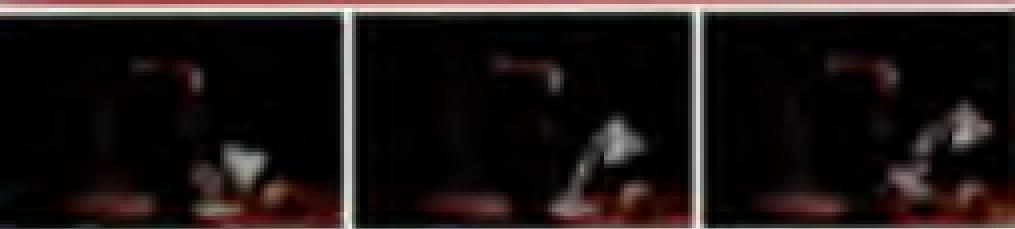


# INTRODUCCIÓN A LA



# GRAFICACIÓN POR COMPUTADOR



FOLEY • VAN DAN • FEINER • HUGHES • PHILLIPS

## INTRODUCCIÓN A LA GRAFICACIÓN POR COMPUTADOR

La graficación por computador es considerada por muchos como el área más emocionante de las ciencias de la computación. Aunque es una rama de la informática, su atractivo se extiende mucho más allá de las fronteras de este campo para abarcar la pintura, la ciencia, la música, la danza, la cinematografía y muchas disciplinas más. La graficación por computador se emplea en la actualidad en varias áreas de la industria, los negocios, el gobierno, la educación y el entretenimiento.

Este libro es una adaptación de la segunda edición de *Computer Graphics: Principles and Practice*, que en la actualidad es la obra más completa y respetada en el campo. Al tiempo que conserva la actualidad y el rigor de su antecesor, esta versión abreviada se concentra en los temas esenciales para quienes se inician en la graficación por computador y ofrece explicaciones más amplias para aquellos lectores con antecedentes menos técnicos. Se han añadido ejemplos resueltos para ilustrar las técnicas y los conceptos, y se ha escrito código de programas en C a fin de incrementar la utilidad del texto.

Este libro se diseñó para usarse en un curso de graficación por computador de uno o dos semestres de duración en cualquier universidad de cuatro años, y suponiendo sólo un poco de preparación matemática, en un curso de un semestre en instituciones de dos años. Esta obra también es el libro ideal para el profesional que desee conocer los rudimentos de este dinámico campo, ya sea para convertirse en practicante o simplemente para apreciar la vasta gama de aplicaciones de la graficación por computador.

Entre los temas que se cubren están la programación elemental de gráficos, el hardware y las aplicaciones. Se incluyen algoritmos importantes para facilitar la implementación de gráficos en dos y tres dimensiones. Un capítulo completo se dedica al paquete de graficación SPHIGS, y coincide con la aparición en el mercado de una versión actualizada de este software. Otro capítulo presenta un panorama conciso de los problemas de la interacción y de las técnicas para implementarla. El libro está profusamente ilustrado, con más de 50 imágenes a todo color, y constituye una introducción accesible a la graficación por computador que establece una base sólida para trabajos posteriores en este fascinante campo.

### OTRAS OBRAS DE INTERÉS PUBLICADAS

POR ADDISON-WESLEY IBEROAMERICANA:

**BEEKMAN:** *Computación & informática hoy. Una mirada a la tecnología del mañana* (65371)

**BERTINO Y MARTINO:** *Sistemas de bases de datos orientadas a objetos* (65356)

**BERGER:** *Graficación por computador con Pascal* (62931)

**BROOKSHEAR:** *Introducción a las ciencias de la computación* (65359)

**EZZELL:** *Programación gráfica en Turbo C++* (60114)

**STROUSTRUP:** *El lenguaje de programación C++*. Segunda edición (60104)

**WEISS:** *Estructuras de datos y análisis de algoritmos* (62571)

Pearson  
Educación



**ADDISON-WESLEY IBEROAMERICANA**

Argentina • Chile • Colombia • España  
Estados Unidos • México • Puerto Rico • Venezuela

9 780201 625998

9 0000>

ISBN 0-201-62591

# **Introducción a la graficación por computador**



# Introducción a la graficación por computador

**James D. Foley**

*Georgia Institute of Technology*

**Andries van Dam**

*Brown University*

**Steven K. Feiner**

*Columbia University*

**John F. Hughes**

*Brown University*

**Richard L. Phillips**

*Los Alamos National Laboratory*

*y The University of Michigan*

Versión en español de

**Ernesto Morales Peake**

*Equilibrio, S.A., México, D.F.*

Con la colaboración técnica de

**Isaac Rudomín**

*Instituto Tecnológico y de Estudios Superiores de Monterrey*

*Campus Estado de México, México*

y

**Alvar Vinacua**

*Universidad Politécnica de Cataluña*

*Barcelona, España*



**ADDISON-WESLEY IBEROAMERICANA**

Argentina • Chile • Colombia • España • Estados Unidos

México • Puerto Rico • Venezuela

Versión en español de la obra titulada *Introduction to Computer Graphics*, de James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes y Richard L. Phillips, publicada originalmente en inglés por Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, E.U.A. © 1994 por Addison-Wesley Publishing Company, Inc.

Esta edición en español es la única autorizada.

**Portada:** Imágenes de "Luxo Jr.", de J. Lasseter, W. Reeves, E. Ostby y S. Lefler. (Derechos reservados © 1986 Pixar.) "Luxo" es una marca registrada de Jac Jacobsen Industrier.

Este libro es una versión condensada y modificada de *Computer Graphics: Principles and Practice*, segunda edición, de Foley, van Dam, Feiner y Hughes, publicada en 1990 en la Addison-Wesley Systems Programming Series, IBM Editorial Board, editores consultores.

Muchos de los nombres que los fabricantes y proveedores emplean para distinguir sus productos se reclaman como marcas registradas. Cuando tales nombres aparecen en este libro y Addison-Wesley sabe que son marcas registradas, se han escrito con mayúscula inicial o totalmente en mayúsculas.

Los programas y aplicaciones presentados en este libro se han incluido por su valor educativo y no están garantizados para ningún propósito específico. Ni el editor ni el autor ofrecen garantías o representaciones, y no aceptan responsabilidad alguna respecto a los programas o aplicaciones.

#### **ADDISON-WESLEY IBEROAMERICANA**

Malabia 2363-2ºG, Buenos Aires 1425, Argentina

Cruz 1469-depto. 21 Independencias

Santiago de Chile

Apartado Aéreo 241-943, Santa Fé de Bogotá, Colombia

Espalier 3 bajo, Madrid 28014, España

1 Jacob Way, Reading, Massachusetts 01867, E.U.A.

Apartado Postal 22-012, México D.F. 14000, México

El Monte Mall 2º Piso Oficina 19-B

Ave. Muñoz Rivera Hato Rey, Puerto Rico 00918

Apartado Postal 51454, Caracas 1050-A, Venezuela

© 1996 por Addison-Wesley Iberoamericana, S.A.

Wilmington, Delaware, E.U.A.

Impreso en Estados Unidos. *Printed in U.S.A.*

ISBN 0-201-62599-7

1 2 3 4 5 6 7 8 9 10-MA-99 98 97 96 95

A Marylou, Heather, Jenn, mis padres y mis maestros

*Jim*

A Debbie, a mi padre, a la memoria de mi madre y a mis hijos Elisa, Lori y Katrin

*Andy*

A Jenni, a mis padres y a mis maestros

*Steve*

A mi familia, a mi maestro Rob Kirby y a la memoria de mi padre

*John*

Y a todos nuestros estudiantes.

---

A Thomas Carlyle, quien, al saber que su sirvienta había quemado accidentalmente su recién terminado manuscrito de *The French Revolution* (La Revolución Francesa), comenzó a escribirlo de nuevo.

*Dick*



---

Este libro es una adaptación de *Computer Graphics: Principles and Practice*, segunda edición (*CGPP*), de Foley, van Dam, Feiner y Hughes. *Introducción a la graficación por computador* se creó a través de la condensación y la modificación de dicha obra exhaustiva de enseñanza y referencia, para satisfacer las necesidades de diversos cursos y requisitos profesionales. Aunque el tamaño de este libro es la mitad del de su antecesor, no se trata simplemente de una versión más reducida. De hecho, se incluye material nuevo y, en algunos casos, un enfoque distinto para la exposición, todo ello aunado a las necesidades del público lector al cual está orientado.

Este libro se diseñó para usarse en un curso de graficación por computador de uno o dos semestres de duración en cualquier universidad de cuatro años y, suponiendo sólo un poco de preparación matemática, en un curso de un semestre en instituciones de dos años. *Introducción a la graficación por computador* también es el libro ideal para el profesional que desee conocer los rudimentos de este dinámico y emocionante campo, ya sea para convertirse en practicante o simplemente para apreciar la vasta gama de aplicaciones de la graficación por computador.

No se pretende que este libro reemplace a *CGPP* como algo más actual o exhaustivo. Sin embargo, hay capítulos en los cuales, debido al abrumador ritmo con el cual avanza este campo, se ha eliminado material viejo y se han actualizado cifras de costo y rendimiento de hardware. Podemos hallar un ejemplo de esta situación en el capítulo 4, donde el enunciado de *CGPP* —el cual, téngase en cuenta, se publicó en 1990— que dice “...una estación de trabajo gráfica usualmente consiste en una UCP capaz de ejecutar al menos varios

millones de instrucciones por segundo (MIPS)...” se actualizó para reflejar el hecho de que hoy día es común encontrar de 20 a 100 MIPS.

A continuación se mencionan otras diferencias y puntos fuertes principales de *Introducción a la graficación por computador*:

- El lenguaje de computación que se emplea en el libro, tanto para fragmentos de programa seudocodificados como para programas completos, es C ANSI moderno. La utilización de C en lugar de Pascal, que se empleó en CGPP, es congruente con las prácticas profesionales y docentes actuales, sobre todo en lo referente a gráficos.
- Como beneficio directo de la utilización de C en este libro, existe ahora una correspondencia uno a uno entre los tipos de datos y las funciones del código empleado en este libro y las de los paquetes de software SRGP y SPHIGS que están disponibles (sin cargo adicional) para complementar al libro (véase la pág. 651).
- El paquete SPHIGS que acabamos de mencionar ha mejorado notablemente con la inclusión de varias características nuevas, por ejemplo fuentes lumínicas múltiples, mejor generación de imágenes y mejor correlación de selección para lograr una superior manipulación interactiva.
- Entre los aspectos más relevantes del libro se encuentran varios ejemplos desarrollados, algunos de ellos bastante extensos. Estos ejemplos están estratégicamente localizados en los capítulos donde mejoran la explicación de conceptos difíciles; como muestra se encuentra un programa completo para definir en forma interactiva curvas cúbicas paramétricas de Bézier.
- Se presenta la importancia de la graficación por computador en el naciente campo del multimedia, a través de la descripción de ejemplos, figuras y referencias de apoyo.
- En el capítulo de transformaciones geométricas se añadió una sección de aspectos matemáticos preliminares. En ella el lector podrá encontrar información suficiente para comprender y aprovechar todo el material de orientación matemática que se presenta subsecuentemente en el libro.

## Programas de estudio

El lector puede seguir varias rutas en este libro. Nosotros proponemos algunas, pero es perfectamente válido seleccionar cualquiera que se adapte a las necesidades del lector; incluso se puede permutar el orden del estudio. Por ejemplo, el material acerca de hardware puede aparecer en un programa de estudio antes o después de lo que propone la posición ordinal del capítulo 4.

**Curso mínimo de un semestre con hincapié en los gráficos bidimensionales.** Si el objetivo es proporcionar una presentación general buena, aunque no rigurosa, de los elementos que competen principalmente a los gráficos bidimensionales,

este programa de estudio será apropiado para estudiantes de una universidad o un colegio de dos o cuatro años.

Capítulo	Secciones
1	Todas
2	Secs. 2.1-2.2
3	Secs. 3.1-3.3
4	Secs. 4.1, 4.2, 4.3 y 4.5
5	Secs. 5.1 (si es necesario), 5.2, 5.3, 5.4
6	Secs. 6.1, 6.2, 6.3, 6.4.1, 6.4.2
8	Todas
9	Secs. 9.1, 9.2.1-9.2.3
11	Secs. 11.1-11.2
12	Lecturas seleccionadas para mostrar capacidades avanzadas

**Curso de un semestre que proporcione un esquema general de los gráficos bidimensionales y tridimensionales.** Este programa de estudio brindará una base sólida de conocimiento de gráficos para aquellos lectores que cuenten con una buena preparación matemática.

Capítulo	Secciones
1	Todas
2	Todas
3	Secs. 3.1-3.5, 3.10, 3.12, 3.15
4	Secs. 4.1, 4.2, 4.3 y 4.5
5	Secs. 5.1 (si es necesario), 5.2-5.5, 5.7, 5.8
6	Secs. 6.1-6.5, 6.6 (excepto 6.6.4), 6.7
7	Secs. 7.1-7.4
8	Todas
9	Secs. 9.1, 9.2.1-9.2.3, 9.3.1-9.3.2
11	Todas
12	Todas
13	Secs. 13.1-13.2
14	Secs. 14.1-14.2

**Curso de dos semestres que abarque gráficos bidimensionales y tridimensionales, modelado y generación de imágenes.** Todos los capítulos (omitiendo quizás temas seleccionados de los capítulos 9 y 10), más temas seleccionados de *CGPP*.

Como muchos de los lectores de *Introducción a la graficación por computador* estarán interesados en consultar a su antecesor, más avanzado y exhaustivo, después de este prefacio se presenta el de *CGPP*. El lector encontrará allí un análisis de las características más importantes de *CGPP* y sugerencias para estructurar cursos con base en dicho libro.

## Reconocimientos

En primer lugar hay que mencionar que aunque todos los autores de *CGPP* participaron de alguna manera en la preparación de este libro, yo asumo toda la

responsabilidad por los nuevos errores que se hayan introducido durante el proceso de adaptación.

David Sklar fue uno de los autores invitados que participaron en *CGPP*, y gran parte del material que aportó a ese libro permanece aquí, en los capítulos 2 y 7. Así mismo, él constituyó una gran ayuda para localizar las versiones electrónicas del código de computación y las ilustraciones del libro original.

Peter Gordon, mi editor, siempre ofreció consejos oportunos, sabios y tranquilizadores durante el proyecto. Jim Rigney, mi supervisor de producción, invertió muchas horas enseñándome los “gajes del oficio”.

Muchas personas ayudaron en diversos aspectos del libro. Entre ellas se encuentran Yvonne Martínez, Chuck Hansen, Tom Rokicki, David Cortesi, Janick (J.) Bergeron, Henry McGilton, Greg Cockcroft, Mike Hawley, Ed Catmull, Loren Carpenter, Harold Borkin, Alan Paeth y Jim White.

Un agradecimiento especial para Ed Angel de la University of New Mexico y sus valientes estudiantes, quienes participaron en las pruebas beta —alfa, en realidad— del primer borrador de este libro, en el tercer trimestre de 1992.

Por último, esto nunca habría llegado a ser realidad sin D. C.

Santa Fe, Nuevo México

R. L. P.

# Prefacio de *Computer Graphics: Principles and Practice*, segunda edición

Ha llegado el momento de la graficación interactiva. Hasta hace poco, se trataba de una misteriosa especialidad que comprendía hardware de presentación muy costoso, grandes recursos de computación y software idiosincrásico. Sin embargo, en los últimos años se ha beneficiado con la reducción constante y en ocasiones espectacular de la relación precio/rendimiento del hardware (p. ej., los computadores personales para hogares u oficinas, que incluyen terminales gráficas estándar) y con el desarrollo de paquetes gráficos de alto nivel, independientes de los dispositivos, que han ayudado a hacer de la programación gráfica una tarea racional y sencilla. Por fin la graficación interactiva está lista para cumplir con la promesa de ofrecernos una comunicación pictórica y facilitar así la interacción entre el ser humano y las máquinas. (Del prefacio de *Fundamentals of Interactive Computer Graphics*, James Foley y Andries van Dam, 1982.)

Este anuncio de la llegada de la graficación por computador se hizo antes de la revolución de la cultura de la computación, generada por los computadores Apple Macintosh e IBM PC y sus clones. En la actualidad, incluso los niños de edad preescolar se sienten a sus anchas con las técnicas de la graficación interactiva, como son la metáfora del escritorio para la manipulación de ventanas y la selección de menús e iconos con un ratón. Las interfaces con el usuario basadas en gráficos han permitido que los neófitos se conviertan en usuarios productivos y un escritorio sin un computador gráfico es algo cada vez más raro.

Al mismo tiempo que la graficación interactiva se ha convertido en algo común para las interfaces con los usuarios y la visualización de datos y objetos, la generación de objetos tridimensionales se vuelve cada vez más realista, como lo demuestran los numerosos comerciales y los efectos especiales cinematográficos generados por computador. Las técnicas que eran experimentales a principios de la década de 1980 son ahora de uso común; los efectos "fotorrealistas", aún más sorprendentes, están a la vuelta de la esquina. Los tipos más sencillos de fotorrealismo que al iniciar la década de 1980 requerían horas de tiempo de computador por imagen, ahora se llevan a cabo en forma rutinaria a tasas de animación (diez o más cuadros/segundo) en computadores personales. De esta manera, las pantallas vectoriales en "tiempo real" de 1981 mostraban objetos alambrados que estaban formados por decenas de miles de vectores sin eliminación de aristas ocultas; en 1990, las pantallas de barrido en tiempo real no sólo pueden mostrar el mismo tipo de dibujos lineales, sino además objetivos móviles compuestos hasta por cien mil triángulos presentados con sombreado de Gouraud o Phong y brillos especulares, así como con eliminación completa

de superficies ocultas. Los sistemas de más alto rendimiento ofrecen correspondencia de textura en tiempo real, eliminación de artefactos de discretización, atenuación atmosférica para representar niebla y bruma, y otros efectos avanzados.

Los estándares de software gráfico también han avanzado considerablemente desde nuestra primera edición. El paquete "SIGGRAPH Core" que existía en 1979, en el cual se basó el paquete SGP de la primera edición, ha desaparecido casi por completo, junto con el tubo de almacenamiento de visualización directa y las pantallas de refrescamiento vectoriales. El paquete PHIGS, mucho más poderoso, que apoya el almacenamiento y la edición de la jerarquía estructural, se ha convertido en un estándar oficial de ANSI e ISO y está disponible para gráficos geométricos en tiempo real de aplicaciones científicas y de ingeniería, así como PHIGS+, que permite establecer iluminación, sombras, curvas y superficies. Los estándares gráficos oficiales complementan otros estándares de nivel más bajo, aunque más eficientes, como QuickDraw de Apple, el paquete de gráficos de barrido bidimensional Xlib de X Window System y la biblioteca gráfica tridimensional GL de Silicon Graphics. También existen implantaciones de la interfaz RenderMan de Pixar para la generación de imágenes fotorrealistas e intérpretes de PostScript para la descripción de páginas impresas y de imágenes en pantalla. Los avances en el software gráfico mejorado se han utilizado para perfeccionar "la vista y el tacto" de las interfaces con los usuarios, y podemos esperar que se incremente el empleo de los efectos tridimensionales, tanto por cuestiones estéticas como para ofrecer nuevas metáforas para la organización y la presentación, así como para navegar a través de la información.

Quizás el más importante de los nuevos movimientos en el mundo de los gráficos sea el interés por el modelado de objetos, no sólo por la creación de sus imágenes. Además, va en aumento el interés por la descripción de la geometría variable en el tiempo y el comportamiento de los objetos tridimensionales. De esta manera, los gráficos tienen que ver cada vez más con la simulación, la animación y un movimiento "de regreso a la física" en el modelado y la generación de imágenes, a fin de crear objetos que se vean y comporten de la manera más realista posible.

Conforme las herramientas y las capacidades disponibles se vuelven cada vez más elaboradas y complejas, surge la necesidad de poder aplicarlas con eficacia. La generación de imágenes ya no es el cuello de botella, de manera que los investigadores comienzan a aplicar técnicas de inteligencia artificial para apoyar el diseño de modelos de objetos, la planificación de movimiento y la disposición de presentaciones gráficas efectivas en dos y tres dimensiones.

Hoy día las fronteras de los gráficos avanzan con rapidez, y un texto que pretenda ser una obra de referencia estándar deberá actualizarse y ampliarse periódicamente. Este libro implicó una reescritura casi total de *Fundamentals of Interactive Computer Graphics*, y aunque esta segunda edición contiene casi el doble de las 623 páginas originales, estamos conscientes de la gran cantidad de material que tuvimos que omitir.

Entre las principales diferencias con respecto a la primera edición se encuentran las siguientes:

- La orientación hacia los gráficos vectoriales se ha reemplazado por la orientación hacia los gráficos de barrido.
- El sencillo paquete gráfico bidimensional de punto flotante (SGP) se ha sustituido con dos paquetes —SRGP y SPHIGS— que reflejan las principales tendencias de la programación de gráficos interactivos. SGP combina características de los paquetes gráficos bidimensionales enteros de barrido QuickDraw y Xlib. SPHIGS, basado en PHIGS, ofrece las características fundamentales de un paquete tridimensional de punto flotante con listas de presentación jerárquicas. Explicamos cómo llevar a cabo la programación de aplicaciones con cada uno de estos paquetes y mostramos cómo implantar los algoritmos básicos de recorte, discretización, visualización y recorrido de lista de presentación que forman la base de estos sistemas.
- Los temas relacionados con las interfaces con el usuario se analizan con gran detenimiento, tanto para metáforas de escritorio bidimensionales como para dispositivos de interacción tridimensionales.
- El tratamiento del modelado se amplió para incluir curvas NURB (*nonuniform rational B-spline*, no uniformes racionales de B *spline*), un capítulo sobre el modelado de sólidos y un capítulo acerca de las técnicas de modelado avanzado, por ejemplo: modelado basado en física, procedimientos como modelos, fractales, sistemas de gramáticas L y sistemas de partículas.
- Mayor cobertura de la generación de imágenes, incluyendo un tratamiento detallado de la eliminación de artefactos de discretización (*aliasing*) y capítulos más extensos sobre la determinación de superficies visibles, iluminación y sombreado, que incluyen modelos de iluminación basados en física, seguimiento de rayos y radiosidad.
- Se añadió material relacionado con las arquitecturas y los algoritmos de los gráficos de barrido avanzados, incluyendo el recorte y la discretización de primitivas complejas y operaciones sencillas de procesamiento de imágenes, como la composición.
- Se agregó una breve introducción a la animación.

Este texto puede ser utilizado por personas sin antecedentes en gráficos y con algunos conocimientos de programación en Pascal, estructuras de datos y algoritmos básicos, arquitectura de computadores y álgebra lineal simple. En un apéndice se repasan los fundamentos matemáticos necesarios. El libro contiene material suficiente para un curso de un año, pero se divide en grupos para permitir una cobertura selectiva. Por lo tanto, el lector puede avanzar a través de una secuencia cuidadosamente diseñada de unidades, comenzando por los

fundamentos más sencillos y de aplicación general y finalizando con los temas más complejos y especializados.

## Grupo básico

En el capítulo 1 se presenta una perspectiva histórica y algunos temas fundamentales sobre hardware, software y aplicaciones. En los capítulos 2 y 3 se describen, respectivamente, el uso y la implantación de SRGP, un sencillo paquete gráfico bidimensional entero. En el capítulo 4 se introduce el hardware gráfico, incluyendo algunas sugerencias para usar el hardware en la implantación de las operaciones descritas en los capítulos anteriores. Los dos capítulos siguientes, 5 y 6, presentan las ideas de las transformaciones en el plano y en el espacio tridimensional, las representaciones con matrices, la utilización de coordenadas homogéneas para unificar transformaciones lineales y afines, y la descripción de vistas tridimensionales, incluyendo las transformaciones de volúmenes de vista arbitrarios a volúmenes de vista canónicos. Por último, en el capítulo 7 se aborda SPHIGS, un paquete gráfico jerárquico tridimensional de punto flotante que es una versión simplificada del estándar PHIGS; además se describe su uso en varias operaciones básicas de modelado. En el capítulo 7 también se analizan las ventajas y desventajas de la jerarquía disponible en PHIGS y la estructura de las aplicaciones que emplean este paquete gráfico.

## Grupo de interfaz con el usuario

En los capítulos 8 a 10 se describe la tecnología actual de los dispositivos de interacción, para luego avanzar hacia temas de mayor nivel del diseño de interfaces con el usuario. Se describen y critican varios paradigmas populares de las interfaces con el usuario. En el capítulo final se trata el software de la interfaz con el usuario, como los administradores de ventanas, los acervos de técnicas de interacción y los sistemas de administración de interfaz con el usuario.

## Grupo de definición de modelos

En los dos primeros capítulos del modelado, 11 y 12, se describen las tecnologías actuales que se utilizan para el modelado geométrico: la representación de curvas y superficies con funciones paramétricas, en especial *splines* cúbicas, así como la representación de sólidos con diversas técnicas, incluyendo las representaciones de frontera y los modelos CSG. En el capítulo 13 se aborda el sistema de visión en color de los seres humanos, varios sistemas de descripción de colores y la conversión entre ellos. En ese capítulo también se trata brevemente el tema de las reglas para usar con eficacia el color.

## Grupo de síntesis de imágenes

En el capítulo 14, el primero de una secuencia de cuatro capítulos, se describe la búsqueda del realismo, desde los primeros dibujos vectoriales hasta los gráficos

sombreados más avanzados. Los artefactos ocasionados por la discretización constituyen un aspecto crucial en los gráficos de barrido y en este capítulo se analizan con gran detalle sus causas y remedios, introduciendo la transformada de Fourier y la convolución. En el capítulo 15 se describen diversas estrategias para determinar superficies visibles, con la suficiente profundidad para que el lector pueda implantar algunas de las más importantes. Los algoritmos de iluminación y sombreado se tratan con detalle en el capítulo 16. En la primera parte de este capítulo se analizan los algoritmos más comunes en el hardware actual, mientras que el resto abarca la textura, las sombras, la transparencia, las reflexiones, los modelos de iluminación basados en física, el rastreo de rayos y los métodos de radiosidad. En el último capítulo del grupo, el 17, se describen la manipulación de imágenes —como el escalamiento, el corte y la rotación de arreglos bidimensionales de píxeles— y las técnicas de almacenamiento de imágenes, incluyendo varios métodos de compresión de imagen.

## Grupo de técnicas avanzadas

Los últimos cuatro capítulos brindan un panorama general de lo más avanzado en el campo (que no permanece estático, por supuesto). En el capítulo 18 se describe el hardware gráfico avanzado que se usa en máquinas comerciales y de investigación de alto nivel; este capítulo es una contribución de Steven Molnar y Henry Fuchs, autoridades en lo referente a arquitecturas gráficas de alto rendimiento. En el capítulo 19 se describen los complejos algoritmos de barrido utilizados en tareas como la discretización de cónicas arbitrarias, la generación de texto con eliminación de artefactos de discretización y la implantación de lenguajes de descripción de páginas, como PostScript. Los dos capítulos finales presentan algunas de las técnicas más importantes en los campos del modelado de alto nivel y la animación por computador.

Los dos primeros grupos sólo contienen material elemental y por ende pueden utilizarse para un curso básico a nivel de licenciatura. En un curso de seguimiento se pueden utilizar los capítulos más avanzados. Como alternativa, los instructores pueden armar cursos a la medida eligiendo capítulos de los distintos grupos.

Por ejemplo, un curso diseñado para introducir a los estudiantes principalmente en el tema de los gráficos bidimensionales incluiría los capítulos 1 y 2, la discretización y los recortes sencillos del capítulo 3, un panorama general de la tecnología haciendo hincapié en las arquitecturas de barrido y los dispositivos de interacción del capítulo 4, las matemáticas homogéneas del capítulo 5 y la vista tridimensional sólo desde la perspectiva de “cómo usarla” de las secciones 6.1 a 6.3. Al grupo de interfaz con el usuario (Caps. 8 a 10) le seguirían secciones introductorias seleccionadas y algoritmos sencillos del grupo de síntesis de imágenes (Caps. 14, 15 y 16).

Un curso general de introducción a los gráficos incluiría los capítulos 1 y 2, los algoritmos básicos del capítulo 3, las arquitecturas de barrido y los dispositivos de interacción del capítulo 4, el capítulo 5 y la mayor parte de los capítulos

6 y 7 sobre visualización y SPHIGS. La segunda mitad del curso incluiría las secciones sobre modelado de los capítulos 11 y 13; sobre síntesis de imágenes de los capítulos 14, 15 y 16; y sobre modelado avanzado del capítulo 20, para proporcionar así una amplia cobertura de estos temas ligeramente más avanzados.

Un curso cuyo tema principal fuera el modelado y la generación de imágenes tridimensionales se iniciaría con las secciones del capítulo 3 acerca de la discretización, el recorte de líneas y polígonos, y la introducción a la eliminación de artefactos de discretización. El curso pasaría después a los capítulos 5 y 6 sobre los aspectos matemáticos básicos de las transformaciones y la visualización, el capítulo 13 acerca del color y luego abarcaría los capítulos 14, 15 y 16 del grupo de síntesis de imágenes. El toque final de la cobertura incluiría selecciones sobre modelado de superficies y sólidos, el capítulo 20 sobre modelado avanzado y el capítulo 21 sobre animación, todos estos temas del grupo de técnicas avanzadas.

## Paquetes gráficos

Los paquetes gráficos SRGP y SPHIGS, diseñados por David Sklar, coautor de los dos primeros capítulos acerca de estos paquetes, están disponibles para los computadores IBM PC (ISBN 0-201-54700-7), Macintosh (ISBN 0-201-54701-5) y estaciones de trabajo UNIX que ejecutan X11, así como varios de los algoritmos para discretización, recortes y visualización.

## Agradecimientos

Este libro no podría haberse producido sin el trabajo dedicado y la indulgencia de muchos amigos y colegas. Nos sentimos en deuda con todas las personas que contribuyeron notablemente a la redacción de uno o más capítulos; muchos otros nos ayudaron con sus comentarios acerca de capítulos individuales, por lo cual también les estamos agradecidos. Lamentamos cualquier omisión involuntaria. Katrina Avery y Lyn Dupré realizaron un trabajo de edición estupendo. Debbie van Dam, Melissa Gold y Clare Campbell ofrecieron una valiosa edición de varias versiones de numerosos capítulos. Agradecemos especialmente a nuestra supervisora de producción, Bette Aaronson, a nuestro director de arte, Joe Vetere, y a nuestro editor, Keith Wollman, no sólo su valiosa ayuda en la producción del libro, sino también su paciencia y su buen humor en circunstancias que, admitimos, fueron bastante críticas: si alguna vez cumplimos con una fecha de entrega en esos cinco frenéticos años, ¡no podemos recordarlo!

La graficación por computador se ha convertido en algo demasiado complejo para que un equipo de cuatro autores principales y tres autores invitados sean expertos en todas las áreas. Nos apoyamos en colegas y estudiantes para ampliar nuestro conocimiento, detectar nuestros errores y ofrecer críticas constructivas acerca de la forma y el contenido. Somos totalmente responsables de cualquier error de omisión o comisión. Las siguientes personas efectuaron detalladas revisiones técnicas de uno o más capítulos: John Airey, Kurt Akeley, Tom Banchoff, Brian Barsky, David Bates, Cliff Besher, Gary Bishop, Peter

Bono, Marvin Bunker, Bill Buxton, Edward Chang, Norman Chin, Michael F. Cohen, William Cowan, John Dennis, Tom Dewald, Scott Draves, Steve Drucker, Tom Duff, Richard Economy, David Ellsworth, Nick England, Jerry Farrell, Robin Forrest, Alain Fournier, Alan Freiden, Christina Gibbs, Melissa Gold, Mark Green, Cathleen Greenberg, Margaret Hagen, Griff Hamlin, Pat Hanrahan, John Heidema, Rob Jacob, Abid Kamran, Mike Kappel, Henry Kauffman, Karen Kendler, David Kurlander, David Laidlaw, Keith Lantz, Hsien-Che Lee, Aaron Marcus, Nelson Max, Deborah Mayhew, Barbara Meier, Gary Meyer, Jim Michener, Jakob Nielsen, Mark Nodine, Randy Pausch, Ari Requicha, David Rosenthal, David Salesin, Hanan Samet, James Sanford, James Sargent, Robin Schaufler, Robert Scheifler, John Schnizlein, Michael Shantzis, Ben Shneiderman, Ken Shoemake, Judith Schrier, John Sibert, Dave Simons, Jonathan Steinhart, Maureen Stone, Paul Strauss, Seth Tager, Peter Tanner, Brice Tebbs, Ben Trumbore, Yi Tso, Greg Turk, Jeff Vroom, Colin Ware, Gary Watkins, Chuck Weger, Kevin Weiler, Turner Whitted, George Wolberg y Larry Wolff.

Varios colegas, incluyendo a Jack Bresenham, Brian Barsky, Jerry Van Aken, Dilip DaSilva (quien sugirió el tratamiento de punto medio uniforme del capítulo 3) y Don Hatfield, no sólo leyeron con cuidado los capítulos sino que también ofrecieron detalladas sugerencias con respecto a los algoritmos.

Recibimos de las siguientes personas una gran ayuda en las tareas de procesamiento de palabras, ayuda de la cual nos sentimos profundamente agradecidos: Katrina Avery, Barbara Britten, Clare Campbell, Tina Cantor, Joyce Cavatoni, Louisa Hogan, Jenni Rodda y Debbie van Dam. Los dibujos para los capítulos 1 y 3 fueron hábilmente creados por Dan Robbins, Scott Snibbe, Tina Cantor y Clare Campbell. Las secuencias de imágenes y figuras creadas para este libro fueron proporcionadas por Beth Cobb, David Kurlander, Allen Paeth y George Wolberg (con ayuda de Peter Karp). Las ilustraciones II.21-37, que muestran una progresión de técnicas de generación de imágenes, fueron diseñadas y producidas en Pixar por Thomas Williams y H. B. Siegel, bajo la dirección de M. W. Mantle, usando el software PhotoRealistic RenderMan de Pixar. Agradecemos a Industrial Light & Magic por permitirnos usar su digitalizador láser para crear las ilustraciones II.24-37, y a Norman Chin por calcular las normales a los vértices para las ilustraciones en color II.30-32. L. Lu y Carles Castellsagué escribieron programas para crear figuras.

Jeff Vogel implantó los algoritmos del capítulo 3, y él y Atul Butte verificaron el código en los capítulos 2 y 7. David Sklar escribió las implantaciones de SRGP y SPHIGS para Macintosh y X11, con la ayuda de Ron Balsys, Scott Boyajian, Atul Butte, Alex Contovounesios y Scott Draves. Randy Pausch y sus estudiantes transportaron los paquetes al ambiente PC.

Hemos instalado un servidor automático de correo electrónico para que nuestros lectores obtengan copias de los algoritmos en un formato legible para computadores\*, con la finalidad de que propongan ejercicios, informen de errores

\* Estas codificaciones, en su forma original en inglés, se encuentran en el apéndice del libro.  
(N. del E.)

en el texto y en SRGP/SPHIGS, y obtengan una lista de erratas en el texto y el software. Envíe su correo electrónico a “graphtext @ cs.brown.edu” con la palabra “Help” en la lista de asunto para recibir la lista actual de servicios disponibles.

*Atlanta, Georgia*  
*Providence, Rhode Island*  
*Nueva York, Nueva York*  
*Providence, Rhode Island*

J. D. F.  
A. v. D.  
S. K. F.  
J. F. H.

---

James Foley (Ph. D., University of Michigan) es fundador y director del Centro de Gráficos, Visualización y Utilización en el Georgia Institute of Technology, donde también ejerce como profesor de ciencias de la computación y de ingeniería eléctrica. Es coautor, con Andries van Dam, de *Fundamentals of Interactive Computer Graphics*, además de ser miembro de ACM, ACM SIGGRAPH, ACM SIGCHI, Human Factors Society, IEEE e IEEE Computer Society. Es editor en jefe de *ACM Transactions on Graphics* y forma parte de los consejos editoriales de *Computers and Graphics* y *Presence*. Sus intereses en la investigación son el UIDE (*User Interface Design Environment*, Ambiente de diseño de interfaz con el usuario, una herramienta de desarrollo de interfaz con el usuario basada en modelos), el software de interfaz con el usuario, la visualización de información, multimedia y los factores humanos de la interfaz con el usuario. El doctor Foley es miembro del IEEE y de las fraternidades Phi Beta Kappa, Tau Beta Pi y Sigma Xi.

Andries van Dam (Ph. D., University of Pennsylvania) fue uno de los fundadores y primer presidente del departamento de ciencias de la computación en Brown University. En la actualidad tiene la mención L. Herbert Ballou University Professor, es profesor de ciencias de la computación en Brown University y científico asesor superior en BLOC Development and Electronic Book Technologies, así como en los consejos de asesoría técnica de ShoGraphics y Microsoft. Es miembro de IEEE Computer Society y ACM, además de ser cofundador de ACM SIGGRAPH. El doctor van Dam ayudó a fundar *Computer Graphics and Image Processing* y *ACM Transactions on Graphics*, y fue editor en estas publicaciones. Fue coautor, con James Foley, del libro *Fundamentals of Interac-*

tive Computer Graphics, texto ampliamente utilizado, y con David Niguidula de *Pascal on the Macintosh: A Graphical Approach*; así mismo, ha publicado más de ocho artículos. En 1984 recibió el premio IEEE Centennial Medal; en 1988, el Governor's Science and Technology Award del estado de Rhode Island; en 1990, el NCGA Academic Award; y en 1991, el premio Steven A. Coons de SIGGRAPH. Los intereses de investigación del doctor van Dam incluyen hipermedia, libros electrónicos y estaciones de trabajo de alto rendimiento para la enseñanza y la investigación.

Steven Feiner (Ph. D., Brown University) es profesor adjunto de ciencias de la computación en Columbia University, donde dirige el laboratorio de graficación por computador e interfaces con el usuario. Sus investigaciones actuales se centran en la síntesis de imágenes, las aplicaciones de la inteligencia artificial en la graficación por computador, las interfaces con el usuario, la animación, los mundos virtuales y los sistemas hipermedia. También participa en el desarrollo de sistemas basados en el conocimiento que automatizan el diseño y la disposición de interfaces gráficas con el usuario. El doctor Feiner pertenece a los consejos editoriales de *Electronic Publishing* y *ACM Transactions on Information Systems*; es miembro de ACM SIGGRAPH y de IEEE Computer Society. En 1991 recibió el premio ONR Young Investigator. El trabajo del doctor Feiner ha aparecido en más de cuarenta publicaciones y se ha presentado en numerosos seminarios y conferencias.

John Hughes (Ph. D., University of California, Berkeley) es profesor adjunto de ciencias de la computación en Brown University, donde dirige, junto con Andries van Dam, el grupo de graficación por computador. Sus intereses de investigación incluyen la aplicación de matemáticas a la graficación por computador para la visualización científica y matemática, la animación por computador automatizada, los fundamentos de la graficación por computador, y la ilustración interactiva. Es miembro de ACM SIGGRAPH y de IEEE Computer Society. En fechas recientes han aparecido artículos suyos en *Computer Graphics* y en *Topology*, y su trabajo acerca de eversiones esféricas se describió en un artículo de *Science News*.

Richard Phillips (Ph. D., University of Michigan) es el principal responsable de esta adaptación de CGPP. El doctor Phillips es profesor emérito de ingeniería eléctrica y computación, y de ingeniería aeroespacial en la University of Michigan. Allí, como director fundador de la Red de ingeniería asistida por computador, fue un elemento clave para el establecimiento de una red de cientos de nodos de estaciones de trabajo para los estudiantes y el personal académico de la facultad de ingeniería. También fue director fundador del *Center for Information Technology Integration*. En la actualidad es miembro del personal técnico de Los Alamos National Laboratory, donde sus intereses de investigación son la visualización científica, las estaciones de trabajo multimedia, la computación distribuida y las publicaciones digitales multimedia. Pertenece a IEEE y ACM, y forma parte del consejo editorial de *Computers and Graphics*.

## NOTAS AL LECTOR

---

El apéndice del libro contiene las versiones originales, totalmente en inglés, de los programas SGRP y PHIGS de los capítulos 2, 3, 7 y 9. En el texto se han traducido los nombres de las variables para facilitar su comprensión, pero para asegurar una completa compatibilidad con las versiones originales que se pueden obtener por correo electrónico (véase pág. xvii), se han incluido dichas versiones en el apéndice.

El presente texto contiene algunos ejemplos de programación en los que se han utilizado identificadores (nombres de variables, de constantes, de funciones, etc.) que contienen caracteres propios del español, como vocales acentuadas, la eñe, y otros. Esto se ha hecho para asegurar la legibilidad. Sin embargo, la mayoría de los compiladores de C no aceptan estos caracteres, por lo que el lector interesado en capturar y compilar los ejemplos deberá eliminar dichos caracteres o sustituirlos por caracteres permitidos.



# ÍNDICE GENERAL

---

<b>1</b>	<b>Introducción: Graficación por computador</b>	1
1.1	Algunas aplicaciones de la graficación por computador	1
1.2	Breve historia de la graficación por computador	7
1.2.1	Tecnología de salida	9
1.2.2	Tecnología de entrada	13
1.2.3	Transportabilidad de software y estándares para gráficos	13
1.3	Ventajas de la graficación interactiva	15
1.4	Marco conceptual para la graficación interactiva	17
1.4.1	Modelado de aplicaciones	18
1.4.2	Presentación del modelo	19
1.4.3	Manejo de la interacción	20
<b>RESUMEN</b>	21	
<b>Ejercicios</b>	22	
<b>2</b>	<b>Programación en el paquete SRGP</b>	25
2.1	Dibujo con SRGP	26
2.1.1	Especificación de primitivas gráficas	26
2.1.2	Atributos	32
2.1.3	Primitivas rellenadas y sus atributos	34
2.1.4	Almacenamiento y recuperación de atributos	39
2.1.5	Texto	39

<b>2.2 Manejo básico de la interacción</b>	<b>42</b>
2.2.1 Factores humanos	42
2.2.2 Dispositivos lógicos de entrada	43
2.2.3 Muestreo y procesamiento dirigido por eventos	44
2.2.4 Modo de muestreo	47
2.2.5 Modo de eventos	48
2.2.6 Correlación de selección para el manejo de interacción	53
2.2.7 Determinación de medidas y atributos de dispositivos	54
<b>2.3 Características de los gráficos de barrido</b>	<b>57</b>
2.3.1 Lienzos	57
2.3.2 Rectángulos de recorte	60
2.3.3 La operación SRGP_copyPixel	61
2.3.4 Modo de escritura u operación de barrido	63
<b>2.4 Limitaciones de SRGP</b>	<b>67</b>
2.4.1 Sistema de coordenadas de aplicación	68
2.4.2 Almacenamiento de primitivas para reespecificación	68
<b>RESUMEN</b>	<b>70</b>
<b>Ejercicios</b>	<b>71</b>
<b>Proyectos de programación</b>	<b>73</b>

## **3 Algoritmos básicos de gráficos de barrido para dibujar primitivas bidimensionales**

75

<b>3.1 Esquema general</b>	<b>76</b>
3.1.1 Implicaciones de la arquitectura del sistema de presentación	76
3.1.2 El ducto de salida en software	80
<b>3.2 Discretización de líneas</b>	<b>81</b>
3.2.1 Algoritmo incremental básico	82
3.2.2 Algoritmo de línea de punto medio	84
3.2.3 Aspectos adicionales	89
<b>3.3 Discretización de círculos</b>	<b>92</b>
3.3.1 Simetría de ocho lados	92
3.3.2 Algoritmo de círculo de punto medio	93
<b>3.4 Rellenado de rectángulos</b>	<b>98</b>
<b>3.5 Rellenado de polígonos</b>	<b>99</b>
3.5.1 Aristas horizontales	102
3.5.2 Astillas	103
3.5.3 Coherencia de aristas y algoritmo de línea de rastreo	103

<b>3.6 Rellenado con patrones 107</b>	
3.6.1 Rellenado con patrones usando discretización 108	
3.6.2 Rellenado con patrones sin discretización repetida 109	
<b>3.7 Primitivas gruesas 112</b>	
3.7.1 Duplicación de píxeles 112	
3.7.2 El pincel móvil 113	
<b>3.8 Recortes en un mundo de barrido de trama 114</b>	
<b>3.9 Recorte de líneas 116</b>	
3.9.1 Recorte de puntos extremos 117	
3.9.2 Recorte de líneas mediante la resolución de ecuaciones simultáneas 117	
3.9.3 Algoritmo de recorte de líneas de Cohen-Sutherland 118	
3.9.4 Algoritmo paramétrico de recorte de líneas 123	
<b>3.10 Recorte de círculos 128</b>	
<b>3.11 Recorte de polígonos 128</b>	
3.11.1 Algoritmo de recorte de polígonos de Sutherland-Hodgman 128	
<b>3.12 Generación de caracteres 132</b>	
3.12.1 Definición y recorte de caracteres 132	
3.12.2 Implantación de una primitiva de salida de texto 135	
<b>3.13 SRGP_copyPixel 136</b>	
<b>3.14 Eliminación de artefactos de discretización (<i>antialiasing</i>) 137</b>	
3.14.1 Aumento de la resolución 137	
3.14.2 Muestreo de área no ponderada 137	
3.14.3 Muestreo de área ponderada 140	
<b>3.15 Temas avanzados 143</b>	
<b>RESUMEN 144</b>	
<b>Ejercicios 145</b>	

## 4 Hardware gráfico 149

<b>4.1 Tecnologías de impresión 150</b>	
<b>4.2 Tecnologías de pantallas 155</b>	
<b>4.3 Sistemas de presentación por barrido de trama 163</b>	
4.3.1 Sistema sencillo de pantalla de barrido 163	
4.3.2 Sistema de presentación por barrido con procesador periférico de dibujo 167	
4.3.3 Funcionalidad adicional del procesador de dibujo 170	

4.3.4 Sistema de dibujo de barrido con procesador de dibujo integrado	173
<b>4.4 Controlador de vídeo</b>	<b>174</b>
4.4.1 Mezclado de vídeo	176
<b>4.5 Dispositivos de entrada para la interacción con el operador</b>	<b>177</b>
4.5.1 Dispositivos localizadores	177
4.5.2 Dispositivos de teclado	180
4.5.3 Dispositivos valuadores	181
4.5.4 Dispositivos de opciones	181
<b>4.6 Digitalizadores de imágenes</b>	<b>181</b>
Ejercicios	183
<b>5 Transformaciones geométricas</b>	<b>185</b>
5.1 Aspectos matemáticos preliminares	185
5.1.1 Los vectores y sus propiedades	186
5.1.2 Producto punto de vectores	188
5.1.3 Propiedades del producto punto	189
5.1.4 Matrices	189
5.1.5 Multiplicación de matrices	190
5.1.6 Determinantes	190
5.1.7 Transpuesta de una matriz	191
5.1.8 Inversa de una matriz	191
5.2 Transformaciones bidimensionales	193
5.3 Coordenadas homogéneas y representación matricial de transformaciones bidimensionales	196
5.4 Composición de transformaciones bidimensionales	201
5.5 Transformación ventana-área de vista	203
5.6 Eficiencia	206
5.7 Representación matricial de transformaciones tridimensionales	206
5.8 Composición de transformaciones tridimensionales	210
5.9 Las transformaciones como un cambio en el sistema de coordenadas	215
Ejercicios	219
<b>6 Vista tridimensional</b>	<b>221</b>
6.1 La cámara sintética y los pasos en la vista tridimensional	221
6.2 Proyecciones	224
6.2.1 Proyecciones de perspectiva	225
6.2.2 Proyecciones paralelas	227

<b>6.3 Especificación de una vista tridimensional arbitraria</b>	<b>229</b>
<b>6.4 Ejemplos de vista tridimensional</b>	<b>235</b>
6.4.1 Proyecciones de perspectiva	236
6.4.2 Proyecciones paralelas	241
6.4.3 Volúmenes de vista finitos	242
<b>6.5 Las matemáticas de las proyecciones geométricas planas</b>	<b>242</b>
<b>6.6 Implantación de proyecciones geométricas planas</b>	<b>246</b>
6.6.1 El caso de la proyección paralela	247
6.6.2 El caso de la proyección de perspectiva	253
6.6.3 Recortes con respecto a un volumen de vista canónico en tres dimensiones	258
6.6.4 Recortes en coordenadas homogéneas	260
6.6.5 Correspondencia a un área de vista	264
6.6.6 Resumen de implantación	265
<b>6.7 Sistemas de coordenadas</b>	<b>266</b>
<b>Ejercicios</b>	<b>268</b>

## **7 Jerarquía de objetos y PHIGS simple (SPHIGS)** **271**

<b>7.1 Modelado geométrico</b>	<b>273</b>
7.1.1 Modelos geométricos	275
7.1.2 Jerarquía en modelos geométricos	275
7.1.3 Relación entre el modelo, el programa de aplicación y el sistema gráfico	279
<b>7.2 Características de los paquetes gráficos de modo retenido</b>	<b>280</b>
7.2.1 El almacenamiento central de estructuras y sus ventajas	281
7.2.2 Limitaciones de los paquetes de modo retenido	282
<b>7.3 Definición y presentación de estructuras</b>	<b>283</b>
7.3.1 Apertura y cierre de estructuras	283
7.3.2 Especificación de primitivas de salida y sus atributos	284
7.3.3 Colocación de estructuras para recorrido de presentación	287
7.3.4 Vista	288
7.3.5 Aplicaciones gráficas que comparten una pantalla a través de la administración de ventanas	292
<b>7.4 Transformaciones de modelado</b>	<b>292</b>
<b>7.5 Redes de estructuras jerárquicas</b>	<b>298</b>
7.5.1 Jerarquía de dos niveles	298
7.5.2 Jerarquía simple de tres niveles	300
7.5.3 Construcción ascendente ( <i>bottom-up</i> ) del robot	300
7.5.4 Programas interactivos de modelado	305

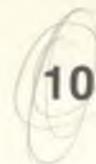
<b>7.6 Composición de matrices en el recorrido de presentación</b>	<b>306</b>
<b>7.7 Manejo de atributos de apariencia en la jerarquía</b>	<b>310</b>
7.7.1 Reglas de herencia	310
7.7.2 Atributos de SPHIGS y texto no afectados por las transformaciones	313
<b>7.8 Actualización de pantalla y modos de presentación</b>	<b>313</b>
<b>7.9 Edición de redes de estructuras para obtener efectos dinámicos</b>	<b>315</b>
7.9.1 Acceso a elementos por medio de índices y etiquetas	316
7.9.2 Operaciones de edición dentro de la estructura	316
7.9.3 Bloques de ejemplares para facilitar la edición	317
7.9.4 Control de la regeneración automática de la imagen en pantalla	319
<b>7.10 Interacción</b>	<b>320</b>
7.10.1 Localizador	320
7.10.2 Correlación de selección	321
<b>7.11 Temas avanzados</b>	<b>328</b>
7.11.1 Características de salida adicionales	328
7.11.2 Aspectos de implantación	329
7.11.3 Optimización de la presentación de modelos jerárquicos	331
7.11.4 Limitaciones del modelado jerárquico en PHIGS	331
7.11.5 Formas alternativas de modelado jerárquico	332
7.11.6 Otros estándares (industriales)	333
<b>RESUMEN</b>	<b>334</b>
<b>Ejercicios</b>	<b>335</b>

## **8 Dispositivos de entrada, técnicas de interacción y tareas de interacción**

<b>8.1 Hardware de interacción</b>	<b>338</b>
8.1.1 Dispositivos localizadores	339
8.1.2 Dispositivos de teclado	341
8.1.3 Dispositivos valuadores	341
8.1.4 Dispositivos de opción	341
8.1.5 Otros dispositivos	342
8.1.6 Dispositivos de interacción tridimensional	342
<b>8.2 Tareas de interacción básicas</b>	<b>345</b>
8.2.1 Tarea de interacción para posicionamiento	345
8.2.2 Tarea de interacción para selección: conjunto de opciones de tamaño variable	346

8.2.3 Tarea de interacción para selección: conjunto de opciones de tamaño relativamente fijo	349
8.2.4 Tarea de interacción para texto	353
8.2.5 Tarea de interacción para cuantificación	353
8.2.6 Tareas de interacción tridimensional	354
<b>8.3 Tareas de interacción compuestas</b>	<b>357</b>
8.3.1 Recuadros de diálogo	357
8.3.2 Técnicas de construcción	358
8.3.3 Manipulación dinámica	359
<b>8.4 Conjuntos de herramientas para técnicas de interacción</b>	<b>361</b>
<b>RESUMEN</b>	<b>362</b>
<b>Ejercicios</b>	<b>362</b>

<b>9 Representación de curvas y superficies</b>	<b>363</b>
9.1 Mallas poligonales	365
9.1.1 Representación de mallas poligonales	366
9.1.2 Ecuaciones de planos	368
9.2 Curvas cúbicas paramétricas	371
9.2.1 Características básicas	372
9.2.2 Curvas de Hermite	376
9.2.3 Curvas de Bézier	380
9.2.4 <i>B-splines</i> uniformes, no racionales	387
9.2.5 <i>B-splines</i> no uniformes, no racionales	391
9.2.6 Segmentos de curva polinomial cúbica racional, no uniforme	393
9.2.7 Ajuste de curvas a puntos digitalizados	348
9.2.8 Comparación de las curvas cúbicas	394
9.3 Superficies bicúbicas paramétricas	396
9.3.1 Superficies de Hermite	397
9.3.2 Superficies de Bézier	399
9.3.3 Superficies <i>B-spline</i>	400
9.3.4 Normales a superficies	401
9.3.5 Dibujo de superficies bicúbicas	401
9.4 Superficies cuádricas	403
9.5 Técnicas de modelado especializado	404
9.5.1 Modelos fractales	405
9.5.2 Modelos gramaticales	410
<b>RESUMEN</b>	<b>414</b>
<b>Ejercicios</b>	<b>415</b>



## 10 Modelado de sólidos

- 10.1 Representación de sólidos 418
- 10.2 Operaciones regularizadas de conjuntos booleanos 419
- 10.3 Generación de ejemplares de primitivas 423
- 10.4 Representaciones de barrido 424
- 10.5 Representaciones de fronteras 426
  - 10.5.1 Poliedros y fórmula de Euler 427
  - 10.5.2 Operaciones de conjuntos booleanos 429
- 10.6 Representaciones de partición espacial 430
  - 10.6.1 Descomposición en celdas 431
  - 10.6.2 Enumeración de ocupación espacial 431
  - 10.6.3 Árboles de octantes 432
  - 10.6.4 Árboles binarios de partición de espacio 436
- 10.7 Geometría sólida constructiva 438
- 10.8 Comparación de representaciones 440
- 10.9 Interfaces con el usuario para el modelado de sólidos 443
  - RESUMEN 443
  - Ejercicios 444

417

## 11 Luz acromática y cromática

- 11.1 Luz acromática 447
  - 11.1.1 Selección de intensidades 448
  - 11.1.2 Aproximación por medios tonos 451
- 11.2 Luz cromática 455
  - 11.2.1 Psicofísica 456
  - 11.2.2 Diagrama de cromaticidad CIE 459
- 11.3 Modelos de colores para gráficos de trama 463
  - 11.3.1 Modelo de colores RGB 464
  - 11.3.2 Modelo de colores CMY 464
  - 11.3.3 Modelo de colores YIQ 466
  - 11.3.4 Modelo de colores HSV 467
  - 11.3.5 Especificación interactiva del color 471
  - 11.3.6 Interpolación en el espacio de colores 472
- 11.4 Utilización del color en la graficación por computador 473
  - RESUMEN 476
  - Ejercicios 476

447

<b>12 Búsqueda del realismo visual</b>	<b>479</b>
12.1 ¿Por qué el realismo? 480	
12.2 Dificultades fundamentales 482	
12.3 Técnicas de generación para dibujos de línea 484	
12.3.1 Vistas ortográficas múltiples 484	
12.3.2 Proyecciones de perspectiva 484	
12.3.3 Indicadores de profundidad 485	
12.3.4 Recortes de profundidad 486	
12.3.5 Textura 486	
12.3.6 Color 486	
12.3.7 Determinación de líneas visibles 487	
12.4 Técnicas de generación para imágenes sombreadas 487	
12.4.1 Determinación de superficies visibles 487	
12.4.2 Iluminación y sombreado 488	
12.4.3 Sombreado interpolado 488	
12.4.4 Propiedades materiales 489	
12.4.5 Modelado de superficies curvas 489	
12.4.6 Iluminación y sombreado mejorados 489	
12.4.7 Textura 489	
12.4.8 Sombras 489	
12.4.9 Transparencia y reflexión 490	
12.4.10 Modelos de cámara mejorados 490	
12.5 Modelos de objetos mejorados 4941	
12.6 Dinámica y animación 4941	
12.6.1 El valor del movimiento 491	
12.6.2 Animación 492	
12.7 Estereoóptica 495	
12.8 Pantallas mejoradas 496	
12.9 Interacción con nuestros otros sentidos 497	
RESUMEN 497	
Ejercicios 498	
<b>13 Determinación de superficies visibles</b>	<b>499</b>
13.1 Técnicas para algoritmos eficientes de superficies visibles 501	
13.1.1 Coherencia 502	
13.1.2 Transformación de perspectiva 503	
13.1.3 Extensiones y volúmenes acotantes 505	
13.1.4 Eliminación de caras posteriores 507	
13.1.5 Partición espacial 509	
13.1.6 Jerarquía 509	

- 13.2 Algoritmo de z-buffer (memoria de profundidad) 510
- 13.3 Algoritmos de línea de barrido 514
- 13.4 Traza de rayos en superficies visibles 519
  - 13.4.1 Cálculo de intersecciones 521
  - 13.4.2 Consideraciones de eficiencia para la traza de rayos en superficies visibles 524
- 13.5 Otros métodos 526
  - 13.5.1 Algoritmos de prioridad de listas 526
  - 13.5.2 Algoritmos de subdivisión de área 531
  - 13.5.3 Algoritmos para superficies curvas 533
- RESUMEN 535**
- Ejercicios 537**

14

## Iluminación y sombreado

541

- 14.1 Modelos de iluminación 542
  - 14.1.1 Luz ambiental 542
  - 14.1.2 Reflexión difusa 543
  - 14.1.3 Atenuación atmosférica 548
  - 14.1.4 Reflexión especular 549
  - 14.1.5 Mejora del modelo de fuente luminosa puntual 552
  - 14.1.6 Fuentes luminosas múltiples 554
  - 14.1.7 Modelos de iluminación físicos 554
- 14.2 Modelos de sombreado para polígonos 557
  - 14.2.1 Sombreado constante 557
  - 14.2.2 Sombreado interpolado 558
  - 14.2.3 Sombreado de malla poligonal 559
  - 14.2.4 Sombreado de Gouraud 560
  - 14.2.5 Sombreado de Phong 561
  - 14.2.6 Problemas con el sombreado interpolado 563
- 14.3 Detalle de superficie 565
  - 14.3.1 Polígonos de detalle de superficie 565
  - 14.3.2 Correspondencia de texturas 565
  - 14.3.3 Correspondencia de protuberancias 567
  - 14.3.4 Otros métodos 568
- 14.4 Sombras 568
  - 14.4.1 Generación de sombras por línea de barrido 569
  - 14.4.2 Volúmenes de sombra 571
- 14.5 Transparencia 572
  - 14.5.1 Transparencia no refractiva 573
  - 14.5.2 Transparencia refractiva 575

<b>14.6 Algoritmos de iluminación global</b>	<b>577</b>
<b>14.7 Trazo de rayos recursivo</b>	<b>579</b>
<b>14.8 Métodos de radiosidad</b>	<b>583</b>
14.8.1 Ecuación de radiosidad	584
14.8.2 Cálculo de factores de forma	586
14.8.3 Refinamiento progresivo	589
<b>14.9 Ducto de generación (<i>rendering</i>)</b>	<b>590</b>
14.9.1 Ductos de iluminación local	591
14.9.2 Ductos de iluminación global	594
14.9.3 Refinamiento progresivo	595
<b>RESUMEN</b>	<b>595</b>
<b>Ejercicios</b>	<b>595</b>
<b>Apéndice: Versión original en inglés de las codificaciones de SRGP y PHIGS</b>	<b>597</b>
<b>Bibliografía</b>	<b>607</b>
<b>Vocabulario técnico bilingüe</b>	<b>625</b>
<b>Índice de materias</b>	<b>637</b>



# **Introducción a la graficación por computador**

Bienvenidos al mundo de la graficación por computador, para algunos el área más emocionante de las ciencias de la computación. La graficación por computador es una rama de la informática, pero su atractivo se extiende mucho más allá de las fronteras de dicho campo relativamente especializado. En su corta vida, la graficación por computador ha atraído a algunas de las personas más creativas del mundo. Éstas provienen de todas las disciplinas: arte, ciencia, música, danza, cinematografía y muchas más. Como ejemplos específicos, los animadores de la compañía Disney usaron la graficación por computador para dar esa aura especial a la escena del salón de baile en *La bella y la bestia*. Merce Cunningham, coreógrafa, emplea la graficación por computador para producir partituras de labanotación, es decir, los guiones con los cuales los bailarines aprenden sus pasos. David Em, un respetado artista que emplea medios convencionales, ahora utiliza con frecuencia la graficación por computador en su trabajo. De hecho, como la mejor forma de dar a conocer la emoción y la diversidad de la graficación por computador es a través de sus aplicaciones, veamos varias de ellas con mayor detalle.

## 1.1 Algunas aplicaciones de la graficación por computador

La graficación por computador se emplea en la actualidad en varias áreas de la industria, los negocios, el gobierno, la educación y el entretenimiento. La lista

de aplicaciones es enorme y crece con rapidez a medida que los computadores con capacidades gráficas se convierten en artículos de consumo. A continuación veremos brevemente algunas de estas áreas.

■ **Interfaces con el usuario.** Quizás no se haya percatado de ello, pero es probable que usted ya haya utilizado la graficación por computador. Si ha trabajado con un Macintosh o con un computador personal compatible con IBM que ejecuta Windows 3.1, usted es prácticamente un avezado usuario de los gráficos. Después de todo, la mayoría de las aplicaciones que se ejecutan en computadores personales o estaciones de trabajo tienen interfaces con el usuario (similares a la que se presenta en la figura 1.1) que se apoyan en sistemas de ventanas para administrar múltiples actividades simultáneas, así como seleccionar elementos de los menús, iconos y objetos de la pantalla. Los programas de procesamiento de texto, hoja de cálculo y de compuendio son aplicaciones típicas que aprovechan estas técnicas de interfaz con el usuario. Como veremos más adelante, la graficación por computador desempeña un papel integral tanto en las funciones de entrada como en las de salida de las interfaces con el usuario.

■ **Graficación interactiva en los negocios, la ciencia y la tecnología.** La siguiente aplicación más usual de los gráficos es para crear gráficos bidimensionales y tridimensionales de funciones matemáticas, físicas y económicas; histogramas y diagramas de barras y circulares; diagramas de programación de actividades; diagramas de inventario y producción; etcétera. Todos estos gráficos

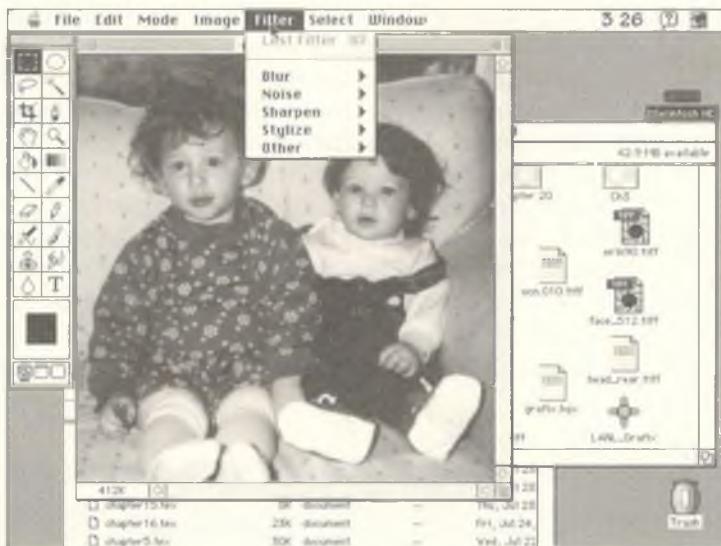
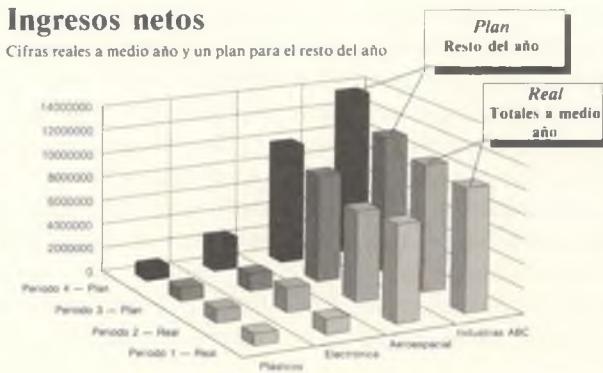


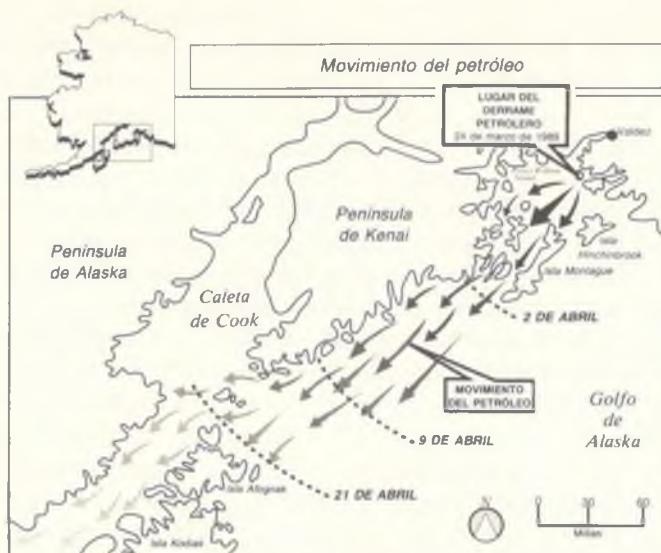
Figura 1.1 Pantalla de Macintosh en la cual se presentan ventanas, un menú e iconos de escritorio.

se usan para presentar de manera significativa y concisa las tendencias y los patrones extraídos de los datos, para así esclarecer fenómenos complejos y facilitar la toma de decisiones informada. En la figura 1.2 se presenta un ejemplo típico de un gráfico tridimensional de datos empresariales.

- **Cartografía.** La graficación por computador se emplea para producir representaciones precisas y esquemáticas de fenómenos geográficos y de otro tipo, a partir de datos de mediciones. Como ejemplos están los mapas geográficos, de relieve, de exploración para perforaciones y minería, oceanográficos, meteorológicos, de contorno y de densidad de población. En la figura 1.3 se muestra un mapa producido para ilustrar algunos aspectos del desastre ocasionado por el derrame del buque tanque *Valdez* de Exxon, en 1989. Este mapa es un ejemplo de los mapas *temáticos*, donde los valores de los datos, como el movimiento del petróleo derramado, se sobreponen a un fondo de mapa.
- **Medicina.** La graficación por computador tiene una función cada vez más importante en campos como la medicina de diagnóstico y la planificación de cirugías. En el segundo caso, los cirujanos usan los gráficos como auxiliares para guiar instrumentos y determinar precisamente dónde hay que eliminar tejidos enfermos. En la figura 1.4 se presenta una aplicación de la graficación por computador en el diagnóstico. En ella se observan tres imágenes obtenidas de la producción de imágenes por resonancia magnética (MRI, *magnetic resonance imaging*) de un cerebro humano. A partir de una serie de rastreos paralelos como los ilustrados en la parte (a), los diagnosticadores puede sintetizar una representación tridimensional del cerebro, la cual se presenta en las partes (b) y (c). El usuario puede manipular el modelo en forma interactiva para revelar información detallada acerca de la condición del cerebro. Otra aplicación médica especialmente interesante de la graficación por computador se presenta en la



**Figura 1.2** Datos empresariales presentados como un gráfico de barras tridimensional.

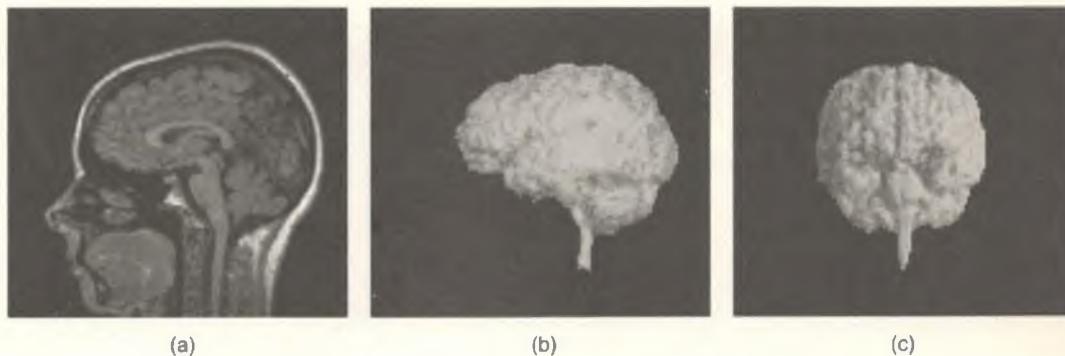


**Figura 1.3** Utilización de un fondo cartográfico para presentar datos relacionados geográficamente. (Cortesía de Tom Poiker, Simon Fraser University.)

figura 1.5, en la cual se observan gráficos tridimensionales de los datos obtenidos al iluminar tejido vivo con luz láser, una técnica conocida como *biopsia óptica*. En este caso, el hígado canino normal (parte a) y el canceroso (parte b) muestran huellas ópticas muy distintas, lo que ofrece la esperanza de un diagnóstico no quirúrgico.

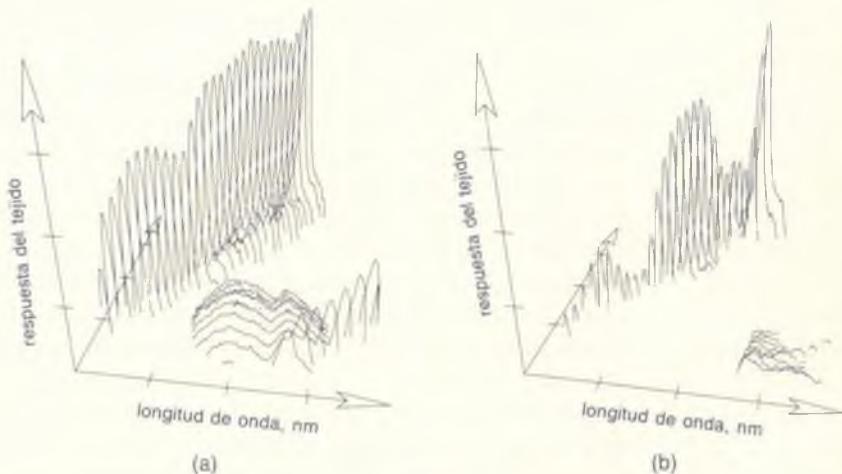
■ **Bosquejo y diseño asistido por computador.** En el diseño asistido por computador (CAD, *computer-aided design*), los usuarios emplean la graficación interactiva para diseñar componentes y sistemas de dispositivos mecánicos, eléctricos, electromecánicos y electrónicos, incluyendo estructuras como edificios, carrocerías de automóviles, cacos de aviones y barcos, pastillas de circuitos integrados a escala muy grande (VLSI, *very large-scale integration*) y redes telefónicas y de computadores. Por lo general, el punto central de la atención es la interacción con un modelo basado en computador del componente o sistema que se diseña, pero en ocasiones el usuario sólo quiere producir dibujos precisos de los componentes y el montaje, como en el caso de bosquejos en línea o los planos arquitectónicos. En la figura 1.6 se presenta un ejemplo del CAD para arquitectura, el dibujo de una planta de energía.

■ **Sistemas multimedia.** La graficación por computador tiene una función crucial en la rápidamente creciente área de los sistemas multimedia. Como su nombre lo indica, los multimedia comprenden más de un medio de comunicación. En

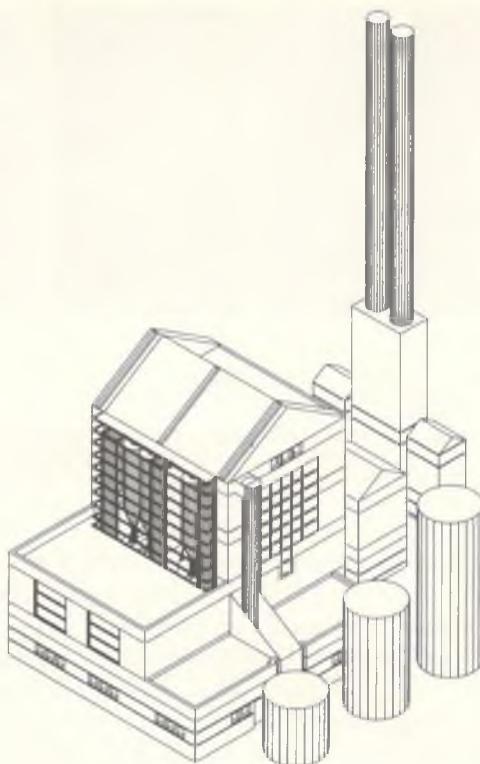


**Figura 1.4** Representación tridimensional del cerebro humano construida a partir de rastreos MRI. Se procesa una serie de imágenes del tipo presentado en (a) para producir las escenas tridimensionales que se muestran en (b), una vista lateral, y (c), la vista posterior. (Cortesía de John George, Los Alamos National Laboratory.)

estos sistemas se emplean convencionalmente texto, gráficos y sonido, pero pueden existir muchos más [PHIL91]. En la figura 1.7 se observa cómo se podrían usar componentes multimedia no convencionales para la educación. Esta figura se obtuvo de un libro de texto multimedia hipotético de gráficos por computador [PHIL92]. En la parte (a) se presenta el procedimiento para dibujar un segmento de línea bidimensional (en el capítulo 3 analizaremos el significado de este código). El código que aparece en la página multimedia está *vivo*; es decir,



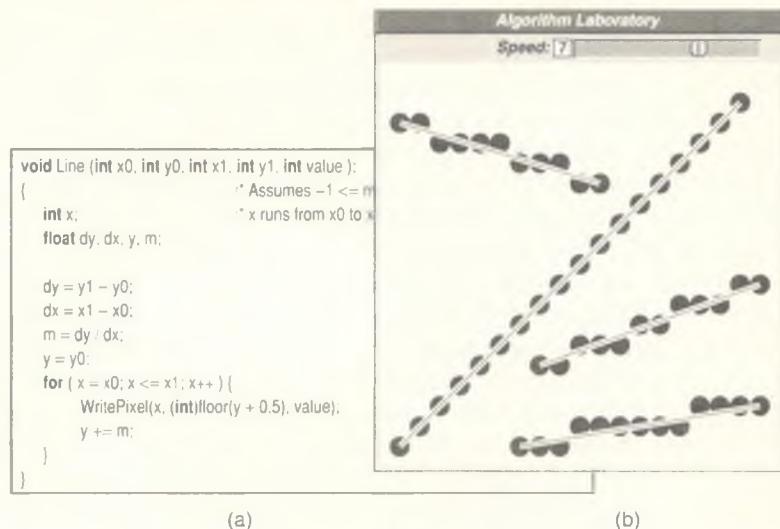
**Figura 1.5** Resultados de una biopsia óptica. (a) Marca de tejido hepático normal. (b) Marca de tejido hepático canceroso. (Cortesía de Thomas Loree, Los Alamos National Laboratory.)



**Figura 1.6** Dibujo de una planta de energía producido por un sistema CAD para arquitectura. (Cortesía de Harold Borkin, Architecture and Planning Research Lab, University of Michigan.)

el lector lo puede ejecutar si apunta a él. Aparece un tablero de algoritmos, presentado en la parte (b), que permite al lector ensayar varias condiciones de puntos extremos y ver las acciones del algoritmo.

- **Simulación y animación para visualización científica y entretenimiento.** Los filmes de animación generados por computador y las presentaciones del comportamiento variable en el tiempo de objetos reales y simulados constituyen una herramienta cada vez más común para la visualización científica y del campo de la ingeniería (véase Ilust. en color 1). Podemos usar estos gráficos para estudiar entidades matemáticas abstractas o modelos matemáticos de fenómenos como el flujo de fluidos, la relatividad, las reacciones nucleares y químicas, la función de los órganos y el sistema fisiológico, y la deformación de estructuras mecánicas sujetas a diferentes tipos de carga. Otra área de tecnología avanzada es la producción de efectos especiales de gran elegancia para filmes (véase Ilust. en color 2 y 3). Existen mecanismos complejos que permiten modelar los objetos y representar las luces y las sombras.



**Figura 1.7** Algoritmo interactivo de un libro de texto multimedia. (a) El código de computador que implanta el algoritmo. (b) Tablero de bosquejo interactivo, que aparece cuando el lector apunta al código. Al especificar los puntos extremos de una línea se ejecuta el código.

## 1.2 Breve historia de la graficación por computador

Este libro se centra en los principios y las técnicas fundamentales que se desarrollaron en el pasado y que aún son aplicables (con toda probabilidad seguirán siéndolo en el futuro). En esta sección veremos brevemente el desarrollo histórico de la graficación por computador a fin de establecer el contexto adecuado para los sistemas actuales. Se pueden hallar tratamientos más completos de la interesante evolución de este campo en [PRIN71], [MACH78], [CHAS81] y [CACM84]. Es más fácil elaborar una crónica de la evolución del hardware que de la del software, ya que la primera ha tenido mayor influencia en el desarrollo de este campo. Por consiguiente, comenzaremos por el hardware.

Incluso en los primeros días de la computación, se elaboraban gráficos rudimentarios en dispositivos de impresión como teletipos e impresoras de línea. El computador Whirlwind, desarrollado en el MIT en el año de 1950, tenía un tubo de rayos catódicos (CRT, *cathode ray tube*) dirigido por computador que se empleaba para presentar las salidas, tanto para el operador como para cámaras que producían versiones impresas. El nacimiento de la graficación interactiva moderna se puede hallar en el importante trabajo doctoral de Ivan Sutherland acerca del sistema de dibujo Sketchpad [SUTH63]. Él introdujo estructuras de datos para almacenar jerarquías de símbolos construidas por medio

de copias de componentes estándar, una técnica similar a la de usar plantillas de plástico para dibujar símbolos de circuitos. Sutherland desarrolló también técnicas de interacción que usaban el teclado y un lápiz de luz (un dispositivo apuntador manual que detecta la luz emitida por los objetos en la pantalla) para seleccionar opciones, apuntar y dibujar; además, formuló otras ideas y técnicas fundamentales que aún se usan en la actualidad. De hecho, muchas de las características presentadas en Sketchpad aparecen en el paquete gráfico PHICS analizado en el capítulo 7.

Al mismo tiempo, los fabricantes de computadores, automóviles y equipo aeroespacial se percataron del enorme potencial que tenían el CAD y la manufactura asistida por computador (CAM, *computer-aided manufacture*) para automatizar la elaboración de bosquejos y otras actividades que requerían mucho trabajo de dibujo. El sistema DAC de General Motors [JACK64] para el diseño de automóviles y el sistema Digitek de Itek [CHAS81] para el diseño de lentes fueron programas pioneros que demostraron la utilidad de la interacción gráfica en los ciclos iterativos de diseño comunes en la ingeniería. A mediados de la década de 1960 ya habían aparecido varios productos comerciales y proyectos de investigación.

Puesto que en esa época la entrada/salida (E/S) de los computadores se realizaba principalmente por medio de tarjetas perforadas, existía la esperanza de que un hallazgo revolucionario permitiera la comunicación interactiva entre el usuario y el computador. La graficación interactiva, como la “ventana en el computador”, sería parte integral de ciclos de diseño interactivo cada vez más rápidos. Sin embargo, los resultados no fueron tan espectaculares, ya que la graficación interactiva permaneció fuera del alcance de casi todos, con excepción de las organizaciones con la mejor tecnología.

De esta manera, la graficación por computador constituyó un pequeño y especializado campo hasta principios de la década de 1980, sobre todo debido al elevado costo del hardware y los escasos programas de aplicación basados en gráficos. Fue entonces que los computadores personales con pantallas gráficas de barrido, como el Apple Macintosh y el IBM y sus clones, popularizaron los gráficos de mapa de bits para la interacción usuario-computador. Un mapa o arreglo bidimensional de bits (*bitmap*) es una representación con unos y ceros de un arreglo rectangular de puntos de la pantalla, llamados pixeles o pels (el término proviene de *picture elements*, que en inglés significa “elementos de imágenes”). Una vez que los gráficos de mapa de bits se convirtieron en algo accesible, sobrevino una explosión de aplicaciones basadas en gráficos, baratas y fáciles de usar. Las interfaces con el usuario basadas en gráficos permitieron que millones de nuevos usuarios controlaran programas de aplicación sencillos y de bajo costo, como hojas de cálculo, procesadores de texto y programas de dibujo.

El concepto de escritorio se convirtió en una metáfora común para la organización del espacio en la pantalla. Usando un administrador de ventanas, el usuario era capaz de crear, colocar y cambiar el tamaño de áreas rectangulares de la pantalla, llamadas ventanas, que actuaban como terminales gráficas virtuales, cada una de las cuales ejecutaba una aplicación. Este método permitió

que los usuarios alternaran entre varias actividades con sólo apuntar a la ventana deseada, por lo general con un dispositivo apuntador llamado **ratón (mouse)**. Las ventanas se podrían sobreponer arbitrariamente, como si fueran papeles sobre un escritorio. Otra parte de esta metáfora de escritorio era la presentación de iconos que representaban no sólo archivos de datos y programas de aplicación, sino también objetos comunes en una oficina —como archiveros, buzones, imprentas y cubos de basura— que realizaban en el computador operaciones equivalentes a las de sus contrapartes en la vida real (véase la Fig. 1.1).

La **manipulación directa** de los objetos usando la técnica “**apunte y oprima**” reemplazó gran parte del teclado que requerían los mandatos utilizados en los computadores más antiguos. Así, los usuarios podían seleccionar iconos para activar los correspondientes programas u objetos, o seleccionar botones en menús descendentes o de ventana para seleccionar opciones. Hoy día casi todos los programas de aplicación interactivos, incluyendo aquellos que se usan para manipular texto (p. ej., procesadores de texto) o datos numéricos (p. ej., programas de hoja de cálculo) usan gran cantidad de gráficos en su interfaz con el usuario para visualizar y manipular los objetos específicos de la aplicación.

### 1.2.1 Tecnología de salida

Los dispositivos de pantalla desarrollados a mediados de la década de 1960 y que fueron de uso común hasta mediados de la década de 1980 se denominan **pantallas de vector, de trazo, de dibujo de líneas o caligráficas**. Aquí, el término *vector* se usa como sinónimo de *línea*; un *trazo* es una línea corta y los caracteres se forman como secuencias de estos trazos. Un sistema vectorial típico consiste en un procesador de pantalla conectado como dispositivo periférico de E/S a una unidad central de procesamiento (UCP), una memoria de pantalla y un CRT. La esencia del sistema vectorial es que el haz de electrones que escribe en el recubrimiento de fósforo del CRT (véase el Cap. 4) se desvía de un punto extremo a otro, según lo determine el orden arbitrario de los mandatos de dibujo; esta técnica se denomina **rastreo aleatorio** (véase la Fig. 1.10b). Como la salida luminosa del fósforo decae en decenas o a lo sumo cientos de microsegundos, el procesador de la pantalla debe efectuar un ciclo por la lista de presentación para **refrescar** el fósforo al menos 30 veces por segundo (30 Hz) a fin de evitar el parpadeo.

A mediados de los años setenta, el desarrollo de gráficos de barrido baratos basados en la tecnología de la televisión contribuyó más al crecimiento del campo que cualquier otra tecnología. Las **pantallas de barrido** almacenan las **primitivas de dibujo** (líneas, caracteres y áreas con sombras sólidas o patrones) en una memoria de refreshamiento en función de los pixeles que componen la primitiva, como se ilustra en la figura 1.8. En algunas pantallas de barrido, un controlador de pantalla en hardware (como el que se muestra en la figura) recibe e interpreta secuencias de mandatos de salida; en otros sistemas más sencillos y comunes, como los de computadores personales, el controlador de pantalla sólo existe como componente de software de la biblioteca gráfica y la memoria

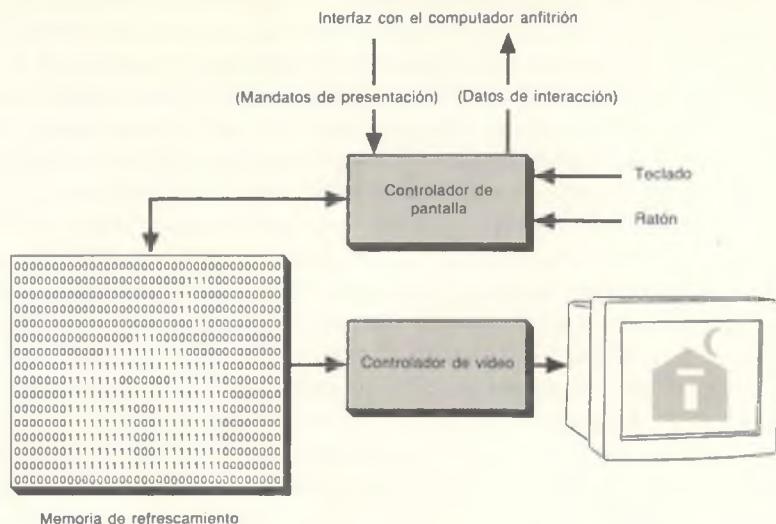


Figura 1.8 Arquitectura de una pantalla de trama.

de refrescamiento únicamente es un pedazo de la memoria de la UCP que puede leer el subsistema de dibujo de imágenes (con frecuencia llamado **controlador de vídeo**) que produce la imagen real en la pantalla.

La imagen completa en una pantalla de barrido se forma a partir de la **trama**, que es un conjunto de **líneas de rastreo** horizontales, cada una de las cuales constituye una fila de pixeles individuales; de esta manera, la trama se almacena como una matriz de pixeles que representa toda el área de la pantalla. El controlador de vídeo rastrea la imagen secuencialmente, de arriba hacia abajo y luego de nuevo hacia arriba (como se muestra en la figura 1.9). En cada pixel la intensidad del haz se ajusta para reflejar la intensidad del pixel; en los sistemas a color se controlan tres haces correspondientes a los colores primarios (rojo, verde y azul) de acuerdo con lo especificado por los tres colores componentes de cada valor de pixel (véanse los Caps. 4 y 11). En la figura 1.10, parte (a), se muestra la diferencia entre el rastreo aleatorio y el de barrido para la presentación de un dibujo bidimensional de líneas de una casa. En la parte (b), los arcos vectoriales están anotados con puntas de flecha para indicar la desviación aleatoria del haz. Las líneas discontinuas denotan la desviación del haz, el cual no se enciende (se **suprime**) para que no se dibuje el vector. En la parte (c) se muestra la casa no rellena presentada con rectángulos, polígonos y arcos, mientras que en la parte (d) aparece una versión rellena. Observe la apariencia serrada de las líneas y los arcos en las imágenes de barrido de trama de las partes (c) y (d); pronto analizaremos este artefacto visual.

En un sistema de barrido es necesario almacenar explícitamente todo el arreglo de, digamos, 1024 por 1024 pixeles, por lo que, a principios de la década

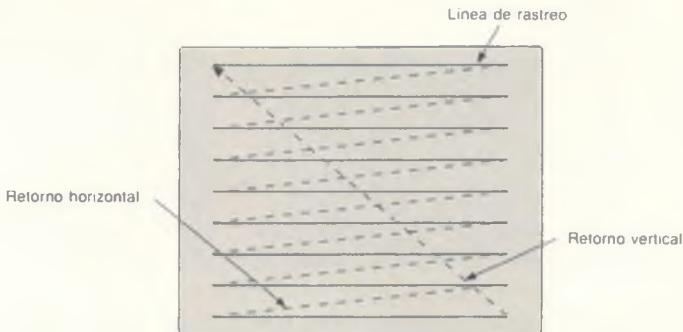
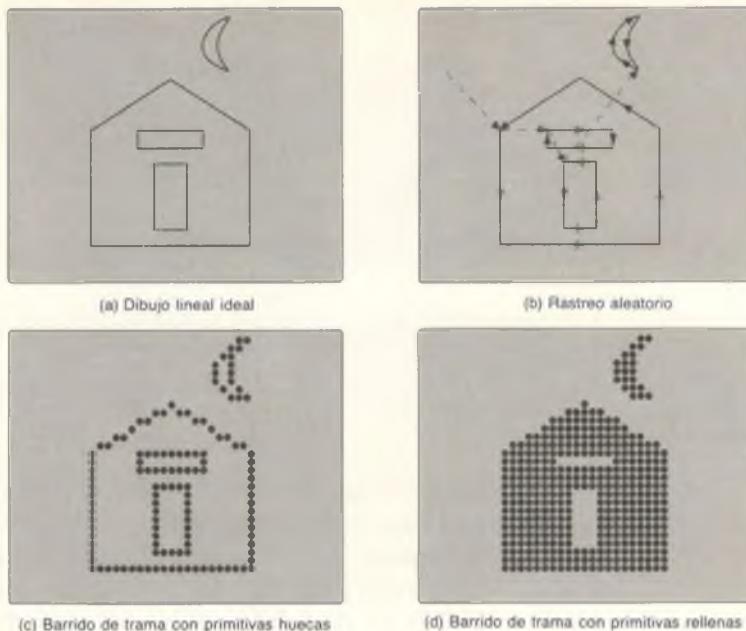


Figura 1.9 Barrido de trama.

de 1970, la disponibilidad de memoria de acceso aleatorio (RAM, *random-access memory*) de bajo costo para los mapas de bits fue el punto clave para que los gráficos de trama se convirtieran en la tecnología de hardware dominante. Los CRT de dos niveles (también conocidos como **monocromáticos**) dibujan imágenes en blanco y negro o en verde y negro; algunas pantallas de plasma usan negro y naranja. Los mapas de bits de dos niveles contienen un solo bit por pixel y todo el mapa de bits de una pantalla de 1024 por 1024 pixeles ocupa únicamente  $2^{20}$  bits, alrededor de unos 128 000 bytes. Los sistemas de color más sencillos usan ocho bits por pixel, lo que permite utilizar de manera simultánea 256 colores; los sistemas más costosos tienen 24 bits por pixel, permitiendo elegir entre 16 millones de colores; y también existen memorias de refrescamiento de 32 bits por pixel y resolución de pantalla de 1280 por 1024 pixeles, disponibles incluso en computadores personales. De estos 32 bits, 24 se dedican a la representación del color y los otros ocho a propósitos de control, como veremos en el capítulo 4. Un sistema de color típico de 1280 por 1024 con 24 bits por pixel requiere 3.75 MB de RAM, lo que representa un costo bajo según los estándares actuales. En un sentido estricto, el concepto **mapa de bits** se aplica exclusivamente a los sistemas de dos niveles de un bit por pixel; en el caso de sistemas de varios bits por pixel usaremos el término **mapa o arreglo bidimensional de pixeles ( pixmap)**, más general. Ya que en la usanza popular un mapa de pixeles se refiere tanto al contenido de la memoria de refrescamiento como a la propia memoria, usaremos el término **memoria gráfica** para hacer referencia a la memoria en sí.

Las principales ventajas de los gráficos de barrido con respecto a los **gráficos vectoriales** incluyen su menor costo y la capacidad para presentar áreas llenas con colores sólidos o patrones, una forma especialmente rica para comunicar información e indispensable para obtener imágenes realistas de objetos tridimensionales.

La principal desventaja de las imágenes de barrido, en comparación con los sistemas vectoriales, se debe a la naturaleza discreta de la representación con



**Figura 1.10** Comparación entre el rastreo aleatorio y el barrido de trama. Simbolizamos la pantalla como un rectángulo redondeado relleno con un matiz gris claro que denota el fondo blanco; la imagen se dibuja en negro sobre este fondo.

pixeles. En primer lugar, las primitivas como líneas y polígonos se especifican en función de sus puntos extremos (vértices) y deben discretizarse a sus pixeles componentes en la memoria gráfica. El término **discretización** se deriva de la notación en la cual el programador especifica las coordenadas de punto final o vértice en el modo de rastreo aleatorio, información que el sistema debe reducir a pixeles para la presentación en modo de barrido de trama. Por lo general, la discretización se efectúa con software en computadores personales y en las estaciones de trabajo más sencillas, donde la UCP es responsable de todos los gráficos.

Otra desventaja de los sistemas de barrido se debe a la naturaleza de la trama. Un sistema vectorial puede dibujar líneas continuas y regulares (incluso curvas regulares) esencialmente de cualquier punto del CRT a otro; en un sistema de barrido, la presentación de primitivas matemáticas curvas como círculos y elipses sólo se puede lograr aproximándolas con pixeles en el arreglo de trama. Esta aproximación puede ocasionar el conocido problema de “escalonamiento” que se ilustra en las figuras 1.10(c) y (d). Este artefacto visual es una manifestación de un error de muestreo conocido como **artefacto de discretización (aliasing)** en la teoría del procesamiento de señales; estas situaciones se presentan cuando la función de una variable continua que manifiesta cambios abruptos en la intensidad, es aproximada con muestras discretas. En el campo de la graficación por computador moderna existe la preocupación por desarrollar técnicas

de eliminación del artefacto de discretización en sistemas de escala de grises o en color. Estas técnicas especifican cambios graduales en la intensidad de los pixeles vecinos en las orillas de las primitivas, en lugar de asignarles únicamente la intensidad máxima o de cero; en el capítulo 3 se analiza con detalle este importante tema.

### 1.2.2 Tecnología de entrada

La tecnología de entrada también ha mejorado en forma notable con el paso de los años. El burdo y frágil lápiz de luz de los sistemas vectoriales ha sido reemplazado por el popular ratón (desarrollado inicialmente por el pionero de la automatización de oficinas Doug Engelbart a mediados de los años sesenta [ENG68]); la tableta de datos; y el tablero transparente, sensible al tacto, montado sobre la pantalla. También son cada vez más comunes los dispositivos de entrada más complejos que no sólo proporcionan localidades ( $x, y$ ) en la pantalla, sino también valores tridimensionales o incluso de más dimensiones (grados de libertad), como se verá en el capítulo 8. La comunicación por audio también tiene mucho potencial, ya que permite la entrada sin tener que usar las manos y la salida natural de instrucciones sencillas, realimentación, etcétera. Con los dispositivos de entrada estándar, el usuario puede especificar operaciones o componentes de la imagen tecleando o dibujando información nueva o bien apuntando hacia la información que existe en la pantalla. Para esta interacción no se requieren conocimientos de programación y hay que usar muy poco el teclado: el usuario elige las opciones con sólo seleccionar botones o iconos de los menús, responde preguntas marcando opciones o tecleando unos cuantos caracteres en un formulario, coloca copias de símbolos predefinidos en la pantalla, dibuja indicando puntos extremos consecutivos que se conectarán con líneas rectas o interpoladas por curvas regulares, pinta moviendo el cursor por la pantalla y rellena áreas cerradas limitadas por polígonos o pinta contornos con tonos de grises, con colores o con diversos patrones.

### 1.2.3 Transportabilidad de software y estándares para gráficos

Como hemos visto, los constantes avances en la tecnología de hardware han hecho posible la evolución de las pantallas gráficas de los dispositivos de salida especiales y únicos a la interfaz estándar entre el usuario y el computador. Nos podríamos preguntar si el software ha seguido el mismo ritmo. Por ejemplo, ¿hasta qué punto hemos resuelto las dificultades iniciales que se presentaban con los complejos, costosos y complicados sistemas gráficos y el software de aplicación? Hemos avanzado de los paquetes de bajo nivel, dependientes del dispositivo, que proporcionaban los fabricantes para sus dispositivos de presentación particulares, a paquetes de mayor nivel, independientes del dispositivo. Estos paquetes pueden dirigir una amplia variedad de dispositivos de presentación, desde impresoras láser y graficadores hasta grabadoras de película fotográfica y pantallas interactivas de alto rendimiento. El propósito principal de usar un paquete independiente del dispositivo junto con un lenguaje de

programación de alto nivel es promover la transportabilidad del programa de aplicación. El paquete ofrece esta transportabilidad en una forma muy similar a la de un lenguaje de alto nivel, independiente de la máquina (como FORTRAN, Pascal o C): aislando al programador de la mayoría de las particularidades de la máquina y ofreciendo características de lenguaje fáciles de implantar en diversos tipos de procesadores.

A mediados de la década de 1970 se hizo evidente la necesidad de estándares para estos gráficos independientes de los dispositivos, lo que culminó con la especificación de un **3D Core Graphics System** (sistema núcleo de gráficos tridimensionales, conocido en forma abreviada como Core), producido por un comité de **ACM SIGGRAPH**<sup>1</sup> en 1977 [GSPC77] y refinado en 1979 [GSPC79].

La especificación Core cumplió con la función que se pretendía como especificación base. No sólo tuvo muchas implantaciones, sino que se usó además en proyectos oficiales (gubernamentales) de estándares tanto en **ANSI (American National Standards Institute, Instituto Nacional Estadounidense de Estándares)** como en **ISO (International Standards Organization, Organización Internacional de Estándares)**. La primera especificación gráfica estandarizada oficialmente fue **GKS, Graphical Kernel System** (Sistema de Kernel Gráfico) [ANSI85], una versión elaborada y depurada de Core que, a diferencia de éste, estaba limitada a dos dimensiones. En 1988, **GKS-3D** [INTE88], una extensión tridimensional de GKS, se convirtió en estándar oficial, al igual que un sistema gráfico mucho más elaborado pero también más complejo de nombre **PHIGS (Programmer's Hierarchical Interactive Graphics System, Sistema Gráfico Interactivo Jerárquico para Programadores)** [ANSI88]. GKS apoya el agrupamiento de primitivas lógicamente relacionadas —como líneas, polígonos y cadenas de caracteres— y sus atributos en colecciones llamadas **segmentos**; estos segmentos no se pueden anidar. PHIGS, como lo sugiere su nombre, sí permite agrupamientos jerárquicos anidados de subprimitivas tridimensionales, llamadas **estructuras**. En PHIGS todas las primitivas, incluyendo las invocaciones a estructuras, están sujetas a transformaciones geométricas (escalamiento, rotación y traslación) para lograr el movimiento dinámico. PHIGS también apoya una base de datos retenida de estructuras que el programador puede editar de manera selectiva. PHIGS actualiza automáticamente la pantalla cuando se altera la base de datos. Además, PHIGS se ha ampliado con un conjunto de características para la moderna presentación seudorrealista<sup>2</sup> de objetos en pantallas de trama; esta

<sup>1</sup> SIGGRAPH es el acrónimo de Special Interest Group on Graphics (Grupo de Interés Especial en Gráficos), uno de los grupos profesionales que integran la Association of Computing Machinery (ACM, Asociación de Maquinaria de Computación). ACM es una de las dos principales sociedades para profesionales de la computación; la otra es la IEEE Computer Society (Sociedad de Computación del IEEE). SIGGRAPH publica una revista de investigaciones y patrocina una conferencia anual que ofrece presentaciones de artículos de investigación en el campo y una exhibición de equipo. La IEEE Computer Society también publica una revista de investigación gráfica.

<sup>2</sup> Una *presentación seudorrealista* es aquella que simula las leyes ópticas simples que describen cómo los objetos reflejan la luz. La *presentación fotorealista* usa aproximaciones más precisas de la manera en que los objetos reflejan y refractan la luz; estas aproximaciones requieren más cálculos pero producen imágenes con calidad mucho más cercana a la fotográfica.

extensión se denomina **PHIGS+** [PHIG88] antes de su presentación a ANSI/ISO y **PHIGS PLUS** en ISO [PHIG92]. Las implantaciones de PHIGS son paquetes muy grandes, debido a las numerosas características y a la complejidad de la especificación. Las implantaciones de PHIGS, y sobre todo de PHIGS PLUS, se ejecutan con mayor rapidez cuando hay apoyo de hardware para sus características de transformación, recorte y presentación.

Además de los estándares oficiales promulgados por las instituciones de estándares nacionales, internacionales o de organizaciones profesionales, también hay estándares no oficiales. Estos estándares, llamados industriales o *de facto*, son desarrollados, promovidos y autorizados por compañías individuales o por consorcios de compañías y universidades. Entre los estándares gráficos de la industria que más se conocen están PostScript, de Adobe; OpenGL, de Silicon Graphics; HOOPS, de Ithaca Software; y X Window System y sus extensiones de protocolo cliente-servidor para gráficos tridimensionales, PEX (véase el Cap. 7), ambos de X-Consortium, encabezado por MIT. Los estándares industriales pueden prevalecer más y por ende son comercialmente más importantes que los oficiales, ya que se pueden actualizar con mayor rapidez, sobre todo aquellos que son un producto comercial clave de una compañía y por lo tanto cuentan con más recursos.

En este libro se analizan con detalle los estándares del software gráfico. El primero que se estudia, en el capítulo 2, es **SRGP** (*Simple Raster Graphics Package*, Paquete gráfico de barrido sencillo), que tiene características del popular paquete entero de gráficos de barrido QuickDraw de Apple [ROSE85] y de X Window System de MIT [SCHE88] para las salidas, así como de GKS y PHIGS para las entradas. Después de ver las aplicaciones más sencillas en este paquete gráfico de trama de bajo nivel, estudiaremos los algoritmos de discretización y de recorte que usan esos paquetes para generar imágenes de primitivas en la memoria gráfica. Más adelante, después de establecer en el capítulo 5 una base matemática para las transformaciones geométricas bidimensionales y tridimensionales, y en el capítulo 6 los fundamentos para la vista tridimensional paralela y de perspectiva, analizaremos un paquete mucho más poderoso llamado **SPHIGS** (*Simple PHIGS*, PHIGS sencillo) en el capítulo 7. SPHIGS es un subconjunto de PHIGS que opera con primitivas definidas en un sistema tridimensional de coordenadas de mundo, abstracto, de punto flotante e independiente de cualquier tipo de tecnología de presentación, que apoya varias de las características sencillas de PHIGS PLUS. Hemos orientado nuestro análisis hacia PHIGS y PHIGS PLUS porque creemos que tendrán mucha mayor influencia en los gráficos tridimensionales interactivos que GKS-3D, especialmente por la disponibilidad de hardware que permite efectuar transformaciones y presentaciones en tiempo real de imágenes seudorrealistas.

## 1.3 Ventajas de la graficación interactiva

Los gráficos ofrecen uno de los medios más naturales para la comunicación con un computador, ya que nuestras altamente desarrolladas habilidades de

reconocimiento de patrones bidimensionales y tridimensionales nos permiten percibir y procesar datos pictóricos con rapidez y eficiencia. En muchos procesos de diseño, implantación y construcción que se realizan hoy día, la información que nos pueden brindar las imágenes es virtualmente indispensable. La visualización científica se convirtió en un campo importante a finales de la década de 1980, cuando los científicos e ingenieros se percataron de que no podían interpretar las enormes cantidades de datos producidos por los supercomputadores sin resumirlos y destacar las tendencias y los fenómenos usando distintas representaciones gráficas.

La graficación interactiva por computador representa el medio más importante para producir imágenes desde la invención de la fotografía y la televisión; además, tiene la ventaja de que, con el computador, no sólo nos permite crear imágenes de objetos concretos y reales, sino también de objetos abstractos, sintéticos y de datos que no tienen geometría inherente, como son los resultados de un sondeo. Así mismo, no estamos limitados a usar imágenes estáticas. Aunque este tipo de imágenes constituye una buena forma de comunicar información, las imágenes con variaciones dinámicas muchas veces son más efectivas, sobre todo para fenómenos variables en el tiempo, tanto reales (p. ej., la desviación del ala de una aeronave durante un vuelo supersónico o el desarrollo del rostro humano de la niñez a la vejez) como abstractos (p. ej., tendencias de crecimiento, como la de la utilización de energía nuclear en Estados Unidos o el movimiento de población de las ciudades a los suburbios y de regreso a las ciudades).

Con la **dinámica de movimiento**, los objetos pueden moverse o girar con respecto a un observador estacionario. También es posible que los objetos permanezcan estacionarios y que el observador se mueva alrededor de ellos, seleccione la porción que deseé visualizar y se acerque o aleje para obtener más o menos detalles, como si viera por el visor de una cámara de video. En muchos casos se mueven tanto los objetos como la cámara. Un ejemplo típico es el simulador de vuelo (Ilusts. en color 4a y 4b), el cual combina una plataforma mecánica que sostiene una réplica de la cabina con pantallas en lugar de ventanas. Los computadores controlan el movimiento de la plataforma, los indicadores y el mundo simulado de los objetos móviles y estacionarios entre los cuales navega el piloto. Los parques de diversiones ofrecen juegos de simulación de movimiento a través de paisajes terrestres y extraterrestres simulados. También se ofrecen juegos de video de destreza basados en gráficos y simuladores de conducción de automóviles de carreras, que aprovechan la dinámica del movimiento (véase la Ilust. en color 5).

La **dinámica de actualización** es el cambio real de la forma, el color u otras propiedades de los objetos que se visualizan o modelan. Por ejemplo, un sistema puede presentar las deformaciones de la estructura de un avión en vuelo o los cambios de estado en el diagrama de bloques de un reactor nuclear como respuesta a la manipulación que hace el operador de las representaciones gráficas de los diversos mecanismos de control. Cuanto más gradual sea el cambio, más realista y significativo es el resultado.

De esta manera, la graficación interactiva por computador permite una extensa y diversa interacción entre el usuario y el computador. Estas interacciones

mejoran notablemente nuestra capacidad para comprender datos, percibir tendencias y visualizar objetos reales o imaginarios.

## 1.4 Marco conceptual para la graficación interactiva

El marco conceptual de alto nivel que se presenta en la figura 1.11 nos puede servir para describir casi cualquier sistema gráfico interactivo. En el nivel de hardware (que no se presenta explícitamente en el diagrama), un computador recibe su entrada de dispositivos interactivos y envía las imágenes a un dispositivo de presentación. El software tiene tres componentes. El primero, el **programa de aplicación**, crea, almacena y lee del segundo componente, el **modelo de aplicación**, que representa los datos u objetos que se mostrarán en la pantalla. El programa de aplicación también maneja las entradas del usuario. Este programa produce vistas enviando al tercer componente, el **sistema gráfico**, una serie de mandatos gráficos de salida que contienen una descripción geométrica detallada de *qué* se verá y los atributos que indican *cómo* deben aparecer los objetos. El sistema gráfico es el responsable de la producción real de la imagen a partir de las descripciones detalladas, así como de pasar las entradas del usuario al programa de aplicación para que se procesen.

Así, el sistema gráfico es un intermediario entre el programa de aplicación y el hardware de presentación, que lleva a cabo una transformación de salida de los objetos en el modelo de aplicación a una vista del modelo. Simétricamente, efectúa una transformación de entrada de las acciones del usuario a entradas para el programa de aplicación, por medio de las cuales la aplicación hará

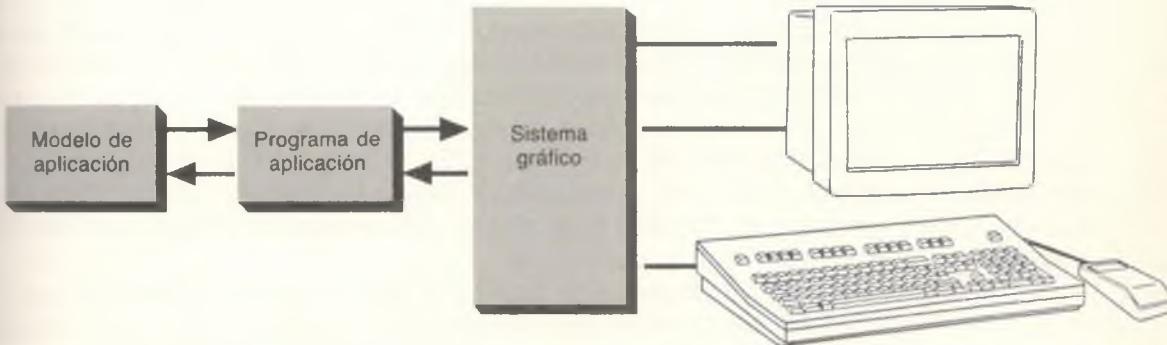


Figura 1.11 Marco conceptual para la graficación interactiva.

cambios al modelo o a la imagen. La tarea fundamental del diseñador de un programa de aplicación de la graficación interactiva es especificar qué clases de elementos de datos u objetos se generarán y representarán pictóricamente, y cómo interactuarán el usuario y el programa de aplicación para crear y modificar el modelo y su representación visual. La mayor parte de las tareas del programador tiene que ver con la creación y edición del modelo, así como con el manejo de la interacción con el usuario, no con la creación de las vistas, ya que esta tarea es realizada por el sistema gráfico.

### 1.4.1 Modelado de aplicaciones

El modelo de aplicación captura todos los datos y objetos, así como las relaciones entre ellos que son importantes para las partes de presentación e interacción del programa de aplicación y cualquier módulo de postprocesamiento no gráfico. Como ejemplos de este tipo de módulos están el análisis del comportamiento transitorio de un circuito o de las tensiones en el ala de un avión, la simulación de un modelo de población o de un sistema meteorológico y los cálculos de precios para un edificio. En las clases de aplicaciones tipificadas por los programas de dibujo como MacPaint y PCPaint, el propósito del programa es producir una imagen permitiendo que el usuario establezca o modifique los pixeles individualmente. En este caso no se requiere un modelo de aplicación explícito, ya que el dibujo es el medio y el fin, y el mapa de bits o de pixeles que se presenta sirve como modelo de aplicación.

Sin embargo, es más usual que exista un modelo de aplicación identificable que representa los objetos de la aplicación a través de una combinación de los datos y una descripción de procedimientos independiente de un dispositivo de presentación en particular. Las descripciones de procedimientos se usan, por ejemplo, para definir fractales, como se describe en la sección 14.11. Un modelo de datos puede ser tan rudimentario como un arreglo de puntos de datos o tan complejo como una lista ligada que representa la estructura de datos de una red o una base de datos relacional que almacena un conjunto de relaciones. Con frecuencia hablaremos de almacenar el **modelo de aplicación** en la **base de datos de la aplicación**; usaremos estos términos en forma indistinta. Los modelos generalmente almacenan descripciones de primitivas (puntos, líneas, curvas y polígonos en dos o tres dimensiones y superficies tridimensionales poliédricas y de forma libre) que definen la forma de los componentes del objeto; **atributos** del objeto, como el estilo de las líneas, el color o la textura de la superficie; y relaciones de **conectividad** y datos de ubicación que describen cómo se enlazan los componentes.

Los datos geométricos en el modelo de aplicación muchas veces están acompañados por información textual o numérica de propiedades no geométricas, la cual es útil para un programa de postprocesamiento o para el usuario interactivo. Como ejemplos de estos datos en las aplicaciones CAD están los datos de manufactura; datos de precios y proveedores; propiedades térmicas, mecánicas, eléctricas o electrónicas; y tolerancias mecánicas o eléctricas.

## 1.4.2 Presentación del modelo

El programa de aplicación crea el modelo como resultado de cálculos previos, como se hace en una simulación científica o de ingeniería en un supercomputador, o como parte de una sesión interactiva en el dispositivo de presentación, durante la cual el usuario guía el proceso de construcción, paso a paso, eligiendo los componentes y los datos geométricos y no geométricos. El usuario puede solicitar en cualquier momento al programa de aplicación que muestre una vista del modelo que ha creado hasta entonces. (La palabra *vista* se usa de manera intencional, tanto en el sentido de una presentación visual de las propiedades geométricas de los objetos que se modelan, como en el sentido técnico de base de datos que se refiere a una presentación bidimensional de las propiedades de un subconjunto del modelo.)

Los modelos son específicos para la aplicación y se crean independientemente de cualquier sistema de presentación. Por consiguiente, el programa de aplicación debe convertir una descripción de la porción del modelo que se visualizará a partir de la representación geométrica interna (ya sea que esté almacenada de manera explícita en el modelo o derivada sobre la marcha), en las llamadas a procedimientos o los mandatos que utilice el sistema gráfico para crear una imagen. Este proceso de conversión tiene dos fases. Primero, el programa de aplicación recorre la base de datos de aplicación que almacena el modelo, para extraer las partes que se visualizarán con base en criterios de consulta o selección. Después, los datos o la geometría extraídos, junto con los atributos, se convierten a un formato que pueda enviarse al sistema gráfico. Los criterios de selección pueden ser geométricos (p. ej., la parte del modelo que se verá se ha desplazado por medio del equivalente gráfico de la operación de giro o de acercamiento/alejamiento de una cámara) o pueden ser similares a los criterios tradicionales de consulta de base de datos.

Los datos extraídos durante el recorrido de la base de datos deben ser geométricos o convertirse a datos geométricos; los datos se pueden describir al sistema gráfico en función tanto de primitivas que puede presentar directamente el sistema como de los atributos que controlan la apariencia de dichas primitivas. Las primitivas de presentación por lo general equivalen a las almacenadas en los modelos geométricos: líneas, rectángulos, polígonos, círculos, elipses y texto en dos dimensiones; y polígonos, poliedros y texto en tres dimensiones.

El sistema gráfico suele consistir en un conjunto de subrutinas de salida que corresponden a las diversas primitivas, atributos y otros elementos. Todos ellos se agrupan en un paquete o biblioteca de subrutinas gráficas que puede invocarse con lenguajes de alto nivel como C, Pascal o LISP. El programa de aplicación especifica primitivas geométricas y atributos a estas subrutinas, las cuales dirigen el dispositivo de presentación específico y hacen que presente la imagen. Así como los sistemas convencionales de E/S crean unidades lógicas de E/S para aislar al programador de la aplicación de los confusos detalles de los manejadores de hardware y los dispositivos, los sistemas gráficos crean un dispositivo lógico de presentación. Esta abstracción del dispositivo de presentación

tiene que ver tanto con la salida de imágenes como con la interacción a través de dispositivos de entrada. Por ejemplo, el ratón, la tableta de datos, la pantalla sensible al tacto, la palanca de mandos bidimensional o la bola rastreadora (*trackball*) puede tratarse como dispositivo lógico de entrada **localizador** que indica una posición ( $x, y$ ) en la pantalla. El programa de aplicación puede pedir entonces al sistema gráfico que **muestre** los dispositivos de entrada (es decir, que solicite sus valores actuales) o esperar en un punto especificado a que se genere un **evento** cuando el usuario active un dispositivo al cual se espera.

### 1.4.3 Manejo de la interacción

El esquema típico del programa de aplicación para manejar la interacción es el **ciclo dirigido por eventos**. Éste puede visualizarse como una máquina de estados finitos con un estado central de espera y transiciones a otros estados ocasionadas por eventos indicados por el usuario. El procesamiento de un mandato puede requerir ciclos de evento anidados del mismo formato, con sus propios estados y transiciones de entrada. Así mismo, un programa de aplicación puede muestrear dispositivos de entrada (como el localizador) solicitando sus valores en un instante cualquiera; el programa usa entonces esos datos como entrada para un procedimiento de procesamiento que también cambia el estado del programa de aplicación, la imagen o la base de datos. El ciclo dirigido por eventos está caracterizado por el siguiente esquema en pseudocódigo:

```

generar pantalla inicial, derivada en forma apropiada del modelo de aplicación
do {
    permitir la selección de mandatos u objetos
    /* El programa hace una pausa indefinida en un estado de espera,
       hasta que actúe el usuario*/
    wait la selección del usuario
    switch ( dependiendo de la selección ) {
        procesar la selección hasta concluir el mandato o procesar el mandato
        terminado, actualizando el modelo y la pantalla según sea necesario
    }
}
while ( !salir ) /* El usuario no ha seleccionado la opción "salir" */

```

Examinemos con mayor detalle la reacción de la aplicación a las entradas. El programa de aplicación suele responder a la interacción del usuario en una de dos formas. Primero, la acción del usuario puede requerir sólo que se actualice la pantalla; por ejemplo, el sistema puede responder realizando un objeto seleccionado o presentando un nuevo menú de opciones. Entonces la aplicación únicamente tiene que actualizar su estado interno y llamar al paquete gráfico para que actualice la pantalla; no tiene que alterar la base de datos. Por otra parte, si el usuario solicita un cambio al modelo —por ejemplo añadir o eliminar un componente— la aplicación debe actualizar el modelo y luego invocar el paquete gráfico para actualizar la pantalla a partir del modelo. Para esto, o bien se

vuelve a recorrer todo el modelo para regenerar la imagen desde cero, o bien la pantalla se actualiza en forma selectiva usando algoritmos más elaborados de actualización incremental. Es importante comprender que no es posible efectuar ningún cambio importante en los objetos de la pantalla si no hay un cambio correspondiente en el modelo. La pantalla es de hecho la ventana al computador, ya que el usuario, por lo general, no sólo manipula una imagen sino el modelo que existe en forma literal y figurada detrás de la imagen. El modelo y la imagen sólo son idénticos en las aplicaciones de dibujo y de tratamiento de imágenes. Por consiguiente, la tarea de la aplicación es interpretar la entrada del usuario. El sistema gráfico no tiene responsabilidad en lo que se refiere a la construcción o la modificación del modelo, ya sea inicialmente o como respuesta a la interacción con el usuario; su único trabajo es crear imágenes a partir de descripciones geométricas y transmitir los datos de entrada del usuario.

El modelo del ciclo de eventos, aunque es fundamental en la práctica actual de la graficación por computador, está limitado por el hecho de que el diálogo entre el usuario y el computador es un modelo *secuencial*, alternante, de acciones de usuario y reacciones del computador. En el futuro quizás veamos más conversaciones *paralelas*, en las cuales se realizan entradas y salidas simultáneas a través de canales de comunicación múltiples, por ejemplo gráficos y verbales. Falta mucho por desarrollar en lo que se refiere a los formalismos, por no mencionar las construcciones de los lenguajes de programación, necesarias para estas conversaciones en formato libre; por ende, no las analizaremos con más detalle en este libro.

## RESUMEN

Las interfaces gráficas han sustituido a las interfaces textuales como el medio habitual para la interacción usuario-computador. Los gráficos también se han convertido en una tecnología fundamental para comunicar ideas, datos y tendencias en la mayoría de las áreas comerciales, científicas, de ingeniería y educativas. Con los gráficos podemos crear mundos artificiales (o virtuales), constituyendo cada uno de ellos un área de exploración para el estudio de objetos y fenómenos en una forma natural e intuitiva que aproveche nuestras altamente desarrolladas habilidades de reconocimiento de patrones visuales.

Hasta finales de la década de 1980, la mayor parte de las aplicaciones de la graficación por computador tenían que ver con objetos bidimensionales; las aplicaciones tridimensionales eran relativamente raras, ya que el software de este tipo es de suyo más complejo que el bidimensional y porque se requiere mucho poder de cálculo para producir imágenes seudorrealistas. Por lo tanto, hasta hace poco la interacción en tiempo real entre el usuario y los modelos tridimensionales e imágenes seudorrealistas sólo era factible en costosas estaciones de trabajo de alto rendimiento que usaban hardware gráfico de propósito especial. El espectacular avance de la tecnología de semiconductores VLSI, responsable del advenimiento de los microprocesadores y la memoria de bajo

costo, fue lo que condujo, a principios de los años ochenta, a la creación de interfaces con computadores personales basadas en gráficos bidimensionales de mapa de bits. La misma tecnología que ha permitido crear, menos de una década después, subsistemas con unos cuantos chips que llevan a cabo animaciones tridimensionales en tiempo real con imágenes sombreadas en color de objetos complejos, por lo general descritos por miles de polígonos. Estos subsistemas se pueden añadir como aceleradores tridimensionales a las estaciones de trabajo o incluso a los computadores personales que emplean microprocesadores comerciales. Es obvio que el explosivo crecimiento de las aplicaciones tridimensionales será paralelo al crecimiento actual de las aplicaciones bidimensionales. Así mismo, las áreas como la presentación fotorrealista, que en una época se consideraban como algo exótico, ahora forman parte de la tecnología de alto nivel y son comunes en el software gráfico y cada vez con mayor frecuencia en el hardware gráfico.

Gran parte del trabajo para crear una comunicación gráfica efectiva, ya sea en dos o en tres dimensiones, se relaciona con el modelado de los objetos cuyas imágenes queremos producir. El sistema gráfico actúa como intermediario entre el modelo de aplicación y el dispositivo de salida. El programa de aplicación es responsable de crear y actualizar el modelo con base en la interacción con el usuario; el sistema gráfico lleva a cabo la parte más rutinaria y mejor comprendida del trabajo, creando vistas de objetos y transmitiendo los eventos del usuario a la aplicación. La literatura cada vez más abundante sobre los diversos tipos de modelado basado en física nos indica que la graficación está evolucionando para incluir algo más que la presentación y el manejo de la interacción. Las imágenes y las animaciones ya no son simples ilustraciones en la ciencia y la ingeniería: se han convertido en parte del contenido científico y de ingeniería y han influido en la forma en que los científicos y los ingenieros llevan a cabo su trabajo cotidiano.

## Ejercicios

1.1 Liste los programas de graficación interactiva que use de manera rutinaria en su trabajo: escritura, cálculos, graficación, programación, depuración, etcétera. ¿Cuáles de ellos trabajarían casi con la misma eficacia usando una terminal exclusivamente alfanumérica? ¿Cuáles serían casi inútiles si no tuvieran capacidad gráfica? Explique sus respuestas.

1.2 Muchas veces se emplean los términos “apariencia” y “tacto” al hacer referencia a la interfaz con el usuario de los programas gráficos. Liste los componentes principales —como iconos, ventanas, barras de desplazamiento y menús— que definen la apariencia de la interfaz gráfica de su programa favorito de procesamiento de texto o administración de ventanas. Liste también las capacidades gráficas que requieren estos componentes. ¿Qué oportunidades pueden tener la aplicación de color y las ilustraciones tridimensionales en lo referente

a la apariencia? Por ejemplo, ¿cómo podría el término “oficina saturada” constituir una metáfora espacial más poderosa para organizar y acceder a información que el término “escritorio desordenado”?

1.3 Use como base el ejercicio 1.2 y describa las oportunidades que según su opinión puedan tener los iconos dinámicos para mejorar o incluso reemplazar los iconos estáticos de las metáforas de escritorio actuales.

1.4 Divida su aplicación gráfica preferida en sus módulos principales, usando como guía el modelo conceptual de la figura 1.11. ¿Cuánto de la aplicación tiene que ver directamente con los gráficos? ¿Cuánto con la creación y el mantenimiento de las estructuras de datos? ¿Cuánto más con los cálculos, como la simulación?

1.5 Los términos *simulación* y *animación* suelen utilizarse en conjunto e incluso indistintamente en la graficación por computador. Este uso es natural cuando visualizamos los cambios de comportamiento (o estructurales) en el tiempo que se presentan en un sistema físico o abstracto. Elabore tres ejemplos de sistemas que podrían beneficiarse con estas visualizaciones. Especifique la forma que tendrían las simulaciones y cómo se podrían ejecutar. Proporcione un ejemplo que distinga una *simulación* de una *animación*.

1.6 Como variante del ejercicio 1.5, elabore un diseño de alto nivel de una simulación para explorar en forma gráfica un área científica, matemática o de ingeniería no trivial. Analice cómo operarían las secuencias de interacción y qué recursos deben estar disponibles para los experimentos del usuario.

1.7 Sin consultar el capítulo 3, elabore un algoritmo directo para discretizar una línea en el primer cuadrante, cuya pendiente sea menor o igual que  $45^\circ$ .

1.8 La discretización puede ocasionar serios problemas al generar artefactos visuales poco agradables o incluso confusos. Indique algunas situaciones en las cuales estos artefactos sean importantes y otras en las cuales no lo sean. Analice varias formas de minimizar los efectos del artefacto de discretización y explique cuál sería el “costo” de estos remedios.



En el capítulo 1 vimos que las pantallas vectoriales y de barrido utilizan dos tecnologías de hardware sustancialmente diferentes para crear imágenes. Las pantallas de barrido son la tecnología dominante porque permiten varias características esenciales para la mayoría de las aplicaciones modernas. En primer lugar, en las pantallas de barrido se pueden llenar áreas con un color uniforme o con la repetición de un patrón en dos o más colores; las pantallas vectoriales a lo sumo pueden simular el relleno de áreas con secuencias muy cercanas de vectores paralelos. En segundo término, las pantallas de barrido almacenan imágenes en una forma que permite la manipulación a un nivel muy fino: es posible leer o escribir pixeles individuales y se pueden copiar o mover porciones arbitrarias de la imagen.

El primer paquete gráfico que analizaremos, **SRGP** (*Simple Raster Graphics Package*, Paquete gráfico simple de barrido) es independiente del dispositivo y explota las capacidades de las tramas. El repertorio de primitivas (líneas, rectángulos, círculos, elipses y cadenas de texto) de SRGP es similar al del popular paquete gráfico de trama Macintosh QuickDraw y al del paquete Xlib de X Window System. Por otra parte, las características de manejo de interacción de SRGP constituyen un subconjunto de las de SPHIGS, el paquete gráfico de mayor nivel que se usa para presentar primitivas tridimensionales (véase el Cap. 7). SPHIGS (*Simple PHIGS*, PHIGS simple) es un dialecto simplificado del paquete gráfico estándar PHIGS (*Programmer's Hierarchical Interactive Graphics System*, Sistema gráfico interactivo jerárquico para programadores), diseñado tanto para hardware de barrido como vectorial. Aunque SRGP y SPHIGS se escribieron específicamente para este texto, siguen las tendencias de la mayoría

de los paquetes gráficos comunes, y casi todo lo que aprenda en este libro podrá aplicarlo de inmediato a los paquetes comerciales. En este libro presentamos ambos paquetes (SRGP y SPHIGS); si desea una descripción más completa, deberá consultar los manuales de referencia que se distribuyen con los paquetes de software.

Iniciamos nuestro análisis de SRGP examinando las operaciones que llevan a cabo las aplicaciones para dibujar en la pantalla: las especificaciones de primitivas y de los atributos que afectan su imagen. (Como las impresoras gráficas presentan la información esencialmente de la misma manera que las pantallas de barrido, no tendremos que preocuparnos por ellas hasta que veamos con mayor detenimiento el hardware, en el capítulo 4.) Después veremos cómo lograr que las aplicaciones sean interactivas usando las funciones de entrada de SRGP. A continuación analizaremos la utilidad de la manipulación de pixeles, disponible sólo en pantallas de barrido. Concluimos con el estudio de algunas limitantes de los paquetes gráficos enteros de barrido como SRGP.

Aunque nuestro análisis de SRGP supone que éste controla toda la pantalla, el paquete se diseñó para operar en ambientes de ventanas (véase el Cap. 10), en cuyo caso controla el interior de una ventana como si fuera una pantalla virtual. Por consiguiente, el programador de la aplicación no tiene que preocuparse por los detalles de la operación bajo el control de un administrador de ventanas.

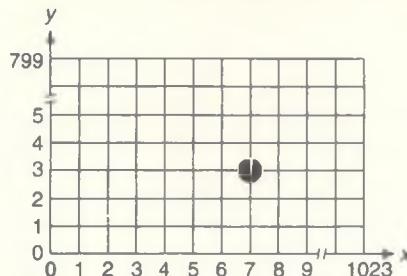
## 2.1 Dibujo con SRGP

### 2.1.1 Especificación de primitivas gráficas

El dibujo en los paquetes gráficos de barrido como SRGP es similar al trazado de gráficos en papel milimétrico con una retícula muy fina. Esta malla varía de 80 a 120 puntos por pulgada en las pantallas convencionales a 300 o más en las de alta resolución. Al aumentar la resolución mejora la apariencia del detalle fino. En la figura 2.1 se presenta una pantalla (o la superficie del papel de la impresora o una película fotográfica) cuadriculada con el sistema de coordenadas cartesianas de SRGP. Observe que los pixeles en SRGP se hallan en la intersección de las líneas.

El origen  $(0, 0)$  se encuentra en la parte inferior izquierda de la pantalla; el eje  $x$  positivo aumenta hacia la derecha y el eje  $y$  positivo aumenta hacia la parte superior. El pixel en la esquina superior derecha es  $(\text{anchura}-1, \text{altura}-1)$ , donde la anchura y la altura son las dimensiones de la pantalla dependientes del dispositivo.

En el papel para gráficos podemos dibujar una línea continua entre dos puntos cualesquiera del papel; sin embargo, en las pantallas de barrido sólo podemos dibujar líneas entre puntos de la malla y es necesario aproximar una línea intensificando los pixeles de los puntos de la malla que están sobre ella o



**Figura 2.1** Sistema de coordenadas cartesianas de una pantalla de 1024 pixeles de anchura por 800 pixeles de altura. Se muestra el pixel (7, 3).

próximo. En forma similar, las figuras sólidas como los círculos o polígonos rellenos se crean intensificando los pixeles en sus interiores y sus fronteras. Como la especificación de cada pixel de una línea o una figura cerrada sería una tarea demasiado laboriosa, los paquetes gráficos permiten que el programador especifique primitivas, como las líneas y los polígonos, por medio de sus vértices; el paquete completa los detalles usando algoritmos de discretización, los cuales se analizan en el capítulo 3.

SRGP apoya una colección básica de primitivas: líneas, polígonos, círculos y elipses, y texto.<sup>1</sup> Para especificar una primitiva, la aplicación envía las coordenadas que definen la forma a la función generadora de primitiva apropiada en SRGP. Es válido que un punto especificado se encuentre fuera del área rectangular que limita la pantalla; por supuesto, sólo serán visibles las porciones de la primitiva que se hallen dentro de estos límites.

Usaremos el lenguaje C ANSI con las siguientes convenciones tipográficas. Las palabras clave de C, los tipos intrínsecos y los tipos definidos por el usuario se presentan en negritas. Las variables utilizadas en el cuerpo del texto se muestran en cursivas. Las constantes simbólicas aparecen en mayúsculas, los comentarios se encierran entre los delimitadores `/*...*/` y el seudocódigo está en cursivas. Por cuestiones de brevedad, omitimos las declaraciones de constantes y de variables cuando sean obvias. Las variables booleanas son de tipo `unsigned char`, con TRUE (verdadero) y FALSE (falso) definidos como 1 y 0 respectivamente. Una vez definidos, los tipos nuevos, como `point` (punto), se usarán en fragmentos de código posteriores sin volver a definirse.

**Líneas y polilíneas.** La siguiente función de SRGP dibuja una línea de  $(x_1, y_1)$  a  $(x_2, y_2)$ :

```
void SRGP_lineCoord (int x1, int y1, int x2, int y2);
```

<sup>1</sup> Las funciones especializadas que dibujan un pixel o un arreglo de pixeles se describen en el manual de referencia de SRGP.

Así, para trazar una linea de (0, 0) a (100, 300) basta hacer la llamada

```
SRGP_lineCoord (0, 0, 100, 300);
```

Como muchas veces es más natural pensar en función de puntos extremos y no en función de coordenadas  $x$  y  $y$  individuales, SRGP ofrece una función alternativa para dibujar líneas:

```
void SRGP_line (point pt1, point pt2);
```

Aquí, “point” (punto) es un tipo definido, una estructura de dos enteros que almacena los valores  $x$  y  $y$  del punto:

```
typedef struct {
    int x, y;
} point;
```

Una secuencia de líneas que conectan vértices sucesivos se denomina **polilínea**. Aunque es posible crear polilíneas llamando repetidamente a las funciones de dibujo de líneas, SRGP las incluye como caso especial. Hay dos funciones de polilíneas, análogas a las formas de coordenadas y puntos de las funciones de dibujo de líneas, que utilizan arreglos como parámetros:

```
void SRGP_polyLineCoord (int num_vértices, vertexCoordinateList arreglo_x,
                         vertexCoordinateList arreglo_y);
void SRGP_polyLine (int num_vértices, vertexList vértices);
```

donde *vertexCoordinateList* (lista de coordenadas de vértices) y *vertexList* (lista de vértices) son tipos definidos por el paquete SRGP: arreglos de enteros y puntos, respectivamente.

El primer parámetro de las dos llamadas de polilínea indica a SRGP cuántos vértices debe esperar. En la primera llamada, el segundo y el tercer parámetros son arreglos enteros de pares de valores  $x$  y  $y$  y la polilínea se dibuja del vértice (*arreglo\_x*[0], *arreglo\_y*[0]) al vértice (*arreglo\_x*[1], *arreglo\_y*[1]) al vértice (*arreglo\_x*[2], *arreglo\_y*[2]), etcétera. Esta forma es conveniente, por ejemplo, cuando se grafican datos en un conjunto de ejes estándar, donde *arreglo\_x* es un conjunto predeterminado de valores de la variable independiente y *arreglo\_y* es el conjunto de datos que se calculan o que introduce el usuario.

Como ejemplo, grafiquemos la salida de un programa de análisis económico que calcula cifras comerciales mensuales y las almacena en un arreglo de datos entero *saldo\_comercial* de 12 entradas. Comenzaremos nuestro gráfico en (200, 200). Para poder apreciar las diferencias entre los puntos sucesivos, los graficaremos cada 10 puntos en el eje  $x$ . De esta manera, creamos un arreglo entero *meses* para representar los 12 meses y asignamos las entradas con los valores  $x$  deseados: 200, 210, ..., 310. En forma parecida, hay que incrementar

en 200 cada valor del arreglo de datos para colocar las 12 coordenadas  $y$  en el lugar correcto. Así, el gráfico de la figura 2.2 se grafica con el código siguiente:

```
/* Graficar los ejes */
SRGP_lineCoord (175, 200, 320, 200);
SRGP_lineCoord (200, 140, 200, 280);

/* Graficar los datos */
SRGP_polyLineCoord (12, meses, saldo_comercial);
```

Podemos emplear la segunda forma de polilínea para dibujar figuras, especificando pares de valores  $x$  y  $y$  como puntos y pasando un arreglo de estos puntos a SRGP. La forma de corbata de moño de la figura 2.3 se crea con la llamada

```
SRGP_polyLine (7, arreglo_corbata_moño);
```

La tabla en la figura 2.3 muestra cómo se definió *arreglo\_corbata\_moño*.

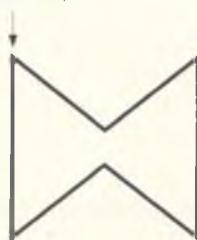
**Marcas y polimarcas.** Con frecuencia es conveniente colocar *marcas* (p. ej., puntos, asteriscos o círculos) en los puntos de datos de los gráficos. Por lo tanto, SRGP ofrece compañeros para las funciones de línea y polilínea. Las siguientes funciones crean un símbolo de marca centrado en  $(x, y)$ :

```
void SRGP_markerCoord (int x, int y);
void SRGP_marker (point pt);
```

También es posible cambiar el estilo y el tamaño de la marca, como explicaremos en la sección 2.1.2. Para crear una secuencia de marcas idénticas en un conjunto de puntos, llamamos a una de las siguientes funciones:

```
void SRGP_polyMarkerCoord (int num_vértices, vertexCoordinateList arreglo_x,
                           vertexCoordinateList arreglo_y);
void SRGP_polyMarker (int num_vértices, vertexList vértices);
```

(100, 100)



	x	y
0	100	100
1	100	60
2	120	76
3	140	60
4	140	100
5	120	84
6	100	100

arreglo\_corbata\_moño

Figura 2.3 Dibujo de una polilínea.

De esta manera, la siguiente llamada adicional añadirá marcas al gráfico de la figura 2.2 para producir la figura 2.4:

```
SRGP_polyMarkerCoord (12, meses, saldo_comercial);
```

**Polygones y rectángulos.** Para dibujar las aristas de un polígono, podemos especificar una polilínea cerrada especificando el vértice inicial y el final para que sean idénticos (como hicimos para dibujar la corbata de moño de la figura 2.3), o podemos emplear la siguiente llamada especializada de SRGP:

```
void SRGP_polygon (int num_vértices, vertexList vértices);
```

Esta llamada cierra automáticamente la figura dibujando una línea del último vértice al primero. Para dibujar la corbata de moño de la figura 2.3 como polígono se usa la siguiente llamada, donde *arreglo\_corbata\_moño* es ahora un arreglo de sólo seis puntos:

```
SRGP_polygon (6, arreglo_corbata_moño);
```

Cualquier rectángulo se puede especificar como un polígono con cuatro vértices, pero un rectángulo vertical (cuyas aristas son paralelas a las orillas de la pantalla) también se puede especificar con la primitiva *rectangle* de SRGP, usando sólo dos vértices (el inferior izquierdo y el superior derecho):

```
void SRGP_rectangleCoord (int x_izquierda, int y_inferior, int x_derecha, int y_superior);
void SRGP_rectanglePt (point izquierda_inferior, point derecha_superior);
void SRGP_rectangle (rectangle rectángulo);
```

La estructura *rectangle* almacena las esquinas inferior izquierda y superior derecha:

```
typedef struct {
    point bottomLeft, topRight;
} rectangle;
```

De esta manera, la siguiente llamada dibuja un rectángulo vertical de 101 pixeles de anchura por 151 pixeles de altura:

```
SRGP_rectangleCoord (50, 25, 150, 175);
```

SRGP proporciona las siguientes utilerías para crear rectángulos y puntos a partir de datos de coordenadas.

```
point SRGP_defPoint (int x, int y);
rectangle SRGP_defRectangle (int x_izquierda, int y_inferior, int x_derecha, int
y_superior);
```

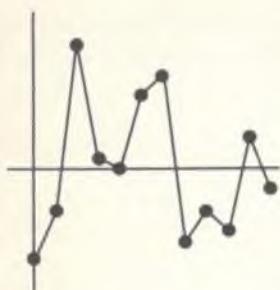


Figura 2.4  
Graficación del arreglo de datos usando marcas.

Con ellas, podríamos haber dibujado nuestro rectángulo ejemplo con

```
rectángulo = SRGB_defRectangle (50, 25, 150, 175);
SRGP_rectangle (rectángulo);
```

**Círculos y elipses.** En la figura 2.5 se presentan arcos circulares y elípticos dibujados con SRGB. Como los círculos son casos especiales de las elipses, usaremos el término **arco elíptico** para todas estas formas, sean arcos circulares o elípticos, cerrados o parciales. SRGB sólo puede dibujar elipses estándar, es decir, aquellas en las cuales los ejes menor y mayor son paralelos a los ejes de las coordenadas.

Aunque existen varios métodos matemáticamente equivalentes para especificar arcos elípticos, es conveniente que el programador especifique los arcos con base en los rectángulos verticales en los cuales están inscritos (Fig. 2.6); estos rectángulos verticales se denominan *cajas acotantes* o *extensiones*.

La función general de la elipse es

```
void SRGB_ellipseArc (rectangle rectángulo_extensión, float ángulo_inicial, float
ángulo_final);
```

La anchura y la altura de la extensión determinan la forma de la elipse. El hecho de que el arco esté cerrado depende de un par de ángulos que especifican dónde inicia el arco y dónde termina. Por conveniencia, los ángulos se miden en *grados rectangulares* en sentido contrario al de las manecillas del reloj, con  $0^\circ$  correspondiendo a la porción positiva del eje  $x$ ,  $90^\circ$  a la porción positiva del eje  $y$  y  $45^\circ$  a la “diagonal” que se extiende del origen a la esquina superior izquierda del rectángulo. Obviamente, los grados rectangulares equivalen a los circulares sólo si la extensión es un cuadrado. La relación general entre los ángulos rectangulares y los circulares es

$$\text{ángulo rectangular} = \arctan\left(\tan(\text{ángulo circular}) \cdot \frac{\text{anchura}}{\text{altura}}\right) + \text{ajuste}.$$

donde los ángulos están en radianes y

$$\text{ajuste} = 0, \text{ para } 0 \leq \text{ángulo circular} < \frac{\pi}{2}$$

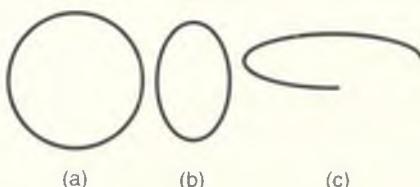
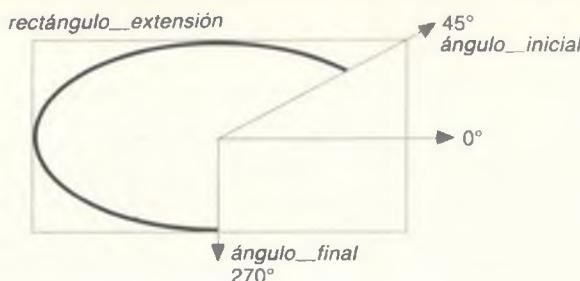


Figura 2.5 Dibujos de arcos elípticos: (a) circular, (b) elíptico cerrado, (c) elíptico.



**Figura 2.6** Especificación de arcos elípticos.

$$\text{ajuste} = \pi, \text{ para } \frac{\pi}{2} \leq \text{ángulo circular} < \frac{3\pi}{2}$$

$$\text{ajuste} = 2\pi, \text{ para } \frac{3\pi}{2} \leq \text{ángulo circular} < 2\pi .$$

## 2.1.2 Atributos

**Estilo y anchura de líneas.** La apariencia de una primitiva se puede controlar con la especificación de sus **atributos**.<sup>2</sup> Los atributos de SRGP aplicables a las líneas, polilíneas, polígonos, rectángulos y arcos elípticos son *estilo de línea*, *anchura de línea*, *color* y *tipo de pincel*.

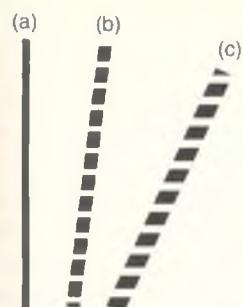
Los atributos se establecen *modalmente*; es decir, son variables de estado globales que conservan sus valores hasta que se cambian de manera explícita. Las primitivas se dibujan con los atributos vigentes al momento de especificar las primitivas; por consiguiente, el cambio del valor del atributo no afecta las primitivas creadas previamente; sólo se afectan las que se especifiquen después de modificar el valor del atributo. Los atributos modales son convenientes porque evitan que el programador tenga que especificar una larga lista de parámetros para cada primitiva (puede haber una docena de atributos diferentes en un sistema de producción).

El estilo y la anchura de una línea se establecen con las llamadas a

```
void SRGP_setLineStyle (lineStyle CONTINUOUS / DASHED / DOTTED / ...);3
void SRGP_lineWidth (int anchura);
```

<sup>2</sup> Las descripciones que se ofrecen aquí de los atributos de SRGP muchas veces carecen de detalles finos, en especial acerca de la interacción entre los distintos atributos. Se omite este nivel de detalle porque el efecto exacto de un atributo es una función de su implantación y, por razones de rendimiento, se emplean implantaciones diversas en distintos sistemas; si desea conocer estos detalles, consulte los manuales de referencia específicos para la implantación.

<sup>3</sup> Aquí y en el texto subsecuente usaremos una notación abreviada. En SRGP, estas constantes simbólicas son en realidad valores de tipo de datos enumerados `lineStyle`.



**Figura 2.7**  
Líneas de anchuras y estilos diversos.

La anchura de una línea se mide en unidades de la pantalla, es decir, en píxeles. Cada atributo tiene un valor por omisión (*default*): el estilo de línea es CONTINUOUS (continuo) y la anchura es 1. En la figura 2.7 se muestran líneas con una variedad de anchuras y estilos; el código que generó la figura se presenta en el programa 2.1.

Podemos considerar el estilo de una línea como una máscara de bits utilizada para escribir selectivamente píxeles cuando SRGP efectúa la discretización de la primitiva. Un cero en la máscara indica que no debe escribirse el pixel y por ende conserva su valor original en la memoria gráfica. Este pixel de la línea se puede pensar como transparente, ya que permite ver el pixel original que está *debajo*. De esta manera, CONTINUOUS corresponde a una cadena formada exclusivamente por unos y DASHED (guiones) a la cadena 1111001111001111[...], con el guion dos veces más largo que los segmentos intermedios transparentes.

Cada atributo tiene un valor por omisión; por ejemplo, el del estilo de línea es CONTINUOUS, el de la anchura es 1, etcétera. En los primeros ejemplos de código no establecimos el estilo de línea para la primera que dibujamos, por lo cual usamos el valor por omisión. Sin embargo, en la práctica no es muy seguro hacer suposiciones acerca del estado actual de los atributos, y en los ejemplos de código que se presentan a continuación los asignamos de manera explícita en cada función, para que éstas sean modulares y se faciliten la depuración y el mantenimiento. En la sección 2.1.4 veremos que una práctica aún más segura es que el programador almacene y restaure explícitamente los atributos para cada función.

### Programa 2.1

Código utilizado para generar la figura 2.7.

```
SRGP_setLineWidth (5);                                /* Línea a */
SRGP_lineCoord (55, 5, 55, 295);
SRGP_setLineStyle (DASHED);
SRGP_setLineWidth (10);                             /* Línea b */
SRGP_lineCoord (105, 5, 155, 295);
SRGP_setLineWidth (15);
SRGP_setLineStyle (DOTTED);
SRGP_lineCoord (155, 5, 285, 255);                /* Línea c */
```

Los atributos que se pueden establecer para la primitiva de marca son

```
void SRGP_setMarkerSize (int tamaño);
void SRGP_setMarkerStyle (markerStyle MARKER_CIRCLE
                         / MARKER_SQUARE / ...);
```

El tamaño de la marca especifica en píxeles la longitud de los lados de la extensión cuadrada de cada marca. El conjunto completo de estilos de marcas se presenta en el manual de referencia; el estilo de círculo es la forma por omisión que se muestra en la figura 2.4.

**Color.** Cada uno de los atributos presentados hasta ahora sólo afecta algunas de las primitivas de SRGP, pero el atributo de color (de valor entero) afecta

todas las primitivas. Obviamente, el significado del atributo de color depende en gran medida del hardware subyacente; los dos valores de color que existen en todos los sistemas son 0 y 1. En los sistemas de dos niveles se puede predecir con facilidad la apariencia de los colores: los pixeles de color 1 son negros y los de color 0 son blancos en los dispositivos que presentan en negro sobre blanco; el verde es 1 y el negro es 0 para los dispositivos de verde sobre negro, etcétera.

El atributo de color entero no especifica el color en forma directa; más bien es un índice que apunta a la **tabla de colores** de SRGP, cada una de cuyas entradas define un color o un valor de escala de grises en una forma que no necesita conocer el programador de SRGP. Hay  $2^d$  entradas en la tabla de colores, donde  $d$  es la *profundidad* (número de bits almacenados para cada pixel) de la memoria gráfica. En las implantaciones de dos niveles la tabla de colores está implantada en hardware; sin embargo, en la mayoría de las implantaciones en color, SRGP permite que la aplicación modifique la tabla. Consulte el manual de referencia para conocer los detalles. En el capítulo 4 se examinan algunas aplicaciones del método indirecto que ofrecen las tablas de colores.

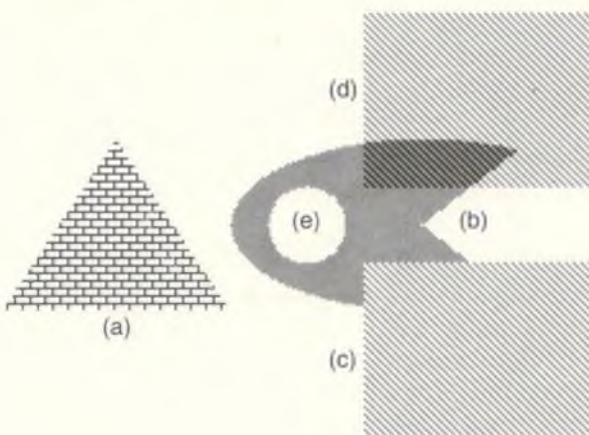
Una aplicación puede emplear uno de dos métodos para especificar colores. Si la independencia de la máquina es un factor importante para la aplicación, deben usarse los enteros 0 y 1 en forma directa, con lo cual la aplicación se ejecutará correctamente en todas las pantallas de dos niveles y en color. Si la aplicación permite el empleo de colores o está escrita para un dispositivo de presentación en particular, puede usar los *nombres de color* dependientes de la implantación que apoya SRGP. Estos nombres son constantes enteras simbólicas que muestran dónde se han colocado ciertos colores estándar dentro de la tabla de colores por omisión del dispositivo de presentación. Por ejemplo, una implantación de negro sobre blanco ofrece los nombres de color COLOR\_BLACK (1, negro) y COLOR\_WHITE (0, blanco); usaremos estos dos valores en los fragmentos de código muestra de este capítulo. Observe que los nombres de colores no son útiles para las aplicaciones que modifican la tabla de colores. La selección de un color se lleva a cabo con la llamada

```
void SRGPSetColor (int indice_color);
```

### 2.1.3 Primitivas rellenas y sus atributos

Las primitivas que encierran áreas (las llamadas *primitivas de definición de área*) se pueden dibujar en dos formas: *huecas* o *rellenadas*. Las funciones descritas en la sección anterior generan el primero de estos estilos: líneas exteriores cerradas con interiores huecos. Las versiones rellenadas de SRGP de las primitivas de definición de área dibujan los pixeles interiores sin la línea exterior. En la figura 2.8 se presenta el repertorio de primitivas rellenadas de SRGP, incluyendo un arco elíptico relleno, o *trozo de pastel*, en la parte (b).

Observe que SRGP no dibuja una línea exterior contrastante, como sería un borde sólido de un pixel de ancho, alrededor del interior; las aplicaciones que requieren este tipo de borde deben dibujarlo en forma explícita. También existe



**Figura 2.8** Patrones de mapa de bits de primitivas rellenas: (a-c) opacos, (d) transparente, (e) sólido.

el aspecto más sutil de si en realidad se deben dibujar los pixeles en el borde de una primitiva de definición de área; es decir, si únicamente deben dibujarse los pixeles del interior. Este problema se analizará con detalle en la sección 3.4.

Para generar un polígono relleno, usamos *SRGP\_fillPolygon* o *SRGP\_fillPolygonCoord*, con las mismas listas de parámetros que se usaron en las versiones huecas de estas llamadas. Definimos de la misma manera las otras primitivas de relleno de área, añadiendo el prefijo “fill” (rellenar) a sus nombres. Como los polígonos pueden ser cóncavos o incluso intersecarse, necesitamos una regla para especificar cuáles son las regiones internas (y por ende las que deben rellenarse) y cuáles las exteriores. Los polígonos en SRGP siguen la regla de *paridad impar*. Para determinar si una región está dentro o fuera de un polígono, se elige como punto de prueba cualquiera que se halle dentro de la región. Después se escoge un rayo que parte del punto de prueba, que se extienda indefinidamente en cualquier dirección y que no pase por ninguno de los vértices. Si este rayo interseca la línea exterior del polígono un número impar de veces, se considera que la región es interior (Fig. 2.9).

### Programa 2.2

Código utilizado para generar la figura 2.8.

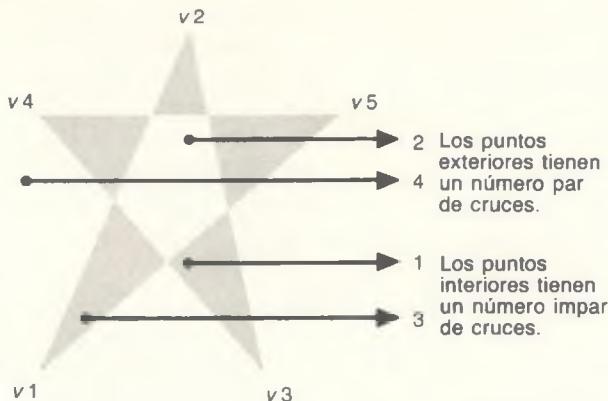
```

SRGP_setFillStyle (BITMAP_PATTERN_OPAQUE);
SRGP_setFillBitmapPattern (BRICK_BIT_PATTERN);           /* Patrón de ladrillos */
SRGP_fillPolygon (3, coords_tríngulo);                  /* a */

SRGP_setFillBitmapPattern (MEDIUM_GRAY_BIT_PATTERN);    /* Gris al 50 por ciento */
SRGP_fillEllipseArc (rect_arco_elipse, 60.0, 290);     /* b */

SRGP_setFillBitmapPattern (DIAGONAL_BIT_PATTERN);
SRGP_fillRectangle (rectángulo_relleno_opaco);          /* c */

```



**Figura 2.9** Regla de paridad impar para determinar el interior de un polígono.

```

SRGP_setFillStyle (BITMAP_PATTERN_TRANSPARENT);
SRGP_fillRectangle (rectángulo_rellenado_transparente); /* d */

SRGP_setFillStyle (SOLID);
SRGPSetColor (COLOR_WHITE);
SRGP_fillEllipse (rectángulo_círculo); /* e */

```

En realidad, SRGP no lleva a cabo esta tarea para cada uno de los pixeles cuando dibuja; más bien, el paquete utiliza la técnica optimizada de discretización de polígonos descrita en el capítulo 3, en la cual se aplica la regla de paridad impar a todo un tramo de pixeles adyacentes que se encuentran en el interior o en el exterior. Además, la prueba de intersección de rayo de paridad impar se emplea en un proceso denominado **correlación de selección** para determinar el objeto que el usuario está seleccionando con el cursor, como se describe en el capítulo 7.

**Estilo y patrón de relleno para áreas.** El atributo de estilo de relleno se puede usar para controlar en cuatro formas distintas la apariencia del interior de una primitiva rellenada, utilizando

```
void SRGP_setFillStyle ,(drawStyle SOLID / BITMAP_PATTERN_OPAQUE /
BITMAP_PATTERN_TRANSPARENT / PIXMAP_PATTERN);
```

La primera opción, SOLID (sólido), produce una primitiva uniformemente rellena da con el valor actual del atributo de color (Fig. 2.8e, con el color establecido en COLOR\_WHITE). Las dos opciones siguientes, BITMAP\_PATTERN\_OPAQUE (patrón de mapa de bits opaco) y BITMAP\_PATTERN\_TRANSPARENT (patrón de mapa de bits transparente), rellenan las primitivas con un

patrón regular, no sólido, el primero reescribiendo los pixeles subyacentes en el color actual o en otro (Fig. 2.8c) y el segundo reescribiendo algunos de los pixeles debajo de la primitiva en el color actual pero permitiendo que otros sean visibles (Fig. 2.8d). La última opción, PIXMAP\_PATTERN (patrón de mapa de pixeles), escribe patrones que contienen un número arbitrario de colores, siempre en modo opaco.

Los patrones de relleno de mapa de bits son arreglos de unos y ceros elegidos en una tabla de patrones disponibles por medio de la llamada

```
void SRGP_setFillBitmapPattern (int índice_patrón);
```

Cada entrada en la tabla de patrones almacena un patrón único; los que se proporcionan con SRGP, indicados en el manual de referencia, incluyen tonos de escala de grises (que van de casi negro a casi blanco) y varios patrones regulares y aleatorios. En el modo transparente los patrones se generan de la siguiente manera. Considere cualquier patrón de la tabla como un pequeño arreglo bidimensional de bits (*mapa de bits, bitmap*) digamos de  $8 \times 8$  que se repetirá cuanto sea necesario (como azulejo) para llenar la primitiva. En un sistema de dos niveles se escribe el color actual (de hecho, el color de *primer plano*) cuando hay unos en el patrón; cuando hay ceros (los *agujeros*) no se escriben los pixeles correspondientes de la imagen original, por lo que se *ven a través* de la primitiva parcialmente transparente que se encuentra encima. Así, el patrón de mapa de bits actúa como *mascarilla de activación de escritura en memoria* para los patrones en el modo transparente, de manera similar a como lo hizo la mascarilla de bits del estilo de líneas para las primitivas de líneas y de bordes.

En el modo más usual, BITMAP\_PATTERN\_OPAQUE, los unos se escriben en el color actual pero los ceros se escriben en otro color, el *color de fondo*, establecido previamente con

```
void SRGP_setBackgroundColor (int índice_color);
```

En las pantallas de dos niveles, cada patrón de mapa de bits en modo OPAQUE (opaco) sólo puede generar dos patrones de relleno distintos. Por ejemplo, un patrón de mapa de bits formado principalmente por unos puede usarse en una pantalla en blanco y negro para generar un patrón de relleno gris oscuro si se asigna el color actual igual a negro (y el fondo a blanco), y se puede usar un patrón de relleno gris claro si el color actual se asigna igual a blanco (y el fondo igual a negro). En una pantalla de color se puede usar cualquier combinación de colores de primer plano y de fondo para generar diversos efectos de dos tonos. Una aplicación típica en una pantalla de dos niveles siempre asigna el color de fondo cuando establece el de primer plano, ya que los patrones de mapa de bits opacos no son visibles si los dos colores son iguales; una aplicación podría crear una función *establecer\_color* para asignar automáticamente el color de fondo de manera que contraste con el de primer plano cuando éste se especifique explícitamente.

La figura 2.8 se creó con el fragmento de código que se presenta en el programa 2.2. La ventaja de tener patrones de mapa de bits de dos tonos es que los colores no se especifican de manera explícita, sino que están determinados por los atributos de color vigentes y por lo tanto se pueden generar en cualquier combinación de colores. La desventaja, y la razón por la cual SRGP también permite utilizar patrones de mapa de pixeles (arreglo bidimensional de pixeles, *pixmap*), es que sólo se pueden generar dos colores. Con frecuencia nos gustaría llenar un área de la pantalla con más de dos colores, usando un patrón explícitamente especificado. De la misma manera que el patrón de mapa de bits se usa como azulejo para embaldosar la primitiva, se puede emplear un pequeño mapa de pixeles para el embaldosado, donde el mapa de pixeles es un arreglo de patrones de índices de la tabla de colores. Como cada pixel se asigna explícitamente en el mapa de pixeles, no existe el concepto de los agujeros y por lo tanto no hay diferencia entre los modos transparente y opaco. Para llenar un área con un patrón de color, seleccionamos un estilo de relleno de PIXMAP\_PATTERN y usamos la función correspondiente de selección de patrón de mapa de pixeles:

```
void SRGP_setFillPixmapPattern (int índice_patrón);
```

Como los patrones de mapa de bits y de mapa de pixeles generan pixeles con valores de color que son índices en la tabla de colores actual, la apariencia de las primitivas llenadas cambia si el programador modifica las entradas de la tabla. El manual de referencia de SRGP indica cómo modificar o añadir entradas de las tablas de patrones de mapa de bits y de pixeles. Así mismo, aunque SRGP ofrece entradas por omisión para la tabla de patrones de mapa de bits, no existe una tabla por omisión para los mapas de pixeles, ya que hay una cantidad indefinida de patrones de mapa de pixeles a color que podrían considerarse útiles.

**Fondo de la pantalla de la aplicación.** Hemos definido el **color de fondo** como el color de los bits 0 en los patrones de mapa de bits usados en modo opaco, pero el término *fondo* se emplea de otra manera no relacionada. Por lo general, el usuario espera que la pantalla presente las primitivas sobre un *patrón de fondo de la pantalla de la aplicación* que cubra de manera uniforme una ventana opaca o toda la pantalla. El patrón de fondo de la pantalla de la aplicación es con frecuencia el color 0, ya que SRGP asigna un valor inicial de la pantalla igual a ese color. Sin embargo, el patrón de fondo en ocasiones no es sólido o bien es un sólido de otro color; en estos casos, la aplicación es responsable de configurar el fondo dibujando un rectángulo que abarque toda la pantalla y que tenga el color deseado, antes de dibujar otras primitivas.

Una técnica común para *borrar* primitivas es redibujarlas con el patrón de fondo de la pantalla de la aplicación, en lugar de redibujar toda la imagen cada vez que se elimina una primitiva. Sin embargo, esta *sucia y rápida* técnica para

la actualización produce una imagen dañada cuando la primitiva borrada se sobreponen a otras primitivas.

Por ejemplo, suponga que el patrón de fondo de la pantalla de la figura 2.8 es el blanco sólido y que borramos el rectángulo de la parte (c) redibujando esta parte con el color sólido COLOR\_WHITE. Esta técnica dejaría un hueco blanco en el arco elíptico relleno (parte b) que está debajo. Para la *reparación de daños* hay que regresar a la base de datos de la aplicación y volver a especificar las primitivas (véase el Ejer. 2.8).

## 2.1.4 Almacenamiento y recuperación de atributos

Como puede ver, SRGP apoya diversos atributos para sus primitivas. Los atributos pueden almacenarse para ser recuperados posteriormente; esta característica es muy útil al diseñar funciones de aplicaciones que lleven a cabo sus tareas sin ocasionar efectos secundarios, es decir, sin afectar el estado global de los atributos. Por conveniencia, SRGP permite consultar y restaurar todo el conjunto de atributos, conocido como *grupo de atributos*, por medio de la llamada

```
void SRGP_inquireAttributes (attributeGroup *grupo);
void SRGP_setAttributes (attributeGroup grupo);
```

El programa de aplicación tiene acceso a todos los campos del registro de “grupo de atributos” definido por SRGP, de manera que el registro que devuelve la función de consulta puede usarse posteriormente para la restauración selectiva.

## 2.1.5 Texto

La especificación y la implantación del dibujo de texto siempre es una tarea compleja en los paquetes gráficos, por la gran cantidad de opciones y atributos que puede tener el texto. Entre estos atributos se encuentran el estilo o **familia tipográfica** de los caracteres (Times, Helvetica, Bodoni, etc.), su apariencia (romanas, **negritas**, **cursivas**, **subrayado**, etc.), su tamaño (generalmente medido en **puntos**<sup>4</sup>) y su anchura, el espaciado entre caracteres, el espaciado entre líneas consecutivas, el ángulo de dibujo de los caracteres (horizontal, vertical o a un ángulo especificado), etcétera.

El recurso más rudimentario, que usualmente se encuentra en hardware y software sencillas, es un espaciado de anchura fija entre caracteres en el cual todos los caracteres tienen la misma anchura e igual espaciado entre ellos. En el otro extremo del espectro, el espaciado proporcional hace variar la anchura de los caracteres y el espaciado entre ellos para que el texto sea más legible y estético. Los libros, revistas y periódicos utilizan espaciado proporcional, así como la mayoría de las pantallas gráficas de barrido y las impresoras láser. SRGP

<sup>4</sup> El punto es una unidad de uso común en la industria editorial; equivale aproximadamente a 1/72 de pulgada.

ofrece una funcionalidad intermedia: el texto se alinea horizontalmente y la anchura de los caracteres varía, pero el espacio entre ellos permanece constante. Con esta sencilla forma de espacio proporcional, la aplicación puede efectuar anotaciones en diagramas gráficos, interactuar con el usuario a través de menús textuales, cuadros de diálogo y formularios para llenado, e incluso implantar sencillos procesadores de palabras. Sin embargo, las aplicaciones que usan mucho texto, como los programas de compuendio para la producción de documentos de alta calidad, requieren paquetes especializados que ofrezcan mayor control sobre la especificación del texto y sus atributos que lo que podría ofrecer SRGP. PostScript [ADOB85] ofrece muchas de estas características avanzadas y se ha convertido en un estándar industrial para describir texto y otras primitivas con gran variedad de opciones y atributos. El texto en SRGP se genera con una llamada a

```
void SRGP_text (point origen, char *texto);
```

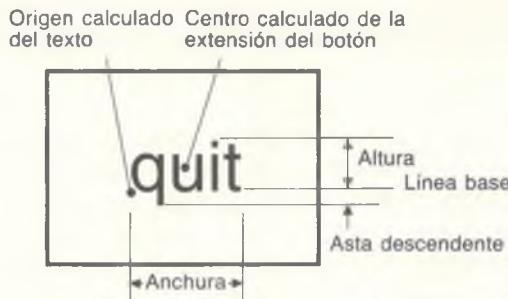
La posición de una primitiva de texto es controlada por la especificación de su **origen**, también conocido como **punto ancla**. La coordenada *x* del origen marca el extremo izquierdo del primer carácter y la coordenada *y* especifica dónde debe aparecer la línea base de la cadena. (La **línea base** es la línea hipotética sobre la cual se apoyan los caracteres, como se muestra en el botón de menú de la figura 2.10. Algunos caracteres, como "y" y "q", tienen una cola, llamada **asta descendente**, que desciende de la línea base.)

La apariencia de una primitiva de texto es determinada por sólo dos atributos: el color actual y la familia tipográfica. La familia tipográfica es un índice en una tabla de familias de diversos tamaños y estilos, dependiente de la implantación:

```
void SRGPSetFont (int índice_valor);
```

Todo carácter se define como un mapa de bits rectangular, y SRGP dibuja un carácter rellenando un rectángulo, con el mapa de bits del carácter como patrón, en modo de patrón de mapa de bits transparente. Los unos del mapa de bits definen el interior del carácter y los ceros especifican los espacios y huecos, como el agujero en "o". (Algunos paquetes más elaborados definen caracteres con mapas de píxeles, lo cual permite que el interior de un carácter tenga un patrón.)

**Formato de texto.** Como las implantaciones de SRGP ofrecen un repertorio limitado de familias tipográficas y tamaños, y dado que las implantaciones en hardware diferente pocas veces ofrecen repertorios equivalentes, una aplicación tiene un control limitado sobre la altura y la anchura de las cadenas de texto. Es necesario contar con información de la extensión del texto para ser capaces de producir composiciones bien balanceadas (p. ej., para centrar una cadena de texto dentro de un marco rectangular), por lo que SRGP ofrece la siguiente



**Figura 2.10** Las dimensiones de un texto centrado en un botón rectangular y los puntos calculados a partir de estas dimensiones para fines de centrado.

función para consultar la extensión de una cadena usando el valor actual del atributo de familia tipográfica:

```
void SRGP_inquireTextExtent (char *texto, int *anchura, int *altura, int *asta_descendente);
```

Aunque SRGP no permite usar el modo opaco de mapa de bits para escribir caracteres, es fácil simularlo. Como ejemplo, la función del programa 2.3 muestra cómo usar la información de extensión y los atributos específicos del texto para producir texto negro, en la familia tipográfica actual, centrado dentro de un rectángulo blanco, como se muestra en la figura 2.10. La función crea primero, al tamaño adecuado, el rectángulo de botón que sirve de fondo, con un borde de separación, y luego centra el texto. El ejercicio 2.9 es una variante de este tema.

### Programa 2.3

Código usado para generar la figura 2.10

```
void crear_botón_quit (rectangle rectángulo_botón)
{
    point centro_botón, origen_texto;
    int anchura, altura, asta_descendente;

    SRGP_setFillStyle (SOLID);
    SRGPSetColor (COLOR_WHITE);
    SRGP_fillRectangle (rectángulo_botón);
    SRGPSetColor (COLOR_BLACK);
    SRGP_setLineWidth (2);
    SRGP_Rectangle (rectángulo_botón);
    SRGP_inquireTextExtent ("quit", &anchura, &altura, &asta_descendente);
    centro_botón.x = (rectángulo_botón.bottomLeft.x + rectángulo_botón.topRight.x)/2;
    centro_botón.y = (rectángulo_botón.bottomLeft.y + rectángulo_botón.topRight.y)/2;
    origen_texto.x = centro_botón.x - (anchura/2);
    origen_texto.y = centro_botón.y - (altura/2);
    SRGP_text (origen_texto, "quit");
}
```

## 2.2 Manejo básico de la interacción

Ahora que sabemos cómo dibujar formas básicas y texto, el siguiente paso es aprender a escribir programas interactivos que establezcan una comunicación efectiva con el usuario a través de los dispositivos de entrada, como el teclado y el ratón. Primero veremos las pautas generales para elaborar programas interactivos efectivos y agradables de usar; después analizaremos el concepto fundamental de los dispositivos de entrada lógicos (abstractos). Por último, veremos los mecanismos que tiene SRGP para tratar diversos aspectos del manejo de la interacción.

### 2.2.1 Factores humanos

El diseñador de un programa interactivo debe considerar varias cuestiones que no surgen en un programa por lotes, no interactivo. Se trata de los llamados **factores humanos** de un programa, como el estilo de interacción (cómo se “*ve y siente*”) y su facilidad de aprendizaje y uso, que tienen la misma importancia que la integridad y la corrección funcionales. En el capítulo 8 se analizan con mayor detalle las técnicas para la interacción usuario-computador que toman en cuenta el factor humano. Las pautas que allí se presentan incluyen las siguientes:

- Ofrecer secuencias interactivas *sencillas y consistentes*.
- *No sobrecargar al usuario* con demasiadas opciones o estilos.
- *Mostrar con claridad las opciones disponibles* en todas las etapas de la interacción.
- *Proporcionar la realimentación apropiada* al usuario.
- Permitir que el usuario se *recupere elegantemente* de sus errores.

Trataremos de seguir estas pautas para tomar en cuenta los factores humanos en nuestros programas muestra. Por ejemplo, generalmente usaremos menús para permitir que el usuario indique cuál será la siguiente función que se ejecutará, usando un ratón para seleccionar un botón de texto en un menú de botones. También son comunes las paletas (menús de iconos) de primitivas geométricas básicas, símbolos específicos para la aplicación y patrones de relleno. Los menús y las paletas satisfacen las tres primeras pautas, ya que sus entradas presentan al usuario una lista de las opciones disponibles y ofrecen una forma única y consistente de elegir entre las opciones. Las opciones no disponibles se pueden eliminar temporalmente o *desvanecerse*, dibujándolas con un patrón de escala de grises de baja intensidad en lugar de un color sólido (véase el proyecto de programación 2.14).

La realimentación ocurre en cada paso de la operación de un menú para cumplir con la cuarta pauta: el programa de aplicación *realizará* la opción del menú o la selección del objeto— por ejemplo, lo presentará en video inverso o encerrado en un rectángulo— para llamar la atención. El paquete también puede incluir un *eco*, por medio del cual se proporcione una respuesta inmediata a la manipulación de un dispositivo de entrada. Por ejemplo, los caracteres aparecen inmediatamente en la posición del cursor al teclear una entrada desde el teclado; al mover el ratón sobre el escritorio o la mesa, el cursor hace un eco de la posición correspondiente en la pantalla. Los paquetes gráficos ofrecen diversas formas de cursor que pueden usarse para que el programa de aplicación refleje el estado del programa. En varios sistemas de pantalla, la forma del cursor puede variar dinámicamente como función de su posición en la pantalla. Por ejemplo, en varios programas de procesamiento de palabras, el cursor aparece como una flecha en las áreas de menú y como una barra vertical parpadeante en las áreas de texto.

La recuperación elegante de los errores, nuestra quinta pauta, generalmente se ofrece con características de *cancelación* y de *deshacer/rehacer*. Éstas requieren que el programa de aplicación mantenga un registro de las operaciones y sus acciones correctivas inversas.

## 2.2.2 Dispositivos lógicos de entrada

**Tipos de dispositivos en SRGP.** Uno de los objetivos principales al diseñar paquetes gráficos es la independencia de los dispositivos, lo cual mejora la transportabilidad de las aplicaciones. SRGP lo logra para la salida gráfica, proporcionando primitivas que se especifican en función de un sistema entero de coordenadas abstractas, con lo cual la aplicación queda aislada de la necesidad de asignar los pixeles individuales en la memoria gráfica. A fin de lograr un nivel de abstracción para la entrada gráfica, SRGP apoya un conjunto de **dispositivos lógicos de entrada** que aislan la aplicación de los detalles de los dispositivos físicos de entrada disponibles. SRGP apoya dos dispositivos lógicos:

- **El localizador**, un dispositivo para especificar coordenadas en la pantalla y el estado de uno o más botones relacionados.
- **El teclado**, un dispositivo para especificar la entrada de cadenas de caracteres.

SRGP establece una correspondencia entre los dispositivos lógicos y los dispositivos físicos disponibles (p. ej., el localizador podría corresponder a un ratón, una palanca de mandos, una tableta o una pantalla sensible al tacto). Esta correspondencia entre lo lógico y lo físico es común en los lenguajes convencionales por procedimientos y los sistemas operativos, en los cuales los dispositivos de E/S como las terminales, los discos y las unidades de cinta se abstraen a archivos lógicos para lograr tanto la independencia del dispositivo como la sencillez de la programación de aplicaciones.

**Manejo de dispositivos en otros paquetes.** El modelo de entrada de SRGP es, en esencia, un subconjunto de los modelos de entrada de GKS y PHIGS. Las implantaciones de SRGP sólo permiten usar un localizador lógico y un dispositivo de teclado, mientras que GKS y PHIGS permiten varios dispositivos de cada tipo. Estos paquetes también ofrecen apoyo a otros tipos de dispositivos: el dispositivo de **trazo** (que devuelve una polilínea de posiciones de cursor registradas con el localizador físico), el dispositivo de **opciones** (abstracción de un teclado de teclas de función que devuelve un identificador de tecla), el **valuador** (abstracción de un control deslizante o una perilla de control que devuelve un número de punto flotante) y el dispositivo **selector** (abstracción de un dispositivo apuntador, como un ratón o una tableta de datos, con un botón relacionado para indicar una selección que devuelve la identificación de la entidad lógica seleccionada). Otros paquetes, como QuickDraw y X Window System, manejan los dispositivos de entrada en una forma más dependiente del dispositivo, que permite al programador tener mayor control sobre la operación del dispositivo, aunque a expensas de una mayor complejidad del programa de aplicación y de la reducción de la transportabilidad a otras plataformas.

En el capítulo 8 se presenta más información acerca de las propiedades de los dispositivos lógicos. Aquí resumimos brevemente y en forma general los modos de interacción con los dispositivos lógicos y después examinamos con mayor detalle las funciones de interacción de SRGP.

### 2.2.3 Muestreo y procesamiento dirigido por eventos

Hay dos técnicas fundamentales para recibir la información creada por las interacciones de los usuarios. En la técnica de **muestreo** (también conocida como **sondeo**), el programa de aplicación obtiene el valor actual de un dispositivo lógico de entrada (conocido como **medida** del dispositivo) y continúa la ejecución. El muestreo se lleva a cabo sin importar si la medida del dispositivo ha cambiado desde el último muestreo; de hecho, la única forma en que la aplicación puede conocer los cambios en el estado del dispositivo es por medio del muestreo continuo. Este modo es costoso para las aplicaciones interactivas, ya que la mayor parte de sus ciclos de UCP se invertirían en pequeños ciclos de muestreo esperando cambios en la medición.

Una alternativa al ciclo de muestreo (el cual requiere mucho tiempo de la UCP) es la interacción **dirigida por interrupciones o eventos**; con esta técnica, la aplicación activa uno o más dispositivos para la entrada y continúa la ejecución normal hasta ser interrumpida por un **evento** de entrada (un cambio en el estado del dispositivo ocasionado por alguna acción del usuario); el control pasa entonces asincrónicamente a un procedimiento de interrupción que responde al evento. Para cada dispositivo de entrada se define un **accionador de evento**: este accionador es la acción del usuario que ocasiona el evento. Generalmente, el accionador es la presión de un botón, como el del ratón o una tecla del teclado.

Para liberar al programador de los complicados y difíciles aspectos de la transferencia asincrónica del control, varios paquetes gráficos, incluyendo

GKS, PHIGS y SRGP, ofrecen una interacción dirigida por eventos como una simulación sincrónica de la interacción dirigida por interrupciones. En esta técnica, una aplicación activa dispositivos y prosigue con la ejecución. En un segundo plano, el paquete vigila los dispositivos y almacena información acerca de cada evento en una cola (Fig. 2.11). Cuando le es conveniente, la aplicación revisa la cola de eventos y los procesa en orden cronológico. De hecho, la aplicación especifica cuándo quiere ser *interrumpida*.

Cuando una aplicación revisa la cola de eventos, especifica si quiere ingresar a un estado de espera. Si la cola contiene uno o más informes de eventos, se elimina el evento a la cabeza (que representa el más antiguo) y su información se pone a disposición de la aplicación. Si la cola está vacía y no se desea un estado de espera, se informa a la aplicación que no hay eventos disponibles y que está libre para continuar con la ejecución. Si la cola está vacía y se desea un estado de espera, la aplicación hace una pausa hasta que ocurra el siguiente evento o hasta que transcurra un intervalo máximo especificado por la aplicación. De hecho, el modo de eventos reemplaza a los sondeos de los dispositivos de entrada con una espera más eficiente en la cola de eventos.

En resumen, en el modo de muestreo se sondea el dispositivo y se recopila una medición de evento, sin importar las actividades del usuario. En el modo de eventos, la aplicación obtiene un informe de evento de una acción previa del usuario o espera a que ocurra una acción de éste (o a que transcurra el tiempo permitido). Este comportamiento que *sólo responde cuando actúa el usuario* es la diferencia esencial entre la entrada muestreada y la dirigida por eventos. La programación dirigida por eventos puede parecer más compleja que el

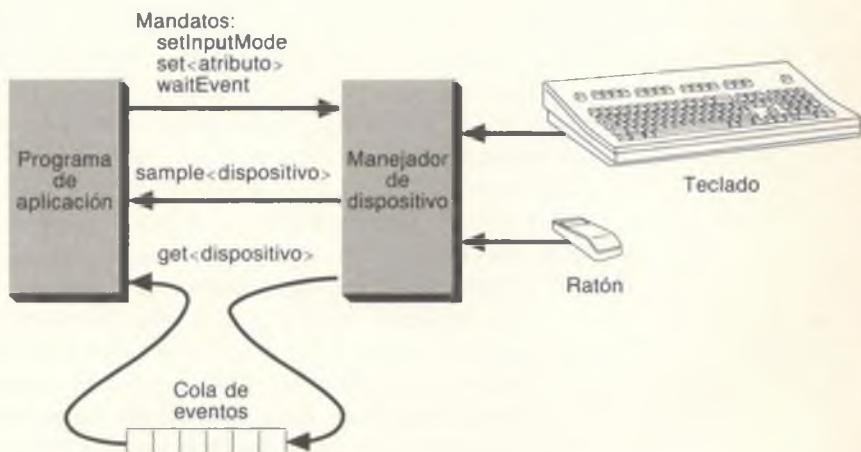


Figura 2.11 Comparación entre el muestreo y el manejo de eventos usando la cola de eventos.

muestreo, pero usted ya está familiarizado con una técnica similar utilizada en la función `scanf` de un programa C interactivo: C activa el teclado y la aplicación espera en `scanf` hasta que el usuario haya terminado de introducir una línea de texto. Algunos ambientes permiten que el enunciado `scanf` tenga acceso a los caracteres que se teclearon e introdujeron a la cola antes de emitir la llamada a `scanf`.

Los programas sencillos dirigidos por eventos en SRGP y en paquetes similares siguen la interacción reactiva *alternante* presentada en la sección 1.4.3 y descrita con seudocódigo en el programa 2.4; esta interacción se puede modelar de manera conveniente como un autómata de estado finito. En el capítulo 8 se analizan estilos de interacción más complejos, que permiten actividades simultáneas del programa y el usuario.

#### **Programa 2.4**

*Esquema de interacción dirigida por eventos*

```
asignar valores iniciales, incluyendo la generación de la imagen inicial;
activar dispositivos interactivos en modo de eventos;
do {           /* ciclo de evento principal */
    esperar evento accionado por usuario en alguno de los dispositivos;
    switch (qué dispositivo ocasionó el evento){
        case DISPOSITIVO_1: recopilar datos de medición de DISPOSITIVO_1, procesar,
                             responder;
        case DISPOSITIVO_2: recopilar datos de medición de DISPOSITIVO_2, procesar,
                             responder;
    }
}
while (usuario no solicita salir);
```

Las aplicaciones dirigidas por eventos normalmente pasan la mayor parte del tiempo en un estado de espera, ya que la interacción es dominada por el *tiempo de pensamiento* durante el cual el usuario decide qué hacer a continuación; incluso en aplicaciones de juego de ritmo muy rápido, el número de eventos que el usuario puede generar en un segundo es una fracción de lo que puede manejar la aplicación. Como SRGP suele implantar el modo de eventos con interrupciones reales (de hardware), el estado de espera en realidad no utiliza tiempo de la UCP. La ventaja es obvia en un sistema multitarea: la aplicación de modo de eventos requiere tiempo de UCP únicamente en pequeñas ráfagas de actividad que siguen a las acciones del usuario, con lo cual la UCP está libre para dedicarse a otras tareas.

Es necesario mencionar otra cuestión relacionada con el empleo correcto del modo de eventos. Aunque el mecanismo de cola permite que el programa y el usuario operen de manera asíncrona, no se debe permitir que el usuario se adelante demasiado al programa, ya que cada evento debe producir eco y cierta realimentación por parte del programa de aplicación. Es cierto que los usuarios experimentados han aprendido a teclear por adelantado para especificar parámetros como nombres de archivo o incluso mandatos de sistema operativo mientras el sistema está procesando solicitudes anteriores, sobre todo si se proporciona de inmediato al menos un eco carácter por carácter. Por otra parte, el

**uso adelantado del ratón** para mandatos gráficos generalmente no es tan útil (pero sí mucho más peligroso), ya que el usuario necesita ver la pantalla actualizada para reflejar el estado actual del modelo de la aplicación antes de la siguiente interacción gráfica.

## 2.2.4 Modo de muestreo

**Activación, desactivación y establecimiento del modo de un dispositivo.** La siguiente función se utiliza para activar o desactivar un dispositivo, usando como parámetros un dispositivo y un modo:

```
void SGRP_SetInputMode (inputDevice LOCATOR / KEYBOARD,
                        inputMode INACTIVE / SAMPLE / EVENT);
```

De esta manera, para establecer el localizador en modo de muestreo hacemos la llamada

```
SGRP_SetInputMode (LOCATOR, SAMPLE);
```

Inicialmente, los dos dispositivos están inactivos. La colocación de un dispositivo en un modo no afecta de ninguna manera al otro: ambos pueden estar activos de manera simultánea e incluso en modos diferentes.

**La medida del localizador.** El localizador es una abstracción lógica de un ratón o una tableta de datos, que devuelve la posición del cursor como un par de coordenadas ( $x, y$ ) en la pantalla, el número del botón que más recientemente experimentó una transición y el estado de los botones como un arreglo de **acorde** (en analogía a un acorde musical, ya que varios botones pueden estar presionados al mismo tiempo). El segundo campo permite que la aplicación sepa cuál fue el botón que accionó el evento.

```
typedef struct {
    point position;
    enum {
        UP, DOWN
    } buttonChord [MAX_BUTTON_COUNT];      /* Usualmente 1 a 3*/
    int buttonOfMostRecentTransition;
} locatorMeasure;
```

Después de activar el localizador en el modo de muestreo con la función *SGRP\_SetInputMode*, podemos solicitar su medida actual usando

```
void SGRP_SampleLocator (locatorMeasure *medida);
```

Examinemos la aplicación prototipo de muestreo que se presenta en la figura 2.5, un sencillo ciclo de pintura que sólo comprende el botón 1 del localizador.

Esta actividad de pintura implica dejar un trazo de pintura donde el usuario ha arrastrado el localizador mientras oprime el botón 1; el localizador se muestrae en un lazo mientras el usuario lo mueve. Primero hay que detectar cuándo comienza a pintar el usuario, muestreando el botón hasta que se haya oprimido; después colocamos la pintura (un rectángulo relleno) en nuestro sencillo ejemplo) en cada punto de muestra hasta que el usuario suelte el botón.

#### **Programa 2.5**

*Ciclo de muestreo para pintar.*

```

establecer atributos de color y patrones y el tamaño del pincel en mitad_altura_pincel y
mitad_anchura_pincel
SRGP_setInputMode (LOCATOR, SAMPLE);

/* Primero muestrear hasta que se oprima el botón */
do {
    SRGP_sampleLocator (medida_localizador);
} while (medida_localizador.buttonChord[0] == UP);

/* Ciclo de pintura:
   Colocar continuamente el pincel y muestrear hasta que se libere el botón */
do {
    rectángulo = SRGP_defRectangle (medida_localizador.position.x - mitad_anchura_
pincel,
                                    medida_localizador.position.y - mitad_altura_pincel,
                                    medida_localizador.position.x + mitad_anchura_pincel,
                                    medida_localizador.position.y + mitad_altura_pincel);
    SRGP_fillRectangle (rectángulo);
    SRGP_sampleLocator (&medida_localizador),
} while (medida_localizador.buttonChord[0] == DOWN);

```

Los resultados de esta secuencia son un tanto burdos: los rectángulos pintados están a distancias arbitrarias y su densidad depende por completo de cuán se desplazó el localizador entre muestreos consecutivos. La tasa de muestreo está determinada en esencia por la velocidad a la que la UCP ejecuta el sistema operativo, el paquete y la aplicación.

El modo de muestreo está disponible para los dos tipos de dispositivo lógico; sin embargo, el teclado casi siempre se opera en modo de eventos, por lo que no veremos aquí las técnicas para muestrearlo.

### **2.2.5 Modo de eventos**

**Utilización del modo de eventos para iniciar el ciclo de muestreo.** Los dos ciclos de muestreo del ejemplo de pintura (uno para detectar la transición de botón presionado y el otro para pintar hasta que ocurra la transición de botón liberado) llevan a cabo el trabajo necesario, pero aplican una carga innecesaria a la UCP. Aunque la sobrecarga quizás no sea algo preocupante en un computador personal, no es recomendable en los sistemas que ejecutan varias tareas, y mucho menos en tiempo compartido. Si bien es definitivamente necesario muestrear varias veces el localizador en el ciclo de pintura (porque tenemos que

conocer la posición del localizador mientras el botón esté presionado), no se requiere un ciclo de muestreo para esperar el evento de botón presionado que inicia la interacción de pintura. Podemos utilizar el modo de eventos, que analizamos a continuación, cuando no se requiere la información de medición mientras se espera un evento.

**SRGP\_waitEvent.** En cualquier instante después de que SRGP\_setInputMode ha activado un dispositivo en el modo de eventos, el programa puede inspeccionar la cola de eventos ingresando al estado de espera con

```
inputDevice SRGP_waitEvent (int tiempo_máximo_espera);
```

La función regresa de inmediato si la cola no está vacía; de lo contrario, el parámetro especifica el tiempo máximo (medido en  $\frac{1}{60}$  de segundo) que debe esperar la función hasta que un evento llene la cola. Un parámetro *tiempo\_máximo\_espera* negativo (especificado con la constante simbólica INDEFINITE) hace que la función espere indefinidamente, mientras que un valor de cero ocasiona su regreso inmediato sin importar el estado de la cola.

La función devuelve la identidad del dispositivo que emitió el evento a la cabeza de la cola, como LOCATOR, KEYBOARD o NO\_DEVICE. El valor especial NO\_DEVICE es devuelto si no hubo un evento disponible en el tiempo límite especificado, es decir, si transcurrió el tiempo correspondiente al dispositivo. Después se puede probar el tipo de dispositivo para determinar cómo hay que recuperar la medida del evento (descrito más adelante en esta sección).

**El dispositivo de teclado.** El evento accionado para el dispositivo de teclado depende del **modo de procesamiento** en el cual se ha colocado el dispositivo. El modo EDIT se utiliza cuando la aplicación recibe cadenas (p. ej., nombres de archivo, mandatos) del usuario, quien teclea y edita la cadena y luego oprime la tecla Retorno (Return) para accionar el evento. En el modo RAW, utilizado para interacciones en las que hay que vigilar de cerca el teclado, cada tecla presionada inicia un evento. La aplicación usa la siguiente función para establecer el modo de procesamiento:

```
void SRGP_setKeyboardProcessingMode (keyboardMode EDIT / RAW);
```

En el modo EDIT (edición), el usuario puede teclear cadenas completas, corregirlas con la tecla de retroceso y luego usar la tecla Retorno o Enter como accionador. Este modo se emplea cuando el usuario debe teclear una cadena completa, como el nombre de un archivo o la etiqueta de una figura. Se ignoran todas las teclas de control excepto el retroceso y Retorno, y la medida es la cadena tal y como aparece al momento de accionar. Por otra parte, en el modo RAW (no interpretado) cada uno de los caracteres tecleados, incluyendo los de control, sirve como accionador y se devuelve individualmente como medida.

Este modo se utiliza cuando los caracteres individuales del teclado sirven como mandatos, por ejemplo para mover el cursor, para operaciones de edición sencillas o para acciones en juegos de video. El modo RAW no ofrece eco, mientras que el modo EDIT presenta en pantalla el eco de la cadena y muestra un **cursor de texto** (como un carácter de subrayado o de bloque) en el lugar donde aparecerá el siguiente carácter que se teclee. Cada retroceso mueve hacia atrás el cursor de texto y borra un carácter.

Cuando SRGP\_waitEvent devuelve el código de dispositivo KEYBOARD, la aplicación obtiene la medida relacionada con el evento invocando

```
void SRGP_getKeyboard (char *medida, int tamaño_memoria);
```

Cuando el dispositivo de teclado está activo en modo RAW, su medida siempre tiene un carácter de longitud. En este caso, el primer carácter de la cadena de medida es el que devuelve la medición de modo RAW.

El programa 2.6 muestra la utilización del modo EDIT. Recibe una lista de nombres de archivos del usuario y elimina cada uno de ellos conforme los recibe. La interacción termina cuando el usuario marca una cadena nula (oprimiendo Retorno sin teclear otros caracteres). Durante la interacción, el programa espera indefinidamente a que el usuario teclee la siguiente cadena.

Aunque este código especifica de manera explícita dónde aparecerá la solicitud de la cadena, no indica dónde se teclea la cadena de entrada del usuario (y dónde se corrige con la tecla de retroceso). El programador especifica la ubicación de este eco tecleado, como veremos en la sección 2.2.7.

**El dispositivo localizador.** El evento accionador para el dispositivo localizador es la presión o la liberación de un botón del ratón. Cuando SRGP\_waitEvent devuelve el código LOCATOR para el dispositivo, la aplicación obtiene la medida relacionada con el evento mediante la llamada

```
void SRGP_getLocator (locatorMeasure *medida);
```

Por lo general, el campo **position** (posición) del localizador se usa para determinar en qué área de la pantalla el usuario designó el punto. Por ejemplo, si el cursor del localizador se halla en una región rectangular en la cual se presentó un botón de menú, el evento se debe interpretar como la solicitud de una acción; si se encuentra en el área de dibujo principal, el punto podría estar dentro de un objeto previamente dibujado para indicar su selección, o bien en una región vacía para indicar dónde se debe colocar un nuevo objeto.

#### Programa 2.6

Interacción de teclado en modo EDIT.

```
# define TAMAÑO_MEDIDA_TECLADO 80
SRGP_setInputMode (KEYBOARD, EVENT);           /* Suponga que sólo hay un teclado activo */
SRGP_setKeyboardProcessingMode (EDIT);
punto = SRGP_defPoint (100, 100);
```

```

SRGP_text (punto, "Especifique uno o más archivos que desee eliminar; oprima Retorno para
salir\n");
/* ciclo principal del evento */
do {
    dispositivo_entrada = SRGP_waitEvent (INDEFINITE);
    SRGP_getKeyboard (medida, TAMAÑO_MEDIDA_TECLADO);
    if (strcoll (medida, ""))
        eliminar_archivo (medida); /* eliminar_archivo lleva a cabo la confirmación, etcétera */
}
while (strcoll (medida, ""));

```

El seudocódigo que se presenta en el programa 2.7 (similar al que se mostró previamente para el teclado) implanta otro uso del localizador, permitiendo que el usuario especifique puntos en los cuales deben colocarse marcas. El usuario concluye el ciclo de colocación de marcas oprimiendo el botón del localizador mientras el cursor apunta a un botón de la pantalla, un rectángulo que contiene el texto *salir*.

En este ejemplo sólo tienen importancia las veces que el usuario oprime el botón 1 del localizador; se ignoran las veces que lo suelta. Observe que el botón debe estar liberado antes de que pueda ocurrir el siguiente evento de botón oprimido: el evento se activa con la transición, no con el estado del botón. Además, para asegurar que todos los eventos que provengan de otros botones no alteren esta interacción, la aplicación indica a SRGP cuáles son los botones que accionarán un evento del localizador, emitiendo la llamada

```
void SRGP_setLocatorButtonMask (int botones_activos);
```

Los valores para la mascarilla del botón son LEFT\_BUTTON\_MASK, MIDDLE\_BUTTON\_MASK y RIGHT\_BUTTON\_MASK, que corresponden al botón izquierdo, central y derecho, respectivamente. Una mascarilla compuesta se forma con un *or* lógico de los valores individuales. La mascarilla por omisión del botón del localizador es 1 pero, sin importar cuál sea la mascarilla, todos los botones siempre tienen medida. En las implantaciones que apoyan menos de tres botones, las referencias a los botones inexistentes son ignoradas por SRGP y sus mediciones siempre contienen UP (arriba).

La función *eligio\_botón\_salida* compara la posición de medida con los límites del rectángulo del botón de salida y devuelve un valor booleano que indica si el usuario eligió el botón de salida. Este proceso es un ejemplo sencillo de la **correlación de selección** descrita en la sección 2.2.6.

#### Programa 2.7

Interacción con el  
localizador.

```

#define SALIDA 0
crear el botón de salida en la pantalla;
SRGP_setLocatorButtonMask (LEFT_BUTTON_MASK);
SRGP_setInputMode (LOCATOR, EVENT); /* Suponga que sólo está activo el localizador */
/* ciclo principal del evento */
terminar = FALSE;
do {

```

```

dispositivo_entrada = SRGP_waitEvent (INDEFINITE);
SRGP_getLocator (&medida);
if (measure.buttonChord[SALIDA] == DOWN) {
    if eligió_boton_salida (measure.position) terminar = TRUE;
    else
        SRGP_marker (measure.position);
}
}
while (!terminar);

```

**Espera de eventos múltiples.** Los fragmentos de código en los programas 2.6 y 2.7 no ilustran la ventaja principal del modo de eventos: la capacidad para esperar a más de un dispositivo al mismo tiempo. SRGP establece la cola de los dispositivos activados en orden cronológico y permite que el programa de aplicación saque el primero de la cola cuando se llama a SRGP\_waitEvent. A diferencia de las interrupciones de hardware, que se procesan de acuerdo con su prioridad, los eventos se procesan estrictamente en el orden temporal. La aplicación examina el código de dispositivo devuelto para determinar cuál fue el dispositivo que ocasionó el evento.

La función presentada en la figura 2.8 permite que el usuario coloque cualquier cantidad de marcas circulares dentro de un área de dibujo rectangular. El usuario coloca la marca apuntando a la posición deseada y oprimiendo el botón 1; para solicitar el fin de la interacción, puede oprimir el botón 3 o teclear “s” o “S”.

Programa 2.8

Utilización simultánea de varios dispositivos.

```

#define BOTÓN_COLOCACIÓN 0
#define BOTÓN_SALIDA 2

generar distribución inicial de la pantalla;
SRGP_setInputMode (KEYBOARD, EVENT);
SRGP_setKeyboardProcessingMode (RAW);
SRGP_setInputMode (LOCATOR, EVENT);
SRGP_setLocatorButtonMask (LEFT_BUTTON_MASK | RIGHT_BUTTON_MASK);
/* Ignorar el segundo botón */

/* Ciclo principal del evento */
terminar = FALSE;
do {
    dispositivo = SRGP_waitEvent (INDEFINITE);
    switch (dispositivo) {
        case KEYBOARD:
            SRGP_getKeyboard (keyMeasure, mem_int);
            terminar = (keyMeasure[0] == 's' || (keyMeasure[0] == 'S'));
            break;
        case LOCATOR: {
            SRGP_getLocator (&locMeasure);
            switch (locMeasure.buttonOfMostRecentTransition) {
                case BOTÓN_COLOCACIÓN:

```

```

if ((locMeasure.buttonChord[BOTÓN_COLOCACIÓN] == DOWN)
    && en área_dibujo (lockMeasure.position))
    SRGPMarker (locMeasure.position);
break;
case BOTÓN_SALIDA:
terminar = TRUE;
break;
} /* case del botón */
} /* case del localizador */
} /* case del dispositivo */
}
while (!terminar);

```

## 2.2.6 Correlación de selección para el manejo de interacción

Una aplicación gráfica usualmente divide el área de la pantalla en regiones dedicadas a propósitos específicos. Cuando el usuario oprime el botón del localizador, la aplicación debe determinar precisamente cuál fue el botón, ícono u objeto en la pantalla que se seleccionó, para que pueda responder en forma apropiada. Esta determinación, denominada **correlación de selección**, es parte fundamental de los gráficos interactivos.

Un programa de aplicación que utiliza SRGPM lleva a cabo la correlación de selección determinando en qué región está ubicado el cursor y después cuál fue el objeto, de existir, que eligió el usuario en dicha región. Los puntos en una subregión vacía podrían ignorarse (p. ej., si el punto está entre botones en un menú) o podría especificarse la posición para un nuevo objeto (si el punto está en el área de dibujo principal). Como gran número de las regiones de la pantalla son rectángulos verticales, casi todo el trabajo correspondiente a la correlación de selección se puede realizar con una función booleana sencilla y de uso frecuente que revisa si un punto determinado se halla en un rectángulo específico. El paquete GEOM distribuido con SRGPM incluye esta función (GEOM\_ptInRect) así como otras utilerías para la aritmética de coordenadas (consulte la sección 7.11.2 para conocer más acerca de la correlación de selección).

Veamos un ejemplo clásico de la correlación de selección. Considere una aplicación de pintura con una **barra de menú** en la parte superior de la pantalla. Esta barra contiene los nombres de varios menús descendentes, llamados **encabezados** de menús. Cuando un usuario selecciona un encabezado (colocando el cursor sobre la cadena de texto del encabezado y oprimiendo un botón del localizador) se presenta el **cuerpo de menú** correspondiente en la pantalla, debajo del encabezado, y se realza éste. Cuando el usuario selecciona una entrada del menú (soltando el botón del localizador), el cuerpo del menú desaparece y deja de realzarse el encabezado. El resto de la pantalla contiene el área de dibujo principal en la cual el usuario puede colocar y seleccionar objetos. Al crear cada uno de estos objetos, la aplicación le asigna un identificador (ID) entero único positivo que devuelve la función de correlación de selección para el procesamiento subsecuente del objeto.

**Programa 2.9**

*Esquema de interacción de alto nivel para el manejo de menús.*

```

void manejador_interacción_alto_nivel (locatorMeasure medida_localizador)
{
    if GEOM_pointInRect (medida_localizador.position, extensión_barra_menu) {
        /* Averiguar si el usuario seleccionó un encabezado de menú y cuál fue;
         * después presentar el cuerpo del menú */
        ID_menu = correlación_barra_menu (medida_localizador.position);
        if (ID_menu > 0) {
            índice_elemento_elegido = efectuar_interacción_menu_descendente
                (ID_menu);
            if (índice_elemento_elegido > 0)
                efectuar_acción_elegida_del_menu (ID_menu, índice_elemento_elegido);
        }
    } else /* El usuario seleccionó dentro del área de dibujo; detectar qué fue y responder */
    {
        ID_objeto = correlación_área_dibujo (medida_localizador.position);
        if (ID_objeto > 0) procesar_objeto (ID_objeto);
    }
}
}

```

Cuando se obtiene un punto de localizador a través de un evento de botón presionado, se ejecuta el esquema de manejo de interacción de alto nivel presentado en el programa 2.9; en esencia, se trata de un procedimiento de reenvío que utiliza la correlación de selección en la barra de menú o en el área de dibujo principal para dividir el trabajo entre las funciones de menú y de selección de objetos. En primer lugar, si el cursor estaba en la barra de menú, un procedimiento de correlación subsidiario determina si el usuario seleccionó un encabezado de menú. De ser así, se llama a un procedimiento (detallado en la sección 2.3.1) para llevar a cabo la interacción del menú, el cual devuelve un índice que especifica cuál fue el elemento que se eligió en el cuerpo del menú. El identificador del menú y el índice del elemento identifican en forma única la acción que debe emprenderse como respuesta. Si el cursor no estaba en la barra de menú sino en el área de dibujo principal, se llama a otro procedimiento de correlación subsidiario para determinar cuál (si acaso) fue el objeto seleccionado. Si se seleccionó un objeto, se llama a un procedimiento de procesamiento para que responda de manera apropiada.

La función *correlación\_barra\_menu* lleva a cabo una correlación más fina llamando a *GEOM\_pointInRect* una vez para cada encabezado de menú en la barra; accede una estructura de datos que almacena la extensión de pantalla rectangular de cada encabezado. La función *correlación\_área\_dibujo* debe efectuar una correlación más compleja ya que, por lo general, los objetos en el área de dibujo pueden estar superpuestos y no son necesariamente rectangulares.

## 2.2.7 Determinación de medidas y atributos de dispositivos

Cada dispositivo de entrada tiene su propio conjunto de atributos y la aplicación los puede establecer para adaptar a la medida la realimentación que el dispositivo presenta al usuario (la mascarilla de botón que se presentó previamente también es un atributo; difiere de los otros que hemos presentado en que no

afecta la realimentación). Los atributos de dispositivo de entrada se establecen modalmente con funciones específicas, igual que los atributos de las primitivas de salida. Los atributos se pueden asignar en cualquier momento, esté o no activo el dispositivo.

Así mismo, cada medida de los dispositivos de entrada, normalmente determinada por las acciones del usuario, también puede ser asignada por la aplicación. A diferencia de los atributos de los dispositivos de entrada, la medida de éstos se restablece a un valor por omisión cuando se desactiva el dispositivo; de esta manera, al reactivarlo, los dispositivos tienen valores iniciales previsibles, algo muy conveniente para el programador y para el usuario. Este restablecimiento automático puede omitirse con la asignación explícita de la medida de un dispositivo mientras esté inactivo.

**Atributos del eco del localizador.** Varios tipos de eco son útiles para el localizador. El programador puede controlar el tipo de eco y la forma del cursor con

```
void SRGP_setLocatorEchoType (echoType NO_ECHO / CURSOR /  
    RUBBER_LINE / RUBBER_RECT);
```

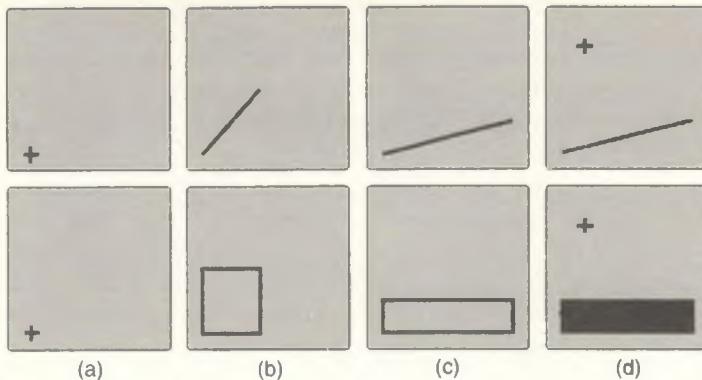
El valor por omisión es CURSOR y las implantaciones de SRGP proporcionan una tabla de cursores de la cual una aplicación selecciona una forma determinada (véase el manual de referencia). Una utilización común de la habilidad para especificar dinámicamente la forma del cursor es proporcionar realimentación alternando la forma del cursor de acuerdo con la región donde se encuentre. Los ecos RUBBER\_LINE (línea elástica) y RUBBER\_RECT (rectángulo elástico) suelen utilizarse para especificar una línea o una caja, respectivamente. Si estos ecos están activados, SRGP dibuja de manera automática una línea o un rectángulo continuamente actualizado conforme el usuario mueve el localizador. La línea o el rectángulo está definido por dos puntos: el punto ancla (otro atributo del localizador) y la posición actual del localizador. En la figura 2.12 se ilustra el uso de estos dos modos para una línea o rectángulo especificado por el usuario.

En la figura 2.12(a), el eco es un cursor en forma de cruz y el usuario está a punto de oprimir el botón del localizador. La aplicación inicia un eco elástico, anclado en la posición actual del cursor, como respuesta a la presión del botón. En las partes (b) y (c) se responde al movimiento del dispositivo localizador con un eco de la primitiva elástica. La posición del localizador en la parte (c) es devuelta cuando el usuario suelta el botón y la aplicación responde dibujando una primitiva de línea o rectángulo y restaurando el eco normal del cursor (véase la parte d).

El punto ancla para el eco elástico se establece con

```
void SRGP_setLocatorEchoRubberAnchor (point posición);
```

Una aplicación por lo general usa el campo *position* (posición) de la medida obtenida del evento más reciente de presión de botón del localizador, ya que esta acción es la que suele dar inicio a la secuencia de eco elástico.



**Figura 2.12** Escenas del eco elástico: (a) al oprimir el botón se inicia el eco; (b) la primitiva elástica presenta el eco del dispositivo localizador; (c) se devuelve a la aplicación la posición del localizador; (d) la aplicación dibuja la línea y restaura el eco.

**Control de medida del localizador.** La porción *position* de la medida del localizador se restablece automáticamente en el centro de la pantalla cuando se desactiva el localizador. A menos que el programador la restablezca de manera explícita, la medida (y la posición de realimentación, si está activo el eco) reciben como valor inicial la misma posición cuando se reactiva el dispositivo. En cualquier instante, se encuentre activo o inactivo el dispositivo, el programador puede restablecer la medida del localizador (la porción *position*, no los campos relacionados con los botones) usando

```
void SRGP_setLocatorMeasure (point posición);
```

El restablecimiento de la medida mientras el localizador está inactivo no tiene efecto inmediato sobre la pantalla, pero si se restablece cuando el localizador se encuentra activo, el eco (si lo hay) varía en forma correspondiente. Por ende, si el programa quiere que el cursor aparezca inicialmente en una posición distinta del centro de la pantalla al activar el localizador, una llamada a `SRGP_setLocatorMeasure` con la posición inicial deseada debe preceder a la llamada a `SRGP_setInputMode`. Esta técnica es de uso común para lograr la continuidad de la posición del cursor: se almacena la última medida antes de desactivar el localizador y se devuelve el cursor a dicha posición cuando se reactiva.

**Control de atributos y medida del teclado.** A diferencia del localizador, cuyo eco se ubica para reflejar el movimiento de un dispositivo físico, no existe una posición obvia para el eco de un dispositivo de teclado. Por consiguiente, la posición es un atributo (con un valor por omisión específico para la implantación) del dispositivo de teclado y puede establecerse con

```
void SRGP_setKeyboardEchoOrigin (point origen);
```

La medida por omisión para el teclado se restablece de manera automática a la cadena nula cuando se desactiva el teclado. La asignación explícita de la medida a un valor inicial distinto de nulo justo antes de activar el teclado es una manera conveniente de presentar una cadena de entrada por omisión (mostrada por SRGP al momento de comenzar el eco), la cual el usuario puede aceptar o modificar antes de oprimir la tecla Retorno, ahorrando esfuerzo de tecleado. La medida del teclado, una cadena de caracteres, se establece con

```
void SRGP_setKeyboardMeasure (char *medida);
```

## 2.3 Características de los gráficos de barrido

Hasta ahora hemos presentado varias de las características de SRGP. En esta sección se analizan los demás recursos que aprovechan en forma especial el hardware de barrido, sobre todo la capacidad para almacenar y restaurar pedazos de la pantalla al sobreponerse otras imágenes, como ventanas o menús temporales. Estas manipulaciones de imágenes se llevan a cabo bajo el control de los programas de aplicación administradores de ventanas y de menús. También presentaremos arreglos bidimensionales de bits fuera de la pantalla para el almacenamiento de ventanas y menús, y analizaremos la utilización de rectángulos de recorte.

### 2.3.1 Lienzos

La mejor manera de lograr que aparezcan y desaparezcan con rapidez iconos o menús complejos es crearlos una vez en la memoria y después copiarlos a la pantalla cuando sea necesario. Los paquetes de gráficos de barrido lo hacen generando las primitivas en mapas de bits o de pixeles del tamaño adecuado, invisibles, fuera de la pantalla, llamados **lienzos** en SRGP, para luego copiarlos de o hacia la memoria de la pantalla. De hecho, esta técnica es un tipo de memoria intermedia. El movimiento de bloques de pixeles por lo general es más rápido que la regeneración de la información, dada la existencia de la rápida operación SRGP\_copyPixel que analizaremos dentro de poco.

Un lienzo de SRGP es una estructura de datos que almacena una imagen como un arreglo bidimensional de pixeles. También almacena cierta información de control relacionada con el tamaño y los atributos de la imagen. Cada lienzo representa una imagen en su propio sistema de coordenadas cartesianas, idéntico al de la pantalla que se presenta en la figura 2.1; de hecho, la pantalla es un lienzo, que sólo se distingue por ser el único que se presenta. Para hacer visible una imagen almacenada en un lienzo fuera de pantalla, la aplicación la debe copiar al lienzo de la pantalla. Sin embargo, antes de hacerlo es posible almacenar la porción de la imagen de la pantalla que será sustituida por la nueva imagen (un menú, por ejemplo), copiando los pixeles de la región a un lienzo

fueras de pantalla. Al efectuarse la selección del menú, la imagen en la pantalla se restaura copiando de regreso estos pixeles.

En cualquier instante sólo existirá un lienzo *activo*: el lienzo en el cual se dibujan las nuevas primitivas y al cual se aplican los valores de los atributos. Este lienzo puede ser el de la pantalla (el que hemos usado por omisión) o uno fuera de pantalla. Las coordenadas que se envían a las funciones primitivas se expresan en términos del espacio de coordenadas locales del lienzo activo. Cada lienzo tiene además su propio conjunto de atributos de SRGP, que afectan todos los dibujos y se establecen con los valores iniciales por omisión al crear el lienzo. Las llamadas a las funciones de asignación de atributos únicamente modifican los atributos en el lienzo activo. Es conveniente pensar en un lienzo como una pantalla virtual de dimensiones especificadas por el programa, con su propio mapa de pixeles, sistema de coordenadas y grupo de atributos. Estas propiedades de un lienzo se conocen como su **estado o contexto**.

Al asignar valores iniciales a SRGP, se crea y activa automáticamente el **lienzo de pantalla**. Todos los programas que hemos presentado hasta ahora generan primitivas únicamente para ese lienzo. Es el único visible en la pantalla y su identificador es SCREEN\_CANVAS, una constante de SRGP. Un lienzo nuevo, fuera de pantalla, se crea con una llamada a la función siguiente, que devuelve el identificador (entero) asignado al nuevo lienzo:

```
int SRGP_createCanvas (int anchura, int altura);
```

Al igual que la pantalla, el origen (0, 0) del sistema de coordenadas locales del nuevo lienzo se encuentra en la esquina inferior izquierda, y la parte superior derecha se halla en (*anchura*-1, *altura*-1). Por lo tanto, un lienzo de 1 por 1 se define con una anchura y una altura de 1, y sus esquinas inferior izquierda y superior derecha son (0, 0). Esto es congruente con nuestro tratamiento de los pixeles como intersecciones de la malla: el único pixel en un lienzo de 1 por 1 se encuentra en (0, 0).

Un lienzo recién creado se activa automáticamente y sus pixeles reciben el color inicial 0 (como se hace con el lienzo de pantalla antes de presentar primitivas). Una vez que se ha creado el lienzo no es posible modificar su tamaño. Además, el programador no puede controlar el número de bits por pixel en un lienzo, ya que SRGP utiliza todos los bits por pixel que permita el hardware. Los atributos de un lienzo se mantienen como parte de su información de estado *local*; así, el programa no tiene que guardar en forma explícita los atributos del lienzo activo antes de crear otro.

La aplicación selecciona un lienzo previamente creado para que sea el activo con la función

```
void SRGP_useCanvas (int ID_lienzo);
```

La activación de un lienzo no implica que se haga visible; para esto es necesario copiar la imagen de un lienzo fuera de pantalla al de la pantalla (usando la función SRGP\_copyPixel que se describe más adelante).

Los lienzos se eliminan con la siguiente función, que no puede emplearse para eliminar el lienzo de pantalla ni el que se encuentre activo:

```
void SRGP_deleteCanvas (int ID_lienzo);
```

Las funciones siguientes permiten consultar el tamaño del lienzo; una devuelve el rectángulo que define el sistema de coordenadas (el punto inferior izquierdo siempre es (0, 0)) y la otra devuelve la anchura y la altura como cantidades separadas.

```
rectangle SRGP_inquireCanvasExtent (int ID_lienzo);
void SRGP_inquireCanvasSize (int ID_lienzo, *anchura, *altura);
```

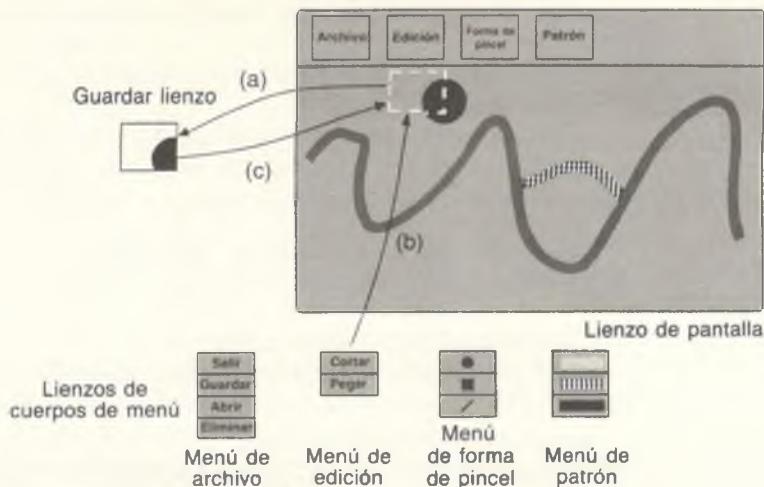
Examinemos la forma de utilizar los lienzos para implantar *efectuar\_interacción\_menué\_descendente* (*PerformPulldownMenulInteraction*), la función llamada por el manejador de interacción de alto nivel presentado en el programa 2.9. La función se implanta con el seudocódigo del programa 2.10 y su secuencia de acciones se ilustra en la figura 2.13. Cada menú tiene un identificador único (indicado por la función *correlación\_barra\_menué* (*CorrelateMenuBar*)) que puede usarse para localizar un registro de datos que contenga la siguiente información acerca de la apariencia del cuerpo del menú.

- El identificador del lienzo que almacena el cuerpo del menú.
- El área rectangular (llamada *extensión\_pantalla\_cuerpo\_menué* en el seudocódigo) especificada en coordenadas de lienzo de pantalla, donde el cuerpo del menú descendente debe aparecer cuando el usuario lo active al oprimir el botón estando el cursor sobre el encabezado.

#### Programa 2.10

Seudocódigo para  
*efectuar\_interacción\_menué\_descendente*.

```
int efectuar_interacción_menué_descendente (int ID_menué);
/* El almacenamiento/copiado de regiones rectangulares de los lienzos se describe más
adelante */
{
    realizar el encabezado de menú en la barra;
    extensión_pantalla_cuerpo_menué = rectángulo del área de la pantalla donde debe
    aparecer el cuerpo del menú
    guardar los pixeles actuales de extensión_pantalla_cuerpo_menué en un lienzo temporal
    /* Véase Fig. 2.13a */
    copiar imagen del cuerpo del menú del lienzo del cuerpo a extensión_pantalla_cuerpo_
    menué
    /* Véase Fig. 2.13b y código C en el programa 2.11 */
    esperar la señal de botón liberado que indica que el usuario hizo una selección, obtener
    después la medida del localizador
    copiar imagen almacenada del lienzo temporal de regreso a extensión_pantalla_cuerpo_menué
    /* Véase Fig. 2.13c*/
    if (GEOM__pointInRect (medida_localizador.position, extensión_pantalla_cuerpo_menué)
        calcular y devolver índice del elemento elegido usando la coordenada y la posición medida
    else
        return 0;
```



**Figura 2.13** Almacenamiento y restauración del área cubierta por un cuerpo de menú.

### 2.3.2 Rectángulos de recorte

Para proteger otras porciones del lienzo, con frecuencia es deseable restringir el efecto de las primitivas gráficas a una subregión del lienzo activo. Para que esta operación sea más sencilla, SRGP mantiene un atributo de **rectángulo de recorte**. Todas las primitivas se recortan a las fronteras de este rectángulo; es decir, no se dibujan las primitivas (o sus porciones) que se encuentren fuera del rectángulo de recorte. Como los demás atributos, el rectángulo de recorte se puede modificar en cualquier momento y su valor más reciente se almacena en el grupo de atributos del lienzo. El rectángulo de recorte por omisión (el que hemos utilizado hasta ahora) es todo el lienzo; puede hacerse más pequeño, pero no se permite que sea mayor que las fronteras del lienzo. Las llamadas pertinentes para establecer y consultar el rectángulo de recorte son

```
void SRGP_setClipRectangle (rectangle rectángulo_recorte);
rectangle SRGP_inquireClipRectangle ();
```

Una aplicación de pintura como la que se presentó en la sección 2.2.4 usaría el rectángulo de recorte para limitar la colocación de pintura a la región de dibujo de la pantalla, asegurando que no se alteren las áreas de menú alrededor de ella. Aunque SRGP sólo ofrece una frontera de recorte formada por un rectángulo vertical, algunos paquetes de software más elaborados, como PostScript, permiten usar regiones de recorte de formas arbitrarias.

### 2.3.3 La operación SRGP\_copyPixel

El poderoso mandato SRGP\_copyPixel es un mandato de barrido típico que también se conoce como bitBlt (transferencia de bloque de bits) o pixBlt (transferencia de bloque de pixeles) cuando se implanta directamente en hardware; estuvo disponible por vez primera en forma de microcódigo en la estación de trabajo pionera ALTO en Xerox Palo Alto Research Center, a principios de la década de 1970 [INGA81]. Este mandato se usa para copiar un arreglo de pixeles de una región rectangular de un lienzo, la región *fuente*, a una región *destino* en el lienzo activo (véase la Fig. 2.14). El recurso en SRGP sólo ofrece una funcionalidad limitada, ya que el rectángulo destino debe ser del mismo tamaño que el fuente. En las versiones más poderosas, la fuente puede copiarse a una región destino de tamaño diferente, escalándose automáticamente para que quede pa. Además, pueden ofrecerse características adicionales, como **mascarillas** para bloquear selectivamente del copiado ciertos pixeles fuente o destino deseados, y **patrones de medios tonos** que pueden usarse para **sombreado** la región destino.

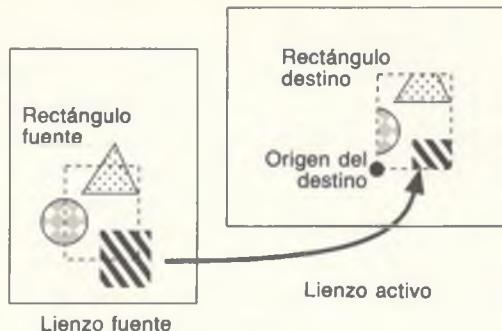
SRGP\_copyPixel puede copiar entre dos lienzos cualesquiera y se especifica de la siguiente manera:

```
void SRGP_copyPixel (int ID_lienzo_fuente, rectangle rectángulo_fuente,  
point vértice_destino);
```

El *rectángulo\_fuente* especifica la región fuente en un lienzo arbitrario y *vértece\_destino* indica el vértice inferior derecho del rectángulo destino en el lienzo activo, cada uno especificado en su propio sistema de coordenadas. La operación de copiado está limitada por el mismo rectángulo de recorte que evita que las primitivas generen pixeles en regiones protegidas del lienzo. Por consiguiente, la región a la cual se copiarán los pixeles es la intersección de la extensión del lienzo destino, la región destino y el rectángulo de recorte, como se ilustra en la región rayada de la figura 2.15.

Para ilustrar la utilización de CopyPixel en el manejo de menús descendentes, implantaremos el cuarto enunciado del seudocódigo (*copiar imagen del cuerpo del menú*) de la función *efectuar\_interacción\_menu\_descendente* (Prog. 2.10). En el tercer enunciado del seudocódigo almacenamos en un lienzo fuera de pantalla la región de la pantalla donde debe ir el cuerpo del menú; ahora queremos copiar el cuerpo a la pantalla.

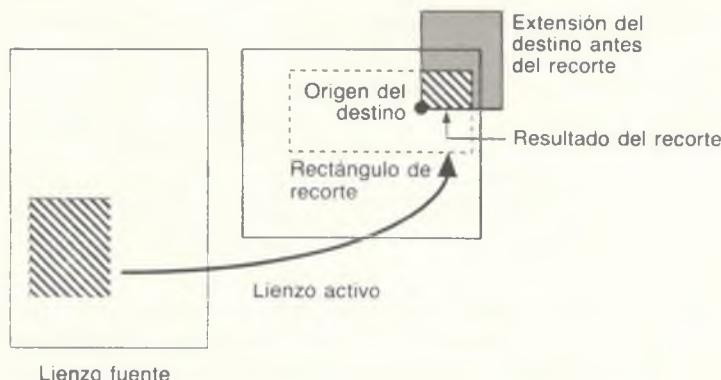
El código C se presenta en el programa 2.11. Debemos estar seguros de distinguir entre los dos rectángulos, que son de tamaño idéntico pero se expresan en sistemas de coordenadas diferentes. El primer rectángulo, que llamamos *extensión\_cuerpo\_menu* en el código, es simplemente la extensión del lienzo del cuerpo del menú en su propio sistema de coordenadas. Esta extensión se usa como rectángulo fuente en la operación SRGP\_copyPixel que coloca el menú en la pantalla. El rectángulo *extensión\_pantalla\_cuerpo\_menu* tiene el mismo tamaño y especifica en coordenadas de la pantalla la posición donde debe aparecer el cuerpo del menú; la esquina izquierda inferior de dicha extensión está horizontalmente alineada con el lado izquierdo del encabezado del menú y



**Figura 2.14** SRGP\_copyPixel.

su esquina superior derecha está a la par de la parte inferior de la barra de menú. (La figura 2.13 simboliza la extensión de la pantalla del menú de edición con un marco de guiones y la extensión de su cuerpo con un marco de línea continua.) La esquina inferior izquierda de *extensión\_pantalla\_cuerpo\_menu* sirve para especificar el destino de la operación SRGP\_copyPixel que copia el cuerpo del menú (Fig. 2.15). También es el rectángulo fuente para el almacenamiento inicial del área de pantalla que será superpuesta por el cuerpo del menú y el destino de la restauración final.

Observe que el estado de la aplicación se almacena y restaura para eliminar los efectos secundarios. Antes de copiar, asignamos el rectángulo de recorte de la pantalla a SCREEN\_EXTENT; alternativamente, podríamos asignarlo al área exacta *extensión\_pantalla\_cuerpo\_menu*.



**Figura 2.15** Recortes durante la ejecución de CopyPixel.

**Programa 2.11**

*Código para copiar el cuerpo del menú a la pantalla.*

```
/* Este fragmento de código copia una imagen de cuerpo de menú a la pantalla, en la posición
almacenada en el registro del cuerpo */

/* Guardar el ID del lienzo activo, que no tiene que ser la pantalla */
ID_lienzo = SRGP_inquireActiveCanvas ( );

/* Guardar el valor de atributo del rectángulo de recorte del lienzo de pantalla */
SRGP_useCanvas (SCREEN_CANVAS);
rectángulo_recorte = SRGP_inquireClipRectangle ( );

/* Asignar temporalmente el rectángulo de recorte para permitir la escritura de toda la pantalla */
SRGP_setClipRectangle (SCREEN_EXTENT);

/* Copiar el cuerpo del menú de su lienzo al área correspondiente debajo del encabezado en la
barra de menú */
SRGP_copyPixel (ID_lienzo_menu, extensión_cuerpo_menu, extensión_pantalla_cuerpo_
menu.llLeft);

/* Restaurar atributos de la pantalla y el lienzo activo */
SRGP_setClipRectangle (rectángulo_recorte);
SRGP_useCanvas (ID_lienzo);
```

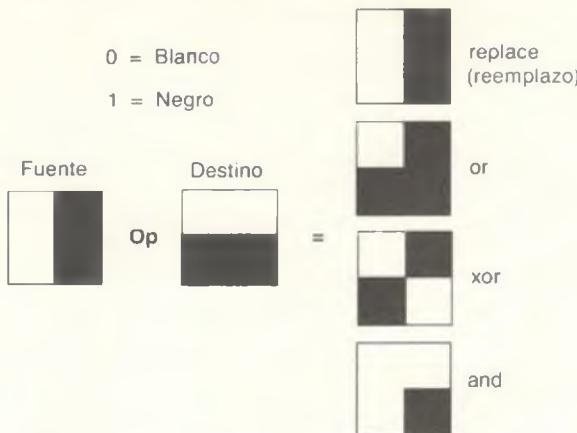
### 2.3.4 Modo de escritura u operación de barrido

SRGP\_copyPixel puede hacer mucho más que mover un arreglo de pixeles de una región fuente a un destino. También puede ejecutar una operación lógica (bit a bit) con cada par correspondiente de pixeles en las regiones fuente y destino, para luego colocar el resultado en la región destino. Esta operación se puede simbolizar como

$$D \leftarrow S \text{ op } D$$

donde **op**, con frecuencia conocida como *operación de barrido (RasterOp)* o *modo de escritura*, consiste por lo general en los 16 operadores booleanos. SRGP sólo permite utilizar los más comunes: **replace**, **or**, **xor** y **and**; estos operadores se ilustran para una imagen de un bit por pixel en la figura 2.16.

El modo de escritura no sólo afecta a SRGP\_copyPixel, sino a cualquier otra primitiva que se escriba en un lienzo. Cada pixel (ya sea de un rectángulo fuente de SRGP\_copyPixel o de una primitiva) se almacena en su localidad de memoria y se escribe con el modo destructivo **replace** (reemplazar), o su valor se combina de manera lógica con el valor previamente almacenado del pixel. (Esta combinación a nivel de bit de los valores fuente y destino es similar a la forma en que el hardware de la UCP lleva a cabo operaciones aritméticas o lógicas con el contenido de una localidad de memoria durante un ciclo de lectura-modificación-escritura de la memoria). Aunque **replace** es el modo más común, **xor** es útil para generar objetos dinámicos, como cursores y ecos elásticos, como veremos dentro de poco.



**Figura 2.16** Modos de escritura para combinar pixeles fuente y destino.

El atributo de modo de escritura se establece con

```
void SRGP_setWriteMode (writeMode WRITE_REPLACE / WRITE_XOR /
WRITE_OR / WRITE_AND);
```

Como todas las primitivas se generan de acuerdo con el modo de escritura actual, el programador de SRGP debe estar seguro de asignar este modo en forma explícita y no depender del valor por omisión de WRITE\_REPLACE.

Para saber cómo funcionan las operaciones de barrido, hay que ver la forma en que SRGP almacena y manipula en realidad los pixeles; éste es el único punto en el cual los aspectos de hardware y de la implantación alteran la perspectiva abstracta de los gráficos de barrido que hemos mantenido hasta el momento.

Las operaciones de barrido se llevan a cabo con valores de pixeles, los cuales son índices en la tabla de colores, no en las especificaciones de color de hardware almacenadas como entradas en la tabla. Por lo tanto, para un sistema de dos niveles de un bit por pixel, la operación de barrido se lleva a cabo con dos índices de un bit cada uno. En el caso de un sistema en color de ocho bits por pixel, la operación de barrido se lleva a cabo como una operación lógica bit a bit con dos índices de ocho bits.

Aunque la interpretación de las cuatro operaciones básicas con imágenes monocromáticas de un bit por pixel presentada en la figura 2.16 es bastante obvia, los resultados de las operaciones distintas de **replace** no son tan evidentes en imágenes de  $n$  bits por pixel ( $n > 1$ ) ya que una operación lógica bit a bit con los índices fuente y destino genera un tercer índice cuyo valor de color quizás no tenga ninguna relación con los colores fuente y destino.

El modo **replace** (reemplazo) implica escribir encima de lo que se encuentra en la pantalla (o en otro lienzo). Esta operación de escritura destructiva es el modo normal para dibujar primitivas y se acostumbra usarlo para mover y presentar ventanas. También puede emplearse para *borrar* primitivas viejas, dibujando encima de ellas con el patrón de fondo de la pantalla de la aplicación.

El modo **or** en las pantallas de dos niveles realiza una suma no destructiva a lo que ya existe en el lienzo. Con el color 0 como fondo blanco y el color 1 como primer plano negro, la operación **or** de un patrón de relleno gris con el fondo blanco cambia los bits subyacentes para mostrar el patrón gris, pero si la operación se aplicara con un área negra no tendría efecto en la pantalla. De esta manera, la aplicación de **or** de una mancha de pintura gris claro sobre un polígono llenado con un patrón de ladrillos únicamente rellena los ladrillos con el patrón del pincel; no borra las orillas negras de los ladrillos, como ocurriría con el modo **replace**. Es por esto que la mayoría de las tareas de pintura se llevan a cabo en el modo **or** (véase el Ejer. 2.6).

El modo **xor** en pantallas de dos niveles se puede emplear para invertir una región destino. Por ejemplo, para realizar un botón seleccionado por el usuario, establecemos el modo **xor** y generamos una primitiva de rectángulo llenado con el color 1, alternando todos los pixeles del botón:  $0 \text{ xor } 1 = 1$ ,  $1 \text{ xor } 1 = 0$ . Para restaurar el estado original del botón permanecemos en el modo **xor** y volvemos a dibujar el rectángulo, alternando los bits de regreso a su estado original. Esta técnica también la utiliza SRGP internamente para ofrecer los modos de eco de línea y rectángulo elásticos del localizador (véase el Ej. 2.1).

En muchas pantallas gráficas de dos niveles, el hardware subyacente (y en algunos casos el software) utiliza la técnica **xor** para presentar la imagen del cursor del localizador en forma no destructiva. Esta sencilla técnica presenta algunas desventajas: cuando el cursor está encima de un fondo con patrón fino que es casi 50 por ciento blanco y 50 por ciento negro, es posible que el cursor apenas sea perceptible. Por lo tanto, muchas pantallas de dos niveles y la mayoría de las pantallas en color utilizan el modo de reemplazo para el eco del cursor y emplean un cursor negro con borde blanco para que sea visible contra cualquier fondo. La incapacidad de usar **xor** complica el hardware o el software de eco (véase el Ejer. 2.4).

El modo **and** se puede utilizar, por ejemplo, para restablecer selectivamente pixeles al color 0 en la región destino, “borrándolos” de hecho.

## Ejemplo 2.1

---

**Problema:** Implementación de una interacción elástica sin emplear el eco de localizador integrado. Esté al pendiente de efectos secundarios, sobre todo al iniciar y termina el ciclo de interacción.

**Respuesta:** Implantaremos un eco de caja elástica (el eco de línea elástica se puede implantar en forma similar). Cuando el ratón se mueve y no se ha seleccionado el valor final del localizador, la función de interacción rastreará el ratón y dibujará el eco elástico. El ciclo de interacción será llamado cuando el

usuario presione un botón del ratón. La interacción permitirá arrastrar el punto actual, con el eco elástico apropiado, hasta que el usuario suelte el botón indicando el final de la interacción.

Al regresar de la función tenemos que restaurar el estado existente al momento de iniciar la función, de manera que los detalles de la implantación de la función queden aislados.

La idea fundamental para crear el eco es dibujar la forma correspondiente (rectángulo) usando el modo **xor**, de manera que al redibujar la misma forma se pueda borrar sin tener que preocuparnos por lo que se reemplazó. Para que aparezca la imagen del eco hay que dibujar la entidad una vez. Para restaurar la pantalla a su estado anterior al eco, es necesario dibujar exactamente la misma forma una segunda vez para que no deje huellas en la pantalla.

Como hay que recopilar los datos del localizador para actualizar el eco, se presenta el eco elástico antes de muestrear por vez primera el localizador y después se muestrea, se borra el eco anterior y se redibuja el eco actualizado repetidamente, en un lazo. Al muestrear se revisa el estado del botón, antes de borrar y redibujar, para determinar si debe concluir la interacción. Así mismo es conveniente, aunque no indispensable, revisar si el ratón realmente se ha movido desde el último muestreo y realizar la operación de borrado-redibujo únicamente si se ha desplazado.

El siguiente código C implanta este método para resolver el problema (no se presenta la función *botones\_oprimidos()*, pero sirve para revisar el estado de los botones del ratón):

*Código para implantar la interacción de eco elástico*

```

void interacción_rectángulo_elástico (point punto_ancla, point punto_actual,
    int bandera_arrastre, int mascarilla_botón, locatorMeasure *posición_final,
    rectangle *rectángulo_final)
{
    attributeGroup atributos;
    locatorMeasure posición_actual;
    int botón_presionado;
    rectangle rectángulo_actual;

    SRGP_inquireAttributes (&atributos);
    SRGP_setLineStyle (CONTINUOUS);
    SRGP_setFillStyle (SOLID);
    SRGP_setInputMode (LOCATOR_SAMPLE);
    SRGP_setWriteMode (WRITE_XOR);

    SRGP_setLocatorEchoType (CURSOR)           /* o NO_ECHO */
    SRGP_setLocatorMeasure (punto_actual)      /* para una buena medida */

/*
 * Queremos que el rectángulo sea insensible si el punto
 * ancla está debajo y a la izquierda del punto actual;
 * esto se asegura con la función de utilería GEOM_.
 */
    rectángulo_actual = GEOM_rectFromDiagPoints (punto_ancla, punto_actual);
}

```

```

/* Presentamos por primera vez la caja elástica: */
SRGP_rectangle (rectángulo_actual);

while (botones_oprimidos (mascarilla_botón, posición_actual.button_chord))) {
    SRGP_sampleLocator (&posición_actual);
    /* El eco se actualiza sólo si se ha movido el ratón */
    if (posición_actual.position.x != posición_actual.x || posición_actual.position.y != 
        posición_actual.y) {
        SRGP_rectangle (rectángulo_actual);
    /* Al llegar a este punto desaparece el rectángulo que dibujamos */
        punto_actual = posición_actual.position;
        rectángulo_actual = GEOM_rectFromDiagPoints (punto_ancla, punto_
            actual);
        SRGP_rectangle (rectángulo_actual);
    /* Ahora la caja del punto ancla aparece en la nueva posición */
    }
}
/* Al llegar a este punto hemos dibujado una vez la caja hasta el último punto */
/* Por lo tanto, debemos borrarla dibujándola de nuevo: */
SRGP_rectangle (rectángulo_actual);
*posición_final = posición_actual;
*rectángulo_final = rectángulo_actual;

/* Se restablece el estado existente al iniciar la función: */
SRGP_setInputMode (LOCATOR, INACTIVE);
SRGP_setAttributes (atributos);
}

```

## 2.4 Limitaciones de SRGP

Aunque SRGP es un paquete poderoso que apoya muchas aplicaciones, tiene limitaciones inherentes que impiden su utilización óptima para varias aplicaciones. La más notoria es que SRGP no ofrece apoyo para aplicaciones que presentan geometría tridimensional. Además, hay otras limitaciones más sutiles que afectan incluso a las aplicaciones bidimensionales:

- El sistema de coordenadas enteras dependiente de la máquina que tiene SRGP es demasiado rígido para usarse en aplicaciones que requieren la mayor precisión, gama de valores y facilidad de uso del punto flotante.
- Como sucede con otros acervos gráficos de barrido bidimensionales, SRGP almacena una imagen en un lienzo de una manera libre de semántica, como matriz de valores de pixel inconexos en lugar de hacerlo como una colección de objetos gráficos (primitivas), y por ende no permite operaciones a nivel de objeto como *eliminar*, *mover* o *cambiar color*. Como SRGP no lleva un registro de las acciones que produjeron la imagen actual en la pantalla, tampoco puede refrescar

la pantalla si la imagen se daña debido a otro software, ni puede volver a discretizar las primitivas para producir una imagen que se presente en un dispositivo con distinta resolución.

### 2.4.1 Sistema de coordenadas de aplicación

En el capítulo 1 presentamos el concepto de que, para la mayoría de las aplicaciones, los dibujos son únicamente un medio para llegar a un fin, y que la función principal de la base de datos de la aplicación es apoyar procesos como el análisis, la simulación, la verificación y la manufactura. Por lo tanto, la base de datos debe almacenar información geométrica utilizando la gama de valores y la precisión que requieren estos procesos, independientemente del sistema de coordenadas y de la resolución del dispositivo de presentación. Por ejemplo, puede ser necesario que un programa CAD/CAM VLSI represente circuitos de 1 o 2 centímetros (cm) de longitud con una precisión de media micra, mientras que un programa de astronomía puede requerir una gama de valores de 1 a  $10^9$  años luz con una precisión de un millón de kilómetros. Para lograr la gama de valores y la flexibilidad máximas, muchas aplicaciones utilizan *coordenadas de mundo* de punto flotante para almacenar la geometría en su base de datos.

Este tipo de aplicación tendría que efectuar la correspondencia entre coordenadas de mundo y las del dispositivo; sin embargo, si consideramos la complejidad de esta correspondencia (la cual analizaremos en el capítulo 6), es conveniente usar un paquete gráfico que acepte primitivas especificadas en coordenadas de mundo y establezca la correspondencia con el dispositivo de presentación en forma independiente de la máquina. La reciente disponibilidad de circuitos de punto flotante de bajo costo, que ofrecen más o menos el mismo nivel de rendimiento que la aritmética entera, ha reducido considerablemente el impacto temporal relacionado con el uso del punto flotante, cuya flexibilidad vale la pena pagar en las aplicaciones que la requieren.

En el caso de gráficos bidimensionales, el software más común que ofrece coordenadas de punto flotante es PostScript, de Adobe, que se utiliza tanto como lenguaje estándar de descripción de páginas para dirigir impresoras que (en una extensión llamada Display PostScript) como paquete gráfico para sistemas de ventanas en algunas estaciones de trabajo. Para los gráficos tridimensionales de punto flotante se encuentran disponibles PHIGS y PHIGS +, y están apareciendo varias extensiones tridimensionales para PostScript.

### 2.4.2 Almacenamiento de primitivas para reespecificación

Considere lo que sucede cuando una aplicación que usa SRGP tiene que redibujar una imagen en un tamaño diferente o bien en el mismo tamaño en un dispositivo de presentación cuya resolución es distinta (p. ej., una impresora de alta resolución). Como SRGP no lleva un registro de las primitivas que ha dibujado, la aplicación debe reespecificar a SRGP todo el conjunto de primitivas después de escalar las coordenadas.

Si mejoráramos SRGP para que mantuviera un registro de todas las primitivas especificadas, la aplicación podría permitir que SRGP las regenerara a

partir de su propio almacenamiento. Además, SRGP podría permitir otra operación de uso común: el refrescamiento de la pantalla. En algunos sistemas gráficos, la imagen en pantalla de la aplicación puede ser dañada por mensajes de otros usuarios y aplicaciones; si no es posible refrescar el lienzo de la pantalla usando una copia redundante almacenada en un lienzo fuera de pantalla, la única forma de reparar el daño sería con la reespecificación de las primitivas.

La principal ventaja de que el paquete almacene las primitivas es el apoyo para las operaciones de edición que son la parte esencial de las aplicaciones de dibujo o construcción, una clase de programas muy distinta de las aplicaciones de pintura que ilustramos en los ejemplos de este capítulo. Un **programa de pintura** permite que el usuario pinte arbitrariamente usando un pincel de tamaño, forma, color y patrón variables. Los programas de pintura más complejos también permiten colocar formas predefinidas, como rectángulos, polígonos y círculos. Cualquier parte del lienzo puede editarse posteriormente a nivel de pixel; las porciones de un objeto pueden cubrirse con pintura, o regiones rectangulares arbitrarias se pueden copiar o mover a otra parte. El usuario no puede apuntar a una forma dibujada o a un trazo pintado previamente y luego eliminarlo o moverlo como un objeto coherente e indivisible. Esta limitación se debe a que el programa de pintura permite que un objeto colocado en el lienzo sea mutilado y fragmentado, perdiendo así su identidad como objeto coherente. Por ejemplo, ¿qué sentido tendría que un usuario apuntara a un fragmento de un objeto partido en trozos que se han colocado en forma independiente en diversas áreas de la pantalla? ¿Se estaría refiriendo el usuario al fragmento o a todo el objeto original? En esencia, la capacidad para afectar pixeles en forma individual hace imposible la correlación de selección y, por ende, la selección y edición de objetos.

Un **programa de dibujo**, por otra parte, permite que el usuario seleccione y edite cualquier objeto en cualquier instante. Estas aplicaciones, también llamadas **editores de distribución o ilustradores gráficos**, permiten que el usuario coloque formas estándar (también conocidas como *símbolos, plantillas u objetos*) y luego edite la distribución eliminando, moviendo, rotando, escalando y cambiando los atributos de estas formas. Los programas interactivos similares que permiten a los usuarios armar objetos tridimensionales a partir de otros más sencillos se denominan **editores geométricos o programas de construcción**.

El escalamiento, el refrescamiento de la pantalla y la edición a nivel de objetos requieren que la aplicación o el paquete gráfico lleve a cabo el almacenamiento y la reespecificación de primitivas; sin embargo, estas operaciones son mucho más complejas de lo que parecen a simple vista. Por ejemplo, se puede eliminar trivialmente una primitiva borrando la pantalla y volviendo a especificar todas las primitivas (por supuesto, con la excepción de la eliminada); sin embargo, un método más eficiente es borrar la imagen de la primitiva dibujando encima el fondo de pantalla de la aplicación y volviendo a especificar las primitivas que pudieran dañarse. Como estas operaciones son complejas y de uso frecuente, existen buenas razones para pasar su funcionalidad al propio paquete gráfico.

Un paquete gráfico geométrico a nivel de objetos, como PHIGS, permite que la aplicación defina objetos usando un sistema de coordenadas bidimensional o tridimensional de punto flotante. El paquete almacena los objetos en forma interna, permite que la aplicación los edite y actualiza la pantalla cuando sea necesario debido a una operación de edición. El paquete también lleva a cabo la correlación de selección, produciendo un identificador de objeto cuando se le especifiquen coordenadas de la pantalla. Como estos paquetes manipulan objetos, no pueden permitir manipulaciones a nivel de pixel (copyPixel y modo de escritura); éste es el precio de conservar la coherencia de objetos. Por lo tanto, ni un paquete gráfico de barrido sin almacenamiento de primitivas ni un paquete gráfico geométrico con almacenamiento de primitivas satisfacen todas las necesidades. En el capítulo 7 se analizan las ventajas y desventajas de la retención de primitivas en el paquete gráfico.

**Escalamiento de imágenes a través de la duplicación de pixeles.** Si ni la aplicación ni el paquete llevan un registro de las primitivas (lo usual en la mayoría de los paquetes de pintura), entonces no se puede efectuar el escalamiento reespecificando las primitivas con coordenadas escaladas para los puntos extremos. Lo único que se puede hacer es escalar el contenido del lienzo usando operaciones de lectura y escritura de pixeles. La manera más sencilla y rápida de agrandar por escalamiento una imagen de mapa de bits o pixeles es con la **duplicación de pixeles**, donde cada pixel se sustituye por un bloque de  $N$  por  $N$  pixeles, con lo cual se amplía la imagen por un factor de escalamiento de  $N$ .

Con la duplicación de pixeles, la imagen se hace más grande pero también más burda, ya que no se proporciona más información que la que contiene la representación original a nivel de pixeles. Así mismo, la duplicación de pixeles puede aumentar el tamaño de una imagen únicamente por un factor entero. Es necesario emplear otra técnica —muestreo de área y filtrado (analizado en el Cap. 3)— para ampliar o reducir correctamente la imagen. El filtrado funciona mejor con mapas de pixeles con profundidad mayor que 1.

El problema del escalamiento de imágenes surge con frecuencia, sobre todo cuando hay que imprimir una imagen creada con un programa de pintura. Consideremos el envío de un lienzo a una impresora que ofrece el doble de resolución que la pantalla. Cada pixel tiene ahora la mitad de su tamaño original, de manera que podemos mostrar la imagen original con el mismo número de pixeles a la mitad de su tamaño o usar la duplicación de pixeles para producir una imagen del tamaño original sin aprovechar la mejor resolución de la impresora. De ambas formas se pierde algo, ya sea tamaño o calidad, y el único método de escalamiento que no sacrifica la calidad es la reespecificación.

## RESUMEN

En este capítulo analizamos un sencillo pero poderoso paquete gráfico de barrido: SRGP. Permite que el programa de aplicación dibuje primitivas bidimensionales sujetas a varios atributos que afectan su apariencia. Los dibujos se pueden efectuar directamente en el lienzo de pantalla o en un lienzo fuera de pantalla de cualquier tamaño que se deseé. El dibujo se puede limitar a una región

rectangular del lienzo usando el atributo del rectángulo de recorte. Además de las formas bidimensionales estándar, SRGP también permite el copiado de regiones rectangulares intralienzos e interlienzos. El copiado y el dibujo pueden verse afectados por el modo de escritura, lo cual permite que el valor actual del pixel destino desempeñe una función en la determinación de su nuevo valor.

SRGP también introduce el concepto de los dispositivos lógicos de entrada, que son abstracciones de alto nivel de los dispositivos de entrada físicos. El dispositivo de teclado de SRGP abstrae un teclado físico, mientras que el dispositivo localizador abstrae dispositivos como el ratón, la tableta de datos y la palanca de mandos. Los dispositivos lógicos pueden operar en modo de muestreo (sondeo) o de eventos. En el modo de eventos, una acción del usuario activa la colocación de un informe de evento en la cola de eventos, que la aplicación puede examinar cuando le sea conveniente. En el modo de muestreo, la aplicación examina en forma continua la medida del dispositivo para detectar cambios importantes.

SRGP convierte por rastreo las primitivas a sus pixeles componentes y no almacena su geometría original, por lo que la única forma de edición permitida en SRGP es alterar los pixeles individuales, ya sea dibujando nuevas primitivas o usando la operación copyPixel con bloques de pixeles. La manipulación de objetos, por ejemplo, mover, eliminar o cambiar su tamaño, debe realizarla el programa de aplicación por medio de la reespecificación de la imagen actualizada.

Otros sistemas ofrecen un conjunto de características gráficas diferentes. Por ejemplo, el lenguaje PostScript ofrece primitivas y atributos de punto flotante, incluyendo formas curvas y recursos de recorte mucho más generales. PHIGS es un paquete de subrutinas que permite manipular objetos modelados jerárquicamente, definidos en un sistema tridimensional de coordenadas de mundo de punto flotante. Estos objetos se almacenan en una base de datos editable; el paquete regenera en forma automática la imagen a partir de esta representación almacenada después de una operación de edición.

SRGP es un paquete de subrutinas, y muchos programadores han descubierto que un lenguaje interpretado, como PostScript de Adobe, ofrece mayor poder y flexibilidad. También hay mucha divergencia en cuanto a cuáles deben ser los estándares: los paquetes de subrutinas (enteras o de punto flotante, con o sin retención de primitivas) o lenguajes de presentación como PostScript, que no retienen primitivas. Cada una tiene su propio dominio de aplicaciones, y creemos que así será por mucho tiempo.

En el capítulo siguiente veremos cómo lleva a cabo SRGP sus dibujos usando la discretización y los recortes. En los capítulos subsecuentes, después de una introducción al hardware, analizaremos las matemáticas de las transformaciones y la visualización tridimensional como preparación para aprender acerca de PHIGS.

**2.1** SRGP opera en un ambiente de ventanas pero no permite que la aplicación aproveche las ventanas múltiples: el lienzo de la pantalla corresponde a una sola ventana y no hay otros lienzos visibles. ¿Qué cambios haría al diseño de SRGP y a la interfaz aplicación-programador para permitir que una aplicación aproveche las ventajas de un sistema de ventanas?

2.2 Una aplicación en SRGP sólo puede ser por completo independiente de la máquina si únicamente utiliza los colores 0 y 1. Desarrolle una estrategia para mejorar SRGP de manera que simule los colores cuando sea necesario, permitiendo que la aplicación sea diseñada para aprovechar el color pero sin dejar de operar de manera útil en una pantalla de dos niveles. Analice los problemas y conflictos que crea esta estrategia.

2.3 Implante una secuencia de animación en la cual se muevan y cambien de tamaño varios objetos triviales. Genere primero cada cuadro borrando la pantalla y luego especificando los objetos en su nueva posición. Después pruebe la doble memoria: use un lienzo fuera de pantalla como memoria intermedia para dibujar cada cuadro antes de copiarlo al lienzo de la pantalla. Compare los resultados de los dos métodos. Considere también la utilización de la función `SRGP_copyPixel`. ¿En qué circunstancias limitadas es útil para la animación?

2.4 Implante un rastreo de cursor no destructivo sin usar el eco de cursor incorporado a SRGP. Use un patrón de mapa de bits o de pixeles para almacenar la imagen del cursor, con cero para representar la transparencia del patrón. Implante un cursor `xor` en una pantalla de dos niveles y un cursor de modo de reemplazo en una pantalla de dos niveles o en colores. Para probar el rastreo debe ejecutar un lazo de muestreo con el dispositivo localizador de SRGP y mover el cursor sobre un área que contiene información previamente escrita.

2.5 Considere la implantación de la siguiente característica en una aplicación de pintura. El usuario puede pintar un trazo `xor` que invierta los colores debajo del pincel. Al parecer, es fácil implantarlo asignando el modo de escritura y luego ejecutando el código del programa 2.5. ¿Qué complicaciones hay? Proponga soluciones.

2.6 Algunas aplicaciones de pintura ofrecen un modo de *pintura en aerosol*, donde el pincel, al pasar sobre un área, afecta una minoría aleatoria de los pixeles. Cada vez que el pincel pasa por un área se tocan pixeles diferentes, de manera que a cada pasada la pintura se vuelve más *densa*. Implante una interacción de pintura en aerosol para una pantalla de dos niveles. (*Precaución:* Los algoritmos más obvios producen manchones o no aumentan la intensidad. Usted debe crear un acervo de mapas de bits o patrones ralos; consulte el manual de referencia SRGP para obtener información acerca de la creación de patrones.)

2.7 Implante texto de fondo transparente para pantallas de dos niveles sin usar la primitiva de texto incorporada a SRGP. Use un lienzo fuera de pantalla a fin de almacenar la forma de mapa de bits para cada carácter; diseñe seis caracteres como máximo, ya que esto no es una lección de diseño de familias tipográficas. (*Sugerencia:* Quizás tenga que usar dos algoritmos para apoyar los colores 0 y 1.)

2.8 Un programa de dibujo puede actualizar la pantalla después de una operación de eliminación rellenando la forma del objeto eliminado con el patrón de

fondo de la pantalla de la aplicación. Por supuesto, esta técnica puede dañar otros objetos. ¿Por qué no es suficiente con reparar el daño volviendo a especificar todos los objetos cuyas extensiones rectangulares intersequen la extensión del objeto eliminado? Analice las soluciones al problema de optimizar la reparación de daños.

2.9 Implante una función que dibuje un botón con texto centrado dentro de un rectángulo opaco con borde delgado. Permita que en la llamada se especifiquen los colores para el texto, el fondo y el borde; la posición en la pantalla donde debe colocarse el centro del botón; un par de dimensiones máximas y mínimas para la anchura y la altura; y la familia tipográfica y la cadena de texto. Si la cadena no cabe en una línea del botón a su máxima longitud, divida la cadena en los lugares apropiados (p. ej., donde haya espacios) para que el texto quede en varias líneas dentro del botón.

2.10 Implante un dispositivo lógico valuador de entrada en pantalla que permita al usuario especificar una temperatura empleando el ratón para variar la longitud de una columna de mercurio simulada. Los atributos del dispositivo deben incluir el intervalo de medición, la medida inicial, la granularidad deseada para la medición (p. ej., precisión de 1° C) y la longitud y posición deseadas para la imagen del termómetro en pantalla. A fin de probar su dispositivo, use una interacción que simule una función *waitEvent* indefinida en la cual el único dispositivo activo sea el valuador.

2.11 Imagine adecuar una implantación de SRGP añadiendo al modelo de entrada un dispositivo valuador en pantalla (como el descrito en el ejercicio 2.10) y apoyándolo para los modos de muestreo y eventos. ¿Qué tipo de problemas podrían surgir si la implantación se instala en una estación de trabajo que sólo tenga un dispositivo localizador físico? Proponga soluciones.

2.12 Implante una primitiva de *rectángulo redondeado*, o sea un rectángulo cuyas esquinas estén redondeadas y que cada esquina sea un arco elíptico de 90 grados rectangulares. Permita que la aplicación controle los radios del arco elíptico y que existan versiones huecas y llenadas.

2.13 Implante el paquete de menú descendente cuyo diseño de alto nivel se presenta en fragmentos de código en las secciones 2.2.6, 2.3.1 y 2.3.3. Haga que el paquete asigne valores iniciales a la barra de menú y a los cuerpos de menú leyendo cadenas de un archivo de entrada. Permita que el programa desactive un menú haciendo desaparecer su encabezado o que lo active (indicando como parámetro su posición horizontal en la barra) para que aparezca el menú.

2.14 Mejore su paquete de menú del proyecto 2.13 implantando la desactivación de elementos seleccionados del menú. Los elementos desactivados deben

aparecer “desvanecidos” en el cuerpo del menú; como SRGP no permite dibujar texto usando un estilo de pincel, para lograr este efecto en una pantalla de dos niveles usted tiene que pintar sobre el texto sólido usando un modo de escritura.

2.15 Mejore su paquete de menú del ejercicio 2.13 realizando el elemento al cual apunta el localizador mientras un usuario escoge un elemento del cuerpo de menú.

2.16 Implante una aplicación de distribución que permita al usuario colocar objetos en una subregión cuadrada de la pantalla. Deben permitirse elipses, rectángulos y triángulos equiláteros. El usuario empleará el cursor y el botón del ratón para seleccionar un tipo de objeto o para iniciar una acción (redibujar la pantalla, restaurar una escena de un archivo o salir).

2.17 Agregue la edición de objetos a su aplicación de distribución del ejercicio 2.16. El usuario debe ser capaz de eliminar, mover o cambiar el tamaño o la escala de los objetos. Use un sencillo método de correlación de selección: rastree los objetos en la base de datos de la aplicación y seleccione el primero cuya extensión rectangular encierre la posición del localizador (demuestre que este sencillo método tiene un efecto secundario desagradable: ¡es posible que no pueda seleccionarse un objeto visible!). Asegúrese de que el usuario reciba retroalimentación realizando el objeto seleccionado.

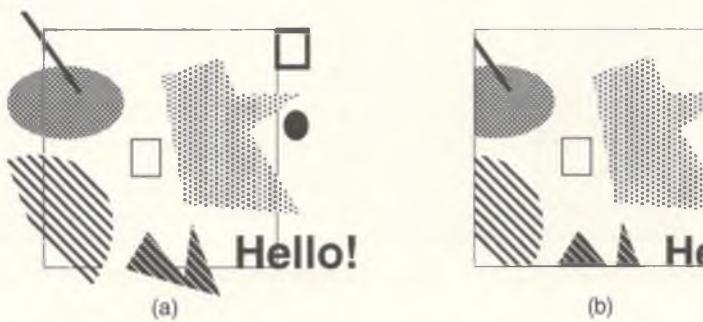
2.18 Añada media dimensión a su aplicación de distribución del ejercicio 2.16 implantando la prioridad de superposición. El usuario debe ser capaz de meter/sacar un objeto (forzar su prioridad como la más alta/baja). Mejore la correlación de selección y use la prioridad de superposición para resolver conflictos. ¿Cómo permite la funcionalidad de meter/sacar, así como el uso de prioridades, que el usuario supere la imprecisión del sencillo método de correlación de selección?

2.19 Optimice el algoritmo de actualización de pantalla de su aplicación de distribución del ejercicio 2.16 usando los resultados del ejercicio 2.8, de manera que se vuelva a especificar la cantidad mínima de objetos en respuesta a una operación de edición.

2.20 Mejore su aplicación de distribución del ejercicio 2.16 para que el teclado y el localizador se encuentren simultáneamente activos, de manera que existan abreviaturas en el teclado para las operaciones más comunes. Por ejemplo, al presionar la letra “e” se eliminaría el objeto seleccionado.

2.21 Diseñe e implante técnicas analíticas para la correlación de selección de los tres tipos de objetos permitidos en su aplicación de distribución del ejercicio 2.16. Sus nuevas técnicas deben ofrecer una precisión total; el usuario ya no tendrá que meter/sacar para elegir un objeto visible de baja prioridad.

Un paquete gráfico de barrido aproxima primitivas matemáticas (*ideales*), descritas en términos de vértices en un sistema de coordenadas cartesianas, como conjuntos de píxeles con la intensidad apropiada de gris o color. Estos píxeles se almacenan como un arreglo bidimensional (mapa) de bits o de píxeles en la memoria de la UCP o en una memoria gráfica. En el capítulo anterior estudiamos las características de SRGP, un paquete típico de gráficos de barrido, desde la perspectiva del programador. El propósito de este capítulo es ver SRGP desde la perspectiva del implantador de un paquete, es decir, en función de los algoritmos fundamentales para discretizar las primitivas a píxeles, sujeto a sus atributos, y para recortarlos con respecto a un rectángulo vertical. En la figura 3.1 se presentan ejemplos de primitivas discretizadas y recortadas.



**Figura 3.1** Recorte de primitivas de SRGP con respecto a una región rectangular: (a) primitivas y rectángulo de recorte; (b) resultados recortados.

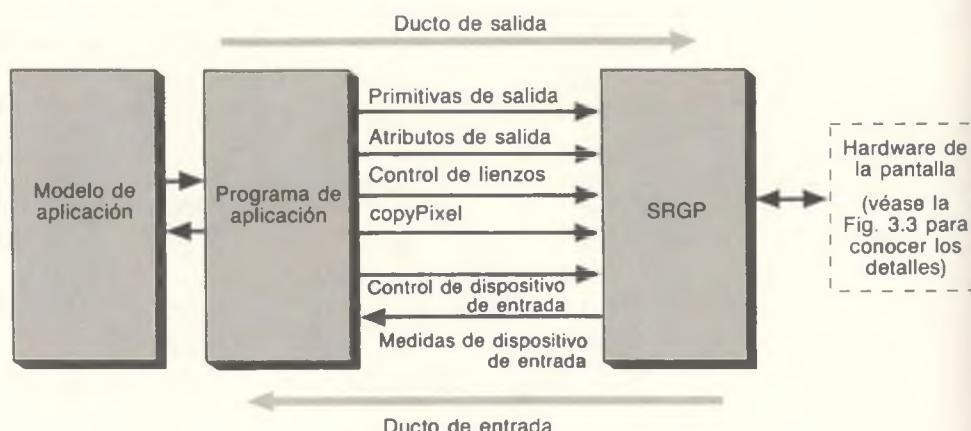
En los paquetes más elaborados y complejos se utilizan algoritmos más avanzados que los incluidos en SRGP. Los algoritmos que se presentan en este capítulo se analizan en función del sistema bidimensional entero de coordenadas cartesianas, pero la mayoría de los algoritmos de discretización se puede extender a punto flotante, mientras que los algoritmos de recorte se pueden extender a punto flotante y a tres dimensiones. En la sección final se introduce el concepto de eliminación del artefacto de discretización (*antialiasing*), es decir, la minimización del serrado usando la capacidad del sistema para variar la intensidad de un pixel.

### 3.1 Esquema general

#### 3.1.1 Implicaciones de la arquitectura del sistema de pantalla

El modelo conceptual fundamental del capítulo 1 presenta un paquete gráfico como un sistema intermedio entre el programa de aplicación (y su modelo/estructura de datos de aplicación) y el hardware de la pantalla. El paquete proporciona al programa de aplicación una interfaz independiente del dispositivo con el hardware, como se ilustra en la figura 3.2, donde las funciones de SRGP se dividen en aquellas que forman un ducto de salida y las que forman un ducto de entrada.

En el **ducto de salida**, el programa de aplicación toma descripciones de los objetos en función de las primitivas y los atributos almacenados o derivados del



**Figura 3.2** SRGP como intermediario entre el programa de aplicación y el sistema gráfico, ofreciendo ductos de entrada y salida.

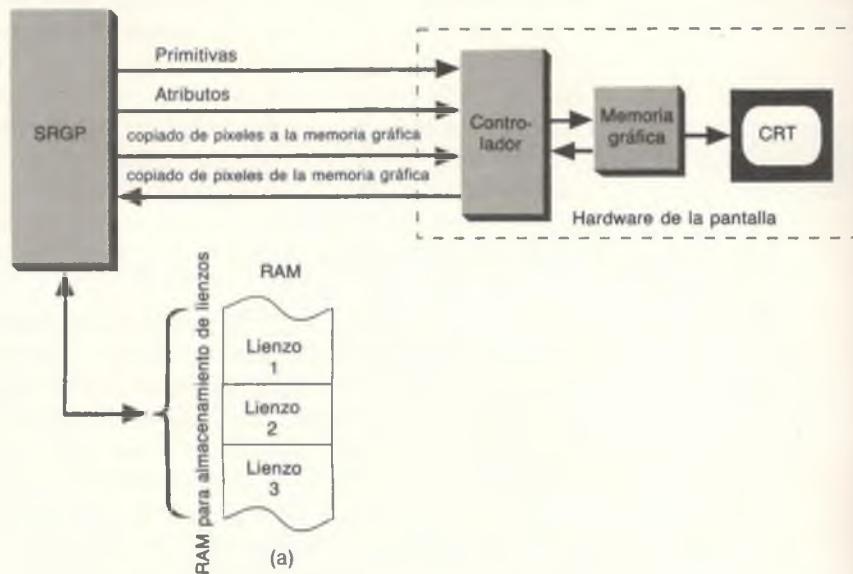
modelo o estructura de datos de la aplicación, y las especifica al paquete gráfico, el cual a su vez las recorta y discretiza a los pixeles que se envían a la pantalla. Las funciones de generación de primitivas del paquete especifican *qué* hay que generar, las funciones de atributos especifican *cómo* deben generarse las primitivas, la función SRGP\_copyPixel especifica *cómo* deben modificarse las imágenes y las funciones de control de lienzo indican *dónde* se deben generar las imágenes. En el **ducto de entrada**, una interacción del usuario en el extremo de la pantalla se convierte a valores de medición devueltos al programa de aplicación por las funciones de entrada por muestreo o dirigidas por eventos; el ducto de entrada suele emplear estos valores para modificar el modelo o la imagen en la pantalla. Las funciones relacionadas con la entrada incluyen las que sirven para asignar valores iniciales y controlar los dispositivos de entrada y las que obtienen sus medidas durante la interacción. En este libro no se abarca la administración de lienzos de SRGP ni su manejo de entrada, ya que estos temas tienen poco que ver con los gráficos de barrido y son principalmente aspectos de estructura de datos y de software de sistema de bajo nivel, respectivamente.

Una implantación de SRGP debe comunicarse con una variedad muy extensa de pantallas. Algunas pantallas están conectadas como periféricos, con memoria gráfica y controladores propios. Estos controladores son procesadores especializados que interpretan y ejecutan mandatos de dibujo que generan pixeles en la memoria gráfica. Otros sistemas, más sencillos, se refrescan directamente desde la memoria utilizada por la UCP. Los subconjuntos de sólo salida de este paquete pueden servir para dirigir dispositivos de impresión. Estos diversos tipos de arquitectura hardware se analizarán con mayor detalle en el capítulo 4. En cualquier arquitectura de sistema de pantalla, la UCP debe ser capaz de leer y escribir pixeles individuales en la memoria gráfica. También es conveniente que pueda mover bloques rectangulares de pixeles de y hacia la memoria gráfica para implantar una operación de tipo copyPixel (bitBlt). Este recurso no se usa para generar primitivas directamente, sino para hacer visibles porciones de los mapas de bits o de pixeles fuera de pantalla, así como para guardar y restaurar pedazos de la pantalla para la administración de ventanas, el manejo de menús, el desplazamiento, etcétera.

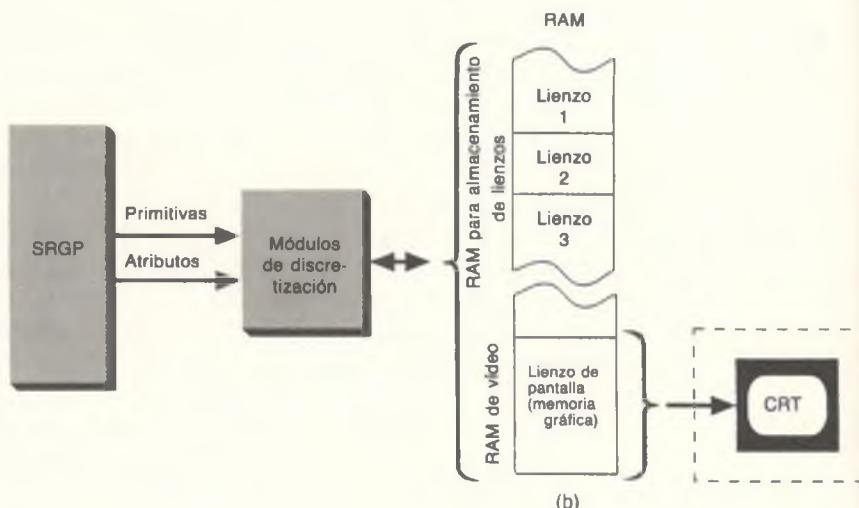
Aunque todas las implantaciones de sistemas que refrescan a partir de la memoria de la UCP son en esencia idénticas, ya que todo el trabajo se lleva a cabo con software, las implantaciones de los sistemas de controlador de pantalla y de impresión pueden variar en forma considerable, dependiendo de lo que puedan hacer los dispositivos de hardware respectivos y lo que reste por hacer al software. Por supuesto, no obstante la arquitectura hay que emplear discretizaciones por software para generar las primitivas y los atributos que no estén apoyados directamente por hardware. Veamos brevemente la gama de arquitecturas e implantaciones.

**Pantallas con memoria gráfica y controlador.** SRGP tiene que efectuar la menor cantidad de trabajo si alimenta a un controlador de pantalla que lleve a

cabo su propia discretización y maneje en forma directa todas las primitivas y los atributos de SRGP. En este caso, lo único que tiene que hacer SRGP es convertir su representación interna de las primitivas, los atributos y los modos de escritura a los formatos aceptados por el periférico de presentación que dibuja la primitiva (Fig. 3.3a).



(a)



(b)

**Figura 3.3** SRGP alimentando a dos tipos de sistemas de pantalla: (a) pantalla con controlador y memoria gráfica; (b) sin controlador de pantalla, con memoria gráfica en memoria compartida.

La arquitectura de controlador de pantalla es más poderosa cuando la correspondencia a memoria permite que la UCP tenga acceso directo a la memoria gráfica y que el controlador de pantalla tenga acceso a la memoria de la UCP. De esta manera, la UCP puede leer y escribir pixeles individuales y copiar bloques de pixeles con instrucciones normales de la UCP, mientras que el controlador de pantalla puede discretizar los lienzos fuera de pantalla y usar su instrucción copyPixel para mover pixeles entre dos memorias o dentro de una memoria gráfica. Cuando la UCP y el controlador de pantalla pueden operar en forma asincrónica, es necesario establecer sincronización para evitar conflictos de memoria. Con frecuencia, el controlador de pantalla es controlado como co-procesador por la UCP. Si el controlador de pantalla del periférico únicamente puede discretizar su propia memoria gráfica y no puede escribir pixeles en la memoria de la UCP, se necesita un mecanismo para crear primitivas en un lienzo fuera de pantalla. El paquete usa entonces el controlador para la discretización al lienzo de pantalla, pero tiene que usar su propio software de discretización para los lienzos fuera de pantalla. Por supuesto, el paquete puede usar copyPixel con imágenes discretizadas por hardware de la memoria gráfica a los lienzos fuera de pantalla.

**Pantallas con sólo memoria gráfica.** En el caso de pantallas sin controlador, SRGP lleva a cabo su propia discretización a los lienzos fuera de pantalla y a la memoria gráfica. En la figura 3.3 (b) se muestra una organización típica de este tipo de implantación SRGP que alimenta una memoria gráfica en memoria compartida. Observe que únicamente se ilustran las partes de la memoria que constituyen la memoria gráfica y que almacenan los lienzos administrados por SRGP; el resto de la memoria está ocupado por el software y los datos usuales, incluyendo por supuesto SRGP.

**Dispositivos de impresión.** Como veremos en el capítulo 4, los dispositivos de impresión varían en cuanto a sus capacidades, aproximadamente en la misma magnitud que los sistemas de pantalla. Los dispositivos más sencillos sólo aceptan una línea de rastreo a la vez y dependen del software para proporcionar la línea de rastreo en el instante preciso en el que hay que pasarla a película o papel. En el caso de este hardware sencillo, SRGP debe generar un mapa de bits o de pixeles, discretizarlo una línea a la vez y enviar ésta al dispositivo de salida. Los dispositivos un poco más inteligentes pueden aceptar todo el marco (página) al mismo tiempo. Los equipos más poderosos incluyen hardware de discretización, conocido como procesador de imagen de barrido (RIP, *raster image processor*). En el extremo más alto de la escala están las impresoras PostScript, que incluyen *procesadores* internos para leer programas PostScript que describen las páginas en forma independiente del dispositivo; interpretan los programas para producir las primitivas y los atributos que después se discretizan. Los algoritmos fundamentales de recorte y discretización son, en esencia, independientes de la tecnología de salida del dispositivo de trama; por lo tanto, en este capítulo no tenemos que analizar con mayor detalle los dispositivos de hardware.

### 3.1.2 El ducto de salida en software

Ahora veremos el ducto de salida que alimenta las pantallas simples con memoria gráfica únicamente para tratar los problemas de recorte y discretización por software. Los diversos algoritmos que presentamos se analizan a nivel general, independiente de la máquina, de manera que son aplicables tanto a implantaciones en software como en hardware (o microcódigo).

Cuando SRGP detecta una primitiva de salida, el paquete la *discretiza*: los pixeles se escriben en el lienzo actual de acuerdo con sus atributos y el modo de escritura vigente. La primitiva también se **recorta** de acuerdo con el rectángulo de recorte; es decir, no se presentan los pixeles pertenecientes a la primitiva que están fuera de la región de recorte. Existen varias formas de efectuar los recortes. La técnica obvia es recortar una primitiva antes de efectuar la discretización, calculando sus intersecciones analíticas con los límites del rectángulo de recorte; estos puntos de intersección se usan entonces para definir los nuevos vértices de la versión recortada de la primitiva. La ventaja de recortar antes de la discretización es, por supuesto, que el discretizador sólo tiene que manejar la versión recortada de la primitiva y no la original (que quizás sea mucho mayor). Esta técnica se usa con frecuencia para recortar líneas, rectángulos y polígonos, cuyos algoritmos de recorte son bastante sencillos y eficientes.

La técnica de recorte más sencilla y burda, llamada **tijereteo**, consiste en discretizar toda la primitiva pero escribir sólo los pixeles visibles en la región de rectángulo de recorte del lienzo. En principio, este procedimiento revisa las coordenadas de cada pixel y las compara con los límites ( $x$ ,  $y$ ) del rectángulo antes de escribir el pixel. En la práctica existen atajos que evitan tener que revisar pixeles adyacentes en una línea de rastreo, como veremos más adelante. Este tipo de recorte se lleva a cabo sobre la marcha; si la revisión de fronteras se puede efectuar con rapidez (p. ej., con un pequeño ciclo interno que se ejecuta en microcódigo o en una memoria caché de instrucciones), este método puede ser más veloz que recortar primero la primitiva y luego discretizarla a las porciones recortadas. También tiene la generalidad para manejar regiones de recorte arbitrarias.

Una tercera técnica es generar toda la colección de primitivas en un lienzo temporal y luego usar la función `copyPixel` para copiar únicamente el contenido del rectángulo de recorte al lienzo destino. Este método desperdicia tiempo y espacio, pero es fácil de implantar y se usa con frecuencia para texto.

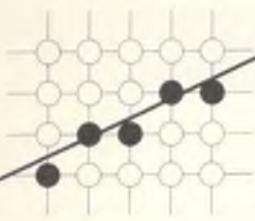
Las pantallas de barrido invocan a los algoritmos de recorte y de discretización cada vez que se crea o modifica una imagen. Por consiguiente, estos algoritmos no sólo deben crear imágenes visualmente satisfactorias, sino además ejecutarse con la mayor rapidez posible. Como veremos con detalle en secciones subsecuentes, los algoritmos de discretización utilizan *métodos incrementales* para minimizar el número de cálculos (sobre todo multiplicaciones y divisiones) que se realizan en cada iteración; así mismo, estos cálculos utilizan aritmética entera y no de punto flotante. La velocidad se puede aumentar aún más usando varios procesadores paralelos para discretizar de manera simultánea primitivas de salida completas o porciones de ellas.

## 3.2 Discretización de líneas

Un algoritmo de discretización de líneas calcula las coordenadas de los pixeles que se encuentran sobre o cerca de una línea ideal, infinitamente delgada, impuesta sobre una trama de barrido bidimensional. En principio, nos gustaría que la secuencia de pixeles estuviera lo más cerca posible de la línea ideal y que fuera lo más recta posible. Considere una aproximación de un pixel de grosor a una línea ideal. ¿Qué propiedades debe tener? En el caso de líneas con pendientes entre  $-1$  y  $1$  inclusive, debe estar iluminado exactamente un pixel en cada columna; en el caso de líneas con pendientes fuera de este intervalo, debe estar iluminado precisamente un pixel en cada fila. Todas las líneas se deben dibujar con intensidad constante, independientemente de su longitud y su orientación, y con la mayor rapidez posible. También debe tenerse en cuenta la posibilidad de dibujar líneas con grosor mayor que un pixel, centradas sobre la línea ideal, que estén afectadas por los atributos de estilo de línea y de estilo de pincel, y que crean otros efectos necesarios en las ilustraciones de alta calidad. Por ejemplo, la forma de las regiones de los puntos extremos debe estar bajo el control del programador para permitir esquinas biseladas, redondeadas y en escuadra. También nos gustaría minimizar el serrado ocasionado por la aproximación discreta de la línea ideal, usando técnicas de eliminación de artefacto de discretización que aprovechen la capacidad de establecer la intensidad de los pixeles individuales en pantallas de  $n$  bits por pixel.

Por el momento sólo consideraremos líneas de un pixel de grosor que tengan exactamente un pixel de dos niveles en cada columna (o en cada fila para línea con pendientes  $> \pm 1$ ). Más adelante en este capítulo veremos las primitivas gruesas y analizaremos los estilos.

Para visualizar la geometría, recordemos que SRGP representa un pixel como un punto circular centrado en la localidad  $(x, y)$  de la trama entera. Esta representación es una aproximación conveniente de la sección transversal más o menos circular del haz de electrones del CRT, pero el espaciado exacto entre los puntos del haz en una pantalla real puede variar notablemente de un sistema a otro. En algunos sistemas se sobreponen los puntos adyacentes; en otros puede de haber espacio entre los pixeles verticales adyacentes; en la mayoría, el espaciado es menor en la dirección horizontal que en la vertical. Otra variante de la representación del sistema de coordenadas se presenta en sistemas como Macintosh, que tratan los pixeles como si estuvieran centrados entre las líneas adyacentes de la trama en lugar de hallarse sobre las líneas. En este esquema, los rectángulos se definen como todos los pixeles interiores del rectángulo matemático definido por los vértices. Esta definición permite crear lienzos con anchura cero (nulos): el rectángulo de  $(x, y)$  a  $(x, y)$  no contiene pixeles, a diferencia del lienzo de SRGP, que tiene un pixel en ese punto. Por ahora seguiremos representando los pixeles como círculos disjuntos centrados en una malla uniforme, aunque haremos algunas variantes menores al analizar la eliminación del artefacto de discretización.



**Figura 3.4**  
Línea discretizada que muestra los píxeles intensificados como círculos negros.

En la figura 3.4 se presenta una vista muy magnificada de una línea de un pixel de grosor y de la línea ideal que aproxima. Los píxeles intensificados se muestran como círculos llenados, mientras que los píxeles no intensificados aparecen como círculos huecos. En una pantalla real, el diámetro de un pixel aproximadamente circular es mayor que el espaciado entre píxeles, de manera que nuestra representación simbólica exagera la discreción de los píxeles.

Dado que las primitivas de SRGP se definen en una malla entera, los puntos extremos de una línea tienen coordenadas enteras. De hecho, si primero recortamos la línea de acuerdo con el rectángulo de recorte, una línea que interseca una arista de recorte puede tener un punto final con valores no enteros de las coordenadas. Esta situación también se presenta al usar un paquete gráfico de barrido de punto flotante (analizaremos estas intersecciones no enteras en la sección 3.2.3). Suponga que nuestra línea tiene pendiente  $|m| \leq 1$ ; las líneas con otras pendientes se pueden manejar con cambios adecuados en el desarrollo siguiente. Además, las líneas más comunes —las horizontales, verticales o con pendiente de  $\pm 1$ — se pueden manejar como casos triviales especiales, ya que pasan únicamente por el centro de los píxeles (véase el Ejer. 3.1).

### 3.2.1 Algoritmo incremental básico

La estrategia más sencilla para discretizar líneas es calcular la pendiente  $m$  como  $\Delta y / \Delta x$  e incrementar  $x$  en 1 a partir del punto del extremo izquierdo para calcular  $y_i = mx_i + B$  de cada  $x_i$  e intensificar el pixel en  $(x_i, \text{round}(y_i))$ , donde  $\text{round}(y_i) = \text{floor}(0.5 + y_i)$ .<sup>1</sup> Con este cálculo se obtiene el pixel más cercano, o sea, aquel cuya distancia a la línea verdadera sea menor. Sin embargo, esta estrategia burda no es muy eficiente, ya que cada iteración requiere una multiplicación y una suma de punto flotante (o de fracción binaria), además de invocar a *floor*. Podemos eliminar la multiplicación al observar que

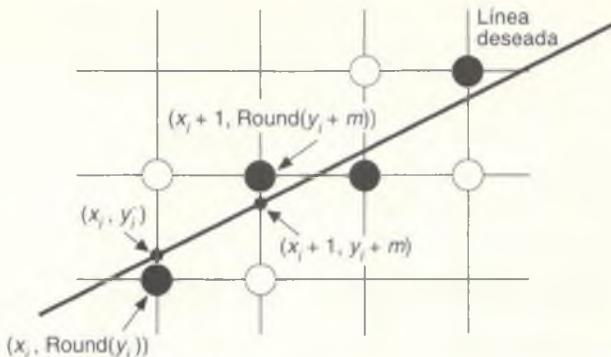
$$y_{i+1} = mx_{i+1} + B = m(x_i + \Delta x) + B = y_i + m\Delta x$$

y que si  $\Delta x = 1$ , entonces  $y_{i+1} = y_i + m$ .

De esta manera, un cambio unitario en  $x$  cambia  $y$  por  $m$ , que es la pendiente de la línea. Para todos los puntos  $(x_i, y_i)$  en la línea (*no* los puntos en la versión de trama de la línea), sabemos que si  $x_{i+1} = x_i + 1$ , entonces  $y_{i+1} = y_i + m$ ; es decir, los valores de  $x$  y  $y$  se definen en función de sus valores anteriores (véase la Fig. 3.5). Esto es lo que define un algoritmo incremental: en cada paso se realizan cálculos incrementales basados en el paso anterior.

Comenzamos el cálculo incremental en  $(x_0, y_0)$ , las coordenadas enteras de un punto extremo. Observe que esta técnica incremental evita la necesidad de tratar con la intersección del eje  $y$ , es decir,  $B$ . Si  $|m| > 1$ , un incremento en  $x$  crea un incremento en  $y$  mayor que 1. Por lo tanto, tenemos que invertir los papeles de  $x$  y  $y$  asignando un incremento unidad a  $y$  y aumentando  $x$  en  $\Delta x = \Delta y / m = 1/m$ . La función *línea* del programa 3.1 implanta la técnica incremental. El

<sup>1</sup> *Round* y *floor* son los nombres usuales de las rutinas que redondean y truncan, respectivamente, un número real, para obtener su parte entera. (N. del T.)



**Figura 3.5** Cálculo incremental de  $(x_i, y_i)$ .

punto de partida debe ser el punto extremo de la izquierda. Así mismo, está limitado al caso  $-1 \leq m \leq 1$ , pero las otras pendientes pueden acomodarse por simetría. Se omite la revisión de los casos especiales de líneas horizontales, verticales o diagonales.

La función *escribir\_pixel*, que utiliza *línea*, es una función de bajo nivel proporcionada por el software de la pantalla que coloca un valor en un lienzo para un pixel cuyas coordenadas se especifican como los dos primeros argumentos.<sup>2</sup> Supondremos aquí que únicamente discretizamos en modo de reemplazo; para los otros modos de escritura de SRGP tenemos que usar una función de bajo nivel *leer\_pixel* para leer el pixel en la localidad destino, combinar lógicamente este valor con el del pixel fuente y escribir el resultado en el destino con *escribir\_pixel*.

Este algoritmo se conoce como algoritmo **analizador diferencial digital (DDA, digital differential analyzer)**. El DDA es un dispositivo mecánico que resuelve ecuaciones diferenciales usando métodos numéricos: rastrea valores  $(x, y)$  sucesivos incrementando simultáneamente  $x$  y  $y$  en pequeñas cantidades proporcionales a la primera derivada de  $x$  y de  $y$ . En nuestro caso, el incremento de  $x$  es 1 y el incremento de  $y$  es  $dy/dx = m$ . Como las variables reales tienen precisión limitada, la suma repetida de una  $m$  inexacta produce una acumulación de errores y por último una desviación con respecto a un valor *round(y<sub>i</sub>)* verdadero; esta situación no ocasionará problemas en la mayoría de las líneas (cortas).

**Programa 3.1**

*El algoritmo incremental de discretización de líneas.*

```
void Línea (int x0, int y0, int x1, int y1, int valor)
{
    /* Supone -1 ≤ m ≤ 1, x0 < x1 */
    int x;                      /* x varia de x0 a x1 en incrementos unitarios. */
    float dy, dx, y, m;
```

<sup>2</sup> Si no dispone de una función de bajo nivel de este tipo, puede usar SRGP\_pointCoord; consulte el manual de referencia.

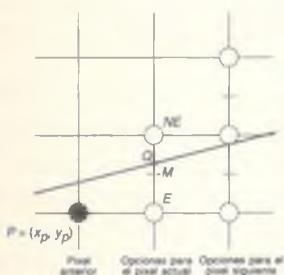
```

dy = y1 - y0;
dx = x1 - x0;
m = dy / dx;
y = y0;
for (x = x0; x <= x1; x++) {
    Escribirpixel (x, (int) floor(y + 0.5), valor); /* Asignar el pixel igual a valor */
    y += m; /* Incrementar y con el valor de la pendiente m */
}
}

```

### 3.2.2 Algoritmo de línea de punto medio

Las desventajas de la función *línea* son que el redondeo de *y* a un entero requiere tiempo y que las variables *y* y *m* deben ser reales o binarias fraccionarias, ya que la pendiente es una fracción. Bresenham desarrolló un algoritmo clásico [BRES65] que resulta atractivo porque sólo emplea aritmética entera con lo cual se evita el empleo de la función *round*, y permite efectuar incrementalmente el cálculo de  $(x_{i+1}, y_{i+1})$ , es decir, usando el cálculo efectuado para  $(x_i, y_i)$ . Se puede aplicar una versión de punto flotante de este algoritmo a líneas con coordenadas de puntos extremos con valores arbitrarios de punto flotante. Así mismo, la técnica incremental de Bresenham se puede aplicar también al cálculo entero de círculos, aunque no es fácil de generalizar a cónicas arbitrarias. Por ello usaremos una formulación un poco distinta, la *técnica de punto medio*, publicada por primera vez por Pitteway [PITT67] y adaptada por Van Aken [VANA84] y otros investigadores. En el caso de líneas y círculos enteros, la formulación de punto medio, como demuestra Van Aken [VANA85], se reduce a la formulación de Bresenham y por ende genera los mismos pixeles. Bresenham demostró que sus algoritmos de líneas y círculos enteros ofrecen las aproximaciones de mejor ajuste a las líneas y los círculos verdaderos, minimizando el error (distancia) a la primitiva verdadera [BRES77]. Kapp analiza los efectos de los diversos criterios de error en [KAPP85].



**Figura 3.6**  
Malla de pixeles para el algoritmo de línea de punto medio, en la cual se muestra el punto medio *M* y los pixeles *E* y *NE* de los cuales se puede escoger.

Suponemos que la pendiente de la línea está entre 0 y 1. Las demás pendientes se pueden manejar con una reflexión adecuada respecto a los ejes principales. Al punto extremo inferior izquierdo lo llamaremos  $(x_0, y_0)$  y al superior derecho,  $(x_1, y_1)$ .

Considere la línea que se presenta en la figura 3.6, donde el pixel previamente seleccionado aparece como un círculo negro y los dos pixeles de los cuales podemos escoger en la etapa siguiente se presentan como círculos huecos. Suponga que acabamos de seleccionar el pixel *P* en  $(x_p, y_p)$  y ahora tenemos que elegir entre el pixel que está un incremento a la derecha (llamado pixel este, *E*) y el pixel que se halla un incremento hacia la derecha y un incremento hacia arriba (llamado pixel noreste, *NE*). Sea *Q* el punto de intersección de la línea que se discretiza y la línea de malla  $x = x_p + 1$ . En la formulación de Bresenham se calcula la diferencia entre las distancias verticales de *E* y *NE* a *Q*, y se usa el signo de la diferencia para seleccionar como mejor aproximación de la línea el pixel cuya distancia de *Q* sea la menor. En la formulación de punto medio se observa a qué lado de la línea se encuentra el punto medio *M*. Es fácil

ver que si el punto medio está por encima de la línea, el pixel  $E$  es el más cercano a la línea; si el punto medio está debajo, el pixel  $NE$  es el más cercano. La línea puede pasar entre  $E$  y  $NE$  o ambos píxeles pueden estar del mismo lado; en cualquier caso, la prueba de punto medio elige el más cercano. Además, el error (la distancia vertical entre el pixel elegido y la línea real) siempre es menor o igual que  $\frac{1}{2}$ .

El algoritmo escoge  $NE$  como el siguiente pixel para la línea presentada en la figura 3.6. Ahora lo que necesitamos es una forma de calcular en qué lado de la linea se encuentra el punto medio. Representemos la línea por medio de una función implícita<sup>3</sup> con coeficientes  $a$ ,  $b$  y  $c$ :  $F(x, y) = ax + by + c = 0$  (el coeficiente  $b$  de  $y$  no está relacionado con la intersección  $B$  del eje  $y$  en la forma de intersección de pendiente). Si  $dy = y_1 - y_0$  y  $dx = x_1 - x_0$ , la forma de intersección de pendiente se puede escribir como

$$y = \frac{dy}{dx}x + B;$$

por lo tanto,

$$F(x, y) = dy \cdot x - dx \cdot y + B \cdot dx = 0.$$

Aquí,  $a = dy$ ,  $b = -dx$  y  $c = B \cdot dx$  en la forma implícita.<sup>4</sup>

Es sencillo verificar que  $F(x, y)$  es cero en la línea, positiva para los puntos debajo de la línea y negativa para los puntos encima de la línea. Para aplicar el criterio del punto medio, sólo tenemos que calcular  $F(M) = F(x_p + 1, y_p + \frac{1}{2})$  y evaluar su signo. Como nuestra decisión se basa en el valor de la función en  $(x_p + 1, y_p + \frac{1}{2})$ , definimos una variable de decisión  $d = F(x_p + 1, y_p + \frac{1}{2})$ . Por definición,  $d = a(x_p + 1) + b(y_p + \frac{1}{2}) + c$ . Si  $d > 0$ , elegimos el pixel  $NE$ ; si  $d < 0$ , escogemos  $E$ ; y si  $d = 0$ , podemos elegir cualquiera y seleccionamos  $E$ .

Después nos preguntamos qué sucede con la ubicación de  $M$  y por ende con el valor de  $d$  para la siguiente línea de la trama; por supuesto, los dos valores dependen de la elección de  $E$  o de  $NE$ . Si elegimos  $E$ ,  $M$  se incrementa una vez en la dirección  $x$ . Por lo tanto,

$$d_{\text{nuevo}} = F(x_p + 2, y_p + \frac{1}{2}) = a(x_p + 2) + b(y_p + \frac{1}{2}) + c,$$

pero

$$d_{\text{viejo}} = a(x_p + 1) + b(y_p + \frac{1}{2}) + c.$$

Al restar  $d_{\text{viejo}}$  de  $d_{\text{nuevo}}$  para obtener la diferencia incremental, escribimos  $d_{\text{nuevo}} = d_{\text{viejo}} + a$ .

<sup>3</sup> Esta forma funcional se extiende sin dificultades a la formulación implícita de círculos.

<sup>4</sup> Para el funcionamiento apropiado del algoritmo de punto medio es importante elegir  $a$  de manera que sea positiva; este criterio se cumple si  $dy$  es positiva, ya que  $y_1 > y_0$ .

El incremento que se usa después de elegir  $E$  se denomina  $\Delta_E$ ;  $\Delta_E = a = dy$ . En otras palabras, con sólo sumar  $\Delta_E$  podemos obtener en forma incremental el valor de la variable de decisión en el siguiente paso a partir del valor en el paso actual, sin tener que calcular  $F(M)$  directamente.

Si elegimos  $NE$ ,  $M$  se incrementa una unidad tanto en la dirección  $x$  como en la  $y$ . Entonces,

$$d_{\text{nuevo}} = F(x_p + 2, y_p + \frac{3}{2}) = a(x_p + 2) + b(y_p + \frac{3}{2}) + c.$$

Al restar  $d_{\text{viejo}}$  a  $d_{\text{nuevo}}$  para obtener la diferencia incremental, escribimos

$$d_{\text{nuevo}} = d_{\text{viejo}} + a + b.$$

El incremento que sumamos a  $d$  después de elegir  $NE$  se llama  $\Delta_{NE}$ ;  $\Delta_{NE} = a + b = dy - dx$ .

Resumamos la técnica incremental del punto medio. En cada paso, el algoritmo escoge entre dos pixeles con base en el signo del cálculo de la variable de decisión en la iteración anterior; después la variable de decisión se actualiza sumando  $\Delta_E$  o  $\Delta_{NE}$  al valor anterior, dependiendo de la elección del pixel.

Como el primer pixel es el primer punto extremo  $(x_0, y_0)$ , podemos calcular en forma directa el valor inicial de  $d$  eligiendo entre  $E$  y  $NE$ . El primer punto medio está en  $(x_0 + 1, y_0 + \frac{1}{2})$  y

$$\begin{aligned} F(x_0 + 1, y_0 + \frac{1}{2}) &= a(x_0 + 1) + b(y_0 + \frac{1}{2}) + c \\ &= ax_0 + by_0 + c + a + b/2 \\ &= F(x_0, y_0) + a + b/2 \end{aligned}$$

Sin embargo,  $(x_0, y_0)$  es un punto en la línea y  $F(x_0, y_0)$  es por lo tanto 0; entonces,  $d_{\text{inicio}}$  es simplemente  $a + b/2 = dy - dx/2$ . Con base en  $d_{\text{inicio}}$  elegimos el segundo pixel, etcétera. Para eliminar la fracción en  $d_{\text{inicio}}$ , redefinimos nuestra  $F$  original multiplicándola por 2;  $F(x, y) = 2(ax + by + c)$ . Así se multiplican por 2 cada constante y la variable de decisión (así como los incrementos  $\Delta_E$  y  $\Delta_{NE}$ ), pero no se afecta el signo de la variable de decisión, que es lo que nos interesa para la prueba del punto medio.

La aritmética necesaria para evaluar  $d_{\text{nuevo}}$  para cualquier iteración es una suma entera sencilla. No se requieren multiplicaciones que consuman mucho tiempo. Además, el ciclo interior es bastante sencillo, como se ve en el algoritmo de punto medio del programa 3.2. El primer enunciado en el ciclo, la evaluación de  $d$ , determina la elección del pixel, pero en realidad incrementamos  $x$  y  $y$  a la posición de ese pixel después de actualizar la variable de decisión (por cuestiones de compatibilidad con los algoritmos para círculos). Observe que esta versión del algoritmo sirve únicamente para líneas con pendiente entre 0 y 1; la generalización del algoritmo queda como tarea para el lector en el ejercicio 3.2. En [SPRO82], Sproull ofrece una elegante derivación de la formulación de Bresenham del algoritmo como una serie de transformaciones de programa a

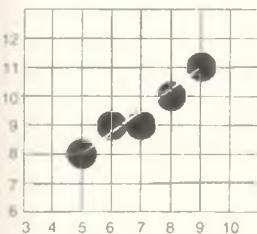
partir del burdo algoritmo original. Aún no se ha presentado ningún equivalente de la derivación para los círculos, pero sí se puede generalizar la técnica de punto medio, como veremos más adelante.

### Programa 3.2

*El algoritmo de punto medio para la discretización de líneas.*

```
void linea_punto_medio (int x0, int y0, int x1, int y1, int valor)
{
    int dx, dy, incr_E, incr_NE, d, x, y;

    dx = x1 - x0;
    dy = y1 - y0;
    d = dy * 2 - dx;
    incr_e = dy * 2;
    incr_NE = (dy - dx) * 2;
    x = x0;
    y = y0;
    escribir_pixel (x, y, valor)
    while (x < x1) {
        if (d <= 0) {
            d += incr_E;
            x++;
        } else {
            d += incr_NE;
            x++;
            y++;
        }
        escribir_pixel (x, y, valor); /* El pixel seleccionado, más cercano a la línea */
    }
}
```



**Figura 3.7**  
La línea de punto medio entre el punto (5, 8) y el punto (9, 11).

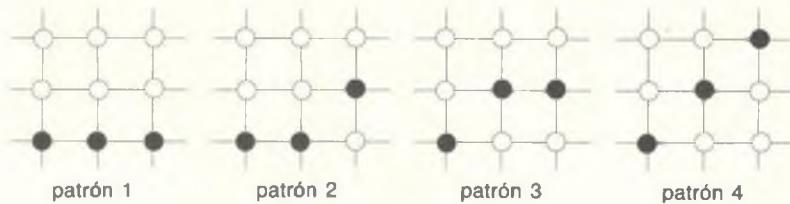
Para una línea que parte del punto (5, 8) al punto (9, 11), los valores sucesivos de  $d$  son 2, 0, 6 y 4, con lo cual se obtiene la selección de  $NE$ ,  $E$ ,  $NE$  y luego  $NE$ , respectivamente, como se ilustra en la figura 3.7. La línea tiene una fuerte apariencia de escalera por la escala muy amplificada del dibujo y el espacio artificialmente grande entre los píxeles que se usa para que quede clara la geometría del algoritmo. Por la misma razón, los dibujos en las secciones siguientes también hacen que las primitivas aparezcan más irregulares que como se verían en realidad en la pantalla.

### Ejemplo 3.1

**Problema:** Desarrolle y programe una versión mejorada del algoritmo de punto medio que analice dos píxeles por adelantado, en lugar de uno.

**Respuesta:** Una alternativa al empleo del algoritmo de punto medio para la discretización de líneas es el algoritmo de doble paso introducido por Wu y Rokne [WU87]. Esta técnica es incremental, como el algoritmo de punto medio;  $(x_{i+1}, y_{i+1})$  se puede calcular a partir de  $(x_i, y_i)$  usando sólo aritmética entera. El algoritmo de punto medio determina primero la dirección (pendiente) de la línea

y luego usa una variable de decisión para elegir entre dos pixeles alternativos en cada paso. El algoritmo de doble paso reduce a la mitad el número de decisiones, es decir, las opciones para el pixel siguiente. Esto se logra examinando el siguiente *par* de pixeles en lugar de evaluar sólo el siguiente. Wu [WU87] demostró que pueden ocurrir cuatro patrones:



Wu mostró que el patrón 1 y el patrón 4 no pueden ocurrir en la misma línea. Así mismo, si la pendiente de la línea es mayor que  $\frac{1}{2}$ , no puede ocurrir el patrón 1. En forma parecida, si la pendiente de la línea es menor que  $\frac{1}{2}$ , no puede presentarse el patrón 4. Por lo tanto, si evaluamos la pendiente, la elección se reduce a uno de tres patrones: 1, 2, 3 o 2, 3, 4. Veamos el caso en el cual la pendiente se halla entre 0 y  $\frac{1}{2}$ . Esto excluye el patrón 4, como ya indicamos. Hay que determinar si se usa el patrón 1 o alguno de los patrones 2 y 3 para llenar la imagen de trama. La variable de decisión  $d$  se asigna inicialmente como  $d = 4dy - dx$ . Entonces, para cada incremento (un paso de dos unidades de trama), se evalúa  $d < 0$  para determinar si se emplea el patrón 1. Si  $d$  es mayor o igual que 0, hay que determinar cuál de los patrones 2 y 3 es el que se usará. Para esto sólo hay que probar  $d < 2dy$ . Para incrementar  $d$  se emplea la regla siguiente:

$$\begin{aligned} d_{i+1} &= d_i + 4dy && \text{si } d_i < 0 \text{ (patrón 1),} \\ d_{i+1} &= d_i + 4dy - 2dx && \text{en caso contrario (patrón 2 o 3).} \end{aligned}$$

El algoritmo de punto medio se puede modificar para aprovechar esta mejora, de la siguiente manera:

Código que implementa el algoritmo mejorado de línea de punto medio.

```
void doble_paso (int x0, int y0, int x1, int y1)
{
    int x_actual, incr_1, incr_2, cond, dx, dy, d;

    /* Código del ciclo interno para el caso (0 < pendiente < 1/2)
       La rutina dibujar_pixeles requiere el patrón y la posición x actual.
       Esto se debe a que para el último pixel quizás sólo se requiera dibujar un pixel y no
       todo el patrón. Si se dibujara todo el patrón, la linea podría extenderse demasiado */

    /* Establecer la asignación de valor iniciales para el ciclo interior */
    dx = x1 - x0;
```

```

dy = y1 - y0;
x_actual = x0;
incr_1 = 4*dy;
incr_2 = 4*dy - 2*dx;
cond = 2 * dy;
d = 4*dy - dx;

while (x_actual < x1) { /* se requiere más discretización */
    if (d < 0) { /* primera decisión */
        dibujar_pixeles (PATTERN_1, x_actual);
        d += incr_1;
    } else {
        if (d < cond) /* no fue el primer caso; es 2 o 3 */
            dibujar_pixeles (PATTERN_2, x_actual);
        else
            dibujar_pixeles (PATTERN_3, x_actual);
        d += incr_2;
    }
    x_actual += 2;
}
}

```

Como ocurre en el algoritmo de punto medio, todos los cálculos se pueden realizar con sumas enteras y multiplicaciones por 2. Si la pantalla es multinivel, en lugar de discriminar entre los patrones 2 y 3, podrían dibujarse ambos a media intensidad para obtener gratuitamente una forma de eliminación de artefacto de discretización! Wyvill [WYVI90] notó que se puede aprovechar la simetría con respecto al punto medio y discretizar la línea simultáneamente desde los dos puntos extremos, con lo cual se duplica la velocidad del algoritmo. El algoritmo completo de doble paso simétrico para líneas, codificado en C, puede hallarse en [WYVI90].

### 3.2.3 Aspectos adicionales

**Orden de los puntos extremos.** Entre las complicaciones que deben considerarse es que hay que asegurar que una línea de  $P_0$  a  $P_1$  contenga el mismo número de pixeles que la línea de  $P_1$  a  $P_0$ , para que la apariencia de la línea sea independiente del orden de especificación de los puntos extremos. El único lugar donde la elección del pixel depende de la dirección de la línea es cuando ésta pasa exactamente por el punto medio y la variable de decisión es cero; si vamos de izquierda a derecha, elegimos  $E$  para este caso. Por simetría, si vamos de derecha a izquierda, elegiríamos  $O$  (oeste) para  $d = 0$ , pero con ello se escogería un pixel una unidad hacia arriba sobre el eje  $y$  con respecto al que se escogió para el rastreo de izquierda a derecha. Por lo tanto, tenemos que elegir  $SO$  (sureste) cuando  $d = 0$  en el rastreo de derecha a izquierda. Hay que efectuar ajustes similares con líneas que tienen otras pendientes.

La solución alternativa de intercambiar los puntos extremos de una línea según se requiera para que la discretización siempre siga la misma dirección, no

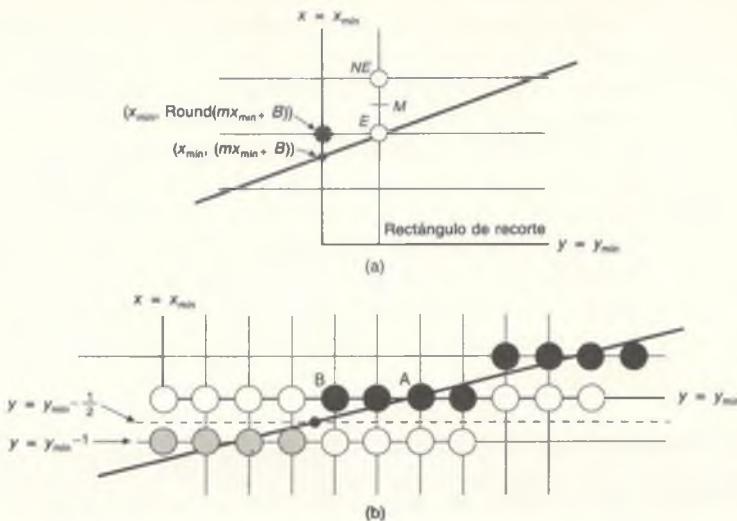
funcionará si usamos estilos de línea. El estilo de la línea siempre *ancla* la mazcarilla de escritura especificada en el punto inicial, que sería el inferior izquierdo, independientemente de la dirección de la línea. Con esto no siempre se obtiene el efecto visual deseado. Específicamente, en el caso de un patrón de línea punto-guion de, digamos, 111100, nos gustaría que el patrón comenzara en el punto inicial especificado, no automáticamente en el inferior izquierdo. Además, si el algoritmo siempre coloca los puntos extremos en orden canónico, el patrón podría ir de izquierda a derecha en un segmento y de derecha a izquierda en el adjunto, como función de la pendiente de la segunda línea; esto crearía una discontinuidad inesperada en el vértice compartido, donde el patrón debería fluir sin interrupciones de un segmento de línea al siguiente.

**Inicio en la arista de un rectángulo de recorte.** También hay que modificar el algoritmo para aceptar una línea que ha sido recortada analíticamente con uno de los algoritmos de la sección 3.9. En la figura 3.8(a) se muestra una línea recortada en la arista izquierda,  $x = x_{\min}$  del rectángulo de recorte. El punto de intersección de la línea con la arista tiene una coordenada  $x$  entera pero una coordenada  $y$  real. El pixel en la arista izquierda,  $(x_{\min}, \text{round}(mx_{\min} + B))$ , es el mismo pixel que se dibujaría en este valor de  $x$  si se usara el algoritmo incremental para la línea no recortada.<sup>5</sup> Dado este valor de pixel inicial, debemos asignar un valor inicial a la variable de decisión como el punto medio entre las posiciones  $E$  y  $NE$  en la siguiente columna. Es importante percibirse de que esta estrategia produce la secuencia de píxeles correcta, algo que no hará el recorte de la línea en la frontera  $x_{\min}$  seguido de la discretización de la línea recortada de  $(x_{\min}, \text{round}(mx_{\min} + B))$  a  $(x_1, y_1)$  usando el algoritmo entero de línea de punto medio: ¡la línea recortada tiene diferente pendiente!

La situación se complica si la línea interseca una arista horizontal en lugar de vertical, como se ilustra en la figura 3.8(b). Para el tipo de líneas poco inclinadas que se muestra, habrá varios píxeles en la línea de rastreo  $y = y_{\min}$  que corresponden a la arista inferior de la región de recorte. Queremos contar cada uno de estos píxeles como si estuvieran dentro de la región de recorte, pero el cálculo de la intersección analítica de la línea con la línea de rastreo  $y = y_{\min}$  para luego redondear el valor  $x$  del punto de intersección produciría el pixel  $A$ , no el punto izquierdo de la secuencia de píxeles que se presenta, el pixel  $B$ . A partir de la figura se puede observar que el pixel del extremo izquierdo de la secuencia,  $B$ , es el que está justo arriba y a la derecha del lugar de la malla donde la primera línea cruza por encima del punto medio  $y = y_{\min} - \frac{1}{2}$ . Por lo tanto, basta hallar la intersección de la línea con la línea horizontal  $y = y_{\min} - \frac{1}{2}$  y redondear hacia arriba el valor  $x$ ; el primer pixel,  $B$ , es entonces el que está en  $(\text{round}(x_{y_{\min} - \frac{1}{2}}), y_{\min})$ .

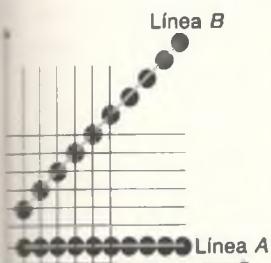
Por último, el algoritmo incremental de punto medio funciona incluso si los puntos extremos se especifican en un paquete gráfico de barrido de punto

<sup>5</sup> Cuando  $mx_{\min} + B$  está exactamente a la mitad del camino entre las líneas horizontales de la malla, debemos redondear hacia abajo. Esto se debe a la elección del pixel  $E$  cuando  $d = 0$ .



**Figura 3.8** Inicio de la línea en una frontera de recorte. (a) Intersección con una arista vertical. (b) Intersección con una arista horizontal (los pixeles en gris están en la línea pero fuera del rectángulo de recorte).

flotante; la única diferencia es que ahora los incrementos serán reales, lo mismo que la aritmética.



**Figura 3.9** Variación de la intensidad de las líneas de rastreo como función de la pendiente.

**Variación de la intensidad de una línea como función de la pendiente.** Considere las dos líneas discretizadas que aparecen en la figura 3.9. La línea *B*, la diagonal, tiene pendiente de 1 y por consiguiente es  $\sqrt{2}$  veces más larga que *A*, la línea horizontal. Sin embargo, se emplea el mismo número de pixeles (10) para representar cada línea. Si la intensidad de cada pixel es *I*, la intensidad por unidad de longitud de la línea *A* es *I*, mientras que para la línea *B* es *I*/ $\sqrt{2}$ ; el observador puede detectar fácilmente esta discrepancia. En una pantalla de dos niveles no hay forma de resolver este problema, pero en un sistema de *n* bits por pixel podemos compensarlo estableciendo la intensidad como función de la pendiente de la línea. La eliminación del artefacto de discretización (*antialiasing*), que se analiza en la sección 3.14, obtiene resultados aún mejores al tratar la línea como un rectángulo delgado y calcular intensidades apropiadas para los pixeles en cada columna que se encuentre sobre el rectángulo o cerca de él.

El tratamiento de la línea como rectángulo es también una manera de crear líneas gruesas. En la sección 3.7 mostraremos cómo modificar los algoritmos de discretización básicos para tratar primitivas gruesas y primitivas cuya apariencia se ve afectada por los atributos de estilo de línea y de estilo de pincel.

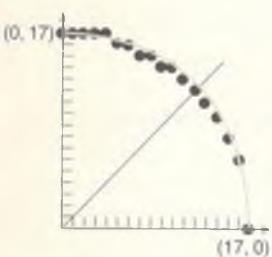
**Primitivas huecas compuestas por líneas.** Una vez que sabemos cómo discretizar líneas, ¿cómo discretizamos las primitivas compuestas por líneas? Las

polilíneas se pueden discretizar un segmento de línea a la vez. La discretización de rectángulos y polígonos como primitivas de definición de áreas también se puede efectuar un segmento de línea a la vez, pero el resultado sería que algunos pixeles se dibujarían fuera del área de la primitiva (en las secciones 3.4 y 3.5 se presentan algoritmos especiales para tratar este problema). Hay que tener cuidado de dibujar sólo una vez los vértices compartidos de las polilíneas, ya que si lo hacemos dos veces se podría alterar su color o asignarse igual al fondo si se escribe en modo `xor` en la pantalla, o bien tratar de escribir con doble intensidad en una grabadora de película fotográfica. De hecho, también es posible que dos segmentos de línea que estén muy próximos o se crucen comparten otros pixeles. En el ejercicio 3.7 se analiza este tema y la diferencia entre una polilínea y una secuencia de segmentos de línea conectados.

### 3.3 Discretización de círculos

Aunque SRGP no ofrece una primitiva de círculo, la implantación se beneficiará con el tratamiento del arco elíptico circular como un caso especial por su simetría de ocho lados, tanto para discretización como para recortes. La ecuación de un círculo centrado en el origen es  $x^2 + y^2 = R^2$ . Los círculos que no están centrados en el origen se pueden trasladar a éste usando cantidades enteras y luego discretizarse, escribiendo los pixeles con el desplazamiento apropiado. Hay varias maneras sencillas pero ineficientes para discretizar un círculo. Si resolvemos para  $y$  en la ecuación implícita del círculo, obtenemos la función explícita  $y = f(x)$  como

$$y = \pm\sqrt{R^2 - x^2}.$$



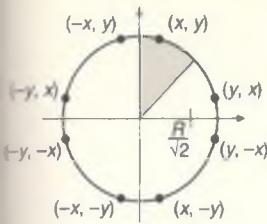
**Figura 3.10**

Cuarto de círculo generado con incrementos unitarios en  $x$ , calculando  $y$  y luego redondeando. Los valores únicos de  $y$  para cada  $x$  producen huecos.

Para dibujar un cuarto de círculo (los otros cuartos se dibujan por simetría), podemos incrementar  $x$  de 0 a  $R$  en pasos unitarios, resolviendo para  $+y$  en cada paso. Este método funciona, pero es ineficiente por las operaciones de multiplicación y raíz cuadrada. Además, el círculo tendrá huecos grandes (a menos que  $R$  sea muy grande) para los valores de  $x$  cercanos a  $R$ , ya que la pendiente del círculo se hacen infinita en ese punto (véase la Fig. 3.10). Un método de similar ineficiencia pero que sí evita los grandes huecos es graficar ( $R \cos\theta$ ,  $R \sin\theta$ ) aumentando  $\theta$  de  $0^\circ$  a  $90^\circ$ .

#### 3.3.1 Simetría de ocho lados

Podemos mejorar el proceso de dibujo de la sección anterior aprovechando más la simetría en un círculo. Considere primero un círculo centrado en el origen. Si el punto  $(x, y)$  se halla en el círculo, podemos calcular trivialmente siete puntos adicionales en éste, como se ilustra en la figura 3.11. Por lo tanto, sólo tenemos



**Figura 3.11**  
Ocho puntos simétricos en un círculo.

que calcular un segmento de  $45^\circ$  para determinar todo el círculo. Para un círculo centrado en el origen, los ocho puntos simétricos se pueden presentar con la función *puntos\_círculo* (la función es fácil de generalizar al caso de círculos con orígenes arbitrarios):

```
void puntos_círculo (float x, float y, int valor);
{
    escribir_pixel (x, y, valor);
    escribir_pixel (y, x, valor);
    escribir_pixel (y, -x, valor);
    escribir_pixel (x, -y, valor);
    escribir_pixel (-x, -y, valor);
    escribir_pixel (-y, -x, valor);
    escribir_pixel (-y, x, valor);
    escribir_pixel (-x, y, valor);
}
```

No es conveniente llamar a *puntos\_círculo* cuando  $x = y$ , ya que cada uno de cuatro puntos se asignaría dos veces; es fácil modificar el código para manejar esta condición limitante.

### 3.3.2 Algoritmo de círculo de punto medio

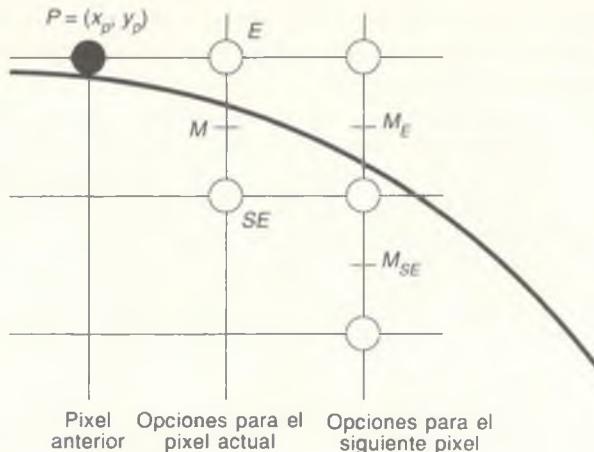
Bresenham [BRES77] desarrolló un generador de círculos incremental más eficiente que los métodos que hemos analizado. El algoritmo se concibió originalmente para usarse en graficadores y genera todos los puntos en un círculo centrado en el origen por medio de incrementos a lo largo del círculo. Obtendremos un algoritmo similar, empleando de nuevo el criterio de punto medio, en el cual, para el caso de punto central y radio enteros, genera el mismo conjunto óptimo de pixeles. Además, el código resultante es en esencia el mismo que está especificado en la patente 4 371 933 [BRES83].

Consideramos sólo  $45^\circ$  de un círculo, el segundo octante de  $x = 0$  a  $x = y = R/\sqrt{2}$  y usamos la función *puntos\_círculo* para mostrar los puntos de todo el círculo. Como hicimos con el algoritmo de línea de punto medio, la estrategia consiste en seleccionar cuál de dos pixeles es el más cercano al círculo, evaluando una función en el punto medio entre los dos pixeles. En el segundo octante, si el pixel  $P$  en  $(x_p, y_p)$  se eligió previamente como el más cercano al círculo, la siguiente elección de pixel es entre el pixel  $E$  y el pixel  $SE$  (véase la Fig. 3.12).

Sea  $F(x, y) = x^2 + y^2 - R^2$ ; esta función es cero en el círculo, positiva fuera del círculo y negativa dentro de él. Podemos ver que si el punto medio entre los pixeles  $E$  y  $SE$  está fuera del círculo, entonces el pixel  $SE$  es el más cercano al círculo. Por otra parte, si el punto medio está dentro del círculo, el pixel  $E$  es el más cercano.

Como se hizo con las líneas, la elección se basa en la variable de decisión  $d$ , la cual constituye el valor de la función en el punto medio,

$$d_{\text{viejo}} = F(x_p + 1, y_p - \frac{1}{2}) = (x_p + 1)^2 + (y_p - \frac{1}{2})^2 - R^2.$$



**Figura 3.12** Cuadrículado de pixeles para el algoritmo de círculo de punto medio, que muestra  $M$  y los pixeles  $E$  y  $SE$  entre los cuales podemos elegir.

Si  $d_{\text{viejo}} < 0$ , se escoge  $E$  y el siguiente punto medio será un incremento en  $x$ . Entonces,

$$d_{\text{nuevo}} = F(x_p + 2, y_p - \frac{1}{2}) = (x_p + 2)^2 + (y_p - \frac{1}{2})^2 - R^2,$$

y  $d_{\text{nuevo}} = d_{\text{viejo}} + (2x_p + 3)$ ; por lo tanto, el incremento  $\Delta_E = 2x_p + 3$ .

Si  $d_{\text{viejo}} \geq 0$ , se elige  $SE$ <sup>6</sup> y el siguiente punto medio será un incremento hacia arriba en  $x$  y un incremento hacia abajo en  $y$ . Entonces,

$$d_{\text{nuevo}} = F(x_p + 2, y_p - \frac{3}{2}) = (x_p + 2)^2 + (y_p - \frac{3}{2})^2 - R^2.$$

Como  $d_{\text{nuevo}} = d_{\text{viejo}} + (2x_p - 2y_p + 5)$ , el incremento  $\Delta_{SE} = 2x_p - 2y_p + 5$ .

Recuerde que, en el caso lineal,  $\Delta_E$  y  $\Delta_{NE}$  eran constantes; sin embargo, en el caso cuadrático,  $\Delta_E$  y  $\Delta_{SE}$  varían en cada paso y son funciones de los valores específicos de  $x_p$  y  $y_p$  en el pixel elegido en la iteración previa. Como estas funciones se expresan en términos de  $(x_p, y_p)$ , llamamos a  $P$  **punto de evaluación**. Las funciones  $\Delta$  se pueden evaluar directamente en cada caso colocando en ellas los valores de  $x$  y  $y$  para el pixel elegido en la iteración anterior. Esta evaluación directa no requiere demasiados cálculos, ya que las funciones son lineales.

En resumen, en cada iteración del algoritmo efectuamos los mismos dos pasos que realizamos para la línea: (1) elegir el pixel con base en el signo de la

<sup>6</sup> La elección es  $SE$  cuando  $d = 0$  difiere de nuestra selección en el algoritmo de línea y es arbitraria. El lector puede simular el algoritmo en forma manual y ver que, para  $R = 17$ , cambia un pixel con esta opción.

variable  $d$  calculada en la iteración previa y (2) actualizar la variable de decisión  $d$  con la  $\Delta$  que corresponda a la elección del pixel. La única diferencia con respecto al algoritmo de la línea es que al actualizar  $d$  se evalúa una función lineal del punto de evaluación.

Lo único que falta es calcular la condición inicial. Si limitamos el algoritmo a los radios enteros en el segundo octante, sabemos que el pixel de partida se encuentra en el círculo en  $(0, R)$ . El siguiente punto medio está entonces en  $(1, R - \frac{1}{2})$  y  $F(1, R - \frac{1}{2}) = 1 + (R^2 - R + \frac{1}{4}) - R^2 = \frac{5}{4} - R$ . Ahora podemos implantar este algoritmo en forma directa, como en el programa 3.3. Observe la similitud entre la estructura de este algoritmo y el de líneas.

El problema con esta versión es que estamos obligados a efectuar aritmética real, por la asignación de un valor inicial fraccionario a  $d$ . Aunque la función se puede modificar con facilidad para manejar círculos que no se localizan en centros enteros o que no tienen radios enteros, nos gustaría contar con una versión más eficiente, completamente entera. Por ende, tenemos que efectuar una sencilla transformación de programa para eliminar las fracciones.

Primero definimos una nueva variable de decisión,  $h$ , como  $h = d - \frac{1}{4}$  y sustituimos este valor en lugar de  $d$  en el código. El valor inicial es ahora  $h = 1 - R$  y la comparación  $d < 0$  se convierte en  $h < -\frac{1}{4}$ . Sin embargo, como  $h$  inicia con un valor entero y se incrementa con valores enteros ( $\Delta_E$  y  $\Delta_{SE}$ ), podemos cambiar la comparación a  $h < 0$ . Tenemos ahora un algoritmo entero en función de  $h$ ; para que sea consistente con el algoritmo de línea, reemplazaremos  $d$  con  $h$  en toda la expresión. El algoritmo final, usando exclusivamente aritmética de enteros, se presenta en el programa 3.4.

### Programa 3.3

```
void círculo_punto_medio (int radio, int valor)

    int x, y;
    float d;

    x = 0; /* Valor inicial */
    y = radio;
    d = 5.0 / 4 - radio;
    puntos_círculo (x, y, valor);
    while (y > x) {
        if (d < 0) /* Seleccionar E */
            d += x * 2.0 + 3;
        x++;
    } else /* Seleccionar SE */
        d += (x - y) * 2.0 + 5;
        x++;
        y--;
    puntos_círculo (x, y, valor);
```

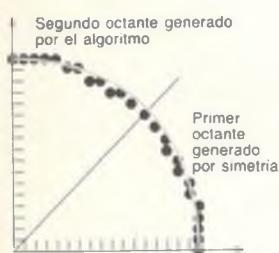
*El algoritmo de  
discretización de círculo  
de punto medio.*

**Programa 3.4**

*El algoritmo entero de discretización de círculo de punto medio.*

```
void circulo_punto_medio (int radio, int valor)
{
    int x, y, d;
    x = 0;                                /* Valor inicial */
    y = radio;
    d = 1 - radio;
    puntos_círculo (x, y, valor);
    while (y > x) {
        if (d < 0)                      /* Seleccionar E */
            d += x * 2 + 3;
        x++;
    } else {                               /* Seleccionar SE */
        d += (x - y) * 2 + 5;
        x++;
        y--;
    }
    puntos_círculo (x, y, valor);
}
```

En la figura 3.13 se muestra el segundo octante de un círculo de radio 17 generado con el algoritmo y el primer octante generado por simetría (compare los resultados con la figura 3.10).



**Figura 3.13**  
Segundo octante de un círculo generado con el algoritmo de punto medio y primer octante generado por simetría.

**Diferencias de segundo orden.** Podemos mejorar el rendimiento del algoritmo de círculo de punto medio usando de manera más exhaustiva la técnica de cálculo incremental. Observamos que las funciones  $\Delta$  son ecuaciones lineales y las calculamos directamente. Sin embargo, cualquier polinomio se puede calcular en forma incremental, como lo hicimos con las variables de decisión de la línea y el círculo. De hecho, estamos calculando **diferencias parciales de primer y segundo orden**, una técnica útil que veremos de nuevo en el capítulo 9. La estrategia es evaluar la función de manera directa en dos puntos adyacentes para calcular la diferencia (la cual, en el caso de los polinomios, siempre es un polinomio de menor grado), para luego aplicar esa diferencia en cada iteración.

Si elegimos  $E$  en la iteración actual, el punto de evaluación se mueve de  $(x_p, y_p)$  a  $(x_p + 1, y_p)$ . Como hemos visto, la diferencia de primer orden es  $\Delta_{E\text{viejo}}$  en  $(x_p, y_p) = 2x_p + 3$ . Por lo tanto,

$$\Delta_{E\text{nuevo}} \text{ en } (x_p + 1, y_p) = 2(x_p + 1) + 3$$

y la diferencia de segundo orden es  $\Delta_{E\text{nuevo}} - \Delta_{E\text{viejo}} = 2$ .

De manera similar,  $\Delta_{SE\text{viejo}}$  en  $(x_p, y_p) = 2x_p - 2y_p + 5$ . Así,

$$\Delta_{SE\text{nuevo}} \text{ en } (x_p + 1, y_p) = 2(x_p + 1) - 2y_p + 5,$$

y la diferencia de segundo orden es  $\Delta_{SE\text{nuevo}} - \Delta_{SE\text{viejo}} = 2$ .

Si elegimos  $SE$  en la iteración actual, el punto de evaluación se mueve de  $(x_p, y_p)$  a  $(x_p + 1, y_p - 1)$ . Por consiguiente,

$$\Delta_{E_{\text{nuevo}}} \text{ en } (x_p + 1, y_p - 1) = 2(x_p + 1) + 3,$$

y la diferencia de segundo orden es  $\Delta_{E_{\text{nuevo}}} - \Delta_{E_{\text{viejo}}} = 2$ . Además,

$$\Delta_{SE_{\text{nuevo}}} \text{ en } (x_p + 1, y_p - 1) = 2(x_p + 1) - 2(y_p - 1) + 5,$$

y la diferencia de segundo orden es  $\Delta_{SE_{\text{nuevo}}} - \Delta_{SE_{\text{viejo}}} = 4$ .

El algoritmo modificado consiste entonces en los siguientes pasos: (1) elegir el pixel con base en el signo de la variable  $d$  calculada durante la iteración previa; (2) actualizar la variable de decisión  $d$  con  $\Delta_E$  o  $\Delta_{SE}$  usando el valor de la  $\Delta$  correspondiente calculada durante la iteración previa; (3) actualizar las  $\Delta$  para tener presente el movimiento hacia el nuevo pixel, usando las diferencias constantes calculadas con anterioridad; y (4) efectuar el movimiento.  $\Delta_E$  y  $\Delta_{SE}$  reciben su valor inicial usando el pixel de inicio  $(0, R)$ . En el programa 3.5 se presenta la función modificada que utiliza esta técnica.

### Programa 3.5

*El algoritmo de punto medio para la discretización de círculos usando diferencias de segundo orden.*

```
void círculo_punto_medio (int radio, int valor)
{
    /* Esta función utiliza diferencias parciales de segundo orden para calcular los
     * incrementos */
    /* en la variable de decisión. Se supone el centro del círculo en el origen */
    int x, y, d, delta_E, delta_SE;
    x = 0;                                /* Valor inicial */
    y = radio;
    d = 1 - radio;
    delta_E = 3;
    delta_SE = 5 - radio * 2;
    puntos_círculo = (x, y, valor);
    while (y > x) {
        if (d < 0) {                      /* Seleccionar E */
            d += delta_E;
            delta_E += 2;
            delta_SE += 2;
            x++;
        } else {                           /* Seleccionar SE */
            d += delta_SE;
            delta_E += 2;
            delta_SE += 4;
            x++;
            y--;
        }
        puntos_círculo (x, y, valor);
    }
}
```

## 3.4 Rellenado de rectángulos

La tarea de llenar primitivas se puede dividir en dos partes: la decisión de cuáles pixeles llenar (esto depende de la forma de la primitiva, según la haya modificado el recorte) y la decisión más sencilla de con qué valor llenarlas. Analizaremos primero el llenado de primitivas no recortadas con un color sólido y en la sección 3.6 hablaremos del llenado con patrones. En términos generales, la determinación de los pixeles que se llenarán consiste en tomar líneas de rastreo sucesivas, de izquierda a derecha, que intersequen la primitiva y llenen *tramos* de pixeles adyacentes que caigan dentro de la primitiva.

Para llenar un rectángulo con un color sólido, asignamos cada pixel que se encuentre en una línea de rastreo que va de la arista izquierda a la derecha con el mismo valor de pixel; es decir, llenamos cada tramo de  $x_{\min}$  a  $x_{\max}$ . Los tramos aprovechan la **coherencia espacial** de las primitivas: el hecho de que las primitivas a menudo no cambien de un pixel a otro en un tramo o de una línea de rastreo a otra. La forma general en que se aprovecha la coherencia es buscando únicamente los pixeles donde ocurren cambios. En el caso de una primitiva con sombreado sólido, todos los pixeles en un tramo se asignan al mismo valor, lo que proporciona la **coherencia de tramos**. El rectángulo con sombreado sólido también presenta una fuerte *coherencia de línea de rastreo*, ya que las líneas de rastreo consecutivas que intersecan el rectángulo son idénticas; más adelante usaremos la **coherencia de aristas** para polígonos generales. Aprovechamos los diversos tipos de coherencia no sólo para discretizar primitivas bidimensionales, sino también para generar primitivas tridimensionales, como se verá en la sección 13.1.

La capacidad para tratar en forma idéntica varios pixeles dentro de un tramo tiene una importancia especial, ya que debe escribirse en la memoria gráfica una palabra a la vez para minimizar la cantidad de accesos a la memoria que ocupan mucho tiempo. En el caso de una pantalla de dos niveles, escribimos entonces 16 o 32 pixeles al mismo tiempo; si los tramos no están alineados por palabra, el algoritmo debe llevar a cabo el enmascaramiento de las palabras que contengan menos que el conjunto completo de pixeles. La necesidad de escribir con eficiencia en la memoria es idéntica en la implantación de `copyPixel`, como veremos dentro de poco en la sección 3.13. En nuestro código nos centramos en la definición de tramos e ignoramos los aspectos de escritura eficiente en la memoria (véase el Cap. 4 y el Ejer. 3.10).

La discretización de un rectángulo no es más que un ciclo `for` anidado:

```
for (y de  $y_{\min}$  a  $y_{\max}$  del rectángulo) {
    for (x de  $x_{\min}$  a  $x_{\max}$ ) {
        escribir_pixel (x, y, valor);
    }
}
```

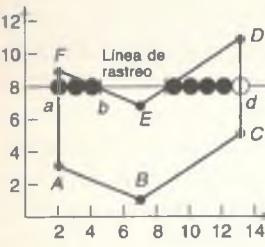
En esta solución directa surge un problema interesante, similar al problema de discretización de una polilínea con segmentos que comparten píxeles. Considere dos rectángulos que comparten una arista común. Si discretizamos cada uno de los rectángulos por separado, escribiremos dos veces cada uno de los píxeles en la arista compartida, lo cual, como señalamos antes, es indeseable. Este problema es una manifestación de un problema mayor de las primitivas de definición de áreas: la determinación de cuáles píxeles pertenecen a una primitiva y cuáles no. Obviamente, los píxeles que caen en el interior matemático de una primitiva de definición de área pertenecen a ella, pero ¿qué sucede con los píxeles en la frontera? Si viéramos un sólo rectángulo (o simplemente pensáramos en forma matemática en el problema) la respuesta más sencilla sería incluir todos los píxeles en la frontera. Sin embargo como deseamos evitar el problema de la doble discretización de las aristas compartidas, tenemos que definir una regla que asigne en forma única los píxeles de las fronteras.

Una regla sencilla es que un pixel de frontera —es decir, un pixel que se encuentra en una arista— no se considera parte de la primitiva si el medio plano definido por la arista y que contiene la primitiva está debajo o a la izquierda de la arista. De esta manera, se dibujarán los píxeles de las aristas izquierda e inferior, pero no aquellos que se encuentren en las aristas superior y derecha. Por lo tanto, una arista vertical compartida por dos rectángulos *pertenece* al del extremo derecho. En realidad, los tramos en un rectángulo representan un intervalo cerrado en el extremo izquierdo y abierto en el derecho.

Hay que señalar varios aspectos relacionados con esta regla. Primero, se aplica a polígonos arbitrarios además de rectángulos. Segundo, el vértice inferior izquierdo de un rectángulo seguiría dibujándose dos veces, por lo que necesitamos otra regla para tratar este caso especial, algo que haremos en la siguiente sección. Tercero, también podemos aplicar la regla a rectángulos y polígonos huecos. Cuarto, esta regla ocasiona que cada tramo no tenga el pixel del extremo derecho y que cada rectángulo no incluya su fila superior. Estos problemas ilustran que no existe una solución *perfecta* para el problema de no escribir dos veces los píxeles que (potencialmente) se encuentran en aristas compartidas, pero los implantadores por lo general consideran que es menor (causa menos distracción visual) si faltan píxeles en las aristas superior y derecha que el hecho de que desaparezcan píxeles o que se asignen con colores inesperados en el modo xor.

## 3.5 Rellenado de polígonos

El algoritmo general de discretización de polígonos que se describe a continuación maneja tanto polígonos convexos como cóncavos, incluso aquellos que se autointersecan o que tienen huecos interiores. Opera calculando los tramos que se hallan entre la arista izquierda y la derecha del polígono. Los extremos del

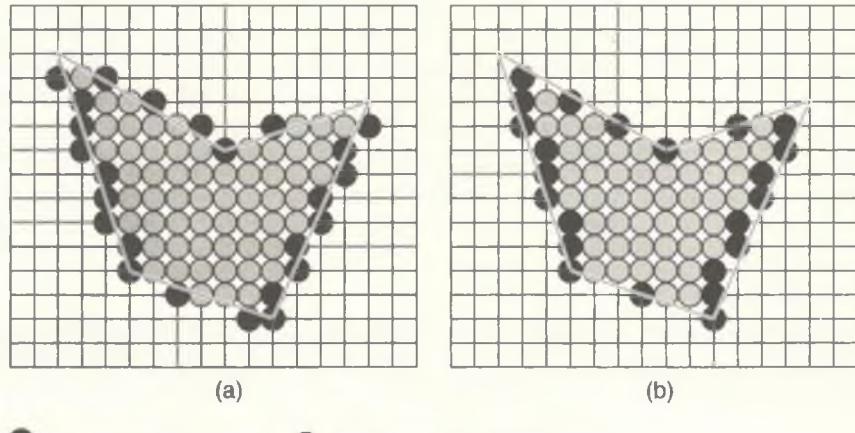


**Figura 3.14**  
Polígono y línea de rastreo 8.

tramo se obtienen con un algoritmo incremental que calcula una intersección arista-línea de rastreo a partir de la intersección con la línea de rastreo anterior. En la figura 3.14, donde se ilustra el proceso básico de discretización de un polígono, se presenta un polígono y una línea de rastreo que pasa por él. Las intersecciones de la línea de rastreo 8 con las aristas  $FA$  y  $CD$  yacen sobre coordenadas enteras, lo cual no ocurre con las de  $EF$  y  $DE$ ; las intersecciones se señalan en la figura con marcas verticales etiquetadas con las letras  $a$  a  $d$ .

Debemos determinar qué pixeles de cada línea de rastreo están dentro del polígono y asignar los pixeles correspondientes (en este caso, los tramos de  $x = 2$  a 4 y de 9 a 13) a sus valores apropiados. Al repetir este proceso para cada línea de rastreo que interseca al polígono, discretizamos todo el polígono, como se ilustra en la figura 3.15 para otro polígono.

En la figura 3.15(a) se muestran en negro los pixeles que definen los extremos de los tramos y en gris los pixeles interiores. Una forma directa de obtener los extremos es aplicar el algoritmo de discretización de líneas de punto medio a cada una de las aristas y mantener una tabla de los extremos de tramo para cada línea de rastreo, actualizando una entrada si se produce un nuevo pixel para una arista que extiende el tramo. Observe que esta estrategia produce algunos pixeles extremo fuera del polígono, los cuales fueron elegidos por el algoritmo de discretización porque eran los que yacían más cerca de una arista, sin considerar de qué lado estaban (el algoritmo de líneas no tiene la noción de lo que es el interior y el exterior). Sin embargo, no queremos dibujar los pixeles que se encuentran fuera de una arista compartida, ya que se entremeterían en las regiones de los polígonos vecinos y el resultado sería extraño si los polígonos fueran de colores diferentes. Evidentemente, es preferible dibujar sólo los pixeles que



**Figura 3.15** Tramos para un polígono. Los extremos se muestran en negro y los pixeles interiores en gris.  
(a) Extremos calculados con el algoritmo de punto medio. (b) Extremos interiores del polígono.

están dentro de la región, incluso si un pixel exterior estuviera más cerca de la arista. Por lo tanto, debemos ajustar de manera correspondiente el algoritmo de discretización; compare la figura 3.15(a) con la figura 3.15(b) y observe que varios píxeles que se hallaban fuera de la primitiva ideal no se dibujaron en la parte (b).

Con esta técnica, un polígono no se entremete (ni siquiera un pixel) en las regiones definidas por otras primitivas. Por consistencia podemos aplicar la misma técnica a los polígonos huecos, o podemos discretizar rectángulos y polígonos un segmento de línea a la vez, ¡en cuyo caso los polígonos huecos y rellenos no contienen los mismos píxeles de frontera!

Como en el algoritmo original de punto medio, se usa un algoritmo incremental para calcular los extremos de los tramos en una línea de rastreo a partir de las correspondientes a la línea de rastreo anterior, sin tener que calcular en forma analítica las intersecciones de la línea de rastreo con cada arista del polígono. Por ejemplo, en la línea de rastreo 8 de la figura 3.14 hay dos tramos de píxeles dentro del polígono. Los tramos se pueden llenar mediante un proceso de tres pasos:

1. Hallar las intersecciones de la línea de rastreo con todas las aristas del polígono.
2. Ordenar las intersecciones aumentando la coordenada  $x$ .
3. Colocar todos los píxeles entre pares de intersecciones que se encuentren dentro del polígono, usando la regla de paridad impar para determinar si un punto está dentro de la región: la paridad es inicialmente par y cada intersección que se detecte invierte el bit de paridad; se dibuja cuando la paridad es impar, no se dibuja si es par.

Los dos primeros pasos de este proceso, la detección de intersecciones y su ordenamiento, se tratan en la sección siguiente. Veamos ahora la estrategia de rellenado del tramo. En la figura 3.14, la lista ordenada de las coordenadas  $x$  es  $(2, 4.5, 8.5, 13)$ . El paso 3 requiere cuatro elaboraciones:

- 3.1 Dada una intersección con un valor  $x$  arbitrario y fraccionario, ¿cómo determinamos cuál de los dos píxeles a cada lado de la intersección es el interior?
- 3.2 ¿Cómo tratamos el caso especial de las intersecciones en coordenadas enteras de los píxeles?
- 3.3 ¿Cómo tratamos el caso especial del paso 3.2 para vértices compartidos?
- 3.4 ¿Cómo tratamos el caso especial del paso 3.2 si los vértices definen una arista horizontal?

Para manejar el caso 3.1, decimos que si nos aproximamos hacia la derecha a una intersección fraccionaria y estamos dentro del polígono, redondeamos hacia abajo la coordenada  $x$  de la intersección para definir el pixel interior; si estamos fuera del polígono, redondeamos hacia arriba para quedar dentro. El caso 3.2 se maneja aplicando el criterio que se utilizó para evitar conflictos en las aristas compartidas de los rectángulos: Si el pixel del extremo izquierdo de

un tramo tiene coordenada  $x$  entera, lo definimos como interior; si el pixel del extremo derecho tiene coordenada  $x$  entera, lo definimos como exterior. Para el caso 3.3, contamos el vértice  $y_{\min}$  de una arista en el cálculo de paridad, pero no incluimos el vértice  $y_{\max}$ ; así, sólo se dibujará un vértice  $y_{\max}$  si es el vértice  $y_{\min}$  de la arista adyacente. Por ejemplo, el vértice  $A$  en la figura 3.14 se cuenta una vez en el cálculo de paridad porque es el vértice  $y_{\min}$  de la arista  $FA$  pero también el vértice  $y_{\max}$  de la arista  $AB$ . De esta manera, tanto las aristas como los tramos se tratan como intervalos cerrados en su valor mínimo y abiertos en su valor máximo. Obviamente, también funcionaría la regla opuesta, pero la que usamos parece más natural ya que trata el punto extremo mínimo como punto de entrada y el máximo como punto de salida. Al tratar el caso 3.4 (aristas horizontales) el efecto que se desea es que, como en los rectángulos, se dibujen las aristas inferiores pero no las superiores. Como veremos en la siguiente sección, esto sucede en forma automática si no contamos los vértices de las aristas, ya que no son vértices  $y_{\min}$  ni  $y_{\max}$ .

Aplicaremos estas reglas a la línea de rastreo 8 de la figura 3.14, que no incluye vértices. Rellenamos todos los píxeles del punto  $a$ , pixel  $(2, 8)$ , al primer pixel a la izquierda del punto  $b$ , pixel  $(4, 8)$ , y luego del primer pixel a la derecha del punto  $c$ , pixel  $(9, 8)$ , a un pixel a la izquierda del punto  $d$ , pixel  $(12, 8)$ . Para la línea de rastreo 3, el vértice  $A$  se cuenta una vez porque es el vértice  $y_{\min}$  de la arista  $FA$  pero el vértice  $y_{\max}$  de la arista  $AB$ ; esto provoca paridad impar, por lo que dibujamos el tramo desde allí hasta un pixel a la izquierda de la intersección con la arista  $CB$ , donde la paridad se asigna como par y termina el tramo. La línea de rastreo 1 sólo toca el vértice  $B$ ; las aristas  $AB$  y  $BC$  tienen sus vértices  $y_{\min}$  en  $B$ , que por ende se cuenta dos veces y deja la paridad como par. Este vértice actúa como tramo nulo: se ingresa en el vértice, se dibuja el pixel y se sale del vértice. Aunque estos mínimos locales dibujan un solo pixel, no se dibuja ninguno en un máximo local, como sería la intersección de la línea de rastreo 9 con el vértice  $F$ , compartido por las aristas  $FA$  y  $EF$ . Los dos vértices son  $y_{\max}$  y por lo tanto no afectan a la paridad, que permanece par.

### 3.5.1 Aristas horizontales

Las aristas horizontales se tratan de manera adecuada si no se cuentan sus vértices, como podemos ver al examinar varios casos en la figura 3.16. Considere la arista inferior  $AB$ . El vértice  $A$  es  $y_{\min}$  para la arista  $JA$ , y  $AB$  no contribuye. Por lo tanto, la paridad es impar y se dibuja el tramo  $AB$ . La arista vertical  $BC$  tiene su vértice  $y_{\min}$  en  $B$ , pero una vez más  $AB$  no contribuye. La paridad se convierte en par y termina el tramo. En el vértice  $J$ , la arista  $IJ$  tiene un vértice  $y_{\min}$ , pero no así la arista  $JA$ , de manera que la paridad se convierte en impar y el tramo se dibuja hasta la arista  $BC$ . El tramo que comienza en la arista  $IJ$  y continúa hasta  $C$  no ve ningún cambio en  $C$ , ya que este vértice es de tipo  $y_{\max}$  para  $BC$ , por lo cual el tramo continúa por la arista inferior  $CD$ ; sin embargo, al llegar a  $D$  la arista  $DE$  tiene un vértice  $y_{\min}$  y se restablece la paridad como par, terminando así el tramo. En  $I$ , la arista  $IJ$  tiene su vértice  $y_{\max}$  y la arista  $HI$

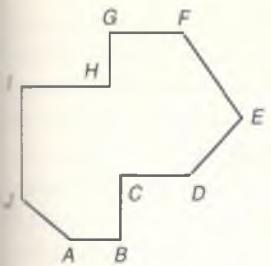


Figura 3.16  
Aristas horizontales en un polígono.

tampoco contribuye, de manera que la paridad permanece como par y no se dibuja la arista superior  $IH$ . Sin embargo, en  $H$  la arista  $GH$  tiene un vértice  $y_{\min}$  y la paridad se convierte en impar, por lo cual se dibuja el tramo desde  $H$  al pixel a la izquierda de la intersección con la arista  $EF$ . Finalmente, no existe un vértice  $y_{\min}$  en  $G$  ni en  $F$ , por lo cual no se dibuja la arista superior  $FG$ .

El algoritmo anterior abarca los vértices compartidos de un polígono, las aristas compartidas por dos polígonos adyacentes y las aristas horizontales. Permite también los polígonos autointersecados. Como señalamos, no funciona de manera perfecta ya que omite pixeles, y algo peor, no puede evitar la escritura múltiple de pixeles compartidos si no mantiene un historial: considere las aristas compartidas por más de dos polígonos o un vértice  $y_{\min}$  compartido por dos triángulos que son disjuntos fuera de este punto (véase el Ejer. 3.11).

### 3.5.2 Astillas

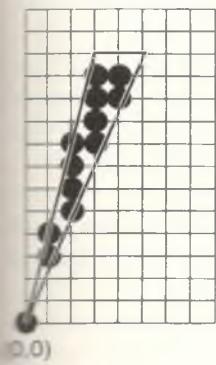


Figura 3.17  
Discretización de astillas  
en polígonos.

Nuestro algoritmo de discretización tiene otro problema que no se puede resolver de manera tan satisfactoria como el de las aristas horizontales: los polígonos con aristas tan cercanas que crean una **astilla**, un área poligonal tan delgada que su interior no contiene un tramo distinto para cada línea de rastreo. Considere, por ejemplo, el triángulo de  $(0, 0)$  a  $(3, 12)$  a  $(5, 12)$  a  $(0, 0)$  que se presenta en la figura 3.17. Por la regla de que sólo se dibujan los pixeles que están dentro o en una arista inferior o izquierda, habrá varias líneas de rastreo con un solo pixel o sin pixeles. El problema de pixeles *faltantes* es otro ejemplo del problema de *artefacto de discretización (aliasing)*, es decir, de la representación de una señal continua con una aproximación discreta. Si tuviéramos varios bits por pixel, podríamos emplear técnicas de eliminación de artefacto de discretización como las que se presentan para líneas en la sección 3.14. Esta eliminación de artefacto implicaría relajar nuestra regla de dibujar *únicamente los pixeles que están dentro o en una arista izquierda o inferior*, para permitir que los pixeles de frontera e incluso los exteriores asuman valores de intensidad que varíen como función de la distancia entre el centro del pixel y la primitiva; de esta manera, varias primitivas pueden contribuir al valor de un pixel.

### 3.5.3 Coherencia de aristas y algoritmo de línea de rastreo

El paso 1 de nuestro procedimiento, el cálculo de las intersecciones, debe realizarse con mucho ingenio o de lo contrario será muy lento. En particular, hay que evitar la técnica burda de evaluar cada arista del polígono para detectar la intersección con cada línea de rastreo nueva. Con frecuencia, sólo algunas de las aristas nos interesan para una línea de rastreo. Además, se observa que varias de las aristas intersecadas por una línea de rastreo  $i$  también son intersecadas por la línea de rastreo  $i + 1$ . Esta *coherencia de aristas* ocurre en una arista para todas las líneas de rastreo que la intersequen. Al avanzar de una línea de rastreo a otra, podemos calcular la nueva intersección  $x$  de la arista con base en la intersección  $x$  anterior, así como calculamos el siguiente pixel a partir del actual en la discretización de líneas de punto medio, usando

$$x_{i+1} = x_i + 1/m,$$

donde  $m$  es la pendiente de la arista. En el algoritmo de punto medio para la discretización evitamos la aritmética fraccionaria al calcular una variable de decisión entera y revisamos únicamente su signo para elegir el pixel más cercano a la línea matemática; en este caso nos gustaría usar aritmética entera a fin de efectuar el redondeo necesario para el cálculo del pixel interior más cercano.

Considere las líneas con pendiente mayor que +1 que son aristas izquierdas; las aristas derechas y las otras pendientes se manejan con argumentos similares, aunque más complicados, y los vértices verticales son casos especiales. (Las aristas horizontales se manejan de manera implícita con las reglas de tramos, como ya vimos.) En el punto extremo  $(x_{\min}, y_{\min})$  es necesario dibujar un pixel. Al incrementar  $y$ , la coordenada  $x$  del punto en la línea ideal aumentará en  $1/m$ , donde  $m = (y_{\max} - y_{\min})/(x_{\max} - x_{\min})$  es la pendiente de la línea. Este aumento dará como resultado que  $x$  tenga una parte entera y una fraccionaria, la cual se puede expresar como una fracción con denominador de  $y_{\max} - y_{\min}$ . Al iterar en este proceso, las partes fraccionarias provocarán un desbordamiento (*overflow*) y tendremos que incrementar la parte entera. Por ejemplo, si la pendiente es  $2/5$  y  $x_{\min}$  es 3, entonces la secuencia de los valores de  $x$  será 3,  $3\frac{2}{5}$ ,  $3\frac{4}{5}$ ,  $3\frac{2}{5} = 4\frac{1}{5}$ , etcétera. Cuando la parte fraccionaria de  $x$  es cero podemos dibujar el pixel  $(x, y)$  que está sobre la línea, pero si la parte fraccionaria es distinta de cero, hay que redondear hacia arriba para obtener un pixel que se halle estrictamente dentro de la línea. Cuando la parte fraccionaria de  $x$  es mayor que 1, incrementamos  $x$  y restamos 1 a la parte fraccionaria; también hay que moverse un pixel a la derecha. Si al incrementar quedamos exactamente encima de un pixel, hay que dibujarlo pero además reducir en uno la fracción para que sea menor que uno.

Podemos evitar el uso de fracciones si observamos el numerador de la fracción y determinamos que la parte fraccionaria es mayor que uno si el numerador es mayor que el denominador. Esta técnica se implanta en el algoritmo del programa 3.6, donde se usa la variable *incremento* para llevar el control del numerador hasta que haya desbordamiento respecto al denominador; en este caso se decrementa el denominador y se incrementa  $x$ .

#### Programa 3.6

*Discretización de arista izquierda de un polígono.*

```
void rastreo_arista_izquierda (int xmin, int ymin, int xmax, int ymax, int valor);
{
    int x, y, numerador, denominador, incremento;
    x = xmin;
    numerador = xmax - xmin;
    denominador = ymax - ymin;
    incremento = denominador;

    for (y = ymin; y < ymax; y++) {
        escribir_pixel (x, y, valor);
        incremento += numerador;
        if (incremento > denominador) {
```

```

/* Desbordamiento; se redondea hacia arriba al siguiente pixel y se reduce el
   incremento */
x += 1;
incremento -= denominador;
}
}

```

Ahora desarrollaremos un **algoritmo de línea de rastreo** que aprovecha esta coherencia de aristas y, para cada línea de rastreo, lleva el control del conjunto de aristas que interseca y los puntos de intersección en una estructura de datos llamada **tabla de aristas activas** (AET, *active-edge table*). Las aristas en la AET se ordenan de acuerdo con el valor de sus intersecciones en  $x$ , para que se puedan llenar los tramos definidos por pares de valores de intersección (redondeados de manera apropiada), o sea, los extremos de los tramos. Al avanzar a la siguiente línea de rastreo en  $y + 1$  se actualiza la AET. Primero se eliminan las aristas que están en la AET pero que no son intersecadas por esta siguiente línea de rastreo (o sea, aquellas donde  $y_{\max} = y$ ). Segundo, se añaden a la AET las aristas nuevas intersecadas por la siguiente línea de rastreo (es decir, aquellas donde  $y_{\min} = y + 1$ ). Por último, se calculan las nuevas intersecciones de  $x$  usando el algoritmo incremental de aristas previamente presentado, para las aristas que estuvieran en la AET pero que aún no se hubieran completado.

Para que la adición de aristas a la AET sea eficiente, se crea inicialmente una **tabla de aristas** (ET, *edge table*) global, que contiene todas las aristas ordenadas de acuerdo con su coordenada y menor. La ET se construye generalmente usando un ordenamiento de cubeta, con el mismo número de cubetas que líneas de rastreo. En cada cubeta, las aristas se mantienen ordenadas de acuerdo con la coordenada  $x$  creciente del punto extremo inferior. Cada entrada en la ET contiene la coordenada  $y_{\max}$  de la arista, la coordenada  $x$  del punto extremo inferior ( $x_{\min}$ ) y el incremento en  $x$  usado al pasar de una línea de rastreo a la siguiente,  $1/m$ . En la figura 3.18 se muestra cómo se ordenarían las seis aristas del polígono de la figura 3.14, y en la figura 3.19 se presenta la AET para las líneas de rastreo 9 y 10 del mismo polígono (en las implantaciones reales, lo más probable es que se añada una bandera para indicar una arista izquierda o derecha).

Una vez que se ha formado la tabla de aristas, se completan los siguientes pasos de procesamiento para el algoritmo de línea de rastreo:

1. Asignar  $y$  igual a la menor coordenada  $y$  que tenga una entrada en la tabla; es decir, la  $y$  de la primera cubeta no vacía.
  2. Asignar la AET para que esté vacía inicialmente.
  3. Repetir hasta que AET y ET estén vacías:
    - 3.1 Mover de la cubeta  $y$  de la ET a la AET las aristas con  $y_{\min} = y$  (aristas de entrada) y luego ordenar la AET con base en  $x$  (más sencillo porque ET se ordenó previamente).
    - 3.2 Rellenar los valores de los pixeles deseados en la línea de rastreo  $y$  usando pares de coordenadas  $x$  de la AET.

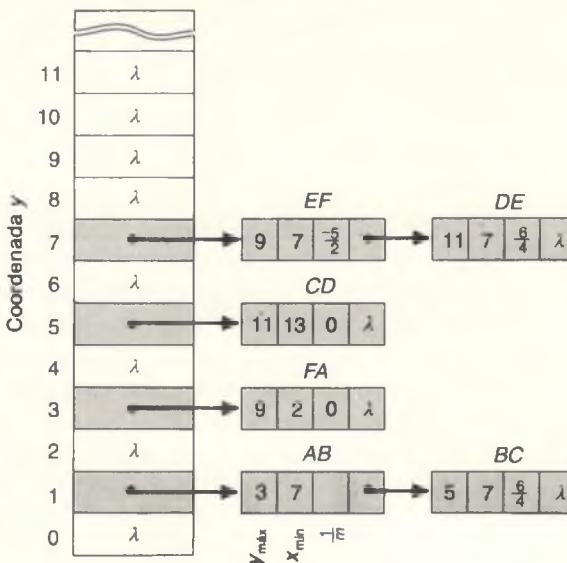
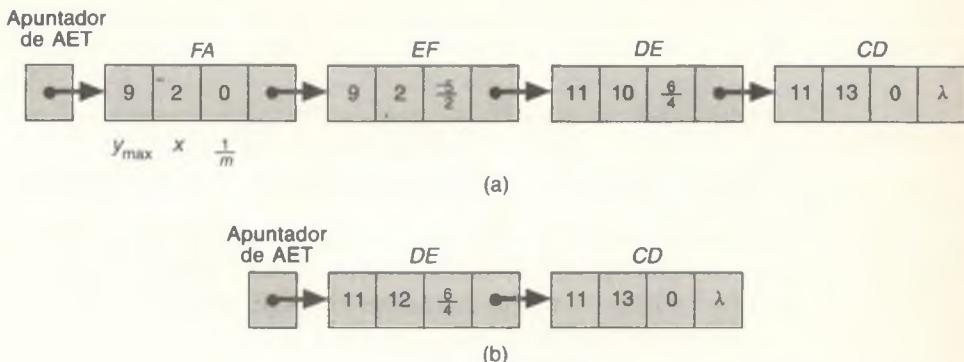


Figura 3.18 Tabla de aristas con ordenamiento de cubeta para el polígono de la figura 3.14.

- 3.3 Eliminar de la AET las entradas donde  $y = y_{\max}$  (aristas que no están comprendidas en la siguiente línea de rastreo).
- 3.4 Incrementar  $y$  en 1 (a la coordenada de la siguiente línea de rastreo).
- 3.5 Actualizar  $x$  para la nueva  $y$  en cada arista no vertical que permanezca en la AET.

Este algoritmo usa la coherencia de aristas para calcular las intersecciones de  $x$  y la coherencia de línea de rastreo (junto con el ordenamiento) para calcular los tramos. Como el ordenamiento opera con un pequeño número de aristas y dado que el reordenamiento del paso 3.1 se aplica a una lista total o parcialmente ordenada, se puede emplear el ordenamiento por inserción o de burbuja. En los capítulos 13 y 14 veremos cómo extender este algoritmo para manejar polígonos múltiples durante la determinación de superficies visibles, incluyendo el caso en que se manejan polígonos transparentes.

Para fines de la discretización, los triángulos y los trapezoides se pueden tratar como casos especiales de polígonos, ya que sólo tienen dos aristas por línea de rastreo (considerando que las aristas horizontales no se discretizan de manera explícita). De hecho, como un polígono arbitrario se puede descomponer en una malla de triángulos que comparten vértices y aristas (véase el Ejer. 3.14), podríamos efectuar la discretización de polígonos generales descomponiendo primero el polígono en una malla de triángulos y luego efectuando la discretización de los triángulos componentes. Esta triangulación es un problema



**Figura 3.19** Tabla de aristas activas para el polígono de la figura 3.14: (a) línea de rastreo 9; (b) línea de rastreo 10. (Observe que la coordenada  $x$  de  $DE$  en (b) ha sido redondeada hacia arriba para la arista izquierda.)

clásico en la geometría computacional [PREP85] y es fácil de efectuar para polígonos convexos; es difícil hacerlo de manera eficiente para polígonos no convexos.

Observe que el cálculo de los tramos es acumulativo. Es decir, si la iteración actual del paso 3.5 del algoritmo de discretización genera varios píxeles que caen en la misma línea de rastreo, hay que actualizar en forma correspondiente los extremos del tramo (el cálculo de tramos para aristas que cruzan y para astillas requiere casos especiales). Podemos calcular todos los tramos en una pasada y llenarlos en otra, o calcular un tramo y llenarlo cuando esté completo. Otra ventaja de usar tramos es que los recortes se pueden efectuar al mismo tiempo que la aritmética del tramo: los tramos pueden recortarse de manera individual en las aristas izquierda y derecha del rectángulo de recorte.

## 3.6 Rellenado con patrones

En las secciones anteriores rellenamos el interior de las primitivas de definición de área de SRGP con un color sólido, pasando el color en el campo **valor** de la función *escribir\_pixel*. Aquí veremos el relleno con patrones, lo cual se lleva a cabo con un poco de control adicional en la parte del algoritmo de discretización encargada de la escritura de los píxeles. En el caso de patrones de mapa de pixeles, este control ocasiona la selección del valor de color de acuerdo con la posición apropiada del arreglo bidimensional en el patrón, como se explica a continuación. Para escribir en forma transparente patrones de mapa de pixeles (*pixmap*, arreglo bidimensional de píxeles), se efectúa una llamada a

*escribir\_pixel* con el color de primer plano para el pixel si existe un 1 en el patrón y se inhibe la llamada a *escribir\_pixel* si se presenta un 0, como se hace en el estilo de línea. Por otra parte, si el patrón de mapa o arreglo bidimensional de bits (*bitmap*) se aplica en modo opaco, los unos y ceros seleccionan el color de primer plano y de fondo, respectivamente.

### 3.6.1 Rellenado con patrones usando discretización

El asunto principal en el relleno con patrones es la relación entre el área del patrón y el área de la primitiva. En otras palabras, debemos decidir dónde está *anclado* el patrón para saber qué pixel del patrón corresponde al pixel actual de la primitiva.

La primera técnica consiste en anclar el patrón en un vértice de un polígono, colocando allí el pixel del extremo izquierdo de la primera fila del patrón. Esta elección permite que el patrón se mueva al mover la primitiva, un efecto visual esperado en patrones con una fuerte organización geométrica, como la malla de rombos que se emplea para sombrear en aplicaciones de bosquejo. Sin embargo, no hay un punto en especial en un polígono que sea obviamente el correcto para servir como ancla relativa, mucho menos puntos de este tipo en primitivas de variación regular como los círculos. Por lo tanto, el programador debe especificar el ancla como un punto sobre la primitiva o dentro de ella. En algunos sistemas, el punto ancla se puede aplicar incluso a un grupo de primitivas.

La segunda técnica, utilizada en SRGP, es considerar toda la pantalla como si estuviera rellenada por el patrón y luego considerar la primitiva como un área hueca o rellenada con bits transparentes que permiten ver el patrón. La posición estándar para este punto ancla absoluto es el origen de la pantalla. Los pixeles de la primitiva se tratan entonces como unos y se lleva a cabo una operación *and* con el patrón. Un efecto secundario de esta técnica es que el patrón no se *pega* a la primitiva si ésta se desplaza ligeramente. En cambio, la primitiva se mueve como si fuera un recorte en un fondo fijo, con patrón, y por ende su apariencia puede cambiar durante el movimiento; en el caso de patrones regulares sin fuerte orientación geométrica, es probable que los usuarios ni siquiera se percaten de este efecto. Además de que sus cálculos son muy eficientes, el anclado absoluto permite la superposición y la adyacencia de primitivas sin que se noten los puntos de unión.

Para aplicar el patrón a la primitiva, se indiza con las coordenadas  $(x, y)$  del pixel actual. Como los patrones se definen con pequeños arreglos bidimensionales de bits o pixeles de  $M$  por  $N$ , usaremos la aritmética modular para repetir el patrón. El pixel  $\text{patrón}[0, 0]$  se considera coincidente con el origen de la pantalla<sup>7</sup> y podemos escribir, por ejemplo, un patrón de bits en modo transparente usando el enunciado

```
if (patrón [x % M, y % N]) escribir_pixel (x, y, valor)
```

<sup>7</sup> En los sistemas de ventanas, el patrón usualmente se ancla en el origen del sistema de coordenadas de la ventana.

Si estamos llenando todo un tramo en modo de escritura de **reemplazo**, (*replace*), podemos copiar al mismo tiempo toda una fila del patrón, suponiendo que se encuentra disponible una versión de bajo nivel de `copyPixel`, para escribir muchos pixeles. Digamos, por ejemplo, que el patrón es una matriz de 8 por 8; por lo tanto, se repite cada tramo de ocho pixeles. Si el punto del extremo izquierdo del tramo está alineado con un byte, es decir, si el valor  $x$  del primer pixel mod 8 es cero, entonces se puede escribir toda la primera fila del patrón usando la función `copyPixel` con un arreglo de 1 por 8; este procedimiento se repite las veces necesarias para llenar el tramo. Si alguno de los extremos del tramo no está alineado con un byte, hay que enmascarar los pixeles fuera del tramo. Los implantadores invierten mucho tiempo para lograr que ciertos casos especiales de los algoritmos de rastreo sean muy eficientes; por ejemplo, realizan pruebas iniciales para eliminar ciclos internos y escriben código en lenguaje ensamblador afinado manualmente para escribir ciclos internos que aprovechen ciertas características especiales del hardware, como memoria caché de instrucciones o instrucciones de ciclo muy eficientes.

### 3.6.2 Rellenado con patrones sin discretización repetida

Hasta ahora hemos analizado el llenado en el contexto de la discretización. Otra técnica es efectuar primero la discretización de la primitiva para obtener un área de trabajo rectangular, y luego escribir cada pixel del mapa de bits en el lugar apropiado del lienzo. Esta *escritura rectangular* en el lienzo no es más que un ciclo **for** anidado en el cual un 1 hace que se escriba el color actual y un 0 que no se escriba nada (para el efecto de transparencia), o que se escriba el color de fondo (para el efecto de opacidad). Este proceso de dos pasos requiere el doble de trabajo que el llenado durante la discretización, por lo que no es conveniente usarlo con primitivas que se detectan y convierten por rastreo sólo una vez. Sin embargo, si es conveniente usarlo con primitivas que de otra manera se discretizarían repetidamente. Esto sucede con caracteres de una familia tipográfica y tamaño determinados, los cuales pueden discretizarse con anticipación a partir de sus versiones huecas. Para caracteres definidos sólo como familias tipográficas de mapa de bits, o para otros objetos, como iconos y símbolos de la aplicación, que se pintan o rastrean como imágenes de mapa de bits, no se emplea la discretización en ninguno de los casos y la única técnica aplicable es la escritura rectangular de sus mapas de bits. La ventaja de un mapa de bits previo a la discretización consiste en que es mucho más rápido escribir cada pixel en la región rectangular sin tener que efectuar aritmética de recorte o de tramo, que discretizar la primitiva desde cero cada vez que se hagan recortes.

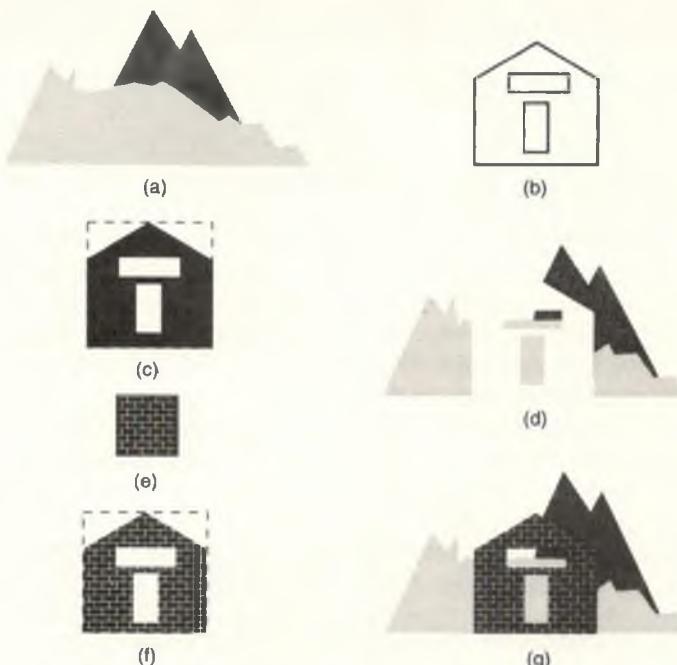
Sin embargo, como tenemos que escribir un mapa de bits rectangular en el lienzo, ¿por qué no usar `copyPixel` para copiar directamente el arreglo bidimensional, en lugar de escribir un pixel a la vez? En el caso de una pantalla de dos niveles en la cual se escribe con el color vigente 1, `copyPixel` funciona muy bien: en el modo transparente se usa el modo de escritura **or**; para el modo opaco, el modo de escritura **replace**. Empero, en las pantallas multinivel

no es posible escribir el mapa de bits directamente con un solo bit por pixel, sino que cada pixel debe convertirse a un valor de color de  $n$  bits antes de escribirlo.

Algunos sistemas tienen una función *copyPixel* más poderosa que puede efectuar copias sujetas a una o más mascarillas de lectura de la fuente o de escritura del destino. Podemos aprovechar estos recursos para el modo transparente (usado para caracteres en SRGP) si podemos especificar el mapa de bits como una mascarailla de escritura de destino y la fuente como un arreglo de color constante (el actual). Los píxeles se escriben entonces en el color actual únicamente si la mascarailla de escritura del mapa de bits tiene unos; la mascarailla de escritura actúa como una región de recorte arbitraria. En cierto sentido, el ciclo **for** anidado explícito que se usa para implantar el rectángulo de escritura en sistemas de  $n$  bits por pixel simula este recurso más poderoso de *CopyPixel con mascarailla de escritura*.

Considere ahora otra variante. Queremos dibujar una letra o alguna otra forma rellena, pero no con un interior sólido, sino con uno que tenga patrón. Por ejemplo, quisiéramos crear una letra “P” gruesa con un patrón gris al 50 por ciento (sombreado el carácter) o un ícono de casa con un patrón de ladrillos de dos tonos. ¿Cómo podemos escribir este objeto en modo opaco sin tener que efectuar la discretización en cada ocasión? El problema es que los *agujeros* dentro de la región, donde hay cero en el mapa de bits, se deben escribir en el color de fondo, mientras que los agujeros fuera de la región (como la cavidad en la “P”) deben escribirse en forma transparente para no afectar la imagen subyacente. En otras palabras, queremos que los ceros en el interior de la forma representen el color de fondo y que los ceros en el exterior, incluyendo cualquier cavidad, correspondan a una mascarailla de escritura utilizada para proteger los píxeles fuera de la forma. Si efectuamos la discretización sobre la marcha, no se presenta el problema de que los ceros representen cosas distintas en regiones diferentes del mapa de bits, ya que nunca vemos los píxeles fuera de las fronteras de la forma.

Para evitar la discretización repetida utilizamos un proceso de cuatro pasos, como se ilustra en la escena de la montaña de la figura 3.20(a). Usando la versión hueca de nuestro ícono que se muestra en la parte (b), el primer paso es crear un *mapa de bits sólido* que se use como mascarailla de escritura/region de recorte, con los píxeles interiores del objeto asignados como unos y los exteriores como ceros; esto se ilustra en la parte (c), donde las porciones blancas representan píxeles de fondo (ceros) y las negras representan unos. Este proceso de discretización sólo se realiza una vez. Como segundo paso, cada vez que se requiere una copia con patrón del objeto, se escribe el mapa de bits sólido en forma transparente con el color de fondo (en modo de reemplazo) sobre el lienzo. Con esto se establece en el color del fondo una región con la forma del objeto, como se ilustra en la parte (d), donde la región en forma de casa se asigna con un fondo blanco dentro de la región de la montaña. El tercer paso es crear una versión con patrón del mapa de bits sólido del objeto aplicando la función *copyPixel* para copiar un rectángulo del patrón —parte (e)— al mapa



**Figura 3.20** Escritura de un objeto con patrón en modo opaco con dos escrituras en modo transparente. (a) Escena de la montaña. (b) Versión hueca del ícono de la casa. (c) Mapa de bits de la versión sólida del ícono de la casa. (d) Borrado de la escena escribiendo con el color de fondo. (e) Patrón de ladrillos. (f) Patrón de ladrillos aplicado al ícono de la casa. (g) Escritura de la pantalla en modo transparente con el ícono de la casa en el patrón deseado.

de bits sólido en modo **and**. Esto enciende algunos píxeles interiores de la forma del objeto, cambiando su valor de 1 a 0 —parte (f)— y se puede considerar como el recorte de una porción arbitrariamente grande con la forma del objeto del patrón. Por último, este mapa de bits se escribe de nuevo en forma transparente en el mismo lugar del lienzo, pero ahora con el color actual del primer plano, como se muestra en la parte (g). Como se hizo en la primera escritura en el lienzo, todos los píxeles fuera de la región de objeto son ceros para proteger los píxeles, mientras que los ceros dentro de la región no afectan el fondo (blanco) previamente escrito; el primer plano (negro) sólo se escribe donde hay unos. Para escribir la casa con un patrón sólido en color rojo ladrillo con líneas de cemento grises, tendríamos que escribir el mapa de bits sólido en gris y el mapa de bits de patrón en rojo; el patrón tendría unos en todas partes excepto por pequeñas franjas de ceros que representarían el cemento. De hecho, hemos reducido la función de escritura de rectángulo que tenía que escribir dos colores sujetos a una máscara de escritura a dos funciones que escriben en forma transparente o de copiado de píxeles con máscara de escritura.

## 3.7 Primitivas gruesas

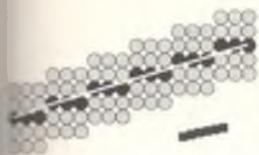
En teoría, las primitivas gruesas se producen trazando la discretización de una primitiva hueca de un solo pixel. Colocamos el centro del pincel de una sección transversal determinada (u otro punto distingible, como la esquina superior izquierda de un pincel rectangular) en cada pixel elegido por el algoritmo de discretización. Podemos considerar una línea de un pixel de grosor como algo dibujado con un pincel del tamaño de un pixel. Sin embargo, esta sencilla descripción encierra varios preguntas complicadas. En primer lugar, ¿qué forma tiene el pincel? Las implantaciones típicas utilizan pinceles circulares y rectangulares. Segundo, ¿cuál es la orientación de un pincel no circular? ¿Se mantiene siempre el pincel rectangular en forma vertical, de manera que el trazo tenga anchura constante, o gira al hacerlo la primitiva, de manera que el eje vertical del pincel esté alineado con la tangente de la primitiva? ¿Cómo deben verse los extremos de una línea gruesa, tanto en forma ideal como en la malla entera? ¿Qué sucede en el vértice de un polígono grueso? ¿Cómo interactúan el estilo de la línea y el estilo del pincel? En esta sección responderemos a las preguntas más sencillas; las otras se analizan en el capítulo 19 de [FOLE90].

Existen cuatro métodos básicos para dibujar primitivas gruesas, los cuales se ilustran en las figuras 3.21 a 3.24. Las primitivas ideales de estas líneas se muestran en forma hueca, en negro sobre blanco; los pixeles generados para definir la primitiva de discretización de un pixel de grosor aparecen en negro; y los pixeles añadidos para formar la primitiva gruesa se presentan en gris. Las versiones en escala reducida muestran la apariencia real de la primitiva gruesa, todavía a baja resolución, con todos los pixeles en negro. El primer método es una aproximación burda que usa más de un pixel por columna (o fila) durante la discretización. En el segundo método se traza la sección transversal del pincel sobre el bosquejo de un pixel de la primitiva. En el tercero se dibujan dos copias de la primitiva a una distancia  $t$  y se llenan los tramos entre estas fronteras interiores y exteriores. El cuarto aproxima todas las primitivas con polilíneas y después usa una línea gruesa para cada segmento de la polilínea.

Veamos brevemente estos métodos y consideremos sus ventajas y desventajas. Todos los métodos producen efectos satisfactorios para la mayoría de los fines, si no es que todos, al menos para la vista en la pantalla. En el caso de la impresión hay que obtener el mayor provecho de la alta resolución, sobre todo porque la velocidad de un algoritmo para impresión no es tan crítica como para la generación de primitivas en línea. Por lo tanto, podemos emplear algoritmos más complejos para producir mejores resultados. Un paquete puede emplear técnicas diferentes para primitivas distintas. Por ejemplo, QuickDraw traza líneas con un pincel rectangular vertical, pero llena los tramos entre fronteras elípticas cofocales.

### 3.7.1 Duplicación de pixeles

Una extensión rápida del ciclo interior de discretización es escribir varios pixeles por cada pixel calculado; este método funciona bastante bien con líneas, donde



**Figura 3.21**  
Línea gruesa dibujada con  
duplicación de columnas.

los pixeles se duplican en las columnas en el caso de líneas con  $-1 < \text{pendiente} < 1$  y en las filas para las demás líneas. Sin embargo, el efecto es que los extremos de la línea siempre son verticales u horizontales, algo no muy agradable en el caso de líneas gruesas, como podrá observarse en la figura 3.21.

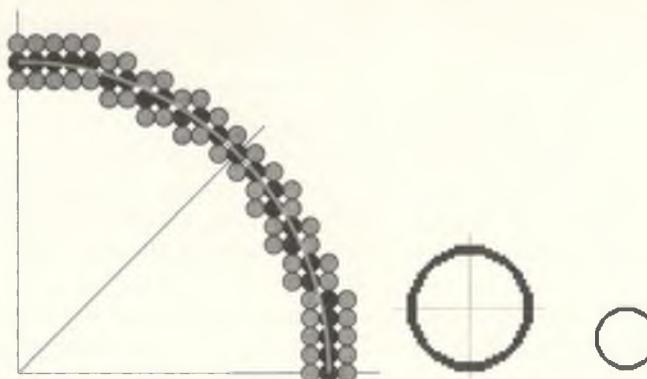
El algoritmo de duplicación de pixeles también produce huecos muy notorios en los lugares donde los segmentos de línea forman ángulos, y omite pixeles cuando hay un desplazamiento de una duplicación horizontal a vertical como función de la pendiente. Esta segunda anomalía se presenta como una delgadez extrema en los arcos elípticos, en las fronteras entre octantes (véase la Fig. 3.22).

Además, el grosor de las líneas horizontales o verticales difiere del de las líneas en ángulo, para las cuales el *grosor* de la primitiva se define como la distancia entre las fronteras de la primitiva perpendiculares a su tangente. De esta manera, si el parámetro de grosor es  $t$ , una línea horizontal o vertical tiene grosor  $t$ , pero una dibujada a  $45^\circ$  tiene un grosor medio de  $t/\sqrt{2}$ . Esta diferencia es otro resultado producido por el hecho de tener menos pixeles en la línea angular, como señalamos por primera vez en la sección 3.2.3; reduce el contraste de brillo con respecto a las líneas horizontales o verticales del mismo grosor. Otro problema con la duplicación de pixeles es el problema genérico de las anchuras pares: no podemos centrar la columna o fila duplicada con respecto al pixel seleccionado, de manera que hay que elegir un lado de la primitiva para que tenga un pixel *adicional*. En conjunto, la duplicación de pixeles es una aproximación burda pero eficiente que genera mejores resultados con primitivas que no son muy gruesas.

### 3.7.2 El pincel móvil

La elección de un pincel rectangular cuyo centro o vértice viaje sobre el trazo de un pixel de la primitiva produce buenos resultados con líneas; genera la línea que se presenta en la figura 3.23. Observe que esta línea es similar a la que se produjo con la duplicación de pixeles, pero sus puntos extremos son más gruesos. Como ocurre en la duplicación de pixeles, el pincel permanece alineado verticalmente, por lo que el grosor percibido de la primitiva varía como función del ángulo de la primitiva, pero en forma opuesta: el grosor es menor en los segmentos horizontales y mayor en los que tienen pendiente de  $\pm 1$ . Por ejemplo, el grosor de un arco elíptico varía en toda su trayectoria, siendo del grosor especificado cuando la tangente es casi horizontal o vertical y  $\sqrt{2}$  veces más grueso alrededor de los  $\pm 45^\circ$  (véase la Fig. 3.24). Este problema se podría eliminar si el cuadrado girara de acuerdo con la trayectoria, pero es mejor utilizar una sección transversal circular para que el grosor sea independiente del ángulo.

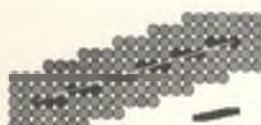
Veamos ahora cómo implantar el algoritmo de pincel móvil para el sencillo caso de una sección transversal vertical rectangular o circular. La solución más sencilla es copiar los pixeles de la sección transversal sólida o con patrón que se requiere (también llamada **huella**), para que su centro o vértice se encuentre en el pixel elegido; en el caso de una huella circular y un patrón dibujado en modo



**Figura 3.22** Círculo grueso dibujado con duplicación de columnas.

opaco, debemos además enmascarar los bits fuera de la región circular, una tarea que no es sencilla si la función `copyPixel` de bajo nivel no tiene una máscara de escritura para la región destino. La solución burda de copiado de pixeles escribe los pixeles más de una vez, ya que las huellas del pincel se sobreponen en pixeles adyacentes. Una mejor técnica, que también maneja el problema de la sección transversal circular, es usar los tramos de la huella para calcular los tramos de las huellas sucesivas en pixeles adyacentes. Como en el llenado de primitivas de definición de áreas, esta combinación de tramos en una línea de barrido no es más que la unión o fusión de segmentos de línea, lo que implica llevar el control de la  $x$  mínima y máxima del tramo acumulado en cada línea de la trama.

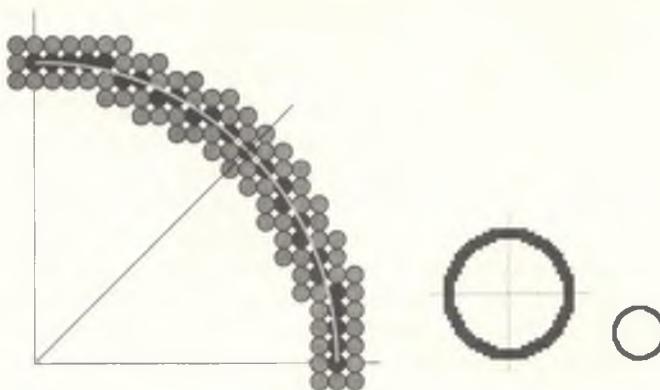
En los capítulos 3 y 19 de [FOLE90] se analizan otros métodos para presentar primitivas gruesas, como el relleno de áreas entre las fronteras interiores y exteriores construidas a una distancia  $t/2$  a cada lado de una trayectoria de un solo pixel.



**Figura 3.23**  
Línea gruesa dibujada  
trazando con un pincel  
rectangular.

### 3.8 Recortes en un mundo de barrido de trama

Como señalamos en la introducción a este capítulo, es indispensable que los recortes y la discretización se hagan con la mayor rapidez posible, para que se presente al usuario una actualización rápida como resultado de los cambios en el modelo de aplicación. Los recortes se pueden efectuar en forma analítica, sobre la marcha durante la discretización, o como parte del copiado de pixeles desde un lienzo que almacena primitivas no recortadas hasta el lienzo destino usando el rectángulo de recorte deseado. Esta tercera técnica es útil cuando



**Figura 3.24** Círculo grueso dibujado con el trazo de un pincel rectangular.

puede generarse por adelantado un lienzo de gran tamaño, y el usuario puede examinar pedazos del lienzo durante un periodo considerable cambiando la perspectiva del rectángulo de recorte, sin actualizar el contenido del lienzo.

La combinación del recorte y la discretización, denominada *tijereteo*, es sencilla si hay primitivas rellenas o gruesas que formen parte de la aritmética de tramos: sólo hay que recortar los extremos y no es necesario examinar pixeles interiores. El tijereteo nos muestra otra ventaja de la coherencia de tramos. Además, si una primitiva hueca no es mucho mayor que el rectángulo de recorte, serán pocos los pixeles, relativamente hablando, que caigan fuera de la región de recorte. En este caso, puede ser más rápido generar cada pixel y recortarlo (es decir, escribirlo de manera selectiva) que realizar de antemano un recorte analítico. En particular, aunque la prueba de los límites corresponde al ciclo interior, se evita la costosa escritura en memoria para los pixeles exteriores, y tanto los cálculos incrementales como las pruebas se pueden efectuar totalmente en una memoria rápida, como la memoria caché de instrucciones de la UCP o la memoria de microcódigo de un controlador de pantalla.

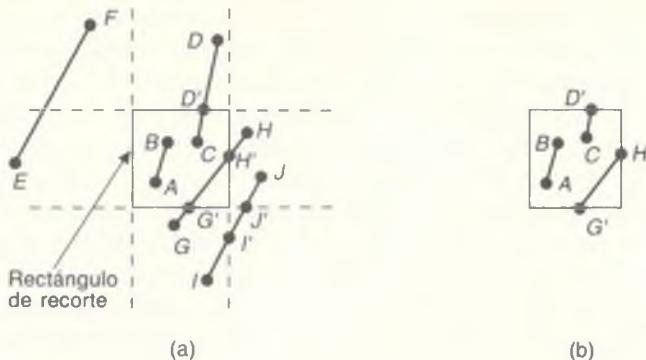
Hay otros trucos que pueden ser de utilidad. Por ejemplo, uno se puede centrar en la intersección de una línea con una arista de corte ejecutando el algoritmo estándar de punto medio para la discretización en cada  $i$ -ésimo pixel y evaluando el pixel elegido contra los límites del rectángulo hasta hallar el primer pixel dentro de la región. Después el algoritmo retrocede, encuentra el primer pixel interior y lleva a cabo la discretización normal. El último pixel interior también podría determinarse en forma similar, o se podría evaluar cada pixel como parte del ciclo de discretización y detener el proceso en cuanto fallara la prueba. La evaluación de cada octavo bit funciona bastante bien, ya que es un buen arreglo entre tener demasiadas pruebas y demasiados pixeles que respaldar.

En los paquetes gráficos que operan en punto flotante, lo mejor es efectuar el recorte analítico en el sistema de coordenadas de punto flotante y luego discretizar las primitivas recortadas, teniendo cuidado de asignar correctamente los valores iniciales de las variables de decisión, como lo hicimos con las líneas en la sección 3.2.3. En los paquetes gráficos enteros como SRGP, existe la opción de recortar primero y luego hacer la discretización o realizar los recortes durante el proceso. Es bastante sencillo llevar a cabo los recortes analíticos de líneas y polígonos, por lo que estas primitivas suelen recortarse antes de la discretización; en el caso de las otras primitivas, es más rápido recortarlas durante el proceso. Así mismo, es bastante común que un paquete gráfico de punto flotante lleve a cabo los recortes en su sistema de coordenadas y luego llame a software de discretización de bajo nivel para que genere verdaderamente las primitivas recortadas; este software gráfico entero puede entonces efectuar un recorte de barrido adicional de acuerdo con fronteras de ventanas rectangulares (o incluso arbitrarias). El recorte analítico de primitivas es útil para los paquetes gráficos enteros e indispensable para los paquetes gráficos bidimensionales y tridimensionales de punto flotante. En este capítulo analizaremos los algoritmos básicos de recorte analítico.

### 3.9 Recorte de líneas

En esta sección se analiza el recorte analítico de líneas con respecto a rectángulos; los algoritmos para recortar otras primitivas se analizan en secciones subsiguientes. Aunque existen algoritmos especializados para el recorte de rectángulos y polígonos, es importante señalar que las primitivas de SRGP construidas con líneas (es decir, polilíneas, rectángulos no rellenos y polígonos) se pueden recortar con la aplicación repetida del recortador de líneas. Además, los círculos se pueden aproximar linealmente por trozos con una secuencia de líneas muy cortas, de manera que las fronteras se pueden tratar como una polilínea o un polígono para el recorte y la discretización. Las cónicas se representan en algunos sistemas como razones de polinomios paramétricos (véase el Cap. 9), una representación que también se presta para una aproximación lineal e incremental por trozos apropiada para un algoritmo de recorte de líneas. El recorte de un rectángulo con respecto a otro genera a lo sumo otro rectángulo. El recorte de un polígono convexo con respecto a un rectángulo produce también a lo sumo un polígono convexo, pero el recorte de un polígono cóncavo puede producir más de un polígono cóncavo. El recorte de un círculo con respecto a un rectángulo genera un máximo de cuatro arcos.

Las líneas que intersecan una región de recorte rectangular (o cualquier polígono convexo) siempre se recortan a un solo segmento de línea; las líneas que yacen en la frontera del rectángulo de recorte se consideran interiores y por lo tanto se presentan. En la figura 3.25 se muestran varios ejemplos de líneas recortadas.



**Figura 3.25** Casos de recorte de líneas.

### 3.9.1 Recorte de puntos extremos

Antes de analizar el recorte de líneas, veamos el problema más sencillo de recortar puntos individuales. Si las fronteras de la coordenada  $x$  del rectángulo de recorte están en  $x_{\min}$  y  $x_{\max}$  y las fronteras de la coordenada  $y$  están en  $y_{\min}$  y  $y_{\max}$ , deben satisfacerse cuatro desigualdades para que un punto en  $(x, y)$  se encuentre en el interior del rectángulo de recorte:

$$x_{\min} \leq x \leq x_{\max}, y_{\min} \leq y \leq y_{\max}$$

Si no se cumple alguna de las cuatro desigualdades, el punto está fuera del rectángulo de recorte.

### 3.9.2 Recorte de líneas mediante la resolución de ecuaciones simultáneas

Para recortar una línea sólo hay que considerar sus puntos extremos, no su infinita cantidad de puntos interiores. Si los dos puntos extremos de una línea caen dentro del rectángulo de recorte (p. ej.,  $AB$  en la figura 3.25), toda la línea cae dentro del rectángulo y se puede aceptar trivialmente. Si un punto extremo se halla dentro del rectángulo y el otro está fuera (p. ej.,  $CD$  en la figura), la línea interseca el rectángulo de recorte y debemos calcular el punto de intersección. Si ambos puntos extremos caen fuera del rectángulo de recorte, es posible que la línea interseque (o no) el rectángulo ( $EF$ ,  $GH$  e  $IJ$  en la figura), por lo que tenemos que realizar cálculos adicionales para determinar si existen estas intersecciones y, de ser así, dónde ocurren.

El enfoque burdo para recortar una línea que no puede aceptarse trivialmente consiste en intersecar la línea con las cuatro aristas del rectángulo de recorte para ver si alguno de los puntos de intersección está en esas aristas; de ser

así, la línea cruza el rectángulo de recorte y se encuentra parcialmente dentro de él. Por lo tanto, para cada línea y cada arista del rectángulo de recorte se toman las dos líneas matemáticamente infinitas que las contienen, y se intersecan. Despues se determina si el punto de intersección es *interior*, o sea, si está dentro del rectángulo de recorte y de la línea; de ser así, existe una intersección con el rectángulo de recorte. En la figura 3.25, los puntos de intersección  $G'$  y  $H'$  son interiores, pero no así  $I'$  y  $J'$ .

Al utilizar este método tenemos que resolver dos ecuaciones simultáneas usando multiplicación y división para cada par  $\langle$ arista, línea $\rangle$ . Podríamos usar la fórmula de intersección de pendiente de líneas que aprendimos en nuestros cursos de geometría analítica, aunque describe líneas infinitas, mientras que en los gráficos y los recortes tratamos líneas finitas (llamadas *segmentos de línea* en matemáticas). Además, la fórmula de intersección de pendiente no trata con líneas verticales, un problema bastante grave ya que usamos un rectángulo de recorte vertical. Una formulación paramétrica de ambas líneas resuelve los dos problemas:

$$x = x_0 + t(x_1 - x_0), \quad y = y_0 + t(y_1 - y_0).$$

Estas ecuaciones describen  $(x, y)$  en el segmento de línea dirigido de  $(x_0, y_0)$  a  $(x_1, y_1)$  para el parámetro  $t$  en el intervalo  $[0, 1]$ , como se puede confirmar con una simple sustitución de  $t$ . Se pueden resolver dos conjuntos de ecuaciones simultáneas de esta forma paramétrica usando los parámetros  $t_{\text{arista}}$  para la arista y  $t_{\text{línea}}$  para el segmento de línea. Los valores de  $t_{\text{arista}}$  y  $t_{\text{línea}}$  se pueden evaluar para ver si ambos están en  $[0, 1]$ ; de ser así, el punto de intersección cae en ambos segmentos y es una verdadera intersección del rectángulo de recorte. Además, hay que probar el caso especial de una línea paralela a la arista del rectángulo de recorte antes de intentar resolver las ecuaciones simultáneas. En conjunto, este enfoque burdo implica gran cantidad de cálculos y pruebas, por lo que no es eficiente.

### 3.9.3 Algoritmo de recorte de líneas de Cohen-Sutherland

El algoritmo de Cohen-Sutherland más eficiente lleva a cabo pruebas iniciales con una línea para determinar si es posible evitar el cálculo de la intersección. En primer lugar se revisan los pares de puntos extremos para determinar la posibilidad de su aceptación trivial. Si la línea no puede aceptarse trivialmente, se efectúan *revisiones de región*. Por ejemplo, dos comparaciones sencillas con  $x$  muestran que los puntos extremos de  $EF$  en la figura 3.25 tienen una coordenada  $x$  menor que  $x_{\min}$  y por ende se hallan en la región ubicada a la izquierda del rectángulo de recorte (es decir, en el medio plano exterior definido por la arista izquierda); por consiguiente, el segmento de línea  $EF$  se puede **rechazar trivialmente** y no hay que recortarlo ni presentarlo. En forma similar, podemos rechazar trivialmente las líneas cuyos dos puntos extremos se encuentren en las regiones ubicadas a la derecha de  $x_{\max}$ , debajo de  $y_{\min}$  y encima de  $y_{\max}$ .

Si el segmento de línea no puede aceptarse o rechazarse trivialmente, se divide en dos segmentos en una arista de recorte, de manera que uno de ellos pueda rechazarse trivialmente. De esta manera, se recorta de manera iterativa un segmento, evaluando la posibilidad de aceptación o rechazo trivial, y luego se subdivide si ninguna de las pruebas tiene éxito, hasta que la porción que quede esté completamente dentro del rectángulo de recorte o pueda rechazarse trivialmente. El algoritmo es muy eficiente en dos casos comunes. En el primero, donde existe un rectángulo de recorte grande que encierra toda o gran parte del área de presentación, la mayoría de las primitivas se pueden aceptar en forma trivial. En el segundo caso existe un rectángulo de recorte pequeño y casi todas las primitivas se pueden rechazar trivialmente. Este segundo caso se presenta en el método estándar de correlación de selección, donde se emplea una pequeña ventana que rodea al cursor, llamada **ventana de selección**, para recortar primitivas y determinar cuáles están dentro de una pequeña área (rectangular) que rodea al **punto de selección** del cursor (véase la Sec. 7.11.2).

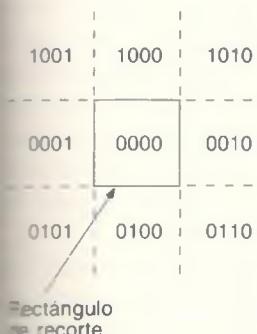


Figura 3.26  
Códigos de región.

Para llevar a cabo las pruebas de aceptación y rechazo trivial, extendemos las aristas del rectángulo de recorte para dividir el plano del rectángulo en nueve regiones (véase la Fig. 3.26). A cada región se le asigna un código de cuatro bits, determinado por la posición de la región con respecto a los medios planos exteriores de las aristas del rectángulo de recorte. Cada bit en el código de región se asigna ya sea a 1 (verdadero) o a 0 (falso); los cuatro bits en el código corresponden a las siguientes condiciones:

Primer bit	fuera del medio plano de la arista superior, encima de la arista superior	$y > y_{\max}$
Segundo bit	fuera del medio plano de la arista inferior, debajo de la arista inferior	$y < y_{\min}$
Tercer bit	fuera del medio plano de la arista derecha, a la derecha de la arista derecha	$x > x_{\max}$
Cuarto bit	fuera del medio plano de la arista izquierda, a la izquierda de la arista izquierda	$x < x_{\min}$

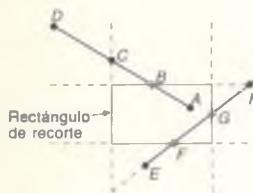
Como la región que está encima y a la izquierda del rectángulo de recorte, por ejemplo, se encuentra fuera del medio plano externo de las aristas superior e izquierda, se le asigna un código de 1001. Una manera muy eficaz de calcular el código de región es observando que el bit 1 es el bit de signo de ( $y_{\max} - y$ ); el bit 2 es el de ( $y - y_{\min}$ ); el bit 3 corresponde al signo de ( $x_{\max} - x$ ); y el bit 4 al de ( $x - x_{\min}$ ). Cada punto extremo del segmento de línea se asigna entonces al código de la región donde se encuentra. Así, podemos utilizar estos códigos de punto extremo para determinar si el segmento de línea cae totalmente dentro del rectángulo de recorte o en el medio plano exterior de una arista. Si los dos códigos de cuatro bits de los puntos extremos son cero, la línea cae por completo dentro del rectángulo de recorte. Sin embargo, si los dos puntos extremos están en el medio plano exterior de determinada arista, como *EF* en la figura 3.25, los códigos de ambos puntos extremos tienen un bit igual a 1 que indica que el punto cae en el medio plano exterior de la arista. En el caso de *EF*, los códigos de región son 0001 y 1001 respectivamente, indicando con el cuarto bit que el segmento de línea se halla en el medio plano exterior de la arista izquierda. Por lo tanto, si la operación lógica **and** aplicada a los códigos de los puntos extremos no es cero, la línea se puede rechazar trivialmente.

Si no es posible aceptar o rechazar una línea en forma trivial, hay que dividirla en dos segmentos tales que uno o ambos se puedan descartar. Esta división se lleva a cabo usando una arista que cruce la línea, para cortar ésta en dos segmentos: la sección en el medio plano exterior de la arista se descarta. Podemos elegir cualquier orden para evaluar las aristas, pero, por supuesto, siempre hay que emplear el mismo orden en el algoritmo; nosotros usaremos el orden arriba-abajo, derecha-izquierda, del código de región. Una propiedad clave del código de región es que los bits iguales a 1 en un código de región distinto de cero corresponden a la aristas cruzadas: si un punto extremo está en el medio plano exterior de una arista y el segmento de línea no aprueba la evaluación de rechazo trivial, entonces el otro punto deberá estar en el medio plano interior de la arista y el segmento de línea debe cruzar la arista. Por ende, el algoritmo siempre elige un punto que se encuentre en el exterior y luego emplea un bit de código de región igual a 1 para determinar una arista de corte; la arista elegida es la primera en el orden arriba-abajo, derecha-izquierda; es decir, se trata del bit 1 que se encuentre más a la izquierda en el código de región.

El algoritmo funciona de la siguiente manera. Se calculan los códigos de región de los dos puntos extremos y se revisa si pueden rechazarse o aceptarse trivialmente. Si fracasan ambas pruebas, encontramos un punto extremo que caiga fuera (al menos habrá uno) y se prueba el código de región para encontrar la arista cruzada y determinar el punto de intersección correspondiente. Después podemos recortar el segmento de línea desde el punto extremo exterior hasta el punto de intersección, reemplazando el punto extremo exterior con el punto de intersección y calculando el código de región de este nuevo punto para la siguiente iteración.

Considere, por ejemplo, el segmento de línea *AD* de la figura 3.27. El punto *A* tiene el código de región 0000 y el punto *D*, el código 1001. La línea no se puede aceptar ni rechazar trivialmente. Por lo tanto, el algoritmo elige *D* como punto externo cuyo código indica que la línea cruza las aristas superior e izquierda. De acuerdo con nuestro orden de evaluación, usamos primero la arista superior para recortar *AD* a *AB* y calculamos el código de región de *B* como 0000. En la siguiente iteración se aplican a *AB* las pruebas de aceptación/rechazo trivial, se acepta trivialmente y se presenta.

La línea *EI* requiere varias iteraciones: el primer punto extremo, *E*, tiene código de región 0100, de manera que el algoritmo lo elige como punto exterior y evalúa el código para determinar si la primera arista de corte de la línea es la inferior, donde *EI* se recorta a *FI*. En la segunda iteración, *FI* no se puede aceptar ni rechazar trivialmente. El código de región del primer punto extremo, *F*, es 0000, por lo que el algoritmo escoge el punto exterior *I* con código de región 1010. La primera arista para el recorte es entonces la superior, que produce *FH*. El código de región de *H* es 0010 y la tercera iteración produce un recorte en la arista derecha para obtener *FG*. Esto se acepta trivialmente en la cuarta y última iteración y se presenta. Si hubiéramos elegido *I* como punto inicial, habríamos obtenido una secuencia de recortes diferente. Con base en su código de región, primero habríamos recortado en la arista superior, luego en la derecha y por último en la inferior.



**Figura 3.27**  
Ilustración del recorte de  
líneas de  
Cohen-Sutherland.

En el código del programa 3.7, para representar el código de región se utiliza una estructura de C con campos como miembros, ya que esta representación es más natural que un arreglo de cuatro enteros. Por cuestiones de modularidad se emplea una función externa para calcular el código de región; sin embargo, para mejorar el rendimiento colocaríamos este código en línea.

**Programa 3.7**

*Algoritmo de recorte de  
líneas de  
Cohen-Sutherland.*

```
typedef struct {
    unsigned todas;
    unsigned izquierda: 1;
    unsigned derecha: 1;
    unsigned inferior: 1;
    unsigned superior: 1;
} código_region;

void recorte_y_dibujo_de_lineas_Cohen_Sutherland (float x0, float y0, float x1,
    float y1, float xmin, float xmáx, float ymin, float ymáx, int valor)
/* Algoritmo de recorte de líneas de Cohen-Sutherland para la línea P0 = (x0, y0) a P1 = (x1,
y1) y */
/* rectángulo de recorte con diagonal de (xmin, ymin) a (xmáx, ymáx) */
{
    boolean aceptar, fin;
    código_region código_region0, código_region1, código_region_salida,
        cálculo_código_region( );
    float x, y;

    aceptar = FALSE;
    fin = FALSE;
    código_region0 = cálculo_código_region (x0, y0, xmin, xmáx, ymin, ymáx);
    código_region1 = cálculo_código_region (x1, y1, xmin, xmáx, ymin, ymáx);
    do {
        if (código_region0.todas == 0 && código_region1.todas == 0) {
            aceptar = TRUE;
            fin = TRUE;
        } else if (código_region0. todas & código_region1.todas != 0)
            fin = TRUE; /* La intersección lógica es verdadera, por lo que se rechaza
trivialmente y termina */
        else {
            if (código_region0.todas != 0)
                código_region_salida = código_region0;
            else
                código_region_salida = código_region1;
            if (código_region_salida.superior) { /* Dividir la línea en la parte superior del
rectángulo de recorte */
                x = x0 + (x1 - x0) * (ymáx - y0) / (y1 - y0);
                y = ymáx;
            } else if ((código_region_salida.inferior)) { /* Dividir la línea en la parte inferior del
rectángulo de recorte */
                x = x0 + (x1 - x0) * (ymin - y0) / (y1 - y0);
                y = ymin;
            }
        }
    } while (!fin);
}
```

```

    } else if ((código_region_salida.derecha) { /* Dividir la línea en la parte derecha
        del rectángulo de recorte */
        y = y0 + (y1 - y0) * (xmáx - x0) / (x1 - x0);
        x = xmáx;
    } else if ((código_region_salida.izquierda) { /* Dividir la linea en la parte izquierda
        del rectángulo de recorte */
        y = y0 + (y1 - y0) * (xmin - x0) / (x1 - x0);
        x = xmin;
    }
    if (código_region_salida.todas == código_region0.todas) {
        x0 = x;
        y0 = y;
        código_salida0 = cálculo_código_salida (x0, y0, xmin, xmáx, ymin, ymax);
    } else {
        x1 = x;
        y1 = y;
        código_salida1 = cálculo_código_salida (x1, y1, xmin, xmáx, ymin, ymax);
    }
} /* Subdividir */
} while (!fin);
if (aceptar)
    linea_punto_medio_real (x0, y0, x1, y1, valor) /* Versión para coordenadas reales */
}

código_region cálculo_código_region (float x, float y, float xmin, float xmáx, float ymin,
                                         float ymax)
{
    código_region código;
    código.todas = 0;
    if (y > ymax){
        código.superior = 1;
        código.todas += código.superior;
    } else if (y < ymin){
        código.inferior = 1;
        código.todas += código.inferior;
    }
    if (x > xmáx){
        código.derecha = 1;
        código.todas += código.derecha;
    } else if (x < xmin){
        código.izquierda = 1;
        código.todas += código.izquierda;
    }
    return código;
}

```

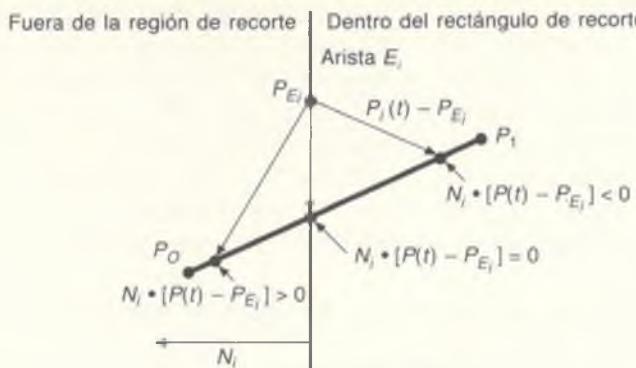
Podemos mejorar ligeramente la eficiencia del algoritmo si no volvemos a calcular las pendientes (véase el Ejer. 3.22). Sin embargo, incluso con esta mejora, el algoritmo no es el más eficiente. Como las pruebas y los recortes se

llevan a cabo siguiendo un orden fijo, en ocasiones el algoritmo efectuará recortes innecesarios. Esta situación se presenta cuando la intersección con la arista del rectángulo es una **intersección externa**, o sea, cuando no cae en la frontera del rectángulo de recorte (p. ej., el punto *H* en *EI* de la figura 3.27). Por otra parte, el algoritmo de Nicholl, Lee y Nicholl [NICH87] evita el cálculo de las intersecciones externas al subdividir el plano en más regiones; este algoritmo se analiza en el capítulo 19 de [FOLE90]. Una ventaja del algoritmo de Cohen-Sutherland, más sencillo, es que se obtiene en forma directa una extensión al volumen de vista ortográfica tridimensional, como se verá en la sección 6.6.3.

### 3.9.4 Algoritmo paramétrico de recorte de líneas

El algoritmo de Cohen-Sutherland es quizás el más común para el recorte de líneas, por ser el más viejo y el de mayor difusión. En 1978, Cyrus y Beck publicaron un algoritmo que utiliza un enfoque totalmente distinto y por lo general más eficaz para el recorte de líneas [CYRU78]. La técnica de Cyrus-Beck se puede utilizar para recortar una línea bidimensional con respecto a un rectángulo o un polígono convexo arbitrario en el plano, o bien una línea tridimensional con respecto a un poliedro convexo arbitrario en el espacio tridimensional. Liang y Barsky desarrollaron en forma independiente un algoritmo paramétrico más eficaz de recorte de líneas que es muy rápido en los casos especiales de regiones de corte bidimensionales y tridimensionales verticales [LIAN84]. Además de aprovechar estas fronteras de recorte sencillas, introdujeron pruebas más eficientes para el rechazo trivial que funcionan para regiones de recorte generales. Aquí seguiremos el desarrollo original de Cyrus-Beck para presentar los recortes paramétricos. Sin embargo, como sólo nos interesan los rectángulos de recorte verticales, reduciremos la formulación de Cyrus-Beck al caso más eficiente de Liang-Barsky al terminar el desarrollo.

Recuerde que el algoritmo de Cohen-Sutherland calcula, para las líneas que no pueden aceptarse o rechazarse trivialmente, la intersección (*x*, *y*) de un segmento de línea con una arista de corte sustituyendo el valor conocido de *x* o *y* por la arista de corte vertical u horizontal, respectivamente. Sin embargo, el algoritmo paramétrico de línea encuentra el valor del parámetro *t* en la representación paramétrica del segmento de línea para el punto donde el segmento interseca la línea infinita donde se encuentra la arista de corte. Como todas las aristas de corte generalmente son intersecadas por líneas, se calculan cuatro valores de *t*. Despues se emplea una serie de comparaciones sencillas para determinar cuál (si acaso hay alguno) de los cuatro valores de *t* corresponde a una intersección real. Sólo entonces se calculan los valores (*x*, *y*) para una o dos de las intersecciones reales. En términos generales, este método ahorra tiempo comparado con el algoritmo de cálculo de intersecciones de Cohen-Sutherland, ya que evita los ciclos repetitivos necesarios para recortar con respecto a varias aristas del rectángulo de recorte. Además, los cálculos en el espacio unidimensional de parámetros son más sencillos que los que se efectúan en el espacio tridimensional de coordenadas. Liang y Barsky han mejorado el algoritmo de



**Figura 3.28** Productos punto para tres puntos fuera, dentro y en la frontera de la región de recorte.

Cyrus-Beck al examinar cada valor de  $t$  conforme se genera, con lo cual se rechazan algunos segmentos de línea antes de haber calculado los cuatro valores de  $t$ .

El algoritmo de Cyrus-Beck se basa en la siguiente formulación de la intersección entre dos líneas. En la figura 3.28 se muestra una arista  $E_i$  del rectángulo de recorte y la normal exterior de la arista,  $N_i$  (es decir, exterior con respecto al rectángulo de recorte),<sup>8</sup> así como el segmento de línea de  $P_0$  a  $P_1$  que debe recortarse en la arista. Para hallar el punto de intersección hay que extender la arista o el segmento de línea.

Como antes, esta línea se representa en forma paramétrica como

$$P(t) = P_0 + t(P_1 - P_0),$$

donde  $t = 0$  en  $P_0$  y  $t = 1$  en  $P_1$ . Elija ahora un punto arbitrario  $P_{E_i}$  en la arista  $E_i$  y considere tres vectores  $P(t) - P_{E_i}$  de  $P_{E_i}$  a tres puntos designados en la línea de  $P_0$  a  $P_1$ : el punto de intersección que se determinará, un punto extremo de la línea en el medio plano interior de la arista y un punto extremo en el medio plano exterior de la arista. Podemos determinar en qué región se encuentra un punto observando el valor del producto punto  $N_i \cdot [P(t) - P_{E_i}]$ . Este valor es negativo para un punto en el medio plano interior, cero para un punto en la línea que contiene la arista y positivo para un punto en el medio plano exterior. Las definiciones de medio plano interior y exterior de una arista corresponden a una numeración, en sentido contrario al de las manecillas del reloj, de las aristas de la región de recorte, una convención que usaremos en este libro. Ahora podemos resolver el valor de  $t$  en la intersección de  $P_0P_1$  con la arista:

$$N_i \cdot [P(t) - P_{E_i}] = 0.$$

<sup>8</sup> Cyrus y Beck usan normales interiores, pero nosotros preferimos usar las exteriores por consistencia con las normales a planos en tres dimensiones, las cuales son exteriores. Por lo tanto, nuestra formulación sólo difiere en lo referente a la evaluación del signo.

Primero se sustituye  $P(t)$ :

$$N_i \cdot [P_0 + t(P_1 - P_0) - P_E] = 0.$$

Después se agrupan términos y se distribuye el producto punto:

$$N_i \cdot [P_0 - P_E] + N_i \cdot t[P_1 - P_0] = 0.$$

Sea  $D = (P_1 - P_0)$  el vector de  $P_0$  a  $P_1$  y resuelva para  $t$ :

$$t = \frac{N_i \cdot [P_0 - P_E]}{-N_i \cdot D} \quad (3.1)$$

Observe que con esto se obtiene un valor válido de  $t$  únicamente si el denominador de la expresión es distinto de cero. Para que esta situación sea verdadera, el algoritmo debe revisar lo siguiente:

$N_i \neq 0$  (es decir, la normal no debe ser 0; esto sólo puede ocurrir por error),

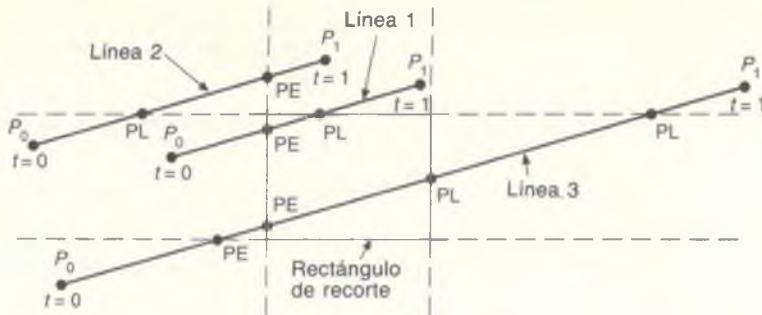
$D \neq 0$  (es decir,  $P_1 \neq P_0$ ),

$N_i \cdot D \neq 0$  (es decir, la arista  $E_i$  y la línea de  $P_0$  a  $P_1$  no son paralelas. Si lo fueran, no existiría una intersección para la arista, por lo cual el algoritmo pasa al siguiente caso).

Podemos usar la ecuación (3.1) para hallar las intersecciones entre  $P_0P_1$  y cada una de las aristas del rectángulo de recorte. Para este cálculo se determina la normal y un punto  $P_E$  arbitrario, —digamos, un punto extremo de la arista— de cada arista de recorte, para luego emplear esos valores en todos los segmentos de línea. Dados los cuatro valores de  $t$  para un segmento de línea, el siguiente paso es determinar cuáles (si los hay) de los valores corresponden a intersecciones interiores del segmento de línea con las aristas del rectángulo de recorte. Como primer paso, se puede descartar cualquier valor de  $t$  fuera del intervalo  $[0, 1]$ , ya que estará fuera de  $P_0P_1$ . Después hay que determinar si la intersección se halla en una frontera de recorte.

Podríamos intentar el ordenamiento de los valores restantes de  $t$  y elegir los valores intermedios como puntos de intersección, como se sugiere en la figura 3.29 para el caso de la línea 1. Sin embargo, ¿cómo distinguimos este caso del de la línea 2, donde ninguna porción del segmento de línea cae dentro del rectángulo de recorte y los valores intermedios de  $t$  corresponden a puntos que no están en la frontera de recorte? Además, ¿cuáles de las cuatro intersecciones de la línea tres son las que están en la frontera de recorte?

Las intersecciones en la figura 3.29 se caracterizan como *potencialmente entrantes* (PE) o *potencialmente salientes* (PL) del rectángulo de recorte, de la



**Figura 3.29** Líneas diagonales con respecto al rectángulo de recorte.

siguiente manera: si el movimiento de  $P_0$  a  $P_1$  hace que crucemos una arista específica para ingresar al medio plano interior de la arista, la intersección es PE; si ocasiona la salida del medio plano interior de la arista, es PL. Observe que, con esta distinción, dos puntos de intersección interior de una línea que interseca el rectángulo de recorte tienen etiquetas opuestas.

Las intersecciones se pueden clasificar de manera formal como PE o PL con base en el ángulo entre  $P_0P_1$  y  $N_i$ ; si el ángulo es menor que  $90^\circ$ , la intersección es PL; si es mayor que  $90^\circ$ , es PE. Esta información está contenida en el signo del producto punto de  $N_i$  y  $P_0P_1$ :

$$N_i \cdot D < 0 \Rightarrow \text{PE} \quad (\text{ángulo mayor que } 90^\circ),$$

$$N_i \cdot D > 0 \Rightarrow \text{PL} \quad (\text{ángulo menor que } 90^\circ)$$

Observe que  $N_i \cdot D$  es el denominador de la ecuación (3.1), lo que significa que, en el proceso de cálculo de  $t$ , la intersección se puede categorizar de manera trivial. Con esta categorización, la línea 3 de la figura 3.29 sugiere el paso final del proceso. Debemos elegir un par (PE, PL) que defina la línea recortada. La porción de la línea infinita que pasa por  $P_0P_1$  y que se encuentra dentro de la región de recorte está limitada por la intersección PE con el mayor valor de  $t$ , la cual llamaremos  $t_E$ , y la intersección de PL con el menor valor de  $t$ ,  $t_L$ . El segmento de línea intersecante se define entonces con el intervalo  $(t_E, t_L)$ . Sin embargo, como nos interesa la intersección de  $P_0P_1$ , no de la línea infinita, es necesario modificar la definición del intervalo para que  $t = 0$  sea una cota inferior de  $t_E$  y  $t = 1$  sea una cota superior de  $t_L$ . ¿Qué sucede si  $t_E > t_L$ ? Esto es exactamente lo que ocurre con la línea 2. Quiere decir que ninguna porción de  $P_0P_1$  está dentro del rectángulo de recorte, y se rechaza toda la línea. Los valores de  $t_E$  y  $t_L$  que corresponden a intersecciones reales se utilizan para calcular las coordenadas  $x$  y  $y$  correspondientes.

En el programa 3.8 se presenta el seudocódigo del algoritmo completo para rectángulos de recorte verticales. La versión completa del código, adaptada de [LIAN84] puede encontrarse en [FOLE90] como figura 3.45.

**Programa 3.8**

```

Seudocódigo para el
algoritmo paramétrico de
recorte de líneas de
Cyrus-Beck.

    {
        calcular previamente  $N_i$  y seleccionar  $P_E$  para cada arista
        for (cada segmento de línea que se recortará) {
            if ( $P_1 = P_0$ )
                la linea es degenerada, por lo que se recorta como punto;
            else {
                 $t_E = 0$ ;
                 $t_L = 1$ ;
                for (cada intersección candidata con arista de corte) {
                    if ( $N_i \cdot D \neq 0$ ) { /* Ignorar las aristas paralelas a la línea */
                        calcular  $t$ ;
                        usar signo de  $N_i \cdot D$  para categorizar como PE o PL;
                        if (PE)  $t_E = \max(t_E, t)$ ;
                        if (PL)  $t_L = \min(t_L, t)$ ;
                    }
                }
                if ( $t_E > t_L$ )
                    return nulo;
                else
                    return  $P(t_E)$  y  $P(t_L)$  como intersecciones de recorte reales;
            }
        }
    }
}

```

En resumen, el algoritmo de Cohen-Sutherland es eficiente cuando la prueba de códigos de región se puede realizar a bajo costo (p. ej., con operaciones bit a bit en lenguaje ensamblador) y la aceptación o el rechazo trivial es aplicable a la mayoría de los segmentos de línea. El recorte paramétrico de líneas es lo más conveniente cuando hay que recortar varios segmentos, ya que se pospone el cálculo real de las coordenadas de los puntos de intersección hasta que en verdad sea necesario, y la evaluación se puede efectuar con valores paramétricos. Sin embargo, el cálculo paramétrico se realiza incluso para puntos extremos que se habrían aceptado trivialmente en la estrategia de Cohen-Sutherland. El algoritmo de Liang-Barsky es más eficiente que la versión de Cyrus-Beck, gracias a las pruebas adicionales de rechazo trivial que evitan el cálculo de los cuatro valores de parámetros para las líneas que no intersecan el rectángulo de recorte. En el caso de líneas que el algoritmo de Cohen-Sutherland no puede rechazar porque no se encuentran en un medio plano invisible, las pruebas de rechazo de Liang-Barsky son obviamente preferibles a los recortes repetidos que se requieren en el algoritmo de Cohen-Sutherland. El algoritmo de Nicholl *et al.* [NICH87] por lo general es más conveniente que el de Cohen-Sutherland o el de Liang-Barsky, pero no se extiende al caso tridimensional, como sucede con los recortes paramétricos. En [DUVA90] se analizan formas de acelerar el algoritmo de Cohen-Sutherland.

## 3.10 Recorte de círculos

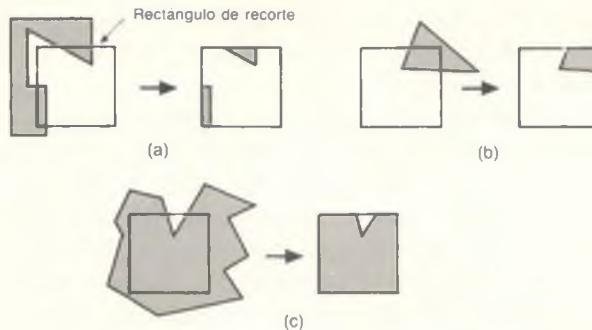
Para recortar un círculo con respecto a un rectángulo, primero podemos efectuar una prueba de aceptación/rechazo trivial intersecando la extensión del círculo (un cuadrado del tamaño del diámetro del círculo) con el rectángulo de recorte, para luego aplicar el algoritmo presentado en la siguiente sección para el recorte de polígonos. Si el círculo interseca el rectángulo, se divide en cuadrantes y se realiza una prueba de aceptación o rechazo trivial para cada uno. A su vez, estas pruebas pueden generar la necesidad de evaluar octantes. Despues podemos calcular en forma analítica la intersección del círculo y la arista, resolviendo sus ecuaciones simultáneamente, y discretizar los arcos resultantes usando el algoritmo con valores iniciales apropiados con los puntos iniciales y finales calculados (y redondeados de manera apropiada). Si la discretización es rápida o el círculo no es demasiado grande, quizás sea más eficaz tijeretear pixel por pixel, evaluando cada pixel de frontera contra las cotas del rectángulo antes de escribirlo. En cualquier caso sería útil revisar la extensión. Si el círculo está llenado, los tramos de pixeles interiores adyacentes de cada línea de rastreo se pueden llenar sin revisión de cota si se recorta cada tramo y luego se llenan sus pixeles interiores.

## 3.11 Recorte de polígonos

Un algoritmo que recorta un polígono debe tratar muchos casos diferentes, como se ilustra en la figura 3.30. El caso de la parte (a) es especialmente notorio, ya que se recorta un polígono cóncavo para obtener dos polígonos separados. En resumen, la tarea de recorte parece ser bastante compleja. Es necesario evaluar cada arista del polígono con cada arista del rectángulo de recorte, añadir nuevas aristas y descartar, conservar o dividir aristas existentes. Al recortar un polígono se pueden generar varios más. Por consiguiente, necesitamos una forma organizada de tratar todos estos casos.

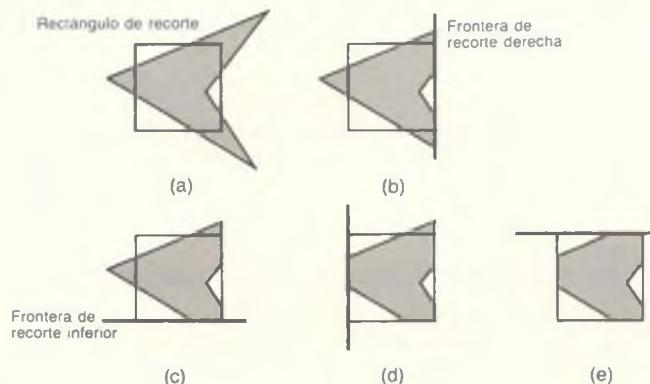
### 3.11.1 Algoritmo de recorte de polígonos de Sutherland-Hodgman

El algoritmo de recorte de polígonos de Sutherland-Hodgman [SUTH74b] utiliza una estrategia de “divide y vencerás”: resuelve una serie de problemas sencillos e idénticos que, al combinarse, resuelven el problema general. El problema sencillo es recortar un polígono con respecto a una arista de corte infinita. Cuatro aristas de corte sucesivas, cada una definiendo una frontera del rectángulo de recorte (Fig. 3.31), recortan sucesivamente un polígono con respecto a un rectángulo.



**Figura 3.30** Ejemplos de recorte de polígonos. (a) Componentes múltiples. (b) Caso convexo simple. (c) Caso cóncavo con varias aristas exteriores.

Observe la diferencia entre esta estrategia para un polígono y el algoritmo de recorte de líneas de Cohen-Sutherland: en el caso de los polígonos se recorta con respecto a cuatro aristas sucesivas, mientras que para las líneas se evalúa el código de región para determinar qué aristas se cruzan, y los recortes sólo se efectúan cuando es necesario. El algoritmo de Sutherland-Hodgman de hecho es más general: se puede recortar un polígono (convexo o cóncavo) con respecto a cualquier polígono de recorte convexo; en tres dimensiones, los polígonos se pueden recortar con respecto a volúmenes poliédricos convexos definidos por planos. El algoritmo acepta una serie de vértices de polígonos  $v_1, v_2, \dots, v_n$ . En dos dimensiones, los vértices definen aristas de polígonos de  $v_i$  a  $v_{i+1}$  y de  $v_n$



**Figura 3.31** Recorte de polígonos, arista por arista. (a) Antes del recorte. (b) Recorte por la derecha. (c) Recorte por la parte inferior. (d) Recorte por la izquierda. (e) Recorte por la parte superior; el polígono está totalmente recortado.

a  $v_1$ . El algoritmo recorta con respecto a una sola arista infinita y produce otra serie de vértices que define el polígono recortado. En una segunda pasada se recorta el polígono parcialmente recortado con respecto a la segunda arista, etcétera.

El algoritmo recorre el polígono de  $v_n$  a  $v_1$  y luego de regreso a  $v_n$ , examinando en cada paso la relación entre los vértices sucesivos y la arista de recorte. En cada paso se añaden cero, uno o dos vértices a la lista de salida de los vértices que definen el polígono recortado. Es necesario analizar cuatro casos posibles, como se ilustra en la figura 3.32.

Consideremos la arista de polígono del vértice  $s$  al vértice  $p$  en la figura 3.32. Suponga que el punto de partida  $s$  fue tratado en la iteración anterior. En el caso 1, cuando la arista del polígono está completamente dentro de las fronteras de recorte, se agrega el vértice  $p$  a la lista de salida. En el caso 2, el punto de intersección  $i$  se produce como vértice porque la arista interseca la frontera. En el caso 3, ambos vértices se hallan fuera de las fronteras, por lo que no hay salida. En el caso 4, el punto de intersección  $i$  y el vértice  $p$  se añaden a la lista de salida.

La función `recorte_polígonos_Sutherland_Hodgman()` que se presenta en el programa 3.9 acepta un arreglo de vértices `arreglo_vértices_entrada` y produce otro, `arreglo_vértices_salida`. Para que el código sea lo más sencillo posible, no efectuamos la revisión de errores de las cotas de arreglo y empleamos la función `salida()` para colocar un vértice en `arreglo_vértices_salida`. La función `intersección()` calcula la intersección de la arista de polígono del vértice  $s$  al vértice  $p$  con `frontera_recorte`, definido por dos vértices en la frontera del polígono de recorte. La función `interior()` devuelve el valor TRUE (verdadero) si el vértice se halla en el interior de la frontera de recorte, donde “interior” se define como “a la izquierda de la frontera de recorte cuando se observa del primer vértice al segundo de la frontera de recorte”. Este sentido corresponde a una numeración de las aristas en sentido contrario al giro de las manecillas del reloj. Para calcular si un punto está fuera de una frontera de

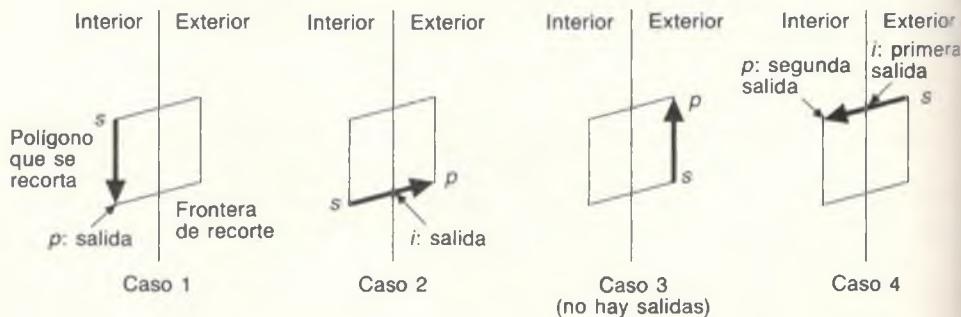


Figura 3.32 Cuatro casos de recorte de polígonos.

recorte, podemos evaluar el signo del producto punto de la normal a la frontera de recorte y la arista del polígono, como se describió en la sección 3.9.4. (Para el sencillo caso de un rectángulo de recorte vertical, sólo hay que evaluar el signo de la distancia horizontal o vertical a la frontera.)

Sutherland y Hodgman muestran cómo estructurar el algoritmo para que sea recursivo [SUTH74b]. Tan pronto como se produce un vértice, el algoritmo se llama a sí mismo con ese vértice. El recorte se efectúa con respecto a la siguiente frontera de recorte, de manera que no hay que utilizar almacenamiento intermedio para el polígono parcialmente recortado. En esencia, el polígono pasa por un *ducto* de recortadores. Cada paso se puede implantar como hardware de propósito especial sin espacio de memoria gráfica. Esta propiedad (y su generalidad) hace que el algoritmo sea muy apropiado para las implantaciones de hardware actuales. Sin embargo, tal y como se presenta el algoritmo, se pueden introducir nuevas aristas en la frontera del rectángulo de recorte. Considere la figura 3.30(a), donde se introduce una arista nueva al conectar la parte superior izquierda del triángulo con la parte superior izquierda del rectángulo. Estas aristas se pueden eliminar con una fase de procesamiento posterior.

### Programa 3.9

*Algoritmo de recorte de polígonos de Sutherland-Hodgman.*

```

typedef struct vértice {
    float x, y;
} vértice;

typedef vértice arista[2];
typedef vértice arreglo_vértices[MÁX];/* MÁX es una constante declarada */

void intersección (vértice primero, vértice segundo, vértice *frontera_recorte, vértice
                    *punto_intersección)
{
    if (frontera_recorte[0].y == frontera_recorte[1].y) {           /* horizontal */
        punto_intersección->y = frontera_recorte[0].y;
        punto_intersección->x = primero.x + (frontera_recorte[0].y - primero.y) * (segundo.x -
            primero.x) / (segundo.y - primero.y);
    } else {                                              /* vertical */
        punto_intersección->x = frontera_recorte[0].x;
        punto_intersección->y = primero.y + (frontera_recorte[0].x - primero.x) * (segundo.y -
            primero.y) / (segundo.x - primero.x);
    }

boolean interior (vértice vértice_prueba, vértice *frontera_recorte)
{
    if (frontera_recorte[1].x > frontera_recorte[0].x)           /* inferior */
        if (vértice_prueba.y >= frontera_recorte[0].y) return TRUE;
    if (frontera_recorte[1].x < frontera_recorte[0].x)           /* superior */
        if (vértice_prueba.y <= frontera_recorte[0].y) return TRUE;
    if (frontera_recorte[1].y > frontera_recorte[0].y)           /* derecha */
        if (vértice_prueba.x <= frontera_recorte[1].x) return TRUE;
    if (frontera_recorte[1].y < frontera_recorte[0].y)           /* izquierda */
        if (vértice_prueba.x >= frontera_recorte[1].x) return TRUE;
}
```

```

if (vértice_prueba.x >= frontera_recorte[1].x) return TRUE;
return FALSE;

}

void salida (vértice vértice_nuevo, int *longitud_salida, vértice * arreglo_vértices_salida)
{
    (*longitud_salida) + +;
    arreglo_vértices_salida[*longitud_salida - 1].x = vértice_nuevo.x;
    arreglo_vértices_salida[*longitud_salida - 1].y = vértice_nuevo.y;
}

void recorte_polígonos_Sutherland_Hodgman (vértice, *arreglo_vértices_entrada,
                                            vértice *arreglo_vértices_salida; int longitud_entrada, int *longitud_salida,
                                            vértice *frontera_recorte)
{
    vértice s, p, i;
    int j;

    *longitud_salida = 0;
    s = arreglo_vértices_entrada[longitud_entrada - 1];           /* Iniciar con el último vértice
        en arreglo_vértices_entrada */
    for (j = 0; j < longitud_entrada; j + +) {
        p = arreglo_vértices_entrada[j];      /* Ahora s y p corresponden a los vértices de la
            figura 3.33 */
        if (interior (p, frontera_recorte))          /* Casos 1 y 4 */
            if (interior (s, frontera_recorte))
                salida (p, longitud_salida, arreglo_vértices_salida) /* Caso 1 */
            else {
                intersección (s, p, frontera_recorte, &i);
                salida (i, longitud_salida, arreglo_vértices_salida);
                salida (p, longitud_salida, arreglo_vértices_salida);
            }
        } else if (interior (s, frontera_recorte)) {           /* Casos 2 y 3 */
            intersección (s, p, frontera_recorte, &i);          /* Caso 2 */
            salida (i, longitud_salida, arreglo_vértices_salida);
        }
    s = p;                                         /* No hay acción para el caso 3 */
}                                                 /* Avanzar al siguiente par de vértices */
}

```

## 3.12 Generación de caracteres

### 3.12.1 Definición y recorte de caracteres

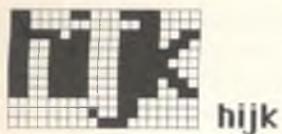
Existen dos técnicas básicas para definir caracteres. La más general pero computacionalmente más costosa es definir cada carácter como un bosquejo curvo o poligonal y discretizarlo cuando se le requiera. Analizaremos primero el otro

método, más sencillo, en el cual cada carácter de una familia tipográfica determinada se especifica como un pequeño mapa de bits. La generación de un carácter comprende entonces simplemente usar la función de copiado de pixeles para copiar la imagen del carácter de un lienzo fuera de pantalla, llamado memoria caché de familias tipográficas, a la posición deseada en la memoria gráfica.

La memoria caché de familias tipográficas puede en realidad estar en la memoria gráfica, de la siguiente manera. En la mayoría de los sistemas gráficos en los cuales la pantalla se refresca a partir de una memoria gráfica privada, esta memoria es mayor que lo que en realidad se necesita para almacenar la imagen presentada. Por ejemplo, los pixeles para una pantalla rectangular se pueden almacenar en una memoria cuadrada, con lo cual queda una franja rectangular de memoria de pantalla *invisible*. Por otra parte, puede haber memoria suficiente para dos pantallas, una que se refresca y una donde se dibuja, para una doble memoria de la pantalla. La memoria caché de las familias tipográficas que se presentan generalmente se almacena en esta memoria de pantalla invisible, ya que la función de copiado de pixeles del controlador del dispositivo opera más rápidamente con la memoria local de la imagen. Una aplicación relacionada de esta memoria invisible es almacenar áreas de pantalla temporalmente ocultas por imágenes como ventanas, menús y formas.

Los mapas o arreglos bidimensionales de bits (*bitmaps*) para la memoria caché de familias tipográficas por lo general se crean digitalizando imágenes ampliadas de los caracteres de familias tipográficas en diversos tamaños; un diseñador de tipos de letras usa entonces un programa de pintura para retocar los pixeles individuales en el arreglo bidimensional de bits de los caracteres. De manera alternativa, el diseñador de tipos puede usar un programa de pintura para crear, desde cero, familias tipográficas especialmente diseñadas para pantallas e impresoras de baja resolución. Como los mapas de bits pequeños no se escalan muy bien, hay que definir más de un arreglo para un carácter de una familia tipográfica a fin de ofrecer varios tamaños estándar. Así mismo, cada tipo de letra requiere su propio conjunto de mapas de bits; por lo tanto, se requiere una memoria caché de familias tipográficas para cada familia cargada por la aplicación.

Los caracteres de mapa de bits son recortados de manera automática por SRGP como parte de su implantación de copyPixel. Cada carácter se recorta pixel por pixel de acuerdo con el rectángulo destino, una técnica que nos permite recortar un carácter en cualquier fila o columna de su mapa de bits. En el caso de sistemas con una función de copiado de pixeles más lenta, un método más rápido, aunque burdo, es recortar el carácter o incluso toda la cadena de caracteres con base en “todo o nada”, realizando una aceptación trivial de la extensión del carácter o la cadena. La función de copiado de pixeles se aplica al carácter o la cadena sólo si la extensión es aceptada trivialmente. Incluso en sistemas con una rutina rápida de copiado de pixeles, puede ser conveniente efectuar una prueba de aceptación/rechazo trivial de la extensión de la cadena como precursora del recorte de los caracteres individuales durante la operación de copiado de pixeles.



**Figura 3.33**  
Parte de un ejemplo de memoria caché de familias tipográficas.

Con la sencilla técnica de memoria caché de familias tipográficas con mapas de bits que emplea SRGP, los caracteres se almacenan lado a lado en un lienzo bastante ancho pero cuya altura es la del carácter más alto; en la figura 3.33 se muestra una porción de esta memoria caché, así como los ejemplares discretos de los mismos caracteres a baja resolución. Cada familia cargada describe con una estructura (declarada en el programa 3.10) que contiene una referencia al lienzo que almacena las imágenes de los caracteres, así como información acerca de la altura de los caracteres y el espacio que debe colocarse entre caracteres en una cadena de texto. (Algunos paquetes almacenan el espacio entre caracteres como parte de la anchura del carácter, a fin de permitir el espacio variable entre caracteres.)

#### Programa 3.10

Declaraciones de tipo para la memoria caché de familias tipográficas.

```
typedef struct ubicación_carácter {
    int x_izquierda, anchura; /* Ubicación horizontal y anchura de la
} ubicación_carácter; imagen en la memoria caché de familias tipográficas */

typedef struct descriptor_caché_familias {
    entero Índice_lienzo caché; /* La altura es constante, la anchura varía */
    int altura_asta_descendente, altura_total; /* Medido en pixeles */
    int espaciado_entre_caracteres;
    ubicación_carácter tabla_ubicación[128];
} descriptor_caché_familias;
```

Como se describió en la sección 2.1.5, la altura del asta descendente y la altura total son constantes para una familia tipográfica determinada; la primera medida es el número de filas o pixeles en la parte inferior de la memoria cache que sólo utilizan los caracteres con asta descendente, mientras que la segunda es simplemente la altura del lienzo de la memoria caché de familias tipográficas. Sin embargo, la anchura de un carácter no se considera constante; por lo tanto, un carácter puede ocupar el espacio que requiera en lugar de forzarlo a un espacio de tamaño fijo. SRGP coloca un espaciado fijo entre caracteres al dibujar una cadena de texto, cantidad que se especifica como parte del descriptor de la familia tipográfica. Una aplicación de procesamiento de texto puede presentar líneas de texto usando SRGP para mostrar palabras individuales del texto y justificar las líneas por la derecha empleando espaciado variable entre palabras y después de los signos de puntuación, de manera que se rellenen las líneas y los caracteres de la derecha estén alineados en el margen derecho. Esta aplicación comprende el uso de los recursos de consulta de extensión de texto para determinar dónde está el extremo derecho de cada palabra y así calcular el inicio de la siguiente. Está de más decir que los recursos de manejo de texto de SRGP son demasiado burdos para tareas de procesamiento de texto más complejas, mucho menos para programas de tipografía, ya que estas aplicaciones requieren un control más estricto sobre el espaciado de las letras individuales para manejar efectos como subíndices, supraíndices, espaciado entre caracteres e impresión de texto que no está alineado horizontalmente.

### 3.12.2 Implantación de una primitiva de salida de texto

En el código del programa 3.11 presentamos la manera en que se implanta internamente el manejo de texto en SRGP. Cada carácter de la cadena se coloca de manera individual y el espacio entre caracteres se determina con el campo correspondiente en el descriptor de la familia tipográfica. Observe que las complejidades como la combinación de familias en una cadena deben ser manejadas por el programa de aplicación.

*Programa 3.11*

*Implantación de la  
ocación de caracteres  
ra la primitiva de texto  
de SRGP.*

```
void SRGP_characterText (point origen, char cadena_ímpresión,
                        fontCacheDescriptor información_familia)
{
    /* El origen es donde se colocará el carácter en el lienzo actual */
    rectangle rectángulo_caché_familias;
    char carácter_ímpresión;
    int i;
    charLocation *fp;

    /* El origen especificado por la aplicación es la linea base y no incluye el asta descendente */
    origen.y -= información_familia.altura_asta_descendente;

    for(i = 0; i < strlen(cadena_ímpresión); i + +){
        carácter_ímpresión = cadena_ímpresión[i];
        fp = &información_familia.tabla_ubicación [carácter_ímpresión];
        /* Encontrar la región rectangular en la memoria caché donde está el carácter */
        SRGP_defPoint (fp -> x_izquierda, 0,
                      rectángulo_caché_familias.inferior_izquierda);
        SRGP_defPoint (fp -> x_izquierda + fp -> anchura - 1, información_familia.altura_total
                      - 1, rectángulo_caché_familias.superior_derecha);
        SRGP_copyPixel (información_familia.caché, rectángulo_caché_familias, origen);
        /* Actualizar el origen para avanzar después del nuevo carácter más el espaciado entre
         * caracteres */
        origen.x += fp -> anchura + espaciado_entre_caracteres;
    }
}
```

Ya mencionamos que la técnica de mapa de bits requiere una memoria caché de familias tipográficas distinta para cada combinación de familia, tamaño y tipo de letra para cada resolución de los dispositivos de salida o presentación que se apoyen. Una sola familia tipográfica en ocho puntajes diferentes y cuatro tipos de letra (romano, negritas, cursivas y negritas cursivas) requiere entonces 32 memorias caché de familias tipográficas! Una manera de resolver este problema de almacenamiento es representar los caracteres en una forma abstracta, independiente del dispositivo, usando bosquejos curvos o poligonales de sus formas definidos con parámetros de punto flotante, para después transformar los caracteres de manera apropiada. Las funciones polinomiales denominadas *splines* (véase el Cap. 9) proporcionan curvas suaves con derivadas continuas de primer orden y mayores, que se usan con frecuencia para codificar

bosquejos de texto. Aunque la definición de cada carácter ocupa más espacio que su representación en la memoria caché de familias tipográficas, se pueden obtener varios tamaños de una sola representación almacenada si se emplea el escalamiento apropiado; así mismo, las cursivas se pueden aproximar rápidamente con una inclinación del bosquejo. Otra ventaja importante del almacenamiento de caracteres en forma independiente del dispositivo es que los bosquejos se pueden trasladar, rotar, escalar o recortar arbitrariamente (incluso se pueden usar como regiones de recorte).

La economía de almacenamiento de los caracteres con *splines* no es tan grande como pudiera inferirse de esta descripción. Por ejemplo, no se pueden obtener todos los puntajes de un carácter a partir del escalamiento de una sola forma abstracta, ya que la forma de una familia tipográfica estética generalmente es una función del puntaje; por lo tanto, cada forma sólo es suficiente para una gama limitada de puntajes. Además, la discretización del texto representado con *splines* requiere mucho más procesamiento que la implantación con copiado de pixeles, ya que la forma independiente del dispositivo debe convertirse a coordenadas de pixeles con base en el tamaño, tipo y atributos de transformación. Por lo anterior, la técnica de memoria caché de familias tipográficas sigue siendo la más común en los computadores personales e incluso se emplea en varias estaciones de trabajo. Una estrategia que nos ofrece lo mejor de ambos métodos es almacenar las familias en forma de bosquejo pero discretizar aquellas que se usan en una aplicación a sus equivalentes en forma de mapa de bits, por ejemplo construyendo sobre la marcha una memoria caché de familias tipográficas. En [FOLE90], sección 19.4, puede hallar un análisis más detallado del texto representado con *splines*.

### 3.13 SRGP\_copyPixel

Si las únicas funciones de bajo nivel disponibles son *escribir\_pixel* y *leer\_pixel*, la función SRGP\_copyPixel se puede implantar como un ciclo con anidamiento doble para cada pixel. Para simplificar, suponga primero que se trabaja con una pantalla de dos niveles y que no hay que considerar los aspectos de bajo nivel de la escritura de bits que no se encuentren alineados con palabras. En el ciclo interior de nuestra sencilla función SRGP\_copyPixel, usamos la función *leer\_pixel* para leer los pixeles fuente y destino, los combinamos lógicamente de acuerdo con el modo de escritura de SRGP y ejecutamos *escribir\_pixel* con el resultado. Si el modo *replace*, el modo de escritura más común, se trata como un caso especial, se puede usar un lazo interno simple que realice una lectura y una escritura de pixel de la fuente al destino sin tener que efectuar la operación lógica. El rectángulo de recorte se utiliza durante el cálculo de direcciones para limitar la región a aquella donde se escriben los pixeles destino.

## 3.14 Eliminación de artefactos de discretización (*Antialiasing*)

### 3.14.1 Aumento de la resolución

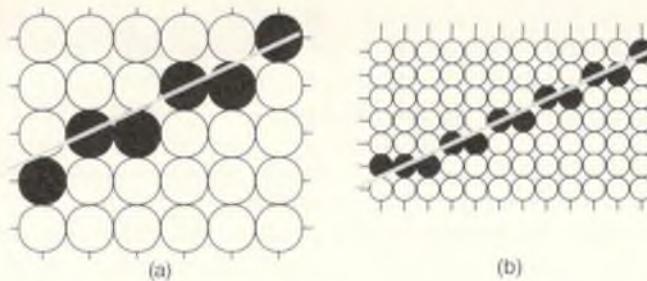
Todas las primitivas que hemos considerado hasta ahora tienen un problema común: sus aristas son irregulares. Este indeseable efecto, conocido como **serrado** o **escalonamiento**, es el resultado del enfoque de “todo o nada” de la discretización, por la cual cada pixel se reemplaza con el color de la primitiva o permanece sin cambio. El serrado es un ejemplo de un fenómeno conocido como **artefacto de discretización** (*aliasing*). La aplicación de técnicas para reducir o eliminar este fenómeno se conoce como **eliminación de artefacto de discretización** (*antialiasing*). En el capítulo 14 de [FOLE90] se analizan las ideas básicas del procesamiento de señales que explican por qué el artefacto de discretización recibió ese nombre, por qué ocurre y cómo reducirlo o eliminarlo al crear imágenes. Por el momento nos conformaremos con una explicación más intuitiva de las razones por las cuales las primitivas de SRGP presentan artefactos de discretización, y describiremos cómo modificar el algoritmo de discretización de líneas desarrollado en este capítulo para generar líneas con eliminación de artefacto de discretización.

Considere la utilización del algoritmo de punto medio para dibujar una línea negra de un pixel de grosor, con pendiente entre 0 y 1, sobre fondo blanco. El algoritmo establece el color del pixel más cercano a la línea en cada columna por la cual pasa. Cada vez que la línea pasa de una columna a otra donde los pixeles más cercanos a la línea no están en la misma fila, hay un cambio abrupto en la línea dibujada en el lienzo, como se ilustra en la figura 3.34(a). Esta situación también se presenta con otras primitivas con discretización que sólo pueden asignar uno de dos valores de intensidad a los pixeles.

Suponga ahora que usamos un dispositivo de presentación con el doble de resolución horizontal y vertical. Como se ilustra en la figura 3.34(b), la línea pasa por dos veces más columnas y en consecuencia tiene el doble de escalones, pero el tamaño de cada escalón es la mitad en  $x$  y en  $y$ . Aunque la imagen resultante se ve mejor, esta mejora es a expensas de cuadruplicar el costo de la memoria, el ancho de banda de la memoria y el tiempo de discretización. El aumento de la resolución es una solución costosa que sólo reduce el problema del serrado, pero no elimina el problema. En las secciones siguientes veremos técnicas de eliminación de artefactos de discretización que, no obstante ser menos costosas, producen imágenes muy superiores.

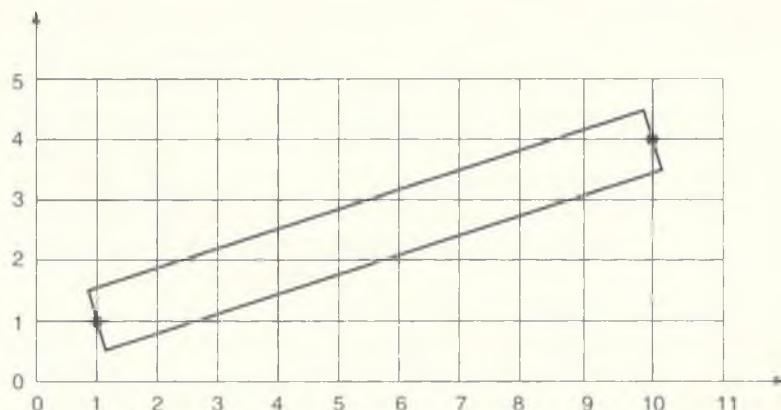
### 3.14.2 Muestreo de área no ponderada

El primer método para mejorar la calidad de las imágenes se puede desarrollar con base en la observación de que, aunque una primitiva ideal (como la línea) tiene anchura de cero, la primitiva que estamos dibujando tiene anchura distinta de cero. Una primitiva discretizada ocupa un área finita en la pantalla;



**Figura 3.34** (a) Línea estándar de punto medio en una pantalla de dos niveles. (b) Misma línea en una pantalla con el doble de resolución lineal.

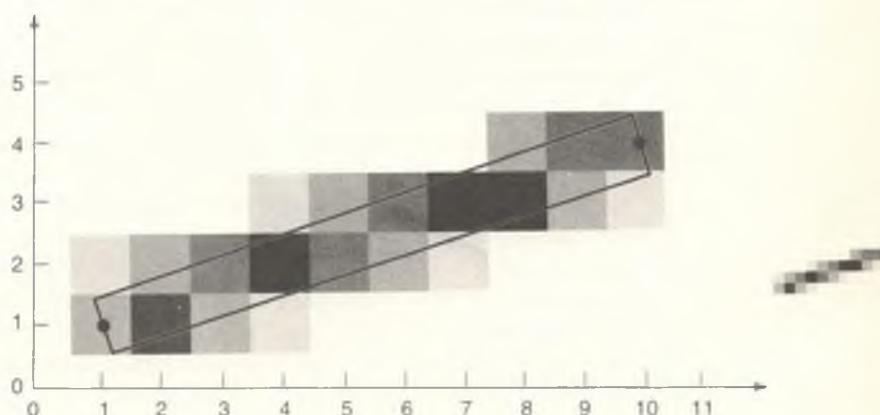
incluso la línea vertical u horizontal más delgada en la superficie de la pantalla tiene un pixel de grosor y las líneas en otros ángulos tienen anchuras que varían en la primitiva. Por ende, consideraremos una línea como un rectángulo del grosor deseado que cubre una porción de la malla, como se ilustra en la figura 3.35. De esto se desprende que una línea no debe establecer a negro la intensidad de un solo pixel en una columna, sino brindar cierta cantidad de intensidad a cada pixel en las columnas que interseca (por supuesto, estas intensidades variables únicamente se pueden mostrar en pantallas que usan varios bits por pixel). Entonces, para las líneas de un pixel de grosor, sólo las líneas horizontales o verticales afectarían exactamente un pixel en su columna o fila. Para las líneas en otros ángulos, se asignaría más de un pixel en cada columna o fila, con la intensidad apropiada.



**Figura 3.35** Línea con anchura distinta de cero que parte del punto (1, 1) al punto (10, 4).

Sin embargo, ¿cuál es la geometría de un pixel? ¿Cuál es su tamaño? ¿Cuánta intensidad debe proporcionar la línea a cada pixel que interseca? Desde el punto de vista computacional, es bastante sencillo suponer que los pixeles forman un arreglo de azulejos cuadrados no superpuestos que cubren la pantalla, centradas en los puntos de la malla, en lugar de considerarlos círculos disjuntos como hicimos antes en este capítulo. [Cuando nos referimos a una primitiva que se sobrepone a un pixel o a una porción de éste, queremos decir que cubre (parte de) el azulejo; para subrayarlo, en ocasiones nos referimos al cuadrado como el *área representada por el pixel*.] También supondremos que una línea contribuye a la intensidad de cada pixel con una cantidad proporcional al porcentaje de la baldosa del pixel que cubre. Un pixel completamente cubierto en una pantalla de negro sobre blanco tendrá color negro, mientras que un pixel parcialmente cubierto será de color gris, con una intensidad que dependerá de la cobertura del pixel. Esta técnica, aplicada a la línea de la figura 3.35, se presenta en la figura 3.36.

En el caso de una línea negra sobre fondo blanco, el pixel (2, 1) es alrededor de un 70 por ciento negro, mientras que el pixel (2, 2) es aproximadamente 25 por ciento negro. Los pixeles como el (2, 3), que no son intersecados por la línea, son completamente blancos. La asignación de la intensidad de un pixel en proporción a la cantidad de área cubierta por la primitiva suaviza la brusca característica de encendido-apagado de la orilla de la primitiva, y ofrece una transición más gradual entre el encendido total y el apagado total. Este aspecto borroso hace que la linea se vea mejor a distancia, a pesar de que distribuye la transición de encendido-apagado por varios pixeles en una columna o fila. Se puede hallar una aproximación burda del área superpuesta al dividir el pixel en una malla más fina de subpixeles rectangulares, para luego contar el número de subpixeles en la línea, por ejemplo debajo de la orilla superior o debajo de la inferior (véase el Ejer. 3.25).



**Figura 3.36** La intensidad de un pixel es proporcional al área cubierta por la línea.

A la técnica de establecer la intensidad en forma proporcional al área cubierta la llamamos **muestreo de área no ponderada**. Esta técnica produce resultados notablemente mejores que al asignar los píxeles a su intensidad total o a intensidad de cero. Sin embargo, existe una estrategia más efectiva denominada **muestreo de área ponderada**. Para explicar las diferencias entre las dos formas de muestreo de área, debemos observar primero que el muestreo de área no ponderada tiene las tres propiedades siguientes. Primero, la intensidad de un pixel intersecado por una línea decrece conforme aumenta la distancia entre el centro del pixel y la orilla: cuanto más lejos esté la primitiva, menor será su influencia sobre la intensidad del pixel. Esta relación es obviamente verdadera porque la intensidad decrece al reducirse el área superpuesta, y esta área se reduce conforme la orilla de la línea se aleja del centro del pixel y se acerca a su frontera. Cuando la línea cubre por completo el pixel, el área de superposición y por ende la intensidad se encuentran al máximo; cuando la orilla de la primitiva es tangente a la frontera, el área y la intensidad son cero.

Una segunda propiedad del muestreo de área no ponderada es que una primitiva no puede influir en la intensidad del pixel si no lo interseca, es decir, si no interseca el azulejo cuadrado que el pixel representa. La tercera propiedad del muestreo de área no ponderada es que las áreas iguales contribuyen con igual intensidad, sin importar la distancia entre el centro del pixel y el área; lo único que importa es el área total superpuesta. Por lo tanto, una pequeña área en la esquina del pixel contribuye lo mismo que un área de igual tamaño cerca del centro.

### 3.14.3 Muestreo de área ponderada

En el muestreo de área ponderada se conservan las dos primeras propiedades del muestreo de área no ponderada (la intensidad decrece al reducirse el área superpuesta y las primitivas sólo contribuyen si se sobreponen al área representada por el pixel), pero se altera la tercera propiedad. Ahora las áreas iguales contribuyen en forma desigual: un área más pequeña cerca del centro del pixel tiene mayor influencia que una lejos del centro. En el capítulo 14 de [FOLE90] se presenta una base teórica para esta modificación; allí se analiza el muestreo de área ponderada en el contexto de la teoría de filtrado.

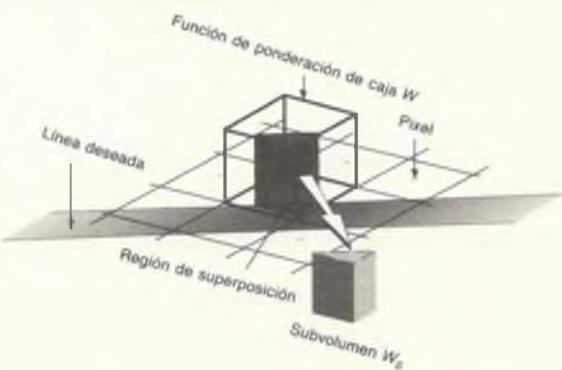
Para conservar la segunda propiedad tenemos que efectuar el siguiente cambio en la geometría del pixel. En el muestreo de área no ponderada, si la arista de una primitiva se encuentra bastante cerca de la frontera de el azulejo cuadrado que hemos usado hasta ahora para representar un pixel, pero en realidad no interseca esta frontera, no contribuirá a la intensidad del pixel. En nuestro nuevo método, el pixel representa un área circular mayor que el azulejo cuadrado; así, la primitiva sí intersecará el área mayor y, por consiguiente, contribuirá a la intensidad del pixel. Observe que esto significa que las áreas relacionadas con píxeles adyacentes en realidad se sobreponen.

Para explicar el origen de los términos *no ponderada* y *ponderada*, definimos una **función de ponderación** que determina la influencia en la intensidad del pixel de una pequeña área  $dA$  de una primitiva, como función de la distancia

de  $dA$  al centro del pixel. Esta función es constante en el muestreo de área no ponderada y disminuye al aumentar la distancia en el caso del muestreo de área ponderada. Considere la función de ponderación como una función  $W(x, y)$  en el plano, cuya altura sobre el plano ( $x, y$ ) indica la ponderación del área  $dA$  en  $(x, y)$ . En el muestreo de área no ponderada, donde los pixeles se representan como azulejos cuadrados, el gráfico de  $W$  es una caja, como se ilustra en la figura 3.37.

En la figura se muestran pixeles cuadrados, con centros indicados por las intersecciones de las líneas de la malla; la función de ponderación aparece como una caja cuya base es el pixel actual. La intensidad que brinda el área del pixel cubierta por la primitiva es el total de las contribuciones de intensidad de todas las pequeñas áreas de la región de superposición de la primitiva y el pixel. La intensidad que aporta cada pequeña área es proporcional al área multiplicada por la altura. Por lo tanto, la intensidad total es la integral de la función de ponderación en el área de superposición. El volumen representado por esta integral,  $W_s$ , siempre es una fracción entre 0 y 1, y la intensidad  $I$  del pixel es  $I_{\max} \cdot W_s$ . En la figura 3.37,  $W_s$  es una rebanada de la caja. La función de ponderación también se conoce como **función de filtrado** y la caja se denomina así mismo **filtro de caja**. En el caso del muestreo de área no ponderada, la altura de la caja se normaliza en 1 para que el volumen de la caja sea 1 lo que ocasiona que una línea gruesa que cubra todo el pixel tenga intensidad  $I = I_{\max} \cdot 1 = I_{\max}$ .

Construyamos ahora una función de ponderación para el muestreo de área ponderada; esta función debe asignar menor peso a las áreas pequeñas lejanas del centro del pixel que a las que se encuentran cerca. Elijamos una función de ponderación que sea la más sencilla de las funciones que decrecen con la distancia; por ejemplo, escogemos una función cuyo máximo corresponda al centro del pixel y que disminuya linealmente al aumentar la distancia con respecto al centro. Debido a la simetría rotacional, el gráfico de esta función forma un cono circular. La base circular del cono (conocida con frecuencia como **soporte**



**Figura 3.37** Filtro de caja para un pixel cuadrado.

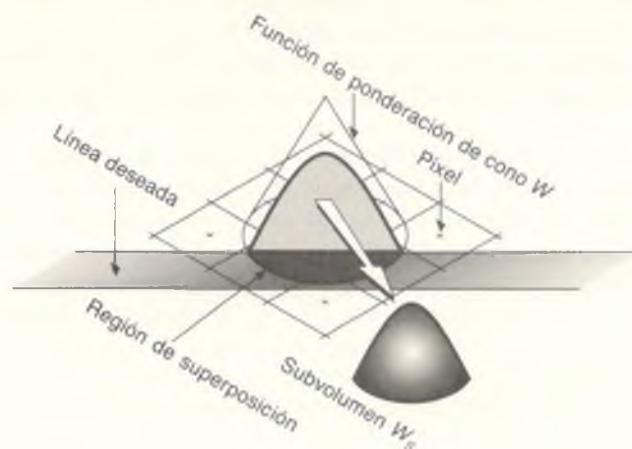
del filtro) debe tener un radio mayor de lo que se esperaría; la teoría de filtrado nos indica que una buena elección para el radio es la distancia unitaria de la malla entera. De esta manera, una primitiva que se encuentre lejos del centro de un pixel todavía puede influir en la intensidad de ese pixel; además, los soportes relacionados con los pixeles vecinos se sobreponen y por consiguiente un pequeño trozo de primitiva puede contribuir en realidad a varios pixeles diferentes (Fig. 3.38). Esta superposición también asegura que no existan áreas de la malla que no estén cubiertas por un pixel, lo que sucedería si los pixeles circulares tuvieran radio igual a la mitad de una unidad de la malla.<sup>9</sup>

Como sucede con el filtro de caja, la suma de todas las contribuciones de intensidad del filtro de cono es el volumen bajo el cono y encima de la intersección de la base del cono y la primitiva; este volumen  $W_s$  es una sección vertical del cono, como se muestra en la figura 3.38. Igual que en el filtro de caja, primero se normaliza la altura del cono para que el volumen debajo de todo el cono sea 1; esto permite que un pixel cuyo soporte está completamente cubierto por una primitiva se presente con la máxima intensidad. Aunque son bastante pequeñas las contribuciones de las áreas de las primitivas que están lejos del centro del pixel pero que sí intersecan el soporte del cono, un pixel cuyo centro esté suficientemente cerca de una línea recibe cierta contribución de intensidad de ella. A la inversa, un pixel que, en el modelo de geometría cuadrada, estaba totalmente cubierto por una línea de grosor unitario<sup>10</sup> no tiene tanto brillo como solía. El efecto neto del muestreo de área ponderada es reducir el contraste entre pixeles adyacentes para proporcionar transiciones más suaves. Específicamente, en el muestreo de área ponderada una línea horizontal o vertical con grosor unitario tiene más de un pixel intensificado en cada fila o columna, lo que no sucedería con el muestreo de área no ponderada.

El filtro cónico tiene dos útiles propiedades: simetría rotacional y reducción lineal de la función al aumentar la distancia radial. Preferimos la simetría rotacional porque no sólo hace que los cálculos de áreas sean independientes del ángulo de la línea, sino que además es teóricamente óptima. Observe, sin embargo, que la pendiente lineal del cono (y su radio) es sólo una aproximación de la función de filtrado óptima, aunque el filtro de cono sigue siendo mejor que el filtro de caja [FOLE90]. Los filtros óptimos son computacionalmente muy costosos y los de caja son los más baratos; por lo tanto, los filtros de cono constituyen un razonable punto medio entre el costo y la calidad. Podríamos integrar sin muchas dificultades el filtro de cono a nuestros algoritmos de discretización. En [FOLE90] puede encontrar detalles de este proceso.

<sup>9</sup> Como señalamos en la sección 3.2, los pixeles que se presentan en un CRT tienen una sección transversal aproximadamente circular, y por lo general los pixeles adyacentes se sobreponen; sin embargo, el modelo de los círculos superpuestos que se emplea en el muestreo de área ponderada no se relaciona en forma directa con este hecho y es aplicable incluso en tecnologías de presentación, como las pantallas de LCD o los tableros de plasma, donde los pixeles físicos son azulejos cuadrados que no se sobreponen.

<sup>10</sup> Ahora decimos “una línea de grosor unitario” en lugar de “una línea con grosor de un pixel” para dejar claro que la unidad del ancho de la línea es el de la malla de SRGP, mientras que el soporte de un pixel tiene que crecer para tener un diámetro de dos unidades.



**Figura 3.38** Filtro de cono para un pixel circular con diámetro de dos unidades de la malla.

## 15 Temas avanzados

En este capítulo sólo hemos tocado algunos de los conceptos de los recortes y la discretización. En la práctica se presentan muchas situaciones complejas que requieren la aplicación de métodos avanzados. Estos métodos reciben un tratamiento completo en el capítulo 19 de [FOLE90], pero es conveniente mencionar algunos aquí.

**Recortes.** Aunque los algoritmos de recorte presentados en este capítulo funcionarán correctamente en la mayoría de los casos, no siempre lo harán con eficiencia. Así mismo, en algunos casos no serán precisos o incluso proporcionarán respuestas incorrectas. Algunos de los algoritmos mejorados son el método de Nicholl-Lee-Nicholl para los recortes bidimensionales, el cual ofrece mucha mayor rapidez que los algoritmos de Liang-Barsky y de Cohen-Sutherland. Además, se presentan situaciones de recorte que ni siquiera hemos considerado, como el caso del recorte de polígonos generales con respecto a otros polígonos generales. En esta situación es útil el algoritmo de polígonos de Weiler.

**Discretización de primitivas.** Sólo hemos considerado la discretización de primitivas simples: líneas, círculos y polígonos. No sólo existen algoritmos más precisos y eficientes para estos objetos, sino además métodos para la discretización de primitivas más complicadas, incluyendo elipses, arcos elípticos, curvas cúbicas y secciones cónicas generales. También hay algoritmos para primitivas gruesas donde la frontera no es únicamente una región matemática, sino que

tiene además anchura arbitraria. En esta clase de problemas se incluye cómo unir segmentos de líneas gruesas de manera atractiva y eficiente. Por último, también hay que considerar el relleno de polígonos autointersecantes, donde no queda claro cuáles son las porciones interiores y cuáles las exteriores.

**Eliminación de artefactos de discretización.** Las situaciones que requieren la eliminación de artefactos de discretización son mucho más numerosas que los casos de líneas rectas que hemos considerado. Así mismo, para la eliminación correcta de estos artefactos se necesita un conocimiento más profundo de la teoría de muestreo. Se requieren algoritmos especiales para círculos, secciones cónicas, curvas generales, además de rectángulos, polígonos y extremos de líneas.

**Texto.** El texto es una entidad altamente especializada, y las técnicas que hemos presentado por lo general no son suficientes. En este capítulo analizamos la utilización de una memoria caché de familias tipográficas para almacenar caracteres que pudieran copiarse directamente al mapa de bits, pero también observamos ciertas limitaciones de este método: se puede requerir una memoria cache diferente para cada tamaño de texto, y el espacio entre caracteres es fijo. Además, aunque algunas versiones de texto en negritas o cursivas se pueden crear a partir de esta memoria caché de familias tipográficas, por lo general no son satisfactorias. Incluso si contamos con un dibujo geométrico preciso del carácter, como el que proporcionaría un diseñador de familias tipográficas, no podríamos discretizarlo trazo por trazo, ya que los resultados no serían aceptables. Por ello se han desarrollado técnicas especializadas para la presentación de texto, las cuales incluyen la eliminación de artefactos de discretización.

**Algoritmos de relleno.** En ocasiones, después de dibujar una secuencia de primitivas queremos colorearlas o deseamos colorear una región definida por un dibujo a mano libre. Por ejemplo, puede ser más sencillo crear un patrón de mosaico construyendo una malla de líneas y luego llenándola con varios colores, en lugar de distribuir de manera uniforme los cuadros coloreados desde el principio. Observe que al emplear la primera técnica no se dibujan primitivas bidimensionales: sólo se emplean áreas bidimensionales que forman el fondo después de dibujar las líneas. De esta manera, la determinación del tamaño de la región que hay que colorear se reduce a la detección de un borde. Los algoritmos que llevan a cabo esta operación se denominan *algoritmos de relleno* e incluyen los de *rellenado de frontera*, *rellenado total* y *rellenado de tinte*. Cada uno de ellos tiene un propósito especial y muchos sistemas gráficos los ofrecen.

## RESUMEN

En este capítulo hemos visto por vez primera los algoritmos fundamentales para recortes y discretización que constituyen el fundamento de los paquetes gráficos de barrido. Aquí sólo hemos abarcado los aspectos básicos; hay que considerar varias elaboraciones y casos especiales para contar con una implantación sólida. En los capítulos 14, 17 y 19 de [FOLE90] se presenta un tratamiento más completo de estos temas.

La idea más importante de este capítulo es que generalmente son mejores los algoritmos incrementales de discretización que sólo utilizan operaciones enteras en sus ciclos internos, ya que la velocidad es esencial en los gráficos de barrido interactivos. Los algoritmos básicos se pueden extender para manejar grosor, además de patrones para fronteras o para relleno de áreas. Mientras que los algoritmos básicos que convierten primitivas de un pixel de ancho tratan de minimizar el error entre los pixeles elegidos en la malla cartesiana y la primitiva ideal definida en el plano, los algoritmos para primitivas gruesas pueden establecer una conciliación entre la calidad y *corrección* y la velocidad. Aunque muchos de los gráficos de barrido bidimensionales en la actualidad operan con primitivas de un solo bit por pixel, incluso en pantallas en color, podemos esperar que prevalezcan las técnicas para la eliminación en tiempo real de artefactos de discretización.

## Ejercicios

- 3.1 Implante el código de caso especial para la discretización de líneas horizontales, verticales y con pendiente de  $\pm 1$ .
- 3.2 Modifique el algoritmo de punto medio para la discretización de líneas (Prog. 3.2) para manejar líneas en cualquier ángulo.
- 3.3 Muestre por qué el error de punto a línea siempre es menor o igual que  $\frac{1}{2}$  en el algoritmo de discretización de punto medio.
- 3.4 Modifique el algoritmo de punto medio para la discretización de líneas del ejercicio 3.2 para manejar el orden de los puntos extremos y las intersecciones con aristas de corte, como se analizó en la sección 3.2.3.
- 3.5 Modifique el algoritmo de punto medio para la discretización de líneas (Ejer. 3.2) para escribir pixeles con intensidad variable como función de la pendiente de la línea.
- 3.6 Modifique el algoritmo de punto medio para la discretización de líneas (Ejer. 3.2) para tratar con puntos extremos que no tengan coordenadas enteras; esto será más fácil si utiliza aritmética de punto flotante en el algoritmo. Como un ejercicio más difícil, maneje líneas con puntos extremos *racionales* usando sólo enteros.
- 3.7 Muestre cómo las polilíneas pueden compartir más que pixeles vértice. Desarrolle un algoritmo que evite la doble escritura de pixeles. *Sugerencia:* Considere como fases separadas la discretización y la escritura en el lienzo en modo **xor**.
- 3.8 Desarrolle una alternativa para el algoritmo de discretización de círculo de punto medio de la sección 3.3.2, con base en una aproximación lineal por trozos del círculo con una polilínea.

3.9 Desarrolle un algoritmo para la discretización de rectángulos redondeados huecos que tengan un radio especificado para las esquinas de cuarto de círculo.

3.10 Escriba una función de discretización para rectángulos verticales con relleno sólido en posiciones arbitrarias en la pantalla. El algoritmo debe escribir de manera eficaz en una memoria gráfica de dos niveles, una palabra de pixeles completa a la vez.

3.11 Construya ejemplos de pixeles *faltantes* o que se escriben varias veces, usando las reglas de la sección 3.5. Trate de desarrollar reglas alternativas, quizás más complejas, que no dibujen dos veces pixeles compartidos en aristas compartidas, pero que no omitan pixeles. ¿Vale la pena el tiempo de proceso adicional que requieren estas reglas?

3.12 Implante el seudocódigo de la sección 3.2 para la discretización de polígonos, teniendo en cuenta el control de tramos de los polígonos astilla potenciales.

3.13 Desarrolle algoritmos de discretización para triángulos y trapezoides que aprovechen la sencilla naturaleza de estas formas. Estos algoritmos son comunes en hardware.

3.14 Investigue algoritmos de triangulación para descomponer un polígono arbitrario, posiblemente cóncavo o autointersecante, en una malla de triángulos con vértices compartidos. ¿Es conveniente limitar el polígono a que, en el peor de los casos, sea cóncavo, sin autointersecciones ni agujeros interiores? (Véase también [PREP85].)

3.15 Extienda el algoritmo de discretización de círculos (Prog. 3.4) para manejar círculos llenados y cuñas circulares (para gráficos circulares), usando tablas de tramos.

3.16 Implante algoritmos de anclado absoluto y relativo para el relleno de polígonos con patrones, analizado en la sección 3.7, y compárelos en lo referente al efecto visual y la eficacia de los cálculos.

3.17 Aplique la técnica de la figura 3.20 para escribir caracteres rellenos con patrones en modo opaco. Muestre cómo se puede aprovechar una función de copiado de pixeles con mascarilla de escritura en esta clase de problemas.

3.18 Implante una técnica para dibujar varios símbolos, como iconos de cursor representados con pequeños arreglos bidimensionales de bits, que puedan observarse sin importar el fondo sobre el cual se encuentren. *Sugerencia:* Defina una mascarilla que “encierre” cada símbolo —es decir, que cubra más pixeles que el símbolo— y que dibuje las mascarillas y los símbolos en distintas pasadas.

3.19 Implante algoritmos para líneas gruesas usando las técnicas presentadas en la sección 3.7. Compare su eficacia y la calidad de los resultados que producen.

3.20 Extienda el algoritmo de punto medio para la discretización de círculos (Prog. 3.4) para manejar círculos gruesos.

3.21 Implante un algoritmo de línea gruesa que maneje tanto el estilo de línea como el estilo de pincel y el patrón.

3.22 Modifique el algoritmo de recorte de líneas de Cohen-Sutherland del programa 3.7 para evitar un nuevo cálculo de las pendientes en pasadas sucesivas. Redefina además la estructura *código\_region* para que sea la unión de *unsigned int todas* y las cuatro banderas de un bit, *izquierda, derecha, inferior y superior*.

3.23 Considere un polígono convexo con  $n$  vértices que se recorta con respecto a un rectángulo. ¿Cuál es el número máximo de vértices en el polígono recortado resultante? ¿Cuál es el número mínimo? Considere el mismo problema para un polígono cóncavo. ¿Cuántos polígonos se podrían producir? Si se genera un solo polígono, ¿cuál es el mayor número de vértices que puede tener?

3.24 Explique por qué el algoritmo de recorte de polígonos de Sutherland-Hodgman funciona únicamente con regiones de corte convexas.

3.25 Desarrolle una estrategia para subdividir un pixel y contar el número de subpixeles cubiertos (al menos hasta un grado significativo) por una línea, como parte de un algoritmo de dibujo de líneas que usa muestreo de área no ponderada.



En este capítulo describiremos la forma en que operan los elementos de hardware más importantes de un sistema de presentación gráfica. En la sección 4.1 se abarca la tecnología de impresión: impresoras, graficadores de pluma, impresoras láser, graficadores de chorro de tinta y grabadoras de película fotográfica. Se analizan brevemente los conceptos tecnológicos básicos que sirven de base para cada tipo de dispositivo, y en una sección final se comparan los diversos dispositivos. En la sección 4.2, que trata las tecnologías de pantalla, se analizan los CRT monocromáticos y de color con mascarilla de sombra, las pantallas de cristal líquido (LCD, *liquid-crystal displays*) y las pantallas electroluminiscentes. Una vez más, se presenta una sección final con las ventajas y desventajas de las diversas tecnologías de pantallas.

Los sistemas de pantalla de barrido, que pueden usar cualquiera de las tecnologías que se analizan aquí, se describen en la sección 4.3. Se introduce primero un sencillo sistema de barrido y luego se mejora en lo referente a la funcionalidad gráfica y la integración de procesadores de barrido y de propósito general en el espacio de direcciones del sistema. En la sección 4.4 se describe la función de la tabla de consultas y del controlador de vídeo en la presentación de imágenes, el control del color y el mezclado de imágenes. En la sección 4.5 se analizan los dispositivos de interacción con el usuario, como las tabletas, el ratón, las pantallas sensibles al tacto, etcétera. Una vez más, se subrayan los detalles de operación y no tecnológicos. En la sección 4.6 se tratan brevemente los dispositivos de entrada de imágenes, como los digitalizadores de fotografías, por medio de los cuales se puede introducir una imagen en un computador.

En la figura 4.1 se muestra la relación entre estos dispositivos. El elemento clave es la integración de la UCP y el procesador de pantalla, conocida como

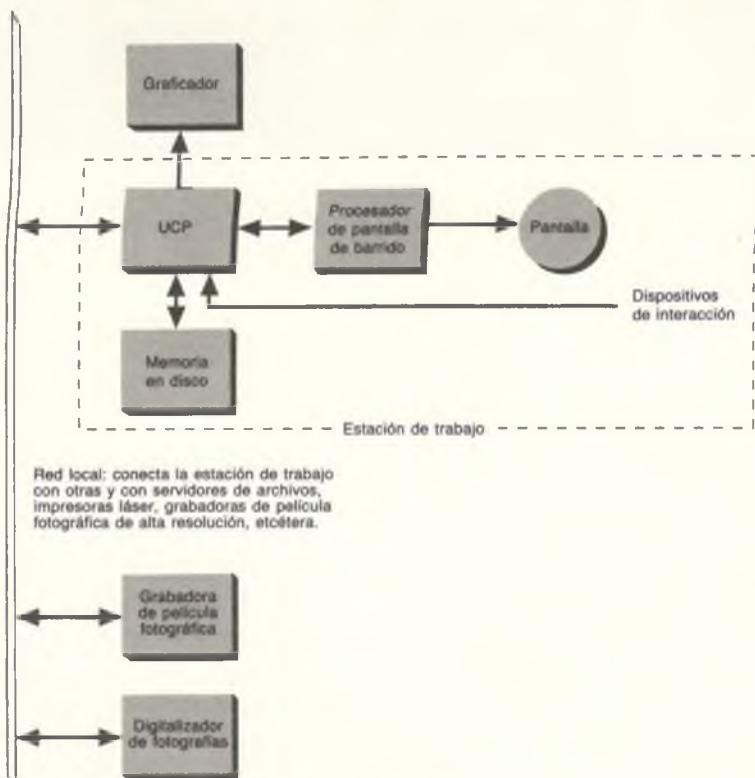


Figura 4.1 Componentes típicos de un sistema gráfico interactivo.

**estación de trabajo gráfica**, que por lo general consiste en una UCP capaz de ejecutar entre 20 y 100 millones de instrucciones por segundo (MIPS) y una pantalla con resolución mínima de  $1000 \times 800$ , o más. La red local conecta varias estaciones de trabajo para que puedan compartir archivos, enviar correo electrónico y compartir periféricos como grabadoras de película fotográfica de alta resolución, discos de gran capacidad, vías de acceso a otras redes y computadores de alto rendimiento.

## 4.1 Tecnologías de impresión

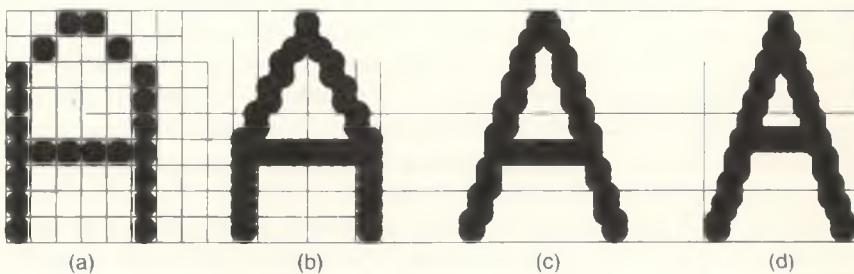
En esta sección analizaremos diversas tecnologías de impresión para luego resumir sus características. Sin embargo, antes de continuar hay que definir varios términos.

La calidad de imagen que se puede obtener con los dispositivos de presentación depende tanto de la capacidad de direccionamiento como del tamaño de punto del dispositivo. El **tamaño de punto** es el diámetro de un punto creado en el dispositivo. La **capacidad de direccionamiento** es el número de puntos por pulgada que se pueden crear; esta cantidad puede variar en las direcciones vertical y horizontal. La capacidad de direccionamiento en  $x$  es el recíproco de la distancia entre los centros de los puntos en las direcciones  $(x, y)$  y  $(x + 1, y)$ ; la capacidad de direccionamiento en  $y$  se define de forma similar. La **distancia entre puntos** es el recíproco de la capacidad de direccionamiento.

Por lo general es deseable que el tamaño de punto sea mayor que la distancia entre puntos, para que se puedan crear formas suaves. Este razonamiento se ilustra en la figura 4.2. Sin embargo, se presentan disyuntivas: un punto con tamaño mucho mayor que la distancia entre puntos permite la impresión de formas muy regulares, mientras que un punto más pequeño permite imprimir detalles más finos.

La **resolución**, que se relaciona con el tamaño del punto y que no puede ser mayor que la capacidad de direccionamiento, es el número de líneas distinguibles por pulgada que puede crear el dispositivo. La resolución se define como el espacio más cercano en el cual un observador puede distinguir líneas negras y blancas adyacentes (una vez más, esto implica que las resoluciones vertical y horizontal pueden variar). Si en una pulgada se pueden distinguir 40 líneas negras alternadas con 40 líneas blancas, la resolución es de 80 líneas por pulgada (también conocida como resolución de 40 pares de líneas por pulgada). La resolución también depende de la distribución de intensidad de la sección transversal del punto. Un punto con orillas muy bien delineadas ofrece mayor resolución que uno con orillas irregulares.

Muchos de los dispositivos que analizaremos sólo pueden crear unos cuantos colores en un punto determinado. Los colores adicionales se obtienen con patrones de punteo (*dither*), los cuales se analizan en el capítulo 11, a expensas de una reducción en la resolución espacial de la imagen resultante.



**Figura 4.2** El efecto de diversas relaciones entre el tamaño de punto y la distancia entre puntos.  
(a) Espaciado entre puntos igual al tamaño de los puntos. (b) Espaciado entre puntos de la mitad del tamaño de punto. (c) Espaciado entre puntos igual a la tercera parte del tamaño de punto. (d) Espaciado entre puntos igual a la cuarta parte del tamaño de punto.

Las **impresoras de matriz de puntos** usan una cabeza de impresión con 724 agujas (pequeños alambres delgados y rígidos) que se pueden disparar individualmente para golpear una cinta contra el papel. La cabeza de impresión se desplaza por el papel una posición a la vez, el papel avanza una línea y la cabeza de impresión se desplaza de nuevo por el papel. Por lo tanto, estas impresoras son dispositivos de salida de barrido y requieren la discretización de imágenes vectoriales antes de la impresión.

Se pueden utilizar cintas de colores para producir impresiones a color. Existen dos métodos para hacerlo. El primero es emplear varias cabezas de impresión, cada una con una cinta de color diferente. El método alternativo, más común, es usar una sola cabeza de impresión con una cinta multicolor.

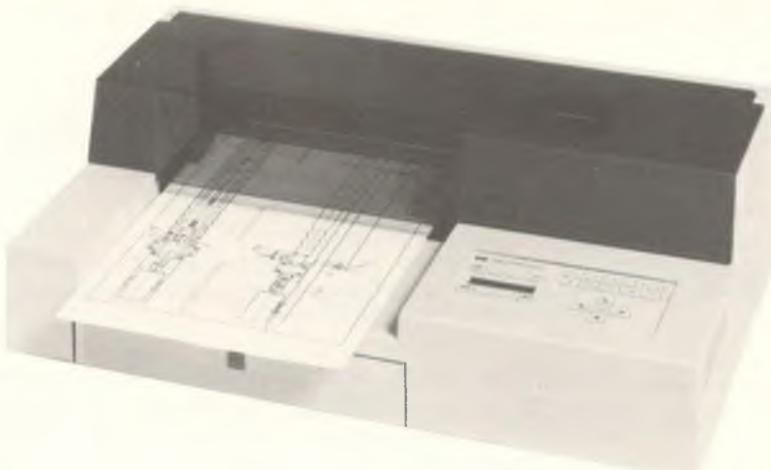
Es posible crear más colores que los que existen en la cinta imprimiendo dos colores en el mismo punto del papel, aunque lo más probable es que el color superior sea más oscuro que el inferior. Con la superposición de tres colores [generalmente *cyan* (azul verdoso), magenta y amarillo] se pueden crear hasta ocho colores en un punto. Sin embargo, el negro que se obtiene con la superposición de los tres colores no es muy bueno, por lo cual es usual que se añada negro verdadero a la cinta.

Un tipo de graficador es el **graficador de pluma**, que desplaza una pluma sobre una hoja de papel en forma aleatoria, de dibujo vectorial. Al dibujar la linea, la pluma se coloca en el inicio de la línea, descende hasta el papel, se mueve a lo largo de una trayectoria recta hasta el punto final, se levanta y se desplaza al inicio de la siguiente línea.

Hay dos variedades principales de graficadores de pluma. El **graficador de cama plana** mueve la pluma por los ejes *x* y *y* sobre un papel colocado en una mesa y sostenido en su lugar por una carga electrostática, por vacío o por simple tensión. Un carro se mueve longitudinalmente sobre la mesa, y sobre él hay una pluma que se desplaza latitudinalmente; esa pluma se puede bajar o subir. Los graficadores de cama plana están disponibles en tamaños de 12 por 18 pulgadas a 6 por 8 pies y mayores.

En contraste, los **graficadores de tambor** mueven el papel sobre un eje y pluma sobre el otro. Por lo general, el papel se extiende sobre un tambor, donde existen agujas que penetran en agujeros previamente perforados en el papel para evitar que se deslice. El tambor puede girar hacia adelante y hacia atrás. Por otra parte, los **graficadores de escritorio** mueven el papel hacia adelante y hacia atrás usando rodillos, mientras la pluma se desplaza sobre el papel (Fig. 4.3).

Las **impresoras láser** rastrean un rayo láser sobre un tambor giratorio con carga positiva recubierto con selenio. Las áreas tocadas por el rayo láser pierden su carga y sólo permanece la carga positiva en las áreas que quedarán en negro. Un pigmento (*toner*) con carga negativa se adhiere a las áreas con carga positiva del tambor y luego se transfiere a papel blanco para formar la impresión. En la xerografía en color este proceso se repite tres veces, una por cada color primario. En la figura 4.4 se presenta un esquema parcial de una impresora láser monocromática. La carga positiva está presente o ausente en un punto determinado del tambor, de manera que será negro o sin color en el punto co-



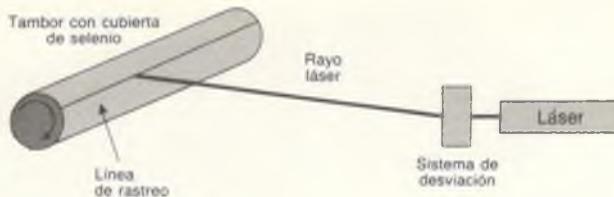
**Figura 4.3** Graficador de escritorio. (Cortesía de Hewlett-Packard Company.)

rrespondiente de la impresión. Por lo tanto, una impresora láser es un dispositivo monocromático de dos niveles o un dispositivo en color de ocho niveles.

Las impresoras láser tienen un microprocesador para realizar la discretización y controlar la impresora. Cada vez más impresoras láser aceptan el lenguaje PostScript de descripción de imágenes y documentos como estándar *de facto* [ADOB85]. PostScript ofrece una descripción por procedimientos de la imagen que se imprimirá y también se puede utilizar para almacenar descripciones de imágenes. La mayoría de las impresoras láser trabajan con papel de 8.5 por 11 pulgadas o de 8.5 por 14 pulgadas, pero hay otras mucho más anchas (hasta 30 pulgadas) para aplicaciones de dibujo de ingeniería y de elaboración de mapas.

Las **impresoras de chorro de tinta** rocían tinta *cyan*, magenta, amarilla y en ocasiones negra sobre el papel. En la mayoría de los casos, las boquillas se montan sobre una cabeza, en un mecanismo parecido al de una impresora convencional. La cabeza de impresión se mueve por el papel para dibujar una línea de rastreo, regresa mientras el papel avanza un espacio entre líneas de rastreo y dibuja la siguiente línea de rastreo. Pueden presentarse pequeñas irregularidades en el espaciado entre líneas si el papel avanza muy poco o demasiado. Además, los colores se depositan simultáneamente, a diferencia de los graficadores y las impresoras láser de varias pasadas. La mayoría de las impresoras de chorro de tinta están limitadas al control de encendido-apagado (es decir, de dos niveles) de cada pixel. Unas cuantas tienen capacidad de tamaño de punto variable.

Las **impresoras de transferencia térmica**, otro dispositivo de impresión de barrido, usan puntas de calentamiento finamente espaciadas (unas 200 por pulgada) para transferir pigmentos de una hoja de cera de color a papel común. El papel encerado y el papel común pasan juntos sobre la tira de puntas de calentamiento, las cuales se calientan selectivamente para transferir el pigmento. En



**Figura 4.4** Organización de una impresora láser (no se muestran el mecanismo de aplicación de tonificador ni el alimentador de papel).

el caso de la impresión en color (la aplicación más común de esta tecnología), el papel encerado se encuentra sobre un rollo con franjas alternas de color *cyan*, magenta, amarillo y negro, cada una con longitud igual a la del tamaño del papel. Como las puntas se calientan y enfrian con rapidez, se puede crear una imagen impresa completa en menos de un minuto. Algunas impresoras de transferencia térmica aceptan una señal de video y la entrada digital de arreglos bidimensionales (mapas) de bits (*bitmaps*), por lo que son convenientes para imprimir imágenes de vídeo.

Las **impresoras de transferencia de tintes por sublimación térmica** operan en forma similar a las de transferencia térmica, excepto que el proceso de calentamiento y transferencia de los tintes permite generar 256 intensidades de *cyan*, magenta y amarillo, a fin de crear imágenes en color de alta calidad con resolución espacial de 200 puntos por pulgada. Este proceso es más lento que la transferencia de cera, pero la calidad es casi fotográfica, por lo cual este tipo de impresora constituye la opción definitiva para pruebas de prensa en color.

Una **cámara** que fotografía una imagen presentada en un **tubo de rayos catódicos (CRT, cathode-ray tube)** se puede considerar como otro dispositivo de impresión. Ésta es la tecnología de impresión más común de nuestro análisis que genera gran número de colores en un solo punto de resolución, ya que la película fotográfica puede capturar muchos colores diferentes.

Las grabadoras de película fotográfica en color utilizan dos técnicas básicas. En una de ellas, la cámara registra la imagen directamente de un CRT en color. La resolución de la imagen es limitada debido a la mascarilla de sombra del monitor en color (véase la Sec. 4.2) y a la necesidad de usar barrido en el monitor de color. En el otro método, se fotografía un CRT en blanco y negro usando filtros de colores, y se presentan en secuencia los componentes de distinto color de la imagen. Esta técnica produce imágenes vectoriales o de barrido de muy alta calidad. Los colores se mezclan con la doble exposición de partes de la imagen a través de dos o más filtros, usualmente con distinta intensidad del CRT.

La entrada de una grabadora de película fotográfica puede ser una señal de vídeo de barrido, un arreglo bidimensional de bits o instrucciones de tipo vectorial. La señal de vídeo puede alimentar directamente a un CRT de color o se pueden separar electrónicamente los componentes rojo, verde y azul para una presentación en secuencia temporal a través de filtros. En ambos casos, la señal

de vídeo debe permanecer constante durante todo el ciclo de grabación, que puede durar hasta un minuto si se emplea película fotográfica relativamente lenta (de baja sensibilidad).

En la tabla 4.1 se resumen las diferencias entre los dispositivos de impresión en color. Si lo desea, puede encontrar información muy detallada acerca de la tecnología de los dispositivos de impresión en [DURB88]. Por supuesto, el ritmo actual de las innovaciones tecnológicas es tan grande que con toda seguridad cambiarán las ventajas y desventajas relativas de algunos de estos dispositivos. Además, algunas de las tecnologías están disponibles en una amplia gama de precios y niveles de rendimiento. Por ejemplo, las grabadoras de película fotográfica y los graficadores de pluma pueden costar entre 500 y 100 000 dólares.

**Tabla 4.1**

## Comparación de varias tecnologías de impresión en color\*

Propiedad	Graficador de pluma	Matriz de puntos	Láser	Chorro de tinta	Foto
Niveles de color por punto	hasta 16	8	8	8-varios	varios
Direcccionamiento, puntos por pulgada	más de 1000	hasta 250	hasta 1500	hasta 200	hasta 800
Tamaño de punto, milésimas de pulgada	15-6	18-10	5	20-8	20-6
Intervalo de costo relativo	B-M	MB	M-A	B-M	M-A
Costo relativo por imagen	B	MB	M	B	A
Calidad de imagen	B-M	B	A	M	M-A
Velocidad	B	B-M	M	M	B

\* MB = muy bajo, B = bajo, M = mediano, A = alto.

Observe que, de todos los dispositivos en color, sólo las grabadoras de película fotográfica, las impresoras de transferencia de tinte por sublimación térmica y algunas impresoras de chorro de tinta pueden capturar una gama amplia de colores. Todas las demás tecnologías usan, en esencia, un control binario de encendido-apagado para los tres o cuatro colores que pueden registrar en forma directa. Observe también que el control de color tiene sus trucos: no hay ninguna garantía de que los ocho colores de un dispositivo se parezcan a los ocho colores de la pantalla o de otro dispositivo de impresión. Consulte la sección 13.4 de [FOLE90], donde se presenta un análisis de las dificultades inherentes de la reproducción de colores.

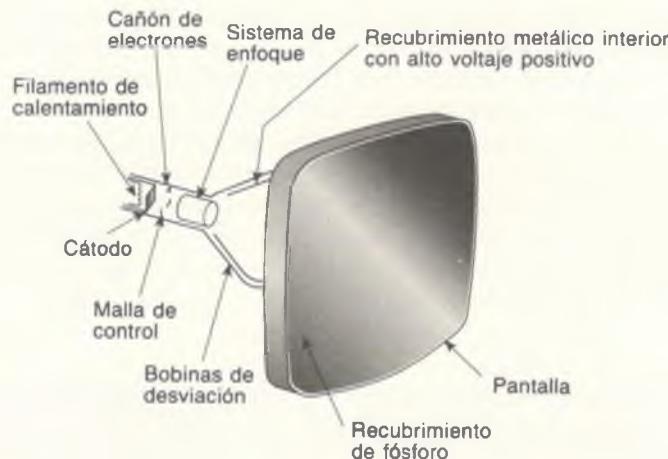
## 2 Tecnologías de pantallas

Los gráficos interactivos por computador requieren dispositivos de presentación cuyas imágenes se puedan modificar rápidamente. Las pantallas con imágenes

no permanentes permiten modificar la imagen, con lo cual es posible el movimiento dinámico de porciones de una imagen. El TRC (tubo de rayos catódicos) es por mucho el dispositivo de presentación más común y lo seguirá siendo por mucho tiempo. Sin embargo, se encuentran en desarrollo tecnologías de estado sólido que quizás, a largo plazo, reduzcan notablemente el dominio del TRC.

Los TRC **monocromáticos** que se utilizan para pantallas gráficas son, en esencia, iguales a los empleados en los televisores domésticos en blanco y negro. En la figura 4.5 se presenta una sección transversal altamente estilizada de un TRC. El cañón de electrones emite un flujo de electrones que se acelera hacia la pantalla con recubrimiento de fósforo, por medio de la aplicación de un alto voltaje positivo cerca del frente del tubo. Al dirigirse hacia la pantalla, el mecanismo de enfoque fuerza los electrones para formar un rayo delgado que se dirige hacia un punto específico de la pantalla mediante el campo magnético producido por las bobinas de desviación. El fósforo emite una luz visible cuando los electrones golpean la pantalla. Como la salida luminosa del fósforo decrece exponencialmente con el tiempo, hay que **refrescar** (redibujar) toda la imagen varias veces por segundo, para que se observe lo que parece ser una imagen constante, sin parpadeo.

La tasa de refrescoamiento de las pantallas de barrido de trama es independiente de la complejidad de la imagen. La tasa de refrescoamiento de los sistemas vectoriales depende directamente de la complejidad de la imagen (el número de líneas, puntos y caracteres): cuanto mayor es la complejidad, más prolongado es el tiempo que requiere un ciclo de refrescoamiento y menor es la tasa de refrescoamiento.



**Figura 4.5** Sección transversal de un TRC (no está a escala).

El flujo de electrones del cátodo calentado se acelera hacia el fósforo con voltaje elevado, por lo general de 15 000 a 20 000 volts, que determina la velocidad alcanzada por los electrones antes de golpear el fósforo. El voltaje de la malla de control determina cuántos electrones hay en el rayo. Si el voltaje de la malla de control es más negativo, pasan menos electrones a través de la malla. Este fenómeno permite controlar la intensidad del punto, ya que la salida luminosa del fósforo disminuye al reducirse el número de electrones en el rayo.

El sistema de enfoque concentra el rayo de electrones para que converja en un pequeño punto al golpear el fósforo. No basta que los electrones en el rayo se muevan en forma paralela entre sí, ya que divergirían debido al rechazo de los electrones. Por lo tanto, el sistema de enfoque debe hacer que converjan para contrarrestar la divergencia. Con la excepción de esta tendencia a divergir, el enfoque de un rayo de electrones es análogo al enfoque de un rayo luminoso.

Cuando el rayo luminoso golpea la pantalla cubierta de fósforo del TRC, los electrones individuales se mueven con energía cinética proporcional al voltaje de aceleración. Parte de esta energía se disipa en forma de calor, pero el resto se transfiere a los electrones de los átomos de fósforo, los cuales saltan a niveles mayores de energía cuántica. Al regresar a sus niveles cuánticos anteriores, estos electrones excitados liberan su energía extra en forma de luz, con frecuencias (es decir, colores) predichas por la teoría cuántica. El fósforo tiene varios niveles cuánticos para un estado sin excitación. Además, en algunos niveles los electrones son menos estables y regresan al estado no excitado con mayor rapidez que otros. La **fluorescencia** del fósforo es la luz emitida cuando estos electrones muy inestables pierden su exceso de energía mientras el fósforo recibe el impacto de los electrones. La **fosforescencia** es la luz emitida por el regreso de los electrones excitados relativamente más estables a su estado de no excitación cuando desaparece la excitación del rayo de electrones. Con los fósforos típicos, la mayor parte de la luz emitida es fosforescencia, ya que la excitación y por ende la fluorescencia suelen durar sólo una fracción de microsegundo. La **persistencia** del fósforo se define como el tiempo entre la eliminación de la excitación y el momento en que la fosforescencia ha disminuido al diez por ciento de la salida luminosa inicial. El intervalo de persistencia de diferentes fósforos puede ser de muchos segundos, pero en la mayoría de los fósforos utilizados en equipo gráfico es de 10 a 60 microsegundos. Esta salida luminosa decae exponencialmente con el tiempo. Las características del fósforo se detallan en [SHER93].

La **tasa de refrescamiento** de un TRC es el número de veces por segundo que se redibuja la imagen; generalmente es de 60 veces por segundo o mayor en las pantallas de barrido. Al disminuir la tasa de refrescamiento se presenta el **parpadeo**, ya que el ojo no puede integrar los impulsos luminosos individuales que provienen de los pixeles. La tasa de refrescamiento por encima de la cual deja de parpadear una imagen y se fusiona en una imagen estable se denomina **frecuencia de fusión crítica** o **CFF** (*critical fusion frequency*). El proceso de fusión es familiar para todos nosotros; ocurre cuando vemos televisión o filmes. Una imagen sin parpadeo aparece constante o fija para el observador aunque, de hecho, cualquier punto está *apagado* más tiempo que el que permanece *encendido*.

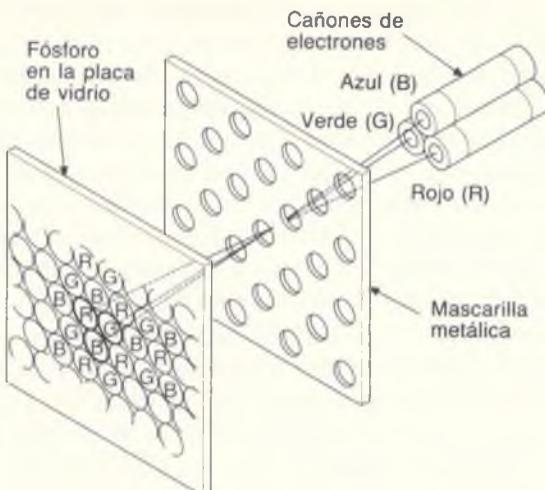
Un factor determinante de la CFF es la persistencia del fósforo: si es mayor la persistencia, menor será la CFF. La relación entre la frecuencia de fusión y la persistencia no es lineal: al duplicar la persistencia no se reduce a la mitad la CFF. Conforme la persistencia aumenta al intervalo de varios segundos, la frecuencia de fusión se reduce bastante. En el otro extremo, se puede usar incluso un fósforo sin persistencia, ya que en realidad el ojo sólo tiene que ver luz por un breve periodo, repetido con una frecuencia superior a la CFF.

La **tasa de rastreo horizontal** es el número de líneas de rastreo que puede presentar el circuito que dirige al CRT. Esta tasa es aproximadamente igual al producto de la tasa de refrescamiento y el número de líneas de rastreo. Para una tasa de rastreo determinada, un aumento en la tasa de refrescamiento representa una reducción en el número de líneas de rastreo.

El **ancho de banda** de un monitor tiene que ver con la velocidad a la cual se puede encender o apagar el cañón de electrones. Para lograr una resolución horizontal de  $n$  pixeles por línea de rastreo, debe ser posible encender el cañón de electrones al menos  $n/2$  veces y apagarlo otras  $n/2$  veces en una línea de rastreo, para crear líneas encendidas y apagadas alternantes. Considere un barrido de trama de 1000 líneas por 1000 pixeles, presentado con una tasa de refrescamiento de 60 Hz. Con unos sencillos cálculos se demuestra que el tiempo requerido para dibujar un pixel es el inverso de la cantidad ( $1000 \text{ pixeles/línea} \times 1000 \text{ líneas/cuadro} \times 60 \text{ cuadros/segundo}$ ), alrededor de 16 nanosegundos. En realidad, existe tiempo de procesamiento adicional en los ciclos de refrescamiento horizontal y vertical, de manera que un pixel se dibuja en unos 11 nanosegundos [WHIT84]. Entonces, el periodo de un ciclo de encendido-apagado es de unos 22 nanosegundos, lo que corresponde a una frecuencia de 45 MHz. Esta frecuencia corresponde al ancho de banda mínimo que se requiere para obtener una resolución de 1000 líneas (500 pares de líneas), pero no se trata del ancho de banda real ya que hemos ignorado el efecto del tamaño del punto. Es necesario compensar el tamaño distinto de cero del punto con un ancho de banda mayor, lo que ocasiona que el rayo se encienda y apague con mayor rapidez y los pixeles tengan orillas mejor definidas. No es raro que el ancho de banda real de un monitor de  $1000 \times 1000$  sea de 100 MHz.

Los televisores y las pantallas de barrido en color utilizan una forma de **TRC con mascarilla de sombra**, donde el interior de la superficie de vista del tubo se cubre con grupos de punto de fósforo rojo, verde y azul, poco espaciados entre sí. Los grupos de puntos son tan pequeños que el observador percibe la luz que emana de los puntos individuales como una mezcla de los tres colores. De esta manera, cada grupo puede producir una amplia gama de colores, dependiendo del nivel de excitación de cada punto de fósforo. Una mascarilla de sombra, que es una delgada placa metálica perforada con varios agujeros pequeños y montada cerca de la superficie de vista, se alinea con cuidado para que los tres rayos de electrones (uno para el rojo, otro para el verde y el otro para el azul) sólo puedan incidir en un tipo de punto de fósforo. De esta manera los puntos se pueden excitar en forma selectiva.

En la figura 4.6 se presenta uno de los TRC de mascarilla de sombra más comunes, un **TRC delta-delta**. Los puntos de fósforo se disponen en un patrón



**Figura 4.6** TRC triángulo-tríangulo con mascarilla de sombra. Los tres cañones y los puntos de fósforo están dispuestos en un patrón triangular. La mascarilla de sombra permite que los electrones de cada cañón golpeen los puntos de fósforo correspondientes.

triangular de **tríada**, al igual que los tres cañones de electrones. Los cañones se desvían en conjunto y se apuntan (convergen) al mismo punto de la superficie de visualización. La mascarilla de sombra tiene un pequeño agujero para cada triada, y estos agujeros se alinean en forma precisa con respecto a las triadas y a los cañones de electrones, de manera que cada punto de la triada esté expuesto a los electrones de un solo cañón. Los TRC triángulo-tríangulo de alta precisión son muy difíciles de mantener alineados. Un arreglo alternativo, el **TRC de triángulo de precisión en línea**, es más fácil de hacer convergir y de fabricar; en la actualidad es la tecnología preferida para la fabricación de monitores de alta resolución (1000 líneas de rastreo). Sin embargo, como el TRC triángulo-tríangulo ofrece mayor resolución, es muy probable que vuelva a surgir como tecnología dominante en la televisión de alta definición (*HDTV, high-definition television*). Aunque aún está en la etapa de investigación en el laboratorio, es probable que pronto sea comercialmente viable el **TRC en color de pantalla plana**, en el cual los rayos de electrones se mueven en forma paralela a la superficie de visualización y luego se hacen girar 90° para golpear la superficie.

La necesidad de la mascarilla de sombra y de las triadas impone un límite a la resolución de los TRC en color que no existe en los TRC monocromáticos. En los tubos de muy alta resolución, las triadas se colocan en centros de aproximadamente 0.21 milímetros; las triadas de los televisores caseros corresponden a centros de unos 0.60 milímetros [esta distancia también se conoce como **paso (pitch)** del tubo]. Como no se puede garantizar que un rayo finamente enfocado golpee exactamente en el centro de un agujero de la mascarilla de sombra, el

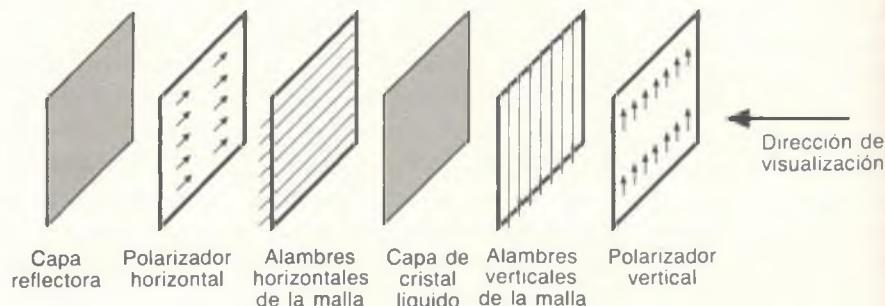
diámetro del rayo (definido como el diámetro cuando la intensidad es el 50 por ciento del máximo) debe ser de alrededor de  $7/4$  del paso. Por consiguiente, en una mascarilla con paso de 0.25 milímetros (0.01 pulg), el rayo tiene aproximadamente 0.018 pulgadas de diámetro y la resolución no puede ser mayor que  $1/0.018 = 55$  líneas por pulgada. En un monitor de 19 pulgadas (medida diagonal) con un paso de 0.25 milímetros, cuyas dimensiones son aproximadamente 15.5 pulgadas de ancho por 11.6 pulgadas de alto [CONR85], la resolución alcanzable es únicamente  $15.5 \times 55 = 850$  por  $11.6 \times 55 = 638$ . Este valor es comparable con la capacidad de direccionamiento típico de  $1280 \times 1024$  o de  $1024 \times 800$ . Como se ilustra en la figura 4.2, es conveniente una resolución algo menor que la capacidad de direccionamiento.

La mayoría de los TRC de alta calidad con mascarilla de sombra tienen medidas diagonales de 15 a 21 pulgadas, con vidrios ligeramente curvos que crean distorsiones ópticas para el observador. En la actualidad comienzan a estar disponibles varios tipos de TRC con pantalla plana.

Una **pantalla de cristal líquido (LCD, liquid-crystal display)** se compone de seis capas, como se muestra en la figura 4.7. La capa frontal es una placa polarizadora vertical. La siguiente es una capa con delgados alambres en malla depositados eléctricamente sobre la superficie, adjuntos a los cristales. La siguiente es una delgada capa (alrededor de 0.0005 pulgadas) de cristal líquido, después sigue una capa con los alambres horizontales de la malla sobre la superficie próxima a los cristales, después un polarizador horizontal y por último un reflector.

El material de cristal líquido está formado por largas moléculas cristalinas. Las moléculas individuales normalmente se disponen en forma de espiral, para que la dirección de polarización de la luz gire  $90^\circ$ . La luz que entra por la capa frontal se polariza en forma vertical. Cuando la luz pasa por el cristal líquido, la polarización gira  $90^\circ$  y queda horizontal, de manera que la luz pasa ahora por el polarizador horizontal posterior, se refleja y regresa por los dos polarizadores y el cristal.

Cuando los cristales se hallan en un campo eléctrico, todos se alinean en la misma dirección y por ende no tienen efecto polarizador. Así, los cristales en el



**Figura 4.7** Capas de una pantalla de cristal líquido (LCD), que se unen para formar un pequeño tablero.

campo eléctrico no cambian la polarización de la luz transmitida, la luz permanece polarizada verticalmente y no pasa por el polarizador posterior. La luz se absorbe y el observador ve un punto oscuro en la pantalla.

Un punto negro en  $(x_1, y_1)$  se crea con el direccionamiento de una matriz. El punto se selecciona aplicando un voltaje negativo  $-V$  al alambre horizontal de la malla  $x_1$  y un voltaje positivo  $+V$  al alambre vertical de la malla  $y_1$ .  $-V$  y  $+V$  no tienen la magnitud suficiente para ocasionar el alineamiento de los cristales, pero su diferencia sí es suficiente para hacerlo. Con ello, los cristales en  $(x_1, y_1)$  ya no hacen girar la dirección de polarización de la luz transmitida, de manera que ésta permanece polarizada de manera vertical y no pasa por el polarizador posterior. Así, la luz se absorbe y el observador contempla un punto oscuro en la pantalla.

Las **pantallas LCD de matriz activa** tienen un transistor en cada punto  $(x_1, y_1)$  de la malla. Los transistores se emplean para ocasionar un rápido cambio de estado de los cristales y para controlar el grado de cambio. Estas dos propiedades permiten usar las LCD en televisores miniatura con imágenes de tono continuo. También se pueden entintar los cristales para proporcionar color. Lo más importante es que el transistor puede servir como memoria del estado de la celda y mantener la celda en dicho estado hasta que cambie. Es decir, la memoria ofrecida por el transistor permite que una celda permanezca todo el tiempo encendida y por consiguiente sea más brillante que si tuviera que refrescarse de manera periódica. Se han construido pantallas LCD en color con resolución de  $800 \times 1000$  en una pantalla diagonal de 14 pulgadas.

Las ventajas de las pantallas LCD son su bajo costo, poco peso, tamaño pequeño y poco consumo de energía. En el pasado, su desventaja principal residía en que las LCD eran pasivas, ya que reflejaban sólo la luz incidente y no creaban su propia luz (aunque esta situación se puede corregir con retroiluminación): cualquier reflejo desvanecía la imagen. En años recientes, la utilización de pantallas activas ha eliminado esta preocupación. De hecho, los computadores portátiles con pantalla en color, que no estaban disponibles hasta hace poco tiempo, usan tecnología LCD tanto activa como pasiva. Además, como las pantallas LCD son pequeñas y ligeras, se pueden usar en pantallas montadas en la cabeza, como las que se analizan en la sección 8.1.6. Conforme las pantallas LCD a color aumentan en tamaño y disminuyen en costo, llegarán a constituir una amenaza para el dominio del TRC de color, aunque no por mucho tiempo.

Las **pantallas electroluminiscentes (EL)** consisten en la misma estructura tipo malla que se usa en las pantallas LCD y de plasma. Entre la pantalla anterior y la posterior se encuentra una delgada capa (de unos 500 nanómetros) de material electroluminiscente, como sulfuro de zinc contaminado con manganeso, que emite luz cuando se encuentra en un campo eléctrico intenso (unos 106 volts por centímetro). Un punto en la pantalla se ilumina usando el método de direccionamiento de matriz, con varios cientos de volts colocados en las líneas de selección horizontales y verticales. También existen pantallas electroluminiscentes en color.

Estas pantallas son brillantes y se pueden encender y apagar con rapidez; además, en cada pixel se pueden emplear transistores para almacenar la imagen. El tamaño típico de la pantalla es de 6 por 8 pulgadas hasta 12 por 16 pulgadas,

con 70 puntos direccionables por pulgada. La mayor desventaja de estas pantallas es que su consumo de energía es mayor que el de una pantalla LCD. Si embargo, gracias a su brillantez se han utilizado en algunos computadores personales.

La mayoría de las pantallas de gran tamaño utiliza algún tipo de **TRC de proyección**, donde la luz de un TRC monocromático pequeño (de unas cuantas pulgadas de diámetro) pero muy intenso se magnifica y proyecta a través de un espejo curvo. Los sistemas en color utilizan tres proyectores con filtros rojo, verde y azul. Un TRC con mascarilla de sombra no crea luz suficiente para proyectarse en una pantalla grande (de unos dos metros de diagonal).

El sistema de proyección de **válvulas de luz** de General Electric se emplea en pantallas muy grandes, para las cuales no bastaría la salida luminosa del TRC de proyección. Una válvula de luz es precisamente lo que implica su nombre: un mecanismo para controlar la cantidad de luz que pasa por la válvula. La fuente luminosa puede tener una intensidad mucho mayor que un TRC. En el método más común, un cañón de electrones traza una imagen en una delgada película de aceite sobre un pedazo de vidrio. La carga del electrón hace que la película cambie de grosor. La luz de la fuente de alta intensidad se dirige al vidrio y se refracta en distintas direcciones debido a la variación en el grosor de la película de aceite. Se emplean entonces dispositivos ópticos especiales que proyectan sobre la pantalla la luz refractada en varias direcciones, y no se proyecta cualquier otra luz. Es posible obtener color con estos sistemas si se emplean tres proyectores o un conjunto de dispositivos ópticos más complejo con un solo proyector. Se proporcionan más detalles al respecto en [SHER93].

Tabla 4.2

## Comparación de tecnologías de pantallas

Propiedad	TRC	Electro-luminiscente	Cristal líquido
Consumo de energía	regular	regular-bueno	excelente
Tamaño de pantalla	excelente	bueno	regular
Profundidad	mala	excelente	excelente
Peso	mal	excelente	excelente
Solidez	regular-buena	bueno-excelente	excelente
Brillo	excelente	excelente	regular-bueno
Direccionamiento	bueno-excelente	bueno	regular-bueno
Contraste	bueno-excelente	bueno	regular
Niveles de intensidad por punto	excelentes	regulares	regulares
Ángulo de visión	excelente	bueno	mal
Capacidad de colores	excelente	buen	buen
Costo relativo	bajo	mediano-alto	bajo

En la tabla 4.2 se resumen las características de las tres tecnologías principales de pantallas. Sin embargo, la tecnología avanza con tanta rapidez que es probable que algunas de las relaciones cambien en los próximos años. Observe

además que las comparaciones del método de cristal líquido son para el direccionamiento pasivo; si se emplea direccionamiento de matriz activa, pueden lograrse niveles de grises y colores.

Se presenta información más detallada acerca de estas tecnologías de pantalla en [APT85; BALD85; CONR85; PERR85; SHER93; y TANN85].

### 3 Sistemas de presentación por barrido de trama

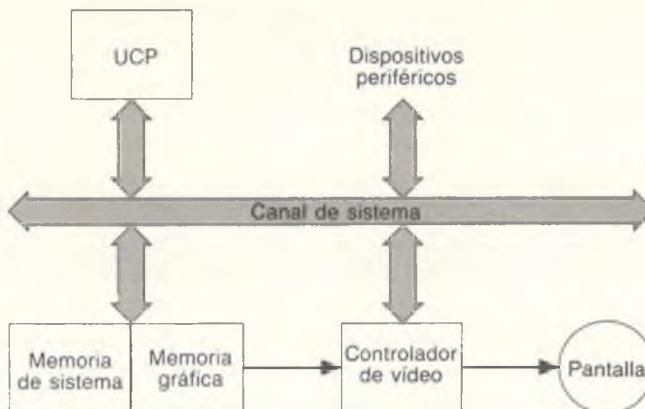
Los conceptos básicos de los sistemas gráficos de barrido se presentaron en el capítulo 1, mientras que en el capítulo 2 se mostraron más detalles acerca de los tipos de operaciones posibles con una pantalla de barrido. En esta sección analizaremos los diversos elementos de una pantalla de barrido, haciendo hincapié en dos aspectos fundamentales que marcan la diferencia entre varios tipos de sistemas de barrido.

En primer lugar, la mayoría de las pantallas de barrido tiene hardware especializado que le ayuda en la discretización de las primitivas de salida para obtener el arreglo bidimensional (mapa) de pixeles ( *pixmap*), así como para realizar las operaciones de barrido consistentes en mover, copiar y modificar pixeles o bloques de pixeles. Este hardware se conoce como **procesador de pantalla gráfica**. La diferencia fundamental entre los sistemas de presentación es cuánto hace el procesador de pantalla comparado con lo que debe ejecutar el paquete de subrutinas gráficas que se ejecuta en la UCP de propósito general, la cual alimenta a la pantalla de barrido. Observe que el procesador de pantalla gráfica también se conoce como **controlador gráfico** (para recalcar su similitud con las unidades de control de otros dispositivos periféricos) o **coprocesador de pantalla**. El segundo punto de distinción en los sistemas de barrido es la relación entre el mapa de pixeles y el espacio de direcciones de la memoria de computador de propósito general, es decir, si el mapa de pixeles forma parte de la memoria del computador o está separado.

En la sección 4.3.1 presentaremos una pantalla de barrido sencilla, que consiste en una UCP con el mapa de pixeles como parte de su memoria, y un controlador de vídeo que alimenta a un TRC. No hay controlador gráfico, de manera que la UCP realiza tanto el trabajo gráfico como el de aplicación. En la sección 4.3.2 se presenta un procesador gráfico con mapa de pixeles por separado; en la sección 4.3.3 se analizan múltiples aspectos funcionales del procesador gráfico. Por último, en la sección 4.3.4 se analizan las formas en que el mapa de pixeles se puede integrar al espacio de direcciones de la UCP, dada la existencia de un procesador gráfico.

#### 4.3.1 Sistema sencillo de pantalla de barrido

*La más común y sencilla forma de organizar un sistema de pantalla de barrido es la que se ilustra en la figura 4.8. La relación entre la memoria y la UCP es*



**Figura 4.8** Arquitectura de un sistema común de pantalla de barrido. Una porción dedicada de la memoria de sistema tiene puerto dual, de manera que el controlador de video puede tener acceso directo sin aumentar la carga del canal de sistema.

exactamente la misma que en un sistema de computación no gráfico. Sin embargo, una porción de la memoria también sirve como mapa de pixeles. El controlador de video presenta la imagen definida en la memoria gráfica, accediendo a la memoria a través de un puerto de acceso separado con la frecuencia determinada por la tasa de barrido de trama. En muchos sistemas, una parte de la memoria se asigna en forma permanente a la memoria gráfica, mientras que en otros hay áreas de memoria intercambiables (en ocasiones llamadas páginas en el mundo de los computadores personales). Otros sistemas pueden designar (por medio de un registro) cualquier parte de la memoria como memoria gráfica.

El programa de aplicación y el paquete de subrutinas gráficas comparten la memoria del sistema y son ejecutados por la UCP. El paquete gráfico incluye procedimientos de discretización, de manera que, por ejemplo, si el programa de aplicación hace la llamada SRGP\_lineCoord ( $x_1, y_1, x_2, y_2$ ), el paquete gráfico puede asignar los pixeles apropiados en la memoria gráfica (los detalles de los procedimientos de discretización se proporcionan en el capítulo 3). La memoria gráfica se encuentra en el espacio de direcciones de la UCP, por lo que el paquete gráfico puede tener un fácil acceso a ella para asignar pixeles e implantar las instrucciones de copiado de pixeles descritas en el capítulo 2.

El controlador de video cicla a través de la memoria gráfica, una línea de barrido a la vez, por lo general 60 veces por segundo. Las direcciones de referencia de la memoria se generan en sincronía con el barrido de trama, y el contenido de la memoria se utiliza para controlar la intensidad o el color del rayo del CRT. El controlador de video se organiza en la forma ilustrada en la figura 4.9. El generador de barrido de trama produce señales de desviación que generan el barrido de trama; controla además los registros de direcciones X y Y, que a su vez definen la siguiente localidad de memoria a la que se tendrá acceso.



Figura 4.9 Organización lógica del controlador de vídeo.

Suponga que la memoria gráfica se direcciona en  $x$  de 0 a  $x_{\max}$  y en  $y$  de 0 a  $y_{\max}$ ; entonces, al iniciar un ciclo de refrescamiento, el registro de dirección X se asigna igual a cero y el registro Y se asigna igual a  $y_{\max}$  (la línea de rastreo superior). Al generar la primera línea de rastreo, X se va incrementando hasta  $x_{\max}$ . Se obtiene el valor de cada pixel y se usa para controlar la intensidad del rayo del TRC. Después de la primera línea de rastreo, la dirección X se restablece en cero y la dirección Y se reduce en uno. El proceso continúa hasta que se genera la última línea de rastreo ( $y = 0$ ).

En esta situación simplificada, se hace un acceso a la memoria gráfica por cada pixel que se dibujará. En una pantalla de alta resolución de 1000 pixeles por 1000 líneas refrescada 60 veces por segundo, una forma sencilla de estimar el tiempo disponible para el dibujo de un pixel de un bit es calculando  $1/(1000 \times 1000 \times 60) = 16$  nanosegundos. En este cálculo se ignora el hecho de que no se dibujan pixeles durante el retorno horizontal y vertical.<sup>1</sup> Sin embargo, las pastillas de memoria RAM tienen tiempos de ciclo de unos 80 nanosegundos: ¡no pueden permitir un acceso cada 16 nanosegundos! Por lo tanto, el controlador de vídeo debe leer varios valores de pixeles en un ciclo de memoria. En nuestro ejemplo, el controlador podría leer 16 bits en un ciclo de memoria, con lo cual el tiempo de refrescamiento sería de  $16 \text{ pixeles} \times 16 \text{ ns/pixel} = 256$  nanosegundos. Los 16 bits se cargan en un registro del controlador de vídeo, para luego desplazarse y controlar la intensidad de rayo del TRC, un bit cada 16 nanosegundos. En los 256 nanosegundos que requiere esta operación, hay tiempo

<sup>1</sup> En un sistema de barrido de trama hay cierta cantidad de tiempo durante la cual no se traza ninguna imagen: el tiempo de retorno horizontal, que ocurre una vez por línea de rastreo, y el tiempo de retorno vertical, que ocurre una vez por cuadro.

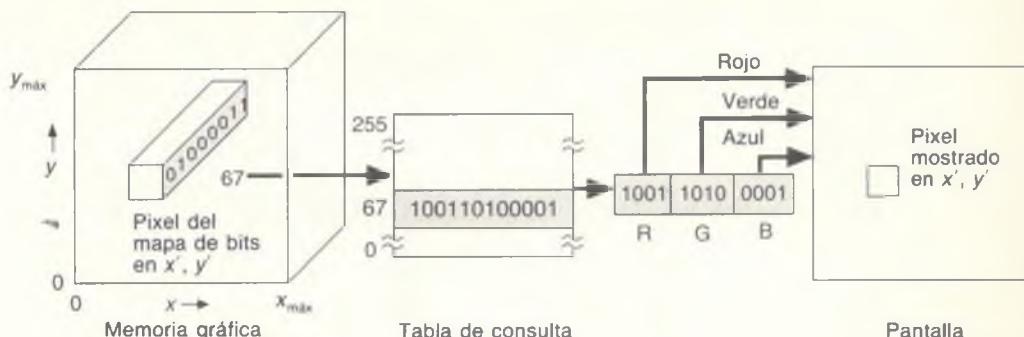
para unos tres ciclos de memoria: uno para el controlador de vídeo y dos para la UCP. Este compartimiento puede forzar a la UCP a esperar para tener acceso a la memoria, lo cual puede reducir proporcionalmente la velocidad de la UCP. Por supuesto, esta situación puede mejorar si se emplea memoria caché en la UCP. Otra estrategia es usar organizaciones no tradicionales para las memorias gráficas. Por ejemplo, el encendido de todos los píxeles de una línea de barrido en un mismo tiempo de acceso reduce el número de ciclos de memoria necesarios para la discretización, sobre todo en áreas rellenas. La organización de RAM para vídeo (VRAM), desarrollada por Texas Instruments, puede leer todos los píxeles de una línea de rastreo en un ciclo, con lo que se reduce la cantidad de ciclos de memoria necesarios para refrescar la pantalla.

Hasta ahora hemos supuesto mapas de bits monocromáticos, con un bit por pixel. Esta suposición es válida para algunas aplicaciones, pero no para otras. El control adicional sobre la intensidad de cada pixel se obtiene almacenando varios bits para cada pixel: dos bits producen cuatro intensidades, etcétera. Los bits se pueden utilizar para controlar no sólo la intensidad, sino también el color. ¿Cuántos bits por pixel se requieren para que una imagen almacenada se perciba con tonos continuos de grises? Por lo general bastan cinco o seis bits, pero pueden requerirse ocho o más. Por lo tanto, en el caso de pantallas a color, un argumento algo simplificado nos sugiere que se necesitan tres veces más bits, o sea, ocho para cada uno de los tres colores primarios aditivos: rojo, azul y verde.

Aunque los sistemas con 24 bits por pixel son relativamente baratos, muchas aplicaciones a color no requieren  $2^{24}$  colores diferentes en la misma imagen (la cual tiene entre  $2^{18}$  y  $2^{20}$  píxeles). Además, muchas veces es necesario que para una imagen o aplicación se tengan unos cuantos colores y que además se cuente con la capacidad para cambiar colores de una imagen o aplicación a otra. Así mismo, en muchas aplicaciones de análisis de imágenes y de mejora de imágenes es deseable cambiar la apariencia visual de la imagen sin cambiar los datos subyacentes que la definen, por ejemplo, para mostrar como negros todos los píxeles cuyo valor esté por debajo de un valor límite, ampliar la gama de intensidades o crear una presentación en seudocolor de una imagen monocromática.

Por todas estas razones, el controlador de vídeo de las pantallas de trama muchas veces incluye una **tabla de consulta de vídeo** [también llamada **tabla de consulta** o **LUT** (*look-up table*)]. La tabla de consulta tiene tantas entradas como valores de píxeles existen. El valor de un pixel no se utiliza para controlar directamente el rayo, pero sí como índice en la tabla de consulta. El valor de la entrada en la tabla se emplea para controlar la intensidad o el color del CRT. Por ejemplo, un pixel con valor 67 ocasionaría el acceso a la entrada 67 de la tabla y usará su contenido para controlar el rayo del CRT. Esta operación de consulta se lleva a cabo para cada pixel en el ciclo de presentación, de manera que el acceso a la tabla debe ser rápido y la UCP debe ser capaz de cargar la tabla de consulta cuando lo indique un mandato de programa.

En la figura 4.10, la tabla de consulta se ubica entre la memoria gráfica y la pantalla CRT. La memoria gráfica tiene ocho bits por pixel y por ende la tabla de consulta tiene 256 entradas.



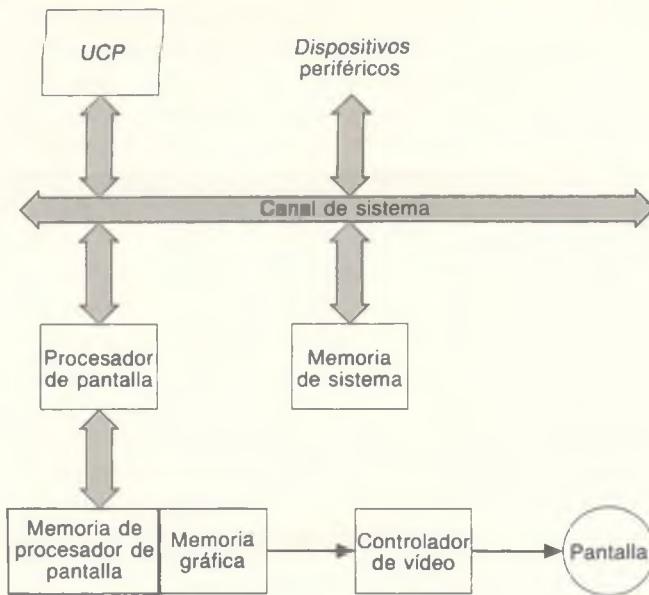
**Figura 4.10** Organización de una tabla de consulta de video. Un pixel con valor 67 (01000011 en binario) se presenta en la pantalla con el cañón de electrones rojo a 9/15 de su máximo, el verde a 9/15 y el azul a 1/15. Esta tabla de consulta se presenta con 12 bits por entrada. Son comunes las tablas con 24 bits.

En muchos computadores personales de bajo costo se emplea la sencilla organización de pantalla de barrido presentada en la figura 4.8. Este tipo de sistema es barato de construir, pero presenta muchas desventajas. En primer lugar, la discretización por software es lenta. Por ejemplo, hay que calcular la dirección ( $x$ ,  $y$ ) de cada pixel en una línea y luego traducirla a una dirección de memoria que consiste en un par formado por un byte y un bit en un byte. Aunque los pasos individuales son sencillos, cada uno se repite varias veces. La discretización basada en software reduce el ritmo global de interacción entre el usuario y la aplicación, lo cual puede causar insatisfacción en el usuario.

La segunda desventaja de esta arquitectura es que al aumentar el número de pixeles o la tasa de refrescamiento de la pantalla, también crece el número de accesos a memoria que realiza el controlador de video y por ende disminuye el número de ciclos de memoria disponibles para la UCP. Ésta se vuelve entonces más lenta, sobre todo en una arquitectura en la cual el acceso a la memoria gráfica debe efectuarse a través del canal de sistema. Si usamos puerto doble en la parte de memoria de sistema de la figura 4.8, la lentitud se presenta únicamente cuando la UCP tiene acceso a la memoria gráfica, por lo general durante las operaciones de rastreo o de barrido. Estas dos desventajas deben ponderarse con respecto a la facilidad que tiene la UCP para acceder a la memoria gráfica y con la sencillez de la arquitectura del sistema.

#### 4.3.2 Sistema de presentación de barrido con procesador periférico de dibujo

El sistema de presentación de barrido con un procesador de pantalla periférico es una arquitectura común (véase la Fig. 4.11) que evita las desventajas de la pantalla de barrido simple, al introducir un procesador gráfico aparte para manejar las funciones gráficas, como la discretización y las operaciones de barrido,



**Figura 4.11** Arquitectura de sistema de barrido con procesador de dibujo periférico.

y una memoria gráfica aparte para el refrescamiento de la imagen. Ahora tenemos dos procesadores: la UCP de propósito general y el procesador de dibujo de propósito específico. También tenemos tres áreas de memoria: la memoria de sistema, la del procesador de dibujo y la memoria gráfica. La memoria de sistema sólo almacena los datos y los programas que se ejecutan en la UCP: el programa de aplicación, el paquete gráfico y el sistema operativo. En forma similar, la memoria del procesador de dibujo contiene los datos y los programas que realizan operaciones de trama y de discretización. La memoria gráfica contiene la imagen presentable creada con las operaciones de discretización y de barrido.

En los casos sencillos, el procesador de dibujo puede consistir en lógica especializada para establecer la correspondencia entre las coordenadas bidimensionales ( $x, y$ ) y una dirección lineal de la memoria. En este caso, la UCP sigue efectuando las operaciones de barrido y de discretización, y no se requiere la memoria del procesador de dibujo; únicamente está presente la memoria gráfica. La mayoría de los procesadores de pantalla periféricos también realizan la discretización.

En esta sección presentamos un sistema prototípico cuyas características son una combinación (en ocasiones simplificada) de muchos sistemas típicos comerciales, como la tarjeta gráfica utilizada en los computadores compatibles con IBM PC.

La memoria gráfica es de  $1024 \times 1024 \times 8$  bits por pixel y existe una tabla de consulta de 256 entradas y 12 bits, cuatro para cada uno de los colores rojo,

verde y azul. El origen está en la esquina inferior izquierda, pero sólo se presentan las primeras 768 filas del arreglo bidimensional de pixeles ( $y$  en el intervalo 0 a 767). La pantalla tiene seis registros de estado, los cuales se asignan con diversas instrucciones y afectan la ejecución de otras instrucciones. Estos registros son CP (posición actual, formado por los registros de posición X y Y), FILL (rellenar), INDEX (índice), WMODE (modo de escritura), MASK (máscara) y PATTERN (patrón). A continuación se explica la forma en que operan.

Algunas de las instrucciones para la pantalla de barrido simple son las siguientes:

**Move ( $x, y$ )** Los registros X y Y que definen la posición actual (CP) se asignan iguales a  $x$  y  $y$ . Como el mapa de pixeles es de  $1024 \times 1024$ ,  $x$  y  $y$  deben estar entre 0 y 1023.

**MoveR ( $dx, dy$ )** Los valores  $dx$  y  $dy$  se suman a los registros X y Y para definir una nueva posición actual. Los valores de  $dx$  y  $dy$  deben estar entre  $-1024$  y  $+1023$  y se representan en notación de complemento a dos. Esta suma puede causar sobreflujo y por ende un desplazamiento circular de los valores de los registros X y Y.

**Line ( $x, y$ )** Se dibuja una línea de CP a  $(x, y)$  y esta posición se convierte en la actual.

**LineR ( $dx, dy$ )** Se dibuja una línea de CP a  $CP + (dx, dy)$  y esta posición se convierte en la actual.

**Point ( $x, y$ )** Se asigna el pixel en  $(x, y)$  y esta posición se convierte en la actual.

**PointR ( $dx, dy$ )** Se asigna el pixel en  $CP + (dx, dy)$  y esta posición se convierte en la actual.

**Rect ( $x, y$ )** Se dibuja un rectángulo entre CP y  $(x, y)$ . La posición actual no cambia.

**RectR ( $dx, dy$ )** Se dibuja un rectángulo entre CP y  $CP + (dx, dy)$ . El parámetro  $dx$  se puede considerar como la anchura del rectángulo y  $dy$  como su altura. La posición actual no cambia.

**Text ( $n$ , dirección)** Se presentan los  $n$  caracteres en la localidad de memoria dirección, comenzando en la posición actual CP. Los caracteres se definen en una malla de 7 por 9 pixeles, con dos pixeles adicionales de espaciado vertical y horizontal para separar caracteres y líneas. La CP se actualiza a la esquina inferior izquierda del área donde se presentaría el carácter  $n + 1$ .

**Circle ( $radio$ )** Se dibuja un círculo centrado en CP. La posición actual no cambia.

**Polygon ( $n$ , dirección)** En la dirección se almacena una lista de vértices  $(x_1, y_1, x_2, y_2, x_3, y_3, \dots, x_n, y_n)$ . Se dibuja un polígono que comienza en  $(x_1, y_1)$ , pasa por todos los vértices hasta  $(x_n, y_n)$  y luego regresa a  $(x_1, y_1)$ . La posición actual no cambia.

**AreaFill (bandera)** La bandera se usa para asignar la bandera FILL en la pantalla de barrido. Cuando la bandera está encendida (con un valor de la bandera distinto de cero), se rellenan todas las áreas creadas con los mandatos Rect, RectR, Circle, CircleSector y Polygon usando el patrón definido con el mandato Pattern.

**RasterOp ( $dx, dy, destx, desty$ )** Una región rectangular de la memoria gráfica, de CP a CP +  $(dx, dy)$ , se combina con la región del mismo tamaño cuyo vértice inferior izquierdo se encuentra en  $(destx, desty)$ , sobre escribiendo esa región. La combinación se controla con el registro WMODE.

Los mandatos y los datos intermedios se transfieren al procesador de dibujo por medio de una memoria intermedia (o sea, una cola) “primero en entrar, primero en salir” (**PEPS**) que se encuentra en una porción dedicada del espacio de direcciones de la UCP. El paquete gráfico coloca mandatos en la cola y la pantalla los lee y ejecuta. En otras localidades específicas de la memoria hay apuntadores al inicio y al final de la memoria intermedia, accesibles para la UCP y la pantalla. El apuntador al inicio de la memoria intermedia es modificado por el procesador de dibujo cada vez que se saca un byte; el apuntador al final de la memoria intermedia es modificado por la UCP cuando se introduce un byte. Se realizan pruebas apropiadas para asegurar que no se lea una memoria intermedia vacía ni se escriba en una memoria intermedia llena. La lectura de los datos direccionados para las instrucciones se efectúa con acceso directo a memoria.

Para pasar mandatos es más atractiva una cola que un registro o una localidad de instrucciones a la cual pueda tener acceso la pantalla. En primer lugar, la longitud variable de las instrucciones favorece el concepto de cola. Segundo, la UCP se puede adelantar a la pantalla y colocar varios mandatos en la cola. Cuando la UCP ha terminado de emitir mandatos para el procesador de dibujo, puede efectuar otras tareas mientras el procesador de dibujo vacía la cola.

La programación del procesador de dibujo es similar al usar el paquete SRGP del capítulo 2. En el capítulo 4 de [FOLE90] se presentan varios ejemplos de programación.

#### 4.3.3 Funcionalidad adicional del procesador de dibujo

Nuestro sencillo procesador de dibujo sólo lleva a cabo algunas de las operaciones gráficas que se pueden implantar. La tentación que se presenta al diseñador de sistemas es descargar la UCP principal y añadir más funcionalidad al procesador de dibujo, por ejemplo usando memoria local para almacenar listas de

instrucciones de dibujo, efectuando recortes y transformaciones ventana-área de vista, y quizás proporcionando lógica de correlación de selección y realimentación automática cuando se selecciona un elemento gráfico. A final de cuentas, el procesador de dibujo se convierte en otra UCP de propósito general que lleva a cabo trabajo gráfico interactivo general, de manera que el diseñador se ve nuevamente tentado a suministrar hardware de propósito especial que reduzca la carga del procesador gráfico.

Esta **rueda de reencarnación** fue identificada por Myer y Sutherland en 1968 [MYER68]. El aspecto que querían destacar es que existe una conciliación entre la funcionalidad de propósito especial y la de propósito general. El hardware de propósito especial suele realizar el trabajo con mayor rapidez que un procesador de propósito general. Sin embargo, el hardware de propósito especial es más costoso y no se puede usar con otros fines. Esto constituye un tema siempre presente en el diseño de sistemas gráficos.

Si se añaden recortes (Cap. 3) al procesador de dibujo, entonces es posible especificar las primitivas de salida en coordenadas distintas de las del dispositivo. Por ejemplo, la especificación se puede efectuar en coordenadas de punto flotante, aunque algunos procesadores de pantalla sólo operan con enteros (esta situación está cambiando rápidamente, con la disponibilidad de pastillas (*chips*) de punto flotante de bajo costo). Si sólo se emplean enteros, las coordenadas utilizadas por el programa de aplicación deben ser enteras, o el paquete gráfico debe establecer una correspondencia entre las coordenadas de punto flotante y las enteras. Para que esta correspondencia sea posible, el programa de aplicación debe proporcionar al paquete gráfico un rectángulo que contenga las coordenadas de todas las primitivas de salida especificadas en el paquete. Se establece una correspondencia entre este rectángulo y el máximo intervalo de enteros, de manera que todo lo que esté en el rectángulo también se encuentre en el intervalo de coordenadas enteras.

Si el paquete de subrutinas es tridimensional, entonces el procesador de dibujo puede efectuar recortes y transformaciones geométricas tridimensionales más complejas (Caps. 5 y 6). Además, si el paquete incluye primitivas de superficies tridimensionales, como áreas poligonales, el procesador de dibujo también puede realizar la determinación de superficies visibles y los pasos de presentación analizados en los capítulos 13 y 14. En el capítulo 18 de [FOLE90] se analizan algunos de los métodos fundamentales que se usan para organizar pastillas VLSI de propósito general y especial de manera que lleven a cabo estos pasos con rapidez. Muchas pantallas comerciales ofrecen estas características.

Otra función que con frecuencia se añade al procesador de dibujo es el almacenamiento local de segmentos, también conocido como **almacenamiento de lista de dibujo**. Las instrucciones de dibujo, agrupadas en segmentos con nombre y con coordenadas enteras no recortadas, se almacenan en la memoria del procesador de dibujo para permitir que el procesador opere con mayor autonomía respecto a la UCP.

¿Qué puede hacer exactamente un procesador de dibujo con estos segmentos almacenados? Puede transformarlos y redibujarlos, por ejemplo en

acercamientos/alejamientos o en desplazamiento. Se pueden jalar localmente los segmentos a nuevas posiciones. Se puede implantar la selección local si el procesador de dibujo compara la posición del cursor con todas las primitivas gráficas (en el capítulo 7 se analizan formas más eficaces de hacerlo). Con el almacenamiento de segmentos también se puede efectuar la regeneración necesaria para llenar los huecos que se crean al eliminar un segmento. Los segmentos se pueden crear, eliminar, editar y hacer visibles o invisibles.

Los segmentos también se pueden copiar o pueden aparecer como referencias, reduciendo así la cantidad de información que debe enviarse de la UCP al procesador de pantalla y ahorrando espacio de almacenamiento en el propio procesador de pantalla. Es posible construir una estructura de datos jerárquica compleja con esta capacidad, y de hecho una gran cantidad de procesadores de dibujo comerciales con memoria local de segmentos pueden copiar o hacer referencia a segmentos. Cuando se presentan los segmentos, toda referencia a otro segmento debe ir precedida por el almacenamiento del estado actual del procesador de pantalla, así como la llamada a una subrutina va precedida por el almacenamiento del estado actual de la UCP. Las referencias se pueden anidar, con lo cual surge un **archivo estructurado de dibujo** o una **lista jerárquica de dibujo**, como en PHIGS [ANSI88], que se analiza con mayor detalle en el capítulo 7.

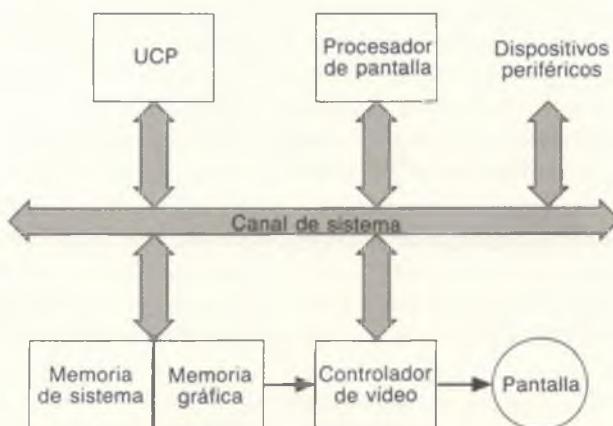
Esta arquitectura de sistema de pantalla de barrido, con su pantalla gráfica y su memoria gráfica separada, tiene muchas ventajas con respecto al sencillo sistema de pantalla de barrido de la sección 4.3.1, pero también tiene sus desventajas. Si el procesador de pantalla es accedido por la UCP como periférico en un puerto de acceso directo a memoria, hay un tiempo de procesamiento adicional considerable por parte del sistema operativo cada vez que se pasa una instrucción (esta situación no se presenta en un procesador de pantalla cuyo registro de instrucciones tenga correspondencia de memoria con el espacio de direcciones de la UCP, pues es fácil que el paquete gráfico asigne los registros en forma directa).

El mandato de operación de barrido es especialmente difícil. En teoría, debe tener cuatro pares fuente-destino posibles: memoria de sistema a memoria de sistema, memoria de sistema a memoria gráfica, memoria gráfica a memoria de sistema y memoria gráfica a memoria gráfica (en este caso se considera que la memoria gráfica y la memoria del procesador de pantalla de la figura 4.11 son idénticas, ya que se encuentran en el mismo espacio de direcciones). Sin embargo, en los sistemas de procesador de pantalla los diferentes pares fuente-destino se manejan en distintas formas, y es posible que no exista el caso de memoria de sistema a memoria de sistema. Esta falta de simetría complica la tarea del programador y reduce la flexibilidad. Por ejemplo, si la porción fuera de pantalla de un mapa de pixeles se rellena con menús, familias tipográficas, etc., es difícil usar la memoria principal como área de sobreflujo. Además, como el uso de mapas de pixeles es tan extenso, no es posible omitir el apoyo a las operaciones de barrido con mapas de pixeles almacenados en memoria principal.

El procesador de dibujo que definimos antes en esta sección, como la mayoría de los procesadores de dibujo reales, mueve imágenes de barrido entre la memoria de sistema y la memoria gráfica por medio de transferencias de E/S en el canal de sistema. Por desgracia, este movimiento puede ser demasiado lento para las operaciones en tiempo real, como son animación, arrastre, presentación de ventanas y menús. El problema lo constituyen el tiempo que requiere el sistema operativo para iniciar las transferencias y la tasa de transferencia del canal. Esta situación puede mejorar en parte si se aumenta la memoria del procesador de dibujo para que almacene más mapas de pixeles fuera de pantalla, pero aun con esto la memoria no es suficiente para otros fines; ¡de por sí casi nunca hay suficiente memoria!

#### 4.3.4 Sistema de dibujo de barrido con procesador de dibujo integrado

Podemos compensar muchas de las desventajas del procesador de dibujo periférico que analizamos en la sección anterior si dedicamos una porción especial de la memoria de sistema para que sea la memoria gráfica y proporcionamos un segundo puerto para el acceso del controlador de vídeo a la memoria gráfica, creando así la arquitectura de espacio único de direcciones (SAS, *single-address-space*) que se ilustra en la figura 4.12. Aquí el procesador de dibujo, la UCP y el controlador de vídeo se encuentran en el canal de sistema y por lo tanto todos tienen acceso a la memoria del sistema. El origen y, en algunos casos, el tamaño de la memoria gráfica se conservan en registros, de manera que para obtener la



**Figura 4.12** Arquitectura de sistema de dibujo de barrido con espacio único de direcciones (SAS) y procesador de dibujo integrado. El procesador de dibujo puede tener memoria privada para algoritmos y almacenamiento temporal. Una parte dedicada de la memoria del sistema tiene puerto doble para que el controlador de vídeo pueda tener acceso directo a ella, sin carga para el canal de sistema.

doble memoria basta volver a cargar el registro de origen; los resultados de la discretización se pueden colocar en la memoria gráfica para presentarse de inmediato o en algún otro lugar de la memoria del sistema y usarse posteriormente. En forma parecida, la fuente y el destino para las operaciones de barrido efectuadas por el procesador de dibujo pueden estar en cualquier parte de la memoria del sistema (la única memoria que nos interesa ahora). Esta disposición es atractiva porque la UCP puede manipular directamente los pixeles en la memoria gráfica con sólo leer o escribir los bits apropiados.

Sin embargo, la arquitectura SAS tiene algunas deficiencias, la más seria de las cuales es la contienda por el acceso a la memoria del sistema. Una solución a este problema es usar una pastilla UCP con memoria caché para instrucciones o datos, con lo cual se reduce la dependencia de la UCP del acceso frecuente y rápido a la memoria del sistema. Por supuesto, esta solución y otras se pueden integrar de varias maneras bastante ingeniosas, como se ve con mayor detalle en el capítulo 18 de [FOLE90].

Otra complicación de diseño ocurre cuando la UCP tiene espacio virtual de direcciones, como en las familias Motorola 680x0 e Intel 80x86 y en varios procesadores con un conjunto reducido de instrucciones (*RISC reduced-instruction-set-computer*). En este caso, las direcciones de memoria generadas por el procesador de dibujo deben pasar por la misma traducción dinámica que las demás direcciones de memoria. Además, muchas arquitecturas de UCP distinguen entre un espacio virtual de direcciones para el kernel del sistema operativo y otro para el programa de aplicación. En muchos casos es deseable que la memoria gráfica (lienzo 0 en terminología de SRGP) se encuentre en el espacio del kernel, para que el manejador de dispositivo de pantalla del sistema tenga acceso directo. Sin embargo, los lienzos asignados por el programa de aplicación deben hallarse en el espacio de direcciones de la aplicación. Por lo tanto, las instrucciones de dibujo que tienen acceso a la memoria gráfica deben distinguir entre los espacios de direcciones del kernel y de la aplicación. Para que haya acceso al kernel, es necesario invocar la instrucción de presentación con una prolongada llamada de servicio del sistema operativo, en lugar de hacerlo con una sencilla llamada a subrutina.

A pesar de estas posibles complicaciones, cada vez más sistemas de dibujo de barrido tienen arquitectura con espacio único de direcciones, usualmente con el estilo presentado en la figura 4.12. La flexibilidad que se obtiene al permitir que la UCP y el procesador de dibujo tengan acceso a cualquier parte de la memoria de manera uniforme y homogénea es muy atractiva y simplifica la programación.

## 4.4 Controlador de vídeo

La tarea más importante de un controlador de vídeo es el refrescamiento constante de la pantalla. Hay dos tipos de refrescamiento principales: **entrelazado** y

**no entrelazado.** El primero se usa en la televisión y en los presentadores de barrido diseñados para alimentar televisores normales. El ciclo de refrescamiento se divide en dos campos, cada uno de los cuales dura  $\frac{1}{60}$  de segundo; por lo tanto, un refrescamiento completo tarda  $\frac{1}{30}$  de segundo. En el primer campo se presentan todas las líneas impares de la trama, y las pares se presentan en la segunda parte. El propósito del barrido entrelazado es colocar información nueva en todas las áreas de la pantalla con una tasa de 60 Hz, ya que una tasa de refrescamiento de 30 Hz tiende a producir parpadeo. El efecto neto del entrelazado es producir una imagen cuya tasa de refrescamiento efectiva se aproxima más a 60 Hz que a 30 Hz. Esta técnica funciona mientras las líneas de rastreo adyacentes presenten información similar; una imagen que consiste en líneas horizontales sobre líneas de rastreo alternas tendría mucho parpadeo. La mayoría de los controladores de vídeo tienen una tasa de refrescamiento de 60 Hz o más y usan rastreo no entrelazado.

La salida del controlador de vídeo tiene una de tres formas: RGB, monocromática o NTSC. En el caso de RGB (*red, green, blue*, rojo, verde, azul), tres cables transportan las señales roja, verde y azul para controlar los tres cañones de electrones de un CRT con mascarilla de sombra, mientras que otro cable transmite la señal de sincronización para indicar el inicio del retorno vertical y horizontal. Existen estándares para los voltajes, las formas de onda y los tiempos de sincronización de las señales RGB. Para las señales monocromáticas de 480 líneas de rastreo, el estándar es RS-170; para color, RS-170A; para señales monocromáticas de mayor resolución, RS-343. Con frecuencia, los tiempos de sincronización se incluyen en el mismo cable que la señal verde; en este caso las señales se denominan video compuesto. Las señales monocromáticas utilizan los mismos estándares pero sólo tienen cables de intensidad y sincronización, o un solo cable que transporta en forma compuesta la intensidad y la sincronización.

El vídeo NTSC (National Television System Committee) es el formato de señal que se emplea en la televisión comercial de Norteamérica. La información de color, intensidad y sincronización se combina en una señal con ancho de banda de unos 5 MHz, transmitida como 525 líneas de rastreo, en dos campos de 262.5 líneas cada una. Sólo 480 líneas son visibles; las otras ocurren durante los períodos de retorno vertical al final de cada campo. Un televisor monocromático usa la información de intensidad y sincronización; un televisor en color también emplea la información de color para controlar los tres cañones de color. El límite del ancho de banda permite que se transmitan varios canales en el intervalo de frecuencias asignadas a la televisión. Por desgracia, este ancho de banda limita la calidad de la imagen a una resolución efectiva de unos  $350 \times 350$ . No obstante, NTSC es el estándar para el equipo de grabación de vídeo de bajo costo. Las grabadoras más costosas almacenan componentes separados de la señal de color en forma analógica o digital. Los estándares de cinta de vídeo y transmisión de televisión de Europa y Rusia son dos estándares de 625 líneas de rastreo, SECAM y PAL.

Algunos controladores de vídeo sobreponen a la memoria gráfica un cursor programable, almacenado en un arreglo bidimensional de pixeles de  $16 \times 16$  o  $32 \times 32$ . Esto evita tener que usar una rutina especial para colocar la forma del cursor en la memoria gráfica en cada ciclo de refrescamiento, lo que reduce ligeramente el procesamiento adicional de la UCP. En forma parecida, algunos controladores de vídeo sobreponen a la memoria gráfica pequeños mapas de pixeles de tamaño fijo, llamados **fantasmas (sprites)**. Esta característica es común en los juegos de vídeo.

#### 4.4.1 Mezclado de vídeo

Otra función útil del controlador de vídeo es el mezclado de vídeo. Dos imágenes, una definida en la memoria gráfica y la otra por una señal de vídeo proveniente de una cámara, grabadora u otra fuente, se pueden combinar para formar una imagen compuesta. En los programas meteorológicos, de noticias y deportivos de la televisión aparecen regularmente ejemplos de esta combinación. En la figura 4.13 se muestra la organización genérica del sistema.

Hay dos tipos de mezclado. En uno de ellos se coloca una imagen gráfica en una imagen de vídeo. El gráfico o diagrama presentado sobre el hombro de un comentarista de noticias es típico de este estilo. El mezclado se logra con hardware que trata el valor de un pixel designado en la memoria gráfica como una bandera para indicar que debe presentarse la señal de video en lugar de la señal de la memoria gráfica. Normalmente, el valor del pixel designado corresponde al color de fondo de la imagen de la memoria gráfica, aunque se pueden lograr efectos interesantes si se emplea otro valor de pixel.

El segundo tipo de mezclado coloca la imagen de vídeo sobre la imagen de la memoria gráfica, como sucede cuando el meteorólogo de la televisión está de pie frente a un mapa que ocupa toda la pantalla. En realidad, la persona se encuentra frente a un telón de fondo cuyo color (generalmente azul) se usa para controlar el mezclado. Cuando el vídeo de entrada es azul se presenta la memoria gráfica; en caso contrario, se presenta la imagen de vídeo. ¡Esta técnica funciona bien mientras la persona no esté vestida con una corbata o una camisa azul!



**Figura 4.13** Controlador de vídeo que mezcla imágenes de una memoria gráfica y una fuente de señal de vídeo.

## 4.5 Dispositivos de entrada para la interacción con el operador

En esta sección describimos la operación de los dispositivos de entrada más comunes. Presentaremos un análisis breve y general del funcionamiento de los tipos de dispositivos disponibles. En el capítulo 8 se analizan las ventajas y desventajas de los dispositivos y se presentan otros más avanzados.

Esta presentación está organizada con base en el concepto de los **dispositivos lógicos**, presentados en el capítulo 2 como parte de SRGP y tratados con mayor detalle en el capítulo 7. Hay cinco tipos básicos de dispositivos lógicos: el **localizador**, para indicar una posición u orientación; el **selector**, para seleccionar una entidad presentada; el **valuador**, para introducir un número real; el **teclado**, para introducir una cadena de texto; y el de **opciones**, para elegir una de varias acciones u opciones. El concepto de dispositivo lógico define clases de equivalencia de dispositivos con base en el tipo de información que proporcionan al programa de aplicación.

### 4.5.1 Dispositivos localizadores

**Tableta.** Una tableta (o **tableta de datos**) es una superficie plana cuyo tamaño varía de unas 6 por 6 pulgadas hasta 48 por 72 pulgadas o más, que puede detectar la posición de un lápiz o un disco trazador móvil sostenido por la mano del usuario. En la figura 4.14 se presenta una pequeña tableta con un lápiz y un



**Figura 4.14** Tableta de datos con lápiz y disco trazador. El lápiz tiene en la punta un conmutador sensible a la presión, que se cierra al presionar el lápiz. El disco trazador tiene varios botones para marcar mandatos y un cursor de cruz para obtener mayor precisión al digitalizar imágenes que se colocan sobre la tableta. (Cortesía de Summagraphics Corporation.)

disco trazador (de aquí en adelante sólo haremos referencia al lápiz, aunque el tratamiento es aplicable a cualquiera). La mayoría de las tabletas utiliza un mecanismo sensor eléctrico para determinar la posición del lápiz. En una de estas configuraciones, dentro de la superficie de la tableta se encuentra una malla de alambres espaciados entre  $\frac{1}{4}$  y  $\frac{1}{2}$  pulgada. Las señales electromagnéticas generadas por pulsos eléctricos aplicados en secuencia a los alambres inducen una señal eléctrica en una bobina de alambre del lápiz. La potencia de la señal también se usa para determinar la posición del lápiz y para determinar en forma aproximada cuán cerca de la tableta [*lejos, cerca* (es decir, aproximadamente a un centímetro) de la tableta, o *tocando*] está el lápiz o el cursor. Cuando la respuesta es *cerca* o *tocando*, usualmente se presenta un cursor en la pantalla para proporcionar retroalimentación visual al usuario. Se envía una señal al computador cuando se presiona el lápiz contra la tableta o cuando se oprime algún botón del disco trazador (los discos trazadores tienen hasta 16 botones). Por lo general se obtiene de 30 a 60 veces por segundo la posición (*x, y*) en la tableta, el estado de los botones y el estado de cercanía [si este estado es *lejos*, no se encuentra disponible la posición (*x, y*)].

Los parámetros relevantes de las tabletas y otros dispositivos localizadores son su resolución (número de puntos distinguibles por pulgada), linealidad, capacidad de repetición y tamaño o gama. Estos parámetros son fundamentales para la digitalización de mapas y dibujos, pero no tienen tanta importancia cuando el dispositivo se usa únicamente para determinar la posición de un cursor en la pantalla, ya que el usuario tiene la realimentación de la posición del cursor en la pantalla para guiar los movimientos de su mano y porque la resolución de una pantalla típica es menor que la de las tabletas, incluso de las más baratas. Otras tecnologías de tabletas emplean acoplamiento de sonido (sónico) y resistivo.

Varios tipos de tabletas son transparentes y por lo tanto se pueden retroiluminar para digitalizar película de rayos X y negativos fotográficos, o bien montarse directamente sobre un CRT. La tableta resistiva es especialmente útil para esta tarea, ya que puede curvarse a la forma del CRT.

**Ratón.** El ratón es un pequeño dispositivo manual cuyo movimiento relativo sobre una superficie se puede medir. Los ratones difieren en cuanto al número de botones y a la forma de detectar su movimiento relativo. En la sección 8.1 se analizan varias aplicaciones importantes de los ratones en tareas de interacción. El movimiento de un rodillo en la base de un **ratón mecánico** se convierte a valores digitales que se usan para determinar la dirección y la magnitud del movimiento. El **ratón óptico** se utiliza sobre una superficie especial con una malla de líneas claras y oscuras alternantes. Un diodo emisor de luz (LED, *light-emitting diode*) en la parte inferior del ratón dirige un rayo de luz hacia la superficie, de la cual se refleja y es detectada por sensores en la parte inferior del ratón. Al mover el ratón, el rayo luminoso reflejado se interrumpe cada vez que se cruza una línea negra. El número de pulsos generados de esta forma, que equivale al número de líneas cruzadas, se usa para informar al computador cuál es el movimiento del ratón.

Como los ratones son dispositivos que miden el movimiento relativo, se pueden levantar, mover y bajar nuevamente sin que existan cambios en la posición indicada (una serie de estos movimientos suele denominarse *golpeo* del ratón). La naturaleza relativa del ratón significa que el computador debe mantener una *posición actual del ratón*, la cual aumenta o disminuye de acuerdo con los movimientos del ratón.

**Bola rastreadora.** Una bola rastreadora (*trackball*) generalmente se describe como un ratón mecánico invertido. El movimiento de la bola rastreadora, que gira libremente en su armazón, es detectado por potenciómetros o codificadores de eje. El usuario normalmente gira la bola rastreadora deslizando la palma de la mano sobre la bola. En el dispositivo se colocan varios commutadores al alcance de los dedos y se usan en forma muy similar a los botones de un ratón o de una tableta de datos.

**Palanca de mandos.** La palanca de mandos (*joystick*, Fig. 4.15) se puede mover a la derecha, a la izquierda, hacia adelante o hacia atrás; una vez más, el movimiento es detectado por potenciómetros. Por lo general se emplean resortes para devolver la palanca de mandos a su posición base en el centro. Algunas palancas de mandos, incluyendo la que aparece en la figura, tienen un tercer grado de libertad: la palanca se puede girar en el sentido en que giran las manecillas del reloj o en el sentido opuesto.

Es difícil usar una palanca de mandos para controlar la posición absoluta del cursor en la pantalla, ya que un pequeño movimiento de la palanca (normalmente bastante corta) se amplifica unas cinco o diez veces en el movimiento del



**Figura 4.15** Palanca de mandos con un tercer grado de libertad. La palanca se puede girar en el sentido que lo hacen las manecillas del reloj o en sentido contrario. (Cortesía de Measurement Systems, Inc.)

cursor. Esto hace que el movimiento del cursor en la pantalla sea muy brusco y no permita un posicionamiento rápido y preciso. Por lo tanto, la palanca de mandos se usa más para controlar la velocidad de movimiento del cursor y no su posición absoluta. Esto quiere decir que la posición actual del cursor en la pantalla cambia de acuerdo con tasas determinadas por la palanca de mandos.

**Pantalla sensible al tacto.** Los ratones, las bolas rastreadoras y las palancas de mandos ocupan espacio en la superficie de trabajo. La pantalla sensible al tacto permite al usuario señalar directamente con el dedo en la pantalla y mover el cursor de un lugar a otro. Para las pantallas sensibles al tacto se emplean varias tecnologías distintas. Las pantallas de baja resolución (de 10 a 50 posiciones determinables en cada dirección) usan una serie de LED infrarrojos y sensores luminosos (fotodiodos o fototransistores) para formar una malla de rayos luminosos invisibles sobre el área de la pantalla. Al tocar la pantalla se interrumpen uno o dos de los rayos horizontales y verticales, indicando la posición del dedo. Si se interrumpen dos rayos paralelos, se supone que el dedo está entre los dos; si se interrumpe uno, se supone que el dedo está en el rayo.

Una pantalla sensible al tacto con acoplamiento capacitivo puede ofrecer cerca de 100 posiciones determinables en cada dirección. Cuando el usuario toca la pantalla de vidrio con cubierta conductiva, los circuitos electrónicos detectan la posición del toque a partir del cambio en la impedancia a través del recubrimiento conductor [INTE85].

Los parámetros más importantes de las pantallas sensibles al tacto son la resolución, la cantidad de presión necesaria para activarlo (no aplicable a la pantalla de rayos luminosos) y la transparencia (una vez más, no aplicable a la pantalla de rayos luminosos). Otro asunto importante con algunas de las tecnologías es el paralaje: si la tableta está a un centímetro de la pantalla, los usuarios tocarán en la posición de la pantalla que esté alineada con sus ojos y el punto deseado en la pantalla, no en la posición de la pantalla directamente perpendicular al punto deseado.

Los usuarios están acostumbrados a cierto tipo de realimentación táctil, pero las pantallas sensibles al tacto no ofrecen ninguna. Por lo tanto, es muy importante que se proporcionen otros tipos de realimentación, como un tono audible o el realce del objetivo o la posición designada.

#### 4.5.2 Dispositivos de teclado

**El teclado alfanumérico** es el prototipo de los dispositivos de entrada de texto. Se emplean varias tecnologías para detectar una tecla oprimida, incluyendo contacto mecánico, cambios en la capacitancia y acoplamiento magnético. La característica funcional importante de un dispositivo de teclado es que crea un código, por ejemplo ASCII, correspondiente en forma única a la tecla presionada. En algunos casos es deseable permitir *acordes* (varias teclas oprimidas al mismo tiempo) en un teclado alfanumérico, para que los usuarios experimentados tengan acceso rápido a diferentes mandatos. Esta situación generalmente no

es posible en un **teclado codificado** estándar, que devuelve un código ASCII por cada carácter tecleado y no devuelve nada si se oprimen dos teclas al mismo tiempo (a menos que la tecla adicional sea el cambio a mayúsculas, el control u otra tecla especial). Por el contrario, un **teclado no codificado** devuelve la identidad de todas las teclas que se opriman simultáneamente, lo cual permite los acordes.

#### 4.5.3 Dispositivos valuadores

La mayoría de los dispositivos valuadores que proporcionan valores escalares se basan en potenciómetros, como los controles de volumen y de tono de un aparato estereofónico. Los valuadores generalmente son potenciómetros giratorios (perillas) montados en grupos de ocho o diez. Los potenciómetros giratorios simples pueden girar unos  $330^\circ$ , pero esto quizás no sea suficiente para proporcionar el intervalo y la resolución necesarios. Los potenciómetros de giro continuo pueden girar libremente en cualquier dirección y por consiguiente su intervalo no tiene límites. Los potenciómetros lineales, que por necesidad son dispositivos limitados, casi no se usan en sistemas gráficos.

#### 4.5.4 Dispositivos de opciones

Las **teclas de función** constituyen el dispositivo de opciones más común. En ocasiones se construyen como una unidad aparte, pero es más usual que estén integradas al teclado. Otros dispositivos de opciones son los **botones** que aparecen en los trazadores de las tabletas y en el ratón. Los dispositivos de opciones generalmente se usan para introducir mandatos o especificar opciones de menú en un programa gráfico. Los sistemas de propósito específico pueden utilizar teclas de función con rótulos permanentes. Para que las etiquetas sean cambiables o “suaves”, se puede incluir una pequeña pantalla LCD o LED junto al botón o en la misma tecla. Otra alternativa es colocar los botones en la orilla de la pantalla, para que las etiquetas aparezcan en la pantalla junto al botón físico.

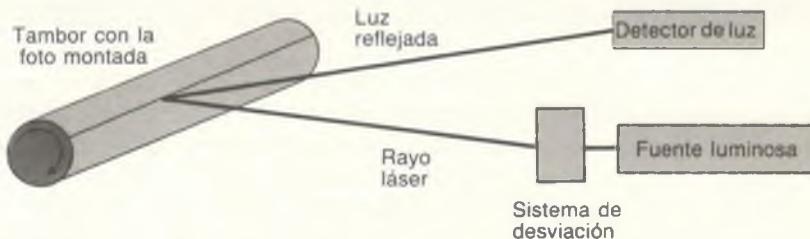
### 4.6 Digitalizadores de imágenes

Aunque es posible utilizar las tabletas de datos para digitalizar manualmente dibujos lineales, se trata de un proceso lento y tedioso, inadecuado para algo más que los dibujos más sencillos, además de que no sirve para imágenes con medios tonos. Los digitalizadores de imágenes constituyen una solución eficaz. Si se usa una cámara de televisión en conjunto con un almacenador de cuadros digitales, se tiene una forma barata de obtener imágenes de barrido de resolución mediana ( $1000 \times 1000$ , con varios niveles de intensidad) de fotografías en blanco y negro o en color. Las cámaras de televisión de barrido lento con dispositivos con acoplamiento de carga (CCD, *charge-coupled-device*) pueden crear

una imagen de  $2000 \times 2000$  en unos 30 segundos. Un método todavía más barato es usar una cabeza de barrido, que consiste en una malla de celdas sensible a la luz, montada sobre la cabeza de impresión de una impresora; con esto se pueden barrer imágenes con una resolución de alrededor de 80 unidades por pulgada. Sin embargo, esta resolución no es aceptable para trabajos impresos de alta calidad. En estos casos se emplea un **digitalizador de fotografías**. La fotografía se monta en un tambor giratorio, se dirige hacia ella un rayo luminoso finamente colimado y una fotocelda mide la cantidad de luz reflejada. Para un negativo, la luz transmitida se mide con una fotocelda dentro del tambor, que es transparente. Conforme gira el tambor (Fig. 4.16), la fuente luminosa se mueve con lentitud de un extremo al otro, generando así un barrido de trama de toda la fotografía. En el caso de fotografías en color se hacen varias pasadas usando filtros frente a la fotocelda para separar los diversos colores. Los digitalizadores de mayor resolución utilizan fuentes luminosas láser y su resolución es superior a las 2000 unidades por pulgada.

Otra clase de digitalizador emplea una larga y delgada tira de CCD, llamada *arreglo CCD*. Un dibujo se digitaliza al pasar por debajo del arreglo CCD, incrementando el movimiento de la imagen de acuerdo con la resolución requerida. De esta manera, en una sola pasada, con duración de uno a dos minutos, se puede digitalizar un dibujo de gran tamaño. La resolución del arreglo CCD es de 200 a 1000 unidades por pulgada, que es inferior a la del digitalizador de fotografías.

Los dibujos de línea se pueden digitalizar con cualquiera de los métodos que hemos descrito. Lo más difícil es derivar cierto significado de los pixeles resultantes. La **vectorización** es el proceso de extraer líneas, caracteres y otras primitivas geométricas de una imagen de trama. Para esta tarea se requieren algoritmos apropiados, no tanto algún hardware de digitalización en especial, y en esencia se trata de un problema de procesamiento de imágenes que comprende varios pasos. En primer lugar, se emplean detección de valores límite y mejora de orillas para limpiar la imagen de trama; es decir, para eliminar manchas y llenar los huecos. Después se utilizan algoritmos de extracción de caracte-



**Figura 4.16** Digitalizador de fotografías. La fuente luminosa se desvía por el eje del tambor y se mide la cantidad de luz reflejada.

ticas para combinar los píxeles activos adyacentes y formar primitivas geométricas, como serían líneas rectas. En un segundo nivel de complejidad, se emplean algoritmos de reconocimiento de patrones para combinar las primitivas simples y formar arcos, letras, símbolos, etcétera. En algunos casos se requiere la interacción con el usuario para resolver ambigüedades causadas por interrupciones en las líneas, manchas y varias líneas que se intersecan en forma muy cercana.

Un problema más difícil es organizar un conjunto de primitivas geométricas para formar estructuras de datos significativas. Una colección de líneas desorganizadas no es muy útil como entrada para un programa de aplicación topográfico (de mapas) o de CAD. Las construcciones geométricas de mayor nivel representadas en los dibujos deben ser reconocidas. De esta manera, las líneas que definen las fronteras de un país se deben organizar como una primitiva de polígono, y el pequeño símbolo “+” que representa el centro de un arco se debe agrupar con el arco mismo. Existen soluciones parciales para estos problemas; los sistemas comerciales dependen de la intervención del usuario cuando la situación se complica, aunque los algoritmos mejoran constantemente.

## Ejercicios

4.1 Si los fósforos de larga persistencia reducen la frecuencia de fusión, ¿por qué no usarlos siempre?

4.2 Escriba un programa para exhibir patrones de prueba en una pantalla de barrido. Deben proporcionarse tres patrones: (1) líneas horizontales de un pixel de ancho, separadas por 0, 1, 2 o 3 píxeles; (2) líneas verticales de un pixel de ancho, separadas por 0, 1, 2 o 3 píxeles; y (3) una malla de puntos de un pixel espaciados a intervalos de cinco píxeles. Cada patrón debe poder presentarse en blanco, rojo, verde o azul y como barras de color alternantes. ¿Qué relación hay entre lo que usted observa al presentar los patrones y nuestro análisis de la resolución de barrido?

4.3 ¿Cuánto tardaría la carga de un mapa de bits de 512 por 512, suponiendo que se empaquetan ocho píxeles por byte y que los bytes se pueden transferir y desempaquetar a una tasa de 100 000 bytes por segundo? ¿Cuánto tardaría cargar un mapa de bits de 1024 por 1280?

4.4 Diseñe la lógica de una unidad de hardware para convertir direcciones de barrido bidimensionales a direcciones de byte más bit en un byte. Las entradas de la unidad son las siguientes: (1)  $(x, y)$ , una dirección de barrido; (2)  $base$ , la dirección del byte de la memoria que tiene la dirección de barrido  $(0, 0)$  en el bit 0; y (3)  $x_{\max}$ , la máxima dirección de barrido  $x$  ( $0$  es la mínima). Las salidas de la unidad son las siguientes: (1)  $byte$ , la dirección del byte que contiene  $(x, y)$  en uno de sus bits; y (2)  $bit$ , el bit en  $byte$  que contiene  $(x, y)$ . ¿Qué simplificaciones son posibles si  $x_{\max} + 1$  es una potencia de dos?



# 5

# Transformaciones geométricas

En este capítulo se presentan las principales transformaciones geométricas bidimensionales y tridimensionales que se emplean en la graficación por computador. Las transformaciones de traslación, escalamiento y rotación que analizamos aquí son indispensables en muchas operaciones gráficas y con frecuencia haremos referencia a ellas en los capítulos subsecuentes.

Las transformaciones son utilizadas directamente por los programas de aplicación y dentro de muchos paquetes de subrutinas gráficas. Un programa de aplicación para la planificación de ciudades usaría la traslación para colocar símbolos de árboles y edificios en los lugares apropiados, la rotación para orientar los objetos y el escalamiento para alterar el tamaño de los símbolos. En términos generales, muchas aplicaciones emplean transformaciones geométricas para cambiar la posición, la orientación y el tamaño de los objetos (también llamados **símbolos o plantillas**) en un dibujo. En el capítulo 6 se usarán versiones tridimensionales de la rotación, la traslación y el escalamiento como parte del proceso de creación de representaciones bidimensionales de objetos tridimensionales. En el capítulo 7 veremos la forma en que un paquete gráfico contemporáneo utiliza las transformaciones como parte de su implantación y las hace disponibles para los programas de aplicación.

## 5.1 Aspectos matemáticos preliminares

En esta sección se repasan los conceptos matemáticos más importantes utilizados en el libro, en especial aquellos relacionados con vectores y matrices. De

ninguna manera se pretende que sea un análisis exhaustivo del álgebra lineal o de la geometría, ni tampoco que constituya una introducción a estos temas. Más bien, en esta sección suponemos que usted ha cursado el equivalente a un año de matemáticas universitarias pero que su familiaridad con temas como el álgebra y la geometría se ha desvanecido un poco. Si usted maneja sin problemas los temas que se presentan en esta sección, puede omitirla y pasar directamente a la sección 5.2. Si sus conocimientos acerca de los conceptos que presentamos aquí son nulos o no están actualizados, esperamos que la sección sirva como un “recetario” que pueda consultar conforme avance a través del libro. Para aquellos que no estén seguros de su habilidad para comprender y aplicar los conceptos que repasamos aquí, tengan la confianza de que toda operación vectorial o matricial no es más que una notación abreviada de una forma algebraica equivalente. Esta notación es tan conveniente y poderosa que le recomendamos que la conozca y maneje sin problemas. Es nuestra experiencia que un programa como *Mathematica*<sup>TM</sup> [WOLF91] —que le permite efectuar operaciones simbólicas y numéricas con vectores y matrices en forma interactiva— puede servir como valiosa herramienta de aprendizaje. Los lectores que estén interesados en examinar con mayor detalle este material deberán consultar [BANC83; HOFF61; MARS85].

### 5.1.1 Los vectores y sus propiedades

Es probable que su primer contacto con el concepto de vector haya sido en un curso de física o mecánica, posiblemente usando el ejemplo de la velocidad y la dirección de una pelota al salir de una raqueta. En ese momento, el estado de la pelota se puede representar con un segmento de línea con una flecha. El segmento de línea apunta en la dirección del movimiento de la pelota y tiene una longitud que representa su velocidad. Este segmento de línea dirigida representa el *vector de velocidad* de la pelota. El vector de velocidad es uno de muchos ejemplos de los vectores que ocurren en problemas físicos. Otros ejemplos son la fuerza, la aceleración y el momento.

El concepto de vector ha demostrado ser de gran valor en la física y en las matemáticas. En la graficación por computador se usa con frecuencia para representar la posición de puntos en un conjunto de datos de coordenadas mundiales, la orientación de superficies en el espacio, el comportamiento de la luz que interactúa con objetos sólidos y transparentes, y para varios fines más. En muchos capítulos subsecuentes volverán a aparecer los vectores. A continuación señalaremos cómo y cuándo usar los vectores conforme describimos sus propiedades.

Aunque en algunos textos los vectores prefieren describirse exclusivamente desde el punto de vista geométrico, nosotros haremos hincapié en su naturaleza dual y aprovecharemos también su definición algebraica. Esta orientación nos conduce a formulaciones concisas y compactas del álgebra lineal, preferibles para las matemáticas de la graficación por computador. Sin embargo, también presentaremos la interpretaciones geométricas cuando ayuden a comprender ciertos conceptos.

Existen definiciones más estrictas, pero para nuestros fines podemos afirmar que un **vector** es una tupla de  $n$  números reales, donde  $n$  es dos para el espacio bidimensional, tres para el tridimensional, etcétera. Denotaremos los vectores con letras en cursivas,<sup>1</sup> por lo general  $u$ ,  $v$  y  $w$  en esta sección. Los vectores pueden emplearse en dos operaciones: suma de vectores y multiplicación de vectores por números reales, lo que se denomina **multiplicación escalar**.<sup>2</sup> Estas operaciones tienen ciertas propiedades. La suma es conmutativa, asociativa y existe un vector, tradicionalmente llamado  $0$ , que tiene la propiedad de que para cualquier vector  $v$ ,  $0 + v = v$ . Estas operaciones también tienen inversas (es decir, para cada vector  $v$  hay otro vector  $w$  con la propiedad de que  $v + w = 0$ ;  $w$  se escribe “ $-v$ ”). La multiplicación escalar satisface las reglas  $(\alpha\beta)v = \alpha(\beta v)$ ,  $1v = v$ ,  $(\alpha + \beta)v = \alpha v + \beta v$  y  $\alpha(v + w) = \alpha v + \alpha w$ .

La suma se define componente a componente, al igual que la multiplicación escalar. Los vectores se escriben verticalmente, de manera que un ejemplo de vector tridimensional sería

$$\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}.$$

Podemos sumar los elementos

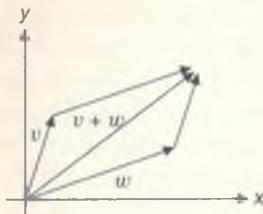
$$\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} + \begin{bmatrix} 2 \\ 3 \\ 6 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ 7 \end{bmatrix}.$$

La mayor parte de la graficación por computador se efectúa en el espacio bidimensional, en el tridimensional o, como veremos en las secciones 5.7 y 6.6.4, en el espacio de cuatro dimensiones.

Al llegar a este punto es importante establecer la diferencia entre los vectores y los puntos. Podemos considerar que un punto está definido por un vector que llega a él, pero esta representación requiere la especificación de *dónde* comienza a dibujarse el vector, o sea, definir lo que conocemos como origen. Hay muchas operaciones que se pueden efectuar con puntos y que no requieren un origen, así como varias operaciones vectoriales (p. ej., la multiplicación) que no están definidas cuando se aplican a puntos. Por lo tanto, es mejor no confundir la naturaleza de los vectores y los puntos, sino tratarlos como conceptos distintos. Sin embargo, existen operaciones como la formación de un vector a partir de la diferencia entre dos puntos, que sí tienen sentido.

<sup>1</sup> En lo que resta de este libro hemos tratado de seguir las convenciones que aparecen en la literatura de investigación, aunque no son consistentes de un artículo al otro. Debido a esta variación que se presenta en la práctica, quizás en algunas ocasiones vea que se usan letras mayúsculas para denotar vectores y que en otras se emplean letras minúsculas.

<sup>2</sup> Los escalares (es decir, los números reales) se denotarán con letras griegas, usualmente las primeras del alfabeto.



**Figura 5.1**  
Suma de vectores en un plano.

Con esta advertencia acerca de la distinción entre los vectores y los puntos, podemos descubrir varias propiedades útiles si tratamos los puntos como vectores. Por ejemplo, considere el espacio bidimensional con un origen especificado. Podemos definir la suma de vectores con la conocida **regla del paralelogramo**: para sumar los vectores  $v$  y  $w$  tomamos una flecha del origen a  $w$ , la trasladamos para que la base esté en el punto  $v$  y definimos  $v + w$  como el nuevo punto extremo de la flecha. Si también dibujamos la flecha del origen a  $v$  y realizamos el proceso correspondiente, obtenemos un paralelogramo, como se ilustra en la figura 5.1. La multiplicación escalar por un número real  $\alpha$  se define en forma parecida: se dibuja una flecha del origen al punto  $v$ , se hace crecer por un factor de  $\alpha$  manteniendo fijo el extremo en el origen, para entonces definir  $\alpha v$  como el punto extremo de la flecha resultante. Por supuesto, se pueden hacer las mismas definiciones para el espacio tridimensional.

Dadas las dos operaciones disponibles en el espacio vectorial, hay cálculos naturales que se pueden efectuar con vectores. Uno de ellos es la formación de **combinaciones lineales**. Una combinación lineal de los vectores  $v_1, \dots, v_n$  es cualquier vector de la forma  $\alpha_1 v_1 + \alpha_2 v_2 + \dots + \alpha_n v_n$ . Las combinaciones lineales de los vectores se emplean para describir muchos objetos. En el capítulo 9 veremos algunas aplicaciones de las combinaciones lineales en la representación de curvas y superficies.

### 5.1.2 Producto punto de vectores

Dados dos vectores bidimensionales

$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad y \quad \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix},$$

su **producto interior** o **producto punto** se define como  $x_1 y_1 + \dots + x_n y_n$ . El producto punto de vectores se denota generalmente con  $v \cdot w$ .

La distancia del punto  $(x, y)$  al origen  $(0, 0)$  en el plano es  $\sqrt{x^2 + y^2}$ . En términos generales, la distancia del punto  $(x_1, \dots, x_n)$  al origen en el espacio  $n$ -dimensional es  $\sqrt{x_1^2 + \dots + x_n^2}$ . Si  $v$  es el vector

$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix},$$

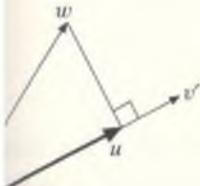
podemos ver que esto no es más que  $\sqrt{v \cdot v}$ . Ésta es nuestra definición de la **longitud** de un vector  $n$ -dimensional, la cual se denota con  $\|v\|$ . La distancia entre dos puntos en el espacio  $n$ -dimensional estándar se define en forma parecida: la distancia entre  $P$  y  $Q$  es la longitud del vector  $Q - P$ .

### 5.1.3 Propiedades del producto punto

El producto punto tiene varias propiedades interesantes. En primer lugar es simétrico:  $v \cdot w = w \cdot v$ . En segundo lugar es **no degenerado**:  $v \cdot v = 0$  sólo si  $v = 0$ . En tercer lugar es **bilineal**:  $v \cdot (u + \alpha w) = v \cdot u + \alpha(v \cdot w)$ .

El producto punto se puede usar para generar vectores cuya longitud sea 1 (esto se conoce como **normalizar un vector**). Para normalizar un vector  $v$  basta calcular  $v' = v/\|v\|$ . El vector resultante tiene longitud de 1 y se denomina **vector unidad**.

El producto punto también se puede emplear para medir ángulos. El **ángulo entre los vectores  $v$  y  $w$**  es



$$\cos^{-1} \left( \frac{v \cdot w}{\|v\| \|w\|} \right).$$

Observe que si  $v$  y  $w$  son vectores unidad, entonces no es necesaria la división.

Si tenemos un vector unidad  $v'$  y otro vector  $w$  y proyectamos  $w$  perpendicularmente sobre  $v'$ , como se ilustra en la figura 5.2, llamando  $u$  al resultado, entonces la longitud de  $u$  debe ser la longitud de  $w$  multiplicada por  $\cos \theta$ , donde  $\theta$  es el ángulo entre  $v$  y  $w$ . En otras palabras,

$$\begin{aligned} \|u\| &= \|w\| \cos \theta \\ &= \|w\| \left( \frac{v \cdot w}{\|v\| \|w\|} \right) \\ &= v \cdot w, \end{aligned}$$

ya que la longitud de  $v$  es 1. Esto nos proporciona otra interpretación del producto punto: el producto punto de  $v$  y  $w$  es la longitud de la proyección de  $w$  sobre  $v$ , si  $v$  es un vector unidad. Encontraremos muchas aplicaciones del producto punto, sobre todo en el capítulo 14, donde se usa para describir la forma en que la luz interactúa con las superficies.

### 5.1.4 Matrices

Una **matriz** es un arreglo rectangular de números que se emplea con frecuencia en operaciones con puntos y vectores. Una matriz se puede considerar como la representación de una regla para transformar sus operandos a través de una transformación lineal. Sus elementos —generalmente números reales— tienen doble índice y por convención el primero es la fila y el segundo la columna. La convención matemática establece que los índices comienzan en 1; ciertos lenguajes de programación emplean índices que comienzan con 0. Dejaremos que los programadores que usan esos lenguajes desplacen una posición todos los índices. De esta manera, si  $A$  es una matriz, entonces  $a_{1,2}$  se refiere al elemento de la

#### ■ 5.2

Proyección de  $w$  sobre  $v$ . La proyección de  $w$  sobre un vector unidad  $v'$  es un vector  $u$  cuya longitud es el coseno del ángulo entre  $u$  y  $v'$ .

tercera fila, segunda columna. Al usar índices simbólicos, como en  $a_{ij}$ , se omite la coma separadora.

Un vector en el espacio  $n$ -dimensional, que hemos escrito en la forma

$$\begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix},$$

se puede considerar como una matriz de  $n \times 1$  y se denomina **vector columna**. En gran parte de este libro usaremos matrices y sus operaciones relacionadas (véanse las Secs. 5.1.5-5.1.8). Las matrices tienen una función muy importante en las transformaciones geométricas (este capítulo), la visualización tridimensional (Cap. 6), los paquetes gráficos tridimensionales (Cap. 7) y la descripción de curvas y superficies (Cap. 9).

### 5.1.5 Multiplicación de matrices

Las matrices se multiplican de acuerdo con la siguiente regla: si  $A$  es una matriz de  $n \times m$  con entradas  $a_{ij}$  y  $B$  es una matriz de  $m \times p$  con entradas  $b_{ij}$ , entonces  $AB$  está definida y es una matriz de  $n \times p$  con entradas  $c_{ij}$ , donde  $c_{ij} = \sum_k a_{ik} b_{kj}$ . Si consideramos las columnas de  $B$  como vectores individuales,  $\vec{B}_1, \dots, \vec{B}_p$ , y las filas de  $A$  como vectores  $\vec{A}_1, \dots, \vec{A}_m$  (pero rotados 90° para que queden horizontales), podemos ver que  $c_{ij}$  es  $\vec{A}_i \cdot \vec{B}_j$ . Se conservan las propiedades usuales de multiplicación, excepto que la multiplicación de matrices no es conmutativa: por lo general,  $AB$  es distinto de  $BA$ . Sin embargo, la multiplicación se distribuye sobre la suma:  $A(B + C) = AB + AC$  y existe un elemento identidad para la multiplicación, específicamente la **matriz identidad**,  $I$ , que es una matriz cuadrada con todas las entradas iguales a 0 excepto por unos en la diagonal (es decir, las entradas son  $\delta_{ij}$ , donde  $\delta_{ij} = 0$  a menos que  $i = j$ , y entonces  $\delta_{ii} = 1$ ). En el ejemplo 5.1 aparece una representación gráfica de la multiplicación de matrices.

### 5.1.6 Determinantes

El **determinante** de una matriz cuadrada es un número que se forma a partir de los elementos de la matriz. El cálculo del determinante es un tanto complicado, ya que la definición es recursiva. El determinante de la matriz de  $2 \times 2$   $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$  es  $ad - bc$ . El determinante de una matriz de  $n \times n$  se define en función de determinantes de matrices más pequeñas. Si  $A_{1i}$  denota el determinante de la matriz de  $(n - 1) \times (n - 1)$  que se obtiene al eliminar la primera fila y la  $i$ -ésima columna de la matriz  $A$  de  $n \times n$ , el determinante de la matriz  $A$  se define como

$$\det A = \sum_{i=1}^n (-1)^{1+i} A_{1i}.$$

Una aplicación especial del determinante tiene lugar en el espacio tridimensional: el **producto cruz**. El producto cruz de dos vectores

$$v = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \quad \text{y} \quad w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix},$$

se calcula tomando el determinante de la matriz,

$$\begin{bmatrix} i & j & k \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{bmatrix},$$

donde las letras  $i, j$  y  $k$  representan vectores unidad dirigidos por los tres ejes de las coordenadas. El resultado es una combinación lineal de las variables  $i, j$  y  $k$  que produce el vector

$$\begin{bmatrix} v_2w_3 - v_3w_2 \\ v_3w_1 - v_1w_3 \\ v_1w_2 - v_2w_1 \end{bmatrix},$$

denotado por  $v \times w$ . Tiene la propiedad de ser perpendicular al plano definido por  $v$  y  $w$ , y su longitud es el producto  $\|v\| \|w\| |\sin \theta|$ , donde  $\theta$  es el ángulo entre  $v$  y  $w$ . En el capítulo 9 aprovecharemos las propiedades del producto cruz, usándolo para determinar la ecuación de plano de un polígono.

### 5.1.7 Transpuesta de una matriz

Se puede reflejar una matriz de  $n \times k$  con respecto a su diagonal (de la parte superior izquierda a la inferior derecha) para formar una matriz de  $k \times n$ . Si la primera matriz tiene entradas  $a_{ij}$  ( $i = 1, \dots, n$ ;  $j = 1, \dots, k$ ), la matriz resultante tendrá entradas  $b_{ij}$  ( $i = 1, \dots, k$ ;  $j = 1, \dots, n$ ), con  $b_{ij} = a_{ji}$ . Esta nueva matriz se conoce como **transpuesta** de la matriz original. La transpuesta de  $A$  se escribe  $A^T$ . Si consideramos un vector en el espacio  $n$ -dimensional como una matriz de  $n \times 1$ , entonces su transpuesta es una matriz de  $1 \times n$  (en ocasiones conocida como **vector fila**). Con la transpuesta se puede obtener una nueva descripción del producto punto; específicamente,  $u \cdot v = u^T v$ .

### 5.1.8 Inversa de una matriz

La multiplicación de matrices difiere de la multiplicación ordinaria en otra forma: es posible que una matriz no tenga inverso multiplicativo. De hecho, las inversas únicamente están definidas para matrices cuadradas, y ni siquiera todas éstas tienen inversas. Para ser precisos, las únicas matrices cuadradas que tienen inversas son aquellas cuyo determinante es distinto de cero.

Si  $A$  y  $B$  son matrices de  $n \times n$  y  $AB = BA = I$ , donde  $I$  es la matriz identidad de  $n \times n$ , se dice que  $B$  es la inversa de  $A$  y se escribe  $A^{-1}$ . En el caso de matrices de  $n \times n$  con números reales como entradas, basta mostrar que  $AB = I$  o que  $BA = I$ ; si una de estas expresiones es verdadera, la otra también lo será.

Si se nos proporciona una matriz de  $n \times n$ , la manera preferida para encontrar su inversa es por eliminación de Gauss, en especial para cualquier matriz mayor que  $3 \times 3$ . Una buena obra de referencia que incluye este método, así como programas para su implantación, es [PRES88].

### Ejemplo 5.1

En la sección 5.7 veremos la importancia de las matrices de  $4 \times 4$  en la graficación por computador, ya que se usan mucho en las transformaciones tridimensionales.

#### Problema:

- a. Encuentre el producto de matrices  $C = AB$  de

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{k} & 1 \end{bmatrix} \quad \text{y} \quad B = \begin{bmatrix} \cos\theta & 0 & \operatorname{sen}\theta & 0 \\ 0 & 1 & 0 & m \\ -\operatorname{sen}\theta & 0 & \cos\theta & n \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Como descubriremos en las secciones 6.5 y 5.7, estas matrices especifican una proyección de perspectiva y una rotación y dos traslaciones, respectivamente.

- b. Escriba una función  $C$  que devuelva el producto matricial  $C$  de dos matrices  $A$  y  $B$  de  $4 \times 4$ .

#### Respuesta:

- a. La regla de multiplicación de matrices, definida en la sección 5.1.5, se puede ilustrar de la siguiente manera:

$$c_{ij} = \sum_{s=1}^m a_{is} b_{sj} \quad \left[ \begin{array}{cccc} b_{11} & b_{12} & | b_{13} & b_{14} \\ b_{21} & b_{22} & | b_{23} & b_{24} \\ b_{31} & b_{32} & | b_{33} & b_{34} \\ b_{41} & b_{42} & | b_{43} & b_{44} \end{array} \right] \cdot \left[ \begin{array}{cccc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array} \right] = \left[ \begin{array}{cccc} c_{11} & c_{12} & | c_{13} & c_{14} \\ c_{21} & c_{22} & | c_{23} & c_{24} \\ c_{31} & c_{32} & | c_{33} & c_{34} \\ c_{41} & c_{42} & | c_{43} & c_{44} \end{array} \right]$$

donde hemos seleccionado el elemento  $c_{43}$  para evaluación. Por lo tanto, la ecuación de la multiplicación nos indica que debemos multiplicar los elementos

de la cuarta fila de  $A$  por los elementos de la tercera columna de  $B$ . Al efectuar esta operación se obtiene el resultado  $c_{43} = (\cos \theta)/k$ . Al aplicar este procedimiento a cada uno de los elementos de  $C$ , tenemos la siguiente matriz resultante:

$$C = \begin{bmatrix} \cos \theta & 0 & \operatorname{sen} \theta & 0 \\ 0 & 1 & 0 & m \\ 0 & 0 & 0 & 0 \\ \frac{-\operatorname{sen} \theta}{k} & 0 & \frac{\cos \theta}{k} & \frac{n}{k} + 1 \end{bmatrix}.$$

```
b. typedef struct matriz_4_struct {
    double elemento[4][4];
} matriz_4;

/* multiplicar las matrices c = ab */
/* observe que c no debe apuntar a ninguna de las matrices de entrada */
matriz_4 *V3MatMul (a, b, c)
matriz_4 *a, *b, *c;
{
    int i, j, k;
    for(i = 0; i < 4; i + +){
        for(j = 0; j < 4; j + +){
            c->elemento[i][j] = 0.0;
            for(k = 0; k < 4; k + +)
                c->elemento[i][j] += a->elemento[i][k] * b->elemento[k][j];
        }
    }
    return (c);
}
```

## 5.2 Transformaciones bidimensionales

Podemos **trasladar** puntos en el plano  $(x, y)$  a nuevas posiciones si añadimos valores de traslación a las coordenadas de los puntos. Para cada punto  $P(x, y)$  que se moverá  $d_x$  unidades en forma paralela al eje  $x$  y  $d_y$  unidades en paralelo al eje  $y$ , para llegar al nuevo punto  $P'(x', y')$ , podemos escribir

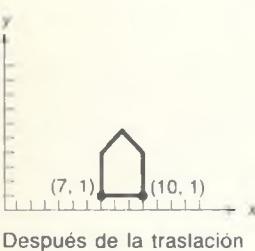
$$x' = x + d_x, \quad y' = y + d_y. \quad (5.1)$$

Si definimos los vectores columna

$$P = \begin{bmatrix} x \\ y \end{bmatrix}, \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \quad T = \begin{bmatrix} d_x \\ d_y \end{bmatrix}, \quad (5.2)$$

la ecuación (5.1) se puede expresar en forma más concisa como

$$P^* = P + T. \quad (5.3)$$



Podemos trasladar un objeto aplicando la ecuación (5.2) a cada uno de sus puntos. Sin embargo, en un objeto cada línea está formada por un número infinito de puntos, por lo que este proceso requeriría un tiempo infinito. Por fortuna, podemos trasladar todos los puntos de una línea con sólo trasladar los puntos extremos y dibujar una nueva línea entre estos puntos trasladados; esta observación también se aplica al escalamiento (estiramiento) y a la rotación. En la figura 5.3 se muestra el efecto de una traslación de  $(3, -4)$  aplicada a un bosquejo de una casa.

Los puntos se pueden **escalar** (estirar) por  $s_x$  sobre el eje  $x$  y por  $s_y$  en el eje  $y$  para obtener nuevos puntos si se aplican las multiplicaciones

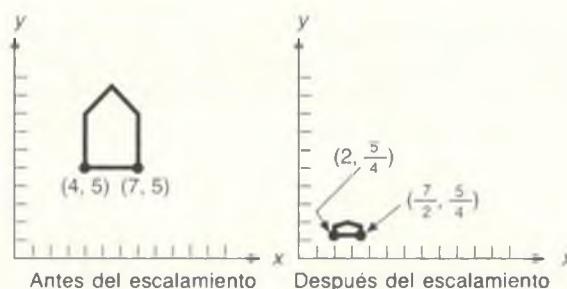
$$x' = s_x \cdot x, \quad y' = s_y \cdot y. \quad (5.4)$$

En forma matricial, esto es

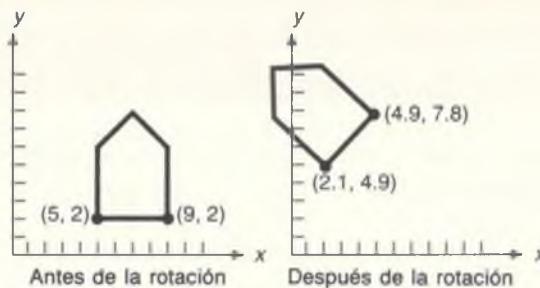
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \quad o \quad P' = S \cdot P, \quad (5.5)$$

donde  $S$  es la matriz en la ecuación (5.5).

En la figura 5.4, la casa se escala por un factor de  $\frac{1}{2}$  en  $x$  y de  $\frac{1}{4}$  en  $y$ . Observe que el escalamiento se lleva a cabo con respecto al origen: la casa es ahora más pequeña y más cercana al origen. Si los factores de escalamiento fueran mayores que 1, la casa sería en cambio más grande y estaría más lejos del origen. En la sección 5.3 se analizan técnicas para escalar con respecto a un punto distinto del origen. Las proporciones de la casa también han cambiado, ya que se ha empleado un escalamiento **diferencial**, donde  $s_x \neq s_y$ . Las proporciones no son afectadas al usar escalamiento **uniforme**, donde  $s_x = s_y$ .



**Figura 5.4** Escalamiento de una casa. El escalamiento no es uniforme y la casa cambia de posición.



**Figura 5.5** Rotación de una casa. La casa también cambia de posición.

Se puede hacer que los puntos **rotén** un ángulo  $\theta$  con respecto al origen. Una rotación se define matemáticamente como

$$x' = x \cdot \cos\theta - y \cdot \sin\theta \quad y' = x \cdot \sin\theta + y \cdot \cos\theta. \quad (5.6)$$

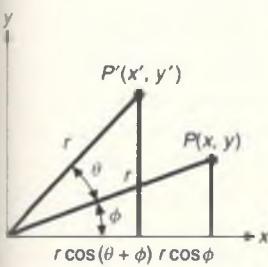
En forma matricial tenemos

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \quad o \quad P' = R \cdot P, \quad (5.7)$$

donde  $R$  es la matriz de rotación en la ecuación (5.7). En la figura 5.5 se presenta la rotación de la casa en  $45^\circ$ . Como ocurre con el escalamiento, la rotación se lleva a cabo con respecto al origen; la rotación con respecto a un punto arbitrario se analiza en la sección 5.3.

Los ángulos positivos se miden en **sentido contrario al giro de las manecillas del reloj**, desde  $x$  hacia  $y$ . En el caso de ángulos positivos (en el sentido del giro de las manecillas), se pueden usar las identidades  $\cos(-\theta) = \cos\theta$  y  $\sin(-\theta) = -\sin\theta$  para modificar las ecuaciones (5.6) y (5.7).

La ecuación (5.6) se obtiene fácilmente de la figura 5.6, en la cual una rotación de  $\theta$  transforma  $P(x, y)$  a  $P'(x', y')$ . Como la rotación es con respecto al origen, las distancias del origen a  $P$  y de  $P$  a  $P'$ , marcadas  $r$  en la figura, son iguales. Con medios trigonométricos simples encontramos que



**Figura 5.6**  
Obtención de la ecuación de rotación

$$x = r \cdot \cos\phi, \quad y = r \cdot \sin\phi \quad (5.8)$$

y que

$$\begin{aligned} x' &= r \cdot \cos(\theta + \phi) = r \cdot \cos\phi \cdot \cos\theta - r \cdot \sin\phi \cdot \sin\theta, \\ y' &= r \cdot \sin(\theta + \phi) = r \cdot \cos\phi \cdot \sin\theta + r \cdot \sin\phi \cdot \cos\theta. \end{aligned} \quad (5.9)$$

Al sustituir la ecuación (5.8) en la ecuación (5.9) se obtiene la ecuación (5.6).

## 5.3 Coordenadas homogéneas y representación matricial de transformaciones bidimensionales

Las representaciones matriciales para la traslación, el escalamiento y la rotación son, respectivamente,

$$P' = T \cdot P, \quad (5.3)$$

$$P' = S \cdot P, \quad (5.5)$$

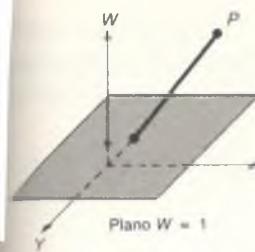
$$P' = R \cdot P. \quad (5.7)$$

Por desgracia, la traslación se trata en forma diferente que el escalamiento y la rotación. Nos gustaría poder tratar las tres transformaciones de una manera consistente, para que se pudieran combinar con facilidad.

Si los puntos se expresan con **coordenadas homogéneas**, las tres transformaciones se pueden tratar como multiplicaciones. Las coordenadas homogéneas fueron desarrolladas por primera vez en la geometría [MAXW46; MAXW51] y se han aplicado en la graficación [ROBE65; BLIN77b; BLIN78a]. Varios paquetes de subrutinas gráficas y procesadores de dibujo trabajan con coordenadas y transformaciones homogéneas.

En las coordenadas homogéneas, se añade una tercera coordenada a un punto. En lugar de representar un punto con un par de números  $(x, y)$ , se representa con tres,  $(x, y, W)$ . Al mismo tiempo, se dice que dos conjuntos de coordenadas homogéneas  $(x, y, W)$  y  $(x', y', W')$  representan el mismo punto si y sólo si uno es múltiplo del otro. De esta manera,  $(2, 3, 6)$  y  $(4, 6, 12)$  son los mismos puntos representados con diferentes triples de coordenadas. Es decir, cada punto tiene varias representaciones con coordenadas homogéneas. Además, al menos una de las coordenadas homogéneas debe ser distinta de cero: no se permite  $(0, 0, 0)$ . Si la coordenada  $W$  es diferente de cero, es posible usarla como divisor:  $(x, y, W)$  representa el mismo punto que  $(x/W, y/W, 1)$ . Por lo general, esta división se lleva a cabo cuando  $W$  es distinto de cero y los números  $x/W$  y  $y/W$  se denominan coordenadas cartesianas del punto homogéneo. Los puntos con  $W = 0$  se conocen como puntos en el infinito y no aparecerán con frecuencia en nuestro análisis.

Los triples de coordenadas usualmente representan puntos en el espacio tridimensional, pero aquí los usamos para representar puntos en el espacio bidimensional. La relación es la siguiente: Si tomamos todos los triples que representan un mismo punto, es decir, todos los triples de la forma  $(tx, ty, tW)$ , con  $t \neq 0$ , obtenemos una línea en el espacio tridimensional. Por lo tanto, cada punto homogéneo representa una *línea* en el espacio tridimensional. Si homogeneizamos el punto (lo dividimos entre  $W$ ), obtenemos un punto de la forma  $(x, y, 1)$ . Por consiguiente, los puntos homogeneizados constituyen el plano



**Figura 5.7**  
El espacio de coordenadas homogéneas  $XYW$ , con el plano  $W = 1$  y el punto  $P(x, y, W)$  proyectado sobre el plano  $W = 1$ .

definido por la ecuación  $W = 1$  en el espacio  $(x, y, W)$ . En la figura 5.7 se muestra esta relación. Los puntos en el infinito no están representados en el plano.

Como los puntos son vectores columna tridimensionales, las matrices de transformación, que multiplican un vector de punto para producir otro vector de punto, deben ser de  $3 \times 3$ . En la forma matricial de  $3 \times 3$  para las coordenadas homogéneas, las ecuaciones de traslación (5.1) son

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \quad (5.10)$$

Debemos advertirle que en algunos libros de texto sobre graficación, incluyendo [FOLE82], se usa la convención de premultiplicar las matrices por los vectores fila en lugar de postmultiplicar por los vectores columna. Para pasar de una convención a la otra hay que transponer las matrices, así como se transponen los vectores fila y columna:

$$(P \cdot M)^T = M^T \cdot P^T.$$

La ecuación (5.10) se puede representar de otra manera como

$$P' = T(d_x, d_y) \cdot P, \quad (5.11)$$

donde

$$T(d_x, d_y) = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix}. \quad (5.12)$$

¿Qué sucede si el punto  $P$  es trasladado por  $T(d_{x1}, d_{y1})$  a  $P'$  y luego por  $T(d_{x2}, d_{y2})$  a  $P''$ ? El resultado que esperamos intuitivamente es una traslación neta  $T(d_{x1} + d_{x2}, d_{y1} + d_{y2})$ . Para confirmarlo, comenzamos con lo que conocemos:

$$P' = T(d_{x1}, d_{y1}) \cdot P, \quad (5.13)$$

$$P'' = T(d_{x2}, d_{y2}) \cdot P'. \quad (5.14)$$

Ahora se sustituye la ecuación (5.13) en la ecuación (5.14) para obtener

$$P'' = T(d_{x2}, d_{y2}) \cdot (T(d_{x1}, d_{y1}) \cdot P) = (T(d_{x2}, d_{y2}) \cdot T(d_{x1}, d_{y1})) \cdot P. \quad (5.15)$$

El producto de matrices  $T(d_{x2}, d_{y2}) \cdot T(d_{x1}, d_{y1})$  es

$$\begin{bmatrix} 1 & 0 & d_{x2} \\ 0 & 1 & d_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & d_{x1} \\ 0 & 1 & d_{y1} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_{x1} + d_{x2} \\ 0 & 1 & d_{y1} + d_{y2} \\ 0 & 0 & 1 \end{bmatrix}. \quad (5.16)$$

La traslación neta es en realidad  $T(d_{x1} + d_{x2}, d_{y1} + d_{y2})$ . El producto de matrices también se conoce como **combinación**, **catenación**, **concatenación** o **composición** de  $T(d_{x1}, d_{y1})$  y  $T(d_{x2}, d_{y2})$ . En este libro normalmente usaremos el término **composición**.

En forma similar, las ecuaciones de escalamiento (5.4) se presentan en forma matricial como

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \quad (5.17)$$

Definiendo

$$S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad (5.18)$$

obtenemos

$$P' = S(s_x, s_y) \cdot P. \quad (5.19)$$

Así como las traslaciones sucesivas son aditivas, podemos esperar que los escalamientos sucesivos sean multiplicativos. Dado

$$P' = S(s_{x1}, s_{y1}) \cdot P, \quad (5.20)$$

$$P'' = S(s_{x2}, s_{y2}) \cdot P'. \quad (5.21)$$

y, sustituyendo la ecuación (5.20) en la ecuación (5.21), se obtiene

$$P'' = S(s_{x2}, s_{y2}) \cdot (S(s_{x1}, s_{y1}) \cdot P) = (S(s_{x2}, s_{y2}) \cdot (S(s_{x1}, s_{y1})) \cdot P). \quad (5.22)$$

El producto de matrices  $S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1})$  es

$$\begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (5.23)$$

Por lo tanto, los escalamientos son multiplicativos.

Por último, las ecuaciones de rotación (5.6) se pueden representar como

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \quad (5.24)$$

Si

$$R(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (5.25)$$

tenemos

$$P' = R(\theta) \cdot P. \quad (5.26)$$

La demostración de que dos rotaciones sucesivas son aditivas queda como actividad para usted en el ejercicio 5.2.

Considere como vectores las dos filas de la submatriz superior izquierda de  $2 \times 2$  en la ecuación (5.25). Se puede demostrar que los vectores tienen tres propiedades:

1. Cada uno es un vector unidad.
2. Cada uno es perpendicular al otro (su producto punto es cero).
3. El primer y el segundo vectores se rotarán por  $R(\theta)$  para que caigan sobre los ejes  $x$  y  $y$  positivos, respectivamente (en presencia de las condiciones 1 y 2, esta propiedad equivale a que la submatriz tenga determinante de 1).

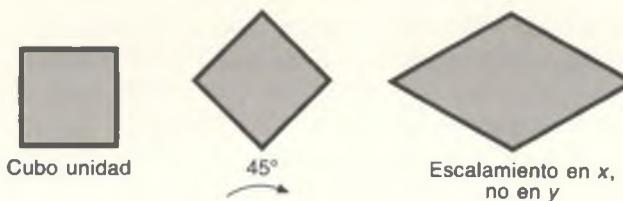
Las dos primeras propiedades también son verdaderas para las columnas de la submatriz de  $2 \times 2$ . Las dos direcciones son aquellas en las cuales los vectores se hacen girar sobre los ejes positivos  $x$  y  $y$ . Estas propiedades sugieren dos maneras útiles de obtener una matriz de rotación cuando se conoce el efecto deseado de la rotación. Una matriz que tiene estas propiedades se denomina **ortogonal especial**.

Una matriz de transformación de la forma

$$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}, \quad (5.27)$$

donde la submatriz superior de  $2 \times 2$  es ortogonal, conserva los ángulos y las longitudes. Es decir, un cuadrado unidad permanece como cuadrado unidad y no se convierte en un rombo con lados iguales a la unidad ni en un cuadrado con lados distintos de la unidad. Estas transformaciones se denominan de **cuerpo rígido**, ya que el cuerpo u objeto que se transforma no se distorsiona de ninguna manera. Una secuencia arbitraria de matrices de rotación y traslación crea una matriz de esta forma.

¿Qué se puede decir acerca del producto de una secuencia arbitraria de matrices de rotación, traslación y escalamiento? Se denominan **transformaciones afines** y tienen la propiedad de conservar el paralelismo de las líneas pero no las longitudes ni los ángulos. En la figura 5.8 se presenta el resultado de aplicar una



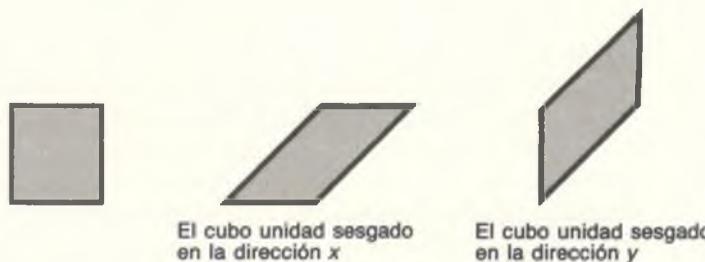
**Figura 5.8** Un cubo unidad se hace girar  $-45^\circ$  y se escala de manera no uniforme. El resultado es una transformación afín del cubo unidad, en la cual se mantiene el paralelismo de las líneas pero no los ángulos ni las longitudes.

rotación de  $-45^\circ$  y luego escalamiento no uniforme al cubo unidad. Es obvio que esta secuencia no ha conservado ni los ángulos ni las longitudes, pero las líneas paralelas aún son paralelas. La aplicación adicional de operaciones de rotación, escalamiento y traslación no ocasionará que las líneas paralelas dejen de serlo.  $R(\theta)$ ,  $S(s_x, s_y)$  y  $T(d_x, d_y)$  también son afines.

Otro tipo de transformación primitiva, llamada **transformación de sesgo**, también es afín. Las transformaciones de sesgo bidimensional son de dos tipos: sesgo sobre el eje  $x$  y sesgo sobre el eje  $y$ . En la figura 5.9 se muestra el efecto de sesgar el cubo unidad sobre cada eje. La operación se representa con la matriz

$$S H_x = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (5.28)$$

El término  $a$  en la matriz de sesgo es la constante de proporcionalidad. Observe que el producto  $S H_x [x \ y \ 1]^T$  es  $[x + ay \ y \ 1]^T$ , lo que demuestra con claridad el cambio proporcional en  $x$  como función de  $y$ .



**Figura 5.9** Operaciones primitivas de sesgo aplicadas al cubo unidad. En ambos casos, la longitud de las líneas oblicuas es ahora mayor que 1.

En forma similar, la matriz

$$S H_x = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.29)$$

se sesga sobre el eje  $y$ .

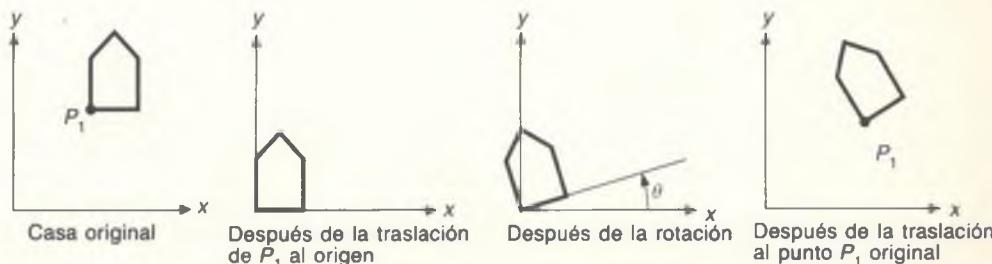
## 5.4 Composición de transformaciones bidimensionales

La idea de la composición se planteó en la sección 5.3. En esta sección usaremos la composición para combinar las matrices fundamentales  $R$ ,  $S$  y  $T$  para producir resultados generales deseados. El propósito básico de la composición de transformaciones es ganar eficiencia al aplicar a un punto una sola transformación compuesta, en lugar de aplicar una serie sucesiva de transformaciones.

Considere la rotación de un objeto con respecto a un punto arbitrario  $P_1$ . Como sólo sabemos rotar con respecto al origen, podemos convertir nuestro problema original (difícil) a tres problemas separados (sencillos). Así, para rotar con respecto a  $P_1$ , necesitamos una secuencia de tres transformaciones fundamentales:

1. Trasladar  $P_1$  para que quede en el origen.
2. Rotar.
3. Trasladar para que el punto en el origen regrese a  $P_1$ .

Esta secuencia se ilustra en la figura 5.10, en la cual la casa se rota con respecto a  $P_1(x_1, y_1)$ . La primera traslación es  $(-x_1, -y_1)$ , mientras que la segunda es la inversa,  $(x_1, y_1)$ . El resultado es bastante diferente del que se obtiene al aplicar sólo la rotación.



**Figura 5.10** Rotación, de una casa con respecto al punto  $P_1$ , con un ángulo  $\theta$ .

La transformación neta es

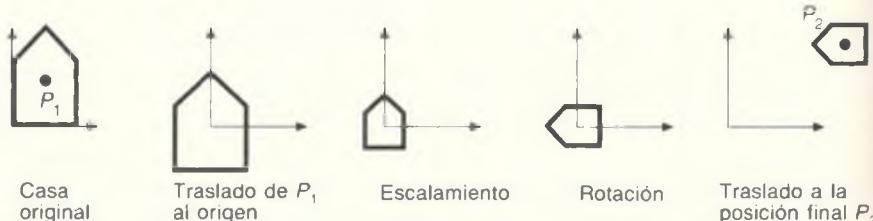
$$\begin{aligned}
 T(x_1, y_1) \cdot R(\theta) \cdot T(-x_1, -y_1) &= \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} \cos\theta & -\sin\theta & x_1(1 - \cos\theta) + y_1\sin\theta \\ \sin\theta & \cos\theta & y_1(1 - \cos\theta) - x_1\sin\theta \\ 0 & 0 & 1 \end{bmatrix} \quad (5.30)
 \end{aligned}$$

Se utiliza una estrategia similar para escalar un objeto con respecto a un punto arbitrario  $P_1$ . Primero se hace el traslado de manera que  $P_1$  pase al origen, después se escala y luego se traslada de regreso a  $P_1$ . En este caso, la transformación neta es

$$\begin{aligned}
 T(x_1, y_1) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1) &= \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} s_x & 0 & x_1(1 - s_x) \\ 0 & s_y & y_1(1 - s_y) \\ 0 & 0 & 1 \end{bmatrix} \quad (5.31)
 \end{aligned}$$

Suponga que queremos escalar, rotar y ubicar la casa presentada en la figura 5.11 usando  $P_1$  como centro para la rotación y el escalamiento. La secuencia es trasladar  $P_1$  al origen, efectuar el escalamiento y la rotación, y luego trasladar del origen a la nueva posición  $P_2$  donde se ubicará la casa. Una estructura de datos que registre esta información podría contener el factor o factores de escalamiento, el ángulo de rotación y las cantidades de traslación, así como el orden de aplicación de las transformaciones, o bien podría registrar únicamente la matriz de transformación compuesta:

$$T(x_2, y_2) \cdot R(\theta) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1) \quad (5.32)$$



**Figura 5.11** Rotación de una casa con respecto al punto  $P_1$ , y ubicación de manera que lo que estaba en  $P_1$  quede en  $P_2$ .

Si  $M_1$  y  $M_2$  representan una traslación, un escalamiento o una rotación fundamental, ¿cuándo es  $M_1 \cdot M_2 = M_2 \cdot M_1$ ? Es decir, ¿cuándo conmutan  $M_1$  y  $M_2$ ? Por supuesto, en general la multiplicación de matrices *no* es conmutativa. Sin embargo, es fácil demostrar que la conmutatividad existe en los siguientes casos especiales:

$M_1$	$M_2$
Traslación	Traslación
Escalamiento	Escalamiento
Rotación	Rotación
Escalamiento (con $s_x = s_y$ )	Rotación

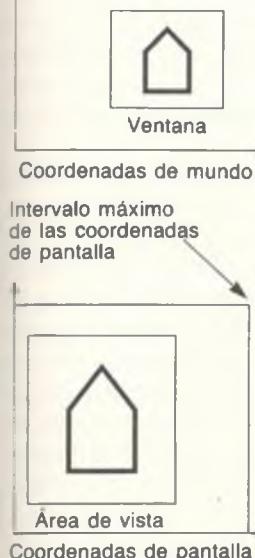
En estos casos no hay que preocuparse por el *orden* de la composición de matrices.

## 5.5 Transformación ventana-área de vista

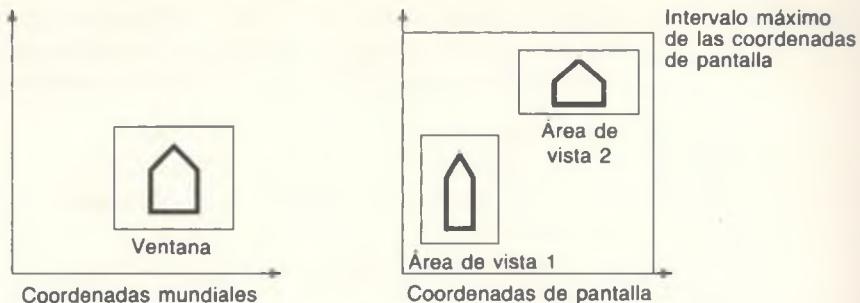
Algunos paquetes gráficos permiten que el programador especifique coordenadas de primitivas de salida en un sistema de **coordenadas de mundo** de punto flotante, usando las unidades que sean relevantes para el programa de aplicación: angstroms, micras, metros, millas, años luz, etcétera. Se emplea el término *de mundo* porque el programa de aplicación representa un mundo que se crea o presenta interactivamente para el usuario.

Como las primitivas de salida se expresan en coordenadas de mundo, hay que indicar al paquete de subrutinas gráficas cómo establecer la correspondencia entre las coordenadas de mundo y las coordenadas de pantalla (usaremos el término específico *coordenadas de pantalla* para relacionar este análisis específicamente con SRGP, pero podrían usarse dispositivos de impresión, en cuyo caso sería más apropiado el término *coordenadas de dispositivo*). Esta correspondencia se puede efectuar si el programador de la aplicación proporciona al paquete gráfico una matriz de transformación para la correspondencia. Otra forma es que el programador de la aplicación especifique una región rectangular en coordenadas de mundo, llamada **ventana de coordenadas mundiales** y una región rectangular correspondiente en coordenadas de pantalla, llamada **área de vista**, con la cual se establece la correspondencia de la ventana de coordenadas mundiales. La transformación que establece la correspondencia entre la ventana y el área de vista se aplica a todas las primitivas de salida en coordenadas de mundo para que correspondan a coordenadas de pantalla. Este concepto se presenta en la figura 5.12. Como se puede ver, si la ventana y el área de vista no tienen la misma razón altura-ancho, ocurre un escalamiento *no uniforme*. Si el programa de aplicación cambia la ventana o el área de vista, las nuevas primitivas de salida que se dibujen en la pantalla se verán afectadas por el cambio, no así las primitivas existentes.

El modificador *coordenadas de mundo* se emplea con *ventana* para subrayar que no se trata de una *ventana de administrador de ventanas*, un concepto distinto y más reciente que por desgracia tiene el mismo nombre. Este modificador se omitirá cuando no exista ninguna ambigüedad con respecto al tipo de ventana que se trate.



**Figura 5.12**  
La ventana en coordenadas mundo y el área de vista en coordenadas de pantalla determinan la correspondencia que se aplica a todas las primitivas de salida en coordenadas de mundo.



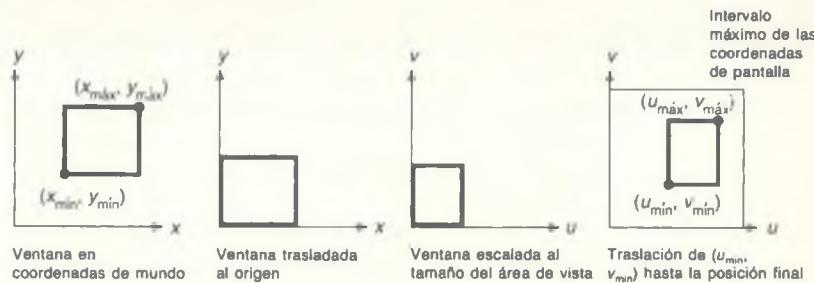
**Figura 5.13** Efecto de dibujar primitivas de salida con dos áreas de vista. Las primitivas de salida que especifican la casa se dibujaron primero con el área de vista 1, que luego se cambió al área de vista 2, para lo cual el programa de aplicación invocó nuevamente al paquete gráfico para dibujar las primitivas de salida.

Si SRGP proporcionara primitivas de salida en coordenadas de mundo, el área de vista se hallaría en el lienzo actual, que por omisión es el lienzo 0, la pantalla. El programa de aplicación podría cambiar en cualquier instante la ventana o el área de vista, en cuyo caso las primitivas de salida que se especificaran subsecuentemente estarían sujetas a una nueva transformación. Si el cambio incluyera un área de vista distinta, las nuevas primitivas de salida se colocarían en el lienzo en posiciones distintas a las anteriores, como se ilustra en la figura 5.13.

Un administrador de ventanas podría establecer la correspondencia entre el lienzo 0 de SRGP y una ventana menor que la pantalla completa; en este caso, no siempre estará visible todo el lienzo ni toda el área de vista.

Si tenemos una ventana y un área de vista, ¿cuál es la matriz de transformación que establece la correspondencia entre las coordenadas mundiales de la ventana y las coordenadas de pantalla del área de vista? Esta matriz se puede desarrollar como una composición de transformación de tres pasos, como se sugiere en la figura 5.14. La ventana, especificada por sus vértices inferior izquierdo y superior derecho, se traslada primero al origen de las coordenadas de mundo. Después se escala el tamaño de la ventana para que sea igual al tamaño del área de vista. Por último, se usa una traslación para colocar el área de vista. La matriz global,  $M_{vv}$ , que se obtiene por medio de la composición de dos matrices de traslación y la matriz de escalamiento, es

$$\begin{aligned}
 M_{vv} &= T(u_{\min}, v_{\min}) \cdot S \left( \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}}, \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} \right) \cdot T(-x_{\min}, -y_{\min}) \\
 &= \begin{bmatrix} 1 & 0 & u_{\min} \\ 0 & 1 & v_{\min} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} & 0 & 0 \\ 0 & \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_{\min} \\ 0 & 1 & -y_{\min} \\ 0 & 0 & 1 \end{bmatrix}
 \end{aligned}$$



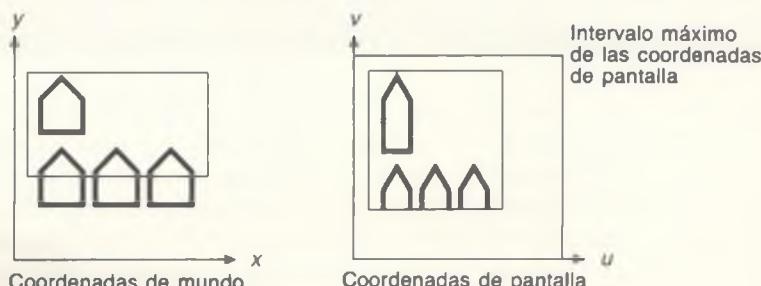
**Figura 5.14** Pasos utilizados en la transformación de una ventana de coordenadas mundiales a un área de vista.

$$= \begin{bmatrix} \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} & 0 & -x_{\min} + \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} + u_{\min} \\ 0 & \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} & -y_{\min} + \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} + v_{\min} \\ 0 & 0 & 1 \end{bmatrix}. \quad (5.33)$$

Al multiplicar  $P = M_{vv}[x \ y \ 1]^T$  se obtiene el resultado esperado:

$$P = \left[ (x - x_{\min}) \cdot \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} + u_{\min} \quad (y - y_{\min}) \cdot \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} + v_{\min} \quad 1 \right]. \quad (5.34)$$

Muchos paquetes gráficos combinan la transformación ventana-área de vista con el recorte de primitivas de salida con respecto a la ventana. El concepto de recorte se presentó en el capítulo 3; en la figura 5.15 se ilustra el recorte en el contexto de ventanas y áreas de vista.



**Figura 5.15** Las primitivas de salida en coordenadas mundiales se recortan con respecto a la ventana. Las que permanecen se presentan en el área de vista.

## 5.6 Eficiencia

La composición más general de las operaciones  $R$ ,  $S$  y  $T$  produce una matriz de la forma

$$M = \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}. \quad (5.35)$$

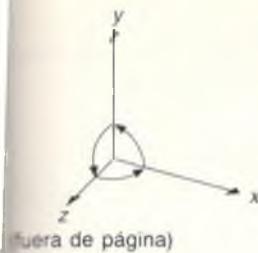
La submatriz superior de  $2 \times 2$  es una matriz compuesta de rotación y escalamiento, mientras que  $t_x$  y  $t_y$  son traslaciones compuestas. Para calcular  $M \cdot P$  como un vector multiplicado por una matriz de  $3 \times 3$  se requieren nueve multiplicaciones y seis sumas. Sin embargo, la estructura fija de la última fila de la ecuación (5.35) simplifica las operaciones reales a

$$\begin{aligned} x' &= x \cdot r_{11} + y \cdot r_{12} + t_x, \\ y' &= x \cdot r_{21} + y \cdot r_{22} + t_y, \end{aligned} \quad (5.36)$$

con lo cual el proceso se reduce a cuatro multiplicaciones y cuatro sumas, una mejora considerable, sobre todo si consideramos que la operación se puede aplicar a cientos o incluso a miles de puntos por imagen. Por lo tanto, aunque las matrices de  $3 \times 3$  sean convenientes y útiles para la composición de transformaciones bidimensionales, podemos usar la matriz final de manera más eficaz en un programa si aprovechamos su estructura especial. Algunos multiplicadores de matrices en hardware tienen sumadores y multiplicadores paralelos para reducir o eliminar esta preocupación.

## 5.7 Representación matricial de transformaciones tridimensionales

Así como las transformaciones bidimensionales se pueden representar con matrices de  $3 \times 3$  usando coordenadas homogéneas, las transformaciones tridimensionales se pueden representar con matrices de  $4 \times 4$ , siempre y cuando usemos representaciones de coordenadas homogéneas de los puntos en el espacio tridimensional. Así, en lugar de representar un punto como  $(x, y, z)$ , lo hacemos como  $(x, y, z, W)$ , donde dos de estos cuádruplos representan el mismo punto si uno es un multiplicador distinto de cero del otro; no se permite el cuádruplo  $(0, 0, 0, 0)$ . Como sucede en el espacio bidimensional, la representación estándar de un punto  $(x, y, z, W)$  con  $W \neq 0$  se indica con  $(x/W, y/W, z/W)$ . La transformación de un punto a esta forma se denomina **homogeneización**.



igual que antes. Además, los puntos cuya coordenada  $W$  es cero se llaman puntos en el infinito. También existe una interpretación geométrica. Cada punto en el espacio tridimensional se representa con una línea que pasa por el origen en el espacio de cuatro dimensiones, y las representaciones homogeneizadas de estos puntos forman un subespacio tridimensional de un espacio de cuatro dimensiones definido por la ecuación  $W = 1$ .

El sistema de coordenadas tridimensionales que se emplea en este libro es de **mano derecha**, como se ilustra en la figura 5.16. Por convención, las rotaciones positivas en el sistema de mano derecha son tales que, al ver hacia un eje positivo desde el origen, una rotación de  $90^\circ$  *en sentido contrario al del giro de las manecillas del reloj* transformará un eje positivo en otro. La tabla siguiente se desprende de esta convención:

Eje de rotación	Dirección de la rotación positiva
$x$	$y$ a $z$
$y$	$z$ a $x$
$z$	$x$ a $y$

$x$	$y$ a $z$
$y$	$z$ a $x$
$z$	$x$ a $y$

Estas direcciones positivas también se ilustran en la figura 5.16. Tenga presente que no todos los libros sobre graficación siguen esta convención.

Usamos el sistema de mano derecha porque se trata de una convención matemática estándar, aunque en la graficación tridimensional es conveniente pensar en un sistema de mano izquierda sobrepuerto a la pantalla (véase la Fig. 5.17), ya que un sistema de mano izquierda da la interpretación natural de que los valores mayores de  $z$  se encuentran más lejos del observador. Observe que en un sistema de mano izquierda, las rotaciones positivas son *en el sentido del giro de las manecillas del reloj* cuando se observa desde un eje positivo hacia el origen. Esta definición de las rotaciones positivas permite que las matrices de rotación que se presentan en esta sección puedan usarse para sistemas de coordenadas de mano derecha o izquierda. La conversión de derecha a izquierda y viceversa se presenta en la sección 5.9.

La traslación en el espacio tridimensional es una simple extensión de la que se lleva a cabo en el espacio bidimensional:

$$T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.37)$$

Es decir,  $T(d_x, d_y, d_z) \cdot [x \ y \ z \ 1]^T = [x + d_x \ y + d_y \ z + d_z \ 1]^T$ .

El escalamiento se extiende en forma similar:

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.38)$$

Al revisar se observa que  $S(s_x, s_y, s_z) \cdot [x \ y \ z \ 1]^T = [s_x \cdot x \ s_y \cdot y \ s_z \cdot z \ 1]^T$ .

La rotación bidimensional de la ecuación (5.26) es simplemente una rotación tridimensional con respecto al eje  $z$ , que es

$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.39)$$

Es fácil verificar esta observación: una rotación de  $90^\circ$  de  $[1 \ 0 \ 0 \ 1]^T$ , que es el vector unidad sobre el eje  $x$ , debe producir el vector unidad  $[0 \ 1 \ 0 \ 1]^T$  sobre el eje  $y$ . Al evaluar el producto

$$\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad (5.40)$$

se obtiene el resultado previsto de  $[0 \ 1 \ 0 \ 1]^T$ .

La matriz de rotación del eje  $x$  es

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.41)$$

La matriz de rotación del eje  $y$  es

$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.42)$$

Las columnas (y las filas) de la submatriz superior izquierda de  $3 \times 3$  de  $R_z(\theta)$ ,  $R_x(\theta)$  y  $R_y(\theta)$  son vectores unidad mutuamente perpendiculares y el determinante de la submatriz es 1, lo que quiere decir que las tres matrices son ortogonales especiales, como se vio en la sección 5.3. Además, la submatriz superior izquierda de  $3 \times 3$  formada por una secuencia arbitraria de rotaciones es ortogonal especial. Recuerde que las transformaciones ortogonales conservan las distancias y los ángulos.

Todas estas matrices de transformación tienen inversas. La inversa de  $T$  se obtiene cambiando el signo de  $d_x$ ,  $d_y$  y  $d_z$ ; la de  $S$ , reemplazando  $s_x$ ,  $s_y$  y  $s_z$  por sus recíprocos; y la inversa de cada una de las tres matrices de rotación, negando el ángulo de rotación.

La inversa de una matriz ortogonal  $B$  es la transpuesta de  $B$ :  $B^{-1} = B^T$ . De hecho, para tomar la transpuesta no es necesario intercambiar los elementos del arreglo que contiene a la matriz: basta intercambiar los índices de las filas y las columnas al acceder a la matriz. Observe que este método para hallar la inversa es consistente con el resultado del cambio de signo de  $\theta$  para hallar la inversa de  $R_x$ ,  $R_y$  y  $R_z$ .

Es posible multiplicar juntas cualquier cantidad de matrices de rotación, escalamiento y traslación. El resultado siempre tiene la forma

$$M = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (5.43)$$

Como en el caso bidimensional, la submatriz superior izquierda de  $3 \times 3 R$  nos da la rotación y el escalamiento combinados, mientras que  $T$  nos da la traslación subsecuente. Podemos obtener mayor eficiencia computacional si efectuamos la transformación en forma explícita, como

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} + T, \quad (5.44)$$

donde  $R$  y  $T$  son submatrices de la ecuación (5.43).

Existen tres matrices de sesgo tridimensional correspondientes a las matrices de sesgo bidimensional presentadas en la sección 5.2. El sesgo  $(x, y)$  es

$$SH_{xy}(sh_x, sh_y) = \begin{bmatrix} 1 & 0 & sh_z & 0 \\ 0 & 1 & sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (5.45)$$

Al aplicar  $SH_{xy}$  al punto  $[x \ y \ z \ 1]^T$  se obtiene  $[x + sh_x \cdot z \ y + sh_y \cdot z \ 1]^T$ . Los sesgos sobre los ejes  $x$  y  $y$  tienen forma similar.

Hasta ahora hemos centrado nuestra atención en la transformación de puntos individuales. La transformación de líneas, definidas éstas por dos puntos, se obtiene transformando los puntos extremos. Los planos, si están definidos por tres puntos, se pueden manejar de la misma forma, pero por lo general se definen con una ecuación de plano y los coeficientes de esta ecuación deben transformarse de manera distinta. También puede ser necesario transformar la normal al plano. Representemos un plano como el vector columna de los coeficientes de la ecuación de plano  $N = [A \ B \ C \ D]^T$ . Entonces, un plano está definido por todos los puntos  $P$  tales que  $N \cdot P = 0$ , donde el símbolo “ $\cdot$ ” es el producto punto vectorial y  $P = [x \ y \ z \ 1]^T$ . Este producto punto da lugar a la

conocida ecuación del plano  $Ax + By + Cz + D = 0$ , que también se puede expresar como el producto del vector fila de los coeficientes de la ecuación del plano multiplicado por el vector columna  $P$ :  $N^T \cdot P = 0$ . Suponga ahora que transformamos todos los puntos  $P$  en el plano con una matriz  $M$ . Para mantener  $N^T \cdot P = 0$  para todos los puntos transformados, quisiéramos transformar  $N$  por una matriz  $Q$  (por determinarse) que dé lugar a la ecuación  $(Q \cdot N)^T \cdot M \cdot P = 0$ . A su vez, esta ecuación se puede reescribir como  $N^T \cdot Q^T \cdot M \cdot P = 0$  usando la identidad  $(Q \cdot N)^T = N^T \cdot Q^T$ . La ecuación será verdadera si  $Q^T \cdot M$  es múltiplo de la matriz identidad. Si el multiplicador es 1, esta situación nos lleva a  $Q^T = M^{-1}$  o  $Q = (M^{-1})^T$ . Por lo tanto, el vector columna  $N'$  de coeficientes de un plano transformado por  $M$  se expresa como

$$N' = (M^{-1})^T \cdot N \quad (5.46)$$

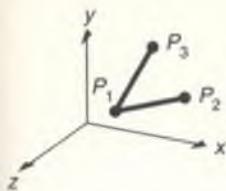
Por lo general no es necesario que exista la matriz  $(M^{-1})^T$ , ya que el determinante de  $M$  puede ser cero. Esta situación ocurriría si  $M$  incluye una proyección (quizás queríamos investigar el efecto de una proyección de perspectiva sobre el plano).

Si sólo se transformará la normal al plano (p. ej., para efectuar los cálculos de sombreado que se analizan en el capítulo 14) y si  $M$  sólo consiste en una composición de matrices de traslación, rotación y escalamiento uniforme, entonces se simplifican las matemáticas. La  $N'$  de la ecuación (5.46) se puede simplificar a  $[A' \ B' \ C' \ 0]^T$ . (Con un componente  $W$  igual a cero, un punto homogéneo representa un punto en el infinito, lo cual se puede considerar como una dirección.)

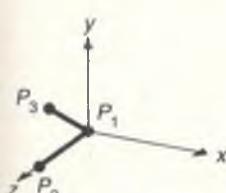
## 5.8 Composición de transformaciones tridimensionales

En esta sección analizaremos la forma de componer matrices de transformación tridimensionales usando un ejemplo que será de utilidad en la sección 6.5. El objetivo es transformar los segmentos de línea dirigida  $P_1P_2$  y  $P_1P_3$  en la figura 5.18 de su posición inicial en la parte (a) a su posición final en la parte (b). De esta manera, el punto  $P_1$  se trasladará al origen,  $P_1P_2$  quedará en el eje z positivo y  $P_1P_3$  quedará en la mitad del eje y positivo del plano ( $y, z$ ). Las longitudes de las líneas no se verán afectadas por la transformación.

Se presentan dos formas de lograr la transformación deseada. El primer método es componer las transformaciones primitivas  $T$ ,  $R_x$ ,  $R_y$  y  $R_z$ . Este método, aunque algo tedioso, es fácil de ilustrar y su comprensión nos ayudará en nuestro conocimiento de las transformaciones. El segundo método, que utiliza las propiedades de las matrices ortogonales especiales que analizamos en la sección 5.7, se explica de manera más breve pero es más abstracto.



(a) Posición inicial



(b) Posición final

**Figura 5.18**

Transformación de  $P_1$ ,  $P_2$  y  $P_3$  de su posición inicial (a) a su posición final (b).

Para trabajar con las transformaciones primitivas, de nuevo dividimos un problema difícil en varios más sencillos. En este caso, la transformación deseada se puede realizar en cuatro pasos:

1. Traslación de  $P_1$  al origen.
2. Rotación sobre el eje  $y$  para que  $P_1P_2$  esté en el plano  $(y, z)$ .
3. Rotación sobre el eje  $x$  para que  $P_1P_2$  esté en el eje  $z$ .
4. Rotación sobre el eje  $z$  para que  $P_1P_3$  esté en el plano  $(y, z)$ .

**Paso 1: Traslación  $P_1$  al origen.** La traslación es

$$T(-x_1, -y_1, -z_1) = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (5.47)$$

Al aplicar  $T$  a  $P_1$ ,  $P_2$  y  $P_3$  se obtiene

$$P'_1 = T(-x_1, -y_1, -z_1) \cdot P_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad (5.48)$$

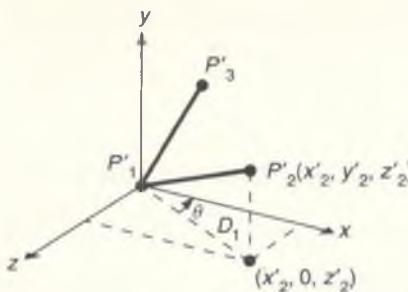
$$P'_2 = T(-x_1, -y_1, -z_1) \cdot P_2 = \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \\ 1 \end{bmatrix}, \quad (5.49)$$

$$P'_3 = T(-x_1, -y_1, -z_1) \cdot P_3 = \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \\ 1 \end{bmatrix}, \quad (5.50)$$

**Paso 2: Rotación sobre el eje  $y$ .** En la figura 5.19 se muestra  $P_1P_2$  después del paso 1, así como la proyección de  $P_1P_2$  sobre el plano  $(x, z)$ . El ángulo de rotación es  $-(90 - \theta) = \theta - 90$ . Entonces,

$$\cos(\theta - 90) = \sin \theta = \frac{z'_2}{D_1} = \frac{z_2 - z_1}{D_1},$$

$$\sin(\theta - 90) = -\cos \theta = \frac{x'_2}{D_1} = -\frac{x_2 - x_1}{D_1}, \quad (5.51)$$



**Figura 5.19** Rotación sobre el eje y. La proyección de  $P_1'P_2'$ , cuya longitud es  $D_1$ , se hace rotar al eje z. El ángulo  $\theta$  muestra la dirección positiva de la rotación sobre el eje y. El ángulo que se usa en realidad es  $-(90 - \theta)$ .

donde

$$D_1 = \sqrt{(z'_2)^2 + (x'_2)^2} = \sqrt{(z_2 - z_1)^2 + (x_2 - x_1)^2}. \quad (5.52)$$

Al sustituir estos valores en la ecuación (5.42) se obtiene

$$P_2'' = R_y(\theta - 90) \cdot P_2' = [0 \ y_2 - y_1 \ D_1 \ 1]^T. \quad (5.53)$$

Como era de esperar, el componente x de  $P_2''$  es cero y el componente z es la longitud  $D_1$ .

**Paso 3: Rotación sobre el eje x.** En la figura 5.20 se muestra  $P_1P_2$  después del paso 2. El ángulo de rotación es  $\phi$ , para el cual

$$\cos \phi = \frac{z''_2}{D_2}, \quad \sin \phi = \frac{y''_2}{D_2}, \quad (5.54)$$

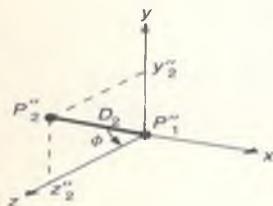
donde  $D_2 = |P_1''P_2''|$ , la longitud de la línea  $P_1''P_2''$ . Sin embargo, la longitud de la línea  $P_1''P_2''$  es igual a la longitud de la línea  $P_1P_2$ , ya que las transformaciones de rotación y traslación conservan la longitud; por lo tanto,

$$D_2 = |P_1''P_2''| = |P_1P_2| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}. \quad (5.55)$$

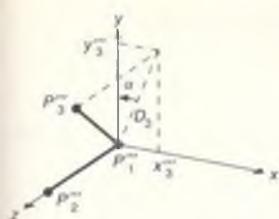
El resultado de la rotación en el paso 3 es

$$\begin{aligned} P_2''' &= R_x(\phi) \cdot P_2'' = R_x(\phi) \cdot R_y(\theta - 90) \cdot P_2' \\ &= R_x(\phi) \cdot R_y(\theta - 90) \cdot T \cdot P_2 = [0 \ 0 \ |P_1P_2| \ 1]^T. \end{aligned} \quad (5.56)$$

Es decir,  $P_1P_2$  coincide ahora con el eje z positivo.



**Figura 5.20** Rotación sobre el eje x:  $P_1P_2$  rota en el eje z con el ángulo positivo  $\phi$ .  $D_2$  es la longitud del segmento de línea. No se presenta el segmento de línea  $P_1''P_3''$ , ya que no se usa para determinar los ángulos de rotación. Las dos líneas se hacen rotar con  $R_x(\phi)$ .

**Figura 5.21**

Rotación sobre el eje  $z$ . El ángulo positivo hace rotar la proyección de  $P_1'P_3'$ , cuya longitud es  $D_3$ , en el eje  $y$ , con lo cual la línea llega al plano  $(y, z)$ .  $D_3$  es la longitud de la proyección.

**Paso 4: Rotación sobre el eje  $z$ .** En la figura 5.21 se muestran  $P_1P_2$  y  $P_1P_3$  después del paso 3, con  $P_2'''$  en el eje  $z$  y  $P_3'''$  en la posición

$$P_3''' = [x_3''' \ y_3''' \ z_3''' \ 1]^T = R_z(\phi) \cdot R_y(\theta - 90) \cdot T(-x_1, -y_1, -z_1) \cdot P_3. \quad (5.57)$$

La rotación a través del ángulo positivo  $\alpha$ , con

$$\cos \alpha = y_3'''/D_3, \quad \sin \alpha = x_3'''/D_3, \quad D_3 = \sqrt{x_3'''^2 + y_3'''^2}. \quad (5.58)$$

Con el paso 4 se obtiene el resultado que se presenta en la figura 5.18(b).

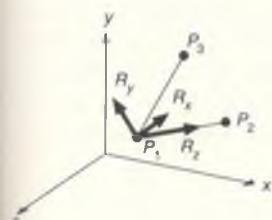
La matriz compuesta

$$R_z(\alpha) \cdot R_x(\phi) \cdot R_y(\theta - 90) \cdot T(-x_1, -y_1, -z_1) = R \cdot T. \quad (5.59)$$

es la transformación requerida, con  $R = R_z(\alpha) \cdot R_x(\phi) \cdot R_y(\theta - 90)$ . Dejaremos que usted aplique esta transformación a  $P_1$ ,  $P_2$  y  $P_3$  para verificar que  $P_1$  se transforme al origen,  $P_2$  se transforme al eje  $z$  positivo y que  $P_3$  se transforma a la mitad  $y$  positiva del plano  $(y, z)$ .

La segunda manera de obtener la matriz  $R$  es usar las propiedades de las matrices ortogonales, analizadas en la sección 5.3. Recuerde que los vectores fila unidad de  $R$  rotan hacia los ejes principales. Al reemplazar los segundos subíndices de la ecuación (5.43) con  $x$ ,  $y$  y  $z$  para que la notación sea más conveniente,

$$R = \begin{bmatrix} r_{1_x} & r_{2_x} & r_{3_x} \\ r_{1_y} & r_{2_y} & r_{3_y} \\ r_{1_z} & r_{2_z} & r_{3_z} \end{bmatrix}. \quad (5.60)$$

**Figura 5.22**

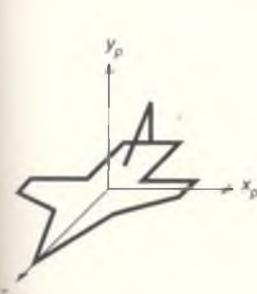
Los vectores unidad  $R_x$ ,  $R_y$  y  $R_z$ , que se transforman hacia los ejes principales.

Como  $R_z$  es el vector unidad sobre  $P_1P_2$  que rotará hacia el eje  $z$  positivo,

$$R_z = [r_{1_z} \ r_{2_z} \ r_{3_z}]^T = \frac{P_1P_2}{|P_1P_2|}. \quad (5.61)$$

Además, el vector unidad  $R_x$  es perpendicular al plano de  $P_1$ ,  $P_2$  y  $P_3$  y rotará hacia el eje  $x$  positivo, de manera que  $R_x$  debe ser el producto cruz normalizado de dos vectores en el plano:

$$R_x = [r_{1_x} \ r_{2_x} \ r_{3_x}]^T = \frac{P_1P_3 \times P_1P_2}{|P_1P_3 \times P_1P_2|}. \quad (5.62)$$



Finalmente,

$$R_y = [r_{1_y} \ r_{2_y} \ r_{3_y}]^T = R_z \times R_x \quad (5.63)$$

**Figura 5.23**

Un aeroplano en el sistema de coordenadas  $(x_p, y_p, z_p)$ .

rotará hacia el eje  $y$  positivo. La matriz compuesta se expresa como

$$\begin{bmatrix} r_{1_x} & r_{2_x} & r_{3_x} & 0 \\ r_{1_y} & r_{2_y} & r_{3_y} & 0 \\ r_{1_z} & r_{2_z} & r_{3_z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot T(-x_1, -y_1, -z_1) = R \cdot T, \quad (5.64)$$

donde  $R$  y  $T$  son como en la ecuación (5.59). En la figura 5.22 se muestran los vectores individuales  $R_x$ ,  $R_y$  y  $R_z$ .

Considere otro ejemplo. En la figura 5.23 se muestra un aeroplano definido en el sistema de coordenadas  $x_p$ ,  $y_p$ ,  $z_p$  y centrado en el origen. Queremos transformar el aeroplano para que apunte en la dirección indicada por el vector  $DDV$  (dirección de vuelo), esté centrado en  $p$  y no esté inclinado, como se muestra en la figura 5.24. La transformación necesaria para llevar a cabo esta reorientación consiste en una rotación para apuntar el aeroplano en la dirección correcta, seguida por una traslación del origen a  $P$ . Para hallar la matriz de rotación sólo hay que determinar en qué dirección apuntan los ejes  $x_p$ ,  $y_p$  y  $z_p$  en la figura 5.24, asegurarse de que las direcciones estén normalizadas y luego usar estas direcciones como vectores columna en una matriz de rotación.

El eje  $z_p$  debe transformarse a la dirección  $DDV$ , mientras que el eje  $x_p$  debe transformarse a un vector horizontal perpendicular a  $DDV$ , o sea, en la dirección  $y \times DDV$ , el producto cruz de  $y$  y  $DDV$ . La dirección  $y_p$  está indicada por  $z_p \times x_p = DDV \times (y \times DDV)$ , el producto cruz de  $z_p$  y  $x_p$ ; por lo tanto, las tres columnas de la matriz de rotación son los vectores normalizados  $|y \times DDV|$ ,  $|DDV \times (y \times DDV)|$  y  $|DDV|$ :

$$R = \begin{bmatrix} |y \times DOF| & |DOF \times (y \times DOF)| & |DOF| & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (5.65)$$

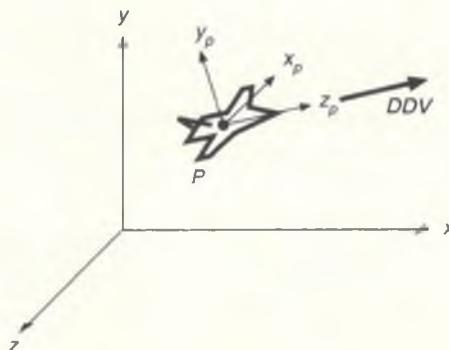


Figura 5.24 El aeroplano de la figura 5.23, ubicado en el punto  $P$  y apuntando en la dirección  $DDV$ .

La situación que se presenta si  $DDV$  se encuentra en la dirección del eje  $y$  es degenerada, ya que existe un conjunto infinito de vectores posibles para el vector horizontal. Esta degeneración se refleja en el álgebra, ya que los productos cruz  $y \times DDV$  y  $DDV \times (y \times DDV)$  son cero. En este caso especial,  $R$  no es una matriz de rotación.

## 5.9 Las transformaciones como un cambio en el sistema de coordenadas

Hasta ahora hemos analizado la transformación de un conjunto de puntos que pertenecen a un objeto para obtener otro conjunto de puntos, cuando los dos conjuntos se hallan en el mismo sistema de coordenadas. Con este método, el sistema de coordenadas permanece sin cambios y el objeto se transforma con respecto al origen del sistema de coordenadas. Una forma alternativa pero equivalente de considerar una transformación es como un cambio en el sistema de coordenadas. Esta perspectiva es útil cuando se combinan varios objetos, cada uno definido en su propio sistema de coordenadas locales, y deseamos expresar las coordenadas de estos objetos en un solo sistema global de coordenadas. Esta situación se hará presente en el capítulo 7.

Definamos  $M_{i \leftarrow j}$  como la transformación que convierte la representación de un punto en un sistema de coordenadas  $j$  a su representación en el sistema de coordenadas  $i$ .

Se define  $P^{(i)}$  como la representación de un punto en el sistema de coordenadas  $i$ ,  $P^{(j)}$  como la representación del punto en el sistema  $j$ , y  $P^{(k)}$  como la representación en el sistema de coordenadas  $k$ ; entonces,

$$P^{(i)} = M_{i \leftarrow j} \cdot P^{(j)} \text{ y } P^{(j)} = M_{j \leftarrow k} \cdot P^{(k)}. \quad (5.66)$$

Al sustituir se obtiene

$$P^{(i)} = M_{i \leftarrow j} \cdot M_{j \leftarrow k} \cdot P^{(k)} = M_{i \leftarrow k} \cdot P^{(k)}, \quad (5.67)$$

de manera que

$$M_{i \leftarrow k} = M_{i \leftarrow j} \cdot M_{j \leftarrow k}, \quad (5.68)$$

En la figura 5.25 se muestran cuatro sistemas de coordenadas. Por inspección se observa que la transformación del sistema de coordenadas 2 en 1 es  $M_{1 \leftarrow 2} = T(4, 2)$ . De manera similar,  $M_{2 \leftarrow 3} = T(2, 3) \cdot S(0.5, 0.5)$  y  $M_{3 \leftarrow 4} = T(6.7, 1.8) \cdot R(-45^\circ)$ . Entonces,  $M_{1 \leftarrow 3} = M_{1 \leftarrow 2} \cdot M_{2 \leftarrow 3} = T(4, 2) \cdot T(2, 3) \cdot S(0.5, 0.5)$ . La figura también muestra un punto que es  $P^{(1)} = (10, 8)$ ,  $P^{(2)} = (6, 6)$ ,  $P^{(3)} = (8, 6)$

y  $P^{(4)} = (4, 2)$  en los sistemas de coordenadas 1 a 4, respectivamente. Es fácil verificar que  $P^{(i)} = M_{i \leftarrow j} \cdot P^{(j)}$  para  $1 \leq i, j \leq 4$ .

También se observa que  $M_{i \leftarrow j} = M_{j \leftarrow i}^{-1}$ . Por lo tanto,  $M_{2 \leftarrow 1} = M_{1 \leftarrow 2}^{-1} = T(-4, -2)$ . Como  $M_{1 \leftarrow 3} = M_{1 \leftarrow 2} \cdot M_{2 \leftarrow 3}$ ,  $M_{1 \leftarrow 3}^{-1} = M_{2 \leftarrow 3}^{-1} \cdot M_{1 \leftarrow 2}^{-1} = M_{3 \leftarrow 2}$ .

En la sección 5.7 se analizaron los sistemas de coordenadas de mano derecha y de mano izquierda. La matriz que convierte los puntos representados en uno de estos sistemas a puntos representados en el otro tiene su propia inversa:

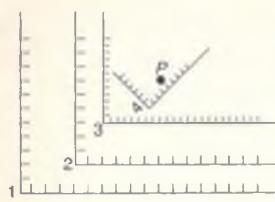


Figura 5.25

El punto  $P$  y los sistemas de coordenadas 1, 2, 3 y 4.

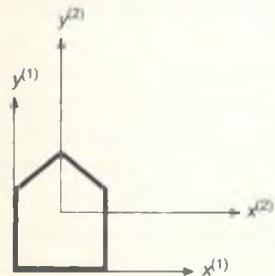


Figura 5.26

La casa y los dos sistemas de coordenadas. Las coordenadas de los puntos en la casa se pueden representar en cualquiera de los sistemas.

$$M_{R \leftarrow L} = M_{L \leftarrow R} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.69)$$

El método que usamos en las secciones anteriores —definir todos los objetos en el sistema de coordenadas de mundo y transformarlos al lugar deseado— implica la noción algo irreal de que todos los objetos se definen uno sobre otro en el mismo sistema de mundo. Es más natural considerar que cada objeto está definido en su propio sistema de coordenadas y que luego se escala, se hace rotar y se traslada por medio de la redefinición de sus coordenadas en un nuevo sistema de mundo. Con este segundo punto de vista, podemos pensar en pedazos de papel, cada uno con un objeto, que se comprimen o estiran, se hacen rotar o se colocan en el plano de coordenadas de mundo. Por supuesto, también podemos imaginar que el plano se comprime o estira, se inclina o se desplaza con respecto a cada trozo de papel. Todas estas perspectivas son matemáticamente idénticas.

Considere el sencillo caso de la traslación al origen del conjunto de puntos que define la casa presentada en la figura 5.10. Esta transformación es  $T(-x_1, -y_1)$ . Si etiquetamos los dos sistemas de coordenadas como se muestra en figura 5.26, podemos ver que la transformación que establece la correspondencia entre el sistema de coordenadas 1 y el 2 (es decir,  $M_{2 \leftarrow 1}$ ) es  $T(x_1, y_1)$ , que es  $T(-x_1, -y_1)^{-1}$ . De hecho, la regla general es que la transformación que se aplica a un conjunto de puntos en un sistema de coordenadas es la inversa de la transformación correspondiente para cambiar el sistema de coordenadas con el cual se representa un punto. Esta relación se puede observar en la figura 5.27, que deriva directamente de la figura 5.11. La transformación para los puntos representados en un sistema de coordenadas es

$$T(x_2, y_2) \cdot R(\theta) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1). \quad (5.32)$$

En la figura 5.27, la transformación de sistema de coordenadas es simplemente

$$M_{5 \leftarrow 1} = M_{5 \leftarrow 4} M_{4 \leftarrow 3} M_{3 \leftarrow 2} M_{2 \leftarrow 1}$$

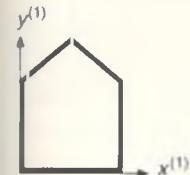
$$= (T(x_2, y_2) \cdot R(\theta) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1))^{-1}$$

$$= T(x_1, y_1) \cdot S(s_x^{-1}, s_y^{-1}) \cdot R(-\theta) \cdot T(-x_2, -y_2), \quad (5.70)$$

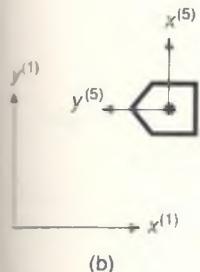
de manera que

$$P^{(5)} = M_{3 \leftarrow 1} P^{(1)} = T(x_1, y_1) \cdot S(s_x^{-1}, s_y^{-1}) \cdot R(-\theta) \cdot T(-x_2, -y_2) \cdot P^{(1)}, \quad (5.71)$$

(a)



(b)

**Figura 5.27**

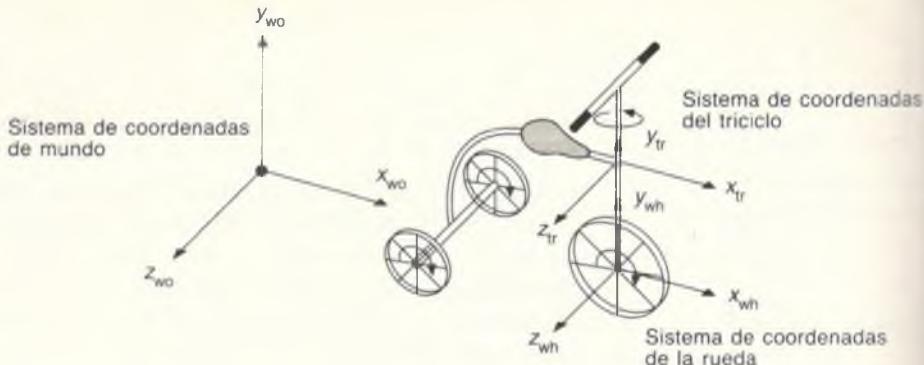
- a) La casa original en su sistema de coordenadas.
- b) La casa transformada en su sistema de coordenadas con respecto al sistema de coordenadas original.

Una pregunta importante relacionada con el cambio de sistemas de coordenadas es cómo cambiar las transformaciones. Suponga que  $Q^{(j)}$  es una transformación en el sistema de coordenadas  $j$ . Por ejemplo, podría ser una de las transformaciones compuestas que obtuvimos en las secciones anteriores. Suponga que queremos hallar la transformación  $Q^{(i)}$  en el sistema de coordenadas  $i$  que pudiera aplicarse a los puntos  $P^{(i)}$  en el sistema  $i$  para producir exactamente los mismos resultados que se obtendrían al aplicar  $Q^{(j)}$  a los puntos correspondientes  $P^{(j)}$  en el sistema  $j$ . Esta igualdad se representa con  $Q^{(i)} \cdot P^{(i)} = M_{i \leftarrow j} \cdot Q^{(j)} \cdot P^{(j)}$ . Al sustituir  $P^{(j)} = M_{j \leftarrow i} \cdot P^{(i)}$ , esta expresión se convierte en  $Q^{(i)} \cdot M_{i \leftarrow j} \cdot P^{(i)} = M_{i \leftarrow j} \cdot Q^{(j)} \cdot P^{(j)}$ . Simplificando, se obtiene  $Q^{(i)} = M_{i \leftarrow j} \cdot Q^{(j)} \cdot M_{j \leftarrow i}^{-1}$ .

El punto de vista del cambio del sistema de coordenadas es útil cuando se especifica información adicional de subobjetos con el sistema local de coordenadas del subobjeto. Por ejemplo, si la rueda frontal del triciclo de la figura 5.28 se hace rotar sobre su coordenada  $z_{wh}$ , todas las ruedas deben girar en forma apropiada y necesitamos saber cómo se mueve el triciclo en su conjunto en el sistema de coordenadas de mundo. Este problema es complejo, ya que ocurren varios cambios sucesivos de sistemas de coordenadas. Primero, los sistemas de coordenadas del triciclo y la rueda frontal tienen posiciones iniciales en el sistema de mundo. Conforme avanza el triciclo, la rueda frontal gira sobre el eje  $z$  del sistema de coordenadas de la rueda, y al mismo tiempo los sistemas de coordenadas de la rueda y del triciclo se mueven con respecto al sistema de coordenadas de mundo. Los sistemas de coordenadas de la rueda y del triciclo se relacionan con el sistema de coordenadas de mundo por medio de traslaciones variables en el tiempo en  $x$  y  $z$ , y por una rotación en  $y$ . Los sistemas de coordenadas de la rueda y del triciclo se relacionan entre sí a través de una rotación variable en el tiempo con respecto a  $y$  al girar el manubrio. (El sistema de coordenadas del triciclo está fijado al cuadro, no al manubrio).

Para que este problema sea un poco más sencillo, supondremos que los ejes de la rueda y del triciclo son paralelos a los ejes de las coordenadas de mundo y que la rueda se mueve sobre una línea recta paralela al eje  $x$  de las coordenadas de mundo. Conforme la rueda rota un ángulo  $\alpha$ , un punto  $P$  en la rueda rota una distancia  $\alpha r$ , donde  $r$  es el radio de la rueda. Como la rueda está sobre el suelo, el triciclo avanza  $\alpha r$  unidades. Por lo tanto, el punto  $P$  en el exterior de la rueda se mueve y gira con respecto al sistema inicial de coordenadas de la rueda con un efecto neto de traslación de  $\alpha r$  y una rotación de  $\alpha$ . Sus nuevas coordenadas  $P'$  en el sistema original de coordenadas de la rueda son entonces

$$P'^{(wh)} = T(\alpha r, 0, 0) \cdot R_z(\alpha) \cdot P^{(wh)}, \quad (5.72)$$



**Figura 5.28** Triciclo estilizado con tres sistemas de coordenadas.

y sus coordenadas en el nuevo (trasladado) sistema de coordenadas de la rueda se indican con la rotación

$$P^{(wh)} = R_z(\alpha) \cdot P^{(wh)}. \quad (5.73)$$

Para hallar los puntos  $P^{(wo)}$  y  $P'^{(wo)}$  en el sistema de coordenadas de mundo, hacemos una transformación del sistema de coordenadas de la rueda al de coordenadas de mundo:

$$P^{(wo)} = M_{wo \leftarrow wh} \cdot P^{(wh)} = M_{wo \leftarrow tr} \cdot M_{tr \leftarrow wh} \cdot P^{(wh)}. \quad (5.74)$$

$M_{wo \leftarrow wh}$  y  $M_{tr \leftarrow wh}$  son traslaciones indicadas por las posiciones iniciales del triciclo y la rueda.

$P'^{(wo)}$  se calcula con las ecuaciones (5.72) y (5.74):

$$P'^{(wo)} = M_{wo \leftarrow wh} \cdot P'^{(wh)} = M_{wo \leftarrow wh} \cdot T(\alpha r, 0, 0) \cdot R_z(\alpha) \cdot P^{(wh)}. \quad (5.75)$$

Alternativamente, se observa que  $M_{wo \leftarrow wh}$  se cambió a  $M_{wo \leftarrow wh'}$  con la traslación del sistema de coordenadas de mundo. Obtenemos el mismo resultado que con la ecuación (5.75), aunque de manera diferente:

$$P'^{(wo)} = M_{wo \leftarrow wh'} \cdot P'^{(wh')} = (M_{wo \leftarrow wh} \cdot M_{wh \leftarrow wh'}) \cdot (R_z(\alpha) \cdot P^{(wh)}). \quad (5.76)$$

Por lo tanto, en términos generales las nuevas  $M_{wo \leftarrow wh'}$  y  $M_{tr \leftarrow wh'}$  se obtienen de sus valores anteriores aplicando las transformaciones apropiadas de las ecuaciones de movimiento de las partes del triciclo. Después se aplican estas transformaciones actualizadas a los puntos actualizados en el sistema de coordenadas locales y se derivan los puntos equivalentes en los sistemas de coordenadas de mundo.

**Ejercicios**

5.1 Demuestre que podemos transformar una línea por medio de la transformación de sus puntos extremos y construyendo una nueva línea entre los puntos extremos transformados.

5.2 Demuestre que dos rotaciones bidimensionales sucesivas son aditivas:  $R(\theta_1) \cdot R(\theta_2) = R(\theta_1 + \theta_2)$ .

5.3 Demuestre que la rotación y el escalamiento bidimensional son comutativos si  $s_x = s_y$  o si  $\theta = n\pi$  para la integral  $n$ , y que no lo son en caso contrario.

5.4 Aplique las transformaciones desarrolladas en la sección 5.8 a los puntos  $P_1$ ,  $P_2$  y  $P_3$ , para verificar que estos puntos se transformen de la manera deseada.

5.5 Repase la sección 5.8 suponiendo que  $|P_1P_2| = 1$ ,  $|P_1P_3| = 1$  y que se especifican los cosenos de dirección de  $P_1P_2$  y  $P_1P_3$  (los *cosenos de dirección* de una línea son los cosenos de los ángulos entre la línea y los ejes  $x$ ,  $y$  y  $z$ ). Para una línea del origen a  $(x, y, z)$ , los cosenos de dirección son  $(x/d, y/d, z/d)$ , donde  $d$  es la longitud de la línea.

5.6 Demuestre que las ecuaciones (5.59) y (5.64) son equivalentes.

5.7 Dado un cubo unidad con un vértice en  $(0, 0, 0)$  y el vértice opuesto en  $(1, 1, 1)$ , obtenga las transformaciones necesarias para que el cubo rote  $\theta$  grados con respecto a la diagonal principal [de  $(0, 0, 0)$  a  $(1, 1, 1)$ ] en sentido contrario al giro de las manecillas del reloj cuando se observa por la diagonal hacia el origen.

5.8 Suponga que la base de la ventana se hace rotar un ángulo  $\theta$  a partir del eje  $x$ , como en el sistema Core [GSPC79]. ¿Cuál es la correspondencia ventana-área de vista? Verifique su respuesta aplicando la transformación a cada vértice de la ventana, para constatar que se transformen en los vértices apropiados en el área de vista.

5.9 Considere una línea del origen de un sistema de coordenadas de mano derecha al punto  $P(x, y, z)$ . Encuentre las matrices de transformación necesarias para que la línea rote en el eje  $z$  positivo de dos maneras, y demuestre con manipulación algebraica que, en cada caso, la  $P$  llega al eje  $z$ . Para cada método calcule los senos y cosenos de los ángulos de rotación.

- Rotación con respecto al eje  $y$  hacia el plano  $(y, z)$ , y rotación sobre el eje  $x$  hasta el eje  $z$ .
- Rotación con respecto al eje  $z$  hacia el plano  $(x, z)$ , y rotación sobre el eje  $y$  hasta el eje  $z$ .

5.10 Se escalará un objeto con un factor  $S$  en la dirección con cosenos  $(\alpha, \beta, \gamma)$ . Obtenga la matriz de transformación.

5.11 Encuentre la matriz de transformación de  $4 \times 4$  que rote un ángulo  $\theta$  sobre una dirección arbitraria indicada por el vector de dirección  $U = (u_x, u_y, u_z)$ . Haga este ejercicio componiendo la matriz de transformación que rote  $U$  en el eje  $z$  (llame a esto  $M$ ) con una rotación de  $R_z(\theta)$ , y combine este resultado con  $M^{-1}$ . El resultado debe ser

$$\begin{bmatrix} u_x^2 + \cos \theta (1 - u_x^2) & u_x u_y (1 - \cos \theta) - u_z \sin \theta & u_z u_x (1 - \cos \theta) + u_y \sin \theta & 0 \\ u_x u_y (1 - \cos \theta) + u_z \sin \theta & u_y^2 + \cos \theta (1 - u_y^2) & u_y u_z (1 - \cos \theta) - u_x \sin \theta & 0 \\ u_z u_x (1 - \cos \theta) - u_y \sin \theta & u_y u_z (1 - \cos \theta) + u_x \sin \theta & u_z^2 + \cos \theta (1 - u_z^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.77)$$

Verifique que si  $U$  es un eje principal, la matriz se reduce a  $R_x$ ,  $R_y$  o  $R_z$ . Vea [FAUX79] para una derivación basada en operaciones vectoriales. Observe que la negación de  $U$  y  $\theta$  no altera el resultado. Explique por qué es verdadero este resultado.

El proceso de vista tridimensional es más complejo que el de vista bidimensional. En el caso bidimensional, basta especificar una ventana en el mundo bidimensional y un área de vista en la superficie de la vista bidimensional. En teoría, los objetos en el mundo se recortan con respecto a la ventana y luego se transforman al área de vista para la presentación. La complejidad adicional de la vista tridimensional es ocasionada en parte por la otra dimensión y en parte por el hecho de que los dispositivos de presentación son bidimensionales. Aunque a primera vista la vista tridimensional puede parecer abrumadora, deja de serlo cuando se le considera como una serie de pasos fáciles de comprender, para muchos de los cuales nos hemos preparado en capítulos anteriores. Por lo tanto, comenzaremos con un resumen del proceso de vista tridimensional que lo guiará por este capítulo.

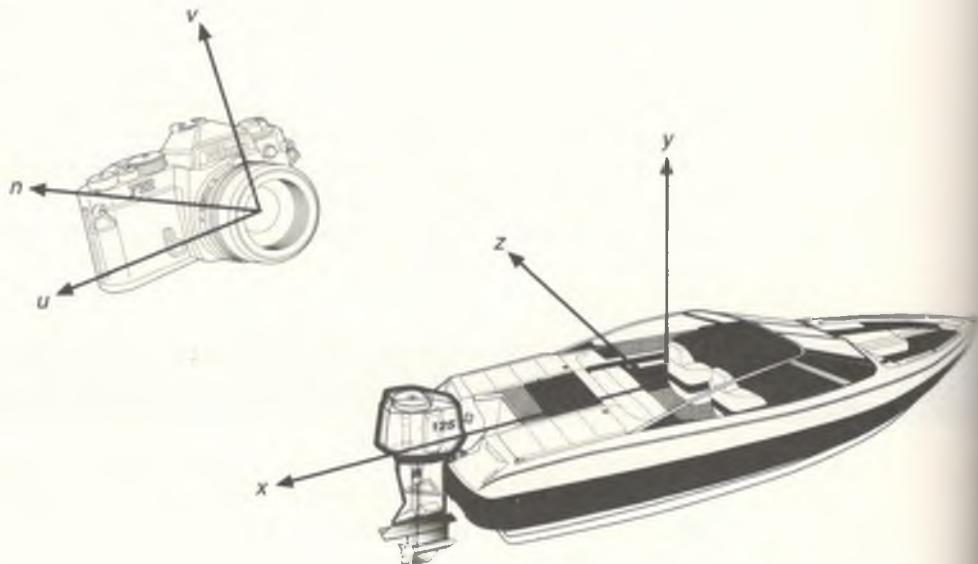
## 6.1 La cámara sintética y los pasos en la vista tridimensional

Una metáfora útil para la creación de escenas tridimensionales es el concepto de la **cámara sintética**, que se ilustra en la figura 6.1. Imagine que podemos mover la cámara a cualquier posición, orientarla como queramos y, con un disparo del obturador, crear una imagen bidimensional de un objeto tridimensional, la lancha en este caso. Si lo deseamos, la cámara se puede convertir en una cámara cinematográfica, lo que nos permite crear una secuencia animada que muestre el objeto en diversas orientaciones y tamaños. Por supuesto, la cámara no es más

que un programa de computación que produce una imagen en la pantalla y el objeto es una base de datos tridimensionales que comprende una colección de puntos, líneas y superficies. En la figura 6.1 también se muestra que la cámara y el objeto tridimensional tienen su propio sistema de coordenadas:  $u$ ,  $v$ ,  $n$  para la cámara y  $x$ ,  $y$ ,  $z$  para el objeto. Más adelante en este capítulo analizaremos la importancia de estos sistemas de coordenadas. Por el momento sólo mencionaremos que ofrecen una importante independencia de la representación.

Aunque la cámara sintética es un concepto útil, se requiere más que presionar un botón para producir una imagen. La creación de nuestra "fotografía" lleva a cabo en varias etapas que describimos a continuación.

- *Especificación del tipo de proyección.* La diferencia entre los objetos tridimensionales y las pantallas bidimensionales se resuelve con la introducción de proyecciones, que transforman objetos tridimensionales a un plano de proyección bidimensional. Gran parte de este capítulo se dedica a las proyecciones: qué son, cuáles son sus fundamentos matemáticos y cómo se usan en un paquete de subrutinas gráficas actual, PHIGS [ANSI88]. Nos centraremos en dos de las proyecciones más importantes: la de perspectiva y la ortográfica paralela. En el capítulo 7 se analiza con mayor detalle la utilización de proyecciones.
- *Especificación de parámetros de visualización.* Una vez que se ha determinado el tipo de proyección que se desea, es necesario especificar las condiciones

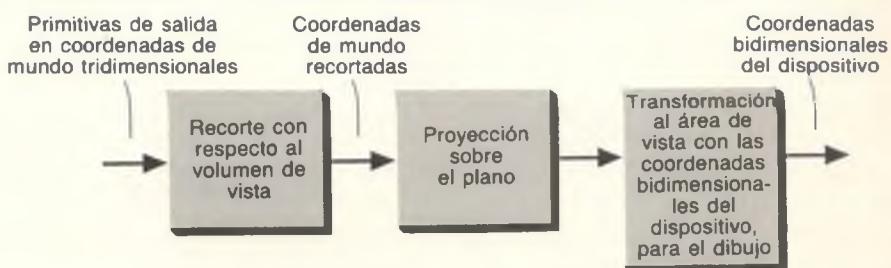


**Figura 6.1** Cámara sintética que fotografía un objeto tridimensional.

en las cuales queremos ver el conjunto de datos tridimensionales del mundo real o la escena que se generará. Con base en las coordenadas de mundo del conjunto de datos, esta información incluye la posición del ojo del observador y la ubicación del plano de observación, o sea, la superficie donde se presentará la proyección. Usaremos dos sistemas de coordenadas, el de la pantalla y otro que llamaremos **sistema de coordenadas del ojo o de vista**. Variando alguno de estos parámetros podemos obtener cualquier representación de la escena, incluyendo la observación de su interior cuando esto tenga sentido.

- **Recorte en tres dimensiones.** Así como debemos confinar la presentación de una escena bidimensional a las fronteras de la ventana especificada, también hay que eliminar de una escena tridimensional las partes que no sean candidatos para la presentación final. De hecho, podemos ignorar las partes de la escena que están detrás de nosotros o que se encuentran demasiado lejos para ser claramente visibles. Para esto es necesario recortar con respecto a un volumen de vista, un proceso mucho más complejo que los representados por los algoritmos estudiados hasta ahora. Debido a la gran variedad de los volúmenes de vista posibles, dedicaremos parte de nuestro estudio a la definición de un **volumen de vista** canónico, con el cual podamos aplicar de manera eficaz un algoritmo de recorte estandarizado.
- **Proyección y presentación.** Por último, el contenido de la proyección del volumen de vista sobre el plano de proyección, llamado **ventana**, se transforma (se establece la correspondencia) al área de vista para la presentación.

En la figura 6.2 se ilustran los pasos principales de este modelo conceptual del proceso de visualización tridimensional; este modelo es el que se presenta a los usuarios de muchos paquetes de subrutinas gráficas tridimensionales. Como sucede en la vista bidimensional, se pueden usar diversas estrategias para implantar el proceso de vista. Las estrategias no tienen que ser idénticas al modelo conceptual, siempre y cuando los resultados sean los definidos por el modelo. En la sección 6.6 se describe una estrategia de implantación típica para los dibujos de alambre. Para los sistemas gráficos que llevan a cabo la determinación de superficies visibles y sombreado, se emplea un ducto distinto, analizado en el capítulo 14.



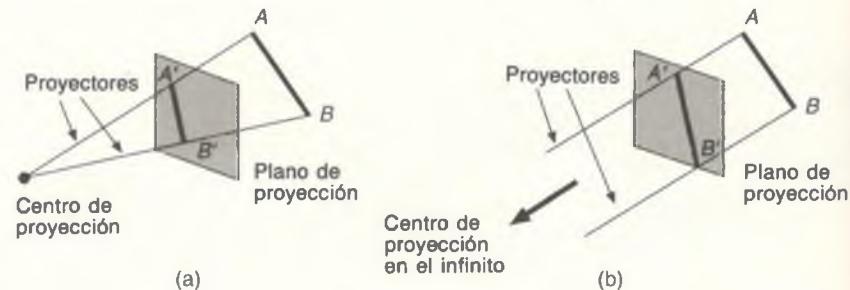
**Figura 6.2** Modelo conceptual del proceso de vista tridimensional.

## 6.2 Proyecciones

En términos generales, las proyecciones transforman puntos en un sistema de coordenadas de dimensión  $n$  a puntos en un sistema de coordenadas con dimensión menor que  $n$ . De hecho, durante mucho tiempo se ha usado la graficación por computador para estudiar objetos  $n$ -dimensionales por medio de su proyección sobre dos dimensiones [NOLL67]. Aquí nos limitaremos a la proyección de tres dimensiones a dos. La proyección de objetos tridimensionales es definida por rayos de proyección rectos, llamados **proyectores**, que emanan de un **centro de proyección**, pasan por cada punto del objeto e intersecan un **plano de proyección** para formar la proyección. Por lo general, el centro de proyección se encuentra a una distancia finita del plano de proyección. Sin embargo, en algunos tipos de proyecciones es conveniente pensar en función de un centro de proyección que tienda a estar infinitamente lejos; este concepto se examinará con mayor detalle en la sección 6.2.1. En la figura 6.3 se presentan dos proyecciones diferentes de la misma línea. Afortunadamente, la proyección de una línea es ~~en~~ sí una línea, de manera que sólo hay que proyectar los puntos extremos.

La clase de proyecciones que trataremos aquí se conoce como **proyecciones geométricas planas**, ya que la proyección es sobre un plano y no sobre una superficie curva y porque usa proyectores rectos y no curvos. Varias proyecciones cartográficas son no planas o no geométricas.

Las proyecciones geométricas planas, que a partir de ahora llamaremos simplemente **proyecciones**, se pueden dividir en dos clases básicas: **de perspectiva** y **paralelas**. La diferencia se debe a la relación entre el centro de proyección y el plano de proyección. Si la distancia entre uno y otro es finita, la proyección es de perspectiva; conforme se aleja el centro de proyección, los proyectores que pasan por un objeto tienden cada vez más a ser paralelos. En la figura 6.3



**Figura 6.3** Dos proyecciones diferentes de la misma línea. (a) Línea  $AB$  y su proyección de perspectiva  $A'B'$ . (b) Línea  $AB$  y su proyección paralela  $A'B'$ . Los proyectores  $AA'$  y  $BB'$  son paralelos.

ilustran los dos casos. La proyección paralela recibe ese nombre porque los proyectores son paralelos si el centro de proyección se encuentra a una distancia infinita. Al definir una proyección de perspectiva se especifica explícitamente su **centro de proyección**; en el caso de una proyección paralela, se indica su **dirección de proyección**. El centro de proyección es un punto, por lo cual tiene coordenadas homogéneas de la forma  $(x, y, z, 1)$ . Como la dirección de proyección es un vector (es decir, la diferencia entre dos puntos), lo podemos calcular restando los dos puntos  $d = (x, y, z, 1) - (x', y', z', 1) = (a, b, c, 0)$ . Por lo tanto, las **direcciones y los puntos en el infinito** tienen una correspondencia natural. En el límite, una proyección de perspectiva cuyo centro de proyección tienda a *un punto en el infinito se convierte en una proyección paralela*.

El efecto visual de una proyección de perspectiva es similar al de los sistemas fotográficos y al del sistema visual humano y se conoce como **reducción frontal de perspectiva**. El tamaño de la proyección de perspectiva de un objeto varía inversamente con la distancia entre el objeto y el centro de proyección. Por lo tanto, aunque la proyección de perspectiva de los objetos tiende a parecer realista, no es muy útil para registrar la forma y las medidas exactas de los objetos; no se pueden tomar las distancias de la proyección, los ángulos sólo se conservan en las caras de los objetos paralelos al plano de proyección y por lo general las líneas paralelas no se proyectan como tales.

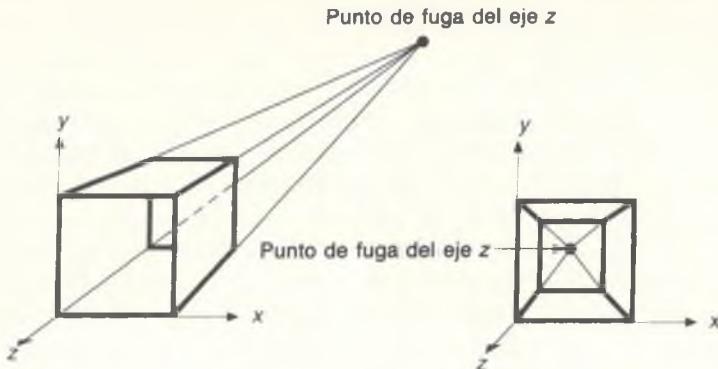
La proyección paralela es una vista menos realista porque no existe la reducción frontal de perspectiva, aunque pueden existir diversas reducciones frontales constantes sobre cada eje. La proyección se puede usar para mediciones exactas, y las líneas paralelas permanecen como tales. Como sucede en la proyección de perspectiva, los ángulos únicamente se conservan en las caras de los objetos paralelos al plano de proyección.

Los distintos tipos de proyecciones paralelas y de perspectiva se analizan e ilustran con detalle en el exhaustivo artículo escrito por Carlom y Paciorek [CARL78]. En las secciones 6.2.1 y 6.2.2 resumimos las definiciones y características básicas de las proyecciones de uso más común; después, en la sección 6.3, pasamos a la forma como se especifican las proyecciones en PHIGS.

## 6.2.1 Proyecciones de perspectiva

Las proyecciones de perspectiva de cualquier conjunto de líneas paralelas que no sean paralelas al plano de conversión convergen en **un punto de fuga**. En el espacio tridimensional, las líneas paralelas sólo se unen en el infinito, de manera que el punto de fuga se puede considerar como la proyección de un punto en el infinito. Por supuesto, hay una infinidad de puntos de fuga, uno para cada una de la infinidad de direcciones en que puede orientarse una línea.

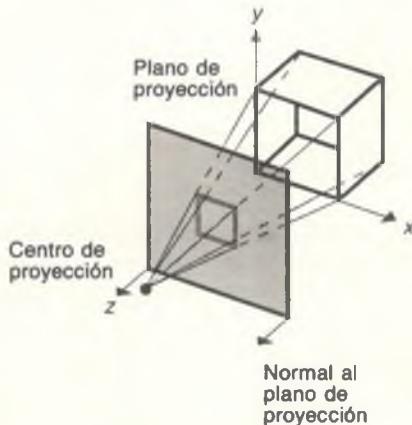
Si el conjunto de líneas es paralelo a uno de los tres ejes principales, el punto de fuga se conoce como **punto de fuga de eje**. A lo sumo hay tres de estos puntos, correspondientes al número de ejes principales cortados por el plano de proyección. Por ejemplo, si el plano de proyección sólo corta el eje  $z$  (y por consiguiente es normal a él), sólo el eje  $z$  tendrá un punto de fuga principal, ya



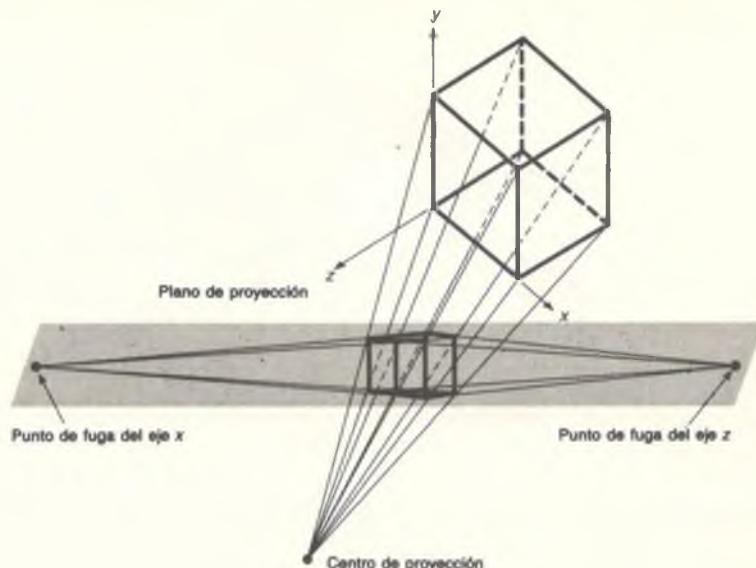
**Figura 6.4** Proyecciones de perspectiva de un punto de un cubo sobre un plano que corta el eje z, mostrando el punto de fuga de las líneas perpendiculares al plano de proyección.

que las líneas paralelas a los ejes  $y$  o  $x$  también serán paralelas al plano de proyección y no tendrán puntos de fuga.

Las proyecciones de perspectiva se clasifican de acuerdo con el número de puntos de fuga principales y por ende con respecto al número de ejes que corta el plano de proyección. En la figura 6.4 se muestran dos proyecciones de perspectiva de un punto para un cubo. Es obvio que hay más proyecciones de punto, ya que las líneas paralelas a los ejes  $x$  y  $y$  no convergen; esto sólo ocurre con las líneas paralelas al eje  $z$ . En la figura 6.5 se presenta la construcción de



**Figura 6.5** Construcción de una proyección de perspectiva de un punto para un cubo sobre un plano que corta el eje z. La normal al plano de proyección es paralela al eje z. (Adaptado de [CARL78], Association for Computing Machinery, Inc.; utilizado con autorización.)



**Figura 6.6** Proyección de perspectiva de dos puntos de un cubo. El plano de proyección corta los ejes  $x$  y  $z$ .

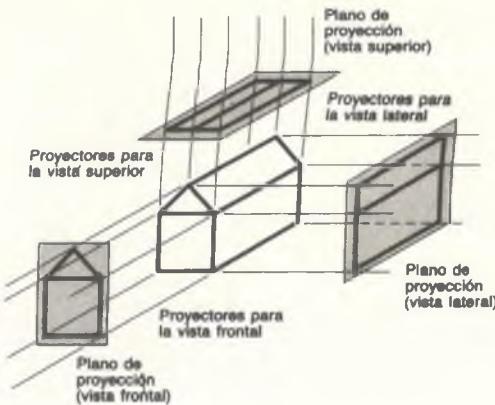
una perspectiva de un punto con algunos de los proyectores y con un plano de proyección que sólo corta el eje  $z$ .

En la figura 6.6 se muestra la construcción de una perspectiva de dos puntos. Observe que las líneas paralelas al eje  $y$  no convergen en la proyección. La perspectiva de dos puntos se usa comúnmente en dibujos de arquitectura, ingeniería, diseño industrial y publicidad. Las perspectivas de tres puntos son menos frecuentes, ya que añaden muy poco al realismo que se puede obtener con la perspectiva de dos puntos.

### 6.2.2 Proyecciones paralelas

Las proyecciones paralelas se clasifican en dos tipos, dependiendo de la relación entre la dirección de la proyección y la normal al plano de proyección. En las proyecciones paralelas **ortográficas**, estas direcciones son las *mismas* (o en sentido contrario), de manera que la dirección de la proyección es normal al plano de proyección. Esto no ocurre en la proyección **paralela oblicua**.

Los tipos más comunes de proyecciones ortográficas son la de **elevación frontal**, **elevación superior** o **elevación de plano** y la de **elevación lateral**. En todas ellas, el plano de proyección es perpendicular al eje principal, que por lo tanto es la dirección de la proyección. En la figura 6.7 se presenta la construcción de estas tres proyecciones, que se usan comúnmente en dibujos de ingeniería para representar piezas de maquinaria, montajes y edificios, ya que



**Figura 6.7** Construcción de tres proyecciones ortográficas.

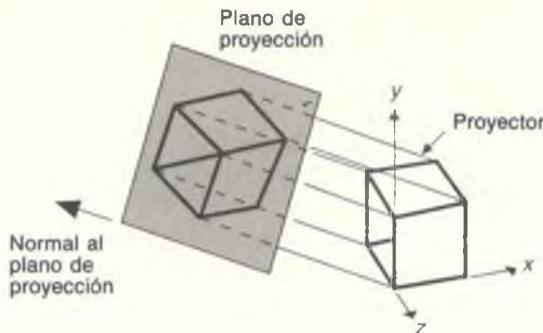
las distancias y los ángulos se pueden medir a partir de las representaciones. Sin embargo, cada proyección sólo muestra una cara del objeto, de manera que puede ser difícil deducir la naturaleza tridimensional del objeto proyectado, incluso si se estudian simultáneamente varias proyecciones del mismo objeto.

Las proyecciones ortográficas axonométricas usan planos de proyección que no son normales a un eje principal y que por ende muestran varias caras de un objeto al mismo tiempo. En este aspecto se parecen a la proyección de perspectiva, pero difieren en lo referente al recorte frontal, que es uniforme y no relacionado con la distancia al centro de proyección. Se conserva el paralelismo de las líneas, pero no los ángulos, y las distancias se pueden medir sobre cualquiera de los ejes principales (por lo general, con factores de escalamiento diferentes).

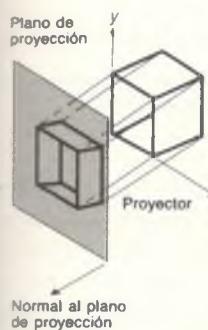
La proyección isométrica es una proyección axonométrica de uso común. La normal al plano de proyección (y por consiguiente la dirección de la proyección) forma ángulos iguales con respecto a cada eje principal. Si la normal al plano de proyección es  $(d_x, d_y, d_z)$ , requerimos que  $|d_x| = |d_y| = |d_z|$  o  $\pm d_x = \pm d_y = \pm d_z$ . Sólo hay ocho direcciones (una en cada octante) que satisfacen esta condición. En la figura 6.8 se muestra la construcción de una proyección isométrica a lo largo de una de estas direcciones,  $(1, -1, -1)$ .

La proyección isométrica tiene una útil propiedad: los tres ejes principales tienen la misma reducción frontal, lo que permite que las mediciones sobre los ejes se realicen con la misma escala (de aquí proviene el nombre: *iso*, que quiere decir “igual”, y *métrico*, de medición). Además, las proyecciones de los ejes principales forman ángulos iguales de  $120^\circ$  entre sí.

Las proyecciones oblicuas, la segunda clase de proyecciones paralelas, difieren de las proyecciones ortográficas en que la normal al plano de proyección y la dirección de la proyección son diferentes. Las proyecciones oblicuas combinan las propiedades de las proyecciones ortográficas frontal, superior y lateral.



**Figura 6.8** Construcción de una proyección isométrica de un cubo unidad. (Adaptado de [CARL78], Association for Computing Machinery, Inc; utilizado con autorización.)



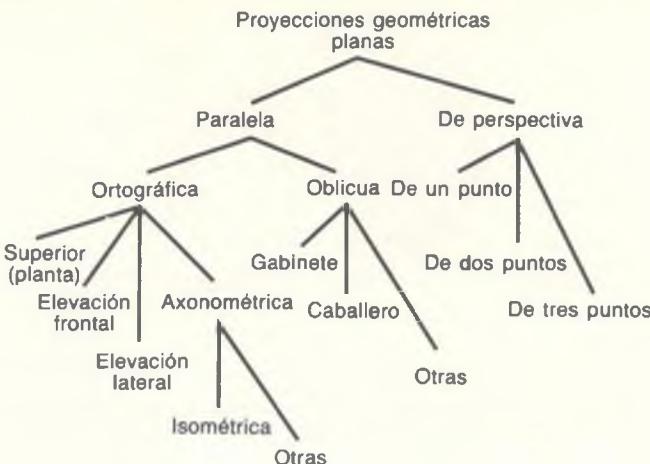
**Figura 6.9** Construcción de una proyección oblicua. (Adaptado de [CARL78], Association for Computing Machinery, Inc; utilizado con autorización.)

con las de una proyección axonométrica: el plano de proyección es normal a un eje principal, de manera que la proyección de la cara del objeto paralela a este plano permite medir ángulos y distancias. También se proyectan otras caras del objeto, lo que permite medir las distancias sobre los ejes principales, aunque no los ángulos. En este texto se emplean mucho las proyecciones oblicuas porque son fáciles de dibujar. En la figura 6.9 se muestra la construcción de una proyección oblicua. Observe que la normal al plano de proyección y la dirección de la proyección no son iguales. En [FOLE90] se describen varios tipos de proyecciones oblicuas.

En la figura 6.10 se muestran las relaciones lógicas entre los diversos tipos de proyecciones. El lazo común de todas las proyecciones es que comprenden un plano de proyección y un centro de proyección para las proyecciones de perspectiva o una dirección de proyección para las proyecciones paralelas. Podemos unificar los casos paralelos y de perspectiva si consideramos que el centro de proyección está definido por la dirección al centro de proyección desde un punto de referencia y la distancia a este punto. Cuando esta distancia aumenta hasta el infinito, la proyección se convierte en paralela. Por lo tanto, podemos decir que el aspecto común de estas proyecciones es que comprenden un plano de proyección, una dirección al centro de proyección y una distancia al centro de proyección. En la sección 6.3 veremos cómo integrar algunos de estos tipos de proyecciones al proceso de visualización tridimensional.

## 6.3 Especificación de una vista tridimensional arbitraria

Como se sugiere en la figura 6.2, la vista tridimensional no sólo comprende una proyección, sino también un volumen de vista con respecto al cual se recorta el mundo tridimensional. El conjunto de la proyección y el volumen de vista



**Figura 6.10** Subclases de las proyecciones geométricas planas. La **vista de planta** es otro término para la vista superior. Las vistas **frontal** y **lateral** muchas veces se emplean sin el término **elevación**.

proporciona toda la información que se necesita para recortar y proyectar en un espacio bidimensional. Así, la transformación bidimensional a coordenadas de un dispositivo físico es bastante sencilla. Ahora conoceremos más detalles de los conceptos de la proyección geométrica plana que presentamos en la sección 6.2 y veremos cómo especificar un volumen de vista. La terminología y el método de visualización que presentamos aquí son los que se emplean en PHIGS.

El plano de proyección, que a partir de ahora llamaremos **plano de vista** para ser consistentes con la literatura sobre graficación, es definido por un punto en el plano denominado **punto de referencia de vista** (**VRP**, *view reference point*) y una normal al plano llamada **normal al plano de vista** (**VPN**, *view-plane normal*). El plano de vista puede estar en cualquier lugar con respecto a los objetos mundiales que se proyectarán: puede encontrarse delante de los objetos, atravesarlos o estar detrás de ellos.

Una vez que se tiene el plano de vista, se requiere una ventana en el plano. La función de la ventana es similar a la de una ventana bidimensional: se establece una correspondencia entre su contenido y el área de vista, y no se presentan las partes del mundo tridimensional que se proyecten al plano de vista fuera de la ventana. Veremos que la ventana también tiene una función importante en la definición del volumen de vista.

Para definir una ventana en el plano de vista necesitamos una manera de especificar coordenadas mínimas y máximas para la ventana y los dos ejes ortogonales en el plano de vista para medir estas coordenadas. Estos ejes forman parte de un sistema de **coordenadas de referencia de vista** (**VRC**, *viewing-reference coordinates*). El origen de un sistema VRC es el VRP. Un eje del VRC es el **VPN** y se denomina **eje *n***. Otro eje de VRC se encuentra con el **vector de vista**

**arriba (VUP, view-up vector)**, que determina la dirección del eje  $v$  en el plano de vista. El eje  $v$  se define de manera que la proyección de VUP paralela a VPN en el plano de vista coincida con el eje  $v$ . La dirección del eje  $u$  se define de manera tal que  $u$ ,  $v$  y  $n$  formen un sistema de coordenadas de mano derecha, como en la figura 6.11. El VRP y los dos vectores de dirección, VPN y VUP, se especifican en el sistema de coordenadas mundiales de mano derecha. (Algunos paquetes gráficos utilizan el eje  $y$  como VUP, pero esta convención es demasiado restrictiva y no funciona si VPN es paralelo al eje  $y$ , ya que en este caso VUP no está definido.)

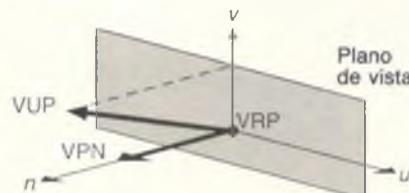
Una vez definido el sistema VRC, es posible determinar las coordenadas  $u$  y  $v$  máximas y mínimas de la ventana, como en la figura 6.12. En esta figura se ilustra que la ventana no tiene que ser simétrica con respecto al VRP, y se muestra explícitamente el centro de la ventana, CW.

El centro de proyección y la dirección de la proyección (DOP) se definen con un **punto de referencia de proyección (PRP)** y un indicador del tipo de proyección. Si el tipo de proyección es de perspectiva, entonces PRP es el centro de proyección. Si la proyección es paralela, la dirección de proyección es de PRP a CW. El centro de la ventana generalmente es distinto del VRP, el cual ni siquiera tiene que estar dentro de los límites de la ventana.

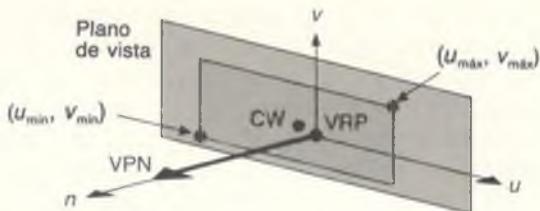
El PRP se especifica en el sistema VRC, no en el sistema de coordenadas de mundo; es decir, la posición del PRP con respecto al VRP no cambia al mover VUP o VRP. La ventaja de este esquema es que el programador puede especificar la dirección de proyección que se requiere y luego cambiar VPN y VUP (cambiando por lo tanto VRC) sin tener que recalcular el PRP necesario para mantener la proyección deseada. Por otra parte, puede ser más difícil mover el PRP para obtener diferentes vistas de un objeto.

El **volumen de vista** limita la porción de un mundo que se recortará y proyectará en el plano de vista. Para una proyección de perspectiva, el volumen de vista es la pirámide semiinfinita con ápice en el PRP y aristas que pasan por los vértices de la ventana. En la figura 6.13 se presenta un volumen de vista de proyección de perspectiva.

Las posiciones detrás del centro de proyección no se incluyen en el volumen de vista y por ende no se proyectan. Por supuesto, en la realidad nuestros ojos



**Figura 6.11** El plano de vista está definido por VPN y VRP; el eje  $v$  se define con la proyección de VUP a lo largo de VPN sobre el plano de vista. El eje  $u$  forma el sistema VRC de mano derecha con VPN y  $v$ .

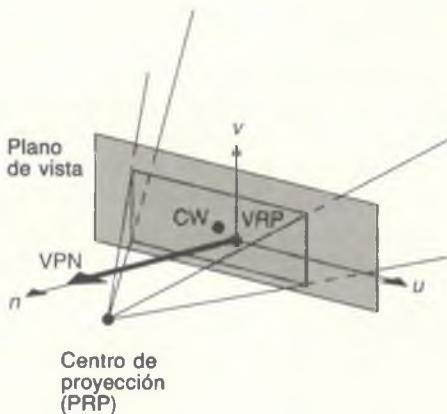


**Figura 6.12** El sistema de coordenadas de referencia de visualización (VRC) es un sistema de mano derecha formado por los ejes  $u$ ,  $v$  y  $n$ . El eje  $n$  siempre es VPN. CW es el centro de la ventana.

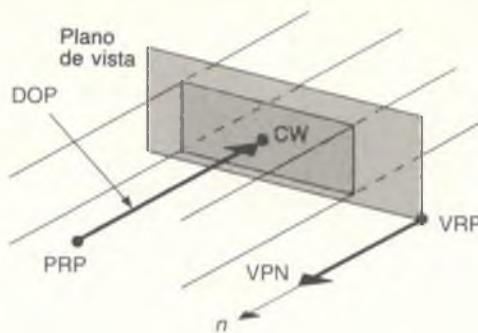
observan un volumen de vista de forma irregular similar a la de un cono. Sin embargo, un volumen de vista piramidal es matemáticamente más sencillo y acorde con el concepto de un área de vista rectangular.

En el caso de las proyecciones paralelas, el volumen de vista es un paralelepípedo infinito con lados paralelos a la dirección de proyección, o sea, a la dirección del PRP al centro de la ventana. En la figura 6.14 se muestra un volumen de vista de proyección paralela y su relación con el plano de vista, la ventana y el PRP.

En ocasiones queremos que el volumen de vista sea finito, para limitar el número de primitivas de salida que se proyectan sobre el plano de vista. En las figuras 6.15 y 6.16 se muestra cómo se hace finito el volumen de vista con un **plano de recorte anterior** y un **plano de recorte posterior**, que a veces se conocen como cercano y lejano (*hither* y *yon*). Estos planos son paralelos al plano de vista; su normal es VPN. Los planos se especifican usando cantidades con signo de **distancia al plano anterior** (*F*) y **distancia al plano posterior** (*B*, relativas al



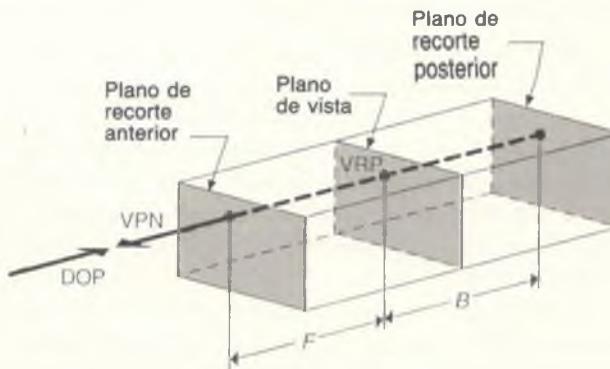
**Figura 6.13** Volumen de vista de pirámide semiinfinita para la proyección de perspectiva. CW es el centro de la ventana.

**Figura 6.14**

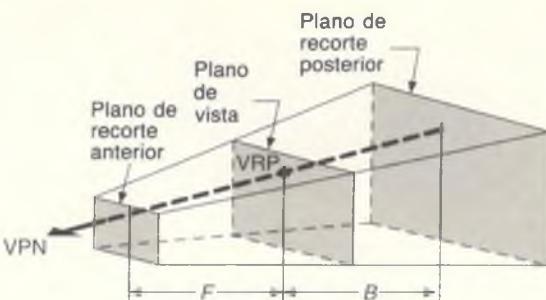
Volumen de vista de paralelepípedo infinito para una proyección paralela ortográfica. VPN y la dirección de proyección (DOP) son paralelas. DOP es el vector de PRP a CW y es paralelo a VPN.

VRP y sobre la VPN, con distancias positivas en la dirección de VPN. Para que el volumen de vista no esté vacío, la distancia al plano anterior debe ser algebraicamente mayor que la distancia al plano posterior.

Esta limitación del volumen de vista puede ser útil para eliminar objetos extraños y permitir que el usuario se centre en una porción particular del mundo. La modificación dinámica de las distancias al plano anterior o posterior puede dar al observador un buen sentido de las relaciones espaciales entre distintas partes del objeto, conforme estas partes aparecen y desaparecen de su vista (véase el Cap. 12). Para las proyecciones de perspectiva hay una motivación adicional. Un objeto muy distante del centro de proyección se proyecta en la superficie de vista como una “mancha” sin forma distinguible. Al presentar este tipo de objeto en un graficador, el instrumento de escritura puede perforar el papel; en una pantalla vectorial, el fósforo del CRT se puede quemar

**Figura 6.15**

Volumen de vista truncado para una proyección paralela ortográfica. DOP es la dirección de la proyección.



**Figura 6.16** Volumen de vista truncado para una proyección de perspectiva.

con el rayo de electrones; y en una grabadora vectorial de película fotográfica, la alta concentración de luz ocasiona la aparición de un área blanca borrosa. Además, un objeto muy cercano al centro de proyección se puede extender por la ventana como si fueran varios palillos inconexos, sin estructura discernible. La especificación apropiada del volumen de vista puede eliminar estos problemas.

¿Cómo se establece la correspondencia entre el contenido del volumen de vista y la superficie de presentación? En primer lugar, considere el cubo unidad que se extiende de 0 a 1 en cada una de las tres dimensiones de las **coordenadas de proyección normalizadas (NPC, normalized projection coordinates)**. El volumen de vista se transforma en un sólido rectangular de NPC que se extiende de  $x_{\min}$  a  $x_{\max}$  sobre el eje  $x$ , de  $y_{\min}$  a  $y_{\max}$  sobre el eje  $y$ , y de  $z_{\min}$  a  $z_{\max}$  sobre el eje  $z$ . El plano de recorte anterior se convierte en el plano  $z_{\max}$  y el plano de recorte posterior se convierte en el plano  $z_{\min}$ . En forma similar, el lado  $u_{\min}$  del volumen de vista es ahora el plano  $x_{\min}$  y el lado  $u_{\max}$  se convierte en el plano  $x_{\max}$ . Por último, el lado  $v_{\min}$  del volumen de vista se convierte en el plano  $y_{\min}$  y el lado  $v_{\max}$  se convierte en el plano  $y_{\max}$ . Esta porción rectangular sólida de NPC, llamada **área de vista tridimensional**, está dentro del cubo unidad.

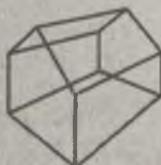
A su vez, se establece una correspondencia entre la cara  $z = 1$  de este cubo unidad y el mayor cuadrado que puede inscribirse en la pantalla. Para crear una presentación alambrada del contenido del área de vista tridimensional (que es el contenido del volumen de vista), simplemente se descarta el componente  $z$  de cada primitiva de salida y se presenta la primitiva. En el capítulo 13 veremos que la eliminación de superficies ocultas simplemente usa el componente  $z$  para determinar cuáles son las primitivas de salida más cercanas al observador y por lo tanto visibles.

PHIGS usa dos matrices de  $4 \times 4$ , la matriz de orientación de vista y la matriz de correspondencia de vista, para representar el conjunto completo de especificaciones de vista. VRP, VPN y VUP se combinan para formar la **matriz de orientación de vista**, que transforma las posiciones representadas en coordenadas de mundo a posiciones representadas en VRC. Esta transformación lleva los ejes  $u$ ,  $v$  y  $n$  a los ejes  $x$ ,  $y$  y  $z$ , respectivamente.

Las especificaciones de volumen de vista, indicadas por PRP,  $u_{\min}$ ,  $u_{\max}$ ,  $v_{\min}$ ,  $v_{\max}$ ,  $F$  y  $B$ , junto con la especificación de área de vista tridimensional, indicada por  $x_{\min}$ ,  $x_{\max}$ ,  $y_{\min}$ ,  $y_{\max}$ ,  $z_{\min}$  y  $z_{\max}$ , se combinan para formar la **matriz de correspondencia de vista**, que transforma puntos en VRC a puntos en coordenadas de proyección normalizadas. En la sección 7.3.4 se analizan las llamadas a subrutinas que forman la matriz de orientación de vista y la matriz de correspondencia de vista se analizan en la sección 7.3.4.

En la sección 6.4 veremos cómo obtener diversas vistas usando los conceptos presentados en esta sección. En la sección 6.5 se presentan los conceptos matemáticos básicos de las proyecciones geométricas planas, mientras que en la sección 6.6 se desarrollan las matemáticas y los algoritmos necesarios para toda la operación de vista.

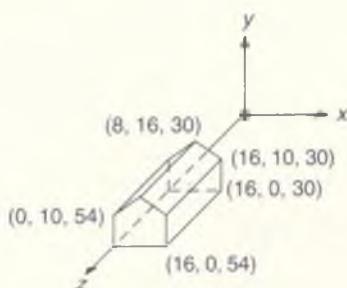
## 6.4 Ejemplos de vista tridimensional



**Figura 6.17**

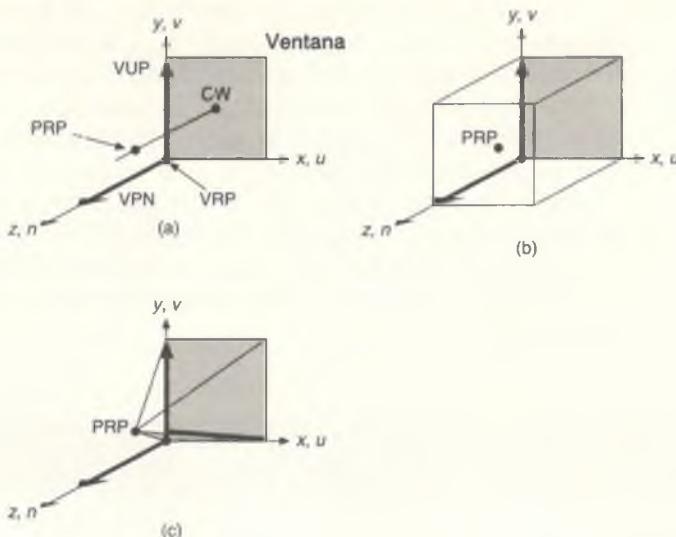
Proyección de perspectiva de dos puntos de una casa.

En esta sección veremos cómo aplicar los conceptos de visualización básicos presentados en la sección 6.3 para crear diversas proyecciones, como la que se presenta en la figura 6.17. Como la casa que aparece en esta figura se utiliza a lo largo de la sección, será útil recordar sus dimensiones y su posición, las cuales se indican en la figura 6.18. En cada vista que analicemos se presentará una tabla con VRP, VPN, VUP, PRP, ventana y tipo de proyección (de perspectiva o paralela). En toda la sección se supone el área de vista tridimensional por omisión, que es el cubo unidad en NPC. La notación (WC) o (VRC) se añade a la tabla como recordatorio del sistema de coordenadas (de mundo o de referencia de vista, respectivamente) para el cual se proporciona el parámetro de vista. La forma de la tabla que se presenta aquí es para la especificación de vista por omisión utilizada por PHIGS. Los valores por omisión se presentan en la figura



**Figura 6.18**

Esta casa se usa a lo largo del presente capítulo como ejemplo de un conjunto de datos de coordenadas de mundo. Sus coordenadas se extienden de 30 a 54 en z, de 0 a 16 en x y de 0 a 16 en y.



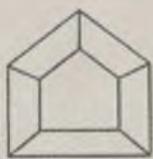
**Figura 6.19** Relación entre las coordenadas de mundo y de referencia de vista. (a) La especificación de vista por omisión: VRP está en el origen, VUP es el eje  $y$  y VPN es el eje  $z$ . Esta disposición hace que el sistema VRC de  $u$ ,  $v$  y  $n$  coincida con el sistema de coordenadas de mundo  $x$ ,  $y$ ,  $z$ . La ventana se extiende de 0 a 1 sobre  $u$  y  $v$ , y PRP está en  $(0.5, 0.5, 1.0)$ . (b) Volumen de vista por omisión de proyección paralela. (c) Volumen de vista si la proyección por omisión fuera de perspectiva.

6.91(a); el volumen de vista correspondiente a estos valores se muestra en la figura 6.19(b). Si el tipo de proyección es de perspectiva en lugar de paralela, el volumen de vista será la pirámide presentada en la figura 6.19(c).

Parámetro de visualización	Valor	Comentarios
VRP(WC)	$(0, 0, 0)$	origen
VPN(WC)	$(0, 0, 1)$	eje $z$
VUP(WC)	$(0, 1, 0)$	eje $y$
PRP(WC)	$(0.5, 0.5, 1.0)$	
ventana (VRC)	$(0, 1, 0, 1)$	
tipo de proyección	paralela	

#### 6.4.1 Proyecciones de perspectiva

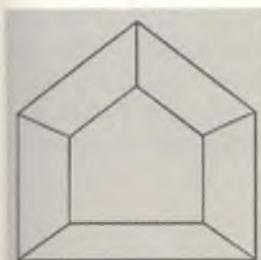
Para obtener la vista en perspectiva frontal de un punto de la casa presentada en la figura 6.20 (esta figura y todas las similares se realizaron con el programa SPHIGS, analizado en el capítulo 7), se coloca el centro de proyección (que puede considerarse como la posición del observador) en  $x = 8$ ,  $y = 6$  y  $z = 84$ . El valor  $x$  se selecciona para que esté en el centro horizontal de la casa y el valor



**Figura 6.20**  
Proyección de perspectiva  
de un punto de la casa.

y para que corresponda a la altura aproximada del ojo de un observador de pie en el plano  $(x, z)$ ; el valor  $z$  es arbitrario. En este caso,  $z$  está a 30 unidades del frente de la casa (plano  $z = 54$ ). La ventana se ha hecho de gran tamaño para garantizar que la casa quepa en el volumen de vista. Los demás parámetros de vista tienen sus valores por omisión, de manera que el conjunto global de parámetros de visualización es el siguiente:

VRP(WC)	(0, 0, 0)
VPN(WC)	(0, 0, 1)
VUP(WC)	(0, 1, 0)
PRP(WC)	(8, 6, 84)
ventana (VRC)	(-50, 50, -50, 50)
tipo de proyección	perspectiva



**Figura 6.21**  
Proyección de perspectiva  
centrada de una casa.

Aunque la imagen en la figura 6.20 es en realidad una proyección de perspectiva de la casa, es muy pequeña y no está centrada en la superficie de vista. Preferiríamos una proyección más centrada de la casa, que abarque casi toda la superficie de vista, como en la figura 6.21. Es más fácil producir este efecto si coinciden el plano de vista y el plano frontal de la casa. Así, como el frente de la casa se extiende de 0 a 16 tanto en  $x$  como en  $y$ , se obtienen resultados razonables con una ventana que se extiende de -1 a 17 en  $x$  y  $y$ .

Colocamos el plano de vista en la cara frontal de la casa ubicando el VRP en cualquier lugar del plano  $z = 54$ ;  $(0, 0, 54)$ , la esquina inferior izquierda frontal de la casa, es un punto adecuado para ello. Para que el centro de la proyección sea el mismo que en la figura 6.20, el PRP, que se halla en el sistema VRC, debe estar en  $(8, 6, 30)$ . En la figura 6.22 se muestra esta nueva disposición de VRC, VRP y PRP, que corresponde al siguiente conjunto de parámetros:

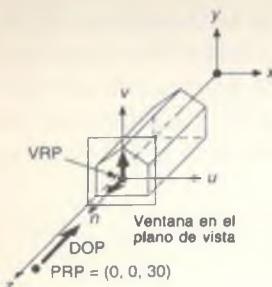
VRP(WC)	(0, 0, 54)
VPN(WC)	(0, 0, 1)
VUP(WC)	(0, 1, 0)
PRP(VRC)	(8, 6, 30)
ventana (VRC)	(-1, 17, -1, 17)
tipo de proyección	perspectiva



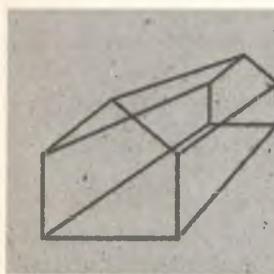
**Figura 6.22**  
Situación de vista para la  
figura 6.21.

Este mismo resultado se puede obtener de otras maneras. Por ejemplo, con VRP en  $(8, 6, 54)$  como en la figura 6.23, el centro de la proyección, indicado por PRP, se convierte en  $(0, 0, 30)$ . También hay que cambiar la ventana, ya que su definición se basa en el sistema VRC, cuyo origen es VRP. La ventana apropiada se extiende de -9 a 9 en  $u$  y de -7 a 11 en  $v$ . Con respecto a la casa, ésta es la misma ventana que se usó en el ejemplo previo, pero ahora se especifica en un sistema VRC diferente. Como la dirección de vista superior es el eje  $y$ , los ejes  $u$  y  $x$  son paralelos, lo mismo que los ejes  $v$  y  $y$ . En resumen, los

siguientes parámetros de visualización, presentados en la figura 6.23, producen también la figura 6.21:



**Figura 6.23**  
Situación de visualización alternativa para la figura 6.21.



**Figura 6.24**  
Proyección de perspectiva de una casa desde (36, 25, 74), con VPN paralela al eje z.

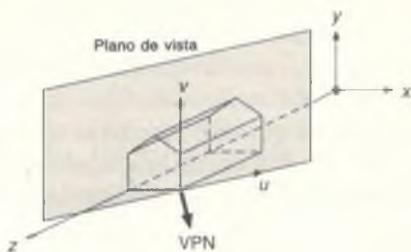
VRP(WC)	(8, 6, 54)
VPN(WC)	(0, 0, 1)
VUP(WC)	(0, 1, 0)
PRP(VRC)	(0, 0, 30)
ventana (VRC)	(-9, 9, -7, 11)
tipo de proyección	perspectiva

Intentemos ahora obtener la proyección de perspectiva de dos puntos que se muestra en la figura 6.17. El centro de proyección es análogo a la posición de una cámara que fotografía objetos en coordenadas de mundo. Teniendo en cuenta esta analogía, el centro de proyección en la figura 6.17 parece estar un poco hacia arriba y a la derecha de la casa, visto desde el eje z positivo. El centro de proyección exacto es (36, 25, 74). Ahora, si se elige como VRP la esquina de la casa en (16, 0, 54), este centro de proyección se encuentra en (20, 25, 20) con respecto a VRP. Si el plano de vista coincide con el frente de la casa (el plano  $z = 54$ ), una ventana que abarca de -20 a 20 en  $u$  y de -5 a 35 en  $v$  tiene ciertamente el tamaño suficiente para contener la proyección. Por lo tanto, podemos especificar la vista de la figura 6.24 con los siguientes parámetros de vista:

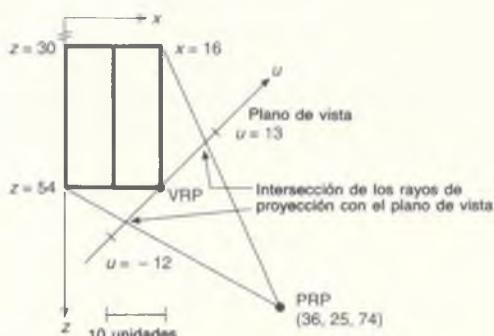
VRP(WC)	(16, 0, 54)
VPN(WC)	(0, 0, 1)
VUP(WC)	(0, 1, 0)
PRP(VRC)	(0, 25, 20)
ventana (VRC)	(-20, 20, -5, 35)
tipo de proyección	perspectiva

Esta vista es similar, pero obviamente distinta, que la que se presentó en la figura 6.17. Una diferencia es que la figura 6.17 es una proyección de perspectiva de dos puntos, mientras que la figura 6.24 es una perspectiva de un punto. Resulta evidente que la figura 6.17 no se puede producir con sólo mover el centro de proyección. De hecho, hay que reorientar el plano de vista para que corte los ejes  $x$  y  $z$ , asignando VPN igual a (1, 0, 1). De esta manera, los parámetros de vista para la figura 6.17 son los siguientes:

VRP(WC)	(16, 0, 54)
VPN(WC)	(1, 0, 1)
VUP(WC)	(1, 0, 1)
PRP(VRC)	(0, 25, 20 $\sqrt{2}$ )
ventana (VRC)	(-20, 20, -5, 35)
tipo de proyección	perspectiva



**Figura 6.25** El plano de vista y el sistema VRC correspondientes a la figura 6.17.



**Figura 6.26** Vista superior (planta) de una casa para determinar un tamaño apropiado de ventana.

En la figura 6.25 se muestra el plano de vista establecido con esta VPN. Se usa el componente  $n 20\sqrt{2}$  de PRP para que el centro de proyección esté a una distancia de  $20\sqrt{2}$  del VRP en el plano ( $x, y$ ), como se muestra en la figura 6.26.

Hay dos maneras de elegir una ventana que rodee totalmente la proyección, como lo hace la ventana de la figura 6.17. Podemos estimar el tamaño de la proyección de la casa sobre el plano de vista usando un bosquejo, como en la figura 6.26, para calcular las intersecciones de los proyectores con el plano de vista. Sin embargo, una mejor alternativa es permitir que los límites de la ventana sean variables en un programa y que se determinen interactivamente con un dispositivo localizador o valuador.

La figura 6.27 se obtiene de la misma proyección que la figura 6.17, pero la ventana tiene distinta orientación. En los ejemplos anteriores, el eje  $v$  del sistema VRC era paralelo al eje  $y$  y del sistema de coordenadas mundiales; así, la ventana (dos de sus lados paralelos al eje  $v$ ) se alineaba con los lados verticales de la casa. La figura 6.27 tiene exactamente los mismos parámetros de vista que la figura 6.17, excepto que el VUP se ha rotado a unos  $10^\circ$  del eje  $y$ .



**Figura 6.27**  
Proyección de la casa  
obtenida por la rotación  
del VUP.

**Ejemplo 6.1**

**Problema:** Una vez que se ha establecido el sistema VRC, todo el procesamiento gráfico subsecuente se lleva a cabo con ese sistema de coordenadas, procedimiento que examinaremos con mayor detalle en la sección 6.6. Antes de estos pasos de procesamiento, hay que transformar las coordenadas de mundo en nuestro conjunto de datos a coordenadas VRC. Esta transformación se puede efectuar con una sola matriz que lleve a cabo la rotación y la traslación. ¿Cuál es la forma general de esta matriz? ¿Qué valores específicos deben tener sus elementos para la situación de visualización ilustrada en la figura 6.17?

**Respuesta:** El método que usaremos se propuso en la sección 5.8 y se ilustra con las ecuaciones (5.60) a (5.64). Allí se trasladaba y rotaba un conjunto arbitrario de líneas para asumir una nueva posición en el sistema de coordenadas  $x$ ,  $y$ ,  $z$ , a través de la composición de una matriz de traslación  $T$  y una matriz de rotación  $R$  para formar la matriz  $M$ . Queremos seguir el mismo procedimiento, pero ahora deseamos reorientar el sistema de coordenadas  $u$ ,  $v$ ,  $n$  para que coincida con el sistema de coordenadas de mundo. La matriz que lleve a cabo esta transformación es la que estamos buscando.

Primero hay que trasladar el sistema VRC al origen. Con base en lo visto en la sección 5.8, esta traslación se obtiene con la matriz  $T$ , la cual es más que

$$T = \begin{bmatrix} 1 & 0 & 0 & -VRP_x \\ 0 & 1 & 0 & -VRP_y \\ 0 & 0 & 1 & -VRP_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

El método utilizado en la sección 5.8, en el cual hallamos una matriz de rotación (una matriz ortogonal especial) a través de la determinación de a dónde enviaba los ejes principales, es el que usaremos aquí. Los componentes de los vectores unidad que yacen sobre las direcciones  $u$ ,  $v$  y  $n$  constituyen los elementos de esta matriz. Así, podemos encontrar los elementos de la matriz de rotación,  $R$ , si observamos que el vector VPN debe rotarse al eje  $z$ , que el eje  $u$  es perpendicular a VUP y VPN y que el eje  $v$  es perpendicular a  $n$  y  $u$ . De esta manera,

$$n = \frac{\text{VPN}}{\| \text{VPN} \|}, \quad u = \frac{\text{VUP} \times \text{VPN}}{\| \text{VUP} \times \text{VPN} \|}, \quad v = n \times u,$$

donde  $u$ ,  $v$  y  $n$  representan vectores unidad. La matriz de rotación resultante es

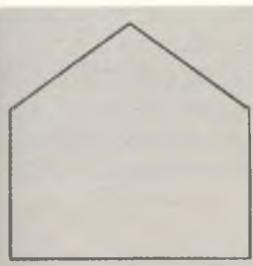
$$R = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Por lo tanto, la matriz que buscamos es

$$M = R \cdot T = \begin{bmatrix} u_x & u_y & u_z & -(u_x \cdot \text{VRP}_x + u_y \cdot \text{VRP}_y + u_z \cdot \text{VRP}_z) \\ v_x & v_y & v_z & -(v_x \cdot \text{VRP}_x + v_y \cdot \text{VRP}_y + v_z \cdot \text{VRP}_z) \\ n_x & n_y & n_z & -(n_x \cdot \text{VRP}_x + n_y \cdot \text{VRP}_y + n_z \cdot \text{VRP}_z) \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Ahora, para los valores específicos aplicables a la figura 6.17, tenemos  $n = [\frac{\sqrt{2}}{2}, 0, \frac{\sqrt{2}}{2}]^T$ ,  $u = [\frac{\sqrt{2}}{2}, 0, -\frac{\sqrt{2}}{2}]^T$  y  $v = [0, 1, 0]^T$ . Como los componentes de VRP son 16.0, 0.0 y 54.0, se tiene que los términos de traslación en  $M$  son 26.8701, 0.0 y -49.4975.

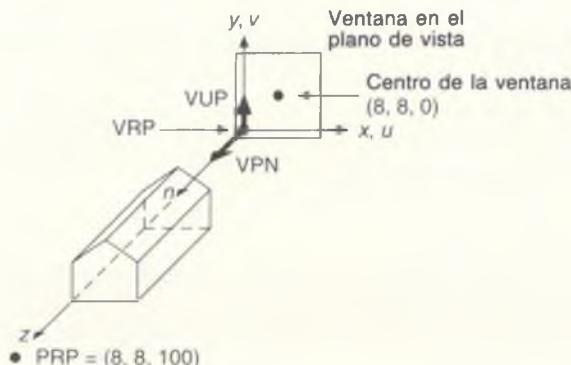
## 6.4.2 Proyecciones paralelas



**Figura 6.28**  
Proyección paralela frontal  
de la casa.

Para crear una proyección paralela frontal de la casa (Fig. 6.28), la dirección de la proyección se hace paralela al eje  $z$ . Recuerde que la dirección de la proyección está determinada por el PRP y el centro de la ventana. Con el sistema VRC por omisión y una ventana de  $(-1, 17, -1, 17)$ , el centro de la ventana es  $(8, 8, 0)$ . Un PRP de  $(8, 8, 100)$  proporciona una dirección de proyección paralela al eje  $z$ . En la figura 6.29 se muestra la situación de visualización que crea la figura 6.28. Los parámetros de visualización son los siguientes:

VRP(WC)	(0, 0, 0)
VPN(WC)	(0, 0, 1)
VUP(WC)	(0, 1, 0)
PRP(VRC)	(8, 8, 100)
ventana (VRC)	(-1, 17, -1, 17)
tipo de proyección	paralela

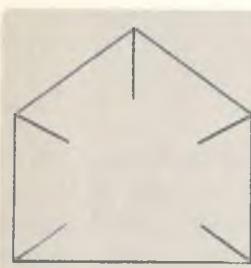


**Figura 6.29** Situación de visualización que crea la figura 6.28, una vista frontal de la casa. El PRP puede ser cualquier punto con  $x = 8$  y  $y = 8$ .

Para crear una vista lateral usaríamos una situación de visualización con el plano ( $y, z$ ) (o cualquier plano paralelo) como plano de vista. La vista superior de la casa se crea usando el plano ( $x, z$ ) como plano de vista y VPN como eje  $y$ . Sin embargo, hay que cambiar la dirección de vista superior de  $+y$  y usar en cambio el eje  $x$  negativo.

Véase [FOLE90] para un tratamiento completo de los casos de vista lateral y superior, así como ejemplos de proyecciones oblicuas.

### 6.4.3 Volúmenes de vista finitos



**Figura 6.30**  
Proyección de perspectiva  
de la casa con plano de  
recorte posterior en  $z = 31$ .

Hasta ahora, en todos los ejemplos hemos supuesto que el volumen de vista es infinito. Los planos de recorte anterior y posterior, descritos en la sección 6.3, ayudan a determinar un **volumen de vista finito**. Estos planos, los dos paralelos al plano de vista, se encuentran a una distancia  $F$  y  $B$ , respectivamente, del VRP, medida a lo largo de VPN. Para evitar un volumen de vista negativo, debemos asegurar que  $F$  sea algebraicamente mayor que  $B$ . O

Con la siguiente especificación de vista a la cual se han agregado  $F$  y  $B$ , se obtiene una vista de perspectiva frontal de la casa con la pared posterior recortada (Fig. 6.30). Si se indica una distancia, se supone un recorte con respecto al plano correspondiente; en caso contrario, no se supone ningún recorte. La especificación de vista es la siguiente:

VRP(WC)	(0, 0, 54)	esquina inferior izquierda de la casa
VPN(WC)	(0, 0, 1)	eje $z$
VUP(WC)	(0, 1, 0)	eje $y$
PRP(VRC)	(8, 6, 30)	
ventana (VRC)	(-1, 17, -1, 17)	
tipo de proyección perspectiva		
$F(VRC)$	+1	una unidad frente a la casa, en $z = 54 + 1 = 55$
$B(VRC)$	-23	una unidad detrás de la casa, en $z = 54 - 23 = 31$

La situación de vista para este caso es la misma que para la figura 6.22, excepto por la adición de los planos de recorte.

Si los planos de recorte anterior y posterior se mueven dinámicamente, la estructura tridimensional del objeto que se observa se puede discernir con mayor facilidad que con una vista estática.

## 6.5 Las matemáticas de las proyecciones geométricas planas

Aquí presentaremos las matemáticas básicas de las proyecciones geométricas planas. Para simplificar, comenzaremos por suponer que, en la proyección de perspectiva, el plano de proyección es normal al eje  $z$  en  $z = d$  y que, en la



proyección paralela, el plano de proyección es el plano  $z = 0$ . Cada una de las proyecciones se puede definir con una matriz de  $4 \times 4$ . Esta representación es conveniente, ya que la matriz de proyección se puede componer con matrices de transformación, lo que permite representar dos operaciones (transformación y luego proyección) como una sola matriz. En la sección 6.6 se analizan los planos de proyección arbitrarios.

En esta sección obtendremos matrices de  $4 \times 4$  para varias proyecciones, comenzando con un plano de proyección paralelo al plano  $xy$  en la posición  $z = d$ , por consiguiente a una distancia  $d$  del origen, y con un punto  $P$  al cual se proyectará. Para calcular  $P_p = (x_p, y_p, z_p)$ , la proyección de perspectiva de  $(x, y, z)$  sobre el plano de proyección en  $z = d$ , usamos los triángulos similares de la figura 6.31 para escribir las razones

$$\frac{x_p}{d} = \frac{x}{z}; \quad \frac{y_p}{d} = \frac{y}{z}. \quad (6.1)$$

Al multiplicar cada lado por  $d$  se obtiene

$$x_p = \frac{d \cdot x}{z} = \frac{x}{z/d}, \quad y_p = \frac{d \cdot y}{z} = \frac{y}{z/d}. \quad (6.2)$$

La distancia  $d$  es un factor de escala que se aplica a  $x_p$  y  $y_p$ . La división entre  $z$  ocasiona que la proyección de perspectiva de los objetos más distantes sea más pequeña que la de los objetos cercanos. Se permiten todos los valores de  $z$  excepto  $z = 0$ . Puede haber puntos detrás del centro de proyección en el eje  $z$  negativo o entre el centro de proyección y el plano de proyección.

La transformación de la ecuación (6.2) se puede expresar como una matriz de  $4 \times 4$ :

$$M_{\text{per}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}. \quad (6.3)$$

Al multiplicar el punto  $P = [x \ y \ z \ 1]^T$  por la matriz  $M_{\text{per}}$  se obtiene un punto homogéneo general  $[X \ Y \ Z \ W]^T$ :

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = M_{\text{per}} \cdot P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \quad (6.4)$$

$$[X \ Y \ Z \ W]^T = \begin{bmatrix} x \\ y \\ z \\ \frac{z}{d} \end{bmatrix}^T. \quad (6.5)$$

Ahora, al dividir entre  $W$  (que es igual a  $z/d$ ) y eliminar la cuarta coordenada para regresar a tres dimensiones, se tiene

$$\left( \frac{X}{W}, \frac{Y}{W}, \frac{Z}{W} \right) = (x_p, y_p, z_p) = \left( \frac{x}{z/d}, \frac{y}{z/d}, d \right); \quad (6.6)$$

estas ecuaciones son el resultado correcto de la ecuación (6.1), más la coordenada  $z$  transformada de  $d$ , que es la posición del plano de proyección sobre el eje  $z$ .

Una formulación alternativa de la proyección de perspectiva coloca el plano de proyección en  $z = 0$  y el centro de proyección en  $z = -d$ , como se muestra en la figura 6.32. Por semejanza de triángulos se obtiene

$$\frac{x_p}{d} = \frac{x}{z+d}, \quad \frac{y_p}{d} = \frac{y}{z+d}. \quad (6.7)$$

Multiplicando por  $d$  se obtiene

$$x_p = \frac{d+x}{z+d} = \frac{x}{(z/d)+1}, \quad y_p = \frac{d+y}{z+d} = \frac{y}{(z/d)+1}. \quad (6.8)$$

La matriz es

$$M'_{\text{per}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix} \quad (6.9)$$

Esta formulación permite que  $d$ , la distancia al centro de proyección, tienda a infinito.

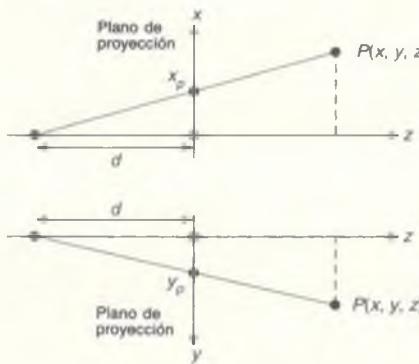


Figura 6.32 Proyección de perspectiva alternativa.

La proyección ortográfica sobre un plano de proyección en  $z = 0$  es bastante sencilla. La dirección de la proyección es igual que la de la normal al plano de proyección, es decir, el eje  $z$  en este caso. De esta manera, el punto  $P$  se proyecta como

$$x_p = x, \quad y_p = y, \quad z_p = 0. \quad (6.10)$$

Esta proyección se expresa con la matriz

$$M_{\text{ort}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.11)$$

Observe que conforme  $d$  tiende a infinito, la ecuación (6.9) se convierte en la ecuación (6.11). Esto se debe a que la proyección ortográfica es un caso especial de la proyección de perspectiva.

$M_{\text{per}}$  se aplica únicamente al caso especial en el cual el centro de proyección está en el origen;  $M_{\text{ort}}$  se aplica sólo si la dirección de proyección es paralela al eje  $z$ . Existe una formulación general, citada en [FOLE90], que no sólo elimina estas restricciones, sino que además integra las proyecciones paralela y de perspectiva.

En esta sección hemos visto cómo formular  $M_{\text{per}}$ ,  $M'_{\text{per}}$  y  $M_{\text{ort}}$ , casos en los cuales el plano de proyección es perpendicular al eje  $z$ . En la sección 6.6 eliminaremos esta restricción y consideraremos el recorte implicado por los volúmenes de vista finitos.

### Ejemplo 6.2

**Problema:** La matriz  $M_{\text{per}}$  define una proyección de perspectiva de un punto. Describa una matriz que defina una proyección de perspectiva de dos puntos y su relación con la matriz  $M_{\text{per}}$  que acabamos de obtener. ¿Cuál es la forma de la matriz que define una perspectiva de tres puntos?

**Respuesta:** Como se propuso en la sección 6.4.1, es necesario orientar el plano de vista de manera que corte más de un eje, el eje  $z$  en este caso. Por ejemplo, especificaremos una rotación sobre el eje  $y$  para que el plano de vista corte tanto el eje  $x$  como el  $z$ . La nueva matriz se obtiene postmultipliando  $M_{\text{per}}$  por la matriz

$$\begin{bmatrix} \cos \theta & 0 & \operatorname{sen} \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\operatorname{sen} \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

donde  $\theta$  es el ángulo de rotación sobre el eje  $y$ . La matriz resultante es

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ -\sin\theta/d & 0 & \cos\theta/d & 0 \end{bmatrix}$$

Observe la aparición de un término distinto de cero en la posición  $a_{41}$  de la matriz compuesta. Esto indica un punto de fuga en el eje  $x$ . Si realizáramos una composición similar con una rotación sobre el eje  $x$ , un término distinto de cero en  $a_{42}$  indicaría un punto de fuga en el eje  $y$ . Si se combinaran rotaciones sobre los ejes  $x$  y  $y$ , se produciría una perspectiva de tres puntos.

## 6.6 Implantación de proyecciones geométricas planas

Dado un volumen de vista y una proyección, consideremos cómo se aplica la operación de vista de recorte y proyección. Como lo sugiere el modelo conceptual para la vista (Fig. 6.2), recortaríamos las líneas con respecto al volumen de vista calculando sus intersecciones con cada uno de los seis planos que definen el volumen de vista. Las líneas que quedaran después del recorte se proyectarían sobre el plano de vista, a través de la solución de ecuaciones simultáneas para la intersección de los proyectores con el plano de vista. Después las coordenadas del sistema de mundo tridimensional se transformarían al sistema bidimensional del dispositivo. Sin embargo, se requieren numerosos cálculos para este proceso, repetidos para gran cantidad de líneas, lo que implica gran tiempo de computación. Por fortuna hay un procedimiento más eficiente, basado en la estrategia “divide y vencerás”, para descomponer un problema difícil en una serie de problemas más sencillos.

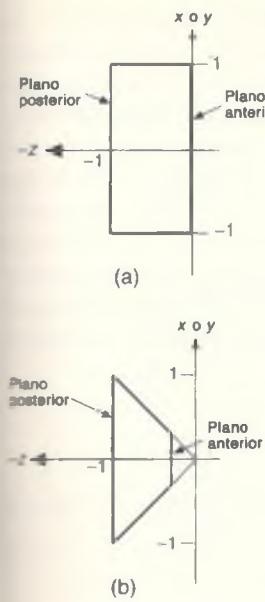
Es más fácil hacer recortes con respecto a ciertos volúmenes de vista que con respecto al general (los algoritmos de recorte se analizan en la sección 6.6.3). Por ejemplo, es fácil calcular las intersecciones de una línea con los planos de un volumen de vista de proyección paralela definido por los seis planos

$$x = -1, \quad x = 1, \quad y = -1, \quad y = 1, \quad z = 0, \quad z = -1. \quad (6.12)$$

Esta situación también se aplica al volumen de vista de proyección de perspectiva definido por los planos

$$x = z, \quad x = -z, \quad y = z, \quad y = -z, \quad z = -z_{\min}, \quad z = -1. \quad (6.13)$$

Estos volúmenes de vista canónicos se presentan en la figura 6.33.



**Figura 6.33**  
Los dos volúmenes de vista canónicos, para las proyecciones (a) paralela y (b) de perspectiva.

Nuestra estrategia es encontrar las transformaciones de normalización  $N_{\text{par}}$  y  $N_{\text{per}}$  que transformen un volumen de vista de proyección paralela o de perspectiva a su volumen de vista canónico paralelo o de perspectiva, respectivamente. Después de esto, se lleva a cabo el recorte y se realiza una proyección a dos dimensiones, usando las matrices de la sección 6.5. Esta estrategia presenta el riesgo de dedicar esfuerzo a la transformación de puntos que posteriormente se descartan en la operación de recorte, pero al menos es sencillo efectuar el recorte.

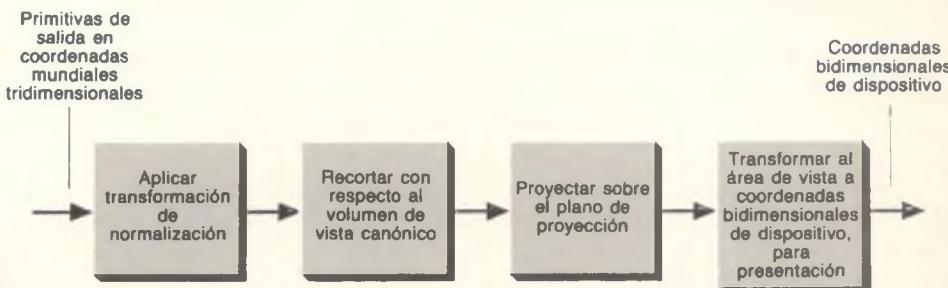
En la figura 6.34 se muestra la secuencia de procesos. Podemos reducir esta secuencia a una de transformación-recorte-transformación si combinamos los pasos 3 y 4 para formar una sola matriz de transformación. Con las proyecciones de perspectiva, también se requiere una división para establecer la correspondencia entre coordenadas homogéneas y coordenadas tridimensionales. Esta división viene después de la segunda transformación de la secuencia combinada. En la sección 6.6.4 se presenta una estrategia alternativa de recorte en coordenadas homogéneas.

Los lectores que estén familiarizados con PHIGS notarán que los volúmenes de vista canónicos de las ecuaciones (6.12) y (6.13) son diferentes de los volúmenes de vista por omisión de PHIGS: el cubo unidad de 0 a 1 en  $x$ ,  $y$  y  $z$  para la proyección paralela y la pirámide con ápice en  $(0.5, 0.5, 1.0)$  y lados que pasan por el cuadrado unidad de 0 a 1 en  $x$  y  $y$  en el plano  $z = 0$  para la proyección de perspectiva. Los volúmenes de vista canónicos se definen para simplificar las ecuaciones de recorte y para ofrecer la consistencia entre proyecciones paralelas y de perspectiva que se analiza en la sección 6.6.4. Por otra parte, los volúmenes de vista por omisión de PHIGS se definen de manera que la visualización bidimensional sea un caso especial de la visualización tridimensional.

En las secciones 6.6.1 y 6.6.2 se obtienen las transformaciones de normalización para las proyecciones paralela y de perspectiva que se usan como primer paso en la secuencia de transformación-recorte-transformación.

### 6.6.1 El caso de la proyección paralela

En esta sección obtendremos la transformación de normalización  $N_{\text{par}}$  para proyecciones paralelas, a fin de transformar posiciones de coordenadas de mundo de



**Figura 6.34** Implantación de la visualización tridimensional.

manera que el volumen de vista se transforme al volumen de vista canónico definido por la ecuación (6.12). Las coordenadas transformadas se recortan con respecto a este volumen de vista canónico y los resultados recortados se proyectan sobre el plano  $z = 0$ , para luego transformarse al área de vista con fines de presentación.

La transformación  $N_{\text{par}}$  se deriva para el caso más general, la transformación paralela oblicua (en lugar de ortográfica). De esta manera,  $N_{\text{par}}$  incluye una transformación de sesgo que hace que la dirección de proyección en las coordenadas de visualización sea paralela a  $z$ , incluso si no es paralela a VPN en las coordenadas ( $u, v, n$ ). Al incluir este sesgo podemos efectuar la proyección sobre el plano  $z = 0$  con sólo asignar  $z = 0$ . Si la proyección paralela es ortográfica, el componente de sesgo de la transformación de normalización se convierte en la identidad.

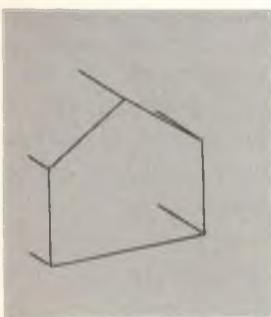
La serie de transformaciones que conforma  $N_{\text{par}}$  es la siguiente:

1. Trasladar VRP al origen.
2. Rotar VRC de manera que el eje  $n$  (VPN) se convierta en el eje  $z$ , el eje  $u$  se convierta en el eje  $x$  y el eje  $n$  se convierta en el eje  $y$ .
3. Sesgar de manera que la dirección de proyección sea paralela al eje  $z$ .
4. Trasladar y escalar al volumen de vista canónico de proyección paralela de la ecuación (6.12).

En PHIGS, los pasos 1 y 2 definen la **matriz de orientación de vista** y los pasos 3 y 4 definen la **matriz de correspondencia de visualización**.

En la figura 6.37 se muestra la secuencia de trasformaciones aplicada a un volumen de vista de proyección paralela y al bosquejo de una casa; en la figura 6.35 se presenta la proyección paralela que se obtiene como resultado.

El paso 1 es simplemente la translación  $T(-VRP)$ . Para el paso 2, usamos las propiedades de las matrices ortogonales especiales analizadas en las secciones 5.3 y 5.7 e ilustradas en la obtención de las ecuaciones (5.64) y (5.65). Los vectores fila de la matriz de rotación necesaria para llevar a cabo el paso 2 son los vectores unidad rotados en  $R$  a los ejes  $x, y$  y  $z$ . VPN gira el eje  $z$ , de manera que



**Figura 6.35**  
Proyección paralela final  
de la casa recortada.

$$R_z = \frac{\text{VPN}}{\|\text{VPN}\|}. \quad (6.14)$$

El eje  $u$ , perpendicular a VUP y a VPN y por consiguiente el producto cruz del vector unidad sobre VUP y  $R_z$  (que se halla en la misma dirección que VPN), se rota al eje  $x$ , de manera que

$$R_x = \frac{\text{VUP} \times R_z}{\|\text{VUP} \times R_z\|}. \quad (6.15)$$

En forma similar, el eje  $v$ , que es perpendicular a  $R_z$  y  $R_x$ , se rota al eje  $y$ , para que

$$R_y = R_z \times R_x. \quad (6.16)$$

Por lo tanto, la rotación en el paso 2 está indicada por la matriz

$$R = \begin{bmatrix} r_{1x} & r_{2x} & r_{3x} & 0 \\ r_{1y} & r_{2y} & r_{3y} & 0 \\ r_{1z} & r_{2z} & r_{3z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (6.17)$$

donde  $r_{1x}$  es el primer elemento de  $R_x$ , etcétera.

El tercer paso es sesgar el volumen de vista a lo largo del eje  $z$  para que todos sus planos sean normales a uno de los ejes del sistema de coordenadas. Esto se hace determinando el sesgo que se aplicará a la dirección de proyección (DOP) para que coincida con el eje  $z$ . Recuerde que DOP es el vector de PRP al centro de la ventana (CW) y que PRP se especifica en el sistema VRC. Los dos primeros pasos de la transformación han hecho que VRC corresponda al sistema de coordenadas mundiales, de manera que PRP está ahora en coordenadas mundiales. Por lo tanto, DOP es CW - PRP. Dado

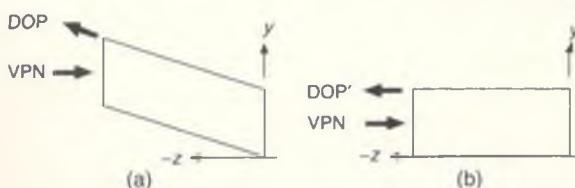
$$\text{DOP} = \begin{bmatrix} dop_x \\ dop_y \\ dop_z \\ 0 \end{bmatrix}, \quad \text{CW} = \begin{bmatrix} \frac{u_{\max} + u_{\min}}{2} \\ \frac{v_{\max} + v_{\min}}{2} \\ 0 \\ 1 \end{bmatrix}, \quad \text{PRP} = \begin{bmatrix} prp_u \\ prp_v \\ prp_n \\ 1 \end{bmatrix}, \quad (6.18)$$

entonces

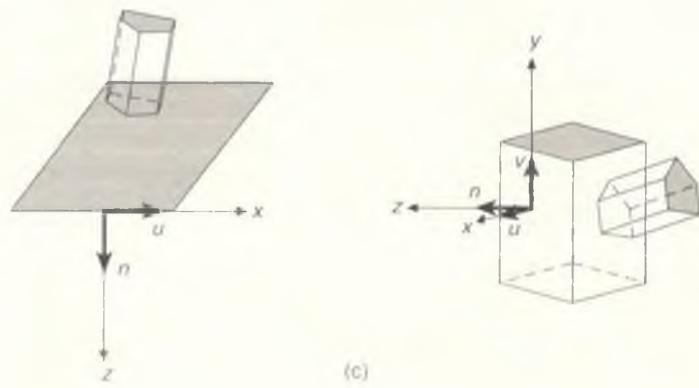
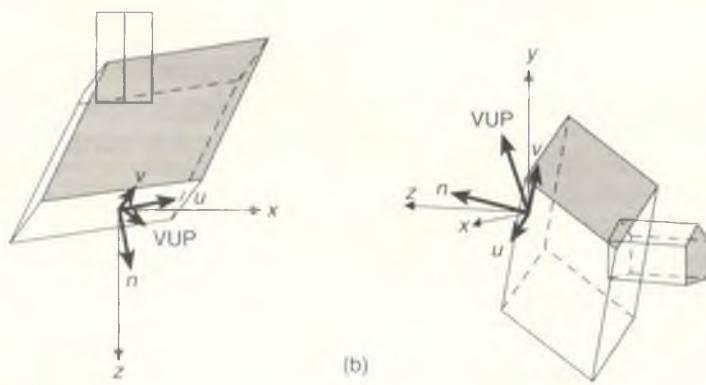
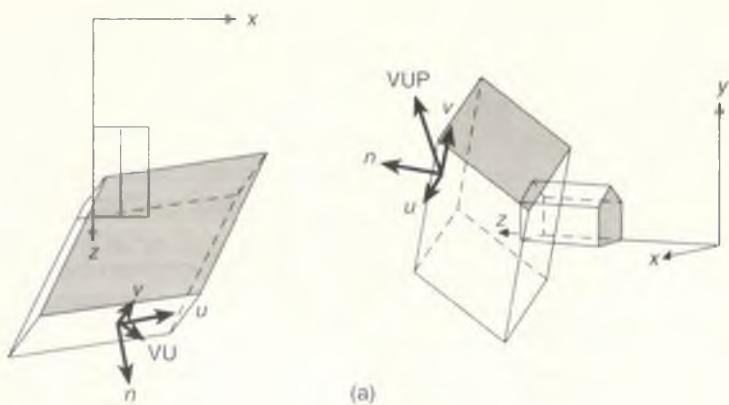
$$\text{DOP} = \text{CW} - \text{PRP}$$

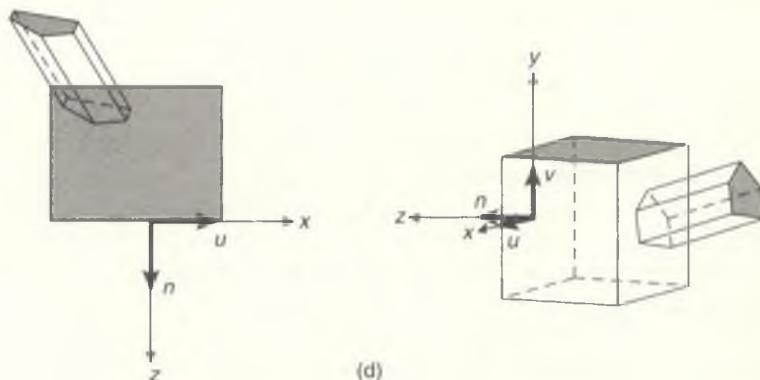
$$= \left[ \frac{u_{\max} + u_{\min}}{2} \quad \frac{v_{\max} + v_{\min}}{2} \quad 0 \quad 1 \right]^T - [prp_u \quad prp_v \quad prp_n \quad 1]^T. \quad (6.19)$$

En la figura 6.36 se muestra la DOP especificada y la DOP' transformada que deseamos.

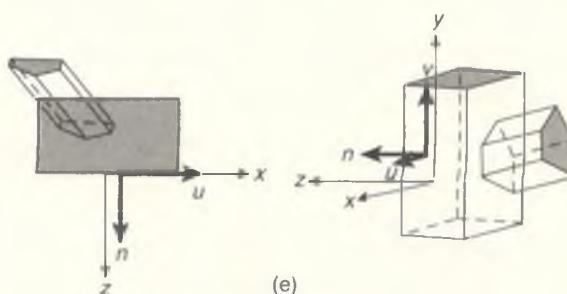


**Figura 6.36** Ilustración del sesgo usando como ejemplo la vista lateral del volumen de vista. El paralelogramo en (a) se sesga al rectángulo en (b); VPN no cambia porque es paralela al eje  $z$ .





(d)



(e)

**Figura 6.37** Resultados en diversas etapas de la secuencia de visualización de proyección paralela. En todos los casos se presenta una proyección paralela superior y fuera de eje. (a) Se presenta la situación de vista original. (b) VRP ha sido trasladado al origen. (c) El sistema de coordenadas \$(u, v, n)\$ se ha hecho rotar para que quede alineado con el sistema \$(x, y, z)\$. (d) El volumen de vista se ha sesgado para que la dirección de proyección (DOP) sea paralela al eje \$z\$. (e) El volumen de vista se ha trasladado y escalado al volumen de vista canónico de proyección paralela. Los parámetros de vista son VRP = \$(0.325, 0.8, 4.15)\$, VPN = \$(0.227, 0.267, 1.0)\$, VUP = \$(0.293, 1.0, 0.227)\$, PRP = \$(0.6, 0.0, -1.0)\$, ventana = \$(-1.425, 1.0, -1.0, 1.0)\$, \$F = 0.0\$, \$B = -1.75\$. (Figuras creadas con un programa escrito por L. Lu, The George Washington University.)

El sesgo se puede obtener con la matriz de sesgo ( $x, y$ ) de la ecuación (5.45) de la sección 5.7. Con coeficientes  $shx_{\text{par}}$  y  $shy_{\text{par}}$ , la matriz de sesgo es

$$SH_{\text{par}} = SH_{xy}(shx_{\text{par}}, shy_{\text{par}}) = \begin{bmatrix} 1 & 0 & shx_{\text{par}} & 0 \\ 0 & 1 & shy_{\text{par}} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (6.20)$$

Como se explicó en la sección 5.7,  $SH_{xy}$  no afecta a  $z$  mientras añade los términos  $z \cdot shx_{\text{par}}$  y  $z \cdot shy_{\text{par}}$ . Queremos encontrar valores para  $shx_{\text{par}}$  y  $shy_{\text{par}}$  tales que

$$\text{DOP}' = [0 \ 0 \ dop_z \ 0]^T = SH_{\text{par}} \cdot \text{DOP}. \quad (6.21)$$

Al llevar a cabo la multiplicación de la ecuación (6.21), seguida por cierta manipulación algebraica, se observa que la igualdad ocurre si

$$shx_{\text{par}} = \frac{dop_x}{dop_z}, \quad shy_{\text{par}} = \frac{dop_y}{dop_z}. \quad (6.22)$$

Observe que, para una proyección ortográfica,  $dop_x = dop_y = 0$ , de manera que  $shx_{\text{par}} = shy_{\text{par}} = 0$  y la matriz de sesgo se reduce a la identidad.

En la figura 6.38 se presenta el volumen de vista después de aplicar estos tres pasos de transformación. Las cotas del volumen son

$$u_{\min} \leq x \leq u_{\max}, \quad v_{\min} \leq y \leq v_{\max}, \quad B \leq z \leq F; \quad (6.23)$$

y  $F$  y  $B$  son las distancias desde VRP a lo largo de VPN, hasta los planos de recorte anterior y posterior, respectivamente.

El cuarto y último paso en este proceso es transformar el volumen de vista sesgado al volumen de vista canónico. Este paso se logra trasladando al origen el centro frontal del volumen de vista de la ecuación (6.23) y escalando al tamaño  $2 \times 2 \times 1$  del volumen de vista canónico final de la ecuación (6.12). Las transformaciones son

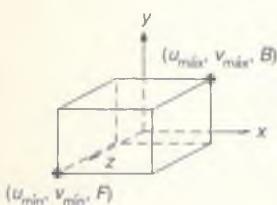
$$T_{\text{par}} = T \left( -\frac{u_{\max} + u_{\min}}{2}, -\frac{v_{\max} + v_{\min}}{2}, -F \right), \quad (6.24)$$

$$S_{\text{par}} = S \left( \frac{2}{u_{\max} + u_{\min}}, \frac{2}{v_{\max} + v_{\min}}, \frac{1}{F - B} \right). \quad (6.25)$$

Si  $F$  y  $B$  no se han especificado (porque no se efectúan recortes de plano anterior y posterior), se puede usar cualquier par de valores que satisfaga  $B \leq F$ . Los valores 0 y 1 son satisfactorios.

En resumen, tenemos

$$N_{\text{par}} = S_{\text{par}} \cdot T_{\text{par}} \cdot SH_{\text{par}} \cdot R \cdot T(-\text{VRP}). \quad (6.26)$$



**Figura 6.38**  
Volumen de vista después de los pasos de transformación 1 a 3.

$N_{\text{par}}$  transforma un volumen de vista arbitrario de proyección paralela al volumen de vista canónico de proyección paralela y por ende permite que las primitivas de salida se recorten con respecto al volumen de vista canónico de la proyección paralela.

### 6.6.2 El caso de la proyección de perspectiva

Ahora desarrollaremos la transformación de normalización  $N_{\text{per}}$  para proyecciones de perspectiva.  $N_{\text{per}}$  transforma posiciones de coordenadas mundiales de manera que el volumen de vista se convierta en el volumen de vista canónico de la proyección de perspectiva, la pirámide truncada con ápice en el origen definida por la ecuación (6.13). Después de aplicar  $N_{\text{per}}$ , se lleva a cabo el recorte con respecto a este volumen canónico y los resultados se proyectan sobre el plano de vista usando  $M_{\text{per}}$  (obtenida en la sección 6.5).

La serie de transformaciones que conforma  $N_{\text{per}}$  es la siguiente:

1. Trasladar VRP al origen.
2. Hacer rotar VRC de manera que el eje  $n$  (VPN) se convierta en el eje  $z$ , el eje  $u$  se convierta en el eje  $x$  y el eje  $v$  se convierta en el eje  $y$ .
3. Trasladar de manera que el centro de proyección (COP), indicado por PRP, esté en el origen.
4. Sesgar de manera que la línea central del volumen de vista se convierta en el eje  $z$ .
5. Escalar para que el volumen de vista se convierta en el volumen de vista canónico de la proyección de perspectiva, la pirámide derecha truncada que es definida por los seis planos de la ecuación (6.13).

En la figura 6.41 se muestra la aplicación de esta secuencia de transformaciones a un volumen de vista de proyección de perspectiva y a una casa. En la figura 6.39 se presenta la proyección de perspectiva resultante.

Los pasos 1 y 2 son los mismos que para la proyección paralela:  $R \cdot T(-\text{VRP})$ . El paso 3 es una traslación del centro de proyección (COP) al origen, como se requiere para el volumen de vista canónico de perspectiva. COP se especifica con respecto a VRP en VRC por medio de  $\text{PRP} = (prp_u, prp_v, prp_n)$ . Los VRC se han transformado a coordenadas mundiales con los pasos 1 y 2, por lo cual la especificación de COP en VRC está ahora en coordenadas mundiales. Por consiguiente, la traslación para el paso 3 es simplemente  $T(-\text{PRP})$ .

Para calcular el sesgo del paso 4, se examina la figura 6.40, donde se muestra una vista lateral del volumen de vista después de los pasos 1 a 3 de la transformación. Observe que la línea central del volumen de vista, que pasa por el origen y el centro de la ventana, no es igual que el eje  $-z$ . El propósito del sesgo es transformar la línea central al eje  $-z$ . La línea central del volumen de vista

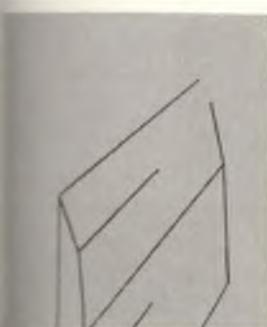
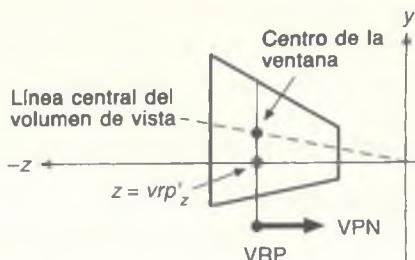


Figura 6.39  
Proyección de perspectiva  
de la casa recortada.



**Figura 6.40** Sección transversal del volumen de vista después de los pasos 1 a 3 de la transformación.

cambia de PRP (que ahora está en el origen) a CW, el centro de la ventana. Por lo tanto, es igual que la dirección para la proyección paralela, o sea, CW – PRP. En consecuencia, la matriz de sesgo es  $SH_{\text{par}}$ , igual que para la proyección paralela! Otra manera de considerar esta situación es que la traslación de –PRP en el paso 3, que llevó el centro de proyección al origen, también trasladó CW en –PRP; por consiguiente, después del paso 3, la línea central del volumen de vista pasa por el origen y CW – PRP.

Después de aplicar el sesgo, la ventana (y por lo tanto el volumen de vista) está centrada en el eje z. Los cotas de la ventana en el plano de proyección son

$$-\frac{u_{\max} - u_{\min}}{2} \leq x \leq \frac{u_{\max} - u_{\min}}{2}, \quad (6.27)$$

$$-\frac{v_{\max} - v_{\min}}{2} \leq y \leq \frac{v_{\max} - v_{\min}}{2}.$$

El VRP, que antes del paso 3 estaba en el origen, ha sido trasladado en el paso 3 y sesgado en el paso 4. La definición de VRP' como VRP después de las transformaciones de los pasos 3 y 4 es

$$\text{VRP}' = SH_{\text{par}} \cdot T(-\text{PRP}) \cdot [0 \ 0 \ 0 \ 1]^T. \quad (6.28)$$

El componente z de VRP', designado como  $vRP_z'$ , es igual a  $-prp_n$  ya que el sesgo ( $x, y$ )  $SH_{\text{par}}$  no afecta las coordenadas z.

El paso final es un escalamiento a lo largo de los tres ejes para crear el volumen de vista canónico definido por la ecuación (6.13) y presentado en la figura 6.42. De esta manera, el escalamiento se puede considerar como un proceso de dos subpasos. En el primero, se escala diferencialmente en x y en y para obtener los dos planos inclinados que acotan la pendiente unidad del volumen de vista. Este subpaso se lleva a cabo escalando la ventana para que su media altura y su media anchura sean  $-vRP_z'$ . Los factores de escalamiento apropiados para x y y son  $-2 \cdot vRP_z' / (u_{\max} - u_{\min})$  y  $-2 \cdot vRP_z' / (v_{\max} - v_{\min})$ , respectivamente.

En el segundo subpaso se escala uniformemente a lo largo de los tres ejes (para mantener las pendientes unidad), de manera que el plano de recorte posterior en  $z = vrp_z' + B$  se convierta en el plano  $z = -1$ . El factor de escalamiento para este subpaso es  $-1/(vrp_z' + B)$ . El factor de escalamiento tiene signo negativo para que el factor de escalamiento sea positivo, ya que  $vrp_z' + B$  es negativo.

Al unir estos dos subpasos se obtiene la transformación de escalamiento de la perspectiva:

$$S_{\text{per}} = S \left( \frac{2vrp_z'}{(u_{\max} - u_{\min})(vrp_z' + B)} \cdot \frac{2vrp_z'}{(v_{\max} - v_{\min})(vrp_z' + B)} \cdot \frac{-1}{vrp_z' + B} \right). \quad (6.29)$$

Al aplicar el escalamiento a  $z$  cambian las posiciones del plano de proyección y de los planos de recorte a las nuevas posiciones:<sup>1</sup>

$$z_{\text{proy}} = -\frac{vrp_z'}{vrp_z' + B}, \quad z_{\min} = -\frac{vrp_z' + F}{vrp_z' + B}, \quad z_{\max} = -\frac{vrp_z' + B}{vrp_z' + B} = -1. \quad (6.30)$$

En resumen, la transformación de normalización de la visualización que convierte el volumen de vista de la proyección de perspectiva en el volumen de vista canónico es

$$N_{\text{per}} = S_{\text{per}} \cdot SH_{\text{par}} \cdot T(-\text{PRP}) \cdot R \cdot T(-\text{VRP}). \quad (6.31)$$

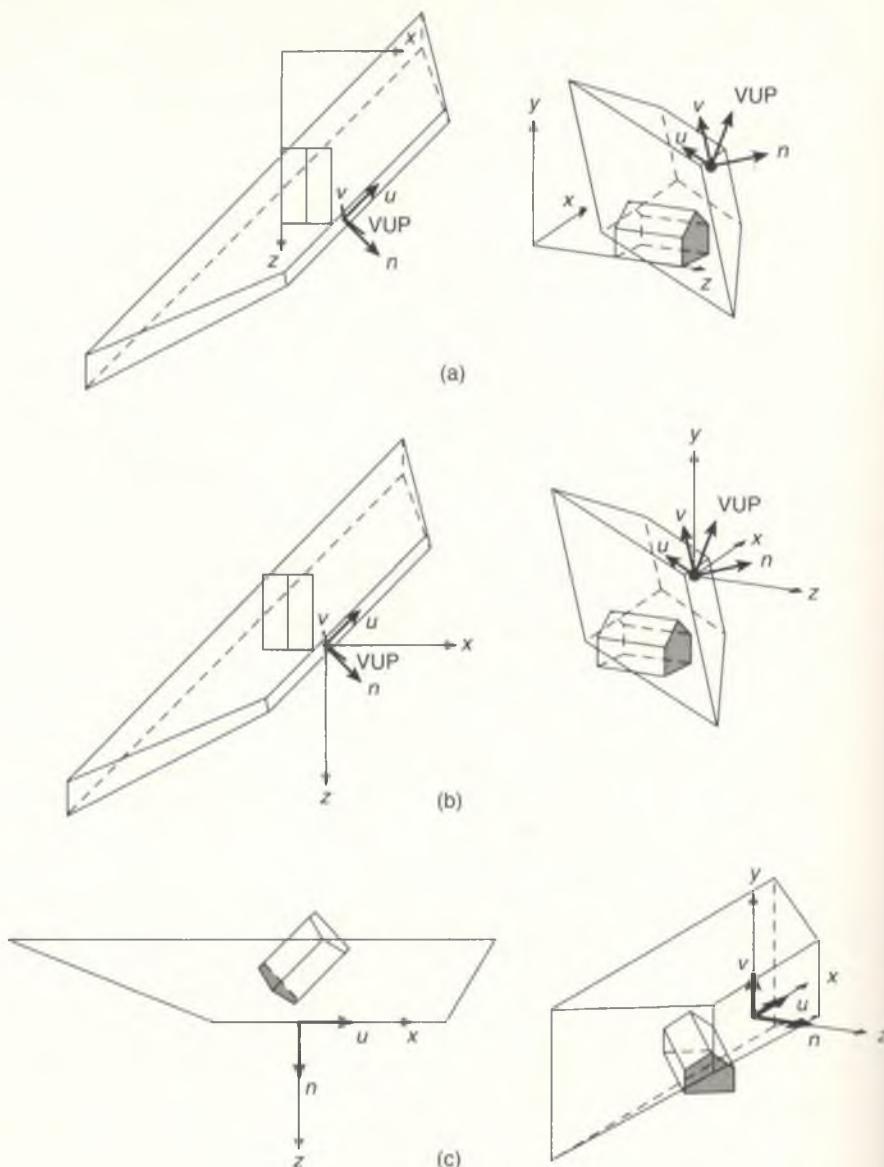
En forma similar, recuerde la transformación de normalización de visualización que convierte el volumen de vista de la proyección paralela al volumen de vista canónico correspondiente:

$$N_{\text{par}} = S_{\text{par}} \cdot T_{\text{par}} \cdot SH_{\text{par}} \cdot R \cdot T(-\text{VRP}). \quad (6.26)$$

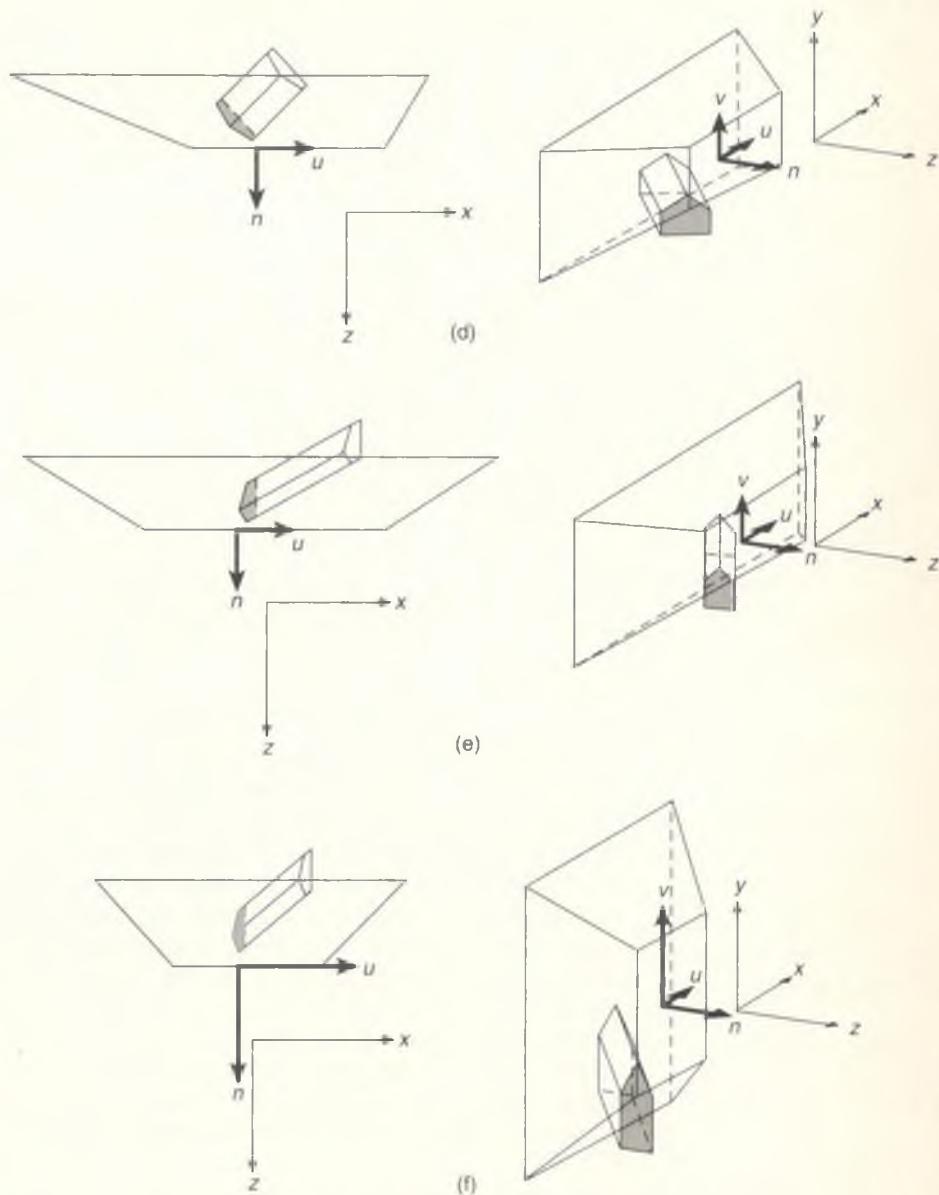
Estas transformaciones ocurren en el espacio homogéneo. ¿En qué condiciones podemos regresar al espacio tridimensional para el recorte? La respuesta es: mientras se cumpla que  $W > 0$ . Esta condición es fácil de comprender. Una  $W$  negativa implica que al dividir entre  $W$ , los signos de  $Z$  y  $z$  serán opuestos. Los puntos con  $Z$  negativa tendrán  $z$  positiva y podrían presentarse incluso si tuvieran que recortarse.

¿Cuándo podemos estar seguros de que  $W > 0$ ? Las rotaciones, las traslaciones, los escalamientos y los sesgos (según lo definido en el capítulo 5) aplicados a puntos, líneas y planos, mantendrán  $W > 0$ ; de hecho,  $W = 1$ . Por lo tanto, ni  $N_{\text{per}}$  ni  $N_{\text{par}}$  afectarán la coordenada homogénea de los puntos transformados, y por lo general no será necesaria la división entre  $W$  para establecer la correspon-

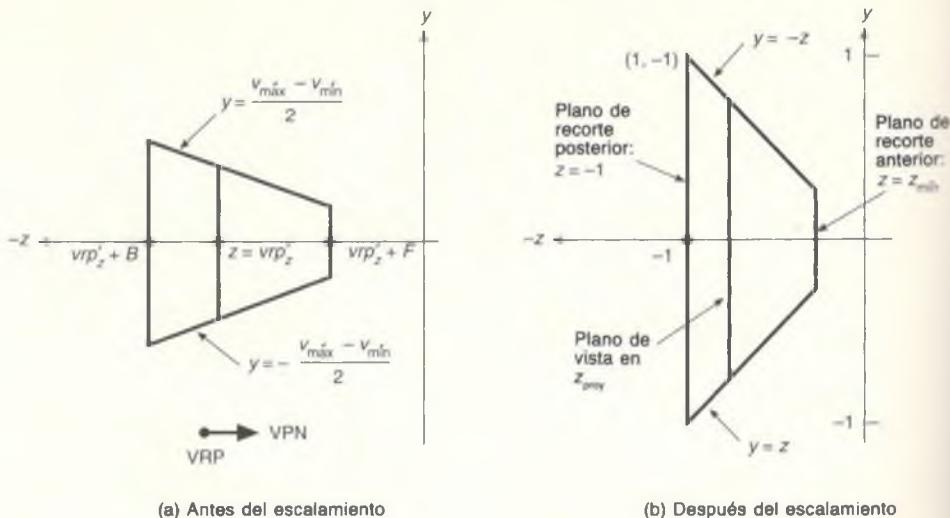
<sup>1</sup>  $z_{\min}$  y  $z_{\max}$  se nombran con base en su relación de valor absoluto, ya que  $z_{\min}$  es algebraicamente mayor que  $z_{\max}$ .



**Figura 6.41** Resultados en diversas etapas de la secuencia de vista de proyección de perspectiva. En todos los casos se presenta una proyección paralela superior y fuera de eje. (a) Se presenta la situación original de vista. (b) VRP se ha trasladado al origen. (c) El sistema de coordenadas  $(u, v, n)$  se ha hecho rotar para que quede alineado con el sistema  $(x, y, z)$ . (d) El centro de proyección (COP) se ha trasladado al origen. (e) El volumen de vista se ha sesgado de manera que la dirección de proyección (DOP) sea paralela al eje  $z$ . (f) El volumen de vista se ha escalado al volumen de vista canónico de la proyección de perspectiva. Los



parámetros de vista son  $\text{VRP} = (1.0, 1.275, 2.6)$ ,  $\text{VPN} = (1.0, 0.253, 1.0)$ ,  $\text{VUP} = (0.414, 1.0, 0.253)$ ,  $\text{PRP} = (1.6, 0.0, 1.075)$ ,  $\text{ventana} = (-1.325, 2.25, -0.575, 0.575)$ ,  $F = 0$ ,  $B = -1.2$ . (Figuras creadas con un programa escrito por L. Lu, The George Washington University.)



**Figura 6.42** Sección transversal del volumen de vista (a) antes del escalamiento y (b) después del escalamiento. En este ejemplo,  $F$  y  $B$  tienen signos opuestos, de manera que los planos anterior y posterior se encuentran en lados opuestos de VRP.

dencia de regreso del espacio tridimensional y puede efectuarse el recorte con respecto al volumen de vista canónico apropiado. Después del recorte con respecto al volumen de vista de proyección de perspectiva, es necesario aplicar la matriz de proyección de perspectiva  $M_{per}$ , que comprende la división.

Es posible que  $W < 0$  si las primitivas de salida incluyen curvas y superficies representadas como funciones en coordenadas homogéneas y mostradas como segmentos de líneas rectas conectadas. Por ejemplo, si el signo de la función para  $W$  cambia de un punto de la curva al siguiente sin que cambie el signo de  $X$ ,  $X/W$  tendrá signos diferentes en los dos puntos de la curva. Las B-splines racionales que se analizan en el capítulo 9 son un ejemplo de este comportamiento. Una  $W$  negativa también puede surgir de la utilización de algunas transformaciones distintas de las que vimos en el capítulo 5, como sería el caso de las sombras “falsas” [BLIN88]. En la sección siguiente se analizan varios algoritmos para recortes tridimensionales. Después, en la sección 6.6.4, se analiza cómo recortar cuando no es posible asegurar que  $W > 0$ .

### **6.6.3 Recortes con respecto a un volumen de vista canónico en tres dimensiones**

Los volúmenes de vista canónicos son un prisma de  $2 \times 2 \times 1$  en las proyecciones paralelas y la pirámide regular truncada derecha para proyecciones de perspectiva. Los algoritmos de recorte de Cohen-Sutherland y Cyrus-Beck que vimos en el capítulo 3 se pueden extender fácilmente a tres dimensiones.

La extensión del algoritmo bidimensional de Cohen-Sutherland para el volumen de vista canónico paralelo usa un código de región de seis bits; un bit es verdadero(1) si se satisface la condición apropiada:

bit 1 — punto encima del volumen de vista	$y > 1$
bit 2 — punto debajo del volumen de vista	$y < -1$
bit 3 — punto a la derecha del volumen de vista	$x > 1$
bit 4 — punto a la izquierda del volumen de vista	$x < -1$
bit 5 — punto detrás del volumen de vista	$z < -1$
bit 6 — punto enfrente del volumen de vista	$z > 0$

Como en dos dimensiones, se acepta trivialmente una línea si los dos puntos extremos tienen código de región de ceros y se rechaza trivialmente si la operación **and** lógica a nivel de bit de los códigos es distinta de sólo ceros. En caso contrario comienza el proceso de subdivisión de la línea. Es posible que se tengan que calcular hasta seis intersecciones, una para cada lado del volumen de vista.

Los cálculos de una intersección usan la representación paramétrica de una línea de  $P_0(x_0, y_0, z_0)$  a  $P_1(x_1, y_1, z_1)$

$$x = x_0 + t(x_1 - x_0), \quad (6.32)$$

$$y = y_0 + t(y_1 - y_0), \quad (6.33)$$

$$z = z_0 + t(z_1 - z_0) \quad 0 \leq t \leq 1. \quad (6.34)$$

Conforme  $t$  varía de 0 a 1, las tres ecuaciones proporcionan las coordenadas de todos los puntos de la línea, de  $P_0$  a  $P_1$ .

Para calcular la intersección de una línea con el plano  $y = 1$  del volumen de vista, reemplazamos la variable  $y$  de la ecuación (6.33) con 1 y resolvemos para hallar que  $t = (1 - y_0)/(y_1 - y_0)$ . Si  $t$  está fuera del intervalo de 0 a 1, la intersección se encuentra en la línea infinita que pasa por los puntos  $P_0$  y  $P_1$  pero no está en la porción de la línea entre  $P_0$  y  $P_1$ , de manera que no nos interesa. Si  $t$  se halla en  $[0, 1]$ , su valor se sustituye en las ecuaciones de  $x$  y  $z$  para encontrar las coordenadas de la intersección:

$$x = x_0 + \frac{(1 - y_0)(x_1 - x_0)}{y_1 - y_0}, \quad z = z_0 + \frac{(1 + y_0)(z_1 - z_0)}{y_1 - y_0}. \quad (6.35)$$

El algoritmo usa códigos de región que hacen innecesaria la prueba de  $t$  en  $[0, 1]$ .

Los bits de código de región para recortes con respecto al volumen de vista canónico de perspectiva son como sigue:

bit 1 — punto encima del volumen de vista	$y > -z$
bit 2 — punto debajo del volumen de vista	$y < z$
bit 3 — punto a la derecha del volumen de vista	$x > -z$

bit 4 — punto a la izquierda del volumen de vista	$x < z$
bit 5 — punto detrás del volumen de vista	$z < -1$
bit 6 — punto enfrente del volumen de vista	$z > z_{\min}$

El cálculo de las intersecciones de líneas con los planos inclinados es bastante sencillo. En el plano  $y = z$ , donde la ecuación (6.33) debe ser igual a la ecuación (6.34),  $y_0 + t(y_1 - y_0) = z_0 + t(z_1 - z_0)$ . Entonces,

$$t = \frac{z_0 - y_0}{(y_1 - y_0) - (z_1 - z_0)}. \quad (6.36)$$

Al sustituir  $t$  en las ecuaciones (6.32) y (6.33) para  $x$  y  $y$  se obtiene

$$x = x_0 + \frac{(x_1 - x_0)(z_0 - y_0)}{(y_1 - y_0) - (z_1 - z_0)}, \quad y = y_0 + \frac{(y_1 - y_0)(z_0 - y_0)}{(y_1 - y_0) - (z_1 - z_0)}. \quad (6.37)$$

Sabemos que  $z = y$ . Ahora queda claro por qué se escogió este volumen de vista canónico: las pendientes unitarias de los planos hacen más sencillo el cálculo de intersecciones que otras pendientes arbitrarias.

Hay otros algoritmos de recorte [CYRU78; LIAN84] basados en expresiones paramétricas para las líneas, los cuales pueden ser más eficientes que el sencillo algoritmo de Cohen-Sutherland. Véanse los capítulos 6 y 19 de [FOLE90].

#### 6.6.4 Recortes en coordenadas homogéneas

Hay dos razones para efectuar recortes en coordenadas homogéneas. La primera tiene que ver con la eficiencia: es posible transformar el volumen de vista canónico de proyección de perspectiva a un volumen de vista canónico de proyección paralela, de manera que siempre es posible usar un solo procedimiento de recorte sencillo, optimizado para el volumen de vista canónico de proyección paralela. Sin embargo, el recorte debe hacerse en coordenadas homogéneas para garantizar la corrección de los resultados. Este tipo de procedimiento de recorte suele proporcionarse en las implantaciones de hardware de la operación de vista. La segunda razón es que hay puntos que surgen como resultado de transformaciones homogéneas poco usuales y de la utilización de *splines* paramétricas racionales (Cap. 9) que pueden tener  $W$  negativa y por lo tanto pueden recortarse correctamente en coordenadas homogéneas pero no en tres dimensiones.

En lo que se refiere a los recortes, puede demostrarse que la transformación del volumen de vista canónico de proyección de perspectiva al volumen de vista canónico de proyección paralela es

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1+z_{\min}} & \frac{-z_{\min}}{1+z_{\min}} \\ 0 & 0 & -1 & 0 \end{bmatrix}, \quad z_{\min} \neq -1. \quad (6.38)$$

Recuerde, de la ecuación (6.30), que  $z_{\min} = -(vRP_z' + F)/(vRP_z' + B)$ , y de la ecuación (6.28) que  $VRP' = SH_{\text{par}} \cdot T(-PRP) \cdot [0 \ 0 \ 0 \ 1]^T$ . En la figura 6.43 se presenta el resultado de la aplicación de  $M$  al volumen de vista canónico de proyección de perspectiva.

La matriz  $M$  se integra con la transformación de normalización  $N_{\text{per}}$  de la proyección de perspectiva:

$$N'_{\text{per}} = M \cdot N_{\text{per}} = M \cdot S_{\text{per}} \cdot SH_{\text{par}} \cdot T(-PRP) \cdot R \cdot T(-VRP). \quad (6.39)$$

Al usar  $N'_{\text{per}}$  en lugar de  $N_{\text{per}}$  para las proyecciones de perspectiva y seguir usando  $N_{\text{par}}$  para las proyecciones paralelas, podemos recortar con respecto al volumen de vista canónico de proyección paralela en lugar de hacerlo con el volumen de vista canónico de proyección de perspectiva.

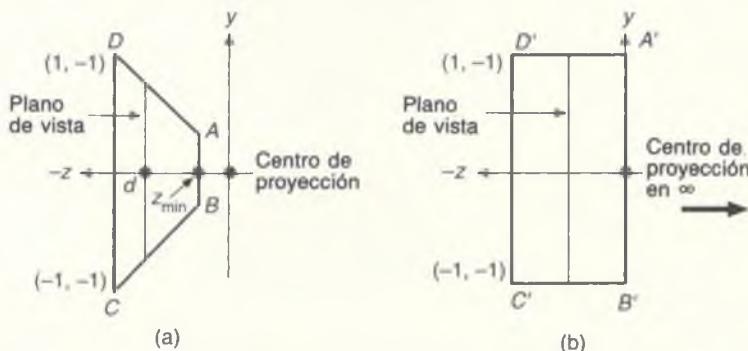
El volumen de vista de proyección paralela tridimensional está definido por  $-1 \leq x \leq 1, -1 \leq y \leq 1, -1 \leq z \leq 0$ . Para hallar las desigualdades correspondientes en coordenadas homogéneas se reemplaza  $x$  por  $X/W$ ,  $y$  por  $Y/W$  y  $z$  por  $Z/W$ , para obtener

$$-1 \leq X/W \leq 1, \quad -1 \leq Y/W \leq 1, \quad -1 \leq Z/W \leq 1. \quad (6.40)$$

Las ecuaciones de plano correspondientes son

$$X = -W, \quad X = W, \quad Y = -W, \quad Y = W, \quad Z = -W, \quad Z = 0. \quad (6.41)$$

Para comprender la forma de usar estos límites y planos, hay que considerar por separado los casos de  $W > 0$  y  $W < 0$ . En el primer caso, las desigualdades de la ecuación (6.40) se pueden multiplicar por  $W$  sin alterar su sentido.



**Figura 6.43** Vistas laterales del volumen de vista de perspectiva normalizada antes (a) y después (b) de la aplicación de la matriz  $M$ .

En el segundo caso, la multiplicación cambia el sentido. Este resultado se puede expresar como

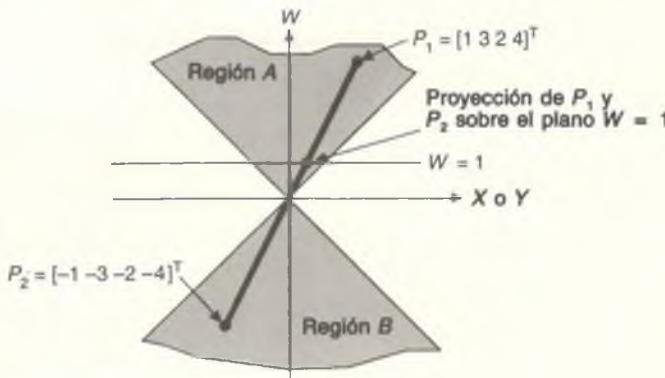
$$W > 0: -W \leq X \leq W, -W \leq Y \leq W, -W \leq Z \leq 0, \quad (6.42)$$

$$W < 0: -W \geq X \geq W, -W \geq Y \geq W, -W \geq Z \geq 0. \quad (6.43)$$

En nuestro caso (recorte de líneas ordinarias y puntos) sólo hay que usar la región indicada por la ecuación (6.42), ya que antes de la aplicación de  $M$  todos los puntos visibles tienen  $W > 0$  (normalmente,  $W = 1$ ).

Sin embargo, como veremos en el capítulo 9, en ocasiones es conveniente representar los puntos directamente en coordenadas homogéneas con coordenadas  $W$  arbitrarias. Por lo tanto, podemos tener  $W < 0$ , lo cual significa que el recorte se debe efectuar con respecto a las regiones indicadas por las ecuaciones (6.42) y (6.43). En la figura 6.44 se muestran estas regiones como  $A$  y  $B$ , y también se ilustra por qué hay que emplear ambas regiones.

El punto  $P_1 = [1 \ 3 \ 2 \ 4]^T$  en la región  $A$  se transforma al punto tridimensional  $(\frac{1}{4}, \frac{3}{4}, \frac{2}{4})$ , que se encuentra en el volumen de vista canónico  $-1 \leq x \leq 1, -1 \leq y \leq 1, -1 \leq z \leq 0$ . El punto  $P_2 = -P_1 = [-1 \ -3 \ -2 \ -4]^T$ , que no está en la región  $A$  pero sí en la región  $B$ , se transforma al mismo punto tridimensional que  $P_1$ , específicamente, a  $(\frac{1}{4}, \frac{3}{4}, \frac{2}{4})$ . Si el recorte sólo se aplicara a la región  $A$ ,  $P_2$  se descartaría de manera incorrecta. Esta posibilidad se presenta porque los puntos  $P_1$  y  $P_2$  en coordenadas homogéneas difieren por un multiplicador constante  $(-1)$ , y sabemos que estos puntos homogéneos corresponden al mismo punto tridimensional (en el plano  $W = 1$  de espacio homogéneo).

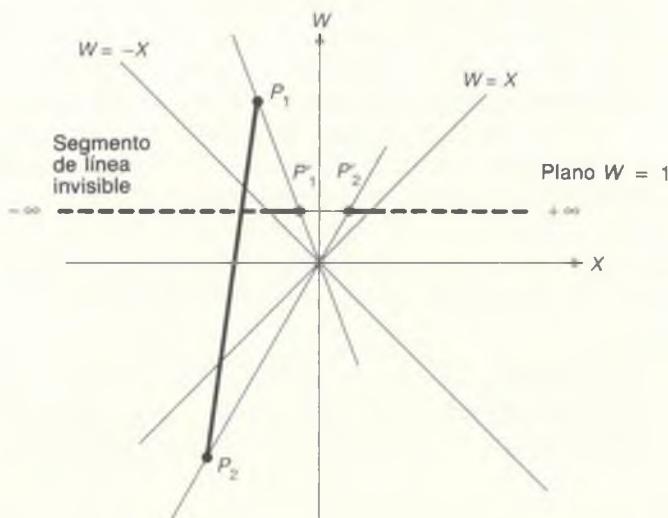


**Figura 6.44** Los puntos  $P_1$  y  $P_2$  corresponden al mismo punto en el plano  $W = 1$ , al igual que los demás puntos en la línea que pasa por el origen y los dos puntos. Los recortes en coordenadas homogéneas exclusivamente con respecto a la región  $A$  rechazarán de manera incorrecta el punto  $P_2$ .

Existen dos soluciones para este problema de los puntos en la región  $B$ . Una es recortar dos veces todos los puntos, una vez con respecto a cada región. Sin embargo, la duplicación de recortes es costosa. Una mejor solución es cambiarle el signo a los puntos, como  $P_2$ , con una  $W$  negativa, y después recortarlos. Así mismo, podemos recortar de manera correcta una línea cuyos puntos extremos se encuentran en la región  $B$  de la figura 6.44 si multiplicamos los puntos extremos por  $-1$  para colocarlos en la región  $A$ .

Otro problema se presenta con las líneas como  $P_1P_2$ , presentada en la figura 6.45, cuyos puntos extremos tienen valores opuestos de  $W$ . La proyección de la línea sobre el plano  $W = 1$  es de dos segmentos, uno de los cuales va a  $+\infty$  y el otro a  $-\infty$ . En este caso la solución es recortar dos veces, una con respecto a cada región, con la posibilidad de que cada recorte devuelva un segmento de línea visible. Una manera sencilla de hacerlo es recortando la línea con respecto a la región  $A$  para cambiarle el signo a los dos puntos extremos de la línea, y luego recortar de nuevo con respecto a la región  $A$ . Este método conserva una de las razones originales para hacer los recortes en las coordenadas homogéneas: usar una sola región de recorte. Los lectores que estén interesados en conocer más detalles pueden consultar [BLIN78a].

Dada la ecuación (6.41), es posible utilizar el algoritmo de Cohen-Sutherland o de Cyrus-Beck para efectuar los recortes. En [LIAN84] se presenta el código para el método de Cyrus-Beck. La única diferencia es que el recorte es en cuatro dimensiones, no en tres.



**Figura 6.45**

La línea  $P_1P_2$  se proyecta como dos segmentos de línea, uno de  $P'_1$  a  $+\infty$  y el otro de  $P'_1$  a  $-\infty$  (se presentan como líneas sólidas gruesas cuando están dentro de la región de recorte y como líneas discontinuas cuando se encuentran fuera de esta región). La línea debe recortarse dos veces, una para cada región.

### 6.6.5 Correspondencia a un área de vista

Las primitivas de salida se recortan en el sistema de coordenadas de la proyección normalizada, también conocido como sistema de coordenadas tridimensional de pantalla. Para nuestro análisis supondremos que se ha usado el volumen de vista canónico de proyección paralela para el recorte (la proyección de perspectiva  $M$  transforma el volumen de vista de proyección de perspectiva al volumen de vista de proyección paralela si esta suposición es incorrecta). Por consiguiente, las coordenadas de todas las primitivas de salida que permanecen se encuentran en el volumen de vista  $-1 \leq x \leq 1, -1 \leq y \leq 1, -1 \leq z \leq 0$ .

El programador de PHIGS especifica un área de vista tridimensional con la cual se establecerá la correspondencia del contenido del volumen de vista. El área de vista tridimensional está contenida en el cubo unidad  $0 \leq x \leq 1, 0 \leq y \leq 1, 0 \leq z \leq 1$ . La cara frontal  $z = 1$  del cubo unidad corresponde al mayor cuadrado que puede inscribirse en la pantalla. Suponemos que la esquina inferior izquierda del cuadrado está en  $(0, 0)$ . Por ejemplo, en una pantalla con resolución horizontal de 1024 y resolución vertical de 800, el cuadrado consiste en los pixeles  $P$  en las localidades  $(P_x, P_y)$  con  $0 \leq P_x \leq 799, 0 \leq P_y \leq 799$ . Presentamos los puntos en el cubo unidad descartando su coordenada  $z$ . De esta manera, el punto  $(0.5, 0.75, 0.46)$  se presentaría en las coordenadas  $(400, 599)$  de la pantalla. En el caso de la determinación de superficies visibles (Cap. 13), la coordenada  $z$  de cada primitiva de salida se emplea para determinar cuáles son visibles y cuales están ocultas por otras primitivas con  $z$  mayor.

Dada un área de vista tridimensional dentro del cubo unidad, definida por las ecuaciones  $x_{v,\min} \leq x \leq x_{v,\max}$ , etc., la correspondencia entre el volumen de vista canónico de proyección paralela y el área de vista tridimensional se puede considerar como un proceso de tres pasos. En el primero, el volumen de vista canónico de proyección paralela se traslada de manera que su vértice  $(-1, -1, -1)$  se convierta en el origen. Esta acción es llevada a cabo por la traslación  $T(1, 1, 1)$ . Después se escala el volumen de vista trasladado al tamaño del área de vista tridimensional, con el escalamiento

$$S\left(\frac{x_{v,\max} - x_{v,\min}}{2}, \frac{y_{v,\max} - y_{v,\min}}{2}, \frac{z_{v,\max} - z_{v,\min}}{1}\right).$$

Finalmente, el volumen de vista escalado se traslada al vértice inferior izquierdo del área de vista con la traslación  $T(x_{v,\min}, y_{v,\min}, z_{v,\min})$ . Así, el volumen de vista canónico compuesto para la transformación de área de vista tridimensional es

$$M_{VV3DV} = T(x_{v,\min}, y_{v,\min}, z_{v,\min}) \cdot S\left(\frac{x_{v,\max} - x_{v,\min}}{2}, \frac{y_{v,\max} - y_{v,\min}}{2}, \frac{z_{v,\max} - z_{v,\min}}{1}\right) \cdot T(1, 1, 1). \quad (6.44)$$

Observe que esta transformación es similar, pero no igual, a la transformación ventana-área de vista  $M_{WV}$  que se desarrolló en la sección 5.5.

## 6.6.6 Resumen de implantación

Hay dos implantaciones de uso general para la transformación de visualización global. La primera, ilustrada en la figura 6.34 y analizada en las secciones 6.6.1 a 6.6.3, es apropiada cuando las primitivas de salida se definen en tres dimensiones y las transformaciones aplicadas a las primitivas de salida crean una  $W$  negativa. Sus pasos son los siguientes:

1. Extender las coordenadas tridimensionales a coordenadas homogéneas.
2. Aplicar la transformación de normalización  $N_{\text{par}}$  o  $N'_{\text{per}}$ .
3. Dividir entre  $W$  para establecer la correspondencia de nuevo a tres dimensiones (en algunos casos se sabe que  $W = 1$  y no es necesaria la división).
4. Recortar en tres dimensiones con respecto al volumen de vista canónico de proyección paralela o de perspectiva, según resulte apropiado.
5. Extender las coordenadas tridimensionales a coordenadas homogéneas.
6. Llevar a cabo la proyección paralela usando  $M_{\text{ori}}$  [Ec. (6.11)] o efectuar la proyección de perspectiva usando  $M_{\text{per}}$  [Ec. (6.3)], con  $d = -1$ .
7. Trasladar y escalar a coordenadas de dispositivo usando la ecuación (6.44).
8. Dividir entre  $W$  para establecer la correspondencia entre coordenadas homogéneas y bidimensionales; la división lleva a cabo la proyección de perspectiva.

Los pasos 6 y 7 se realizan con una multiplicación de matrices y corresponden a las etapas 3 y 4 de la figura 6.34.

La segunda forma de implantar la operación de visualización es necesaria cuando las primitivas de salida se definen en coordenadas homogéneas y pueden tener  $W < 0$ , cuando las transformaciones aplicadas a las primitivas de salida pueden crear una  $W$  negativa, o cuando se implanta un solo algoritmo de recorte. Como vimos en la sección 6.6.4, sus pasos son los siguientes:

1. Extender las coordenadas tridimensionales a coordenadas homogéneas.
2. Aplicar la transformación de normalización  $N_{\text{par}}$  o  $N'_{\text{per}}$  [que incluye  $M$ , Ec. (6.38)].
3. Si  $W > 0$ , recortar en coordenadas homogéneas con respecto al volumen definido por la ecuación (6.42); en caso contrario, recortar en coordenadas homogéneas con respecto a los dos volúmenes de vista definidos por las ecuaciones (6.42) y (6.43).
4. Trasladar y escalar a coordenadas de dispositivos usando la ecuación (6.44).

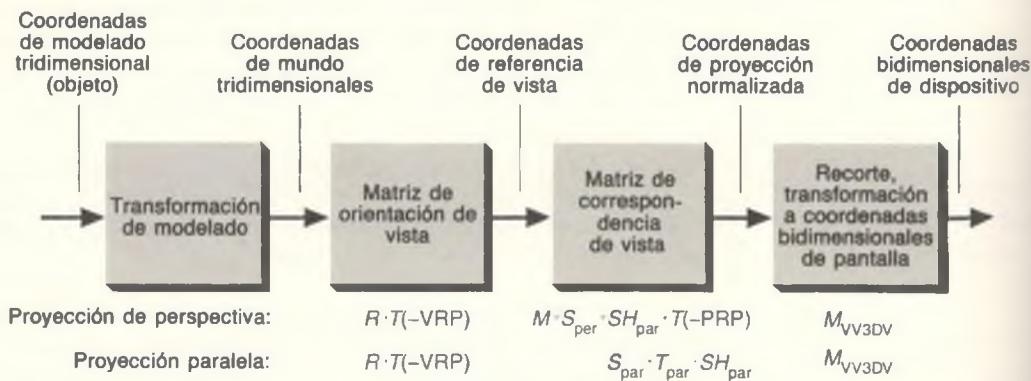
5. Dividir entre  $W$  para establecer la correspondencia entre las coordenadas homogéneas y las coordenadas bidimensionales; con la división se realiza la proyección de perspectiva.

## 6.7 Sistemas de coordenadas

En los capítulos 5 y 6 se han usado diferentes sistemas de coordenadas. En esta sección se resumen todos los sistemas y se analiza la relación que existe entre ellos. Así mismo, se presentan los sinónimos utilizados en varias obras de referencia y en paquetes de subrutinas gráficas. En la figura 6.46 se muestra la secuencia de sistemas de coordenadas, usando los términos que generalmente aparecen en este libro; en un paquete de subrutinas gráficas sólo se usan algunos de los sistemas de coordenadas. Para los diversos sistemas de coordenadas, hemos elegido nombres que reflejen el uso común; por lo tanto, algunos de los nombres no son consistentes con otros. Observe que el término *espacio* en ocasiones se usa como sinónimo de *sistema*.

Comenzando por el sistema de coordenadas que está más lejos del dispositivo de presentación, a la izquierda en la figura 6.46, los objetos individuales se definen con un **sistema de coordenadas de objeto**. En PHIGS esto se conoce como **sistema de coordenadas de modelado**, y también es común el término **sistema de coordenadas locales**. Como veremos en el capítulo 7, con frecuencia existe una jerarquía de sistemas de coordenadas de modelado.

Los objetos se transforman a un **sistema de coordenadas de mundo**, el sistema en el cual se representa en el computador una escena o un objeto completo; esto se hace por medio de la **transformación de modelado**. Este sistema también



**Figura 6.46** Sistemas de coordenadas y su relación. Las matrices debajo de cada etapa llevan a cabo la transformación aplicada en la etapa para las proyecciones de perspectiva y paralela.

se conoce como **sistema de coordenadas del problema o sistema de coordenadas de la aplicación**.

El **sistema de coordenadas de referencia de vista** se usa en PHIGS como sistema de coordenadas para definir un volumen de vista. También se le llama sistema ( $u, v, n$ ) o sistema ( $u, v, \text{VPN}$ ). El sistema Core [GSPC79] usaba un sistema de mano izquierda similar, aunque sin nombre. El sistema de mano izquierda se emplea para que, cuando el ojo o la cámara en el origen ve hacia  $+z$ , los valores mayores de  $z$  están más lejos del ojo,  $x$  está a la derecha y  $y$  está hacia arriba.

Otros paquetes, como RenderMan de Pixar [PIXA88], aplican restricciones al sistema de coordenadas de referencia de vista, requiriendo que el origen se encuentre en el centro de proyección y que el plano de vista sea normal al eje  $z$ . A esto lo llamamos **sistema de coordenadas oculares**. RenderMan y otros sistemas emplean el término **sistema de coordenadas de cámara**. Haciendo referencia a la sección 6.6, los tres primeros pasos de la transformación de normalización de proyección de perspectiva hacen la conversión del sistema de coordenadas de mundo al sistema de coordenadas oculares. El sistema de coordenadas oculares en ocasiones es de mano izquierda.

A partir de las coordenadas oculares, pasamos al **sistema de coordenadas de proyección normalizada o coordenadas tridimensionales de pantalla**, el sistema de coordenadas del volumen de vista canónico de proyección paralela (y del volumen de vista canónico de proyección de perspectiva después de la transformación de perspectiva). El sistema Core llama a este sistema **coordenadas tridimensionales de dispositivo normalizadas**. En ocasiones el sistema se conoce como **coordenadas tridimensionales de dispositivo lógico**. El término *normalizado* generalmente quiere decir que todos los valores de coordenadas se encuentran en el intervalo  $[0, 1]$  o  $[-1, 1]$ , mientras que el término *lógico* usualmente indica que los valores de coordenadas se hallan en otro intervalo previamente especificado, como  $[0, 1023]$ , que por lo general se define de manera que corresponda al sistema de coordenadas de un dispositivo de uso común. En algunos casos este sistema no está normalizado.

La proyección de tres dimensiones a dos crea lo que se conoce como **sistema de coordenadas bidimensionales de dispositivo**, también llamado **sistema de coordenadas de dispositivo normalizadas**, **sistema de coordenadas de imagen** en [SUTH74a] o **sistema de coordenadas de pantalla** en RenderMan. Otros términos utilizados son **coordenadas de pantalla**, **coordenadas de dispositivo**, **coordenadas bidimensionales de dispositivo** y **coordenadas de dispositivo físico** (a diferencia de la coordenada de dispositivo lógico que mencionamos antes). En RenderMan, la forma física se denomina **coordenadas espaciales de trama**.

Por desgracia, no hay una utilización estándar de muchos de estos términos. Por ejemplo, el término **sistema de coordenadas de pantalla** es usado por distintos autores para referirse a los últimos tres sistemas que analizamos, que abarcan tanto coordenadas bidimensionales como tridimensionales, así como coordenadas lógicas y físicas.

**Ejercicios**

6.1 Escriba un programa que acepte una especificación de vista, calcule  $N_{\text{par}}$  o  $N_{\text{per}}$  y presente la casa cuyas coordenadas se definen en la figura 6.18.

6.2 Implante algoritmos de recorte tridimensional para las proyecciones paralela y de perspectiva.

6.3 Muestre que, para una proyección paralela con  $F = -\infty$  y  $B = +\infty$ , el resultado del recorte tridimensional y la proyección a dos dimensiones es igual que el resultado de la proyección a dos dimensiones y el recorte bidimensional.

6.4 Muestre que, si todos los objetos se encuentran frente al centro de proyección y que si  $F = -\infty$  y  $B = +\infty$ , el resultado del recorte en tres dimensiones con respecto al volumen de vista canónico de proyección de perspectiva, seguido por la proyección de perspectiva, es igual que efectuar primero la proyección de perspectiva a dos dimensiones y luego hacer un recorte bidimensional.

6.5 Verifique que  $S_{\text{per}}$  (Sec. 6.6.2) transforme el volumen de vista de la figura 6.42(a) al de la figura 6.42(b).

6.6 Escriba el código para el recorte tridimensional con respecto al cubo unidad. Generalice el código para recortes con respecto a cualquier sólido rectangular con caras normales a los ejes principales. ¿Es el código generalizado más o menos eficiente que el del caso del cubo unidad? Explique su respuesta.

6.7 Escriba el código para el recorte tridimensional con respecto al volumen de vista canónico de proyección de perspectiva. Ahora generalice al volumen de vista definido por

$$-a \cdot z_v \leq x_v \leq b \cdot z_v, \quad -c \cdot z_v \leq y_v \leq d \cdot z_v, \quad z_{\min} \leq z_v \leq z_{\max}.$$

Estas relaciones representan la forma general del volumen de vista después de los pasos 1 a 4 de la transformación de normalización de perspectiva. ¿Cuál caso es más eficiente? Explique su respuesta.

6.8 Escriba el código para el recorte tridimensional con respecto a un volumen de vista poliédrico general con seis caras definidas por

$$A_i x + B_i y + C_i z + D_i = 0, \quad 1 \leq i \leq 6.$$

Compare el esfuerzo computacional requerido con el de los siguientes casos:

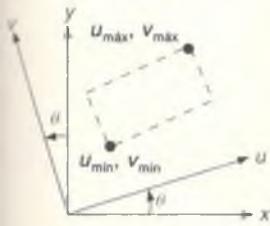
- Recortes con respecto a alguno de los volúmenes de vista canónicos.
- Aplicar  $N_{\text{par}}$  y luego recortar con respecto al cubo unidad.

6.9 Considere una línea en tres dimensiones entre los puntos de coordenadas mundiales  $P_1(6, 10, 3)$  y  $P_2(-3, -5, 2)$  y una pirámide de visualización semi-infinita en la región  $-z \leq x \leq z, -z \leq y \leq z$ , acotado por los planos  $z = +x$ ,  $z = -x$ ,  $z = +y$ ,  $z = -y$ . El plano de proyección se encuentra en  $z = 1$ .

- Recorte la línea en tres dimensiones (usando las ecuaciones paramétricas de líneas) y proyectela al plano. ¿Cuáles son los puntos extremos recortados en el plano?
- Proyecte la línea sobre el plano y recorte las líneas con cálculos bidimensionales. ¿Cuáles son los puntos extremos recortados en el plano?

(Sugerencia: Si sus respuestas en las partes (a) y (b) no son idénticas, ¡intente de nuevo!)

6.10 Muestre lo que sucede si un objeto *detrás* del centro de proyección es proyectado por  $M_{\text{per}}$  y luego recortado. Su respuesta debe demostrar por qué, en términos generales, no se puede proyectar y luego recortar.



**Figura 6.47**  
Ventana rotada.

6.11 Considere la operación de vista bidimensional con una ventana rotada. Elabore una transformación normalizada para convertir la ventana al cuadrado unidad. La ventana está especificada por  $u_{\text{min}}, v_{\text{min}}, u_{\text{máx}}, v_{\text{máx}}$  en el sistema de coordenadas VRC, como en la figura 6.47. Muestre que esta transformación es la misma que para el caso general de  $N'_{\text{par}}$  tridimensional cuando el plano de proyección es el plano  $(x, y)$  y VUP tiene un componente  $x$  de  $-\sin \theta$  y un componente  $y$  de  $\cos \theta$  (es decir, la proyección paralela de VUP sobre el plano de vista es el eje  $v$ ).

6.12 ¿Cuál es el efecto de aplicar  $M_{\text{per}}$  a puntos cuya coordenada  $z$  sea menor que cero?

6.13 Diseñe e implante un conjunto de subrutinas de utilería para generar una matriz de transformación de  $4 \times 4$  a partir de una secuencia arbitraria de transformaciones primitivas  $R$ ,  $S$  y  $T$ .

6.14 Dibuje un árbol de decisión que pueda usar cuando determine el tipo de proyección que empleará al crear una imagen. Aplique este árbol de decisión a las figuras de este capítulo que sean proyecciones de tres dimensiones.

6.15 El volumen de vista canónico para la proyección paralela se consideró como un paralelepípedo rectangular de  $2 \times 2 \times 1$ . Suponga que se usa en cambio el cubo unidad en el octante positivo, con un vértice en el origen.

- Encuentre la normalización  $N'_{\text{par}}$  para este volumen de vista.
- Encuentre el volumen de vista correspondiente de coordenadas homogéneas.

6.16 Proporcione los parámetros de visualización para las vistas frontal, superior y lateral de la casa de la figura 6.18 con el VRP en medio de la ventana. ¿Debe variar PRP para cada una de las vistas? Explique su respuesta.

6.17 Los pares estéreo son dos vistas de la misma escena desde puntos de referencia de proyección ligeramente distintos, pero con el mismo VRP. Sea  $d$  la separación estéreo, es decir, la distancia entre los dos puntos de referencia. Si pensamos en estos puntos de referencia como nuestros ojos, entonces  $d$  es la distancia entre los ojos. Sea  $P$  el punto intermedio entre los ojos. Con base en  $P$ ,  $d$ , VRP, VPN y VUP, obtenga expresiones para los dos puntos de referencia de proyección.

Un paquete gráfico es un intermediario entre un programa de aplicación y el hardware gráfico. Las primitivas de salida y los dispositivos de interacción que apoya un paquete gráfico pueden ir desde los más rudimentarios hasta los más complejos. En el capítulo 2 describimos el paquete SRGP, bastante sencillo y de bajo nivel, e indicamos algunas de sus limitaciones. En este capítulo describiremos un paquete basado en un paquete gráfico estándar mucho más completo pero a la vez más complicado, **PHIGS** (*Programmer's Hierarchical Interactive Graphics System*, Paquete gráfico interactivo jerárquico para programadores<sup>1</sup>). Un **paquete gráfico estándar** como PHIGS o GKS (*Graphical Kernel System*, Sistema de kernel gráfico) implanta una especificación diseñada como estándar por un organismo normativo oficial a nivel internacional o nacional; GKS y PHIGS han sido diseñados de esta manera por ANSI (*American National Standards Institute*, Instituto Nacional Estadounidense de Estándares) y por ISO (*International Standards Organization*, Organización Internacional de Estándares). El propósito principal de estos estándares es promover la transportabilidad de los programas de aplicación y los programadores. Los estándares no oficiales de la industria, analizados con mayor detenimiento en la sección 7.11.6, también son importantes en el campo de los gráficos interactivos. Entre éstos se incluyen OpenGL, de Silicon Graphics [NEID93] y HOOPS™, de

<sup>1</sup> En este capítulo el término **PHIGS** también incluye un conjunto de extensiones a PHIGS, llamado **PHIGS PLUS**, que apoya el uso de primitivas geométricas avanzadas como poliedros, curvas y superficies, así como técnicas de generación de imágenes que utilizan iluminación y sombreado, analizadas en los capítulos 12 a 14.

Ithaca Software [BASS90]. Aunque se ha hablado mucho de las diferencias entre PHIGS PLUS, OpenGL y HOOPS, tienen más similitudes que diferencias, cuando menos en lo que se refiere a sus capacidades básicas si no es que en cuanto a su Interfaz de programa de aplicación (API, *Application Program Interface*). Casi todo lo que aprenderá en este capítulo será aplicable, con pequeñas modificaciones, a los demás paquetes, ya que todos apoyan, por ejemplo, la jerarquía de objetos con las transformaciones.

El paquete que describimos aquí se llama **SPHIGS** (PHIGS simple) porque en esencia es un subconjunto de PHIGS. Conserva la mayoría de las capacidades y el poderío de PHIGS, pero simplifica o modifica varias características para permitir su aplicación directa. SPHIGS también incluye varias mejoras adaptadas de las extensiones de PHIGS PLUS. Nuestro objetivo al diseñar SPHIGS ha sido introducir los conceptos de la manera más sencilla posible, no proporcionar un paquete que sea directamente compatible con PHIGS. Sin embargo, es fácil adaptar una aplicación SPHIG para su ejecución en PHIGS. En las notas de pie de página se presentan algunas de las diferencias más importantes entre SPHIGS y PHIGS; por lo general, una característica de SPHIGS existe en PHIGS, a menos que se especifique lo contrario.

Hay tres diferencias principales entre SPHIGS y los paquetes enteros de barrido, como SRGP o el paquete Xlib del sistema X Window. En primer lugar, para las aplicaciones de ingeniería y científicas, SPHIGS usa un sistema de coordenadas tridimensionales de punto flotante e implanta la secuencia de visualización tridimensional descrita en el capítulo 6.

La segunda diferencia, que tiene mayor repercusión, es que SPHIGS mantiene una base de datos de estructuras. Una **estructura** es un agrupamiento lógico de primitivas, atributos y otra información. El programador puede modificar las estructuras en la base de datos con unos cuantos mandatos de edición; SPHIGS asegura que la imagen en pantalla sea una representación precisa del contenido almacenado en la base de datos. Las estructuras no sólo contienen especificaciones de primitivas y atributos, sino también invocaciones de estructuras subordinadas. Así, las estructuras exhiben algunas de las propiedades de los procedimientos en los lenguajes de programación. Específicamente, así como la jerarquía de procedimientos es inducida por las llamadas a subprocedimientos, la jerarquía de estructuras se induce con estructuras que invocan subestructuras. Esta composición jerárquica es especialmente poderosa cuando se puede controlar la geometría (posición, orientación, tamaño) y la apariencia (color, estilo, grosor, etc.) de la invocación de una subestructura.

La tercera diferencia es que SPHIGS opera en un sistema abstracto de coordenadas de mundo tridimensionales, no en el espacio bidimensional de la pantalla, y por consiguiente no apoya la manipulación directa de pixeles. Debido a estas diferencias, SPHIGS y SRGP están orientados a diferentes conjuntos de necesidades y aplicaciones; como señalamos en el capítulo 2, cada uno tiene su lugar: ningún paquete gráfico satisface por sí solo todas las necesidades.

Gracias a su capacidad para apoyar la jerarquía de estructuras, SPHIGS es muy apropiado para aplicaciones basadas en modelos con jerarquía de componentes-subcomponentes; de hecho, la jerarquía de estructuras de SPHIGS se

puede ver como una sencilla jerarquía de modelado de propósito general. Por lo tanto, en la sección 7.1 veremos los aspectos generales del modelado antes de analizar los puntos específicos del modelado geométrico con SPHIGS. En las secciones 7.2 a 7.9 se muestra cómo crear, presentar y editar la base de datos de estructuras de SPHIGS. En la sección 7.10 se presenta la interacción, sobre todo la correlación de selección.

## 7.1 Modelado geométrico

En sus cursos de ciencias físicas y sociales usted ha conocido muchos ejemplos de modelos. Por ejemplo, es probable que esté familiarizado con el modelo de Bohr del átomo, donde las esferas que representan a los electrones giran en una órbita alrededor de un núcleo esférico que contiene las esferas de los neutrones y los protones. Otros ejemplos son el modelo de crecimiento exponencial ilimitado en biología o los modelos macro o microeconómicos que intentan describir algún aspecto económico. Un modelo es una representación de algunas características (no necesariamente todas) de una entidad concreta o abstracta. El propósito del modelo de una entidad es permitir que la gente visualice y comprenda la estructura o el comportamiento de la entidad y proporcionar un medio conveniente para realizar experimentos y predecir los efectos de entradas o cambios en el modelo. Los modelos cuantitativos comunes en las ciencias físicas y sociales generalmente se expresan como sistemas de ecuaciones, y el modelador experimentará con cambios en los valores de las variables independientes, los coeficientes y los exponentes. Los modelos muchas veces simplifican la estructura o el comportamiento real de la entidad modelada para que el modelo sea más fácil de visualizar o, en el caso de modelos representados con sistemas de ecuaciones, para que sea más fácil el manejo del modelo.

En este libro nos limitaremos al análisis de los modelos basados en computadores, específicamente de aquellos que se prestan para la interpretación gráfica. Los gráficos se pueden utilizar para crear y editar el modelo, para obtener valores para sus parámetros y para visualizar su comportamiento y estructura. El modelo y el medio gráfico para crearlo y visualizarlo son distintos; los modelos, como los de población, no tienen aspectos gráficos inherentes. Entre los tipos de modelos para los cuales se emplean gráficos por computador están los siguientes:

- Los *modelos organizacionales* son jerarquías que representan burocracias institucionales y taxonomías, como los esquemas de clasificación en bibliotecas y las taxonomías biológicas. Estos modelos tienen varias representaciones de gráficos dirigidos, como el diagrama de organización.
- Los *modelos cuantitativos* son ecuaciones que describen sistemas económicos, financieros, sociológicos, demográficos, climáticos, químicos, físicos y matemáticos. Con frecuencia se ilustran con gráficos o diagramas estadísticos.

- Los *modelos geométricos* son colecciones de componentes con geometría bien definida y, con frecuencia, interconexiones entre componentes, incluyendo estructuras de ingeniería y arquitectura, estructuras moleculares y otras de carácter químico, estructuras geográficas y vehículos. Estos modelos suelen representarse con diagramas de bloque o con “fotografías sintéticas” seudorealistas.

El modelado asistido por computador permite que los diseñadores de farmacéuticos modelen el comportamiento químico de nuevos compuestos que pueden ser efectivos en la lucha contra ciertas enfermedades, que los ingenieros aeronáuticos predigan la deformación de las alas durante vuelos supersónicos, que los pilotos aprendan a volar, que los expertos en reactores nucleares predigan el efecto de diversas averías en la planta y desarrollos los remedios apropiados, y que los diseñadores de automóviles prueben la integridad del compartimiento de pasajeros durante los accidentes. En estos casos y en muchos más es más fácil, barato y seguro experimentar con un modelo que con una entidad real. De hecho, en diversas situaciones, como el entrenamiento de pilotos de naves espaciales y el estudio de la seguridad de un reactor nuclear, el modelado y la simulación constituyen el único método factible para aprender acerca del sistema. Por estas razones, el modelado por computador ha sustituido cada vez más a las técnicas tradicionales, como las pruebas de túnel de viento. Los ingenieros y los científicos ahora pueden realizar muchos de sus experimentos con versiones digitales de túneles de viento, microscopios, telescopios, etcétera. Esta simulación y animación con base numérica de los modelos se está convirtiendo rápidamente en un nuevo paradigma de la ciencia, ocupando un lugar junto a las ramas tradicionales de la teoría y la experimentación física. Está de más decir que el modelado y la simulación sólo serán tan buenos como el modelo y sus entradas; en el modelado se aplica estrictamente aquello de “si entra basura, sale basura”.

Los modelos no siempre contienen datos intrínsecamente geométricos: abstracciones como los modelos organizacionales no tienen orientación espacial. No obstante, la mayoría de estos modelos se puede representar en forma geométrica; por ejemplo, un modelo organizacional se puede representar con un diagrama de la organización, o los resultados de la evaluación de un medicamento se pueden representar con un histograma. Incluso si el modelo representa un objeto intrínsecamente geométrico, no se dicta ninguna representación gráfica fija en el modelo o en una vista del modelo. Por ejemplo, podemos elegir si representamos un robot como una colección de poliedros o de superficies curvas y podemos especificar cómo se “fotografiará” el robot: desde qué punto de vista, con qué tipo de proyección geométrica y con qué grado derealismo. Además, podemos escoger si se muestra en forma pictórica la estructura o el comportamiento de un modelo; por ejemplo, podemos observar la disposición física de un circuito VLSI en una pastilla y su comportamiento lógico como función de las entradas y el tiempo.

### 7.1.1 Modelos geométricos

Los modelos geométricos o gráficos describen componentes con propiedades geométricas inherentes y por lo tanto se prestan en forma natural a la representación gráfica. Entre los ingredientes que puede representar un modelo geométrico están los siguientes:

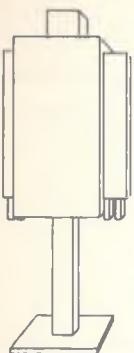
- Distribución espacial y forma de los componentes (es decir, la *geometría* de la entidad) y otros atributos que afectan la apariencia de los componentes, como el color.
- La conectividad de los componentes (o sea, la estructura o *topología* de la entidad); observe que la información de conectividad se puede especificar de manera abstracta (digamos, en una matriz de adyacencia para redes o en una estructura de árbol para una jerarquía) o puede tener su propia geometría intrínseca (las dimensiones de los canales en un circuito integrado).
- Los valores de datos específicos para la aplicación y las propiedades relacionadas con los componentes, como las características eléctricas o texto descriptivo.

También es posible que estén asociados al modelo algoritmos de procesamiento, como análisis de circuitos lineales para modelos de circuitos discretos, análisis de elementos finitos para estructuras mecánicas y minimización de energía para modelos moleculares.

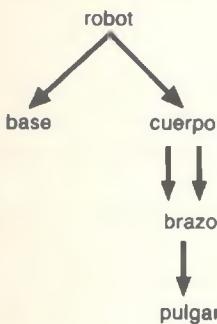
Hay una conciliación entre lo que se almacena de manera explícita en el modelo y lo que hay que calcular antes del análisis o la presentación, la clásica conciliación entre espacio y tiempo. Por ejemplo, el modelo de una red de computadores puede almacenar las líneas de conexión en forma explícita o calcularlas a partir de una matriz de conectividad con un sencillo algoritmo de distribución gráfica cada vez que se solicite una vista nueva. Es necesario almacenar información suficiente en el modelo para permitir el análisis y la presentación, pero el formato exacto y la elección de las técnicas de codificación dependen de la aplicación y de la conciliación entre espacio y tiempo.

### 7.1.2 Jerarquía en modelos geométricos

Los modelos geométricos muchas veces tienen estructuras jerárquicas inducidas por un proceso de construcción ascendente: Los componentes se usan como bloques básicos para crear entidades de nivel superior, que a su vez sirven como bloques para entidades de nivel aún más alto, etcétera. Como los grandes sistemas de programación, las jerarquías rara vez se construyen en forma estrictamente ascendente o descendente; lo que importa es la jerarquía final, no el proceso exacto de construcción. Las jerarquías de objetos son comunes porque pocas entidades son monolíticas (indivisibles); una vez que se descompone una entidad en un conjunto de partes, hemos creado al menos una jerarquía de dos niveles. En el caso poco común de que cada objeto se incluya una sola vez en un

**Figura 7.1**

Vista en perspectiva de un robot androide simplificado.

**Figura 7.2(a)**

Representación de gráfico acíclico dirigido (DAG) de la jerarquía de un robot.

objeto de nivel superior, la jerarquía se puede simbolizar como un árbol, con objetos como nodos y las relaciones de inclusión entre objetos como aristas. En el caso más común de objetos que se incluyen varias veces, la jerarquía se simboliza con un *gráfico acíclico dirigido* (DAG, *directed acyclic graph*). Como ejemplo sencillo de la jerarquía de objetos, en la figura 7.1 se presenta una vista en perspectiva de un robot *androide* rudimentario; en la figura 7.2(a) se presenta la estructura del robot como DAG. Observe que podemos duplicar los objetos que se incluyen varias veces para convertir el DAG en un árbol [Fig. 7.2(b)]. Por convención, las flechas se eliminan por ser redundantes, ya que la relación de orden entre los nodos se indica con las posiciones relativas de los nodos en el árbol: si el nodo A está encima del nodo B, entonces A incluye a B.

El robot está compuesto por un cuerpo que se mueve sobre una base. El cuerpo está compuesto por una cabeza que rota con respecto al tronco; así mismo, conectados al tronco hay dos brazos idénticos que pueden girar en forma independiente sobre un eje horizontal *en el hombro*. El brazo está formado por una parte fija, *la mano*, y un pulgar que se desliza en forma paralela a la mano para formar un elemento primitivo de sujeción. De esta manera, el objeto pulgar se invoca una vez en cada brazo y el objeto brazo se invoca dos veces en el tronco. A lo largo de este capítulo analizaremos la creación de este robot; su forma también se presenta en tres proyecciones ortográficas en las ventanas 2 a 4 de la pantalla presentada en la figura 7.5(b).

Aunque un objeto en la jerarquía está compuesto por primitivas geométricas e inclusiones de subobjetos de nivel inferior, el DAG y el árbol que representan la jerarquía de robot sólo muestran referencias a subobjetos. Esto es análogo al diagrama de jerarquía de procedimientos que se usa comúnmente para mostrar la estructura de llamadas de un programa en un lenguaje estructurado de alto nivel. Es importante observar que la decisión de cómo se construirá exactamente la jerarquía de un objeto compuesto depende del diseñador. Por ejemplo, el robot pudo haberse modelado como una jerarquía de dos niveles, con un objeto raíz consistente en una base, una cabeza y un cuerpo como primitivas geométricas (paralelepípedos, por ejemplo) y dos referencias a un objeto brazo *atómico* formado únicamente por primitivas geométricas.

Muchos sistemas, incluyendo las redes de computadores y las plantas químicas, se pueden representar con diagramas de red, en los cuales los objetos no sólo se incluyen varias veces, sino que además están interconectados en forma arbitraria. Estas redes también se modelan como gráficos e incluso pueden tener ciclos, pero siguen mostrando las propiedades de la jerarquía de inclusión de objetos cuando las subredes ocurren varias veces.

Para simplificar la tarea de construcción de objetos complejos (y sus modelos), con frecuencia se usan componentes atómicos específicos para la aplicación como bloques básicos. En dos dimensiones, estos componentes generalmente se dibujan usando plantillas de plástico o dibujadas por computador que contienen formas simbólicas estándar (también llamadas símbolos o esténciles). En los programas de dibujo, estas formas están compuestas a su vez de primitivas geométricas, como líneas, rectángulos, polígonos, arcos elípticos,



**Figura 7.2(b)**  
Representación de árbol  
de la jerarquía de un robot.

etcétera. En tres dimensiones se usan como bloques básicos formas como cilindros, paralelepípedos, esferas, pirámides y superficies de revolución. Estas formas tridimensionales pueden definirse en función de primitivas geométricas de nivel más bajo, como los polígonos tridimensionales; en este caso, las superficies curvas suaves deben aproximarse con superficies poligonales, lo que implica una disminución en la resolución. Por otra parte, en los sistemas que tratan directamente con superficies o volúmenes de forma libre, las formas como superficies polinomiales paramétricas y los sólidos como cilindros, esferas y conos son en sí primitivas geométricas y se pueden definir de manera analítica, sin pérdida de resolución (véanse los Caps. 9 y 10). En este capítulo hemos usado el término **objeto** para los componentes bidimensionales o tridimensionales que se definen en su propio sistema de coordenadas de modelado, en función de primitivas geométricas y objetos de nivel inferior, y que por lo general no sólo tienen datos geométricos, sino además datos asociados de la aplicación. De esta manera, un objeto es una forma (compuesta) y todos sus datos.

Por lo tanto, una jerarquía se crea para varios propósitos:

- Para construir objetos complejos en forma modular, por lo general empleando invocaciones repetidas a bloques básicos que varían en cuanto a atributos geométricos y de apariencia.
- Para aumentar la economía de almacenamiento, ya que basta almacenar sólo las referencias a los objetos que se usan de manera repetida en lugar de almacenar cada vez la definición completa del objeto.
- Para permitir una sencilla propagación de actualizaciones, ya que un cambio en la definición de un objeto básico se propaga automáticamente a todos los objetos de nivel superior que lo utilizan (puesto que ahora hacen referencia a una versión modificada); en este caso, la analogía entre la jerarquía de objetos y la de procedimientos es muy útil en cuanto a que los cambios en el cuerpo de un procedimiento están reflejados en todas las llamadas al procedimiento.

La aplicación puede usar varias técnicas para codificar modelos jerárquicos. Por ejemplo, se puede usar una red o una base de datos relacional para almacenar la información acerca de los objetos y las relaciones entre ellos. Alternativamente, el programa de aplicación puede mantener una estructura de lista ligada especial, más eficiente, con registros para los objetos y apuntadores para las relaciones. En algunos modelos, las conexiones entre objetos son en sí objetos y también deben representarse con registros de datos en el modelo. Otra alternativa es utilizar una base de datos orientada a objetos [ZDON90]. Los ambientes de programación orientada a objetos como SmallTalk [GOLD83], MacApp [SCHM86] y C++ [STRO91] se usan cada vez más para almacenar información de modelado para los objetos geométricos en programas de aplicación gráfica.

**Interconexiones.** En la mayoría de las redes, los objetos se colocan en localidades específicas (ya sea interactivamente por parte del usuario o en forma automática por parte del programa de aplicación) y después se interconectan. Las interconexiones pueden ser abstractas y por ende de forma arbitraria (p. ej., en diagramas de red o de jerarquía, como los diagramas organizacionales o de programación de proyectos) o pueden tener su propia geometría relevante (p. ej., una pastilla VLSI). Si las conexiones son abstractas, podemos emplear varias convenciones estándar para distribuir los diagramas jerárquicos o de red y usar atributos como el estilo de línea, la anchura de línea o el color para denotar diversos tipos de relaciones (p. ej., *responsabilidad indirecta* en un diagrama organizacional). En el caso de conexiones cuya forma sea importante, como los canales que conectan transistores y otros componentes en un circuito VLSI, son esencia objetos. Las conexiones abstractas y no abstractas muchas veces están **restringidas** a tener orientación horizontal o vertical (lo que en ocasiones se denomina esquema de distribución **Manhattan**) para simplificar la visualización y la construcción física.

**Pase de parámetros en la jerarquía de objetos.** Los objetos invocados como bloques básicos deben colocarse en el lugar preciso dentro de sus objetos padre y, para que quepan, muchas veces hay que alterar su tamaño y orientación. En el capítulo 5 se usaron las matrices de coordenadas homogéneas para transformar primitivas y en el capítulo 6 para normalizar el volumen de vista, por lo que no debe ser ninguna sorpresa que en un modelo jerárquico se apliquen con frecuencia matrices de escalamiento, rotación y traslación a los subobjetos. Sutherland fue el primero en usar este recurso para el modelado gráfico en Sketchpad [SUTH63], acuñando el término **maestro** para definir un objeto y **ejemplar (instance)** para una invocación transformada geométricamente. Como vimos en la sección 4.3.3, los sistemas gráficos que utilizan **listas de presentación jerárquicas** (también conocidas como **listas de presentación estructuradas**) implantan en hardware la jerarquía maestro-ejemplar, usando saltos a subrutinas y unidades aritméticas de punto flotante de alta velocidad para las transformaciones. Como deseamos distinguir las transformaciones geométricas utilizadas en la normalización del volumen de vista de aquellas empleadas en la construcción de la jerarquía de objetos, haremos referencia a éstas como **transformaciones de modelado**. Por supuesto, no existe ninguna diferencia desde el punto de vista matemático entre las transformaciones de normalización y las de modelado.

Una vez más, haciendo una analogía con la jerarquía de procedimiento, muchas veces decimos que un objeto padre *llama* al objeto hijo en la jerarquía y que le pasa **parámetros geométricos** que corresponden a su escala, orientación y posición en el sistema de coordenadas del padre. Como veremos en breve, los paquetes gráficos que permiten la jerarquía de objetos, como SPHIGS, pueden almacenar, componer y aplicar matrices de transformación a los vértices de las primitivas, así como a los vértices de los ejemplares de objetos hijo. Además, es posible pasar a los ejemplares de objetos atributos que afectan

la apariencia. Sin embargo, en la sección 7.5.3 veremos que el mecanismo de pase de parámetros de SPHIGS no es tan general como el de un lenguaje de procedimientos.

### 7.1.3 Relación entre el modelo, el programa de aplicación y el sistema gráfico

Hasta ahora hemos visto los modelos en general y los modelos geométricos con transformaciones de jerarquía y de modelado en particular. Antes de comenzar el estudio de SPHIGS, repasemos brevemente el modelo conceptual gráfico que presentamos en la figura 1.11 y que se repite en forma más elaborada en la figura 3.2, para mostrar la relación que existe entre el modelo, el programa de aplicación y el sistema gráfico. En el diagrama de la figura 7.3, los programas de aplicación se dividen en cinco subsistemas, rotulados con las letras (a) a (e):

- a. Creación, modificación y mantenimiento del modelo a través de la adición, la eliminación y el reemplazo de su información.
- b. Recorrido (rastreo) por el modelo para extraer información para la presentación.
- c. Recorrido por el modelo para extraer información utilizada en el análisis del comportamiento y rendimiento del modelo.
- d. Presentación o dibujo de la información (p. ej., generación de un modelo geométrico, producción de un análisis) y las *herramientas* de interfaz con el usuario (p. ej., menús, recuadros de diálogo).
- e. Realización de tareas diversas de la aplicación que no se relacionan directamente con el modelo o con la presentación (p. ej., administración interna).

El término **subsistema** no implica módulos principales de código; unas cuantas llamadas o un procedimiento corto pueden ser suficientes para implantar un subsistema. Además, un subsistema puede estar distribuido a lo largo del programa de aplicación, en lugar de estar agrupado en un módulo de programa por separado. De esta manera, en la figura 7.3 sólo se muestran los componentes lógicos, no necesariamente los componentes de la estructura del programa; así mismo, aunque sirve para distinguir los procedimientos que construyen, modifican, analizan o presentan el modelo, no siempre queda claro si se debe considerar que un módulo específico es parte del modelo o parte del código de mantenimiento del modelo. Por ejemplo, se podría argumentar que un módulo de análisis de circuitos en realidad forma parte de la definición del modelo, ya que describe su comportamiento. En el caso de un programador que utiliza un lenguaje de procedimientos tradicional como Pascal o C, la figura 7.3 genera mejores resultados si el modelo se considera como algo que contiene sobre todo datos. Las personas familiarizadas con los lenguajes de programación orientada a objetos verán como algo natural la combinación de datos y procedimientos.

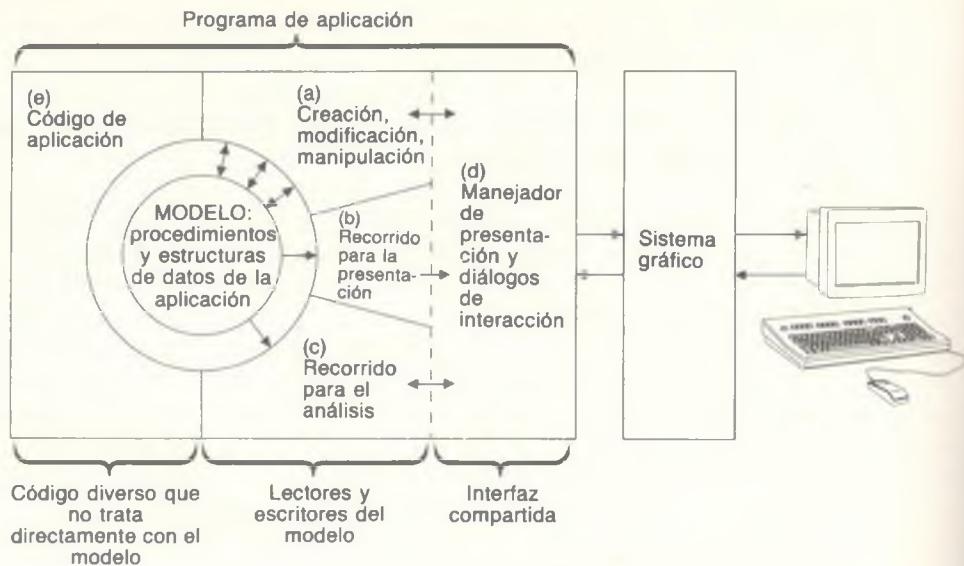


Figura 7.3 El modelo de aplicación y sus lectores y escritores.

En muchos programas de aplicación, sobre todo los industriales, se mantiene la regla 80-20: La mayor parte del programa (el 80 por ciento) tiene que ver con el modelado de entidades, mientras que una porción menor (el 20 por ciento) tiene que ver con la producción de imágenes de las entidades. En otras palabras, en muchas aplicaciones, como CAD, la representación pictórica de los modelos es un medio para alcanzar un fin, y éste es el análisis, la construcción física, el maquinado controlado numéricamente o algún otro tipo de postprocesamiento. Por supuesto, también hay aplicaciones donde *lo fundamental es el objeto*, por ejemplo pintura, bosquejos, producción de filmes y videos y la animación de escenas para simuladores de vuelo. De éstos, todos, con excepción de la pintura, requieren también un modelo a partir del cual se generan las imágenes. En resumen, la mayoría de los gráficos implican mucho modelado (y simulación) y hay bases bastante firmes para afirmar que “los gráficos son el modelado”; los capítulos 9 y 10 están dedicados a este tema de tanta importancia.

## 7.2 Características de los paquetes gráficos de modo retenido

Al analizar las funciones del programa de aplicación, el modelo de aplicación y el paquete gráfico, hemos visto con detalle cuáles son las capacidades que tiene el paquete gráfico y qué sucede al modificar el modelo. SRGP, descrito en

el capítulo 2, opera en **modo inmediato** y no mantiene ningún registro de las primitivas y los atributos que recibe. Por lo tanto, para eliminar y modificar los objetos de la aplicación hay que eliminar la información en la pantalla y por ende regenerar toda la pantalla o efectuar una modificación selectiva; estos dos métodos requieren que la aplicación vuelva a especificar las primitivas a partir del modelo. Por otra parte, PHIGS opera en **modo retenido**: mantiene un registro de todas las primitivas y la información relacionada con ellas para permitir la edición y la actualización automática de la presentación, con lo cual se aligera la carga del programa de aplicación.

### 7.2.1 El almacenamiento central de estructuras y sus ventajas

PHIGS almacena la información en una base de datos de propósito especial llamada **almacenamiento central de estructuras (CSS, central structure storage)**. En PHIGS, una **estructura** es una secuencia de **elementos** —primitivas, atributos de apariencia, matrices de transformación e invocaciones de estructuras subordinadas— cuyo propósito es definir un objeto geométrico coherente. De esta manera, PHIGS de hecho almacena una jerarquía de modelado de propósito especial, completa con transformaciones de modelado y otros atributos que se pasan como *parámetros* a las estructuras subordinadas. Observe que las semejanzas entre la jerarquía de modelado CSS y la lista jerárquica de dibujo en hardware que almacena una jerarquía de maestro-ejemplar. De hecho, podemos considerar que PHIGS es la especificación de un paquete de lista de dibujo jerárquica independiente del dispositivo; por supuesto, una implantación específica se optimiza para un dispositivo de presentación en particular, pero el programador de la aplicación no tiene que preocuparse por ello. Aunque la mayoría de las implantaciones de PHIGS son exclusivamente software, la disposición más común es llevar a cabo en software la manipulación del CSS y emplear una combinación de hardware y software para la generación de imágenes.

El CSS, como la lista de presentación, duplica la información geométrica almacenada en el modelo o base de datos de propósito más general de la aplicación para facilitar un *recorrido de dibujo* rápido, es decir, el recorrido que sirve para calcular una nueva vista del modelo. La ventaja principal del CSS es entonces una regeneración automática y rápida de la pantalla cuando la aplicación actualiza el CSS. Por esta característica vale la pena duplicar los datos geométricos en la base de datos de la aplicación y en el CSS, sobre todo si la implantación de PHIGS utiliza un procesador por separado como *máquina de recorrido* para descargar el recorrido de pantalla de la UCP que ejecuta la aplicación. Las ediciones pequeñas, como el cambio de una matriz de transformación, también se llevan a cabo de manera eficiente en PHIGS.

Otra ventaja del CSS es la correlación de selección automática: el paquete determina la identidad y el lugar en la jerarquía de la primitiva seleccionada por el usuario (véase la Sec. 7.10.2). El recurso de correlación de selección ejemplifica una técnica común que consiste en mover al paquete gráfico subyacente la funcionalidad que se requiere con frecuencia.

Una tercera ventaja del CSS es que los recursos de edición, combinados con las características del modelado jerárquico, facilitan la creación de varios efectos dinámicos —por ejemplo, la dinámica de movimiento— donde se usan transformaciones variables en el tiempo para escalar, rotar y colocar subobjetos dentro de los objetos padre. Como muestra, podemos modelar nuestro sencillo robot para que cada articulación esté representada por una rotación aplicada a una subestructura (p. ej., el brazo es un subordinado rotado del tronco) y rotar el brazo en forma dinámica con la edición de una sola matriz de rotación.

## 7.2.2 Limitaciones de los paquetes de modo retenido

El CSS (una entidad de propósito especial construida principalmente para la presentación y la actualización incremental rápida) hace más sencillas ciertas operaciones comunes del modelado, pero no es necesario ni suficiente para todos los propósitos del modelado. No es necesario porque una aplicación puede llevar a cabo su propia regeneración de pantalla al alterar el modelo, puede efectuar su propia correlación de selección (aunque es una tarea considerable) y puede implantar su propia jerarquía de objetos a través de procedimientos que definen objetos y aceptan transformaciones y otros parámetros. El CSS por lo general no es suficiente en este caso, ya que en la mayoría de las aplicaciones se requiere una estructura de datos de aplicación construida y actualizada por separado para registrar todos los datos apropiados para cada objeto de la aplicación. Por lo tanto, hay duplicidad de todos los datos geométricos y las dos representaciones se deben sincronizar correctamente. Por todas estas razones, algunos paquetes gráficos permiten utilizar coordenadas de punto flotante y recursos de visualización bidimensional y tridimensional sin ningún tipo de almacenamiento de estructuras. La razón fundamental de estos paquetes de modo inmediato es que el mantenimiento del CSS muchas veces no compensa el tiempo de procesamiento adicional, ya que la aplicación por lo general mantiene un modelo de aplicación suficiente para la regeneración de la imagen en pantalla.

En el caso de aplicaciones en las cuales hay cambios estructurales considerables en imágenes sucesivas, no conviene utilizar un paquete de modo retenido. Por ejemplo, en un análisis de *túnel de viento digital* del ala de un aeroplano, donde la superficie se representa con una malla de triángulos, la mayoría de los vértices se desplazan ligeramente al someter el ala a las fuerzas aerodinámicas. No tiene sentido editar una base de datos de estructuras para este caso, ya que en cada imagen nueva se sustituye la mayoría de los datos. De hecho, no se recomienda la edición de la base de datos de estructuras de PHIGS a menos que el número de elementos que se editará sea pequeño con respecto al tamaño de las redes que se presentan. Las herramientas de edición ofrecidas por PHIGS son rudimentarias; por ejemplo, es fácil cambiar una transformación de modelado, pero para modificar un vértice de un polígono hay que eliminar el polígono y especificar la versión modificada. Lo más usual es que las implantaciones se optimicen para un recorrido de presentación y no para la edición masiva, ya que la primera es la operación más común. Además, de cualquier manera hay

que actualizar el modelo de la aplicación, y es más fácil y rápido actualizar una base de datos que dos.

Debido a estas limitaciones, muchas implantaciones de PHIGS ofrecen un recurso de salida en modo inmediato o incluso un modo híbrido donde las primitivas especificadas en modo inmediato se pueden combinar con primitivas retenidas.

## 7.3 Definición y presentación de estructuras

En la sección anterior se analizaron las propiedades generales de PHIGS y SPHIGS. En esta sección comenzaremos con la descripción detallada del paquete SPHIGS; a menos que se indique lo contrario, el análisis también es aplicable a PHIGS. Entre las manipulaciones permitidas con las estructuras de SPHIGS se encuentran las siguientes:

- Apertura (para iniciar la edición) y cierre (para concluir la edición).
- Eliminación.
- Inserción de **elementos de estructura** (los tres tipos primarios de elementos de estructura son **primitivas**, **atributos**, incluyendo aquellas que especifican las transformaciones de modelado, y elementos que invocan subestructuras). Un elemento es un registro de datos que se crea e inserta en la estructura que esté abierta actualmente al invocar un procedimiento **generador de elementos** y que almacena los parámetros del procedimiento.
- Eliminación de elementos de estructura.
- **Colocación para presentación** (similar a la colocación de una fotografía en un tablero de mensajes), sujeta a una **operación de visualización** que especifica cómo se establecerá la correspondencia entre las coordenadas de punto flotante y el sistema de coordenadas de la pantalla.

### 7.3.1 Apertura y cierre de estructuras

Para crear una estructura —por ejemplo la colección de primitivas y atributos que forma el componente de brazo del robot de la figura 7.2— se efectúan llamadas a las funciones generadoras de elementos:

```
void SPH_openStructure (int ID_estructura);
void SPH_closeStructure ( );
```

En esencia estas funciones llevan a cabo la misma tarea que los mandatos de apertura y cierre de archivos en disco. Sin embargo, a diferencia de los archivos en disco, sólo puede haber una estructura abierta en un instante determinado, y

todos los elementos que se especifican mientras esté abierta se almacenan en ella. Una vez cerrada una estructura, puede reabrirse para la edición (véase la Sec. 7.9).

Observamos que las estructuras tienen dos propiedades adicionales. Primero, las primitivas y los atributos sólo se pueden especificar como elementos de una estructura. No hay reglas acerca de cuántos elementos se pueden almacenar en una estructura; puede estar vacía o puede contener una cantidad arbitrariamente grande de elementos, con la única limitación del espacio de memoria. Por supuesto, los elementos que constituyen una estructura por lo general deben formar un conjunto lógicamente cohesivo que defina un solo objeto.

Segundo, los identificadores de las estructuras son números enteros. Como usualmente son utilizados por el programa de aplicación, no por el usuario interactivo, no requieren la forma más general de las cadenas de caracteres, aunque el programador de la aplicación puede definir constantes simbólicas para los identificadores de las estructuras. Los identificadores enteros también permiten establecer una correspondencia práctica entre los objetos en la estructura de datos de la aplicación y los identificadores de estructura de los objetos correspondientes.

### 7.3.2 Especificación de primitivas de salida y sus atributos

Las funciones que generan elementos de primitivas de salida son similares a sus contrapartes en SRGP, pero existen diferencias importantes. En primer lugar, los puntos se especifican con tres coordenadas de punto flotante ( $x$ ,  $y$ ,  $z$ ). Así mismo, estas funciones colocan elementos en la estructura abierta del CSS en lugar de alterar directamente la imagen en pantalla (el dibujo de las estructuras es una operación aparte que se trata en la sección 7.3.3). En este capítulo utilizamos el término **primitiva** como abreviatura de tres entidades relacionadas: una función de generación de elementos, como SPH\_polyLine; el elemento de estructura generado por la función (p. ej., el elemento *polilínea*); y la imagen para presentación que se crea cuando se ejecuta un elemento primitivo durante el recorrido de presentación del almacenamiento central de estructuras. SPHIGS **ejecuta** un elemento primitivo transformando las coordenadas de la primitiva a través de transformaciones de modelado y una operación de vista, incluyendo su recorte con respecto al volumen de vista, para luego **discretizarlo** (es decir, convertirlo a pixeles). Los atributos son más especializados que en SRGP, ya que cada tipo de primitiva tiene sus propios atributos. De esta manera, los atributos como el color y el estilo de línea se *teclean*, con lo cual el programador puede, por ejemplo, restablecer el color actual para las líneas mientras se mantienen los colores vigentes para poliedros y texto.

**Primitivas.** SPHIGS tiene menos primitivas de salida que SRGP, ya que los equivalentes sólidos tridimensionales de algunas primitivas de SRGP (p. ej., un elipsoido) son muy costosos de implantar debido al gran número de cálculos, sobre todo si se compara con las transformaciones, los recortes y las discretizaciones.

La mayoría de las primitivas de SPHIGS son idénticas a sus contrapartes en SRGP en lo que se refiere a sus métodos de especificación (excepto que los puntos son tridimensionales):

```
void SPH_polyLine (int num_vértices, pointList vértices);
void SPH_polyMarker (int num_vértices, pointList vértices);
void SPH_fillArea (int num_vértices, pointList vértices);
/* Como SRGP_polygon */
void SPH_text (point origen, char *cadena);
/* No es completamente tridimensional; véase la Sec. 7.7.2 */
```

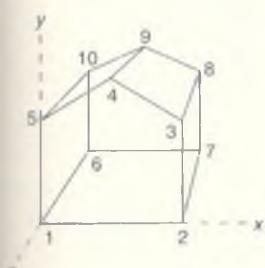
Observe que SPHIGS no verifica que las áreas de relleno (o facetas, descritas a continuación) sean planas y los resultados no estarán definidos si no lo son.

Considere ahora la definición de una casa sencilla, ilustrada en la figura 7.4. Podemos describir esta casa a SPHIGS especificando cada cara (o **faceta**) como un área de relleno, a expensas de una duplicación innecesaria de especificación y almacenamiento (en CSS) de cada vértice compartido. Esta duplicación también reduce la velocidad de generación de la presentación, ya que los cálculos de la operación de visualización se llevarían a cabo más de una vez para cada vértice. El almacenamiento y el tiempo de procesamiento son mucho más eficientes si las facetas se especifican con referencias indirectas a los vértices compartidos. De esta manera, un poliedro se considera como una colección de facetas, cada una de las cuales constituye una lista de índices de vértices y cada índice un apuntador a una lista de vértices. La especificación de un poliedro se puede describir con la siguiente notación:

```
Poliedro = { lista_vértices, lista_facetas }
Lista_vértices = { V1, V2, V3, V4, V5, V6, V7, V8, V9, V10 }
V1 = (x1, y1, z1), V2 = (x2, y2, z2), ...
Lista_facetas = { frontal = { 1, 2, 3, 4, 5 }, derecha = { 2, 7, 8, 3 }, ...
inferior = { ... } }
```

SPHIGS ofrece esta forma eficiente de especificación con su primitiva **polyhedron** (poliedro). En terminología de SPHIGS, un poliedro es una colección de facetas que puede o no encerrar un volumen. En un poliedro cerrado como nuestra casa, los vértices suelen ser compartidos por al menos tres facetas, de manera que la eficiencia del método indirecto de especificación es muy alta. La apariencia de un poliedro es afectada por los mismos atributos que se aplican al relleno de áreas.

La lista de facetas se presenta al generador de elementos poliédricos como un arreglo de enteros (tipo “vertexIndexList” de SPHIGS) que almacena un conjunto concatenado de descripciones de facetas. Cada descripción de faceta es una secuencia de ( $V + 1$ ) enteros, donde  $V$  es el número de vértices en la faceta. Los primeros  $V$  enteros son índices en la lista de vértices; el último entero es (-1) y actúa como centinela que indica el fin de la especificación de faceta. De esta manera, las facetas de la casa se especificarían (a través del cuarto pará-



**Figura 7.4**  
Casa simple definida como  
un conjunto de vértices y  
facetas.

metro de la función, descrita a continuación) enviando el arreglo: 1, 2, 3, 4, 5, -1, 2, 7, 8, 3, -1, ...

```
void SPH_polyhedron
    (int num_vertices, int num_facetas, pointList vertices, vertexIndexList facetas);
```

Observe que los algoritmos de generación de imágenes de SPHIGS requieren que los dos lados de una faceta sean distinguibles (externos e internos). Por lo tanto, los vértices se tienen que especificar en orden contrario al del giro de las manecillas del reloj (regla de la mano derecha) al examinar el lado externo de la faceta.<sup>2</sup>

Como ejemplo sencillo, el código C que se presenta a continuación crea una estructura consistente en un poliedro que modela la casa de la figura 7.4:

```
SPH_openStructure (ESTRUCTURA_CASA);
    SPH_polyhedron (10, 7, lista_vertices_casa, descripción_facetas_casa);
SPH_closeStructure( );
```

En esencia, SPHIGS sólo apoya la geometría poligonal. Más adelante veremos primitivas de modelado tridimensional más avanzadas: las curvas suaves definidas polinomialmente en el capítulo 9 y las primitivas sólidas definidas en el capítulo 10.

**Atributos.** Las funciones que se listan en el programa 7.1 generan elementos de atributos. Durante el recorrido de dibujo, la ejecución de un elemento de atributo cambia el valor del atributo en forma modal: el nuevo valor permanece activo hasta que se cambie explícitamente. Los atributos están unidos a las primitivas durante el recorrido de presentación, como veremos en la sección siguiente y en la sección 7.7.

#### Programa 7.1

Funciones que generan elementos de atributos.

polilíneas:

```
void SPH_setLineStyle (style CONTINUOUS / DASHED / DOTTED / DOT_DASHED);
void SPH_setLineWidthScaleFactor (double factor_escala);
void SPH_setLineColor (int indice_color);
```

relleno de áreas y poliedros:

```
void SPH_setInteriorColor (int indice_color);
void SPH_setEdgeFlag (flag EDGE_VISIBLE / EDGE_INVISIBLE);
void SPH_setEdgeStyle (style CONTINUOUS / DASHED / DOTTED / DOT_DASHED);
void SPH_setEdgeWidthScaleFactor (double factor_escala);
void SPH_setEdgeColor (int indice_color);
```

<sup>2</sup> SPHIGS requiere que un lado de cada faceta se defina como *externo*, incluso si las facetas de un poliedro no forman un objeto cerrado. Además, el lado *interno* de una faceta de poliedro nunca es visible.

polimarcas:

```
void SPH_setMarkerStyle (style MARKER_CIRCLE / MARKER_SQUARE / ...);
void SPH_setMarkerSizeScaleFactor (double factor_escala);
void SPH_setMarkerColor (int indice_color);
```

texto:

```
void SPH_setTextFont (int indice_familia);
void SPH_setTextColor (int indice_color);
```

Los atributos de relleno de áreas son distintos de los de relleno de polígonos en SRGP. Las primitivas de relleno de áreas y poliedros tienen interiores y aristas cuyos atributos se especifican por separado. El interior sólo tiene el atributo de color, mientras que la arista tiene atributos de estilo, anchura y color. Así mismo, la visibilidad de las aristas se puede desactivar con el atributo de bandera de arista, lo cual es útil para varios modos de generación de imágenes, como veremos en la sección 7.8.

La anchura de líneas o aristas y el tamaño de marcas se especifican en forma *no geométrica*: no se definen usando unidades del sistema de coordenadas de mundo y por consiguiente no se sujetan a transformaciones geométricas. Por lo tanto, las transformaciones de modelado y la operación de vista pueden cambiar la longitud aparente de una línea, pero no su anchura. En forma parecida, la longitud de los guiones en el estilo de línea discontinua es independiente de las transformaciones aplicadas a la línea. Sin embargo, a diferencia de SRGP, los pixeles no se usan como unidades de medición, ya que su tamaño depende del dispositivo. En cambio, se establece previamente una anchura/tamaño nominal para cada dispositivo, de manera que la anchura/tamaño tendrá aproximadamente la misma apariencia en cualquier dispositivo de salida; la aplicación de SPHIGS especifica un múltiplo (no entero) de la anchura/tamaño nominal.

SPHIGS no permite patrones, por tres razones. Primero, SPHIGS reserva los patrones para simular el sombreado de color en sistemas de dos niveles. Segundo, el sombreado suave de áreas con patrón en un sistema a color requiere cálculos que son excesivos para la mayoría de los sistemas de presentación. Tercero, el tipo de patrón geométrico llamado **sombreado con líneas paralelas (hatching)** en PHIGS también consume mucho tiempo, incluso en sistemas de presentación con hardware de transformación en tiempo real.

### 7.3.3 Colocación de estructuras para recorrido de presentación

SPHIGS registra una estructura de nueva creación en el CSS, pero no la presenta hasta que la aplicación *coloque* la estructura sujeta a una especificación de visualización específica.<sup>3</sup> SPHIGS lleva a cabo entonces un **recorrido de dibujo**

<sup>3</sup> Esta manera de especificar la presentación de la estructura es la diferencia más importante entre PHIGS y SPHIGS. En el mecanismo más general de PHIGS, la especificación de vista es un elemento de estructura; esto permite cambiar la vista durante el recorrido de presentación y editarla como si fuera cualquier otro elemento. Varias implantaciones actuales de PHIGS también permiten usar el mecanismo de colocación más sencillo que se usa en SPHIGS.

de los elementos de estructura en el CSS, ejecutando cada elemento en secuencia, del primero al último. La ejecución de un elemento primitivo contribuye a la imagen en pantalla (si es visible una porción de la primitiva). La ejecución de un elemento de atributo (tanto atributos de apariencia como de transformación geométrica) cambia la colección de atributos almacenada en un vector de estado (**el estado de recorrido de atributos**) que se aplica a las primitivas subsecuentes conforme se detectan, en forma modal. Así, los atributos se aplican a las primitivas en el orden del recorrido de presentación.

La siguiente función añade una estructura a la lista de estructuras colocadas que mantiene SPHIGS internamente:

```
void SPH_postRoot (int ID_estructura, int índice_vista);
```

El término *root* (raíz) indica que, al colocar una estructura *S* que invoca subestructuras, en realidad estamos colocando el DAG jerárquico, llamado **red de estructuras**, cuya raíz es *S*. Incluso si una estructura colocada no invoca subestructuras, se le denomina raíz; todas las estructuras colocadas son raíces.

El parámetro *índice\_vista* elige una entrada de la tabla de **vistas** (descrita en la sección siguiente); esta entrada especifica cómo se establecerá la correspondencia entre las coordenadas de las primitivas de la estructura y el espacio de coordenadas enteras de la pantalla.

Podemos borrar de la pantalla la imagen de un objeto si eliminamos las estructuras (o elementos) del CSS (véase la Sec. 7.9) o si se usa la función **SPH\_unpostRoot**, menos drástica, que elimina la raíz de la lista de raíces colocadas sin eliminarla del CSS:

```
void SPH_unpostRoot (int ID_estructura, int índice_vista);
```

### 7.3.4 Vista

**La cámara sintética.** Como señalamos en el capítulo 6, conviene pensar en un paquete de gráficos tridimensionales como una cámara sintética que *fotografi* un mundo tridimensional habitado por objetos definidos geométricamente. La creación de una estructura equivale a ubicar un objeto en un estudio fotográfico; la colocación de una estructura es análoga a activar una cámara previamente establecida para abarcar la escena, para luego colocar la fotografía de la escena en un tablero. Como veremos en breve, cada vez que algo cambia en la escena, nuestra cámara sintética produce en forma automática una imagen nueva y actualizada que se coloca en lugar de la anterior. Para producir una animación se muestran varias imágenes estáticas en una sucesión rápida, como lo hace una cámara de cine.

Continuando con la metáfora, consideremos cómo se produce la *imagen* sintética. Primero, el operador debe ubicar y orientar la cámara, y luego decidir

cuánto de la escena aparecerá. Por ejemplo, ¿la imagen será un acercamiento de una parte del objeto que nos interesa o será una vista a distancia de toda la escena? Después, el fotógrafo debe decidir de qué tamaño será la impresión de la fotografía que colocará en el tablero: ¿será una fotografía tamaño cartera o un cartel? Por último, hay que determinar el lugar en el tablero donde se colocará la fotografía. En SPHIGS, estos criterios se representan en una *vista* que incluye la especificación de una operación de vista; el *área de vista* de esta operación especifica el tamaño de la fotografía y su posición en el tablero. No es necesario fotografiar todos los objetos en la base de datos de estructuras con la misma *configuración de cámara*. De hecho se pueden especificar varias vistas para el tablero, como veremos más adelante.

**El área de vista.** Como vimos en el capítulo anterior, el área de vista especifica un paralelepípedo en el sistema NPC (coordenadas de proyección normalizadas) con el cual se establecerá la correspondencia del contenido del volumen de vista definido en VRC (coordenadas de referencia de visualización). Como se establece de manera fija una correspondencia entre el sistema NPC y el sistema de coordenadas enteras del dispositivo físico, el área de vista también especifica dónde aparecerá la imagen en la pantalla. El sistema NPC tridimensional tiene la siguiente correspondencia con el sistema bidimensional de coordenadas de pantalla: el cubo unidad NPC con una esquina en  $(0, 0, 0)$  y la esquina opuesta en  $(1, 1, 1)$  corresponde al cuadrado más grande que puede inscribirse en la pantalla, ignorando la coordenada  $z$ . Por ejemplo, en un dispositivo de presentación con resolución horizontal de 1024 y vertical de 800, un punto  $(0.5, 0.75, z)_{\text{NPC}}$  corresponde a  $(511, 599)_{\text{DC}}$ . Por cuestiones de transportabilidad, una aplicación no debe usar el espacio NPC que esté fuera del cubo unidad; sin embargo, las ventajas de aprovechar un tamaño de pantalla no cuadrada pueden ser superiores al costo de la transportabilidad.

**La tabla de vistas.** SPHIGS mantiene una tabla de vistas con un número de entradas que depende de la aplicación. Cada vista consiste en una especificación del volumen de vista y el área de vista, llamada **representación de vista**, y una lista (inicialmente vacía) de las raíces colocadas en ella. La entrada 0 de la tabla define una **vista por omisión** (default) con el volumen descrito en la figura 6.19(b), con planos anterior y posterior en  $z = 0$  y  $z = -\infty$ , respectivamente. El área para esta vista por omisión es el cubo unidad NPC.

Las representaciones de vistas para todas las entradas en la tabla (excepto la vista 0) se pueden editar desde la aplicación con

```
void SPH_setViewRepresentation (
    int indice_vista, matrix_4x4 matriz_ov, matrix_4x4 matriz_cv,
    double área_vista_NPC_xmín, double área_vista_NPC_xmáx,
    double área_vista_NPC_ymín, double área_vista_NPC_ymáx,
    double área_vista_NPC_zmín, double área_vista_NPC_zmáx);
```

Las dos matrices de  $4 \times 4$  de coordenadas homogéneas son las matrices de orientación de vista y de correspondencia de vista descritas en el capítulo 6. Se producen con las funciones presentadas en el programa 7.2.

### Programa 7.2

*Utilerías para calcular matrices de transformación de visualización.*

```

/* Para establecer el sistema de coordenadas de referencia de visualización UVN */
void SPH_evaluateViewOrientationMatrix (point_3D punto_ref_vista,
                                         vector_3D normal_plano_vista, vector_3D vector_ascendente_vista,
                                         matrix_4x4 *matriz_ov);

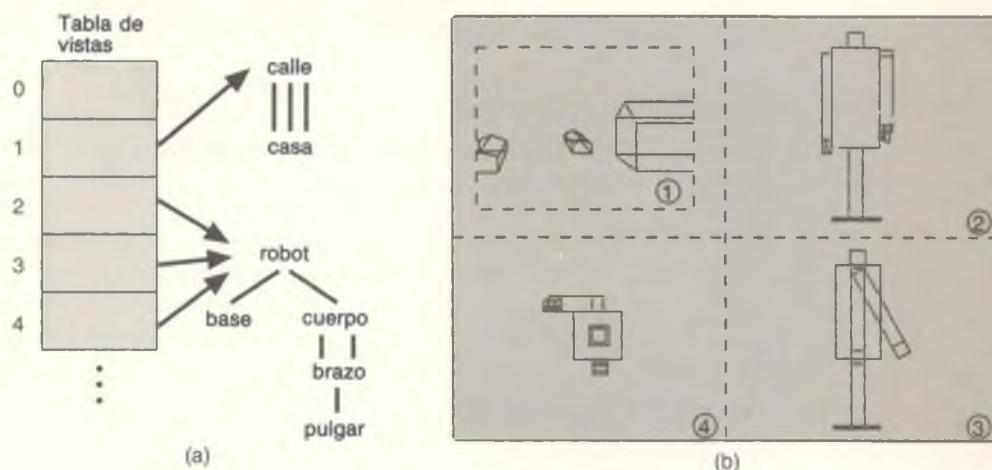
/* Para establecer el volumen de vista y describir como corresponde al espacio NPC */
void SPH_evaluateViewMappingMatrix (
    /* Primero se seleccionan las fronteras del plano de vista */
    double umín, double umáx, double vmin, double vmáx,
    int tipo_proyección,           /* PARALLEL / PERSPECTIVE */
    point_3D punto_ref_proyección           /* En VRC */
    double dist_plano_anterior, double dist_plano_posterior, /* Planos de recorte */
    /* Despues se especifica el área de vista NPC */
    double av_NPC_xmín, double av_NPC_xmáx,
    double av_NPC_ymín, double av_NPC_ymáx,
    double av_NPC_zmín, double av_NPC_zmáx,
    matrix_4x4 *matriz_vm);

```

**Vistas múltiples.** El índice de vistas especificado durante la colocación se refiere a un área de vista NPC específica que describe en qué lugar de la pantalla (tablero) aparecerá la imagen de la estructura (fotografía). Así como se pueden colocar varias fotografías en un tablero, una aplicación puede dividir la pantalla en varias áreas de vista. La utilización de vistas múltiples puede ser muy poderosa. Se pueden presentar simultáneamente varias estructuras en áreas individuales de la pantalla, colocándolas con vistas diferentes. En la figura 7.5(a) aparece una representación esquemática de la tabla de vistas, en la cual sólo se muestran los apuntadores a las listas de redes de estructuras colocadas en cada vista. Podemos ver que hay una vista que presenta una escena callejera; además hay tres vistas separadas de un robot. La estructura del robot se colocó tres veces, cada una con índice de vista distinto. En la figura 7.5(b) se presenta la imagen resultante en la pantalla. Las vistas múltiples del robot no sólo varían en lo que se refiere a sus especificaciones de área de vista, sino también en cuanto a sus especificaciones de volumen de vista.

La situación anterior implica que cada vista tiene a lo sumo una estructura colocada; no obstante, en una misma vista puede colocarse cualquier número de raíces. Así, podemos presentar diferentes estructuras raíz en una imagen unificada si las colocamos juntas en una vista. En este caso, nuestra cámara metafórica tomaría una fotografía compuesta de una escena que contiene muchos objetos.

Otra propiedad de las áreas de vista es que son transparentes, a diferencia de las fotografías reales y las ventanas de los administradores de ventanas. Es



**Figura 7.5** Vistas múltiples que comparten espacio en pantalla. (a) Diagrama esquemático de la tabla de vistas. Cada entrada apunta a una lista de las raíces colocadas en la vista. (b) Imagen resultante. Las extensiones marcadas con guiones y los números en círculos indican las áreas de vista y sus índices de vista relacionados.

En la práctica, muchas aplicaciones *embaldosan* las áreas de vista para evitar superposiciones; sin embargo, éstas también se pueden, si resulta conveniente. Por ejemplo, podemos *componer* dos imágenes distintas creadas con diferentes transformaciones de vista o mostrar objetos definidos con unidades de medición diferentes; así, al construir un diagrama de acercamiento de una pieza de maquinaria, podríamos incorporar un pequeño recuadro donde se muestre toda la máquina (para comunicar el contexto) superpuesto a la imagen del acercamiento (esto se haría seleccionando un área del acercamiento que sólo contuviera fondo).

Para regenerar la pantalla, SPHIGS presenta las redes colocadas recorriendo las raíces colocadas en cada vista dentro de la tabla de vistas, en orden ascendente de índice de vista, comenzando por el número 0; de esta manera, las imágenes colocadas en la vista  $N$  tienen **prioridad de presentación** sobre las imágenes de objetos colocadas en una vista con índice menor que  $N$ , y por consiguiente pueden sobreponerse a ellas. (Note que este trivial sistema de prioridad de vistas es menos complicado que el de PHIGS, que permite a la aplicación asignar prioridades de vista explícitas.)

Observe que una aplicación puede producir varios espacios de coordenadas de mundo independientes y utilizar cualquier unidad de medición que desee. Por ejemplo, en la figura 7.5 la estructura de la calle está definida en un espacio de coordenadas de mundo donde cada incremento en el eje representa 10 metros, mientras que el robot se define en un espacio de coordenadas de mundo totalmente independiente que se mide en centímetros. Aunque cada estructura raíz se modela en su propio espacio de coordenadas de mundo, sólo hay un

espacio NPC por dispositivo de presentación, compartido por todas las estructuras colocadas, ya que dicho espacio es una abstracción del dispositivo de presentación.

### 7.3.5 Aplicaciones gráficas que comparten una pantalla a través de la administración de ventanas

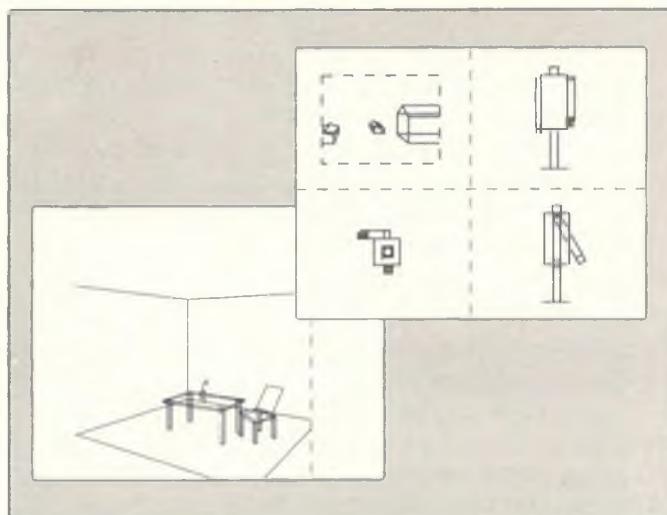
Cuando se diseñaron los primeros paquetes gráficos interactivos estándar a principios de la década de 1970, sólo podía ejecutarse una aplicación gráfica en un instante determinado y ocupaba toda la pantalla. El diseño de PHIGS se inició a finales de los años setenta, cuando este modo de operación era el predominante y antes de que los administradores de ventanas estuvieran disponibles en forma general. Por ello, el cubo unidad del espacio NPC tradicionalmente correspondía a toda la pantalla.

Las modernas estaciones de trabajo gráficas con sistemas operativos multitarea permiten que varias aplicaciones gráficas se ejecuten al mismo tiempo, compartiendo los recursos, la pantalla y el conjunto de dispositivos de entrada de la estación de trabajo, todo bajo el control de un administrador de ventanas. En este ambiente, a cada aplicación se le asigna su propia ventana, la cual actúa como *pantalla virtual*. El usuario puede alterar el tamaño de estas ventanas y moverlas invocando la funcionalidad del administrador de ventanas. La mayor ventaja es que cada aplicación puede actuar como si controlara toda una pantalla; no necesita saber que su pantalla en realidad es una parte de la pantalla real del dispositivo. Por lo tanto, no hay que modificar una aplicación de SPHIGS para usarse en un ambiente de administrador de ventanas; el paquete y el administrador de ventanas cooperan para establecer la correspondencia entre el espacio NPC y una ventana asignada, en lugar de hacerlo con toda la pantalla.

En la figura 7.6 se muestran dos aplicaciones de SPHIGS ejecutándose de manera concurrente en una estación de trabajo gráfica. Como SPHIGS establece una correspondencia entre el espacio NPC y el mayor cuadrado que se ajusta dentro de la ventana de administrador de ventanas, no se puede disponer, para la aplicación SPHIGS, de una parte de cualquier ventana que no sea cuadrada, como se ilustra con la ventana SPHIGS que muestra la tabla y la escena de la silla.

## 7.4 Transformaciones de modelado

En la sección 7.3.2 se presentó un fragmento de código C que creaba una estructura sencilla para modelar una casa. Por cuestiones de sencillez colocamos una de las esquinas de la casa en el origen, alineamos los lados de la casa con los ejes principales y le dimos dimensiones con unidades enteras. Diremos que un objeto definido en el origen y (en su mayor parte) alineado con los ejes prin-



**Figura 7.6** Dos aplicaciones de SPHIGS, cada una en ejecución en su propia ventana de administrador de ventanas. Las líneas punteadas ilustran los límites de las áreas de vista.

cipales está *estandarizado*; no sólo es más fácil definir (determinar las coordenadas de los vértices de) un objeto estandarizado que uno ubicado arbitrariamente en el espacio, sino que además es más fácil manipular la geometría de un objeto estandarizado para modificar su tamaño, reorientarlo o reubicarlo.

Digamos que queremos que la casa aparezca en un lugar distinto, lejos del origen. Por supuesto, podríamos recalcular los vértices de la casa y crear su estructura con el mismo código C que se presenta en la sección 7.3.2 (cambiando únicamente las coordenadas de los vértices). Sin embargo, veremos una poderosa técnica que consiste en transformar un objeto básico estandarizado para cambiar sus dimensiones o su ubicación.

Como vimos en el capítulo 5, podemos transformar una primitiva, como un polígono, si multiplicamos cada vértice, representado como un vector columna  $[x \ y \ z]^T$ , por una matriz  $4 \times 4$  de transformación de coordenadas homogéneas. Las funciones utilitarias que se listan a continuación crean estas matrices:

```

matrix_4x4 SPH_scale (double escala_x, double escala_y, double escala_z);
matrix_4x4 SPH_rotateX (double ángulo);
matrix_4x4 SPH_rotateY (double ángulo);
matrix_4x4 SPH_rotateZ (double ángulo);
matrix_4x4 SPH_translate (double delta_x, double delta_y, double delta_z);

```

Se puede especificar un factor de escala diferente para cada eje, de manera que es posible *estirar* o *comprimir* un objeto de manera no uniforme. Para la rotación, el parámetro de ángulo, expresado en grados, representa un movimiento

en sentido contrario al giro de las manecillas del reloj sobre el eje principal designado, desde el punto de vista de alguien que observa por el eje de  $+ \infty$  al origen.

Las matrices se pueden emplear para crear un elemento de transformación que se coloque en una estructura. La siguiente función es el generador de elementos:

```
void SPH_setLocalTransformation (matrix_4x4 matriz, int REPLACE / PRECONCATENATE /
                                POSTCONCATENATE);
```

En este caso, el prefijo *local* se refiere a la forma en que SPHIGS presenta una estructura. Conforme SPHIGS recorre una estructura, almacena una *matriz local* como información de estado aplicable únicamente a la estructura que se recorre. En un principio la matriz local es, por omisión, la matriz identidad. La matriz local se transforma cuando se detecta un elemento *setLocalTransformation*: se reemplaza o se modifica con una operación de multiplicación, según lo especificado por el parámetro de *modo*. Al detectar una primitiva durante el recorrido, cada uno de sus vértices es transformado por la matriz local y luego se somete a una transformación de vista para la presentación (como veremos más adelante, la jerarquía complica esta regla).

El siguiente código crea una estructura que contiene la casa en una localidad arbitraria y coloca esta estructura para presentación usando la vista por omisión. La casa mantiene su tamaño y orientación originales, estandarizados.

```
SPH_openStructure (ESTRUCTURA_CASA);
SPH_setLocalTransformation (SPH_translate(...), REPLACE);
SPH_polyhedron (...);           /* estos vértices están estandarizados, como antes */
SPH_closeStructure;
SPH_postRoot (ESTRUCTURA_CASA, 0);
```

Las transformaciones sencillas, como ésta, no son muy comunes. Por lo general no sólo queremos trasladar el objeto, sino además alterar su tamaño y orientación. Cuando se desean transformaciones múltiples de primitivas, la aplicación construye la matriz local concatenando (es decir, componiendo) las matrices de transformación individuales en el orden preciso en que se aplicarán. Por lo general, los objetos básicos estandarizados primero se escalan, luego se hacen rotar y por último se trasladan a su ubicación deseada; como vimos en el capítulo 5, este orden evita traslaciones o sesgos indeseados.

El código siguiente crea y coloca una estructura de casa que se desplaza lejos del origen y que se hace rotar a una posición donde se ve su lado en lugar de su frente:

```
SPH_openStructure (ESTRUCTURA_CASA_MOVIDA);
SPH_setLocalTransformation (SPH_rotate(...), REPLACE);
SPH_setLocalTransformation (SPH_translate(...), PRECONCATENATE);
SPH_polyhedron (...);           /* estos vértices están estandarizados, como antes */
```

```
SPH_closeStructure;  
SPH_postRoot (ESTRUCTURA_CASA_MOVIDA, 0);
```

La utilización del modo PRECONCATENATE (preconcatenación) para la matriz de traslación asegura el empleo de la premultiplicación para componer la matriz de traslación con la de rotación, de manera que el efecto de la traslación se aplique después del de la rotación. Por lo tanto, es más común usar la premultiplicación que la postmultiplicación, ya que corresponde al orden de los elementos de transformación individuales. Como SPHIGS lleva a cabo el escalamiento y la rotación con respecto a los ejes principales, si el programador quiere transformar con respecto a un eje arbitrario, primero tendrá que generar las matrices necesarias para la correspondencia entre dicho eje y uno de los principales, después efectuará las transformaciones y luego reestablecerá la correspondencia, como se explicó en el capítulo 5.

SPHIGS lleva a cabo la composición de los elementos de transformación durante el *tiempo de recorrido*; así, cada vez que se regenera la presentación, hay que efectuar la composición. Un método alternativo para especificar una secuencia de transformaciones mejora la eficiencia del proceso de recorrido de pantalla: en lugar de crear un elemento de estructura para cada una, las componemos nosotros en el *momento de la especificación* y generamos un solo elemento de transformación. La función que sigue efectúa una multiplicación de matriz en el momento de la especificación:

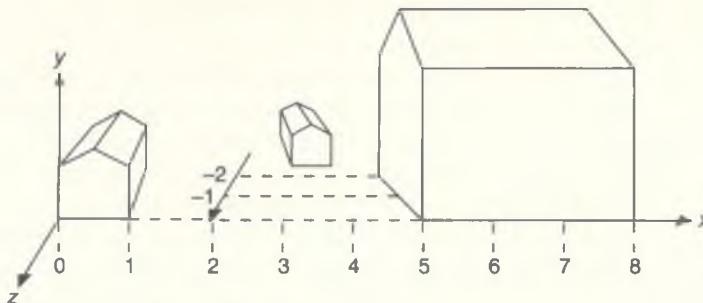
```
matrix_4x4 SPH_composeMatrix (matrix_4x4 mat1, matrix_4x4 mat2);
```

Los dos elementos setLocalTransformation en el código precedente se pueden sustituir entonces por

```
SPH_setLocalTransformation (   
    SPH_composeMatrix (SPH_translate(...), SPH_rotate(...)), REPLACE);
```

La desventaja de este método reside en que ya no es posible efectuar cambios dinámicos en el tamaño, la orientación o la posición de una primitiva “editando” selectivamente el miembro deseado de la secuencia de elementos de transformación local; en lugar de ello, hay que calcular y especificar de nuevo todo el compuesto. La regla empírica de eficiencia es entonces usar la composición en el momento de la especificación a menos que se vayan a actualizar selectivamente las transformaciones individuales, en cuyo caso deberán especificarse en forma individual.

Construyamos la estructura de calle que contiene tres copias de nuestra sencilla casa, como apareció por primera vez en la figura 7.4. En la figura 7.7(a) se presenta una vista en perspectiva de la *casa* a la izquierda, la *mansión* a la derecha y la *choza* en el centro. Hemos agregado líneas punteadas paralelas al eje *x* y marcas para este eje a fin de indicar la posición relativa de las casas; así mismo, hemos empleado un modo de presentación de SPHIGS que muestra las



**Figura 7.7(a)** Modelado de una calle con tres casas. Vista en perspectiva.

aristas alambradas de los poliedros, eliminando las aristas ocultas (véase la Sec. 7.8). En la figura, la casa de la izquierda es un ejemplo no transformado de nuestra casa estandarizada, mientras que las otras dos difieren en cuanto a tamaño, orientación y posición.

La forma directa de crear esta estructura de calle es especificar tres veces el poliedro de la casa estandarizada, precedido por los elementos de transformación deseados, como se ilustra en la representación esquemática de la estructura en la figura 7.7(b). Presentamos un bloque de elementos de transformación consecutivos como una sola unidad; el primer elemento utiliza el modo de reemplazo y los demás el modo de preconcatenación, usando el símbolo de multiplicación ( $\cdot$ ) como separador para indicar la composición. El código que genera la estructura aparece en el programa 7.3.

### Programa 7.3

Código para generar la figura 7.7(a).

```

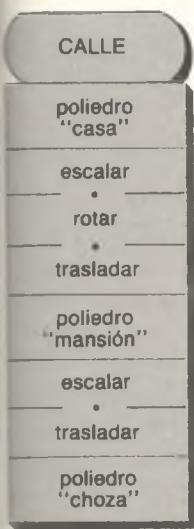
SPH_openStructure (ESTRUCTURA_CALLE);
/* Definir la primera casa en forma estándar */
SPH_polyhedron (...);

/* La mansión se escala por 2 en x, por 3 en y, por 1 en z, se hace rotar 90° sobre y y
después se traslada.
Observe que el lado izquierdo queda hacia adelante y sobre el plano (x, y). */
SPH_setLocalTransformation (SPH_scale (2.0, 3.0, 1.0), REPLACE);
SPH_setLocalTransformation (SPH_rotateY (90.0), PRECONCATENATE);
SPH_setLocalTransformation (SPH_translate (8.0, 0.0, 0.0), PRECONCATENATE);
SPH_polyhedron (...);

/* La choza se escala uniformemente por 0.75, no se hace rotar y se mueve hacia atrás en
z y hacia la derecha en x. */
SPH_setLocalTransformation (SPH_scale (0.75, 0.75, 0.75), REPLACE);
SPH_setLocalTransformation (SPH_translate (3.5, 0.0, -2.5), PRECONCATENATE);
SPH_polyhedron (...);

SPH_closeStructure ( );
SPH_postStructure (ESTRUCTURA_CALLE, 0);

```



**Figura 7.7(b)**  
Estructura del modelo calle-casa.

Podemos eliminar las especificaciones redundantes del poliedro de la casa estandarizada si definimos una función C que haga la llamada para generar el poliedro de la casa, como se muestra en el seudocódigo del programa 7.4. Como esta casa se define con una sola llamada de poliedro, en este ejemplo no es obvia la eficiencia de la técnica; sin embargo, si la casa fuera más compleja y requiriera una serie de especificaciones de atributos y primitivas, este método obviamente necesitaría menos código C. Además, la técnica ofrece otra ventaja de la modularización: podemos cambiar la forma o el estilo de la casa con sólo editar la función correspondiente a la casa, sin tener que modificar el código que crea la calle.

Una función como *casa*, que genera una secuencia de elementos que define un bloque básico y que se usa repetidamente con transformaciones geométricas arbitrarias, se conoce como **función de plantilla**. La función de plantilla es muy conveniente para el programador y constituye además una buena práctica de programación. Sin embargo, observe que a pesar de que *casa* añade un nivel a la jerarquía de funciones del programa en C, no se crea ninguna jerarquía de estructura: el modelo de la calle sigue siendo *plano*. De hecho, la red de estructuras producida por el código del programa 7.4 es idéntica a la generada por el código del programa 7.3. No se obtiene ningún ahorro en términos del número de elementos producidos para la estructura.

Otro cambio que podemos efectuar en nuestra función de plantilla es que acepte una matriz de transformación como parámetro, la cual usaría para especificar un elemento de transformación local. Aunque en algunos casos es conveniente pasar parámetros de transformación, este método carece de la generalidad inherente en nuestro método original, con el cual podemos especificar una cantidad arbitraria de transformaciones antes de llamar a la plantilla.

#### Programa 7.4

Utilización de una función de plantilla para modelar la calle.

```

void casa
{
    SPH_polyhedron (...);

main() /* Secuencia principal */
{
    SPH_openStructure (ESTRUCTURA_CALLE);
    casa ( );
    establecer matriz de transformación local;
    casa ( );
    establecer matriz de transformación local;
    casa ( );
    SPH_closeStructure ( );
    SPH_postStructure (ESTRUCTURA_CALLE, 0);
}
  
```

## 7.5 Redes de estructuras jerárquicas

### 7.5.1 Jerarquía de dos niveles

Hasta ahora hemos trabajado con tres tipos de elementos de estructura: primitivas de salida, atributos de apariencia y transformaciones. A continuación veremos que gran parte del poder de SPHIGS se deriva de la jerarquía de estructuras, implantada a través de un elemento que *llama* a una subestructura cuando se le ejecuta durante el recorrido. No debe confundirse la jerarquía de estructuras con la jerarquía de procedimientos de plantilla de la sección anterior. La jerarquía de procedimientos de plantilla se resuelve en el momento de la especificación, cuando se edita el CSS, y produce elementos en línea, no invocaciones a subestructuras. En cambio, la jerarquía de estructuras inducida por la invocación de subestructuras se resuelve durante el recorrido del CSS para la presentación: la ejecución de un elemento de invocación actúa como llama da a una subrutina.

El **elemento de ejecución de estructura** que invoca a una subestructura se crea con

```
void SPH_executeStructure (int ID_estructura);
```

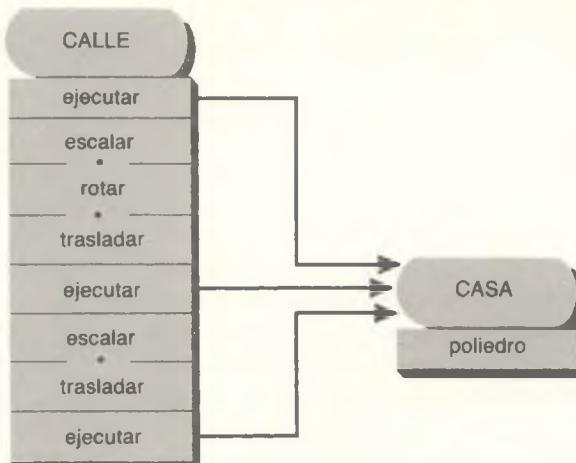
Reemplazemos la función de plantilla de la sección anterior con una función que construya una estructura de casa en el CSS (véase el Prog. 7.5). La función principal invoca esta función sólo una vez y ESTRUCTURA\_CASA nunca es colocada; más bien, su presentación debe invocarse como subobjeto de la estructura de la calle. Observe que las únicas diferencias en la especificación de ESTRUCTURA\_CALLE son el agregar la llamada a la función que construye la estructura de la casa y la sustitución de las llamadas a la función de plantilla con la generación de un elemento de ejecución de estructura. Aunque la imagen presentada es la misma que en la figura 7.7(a), la red de estructuras es diferente, como se muestra en la figura 7.8, donde el elemento de ejecución de estructura se representa con una flecha.

Programa 7.5

*Utilización de una estructura subordinada para modelar la calle.*

```
void construir_casa_estandarizada
{
    SPH_openStructure (ESTRUCTURA_CASA);
    SPH_polyhedron (...);
    SPH_closeStructure ;
}

main ()
{
    construir_casa_estandarizada ( );
    SPH_openStructure (ESTRUCTURA_CALLE);
    /* Secuencia principal */
}
```



**Figura 7.8** Red de estructuras que muestra la invocación de una estructura subordinada.

```

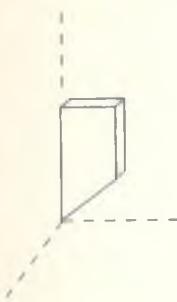
SPH_executeStructure (ESTRUCTURA_CALLE);
/* Primera casa */
establecer matriz de transformación local;
SPH_executeStructure (ESTRUCTURA_CASA);
/* Mansión */
establecer matriz de transformación local;
SPH_executeStructure (ESTRUCTURA_CASA);
/* Choza */
SPH_closeStructure ( );
SPH_postRoot (ESTRUCTURA_CALLE);
  
```

La colocación de ESTRUCTURA\_CALLE indica a SPHIGS que debe actualizar la pantalla recorriendo la red de estructura ESTRUCTURA\_CALLE; el recorrido se lleva a cabo por un árbol de acuerdo a la profundidad (*depth-first*), tal como se ejecuta una jerarquía de función/subrutina. En el ejemplo anterior, durante el recorrido la matriz identidad se asigna como valor inicial de la matriz local de la estructura de la calle y luego se lleva a cabo la primera invocación a la subestructura de la casa, aplicando la matriz local de la estructura de la calle a cada uno de los vértices de la casa, como si el poliedro de la casa fuera un elemento primitivo de la estructura de la calle. Al regresar de la primera invocación, asigna la matriz local como una transformación compuesta deseada y se efectúa la segunda invocación, aplicando la nueva matriz compuesta a los vértices de la casa para crear el segundo ejemplar. Al regresar se cambia de nuevo la matriz local, y el nuevo compuesto se aplica a los vértices de la casa para producir el tercer ejemplar.

Una estructura se considera un objeto independiente, con sus primitivas definidas en su propio sistema de coordenadas de modelado (MCS, *modeling-coordinate system*) de punto flotante; esta forma de pensar facilita la construcción de bloques básicos estandarizados de bajo nivel. Como señalamos

en la sección 5.9, una transformación establece la correspondencia entre los vértices en dos sistemas de coordenadas; aquí, SPHIGS usa la matriz local de la estructura *S* para transformar las primitivas de las estructuras al MCS de *S*.

### 7.5.2 Jerarquía simple de tres niveles



**Figura 7.9(a)**

Chimenea estandarizada para la jerarquía de tres niveles de la calle.

Como ejemplo sencillo de una jerarquía de tres niveles, extendemos la casa en nuestro ejemplo de la calle. La nueva casa está compuesta por la casa estandarizada original (renombrada ESTRUCTURA\_CASA\_SIMPLE) y una chimenea escalada y trasladada para que quede en el lugar correcto sobre la casa. Podríamos modificar la estructura de la casa y añadir las facetas de la chimenea directamente al poliedro original o añadir un segundo poliedro a la estructura, pero en este caso induciremos una jerarquía de tres niveles descomponiendo la casa en dos subobjetos. Una ventaja de esta modularización es que podemos definir la chimenea en forma estandarizada (en el origen, de tamaño unidad) en su propio MCS (como se muestra en la figura 7.9a), y después usar el escalamiento y la rotación para colocarla sobre el techo en el MCS de la casa. Si tuviéramos que definir la chimenea para que ajustara exactamente con el techo y después efectuáramos la correspondencia con el MCS de la casa sin escalar, el cálculo explícito de los vértices sería muy complicado. Sin embargo, con la modularidad basta definir la chimenea estandarizada de manera que su faceta inferior tenga la misma pendiente que el techo de la casa; si se cumple esta condición, podemos aplicar el escalamiento uniforme y la traslación arbitraria.

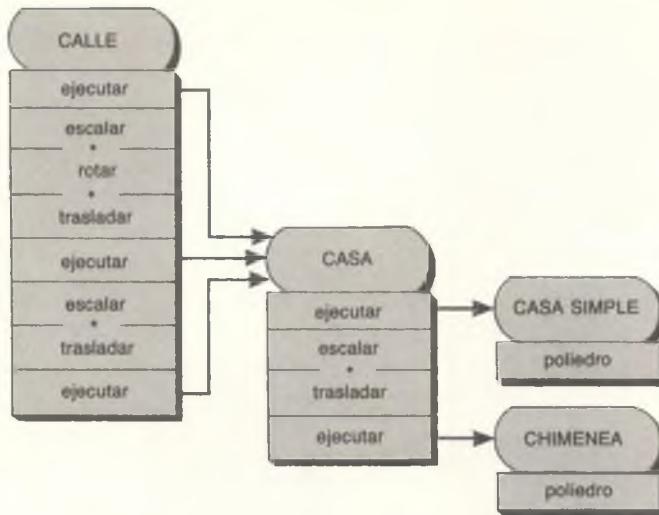
La estructura de la casa modificada se construye con

```
SPH_openStructure (ESTRUCTURA_CASA);
SPH_executeStructure (ESTRUCTURA_CASA_SIMPLE);
    establecer matriz local para escalar/trasladar la chimenea estandarizada al techo de la casa
    simple;
    SPH_executeStructure (ESTRUCTURA_CHIMENEA);
SPH_closeStructure ();
```

¿Qué sucede si la estructura de calle usa una transformación para generar un ejemplar de este objeto de casa de dos niveles y producir la jerarquía de tres niveles que se ilustra en la figura 7.9(b)? Como SPHIGS transforma un padre a través de la transformación de sus subestructuras y elementos componentes, podemos estar seguros de que las dos primitivas componentes (la casa y la chimenea) se transforman juntas como si fueran un objeto unificado (Fig. 7.9c). El punto clave es que no fue necesario modificar la especificación de la estructura de la calle. Así, el diseñador de la estructura de la calle no tiene que preocuparse por los detalles internos de la construcción de la casa o de su edición subsecuente: es una caja negra.

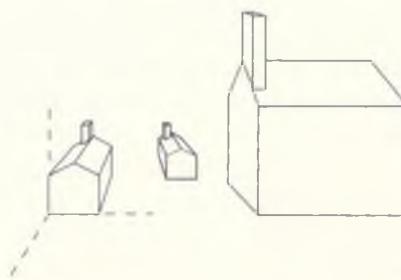
### 7.5.3 Construcción ascendente (*bottom-up*) del robot

Ahora veremos un ejemplo más interesante, nuestro sencillo robot, que combina los conceptos clave del modelado con jerarquía de estructuras y de la edición



**Figura 7.9(b)** Red de estructuras para una jerarquía de tres niveles de la calle.

repetida de transformaciones para lograr la dinámica de movimiento. Un objeto complejo o una jerarquía de sistemas por lo general se conceptualiza y se describe informalmente como descendente (*top-down*). Por ejemplo, el edificio de un departamento de ciencias de la computación está compuesto por pisos, los cuales están formados por oficinas y laboratorios, que a su vez están integrados por mobiliario y equipo, etcétera. Recuerde que nuestro sencillo robot androide está compuesto por un cuerpo y una base de pedestal; el cuerpo está formado por un tronco, una cabeza y dos brazos idénticos, cada uno compuesto por una mano fija y un pulgar *corredizo* (trasladante) para sujeción.



**Figura 7.9(c)** Imagen obtenida con la aplicación de la transformación de jerarquía de tres niveles.

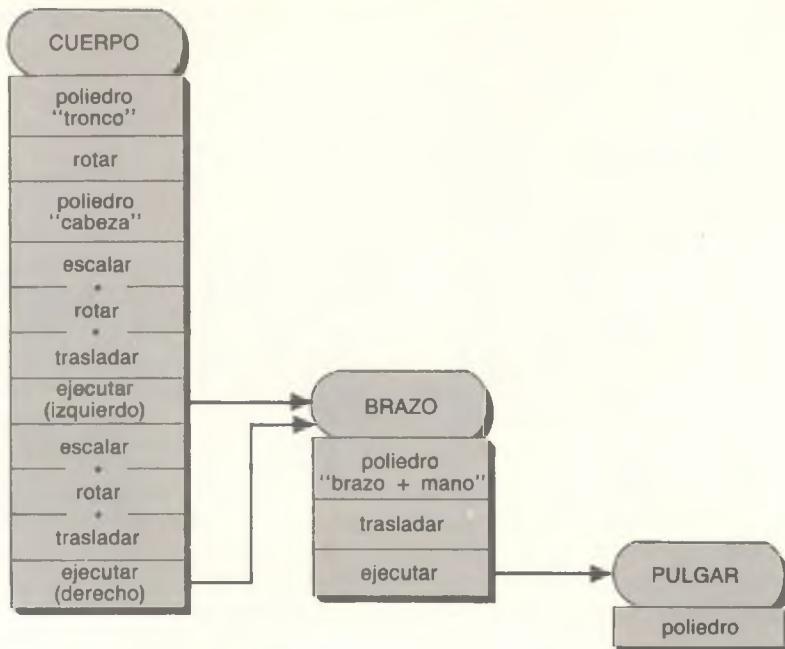
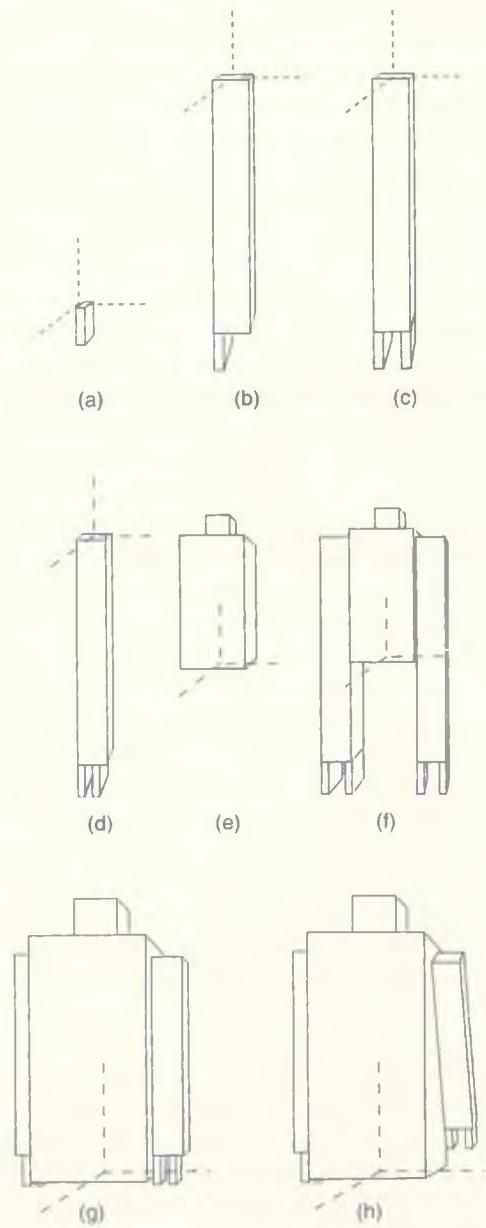


Figura 7.10 Jerarquía de estructuras para el cuerpo del robot.

Aunque el diseño se lleva a cabo en forma descendente, con frecuencia se implanta en forma ascendente, definiendo los bloques básicos que se utilizarán en la definición de los bloques básicos superiores, etc., hasta crear una jerarquía de bloques. De esta manera, durante la construcción del robot se define el bloque del pulgar para el brazo del robot, después el brazo y luego se unen al cuerpo dos ejemplares de bloque del brazo, etc., como se muestra en la jerarquía simbólica de partes de la figura 7.2 y en el diagrama de red de estructuras más detallado de la figura 7.10, correspondiente al cuerpo.

Veamos con mayor detalle el proceso de construcción ascendente para ver cuáles son las transformaciones y la geometría que implica. Tiene sentido diseñar el brazo y el pulgar con las mismas unidades de medición, para que puedan unirse con facilidad. La estructura del pulgar se define con una posición estandarizada en su propio MCS, en el cual *cuelga* sobre el eje y (Fig. 7.11a). La estructura del brazo se define con la misma unidad de medición que se usó para el pulgar; consiste en el poliedro brazo + mano (estandarizado, colgando a lo largo del eje y, como se ilustra en la figura 7.11b) y una invocación trasladada de la estructura del pulgar. El elemento de traslación que precede a la invocación del pulgar es responsable del movimiento del pulgar desde su posición estandarizada en el origen al lugar correcto en la muñeca del brazo, como se presenta en la parte (c).



**Figura 7.11** Construcción del cuerpo del robot. (a) Pulgar. (b) Brazo fijo más mano. (c) Brazo completo. (d) Brazo completo con pulgar trasladado. (e) Tronco y cabeza. (f) Cuerpo con brazos demasiado grandes. (g) Brazos corregidos. (h) Brazo izquierdo rotado.

La red donde el brazo invoca al pulgar es una jerarquía de dos niveles similar al ejemplo calle-casa. Editando el elemento de traslación en la estructura del brazo podemos *desplazar* el pulgar por la muñeca del brazo (Fig. 7.11d).<sup>4</sup>

Después construimos el cuerpo. Como queremos rotar la cabeza, primero se especifica el poliedro del tronco, después una rotación y luego el poliedro de la cabeza (Fig. 7.11e). El siguiente paso es que la estructura del cuerpo invoque dos veces a la estructura del brazo. ¿Qué transformaciones deben preceder a estas invocaciones? Si lo único que nos interesa es la ubicación (es decir, la traslación) correcta de cada brazo en el MCS del cuerpo, podemos producir una imagen como la figura 7.11(f), donde el brazo y el cuerpo se diseñaron con escalas diferentes. Esta situación es fácil de remediar: basta añadir una transformación de escalamiento que preceda a la traslación (Fig. 7.11g). Sin embargo, un escalamiento y una traslación no son suficientes si queremos que los brazos se muevan sobre el eje que conecta los dos hombros (movimiento similar al del brazo de un soldado que marcha); para esto colocaríamos un elemento de rotación en la estructura, antes de la traslación. Hemos completado nuestra definición de la estructura del cuerpo; la construcción de todo el robot queda como tarea para usted en el ejercicio 7.1. En la figura 7.11(h) se muestra el cuerpo cuando el elemento de rotación del brazo izquierdo es distinto de cero.

Como cada invocación del brazo está precedida por una secuencia independiente de transformaciones, el movimiento de cada brazo se puede controlar en forma independiente. Un brazo puede estar colgando hacia abajo mientras el otro se mueve, como en la figura 7.11(h). De hecho, la diferencia en las transformaciones es lo que distingue al brazo derecho del izquierdo. Sin embargo, note que el pulgar móvil no sólo está en el mismo lado, sino además a la misma distancia de la mano fija en el brazo derecho y en el izquierdo, ya que es parte de la definición interna del brazo. (Es más, si la aplicación simplemente cambiara el elemento de traslación en la estructura del brazo, ¡de repente se moverían todos los pulgares en los ejemplares del robot!). Por lo tanto, debemos hacer rotar uno de los brazos 180° sobre el eje y para que los brazos sean simétricos. Una estructura que invoca al brazo puede controlar el tamaño, la orientación y la posición del brazo únicamente *como un todo* y no puede alterar de ninguna manera la construcción interna del brazo. Como mencionamos antes, una subestructura es en esencia una caja negra; el invocador sólo tiene que conocer qué geometría está definida por la subestructura, pero no cómo se creó la subestructura, y de hecho no puede afectar sus aspectos internos.

En resumen, al especificar una estructura trabajamos exclusivamente con las primitivas y las subestructuras de menor nivel que se incluyen en ella, con las transformaciones de modelado que deben usarse para ubicar sus partes componentes y con los atributos que deben emplearse para alterar su apariencia. No necesitamos, ni podemos, tratar los aspectos internos de las estructuras de nivel

<sup>4</sup> El lector observador podrá preguntarse cómo se desliza el pulgar en este modelo, ya que en realidad no está unido a la mano del brazo. De hecho, ninguno de los componentes del modelo del robot está unido a otro a través de objetos que representen articulaciones.

inferior. Así mismo, los componentes se diseñan sin tener en cuenta cómo serán usados por las estructuras que los invocan, ya que las transformaciones se pueden utilizar para obtener el tamaño, la orientación y la posición que se desea. En la práctica es conveniente estandarizar un componente en su MCS local para que sea fácil de escalar y hacer rotar sobre los ejes principales.

Hay que señalar dos aspectos más. Primero, un programador no tiene que diseñar en forma exclusivamente descendente ni implantar en forma ascendente pura; por analogía con la técnica de programación de usar *stubs* y *manejadores*, se pueden usar subestructuras ficticias en la jerarquía de estructuras. Una estructura ficticia puede ser una estructura vacía; de hecho, SPHIGS permite que una estructura ejecute una subestructura que aún no se ha creado, en cuyo caso SPHIGS crea automáticamente la subestructura de la referencia y la asigna como vacía. En algunos casos es deseable que la estructura ficticia sea un objeto sencillo (digamos, un paralelepípedo) de aproximadamente el mismo tamaño que la versión más compleja que se especifica después. Esta técnica permite la depuración descendente de las jerarquías de estructuras complejas antes de haber definido la totalidad de los componentes. Segundo, no hemos comparado la jerarquía de estructuras con la jerarquía de funciones de plantilla, que se analiza en la sección 7.15 de [FOLE90]. Simplemente diremos que la jerarquía de estructuras funciona bien si hay que generar varios ejemplares de un bloque básico y controlar (hasta cierto punto) la apariencia y la ubicación de cada copia, pero no su definición interna.

#### 7.5.4 Programas interactivos de modelado

Los programas interactivos de modelado y construcción tridimensional facilitan la construcción de jerarquías de objetos a través del proceso de montaje ascendente que acabamos de describir. La mayoría de estos programas ofrece una paleta de iconos que representan el conjunto fundamental de bloques básicos del programa. Si el programa de dibujo es específico para la aplicación, también lo serán los bloques básicos; en caso contrario, son átomos comunes como polilíneas, polígonos, cubos, paralelepípedos, cilindros y esferas. El usuario puede seleccionar un ícono para generar un ejemplo en el bloque básico relacionado, y después especificar transformaciones que se aplicarán al ejemplar del bloque básico. Esta especificación suele hacerse por medio de dispositivos de entrada, como el ratón o las perillas de control, que permitan al usuario experimentar con el tamaño, la orientación y la posición del ejemplar hasta que *se vea bien*. Como es difícil juzgar las relaciones espaciales en las proyecciones bidimensionales de escenas tridimensionales, en algunas técnicas de interacción más elaboradas se emplean mallas tridimensionales, escalas numéricas, potenciómetros, controles deslizantes de diversos tipos, realimentación numérica de la posición de los vértices, etcétera (véase la Sec. 8.2.5). Algunos programas de construcción permiten que el usuario combine ejemplares de los bloques básicos gráficos fundamentales para crear un bloque de mayor nivel, el cual se agrega a la paleta de bloques básicos y puede usarse para construir objetos de mayor nivel.

## 7.6 Composición de matrices en el recorrido de presentación

Hasta ahora hemos visto cómo un programador construye un modelo usando el diseño descendente y una implantación (más o menos) ascendente. Sin importar la forma de construir el modelo, SPHIGS presenta el modelo a través de una búsqueda descendente, por profundidades, del DAG raíz de la estructura colocada. Durante el recorrido, SPHIGS procesa toda la geometría especificada con varios niveles de transformación e invocación. Para ver lo que implica, se observa que durante un recorrido descendente, cuando una estructura raíz  $A$  invoca la estructura  $B$ , que a su vez invoca la estructura  $C$ , esto equivale a decir que  $B$  se construyó en forma ascendente con una transformación de las primitivas definidas en el MCS de  $C$  al MCS de  $B$ , y que  $A$  se construyó en forma similar con la transformación de primitivas de  $B$  (incluyendo aquellas definidas por medio de una invocación a  $C$ ) al MCS de  $A$ . El efecto neto es que las primitivas de  $C$  se transformaron dos veces, primero de  $MCS_C$  a  $MCS_B$  y después de  $MCS_B$  a  $MCS_A$ .

Si usamos la notación desarrollada en la sección 5.9,  $M_{B \leftarrow C}$  denota el valor de la matriz local de la estructura  $B$  que, al momento de invocar a  $C$ , establece la correspondencia entre los vértices en  $MCS_C$  y sus posiciones correctamente transformadas en  $MCS_B$ . Así, para establecer la correspondencia entre un vértice en  $MCS_C$  y uno en  $MCS_B$ , se escribe  $V^{(B)} = M_{B \leftarrow C} \cdot V^{(C)}$  (donde  $V^{(H)}$  indica el vector que representa a un vértice cuyas coordenadas se expresan en el sistema de coordenadas  $H$ ) y, en forma similar,  $V^{(A)} = M_{A \leftarrow B} \cdot V^{(B)}$ . Para imitar el proceso de construcción ascendente, al efectuar el recorrido hay que aplicar sucesivamente las transformaciones que establecen la correspondencia entre los vértices de  $C$  a  $B$  y luego de  $B$  a  $A$ :

$$V^{(A)} = M_{A \leftarrow B} \cdot V^{(B)} = M_{A \leftarrow B} \cdot (M_{B \leftarrow C} \cdot V^{(C)}). \quad (7.1)$$

Por asociatividad de matrices,  $V^{(A)} = (M_{A \leftarrow B} \cdot M_{B \leftarrow C}) \cdot V^{(C)}$ . Por lo tanto, en el recorrido se forma la composición de las dos matrices locales y se aplica la matriz resultante a cada uno de los vértices de  $C$ .

Si usamos la notación de árbol, la raíz será el nivel 1 y los hijos sucesivos serán los niveles 2, 3, 4, .... Entonces, por inducción, para cualquier estructura en el nivel  $j$  ( $j > 4$ ) se puede transformar un vértice  $V^{(j)}$  en el MCS de esa estructura al vértice  $V^{(1)}$  en el sistema de coordenadas de la raíz, usando

$$V^{(1)} = (M_{1 \leftarrow 2} \cdot M_{2 \leftarrow 3} \dots \cdot M_{(j-1) \leftarrow j}) \cdot V^{(j)}. \quad (7.2)$$

Como SPHIGS permite transformar con la matriz local las primitivas en el MCS local, un vértice  $V^{(j)}$  se obtiene aplicando la matriz local a los valores de coordenadas de la primitiva:

$$V^{(j)} = M^{(j)} \cdot V^{(\text{prim})}. \quad (7.3)$$

Usamos  $M^{(j)}$  para denotar la matriz local mientras se recorre la estructura con el fin de indicar que la matriz se usa para transformar primitivas en el MCS de nivel  $j$  de la estructura. Si la estructura invoca subsecuentemente a una subordinada, cambia la utilización de la matriz; en este caso se usa para transformar la estructura invocada en el nivel  $j + 1$  al MCS del nivel  $j$ , lo que denotamos con  $M_{j \rightarrow (j+1)}$ . Esto no implica que cambie el valor de la matriz, sólo su utilización.

Al combinar las ecuaciones (7.2) y (7.3) por medio de la asociatividad se obtiene

$$V^{(1)} = (M_{1 \rightarrow 2} \cdot M_{2 \rightarrow 3} \dots \cdot M_{(j-1) \rightarrow j} \cdot M^{(j)} \cdot V^{(\text{prim})}). \quad (7.4)$$

Entonces, para transformar una primitiva en el nivel  $j$  de la jerarquía al MCS de la raíz (que es el espacio de coordenadas mundiales), lo único que tenemos que hacer es aplicar la composición de los valores actuales de la matriz local para cada estructura desde la raíz hasta la estructura en la cual se define la primitiva. Esta composición de matrices locales —el término entre paréntesis en la ecuación (7.4)— se denomina **matriz de transformación de modelado compuesta** (CMTM, *composite modeling transformation matrix*). Cuando el estado de un recorrido es tal que se ejecutan los elementos de la estructura de nivel  $j$ , la CMTM es la composición de las matrices  $j$ . Sólo la última de estas matrices ( $M^{(j)}$ ) puede ser modificada por la estructura, ya que ésta sólo puede modificar su matriz local. Por consiguiente, mientras la estructura esté activa, las primeras  $j - 1$  matrices en la lista CMTM son constantes. La composición de estas  $j - 1$  matrices es la **matriz global** (GM, *global matrix*) —el término entre paréntesis en la ecuación (7.2)— para la estructura de nivel  $j$  que se ejecuta. Es conveniente que SPHIGS mantenga la GM durante el recorrido de la estructura; cuando un elemento de asignación de transformación local modifica la matriz local (LM), SPHIGS puede calcular fácilmente la nueva CMTM con la postconcatenación de la nueva matriz local y la matriz global.

Ahora podemos resumir el algoritmo de recorrido. SPHIGS lleva a cabo un recorrido por profundidades, almacenando CMTM, GM y LM justo antes de invocar una estructura; después asigna la CMTM heredada como valor inicial de la GM y la CMTM de la estructura, y la matriz identidad como valor inicial de la matriz local. Después se aplica la CMTM a los vértices y se actualiza con los cambios a la matriz local. Por último, cuando regresa del recorrido, restaura la CMTM, GM y LM de la estructura padre y continúa. Gracias al almacenamiento y la restauración de las matrices, los padres afectan a los hijos pero no viceversa.

Veamos la forma en que SPHIGS recorre la jerarquía de tres niveles cuerpo-brazo-pulgar de la figura 7.10. Hemos colocado la estructura CUERPO como raíz. En la figura 7.12(a) se presenta una secuencia de fotografías del estado del recorrido; se ha creado una fotografía para cada punto marcado con un número en el diagrama de red de estructuras de la figura 7.12(b).

El estado del recorrido se mantiene con una pila, presentada en la figura 7.12(a), que crece en forma descendente, con la estructura actualmente activa en

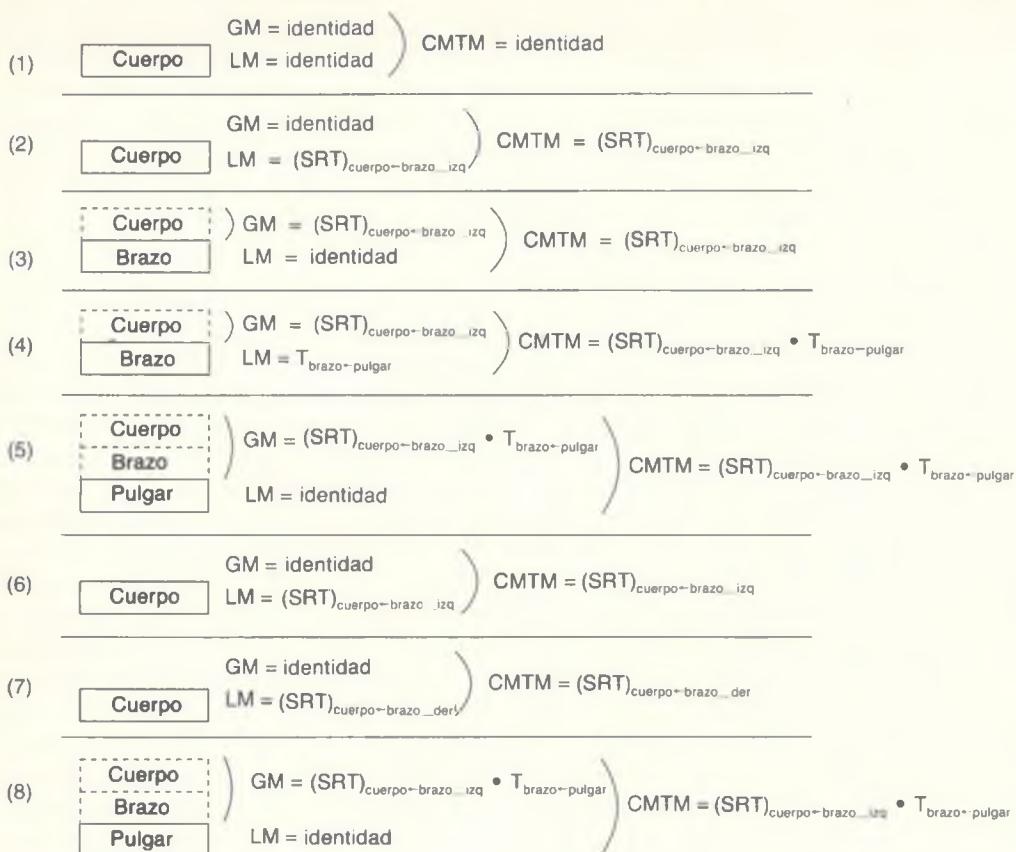
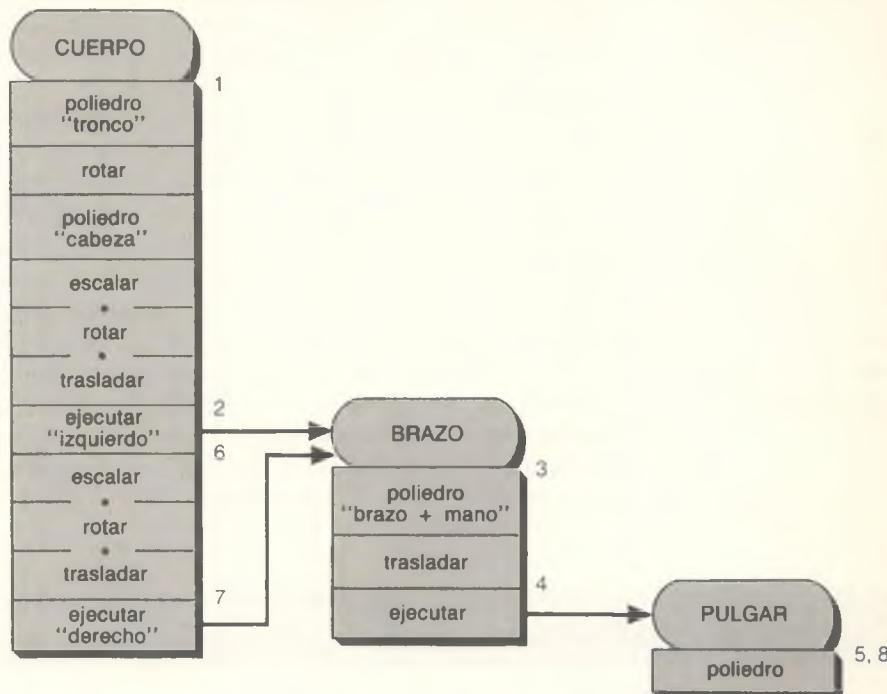


Figura 7.12(a) Fotografías de la pila de estado de recorrido en una jerarquía de tres niveles.

un rectángulo sólido y sus ancestros en rectángulos punteados. Los valores de las tres matrices de estado para la estructura activa se presentan a la derecha del diagrama de pila. Los arcos indican el alcance de una transformación: el arco de la matriz global indica que los ancestros de una estructura contribuyen a la matriz global y el arco CMTM muestra que la CMTM es el producto de la matriz global y la local. Recuerde que en cada grupo de transformaciones, la primera es en modo de reemplazo y las demás en modo de preconcatenación. Por lo tanto, el primer término *rotar* en la estructura sólo se aplica a la cabeza, ya que será reemplazado con el primer término *escalar* que se aplique al brazo izquierdo.

En el punto 1 de la figura 7.12(b), el recorrido está a punto de ejecutar el primer elemento en la raíz. Como la raíz no tiene ancestros, su matriz global es la matriz identidad; la matriz local también es la matriz identidad, como sucede siempre que se inicia la ejecución de una estructura. En el punto 2, la matriz



**Figura 7.12(b)** Red anotada de estructuras para la jerarquía de tres niveles.

local se asigna igual a la composición de la tríada de transformación (escalamiento, rotación, traslación o SRT). Por lo tanto, la CMTM se actualiza con el producto de la matriz identidad global y la composición SRT, con lo cual queda lista para usarse en la transformación del subobjeto brazo al MCS del cuerpo, de manera que se convierta en el brazo izquierdo con  $(SRT)_{cuerpo \leftarrow brazo\_izq}$ . Después, en el punto 3, el recorrido está a punto de ejecutar el primer elemento de la estructura del brazo. La matriz global para la ejecución del brazo es, como era de esperar en un ejemplo de nivel 2, la matriz local del padre en el punto de invocación.

En el punto 4, la matriz local del brazo se asigna a la posición con el pulgar en el brazo ( $T_{brazo-pulg}$ ) y la CMTM se actualiza con el producto de las matrices global y local. Esta CMTM de nivel 2 se convierte en la matriz global para el exemplar de nivel 3 del pulgar (punto 5). Como la matriz local del pulgar es la matriz identidad, la CMTM del pulgar tiene el efecto deseado de transformar primero las coordenadas del pulgar a coordenadas del brazo y después a coordenadas del cuerpo. En el punto seis, el recorrido vuelve el cuerpo luego de las invocaciones al pulgar y al brazo. Las matrices para la estructura del cuerpo son las que existían antes de la invocación, ya que los subordinados no pueden cambiar su matriz local. En el punto 7, la matriz local del cuerpo se reemplaza con

una nueva composición para el brazo derecho. Al descender hasta la estructura del pulgar del brazo derecho (punto 8), la CMTM es casi idéntica a la del punto 5; la única diferencia es que se trata de la matriz de nivel 2 que mueve el brazo a la posición correcta en el cuerpo.

Para animar un objeto compuesto, como el robot, sólo tenemos que pensar en cómo afectará el padre a cada estructura hijo y definir los elementos de transformación apropiados para cada componente que pueda editarse en forma dinámica más adelante. Así, para que el robot gire sobre su eje, alce un brazo y abra ambas manos, se cambian: la matriz de rotación en la estructura del robot para afectar el cuerpo, la matriz de rotación en la estructura del cuerpo para afectar el brazo deseado y la matriz de traslación en la estructura del brazo para afectar el pulgar. Las transformaciones se llevan a cabo independientemente en cada nivel de la jerarquía, pero el efecto es acumulativo. Lo más difícil de la especificación de una animación es trabajar hacia atrás a partir del resultado deseado, como “el robot se mueve a la esquina noroeste de la habitación y recoge un bloque que está sobre la mesa”, para derivar la secuencia de transformaciones que produce dicho resultado.

## 7.7 Manejo de atributos de apariencia en la jerarquía

### 7.7.1 Reglas de herencia

El estado del recorrido de atributos es asignado por los elementos de atributo durante el recorrido y, como en SRGP, se aplica modalmente a todas las primitivas que se detecten. Ya vimos la forma en que los padres afectan a sus hijos por medio de transformaciones geométricas. ¿Qué reglas se aplican a los atributos de apariencia? En nuestro ejemplo de la calle, todas las casas tienen el color por omisión. Para que un objeto tenga un color específico (p. ej., para que una casa sea marrón) podemos especificar el color como elemento inicial en la estructura del objeto, pero con ello el color del objeto será algo intrínseco y no podrá modificarse durante el recorrido. Sería mejor “pasar el color como parámetro”, de manera que el hijo lo pueda heredar, así como hereda la CMTM como su matriz global.

De hecho, en SPHIGS cada subestructura hereda el estado de recorrido tal y como éste existe al momento de la invocación de la subestructura, y puede modificarlo como desee sin afectar a sus ancestros. En otras palabras, los atributos y las transformaciones están ligados de modo dinámico durante el recorrido, no estáticamente en el momento de la especificación. Este lazo dinámico es una de las características principales de SPHIGS, que facilita la adecuación de ejemplares de una subestructura.

Lo que hagan las subestructuras con el estado heredado dependerá del tipo de datos. Ya vimos que, en el caso de las transformaciones geométricas, la subestructura hereda la matriz global pero no puede ir más allá de la herencia.

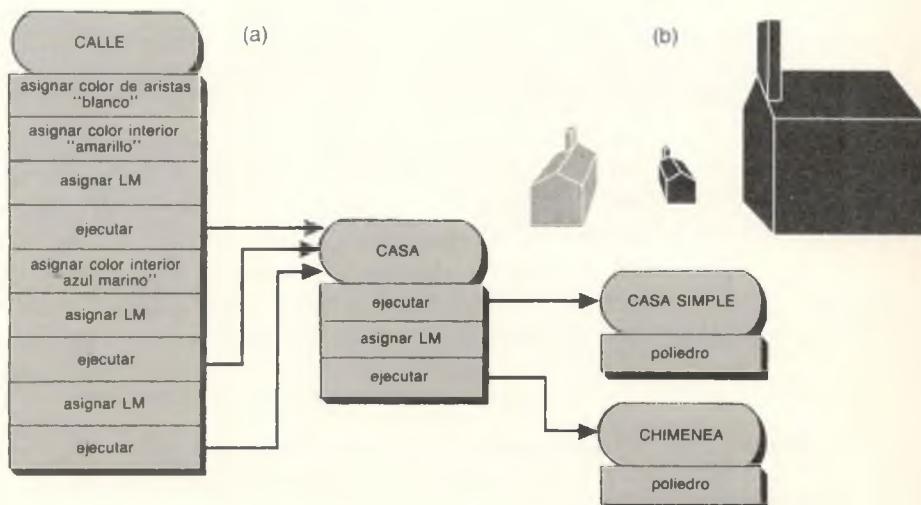
puesto que sólo puede afectar a la matriz local. Los atributos son más sencillos, ya que la subestructura hereda los atributos del padre como valores iniciales de su estado local de atributos pero después puede modificar su estado local. No es necesario distinguir entre los atributos globales y locales, ya que no existe el concepto de la composición de atributos. Observe que este mecanismo tiene el mismo problema que se presentó con la herencia de transformaciones: así como los dos ejemplares del brazo del robot no pueden tener distintas transformaciones para el pulgar, dos ejemplares del brazo no pueden tener el mismo color para la parte fija y colores distintos para el pulgar.

En la red de estructuras de la figura 7.13(a), la estructura de la calle establece los colores para la subestructura de la casa. La imagen resultante se presenta en la figura 7.13(b) y el código que genera la red aparece en el programa 7.6.

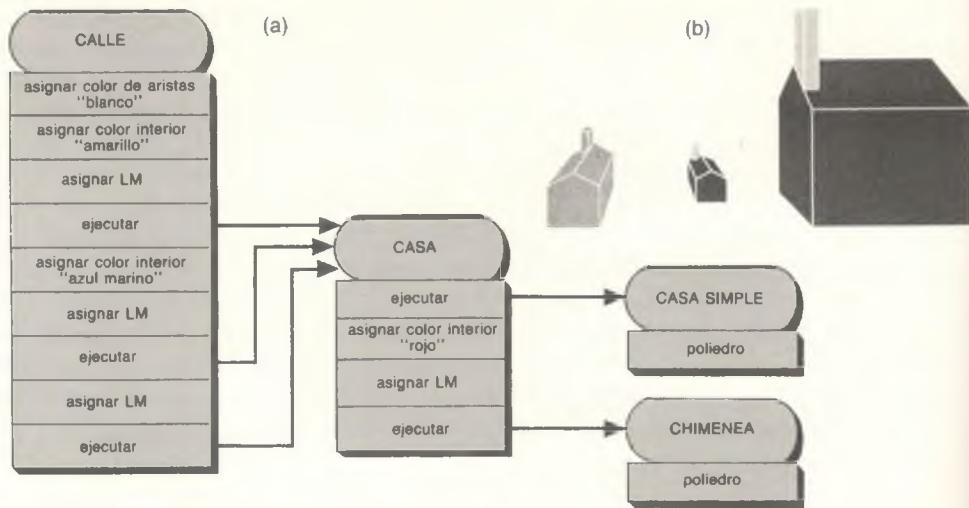
Se puede restablecer un atributo en una subestructura para reemplazar el valor heredado. El siguiente fragmento de código especifica una estructura de casa modificada cuya chimenea siempre es roja:

```
SPH_openStructure (ESTRUCTURA_CASA);
SPH_executeStructure (ESTRUCTURA_CASA_SIMPLE);
SPH_setInteriorColor (COLOR_RED);/* Color rojo */
establecer transformaciones;
SPH_executeStructure (ESTRUCTURA_CHIMENEA);
SPH_closeStructure ( );
```

Usemos esta nueva estructura de casa en conjunto con la estructura de calle generada con el código del programa 7.6. En la figura 7.14 se muestran la red



**Figura 7.13** Utilización de la herencia de atributos para modelar una calle con casas coloreadas. (a) Red de estructuras. (b) Imagen resultante. (Los colores interiores se simulan con patrones).



**Figura 7.14** Estructura subordinada que reemplaza un atributo heredado. (a) Red de estructuras. (b) Vista resultante.

de estructuras y la imagen resultante. El recorrido se inicia en ESTRUCTURA\_CALLE; los atributos de color de los interiores y las aristas tienen sus valores por omisión. El color de las aristas es blanco, un valor que conserva durante todo el recorrido de presentación de la red. La primera asignación de color interior hace que el primer ejemplar de ESTRUCTURA\_CASA herede el color amarillo, que a su vez pasa este color a ESTRUCTURA\_CASA\_SIMPLE, cuyo políedro se presenta en ese color. Cuando el recorrido regresa de ESTRUCTURA\_CASA\_SIMPLE a ESTRUCTURA\_CASA, el atributo del color interior es cambiado de inmediato a rojo por el siguiente elemento. La invocación de ESTRUCTURA\_CHIMENEA produce entonces una chimenea roja con aristas blancas. Por supuesto, ninguna de estas operaciones afecta el grupo de atributos para ESTRUCTURA\_CALLE; cuando el recorrido regresa de ESTRUCTURA\_CASA, se restablece en amarillo el atributo de color interior de ESTRUCTURA\_CALLE. Después se cambia el atributo de color interior a azul marino en preparación para el dibujo de dos casas azul marino.

#### Programa 7.6

```
SPH_openStructure (ESTRUCTURA_CALLE);
SPH_setEdgeColor (COLOR_WHITE);/* Blanco */
```

Código usado para  
generar la figura 7.13.

```
SPH_setInteriorColor (COLOR_YELLOW);/* Amarillo */  
establecer transformaciones;  
SPH_executeStructure (ESTRUCTURA_CASA);
```

```
SPH_setInteriorColor (COLOR_NAVY);/* Azul marino */  
establecer transformaciones;  
SPH_executeStructure (ESTRUCTURA_CASA);  
  
establecer transformaciones;  
SPH_executeStructure (ESTRUCTURA_CASA);  
SPH_closeStructure ( );
```

## 7.7.2 Atributos de SPHIGS y texto no afectados por las transformaciones

En las implantaciones reales de PHIGS, el texto se puede someter a transformaciones como si fuera cualquier otra primitiva. De esta manera, los caracteres de texto en los costados de un camión, presentado en perspectiva, se hacen rotar y se presentan con el recorte frontal de perspectiva apropiado, como si las letras estuvieran formadas por polilíneas o áreas rellenas individuales. Así mismo, los guiones en una línea estarían sujetos a transformaciones geométricas y recortes frontales en perspectiva. Sin embargo, así como los atributos en SPHIGS no son geométricos por cuestiones de rendimiento, tampoco lo es el texto. Como en SRGP, la familia tipográfica del texto determina el tamaño de pantalla para el texto, y una cadena de texto ni siquiera se puede hacer rotar: su imagen siempre está vertical en el plano de la pantalla y nunca se comprime o expande. Por lo tanto, la rotación y el escalamiento afectan el origen del texto, pero no su tamaño ni su orientación. Por ello, el texto primitivo en SPHIGS se usa sobre todo para rotulación.

## 7.8 Actualización de pantalla y modos de presentación

SPHIGS actualiza constantemente la imagen en pantalla para que concuerde con el estado actual del CSS y la tabla de vistas. Las acciones que siguen pueden hacer obsoleta una cantidad arbitraria de la imagen en la pantalla:

- La modificación de una entrada en la tabla de vistas.
- El cierre de una estructura (después de haberse abierto y editado).
- La eliminación de una estructura.
- La colocación o descolocación de una estructura.

Cuando se solicita a SPHIGS que lleve a cabo alguna de estas acciones, debe regenerar la imagen en pantalla para reflejar el estado actual de todas las redes colocadas. La forma que elige SPHIGS para generar la imagen es una

función del modo de generación que ha seleccionado la aplicación. Estos modos presentan una opción entre la calidad y la velocidad de la regeneración. Si aumenta la calidad, la imagen tarda más en generarse. El modo de generación para una vista se establece con

```
void SPH_setRenderingMode (int índice_vista, int WIREFRAME / FLAT / LIT_FLAT/GOURAUD);
```

Aquí presentamos un resumen de los cuatro modos de generación de SPHIGS; se analizan con mayor detalle en los capítulos 12 a 14.

**Modo de generación de alambrado (WIREFRAME).** El modo de alambrado es el más rápido pero el menos realista para presentar una imagen. Los objetos se dibujan como si estuvieran formados por alambres, presentando sólo sus aristas. Se presentan por completo las porciones visibles (dentro del volumen de vista) de las aristas de todos los objetos, sin eliminar las aristas ocultas. Las primitivas se dibujan en orden temporal, o sea, en el orden en que las detecta el recorrido en las redes de estructuras colocadas en la base de datos; este orden es afectado por la prioridad de presentación determinada con el índice de vista, como señalamos en la sección 7.3.4.

En este modo, todos los atributos de aristas afectan la aparición en pantalla en la forma habitual; de hecho, si la bandera de arista se asigna como EDGE\_INVISIBLE (arista invisible), las áreas rellenas y los poliedros son completamente invisibles en este modo.

**Modos de generación sombreados.** En los otros tres modos de generación, SPHIGS presenta de una manera más realista las áreas rellenas y los poliedros, dibujando las áreas y las facetas como polígonos llenos. La adición de áreas sombreadas al proceso de generación aumenta considerablemente su complejidad, ya que el ordenamiento espacial se convierte en algo importante: no deben presentarse las porciones de los objetos que están ocultas (debido a que están oscurecidas por porciones de objetos *más cercanos*). En el capítulo 13 se analizan los métodos de determinación de superficies visibles (también conocida ésta como eliminación de superficies ocultas).

En los tres modos de generación sombreados, SPHIGS *sombrea* los pixeles interiores de las porciones visibles de las facetas; la calidad de la generación varía con el modo. En el modo FLAT (plano), que hemos utilizado con frecuencia en las figuras de este capítulo, todas las facetas de un poliedro se generan con el color interior actual, sin influencia de las fuentes de luz en la escena. Las porciones visibles de las aristas se presentan (si la bandera de aristas 'es EDGE\_VISIBLE) como aparecerían en el modo de alambrado. Si el color interior es igual al color de fondo de la pantalla, sólo se presentan las aristas [esta forma de utilizar la generación FLAT, con la cual se produjeron las figuras 7.7(a) y 7.9(c), simula el alambrado con eliminación de aristas ocultas].

Los dos modos de generación de mayor calidad producen imágenes iluminadas por una fuente de luz;<sup>5</sup> los modelos de iluminación y sombreado se analizan en el capítulo 14. Estas imágenes se *sombreadan* de manera no uniforme; los colores de los pixeles se basan en el valor del atributo de color interior, pero no son iguales. En el modo LIT\_FLAT (iluminación plana), todos los pixeles en una faceta tienen el mismo color, determinado por el ángulo de incidencia de la luz. Como cada faceta tiene color uniforme, la imagen posee una apariencia *facetada* y es notorio el contraste en las aristas entre caras adyacentes. El modo GOURAUD colorea los pixeles para proporcionar una apariencia de sombreado suave que elimina la apariencia facetada.

En el modo FLAT hay que asignar el atributo de bandera de aristas como EDGE\_VISIBLE, ya que si las aristas no fueran visibles, el observador sólo podría determinar la frontera de silueta del objeto. Sin embargo, en los dos modos de mayor calidad generalmente se desactiva la visibilidad de las aristas, ya que el sombreado ayuda al usuario en la determinación de la forma del objeto.

## 7.9 Edición de redes de estructuras para obtener efectos dinámicos

Como ocurre con cualquier base de datos, no sólo debemos tener la capacidad de crear y consultar la base de datos de estructura de SPHIGS (para poder efectuar la presentación), sino además para editarla en una forma práctica. Una aplicación edita una estructura a través de las funciones descritas en esta sección; si la aplicación también mantiene un modelo, debe asegurar que las dos representaciones se editen a la vez. Para la **dinámica de movimiento** se requiere la modificación de las transformaciones de vista y modelado; para la **dinámica de actualización** se necesitan cambios o reemplazos de estructuras. El programador puede editar la lista interna de elementos de la estructura si los cambios son relativamente pequeños; si la edición es de mayor magnitud, lo más común es eliminar y luego volver a especificar toda la estructura.

En lo que resta de esta sección presentaremos métodos de edición intraestructural; consulte el manual de referencia de SPHIGS para obtener más información acerca de las operaciones de edición que afectan estructuras completas (p. ej., eliminación, vaciado), así como para obtener descripciones más detalladas de las funciones que presentamos aquí.

<sup>5</sup> La extensión PHIGS PLUS ofrece varios recursos para controlar la generación, incluyendo la especificación de la colocación y los colores de varias fuentes luminosas, de las propiedades materiales de los objetos que caracterizan su interacción con la luz, etc.; véanse capítulos 12 a 14.

### 7.9.1 Acceso a elementos por medio de índices y etiquetas

Los rudimentarios recursos de edición de SPHIGS y PHIGS se parecen a los de los viejos programas editores de líneas que empleaban números de línea. Los elementos de una estructura se indizan de 1 a  $N$ ; cuando se inserta o elimina un elemento, se incrementa o decrementa el índice asociado con cada elemento que tenga índice mayor en la misma estructura. El **elemento actual** único es aquel elemento cuyo índice se encuentra almacenado en la variable de estado **apuntador de elemento**. Cuando se abre una estructura con la llamada a `SPH_openStructure`, el apuntador de elemento se asigna igual a  $N$  (apuntando al último elemento) o a 0 para una estructura vacía. El apuntador se incrementa al añadir un elemento después del actual, y se reduce al eliminar el elemento actual. El apuntador también puede asignarse en forma explícita con los mandatos de posicionamiento absoluto y relativo:

```
void SPH_setElementPointer (int índice);
void SPH_offsetElementPointer (int delta);
/* + para desplazamiento hacia adelante, - para desplazamiento hacia atrás */
```

Como el índice de un elemento cambia cuando se añade o elimina un elemento precedente en su estructura padre, la utilización de índices para ubicar el apuntador de elementos propicia errores. Por ello, SPHIGS permite que una aplicación asigne elementos de *ubicación*, llamados **etiquetas**, dentro de una estructura. Un elemento de etiqueta recibe un identificador entero al generarse:

```
void SPH_label (int id);
```

La aplicación puede mover el apuntador de elementos con

```
void SPH_moveElementPointerToLabel (int id);
```

Con esto, el apuntador se mueve hacia adelante en busca de la etiqueta especificada. Si se llega al extremo de la estructura antes de encontrar la etiqueta, la búsqueda termina sin éxito. Por lo tanto, es recomendable mover el apuntador al comienzo de la estructura (índice 0) antes de buscar una etiqueta.

### 7.9.2 Operaciones de edición dentro de la estructura

La acción de edición más común es la inserción de elementos nuevos en una estructura. Cuando se invoca una función generadora de elementos, el nuevo elemento se coloca inmediatamente después del elemento actual, y el apuntador de elementos se incrementa para que apunte al nuevo. Los elementos se eliminan con las funciones siguientes:

```
void SPH_deleteElement ( );
void SPH_deleteElementsInRange (int primer_indice, int segundo_indice);
void SPH_deleteElementsBetweenLabels (int primera_etiqueta, int segunda_etiqueta);
```

En todos los casos, después de la eliminación el apuntador se mueve al elemento que está inmediatamente antes de los eliminados, y los elementos restantes se renumeran. La primera función elimina el elemento actual. La segunda elimina los elementos que se encuentran entre los dos especificados (incluyendo a éstos). La tercera función es parecida, pero no elimina los dos elementos de las etiquetas.

Observe que estos recursos de edición afectan todo un elemento o un conjunto de elementos; no hay recursos para la edición selectiva de campos de datos dentro de un elemento. Entonces, si por ejemplo hay que actualizar un solo vértice, el programador tiene que especificar de nuevo todo el poliedro.

**Ejemplo de edición.** Veamos una modificación de nuestro sencillo ejemplo de la calle. La calle consiste ahora en sólo la primera casa y la choza; la primera es fija y la segunda es móvil. Creamos una etiqueta frente a la choza, para que podamos editar posteriormente la transformación que mueve la choza.

Para mover la choza, reabrimos la estructura de la calle, movemos el apuntador a la etiqueta y luego hacemos un desplazamiento hacia el elemento de transformación, lo reemplazamos y cerramos la estructura. La pantalla se actualiza en forma automática después de cerrar la estructura, para presentar la choza en su nueva posición. Este código se presenta en el programa 7.7 y su secuencia de operaciones se ilustra en la figura 7.15.

### Programa 7.7

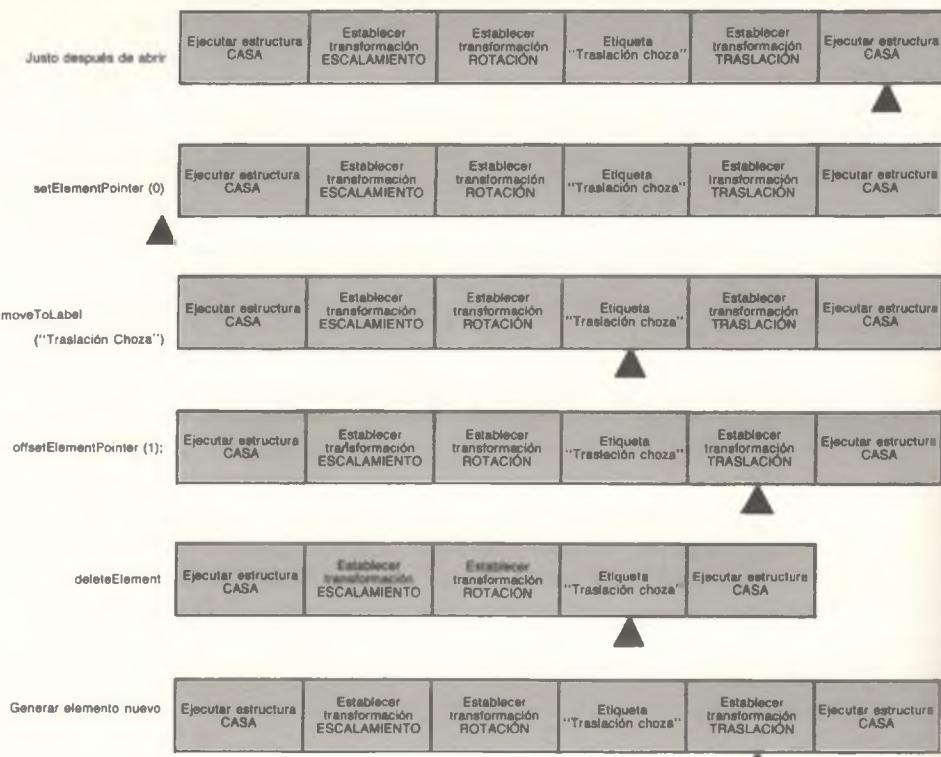
Código para editar la estructura de calle en la figura 7.15.

```
SPH_openStructure (ESTRUCTURA_CALLE);
/* Cuando se abre una estructura, el apuntador de elementos se encuentra al final.
Primero hay que mover el apuntador al inicio para poder buscar etiquetas. */
SPH_setElementPointer (0);
SPH_moveElementPointerToLabel (ETIQUETA_TRASLACIÓN_CHOZA);
SPH_offsetElementPointer (1);/* El apuntador apunta al elemento de transformación */
SPH_deleteElement ( );/* Se reemplaza con una combinación de eliminación e inserción */
SPH_setLocalTransformation (newTranslationMatrix, PRECONCATENATE);
SPH_closeStructure ( );
```

### 7.9.3 Bloques de ejemplares para facilitar la edición

El ejemplo de edición anterior sugiere que debe colocarse una etiqueta frente a cada elemento que queramos editar, pero es obvio que la creación de tantas etiquetas es demasiado laboriosa. Hay varias técnicas para evitar el tedio. La primera consiste en encerrar un grupo de elementos editables entre dos etiquetas y usar estas etiquetas para eliminar o reemplazar todo el grupo. Otra técnica común es agrupar el conjunto de elementos en un formato fijo e introducir el grupo con una sola etiqueta. Para editar cualquier miembro del grupo, el apuntador de elementos se mueve a la etiqueta y después se desplaza de la etiqueta al grupo. Como el formato del grupo es fijo, es fácil determinar el desplazamiento con un número entero pequeño.

Un caso especial de esta técnica consiste en diseñar una forma estándar para la creación de ejemplares de subestructuras, haciendo preceder el elemento de



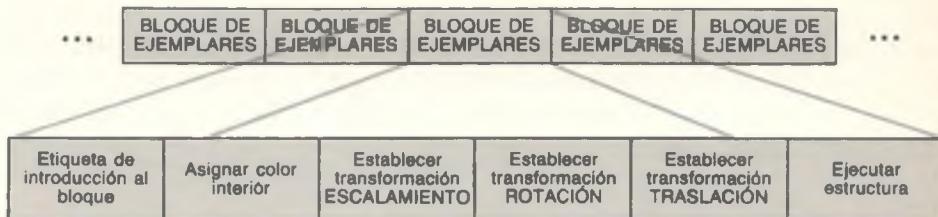
**Figura 7.15** Secuencia de fotografías de la estructura durante la edición. Los triángulos negros indican la posición del apuntador de elementos. (La sintaxis de las llamadas se ha abreviado por cuestiones de ilustración.)

ejecución de estructura con una lista común de elementos de asignación de atributos. En la figura 7.16 se presenta un formato típico de esta secuencia de elementos, llamado **bloque de ejemplares**; primero está la etiqueta que identifica al bloque, después vienen la asignación de color interior, las tres transformaciones básicas y por último una invocación a la estructura del símbolo.

Podemos crear un conjunto de constantes simbólicas para proporcionar los desplazamientos:

```
# define DESP__COLOR__INTERIOR      1
# define DESP__ESCALAMIENTO        2
# define DESP__ROTACIÓN            3
# define DESP__TRASLACIÓN          4
```

Con la utilización del formato fijo para el bloque se garantiza que siempre se modifique de la misma manera un atributo para cualquier ejemplar. Para



**Figura 7.16** Ejemplo de formato de bloque de ejemplares.

cambiar la transformación de rotación de un ejemplar se emplea el siguiente código:

```

SPH_openStructure (ID de la estructura que se editará);
SPH_setElementPointer (0);
SPH_moveElementPointerToLabel (etiqueta del bloque del ejemplar deseado);
SPH_offsetElementPointer (DESP_ROTACIÓN);
SPH_deleteElement ();
SPH_setLocalTransformation (matriz_nueva, modo);
SPH_closeStructure ();
  
```

Otra característica interesante de los bloques de ejemplares es que resulta fácil definir la etiqueta de introducción a cada bloque. Si una aplicación mantiene una base de datos interna que identifica todos los ejemplares de los objetos, algo que sucede con frecuencia, se puede asignar a la etiqueta un número único que use la aplicación para identificar internamente el ejemplar.

#### 7.9.4 Control de la regeneración automática de la imagen en pantalla

SPHIGS actualiza constantemente la imagen en pantalla para reflejar el estado actual de su base de datos de almacenamiento de estructuras y su tabla de vista. Sin embargo, en ocasiones queremos inhibir esta regeneración, ya sea para aumentar la eficiencia o para evitar la presentación al usuario de una imagen con cambios constantes que resulta confusa y muestra etapas intermedias irrelevantes de la edición.

Como hemos visto, SPHIGS suprime la regeneración durante la edición de una estructura; no importa cuántos cambios se realicen, la imagen sólo se regenera al cerrar la estructura. El *agrupamiento en lotes* se hace por cuestiones de eficiencia, ya que cualquier eliminación o transformación de una primitiva puede ocasionar daños a la imagen en pantalla, daños que requieren reparaciones selectivas o un nuevo recorrido por todas las redes colocadas en una o más vistas. Obviamente, es más rápido para SPHIGS calcular el efecto acumulado de varias ediciones consecutivas antes de efectuar la regeneración.

Se presenta una situación similar cuando se aplican varios cambios sucesivos a estructuras diferentes, por ejemplo, cuando se eliminan una estructura y sus subestructuras con llamadas consecutivas a `deleteStructure`. Para evitar este problema, la aplicación puede suprimir la regeneración automática antes de efectuar una serie de cambios, para activarla posteriormente:

```
void SPH_setImplicitRegenerationMode (int ALLOWED / SUPPRESSED);
```

Incluso si la regeneración implícita está suprimida, la aplicación puede solicitar una regeneración de pantalla en forma explícita con la llamada

```
void SPH_regenerateScreen ( );
```

## 7.10 Interacción

Los módulos de interacción de SRGP y SPHIGS se basan en la especificación de PHIGS y, por lo tanto, cuentan con los mismos recursos para establecer modos y atributos de dispositivos y para obtener medidas. El dispositivo de teclado de SPHIGS es idéntico al de SRGP, excepto que el origen del eco se especifica en el espacio NPC ignorando la coordenada  $z$ . La medida del dispositivo localizador de SPHIGS tiene un campo adicional para la coordenada  $z$ , pero no varía en los demás aspectos. SPHIGS también añade dos nuevos recursos de interacción. El primero es la **correlación de selección**, que aumenta la funcionalidad del localizador para proporcionar una identificación del objeto seleccionado por el usuario. El segundo es el dispositivo de **opciones**, descrito en el manual de referencia, que apoya la utilización de menús.

### 7.10.1 Localizador

El localizador de SPHIGS devuelve la posición del cursor en coordenadas NPC, con  $z_{\text{NPC}} = 0$ . También devuelve el índice de la vista de mayor prioridad cuya área de vista encierra al cursor.

```
typedef struct {
    point    position;           /* Posición en pantalla [x, y, 0]_NPC */
    int      viewIndex;          /* Índice de la vista de mayor prioridad
                                  cuya área encierra al cursor */
    int      buttonOfMostRecentTransition;
    enum {
        UP, DOWN
    } buttonChord [MAX_BUTTON_COUNT]; /* Por lo general 1 ... 3 */
    locatorMeasure;
}
```

Cuando dos áreas de vista se sobreponen y la posición del cursor se encuentra en la intersección de sus fronteras, el área de vista con mayor índice (en la tabla de vistas) es la que se devuelve en el segundo campo. Así, el índice de vista

utiliza para establecer la prioridad de vistas tanto en la entrada como en la salida. El campo de índice de vista es útil por varias razones. Considere una aplicación que permite al usuario especificar de manera interactiva los límites de un área de vista, de modo muy similar a como se puede mover o cambiar el tamaño de una ventana con un administrador de ventanas. Como respuesta a la solicitud de cambio de tamaño, el usuario puede seleccionar cualquier lugar en el área de vista. El programa de aplicación usa entonces el campo *viewIndex* para determinar cuál fue la vista seleccionada, en lugar de tener que usar una prueba de punto en rectángulo con respecto a las fronteras del área de vista. El índice de vista también se emplea en aplicaciones que tienen algunas vistas que son sólo de salida; estas aplicaciones pueden examinar el índice de vista devuelto para determinar si es necesario invocar la función de correlación.

### 7.10.2 Correlación de selección

El programador de SPHIGS piensa en función de objetos modelados en lugar de píxeles que componen sus imágenes, por lo cual es conveniente que la aplicación sea capaz de determinar la identidad de un objeto cuya imagen ha seleccionado el usuario. Por lo tanto, la función principal del localizador es proporcionar un punto NPC como entrada para la función de correlación de selección analizada en esta sección. Como vimos en SRGP, la correlación de selección en un mundo plano es una forma directa de detectar **aciertos**, es decir, primitivas cuya imagen está suficientemente cerca de la posición del localizador para considerar que ha sido elegida por el usuario. Si hay más de un acierto, debido a la superposición de primitivas cerca del cursor, se elimina la ambigüedad escogiendo la que se haya dibujado más recientemente, ya que es la que está *encima*. Así, en la correlación de selección bidimensional se examinan las primitivas en orden temporal inverso y se selecciona el primer acierto. La selección de objetos en un mundo tridimensional jerárquico es mucho más complicada, por las razones que describiremos a continuación; por fortuna, SPHIGS evita que la aplicación tenga que realizar esta tarea.

**Selección en una jerarquía.** Considere la complejidad que introduce la jerarquía. Primero, ¿qué información debe devolver la utilería de correlación de selección para identificar el objeto seleccionado? No basta el identificador de la estructura, porque no distingue entre varios ejemplares. Sólo la **trayectoria completa** —una descripción “genealógica” desde la raíz hasta la primitiva seleccionada— ofrece una identificación única.

Segundo, cuando se selecciona una primitiva, ¿a qué nivel de la jerarquía hace referencia el usuario? Por ejemplo, si el cursor se coloca cerca de uno de los pulgares del robot, ¿se refiere el usuario al pulgar, el brazo, el cuerpo o a todo el robot? En algunas ocasiones se desea seleccionar la primitiva, en otras la estructura de la rama, o quizás cualquier otro nivel, hasta la raíz. Algunas aplicaciones resuelven este problema con un mecanismo de realimentación que permite al usuario avanzar por los niveles desde la primitiva hasta la raíz, para especificar exactamente cuál es el nivel que desea (véase el Ejer. 7.8).

**Criterio de comparación.** ¿Cómo se define la proximidad a un objeto si la comparación se tiene que efectuar en tres dimensiones? Como el dispositivo localizador produce efectivamente un valor NPC bidimensional, no hay ninguna base para comparar las coordenadas  $z$  de las primitivas con la posición del localizador. Por lo tanto, SPHIGS puede comparar la posición del cursor sólo con las imágenes de las primitivas en pantalla, no con la ubicación en coordenadas de mundo de dichas primitivas. Si una primitiva es un acierto, se le considera *candidato* para la correlación. En el modo de alambrado, SPHIGS selecciona el primer candidato que detecta durante el recorrido; la razón por la cual se usa esta estrategia es que no existe información de profundidad evidente en la imagen alambrada, por lo que el usuario no espera que la correlación de selección tenga en cuenta la profundidad relativa (un efecto secundario de esta estrategia es que optimiza la correlación de selección). En los modos de generación con sombreado, SPHIGS selecciona el candidato cuyo **punto de acierto** [el punto NPC, en la superficie normalizada (NPC tridimensional) de la primitiva, al cual apunta directamente el usuario] sea el más cercano al punto de vista: el que esté *enfrente*.

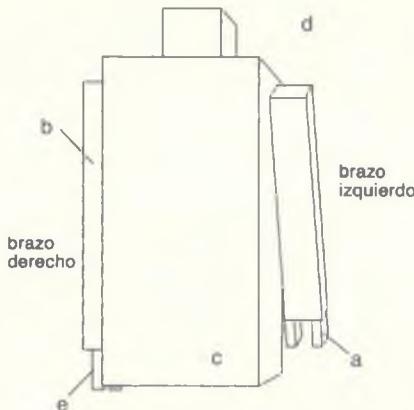
**Utilería de correlación de selección.** Para efectuar la correlación de selección, el programa de aplicación llama a una utilería de correlación de selección de SPHIGS pasando como parámetros un punto NPC y un índice de vista, usualmente obtenidos de una interacción previa con el localizador:

```
void SPH_pickCorrelation (point_3D posición, int índice_vista, pickInformation
    *info_selección);
```

La información devuelta identifica la primitiva seleccionada y sus ancestros a través de una *trayectoria de selección*, descrita con los tipos de datos C del programa 7.8.

Cuando ninguna primitiva está lo suficientemente cerca de la posición del cursor, el valor de *pickLevel* (nivel de selección) que se devuelve es cero y el campo *path* (trayectoria) no está definido. Cuando *pickLevel* es mayor que cero, especifica la longitud de la trayectoria desde la raíz hasta la primitiva seleccionada, o sea, la profundidad de la primitiva en la estructura. En este último caso, las entradas [1] a [*pickLevel*] del arreglo *path* devuelven la identificación de los elementos de la estructura que se encuentran en la trayectoria de la raíz a la primitiva seleccionada. En el nivel más profundo (entrada [*pickLevel*]), el elemento identificado es la primitiva que se seleccionó; en los demás niveles (entradas [*pickLevel* - 1] a [1]) todos los elementos son ejecuciones de estructuras. Cada entrada en *path* identifica un elemento con un registro que proporciona el identificador de la estructura que contiene el elemento, el índice del elemento en la estructura, un código que indica el tipo de elemento y el identificador de selección del elemento (esto se explicará más adelante).

En la figura 7.17 se usa la red de estructuras de la figura 7.10 para el cuerpo del robot, y se presenta la información de selección devuelta por varias selecciones en la imagen que se presenta de la estructura.



Remítase a la red de estructuras presentada en la figura 7.10.

- (a) nivel = 3  
path[1] : estructura CUERPO, elemento 7  
path[2] : estructura BRAZO, elemento 3  
path[3] : estructura PULGAR, elemento 1
- (b) nivel = 2  
path[1] : estructura CUERPO, elemento 11  
path[2] : estructura BRAZO, elemento 1
- (c) nivel = 1  
path[1] : estructura CUERPO, elemento 1
- (d) nivel = 0
- (e) nivel = 3  
path[1] : estructura CUERPO, elemento 11  
path[2] : estructura BRAZO, elemento 3  
path[3] : estructura PULGAR, elemento 1

**Figura 7.17** Ejemplo de correlación de selección.

¿Cómo identifica la trayectoria de selección en forma única cada ejemplar de una estructura que se invoca varias veces en la jerarquía? Por ejemplo, ¿cómo distinguimos una selección en el pulgar izquierdo del robot de una selección en el pulgar derecho? Las trayectorias de selección de los dos pulgares son idénticas excepto en el nivel de la raíz, como lo demuestran los puntos *a* y *e* en la figura 7.17.

#### Programa 7.8

Tipos de almacenamiento de trayectoria de selección.

```
typedef struct {
    int structureID; /* identificador de estructura */
    int elementIndex; /* índice de elemento */
    enum {
        POLYLINE, POLYHEDRON, EXECUTE_STRUCTURE
    } elementType; /* tipo de elemento */
    int pickID; /* identificador de selección */
} pickPathItem; /* elemento de trayectoria de selección */
```

```
typedef pickPathItem pickPath[MAX_HIERARCHY_LEVEL];
```

```
typedef struct {
    int pickLevel; /* nivel de selección */
    pickPath path; /* trayectoria */
} pickInformation; /* información de selección */
```

El *identificador de selección* puede proporcionar una correlación de selección con mayor resolución que el identificador de la estructura. Aunque pode-

mos usar el índice del elemento para identificar los elementos individuales, está sujeto a cambios cuando se edita la estructura. Por lo tanto, es más fácil emplear el identificador de selección, ya que éste no se ve afectado por la edición de otros elementos. Su valor por omisión es cero y se asigna modalmente en una estructura. Un elemento de identificador de selección se genera con

```
void SPH_SetPickIdentifier (int ID);
```

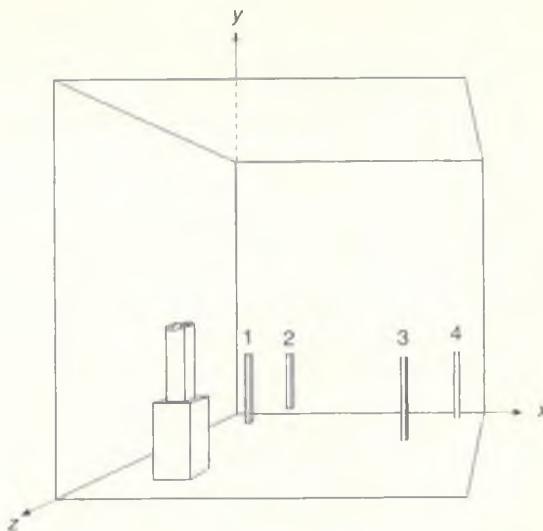
El elemento de identificador de selección es ignorado durante el recorrido de presentación. Además, un identificador de selección no tiene nociones de herencia; su valor inicial es cero cuando SPHIGS inicia el recorrido de una estructura, ya sea ésta una raíz o una subestructura. Por estas dos razones, los identificadores de selección no se comportan como atributos. Varias primitivas en una estructura pueden tener identificadores únicos o compartir un mismo identificador; esto permite una resolución arbitrariamente fina de la correlación de selección en una estructura, según lo requiera la aplicación. Aunque las etiquetas y los identificadores de selección son por ende mecanismos distintos, el primero para la edición y el segundo para la correlación de selección, muchas veces se utilizan en conjunto. Específicamente, cuando las estructuras se organizan con la técnica de bloques de ejemplares descrita en la sección 7.9.2, un elemento de identificador de selección también forma parte del bloque y por lo general se le asigna el mismo valor entero que tiene la etiqueta del bloque.

### Ejemplo 7.1

**Problema:** Mejore la animación del robot para permitir la interacción con el usuario. Permita que existan varios objetos en la habitación y que el usuario seleccione (con el localizador) cuál será el que deba recoger el robot. Para que el problema sea más sencillo, el robot puede tener un solo brazo.

**Respuesta:** *Perspectiva del usuario.* El usuario ve una habitación simple, con el robot de un solo brazo en la parte frontal izquierda. En la parte posterior están colgadas varias barras. Con el ratón y su botón izquierdo, el usuario puede seleccionar cualquiera de estos objetos y el robot se deshará de él. El usuario puede salir en cualquier momento oprimiendo la tecla ‘s’.

*Ubicación de los objetos.* Ubicamos los objetos que puede manipular el robot de manera que sea suficiente una estrategia sencilla para aproximarse a uno de ellos, sujetarlo y eliminarlo. Los objetos se ubican espaciados en forma lateral uniforme (en  $x$ ) y limitados por la profundidad de la región de la habitación donde deben ubicarse (es decir, restringidos en  $z$ ); flotan en el espacio a una altura conveniente. A continuación se muestra una disposición típica:



Esta disposición de los objetos permite que el robot se mueva por una trayectoria paralela al eje  $x$ , frente a los objetos, hasta quedar en línea con uno de ellos (alineados en  $x$ ). El movimiento subsecuente hacia el objeto es paralelo al eje  $z$ , y es fácil determinar la cantidad de movimiento hacia adelante necesario para colocar el robot al alcance del objeto. A partir de esa posición, el robot puede sujetar el objeto, levantarla y luego girar y regresar por el mismo camino: hacia adelante en una línea paralela al eje  $z$ . Permitimos que el robot avance hasta una coordenada  $z$  fuera de la habitación (un pasillo, por ejemplo) y lo movemos lateralmente (paralelo al eje  $x$ ). Observe que este esquema es aplicable a todos los objetos y sólo se requieren como parámetros sus coordenadas  $x$  y  $z$ . En la siguiente secuencia de ilustraciones se muestra cómo se aproxima el robot a una barra seleccionada (3), se inclina para sujetarla y se prepara para abandonar la habitación y deshacerse de la barra que acaba de seleccionar.



Una vez que el robot está fuera de la pantalla, el usuario no sabe cuál es la acción que aquél lleva a cabo. Por lo tanto, queremos deshacernos del objeto que carga el robot de la manera más sencilla posible y reorientar el robot para

que regrese a la pantalla. Para ello, basta retirar el objeto de la jerarquía del robot, restablecer la posición de los pulgares y dejar que el robot regrese a su área de espera.

A continuación se presenta el código C para implantar este algoritmo.

*Programa para permitir que el usuario interactúe y controle la animación del robot.*

```

/* definir las constantes necesarias para usarse en el programa */
/* definir la geometría necesaria para la habitación, los objetos y el robot */

main (argc, argv)
int argc;
char **argv;

{
    /* establecer vistas iniciales, colores y parámetros de SPHIGS */

    /* Prepararse para la entrada del evento */
    SPH_setInputMode (KEYBOARD, EVENT);
    SPH_setKeyboardProcessingMode (RAW);

    SPH_setInputMode (LOCATOR, EVENT);
    SPH_setLocatorButtonMask (LEFT_BUTTON_MASK);

    mostrar_mundo (HABITACIÓN)/* presentar la vista inicial de la habitación */
    fin = NO;
    do {
        dispositivo = SPH_waitEvent (INDEFINITE);
        switch (dispositivo) {
            case KEYBOARD;
                SPH_getKeyboard (medida_tecla, KEYMEASURE_SIZE);
                if (medida_tecla[0] == 'S' || medida_tecla[0] == 's')
                    fin = Sí;
                break;
            case LOCATOR; {
                SPH_getLocator (&pos_actual_localizador);
                /* si se oprimió el botón, se revisa si hay una selección... */
                if (pos_actual_localizador.button_chord[LEFT_BUTTON] == DOWN) {
                    SPH_pickCorrelate (pos_actual_localizador.posición,
                        pos_actual_localizador.indice_vista, &info_selección);
                    if (info_selección.pickLevel > 2) /* objeto en segundo nivel */
                        if (info_selección.path[1].structureID == OBJECT_SET)
                            plan_eliminación (info_selección.path[1].pickID);
                }
            }
        }
    }
}
while (!fin);
/* LOCATOR */
/* switch */
/* do */

```

```
SPH_end ( );  
}  
  
void plan_eliminación (int ID_objeto)  
{  
    double x, z;  
  
    x = info_objetos[ID_objeto].x;  
    z = info_objetos[ID_objeto].z;  
  
    /* movimiento para alinear el robot con el objeto en x: */  
    mover_robot (x, Z_ESPERA);  
    /* encarar el objeto y aproximarse a él: */  
    girar_robot (NORTE, SENTIDO_MANECILLAS);  
    mover_robot (x, z + DISTANCIA);  
  
    /* tomar el objeto */  
    bajar_brazo ( );  
    sujetar ( );  
    recoger_objeto (ID_objeto);  
    levantar_brazo ( );  
    /* objeto sujetado. */  
  
    /* llevarse el objeto */  
    girar_robot (ESTE, CONTRA_MANECILLAS);  
    girar_robot (SUR, CONTRA_MANECILLAS);  
    mover_robot (x, Z_PASILLO);  
    girar_robot (ESTE, SENTIDO_MANECILLAS);  
    mover_robot (X_FUERA_PANTALLA, Z_PASILLO);  
  
    /* soltar el objeto */  
    bajar_brazo ( );  
    liberar ( );  
    soltar ( );  
    levantar_brazo ( );  
  
    /* regresar al área de espera */  
    girar_robot (OESTE, CONTRA_MANECILLAS);  
    mover_robot (X_ESPERA, Z_PASILLO);  
    girar_robot (NORTE, CONTRA_MANECILLAS);  
    mover_robot (X_ESPERA, Z_ESPERA);  
    girar_robot (ESTE, CONTRA_MANECILLAS);  
}  
  
void recoger (int ID)  
/* Elimina el objeto indicado de la habitación y lo agrega  
a la mano (brazo) del robot */
```

```

{
    matrix matriz_temp;                                /* matriz de transformación */

    SPH_openStructure (CONJUNTO_OBJETOS);
    SPH_setElementPointer (0);
    SPH_moveElementPointerToLabel (info_objetos[ID].etiqueta);
    SPH_offsetElementPointer (1);
    SPH_deleteElement ( );
    SPH_executeStructure (OBJETO_NULO);
    SPH_closeStructure ( );

    SPH_openStructure (BRAZO);
    SPH_setElementPointer (0);
    SPH_moveElementPointerToLabel (BRAZO_DIBUJAR_BARRA);
    SPH_offsetElementPointer (1);
    SPH_deleteElement ( );
    SPH_executeStructure (info_objetos[ID].struct_id);
    SPH_closeStructure ( );
    /* mostrar el cuadro: */
    SPH_regenerateScreen ( );
}

```

## 7.11 Temas avanzados

Hay varios aspectos adicionales relacionados con SPHIGS y PHIGS estándar que no es apropiado tratar con detalle en este libro. Resumiremos algunos de los más importantes, pero para un tratamiento completo deberá consultar el capítulo 7 de [FOLE90].

### 7.11.1 Características de salida adicionales

**“Manojos” de atributos.** PHIGS estándar incluye un mecanismo para establecer valores de atributos en forma indirecta. Una aplicación puede, durante la secuencia de asignación de valores iniciales, almacenar una colección de valores de atributos en un **“manojo” de atributos**. Se pueden cambiar las definiciones de los “manojos” en la tabla sin alterar las redes de estructuras, lo que representa una forma sencilla de cambiar dinámicamente la apariencia de los objetos.

**Conjuntos de nombres para realce e invisibilidad.** SPHIGS permite usar dos técnicas de realimentación tradicional que las aplicaciones suelen usar junto con el recurso de selección de SPHIGS: realzar objetos y hacerlos invisibles. La primera técnica se emplea generalmente para proporcionar realimentación cuando el usuario selecciona un objeto; la segunda sirve para “asear” la pantalla y presentar sólo el detalle deseado. En el manual de referencia de SPHIGS se

describen estructuras de elementos que, al ejecutarse durante el recorrido, añaden o eliminan nombres en el conjunto de nombres.

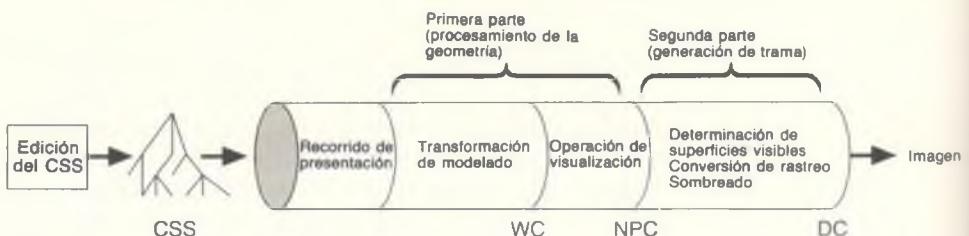
**Intercambio de imágenes y metaarchivos.** Aunque PHIGS y otros paquetes gráficos estándar son independientes del sistema y de los dispositivos para ofrecer transportabilidad, la implantación de uno de estos paquetes en un ambiente específico seguramente se optimizará en forma no portátil por cuestiones de rendimiento. El archivo de respaldo de PHIGS, definido por el comité de normas gráficas, es una fotografía portátil de la base de datos de estructuras en un instante determinado y permite que diferentes implantaciones de PHIGS comparan modelos geométricos. Las implantaciones de PHIGS también pueden permitir la escritura de un metaarchivo, el cual puede contener una fotografía de lo que la aplicación presenta en ese momento en la pantalla.

### 7.11.2 Aspectos de implantación

Para presentar una red de estructuras, SPHIGS visita los elementos que la componen usando un recorrido descendente, por profundidad, y lleva a cabo la acción indicada para cada elemento con base en su tipo. Este proceso de presentación, que establece una correspondencia entre un modelo y una imagen en la pantalla (o impresa), se conoce como recorrido de presentación en el contexto de PHIGS, pero más comúnmente como **generación de imágenes**; su implantación en software y hardware se conoce como **ducto de generación de imágenes**.

**Recorrido.** Es bastante difícil optimizar la regeneración para recorrer la menor porción posible del CSS, ya que el efecto de una operación trivial puede ser enorme. Por ejemplo, es difícil determinar cuánto hay que regenerar en la pantalla debido a la edición de una estructura.

**Generación de imágenes.** El ducto de generación de imágenes conceptual que implanta el recorrido de presentación se ilustra en la figura 7.18. Su primera etapa es el recorrido por profundidad del CSS. (Alternativamente, si se emplea un paquete gráfico de modo inmediato, la aplicación puede recorrer el modelo de la aplicación o generar primitivas o atributos por medio de procedimientos.) Cada primitiva que se detecte durante el recorrido se envía a lo largo del ducto: primero se aplican las transformaciones de modelado (descritas en el capítulo 5) para establecer la correspondencia entre las coordenadas de modelado de la primitiva y las coordenadas de mundo. Después se aplica la operación de vista para transformar y recortar la primitiva al volumen de vista canónico, y luego establecer la correspondencia con el paralelepípedo NPC (descrito en el capítulo 6). Como estos procesos son independientes del dispositivo de presentación y se relacionan con la geometría de vértices en coordenadas de punto flotante, esta parte del ducto que está inmediatamente después del recorrido se conoce como subsistema de procesamiento de geometría.



**Figura 7.18** Ducto de generación de imágenes de SPHIGS.

La segunda porción del ducto recibe las primitivas transformadas y recortadas y produce pixeles; este procesamiento de los pixeles se conoce como *generación de trama*. Por supuesto, el proceso es bastante sencillo para el modo alambrado: se establece la correspondencia de las coordenadas NPC (por medio del escalamiento y la traslación, ignorando z) a coordenadas enteras del dispositivo y se invoca la función de dibujo de líneas del paquete gráfico de trama subyacente para efectuar el proceso de discretización. Sin embargo, la generación de imágenes sombreadas es bastante compleja y se divide en tres subprocesos: **determinación de superficies visibles** (determinar cuáles son las partes de una primitiva que realmente son visibles desde el punto de vista de la cámara sintética), **discretización** (determinar los pixeles que estarán cubiertos por la imagen de la primitiva) y **sombreado** (determinar qué color se asignará a cada uno de los pixeles cubiertos). El orden exacto de ejecución de estos subprocesos varía de acuerdo con el modo de generación de imágenes y el método de implantación. En los capítulos 12 y 14 se presentan descripciones detalladas de los subprocesos de generación de trama.

**Optimización por medio de revisión de extensiones.** La estrategia de recorrido que hemos presentado transita incondicionalmente por todo el contenido de la red. Sin embargo, muchas veces no todos los objetos de la red son visibles, ya que las transformaciones de modelado y visualización vigentes al realizar el recorrido pueden ocasionar que gran parte de la red se encuentre fuera del volumen de vista. Para implantar una optimización necesitamos un método sencillo con el cual calcular los límites de un objeto arbitrariamente complejo, y una forma eficiente de comparar estos límites con el área de vista NPC.

**Animación y memoria doble.** Un efecto secundario de algunas implantaciones sencillas de SPHIGS es que entre los *cuadros* de una animación, los observadores ven que se borra la pantalla y pueden ver (más o menos) cómo se dibujan los objetos mientras el recorrido lleva a cabo la regeneración. Este artefacto visual se puede reducir usando memoria doble: un lienzo o arreglo bidimensional de bits fuera de pantalla para almacenar el siguiente cuadro mientras se dibuja.

luego cuando termina la regeneración, el contenido de este lienzo se copia a la pantalla.

**Correlación de selección.** En la correlación de selección, SPHIGS recorre las redes colocadas en el área de vista en la cual se ha especificado el punto NPC. El recorrido es casi idéntico al que se efectúa durante la presentación: se mantienen las matrices de transformación de modelado y se lleva a cabo la mayor parte del ducto de generación de imágenes. El recorrido de correlación de selección se puede optimizar en varias formas, incluyendo la detección analítica de aciertos y la detección de aciertos por medio de recortes. Un ejemplo del primer método es una función *punto\_en\_polígono*, que se usa para probar los aciertos en áreas de relleno y facetas en modo de generación sombreados. Un algoritmo común para determinar si la posición del cursor NPC está dentro de un polígono, basado en la regla de paridad impar descrita en la sección 2.1.3, lanza un rayo desde la posición del localizador y determina cuántas veces el rayo interseca el polígono. El algoritmo recorre la lista de aristas y determina si hay intersecciones y casos especiales (p. ej., intersecciones en vértices, colinealidad arista-rayo). El algoritmo de discretización de polígonos presentado en la sección 3.5 resuelve un problema muy similar y puede adaptarse como función *punto\_en\_polígono*.

### 7.11.3 Optimización de la presentación de modelos jerárquicos

**Elisión.** Podemos modelar un edificio como una jerarquía de partes si decimos que consiste en pisos, los pisos consisten en oficinas, etc.; no hay primitivas en los nodos de la jerarquía hasta llegar al nivel de los ladrillos, tablas y lozas de concreto que consisten en poliedros. Aunque esta representación puede ser útil para fines de construcción, no lo es para la presentación, ya que en ocasiones queremos ver una imagen más cruda y simplificada que elimine los detalles confusos e innecesarios. El término *elisión* se refiere a la decisión que se toma durante el recorrido de presentación para omitir el descenso a una subestructura. La elisión se puede efectuar con **poda**, **eliminación de superficies** y considerando el **nivel de detalle**.

**Referencia a la estructura.** Ciertas implantaciones de PHIGS y PHIGS PLUS permiten una forma no estándar de ejecución de estructuras, llamada **referencia**, que omite los costosos procesos de almacenamiento y restauración del mecanismo de ejecución de estructuras. Una optimización consiste en añadir una operación de referencia a estructuras que pueda usar el programador en aquellos casos en que una estructura hija invocada no tenga elementos de asignación de atributos.

### 7.11.4 Limitaciones del modelado jerárquico en PHIGS

**Limitaciones de la jerarquía simple.** Como mencionamos en la sección 1.3, algunas aplicaciones no tienen una estructura real para sus datos (p. ej., datos

para gráficos de dispersión) o a lo sumo tienen un ordenamiento (parcial) de los datos (p. ej., una función representada en forma algebraica). Otras aplicaciones se pueden expresar en forma más natural como redes, es decir, como grafos (dirigidos) generales (que pueden tener subredes jerárquicas). Entre éstas se incluyen los diagramas de circuitos y de alambrado eléctrico, las redes de comunicación y transporte, y los diagramas de tuberías de plantas químicas. Otro ejemplo de la insuficiencia de una jerarquía simple para ciertos tipos de modelos es el cubo de Rubik, una colección de componentes en la cual la red y la jerarquía (digamos de niveles, filas y columnas) es alterada fundamentalmente después de una transformación.

En otros tipos de modelos no es suficiente una sola jerarquía. Por ejemplo, el sujetador de pluma en un graficador ( $x, y$ ) es movido por el brazo vertical y el horizontal, y por ende “pertenece” a ambos. En resumen, cuando el modelo de aplicación presenta jerarquía pura, red pura sin jerarquía, jerarquía en una red con enlaces cruzados o jerarquías múltiples, se puede usar SPHIGS para presentarlo pero quizás no sea posible o deseable aprovechar toda la generalidad de la jerarquía de estructuras.

**Limitaciones de SPHIGS en el “pase de parámetros”.** La naturaleza de caja negra de las estructuras es buena para la modularidad pero, como vimos en el ejemplo del robot, puede ser limitante. Por ejemplo, ¿cómo construimos un robot con dos brazos idénticos y logramos que el robot use el brazo derecho para recoger un cilindro de una mesa y que se lleve el cilindro? La jerarquía de estructuras no apoya el uso de varios ejemplares de estructuras que difieren en cuanto a la configuración de las transformaciones en diversos niveles jerárquicos, ya que la jerarquía no tiene el mecanismo general de pase de parámetros de la jerarquía de funciones ni construcciones generales de flujo de control.

### 7.11.5 Formas alternativas de modelado jerárquico

**Jerarquía de funciones.** En el espectro que abarca desde la jerarquía pura de datos hasta la jerarquía pura de funciones, la jerarquía de estructuras está casi en el extremo de los datos, ya que carece de un flujo de control general. En cambio, una función de plantilla (es decir, una función que define un objeto de plantilla, consistente en primitivas o llamadas a una función de plantilla subordinada) puede usar parámetros y flujo de control arbitrario.

**Jerarquía de datos.** A diferencia de la jerarquía de funciones, la jerarquía de datos es apropiada para la creación dinámica. Como la jerarquía de funciones de plantilla, se puede usar en conjunto con paquetes gráficos de modo inmediato o retenido.

**Utilización de sistemas de base de datos.** Como una base de datos de propósito general tiene más poder que una de propósito especial, debemos considerar la utilización de sistemas estándar de base de datos para la graficación por

computador. Por desgracia, estas bases de datos están diseñadas para operar con grandes volúmenes de datos en almacenamiento secundario y para proporcionar tiempos de respuesta medidos en una escala temporal humana.

### 7.11.6 Otros estándares (industriales)

En la sección 7.1 declaramos que muchas de las características analizadas en el contexto de PHIGS también están disponibles en sus competidores, como OpenGL y HOOPS. En esta sección veremos brevemente las diferencias entre estos paquetes y la forma en que PEX se relaciona con PHIGS.

En primer lugar es necesario distinguir entre los protocolos gráficos tridimensionales de cliente/servidor y las API por procedimientos como PHIGS que se emplean para implantar aplicaciones. Las API, es decir, las bibliotecas o paquetes a nivel de aplicación, usan los protocolos cliente/servidor de nivel inferior con el fin de implantar la intercomunicación necesaria para la computación distribuida. El sistema X Window (X) ha sido por mucho tiempo el estándar industrial para aplicaciones gráficas bidimensionales que operan en un ambiente de cliente-servidor. Muchas veces los procesos cliente y servidor se ejecutan en máquinas diferentes conectadas por una red local (LAN) o de mayor alcance (WAN). En X, el proceso servidor administra la presentación y el proceso cliente contiene el código de la aplicación y el paquete gráfico; los dos procesos se comunican a través de un protocolo de comunicación entre procesos (IPC, *interprocess communications protocol*) que tiene la forma de secuencias de mandatos, cada uno de los cuales consiste generalmente en una operación y sus parámetros.

PEX [originalmente una abreviatura de “PHIGS Extensions to X” (extensiones de PHIGS para X), pero ahora no tan centrado en PHIGS] es la extensión gráfica tridimensional distribuida para el sistema X Window. PEX sólo maneja la *salida*; la entrada es manejada por el protocolo estándar de entrada de X.

PHIGS, HOOPS y PEXlib son las API más comunes que se basan en el protocolo PEX. PEXlib, por analogía con Xlib para los gráficos bidimensionales, es un *revestimiento delgado* encima del protocolo PEX que ofrece acceso a toda la funcionalidad del protocolo pero no permite un gran nivel de abstracción. PHIGS y HOOPS ocultan la funcionalidad de bajo nivel pero son menos eficientes que las llamadas directas de PEXlib. OpenGL no reside sobre PEX, sino que incluye su propio protocolo para los gráficos distribuidos.

OpenGL es una API exclusiva para la generación de imágenes, independiente de la marca de equipo, que ofrece funciones gráficas bidimensionales y tridimensionales, incluyendo modelado, transformaciones, color, iluminación, sombreado suave y varias características avanzadas, como la correspondencia (*mapping*) de texturas (Sec. 14.3), NURBS (Sec. 9.2) y borrosidad causada por movimiento (Sec. 12.4). Sus características de generación de imágenes son más avanzadas que las de PHIGS PLUS, ya que originalmente eran apoyadas por el hardware de Silicon Graphics. OpenGL, como PEX, permite

usar los modos gráficos inmediato y retenido. Es independiente del sistema operativo y del sistema de ventanas y ha sido integrado con el sistema X Window en Unix.

HOOPS difiere de PHIGS en varios aspectos importantes. Como ha sido diseñado e implantado por una sola compañía, todas las implantaciones son literalmente compatibles, algo que no sucede con las implantaciones comerciales de PHIGS. Los estándares oficiales ofrecen demasiada flexibilidad para las implantaciones individuales.

HOOPS ofrece un apoyo más completo de familias tipográficas, datos de imágenes y generación de imágenes avanzada que PHIGS. Además de las capacidades de generación ofrecidas por PHIGS PLUS, HOOPS también permite varias formas de modelos de iluminación global, incluyendo la traza de rayos, la radiosidad y un generador combinado de traza de rayos-radiosidad (véase el Cap. 14).

Como PHIGS, HOOPS permite usar la jerarquía de estructuras, pero con una diferencia fundamental. Las estructuras de PHIGS contienen listas dependientes del orden de los elementos y atributos gráficos. Para la edición de estas listas se requiere que el programador tenga un conocimiento detallado de cómo y cuándo se asignan y modifican los atributos. En cambio, todas las primitivas en un *segmento* de HOOPS tienen el mismo estado de atributo, descrito en el encabezado del segmento. Esto simplifica la tarea del programador y produce modelos más modulares, independientes del orden, además de permitir una implantación de mayor rendimiento, ya que se simplifican los aspectos administrativos y se reducen los cambios de contexto en el ducto de generación.

## RESUMEN

En este capítulo presentamos una introducción general a los modelos geométricos, destacando los modelos jerárquicos que representan el montaje de partes. Aunque muchos tipos de datos y objetos no son jerárquicos, la mayoría de los objetos de fabricación humana lo son en cierta medida. PHIGS y sus alternativas (p. ej., OpenGL, HOOPS y PEXlib) ofrecen un alto nivel de funcionalidad a expensas de una considerable complejidad. Proporcionan recursos para el modelado geométrico jerárquico con polígonos, poliedros, curvas y superficies, y pueden coexistir en forma pacífica con administradores de ventanas y la computación distribuida cliente/servidor, como la que ofrece el sistema X Window.

SPHIGS, un subconjunto de PHIGS, está diseñado para proporcionar representaciones eficientes y naturales de objetos geométricos almacenados esencialmente como jerarquías de polígonos y poliedros. Como estos paquetes almacenan una base de datos interna de los objetos, un programador puede efectuar sin mucho esfuerzo ligeras modificaciones a la base de datos para que el paquete produzca automáticamente una vista actualizada. De esta manera, el programa de aplicación construye y edita la base de datos, por lo general en respuesta a una entrada del usuario, y el paquete es el responsable de producir las vistas especificadas por la base de datos. Estas vistas usan diversas técnicas de generación de imágenes para conciliar entre la calidad y la velocidad. El

paquete también ofrece dispositivos de entrada de tipo localizador y de opciones, así como correlación de selección para permitir la elección de objetos en cualquier nivel de la jerarquía. Se pueden usar filtros de realce y visibilidad para activar y desactivar selectivamente como otra forma de control de la apariencia de los objetos.

Debido a la naturaleza restringida de las estructuras y a los limitados medios para buscarlas y editarlas, un sistema de propósito especial como éste es más apropiado para la dinámica de movimiento y la dinámica de actualización de luces, sobre todo si la base de datos de estructuras se puede mantener en un subsistema de presentación optimizado para servir como periférico de SPHIGS. Si hay que actualizar gran parte de la base de datos de estructuras entre imágenes sucesivas o si se puede recorrer rápidamente la base de datos de la aplicación sin que exista un cuello de botella entre el computador y el subsistema de presentación, es más eficiente utilizar un paquete gráfico en modo inmediato, sin retener información.

La jerarquía de estructuras es intermedia entre la jerarquía de datos pura y la de funciones pura. Tiene la ventaja de la edición dinámica, característica de las jerarquías de datos. También permite una forma sencilla de pase de parámetros a las subestructuras (de atributos geométricos o de apariencia), usando el mecanismo de estado de recorrido de atributos. Sin embargo, por la carencia de instrucciones generales de flujo de control, el mecanismo para pasar parámetros es limitado y no se pueden asignar selectivamente atributos diferentes de las estructuras en ejemplares distintos de una subestructura. En cambio, se pueden emplear funciones de plantilla para establecer varias copias de estructuras (jerárquicas) con estructura idéntica pero diferentes en cuanto a los atributos geométricos o de apariencia de las subestructuras. Alternativamente, se pueden usar para alimentar a un paquete de modo inmediato.

SPHIGS está orientado a modelos geométricos formados esencialmente por polígonos y poliedros, sobre todo aquellos que presentan jerarquía; en los capítulos 9 y 10 veremos los modelos geométricos que tienen primitivas más complejas y combinaciones de primitivas. Sin embargo, antes de pasar a estos temas más avanzados del modelado veremos las interfaces con el usuario, las técnicas y las herramientas para la interacción.

## Ejercicios

7.1 a. Complete el modelo del robot de la figura 7.11, añadiendo una base sobre la cual se mueva el cuerpo, y cree una animación simple de su movimiento por una habitación.

b. Cree una aplicación de SPHIGS que produzca una animación en la cual un robot de un solo brazo se aproxime a una mesa donde yace un objeto, recoja el objeto y se aleje con él.

7.2 Mejore la animación de un robot para que se presenten simultáneamente tres vistas de la animación, incluyendo una vista ortográfica superior y una vista desde el *ojo del robot* que muestre lo que ve el robot al moverse.

7.3 Actualice el recorrido de presentación recursivo para que mantenga la información de extensión en coordenadas de modelado para cada estructura. Suponga que al cerrar una estructura  $S$  tras la edición, se asigna un campo *extensión\_obsoleta* de tipo booleano en el registro de  $S$ . Suponga también que hay funciones que, dada una primitiva, devuelven la extensión NPC de la primitiva.

7.4 Diseñe un algoritmo para calcular analíticamente el punto de acierto para una línea candidata, dados los puntos extremos NPC de la línea y la medida del localizador.

7.5 Diseñe un algoritmo para calcular analíticamente el punto de acierto para un área de relleno candidata.

7.6 Diseñe, usando seudocódigo, un recorrido recursivo de correlación de selección que sólo apoye el modo alambrado.

7.7 Implante la función *punto\_en\_polígono* para usarla en la correlación de selección. Trate los casos especiales de rayos que pasan por los vértices o que coinciden con las aristas. Consulte [PREP85] y [FORR85] para conocer más acerca de los puntos finos de este problema.

7.8 Diseñe una interfaz de selección que permita al usuario indicar el nivel deseado de una jerarquía. Implante y pruebe su interfaz con el modelo de robot, escribiendo una aplicación que permita al usuario realizar porciones de la anatomía del robot, desde partes individuales hasta subsistemas completos.

Las interfaces de alta calidad con el usuario son, en muchas formas, la *frontera final* al ofrecer la computación a una amplia diversidad de usuarios, ya que los costos del hardware y el software son ahora suficientemente bajos para proporcionar capacidad de computación a los hogares y las oficinas. Así como la ingeniería de software ha creado recientemente la estructura para una actividad que antes se realizaba por completo en forma *ad hoc*, la nueva área de la ingeniería de interfaces con el usuario genera los principios y las metodologías de diseño de las interfaces.

La calidad de la interfaz con el usuario muchas veces determina si el usuario acepta o rechaza un sistema, si los diseñadores del sistema son elogiados o reprobados y si un sistema tiene éxito o fracasa en el mercado. El diseñador de una aplicación gráfica interactiva debe tener presente el deseo del usuario de contar con una interfaz fácil de aprender pero a la vez poderosa.

La metáfora del escritorio para interfaces con el usuario, con sus ventanas, iconos y menús descendentes, todo lo cual usa extensivamente gráficos de barriado, se ha vuelto popular porque es sencilla de aprender y no requiere muchas habilidades de teclado. Muchos usuarios de estos sistemas no son programadores de computadores y tienen poca simpatía por las interfaces antiguas, difíciles de aprender, de lenguaje de mandatos tecleados, que muchos programadores consideran algo natural. El proceso de diseño, pruebas e implantación de una interfaz con el usuario es algo complejo; en [FOLE90; SHNE86; MAYH90] se presentan pautas y metodologías.

En este capítulo nos centramos en los dispositivos de entrada, las tecnologías de interacción y las tareas de interacción, que constituyen los bloques básicos para construir interfaces con el usuario. Los dispositivos de entrada son pie-

zas de hardware que el usuario usa para introducir información a un sistema de computación. Ya analizamos varios de estos dispositivos en el capítulo 4. En este capítulo presentaremos otros dispositivos y analizaremos las razones para preferir un dispositivo en lugar de otro. En la sección 8.1.6 describiremos los dispositivos de entrada orientados especialmente a la interacción tridimensional. Seguimos usando las categorías de dispositivos lógicos de localizador, teclado, de opción, valuador y selección utilizados por SRGP, SPHIGS y otros paquetes gráficos independientes del dispositivo. También analizaremos los elementos básicos de las interfaces con el usuario: las **técnicas de interacción** y las **tareas de interacción**. Las técnicas de interacción son formas de usar los dispositivos de entrada para registrar información en el computador, mientras que las tareas de interacción clasifican los tipos fundamentales de información que se registran con las técnicas de interacción. Estas técnicas son los bloques básicos primitivos para armar una interfaz con el usuario.

Una **tarea de interacción** es el registro de una información por parte del usuario. Las cuatro tareas de interacción básicas son **posicionamiento, texto, selección y cuantificación**. La unidad de información que se registra en una tarea de interacción de posicionamiento es, por supuesto, una posición. En forma similar, la tarea de texto produce una cadena de texto; la tarea de selección produce la identificación de un objeto; y la tarea de cuantificación produce un valor numérico. Para una tarea de interacción se pueden emplear distintas **técnicas de interacción**. Por ejemplo, una tarea de selección se puede realizar con el ratón para seleccionar elementos de un menú, con un teclado para especificar el nombre de la selección, oprimiendo una tecla de función o utilizando un reconocedor de voz. En forma semejante, se puede utilizar un mismo dispositivo para varias tareas: con frecuencia se emplea el ratón para posicionamiento y selección.

Las tareas de interacción son diferentes de los dispositivos lógicos de entrada que analizamos en capítulos anteriores. Estas tareas se definen de acuerdo con *qué logra el usuario*, mientras que los dispositivos lógicos de entrada categorizan *cómo* se lleva a cabo la tarea a través del programa de aplicación y el paquete gráfico. Las tareas de interacción se centran en el usuario, mientras que los dispositivos lógicos de entrada son un concepto de programación y del paquete gráfico.

Varios de los temas en este capítulo se analizan con mayor detalle en otras obras; consulte los libros de Baecker y Buxton [BAEC87], Hutching, Hollan y Norman [HUTC86], Mayhew [MAYH90], Norman [NORM88], Rubenstein y Hersh [RUBE84], Shneiderman [SHNE86] y [FOLE90]; el libro de consulta de Salvendy [SALV87]; y el estudio de Foley, Wallace y Chan [FOLE84].

## 8.1 Hardware de interacción

Aquí presentamos algunos de los dispositivos de interacción que no se analizaron en la sección 4.5, detallamos su funcionamiento y analizamos las ventajas y

desventajas de los diversos dispositivos. La presentación se organiza con respecto a la categorización de dispositivos lógicos de la sección 4.5 y se puede considerar como una continuación más detallada de lo que se presentó en esa sección.

Las ventajas y desventajas de los diversos dispositivos de interacción se pueden analizar en tres niveles: dispositivo, tarea y diálogo (es decir, la secuencia de varias tareas de interacción). El **nivel de dispositivo** se centra en las características de hardware y no trata los aspectos de la utilización del dispositivo controlados por el software. Por ejemplo, en el nivel de dispositivo mencionamos que una forma de ratón puede ser más cómoda de usar que otra, y que una tableta de datos ocupa más espacio que una palanca de mandos.

En el **nivel de tarea** podemos comparar las técnicas de interacción que utilizan diferentes dispositivos para la misma tarea. De esta manera podemos determinar que los usuarios experimentados muchas veces pueden registrar mandatos con mayor rapidez a través de teclas de función o del teclado que usando la selección de menús, o que los usuarios pueden seleccionar objetos presentados más rápidamente si usan un ratón en lugar de una palanca de mandos o teclas de control del cursor.

En el **nivel de diálogo** no sólo consideramos las tareas de interacción individuales, sino también las secuencias de dichas tareas. Los movimientos de la mano de un dispositivo a otro requieren tiempo: aunque la tarea de posicionamiento por lo general es más rápida con un ratón que con teclas de control del cursor, éstas pueden ser más rápidas que el ratón *si* las manos del usuario ya están sobre el teclado y tienen que permanecer allí para la siguiente tarea de la secuencia después de reubicar el cursor.

Algunos de los aspectos importantes a nivel de dispositivo, los cuales veremos en esta sección, son las huellas de los dispositivos (la **huella** de una pieza de equipo es el área de trabajo que ocupa), la fatiga del operador y la resolución del dispositivo. Otros asuntos importantes relacionados con los dispositivos, como el costo, la confiabilidad y la facilidad de mantenimiento, cambian con demasiada rapidez gracias a los avances tecnológicos, y no los consideraremos aquí.

### 8.1.1 Dispositivos localizadores

Es conveniente clasificar los dispositivos localizadores de acuerdo con tres características importantes: absolutos o relativos, directos o indirectos, y discretos o continuos.

Los dispositivos **absolutos**, como la tableta de datos y la pantalla sensible al tacto, tienen un marco de referencia, u origen, e informan de las posiciones de acuerdo con ese origen. Los dispositivos **relativos**, como los ratones, las bolas rastreadoras (*trackballs*) y las palancas de control de velocidad, no tienen origen absoluto y sólo informan de los cambios con respecto a su posición anterior. Se puede usar un dispositivo relativo para especificar un cambio arbitrariamente grande en la posición: el usuario puede mover el ratón sobre el escritorio, levantarla y colocarlo de nuevo en su posición inicial, para luego moverlo una vez más. Se puede programar una tableta de datos para que se comporte como dis-

positivo relativo: la primera coordenada ( $x, y$ ) que se lee después de mover el lápiz de un estado *lejano* a uno *cercano* (es decir, cercano a la tableta) se resta de las coordenadas leídas subsecuentemente para proporcionar sólo los cambios en  $x$  y en  $y$ , los cuales se suman a la posición ( $x, y$ ) previa. Este proceso continúa hasta que el lápiz regresa a un estado *lejano*.

No es fácil usar los dispositivos relativos para digitalizar imágenes, pero si los absolutos. La ventaja de un dispositivo relativo es que el programa de aplicación puede reubicar el cursor en cualquier lugar de la pantalla.

Con un dispositivo **directo**, como una pantalla sensible al tacto, el usuario apunta directamente a la pantalla con un dedo o un dispositivo similar; con un dispositivo **indirecto**, como una tableta, un ratón o una palanca de mandos, el usuario mueve un cursor en la pantalla usando un dispositivo que no está en la pantalla. Para el segundo tipo es necesario aprender nuevas formas de coordinación motriz; sin embargo, la proliferación de juegos de video en los hogares y en centros comerciales ha creado un ambiente en el cual muchos usuarios casuales de los computadores ya han desarrollado estas habilidades. No obstante, apuntar directamente puede cansar el brazo, sobre todo entre usuarios casuales.

Un dispositivo **continuo** es aquel en el cual un movimiento regular de la mano puede crear un movimiento regular del cursor. Las tabletas, las palancas de mandos y los ratones son dispositivos continuos, mientras que las teclas de control del cursor son dispositivos **discretos**. Los dispositivos continuos generalmente permiten un movimiento más natural, sencillo y rápido del cursor que los discretos. Además, la mayoría de los dispositivos continuos permite efectuar movimientos en direcciones arbitrarias con mayor facilidad que las teclas de control del cursor.

La velocidad de posicionamiento del cursor con un dispositivo continuo se ve afectada por la **razón entre el control y la presentación**, comúnmente conocida como **razón C/D (control-to-display ratio)** [CHAP72]; se trata de la razón entre el movimiento de la mano (el control) y el movimiento del cursor (la presentación). Una razón elevada es buena para una ubicación precisa, pero hace tediosos los movimientos rápidos; una razón pequeña es buena para la velocidad pero inconveniente para la precisión. Por fortuna, en un dispositivo de posicionamiento relativo no es necesario que la razón sea constante, y esta puede adaptarse como función de la velocidad del movimiento. Los movimientos rápidos indican que el usuario está realizando un movimiento abrupto y se usa una razón pequeña; al reducir la velocidad, la razón C/D aumenta. Esta variación de la razón C/D se puede establecer de manera que los usuarios puedan emplear el ratón para ubicar el cursor con precisión en una pantalla de 15 pulgadas ¡sin tener que mover toda la muñeca! En el caso de los dispositivos discretos indirectos (teclas de control del cursor) se utiliza una técnica similar: la distancia que se mueve el cursor en una unidad de tiempo aumenta como función del tiempo que se ha mantenido presionada la tecla.

El posicionamiento preciso con los dispositivos directos se dificulta si el brazo no tiene apoyo y se extiende directamente hacia la pantalla. Trate de escribir su nombre en un pizarrón de esta manera y compare el resultado con la rúbrica habitual. Este problema se puede reducir si la pantalla se encuentra

ángulo casi horizontal. Por otra parte, los dispositivos indirectos permiten que la mano se apoye, y es posible aplicar un control motriz más fino de los dedos. Sin embargo, no todos los dispositivos indirectos continuos son igualmente buenos para dibujar. Trate de escribir su nombre con una palanca de mandos, un ratón y el lápiz de una tableta. Con el lápiz se obtienen los resultados más rápidos y agradables.

### 8.1.2 Dispositivos de teclado

El conocido teclado QWERTY (este nombre se refiere a las teclas del primer renglón del teclado usado normalmente) ha estado presente durante muchos años. Lo irónico es que este teclado se diseñó originalmente para *reducir la velocidad* de los mecanógrafos, de modo que no fuera tan probable que se atasaran los martillos de las máquinas de escribir. Varios estudios han demostrado que el teclado Dvorák [DVOR43], que coloca las vocales y otros caracteres de uso frecuente cerca de los lugares donde descansan los dedos, es un poco más rápido que el diseño QWERTY [GREE87]; sin embargo, no ha sido muy aceptado. Los teclados organizados en forma alfabética se usan en algunas ocasiones cuando los usuarios no saben teclear. No obstante, cada vez más personas utilizan los teclados QWERTY, y diversos experimentos no han mostrado ventajas de los teclados alfabéticos sobre los QWERTY [HIRS70; MICH71].

Otros aspectos relacionados con el teclado, que tienen que ver con el diseño del software y no del hardware, son arreglos que permiten al usuario registrar caracteres de puntuación o corrección de uso frecuente sin necesidad de oprimir simultáneamente las teclas de control o de mayúsculas, además de asignar las acciones peligrosas (como la eliminación) a teclas que están lejos de las de uso frecuente.

### 8.1.3 Dispositivos valuadores

Algunos valuadores están **limitados**, como el control de volumen en una radio: la perilla sólo puede girar hasta cierto punto antes de que se detenga el movimiento. Un valuador limitado introduce una cantidad absoluta. Por otra parte, un potenciómetro de giro continuo puede girar un número **ilimitado** de veces en cualquier dirección. Si se parte de un valor inicial, el potenciómetro ilimitado se puede usar para producir valores absolutos; en caso contrario, los valores recibidos se consideran relativos. Si se ofrece algún tipo de eco, el usuario puede determinar cuál es el valor relativo o absoluto que se especifica. El tema de la razón C/D, analizado en el contexto de los dispositivos de posicionamiento, también está presente al utilizar potenciómetros deslizantes y giratorios para introducir valores.

### 8.1.4 Dispositivos de opción

Las teclas de función constituyen un dispositivo de opción común. Su colocación afecta la facilidad de uso. Las teclas montadas en la orilla de la pantalla

son más difíciles de usar que aquellas montadas en el teclado o en una unidad separada pero cercana. En las aplicaciones donde ambas manos del usuario están ocupadas pero se requiere cerrar con frecuencia un conmutador, puede utilizarse un conmutador de pie.

### 8.1.5 Otros dispositivos

Ahora analizaremos algunos de los dispositivos de interacción bidimensional menos comunes y, en algunos casos, experimentales. Los reconocedores de voz, útiles porque liberan las manos para otras tareas, aplican un método de reconocimiento de patrones a las formas de onda creadas al hablar. La forma de onda se separa en varias bandas de distinta frecuencia, cuya variación en magnitud con el paso del tiempo en cada banda forma la base para la equivalencia de patrones. Sin embargo, pueden ocurrir errores en la comparación de patrones, por lo que es muy importante que la aplicación que utilice un reconocedor de voz incluya capacidades de corrección.

Los reconocedores de voz difieren en lo que se refiere a la necesidad de entrenarlos para que reconozcan las formas de onda de una persona en particular, o en cuanto a su capacidad de reconocer el habla compuesta y no sólo palabras o frases aisladas. Los reconocedores independientes de la persona que habla tienen vocabularios que incluyen los dígitos y hasta 1000 palabras.

La tableta de datos se ha extendido en varias formas. Hace muchos años, Herot y Negroponte usaron un lápiz experimental sensible a la presión [HERO76]: si la presión era alta y la velocidad de dibujo lenta, esto implicaba que el usuario dibujaba una línea con cuidado, en cuyo caso la línea se registraba tal y como se dibujaba; si la presión era baja y la velocidad alta, esto implicaba que la línea se dibujaba con rapidez, y entonces se registraba una línea recta que conectaba los puntos extremos. Una tableta comercial disponible en años recientes [WACO93] incorpora este lápiz sensible a la presión. Los tres grados de libertad que indica la tableta se pueden usar de varias maneras creativas.

### 8.1.6 Dispositivos de interacción tridimensional

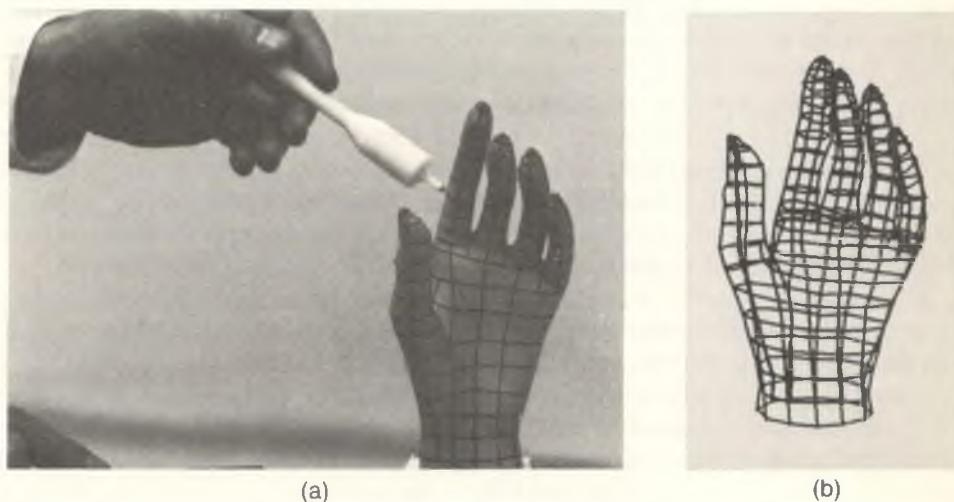
Algunos de los dispositivos de interacción bidimensional se pueden extender fácilmente a tres dimensiones. Las palancas de mando pueden tener un bastón que gire para indicar la tercera dimensión (véase la Fig. 4.15). Las bolas rastreadoras pueden detectar la rotación sobre el eje vertical además de los dos horizontales. Sin embargo, en ambos casos no hay una relación directa entre el movimiento de la mano y el movimiento correspondiente en el espacio tridimensional.

Muchos dispositivos pueden registrar movimientos manuales tridimensionales. Por ejemplo, el sensor de posición y orientación tridimensional Polhemus 3SPACE usa el acoplamiento electromagnético entre tres antenas transmisoras y tres antenas receptoras. Las bobinas de la antena transmisora, dispuestas en ángulos rectos entre sí para formar un sistema de coordenadas cartesianas, se pulsan en sucesión. El receptor tiene tres antenas receptoras dispuestas en forma

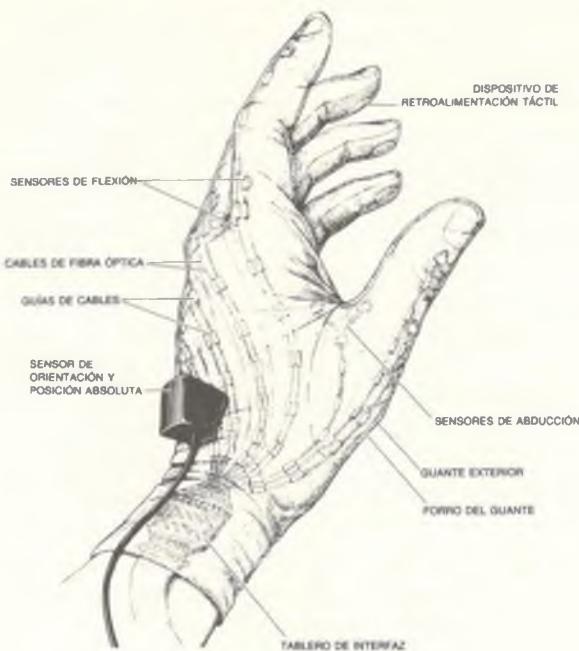
similar; cada vez que se pulsa una bobina transmisora se induce una corriente en las bobinas receptoras. La intensidad de la corriente depende de la distancia entre el receptor y el transmisor, así como de la orientación relativa de las bobinas transmisora y receptora. La combinación de los nueve valores de corriente inducidos por los tres pulsos sucesivos se emplea para calcular la posición y la orientación tridimensionales del receptor. En la figura 8.1 se muestra la utilización de este dispositivo en una de sus aplicaciones más comunes: la digitalización de un objeto tridimensional.

El DataGlove registra la posición y la orientación de la mano, así como los movimientos de los dedos. Como se ilustra en la figura 8.2, se trata de un guante cubierto con sensores pequeños y ligeros. Cada sensor es un pequeño cable de fibra óptica con un diodo emisor de luz (LED) en un extremo y un fototransistor en el otro. La superficie del cable es áspera en los lugares donde debe ser sensible a la flexión. Al flexionar el cable se pierde parte de la luz del LED, de manera que el fototransistor recibe menos luz. Además, un sensor Polhemus de posición y orientación registra los movimientos de la mano. Con el DataGlove, el usuario puede sujetar objetos, moverlos, hacerlos rotar y después soltarlos, con lo cual se obtiene una interacción tridimensional muy natural [ZIMM87]. La ilustración en color 6 muestra este concepto.

Se ha dedicado mucho esfuerzo para crear lo que se denomina **realidades artificiales** o **realidades virtuales**; se trata de ambientes completamente generados por computador con apariencia, comportamiento y técnicas de interacción realistas [FOLE87]. En una versión, el usuario usa una pantalla estéreo montada



**Figura 8.1** (a) Sensor de posición tridimensional Polhemus utilizado para digitalizar un objeto tridimensional. (b) Presentación alambrada del resultado. (Digitalizador 3Space cortesía de Polhemus, Inc., Colchester, Vermont.)



**Figura 8.2** El DataGlove VPL, que muestra los cables de fibra óptica utilizados para detectar los movimientos de los dedos, y el sensor de orientación y posición Polhemus. (De J. Foley, *Interfaces for Advanced Computing*, Copyright © 1987, *Scientific American, Inc.* Derechos reservados.)

en la cabeza donde se presentan las vistas apropiadas para el ojo derecho y el izquierdo; un sensor Polhemus en la cabeza hace que los cambios en la posición y la orientación de la cabeza ocasionen cambios en la pantalla estéreo, un DataGlove permite la interacción tridimensional y se usa un micrófono para emitir mandatos verbales. En la ilustración en color 7 se muestra esta combinación de equipo.

Para registrar posiciones tridimensionales se pueden utilizar varias tecnologías más. En una se usan sensores ópticos y se montan LED en el usuario (ya sea en un solo punto, como la punta del dedo, o en todo el cuerpo, para medir los movimientos corporales). Los sensores de luz se montan en las esquinas de una habitación pequeña, a media luz, en la cual trabaje el usuario, y se intensifican sucesivamente los LED. Los sensores pueden determinar el plano donde se encuentra el LED, y su ubicación es entonces la intersección de los tres planos. (Con frecuencia se usa un cuarto sensor, por si alguno de los sensores no puede ver el LED.) En lugar de los LED se pueden emplear pequeños reflectores en las puntas de los dedos o en otros puntos de interés; los sensores captan la luz reflejada en lugar de la emitida por el LED.

Krueger [KRUE83] desarrolló un sensor para registrar el movimiento bidimensional de manos y dedos. Una cámara de televisión registra el movimiento de la mano; después se emplean técnicas de procesamiento de imágenes de mejora de contraste y detección de aristas para encontrar el delineamiento de la mano y los dedos. Las distintas posiciones de los dedos se pueden interpretar como mandatos y el usuario puede sujetar y manipular objetos, como en la ilustración en color 8. Esta técnica se puede extender a tres dimensiones utilizando varias cámaras.

## 8.2 Tareas de interacción básicas

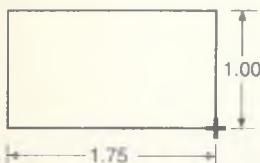
Con una tarea de interacción básica, el usuario de un sistema interactivo registra una unidad de información significativa en el contexto de la aplicación. ¿Cuál es la magnitud de esta unidad? Por ejemplo, ¿se registra una unidad de información al mover el dispositivo de posicionamiento una distancia pequeña? La respuesta es sí, si la nueva posición es utilizada por la aplicación, por ejemplo para reubicar un objeto o especificar el punto final de una línea. Empero, la respuesta también puede ser no si el cambio de posición es sólo parte de una secuencia de cambios conforme el usuario mueve el cursor para colocarlo sobre un elemento de menú; en este caso, la unidad de información es la opción del menú.

Las tareas de interacción básicas son indivisibles; es decir, si se dividieran en unidades de información más pequeñas, las unidades menores no tendrían sentido para la aplicación. En esta sección analizamos las tareas de interacción básicas. En la sección 8.3 veremos las tareas de interacción compuestas, que son agregados de las tareas de interacción básicas que describimos aquí. Si pensamos en las tareas básicas como átomos, entonces las tareas compuestas serían moléculas.

Un conjunto completo de tareas de interacción básicas para los gráficos interactivos está formado por la ubicación, la selección, el registro de texto y el registro de cantidades numéricas. En esta sección se describe cada una de las tareas de interacción básicas y se presentan algunas de las muchas técnicas de interacción para ellas. Sin embargo, la cantidad de técnicas de interacción es demasiado vasta para presentar una lista exhaustiva, y tampoco podemos anticipar el desarrollo de nuevas técnicas. Siempre que sea posible, veremos las ventajas y las desventajas de cada técnica; recuerde que una técnica de interacción específica puede ser buena en ciertas situaciones e inconveniente en otras.

### 8.2.1 Tarea de interacción para posicionamiento

La tarea de posicionamiento implica especificar una posición ( $x, y$ ) o ( $x, y, z$ ) al programa de aplicación. Las técnicas de interacción acostumbradas para esta tarea comprenden el movimiento de un cursor en la pantalla hasta la posición



**Figura 8.3**

Realimentación numérica que indica el tamaño de un objeto que se construye. La altura y la anchura cambian al mover el cursor (+), de manera que el usuario pueda ajustar el objeto al tamaño deseado.

deseada para luego oprimir un botón, o teclear las coordenadas deseadas de la posición en un teclado real o simulado. El dispositivo de posicionamiento puede ser directo o indirecto, continuo o discreto, absoluto o relativo. Además, los mandatos de movimiento del cursor se pueden teclear explícitamente en un teclado, como “arriba”, “abajo”, etc., o bien los mismos mandatos se pueden especificar en forma verbal a una unidad de reconocimiento de voz. Así mismo, es posible combinar estas técnicas: se puede usar un ratón para controlar el cursor y obtener una posición aproximada, para luego emplear las teclas de control de cursor y hacer avanzar éste una unidad de pantalla a la vez hasta llegar a la posición precisa.

Hay dos tipos de tareas de posicionamiento: espaciales y lingüísticas. En una tarea de posicionamiento **espacial**, el usuario sabe dónde está la posición deseada, por relación espacial con los elementos cercanos, como ocurre al dibujar una línea entre dos rectángulos o centrar un objeto entre otros dos. En una tarea de posicionamiento **lingüístico**, el usuario conoce los valores numéricos de las coordenadas ( $x, y$ ) de la posición. En el primer caso, el usuario quiere realimentación que le indique la posición real en la pantalla; en el segundo, se requieren las coordenadas de la posición. Si se proporciona la forma de realimentación incorrecta, el usuario tiene que efectuar una conversión mental. Es posible proporcionar ambos tipos de realimentación si se presenta el cursor y sus coordenadas numéricas, como en la figura 8.3.

### 8.2.2 Tarea de interacción para selección: conjunto de opciones de tamaño variable

La tarea de selección es aquella en la cual se elige un elemento de un **conjunto de opciones**. Los conjuntos de opciones típicos son mandatos, valores de atributos, clases de objetos y ejemplares de objetos. Por ejemplo, el menú de estilo de línea en un programa de pintura típico es un conjunto de valores de atributos, mientras que el menú de tipo de objeto (línea, círculo, rectángulo, texto, etc.) en estos programas es un conjunto de clases de objeto. Se pueden utilizar algunas técnicas de interacción para seleccionar de cualquiera de estos conjuntos de opciones; otras técnicas no son tan generales. Por ejemplo, podemos apuntar a la representación visual del elemento del conjunto para seleccionarlo, sin importar el tipo de conjunto. Por otra parte, aunque las teclas de función a menudo son muy prácticas para seleccionar de un conjunto de mandatos, clases de objetos, o atributos, es difícil asignar una tecla de función a cada ejemplar de objeto en un dibujo, ya que el tamaño del conjunto de opciones es variable, a menudo es grande (mayor que el número de teclas de función disponibles) y cambia con frecuencia conforme el usuario crea y elimina objetos.

Usaremos los términos *conjunto de opciones de tamaño (relativamente) fijo* y *conjunto de opciones de tamaño variable*. El primer término caracteriza los conjuntos de opciones de mandatos, atributos y clases de objetos; el segundo, los conjuntos de opciones de ejemplares de objetos. El modificador *relativamente* reconoce que cualquiera de estos conjuntos puede cambiar al definir nue-

vos mandatos, atributos o clases de objetos (como los símbolos en un sistema de bosquejo). Sin embargo, el tamaño del conjunto no cambia con frecuencia ni varía mucho. Por otra parte, los conjuntos de opciones de tamaño variable pueden ser muy grandes y cambiar con frecuencia.

En esta sección analizaremos las técnicas que se adaptan bien a los potencialmente grandes conjuntos de opciones de tamaño variable; estas técnicas incluyen especificar nombres y apuntar. En la sección 8.2.3 veremos las técnicas de selección apropiadas para los conjuntos de opciones de tamaño (relativamente) fijo. Estos conjuntos tienden a ser pequeños, excepto por los grandes (pero de tamaño relativamente fijo) conjuntos de mandatos que aparecen en las aplicaciones complejas. Las técnicas que se analizan incluyen: teclear o pronunciar el nombre, abreviatura u otro código que represente al elemento del conjunto; oprimir una tecla de función relacionada con el elemento del conjunto (este proceso equivale a teclear un carácter en el teclado); apuntar a una representación visual (textual o gráfica) del elemento del conjunto en un menú; desplazar el conjunto hasta que aparezca el elemento deseado; y efectuar un movimiento distintivo con un dispositivo de posicionamiento continuo.

**Selección de objetos por nombre.** El usuario puede teclear el nombre de la opción. La idea es sencilla, pero, ¿qué sucede si el usuario no conoce el nombre del objeto, algo que puede ocurrir fácilmente si se presentan cientos de objetos o si el usuario no tiene por qué conocer sus nombres? No obstante, esta técnica es útil en varias situaciones. En primer lugar, es probable que el usuario conozca los nombres de varios objetos, así como el comandante de una flota conoceería los nombres de sus navíos, por lo que es razonable referirse a ellos por nombre, lo cual puede ser más rápido que apuntar hacia ellos, sobre todo si el usuario tuviera que desplazarse por la pantalla para que apareciera el objeto deseado. En segundo lugar, si se dificulta seleccionar apuntando porque la pantalla está llena de objetos y no es factible realizar un acercamiento (quizá porque el hardware gráfico no lo permite y el acercamiento por software es demasiado lento), la especificación de nombres puede ser el último recurso. Si el amontonamiento es un problema, sería útil un mandato para activar o desactivar nombres de objetos.

El tecleado nos permite efectuar selecciones múltiples a través de caracteres comodines si los elementos del conjunto de opciones han sido nombrados de manera significativa. La selección por nombre es la más apropiada para los usuarios regulares y experimentados, pero no para los casuales.

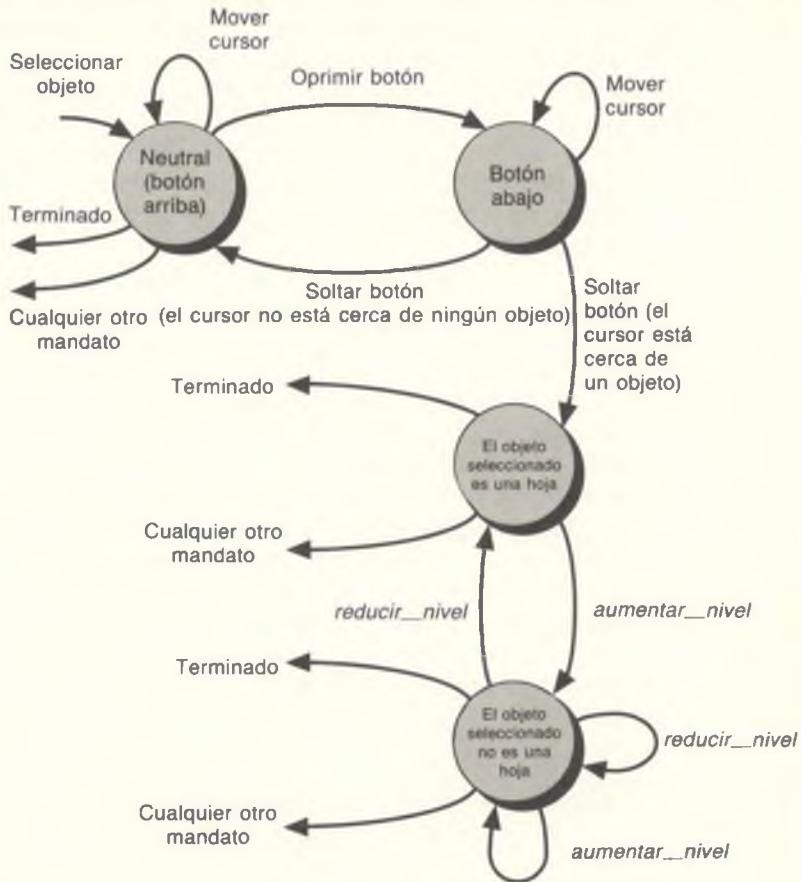
Si es necesario especificar nombres a través del tecleado, una forma de realimentación útil es presentar, inmediatamente después de oprimir cada tecla, la lista (o lista parcial si la completa es demasiado larga) de los nombres en el conjunto de selección que equivalen a la secuencia de caracteres tecleados hasta el momento. Esta presentación puede hacer recordar al usuario la forma correcta de escribir el nombre después de haber especificado los primeros caracteres. En cuanto se ha tecleado una equivalencia no ambigua, se puede realizar automáticamente el nombre en la lista. De manera alternativa, el nombre se puede

completar en forma automática en cuanto se haya tecleado una equivalencia no ambigua. Esta técnica, llamada **autocompleción**, puede ocasionar desconcierto en usuarios nuevos, por lo que se recomienda actuar con precaución. Otra estrategia para el tecleo de nombres es la corrección ortográfica [en ocasiones llamada *Do What I Mean* ("haz lo que quiero decir") o DWIM]. Si el nombre tecleado no equivale a uno conocido por el sistema, se presentan al usuario como alternativa los que se aproximen al nombre tecleado. La determinación de la aproximación puede ser sencilla y limitarse a errores en un solo carácter, o incluir errores en varios caracteres y caracteres faltantes.

Si se emplea un reconocedor de voz, el usuario puede pronunciar el nombre, abreviatura o código en lugar de teclearlo. La entrada de voz es una forma sencilla de distinguir los mandatos de los datos: los mandatos se registran en forma verbal y los datos por medio del teclado u otros medios. En un ambiente de teclado, esta característica elimina la necesidad de modos o caracteres especiales para distinguir datos y mandatos.

**Selección de objetos con apuntador.** Cualquiera de las técnicas apuntadoras mencionadas en la introducción a la sección 8.2 se puede usar para seleccionar un objeto, apuntando primero y luego indicando (con la presión de un botón) que se apunta al objeto deseado. Sin embargo, ¿qué pasa si el objeto tiene varios niveles de jerarquía, como el robot del capítulo 7? Si el cursor está sobre la mano del robot, no es evidente si el usuario apunta a la mano, al brazo o a todo el robot. Se pueden usar mandatos como *seleccionar\_robot* y *seleccionar\_brazo* para especificar el nivel de la jerarquía. Por otra parte, si el nivel al cual trabaja el usuario no cambia con frecuencia, el usuario podrá operar con mayor velocidad si usa un mandato aparte, como *establecer\_nivel\_selección*, para cambiar el nivel de la jerarquía.

Se requiere otro método si el diseñador del sistema no conoce el número de niveles jerárquicos y éste puede ser considerable (como en un sistema de bosquejo, donde los símbolos se forman con primitivas gráficas y otros símbolos). Se necesitan al menos dos mandatos de usuario: *aumentar\_nivel* y *reducir\_nivel*. Cuando el usuario selecciona algo, el sistema realza el objeto de menor nivel que detecta. Si éste es el deseado, el usuario puede continuar. En caso contrario, el usuario emite el primer mandato: *aumentar\_nivel*. Se realza todo el objeto de primer nivel al cual pertenece el objeto detectado. Si esto no es lo que desea el usuario, continúa el ascenso por la jerarquía mientras se realzan cada vez más porciones de la imagen. Si el usuario asciende demasiado, puede usar el mandato *reducir\_nivel* para invertir la dirección del recorrido por la jerarquía. Además, puede ser de utilidad un mandato *regresar\_al\_nivel\_más\_bajo* si la jerarquía es muy profunda; también puede servir un diagrama de jerarquía en otra ventana, que presente la ubicación de la selección actual en la jerarquía. El diagrama de estado de la figura 8.4 muestra un método para la selección jerárquica. Por otra parte, se puede emplear un solo mandato, digamos *ascender\_por\_la\_jerarquía*, para regresar directamente al nodo seleccionado después de llegar al nodo raíz.



**Figura 8.4** Diagrama de estado para una técnica de selección de objetos en una jerarquía con cantidad arbitraria de niveles. Los mandatos *aumentar\_nivel* y *reducir\_nivel* sirven para ascender o descender por la jerarquía. El mandato *reducir\_nivel* no está disponible en el estado “El objeto seleccionado es una hoja”. El usuario selecciona un objeto apuntando a él con el cursor y luego oprimiendo y soltando un botón.

### **8.2.3 Tarea de interacción para selección: conjunto de opciones de tamaño relativamente fijo**

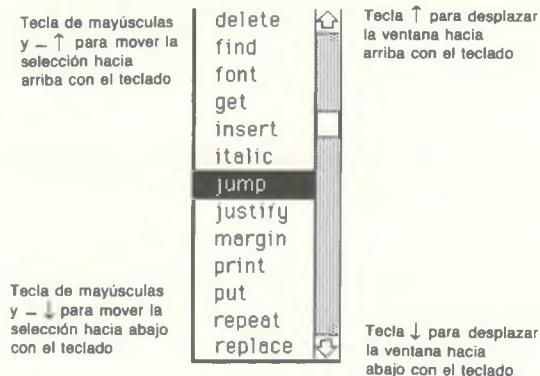
La selección por medio de menús es una de las técnicas más ricas para seleccionar a partir de un conjunto de opciones de tamaño relativamente fijo. A continuación analizaremos los factores clave en el diseño de menús.

**Diseño jerárquico y de un solo nivel.** Una de las decisiones fundamentales en el diseño de menús se presenta cuando el conjunto de opciones es demasiado grande para presentarlo todo a la vez. Este menú se puede dividir en una jerarquía

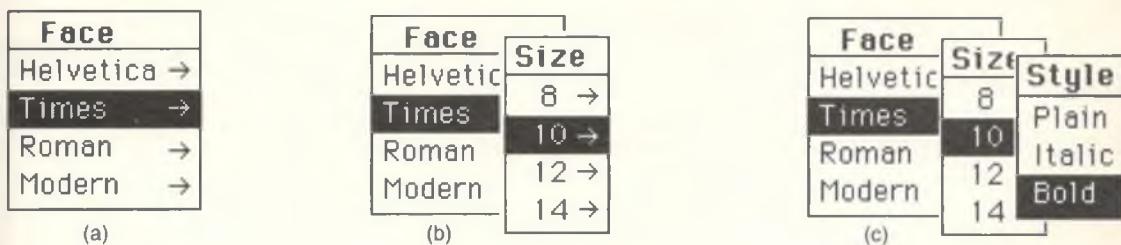
lógicamente estructurada o presentarse como una secuencia lineal de opciones que se paginará o desplazará. Una barra de desplazamiento, (*scrollbar*) como la que se utiliza en muchos administradores de ventanas, permite presentar en forma concisa los mandatos de paginación y desplazamiento. También se puede proporcionar una alternativa rápida orientada al teclado para apuntar a los mandatos de desplazamiento; por ejemplo, las teclas de flecha pueden servir para desplazar la ventana y la tecla de mayúsculas, combinada con las teclas de flecha, para mover la selección dentro de la ventana visible, como se ilustra en la figura 8.5.

En un menú jerárquico, el usuario selecciona primero el conjunto de opciones en la parte superior de la jerarquía, con lo cual se presenta un segundo conjunto de opciones. El proceso se repite hasta seleccionar un nodo hoja (es decir, un elemento del conjunto de opciones) del árbol de jerarquía. Como sucede con la selección jerárquica de objetos, se requieren mecanismos de navegación que permitan al usuario retroceder por la jerarquía si seleccionó un subárbol incorrecto. También es necesaria la realimentación visual que indique al usuario dónde se encuentra dentro de la jerarquía.

Las jerarquías de menú se pueden presentar en varias formas. Por supuesto, los niveles sucesivos de la jerarquía pueden reemplazar a los anteriores en la pantalla conforme se seleccionan, pero este método no ofrece al usuario una indicación clara de la posición dentro de la jerarquía. La jerarquía en cascada, ilustrada en la figura 8.6, es mucho más atractiva. Es necesario presentar una porción suficiente de cada menú para que toda la trayectoria de selección realizada sea visible, y hay que proporcionar además alguna forma para indicar si el elemento del menú es un nodo hoja o el nombre de un menú de nivel inferior (en la figura, la flecha que apunta a la derecha lleva a cabo esta función). Otra posibilidad es mostrar únicamente el nombre de cada selección que se ha hecho en el recorrido de la jerarquía, además de todas las opciones disponibles en el nivel actual.



**Figura 8.5** Menú con ventana desplazable. El usuario controla el desplazamiento seleccionando las flechas hacia arriba o hacia abajo o arrastrando el recuadro en la barra de desplazamiento.



**Figura 8.6** Menú jerárquico de ventanas. (a) El primer menú aparece donde está el cursor, como respuesta al oprimir un botón. El cursor se puede mover hacia arriba o hacia abajo para seleccionar el tipo de letra deseado. (b) El cursor se mueve a la derecha para presentar el segundo menú. (c) Se repite el proceso para el tercer menú.

Al diseñar un menú jerárquico siempre está presente el tema de la profundidad y la extensión. Snowberry *et al.* [SNOW83] han determinado experimentalmente que el tiempo y la precisión mejoran cuando se usan menús más extensos con menos niveles de selección. Landauer y Nachbar [LAND85] y otros investigadores presentan resultados similares. Sin embargo, estos resultados no pueden generalizarse a jerarquías de menú que no tienen una estructura natural y fácil de comprender.

La selección en menús jerárquicos casi siempre requiere una técnica de teclado o de tecla de función aceleradora para hacer más rápida la selección en el caso de usuarios experimentados (llamados **poderosos**). Esto es fácil de hacer si cada nodo del árbol tiene un nombre único, de manera que el usuario pueda registrar el nombre en forma directa, y si el sistema de menú ofrece un respaldo en caso de que el usuario no pueda recordar el nombre. Si los nombres sólo son únicos en el nivel de la jerarquía, el usuario experimentado deberá teclear el nombre completo de la trayectoria hasta el nodo hoja deseado.

**Colocación de menús.** Los menús que se presentan en pantalla pueden ser estáticos y permanentemente visibles, o pueden aparecer en forma dinámica cuando se les solicite (menús desprendibles, de ventana, descendentes, laterales).

Un menú de ventana aparece en la pantalla cuando se hace una selección, ya sea como respuesta a una acción explícita del usuario (por lo general, oprimiendo un botón del ratón o de un disco rastreador) o en forma automática porque el siguiente paso del diálogo requiere la selección de un menú. El menú suele aparecer en la posición del cursor, que en la mayoría de los casos es el centro de la atención visual del usuario, manteniendo así la continuidad. Una característica atractiva de los menús de ventana es el realce inicial de la selección más reciente si es más probable que se seleccione por segunda vez ésta y no otra opción; el menú se coloca de manera que el cursor quede sobre la entrada correspondiente.

Los menús de ventana y otros que aparecen en pantalla ahorran espacio de presentación, uno de los recursos máspreciados en el diseño de interfaces con el

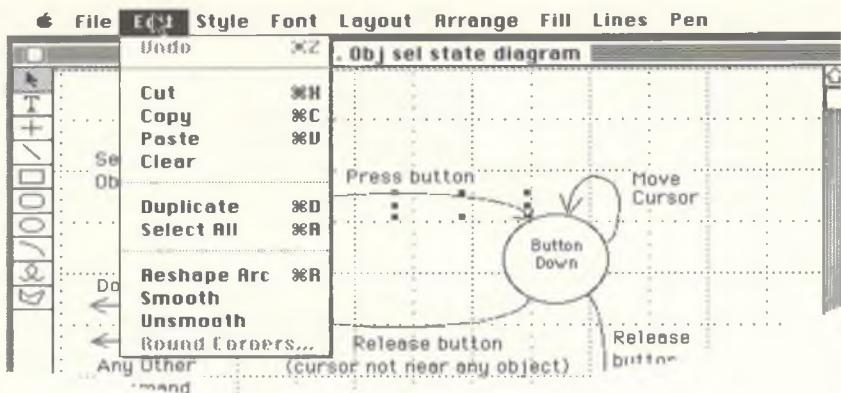


Figura 8.7

Menú descendente en Macintosh. El último elemento del menú es gris y no negro para indicar que no está disponible en ese momento (el objeto seleccionado actualmente, el arco, no tiene esquinas que se puedan redondear). El mandato *Undo* (cancelar último mandato) también está en gris, ya que no puede cancelarse el efecto del último mandato. Las abreviaturas son teclas aceleradoras para usuarios experimentados. (Copyright 1988 Claris Corporation. Derechos reservados.)

usuario. Su utilización es más sencilla si existe una instrucción rápida de operación de trama, descrita en el capítulo 2.

A diferencia de los menús de ventana, los menús descendentes están anclados en la barra de menú en la parte superior de la pantalla. Todas las interfaces gráficas populares (Apple Macintosh, Microsoft Windows, OPEN LOOK, Motif) usan menús descendentes. Los menús de Macintosh, ilustrados en la figura 8.7, también presentan teclas aceleradoras y sensibilidad al contexto.

**Selección actual.** Si un sistema incluye el concepto de *elemento seleccionado actualmente* de un conjunto de opciones, la selección del menú permite realizar este elemento. En algunos casos el sistema proporciona una configuración inicial por omisión, que se usa a menos que el usuario la cambie. El elemento seleccionado actualmente se puede presentar en varias formas. Una es con la técnica de interacción de **botón de radio** (Fig. 8.8), diseñada teniendo como base los botones de sintonía de los radios de los automóviles. Una vez más, algunos menús de ventana realzan el elemento más reciente que se haya seleccionado y lo colocan bajo el cursor, suponiendo que es más probable que el usuario vuelva a seleccionar ese elemento que otro.

**Tamaño y forma de los elementos del menú.** La precisión y la velocidad apuntar son afectadas por el tamaño de cada elemento del menú. Los elementos más grandes son más fáciles de seleccionar, tal y como lo predice la ley de Fitts [FITT54; CARD83]; por otra parte, los elementos menores ocupan menos espacio y permiten presentar más elementos de menú en un área fija, aunque ocasionalmente causan más errores durante la selección. Por lo tanto, existe un conflicto entre usabilidad y eficiencia.

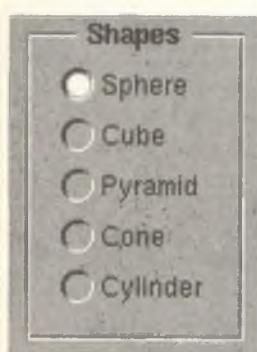


Figura 8.8

Técnica de botón de radio para seleccionar un elemento en un conjunto de alternativas mutuamente excluyentes. (Cortesía de NeXT, Inc. © 1989 NeXT, Inc.)

elementos pequeños para preservar el espacio en pantalla y usar elementos mayores para reducir el tiempo de selección y reducir los errores.

**Reconocimiento de patrones.** En las técnicas de selección que utilizan reconocimiento de patrones, el usuario hace secuencias de movimientos con un dispositivo de posicionamiento continuo, como una tableta de datos o un ratón. El reconocedor de patrones compara automáticamente la secuencia contra un conjunto de patrones definidos, cada uno de los cuales corresponde a un elemento del conjunto de selección. Las marcas de edición de textos, que indican eliminación, mayúsculas, movimiento, etc., son buenos candidatos para este método [WOLF87].

Los avances más recientes en los algoritmos de reconocimiento de caracteres han producido sistemas operativos y computadores portátiles, como Apple Newton, basados en la escritura. Los patrones se registran en una tableta y se reconocen e interpretan como mandatos, números y letras.

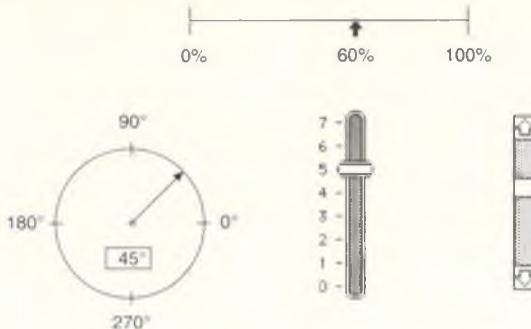
**Teclas de función.** Los elementos del conjunto de opciones se pueden asociar a teclas de función. (Podemos pensar en entradas que comprenden oprimir una sola tecla como si fueran teclas de función.) ¡Por desgracia, parece que casi siempre faltan teclas! Las teclas se pueden emplear en forma de selección jerárquica y su significado se puede alterar usando acordes, por ejemplo oprimiendo las teclas de mayúsculas y de control del teclado junto con la tecla de función. Por ejemplo, Microsoft Word en Macintosh usa “mayúsculas-opción->” para aumentar el puntaje y la combinación simétrica “mayúsculas-opción-<” para reducir el puntaje; “mayúsculas-opción-I” convierte texto común a cursivas y el texto en cursivas a común, mientras que “mayúsculas-opción-U” trata el texto subrayado en forma similar.

#### 8.2.4 Tarea de interacción para texto

La tarea de registro de cadenas de texto comprende introducir una cadena de caracteres a la cual la aplicación no asigna ningún significado especial. De esta manera, el registro de un nombre de mandato *no* es una tarea de registro de texto. En cambio, el tecleado de leyendas para un gráfico y el registro de texto en un procesador de texto *sí* son tareas de entrada de texto. Por supuesto, la técnica más común para registrar texto es a través del teclado QWERTY.

#### 8.2.5 Tarea de interacción para cuantificación

La tarea de interacción para cuantificación implica especificar un valor numérico entre ciertos valores mínimo y máximo. Las técnicas de interacción más usuales son teclear el valor, girar una perilla al valor deseado y usar un contador ascendente-descendente para seleccionar el valor. Al igual que la tarea de posicionamiento, esta tarea puede ser lingüística o espacial. Si es lingüística, el usuario conoce el valor específico que se registrará; si es espacial, el usuario aumenta o reduce un valor en una cantidad determinada, quizás con una idea



**Figura 8.9** Varios indicadores que puede emplear el usuario para introducir valores, deslizando el apuntador de control. La realimentación es proporcionada por el indicador y, en dos casos, por un valor numérico. (Deslizadores verticales © Apple Computer, Inc.)

aproximada del valor final deseado. En el primer caso, la técnica de interacción debe comprender la realimentación numérica del valor seleccionado (una forma de lograrlo es que el usuario teclee el valor real); en el segundo caso, es más importante ofrecer una idea general del valor. Esto suele hacerse con una técnica de realimentación de orientación espacial, como la presentación de un medidor o cuadrante donde se presenta el valor actual (y quizás el anterior).

Otra forma de introducir valores es con un potenciómetro. La decisión de usar un potenciómetro lineal o uno giratorio debe considerar si la realimentación de un cambio de valor es rotatoria (p. ej., girar la manecilla de un reloj) o lineal (p. ej., un medidor de temperatura ascendente). La posición actual de un potenciómetro deslizante o de un grupo de ellos es más fácil de comprender a primera vista que un grupo de potenciómetros giratorios, incluso si las perillas tienen apuntadores. Por otra parte, los potenciómetros giratorios son más fáciles de ajustar. La disponibilidad de ambos tipos de potenciómetros puede ayudar a los usuarios a relacionar el significado con cada dispositivo. Es importante usar las direcciones en forma consistente. Lo usual es que el movimiento ascendente o en el sentido del giro de las manecillas del reloj incremente un valor.

En la manipulación de escala continua, el usuario apunta al indicador de valor actual en la escala o medidor que se presenta, oprime el botón de selección, arrastra el indicador por la escala hasta el valor deseado y suelta el botón de selección. Generalmente se usa un apuntador para indicar el valor seleccionado en la escala y también puede proporcionarse un eco numérico. En la figura 8.9 se muestran estos indicadores y su realimentación.

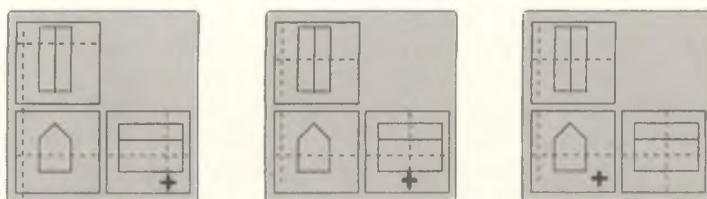
## 8.2.6 Tareas de interacción tridimensional

Dos de las cuatro tareas de interacción que describimos para aplicaciones bidimensionales son más complicadas en tres dimensiones: la posición y la selec-

ción. La primera parte de esta sección trata con una técnica para el posicionamiento y la selección, que están estrechamente relacionadas. En esta sección también presentaremos una tarea adicional de interacción tridimensional: la rotación (en el sentido de la orientación de un objeto en el espacio tridimensional). La razón principal de las complicaciones es la dificultad para percibir las relaciones de profundidad tridimensional del cursor o de un objeto con respecto a otros. Esto difiere notablemente de la interacción bidimensional, en la cual el usuario puede percatarse con facilidad si el cursor está arriba, a un lado o sobre un objeto. Otra complicación reside en que los dispositivos de interacción más comunes, como el ratón y las tabletas, son dispositivos bidimensionales, y necesitamos una forma de establecer la correspondencia entre los movimientos de estos dispositivos bidimensionales y el espacio tridimensional.

La presentación de pares estéreo, correspondientes a las vistas del ojo izquierdo y el derecho, ayuda a comprender las relaciones generales de profundidad, pero su precisión es limitada como método de localización. En el capítulo 12 y en [HODG85] se analizan métodos para presentar pares estéreo a la vista humana. En los capítulos 12 a 14 se describen otras formas de indicar las relaciones de profundidad.

En la figura 8.10 se muestra una forma común de determinar la posición en tres dimensiones. El cursor bidimensional, controlado por un ratón (por ejemplo) se mueve libremente por las tres vistas. El usuario puede seleccionar cualquiera de las líneas punteadas del cursor tridimensional y arrastrar la línea por medio de una secuencia “oprimir botón-arrastrar-soltar botón”. Si el evento de botón oprimido tiene lugar cerca de la intersección de dos líneas punteadas del cursor, se seleccionan ambas y se mueven con el ratón. Aunque este método puede parecer restrictivo por obligar al usuario a trabajar en una o dos dimensiones a la vez, en ocasiones es provechoso descomponer la tarea de manipulación tridimensional en tareas más sencillas con menos dimensiones. La selección y la localización se simplifican al usar varias vistas: los objetos superpuestos y por lo tanto difíciles de distinguir en una sola vista quizás no se sobrepongan en otra.



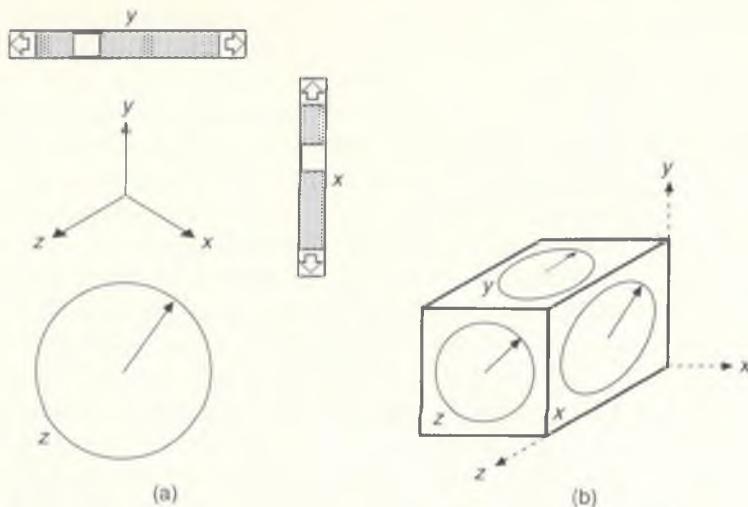
Oprima el botón cuando el cursor bidimensional esté sobre el cursor tridimensional punteado

Arrastre el cursor tridimensional; todas las vistas se actualizan en forma apropiada

Suelte el botón; el cursor bidimensional ya no controla al tridimensional

**Figura 8.10**

Técnica de posicionamiento tridimensional usando tres vistas de la misma escena (una casa). El cursor bidimensional (+) se usa para seleccionar una de las líneas punteadas de cursor tridimensional.



**Figura 8.11** Dos métodos para la rotación tridimensional. (a) Dos controles deslizantes para realizar la rotación sobre los ejes  $x$  y  $y$  de la pantalla y un indicador de rotación con respecto al eje  $z$  de la pantalla. El sistema de coordenadas representa coordenadas mundiales e indica cómo se relacionan las coordenadas mundiales con las coordenadas de pantalla. (b) Tres indicadores para controlar la rotación sobre los tres ejes. La colocación de los indicadores en el cubo ofrece gran compatibilidad estímulo-respuesta.

Como sucede en la localización y en la selección, los aspectos importantes en la rotación tridimensional son comprender las relaciones de profundidad, establecer la correspondencia entre los dispositivos de interacción bidimensionales y el espacio tridimensional, y asegurar la compatibilidad estímulo respuesta (compatibilidad E-R).<sup>1</sup> Una técnica de rotación tridimensional fácil de implementar utiliza controles deslizantes para especificar la rotación en los tres ejes. La compatibilidad estímulo-respuesta sugiere que los tres ejes deben estar en el sistema de coordenadas de la pantalla:  $x$  hacia la derecha,  $y$  hacia arriba y  $z$  hacia afuera (o hacia adentro) de la pantalla [BRIT78]. Por supuesto, el centro de rotación debe especificarse de manera explícita en un paso aparte, o en forma implícita (por lo general el origen de las coordenadas de la pantalla, el origen del objeto o el centro del objeto). Es bastante sencillo proporcionar la rotación sobre los ejes  $x$  y  $y$  de la pantalla, como se sugiere en la figura 8.11(a). El sistema de coordenadas ( $x$ ,  $y$ ,  $z$ ) relacionado con los controles deslizantes gira al mover estos controles, para mostrar el efecto de la rotación. El método de rotación de dos ejes se puede generalizar fácilmente a tres ejes añadiendo un control para la rotación sobre el eje  $z$  (por cuestiones de compatibilidad estímulo-respuesta, es preferible un control giratorio que uno deslizante). Se obtiene au-

<sup>1</sup> El principio de los factores humanos, el cual establece que las respuestas del sistema a las acciones del usuario deben ser en la misma dirección u orientación, y que la magnitud de las respuestas debe ser proporcional a las acciones.

más compatibilidad E-R si los indicadores se disponen sobre las caras de un cubo, como se ilustra en la figura 8.11(b), que sugiere con claridad los ejes controlados por cada indicador. En lugar de los indicadores se puede usar una bola rastreadora tridimensional.

Muchas veces hay que combinar tareas de interacción tridimensional. Por lo tanto, la rotación requiere una tarea de selección para el objeto que se rotará, una tarea de posición para el centro de la rotación y una tarea de orientación para la rotación real. La especificación de una vista tridimensional se puede considerar como la combinación de tareas de posicionamiento (dónde está el ojo), orientación (cómo se orienta el ojo) y escalamiento (campo visual, o qué porción del plano de proyección corresponde al área de vista). Podemos crear una tarea de este tipo combinando algunas de las técnicas que hemos analizado, o diseñando una capacidad de *vuelo* con la cual el observador “vuela” en un aeroplano imaginario alrededor de un mundo tridimensional. Los controles habituales son los de un avión: cabeceo, balanceo y guíñada, más velocidad para acelerar o frenar. Con el concepto de vuelo, el usuario necesita una vista superior, como una vista de planta bidimensional, que indique la posición y la dirección del aeroplano imaginario con respecto a tierra.

## 8.3 Tareas de interacción compuestas

Las tareas de interacción compuestas se construyen a partir de las tareas de interacción básicas descritas en la sección anterior, y por lo general son combinaciones de tareas básicas integradas en una unidad. Hay tres formas principales de tareas de interacción compuestas: recuadros de diálogo, que se usan para especificar varias unidades de información; de construcción, empleadas para crear objetos que requieren dos o más posiciones; y de manipulación, que sirven para alterar la forma de objetos geométricos existentes.

### 8.3.1 Recuadros de diálogo

Muchas veces hay que seleccionar varios elementos de un conjunto de selección. Por ejemplo, los atributos de texto, como cursivas, negritas, subrayado, huecos y versales, no son mutuamente excluyentes y quizás el usuario desee seleccionar dos o más a la vez. Además, puede haber varios conjuntos de atributos relevantes, como el estilo de letra y la familia tipográfica. Algunos de los métodos de menú que sirven para seleccionar un elemento de un conjunto de selección no son adecuados para selecciones múltiples. Por ejemplo, los menús descendentes y de ventana generalmente desaparecen al efectuar una selección, por lo que se requiere una segunda activación para la otra selección.

Este problema se puede solucionar con recuadros de diálogo, un tipo de menú que permanece visible hasta que el usuario lo quita explícitamente. Además, los recuadros de diálogo permiten seleccionar entre más de un conjunto y también pueden incluir áreas para introducir texto y valores. Las selecciones

realizadas en los recuadros de diálogo se pueden corregir de inmediato. Una vez que se ha introducido toda la información en el recuadro de diálogo, éste se quita con un mandato explícito. Los atributos y otros valores especificados en un recuadro de diálogo se pueden aplicar de inmediato, lo que permite al usuario anticipar el efecto de un cambio de tipo de letra o de estilo de línea.

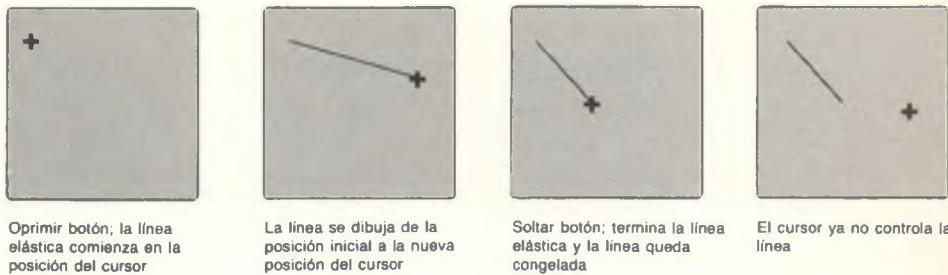
### 8.3.2 Técnicas de construcción

Una manera de construir una línea es que el usuario indique un punto extremo y luego el otro; una vez que se ha especificado el segundo punto extremo, se dibuja una línea entre los dos puntos. Sin embargo, con esta técnica el usuario no puede ensayar fácilmente diferentes posiciones de la línea antes de decidir cuál será la definitiva, ya que la línea en realidad no se dibuja hasta especificar el segundo punto extremo. Con este estilo de interacción, el usuario tiene que invocar un mandato cada vez que se reubique un punto extremo.

Un método mucho mejor es el **elástico**, descrito en el capítulo 2. Cuando el usuario oprime un botón (generalmente un botón del ratón o la punta del lápiz de la tableta), el cursor (usualmente, controlado por un dispositivo de posicionamiento continuo, aunque no siempre) establece la posición inicial de la línea. Al mover el cursor también se mueve el punto extremo de la línea; al soltar el botón, el punto extremo queda congelado. En la figura 8.12 se presenta la secuencia de dibujo de una línea elástica. El estado *elástico* se encuentra activo sólo mientras esté oprimido el botón. Los movimientos del cursor alteran la línea actual mientras se encuentre activo este estado.

A partir del dibujo de líneas elásticas se deriva toda una gama de técnicas de interacción. La técnica de **rectángulo elástico** comienza por anclar un vértice de un rectángulo al oprimirse un botón, y después de esta acción el vértice opuesto se enlaza dinámicamente con el cursor hasta que se suelte el botón. El diagrama de estado de esta técnica difiere del que corresponde al dibujo de líneas elásticas, aunque sólo en que la realimentación dinámica presenta un rectángulo y no una línea. La técnica de **círculo elástico** crea un círculo centrado en la posición inicial del cursor y que pasa por la posición actual del cursor, o bien un círculo que está dentro del cuadrado definido por vértices opuestos. Todas estas técnicas tienen un aspecto común: la secuencia de acciones del usuario, que consiste en oprimir el botón, mover el localizador observando la realimentación, y soltar el botón.

Se pueden aplicar **restricciones** de varios tipos a la posición del cursor en estas técnicas. Por ejemplo, en la figura 8.13 se muestra una secuencia de líneas dibujadas con las mismas posiciones del cursor que en la figura 8.12, pero aplicando una restricción horizontal. De esta manera también se puede dibujar una línea vertical o con otra orientación. Las polilíneas formadas exclusivamente por líneas horizontales y verticales, como en las tablillas de circuitos impresos, las pastillas VLSI y los mapas de algunas ciudades, son fáciles de crear; los ángulos rectos se introducen como respuesta a un mandato del usuario o forma automática al cambiar la dirección del cursor. El concepto se puede generalizar a cualquier forma, como serían círculos, elipses o cualquier otra curva.



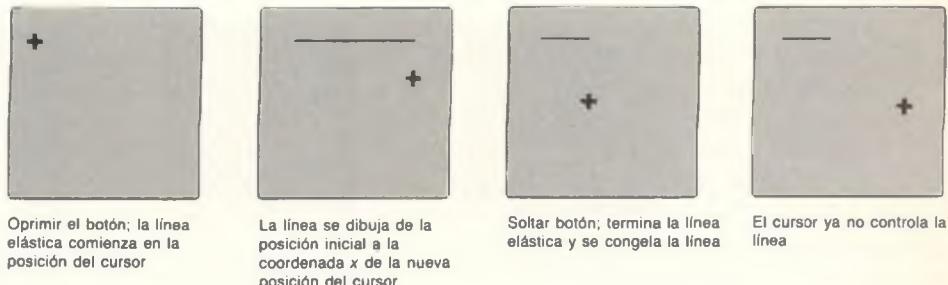
**Figura 8.12** Dibujo de una linea elástica.

va; la curva se inicia en una posición y los movimientos del cursor determinan cuánto de la curva se presentará. En términos generales, la posición del cursor sirve como entrada para una función de restricción cuya salida se emplea para presentar la porción apropiada del objeto.

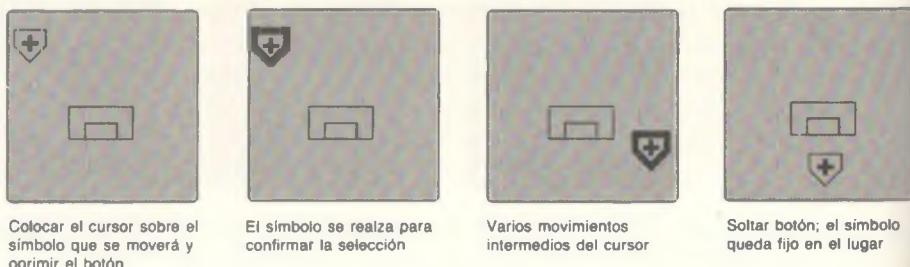
### 8.3.3 Manipulación dinámica

No es suficiente poder crear líneas, rectángulos y otros objetos. En muchos casos el usuario debe ser capaz de modificar entidades geométricas previamente creadas.

El arrastre mueve un símbolo seleccionado de una posición a otra bajo el control de un cursor, como se ilustra en la figura 8.14. El arrastre generalmente se inicia al oprimir un botón (en algunos casos, la acción de botón oprimido también se usa para seleccionar el símbolo que está bajo el cursor y que será arrastrado); después, al soltar el botón el símbolo queda fijo en el lugar, de manera que no lo afecten movimientos ulteriores del cursor. Esta secuencia de “oprimir botón-arrastrar-soltar botón” también se conoce como interacción “oprima y arrastre”.



**Figura 8.13** Dibujo de línea elástica con restricción horizontal.

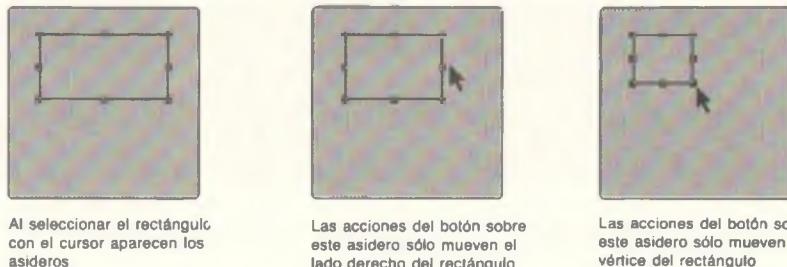


**Figura 8.14** Arrastre de un símbolo a una nueva posición.

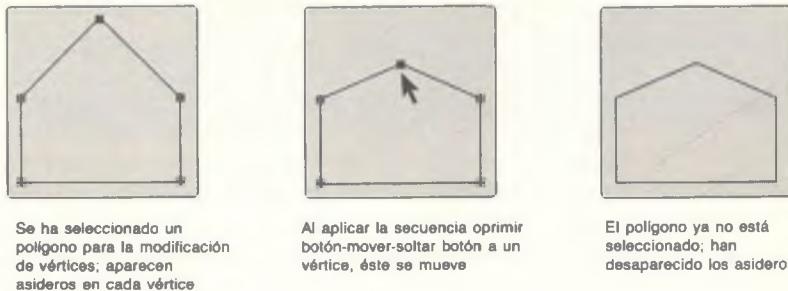
El concepto de los **asideros** es útil para permitir el escalamiento de un objeto. En la figura 8.15 se muestra un objeto con ocho asideros, los cuales se presentan como pequeños cuadros en los vértices y en los lados de la caja imaginaria que rodea al objeto. El usuario selecciona uno de los asideros y lo arrastra para escalar el objeto. Si el asidero está en un vértice, el vértice diagonalmente opuesto queda fijo en su lugar. Si el asidero está a la mitad de un lado, el lado opuesto es el que queda fijo.

Al integrar esta técnica a una interfaz completa con el usuario, los asideros sólo aparecen cuando se selecciona un objeto para aplicar una operación. Los asideros también sirven como código visual único que indica la selección del objeto, ya que otras formas de codificación visual (p. ej., el grosor de las líneas, las líneas punteadas o el cambio de intensidad) también podrán formar parte del dibujo.

El arrastre, la rotación y el escalamiento afectan a todo un objeto. ¿Qué sucede si queremos mover puntos individuales, como los vértices de un polígono? Podrían nombrarse los vértices y el usuario podría especificar el nombre del vértice y sus nuevas coordenadas ( $x, y$ ). Sin embargo, es más práctica la estrategia de apunte y arrastre que usamos para mover el objeto. En este caso, el usuario



**Figura 8.15** Asideros utilizados para cambiar el tamaño de objetos.



**Figura 8.16** Uso de asideros para cambiar la posición de los vértices de un polígono.

apunta a un vértice, lo selecciona y lo arrastra a una nueva posición. Los vértices adyacentes al seleccionado permanecen seleccionados a través de líneas elásticas. Para facilitar la selección de un vértice se puede hacer que parpadee cuando esté cerca el cursor o superponer asideros a los vértices, como en la figura 8.16. En forma similar, el usuario puede mover la arista de un polígono por medio de su selección y arrastre, manteniendo la arista su pendiente original. En el caso de curvas y superficie suaves, se pueden ofrecer asideros para que el usuario manipule puntos que controlan su forma, como veremos con mayor detalle en el capítulo 9.

## B.4 Conjuntos de herramientas para técnicas de interacción

La apariencia y el comportamiento de la interfaz de un computador con el usuario dependen en gran medida del conjunto de técnicas de interacción que se ofrezcan. Recuerde que las técnicas de interacción proporcionadas son el punto de enlace con el hardware del diseño de la interfaz con el usuario. El diseño y la implantación de un buen conjunto de técnicas de interacción constituyen una tarea que requiere mucho tiempo: los conjuntos de herramientas para técnicas de interacción, que son bibliotecas de subrutinas de técnicas de interacción, son mecanismos que ponen una colección de técnicas al alcance de los programadores de aplicaciones. Este método, que ayuda a asegurar una apariencia y un comportamiento consistentes entre programas de aplicación, es definitivamente una buena práctica de la ingeniería de software.

Los conjuntos de herramientas para técnicas de interacción no sólo puede ser utilizados por los programas de aplicación, sino además por el administrador de ventanas residente, que después de todo es otro programa. La utilización del mismo conjunto de herramientas es una estrategia común e importante para ofrecer una apariencia y un comportamiento que unifiquen tanto diversas apli-

caciones como el propio ambiente de ventanas. Por ejemplo, el estilo de menú que se utiliza para seleccionar las operaciones de ventana debe ser el mismo estilo que se emplee en la aplicación.

Se puede implantar un conjunto de herramientas encima de un **sistema de administración de ventanas** [FOLE90]. Si no existe el sistema de ventanas, los conjuntos de herramientas se pueden implantar directamente sobre el paquete de subrutinas gráficas; sin embargo, como los elementos del conjunto incluyen menús, recuadros de diálogo, barras de desplazamiento, etc., todos los cuales se pueden implantar en ventanas, se emplea normalmente el subestrato del sistema de ventanas. Los conjuntos de herramientas de mayor uso son el de Macintosh [APPL85], OSF/Motif [OPEN89] e InterViews [LINT89] para el sistema X Window, así como varios conjuntos que sirven para implantar OPEN LOOK [SUN89]. En la ilustración en color 9 se presenta la interfaz OSF/Motif; en la 10 se muestra la interfaz OPEN LOOK.

## RESUMEN

Hemos presentado algunos de los conceptos más importantes de las interfaces con los usuarios: los dispositivos de entrada, las técnicas de interacción y las tareas de interacción. Sin embargo, hay varios aspectos del diseño y las técnicas de interfaces con el usuario que no hemos analizado, como son las ventajas y desventajas de los distintos estilos de diálogo —como el llamado *what you see is what you get* (WYSIWYG, lo que se ve es lo que se obtiene), los lenguajes de mandatos y la manipulación directa— y los aspectos relacionados con los administradores de ventanas que afectan la interfaz con el usuario. En [FOLE90] podrá hallar un tratamiento completo de estos temas.

## Ejercicios

8.1 Examine una interfaz computador-usuario con la cual esté familiarizado. Liste cada una de las tareas de interacción que se utilicen. Clasifique cada tarea como una de las cuatro tareas de interacción básicas de la sección 8.2. Si una interacción no se ajusta a este método de clasificación, trate de descomponerla.

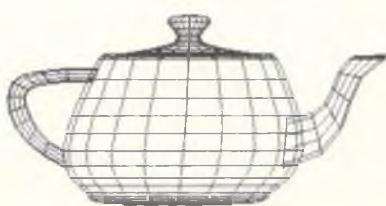
8.2 Extienda el diagrama de estado de la figura 8.4 para incluir un mandato de “retorno al nivel más bajo” que devuelva la selección al menor nivel de la jerarquía, de manera que se seleccione de nuevo lo que se seleccionó originalmente.

8.3 Implante un paquete de menús en una pantalla de trama en color usando una tabla de consulta tal que la presentación del menú sea con un color fuerte brillante, pero parcialmente transparente, y los colores debajo del menú aparezcan en gris.

8.4 Implante cualquiera de las técnicas de interacción tridimensional presentadas en este capítulo.

8.5 Dibuje el diagrama de estado que controle menús jerárquicos de ventana. Dibuje el diagrama de estado que controle menús jerárquicos de tablero.

La clásica tetera, presentada en la figura 9.1, es quizás el ícono más conocido de la graficación por computador. Desde que Martin Newell [CROW87] la modeló en 1975, ha sido utilizada por docenas de investigadores como una estructura para demostrar las técnicas más recientes en la producción de superficies y texturas realistas. Para modelar la elegante tetera se tuvo que especificar su forma como una colección de elementos superficiales suaves, conocidos como **parches bicubicos**. En muchas aplicaciones de la graficación por computador es necesario generar curvas y superficies suaves. Gran número de los objetos reales son inherentemente suaves y gran parte de la graficación por computador está orientada al modelado del mundo real. El diseño asistido por computador (CAD), los tipos de letra de alta calidad, los gráficos de datos y los dibujos de artistas contienen curvas y superficies suaves. La trayectoria de una cámara o un objeto en una secuencia de animación casi siempre es suave; así mismo, una trayectoria



**Figura 9.1** La famosa *tetera*, modelo que consiste en un conjunto de superficies curvas suaves.

a través de un espacio de intensidades o colores (Caps. 11 y 12) por lo general debe ser suave.

La necesidad de representar curvas y superficies surge en dos casos: al modelar objetos existentes (un automóvil, un rostro o una montaña) y al modelar *a partir de cero*, cuando no se representa un objeto físico previamente existente. En el primer caso es posible que no exista una descripción matemática del objeto. Por supuesto, podemos usar como modelo las coordenadas de la infinidad de puntos en el objeto, pero esta estrategia no es factible en un computador con capacidad de almacenamiento limitada. Es más usual aproximar el objeto con pedazos de planos, esferas y otras formas que sean fáciles de describir de manera matemática, lo cual requiere que los puntos en el modelo estén cerca de los puntos correspondientes en el objeto.

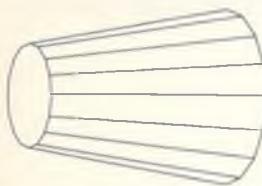
En el segundo caso, cuando no hay un modelo previo que modelar, el usuario crea el objeto durante el proceso de modelado; por consiguiente, el objeto equivale exactamente a su representación, ya que ésta es su única encarnación. Para crear el objeto, el usuario puede esculpirlo en forma interactiva, describirlo matemáticamente o proporcionar una descripción aproximada que será *completada* por un programa. En CAD, la representación por computador se usa posteriormente para generar realizaciones físicas del objeto diseñado en forma abstracta.

En este capítulo se presenta el área general del **modelado de superficies**. Se trata de un área muy extensa, y sólo presentaremos las tres representaciones más comunes de superficies tridimensionales: superficies de malla poligonal, superficies paramétricas y superficies cuádricas. También analizaremos las curvas paramétricas, ya que son un concepto interesante y porque las superficies paramétricas constituyen una generalización de las curvas.

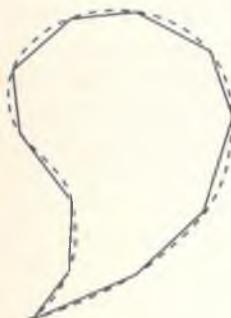
El **modelado de sólidos**, que se presenta en el capítulo siguiente, es la representación de volúmenes completamente rodeados por superficies, como son un cubo, un aeroplano o un edificio. La representación de superficies que se analiza en este capítulo puede utilizarse en el modelado de sólidos para definir las superficies que acotan el volumen.

Una **malla poligonal** es un conjunto de superficies planas limitadas por polígonos conectados entre sí. Las cajas abiertas, los gabinetes y los exteriores de edificios se pueden representar fácil y naturalmente con mallas poligonales, lo mismo que los volúmenes acotados por superficies planas. Las mallas poligonales también se pueden utilizar, aunque con menor facilidad, para representar objetos con superficies curvas, como en la figura 9.2; sin embargo, esta representación sólo es aproximada. En la figura 9.3 se presenta la sección transversal de una forma curva y la malla poligonal que la representa. Podemos hacer que los errores de la representación sean arbitrariamente pequeños si usamos más polígonos para crear mejores aproximaciones lineales por trozos, pero esta estrategia aumenta los requisitos de espacio y el tiempo de ejecución de los algoritmos que procesan la representación. Así mismo, al ampliar la imagen vuelven a ser obvias las aristas rectas.

Las **curvas polinomiales paramétricas** definen puntos en una curva tridimensional usando tres polinomios en un parámetro  $t$  para  $x$ ,  $y$  y  $z$ . Los coefi-



**Figura 9.2**  
Objeto tridimensional representado con polígonos.



**Figura 9.3**  
Sección transversal de una forma curva (línea punteada) y su representación poligonal (líneas sólidas).

cientes de los polinomios se seleccionan de manera que la curva siga la trayectoria deseada. Aunque se pueden emplear polinomios de diversos grados, sólo presentaremos el caso más común: los polinomios cúbicos (cuyos parámetros pueden tener hasta la tercera potencia). Con frecuencia utilizaremos el término **curva cónica** para estas curvas.

Los **parches de superficie polinomial paramétrica de dos variables** definen las coordenadas de los puntos en una superficie curva usando tres polinomios de dos variables, que corresponden a  $x$ ,  $y$  y  $z$ . Las fronteras de los parches son curvas polinomiales paramétricas. Se requieren mucho menos parches de superficie polinomial de dos variables que parches poligonales para aproximar una superficie curva con una precisión determinada. Sin embargo, los algoritmos para trabajar con los polinomios de dos variables son más complejos que los que se utilizan para polígonos. Como ocurre con las curvas, se pueden emplear polinomios de varios grados, pero aquí sólo analizaremos el caso común de los polinomios en los cuales ambos parámetros son cúbicos. Estas superficies se conocen como **superficies bicubáticas**.

Las **superficies cuádricas** son aquellas que se definen implícitamente con una ecuación  $f(x, y, z) = 0$ , donde  $f$  es un polinomio cuádrico en  $x$ ,  $y$  y  $z$ . Las superficies cuádricas constituyen una representación práctica de la esfera, el elipsoide y el cilindro.

En el capítulo 10, acerca del modelado de sólidos, se incorporan estas representaciones a sistemas para representar no sólo superficies, sino también volúmenes acotados (sólidos). Las representaciones de superficies que se describen en este capítulo en ocasiones se usan combinadas con otras para acotar un volumen tridimensional.

## 9.1 Mallas poligonales

Una **malla poligonal** es una colección de aristas, vértices y polígonos conectados de manera que cada arista esté compartida a lo sumo por dos polígonos. Una arista conecta dos vértices y un polígono es una secuencia cerrada de aristas. Una arista puede ser compartida por dos polígonos adyacentes, un vértice es compartido al menos por dos aristas y cada arista forma parte de *algún* polígono. Una malla poligonal se puede representar en varias formas, cada una con sus ventajas y desventajas. La tarea del programador de la aplicación es elegir la representación más apropiada. En una sola aplicación se pueden emplear varias representaciones: una para el almacenamiento externo, otra para uso interno y otra más para que el usuario pueda crear la malla en forma interactiva.

Para evaluar las representaciones se pueden emplear dos criterios básicos: espacio y tiempo. Las operaciones típicas con una malla poligonal comprenden la determinación de todas las aristas incidentes a un vértice, detectar los polígonos que comparten una arista o un vértice, hallar los vértices conectados por

una arista, encontrar las aristas de un polígono, dibujar la malla e identificar errores en la representación (p. ej., aristas, vértices o polígonos faltantes). En términos generales, cuanto más explícitamente se representen las relaciones entre polígonos, vértices y aristas, más rápidas serán las operaciones y más espacio necesitará la representación. Woo [WOO85] ha analizado la complejidad temporal de nueve operaciones básicas de acceso y nueve operaciones básicas de actualización con estructuras de datos de malla poligonal.

En las secciones 9.1.1 y 9.1.2 se analizan varios temas relacionados con las mallas poligonales: la representación de mallas poligonales, asegurar que la representación sea correcta y calcular los coeficientes del plano de un polígono.

### 9.1.1 Representación de mallas poligonales

En esta sección analizaremos tres representaciones de mallas poligonales: explícitas, apuntadores a una lista de vértices y apuntadores a una lista de aristas. En la **representación explícita**, cada polígono se representa con una lista de coordenadas de vértices:

$$P = ((x_1, y_1, z_1), (x_2, y_2, z_2), \dots, (x_n, y_n, z_n)).$$

Los vértices se almacenan en el orden en que los detectaríamos en un recorrido por el polígono. Hay aristas entre los vértices sucesivos en la lista, así como entre el primer y el último vértices. En el caso de un solo polígono, esta representación es eficiente en cuanto a espacio; sin embargo, en el caso de una malla poligonal se desperdicia mucho espacio ya que las coordenadas de los vértices compartidos se duplican. Un problema aún mayor es que no hay una representación explícita de las aristas y los vértices compartidos. Por ejemplo, para arrastrar interactivamente un vértice y todas sus aristas incidentes, es necesario hallar todos los polígonos que comparten el vértice. Para esta búsqueda se requiere la comparación de los triples de coordenadas de un polígono con los de los demás polígonos. La forma más eficiente de hacerlo sería ordenando los  $N$  triples de coordenadas, pero este proceso requiere, en el mejor de los casos,  $N \log_2 N$  pasos y no se elimina el peligro de que el mismo vértice pueda tener, debido al redondeo durante los cálculos, valores ligeramente distintos en las coordenadas para cada polígono, de tal forma que quizás nunca se detectaría una equivalencia exacta.

Con esta representación, para presentar la malla como polígonos llenados o huecos es necesario transformar cada vértice y recortar cada arista de cada polígono. Si se dibujan las aristas, cada arista compartida se dibuja dos veces, lo cual ocasiona problemas en los graficadores de pluma, las grabadoras de película y las pantallas vectoriales, debido a la sobreescritura. También puede surgir un problema en las pantallas de barrido si las aristas se dibujan en direcciones opuestas, en cuyo caso pueden intensificarse pixeles adicionales.

Los polígonos definidos con **apuntadores a una lista de vértices**, el método que utiliza SPHIGS, almacenan sólo una vez cada vértice de la malla poligonal

en la lista de vértices  $V = ((x_1, y_1, z_1), \dots, (x_n, y_n, z_n))$ . Un polígono se define con una lista de índices (o apuntadores) a la lista de vértices. Así, un polígono formado por los vértices 3, 5, 7 y 10 de la lista se representaría como  $P = (3, 5, 7, 10)$ .

En la figura 9.4 se presenta un ejemplo de esta representación, la cual tiene varias ventajas con respecto a la representación poligonal explícita. Como cada vértice se almacena una sola vez, se ahorra bastante espacio. Además, las coordenadas de un vértice se pueden modificar con facilidad. Por otra parte, sigue siendo difícil encontrar polígonos que comparten una arista y aristas compartidas que se dibujen dos veces al presentar todos los polígonos. Estos problemas se pueden eliminar representando explícitamente las aristas, como se hace en el método siguiente.

Al definir polígonos con **apuntadores a una lista de aristas**, una vez más tenemos la lista de vértices  $V$ , pero un polígono no se representa con una lista de apuntadores a la lista de vértices, sino a una lista de aristas, donde cada arista se incluye una sola vez. A su vez, cada arista de la lista apunta a los dos vértices de la lista de vértices que definen la arista y además a los polígonos (uno o dos) a los que pertenece la arista. De esta manera, un polígono se describe como  $P = (E_1, \dots, E_n)$  y una arista como  $E = (V_1, V_2, P_1, P_2)$ . Cuando una arista sólo pertenece a un polígono,  $P_1$  o  $P_2$  será nulo. En la figura 9.5 se presenta un ejemplo de esta representación.

Los polígonos huecos se muestran presentando todas las aristas, en lugar de todos los polígonos; de esta manera se evitan recortes, transformaciones y discretizaciones redundantes. Los polígonos rellenos también son fáciles de presentar. En algunas situaciones, como en la descripción de una estructura de hoja metálica en forma de panal tridimensional, algunas de las aristas son compartidas por tres polígonos. En estos casos, las descripciones de las aristas se pueden extender para incluir un número arbitrario de polígonos:  $E = (V_1, V_2, P_1, P_2, \dots, P_n)$ .

En ninguna de estas tres representaciones (es decir, polígonos explícitos, apuntadores a vértices y apuntadores a una lista de aristas) es sencillo determinar cuáles son las aristas incidentes a un vértice: hay que inspeccionar todas las aristas. Por supuesto, se puede añadir información explícitamente para permitir la determinación de estas relaciones. Por ejemplo, la representación de arista alada que usa Baumgart [BAUM75] expande la descripción de las aristas para

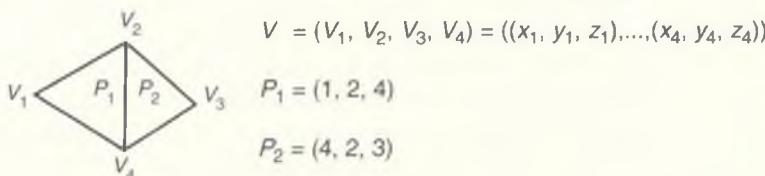
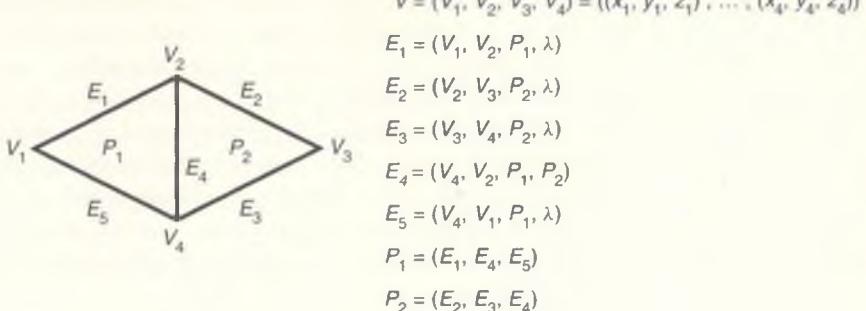


Figura 9.4 Malla poligonal definida con índices en una lista de vértices.



**Figura 9.5** Malla poligonal definida con listas de aristas para cada polígono ( $\lambda$  representa un valor nulo).

incluir apuntadores a los dos vértices adjuntos de cada polígono, mientras que la descripción de los vértices incluye un apuntador a una arista incidente (arbitraria) al vértice, con lo cual se obtiene más información acerca de los polígonos y los vértices.

### 9.1.2 Ecuaciones de planos

Al trabajar con polígonos o con mallas poligonales, muchas veces es necesario conocer la ecuación del plano donde se encuentra el polígono. Por supuesto, en algunos casos se conoce la ecuación en forma implícita, a través de los métodos de construcción interactivos que se usan para definir el polígono. Si no se conoce la ecuación, es posible utilizar las coordenadas de los tres vértices para encontrar el plano. Recuerde la ecuación de un plano:

$$Ax + By + Cz + D = 0. \quad (9.1)$$

Los coeficientes  $A$ ,  $B$  y  $C$  definen la normal al plano,  $[A \ B \ C]$ . Dados los puntos  $P_1$ ,  $P_2$  y  $P_3$  en el plano, la normal al plano se puede calcular como el producto cruz de vectores  $P_1P_2 \times P_1P_3$  (o  $P_2P_3 \times P_2P_1$ , etc.). Si el producto cruz es cero, los tres puntos son colineales y no definen un plano. En este caso se pueden usar otros vértices, si los hay. Una vez que se ha obtenido un producto cruz distinto de cero, podemos hallar  $D$  sustituyendo la normal a  $[A \ B \ C]$  y uno de los tres puntos en la ecuación (9.1).

Si hay más de tres vértices, quizás no estén todos sobre un mismo plano, posiblemente por razones numéricas o debido al método por el cual se generaron los polígonos. En este caso es mejor usar una técnica que encuentre los coeficientes  $A$ ,  $B$  y  $C$  de un plano que se aproxime a todos los vértices y no el método del producto cruz. Se puede demostrar que  $A$ ,  $B$  y  $C$  son proporcionales a las áreas con signo de las proyecciones del polígono sobre los planos ( $y$ ,  $z$ ), ( $z$ ,  $x$ ) y ( $x$ ,  $y$ ), respectivamente. Por ejemplo, si el polígono es paralelo al plano

$(x, y)$ , entonces  $A = B = 0$ , como era de esperar: las proyecciones del polígono sobre los planos  $(y, z)$  y  $(z, x)$  tienen área igual a cero. Este método es mejor que el de producto cruz porque las áreas de las proyecciones son función de las coordenadas de todos los vértices y por ende no pueden ser afectadas por la elección de unos cuantos vértices que quizás no sean coplanares con los demás vértices o con la mayoría de ellos, o que posiblemente sean colineales. Por ejemplo, el área (y por lo tanto el coeficiente)  $C$  del polígono proyectado sobre el plano  $(x, y)$  en la figura 9.6 es el área del trapezoide  $A_3$  menos las áreas de  $A_1$  y  $A_2$ . Por lo general,

$$C = \frac{1}{2} \sum_{i=1}^n (y_i + y_{i+1})(x_{i+1} - x_i), \quad (9.2)$$

donde el operador  $\oplus$  es la suma usual excepto que  $n \oplus 1 = 1$ . Las áreas para  $A$  y  $B$  se expresan en forma similar, excepto que el área de  $B$  está negada (véase el Ej. 9.1).

La ecuación (9.2) proporciona la suma de las áreas de todos los trapezoides formados por aristas sucesivas de los polígonos. Si  $x_{i+1} < x_i$ , el área contribuye en forma negativa a la suma. El signo de la suma también es útil: si los vértices han sido numerados en el sentido del giro de las manecillas del reloj (según la proyección sobre el plano), el signo será positivo; en caso contrario, será negativo.

Una vez que determinamos la ecuación del plano usando todos los vértices, podemos estimar la no planaridad del polígono calculando la distancia perpendicular del plano a cada uno de los vértices. Esta distancia  $d$  para el vértice en  $(x, y, z)$  es

$$d = \frac{Ax + By + Cz + D}{\sqrt{A^2 + B^2 + C^2}}. \quad (9.3)$$

Esta distancia es positiva o negativa, dependiendo del lado del plano donde se encuentre el punto. Si el vértice está sobre el plano,  $d = 0$ . Por supuesto,

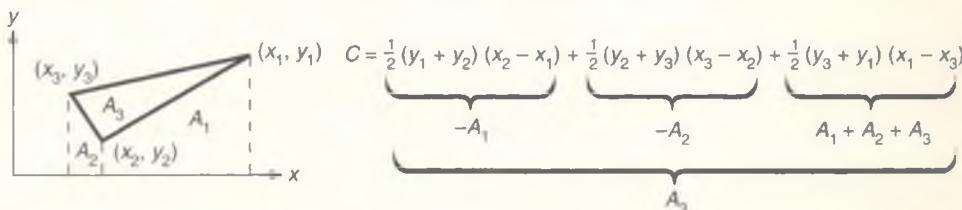


Figura 9.6 Cálculo del área  $C$  de un triángulo usando la ecuación (9.2).

para determinar en qué lado del plano está un punto, lo único importante es el signo de  $d$ , y la división entre la raíz cuadrada no es necesaria.

La ecuación del plano no es única; cualquier constante multiplicadora  $k$  distinta de cero cambia la ecuación, pero no el plano. Muchas veces es conveniente almacenar los coeficientes del plano con una normal normalizada; esto se puede hacer escogiendo

$$k = \frac{1}{\sqrt{A^2 + B^2 + C^2}}, \quad (9.4)$$

que es el recíproco de la longitud de la normal. Es más fácil calcular las distancias con la ecuación (9.3), ya que el denominador es 1.

### Ejemplo 9.1

**Problema:** Escriba una función que calcule los coeficientes de la ecuación del plano dados  $n$  vértices de un polígono que es aproximadamente plano. Suponga que los vértices del polígono se numeran en sentido contrario al giro de las manecillas, vistos hacia el plano desde su lado positivo. Los vértices y el número de vértices son argumentos que se pasan a la función.

**Respuesta:** Usando la ecuación (9.2) y ecuaciones similares para  $A$  y  $B$ , el programa es:

*Esta función calcula los coeficientes de la ecuación del plano.*

```
hallar_coeficientes_plano (float x[ ], float y[ ], float z[ ], int *núm_vértices,
                           float *a, float *b, float *c, float *d)
{
    float A, B, C, D;
    int i, j;

    A = B = C = 0.0;
    for (i = 0; i < núm_vértices; i++) {
        j = (i + 1) % núm_vértices;
        A += (z[i] + z[j]) * (y[j] - y[i]);
        B += -(x[i] + x[j]) * (z[j] - z[i]);
        C += (y[i] + y[j]) * (x[j] - x[i]);
    }
    A /= 2.0; B /= 2.0; C /= 2.0;
    D = -(A * x[0] + B * y[0] + C * z[0]);

    *a = A;
    *b = B;
    *c = C;
    *d = D;
```

## 9.2 Curvas cúbicas paramétricas

Las polilíneas y los polígonos son aproximaciones por trozos de primer grado de curvas y superficies, respectivamente. A menos que las curvas o superficies que se aproximan también sean lineales por trozos, hay que crear y almacenar gran cantidad de coordenadas de puntos extremos para lograr una precisión razonable. La manipulación interactiva de los datos para aproximar una curva es algo tediosa, ya que hay que ubicar con precisión gran cantidad de puntos.

En esta sección desarrollamos una representación más compacta y manejable de las curvas suaves por trozos; en la sección 9.3 se generaliza el desarrollo matemático a las superficies. El método general consiste en emplear funciones que sean de un grado mayor que el de las funciones lineales. Estas funciones por lo general sólo aproximan la forma deseada, pero requieren menos espacio de almacenamiento y ofrecen mayor facilidad de manipulación interactiva que las funciones lineales.

Las aproximaciones de mayor grado se pueden basar en uno de tres métodos. Primero, podemos expresar  $y$  y  $z$  como funciones *explícitas* de  $x$ , de manera que  $y = f(x)$  y  $z = g(x)$ . Los problemas con este método son los siguientes: (1) es imposible obtener valores múltiples de  $y$  para una sola  $x$ , de manera que las curvas como los círculos y las elipses se tienen que representar con varios segmentos de curva; (2) esta definición no es rotacionalmente invariante (para describir una versión rotada de la curva se requiere mucho trabajo y por lo general hay que dividir un segmento de curva en varios segmentos más); y (3) la descripción de curvas con tangentes verticales es muy difícil, ya que resulta complicado representar una pendiente infinita.

Segundo, podemos modelar las curvas como soluciones a ecuaciones *implícitas* de la forma  $f(x, y, z) = 0$ ; este método está plagado de peligros. Primero, la ecuación dada puede tener más soluciones que las que deseamos. Por ejemplo, al modelar un círculo podríamos usar  $x^2 + y^2 = 1$ , lo cual está bien. Sin embargo, ¿cómo modelamos medio círculo? Es necesario añadir restricciones como  $x \geq 0$ , que no pueden estar contenidas en la ecuación implícita. Así mismo, si se unen dos segmentos de curva definidos de manera implícita, puede ser difícil determinar si sus direcciones tangentes concuerdan en el punto de unión. La continuidad de tangentes es crítica en muchas aplicaciones.

Estas dos formas matemáticas sí permiten determinar con rapidez si un punto está en la curva o a qué lado de la curva yace el punto, como hicimos en el capítulo 3. Las normales a la curva también son fáciles de calcular. Por lo tanto, analizaremos brevemente la forma implícita en la sección 9.4.

Con la **representación paramétrica** de las curvas,  $x = x(t)$ ,  $y = y(t)$ ,  $z = z(t)$ , se superan los problemas ocasionados por las formas funcionales e implícitas y se ofrece una variedad de atractivos que serán evidentes en lo que resta de este capítulo. Las curvas paramétricas reemplazan la utilización de pendientes geométricas (que pueden ser infinitas) con vectores tangente paramétricos (que,

como veremos, nunca pueden ser infinitos). En este caso, una curva se approxima con una **curva polinomial por trozos** en lugar de la curva lineal por trozos que usamos en la sección 9.1. Cada segmento  $Q$  de la curva global está indicado por tres funciones,  $x$ ,  $y$  y  $z$ , que son polinomios cúbicos en el parámetro  $t$ .

Los polinomios cúbicos son los que se emplean con mayor frecuencia, ya que los polinomios de grado menor no ofrecen mucha flexibilidad para controlar la forma de la curva y los de mayor grado pueden introducir ondulaciones indeseadas y además requieren más cálculos. Ninguna representación de menor grado permite que un segmento de curva interpole (pase por) dos puntos extremos especificados con derivadas específicas en cada punto extremo. Dado un polinomio cúbico con sus cuatro coeficientes, se usan cuatro incógnitas para resolver los coeficientes desconocidos. Los cuatro valores conocidos podrán ser los dos puntos extremos y las derivadas en estos puntos. Así mismo, los dos coeficientes de un polinomio de primer orden (línea recta) se determinan con los dos puntos extremos. En el caso de una línea recta, las derivadas en cada extremo se determinan con la propia línea y no se pueden controlar en forma independiente. En el caso de polinomios cuadráticos (de segundo grado), que por lo tanto tienen tres coeficientes, se pueden especificar los dos puntos extremos y otra condición, como la pendiente o un punto adicional.

Así mismo, las cúbicas paramétricas son las curvas de menor grado que no son planas en tres dimensiones. Este hecho se puede observar sabiendo que los tres coeficientes de un polinomio de segundo orden se pueden especificar por completo con tres puntos y que tres puntos definen un plano donde se encuentra el polinomio.

Las curvas de mayor grado requieren más condiciones para determinar los coeficientes y pueden *ondular* en formas difíciles de controlar. A pesar de estas complejidades, las curvas de mayor grado se emplean en aplicaciones —como el diseño de automóviles y aeroplanos— donde es necesario controlar derivadas de mayor grado para crear superficies aerodinámicamente eficientes. De hecho, el desarrollo matemático de las curvas y superficies paramétricas muchas veces se expresa en función de un grado arbitrario  $n$ . En este capítulo, mantenemos  $n$  fijo en 3.

### 9.2.1 Características básicas

Los polinomios cúbicos que definen un segmento de curva  $Q(t) = [x(t) \ y(t) \ z(t)]^T$  tienen la forma

$$\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x, \\ y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y, \\ z(t) &= a_z t^3 + b_z t^2 + c_z t + d_z, \quad 0 \leq t \leq 1. \end{aligned} \quad (9.5)$$

Para tratar con segmentos finitos de la curva, limitamos el parámetro  $t$ , sin perder generalidad, al intervalo  $[0, 1]$ .

Con  $T = [t^3 \ t^2 \ t \ 1]^T$  y definiendo la matriz de coeficientes de los tres polinomios como

$$C = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \end{bmatrix}, \quad (9.6)$$

podemos reescribir la ecuación (9.5) como

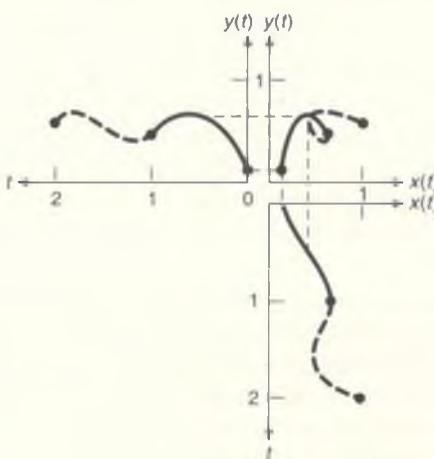
$$Q(t) = [x(t) \ y(t) \ z(t)]^T = C \cdot T. \quad (9.7)$$

Esta representación es una forma compacta de expresar la ecuación (9.5).

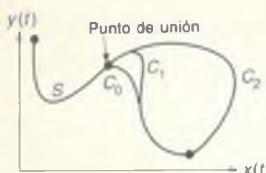
En la figura 9.7 se presentan dos segmentos unidos de curvas paramétricas y sus polinomios; también se ilustra la capacidad de las paramétricas para representar fácilmente varios valores de  $y$  para un solo valor de  $x$  en el caso de polinomios que en sí sólo tienen un valor. (Esta figura de una curva, como todas las demás en esta sección, presenta curvas bidimensionales representadas por  $[x(t) \ y(t)]^T$ .)

**Continuidad entre segmentos de curva.** La derivada de  $Q(t)$  es el **vector tangente** paramétrico de la curva. Al aplicar esta definición a la ecuación (9.7) obtenemos

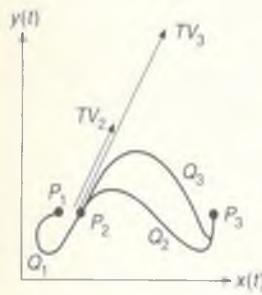
$$\begin{aligned} \frac{d}{dt} Q(t) = Q'(t) &= \left[ \frac{d}{dt} x(t) \quad \frac{d}{dt} y(t) \quad \frac{d}{dt} z(t) \right]^T = \frac{d}{dt} C \cdot T = C \cdot [3t^2 \quad 2t \quad 1 \quad 0]^T \\ &= [3a_x t^2 + 2b_x t + c_x \quad 3a_y t^2 + 2b_y t + c_y \quad 3a_z t^2 + 2b_z t + c_z]^T. \end{aligned} \quad (9.8)$$



**Figura 9.7** Dos segmentos unidos de curvas paramétricas bidimensionales y los polinomios que los definen. Las líneas punteadas entre el gráfico  $(x, y)$  y los gráficos  $x(t)$  y  $y(t)$  muestran la correspondencia entre los puntos en la curva  $(x, y)$  y los polinomios cúbicos de definición. Los gráficos  $x(t)$  y  $y(t)$  para el segundo segmento han sido trasladados para que inicien en  $t = 1$ , no en  $t = 0$ , para mostrar la continuidad de las curvas en su punto de unión.



**Figura 9.8**  
Segmento de curva  $S$  unido a los segmentos  $C_0$ ,  $C_1$  y  $C_2$  con grados 0, 1 y 2 respectivamente, de continuidad paramétrica. La diferencia visual entre  $C_1$  y  $C_2$  es muy pequeña cerca de la unión, pero notoria lejos de ella.



**Figura 9.9**  
Los segmentos de curva  $Q_1$ ,  $Q_2$  y  $Q_3$  se unen en el punto  $P_2$  y son idénticos excepto por sus vectores tangente en  $P_2$ .  $Q_1$  y  $Q_2$  tienen vectores tangente idénticos y por lo tanto ambos tienen continuidad  $G^1$  y  $C^1$  en  $P_2$ .  $Q_1$  y  $Q_3$  tienen vectores tangente en la misma dirección, pero  $Q_3$  tiene el doble de magnitud, de manera que sólo tienen continuidad  $G^1$  en  $P_2$ . El vector tangente mayor de  $Q_3$  significa que la curva es jalada más hacia la dirección del vector tangente antes de dirigirse hacia  $P_3$ . El vector  $TV_2$  es el vector tangente de  $Q_2$  mientras que  $TV_3$  es el de  $Q_3$ .

Si dos segmentos de curva se unen, la curva tiene **continuidad geométrica  $G^0$** . Si las direcciones (aunque no necesariamente las magnitudes) de los vectores tangente de las dos curvas son iguales en el punto de unión, la curva tiene continuidad geométrica  $G^1$ . En el diseño de objetos asistido por computador, con frecuencia se requiere continuidad  $G^1$  entre segmentos de curva. La continuidad  $G^1$  significa que las pendientes geométricas de los segmentos son iguales en el punto de unión. Para que dos vectores tangente  $TV_1$  y  $TV_2$  tengan la misma dirección, es necesario que uno sea múltiplo escalar del otro:  $TV_1 = k \cdot TV_2$ , donde  $k > 0$  [BARS88].

Si los vectores tangente de dos segmentos de curva cúbica son iguales (es decir, si sus direcciones y sus magnitudes son iguales) en el punto de unión de los segmentos, la curva tiene continuidad de primer grado en el parámetro  $t$ , o **continuidad paramétrica**, y se dice que tiene continuidad  $C^1$ . Si la dirección y la magnitud de  $d^n/dt^n[Q(t)]$  hasta la  $n$ -ésima derivada son iguales en el punto de unión, se dice que la curva tiene **continuidad  $C^n$** . En la figura 9.8 se presenta una curva con tres grados de continuidad diferentes. Observe que un segmento de curva paramétrica siempre es continuo en todos sus puntos; la continuidad que nos interesa aquí es la que existe en los puntos de unión.

El vector tangente  $Q'(t)$  es la *velocidad* de un punto en la curva con respecto al parámetro  $t$ . En forma similar, la segunda derivada de  $Q(t)$  es la *aceleración*. Suponga que se mueve una cámara por la curva cúbica paramétrica en incrementos de tiempo iguales y que se toma una fotografía después de cada incremento; el vector tangente indica la velocidad de la cámara a lo largo de la curva. Para evitar los movimientos irregulares en la secuencia de animación, la velocidad de la cámara y la aceleración en los puntos de unión deben ser continuas. Esta continuidad de la aceleración por el punto de unión de la figura 9.8 es la que hace que la curva  $C^2$  continúe más hacia la derecha que la curva  $C^1$  antes de doblar hacia el punto extremo.

En términos generales, la continuidad  $C^1$  implica  $G^1$ , pero no siempre se aplica la situación inversa. Es decir, la continuidad  $G^1$  usualmente es menos restrictiva que  $C^1$ , de manera que las curvas pueden tener continuidad  $G^1$  pero no necesariamente  $C^1$ . Sin embargo, los puntos de unión con continuidad  $G^1$  aparecerán tan suaves como los que poseen continuidad  $C^1$ , como puede verse en la figura 9.9.

El gráfico de una curva paramétrica es muy distinto del gráfico de una función ordinaria, en el sentido de que la variable independiente se grafica sobre el eje  $x$  y la variable dependiente sobre el eje  $y$ . En los gráficos de curvas paramétricas nunca se grafica la variable independiente  $t$ . Por lo tanto, no es posible determinar el vector tangente a la curva con sólo observar el gráfico de la curva paramétrica. Es posible determinar la dirección del vector, pero no su magnitud. Se podrá comprender por qué ocurre esto si se considera de la siguiente manera: si  $\gamma(t)$ ,  $0 \leq t \leq 1$ , es una curva paramétrica, su vector tangente en el instante 0 es  $\gamma'(0)$ . Si  $\eta(t) = \gamma(2t)$ ,  $0 \leq t \leq \frac{1}{2}$ , los gráficos paramétricos de  $\eta$  son idénticos. Por otra parte,  $\eta'(0) = 2\gamma'(0)$ . Por lo tanto, dos curvas cuyos gráficos son idénticos pueden tener vectores tangente diferentes. Este hecho es

base de la definición de la **continuidad geométrica**: para que dos curvas se unan suavemente, sólo se requiere que concuerden las direcciones de sus vectores tangente; no es necesario que concuerden sus magnitudes.

**Relación con las restricciones.** Un segmento de curva  $Q(t)$  se define con restricciones aplicables a los puntos extremos, vectores tangente y continuidad entre segmentos de curva. Cada polinomio cúbico de la ecuación (9.5) tiene cuatro coeficientes, de manera que se requieren cuatro restricciones, lo que nos permite formular cuatro ecuaciones para las cuatro incógnitas y resolver éstas. Los tres tipos principales de curvas que se analizan en esta sección son las de **Hermite** o **hermitianas**, definidas por dos puntos extremos y dos vectores tangente de los puntos extremos; las de **Bézier**, definidas por dos puntos extremos y otros dos puntos que controlan los vectores tangente de los puntos extremos; y varios tipos de **splines**, cada una de ellas definida por cuatro puntos de control. Las *splines* tienen continuidad  $C^1$  y  $C^2$  en los puntos de unión y se acercan a sus puntos de control, pero generalmente no los interpolan. Los tipos de *splines* son las **B-splines** uniformes y las **B-splines** no uniformes.

Para ver la forma en que los coeficientes de la ecuación (9.5) pueden depender de las cuatro restricciones, recordemos que una curva cónica paramétrica se define con  $Q(t) = C \cdot T$ . Reescribimos la matriz de coeficientes como  $C = G \cdot M$ , donde  $M$  es una matriz base  $4 \times 4$  y  $G$  es una matriz de cuatro elementos de restricciones geométricas, llamada matriz geométrica. Las restricciones geométricas son las condiciones, como los puntos extremos o los vectores tangente, que definen la curva. Usaremos  $G_x$  para referirnos al vector fila de los componentes  $x$  de la matriz geométrica;  $G_y$  y  $G_z$  tienen significado similar.  $G$  o  $M$ , o bien tanto  $G$  como  $M$ , difieren para cada tipo de curva.

Los elementos de  $G$  y  $M$  son constantes, de manera que el producto  $G \cdot M \cdot T$  es tres polinomios cúbicos en  $t$ . Al expandir el producto  $Q(t) = G \cdot M \cdot T$  se obtiene

$$Q(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = [G_1 \quad G_2 \quad G_3 \quad G_4] \begin{bmatrix} m_{11} & m_{21} & m_{31} & m_{41} \\ m_{12} & m_{22} & m_{32} & m_{42} \\ m_{13} & m_{23} & m_{33} & m_{43} \\ m_{14} & m_{24} & m_{34} & m_{44} \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}. \quad (9.9)$$

Esta ecuación se puede leer en otra forma: el punto  $Q(t)$  es una suma ponderada de las columnas de la matriz geométrica  $G$ , cada una de las cuales representa un punto o un vector en el espacio tridimensional.

Al multiplicar sólo  $x(t) = G_x \cdot M \cdot T$  se obtiene

$$x(t) = (t^3 m_{11} + t^2 m_{21} + t m_{31} + m_{41}) g_{1x} + (t^3 m_{12} + t^2 m_{22} + t m_{32} + m_{42}) g_{2x} + (t^3 m_{13} + t^2 m_{23} + t m_{33} + m_{43}) g_{3x} + (t^3 m_{14} + t^2 m_{24} + t m_{34} + m_{44}) g_{4x}. \quad (9.10)$$

La ecuación (9.10) hace énfasis en que la curva es una suma ponderada de los elementos en la matriz geométrica. Las ponderaciones son polinomios cúbicos

de  $t$  y se denominan **funciones de mezcla**. Las funciones de mezcla  $B$  se obtienen de  $B = M \cdot T$ . Observe la similitud con la aproximación lineal por trozos, donde sólo se requieren dos restricciones geométricas (los puntos extremos de la línea), de manera que cada segmento de curva es una línea recta definida por los puntos extremos  $G_1$  y  $G_2$ :

$$x(t) = g_{1x}(1-t) + g_{2x}(t),$$

$$y(t) = g_{1y}(1-t) + g_{2y}(t),$$

$$z(t) = g_{1z}(1-t) + g_{2z}(t), \quad (9.11)$$

Las cúbicas paramétricas en realidad son sólo una generalización de las aproximaciones de líneas rectas. La curva cónica  $Q(t)$  es una combinación de las *cuatro* columnas de la matriz geométrica, así como un segmento de línea es una combinación de *dos* vectores columna.

Para ver cómo se calcula la matriz base  $M$ , pasamos ahora a formas específicas de curvas cúbicas paramétricas.

### 9.2.2 Curvas de Hermite

La forma de Hermite o hermitiana (llamada así en honor del matemático) de segmento de curva polinomial cónica está determinada por las restricciones de los puntos extremos  $P_1$  y  $P_4$  y los vectores tangente en los puntos extremos  $R_1$  y  $R_4$ . (Se usan los índices 1 y 4, en lugar de 1 y 2, por cuestiones de consistencia con secciones subsecuentes, donde se usarán los puntos intermedios  $P_2$  y  $P_3$ , en lugar de vectores tangente, para definir la curva).

Para hallar la **matriz base hermitiana**  $M_H$ , que relaciona el **vector geométrico hermitiano**  $G_H$  con los coeficientes de los polinomios, escribimos cuatro ecuaciones, una para cada restricción, en los cuatro coeficientes polinomial desconocidos y resolvemos las incógnitas.

Si definimos  $G_H$ , el componente  $x$  de la matriz geométrica hermitiana, como

$$G_H = [P_1 \ P_4 \ R_1 \ R_4], \quad (9.12)$$

y reescribimos  $x(t)$  de las ecuaciones (9.5) y (9.9) como

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x = C_x \cdot T = G_H \cdot M_H \cdot T = G_H \cdot M_H [t^3 \ t^2 \ t \ 1]^T, \quad (9.13)$$

las restricciones de  $x(0)$  y  $x(1)$  se pueden encontrar por sustitución directa en la ecuación (9.13):

$$x(0) = P_1 = G_H \cdot M_H [0 \ 0 \ 0 \ 1]^T, \quad (9.14)$$

$$x(1) = P_4 = G_H \cdot M_H [1 \ 1 \ 1 \ 1]^T. \quad (9.15)$$

Así como en el caso general diferenciamos la ecuación (9.7) para hallar la ecuación (9.8), ahora diferenciamos la ecuación (9.13) para encontrar  $x'(t) = G_H \cdot M_H [3t^2 \ 2t \ 1 \ 0]^T$ . Por lo tanto, las ecuaciones de restricción de vectores tangentes se pueden escribir como

$$x'(0) = R_1 = G_H \cdot M_H [0 \ 0 \ 1 \ 0]^T, \quad (9.16)$$

$$x'(1) = R_4 = G_H \cdot M_H [3 \ 2 \ 1 \ 0]^T. \quad (9.17)$$

Las cuatro restricciones de las ecuaciones (9.14)-(9.17) se pueden reescribir en forma matricial:

$$\begin{bmatrix} P_1 & P_4 & R_1 & R_4 \end{bmatrix} = G_H = G_{H_i} \cdot M_H \cdot \begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}. \quad (9.18)$$

Para que se satisfaga esta ecuación (y las expresiones correspondientes de  $y$  y  $z$ ),  $M_H$  debe ser la inversa de la matriz de  $4 \times 4$  en la ecuación (9.18):

$$M_H = \begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}. \quad (9.19)$$

Ahora podemos usar  $M_H$  en  $x(t) = G_H \cdot M_H \cdot T$  para hallar  $x(t)$  con base en el vector de geometría  $G_H$ . En forma similar,  $y(t) = G_H \cdot M_H \cdot T$  y  $z(t) = G_H \cdot M_H \cdot T$ , por lo que podemos escribir

$$Q(t) = [x(t) \ y(t) \ z(t)]^T = G_H \cdot M_H \cdot T, \quad (9.20)$$

donde  $G_H$  es la matriz

$$[P_1 \ P_4 \ R_1 \ R_4].$$

Al expandir el producto  $M_H \cdot T$  en  $Q(t) = G_H \cdot M_H \cdot T$  se obtienen las **funciones de mezcla de Hermite  $B_H$**  como polinomios que ponderan cada elemento de la matriz de geometría:

$$\begin{aligned} Q(t) &= G_H \cdot M_H \cdot T = G_H \cdot B_H \\ &= (2t^3 - 3t^2 + 1)P_1 + (-2t^3 + 3t^2)P_4 + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_4. \end{aligned} \quad (9.21)$$

En la figura 9.10 se muestran las cuatro funciones de mezcla. Observe que, en  $t = 0$ , sólo la función  $P_1$  es distinta de cero: únicamente  $P_1$  afecta la curva en

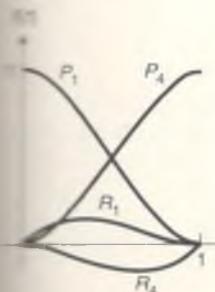
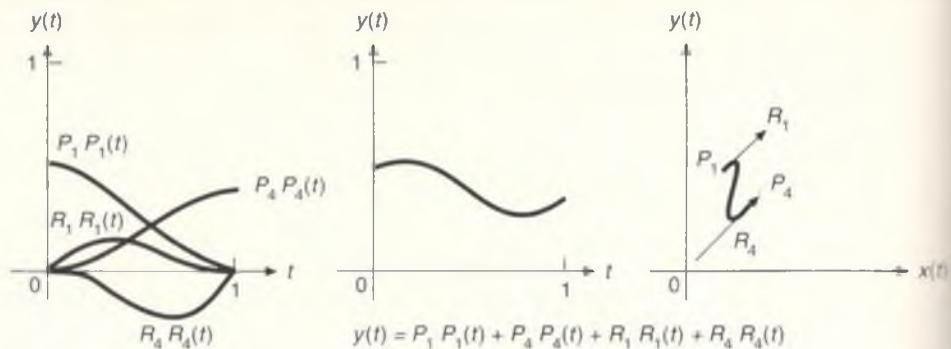


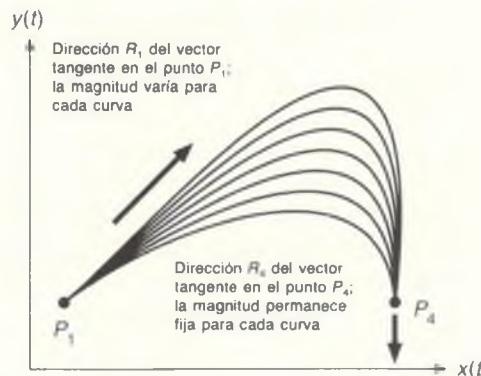
Figura 9.10 Las funciones de mezcla de Hermite, rotuladas con los elementos del vector de geometría que ponderan.



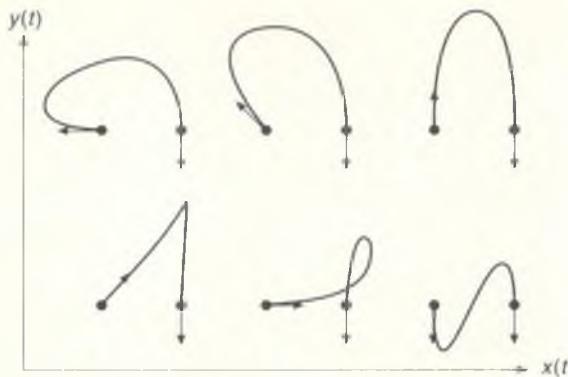
**Figura 9.11** Curva hermitiana que muestra los cuatro elementos del vector de geometría ponderado por las funciones de mezclado (las cuatro curvas de la izquierda), su suma  $y(t)$  y la curva bidimensional (a la derecha).  $x(t)$  se define con una suma ponderada similar.

$t = 0$ . Tan pronto como  $t$  es mayor que cero,  $R_1$ ,  $P_4$  y  $R_4$  comienzan a influir. En la figura 9.11 se presentan las cuatro funciones ponderadas por los componentes  $y$  y de un vector de geometría específico, su suma  $y(t)$  y la curva  $Q(t)$ .

En la figura 9.12 se muestra una serie de curvas hermitianas. La única diferencia entre ellas es la longitud del vector tangente  $R_1$ : las direcciones de los vectores tangente son fijas. Cuanto más aumenta la longitud de los vectores, mayor es su efecto sobre la curva. En la figura 9.13 aparece otra serie de curvas hermitianas, con vectores tangente de longitud constante pero direcciones diferentes. En un sistema gráfico interactivo, los puntos extremos y los vectores tangente de una curva son manipulados por el usuario para dar forma a la curva. En la figura 9.14 se presenta una manera de implantar este tipo de interacción.



**Figura 9.12** Familia de curvas cúbicas paramétricas hermitianas. Sólo  $R_1$ , el vector tangente en  $P_1$ , varía para cada curva, aumentando en magnitud para las curvas superiores.



**Figura 9.13** Familia de curvas cúbicas paramétricas hermitianas. Sólo varía la dirección del vector tangente en el punto de partida de la izquierda; todos los vectores tangente tienen la misma magnitud. Si la magnitud fuera menor, se eliminaría el lazo en la curva.

**Dibujo de curvas paramétricas.** Las curvas cúbicas paramétricas hermitianas y otras similares son fáciles de dibujar: se evalúa la ecuación (9.5) en  $n$  valores sucesivos de  $t$  separados por un incremento  $\delta$ ; el código se presenta en el programa 9.1. La evaluación en (...) del ciclo **for** requiere 11 multiplicaciones y 10 sumas por punto tridimensional. Si se aplica la regla de Horner para la factorización de polinomios,

$$f(t) = at^3 + bt^2 + ct + d = ((at + b)t + c)t + d, \quad (9.22)$$

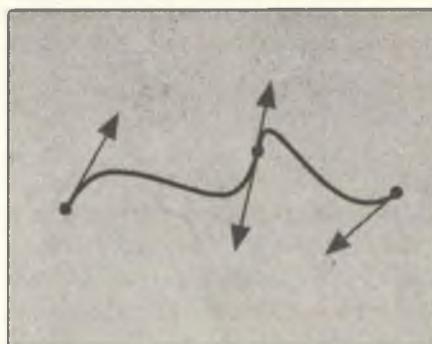
el trabajo se reduciría a nueve multiplicaciones y 10 sumas por punto tridimensional.

En [FOLE90] se presentan formas más eficientes de presentar estas curvas usando técnicas de diferencia hacia adelante.

**Programa 9.1**  
Programa para presentar  
una curva paramétrica  
cúbica.

```
typedef float arreglo_coeficientes[4];
void dibujar_curva (arreglo_coeficientes *cx, arreglo_coeficientes *cy,
                     arreglo_coeficientes *cz, int n)
{
    /* cx, cy y cz son coeficientes para x(t), y(t) y z(t) */
    /* por ejemplo, Cx = Gx · M, etc. */
    /* n es el número de pasos */
    {
        float x, y, z, delta, t, t2, t3;
        int i;

        mov_abs_3 (cx[3], cy[3], cz[3]);
        delta = 1.0 / n;
        for (i = 1; i <= n; i++) {
            t = i * delta;
            t2 = t * t;
            t3 = t2 * t;
            x = cx[0] + cx[1] * t + cx[2] * t2 + cx[3] * t3;
            y = cy[0] + cy[1] * t + cy[2] * t2 + cy[3] * t3;
            z = cz[0] + cz[1] * t + cz[2] * t2 + cz[3] * t3;
            /* Dibujar la curva */
        }
    }
}
```



**Figura 9.14** Dos segmentos de curva cúbica hermitiana presentados con controles para facilitar la manipulación interactiva. El usuario puede cambiar la posición de los puntos extremos arrastrando los puntos, y alterar los vectores tangente arrastrando las puntas de flecha. Los vectores tangente en el punto de unión se limitan a ser colineales (para proporcionar la continuidad  $C^0$ ): el usuario generalmente dispone de un mandato para forzar la continuidad  $C^1$ ,  $C^2$ ,  $G^1$  o establecer que no habrá continuidad. Los vectores tangente en el extremo  $t = 1$  de cada curva se dibujan en sentido contrario al de la dirección usada en la formulación matemática de la curva hermitiana, por cuestiones de claridad y para que sea más sencilla la interacción del usuario.

```

t = i * delta;
t2 = t * t;
t3 = t2 * t;
x = cx[0] * t3 + cx[1] * t2 + cx[2] * t + cx[3];
y = cy[0] * t3 + cy[1] * t2 + cy[2] * t + cy[3];
z = cz[0] * t3 + cz[1] * t2 + cz[2] * t + cz[3];
dibujar_abs_3 (x, y, z);
}
    
```

Como las curvas cúbicas son combinaciones lineales (sumas ponderadas) de los cuatro elementos del vector de geometría, como puede observarse en la ecuación (9.10), podemos transformar las curvas transformando el vector de geometría y luego usándolo para generar la curva transformada, lo que equivale a decir que las curvas son invariantes con la rotación, el escalamiento y la traslación. Esta estrategia es más efectiva que la generación de la curva como serie de segmentos de línea cortos para luego transformar cada línea individual. Las curvas *no* son invariantes en la proyección de perspectiva, como veremos en la sección 9.2.6.

### 9.2.3 Curvas de Bézier

La forma de Bézier [BEZI70; BEZI74] del segmento de curva polinomial cónica, nombrada así en honor de Pierre Bézier, quien la desarrolló para el diseño

de automóviles en la compañía Renault, especifica de manera indirecta el vector tangente de punto extremo a través de la especificación de dos puntos intermedios que no están en la curva (véase la Fig. 9.15). Los vectores tangente inicial y final se determinan con los vectores  $P_1P_2$  y  $P_3P_4$  y se relacionan con  $R_1$  y  $R_4$  mediante la ecuación

$$R_1 = Q'(0) = 3(P_2 - P_1), \quad R_4 = Q'(1) = 3(P_4 - P_3). \quad (9.23)$$

La curva de Bézier interpola los dos puntos de control extremos y aproxima los otros dos. Consulte el ejercicio 9.9 para saber por qué se usa la constante 3 en la ecuación (9.23). La **matriz de geometría de Bézier**  $G_B$ , que consiste en cuatro puntos, es

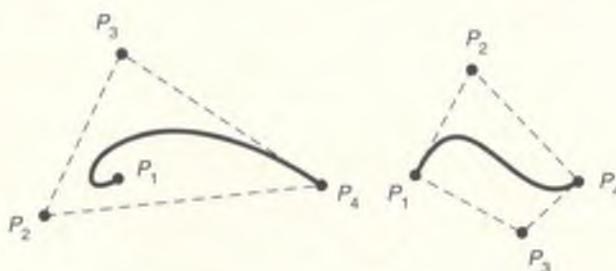
$$G_B = [P_1 \ P_2 \ P_3 \ P_4]. \quad (9.24)$$

Por lo tanto, la matriz  $M_{HB}$  que define la relación  $G_H = G_B \cdot M_{HB}$  entre la matriz de geometría hermitiana  $G_H$  y la matriz de geometría de Bézier  $G_B$ , no es más que la matriz de  $4 \times 4$  en la siguiente ecuación, que reescribe la ecuación (9.24) en forma matricial:

$$G_H = [P_1 \ P_4 \ R_1 \ R_4] = [P_1 \ P_2 \ P_3 \ P_4] \begin{bmatrix} 1 & 0 & -3 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 3 \end{bmatrix} = G_B \cdot M_{HB}. \quad (9.25)$$

Para hallar la **matriz base de Bézier**  $M_B$ , se emplea la ecuación (9.20) para la forma hermitiana, se sustituye  $G_H = M_{HB} \cdot G_B$  y se define  $M_B = M_H \cdot M_{HB}$ :

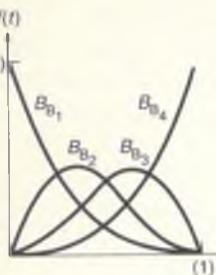
$$Q(t) = G_H \cdot M_H \cdot T = (G_B \cdot M_{HB}) \cdot M_H \cdot T = G_B \cdot (M_{HB} \cdot M_H) \cdot T = G_B \cdot M_B \cdot T. \quad (9.26)$$



**Figura 9.15** Dos curvas de Bézier y sus puntos de control. Observe que la envolvente convexa (el mínimo polígono convexo que contiene a los puntos de control), presentadas con líneas punteadas, no tienen que tocar los cuatro puntos de control.

Al desarrollar la multiplicación  $M_B = M_{HB} \cdot M_H$  se obtiene

$$M_B = M_{HB} \cdot M_H = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad (9.27)$$



y el producto  $Q(t) = G_B \cdot M_B \cdot T$  es

$$Q(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t)P_3 + t^3 P_4. \quad (9.28)$$

**Figura 9.16** Los polinomios de Bernstein, que son las funciones de ponderación para las curvas de Bézier. En  $t = 0$ , sólo  $B_{B_1}$  es distinto de cero, de manera que la curva interpola  $P_1$ ; en forma similar, en  $t = 1$  sólo  $B_{B_4}$  es distinto de cero y la curva interpola  $P_4$ .

Los cuatro polinomios  $B_B = M_B \cdot T$ , que son las ponderaciones en la ecuación (9.28), se denominan **polinomios de Bernstein** y se presentan en la figura 9.16.

**Unión de segmentos de curva.** En la figura 9.17 se presentan dos segmentos de curva de Bézier con un punto extremo común. Hay continuidad  $G^1$  en el punto extremo cuando  $P_3 - P_4 = k(P_4 - P_5)$ ,  $k > 0$ . Es decir, los tres puntos  $P_3$ ,  $P_4$  y  $P_5$  deben ser distintos y colineales. En el caso más restrictivo de  $k = 1$ , hay continuidad  $C^1$  además de la  $G^1$ .

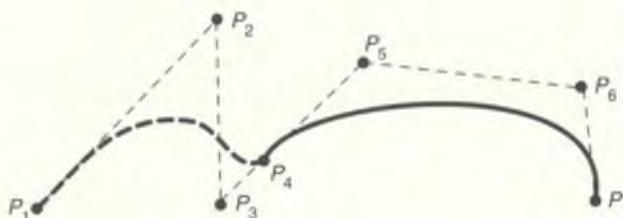
Si nos referimos a los polinomios de los dos segmentos de curva como  $x^l$  (para el segmento izquierdo) y  $x^r$  (para el segmento derecho), podemos encontrar las condiciones para la continuidad  $C^0$  y  $C^1$  en su punto de unión:

$$x^l(1) = x^r(0), \quad \frac{d}{dt}x^l(1) = \frac{d}{dt}x^r(0). \quad (9.29)$$

Al trabajar con el componente  $x$  de la ecuación (9.29) tenemos

$$x^l(1) = x^r(0) = P_4, \quad \frac{d}{dt}x^l(1) = 3(P_4 - P_3), \quad \frac{d}{dt}x^r(0) = 3(P_5 - P_4). \quad (9.30)$$

Como siempre, se aplican las mismas condiciones a  $y$  y  $z$ . De esta manera, tenemos continuidad  $C^0$  y  $C^1$  cuando  $P_4 - P_3 = P_5 - P_4$ , como se esperaba.



**Figura 9.17** Dos curvas de Bézier unidas en  $P_4$ . Los puntos  $P_3$ ,  $P_4$  y  $P_5$  son colineales.

**Importancia de la envolvente convexa.** Al examinar en la figura 9.16 los cuatro polinomios  $B_8$  de la ecuación (9.28), observamos que su suma siempre es la unidad y que cada polinomio siempre es no negativo para  $0 \leq t < 1$ . Por lo tanto,  $Q(t)$  es simplemente un promedio ponderado de los cuatro puntos de control. Esta condición significa que cada segmento de curva, que a su vez es la suma de cuatro puntos de control ponderados por los polinomios, está totalmente contenida en la **envolvente convexa** de los cuatro puntos de control. La envolvente convexa de las curvas bidimensionales es el mínimo polígono convexo que contiene a los cuatro puntos de control: piense en este polígono como el que se obtendría al colocar una liga elástica alrededor de los puntos (Fig. 9.15). En el caso de las curvas tridimensionales, la envolvente convexa es el mínimo poliedro convexo que contiene por los puntos de control: piense que este poliedro es como el que se obtendría al extender un papel plástico elástico alrededor de los cuatro puntos.

Esta propiedad de envolvente convexa es válida para todas las cúbicas definidas por la suma ponderada de puntos de control si las funciones de mezcla no son negativas y su suma es uno. En términos generales, el promedio ponderado de  $n$  puntos está dentro de la envolvente convexa de  $n$  puntos; esto se puede observar en forma intuitiva para  $n = 2$  y  $n = 3$ , y de aquí se desprende la generalización. Otra consecuencia de que los cuatro polinomios sumen la unidad es que podemos determinar el valor del cuarto polinomio para cualquier valor de  $t$  restando los tres primeros a la unidad, un hecho que podemos aprovechar para reducir el tiempo de computación.

La propiedad de la envolvente convexa también es útil para recortar segmentos de curva: en lugar de recortar cada trozo de línea de un segmento curvo para determinar su visibilidad, se aplica primero un algoritmo de recorte de polígonos, por ejemplo el de Sutherland-Hodgman analizado en el capítulo 3, para recortar la envolvente convexa o su extensión con respecto a la región de recorte. Si la envoltura convexa (extensión) se encuentra totalmente dentro de la región de recorte, también lo estará todo el segmento de curva. Si la envoltura convexa (extensión) está totalmente fuera de la región de recorte, todo el segmento de curva también estará fuera de ella. Sólo hay que examinar el segmento de curva si la envolvente convexa (extensión) interseca la región de recorte.

### Ejemplo 9.2

**Problema:** Escriba un programa, usando SRGP, que permita al usuario especificar los cuatro puntos de control para una curva bidimensional de Bézier y que luego dibuje la curva usando el método del programa 9.1. Usted debe proporcionar mecanismos para especificar un número arbitrario de curvas de Bézier, limpiar la ventana de SRGP y terminar el programa.

**Respuesta:** Implantamos la función *dibujar\_curva* usando la ecuación (9.28), que relaciona la curva  $Q(t)$  con los cuatro puntos de control. En términos generales, esta implantación sacrifica eficiencia para obtener claridad. Sin embargo,

empleamos la función SRGP\_\_polyLine, que es la forma más eficiente de dibujar la curva. El resto de la implantación sigue el modelo del programa 9.1.

Hemos especificado arbitrariamente el tamaño de la ventana y el número de pasos para aproximar la curva, con valores de 400 y 20 respectivamente. Hay varias formas de implantar la parte interactiva del programa, pero hemos elegido una combinación de dispositivos localizador y de teclado. El botón derecho del localizador sirve para especificar el inicio de una nueva secuencia de puntos de control, mientras que el izquierdo se utiliza para definir los tres puntos restantes. Un eco de línea elástica ayuda a guiar la distribución de los puntos. La curva de Bézier se dibuja en cuanto se especifica el último punto.

Finalmente, la ventana se limpia cuando el usuario oprime la tecla “I”; al oprimir la tecla “s” termina el programa. En la figura adjunta se presenta un conjunto típico de curvas producidas por el programa.

*Programa interactivo de curva de Bézier.*

```
#include "srqp.h"
#include <stdio.h>

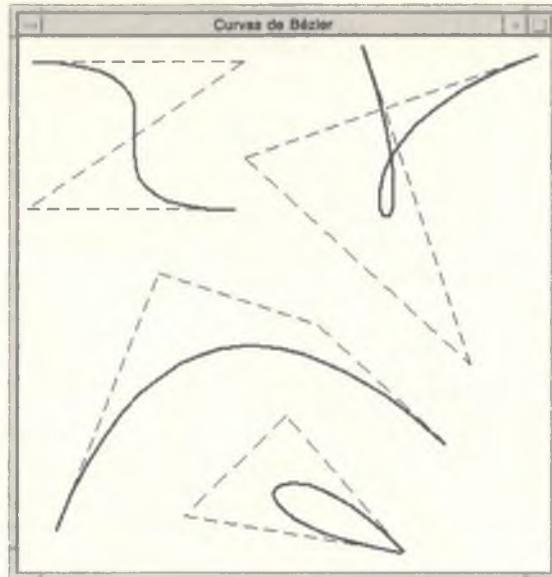
#define TAMAÑO_MEDIDA_TECLA 80
#define TAMAÑO_VENTANA 400
#define NÚM_PASOS 20

void dibujar_curva (point *puntos_control, int n)
{
    int i;
    float t, delta;
    point puntos_curva[n];

    puntos_curva[0].x = puntos_control[0].x; /* Las curvas de Bézier interpolan el primero */
    puntos_curva[0].y = puntos_control[0].y; /* y el último de los puntos de control */
    delta = 1.0 / n; /* La curva se aproximarán con n puntos */
    /* t está en el intervalo 0.0 a 1.0 */

    for(i = 1; i <= n; i++) {
        t = i * delta;
        puntos_curva[i].x = puntos_control[0].x * (1.0 - t) * (1.0 - t) * (1.0 - t)
            + puntos_control[1].x * 3.0 * t * (1.0 - t) * (1.0 - t)
            + puntos_control[2].x * 3.0 * t * t * (1.0 - t)
            + puntos_control[3].x * 3.0 * t * t * t;

        puntos_curva[i].y = puntos_control[0].y * (1.0 - t) * (1.0 - t) * (1.0 - t)
            + puntos_control[1].y * 3.0 * t * (1.0 - t) * (1.0 - t)
            + puntos_control[2].y * 3.0 * t * t * (1.0 - t)
            + puntos_control[3].y * 3.0 * t * t * t;
    }
    SRGP_polyLine (n + 1, puntos_curva);/* Dibujar la curva completa */
}
```



Salida típica del programa de curvas de Bézier

```

main ( )
{
    locator_measure medida_loc, medida_loc_ant;
    char medida_tecla[TAMAÑO_MEDIDA_TECLA];
    int dispositivo;
    int num_ctl;
    boolean fin;
    rectangle pantalla;
    point puntos_control[4];

    SRGP_begin ("Curvas de Bézier", TAMAÑO_VENTANA, TAMAÑO_VENTANA, 1,
                FALSE);
    SRGP_setLocatorEchoType (CURSOR);
    SRGP_setLocatorButtonMask (LEFT_BUTTON_MASK | RIGHT_BUTTON_MASK);
    medida_loc_ant.position = SRGP_defPoint (-1, -1); /* Valor inicial arbitrario para la
                                                posición */
    SRGP_setLocatorMeasure (medida_loc_ant.position);
    SRGP_setKeyboardProcessingMode (RAW);
    SRGP_setInputMode (LOCATOR, EVENT);/* Se activan el localizador (ratón) y el teclado */
    SRGP_setInputMode (KEYBOARD, EVENT);
    pantalla = SRGP_defRectangle (0, 0, TAMAÑO_VENTANA - 1, TAMAÑO_VENTANA - 1);

    /* Lazo principal de eventos */
    fin = FALSE;
    do {
        dispositivo = SRGP_waitEvent (INDEFINITE);
        switch (dispositivo){
            case KEYBOARD: {

```

```

SRGP_getKeyboard (medida_tecla, TAMAÑO_MEDIDA_TECLA);
switch (medida_tecla [0]){
    case 's':                                /* Salir del programa */
        fin = TRUE;
        break;
    case 'l':                                /* Limpiar la ventana */
        SRGB_setColor (0);
        SRGB_fillRectangle (pantalla);
        SRGBSetColor (1);
        break;
    }
    break;
}
/* case del teclado */

case LOCATOR: {
    SRGB_getLocator (&medida_loc);
    switch (medida_loc.buttonOfMostRecentTransition) {
        case LEFT_BUTTON:/* Definición de los puntos de control restantes */
            if ((medida_loc.buttonChord[LEFT_BUTTON] == DOWN) &&
                medida_loc_ant.position.x > 0) {
                SRGB_setLocatorEchoRubberAnchor (medida_loc.position);
                SRGB_line (medida_loc_ant.position, medida_loc.position);
                medida_loc_ant = medida_loc;
                puntos_control[num_ctl] = medida_loc.position;
                num_ctl++;
                if (num_ctl == 4) {
                    SRGB_setLineStyle (CONTINUOUS); /* Para dibujar la curva
                    SRGB_setLineWidth (2);
                    dibujar_curva (puntos_control, NUM_PASOS);
                    medida_loc_ant.position.x = -1;
                    SRGB_setLocatorEchoType (CURSOR);
                }
                break;
            }
        case RIGHT_BUTTON: /* Iniciar nuevo conjunto de puntos de control */
            SRGB_setLocatorEchoRubberAnchor (medida_loc.position);
            medida_loc_ant = medida_loc;
            SRGB_setLocatorEchoType (RUBBER_LINE);
            SRGB_setLineStyle (DASHED); /* Para dibujar el polígono */
            control =
            SRGB_setLineWidth (1);
            puntos_control[0] = medida_loc.position;
            num_ctl = 1;
            break;
        }
    }
}
/* manejo de botones */
/* case del localizador */
/* switch de dispositivo */
}

while (!fin);
SRGP_end ( );

```

### 9.2.4 B-splines uniformes, no racionales

El término *spline* proviene de las largas tiras flexibles de metal que usaban los dibujantes para establecer las superficies de aeroplanos, automóviles y barcos. A las *splines* se le colocaban pesos que servían para jalar la *spline* en varias direcciones. A menos que se sometieran a grandes tensiones, las *splines* metálicas tenían continuidad de segundo orden. El equivalente matemático de estas tiras, la *spline cúbica natural*, es un polinomio cúbico con continuidad  $C^0$ ,  $C^1$  y  $C^2$  que interpola (pasa por) los puntos de control. Este polinomio tiene un grado de continuidad más que la continuidad inherente en las formas de Hermite y Bézier. Por lo tanto, las *splines* son más suaves que las formas anteriores.

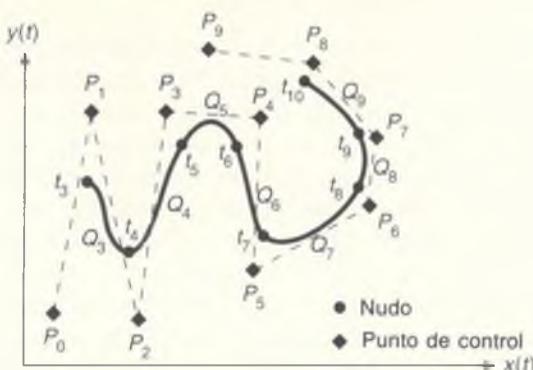
Sin embargo, los coeficientes polinomiales para las *splines* cúbicas naturales dependen de los  $n$  puntos de control y su cálculo implica invertir una matriz de  $n + 1$  por  $n + 1$  [BART87]. Esta característica tiene dos desventajas: al mover un punto de control se afecta toda la curva, y el tiempo de computación necesario para invertir la matriz puede interferir con el cambio rápido de la forma de una curva.

Las *B-splines* que analizamos en esta sección, consisten en segmentos de curva cuyos coeficientes polinomiales dependen de unos cuantos puntos de control. Ese comportamiento se conoce como **control local**. Por lo tanto, mover un punto de control afecta sólo una pequeña parte de la curva. Además, se reduce en forma considerable el tiempo que se requiere para calcular los coeficientes. Las *B-splines* tienen la misma continuidad que las *splines* naturales, pero no interpolan sus puntos de control.

En el siguiente análisis modificaremos ligeramente nuestra notación, ya que debemos examinar una curva completa, consistente en varios segmentos de curva, y no sus segmentos individuales. Un segmento de curva no tiene que pasar por sus puntos de control, y las dos condiciones de continuidad en un segmento provienen de los segmentos adyacentes. Este comportamiento surge de compartir puntos de control entre segmentos, por lo que es más conveniente describir el proceso en función de todos los segmentos al mismo tiempo.

Las *B-splines* cúbicas aproximan una serie de  $m + 1$  puntos de control  $P_0, P_1, \dots, P_m$ ,  $m \geq 3$ , con una curva que consiste en  $m - 2$  segmentos de curva polinomial cúbica  $Q_3, Q_4, \dots, Q_m$ . Aunque estas curvas cúbicas se pueden definir en su propio dominio  $0 \leq t < 1$ , podemos ajustar el parámetro (con una sustitución de la forma  $t = t + k$ ) de manera que los dominios de parámetros de los segmentos de curva sean secuenciales. Por lo tanto, decimos que el intervalo de parámetros donde está definido  $Q_i$  es  $t_i \leq t < t_{i+1}$ , para  $3 \leq i \leq m$ . En el caso específico de  $m = 3$ , hay un solo segmento de curva  $Q_3$ , que está definido en el intervalo  $t_3 \leq t < t_4$  por cuatro puntos de control,  $P_0$  a  $P_3$ .

Para cada  $i \geq 4$  hay un punto de unión o **nudo** entre  $Q_{i-1}$  y  $Q_i$  en el valor  $t_i$  del parámetro; el valor del parámetro en este punto se conoce como **valor de nudo**. Los puntos inicial y final en  $t_3$  y  $t_{m+1}$  también se denominan nudos, por lo que hay un total de  $m - 1$  nudos. En la figura 9.18 se muestran una curva bidimensional de *spline B* y sus nudos. Es fácil crear una curva cerrada de *spline*



**Figura 9.18** *Spline B con segmentos de curva  $Q_3$  a  $Q_9$ . Esta figura y muchas más de este capítulo se crearon con un programa escrito por Charles Castellsaqué.*

B: los puntos de control  $P_0, P_1, P_2$  se repiten al final de la secuencia:  $P_0, P_1, \dots, P_m, P_0, P_1, P_2$ .

El término **uniforme** quiere decir que los nudos están espaciados a intervalos iguales del parámetro  $t$ . Podemos suponer, sin perder generalidad, que  $t_1 = 0$  y que el intervalo  $t_{i+1} - t_i = 1$ . En la sección 9.2.5 se analizan las *B-splines* no uniformes, no racionales, que permiten el espaciado desigual entre nudos. (De hecho, el concepto de nudo se ha introducido en esta sección a fin de preparar el terreno para las *splines* no uniformes.) El término **no racional** se emplea para distinguir estas *splines* de las curvas polinomiales cúbicas racionales, analizadas en la sección 9.2.6, donde  $x(t), y(t)$  y  $z(t)$  se definen como la razón de dos polinomios cúbicos. La “B” representa la base, ya que las *splines* se pueden representar como sumas ponderadas de funciones base polinomiales, a diferencia de las *splines* naturales, donde no es aplicable la propiedad de la suma ponderada.

Cada uno de los  $m - 2$  segmentos de una curva de *B-spline* está definido por cuatro de los  $m + 1$  puntos de control. Específicamente, el segmento de curva  $Q_i$  está definido por los puntos  $P_{i-3}, P_{i-2}, P_{i-1}$  y  $P_i$ . Por consiguiente, la **matriz de geometría de B-spline**  $G_{Bs}$  para el segmento  $Q_i$  es

$$G_{Bs} = [P_{i-3} \ P_{i-2} \ P_{i-1} \ P_i], \quad 3 \leq i \leq m. \quad (9.31)$$

El primer segmento de curva,  $Q_3$ , está definido por los puntos  $P_0$  a  $P_3$ , en el intervalo de parámetros  $t_3 = 0$  a  $t_4 = 1$ ;  $Q_4$  está definido por los puntos  $P_1$  a  $P_4$ , en el intervalo de parámetros  $t_4 = 1$  a  $t_5 = 2$ ; y el último segmento de curva,  $Q_m$ , está definido por los puntos  $P_{m-3}, P_{m-2}, P_{m-1}$  y  $P_m$  en el intervalo de parámetros  $t_m = m - 3$  a  $t_{m+1} = m - 2$ . Por lo general, el segmento de curva  $Q_i$  comienza en un lugar cercano al punto  $P_{i-2}$  y termina cerca del punto  $P_{i-1}$ . Veremos que las funciones de mezcla de *B-splines* son no negativas en todos los

puntos y que suman la unidad, de manera que el segmento de curva  $Q_i$  está limitado a la envolvente convexa de sus cuatro puntos de control.

Así como cada segmento de curva está definido por cuatro puntos de control, cada punto de control (excepto el inicial y el final de la secuencia  $P_0, P_1, \dots, P_m$ ) influye en cuatro segmentos de curva. Al mover un punto de control en una dirección determinada se mueven en la misma dirección los cuatro segmentos de curva a los cuales afecta; los demás segmentos de curva no son afectados en absoluto (véase la Fig. 9.19). Este comportamiento es la propiedad de control local de las *B-splines* y de las demás *splines* que se analizan en este capítulo.

Si definimos  $T_i$  como el vector columna  $[(t - t_i)^3 \ (t - t_i)^2 \ (t - t_i) \ 1]^T$ , entonces la formulación de *B-spline* para un segmento de curva es

$$Q_i(t) = G_{Bs_i} = M_{Bs} \cdot T_i, \quad t_i \leq t \leq t_{i+1}. \quad (9.32)$$

La curva completa se genera aplicando la ecuación (9.32) con  $3 \leq i \leq m$ .

La matriz base de *B-spline*,  $M_{Bs}$ , relaciona las restricciones geométricas  $G_{Bs}$  con las funciones de mezcla y los coeficientes polinomiales:

$$M_{Bs} = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}. \quad (9.33)$$

Esta matriz se deriva en [BART87].

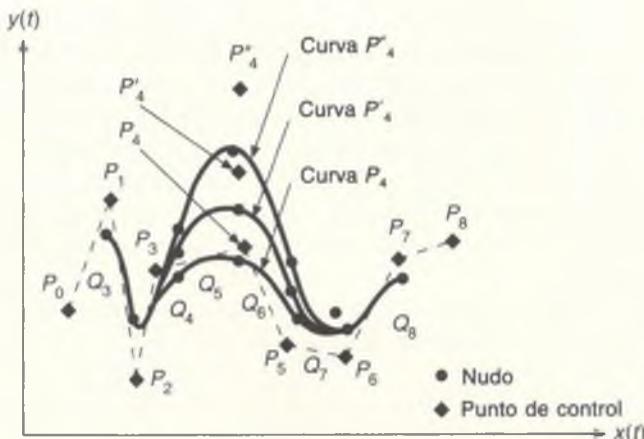
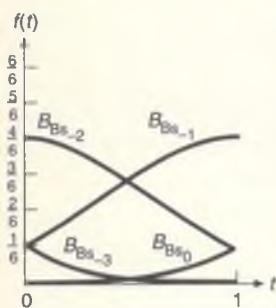


Figura 9.19 *B-Spline* con el punto de control  $P_4$  en distintos lugares.

Las funciones de mezclado de *B-spline*,  $B_{Bs}$ , se obtienen del producto  $M_{Bs} \cdot T_i$ , en forma análoga a como se obtuvieron en las formulaciones de Bézier y de Hermite. Observe que las funciones de mezcla de cada segmento de curva son exactamente iguales ya que, para cada segmento  $i$ , los valores de  $t - t_i$  se encuentran en el intervalo de 0 en  $t = t_i$  a 1 en  $t = t_{i+1}$ . Si reemplazamos  $t - t_i$  con  $t$  y el intervalo  $[t_i, t_{i+1}]$  por  $[0, 1]$ , obtenemos

$$\begin{aligned} B_{Bs} &= M_{Bs} \cdot T = [B_{Bs-3} \quad B_{Bs-2} \quad B_{Bs-1} \quad B_{Bs0}]^T \\ &= \frac{1}{6} [-t^3 + 3t^2 - 3t + 1 \quad 3t^3 - 6t^2 + 4 \quad -3t^3 + 3t^2 + 3t + 1 \quad t^3]^T \\ &= \frac{1}{6} [(1-t)^3 \quad 3t^3 - 6t^2 + 4 \quad -3t^3 + 3t^2 + 3t + 1 \quad t^3]^T, \quad 0 \leq t < 1. \quad (9.34) \end{aligned}$$



**Figura 9.20** Las cuatro funciones de mezcla de *B-spline* de la ecuación (9.34). Sólo tres de las funciones son distintas de cero en  $t = 0$  y en  $t = 1$ .

En la figura 9.20 se muestran las funciones de mezcla de *B-spline*  $B_{Bs}$ . Como las cuatro funciones suman 1 y no son negativas, la propiedad de envolvente convexa es válida para cada uno de los segmentos de curva de una *B-spline*. Consulte [BART87] para comprender la relación entre estas funciones de mezcla y las funciones base de polinomios de Bernstein.

Al expandir la ecuación (9.32), reemplazando  $t - t_i$  por  $t$  en el segundo signo de igualdad, obtenemos

$$\begin{aligned} Q_i(t - t_i) &= G_{Bs_i} \cdot M_{Bs} \cdot T_i = G_{Bs_i} \cdot M_{Bs} \cdot T \\ &= G_{Bs_i} \cdot B_{Bs} = P_{i-3} \cdot B_{Bs-3} + P_{i-2} \cdot B_{Bs-2} + P_{i-1} \cdot B_{Bs-1} + P_i \cdot B_{Bs0} \\ &= \frac{(1-t)^3}{6} P_{i-3} + \frac{3t^3 - 6t^2 + 4}{6} P_{i-2} + \frac{-3t^3 + 3t^2 + 3t + 1}{6} P_{i-1} \\ &\quad + \frac{t^3}{6} P_i, \quad 0 \leq t < 1. \quad (9.35) \end{aligned}$$

Es fácil demostrar que  $Q_i$  y  $Q_{i+1}$  tienen continuidad  $C^0$ ,  $C^1$  y  $C^2$  en el punto donde se unen. La continuidad adicional que ofrecen las *B-splines* es atractiva, pero se obtiene a expensas de lograr menor control respecto a dónde pasa la curva. Podemos forzar a que la curva interpole puntos específicos si duplicamos los puntos de control; esto es conveniente en los puntos extremos e intermedios de la curva. Por ejemplo, si  $P_{i-2} = P_{i-1}$ , la curva se acerca a este punto porque el segmento de curva está definido sólo por tres puntos diferentes, y el punto  $P_{i-2} = P_i$  es ponderado dos veces en la ecuación (9.35), una vez por  $B_{Bs-2}$  y otra por  $B_{Bs-1}$ .

Si un punto de control se emplea tres veces (p. ej., si  $P_{i-2} = P_{i-1} = P_i$ ), entonces la ecuación (9.35) se convierte en

$$Q_i(t) = P_{i-3} B_{Bs-3} + P_i \cdot (B_{Bs-2} + B_{Bs-1} + B_{Bs0}). \quad (9.36)$$

Obviamente,  $Q_i$  es una línea recta. Además, el punto  $P_{i-2}$  es interpolado por la línea en  $t = 1$ , donde las tres ponderaciones aplicadas a  $P_i$  suman 1, pero  $P_{i-2}$  por lo general no es interpolado en  $t = 0$ . Otra manera de considerar este com-

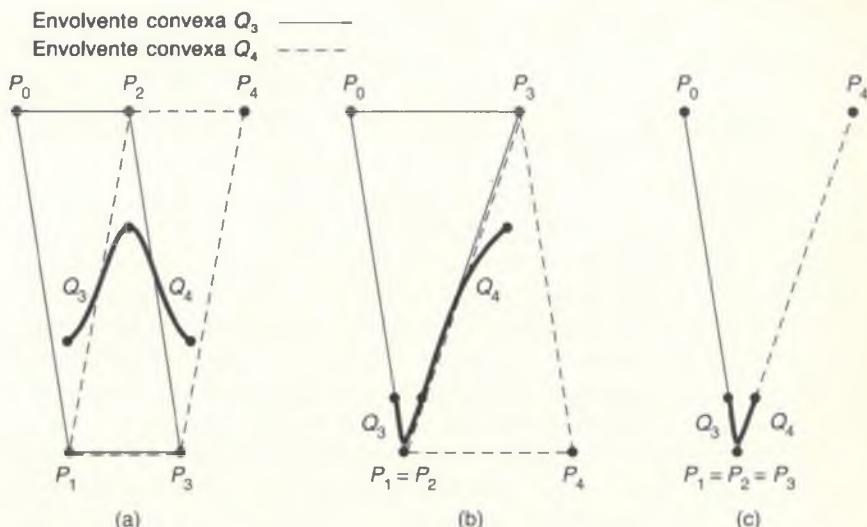
portamiento es que la envolvente convexa de  $Q_i$  ahora está definida por dos puntos diferentes, de manera que  $Q_i$  tiene que ser una línea. En la figura 9.21 se muestra el efecto de varios puntos de control en el interior de una *B-spline*.

En [BARS83;BAR87] se analiza otra técnica para interpolar puntos extremos, los vértices fantasma. En la sección siguiente veremos que, con las *B-splines* no uniformes, es posible interpolar los puntos extremos e internos en una manera más natural que con las *B-splines* uniformes.

### 9.2.5 *B-Splines* no uniformes, no racionales

La *B-splines* no uniformes, no racionales difieren de las *B-splines* uniformes no racionales analizadas en la sección 9.2.4 en que el intervalo de parámetros entre valores de nudo sucesivos no siempre es uniforme. La secuencia no uniforme de valores de nudo significa que las funciones de mezclado ya no son las mismas para cada intervalo, sino que varían de un segmento de curva a otro.

Estas curvas tienen varias ventajas con respecto a las *B-spline* uniformes. En primer lugar, la continuidad en los puntos de unión seleccionados se puede reducir de  $C^2$  a  $C^1$  a  $C^0$  a ninguna. Si la continuidad se reduce a  $C^0$ , entonces la



**Figura 9.21** Efecto de varios puntos de control sobre una curva de *B-spline* uniforme. En (a) no hay puntos de control múltiples. Las envolventes convexas de las dos curvas se superponen; el punto de unión entre  $Q_3$  y  $Q_4$  está en la región compartida por las dos envolventes convexas. En (b) hay un doble punto de control, de manera que las envolventes convexas comparten la arista  $P_2P_3$ ; por consiguiente, el punto de unión está restringido a esta arista. En (c) hay un punto de control triple y las envolventes convexas son líneas rectas que comparten el punto triple; por consiguiente, el punto de unión también está en el punto triple. Como las envolventes convexas son líneas rectas, los dos segmentos de curva también deben ser líneas rectas. En la unión hay continuidad  $C^2$  pero sólo continuidad  $G^0$ .

curva interpola un punto de control pero sin el efecto indeseable de las *B-splines* uniformes, donde los segmentos de curva a cada lado del punto de control interpolado son líneas rectas. Además, los puntos de inicio y fin se pueden interpolar con exactitud, sin introducir al mismo tiempo segmentos lineales. Como se analiza en [FOLE90], es posible añadir nudos y puntos de control a las *B-splines* no uniformes, de manera que se pueda cambiar fácilmente la forma de la curva resultante, una modificación que no puede realizarse con las *B-splines* uniformes.

La mayor generalidad de las *B-splines* no uniformes requiere una notación ligeramente diferente de la que se usa para las *B-splines* uniformes. Como antes, la *spline* es una curva continua por trozos formada por polinomios cúbicos que aproxima los puntos de control  $P_0$  a  $P_m$ . La secuencia de valores de nudo es una secuencia no decreciente de valores de nudo entre  $t_0$  y  $t_{m+4}$  (es decir, hay cuatro nudos más que puntos de control). Como el menor número posible de puntos de control es cuatro, la menor secuencia de nudos tiene ocho valores y la curva está definida en el intervalo de parámetros de  $t_3$  a  $t_4$ .

La única restricción que se aplica a la secuencia de nudos es que no sea decreciente, lo que permite que haya valores de nudos sucesivos iguales. Cuando esto ocurre, el valor del parámetro se conoce como **nudo múltiple** y el número de valores de parámetro idénticos se denomina **multiplicidad** del nudo (un nodo único tiene multiplicidad de 1). Por ejemplo, en la secuencia de nudos (0, 0, 0, 0, 1, 1, 2, 3, 4, 4, 5, 5, 5, 5), el valor de nudo 0 tiene multiplicidad de 4; el valor 1 tiene multiplicidad de 2; los valores 2 y 3 tienen multiplicidad de 1; el valor 4 tiene multiplicidad de 2; y el valor 5 tiene multiplicidad de 4.

El segmento de curva  $Q_i$  está definido por los puntos de control  $P_{i-3}$ ,  $P_{i-2}$ ,  $P_{i-1}$  y  $P_i$  y por las funciones de mezcla  $B_{i-3,4}(t)$ ,  $B_{i-2,4}(t)$ ,  $B_{i-1,4}(t)$ ,  $B_{i,4}(t)$  como la suma ponderada

$$Q_i(t) = P_{i-3} \cdot B_{i-3,4}(t) + P_{i-2} \cdot B_{i-2,4}(t) + P_{i-1} \cdot B_{i-1,4}(t) + P_i \cdot B_{i,4}(t)$$

$$3 \leq i \leq m, \quad t_i \leq t < t_{i+1}. \quad (9.37)$$

La curva no está definida fuera del intervalo  $t_3$  a  $t_{m+1}$ . Cuando  $t_i = t_{i+1}$  (un nudo múltiple), el segmento de curva  $Q_i$  es un solo punto. Este concepto de un segmento de curva que se reduce a un punto es lo que proporciona la flexibilidad adicional de las *B-splines* no uniformes.

No existe un conjunto único de funciones de mezcla, como sucedió con los otros tipos de *spline*. Las funciones dependen de los intervalos entre valores de nudos y se definen recursivamente en términos de funciones de mezcla de orden inferior.  $B_{i,j}(t)$  es la función de mezcla de  $j$ -ésimo orden para ponderar el punto de control  $P_i$ . Puesto que trabajamos con *B-splines* de cuarto orden (es decir, de tercer grado, o cúbicas), la definición recursiva termina en  $B_{i,4}(t)$  y se puede presentar fácilmente en forma “desenrollada”. La recurrencia para las *B-splines* cúbicas es

$$B_{i,1}(t) = \begin{cases} 1, & t_i \leq t < t_{i+1} \\ 0, & \text{o bien} \end{cases}$$

$$\begin{aligned}
 B_{i,2}(t) &= \frac{t - t_i}{t_{i+1} - t_i} B_{i,1}(t) + \frac{t_{i+2} - t}{t_{i+2} - t_{i+1}} B_{i+1,1}(t), \\
 B_{i,3}(t) &= \frac{t - t_i}{t_{i+2} - t_i} B_{i,2}(t) + \frac{t_{i+3} - t}{t_{i+3} - t_{i+1}} B_{i+1,2}(t), \\
 B_{i,4}(t) &= \frac{t - t_i}{t_{i+3} - t_i} B_{i,3}(t) + \frac{t_{i+4} - t}{t_{i+4} - t_{i+1}} B_{i+1,3}(t).
 \end{aligned} \tag{9.38}$$

Se puede demostrar que las funciones de mezcla son no negativas y que suman uno, de manera que los segmentos de curva de *B-spline* no uniforme yacen dentro de las envolventes convexas de sus cuatro puntos de control. En el caso de nudos con multiplicidad mayor que uno, los denominadores pueden ser cero ya que los valores de nudos sucesivos pueden ser iguales: la división entre cero se define como igual a cero.

El aumento en la multiplicidad de nudos tiene dos efectos. Primero, el *spline*, evaluada en cualquier valor de nudo conocido  $t_i$ , producirá automáticamente un punto dentro de la envoltura convexa de los puntos  $P_{i-3}$ ,  $P_{i-2}$  y  $P_{i-1}$ . Si  $t_i$  y  $t_{i+1}$  son iguales, deben estar en la envoltura convexa de  $P_{i-3}$ ,  $P_{i-2}$  y  $P_{i-1}$ , y en la envoltura convexa de  $P_{i-2}$ ,  $P_{i-1}$  y  $P_i$ . Por consiguiente, deben estar en realidad sobre el segmento de línea entre  $P_{i-2}$  y  $P_{i-1}$ . De la misma manera, si  $t_i = t_{i+1} = t_{i+2}$ , el nudo debe estar en  $P_{i-1}$ . Si  $t = t_{i+1} + t_{i+2} = t_{i+3}$ , entonces el nudo tiene que estar tanto en  $P_{i-1}$  como en  $P_i$  y la curva se parte. En segundo lugar, los nudos múltiples reducirán la continuidad paramétrica: de la continuidad  $C^2$  a  $C^1$  en el caso de un nudo adicional (multiplicidad 2); de  $C^1$  a  $C^0$  con dos nudos adicionales (multiplicidad 3); y de  $C^0$  a no continuidad en el caso de tres nudos adicionales (multiplicidad 4).

La creación interactiva de *splines* no uniformes generalmente implica apuntar a los puntos de control, indicando puntos múltiples con la selección sucesiva del mismo punto. Otra forma es apuntar directamente a la curva con un ratón de varios botones: presionando dos veces un botón se puede indicar un punto de control doble; oprimiendo dos veces otro botón se puede indicar un nudo doble.

## 9.2.6 Segmentos de curva polinomial cúbica racional, no uniforme

Los segmentos de curva cúbica racional son razones de polinomios:

$$x(t) = \frac{X(t)}{W(t)}, \quad y(t) = \frac{Y(t)}{W(t)}, \quad z(t) = \frac{Z(t)}{W(t)}, \tag{9.39}$$

donde  $X(t)$ ,  $Y(t)$ ,  $Z(t)$  y  $W(t)$  son curvas polinomiales cúbicas cuyos puntos de control están definidos en coordenadas homogéneas. También podemos considerar que la curva existe en el espacio homogéneo como  $Q(t) = [X(t) \ Y(t) \ Z(t) \ W(t)]^T$ . Como siempre, el cambio del espacio homogéneo al espacio tridimensional implica la división entre  $W(t)$ . Podemos transformar cualquier curva no racional en una racional añadiendo  $W(t) = 1$  como cuarto elemento. En términos

generales, los polinomios en una curva racional pueden ser de Bézier, de Hermite o de otro tipo. Cuando son *B-splines*, obtenemos *B-splines* racionales no uniformes, en ocasiones denominadas **NURBS** (*nonuniform rational B-splines*) [FORR80].

Las curvas racionales son útiles por varias razones. La primera y más importante es que son invariantes al aplicar transformaciones de rotación, escalamiento, traslación y perspectiva de los puntos de control (las curvas no racionales sólo son invariantes con la rotación, el escalamiento y la traslación). Por lo tanto, la transformación de perspectiva únicamente tiene que aplicarse a los puntos de control, los cuales pueden emplearse después para generar la transformación de perspectiva de la curva original. La alternativa a convertir una curva no racional en una racional antes de efectuar la transformación de perspectiva es generar primero los puntos en la curva misma para luego aplicar la transformación de perspectiva a *cada uno* de los puntos, un proceso mucho menos eficiente. Una observación análoga es que la transformación de perspectiva de una esfera no es lo mismo que una esfera cuyo centro y radio son el centro y el radio transformados de la esfera original.

Otra ventaja de las *splines* racionales es que, a diferencia de las no racionales, pueden definir de manera precisa cualquiera de las secciones cónicas. Con las no racionales sólo es posible aproximar una cónica, usando varios puntos de control cercanos a ella. Esta segunda propiedad es útil en las aplicaciones, especialmente en CAD, donde se requieren curvas y superficies generales además de cónicas. Con las NURBS se pueden definir ambos tipos de entidades.

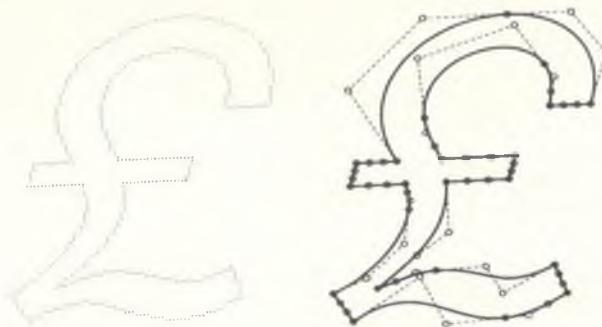
Para conocer más acerca de las cónicas y las NURBS, véase [FAUX79; BÖHM84; TILL83].

### 9.2.7 Ajuste de curvas a puntos digitalizados

Un ingeniero o un artista muchas veces tiene una representación no electrónica de una forma compleja, la cual se puede digitalizar como una serie de puntos discretos. Por ejemplo, quizás lo único disponible sea una copia de la forma impresa en papel. Para la manipulación adicional de la forma nos gustaría ajustar una curva suave o una serie de curvas a la representación digitalizada (generalmente) imprecisa. Se han publicado varias técnicas de ajuste de curvas, todas las cuales tienen ventajas y desventajas. Schneider [SCHN90] ha desarrollado un método para aproximar curvas digitalizadas con segmentos por trozos de Bézier. Las ventajas con respecto a los métodos anteriores son la continuidad geométrica, la estabilidad y la facilidad de implantación. En [SCHN90] puede hallar una implantación completa en C del algoritmo. En la figura 9.22 se presenta un ejemplo de la aplicación del método a una forma digitalizada.

### 9.2.8 Comparación de las curvas cúbicas

Los diversos tipos de curvas cúbicas paramétricas se pueden comparar con base en distintos criterios, como son la facilidad de manipulación interactiva, el grado de continuidad en los puntos de unión, la generalidad y la velocidad de la



**Figura 9.22** Carácter digitalizado, donde se presentan la muestra original, las curvas ajustadas y los puntos de control de Bézier. (Cortesía de Academic Press, Inc.)

cálculos que usan estas representaciones. Por supuesto, no es necesario elegir una sola representación, ya que podemos efectuar conversiones entre las representaciones, como se describe en [FOLE90]. Por ejemplo, las *B-splines* racionales no uniformes se pueden usar como representación interna, mientras el usuario manipula en forma interactiva puntos de control de Bézier o puntos de control y vectores tangente de Hermite. Algunos editores gráficos interactivos ofrecen al usuario curvas hermitianas a la vez que las representan internamente en la forma de Bézier apoyada por PostScript [ADOB85]. Por lo general, el usuario de un sistema CAD interactivo puede elegir entre varias opciones, como serían formas de Hermite, de Bézier, *B-splines* uniformes y *B-splines* no uniformes. Lo más probable es que la representación de *B-spline* racional no uniforme se use en forma interna, ya que es la más general.

En la tabla 9.1 se compara la mayoría de las formas de curva mencionadas en esta sección. En la tabla no se incluye de manera explícita la facilidad de manipulación interactiva, ya que este atributo depende de la aplicación. El rubro *número de parámetros que controlan un segmento de curva* consiste en las cuatro restricciones geométricas más otros parámetros, como el espaciado de nudos en el caso de las *splines* no uniformes. El rubro *continuidades fácilmente logradas* se refiere a las restricciones, por ejemplo forzar los puntos de control para que sean colineales a fin de permitir la continuidad  $G^1$ . Como la continuidad  $C^n$  es más restrictiva que la  $G^n$ , cualquier forma que pueda obtener  $C^n$  puede, por definición, obtener al menos continuidad  $G^n$ .

Cuando sólo se requiere continuidad geométrica, como ocurre con frecuencia en CAD, la selección se reduce a los diversos tipos de *splines*, todos los cuales obtienen continuidad  $G^1$  y  $G^2$ . De los tres tipos de *splines* en la tabla, los más limitantes son las *B-splines* uniformes. La posibilidad de nudos múltiples que ofrecen las *B-splines* permite que el usuario tenga más control sobre la forma. Por supuesto, es importante una buena interfaz que permita al usuario aprovechar esta capacidad.

Tabla 9.1

## Comparación de cuatro formas diferentes de curvas cúbicas paramétricas

	Hermite	Bézier	<i>B-Spline</i> uniforme	<i>B-Spline</i> no uniforme
Envoltura convexa definida por puntos de control	N/A	Sí	Sí	Sí
Interpola algunos puntos de control	Sí	Sí	No	No
Interpola todos los puntos de control	Sí	No	No	No
Facilidad de subdivisión	Buena	La mejor	Mediana	Alta
Continuidades inherentes en la representación	$C^0$ $G^0$	$C^0$ $G^0$	$C^2$ $G^2$	$C^2$ $G^2$
Continuidades fácilmente logradas	$C^1$ $G^1$	$C^1$ $G^1$	$C^2$ $G^2*$	$C^2$ $G^2*$
Número de parámetros que controlan un segmento de curva	4	4	4	5

\*Excepto para el caso especial analizado en la sección 9.2.

Es común ofrecer al usuario la capacidad para arrastrar puntos de control o vectores tangente en forma interactiva, presentando continuamente la *spline* actualizada. En la figura 9.19 se muestra una secuencia de este tipo para *B-splines*. Una de las desventajas de las *B-splines* en algunas aplicaciones es que los puntos de control no están en la *spline*. Sin embargo, es posible omitir dibujar los puntos de control y permitir en cambio que el usuario interactúe con los nudos (los cuales deberán estar marcados para que puedan seleccionarse).

### 9.3 Superficies bicubáticas paramétricas

Las superficies bicubáticas paramétricas constituyen una generalización de las curvas cúbicas paramétricas. Recuerde la forma general de la curva cúbica paramétrica,  $Q(t) = G \cdot M \cdot T$ , donde  $G$ , la matriz de geometría, es una constante

te. En primer lugar, por cuestiones de facilidad de notación, reemplazamos  $t$  con  $s$  para obtener  $Q(s) = G \cdot M \cdot S$ . Si permitimos que los puntos en  $G$  varíen en tres dimensiones sobre una trayectoria parametrizada en  $t$ , obtenemos

$$Q(s, t) = [G_1(t) \ G_2(t) \ G_3(t) \ G_4(t)] \cdot M \cdot S. \quad (9.40)$$

Ahora, para una  $t_1$  fija,  $Q(s, t_1)$  es una curva porque  $G(t_1)$  es constante. Si  $t$  asume un valor distinto, digamos  $t_2$ , donde  $t_2 - t_1$  es muy pequeño, entonces  $Q(s, t)$  es una curva ligeramente diferente. Repitiendo este proceso para varios valores de  $t_2$  entre 0 y 1 definimos una familia completa de curvas, cada uno de cuyos miembros está arbitrariamente cerca de otra curva. El conjunto de estas curvas define una superficie. Si las  $G_i(t)$  son por si mismas cúbicas, se dice que es una **superficie bicubica paramétrica**.

Continuando con el caso en que las  $G_i(t)$  son cúbicas, cada una de ellas se puede representar como  $G_i(t) = G_i \cdot M \cdot T$ , donde  $G_i = [g_{i1} \ g_{i2} \ g_{i3} \ g_{i4}]$  (las letras  $G$  y  $g$  se usan para distinguirlas de la  $G$  utilizada para la curva). De esta manera,  $g_{i1}$  es el primer elemento de la matriz de geometría para la curva  $G_i(t)$ , etcétera.

Transpongamos ahora la ecuación  $G_i(t) = G_i \cdot M \cdot T$  usando la identidad  $(A \cdot B \cdot C)^T = C^T \cdot B^T \cdot A^T$ . El resultado es  $G_i(t) = T^T \cdot M^T \cdot G_i T = T^T \cdot M^T \cdot [g_{i1} \ g_{i2} \ g_{i3} \ g_{i4}]^T$ . Si sustituimos este resultado en la ecuación (9.40) para cada uno de los cuatro puntos, obtenemos

$$Q(s, t) = T^T \cdot M^T \cdot \begin{bmatrix} g_{11} & g_{21} & g_{31} & g_{41} \\ g_{12} & g_{22} & g_{32} & g_{42} \\ g_{13} & g_{23} & g_{33} & g_{43} \\ g_{14} & g_{24} & g_{34} & g_{44} \end{bmatrix} \cdot M \cdot S. \quad (9.41)$$

o

$$Q(s, t) = T^T \cdot M^T \cdot G \cdot M \cdot S, \quad 0 \leq s, t \leq 1. \quad (9.42)$$

Si escribimos por separado para  $x$ ,  $y$  y  $z$ , la forma es

$$x(s, t) = T^T \cdot M^T \cdot G_x \cdot M \cdot S,$$

$$y(s, t) = T^T \cdot M^T \cdot G_y \cdot M \cdot S,$$

$$z(s, t) = T^T \cdot M^T \cdot G_z \cdot M \cdot S, \quad (9.43)$$

A partir de esta forma general, pasamos ahora a examinar maneras específicas de establecer superficies usando diferentes matrices de geometría.

### 9.3.1 Superficies de Hermite

Las superficies de Hermite o hermitianas están totalmente definidas por una matriz de geometría  $G_H$  de  $4 \times 4$ . La obtención de  $G_H$  sigue las mismas pautas

que se emplearon para hallar la ecuación (9.42). Aquí desarrollaremos un poco más esta obtención, aplicándola sólo a  $x(s, t)$ . Primero reemplazamos  $t$  por  $s$  en la ecuación (9.13) para obtener  $x(s) = G_{H_s} \cdot M_H \cdot S$ . Al reescribir la ecuación de manera que la matriz de geometría hermitiana  $G_{H_s}$  no sea constante sino una función de  $t$ , obtenemos

$$x(s, t) = G_{H_s}(t) \cdot M_H \cdot S = [P_{1_s}(t) P_{4_s}(t) R_{1_s}(t) R_{4_s}(t)] \cdot M_H \cdot S. \quad (9.44)$$

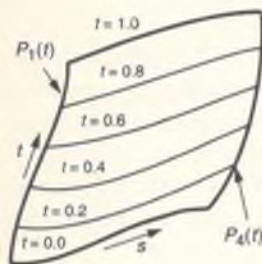


Figura 9.23

Línea de valores de parámetro constante en una superficie bicubica:  $P_1(t)$  está en  $s = 0$  y  $P_4(t)$  está en  $s = 1$ .

Las funciones  $P_{1_s}(t)$  y  $P_{4_s}(t)$  definen los componentes  $x$  de los puntos inicial y final de la curva en el parámetro  $s$ . En forma similar,  $R_{1_s}$  y  $R_{4_s}(t)$  son los vectores tangente en estos puntos. Para cualquier valor de  $t$  hay dos puntos extremos y vectores tangente específicos. En la figura 9.23 se muestran  $P_1(t)$ ,  $P_4(t)$  y la cónica en  $s$  que se define cuando  $t = 0.0, 0.2, 0.4, 0.6, 0.8$  y  $1.0$ . El parche de superficie es, en esencia, una interpolación cónica entre  $P_1(t) = Q(0, t)$  y  $P_4(t) = Q(1, t)$  o, alternativamente, entre  $Q(s, 0)$  y  $Q(s, 1)$ .

En el caso especial de que los cuatro interpolantes  $Q(0, t)$ ,  $Q(1, t)$ ,  $Q(s, 0)$  y  $Q(s, 1)$  sean líneas rectas, el resultado es una **superficie reglada**. Si los interpolantes también son coplanares, entonces la superficie es un polígono plano de cuatro lados.

Continuando con la obtención, representemos  $P_{1_s}(t)$ ,  $P_{4_s}(t)$ ,  $R_{1_s}(t)$  y  $R_{4_s}(t)$  en forma hermitiana como

$$P_{1_s}(t) = [g_{11} g_{12} g_{13} g_{14}]_x \cdot M_H \cdot T, \quad P_{4_s}(t) = [g_{21} g_{22} g_{23} g_{24}]_x \cdot M_H \cdot T,$$

$$R_{1_s}(t) = [g_{31} g_{32} g_{33} g_{34}]_x \cdot M_H \cdot T, \quad R_{4_s}(t) = [g_{41} g_{42} g_{43} g_{44}]_x \cdot M_H \cdot T, \quad (9.45)$$

Estas cuatro cónicas se pueden reescribir juntas en una sola ecuación:

$$[P_{1_s}(t) P_{4_s}(t) R_{1_s}(t) R_{4_s}(t)]^T = G_{H_s} \cdot M_H \cdot T, \quad (9.46)$$

donde

$$G_{H_s} = \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix}_x \quad (9.47)$$

Al transponer ambos lados de la ecuación (9.46) se obtiene

$$[P_{1_s}(t) P_{4_s}(t) R_{1_s}(t) R_{4_s}(t)] = T^T \cdot M_H^T \cdot G_{H_s}^T \cdot S = T^T \cdot M_H^T \cdot G_{H_s} \cdot S, \quad (9.48)$$

Si sustituimos la ecuación (9.48) en la ecuación (9.44) obtenemos

$$x(s, t) = T^T \cdot M_H^T \cdot G_{H_x} \cdot M_H \cdot S; \quad (9.49)$$

y en forma similar

$$y(s, t) = T^T \cdot M_H^T \cdot G_{H_y} \cdot M_H \cdot S, \quad z(s, t) = T^T \cdot M_H^T \cdot G_{H_z} \cdot M_H \cdot S. \quad (9.50)$$

Las tres matrices de  $4 \times 4$ ,  $G_{H_x}$ ,  $G_{H_y}$  y  $G_{H_z}$ , tienen una función similar en las superficies hermitianas que la matriz  $G_H$  en el caso de las curvas.

La bicúbica hermitiana permite continuidad  $C^1$  y  $G^1$  de un parche a otro, en una forma muy parecida a como la cúbica hermitiana permite continuidad  $C^1$  y  $G^1$  de un segmento de curva al otro. En el capítulo 11 de [FOLE90] encontrará los detalles al respecto.

### 9.3.2 Superficies de Bézier

La formulación bicúbica de Bézier se puede derivar exactamente de la misma manera que en el caso de la cúbica hermitiana. Los resultados son

$$\begin{aligned} x(s, t) &= T^T \cdot M_B^T \cdot G_{B_x} \cdot M_B \cdot S, \\ y(s, t) &= T^T \cdot M_B^T \cdot G_{B_y} \cdot M_B \cdot S, \\ z(s, t) &= T^T \cdot M_B^T \cdot G_{B_z} \cdot M_B \cdot S. \end{aligned} \quad (9.51)$$

La matriz de geometría de Bézier,  $G$ , consiste en 16 puntos de control, como se ilustra en la figura 9.24. Las superficies de Bézier son atractivas para el diseño interactivo por la misma razón que las curvas de Bézier: algunos de los puntos de control interpolan la superficie, permitiendo un control preciso,

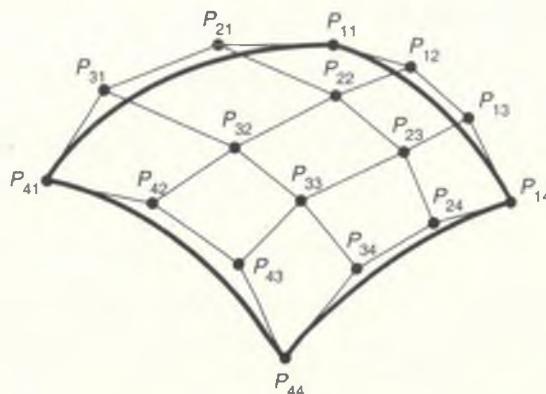
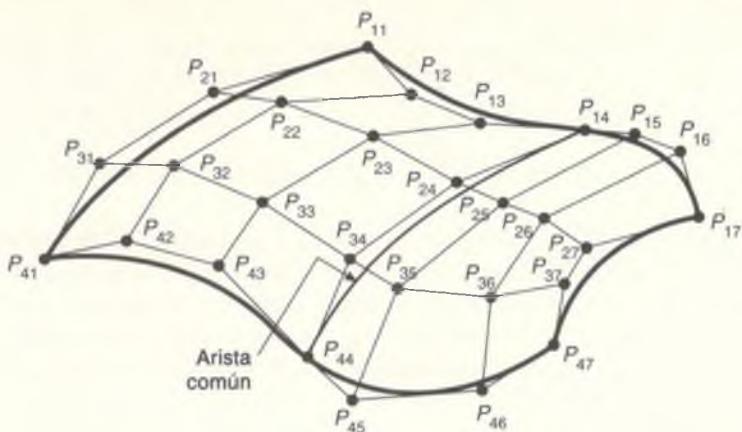


Figura 9.24 16 puntos de control para un parche bicúbico de Bézier.



**Figura 9.25** Dos parches de Bézier unidos en la arista  $P_{14}$ ,  $P_{24}$ ,  $P_{34}$  y  $P_{44}$ .

mientras que los vectores tangente se pueden controlar explícitamente. La propiedad de la envolvente convexa también es atractiva cuando se usan las superficies de Bézier como representación interna.

Obtendremos continuidad  $C^0$  y  $G^0$  en las aristas de los parches igualando los cuatro puntos de control comunes. La continuidad  $G^1$  se presenta cuando los dos conjuntos de cuatro puntos de control a cada lado de la arista son colineales con los puntos en la arista. En la figura 9.25, los siguientes conjuntos de puntos de control son colineales y definen cuatro segmentos de línea cuyas longitudes tienen la misma razón  $k$ :  $(P_{13}, P_{14}, P_{15})$ ,  $(P_{23}, P_{24}, P_{25})$ ,  $(P_{33}, P_{34}, P_{35})$  y  $(P_{43}, P_{44}, P_{45})$ . La tetera presentada en la figura 9.1 se modeló con 32 parches de Bézier, unidos para asegurar la continuidad  $G^1$ .

### 9.3.3 Superficies *B-spline*

Los parches *B-spline* se representan como

$$\begin{aligned} x(s, t) &= T^T \cdot M_{Bs_x}^T \cdot G_{Bs_x} \cdot M_{Bs} \cdot S, \\ y(s, t) &= T^T \cdot M_{Bs_y}^T \cdot G_{Bs_y} \cdot M_{Bs} \cdot S, \\ z(s, t) &= T^T \cdot M_{Bs_z}^T \cdot G_{Bs_z} \cdot M_{Bs} \cdot S. \end{aligned} \quad (9.5)$$

La continuidad  $C^2$  entre fronteras es automática en las *B-splines*; no se requiere ninguna disposición especial de los puntos de control, excepto para evitar puntos de control duplicados, los cuales crean discontinuidades.

Las superficies bicubicas *B-splines* no uniformes y racionales, así como otras superficies racionales, son análogas a sus contrapartes cúbicas. Todas las técnicas de presentación se transfieren directamente al caso bicúbico.

### 9.3.4 Normales a superficies

Es fácil determinar la normal a una superficie bicubica, la cual es necesaria para el sombreado (Cap. 14), para efectuar la detección de interferencias en robótica, para calcular desplazamientos en el maquinado controlado numéricamente y para realizar otros cálculos. A partir de la ecuación (9.42), el vector tangente  $s$  de la superficie  $Q(s, t)$  es

$$\begin{aligned}\frac{\partial}{\partial s} Q(s, t) &= \frac{\partial}{\partial s}(T^T \cdot M^T \cdot G \cdot M \cdot S) = T^T \cdot M^T \cdot G \cdot M \cdot \frac{\partial}{\partial s}(S) \\ &= T^T \cdot M^T \cdot G \cdot M \cdot [3s^2 \ 2s \ 1 \ 0]^T,\end{aligned}\quad (9.53)$$

y el vector tangente  $t$  es

$$\begin{aligned}\frac{\partial}{\partial t} Q(s, t) &= \frac{\partial}{\partial t}(T^T \cdot M^T \cdot G \cdot M \cdot S) = \frac{\partial}{\partial t}(T^T \cdot M^T \cdot G \cdot M \cdot S) \\ &= [3t^2 \ 2t \ 1 \ 0]^T \cdot M^T \cdot G \cdot M \cdot S.\end{aligned}\quad (9.54)$$

Los dos vectores tangente son paralelos a la superficie en el punto  $(s, t)$ , y su producto cruz es entonces perpendicular a la superficie. Observe que, si ambos vectores tangente son cero, el producto cruz es cero y no existe una normal a la superficie. Recuerde que un vector tangente puede ser cero en puntos de unión que tienen continuidad  $C^1$  pero no  $G^1$ .

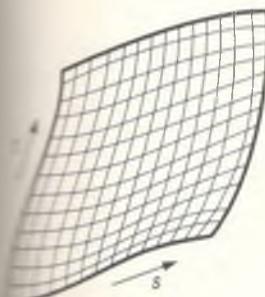
Por supuesto, cada uno de los vectores tangente es una 3-tupla, ya que la ecuación (9.42) representa los componentes  $x$ ,  $y$  y  $z$  de la bicubica. Con la notación  $x_s$  para el componente  $x$  del vector tangente  $s$ ,  $y_s$  para el componente  $y$  y  $z_s$  para el componente  $z$ , la normal es

$$\frac{\partial}{\partial s} Q(s, t) \times \frac{\partial}{\partial t} Q(s, t) = [y_s z_t - y_t z_s, \ z_s x_t - z_t x_s, \ x_s y_t - x_t y_s]. \quad (9.55)$$

La normal a la superficie es un polinomio biquintico (dos variables, quinto grado) y por ende es bastante difícil de calcular. En [SCHW82] se presenta una aproximación bicubica que resulta satisfactoria mientras el parche sea relativamente suave.

### 9.3.5 Dibujo de superficies bicubicas

Las superficies, así como las curvas, se pueden presentar con una evaluación iterativa de los polinomios bicubicos. La evaluación iterativa se adapta mejor al dibujo de parches bicubicos cuyo estilo es el que se muestra en la figura 9.26. Cada una de las curvas de  $s$  constante y  $t$  constante de la superficie es en sí una cúbica, de manera que el dibujo de las curvas es bastante sencilla, como se ilustra en el programa 9.2.



**Figura 9.26**  
Parche de superficie presentado como curvas de constante  $s$  y  $t$  constante.

La técnica burda para la evaluación iterativa de superficies es más costosa que la de las curvas, ya que las ecuaciones de superficie tienen que evaluarse unas  $2/\delta^2$  veces. Para  $\delta = 0.1$ , este valor es 200; para  $\delta = 0.01$ , es 20 000. Estas cantidades hacen que el método alternativo de diferencia adelantada sea más atractivo que en el caso de las curvas. En [FOLE90; FORR79] se presentan este método y otras formas útiles de presentar superficies bicúbicas.

### Programa 9.2

*Función para dibujar un parche bicúbico como una malla. Las funciones  $X(s, t)$ ,  $Y(s, t)$  y  $Z(s, t)$  evalúan la superficie usando los coeficientes de la matriz de coeficientes.*

```

typedef float coefs[4][4][3];

void dibujar_superficie (coefs *coeficientes, int ns, int nt, int n)
    /* los coeficientes variables son los coeficientes para Q(s, t) */
    /* ns y nt son el número de curvas de s y t constante que se dibujarán */
{
    float del, dels, delt, s, t;
    int i, j;
    /* Asignar valores iniciales */
    del = 1.0 / n;           /* Incremento que se usará para dibujar las curvas */
    dels = 1.0 / (ns - 1);   /* Incremento en s al pasar a la siguiente curva de t constante */
    delt = 1.0 / (nt - 1);   /* Incremento en t al pasar a la siguiente curva de s constante */
    /* Dibujar ns curvas de s constante, para s = 0.0, dels, 2dels,..., 1.0 */
    for (i = 0; i < ns; i++) {
        s = i * dels;
        /* Dibujar una curva de s constante, variando t de 0.0 a 1.0 */
        /* X, Y y Z son funciones para evaluar las bicúbicas de una s y t determinadas */
        mover_abs3 (X(s, 0.0), Y(s, 0.0), Z(s, 0.0));
        for (j = 0; j < n; j++) {
            t = j * del;
            /* se usan n pasos al variar t de 0.0 a 1.0 para cada curva */
            linea_abs3 (X(s, t), Y(s, t), Z(s, t));
        }
    }

    /* Dibujar nt curvas de t constante, para t = 0.0, delt, 2delt, ..., 1.0 */
    for (i = 0; i < nt; i++) {
        t = i * delt;
        /* Dibujar una curva de t constante, variando s de 0.0 a 1.0 */
        mover_abs3 (X(0.0, t), Y(0.0, t), Z(0.0, t));
        for (j = 0; j < n; j++) {
            s = j * del;
            /* se usan n pasos al variar s de 0 a 1 para cada curva */
            linea_abs3 (X(s, t), Y(s, t), Z(s, t));
        }
    }
}

```

Las funciones  $X(s, t)$ ,  $Y(s, t)$  y  $Z(s, t)$  se pueden desarrollar fácilmente para un tipo de superficie en particular. Como ejemplo, consideraremos una superfi-

cie de Bézier. A partir de la ecuación (9.51), la ecuación  $x(s, t)$  se puede reescribir como

$$x(s, t) = [(1-t)^3 \ 3t(1-t)^2 \ 3t^2(1-t) \ t^3] \cdot G_{B_x} \cdot \begin{bmatrix} .(1-s)^3 \\ 3s(1-s)^2 \\ 3s^2(1-s) \\ s^3 \end{bmatrix}, \quad (9.56)$$

con sólo desarrollar las matrices  $T^T \cdot M_B T$  y  $M_B \cdot S$ . Recuerde que  $G_{B_x}$  es la matriz del componente  $x$  de los puntos de control, los cuales se presentan en la figura 9.24, de manera que se puede escribir como

$$G_{B_x} = \begin{bmatrix} P_{11} & P_{21} & P_{31} & P_{41} \\ P_{12} & P_{22} & P_{32} & P_{42} \\ P_{13} & P_{23} & P_{33} & P_{43} \\ P_{14} & P_{24} & P_{34} & P_{44} \end{bmatrix}_x.$$

Por último, la forma totalmente desarrollada de la ecuación (9.56) se puede escribir como

$$\begin{aligned} x(s, t) = & (1-s)^3(P_{11_x}(1-t)^3 + 3P_{12_x}(1-t)^2t + 3P_{13_x}(1-t)t^2 + P_{14_x}t^3) \\ & + 3(1-s)^2s(P_{21_x}(1-t)^3 + 3P_{22_x}(1-t)^2t + 3P_{23_x}(1-t)t^2 + P_{24_x}t^3) \\ & + 3(1-s)s^2(P_{31_x}(1-t)^3 + 3P_{32_x}(1-t)^2t + 3P_{33_x}(1-t)t^2 + P_{34_x}t^3) \\ & + s^3(P_{41_x}(1-t)^3 + 3P_{42_x}(1-t)^2t + 3P_{43_x}(1-t)t^2 + P_{44_x}t^3). \end{aligned}$$

Las ecuaciones para  $y(s, t)$  y  $z(s, t)$  se obtienen en forma similar. Las funciones  $X(s, t)$ ,  $Y(s, t)$  y  $Z(s, t)$ , necesarias para el programa 9.2, se pueden codificar directamente a partir de las expresiones  $x(s, t)$ ,  $y(s, t)$  y  $z(s, t)$ . Las funciones para dibujar otros tipos de superficies se pueden desarrollar con la misma facilidad que en el caso de la superficie de Bézier.

## 9.4 Superficies cuádricas

La ecuación de superficie implícita de la forma

$$f(x, y, z) = ax^2 + by^2 + cz^2 + 2dxy + 2eyz + 2fxz + 2gx + 2hy + 2jz + k = 0 \quad (9.57)$$

define la familia de superficies cuádricas. Por ejemplo, si  $a = b = c = -k = 1$  y los demás coeficientes son cero, una esfera unidad está definida en el origen. Si de  $a$  a  $f$  son cero, se define un plano. Las superficies cuádricas son muy útiles en aplicaciones especializadas, como el modelado molecular [PORT79;

MAX79], y también se han integrado a los sistemas de modelado de sólidos. Recuerde así mismo que las curvas cúbicas racionales pueden representar secciones cónicas; en forma similar, las superficies bicúbicas racionales pueden representar cuádricas. Por consiguiente, la ecuación cuadrática implícita es una alternativa a las superficies racionales si sólo se representan superficies cuádricas. Otras razones para usar cuádricas incluyen la facilidad para:

- Calcular la normal a la superficie.
- Determinar si un punto está en la superficie [basta sustituir el punto en la ecuación (9.57), evaluar y probar un resultado con una  $\epsilon$  de cero].
- Calcular  $z$  a partir de  $x$  y  $y$  (importante en los algoritmos de superficies ocultas; véase el Cap. 13)
- Calcular intersecciones entre superficies.

Una representación alternativa de la ecuación (9.57) es

$$P^T \cdot Q \cdot P = 0, \quad (9.58)$$

$$\text{con } Q = \begin{bmatrix} a & d & f & g \\ d & b & e & h \\ f & e & c & j \\ g & h & j & k \end{bmatrix} \quad \text{y} \quad P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \quad (9.59)$$

La superficie representada por  $Q$  puede trasladarse y escalarse con facilidad. Dada una matriz de transformación  $M$  de  $4 \times 4$ , de la forma desarrollada en el capítulo 5, la superficie cuádrica transformada  $Q'$  se expresa como

$$Q' = (M^{-1})^T \cdot Q \cdot M^{-1}. \quad (9.60)$$

La normal a la superficie implícita definida por  $f(x, y, z) = 0$  es el vector  $[df/dx, df/dy, df/dz]$ . Esta normal a la superficie es mucho más fácil de evaluar que la normal a una superficie bicubica que vimos en la sección 9.3.4.

## 9.5 Técnicas de modelado especializado

En lo que va de este capítulo nos hemos centrado en modelos geométricos, como también lo haremos en el capítulo 10. Estos modelos serían suficientes en el mundo formado exclusivamente por objetos geométricos simples. Sin embargo, muchos fenómenos naturales no se pueden representar de manera eficaz con modelos geométricos, por lo menos no a gran escala. Por ejemplo, la niebla

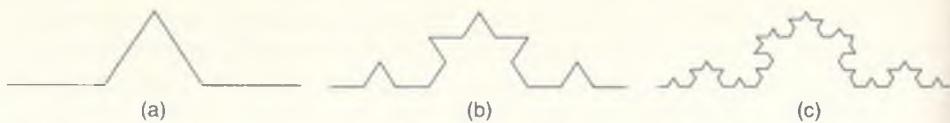
está formada por diminutas gotas de agua, pero no es posible usar un modelo en el cual se coloque cada gota. Así mismo, este modelo de gotas de agua no representa correctamente nuestra percepción de la niebla, ya que la vemos como una borrosidad en el aire frente a nosotros, no como millones de gotas. Nuestra percepción visual de la niebla se basa en la forma en que ésta altera la luz que llega a nuestros ojos, no en la forma ni la colocación de las gotas individuales. Por lo tanto, necesitamos un modelo diferente para representar con eficacia el efecto de percepción de la niebla. De la misma manera, la forma de una hoja de árbol se puede modelar con polígonos y su tallo con un tubo de *spline*, pero sería demasiado laborioso y tardado colocar explícitamente cada tronco, rama, tallo, y hoja de un árbol.

En el capítulo 20 de [FOLE90] encontrará un análisis exhaustivo de las técnicas de modelado avanzado; aquí veremos dos métodos especializados, los cuales son sorprendentemente sencillos de implantar y producen imágenes con un realismo asombroso.

### 9.5.1 Modelos fractales

En fechas recientes, los fractales han llamado mucho la atención [VOSS87; MAND82; PEIT86]. Las imágenes que se obtienen con ellos son espectaculares, y se han desarrollado varios métodos para generar fractales. El término **fractal** ha sido generalizado por la comunidad de la graficación por computador para incluir objetos fuera de la definición original de Mandelbrot; ahora se refiere a cualquier cosa con gran cantidad de autosimilitud exacta o estadística. Aquí usaremos esta acepción, aunque la definición matemática precisa requiere la autosimilitud estadística en todas las resoluciones. Por lo tanto, los únicos objetos fractales verdaderos son aquellos generados por procesos infinitamente recursivos. Por otra parte, los que se generan con procesos finitos pueden exhibir cambios cuyo detalle sea imperceptible después de alcanzar cierta etapa, por lo cual son aproximaciones de lo ideal. La mejor forma de ilustrar lo que significa la **autosimilitud** es con un ejemplo: el copo de nieve de von Koch. A partir de una línea recta con una protuberancia, como se ilustra en la figura 9.27, reemplazamos cada segmento de línea con una figura exactamente igual a la línea original. Este proceso se repite: cada segmento de la parte (b) de la figura se reemplaza con una forma idéntica a toda la figura. (No importa si el reemplazo se lleva a cabo con la forma presentada en la parte (a) o con la que se muestra en la parte (b); si se utiliza la de la parte (a), el resultado después de  $2^n$  pasos es igual que el que se obtiene después de  $n$  pasos si se sustituye cada segmento de la figura actual con toda la figura en cada paso.) Si repetimos este proceso un número infinito de veces, se dice que el resultado es **autosimilar**: todo el objeto es similar (es decir, se puede trasladar, rotar y escalar) a una subporción de él mismo.

A este concepto de la autosimilitud se suma el de la **dimensión fractal**. Para definir la dimensión fractal, examinaremos algunas propiedades de los objetos cuyas dimensiones conocemos. Un segmento de línea es unidimensional; si divi-



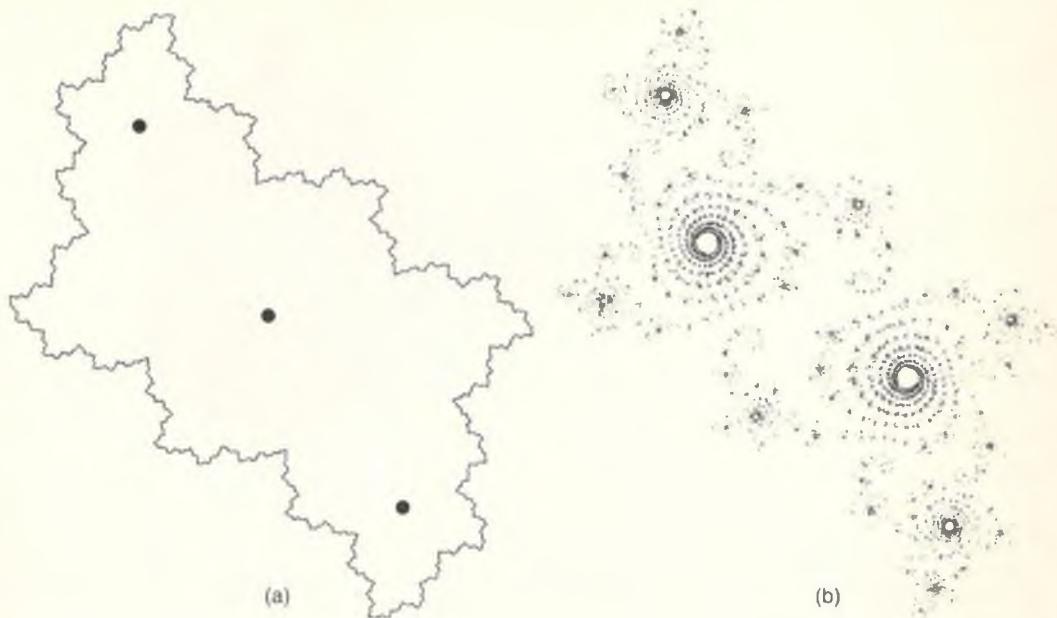
**Figura 9.27** Construcción del copo de nieve de von Koch: cada segmento en (a) es reemplazado por una copia exacta de la figura completa, reducida por un factor de 3. Se aplica el mismo proceso a los segmentos en (b) para generar los de (c).

dimos una línea en  $N$  partes iguales, las partes se parecerán a la línea original reducidas por un factor de  $N = N^{1/1}$ . Un cuadrado es bidimensional: si lo dividimos en  $N$  partes, cada una de ellas se parecerá a la original reducida por un factor de  $\sqrt{N} = N^{1/2}$ . (P. ej., un cuadrado se divide en nueve subcuadrados, cada uno de los cuales se parece al original escalado por un factor de  $1/3$ .) ¿Qué ocurre con el copo de nieve de von Koch? Cuando se divide en cuatro partes [las partes relacionadas con los cuatro segmentos originales de la figura 9.27(a)], cada trozo resultante se parece al original reducido por un factor de 3. Nos gustaría decir que tiene una dimensión  $d$ , donde  $4^{1/d} = 3$ . El valor de  $d$  debe ser  $\log(4)/\log(3) = 1.26\dots$  Ésta es la definición de una dimensión fractal.

Vale la pena mencionar los dos objetos fractales más famosos: el conjunto de Julia-Fatou y el conjunto de Mandelbrot. Estos objetos se generan a partir del estudio de la regla  $x \rightarrow x^2 + c$  (y de muchas reglas más, pero ésta es la más conocida). Aquí  $x$  es un **número complejo**,<sup>1</sup>  $x = a + bi$ . Si un número complejo tiene módulo menor que 1, al elevarlo repetidamente al cuadrado se aproxima a cero; si el módulo es mayor que 1, este proceso generará números cada vez mayores. Los números con módulo 1 conservan el mismo módulo después de elevar al cuadrado repetidamente. Por lo tanto, algunos números complejos “caen a cero” cuando se les eleva repetidamente al cuadrado, otros “caen a infinito” y otros más no corresponden a ninguna de las dos categorías; este último grupo forma la frontera entre los números atraídos hacia el cero y los atraídos hacia el infinito.

Suponga que aplicamos repetidamente la correspondencia  $x \rightarrow x^2 + c$  a cada número complejo  $x$  para un valor distinto de cero de  $c$ , como  $c = -0.12375 + 0.056805i$ ; algunos números complejos serán atraídos al infinito, otros a números finitos y otros más no tenderán a ningún lado. Al dibujar el conjunto de puntos que no tienden a ninguno de los extremos se obtiene el conjunto de Julia-Fatou presentado en la figura 9.28(a).

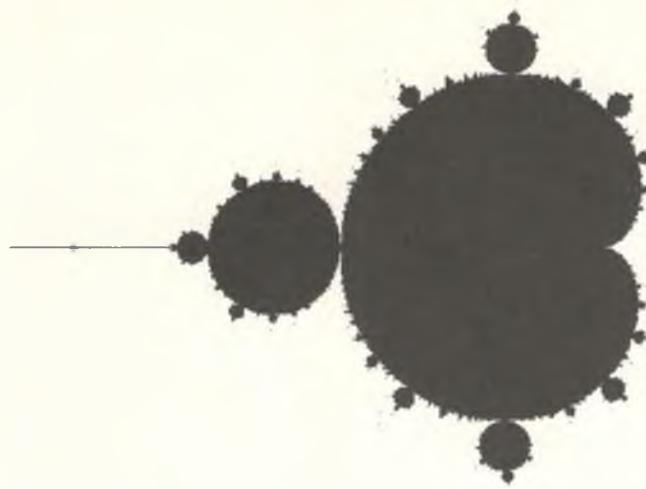
<sup>1</sup> Si no está familiarizado con los números complejos, basta con tratar  $i$  como un símbolo especial y sólo conocer las definiciones de la suma y la multiplicación de los números complejos. Si  $z = c + di$  es un segundo número complejo, entonces se define que  $x + z$  es  $(a + c) + (b + d)i$ , y  $xz$  se define como  $(ac - bd) + (ad + bc)i$ . Podemos representar los números complejos como puntos en el plano identificando el punto  $(a, b)$  con el número complejo  $(a + bi)$ . El **módulo** del número  $a + bi$  es el número real  $(a^2 + b^2)^{1/2}$ , que nos da una medida del “tamaño” del número complejo.



**Figura 9.28** Conjunto de Julia-Fatou. (a)  $c = -0.12375 + 0.056805i$ . (b)  $c = -0.012 + 0.74i$ .

Observe que la región en la figura 9.28(b) no está tan bien conectada como la que se muestra en la parte (a) de la figura. En la parte (b), algunos puntos caen hacia los tres puntos negros, otros van hacia el infinito y otros no corresponden a ninguna de estas dos categorías. Estos últimos puntos son los que se dibujan como contorno de la forma en la parte (b). La forma del conjunto de Julia-Fatou depende obviamente del valor del número  $c$ . Si calculamos los valores de los conjuntos de Julia para todos los valores posibles de  $c$  y coloreamos de negro el punto  $c$  cuando el conjunto de Julia-Fatou está conectado (es decir, está compuesto por una sola pieza, no dividido en “islas” disjuntas) y de blanco cuando el conjunto no está conectado, obtenemos el objeto que se presenta en la figura 9.29, conocido como **conjunto de Mandelbrot**. Observe que el conjunto de Mandelbrot es autosimilar, ya que alrededor de la orilla del disco grande del conjunto hay varios conjuntos más pequeños, cada uno de los cuales se parece mucho a una versión reducida del grande.

Por fortuna, hay una manera más sencilla de generar aproximaciones al conjunto de Mandelbrot: para cada valor de  $c$ , tome el número complejo  $0 = 0 + 0i$  y aplique a él el proceso  $x \rightarrow x^2 + c$  un número finito de veces (digamos 1000). Si después de este número de iteraciones está fuera del disco definido por módulo  $< 100$ , entonces  $c$  se colorea de blanco; en caso contrario, se le asigna el color negro. Conforme aumenta el número de iteraciones y el radio del disco, la imagen resultante se convierte en una mejor aproximación al conjunto. Peit-

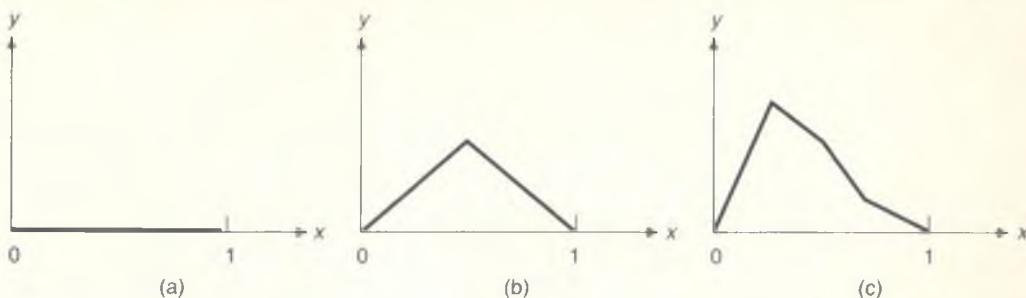


**Figura 9.29** Conjunto de Mandelbrot. Cada punto  $c$  en el plano complejo se presenta en negro si el conjunto de Julia para el proceso  $x \rightarrow x^2 + c$  está conectado.

gen y Richter [PEIT86] proporcionan direcciones explícitas para generar varias imágenes espectaculares de los conjuntos de Mandelbrot y de Julia-Fatou.

Estos resultados son muy sugerentes para el modelado de formas naturales, ya que muchos objetos naturales parecen exhibir una gran autosimilitud. Las montañas tienen picos, picos más pequeños, rocas y grava, todo lo cual parece similar; los árboles tienen troncos, ramas y tallos, que también se ven similares; las costas tienen bahías, caletas, estuarios, riberas y zanjas de drenado, que también son de apariencia similar. Por lo tanto, el modelado de la autosimilitud a cierta escala parece ser una forma de generar modelos atractivos de fenómenos naturales. La escala a la cual se detiene la autosimilitud no es un tema importante en este momento, ya que el propósito es el modelado y no las matemáticas. Por ello, cuando un objeto se ha generado recursivamente a través de suficientes pasos, de manera que los cambios adicionales ocurren por debajo de la resolución de los pixeles, no es necesario continuar.

Fournier, Fussell y Carpenter [FOUR82] desarrollaron un mecanismo para generar una clase de montañas fractales con base en subdivisiones recursivas. Es más fácil explicar este proceso en una dimensión. Suponga que comenzamos con un segmento de línea sobre el eje  $x$ , como se muestra en la figura 9.30(a). Si subdividimos la línea en mitades y luego movemos el punto medio cierta distancia en la dirección  $y$ , obtenemos la forma presentada en la figura 9.30(b). Para continuar con la subdivisión de segmentos, se calcula un nuevo valor para el punto medio del segmento de  $(x_i, y_i)$  a  $(x_{i+1}, y_{i+1})$ , de la siguiente manera:  $x_{\text{nueva}} = \frac{1}{2}(x_i + x_{i+1})$ ,  $y_{\text{nueva}} = \frac{1}{2}(y_i + y_{i+1}) + P(x_{i+1} - x_i) R(x_{\text{nueva}})$ , donde  $P( )$  es una función que determina la extensión de la perturbación en términos del tamaño



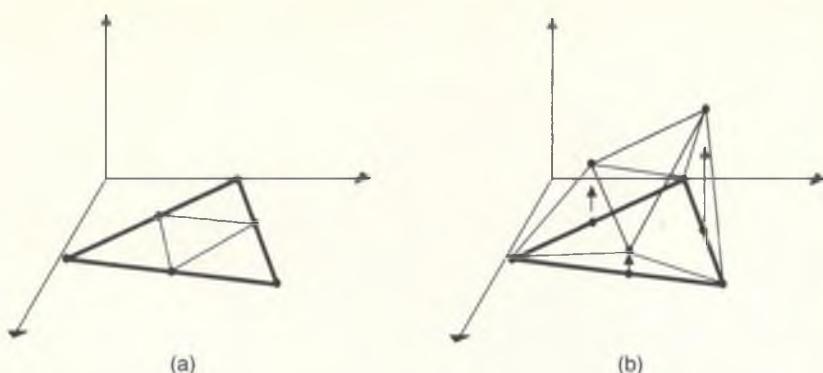
**Figura 9.30** Segmento de línea en el eje  $x$ . (b) El punto medio de la línea se ha trasladado en la dirección y en una cantidad aleatoria. (c) Resultado de una iteración adicional.

de la linea que se perturba, y  $R(\cdot)$  es un número aleatorio<sup>2</sup> entre 0 y 1 seleccionado con base en  $x_{\text{nueva}}$  (véase la Fig. 9.30c). Si  $P(s) = s$ , el primer punto no se puede desplazar más de uno, cada uno de los dos puntos siguientes (que a lo sumo están a una altura de  $\frac{1}{2}$ ) no se puede desplazar más de  $\frac{1}{2}$ , etcétera. De esta manera, todos los puntos resultantes caben en un cuadrado unidad. Para  $P(s) = s^a$ , la forma del resultado depende del valor de  $a$ ; los valores menores de  $a$  producen perturbaciones mayores y viceversa. Por supuesto, pueden usarse otras funciones, como  $P(s) = 2^{-s}$ .

Fournier, Fussell y Carpenter usan este proceso para modificar formas bidimensionales de la siguiente manera. Comienzan con un triángulo, marcan el punto medio de cada arista y conectan los tres puntos medios, como se muestra en la figura 9.31(a). La coordenada  $y$  de cada punto medio se modifica en la forma que acabamos de describir, de manera que el conjunto resultante de cuatro triángulos tiene la apariencia presentada en la figura 9.31(b). Este proceso, al iterarse, produce montañas bastante realistas, como se muestra en la ilustración en color 11 (aunque, desde una vista superior, se percibe una estructura poligonal muy regular).

Observe que podemos comenzar con una disposición de triángulos que tenga cierta forma y luego aplicamos este proceso para generar detalles más finos. Esta capacidad es muy importante en algunas aplicaciones de modelado, donde la disposición de los objetos en una escena puede ser estocástica a bajo nivel pero ordenada a mayor nivel: el follaje de un jardín ornamental se puede generar con un mecanismo estocástico, pero su disposición en el jardín debe seguir reglas estrictas. Por otra parte, el hecho de que la estructura de alto nivel de la

<sup>2</sup>  $R(\cdot)$  es en realidad una *variable aleatoria*, es decir, una función que recibe números reales y produce números distribuidos aleatoriamente entre 0 y 1. Si esto se implanta con un generador de números seudoaleatorios, tiene la ventaja de que los fractales pueden repetirse: se pueden generar de nuevo al proporcionar la misma semilla al generador de números seudoaleatorios.



**Figura 9.31** (a) Subdivisión de un triángulo en cuatro más pequeños. Los puntos medios del triángulo original se perturban en la dirección y para generar la forma de la parte (b).

disposición inicial de los triángulos persista en las subdivisiones iteradas quizás no sea apropiado en ciertas aplicaciones (específicamente, el fractal generado esta manera no tiene todas las autosimilitudes estadísticas que están presentes en los fractales basados en el movimiento browniano [MAND82]). Además, como la posición de cualquier vértice se ajusta una sola vez y de ahí en adelante permanece estacionario, tienden a desarrollarse pliegues en la superficie a lo largo de las aristas entre los triángulos originales, un resultado que quizás no parezca natural.

La generación de imágenes fractales puede ser difícil. Si los fractales se generan en una memoria  $\mathbb{Z}$ , dibujar todo el objeto requiere mucho tiempo debido a la enorme cantidad de polígonos. En la generación por línea de barrido, requiere mucho tiempo para ordenar todos los polígonos de manera que solo se consideren aquellos que intersecan la línea de rastreo. Sin embargo, la traza de rayos con fractales es muy complicada, ya que es necesario revisar cada rayo para determinar la intersección con cada uno de los polígonos, que pueden llegar a miles o millones. Kajiya [KAJI83] presentó un método para la traza de rayos en objetos fractales de la clase descrita en [FOUR82] y Bouville [BOUV85] mejoró este algoritmo encontrando un mejor volumen acotante para los objetos.

### 9.5.2 Modelos gramaticales

Smith [SMIT84] presenta un método, desarrollado originalmente por Lindenmayer [LIND68], para describir la estructura de ciertas plantas, usando lenguajes gramaticales (**gramáticas L**) de grafos paralelos, lo que Smith denomina **graftales**. Estos lenguajes se describen con una gramática que consiste en una colección de producciones que se aplican todas a la vez. Lindenmayer extendió los lenguajes para que incluyeran corchetes, de manera que el alfabeto contenga

ra los dos símbolos especiales “[” y “]”. Un ejemplo típico es la gramática con el alfabeto {A, B, [, ]} y dos reglas de producción:

1. A → AA
2. B → A[B]AA[B]

Comenzando por el axioma A, las primeras generaciones son A, AA, AAAA, etc.; si comenzamos por el axioma B, las primeras generaciones son

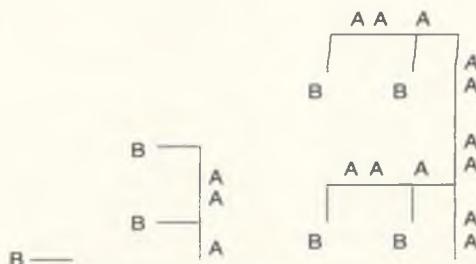
0. B
1. A[B]AA[B]
2. AA[A[B]AA[B]]AAAA[A[B]AA[B]]

etcétera. Si decimos que una palabra en el lenguaje representa una secuencia de segmentos en una estructura gráfica y que las porciones entre corchetes representan porciones que se ramifican a partir del símbolo que las precede, entonces las figuras relacionadas con estos tres niveles son las que se presentan en la figura 9.32.

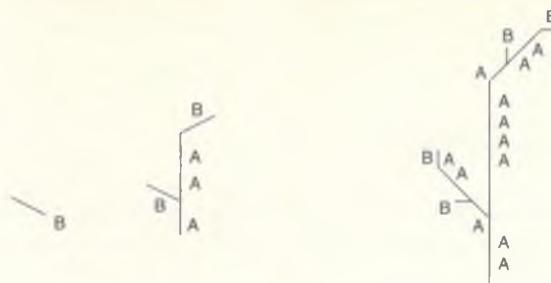
Esta secuencia de imágenes tiene una agradable estructura de ramificación, pero sería más atractivo un árbol un poco más balanceado. Si añadimos al lenguaje los símbolos de paréntesis, “(” y “)”, y alteramos la segunda producción para que sea A[B]AA(B), la segunda generación será

2. AA[A[B]AA(B)]AAAA(A[B]AA(B))

Si decimos que los corchetes denotan una rama izquierda y que los paréntesis denotan una rama derecha, las imágenes relacionadas son las que se presentan en la figura 9.33. Al avanzar hacia generaciones posteriores en este lenguaje, se obtienen grafos que representan patrones muy complejos. Estos grafos tienen un tipo de autosimilitud, ya que el patrón descrito por la palabra de la  $n$ -ésima



**Figura 9.32** Representaciones de árbol de las tres primeras palabras del lenguaje. Todas las ramas se dibujan a la izquierda del eje principal actual.



**Figura 9.33** Representaciones de árbol de las tres primeras palabras, pero con el lenguaje de ramificación a ambos lados. Hemos acortado los segmentos del árbol conforme avanzamos por las generaciones.

generación está contenido (repetidamente, en este caso) en la palabra de la ( $n - 1$ )-ésima generación.

La generación de un objeto a partir de una palabra es un proceso distinto del de la generación de la palabra en sí. En este caso, los segmentos del árbol han dibujado con longitudes cada vez menores, todos los ángulos de ramificación han sido de  $45^\circ$  y las ramas van hacia la izquierda o hacia la derecha. Si elegimos diferentes ángulos de ramificación para las ramas a distinta profundidad, y grosor diferente para las líneas (o incluso cilindros) que representan segmentos, varían los resultados; dibujar una “flor” o una “hoja” en los nodos terminales del árbol mejora aún más la imagen. La gramática no tiene contenido geométrico inherente, de manera que la utilización de un modelo gramatical requiere una interpretación tanto gramatical como geométrica del lenguaje.

Este tipo de extensión de los lenguajes y la interpretación de las palabras en el lenguaje (es decir, las imágenes formadas a partir de las palabras) ha desarrollado por varios investigadores [REFF88; PRUS88]. Las gramáticas han enriquecido para permitirnos llevar un registro de la “edad” de la letra en una palabra, de modo que las letras viejas y jóvenes se transformen de distinta manera (este registro de edades se puede efectuar con reglas de la forma  $A \rightarrow B \rightarrow C, C \rightarrow D, \dots, Q \rightarrow QG[Q]$ , para que no ocurran transiciones interesadas hasta que “envejezca” la planta). Gran parte del trabajo se ha centrado en la obtención de gramáticas que representen con precisión la biología de plantas durante el desarrollo.

Sin embargo, al llegar a cierto punto una gramática deja de ser manejable como descriptora de plantas: se han añadido demasiadas características adicionales a la gramática o a la interpretación de una palabra de ella. En el modelo de Reffye [REFF88], la simulación del crecimiento de una planta es controlada por una pequeña colección de parámetros que se describen en términos biológicos y que se pueden incorporar al algoritmo. Las producciones de la gramática se aplican en forma probabilística y no determinista.

En este modelo comenzamos, como antes, con un solo tallo. En la punta de este tallo se encuentra un brote que puede sufrir varias transformaciones:

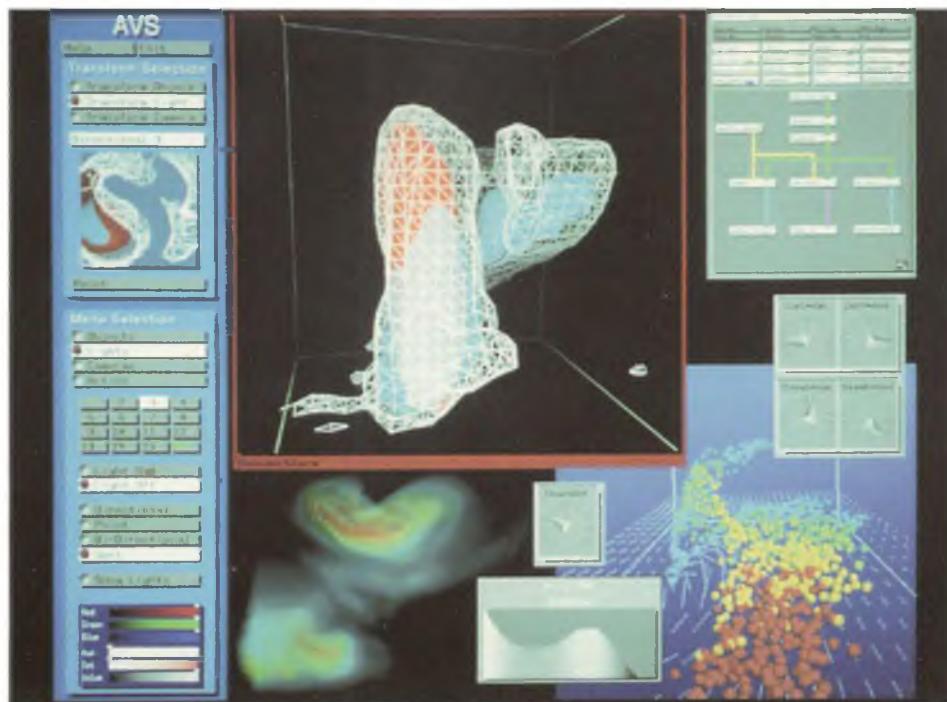


Ilustración 1. Tormenta de tornado rigurosa, por R. Williamson, L. Wicker y C. Shaw, NCSA, University of Illinois. (Application Visualization System por Stardent Computer.)



Ilustración 2. *The Abyss* — Secuencia de seudópodo. (Copyright © 1989 Twentieth Century Fox. Derechos Reservados. Cortesía de Industrial Light & Magic, Computer Graphics Division.)

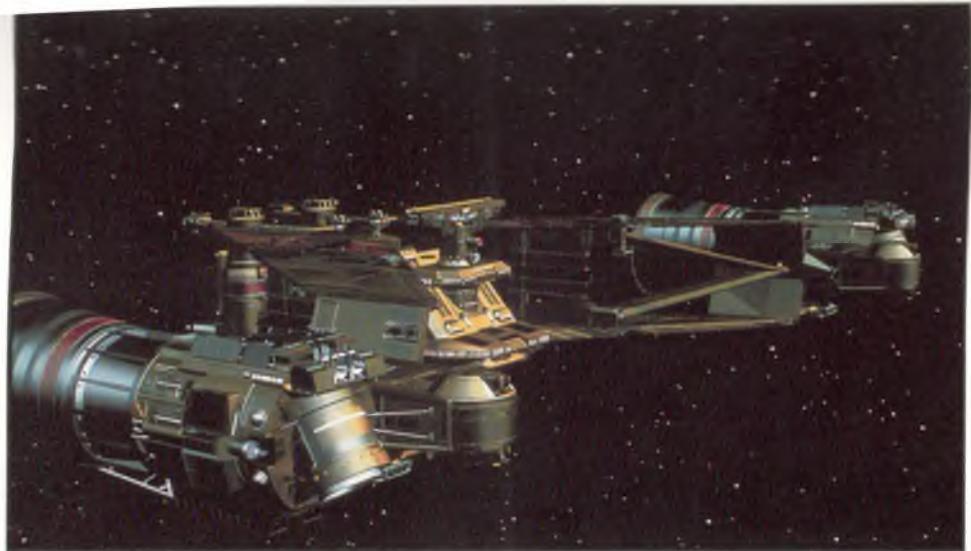


Ilustración 3. Nave de mando de *The Last Starfighter*. La nave con correspondencia de textura tiene 450 000 polígonos. (Copyright © 1984 Digital Productions. Cortesía de G. Demos.)



(a)



(b)

Ilustración 4. (a) Cabina de un simulador de vuelo F5; la vista del piloto se proyecta sobre un domo que rodea la cabina. (b) La vista desde la cabina del simulador de vuelo. El avión de combate se modela geométricamente, mientras que el terreno tiene textura de fotografía. (Cortesía de R. Economy, General Electric Company.)



Ilustración 5. Juego de video *Hard Drivin'*. (Cortesía de Atari Games Corporation, copyright © 1988, Atari Games Corporation.)

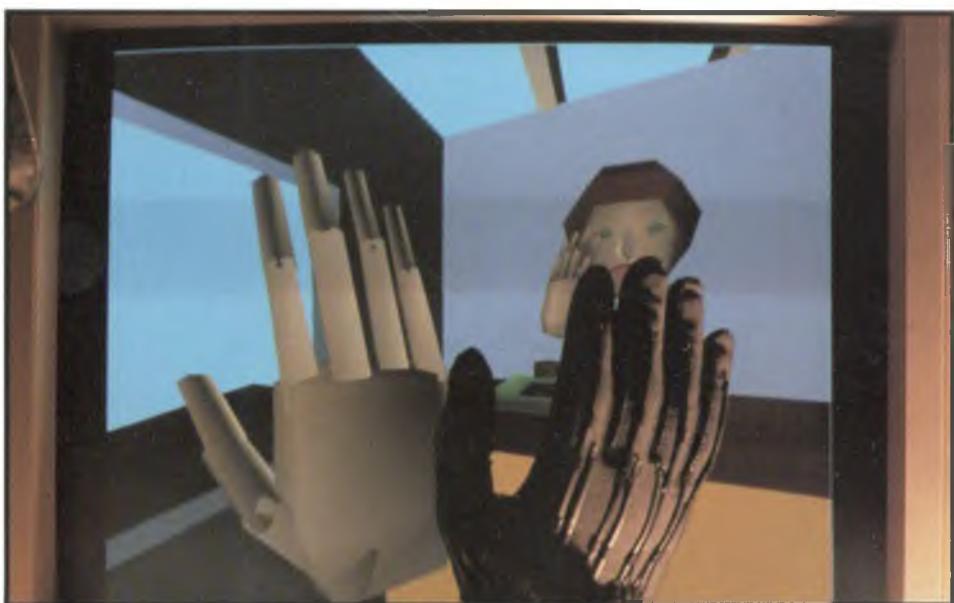
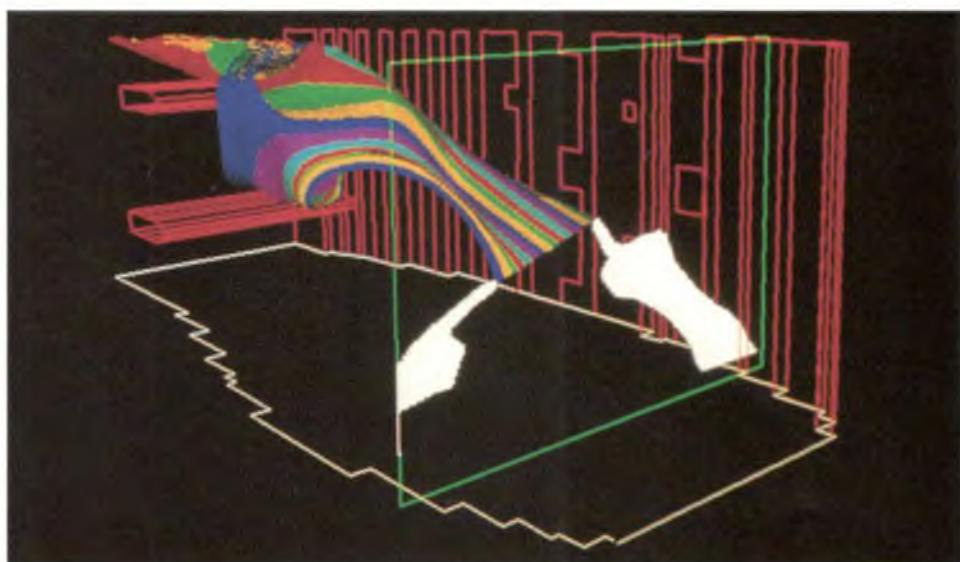


Ilustración 6. DataGlove (derecha) e imagen por computador del guante. El DataGlove mide los movimientos de los dedos y la orientación y posición de la mano. La imagen por computador rastrea los cambios. (Cortesía de Jaron Lanier, VPL.)



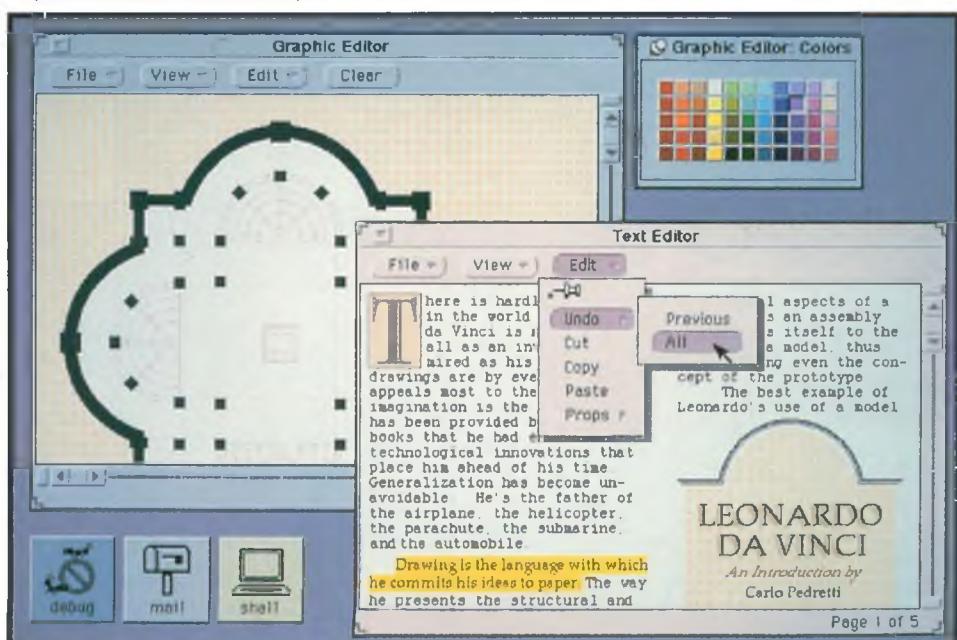
**Ilustración 7.** Usuario con una pantalla estéreo montada en la cabeza, DataGloves y micrófono para emitir mandatos. Estos dispositivos se emplean para crear una realidad virtual para el usuario, cambiando la presentación en la pantalla estéreo cuando se mueve la cabeza y utilizando los DataGloves para manipular objetos generados por computador. (Cortesía de Michael Mc Greevey y Scott Fisher, NASA Ames Research Center, Moffett Field, California.)



**Ilustración 8.** Sistema Videotouch de Krueger, en el cual los movimientos de la mano del usuario se usan para manipular un objeto. El contorno de la mano se presenta junto con los objetos para proporcionar una retroalimentación natural. (Cortesía de Myron Krueger, Artificial Reality Corp.)



**Ilustración 9.** Interfaz con el usuario OSF/Motif. Las barras deslizantes de color se emplean para definir los colores que se usarán en las ventanas. Observe la utilización del sombreado en las orillas de los botones, menús, etc., para crear un efecto tridimensional. (Cortesía de Open Software Foundation.)



**Ilustración 10.** Interfaz con el usuario OPEN LOOK. Se usa el amarillo para realizar el texto seleccionado. Se emplean sombras suaves para el fondo y los bordes de las ventanas. (Cortesía de Sun Microsystems.)

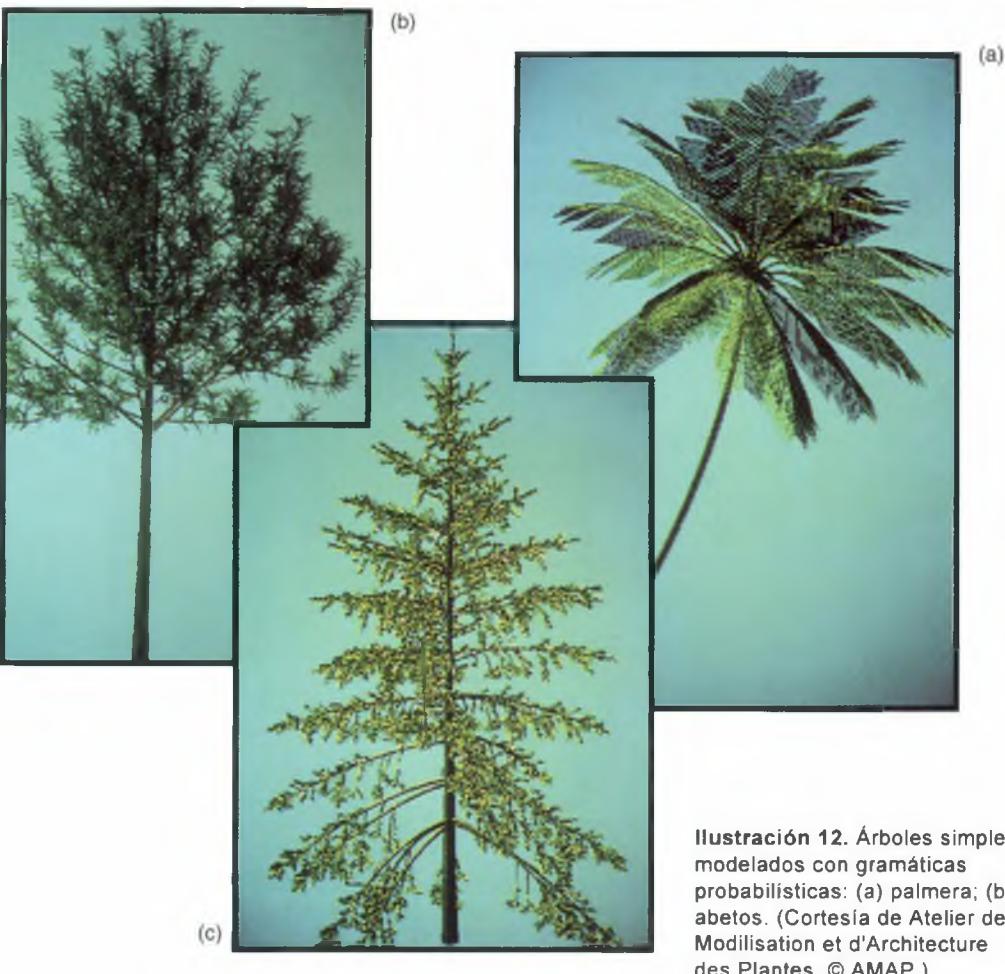


Ilustración 12. Árboles simples modelados con gramáticas probabilísticas: (a) palmera; (b) y (c) abetos. (Cortesía de Atelier de Modélisation et d'Architecture des Plantes, © AMAP.)

Página opuesta: Ilustración 11. "Vol. Libre Ridge": montañas fractales generadas con el algoritmo de Fournier-Fussell-Carpenter. (Cortesía de Loren Carpenter.)



Ilustración 13. Playa al atardecer. (Cortesía de Bill Reeves, Pixar y Alan Fournier, University of Toronto.)

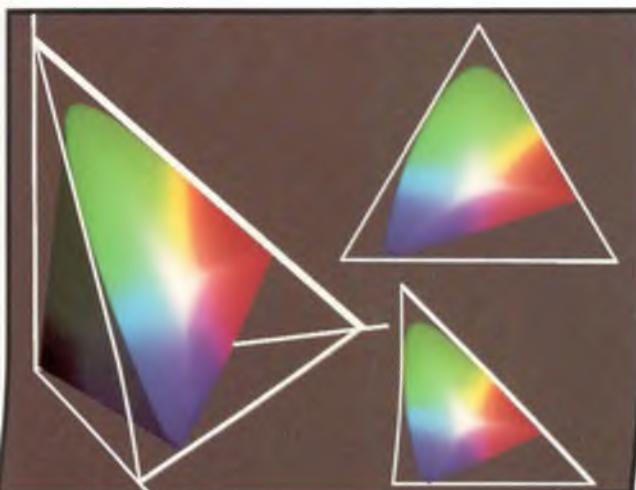
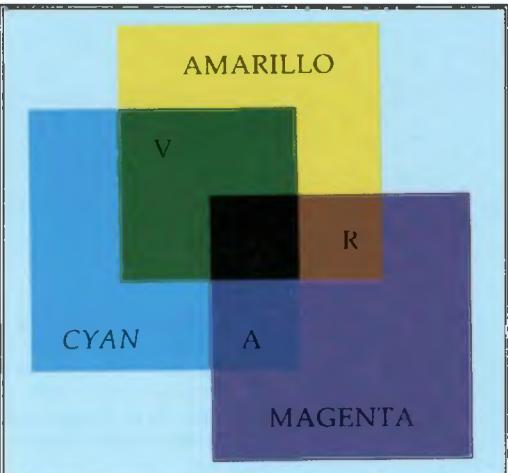


Ilustración 14. Varias vistas del plano  $X + Y + Z = 1$  del espacio CIE. Izquierda: el plano incorporado en el espacio CIE. Superior derecha: vista perpendicular al plano. Inferior derecha: proyección sobre el plano ( $X, Y$ ) (es decir, el  $Z = 0$ ), que es el diagrama de cromaticidad. (Cortesía de Barbara Meier, Brown University.)

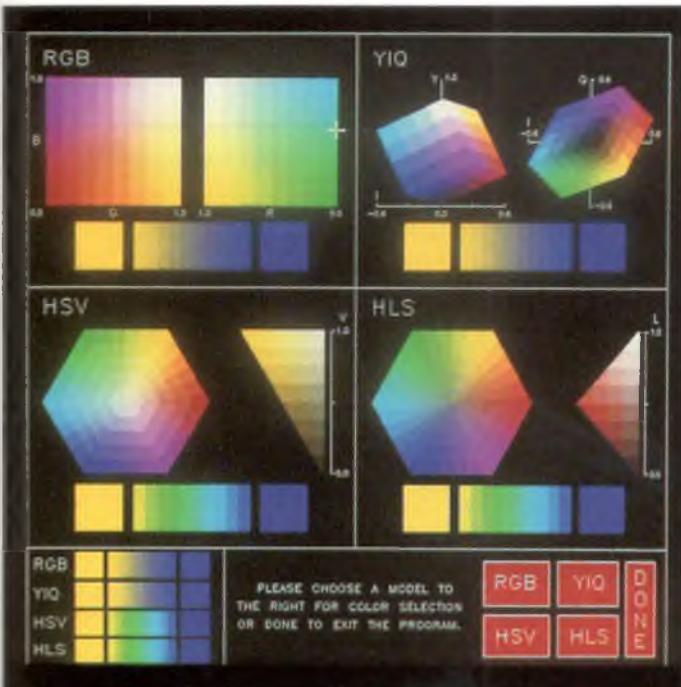
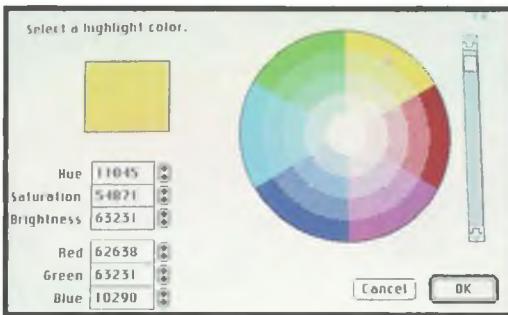
**Ilustración 15.** Diagrama de cromaticidad CIE, que presenta las gamas de color típicas para una imprenta por offset, un monitor en color y película para diapositivas. Los colores de impresión representan los colores estándar S.W.O.P. de la Graphic Arts Technical Foundation medidos bajo una luz de artes gráficas con temperatura de color de 5000° K. El monitor en color es un Barco CTVM 3/51 con un punto blanco ajustado a 6500° K y la película de diapositivas es Kodak Ektachrome 5017 ISO 64 caracterizada bajo una fuente CIE A: un objeto negro de 2653° K que aproxima cercanamente una lámpara de tungsteno. La x, el círculo y el cuadrado indican los puntos blancos para las gamas de impresión, el monitor en color y la película, respectivamente. (Cortesía de M. Stone, Xerox Palo Alto Research Center. Gama de película medida por A. Paeth, Computer Graphics Lab., University of Waterloo; véase también el primer apéndice de [PAET89]).

**Ilustración 16.** Colores aditivos. El rojo más el verde forman el amarillo, el rojo más el azul forman el magenta, el verde más el azul forman el cyan, y el rojo más el verde más el azul forman el blanco.



**Ilustración 17.** Colores sustractivos. El amarillo y el magenta restados al blanco forman el rojo, el amarillo y el cyan restados al blanco forman el verde, el cyan y el magenta restados al blanco forman el azul.

**Ilustración 18.** Técnica de interacción utilizada en el Macintosh para especificar colores en el espacio HSV. El tinte y la saturación se presentan en el área circular, mientras que el valor se indica con un control deslizante. El usuario puede mover la marca en el área circular y modificar el control deslizante, o puede teclear nuevos valores HSV o RGB. El área cuadrada con color (parte superior izquierda) muestra el color actual y el nuevo color.



**Ilustración 19.** Programa interactivo que permite al usuario especificar e interpolar colores en cuatro espacios de colores diferentes: RGB, YIQ, HSV y HLS. Los colores inicial y final para una interpolación lineal se especifican apuntando a las diversas proyecciones de los espacios de colores. Las interpolaciones se presentan debajo de cada espacio de color, y juntas en la parte inferior izquierda para fines comparativos. (Cortesía de Paul Charlton, The George Washington University.)

**Ilustración 20.** Imagen en seudocolor que muestra la topografía de Venus. La escala de color a la izquierda indica las altitudes de -22 km a +2 km con respecto a un radio medio de 6052 km para Venus. El Lunar and Planetary Institute calculó los datos con base en observaciones altimétricas por radar desde la nave espacial Pioneer Venus Orbiter de NASA. La imagen se creó con el National Space Science Data Center Graphics System. (Cortesía de Lloyd Treinish, NASA Goddard Space Flight Center.)

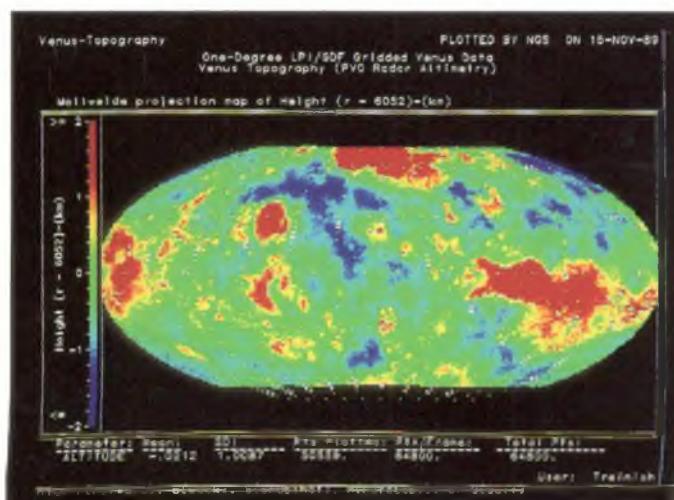




Ilustración 21. Chevrolet Camaro iluminado por cinco luces con controles de iluminación de Warn. (Cortesía de David R. Warn, General Motors Research Laboratories.)

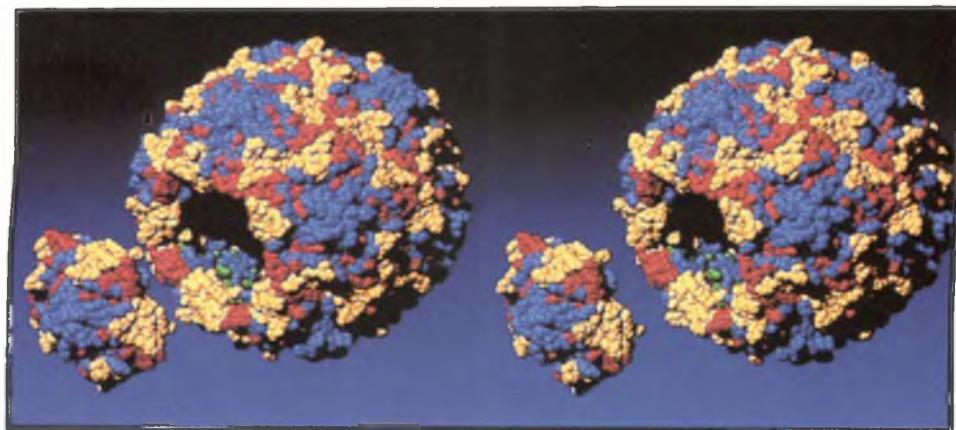


Ilustración 22. Par estéreo del virus de polio, generado colocando una esfera con radio de 0.5 nm en cada posición de carbono alfa. Se ha desprendido un pentámero para mostrar el interior. Coordenadas por cortesía de J. Hogle. (Cortesía de David Goodsell y Arthur Olsen.)

Copyright

© 1989, Research Institute of Scripps Clinic.)

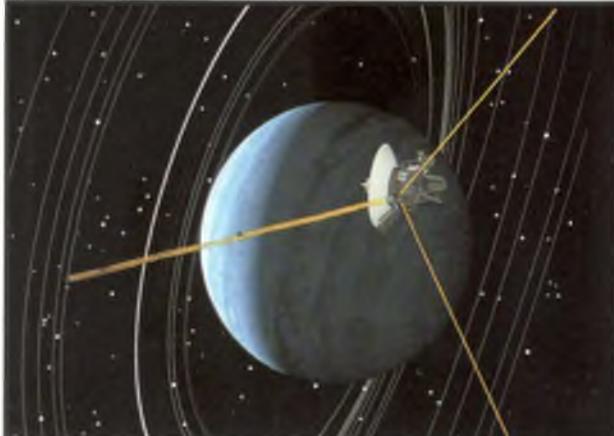
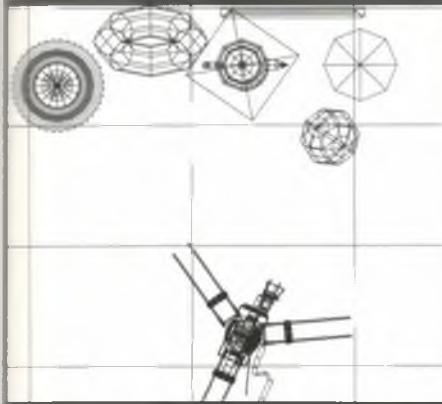
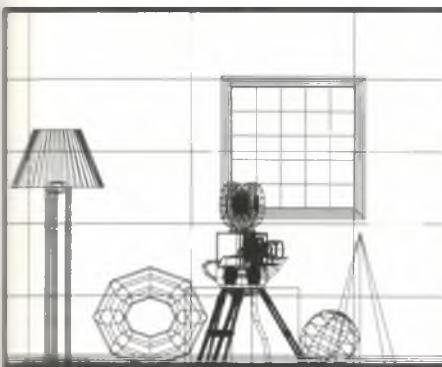


Ilustración 23. Simulación de un vuelo por Urano con anillos y órbita. (Cortesía de Jim Blinn, Computer Graphics Lab, Jet Propulsion Lab, California Institute of Technology.)

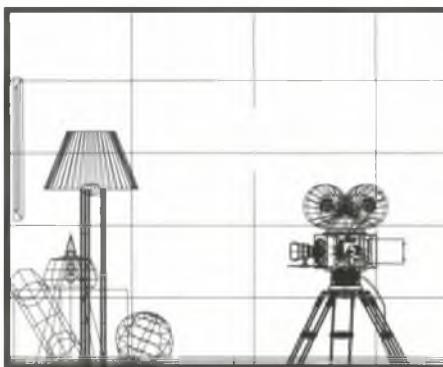
**Ilustración 24.** *Shutterbug*. Escena de sala con cámara de cine. Proyecciones ortográficas (Secs. 6.2.2 y 12.3.1). (a) Vista de planta. (b) Vista frontal. (c) Vista lateral. Modelos poligonales generados con parches de *spline*. (Copyright © 1990, Pixar. Generado por Thomas Williams y H. B. Siegal usando software PhotoRealistic Renderman™ de Pixar.)



(a)

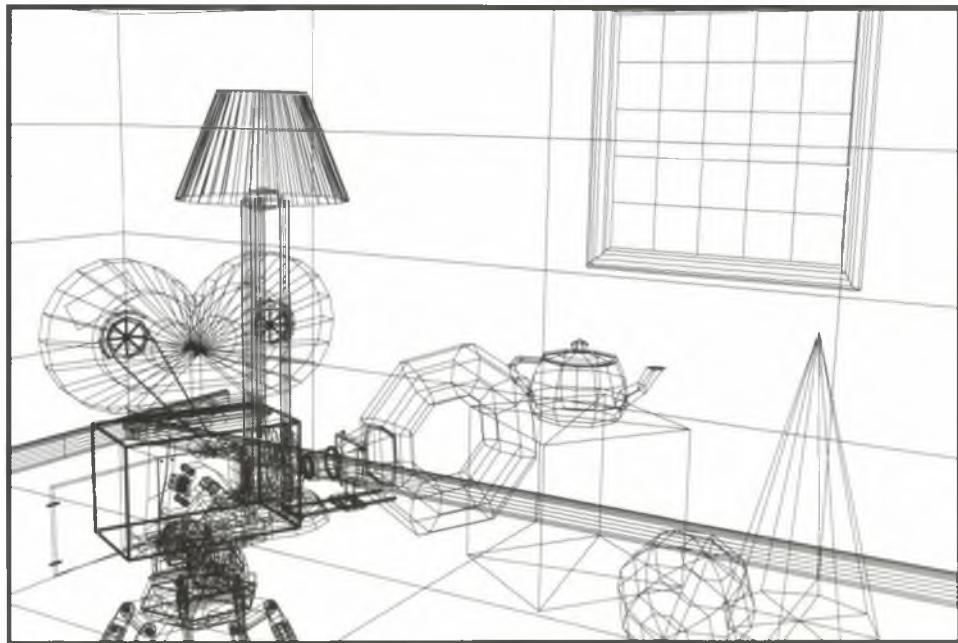


(b)



(c)

**Ilustración 25.** *Shutterbug*. Proyección de perspectiva (Secs. 6.2.1 y 12.3.2). (Copyright © 1990, Pixar. Generado por Thomas Williams y H.B. Siegal usando software PhotoRealistic Renderman™ de Pixar.)



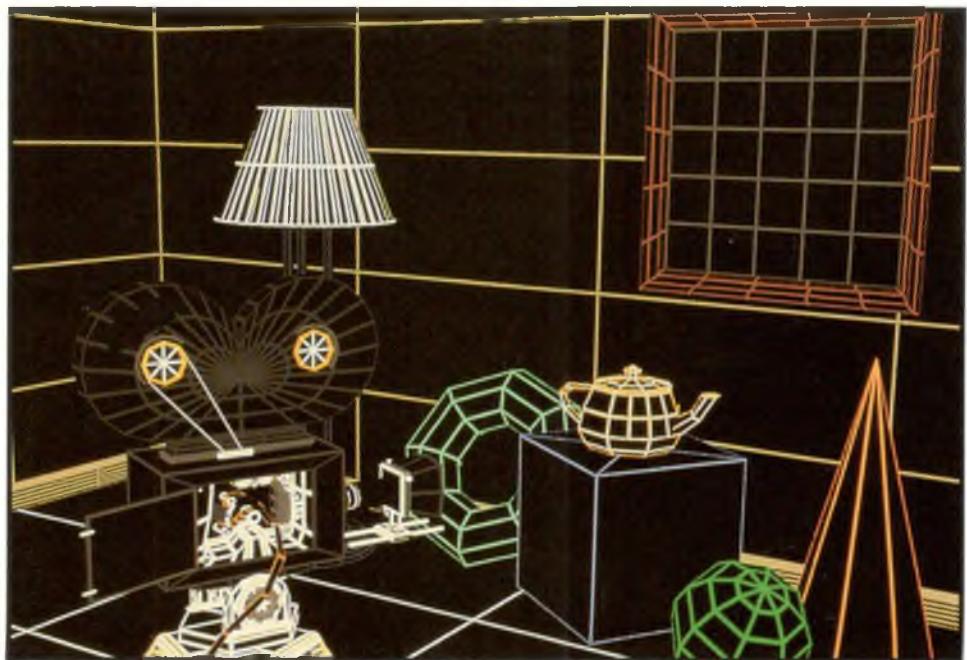


Ilustración 26. *Shutterbug*. Determinación de líneas visibles (Sec. 12.3.7). (Copyright © 1990, Pixar. Generado por Thomas Williams y H.B. Siegal usando software PhotoRealistic Renderman™ de Pixar.)



Ilustración 27. *Shutterbug*. Determinación de superficies visibles usando sólo iluminación ambiental (Secs. 12.4.1 y 14.1.1). (Copyright © 1990, Pixar. Generado por Thomas Williams y H.B. Siegal usando software PhotoRealistic Renderman™ de Pixar.)

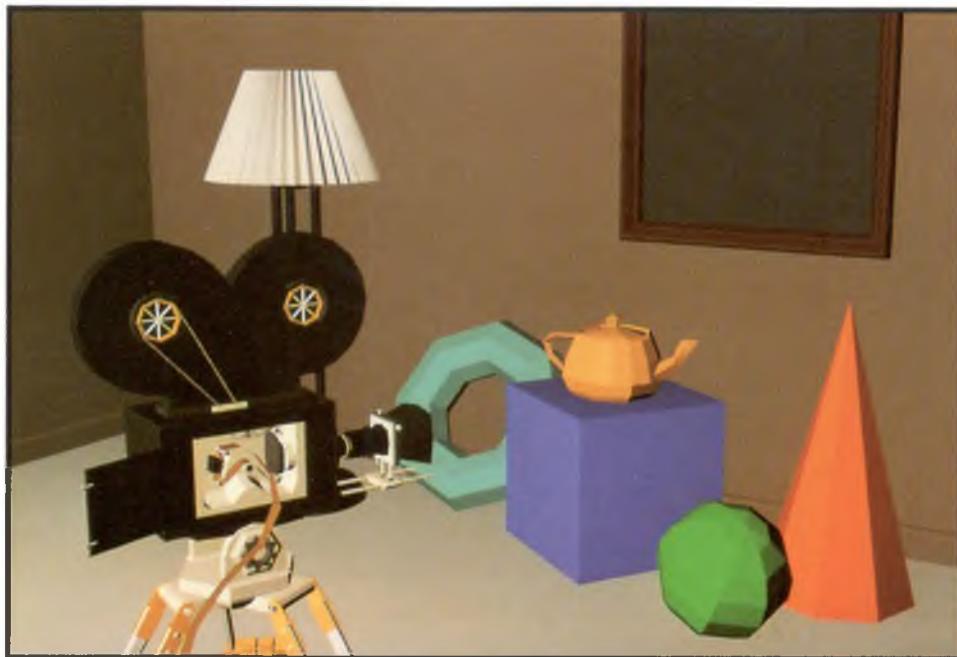
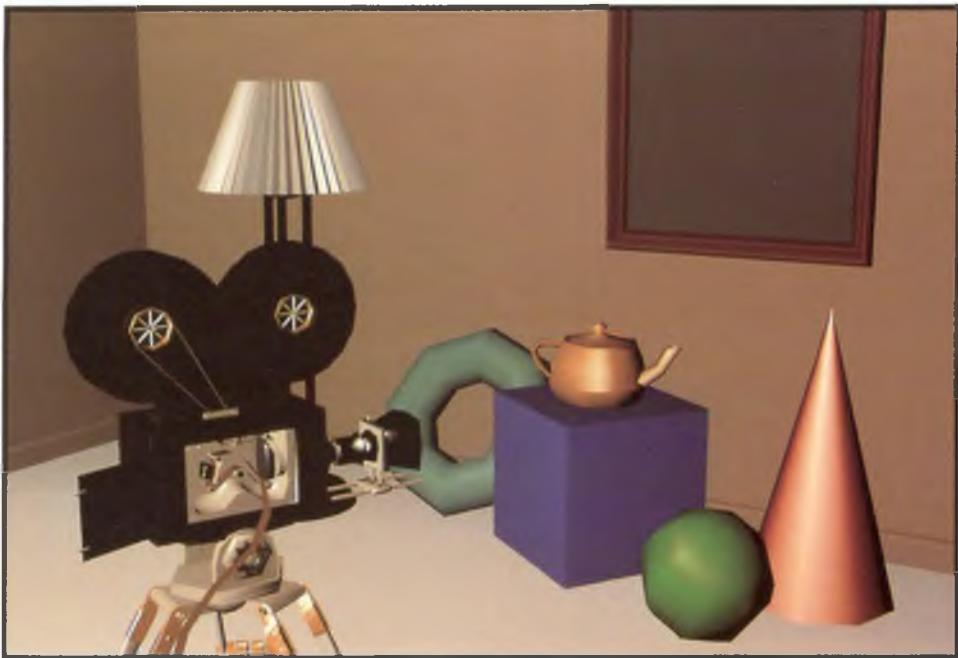


Ilustración 28. *Shutterbug*. Polígonos sombreados individualmente con reflexión difusa (Secs. 12.4.2 y 14.2.3). (Copyright © 1990, Pixar. Generado por Thomas Williams y H.B. Siegal usando software PhotoRealistic Renderman™ de Pixar.)



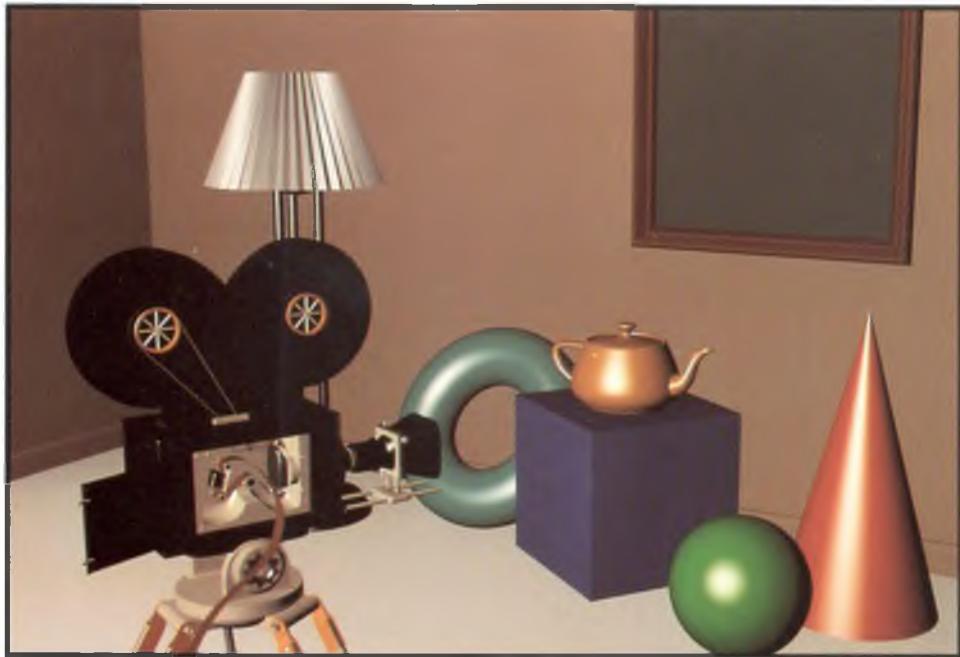
Ilustración 29. *Shutterbug*. Polígonos con sombreado de Gouraud y reflexión difusa (Secs. 12.4.2 y 14.2.3). (Copyright © 1990, Pixar. Generado por Thomas Williams y H.B. Siegal usando software PhotoRealistic Renderman™ de Pixar.)



**Ilustración 30.** *Shutterbug*. Polígonos con sombreado de Gouraud y reflexión especular (Secs. 12.4.4 y 14.2.4). (Copyright © 1990, Pixar. Generado por Thomas Williams y H.B. Siegal usando software PhotoRealistic Renderman™ de Pixar.)



**Ilustración 31.** *Shutterbug*. Polígonos con sombreado de Phong y reflexión especular (Secs. 12.4.4 y 14.2.5). (Copyright © 1990, Pixar. Generado por Thomas Williams y H.B. Siegal usando software PhotoRealistic Renderman™ de Pixar.)



**Ilustración 32.** *Shutterbug*. Superficies curvas con reflexión especular (Sec. 12.4.5). (Copyright © 1990, Pixar. Generado por Thomas Williams y H.B. Siegal usando software PhotoRealistic Renderman™ de Pixar.)



**Ilustración 33.** *Shutterbug*. Modelo de iluminación mejorado y luces múltiples (Secs. 12.4.6 y 14.1). (Copyright © 1990, Pixar. Generado por Thomas Williams y H.B. Siegal usando software PhotoRealistic Renderman™ de Pixar.)



Ilustración 34. *Shutterbug*. Correspondencia de textura (Secs. 12.4.7 y 14.3.2). (Copyright © 1990, Pixar. Generado por Thomas Williams y H.B. Siegal usando software PhotoRealistic Renderman™ de Pixar.)



Ilustración 35. *Shutterbug*. Correspondencia de desplazamiento (Secs. 12.4.7 y 14.3.4). (Copyright © 1990, Pixar. Generado por Thomas Williams y H.B. Siegal usando software PhotoRealistic Renderman™ de Pixar.)

**Ilustración 36.**  
*Shutterbug.*  
Correspondencia  
de reflexión  
(Sec. 12.4.9).  
(Copyright  
© 1990, Pixar.  
Generado por  
Thomas  
Williams y H.B.  
Siegal usando  
software  
PhotoRealistic  
Renderman™  
de Pixar.)



a)



**Ilustración 37.** Profundidad de campo, implantada con postprocesamiento (Sec. 12.4.10). (a) Enfocado en el cubo (550 mm), apertura f/11. (b) Enfocado en la esfera (290 mm), apertura f/11. (Cortesía de Michael Potmesil e Indranil Chakravarty, RPI.)



(b)



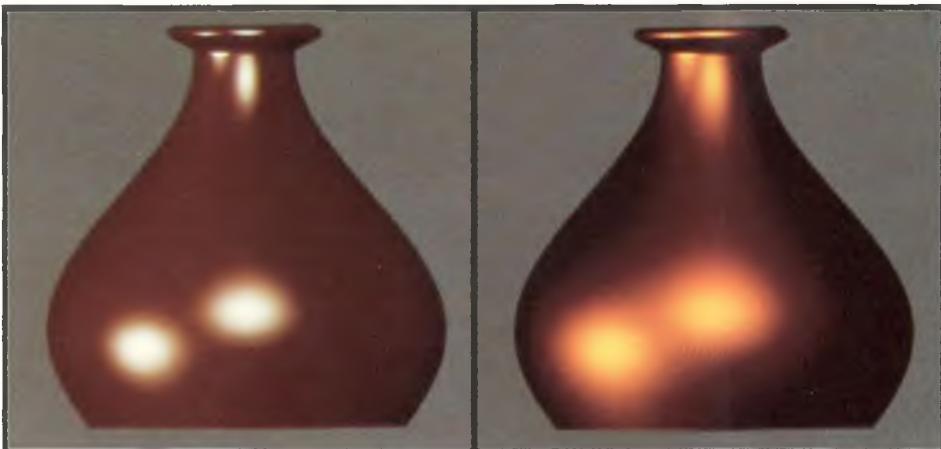
**Ilustración 38.** Toro con correspondencia de protuberancias creado usando una función de protuberancias generadas a mano (Sec. 14.3.3). (Por Jim Blinn. Cortesía de University of Utah.)

**Ilustración 39.** Fresa con correspondencia de protuberancias creada usando una función de protuberancias generadas a mano (Sec. 14.3.3). (Por Jim Blinn. Cortesía de University of Utah.)

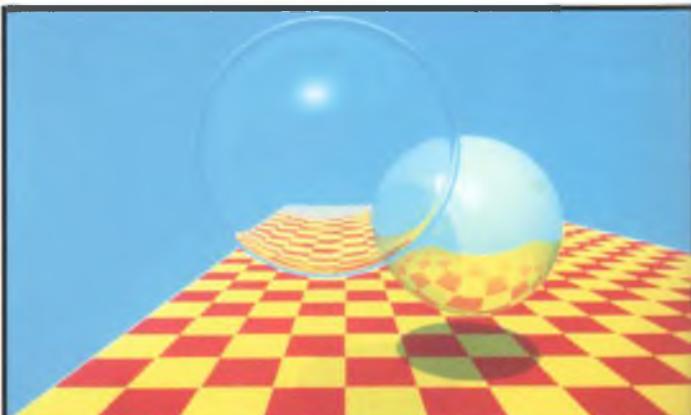


(a)

(b)



**Ilustración 40.** Dos jarrones generados con el modelo de iluminación de Cook-Torrance (Sec. 14.1.7). Ambos jarrones están iluminados por dos luces con  $I_a = I_s =$  iluminante CIE estándar D6500,  $d\omega_{I_1} = 0.0001$  y  $d\omega_{I_2} = 0.0002$ ;  $I_a = 0.01 I_s$ ;  $p$  = reflectividad bidireccional del cobre para la incidencia normal;  $\rho_s = \pi p_d$ . (a) Plástico con color de cobre;  $k_s = 0.1$ ;  $F$  = reflectividad de un espejo de vinilo;  $D$  = función de Beckmann con  $m = 0.15$ ;  $k_d = 0.9$ . (b) Metal de cobre;  $k_s = 1.0$ ;  $F$  = reflectividad de un espejo de cobre;  $D$  = función de Beckmann con  $m = 0.4$ ,  $w_1 = 0.4$ ,  $m_2 = 0.2$ ,  $w_2 = 0.6$ ,  $k_d = 0.0$ . (Por Robert Cook, Program of Computer Graphics, Cornell University.)



**Ilustración 41.** Esferas y tablero de ajedrez. Una de las primeras imágenes producidas con traza de rayos recursiva (Sec. 14.7). (Cortesía de Turner Whitted, Bell Laboratories.)

**Ilustración 42.** Imágenes con traza de rayos (Sec. 14.7). (a) Escena del cortometraje *Quest* (1985). (Michael Sciuilli, James Arvo y Melissa White. © Hewlett-Packard.) (b) "Haute Air". Se usaron funciones para modificar el color, las normales a las superficies y la transparencia en casi todos los píxeles. (Cortesía de David Kurlander, Columbia University, 1986.)



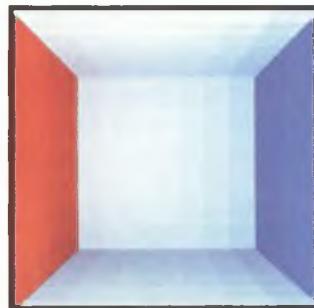
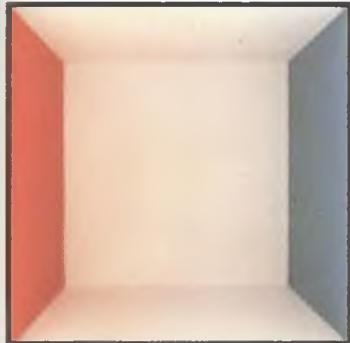
(a)



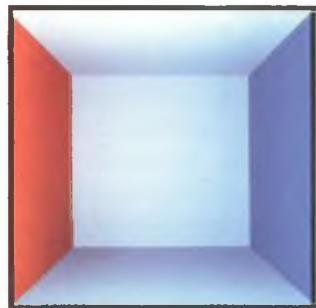
(b)

**Ilustración 43.** Radiosidad (Sec. 14.8.1). Cubo con seis paredes difusas (no se presenta la pared blanca frontal emisiva). (a) Fotografía del cubo real. (b) Modelo generado con 49 parches por lado, usando sombreado constante. (c) Modelo generado con 49 parches por lado, usando sombreado interpolado. (Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg y Bennett Battaille, Program of Computer Graphics, Cornell University, 1984.)

(a)



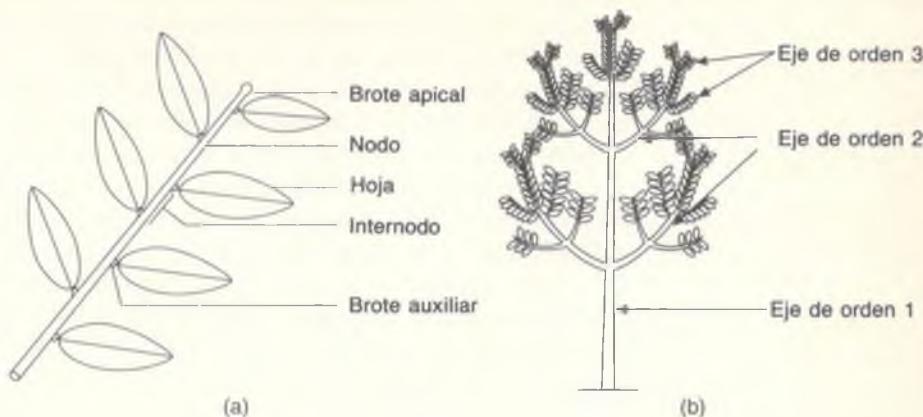
(b)



(c)

**Ilustración 44.** Oficina generada usando un algoritmo de radiosidad de hemicubo con refinamiento progresivo (Sec. 14.8.3); 500 parches, 7000 subparches. Se añade la radiosidad ambiental estimada. El cálculo y la presentación de cada iteración requirió unos 15 segundos en un computador HP 9000 modelo 825 SRX. (a) 1 iteración. (b) 2 iteraciones. (c) 24 iteraciones. (d) 100 iteraciones. (Shenchang Eric Chen, Michael F. Cohen, John R. Wallace y Donald P. Greenberg, Program of Computer Graphics, Cornell University, 1988.)





**Figura 9.34** Ejemplos del crecimiento de una planta. (a) El brote en la punta de un segmento de la planta se puede convertir en internodo; al hacerlo crea un nuevo brote (el **brote auxiliar**), un nuevo segmento (el **internodo**) y un nuevo brote en la punta (el **brote apical**). (b) Una planta más compleja, con órdenes asignados a los diversos internodos.

morir, puede florecer y morir, puede dormir por cierto tiempo o puede convertirse en **internodo**, un segmento de planta entre brotes. El proceso de conversión a un internodo tiene tres etapas: el brote original puede generar uno o más **brotes auxiliares** (brotes a un lado de la unión entre internodos), un proceso denominado **ramificación**; se añade el internodo; y el final del nuevo internodo se convierte en un **brote apical** (un brote en el extremo de una secuencia de internodos). En la figura 9.34(a) se presentan ejemplos de la transición de brote a internodo.

Cada uno de los brotes en el objeto resultante puede pasar por transiciones similares. Si decimos que el segmento inicial del árbol es de **orden 1**, podemos definir en forma inductiva el orden los demás internodos: los internodos generados del brote apical de un internodo de orden  $i$  también son de orden  $i$ ; los que se generan de los brotes auxiliares de un internodo de orden  $i$  son de orden  $(i + 1)$ . De esta manera, todo el tronco de un árbol es de orden 1, las ramas son de orden 2, los tallos en esas ramas son de orden 3, etcétera. En la figura 9.34(b) se presentan una planta más complicada y los órdenes de varios internodos de la planta.

Para convertir esta descripción en una imagen real de un árbol se requiere un modelo para las formas de sus diversos componentes: un internodo de orden 1 puede ser un cilindro ahusado de gran tamaño, y un internodo de orden 7 puede ser una pequeña línea verde. El único requisito es que debe haber una hoja en cada nodo auxiliar (aunque la hoja pueda desprenderse en un momento determinado).

Por último, para simular el crecimiento de la planta en este modelo se necesita la siguiente información biológica: la edad actual del modelo, la tasa de

crecimiento de cada orden de los internodos, el número de brotes auxiliares en el comienzo de cada internodo (como función del orden del internodo) y las probabilidades de muerte, pausa, ramificación y reiteración como funciones de la edad, la dimensión y el orden. También necesitamos cierta información geométrica: la forma de cada internodo (como función del orden y la edad), los ángulos de ramificación para cada orden y edad, y el tropismo de cada eje (si la secuencia de internodos de orden  $i$  es una línea recta o se curva hacia la horizontal o la vertical). Para dibujar una imagen de la planta requerimos más información: el color y la textura de cada una de las entidades que se dibujaran: internodos de diversos órdenes, hojas y flores de distintas edades. Con los modelos gramaticales se pueden producir modelos de árboles muy convincentes (véase la Ilust. en color 12).

## RESUMEN

En este capítulo sólo se tocaron los puntos más importantes relacionados con la representación de curvas y superficies, pero contiene información suficiente para que usted pueda implantar sistemas interactivos usando estas representaciones. Puede hallar el tratamiento teórico de este material en libros como [BART87; DEBO78; FAUX79; MORT85].

Las mallas poligonales, que son lineales por trozos, son apropiadas para la representación de objetos con caras planas, pero pocas veces son satisfactorias para objetos con caras curvas. Las curvas cúbicas y las superficies bicubicas paramétricas continuas por trozos son de uso común en la graficación por computador y en CAD para representar objetos con caras curvas, por lo siguiente:

- Permiten valores múltiples para un solo valor de  $x$  o  $y$ .
- Representan pendientes infinitas.
- Permiten el control local, de manera que al cambiar un punto de control afecte únicamente una porción local de la curva.
- Pueden establecerse de manera que interpolen o aproximen puntos de control, dependiendo de los requisitos de la aplicación.
- Son computacionalmente eficientes.
- Se transforman fácilmente por medio de transformaciones de los puntos de control.

Aunque sólo hemos visto las superficies cúbicas, se pueden emplear otras de orden mayor o menor. Los textos que mencionamos antes generalmente desarrollan las curvas y superficies paramétricas para el caso general de orden  $n$ .

También analizamos en forma breve algunas de las técnicas para modelar fenómenos naturales en particular los métodos fractales y gramaticales.

**Ejercicios**

9.1 Encuentre la matriz de geometría y la matriz base para la representación paramétrica de una línea recta proporcionada en la ecuación (9.11).

9.2 Demuestre que, para una curva bidimensional  $[x(t) \ y(t)]^T$ , la continuidad  $G^1$  significa que la pendiente geométrica  $dy/dx$  es igual en los puntos de unión entre segmentos.

9.3 Sea  $\gamma(t) = (t, t^2)$  para  $0 \leq t \leq 1$  y sea  $\eta(t) = (2t + 1, t^3 + 4t + 1)$  para  $0 \leq t \leq 1$ . Observe que  $\gamma(1) = (1, 1) = \eta(0)$ , de manera que  $\gamma$  y  $\eta$  se unen con continuidad  $C^0$ .

- Grafique  $\eta(t)$  y  $\gamma(t)$  para  $0 \leq t \leq 1$ .
- Determine si  $\eta(t)$  y  $\gamma(t)$  cumplen con la continuidad  $C^1$  en el punto de unión. (Usted tendrá que calcular los vectores  $d\frac{\gamma}{dt}(1)$  y  $d\frac{\eta}{dt}(0)$  para verificar su respuesta.)
- Determine si  $\eta(t)$  y  $\gamma(t)$  cumplen con la continuidad  $G^1$  en el punto de unión. (Tendrá que revisar las razones de la parte (b) para verificar su respuesta.)

9.4 Considere las trayectorias

$$\gamma(t) = (t^2 - 2t + 1, t^3 - 2t^2 + t) \text{ y } \eta(t) = (t^2 + 1, t^3),$$

definidas en el intervalo  $0 \leq t \leq 1$ . Las curvas se unen, ya que  $\gamma(1) = \eta(1, 0) = (0)$ . Demuestre que se unen con continuidad  $C^1$  pero no con continuidad  $G^1$ . Grafique ambas curvas como funciones de  $t$  para demostrar exactamente el porqué de este comportamiento.

9.5 Muestre que las dos curvas  $\gamma(t) = (t^2 - 2t, t)$  y  $\eta(t) = (t^2 + 1, t + 1)$  tienen continuidad  $C^1$  y  $G^1$  cuando se unen en  $\gamma(1) = \eta(0)$ .

9.6 Analice el efecto de un *B-spline* con cuatro puntos de control colineales en secuencia.

9.7 Escriba un programa para aceptar una matriz de geometría, una matriz base y una lista de control arbitrarias y dibuje la curva correspondiente.

9.8 Encuentre las condiciones en las cuales dos curvas hermitianas unidas tienen continuidad  $C^1$ .

9.9 Suponga que las ecuaciones que relacionan la geometría de Hermite con la de Bézier tienen la forma  $R_1 = \beta(P_2 - P_1)$ ,  $R_4 = \beta(P_4 - P_3)$ . Considere los cuatro puntos de control de Bézier con espacio uniforme  $P_1 = (0, 0)$ ,  $P_2 = (1, 0)$ ,  $P_3 = (2, 0)$  y  $P_4 = (3, 0)$ . Muestre que, para que la curva paramétrica  $Q(t)$  tenga velocidad constante de  $P_1$  a  $P_4$ , el coeficiente  $\beta$  debe ser igual a 3.

9.10 Explique por qué la ecuación (9.35) para las *B-splines* uniformes se escribe como  $Q_i(t - t_i)$ , mientras que la ecuación (9.37) para las *B-splines* no uniformes se escribe como  $Q_i(t)$ .

9.11 Dada una *B-spline* no uniforme bidimensional y un valor  $(x, y)$  en la curva, escriba un programa para hallar el valor correspondiente de  $t$ . Asegúrese de considerar la posibilidad de que, para un valor de  $x$  (o  $y$ ), existan varios valores de  $y$  (o  $x$ ).

9.12 Aplique la metodología utilizada para obtener las ecuaciones (9.49) y (9.50) de superficies hermitianas para obtener la ecuación (9.51) de superficies de Bézier.

9.13 Sea  $t_0 = 0, t_1 = 1, t_2 = 3, t_3 = 4, t_4 = 5$ . Use estos valores para calcular  $B_{0,4}$  y cada una de las funciones utilizadas en su definición. Después grafique las funciones para el intervalo  $-3 \leq t \leq 8$ .

9.14 Desarrolle un programa, similar al ejemplo 9.2, para dibujar parches de superficie de Bézier usando el esquema del programa 9.2. Su programa debe ofrecer la opción de dibujar los puntos de control para un parche especificado de manera que el usuario pueda seleccionar cualquiera de ellos (empleando un localizador) y moverlo a una nueva ubicación. El parche debe redibujarse entonces para reflejar la nueva restricción geométrica. Puesto que la entrada de localizador es bidimensional, ¿cómo la relacionaría con un punto de control tridimensional? Puede especificar su propia geometría de parches o utilizar los existentes, como los de la tetera de la figura 9.1. Los datos completos para la tetera se incluyen en [CROW87].

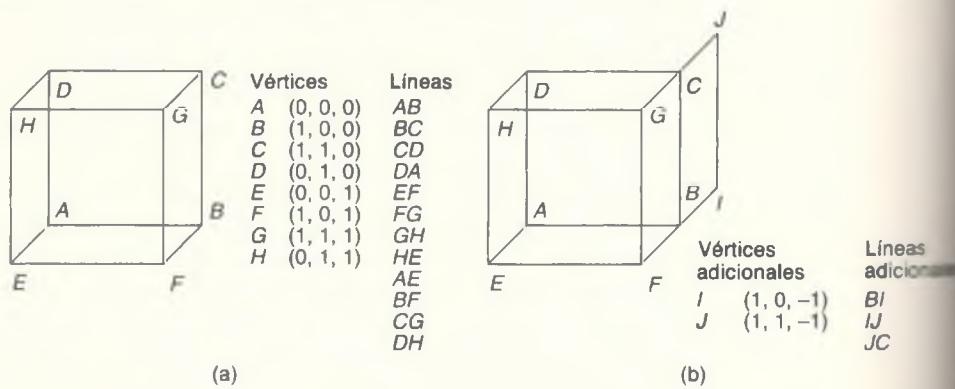
Las representaciones analizadas en el capítulo 9 nos permiten describir curvas y superficies en dos y tres dimensiones. Así como un conjunto de líneas y curvas bidimensionales no necesariamente tiene que describir la frontera de un área cerrada, una colección de planos y superficies tridimensionales no tiene que encerrar un volumen. Sin embargo, en muchas aplicaciones es importante distinguir entre las partes interna, externa y superficial de un objeto tridimensional y poder calcular propiedades del objeto que dependan de esta distinción. Por ejemplo, en CAD/CAM, si un objeto sólido se puede modelar en una forma que capture adecuadamente su geometría, es posible llevar a cabo varias operaciones útiles antes de fabricar el objeto. Nos gustaría determinar si dos objetos interfieren entre sí, por ejemplo si el brazo de un robot golpeará objetos en su ambiente o si una herramienta de corte sólo cortará el material que debe eliminar. Al simular mecanismos físicos, como una caja de velocidades, puede ser importante calcular propiedades como el volumen y el centro de masa. A los modelos de sólidos se les aplica el análisis de elementos finitos para determinar la respuesta a factores como la tensión y la temperatura. La representación satisfactoria de un objeto sólido permite incluso generar instrucciones automáticamente para máquinas herramienta controladas por computador que produzcan el objeto o crear rápidamente prototipos usando una técnica como la estereolitografía, un proceso en el cual se emplea un rayo láser para formar un objeto duro a partir de un baño de plástico fundido. Además, algunas técnicas gráficas, como el modelado de la transparencia con refracción, dependen de la posibilidad de determinar por dónde entra y sale la luz en un objeto sólido. Estas aplicaciones son ejemplos del **modelado de sólidos**. La necesidad de modelar

objetos como sólidos ha dado lugar al desarrollo de diversas técnicas especializadas para representarlos. En este capítulo se presenta una breve introducción a estas representaciones.

## 10.1 Representación de sólidos

La capacidad de una representación para codificar cosas que *parecen* sólidos no quiere decir que la representación sea adecuada para representar sólidos. Considere la forma en que hasta ahora hemos representado objetos, como colecciones de líneas rectas, curvas, polígonos y superficies. Las líneas de la figura 10.1(a) ¿definen un cubo sólido? Si suponemos que cada conjunto de cuatro líneas a los lados del objeto acota una cara cuadrada, la figura es un cubo. Sin embargo, no hay nada en la representación que requiera dicha interpretación de las líneas. Por ejemplo, se usaría el mismo conjunto de líneas para dibujar la figura aunque faltaran caras. ¿Qué pasa si decidimos que cada ciclo plano de líneas conectadas en el dibujo determina, por definición, una cara poligonal? En este caso, la figura 10.1(b) consistiría en todas las caras de la figura 10.1(a) más una cara “colgante” adicional, produciendo un objeto que no acota un volumen. Como veremos en la sección 10.5, se requieren algunas restricciones adicionales para garantizar que una representación de este tipo únicamente modele sólidos.

Requicha [REQU80] presenta una lista de las propiedades deseables en un esquema de representación de sólidos. El **dominio** de la representación debe tener el tamaño suficiente para permitir la representación de un conjunto útil de objetos físicos. De manera ideal, la representación debe ser *no ambigua*: no deben existir dudas acerca de qué es lo que se representa, y una representación debe corresponder a un sólido y sólo uno, a diferencia de lo que aparece en la



**Figura 10.1** (a) Cubo alambrado compuesto por 12 líneas rectas. (b) Cubo alambrado con una cara adicional.

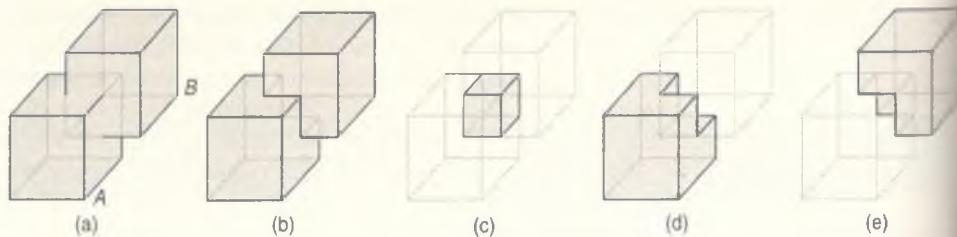
figura 10.1(a). También se dice que una representación no ambigua es **completa**. Una representación es **única** si se puede usar para codificar un sólido determinado en solamente una forma. Si una representación puede asegurar la unicidad, se facilitan operaciones como la prueba de igualdad de dos objetos. Una representación *precisa* permite representar un objeto sin aproximaciones. Así como un sistema gráfico que sólo puede dibujar líneas rectas nos obliga a crear aproximaciones de curvas suaves, algunas representaciones de modelado de sólidos describen muchos objetos como aproximaciones. En teoría, un esquema de representación debe hacer *imposible la creación de una representación inválida* (es decir, una que no corresponda a un sólido), como la figura 10.1(b). Además, debe ser *fácil crear una representación válida*, generalmente con ayuda de un sistema interactivo de modelado de sólidos. Nos gustaría que los objetos mantuvieran la **cerradura** bajo rotación, traslación y otras operaciones. Así, al realizar estas operaciones con sólidos válidos sólo deben obtenerse sólidos válidos. Una representación debe ser *compacta* para ahorrar espacio, lo que a su vez puede ahorrar tiempo de comunicación en un sistema distribuido. Por último, una representación debe permitir la utilización de algoritmos *eficientes* para calcular las propiedades físicas deseadas y, lo más importante para nosotros, para crear imágenes.

El diseño de una representación con todas estas propiedades es algo difícil, y muchas veces hay que conciliar. Conforme analicemos la principales representaciones que se usan en la actualidad, nos concentraremos en proporcionar detalles suficientes para poder comprender cómo se pueden establecer interfaces entre estas representaciones y el software gráfico. Si desea conocer más detalles, con hincapié en los aspectos del modelado de sólidos, puede consultar [REQU80; MORT85; MÄNT88].

## 10.2 Operaciones regularizadas de conjuntos booleanos

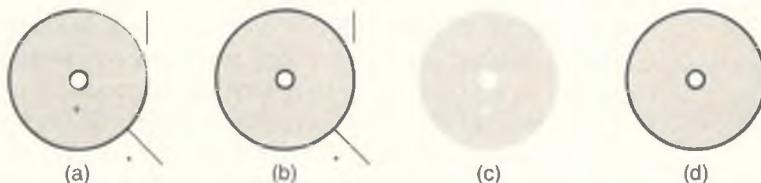
Independientemente de cómo representemos los objetos, nos gustaría poder combinarlos para formar nuevos. Uno de los métodos más intuitivos y comunes para combinar objetos son las **operaciones booleanas de conjuntos**, como la unión, la diferencia y la intersección, ilustradas en la figura 10.2. Estas operaciones son los equivalentes tridimensionales de las operaciones booleanas bidimensionales familiares. Sin embargo, la aplicación de una operación booleana ordinaria de conjuntos a dos objetos sólidos no necesariamente produce un objeto sólido. Por ejemplo, la intersección ordinaria de dos cubos que se unen en un solo vértice es un punto.

En lugar de emplear los operadores booleanos ordinarios de conjuntos, usaremos los **operadores booleanos regularizados de conjuntos** [REQU77], denominados como  $\cup^*$ ,  $\cap^*$  y  $-^*$ , y definidos de manera que las operaciones con sólidos siempre generen sólidos. Por ejemplo, la intersección booleana regularizada de dos cubos que se unen en un solo vértice es el objeto nulo.



**Figura 10.2** Operaciones booleanas. (a) Objetos **A** y **B**, (b)  $A \cup B$ , (c)  $A \cap B$ , (d)  $A - B$  y (e)  $B - A$ .

Para examinar la diferencia entre los operadores ordinarios y los regulares, podemos considerar cualquier objeto como definido por un conjunto de puntos y dividido en puntos interiores y puntos de frontera, como se ilustra en la figura 10.3(a). Los **puntos de frontera** son aquellos cuya distancia al objeto o al complemento del objeto es cero. Los puntos de frontera no tienen que formar parte del objeto. Un **conjunto cerrado** contiene todos sus puntos de frontera, mientras que un **conjunto abierto** no contiene ninguno. La unión de un conjunto con el conjunto de sus puntos de frontera se conoce como **cerradura** del conjunto, como se muestra en la figura 10.3(b), que es en sí un conjunto cerrado. La **frontera** de un conjunto cerrado es el conjunto de sus puntos de frontera, mientras que el **interior**, presentado en la figura 10.3(c), consiste en todos los demás puntos del conjunto y por ende es el complemento de la frontera respecto al objeto. La **regularización** de un conjunto se define como la cerradura de los puntos interiores del conjunto. En la figura 10.3(d) se muestra la cerradura del objeto de la figura 10.3(c) y, por lo tanto, la regularización del objeto de la figura 10.3(a). Un conjunto que es igual a su propia regularización se conoce como **conjunto regular**. Observe que un conjunto regular no puede contener ningún punto de frontera que no sea adyacente a un punto interior; es decir,



**Figura 10.3** Regularización de un objeto. (a) El objeto se define con puntos interiores, dibujados en color gris claro, y puntos de frontera. Los puntos de frontera que forman parte del objeto se presentan en negro; los demás en gris oscuro. El objeto tiene puntos y líneas colgantes desconectados y hay un punto de frontera en el interior que no forma parte del objeto. (b) Cerradura del objeto. Todos los puntos de frontera forman parte del objeto. El punto de frontera incorporado en el interior de la parte (a) es ahora parte del interior. (c) Interior del objeto. Se han eliminado los puntos y líneas colgantes y desconectados. (d) La regularización del objeto es la cerradura de su interior.

no puede tener puntos, líneas o superficies de frontera “colgantes”. Podemos definir cada operador booleano regularizado de conjuntos en función de un operador ordinario, como

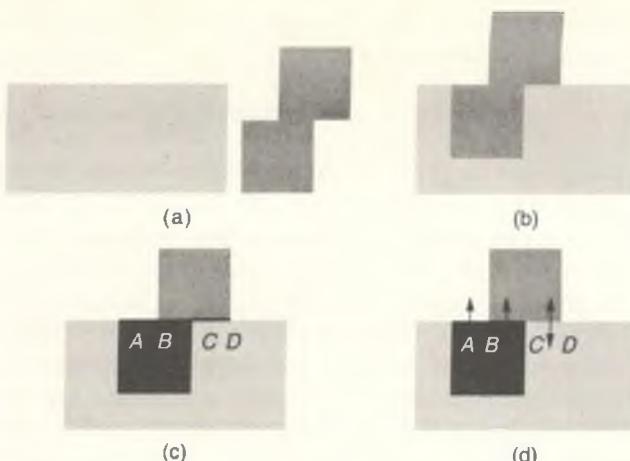
$$A \text{ op}^* B = \text{cerradura}(\text{interior}(A \text{ op } B)), \quad (10.1)$$

donde  $\text{op}$  es  $\cup$ ,  $\cap$ , o  $-$ . Los operadores booleanos regularizados de conjuntos producen sólo conjuntos regulares cuando se aplican a conjuntos regulares.

Comparemos ahora las operaciones booleanas ordinarias y regularizadas de conjuntos aplicadas a conjuntos regulares. Considere los dos objetos de la figura 10.4(a), ubicados como se muestra en la figura 10.4(b). La intersección booleana ordinaria de los dos objetos contiene la intersección del interior y la frontera de cada objeto con el interior y la frontera del otro, como se muestra en la figura 10.4(c). En cambio, la intersección booleana regularizada de los dos objetos, presentada en la figura 10.4(d), contiene la intersección de sus interiores y la intersección del interior de cada una con la frontera de la otra, pero sólo un subconjunto de la intersección de sus fronteras. El criterio que se emplea para definir este subconjunto determina la forma en que la intersección booleana regularizada difiere de la ordinaria, en la cual se incluyen todas las partes de la intersección de las fronteras.

Por intuición, una porción de la intersección frontera-frontera será incluida en la intersección booleana regularizada si y sólo si los interiores de ambos objetos se encuentran en el mismo lado de esta porción de frontera compartida. Como los puntos interiores de los dos objetos que son directamente adyacentes a la porción de la frontera están en la intersección, también hay que conservar la porción de la frontera para mantener la cerradura. Considere el caso de una porción de frontera compartida que se encuentre en caras coplanares de dos poliedros. Es fácil determinar si los interiores se hallan en el mismo lado de la frontera compartida si los dos objetos se definen de manera que las normales a sus superficies apunten hacia afuera (o hacia adentro). Los interiores estarán del mismo lado si sus normales apuntan en la misma dirección. Por lo tanto, se incluye el segmento  $AB$  en la figura 10.4(d). Recuerde que siempre se incluyen aquellas partes de la frontera de un objeto que intersecan el interior del otro objeto, como el segmento  $BC$ .

Considere lo que sucede si los interiores de los objetos se encuentran en lados opuestos de la frontera compartida, como ocurre con el segmento  $CD$ . En estos casos no se incluye en la intersección ninguno de los puntos interiores adyacentes a la frontera. Por consiguiente, la porción de la frontera compartida no es adyacente a ninguno de los puntos interiores del objeto resultante y, por ende, no se incluye en la intersección regularizada. Esta restricción adicional con respecto a cuáles son las porciones incluidas de la frontera compartida garantiza que el objeto resultante sea un conjunto regular. La normal a la superficie de cada cara de la frontera del objeto resultante es la normal a la superficie que contribuyó dicha parte de la frontera. (Como veremos en el capítulo 14, las normales a la superficie son importantes para sombrear objetos.) Una vez que se ha determinado cuáles caras se encuentran en la frontera, se incluye una aris-



**Figura 10.4** Intersección booleana. (a) Dos objetos, mostrados en sección transversal. (b) Posiciones de los objetos antes de la intersección. (c) Intersección booleana ordinaria que produce una cara colgante, mostrada como la línea  $CD$  en la sección transversal. (d) Intersección booleana regularizada que incluye una porción de la frontera compartida en la frontera resultante si los dos objetos yacen en el mismo lado ( $AB$ ), y la excluye si los objetos yacen en lados opuestos ( $CD$ ). Las intersecciones frontera-interior ( $BC$ ) siempre se incluyen.

ta o un vértice de la intersección frontera-frontera en la frontera de la intersección si la arista o el vértice es adyacente a una de dichas caras.

Los resultados de cada operador regularizado se pueden definir en función de los operadores ordinarios aplicados a las fronteras y a los interiores de los objetos. En la tabla 10.1 se indica la manera en que se definen los operadores regularizados para objetos  $A$  y  $B$  cualesquiera; en la figura 10.5 se presenta el resultado de efectuar las operaciones.  $A_b$  y  $A_i$  son la frontera y el interior de  $A$ , respectivamente.  $A_b \cap B_b$  iguales es la parte de la frontera compartida por  $A$  y  $B$  donde  $A$  y  $B$  están en el mismo lado. Este resultado es el caso de un punto  $i$  en la frontera compartida si al menos un punto  $i$  adyacente a él pertenece a  $A$  o a  $B$ .  $A_b \cap B_b$  diferentes es la parte de la frontera compartida por  $A$  y  $B$  para el cual  $A_i$  y  $B_i$  caen en lados opuestos. Esto se aplica a  $b$  si no es adyacente a ningún punto  $i$ . Cada uno de los operadores regularizados está definido por la unión de los conjuntos asociados con las filas que tienen un punto (\*) en la columna del operador.

Observe que, en todos los casos, cada porción de la frontera del objeto resultante se encuentra en la frontera de uno o de los dos objetos originales. Para calcular  $A \cup^* B$  o  $A \cap^* B$ , la normal a la superficie de una cara del resultado se hereda de la normal a la superficie de la cara correspondiente de uno o ambos objetos originales. Sin embargo, en el caso de  $A -^* B$ , la normal a la superficie de cada cara del resultado en donde se ha usado  $B$  para excavar  $A$ , debe apuntar en la dirección opuesta a la normal a la superficie de  $B$  en dicha cara. Esto corresponde a las porciones de frontera  $A_b \cap B_b$  diferentes y  $B_b \cap A_i$ . Alternativamente,

Tabla 10.1

## Operaciones booleanas regularizadas de conjuntos

Conjunto	$A \cup^* B$	$A \cap^* B$	$A -^* B$
$A_i \cap B_i$	•	•	•
$A_i - B$	•	•	•
$B_i - A$	•	•	•
$A_b \cap B_i$	•	•	•
$B_b \cap A_i$	•	•	•
$A_b - B$	•	•	•
$B_b - A$	•	•	•
$A_b \cap B_b$ iguales	•	•	•
$A_b \cap B_b$ diferentes	•	•	•

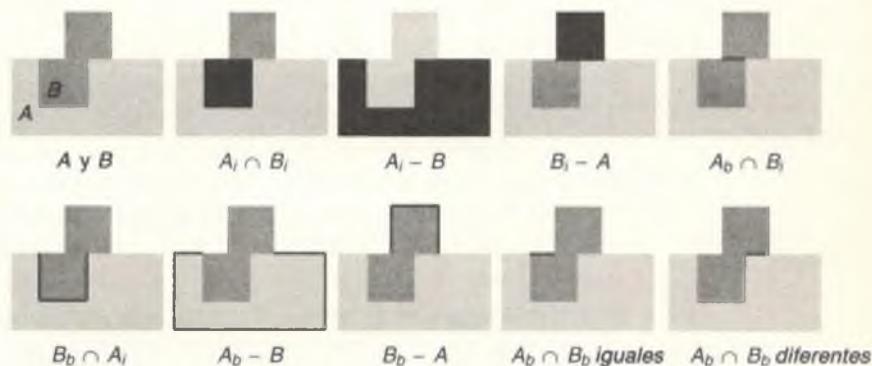


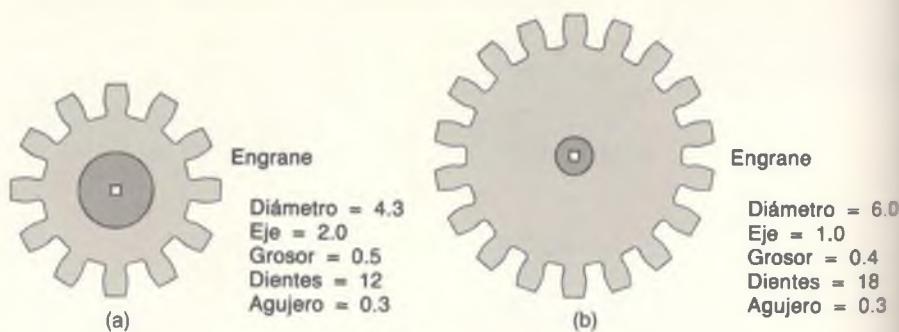
Figura 10.5 Operaciones booleanas ordinarias aplicadas a subconjuntos de dos objetos.

$A -^* B$  se puede reescribir como  $A \cap^* \bar{B}$ . Podemos obtener  $\bar{B}$  (el complemento de  $B$ ) complementando el interior de las  $B$  e invirtiendo las normales a su frontera.

En la mayoría de los métodos que analizaremos, los operadores booleanos regularizados de conjuntos se han utilizado como técnica de interfaz con el usuario para construir objetos complejos a partir de objetos sencillos. Estos operadores también se incluyen explícitamente en uno de los métodos, la geometría sólida constructiva. En las secciones que siguen describiremos diversas maneras de representar objetos sólidos sin ambigüedades.

## 10.3 Generación de ejemplares de primitivas

En la generación de ejemplos de primitivas, el sistema de modelado define un conjunto de formas sólidas primitivas tridimensionales que son relevantes para



**Figura 10.6** Dos engranes definidos por la generación de ejemplares de primitivas.

el área de aplicación. Estas primitivas suelen parametrizarse no sólo en función de las transformaciones del capítulo 7, sino también con base en otras propiedades. Por ejemplo, un objeto primitivo puede ser una pirámide regular con un número (definido por el usuario) de caras que se unen en el ápice. Los ejemplares de primitivas son similares a objetos parametrizados, como los menús del capítulo 2, excepto que los objetos son sólidos. Una primitiva parametrizada se puede considerar como la definición de una familia de partes cuyos miembros varían en unos cuantos parámetros, un importante concepto de CAD conocido como **tecnología de grupos**. La generación de ejemplares de primitivas se usa con frecuencia en objetos de complejidad relativa, como engranes o tuercas, que es tedioso definir en función de combinaciones booleanas de objetos más sencillos pero que se pueden caracterizar fácilmente con unos cuantos parámetros de alto nivel. Por ejemplo, un engrane se puede parametrizar por su diámetro o su número de dientes, como se ilustra en la figura 10.6.

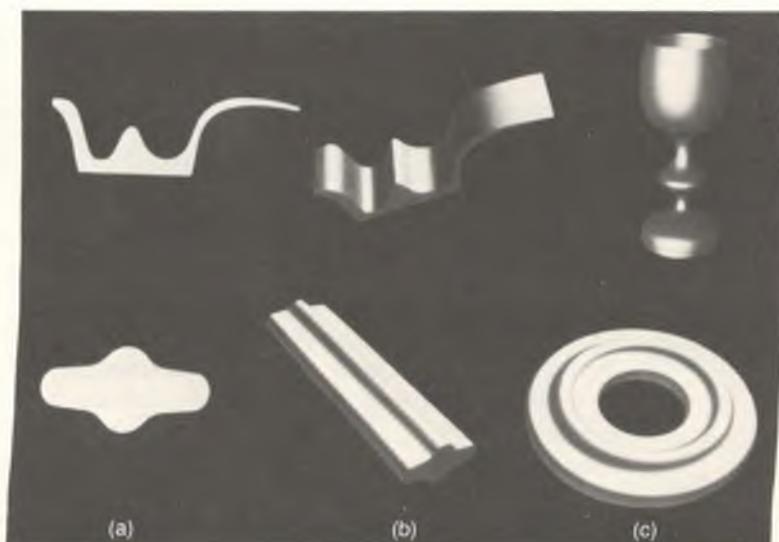
Aunque podemos construir una jerarquía de ejemplares de primitivas, cada ejemplar de nodo hoja sigue siendo un objeto definido por separado. En la generación de ejemplares de primitivas no hay medios para combinar objetos para crear uno nuevo de mayor nivel usando, por ejemplo, las operaciones booleanas regularizadas de conjuntos. Por lo tanto, la única forma de crear un nuevo tipo de objeto es escribiendo el código que lo define. Así mismo, es necesario escribir individualmente, para cada primitiva, las rutinas que dibujan el objeto y determinan sus propiedades de masa.

## 10.4 Representaciones de barrido

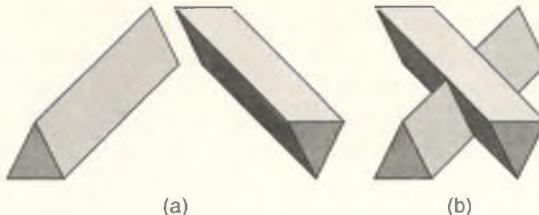
Al barrer un objeto a lo largo de una trayectoria por el espacio se define un objeto nuevo, llamado **barrido**. El tipo de barrido más sencillo es el definido

por un área bidimensional barrida por una trayectoria lineal normal al plano del área, para crear un volumen. Este proceso se conoce como **barrido traslacional** o **extrusión**, y es una forma natural de representar objetos formados por la extrusión de metal o plástico a través de un molde con la sección transversal deseada. En los casos más sencillos, cada volumen de barrido no es más que el área del objeto que se barre multiplicada por la longitud del barrido. Las extensiones sencillas comprenden el escalamiento de la sección transversal durante el barrido para producir un objeto ahulado o barrer la sección transversal a lo largo de una trayectoria lineal que no es la normal. Los **barridos rotacionales** se definen mediante la rotación de un área con respecto a un eje. En la figura 10.7 se presentan dos objetos y los barridos traslacionales y rotacionales simples que se generan con ellos.

El objeto que se barre no tiene que ser bidimensional. Los barridos de sólidos son útiles para modelar la región barrida por la cabeza de corte de una máquina herramienta o un robot que sigue una trayectoria. Los barridos donde el área o el volumen que generan cambia de tamaño, forma u orientación y que siguen una trayectoria curva arbitraria se denominan **barridos generales**. Los barridos generales de secciones transversales bidimensionales se conocen como **cilindros generalizados** en visión computadorizada [BINF71], y generalmente se modelan como secciones transversales bidimensionales parametrizadas barridas en ángulos rectos sobre una curva arbitraria. Los barridos generales son muy difíciles de modelar en forma eficiente. Por ejemplo, la trayectoria y la forma



**Figura 10.7** Barridos de (a) áreas bidimensionales que se usan para definir, (b) barridos traslacionales y (c) barridos rotacionales. (Creados con el sistema de modelado Alpha\_1. Cortesía de University of Utah.)



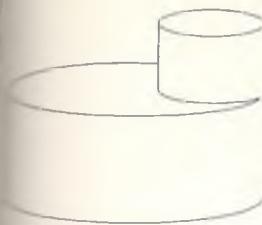
**Figura 10.8** (a) Dos barridos simples de objetos bidimensionales (triángulos). (b) La unión de los barridos presentados en (a) no es un barrido simple de un objeto bidimensional.

del objeto que producen pueden ocasionar que el objeto se interseque, complicando los cálculos de volumen. Además, los barridos generales no siempre generan sólidos. Por ejemplo, el barrido de un área bidimensional en su propio plano genera otra área bidimensional.

En términos generales, es difícil aplicar las operaciones regularizadas de conjuntos booleanos a los barridos sin antes convertirlos a otra representación. Incluso los barridos más simples no son cerrados con las operaciones booleanas regularizadas de conjuntos. Por ejemplo, la unión de dos barridos simples generalmente no es un barrido simple, como se ilustra en la figura 10.8. Sin embargo, a pesar de los problemas de cerradura y de cálculo, los barridos son una forma natural e intuitiva de construir diversos objetos. Por ello, muchos sistemas de modelado de sólidos permiten a los usuarios construir objetos como barridos, pero los almacenan con alguna de las representaciones que analizaremos.

## 10.5 Representaciones de fronteras

Las **representaciones de fronteras** (también conocidas como **b-rep**) son similares a las sencillas representaciones que analizamos en la sección 10.1, ya que describen un objeto en función de sus fronteras superficiales: vértices, aristas y caras. Algunas representaciones de fronteras están limitadas a fronteras poligonales planas, y pueden requerir que las caras sean triángulos o polígonos convexos. La determinación de lo que constituye una cara puede ser difícil si se permite superficies curvas, como en la figura 10.9. Las caras curvas muchas veces se aproximan con polígonos. Alternativamente, las superficies curvas también pueden representarse como parches de superficie si los algoritmos que procesan la representación pueden tratar las curvas de intersección resultantes, las cuales, por lo general, serán de orden mayor que las superficies originales. Las representaciones de fronteras se obtienen de las sencillas representaciones vectoriales que se usaron en capítulos anteriores y se emplean en muchos sistemas de modelado actuales. Como son comunes en la graficación, se han desarrollado

**Figura 10.9**

¿Cuántas caras tiene este objeto?

rias técnicas eficientes para crear imágenes de objetos poligonales con sombreado suave; varias de estas técnicas se analizan en el capítulo 14.

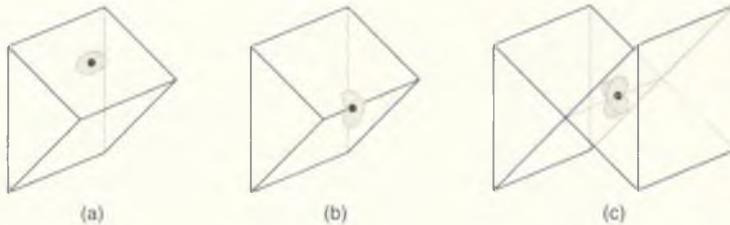
Muchos sistemas de representación de fronteras sólo permiten usar sólidos cuyas fronteras sean **2-variedades**. Por definición, cada punto en una 2-variedad tiene una vecindad, arbitrariamente pequeña, de puntos que pueden considerarse topológicamente iguales a un disco en el plano. Esto quiere decir que hay una correspondencia continua uno a uno entre la vecindad de los puntos y el disco, como se ilustra en la figura 10.10(a) y (b). Por ejemplo, si más de dos caras comparten una arista, como en la figura 10.10(c), cualquier vecindad de un punto en la arista contiene puntos de cada una de esas caras. Por intuición, es obvio que no hay una correspondencia uno a uno continua entre esta vecindad y un disco en el plano, aunque la demostración matemática no es trivial. Por lo tanto, la superficie en la figura 10.10(c) no es una 2-variedad. Aunque algunos sistemas actuales no tienen esta restricción, limitaremos nuestro análisis de las representaciones de fronteras a las 2-variedades, excepto si se indica lo contrario.

### 10.5.1 Poliedros y fórmula de Euler

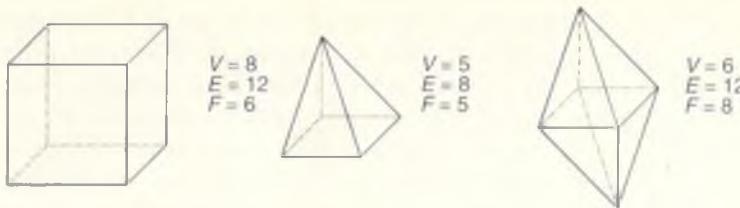
Un **poliedro** es un sólido acotado por un conjunto de polígonos cuyas aristas pertenecen a un número par de polígonos (exactamente dos, en el caso de las 2-variedades) y que satisface algunas restricciones adicionales (que veremos más adelante). Un **poliedro simple** es aquel que se puede deformar para obtener una esfera, es decir, un poliedro que, a diferencia de un toro, no tiene agujeros. La representación de frontera de un poliedro simple satisface la fórmula de Euler, que expresa una relación invariable entre el número de vértices, aristas y caras de un poliedro simple:

$$V - E + F = 2 \quad (10.2)$$

donde  $V$  es el número de vértices,  $E$  es el número de aristas y  $F$  es el número de caras. En la figura 10.11 se presentan algunos poliedros simples y su número

**Figura 10.10**

En una 2-variedad cada punto tiene una vecindad de puntos que lo rodean y forman un disco topológico, mostrado en gris en (a) y (b). (c) Si un objeto no es una 2-variedad, entonces no tiene puntos cuya vecindad sea un disco topológico.



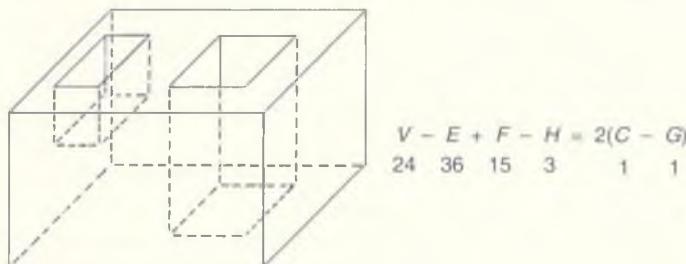
**Figura 10.11** Algunos poliedros simples con sus valores de vértices ( $V$ ), aristas ( $E$ ) y caras ( $F$ ). En todos los casos,  $V - E + F = 2$ .

de vértices, aristas y caras. Observe que la fórmula aún es aplicable si se permiten aristas curvas y caras no planas. Por sí sola, la fórmula de Euler establece condiciones necesarias, aunque no suficientes, para que un objeto sea un poliedro simple. Se pueden construir objetos que satisfagan la fórmula pero no acierten un volumen, añadiendo una o más caras o aristas colgantes a un sólido que de otra manera sería válido, como en la figura 10.1(b). Se requieren otras restricciones para garantizar que el objeto sea un sólido: cada arista debe conectar dos vértices y estar compartida por exactamente dos caras, al menos tres aristas deben unirse en cada vértice y las caras no deben ser interpenetrantes.

Una generalización de la fórmula de Euler se aplica a 2-variedades que tienen caras con agujeros:

$$V - E + F - H = 2(C - G), \quad (10.3)$$

donde  $H$  es el número de agujeros en las caras,  $G$  es el número de agujeros que atraviesan el objeto y  $C$  es el número de componentes separados (partes) del objeto, como se muestra en la figura 10.12. Si un objeto tiene un solo componente, su  $G$  se conoce como **género**; si tiene varios componentes, entonces su  $G$  es la suma de los géneros de sus componentes. Como antes, se necesitan restricciones adicionales para garantizar que los objetos sean sólidos.



**Figura 10.12** Poliedro clasificado de acuerdo con la ecuación (10.3), con dos agujeros en la cara superior y uno en la cara inferior.

Baumgart introdujo el concepto de **operadores de Euler**, los cuales operan con objetos que satisfacen la fórmula de Euler y transforman los objetos para formar otros que también obedecen a la fórmula, añadiendo vértices, aristas y caras [BAUM74]. Braid, Hillyard y Stroud [BRAI78] mostraron cómo se puede combinar un pequeño número de operadores de Euler para transformar objetos, siempre y cuando no se requiera que los objetos intermedios sean sólidos válidos, mientras que Mäntylä [MÄNT88] demuestra que todas las representaciones de fronteras válidas se pueden construir con una secuencia finita de operadores de Euler. Se pueden definir también otros operadores que realizan *ajustes finos* a un objeto, moviendo los vértices, aristas o caras existentes sin afectar su número.

Quizás la representación de frontera más simple sea una lista de caras poligonales, cada una representada por una lista de coordenadas de vértices. Para representar la dirección hacia la cual queda la cara, los vértices del polígono se listan en un orden acorde al sentido del giro de las manecillas del reloj, visto desde el exterior del sólido. Para evitar la duplicación de coordenadas compartidas por las caras, cada vértice se puede representar con un índice en una lista de coordenadas. En esta representación, las aristas se representan implícitamente con pares de vértices adyacentes en las listas de vértices de los polígonos. Las aristas también se pueden representar en forma explícita como pares de vértices, definiendo cada cara como una lista de índices en la lista de vértices. Estas representaciones se analizan con mayor detalle en la sección 9.1.1.

### 10.5.2 Operaciones de conjuntos booleanos

Las representaciones de fronteras se pueden combinar usando los operadores regularizados de conjuntos booleanos, para crear así nuevas representaciones de fronteras [REQU85]. Sarraga [SARR83] y Miller [MILL87] analizan algoritmos para determinar las intersecciones entre superficies cuádricas. Los algoritmos para combinar objetos poliédricos se presentan en [TURN84; REQU85; PUTN86; LAID86], mientras que Thibault y Naylor [THIB87] describen un método basado en la representación de sólidos con un árbol binario para partición de espacio, que se analiza en la sección 10.6.4.

Un método [LAID86] consiste en inspeccionar los polígonos de ambos objetos, dividiéndolos de ser necesario para asegurar que la intersección de un vértice, arista o cara de un objeto con un vértice, arista o cara del otro sea un vértice, arista o cara de ambos. Despues se clasifican los polígonos de cada objeto con respecto al otro para determinar si están dentro, fuera o en la frontera. Si nos remitimos a la tabla 10.1, observamos que, por tratarse de una representación de frontera, sólo nos interesan las seis últimas filas, que representan una parte de una o ambas fronteras de los objetos originales,  $A_b$  y  $B_b$ . Despues de la división, cada polígono de un objeto está totalmente dentro ( $A_b \cap B_b$  o  $B_b \cap A_b$ ) o fuera ( $A_b - B$  o  $B_b - A$ ) del otro, o bien forma parte de la frontera compartida ( $A_b \cap B_b$  iguales o  $A_b \cap B_b$  diferentes).

Podemos clasificar un polígono usando la técnica de traza de rayos descrita en la sección 13.4.1. Por ahora, construimos un vector en la dirección de la

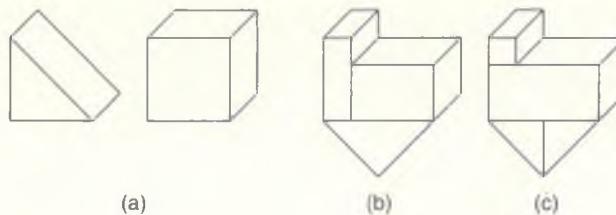
normal a la superficie del polígono, desde un punto en el interior del polígono, para luego hallar el polígono más cercano que interseca el vector en el otro objeto. Si no se interseca ningún polígono, el polígono original se encuentra fuera del otro objeto. Si el polígono intersecante más cercano es coplanar con el original, esta intersección es de tipo frontera-frontera y la comparación de las normales de los polígonos indica el tipo de intersección ( $A_b \cap B_b$  iguales o  $A_b \cap B_b$  diferentes). En caso contrario, se revisa el producto punto de las normales de los dos polígonos. Un producto punto positivo indica que el polígono original se halla dentro del otro objeto; si el resultado es negativo, el polígono original está fuera. El producto punto igual a cero puede ocurrir si el vector se encuentra en el plano del polígono intersecado; en este caso, el vector se perturba ligeramente e interseca de nuevo con los polígonos del otro objeto.

Es posible usar la información de adyacencia de vértices para evitar el procesamiento adicional necesario para clasificar los polígonos de esta manera. Si un polígono es adyacente a un polígono clasificado (o sea, comparte vértices con él) y no tiene contacto con la superficie del otro objeto, se asigna al polígono la misma clasificación. Todos los vértices en la frontera común entre objetos se pueden marcar durante la fase inicial de división de polígonos. Podemos determinar si un polígono entra en contacto con la superficie del otro revisando si tiene vértices de frontera.

La clasificación de cada polígono determina si se conserva o descarta el polígono durante la operación de creación del objeto compuesto, como se describe en la sección 10.2. Por ejemplo, al formar la unión, la operación descarta todo polígono que pertenezca a un objeto que esté dentro del otro. La operación conserva los polígonos de cualquiera de los objetos que no se encuentre dentro del otro, excepto en el caso de los polígonos coplanares. Los polígonos coplanares se descartan si tienen normales a la superficie opuestas, y sólo se conserva uno de los dos si estas direcciones son iguales. Es importante la decisión de cuál polígono conservar si los objetos están formados por materiales diferentes. Aunque  $A \cup^* B$  tiene el mismo significado geométrico que  $B \cup^* A$ , en este caso las dos operaciones pueden tener resultados visiblemente distintos, de manera que la operación se pueda definir para favorecer a uno de los operandos en el caso de los polígonos coplanares.

## 10.6 Representaciones de partición espacial

En las representaciones de **partición espacial**, un sólido se descompone en una colección de sólidos adjuntos, no intersecantes, que son más primitivos que el sólido original, aunque no necesariamente del mismo tipo. Las primitivas pueden variar en tipo, tamaño, posición, parametrización y orientación, casi como las piezas de un juego de bloques de construcción para niños. El alcance de la descomposición de los objetos depende de cuán primitivos deben ser los sólidos para poder efectuar con facilidad las operaciones que nos interesan.



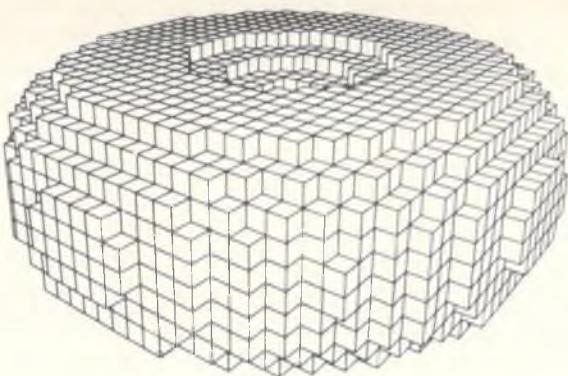
**Figura 10.13** Las celdas presentadas en (a) se pueden transformar de distintas maneras para construir el mismo objeto que aparece en (b) y (c). Incluso un solo tipo de celda es suficiente para ocasionar ambigüedades.

### 10.6.1 Descomposición en celdas

Una de las formas más generales de la partición espacial se denomina **descomposición en celdas**. Cada sistema de descomposición en celdas define un conjunto de celdas primitivas que por lo general se parametrizan y con frecuencia son curvas. La descomposición en celdas difiere de la generación de ejemplares de primitivas en que se pueden componer objetos más complejos a partir de otros más sencillos y primitivos, “pegándolos” en forma ascendente. La operación de *pegado* se puede considerar como una forma restringida de la unión, en la cual los objetos no pueden intersecarse. Otras restricciones que se aplican al pegado de objetos pueden requerir que dos celdas comparten un punto, arista o cara. Aunque la representación de descomposición en celdas no es ambigua, es posible que no sea única, como se ilustra en la figura 10.13. La descomposición en celdas también es difícil de validar, ya que en potencia hay que evaluar cada par de celdas para determinar si hay intersecciones. No obstante, la descomposición en celdas es una representación importante para el análisis de elementos finitos.

### 10.6.2 Enumeración de ocupación espacial

La **enumeración de ocupación espacial** es un caso especial de la descomposición en celdas, en la cual el sólido se descompone en celdas idénticas dispuestas sobre una malla regular fija. Estas celdas con frecuencia se denominan **elementos de volumen** (o voxels, por analogía con los pixeles). En la figura 10.14 se muestra un objeto representado con enumeración de ocupación espacial. El tipo de celda más común es el cubo, y la representación del espacio como un arreglo regular de cubos se denomina **cuberil**. Cuando se representa un objeto con enumeración de ocupación espacial, sólo se controla la presencia o la ausencia de una celda en cada posición de la malla. De esta manera, un objeto se puede codificar con una lista única y no ambigua de celdas ocupadas. Es más sencillo determinar si una celda está dentro o fuera del sólido, y también si dos objetos son adyacentes. La enumeración de ocupación espacial se usa con frecuencia en aplicaciones biomédicas para representar datos volumétricos obtenidos de fuentes como rastreo de tomografía axial computadorizada (CAT, computerized axial tomography).



**Figura 10.14** Toro representado por numeración de ocupación espacial. [Por A. H. J. Christensen, SIGGRAPH '80 Conference Proceedings, *Computer Graphics* (14)3, julio de 1980. Cortesía de Association for Computing Machinery, Inc.]

No obstante todas sus ventajas, la enumeración de ocupación espacial tiene varias deficiencias obvias, similares a las de la representación de una imagen bidimensional con un arreglo bidimensional de un bit de profundidad. No existe el concepto de ocupación *parcial*; por lo tanto, varios sólidos, como el toro de la figura 10.14, sólo pueden aproximarse. Si las celdas son cubos, los únicos objetos que se pueden representar con exactitud son aquellos cuyas caras son paralelas a los lados de cubo y cuyos vértices corresponden a la malla. Las celdas, como los pixeles en un arreglo bidimensional de bits, pueden en principio ser tan pequeñas como se desee para aumentar la precisión de la representación. Sin embargo, el espacio se convierte en un asunto de importancia, ya que requieren hasta  $n^3$  celdas ocupadas para representar un objeto con una resolución de  $n$  elementos de volumen en cada una de las tres dimensiones.

### 10.6.3 Árboles de octantes

Los **árboles de octantes** (*octrees*) constituyen una variante jerárquica de la enumeración de ocupación espacial, diseñada para considerar los exigentes requisitos de almacenamiento del método. Los árboles de octantes se derivan de los **árboles de cuadrantes** (*quadtrees*), un formato de representación bidimensional que se utiliza para codificar imágenes. Como se detalla en el exhaustivo estudio de Samet [SAME84], al parecer ambas representaciones fueron descubiertas de forma independiente por varios investigadores: los árboles de cuadrantes a finales de la década de 1960 y principios de la de 1970 [p. ej., WARN69; KLIN71] y los árboles de octantes a finales de los años setenta y principios de los años ochenta [p. ej., HUNT78; REDD78; JACK80; MEAG80; MEAG82].

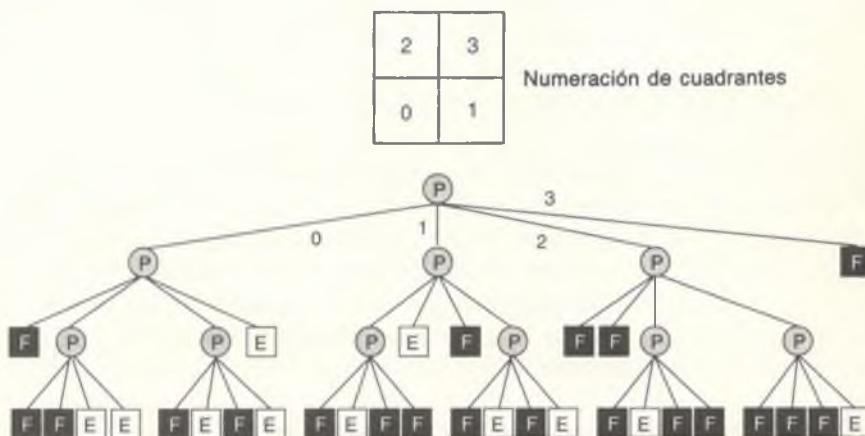
La idea que sustenta los árboles de cuadrantes y de octantes es el poder “divide y vencerás” de la subdivisión binaria. Un árbol de cuadrantes se forma



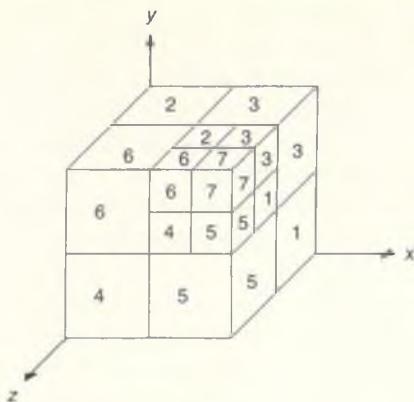
**Figura 10.15**  
Objeto representado con enumeración de partición espacial para tener (b) un árbol de cuadrantes.

dividiendo sucesivamente un plano bidimensional en ambas direcciones para formar cuadrantes, como se ilustra en la figura 10.15. Cuando se emplea un árbol de cuadrantes para representar un área en un plano, cada cuadrante puede estar lleno, parcialmente lleno o vacío (también llamado negro, gris o blanco, respectivamente), dependiendo de la cantidad de área que interseque el cuadrante. Un cuadrante parcialmente lleno se subdivide de manera recursiva en subcuadrantes. La subdivisión continúa hasta que todos los cuadrantes son homogéneos (llenos o vacíos) o hasta alcanzar una profundidad límite previamente determinada. Cuando cuatro cuadrantes hermanos están llenos o vacíos, se eliminan y su padre parcialmente lleno se reemplaza con un nodo totalmente lleno o vacío. (Se puede utilizar en cambio un enfoque ascendente (*bottom-up*) para evitar este problema de eliminación y fusión [SAME90b].) En la figura 10.15, cualquier nodo parcialmente lleno en la profundidad límite se clasifica como lleno. Las subdivisiones sucesivas se pueden representar como un árbol con cuadrantes parcialmente llenos en los nodos internos y cuadrantes llenos o vacíos en las hojas, como se presenta en la figura 10.16. Este concepto se puede comparar con el algoritmo de Warnock para la subdivisión de área, analizado en la sección 13.5.2. Si relajamos los criterios para clasificar un nodo como homogéneo, de manera que los nodos que se encuentren encima o debajo de cierto límite se puedan clasificar como llenos o vacíos, la representación será más compacta, aunque menos precisa. El árbol de octantes es similar al de cuadrantes, excepto que en sus tres dimensiones se subdivide para obtener octantes, como se ilustra en la figura 10.17.

Los cuadrantes generalmente se indican con los números 0 a 3 y los octantes con los números 0 a 7. Como no se ha desarrollado ningún mecanismo estándar para la numeración, también se emplean nombres mnemónicos. Los cuadrantes



**Figura 10.16** Estructura de árbol de cuadrantes para el objeto de la figura 10.15. F = lleno, P = parcialmente lleno, E = vacío.



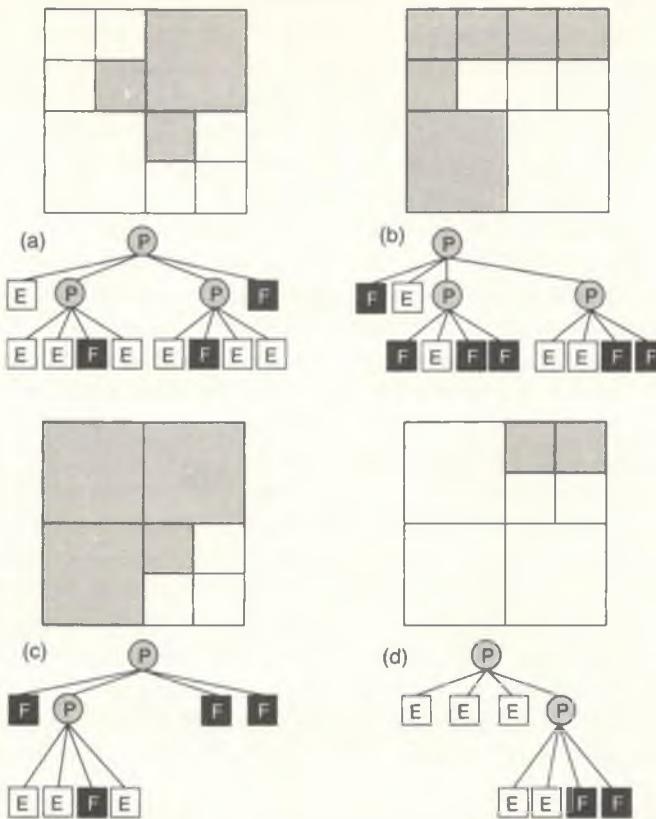
**Figura 10.17** Numeración de árbol de octantes. El octante cero no es visible.

se nombran de acuerdo con su dirección de brújula respecto al centro de su padre: NO, NE, SO y SE (o NW, NE, SW y SE, en inglés). Los octantes nombran en forma similar, distinguiendo entre la izquierda (L) y la derecha (R), arriba (U) y abajo (D), y anterior (F) y posterior (B): LUF, LUB, LDF, LD, RUF, RUB, RDF y RDB.

Con excepción de los peores casos, se puede demostrar que el número de nodos en una representación de árbol de cuadrantes u octantes es proporcional al perímetro o a la superficie del objeto, respectivamente [JUNT78; MEAG80]. Esta relación existe porque la subdivisión de nodos surge exclusivamente por necesidad de representar la frontera del objeto que se codifica. Los únicos nodos internos que se dividen son aquellos por los cuales pasa una parte de frontera. Por ende, cualquier operación con una de estas estructuras de datos que sea lineal en el número de nodos que contiene también se ejecuta en tiempo proporcional al tamaño del perímetro o área.

**Transformaciones y operaciones booleanas de conjuntos.** Se ha efectuado mucho trabajo en el desarrollo de algoritmos eficaces para almacenar y procesar árboles de cuadrantes y octantes [SAME84; SAME90a; SAME90b]. Por ejemplo, las operaciones booleanas de conjuntos tienen una aplicación bastante directa en árboles tanto de cuadrantes como de octantes [HUNT79]. Para calcular la unión o la intersección  $U$  de dos árboles,  $S$  y  $T$ , se efectúa un recorrido paralelo descendente por los dos árboles.

En la figura 10.18 se muestran las operaciones para árboles de cuadrantes; la generalización a los árboles de octantes es bastante sencilla. Se examina cada par correspondiente de nodos. Considere el caso de la unión. Si algunos de los nodos en el par es negro, se agrega un nodo negro correspondiente a  $U$ . Si ninguno de los nodos del par es blanco, se crea en  $U$  el nodo correspondiente con el valor



**Figura 10.18** Operaciones booleanas de conjuntos con árboles de cuadrantes. (a) El objeto  $S$  y su árbol de cuadrantes. (b) El objeto  $T$  y su árbol de cuadrantes. (c)  $S \cup T$ . (d)  $S \cap T$ . ( $F$  = lleno,  $P$  = parcialmente lleno,  $E$  = vacío.)

del otro nodo en el par. Si ambos nodos en el par son grises, se añade un nodo gris a  $U$  y se aplica recursivamente el algoritmo a los hijos del par. En este caso, hay que inspeccionar los hijos del nuevo nodo en  $U$  después de aplicar el algoritmo. Si todos son negros, se eliminan y su padre en  $U$  cambia de gris a negro.

Es fácil efectuar transformaciones sencillas con árboles de cuadrantes y de octantes. Por ejemplo, la rotación sobre un eje en múltiplos de  $90^\circ$  se logra rotando recursivamente los hijos en cada nivel. El escalamiento por potencias de 2 y las reflexiones también son bastante sencillos. Las traslaciones pueden ser más complejas, al igual que las transformaciones generales. Así mismo, como ocurre por lo general en la numeración de ocupación espacial, el problema del artefacto de discretización en las transformaciones generales es serio.

**Detección de vecinos.** Una operación importante en los árboles de cuadrantes y octantes es detectar el *vecino* de un nodo, es decir, hallar un nodo que sea ady-

cente al original (que comparta una cara, una arista o un vértice) y cuyo tamaño sea igual o mayor. Un nodo de árbol de cuadrantes tiene vecinos en ocho direcciones posibles. Sus vecinos en N, S, E y O lo son sobre una arista común, mientras que sus vecinos en NO, NE, SO y SE lo son con respecto a un vértice común. Un nodo de árbol de octantes tiene vecinos en 26 direcciones posibles: 6 vecinos sobre una cara, 12 sobre una arista y 8 sobre un vértice.

Samet [SAME89a] describe una forma de hallar al vecino de un nodo en una dirección especificada. El método comienza en el nodo original y asciende por el árbol de cuadrantes o de octantes hasta encontrar el primer antepasado común del nodo original y del vecino. El recorrido continúa entonces en forma descendente hasta encontrar el vecino deseado. En este caso hay que resolver de manera eficiente dos problemas: encontrar el antepasado común y determinar cuál de sus descendientes es el vecino. El caso más sencillo es encontrar el vecino de un nodo de árbol de octantes en la dirección  $d$  de una de sus caras: L, R, U, D, F o B. Al ascender por el árbol, iniciando en el nodo original, el antepasado común será el primer nodo que no pueda alcanzarse desde un hijo en el lado  $d$  del nodo. Por ejemplo, si se busca un vecino L, el primer antepasado común es el primer nodo que no pueda alcanzarse desde un hijo LUF, LUB, LDF o LDB. Esto se debe a que el nodo que se ha alcanzado desde uno de estos hijos no puede tener un hijo a la izquierda (es decir, un vecino L) del nodo original. Al encontrar el antepasado común, se desciende por su subárbol siguiendo la trayectoria desde el nodo original al antepasado, reflejada con respecto a la frontera común. Sólo se sigue parte de la trayectoria reflejada si el vecino es mayor que el nodo original.

#### 10.6.4 Árboles binarios de partición de espacio

Los árboles de octantes dividen recursivamente el espacio en planos que siempre son mutuamente perpendiculares y que bisecan las tres dimensiones de cada nivel del árbol. En cambio, los **árboles binarios de partición de espacio (BSP, binary space-partitioning trees)** dividen recursivamente el espacio en pares de subespacios, cada uno separado por un plano de orientación y posición arbitrarias. La estructura de datos de árbol binario que se crea se utilizó originalmente en la determinación de superficies visibles de gráficos, como se describe en la sección 13.5. Thibault y Naylor [THIB87] introdujeron después la utilización de los árboles BSP para representar poliedros arbitrarios. Cada nodo interno del árbol BSP está relacionado con un plano y tiene dos apuntadores a hijos, uno para cada lado del plano. Suponiendo que las normales apunten hacia afuera del objeto, el hijo de la izquierda está detrás o dentro del plano, mientras que el de la derecha se encuentra frente o fuera del plano. Si se divide aún más el medio espacio a un lado del plano, entonces su hijo es la raíz de un subárbol: el medio espacio es homogéneo, entonces el hijo es una hoja y representa una región que está completamente dentro o fuera del poliedro. Estas regiones homogéneas se denominan celdas *dentro* y *fuerza*. Para considerar la precisión numérica limitada con la cual se realizan las operaciones, cada nodo tiene

*grosor* relacionado con su plano. Cualquier punto que se encuentre dentro de esta tolerancia del plano se considera *dentro* del plano.

El concepto de subdivisión que sirve de base a los árboles BSP, como el de los árboles de cuadrantes y octantes, es independiente de la dimensión. De esta manera, en la figura 10.19(a) se muestra un polígono cóncavo acotado por líneas negras. Las celdas *dentro* se sombrean en color gris claro y las líneas que definen los medios espacios aparecen en gris oscuro, con las normales apuntando hacia afuera. El árbol BSP correspondiente se muestra en la figura 10.19(b). En dos dimensiones, las regiones *dentro* y *fuera* forman un teselado poligonal convexo del plano; en tres dimensiones, estas regiones forman un teselado poliédrico convexo del espacio tridimensional. Por lo tanto, un árbol BSP puede representar un sólido cóncavo arbitrario con agujeros como la unión de regiones *dentro* convexas.

Considere la tarea de determinar si un punto está dentro, fuera o sobre un sólido, un problema conocido como **clasificación de puntos** [TILO80]. Se puede usar un árbol BSP para clasificar un punto, filtrando el punto por el árbol a partir de la raíz. En cada nodo se sustituye el punto en la ecuación de plano del nodo y se pasa recursivamente al hijo de la izquierda si se encuentra detrás (dentro) del plano o al hijo de la derecha si está enfrente (fuera) del plano. Si el nodo es una hoja, el punto está indicado por el valor de la hoja, ya sea como *dentro* o *fuera*. Si el punto se halla en el plano de un nodo, se pasa a ambos hijos y se comparan las clasificaciones. De ser iguales, el punto recibe ese valor; cuando son diferentes, el punto se ubica en la frontera entre las regiones *fuera* y *dentro* y se clasifica como *sobre*. Este método se puede extender a la clasificación de líneas y polígonos. Sin embargo, a diferencia de los puntos, una línea o un polígono puede estar parcialmente en ambos lados del plano. Por consiguiente, en cada nodo cuyo plano interseca la línea o el polígono, hay que dividir (recortar) la línea o el polígono y obtener partes que estén enfrente, detrás o sobre el plano, para luego clasificar las partes por separado.

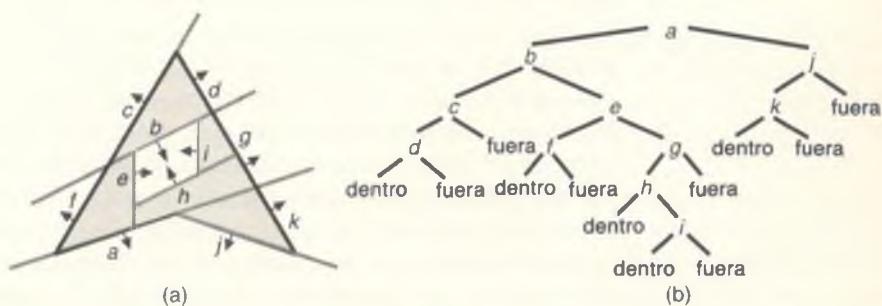


Figura 10.19 Representación de árbol BSP en dos dimensiones. (a) Polígono cóncavo acotado por líneas negras. Las líneas que definen los medios espacios están en gris oscuro y las celdas *dentro* están en gris claro. (b) Árbol BSP.

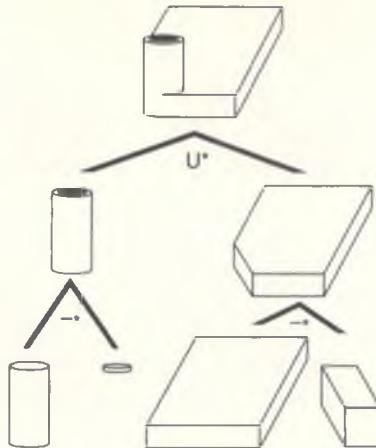
Thibault y Naylor describen algoritmos para construir un árbol BSP a partir de una representación de frontera, para efectuar operaciones de conjuntos booleanos a fin de combinar un árbol BSP con una representación de frontera y para determinar las porciones poligonales que se encuentran en la frontera de un árbol BSP [THIB87]. Estos algoritmos operan con árboles BSP cuyos nodos están asociados a listas de polígonos incorporados en el plano del nodo. Los polígonos se insertan en el árbol usando una variante del algoritmo de construcción de árbol BSP presentado en la sección 13.5.

Aunque los árboles BSP constituyen una forma de representación sencilla y elegante, los polígonos se subdividen conforme se construye el árbol y al efectuar las operaciones de conjuntos booleanos, por lo cual la notación quizás sea tan compacta como otras representaciones. Sin embargo, si se aprovecha la inherente independencia de la dimensión del árbol BSP, es posible desarrollar un álgebra booleana cerrada para árboles BSP tridimensionales que se base cursivamente en las representaciones de polígonos como árboles bidimensionales, aristas como árboles unidimensionales y puntos como árboles de una dimensión [NAYL90].

## 10.7 Geometría sólida constructiva

En la **geometría sólida constructiva** (CSG, *constructive solid geometry*), las primitives simples se combinan a través de operadores booleanos regularizados en conjuntos que se incluyen directamente en la representación. Un objeto se almacena como un árbol con operadores en los nodos internos y primitives simples en las hojas (Fig. 10.20). Algunos nodos representan operadores booleanos mientras que otros llevan a cabo traslaciones, rotaciones y escalamientos, en forma muy parecida a las jerarquías del capítulo 7. Como las operaciones booleanas por lo general no son conmutativas, las aristas del árbol están ordenadas.

Para determinar las propiedades físicas o para crear imágenes, debemos ser capaces de combinar las propiedades de las hojas para obtener las propiedades de la raíz. La estrategia general del procesamiento es un recorrido por profundidades a lo largo del árbol, como en el capítulo 7, para combinar los nodos a partir de las hoja. La complejidad de esta tarea depende de la representación de los objetos hoja y de si debe producirse realmente una representación completa del objeto compuesto en la raíz del árbol. Por ejemplo, los algoritmos de combinaciones de conjuntos booleanos para representaciones de fronteras, analizados en la sección 10.5, combinan las representaciones de fronteras de dos nodos para crear una tercera representación de frontera y son difíciles de implantar. El algoritmo CSG, que se examina en el capítulo 15 de [FOLE90], es mucho más sencillo y produce una imagen procesando las representaciones de las hojas y combinarlas de manera explícita.

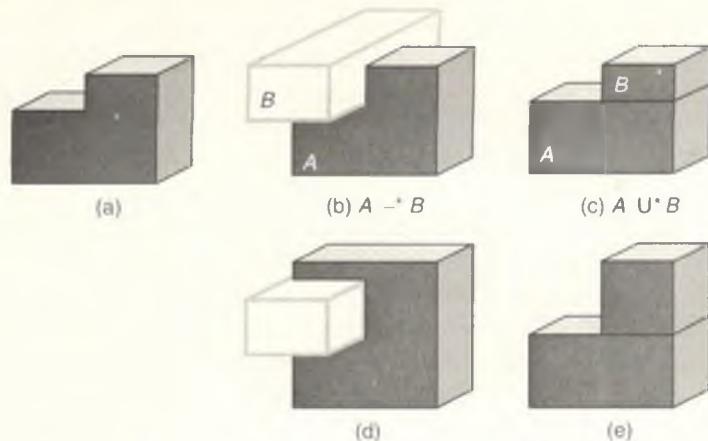


**Figura 10.20** Objeto definido por CSG y su árbol.

En algunas implantaciones, las primitivas son sólidos simples, como cubos o esferas, para asegurar que todas las combinaciones regularizadas también sean sólidos válidos. En otros sistemas, las primitivas incluyen medios espacios, que en sí no son sólidos acotados. Por ejemplo, un cubo se puede definir como la intersección de seis medios espacios, o un cilindro finito como un cilindro infinito cerrado en las partes superior e inferior por un medio espacio plano. La utilización de medios espacios introduce un problema de validez, ya que no todas las combinaciones producen sólidos. Sin embargo, los medios espacios son útiles para operaciones como el rebanado de un objeto por un plano, operación que de otra manera se llevaría a cabo usando la cara de otro objeto sólido. Si no se emplean medios planos surge la necesidad de tiempo de procesamiento adicional, ya que las operaciones regularizadas de conjuntos booleanos deben realizarse con el objeto completo durante la división, incluso si sólo hay que cortar una cara.

Podemos considerar la descomposición en celdas y la numeración de ocupación espacial como casos especiales de la CSG en los cuales el único operador es el operador implícito de pegado: la unión de dos objetos que se pueden tocar pero que deben tener interiores disjuntos (es decir, los objetos deben tener una intersección booleana regularizada nula).

La CSG no ofrece una representación única. Esta característica puede ser muy confusa en un sistema que permite al usuario manipular los objetos hoja con operadores de ajuste fino. Al aplicar la misma operación a dos objetos que son inicialmente iguales se pueden generar dos resultados diferentes, como se muestra en la figura 10.21. No obstante, la capacidad de editar modelos a través de la eliminación, adición, reemplazo y modificación de subárboles, aunada a la forma relativamente compacta de almacenamiento de los modelos, han hecho de CSG una de las formas predominantes para el modelado de sólidos.



**Figura 10.21** El objeto presentado en (a) se puede definir con varias operaciones CSG, como se ilustra en (b) y (c). El ajuste fino hacia arriba de la cara superior de (b) y (c) produce objetos diferentes, presentados en (d) y (e).

## 10.8 Comparación de representaciones

Hemos analizado cinco tipos principales de representaciones: generación ejemplares de primitivas, barridos, representaciones de fronteras, partición de espacio (incluyendo la descomposición en celdas, la numeración de ocupación espacial, los árboles de octantes y los árboles BSP) y CSG. Compare ahora estos tipos con base en los criterios presentados en la sección 10.1.

- **Precisión.** Los métodos de partición de espacio y de representaciones ligeramente de fronteras sólo producen aproximaciones a varios objetos. En algunas aplicaciones, como la determinación de una trayectoria para un robot, esta limitación no es una desventaja, siempre y cuando la aproximación se calcule con una resolución apropiada (por lo general no muy fina). Sin embargo, la resolución necesaria para producir gráficos visualmente atractivos o para calcular con precisión suficiente la interacción de objetos puede ser demasiado elevada para ser práctica. Las técnicas de sombreado suave que se analizan en el capítulo 14 no corregen los artefactos visuales ocasionados por las obvias aristas de los polígonos. Por lo tanto, los métodos que permiten utilizar gráficos de alta calidad generalmente emplean CSG con primitivas no poliédricas y representaciones de frontera que permiten superficies curvas. La generación de ejemplares de primitivas también puede producir imágenes de alta calidad, pero no permite combinar dos objetos sencillos usando operadores de conjuntos booleanos.

- *Dominio.* El dominio de los objetos que se pueden representar con la generación de ejemplares de primitivas y los barridos es limitado. En cambio, los métodos de partición espacial pueden representar cualquier sólido, aunque muchas veces sólo como una aproximación. Si se ofrecen otros tipos de caras y aristas además de los polígonos acotados por líneas rectas, es posible utilizar las representaciones de fronteras para representar una gama muy variada de objetos. Sin embargo, varios sistemas de representación de fronteras están restringidos a tipos de superficies y topologías sencillas. Por ejemplo, es probable que sólo puedan codificar combinaciones de cuádricas que sean 2-variedades.
- *Unicidad.* Los únicos métodos que garantizan la unicidad de una representación son el árbol de octantes y la numeración de ocupación espacial, ya que ofrecen una sola forma de representar un objeto de tamaño y posición específicos. En el caso de los árboles de octantes, hay que efectuar cierto procesamiento para asegurarse de que la representación se haya reducido totalmente (es decir, que ningún nodo gris tenga todos los hijos negros o blancos). La generación de ejemplares de primitivas por lo general no garantiza la unicidad; por ejemplo, una esfera se puede representar con una primitiva esférica o elíptica. Sin embargo, la unicidad se puede asegurar si se elige con cuidado el conjunto de primitivas.
- *Validez.* De todas las representaciones, las de frontera son las que se distinguen por ser las más difíciles de validar. No sólo es posible que las estructuras de datos de vértices, caras y aristas sean inconsistentes, sino que también puede haber caras o aristas que se intersequen. En cambio, cualquier árbol BSP representa un conjunto espacial válido, aunque no necesariamente un sólido acotado. Sólo hay que efectuar una sencilla revisión sintáctica local para validar un árbol CSG (que siempre está acotado si sus primitivas están acotadas) o un árbol de octantes, y no se requiere verificación para la numeración de ocupación espacial.
- *Cerradura.* Las primitivas creadas utilizando la generación de ejemplares de primitivas no se pueden combinar y los barridos simples no son cerrados al aplicarse operaciones booleanas. Por lo tanto, ninguno de estos métodos se emplea como representación interna en sistemas de modelado. Aunque ciertas representaciones de fronteras pueden presentar problemas de cerradura con operaciones booleanas (p. ej., la incapacidad de representar las que no sean 2-variedades), estos casos generalmente se pueden evitar.
- *Densidad y eficiencia.* Los métodos de representación generalmente se clasifican de acuerdo con su capacidad para producir modelos *evaluados* o *no evaluados*. Los modelos *no evaluados* contienen información que debe procesarse (o evaluarse) para efectuar operaciones básicas, como la determinación de la frontera del objeto. En lo que respecta a la utilización de operaciones booleanas, la CSG crea modelos no evaluados, ya que cada vez que se realizan cálculos hay que recorrer el árbol y evaluar las expresiones.

Por lo tanto, las ventajas de la CSG son lo compacto de la representación y su capacidad para registrar rápidamente operaciones booleanas y cambios de transformaciones, así como deshacer éstas con rapidez, ya que sólo hay que construir nodos del árbol. Los árboles de octantes y BSP también se pueden considerar como modelos no evaluados, al igual que una secuencia de operadores de Euler que crea una representación de frontera. Por otra parte, las representaciones de fronteras y la numeración de ocupación espacial a menudo se consideran modelos *evaluados* en lo que se refiere a que ya se han realizado varias de las operaciones booleanas utilizadas para crear un objeto. Observe que la utilización de estos términos es relativa; por ejemplo, si la operación que se realizará consiste en la determinación de si un punto está dentro de un objeto, se puede llevar a cabo más trabajo al evaluar una representación de frontera que al evaluar el árbol CSG equivalente.

Como veremos en el capítulo 13, existen varios algoritmos eficientes para generar imágenes de objetos codificados con representaciones de fronteras. Aunque la numeración de ocupación espacial y los árboles de octantes sólo pueden ofrecer aproximaciones burdas de muchos objetos, los algoritmos que se emplean para manipularlas son generalmente más sencillos que sus equivalentes para otras representaciones. Debido a ello, se han utilizado en sistemas de modelado de sólidos basados en hardware, diseñados para aplicaciones donde el aumento de velocidad de ejecución de las operaciones booleanas de conjuntos supera la poca resolución de las imágenes resultantes.

Algunos sistemas emplean varias representaciones, ya que algunas operaciones son más eficientes en una representación que en otra. Por ejemplo, GMSOLID [BOYS82] usa CSG para lograr una representación más compacta y una representación de frontera para recuperar rápidamente los datos útiles que no se especifican de manera explícita en CSG, como sería la conectividad. Aunque la representación CSG de GMSOLID siempre refleja el estado actual del objeto que se modela, su representación de frontera sólo se actualiza cuando se ejecutan las operaciones que la requieren. Además de los sistemas que mantienen dos representaciones separadas, obteniendo una a partir de la otra cuando es necesario, también hay sistemas híbridos que bajan hasta cierto nivel de detalle en un esquema y luego cambian a otro, pero nunca duplican información. En [MILL89] se analizan algunos de los aspectos importantes que se presentan al utilizar representaciones múltiples e híbridas.

Como señalamos en la sección 10.1, las representaciones alambradas sólo contienen información de vértices y aristas, sin hacer referencia a caras, y por ende son inherentemente ambiguas. Sin embargo, Markowsky y Wesley han desarrollado un algoritmo para obtener todos los poliedros que podrían representarse con un alambrado [MARK80], así como un algoritmo complementario que genera todos los poliedros que podrían producir una proyección bidimensional en particular [WESL81].

## 10.9 Interfaces con el usuario para el modelado de sólidos

El desarrollo de la interfaz con el usuario de un sistema de modelado de sólidos nos ofrece una oportunidad excelente para poner en práctica las técnicas de diseño de interfaces analizadas en el capítulo 8. Hay diversas técnicas que se prestan para las interfaces gráficas, incluyendo la aplicación directa de operadores booleanos regularizados de conjuntos el ajuste fino y los operadores de Euler. En los sistemas CSG se puede permitir que el usuario edite el objeto modificando o reemplazando uno de los subárboles o sólidos de hoja. Podemos definir operaciones de fusión y biselado para suavizar la transición de una superficie a otra. Las interfaces con el usuario de los sistemas que han tenido éxito son muy independientes de la representación interna que se elija. Sin embargo, la excepción es la generación de ejemplares de primitivas, ya que alienta a los usuarios a pensar en los objetos en función de parámetros de propósito especial.

En el capítulo 9 señalamos que hay varias formas equivalentes para describir la misma curva. Por ejemplo, la interfaz entre el usuario y un sistema de dibujo de curvas puede permitir que el usuario registre curvas controlando los vectores tangente hermitianos o especificando puntos de control de Bézier, mientras almacena las curvas exclusivamente como puntos de control de Bézier. En forma similar, un sistema de modelado de sólidos puede permitir que el usuario cree objetos en función de varias representaciones y los almacene con otra. Como sucede con las representaciones de curvas, cada representación de entrada puede tener ventajas que hagan de ella la opción más natural para crear el objeto.

La precisión con que deben especificarse los objetos muchas veces establece que deben ofrecerse medios para determinar mediciones en forma precisa, por ejemplo a través del dispositivo localizador o con una entrada numérica. Puesto que la posición de un objeto muchas veces depende de la de otros, las interfaces con frecuencia permiten restringir un objeto con respecto a otro. Una técnica relacionada consiste en proporcionar al usuario la capacidad para definir líneas de malla a fin de restringir la posición de los objetos.

Algunos de los problemas fundamentales del diseño de una interfaz para modelado de sólidos son los ocasionados por la necesidad de manipular y presentar objetos tridimensionales usando dispositivos de interacción y pantallas bidimensionales. Estos temas generales se analizaron en el capítulo 8. Muchos sistemas tratan algunos de estos problemas ofreciendo varias ventanas de presentación que permiten al usuario ver el objeto simultáneamente desde posiciones diferentes.

### RESUMEN

Como hemos visto, el modelado de sólidos es importante tanto en el CAD/CAM como en la graficación. Aunque existen varios sistemas y algoritmos que manejan los objetos descritos hasta ahora, aún hay varios problemas sin resolver. Uno de los más importantes es la cuestión de la robustez. Los sis-

temas de modelado de sólidos usualmente están plagados de inestabilidades numéricas. Los algoritmos de uso común requieren mayor precisión que la disponible en hardware para almacenar resultados intermedios de punto flotante. Por ejemplo, los algoritmos de operaciones de conjuntos booleanos pueden fracasar al tener que manejar dos objetos, uno de los cuales es una copia ligeramente transformada del otro.

Para lograr objetos articulados, flexibles y no rígidos se requieren mejores representaciones. Varios objetos no se pueden especificar con precisión absoluta; en cambio, sus formas se definen con parámetros restringidos para que caigan dentro de un intervalo de valores. Estos objetos, conocidos como “con tolerancias” (*toleranced*), corresponden a objetos reales producidos por máquinas como los tornos y las estampadoras [REQU84]. Se han desarrollado nuevas representaciones para codificar objetos con tolerancias [GOSS88].

Un concepto común en todos los objetos diseñados es el de las *características*, como agujeros y biseles, que se diseñan para fines específicos. Una de las áreas de investigación actual explora la posibilidad de reconocer características en forma automática y deducir el propósito del diseñador para cada característica [PRAT84]. Este proceso permitirá revisar el diseño y garantizar que las características actúen como se pretende que lo hagan. Por ejemplo, si se diseñan ciertas características para aumentar la resistencia de una pieza a la presión, entonces su capacidad para realizar esta función se podría validar en forma automática. Así mismo, las operaciones efectuadas posteriormente con el objeto se podrían revisar para no poner en riesgo la funcionalidad de las características.

## Ejercicios

10.1 Defina los resultados de aplicar  $\cup^*$  y  $-^*$  a dos objetos poliédricos, en la misma forma que se definió el resultado de  $\cap^*$  en la sección 10.2. Explique cómo el objeto resultante se limita a ser un conjunto regular, y especifique cómo se determina la normal a cada una de las caras del objeto.

10.2 Considere la tarea de determinar si un sólido legal es el objeto nulo (que no tiene volumen). ¿Cuán difícil es efectuar esta prueba en cada una de las presentaciones analizadas?

10.3 Considere un sistema cuyos objetos se representan como barridos y para los cuales se pueden aplicar operadores booleanos regularizados de conjuntos. ¿Qué restricciones hay que aplicar a estos objetos para garantizar la cerradura?

10.4 Explique por qué una implantación de operaciones de conjuntos booleanos con árboles de cuadrantes o de octantes no tiene que distinguir entre las operaciones ordinarias y las regularizadas que se describieron en la sección 10.2.

10.5 Aunque las consecuencias geométricas de la aplicación de operadores booleanos regularizados de conjuntos no son ambiguas, no es tan obvio cuáles se tratarán las propiedades del objeto. Por ejemplo, ¿qué propiedades de

asignarse a la intersección de dos objetos formados por materiales diferentes? Esta pregunta no tiene mucha importancia cuando se modelan objetos reales, pero en el mundo artificial de los gráficos es posible intersecar dos materiales cualesquiera. ¿Qué soluciones cree usted que serían útiles?

10.6 Explique cómo podría usarse un árbol de cuadrantes o de octantes para acelerar la selección bidimensional o tridimensional en un paquete de graficación.

10.7 Describa cómo realizar la clasificación de puntos en la generación de ejemplares de primitivas, las representaciones de fronteras, la numeración de ocupación espacial y la CSG.



Es indispensable que el estudiante de la graficación por computador moderna comprenda la teoría y la aplicación de la luz y el color. Incluso la aplicación sensata de unas cuantas sombras de color gris puede mejorar notablemente la apariencia de un objeto generado. La utilización del color es la responsable de gran parte del impacto que tienen las imágenes en la sección de ilustraciones en color. El color es un tema muy complejo, que utiliza conceptos de la física, la fisiología, la psicología, el arte y el diseño gráfico. En este capítulo presentaremos las áreas del color que tienen mayor importancia en la graficación por computador.

El color de un objeto no sólo depende del propio objeto, sino además de la fuente de luz que lo ilumina, del color del área que lo rodea y del sistema visual humano. Además, ciertos objetos reflejan la luz (paredes, escritorios, papel), mientras que otros la transmiten (celofán, vidrio). Cuando una superficie que sólo refleja luz azul pura es iluminada por luz roja pura, aparece negra. Así mismo, una luz verde pura observada a través de un vidrio que sólo transmite rojo puro también aparecerá negra. Pospondremos el tratamiento de algunos de estos temas e iniciaremos nuestro análisis con las sensaciones acromáticas, es decir, aquellas que se describen con blanco, gris y negro.

## 11.1 Luz acromática

*La luz acromática (literalmente la ausencia de color) es la que se observa en un televisor o pantalla de computador en blanco y negro. El observador de la luz*

acromática no experimenta ninguna de las sensaciones que relacionamos con el rojo, el azul, el amarillo, etcétera. El único atributo de la luz acromática es la cantidad de luz. La cantidad de luz se puede analizar en el sentido físico de la energía, en cuyo caso se emplean los términos **intensidad** o **luminancia**, o en el sentido psicológico de la intensidad percibida, en cuyo caso se emplea el término **brillantez**. Como veremos un poco más adelante, estos dos conceptos están relacionados pero no son iguales. Es conveniente relacionar un valor escalar con los diversos niveles de intensidad, definiendo el negro como 0 y el blanco como 1, mientras que los niveles de intensidad entre 0 y 1 representan distintos grises.

Un televisor en blanco y negro puede producir distintas intensidades en la misma posición de pixel. Las impresoras de línea, los graficadores de pluma y los graficadores electrostáticos sólo producen dos niveles: el blanco (o gris claro) del papel y el negro (o gris oscuro) de la tinta o el tonificador depositado en el papel. Ciertas técnicas, que analizaremos en secciones posteriores, permiten que estos dispositivos, que inherentemente son de **dos niveles**, produzcan **niveles** de intensidad adicionales.

### 11.1.1 Selección de intensidades

Suponga que queremos presentar 256 intensidades diferentes. Seleccionamos este número porque la brillantez de cada pixel de muchas imágenes se representa con ocho bits de datos. ¿Cuáles son los 256 niveles de intensidad que debemos usar? Obviamente, no queremos usar 128 en el intervalo de 0 a 0.1 ni 128 en el intervalo de 0.9 a 1.0, ya que la transición de 0.1 a 0.9 aparecería como discontinuidad. Inicialmente podríamos distribuir los niveles de manera uniforme en el intervalo de 0 a 1, pero esta opción ignora una característica importante del ojo humano: es sensible a razones de niveles de intensidad, no a sus valores absolutos. Es decir, percibimos la diferencia entre las intensidades 0.10 y 0.11 igual que la diferencia entre las intensidades 0.50 y 0.55. (Esta no linealidad es fácil de observar: vaya probando los valores de una bombilla casera de 50-100-150 watts; verá que el incremento de 50 a 100 parece mayor que el de 100 a 150.) Es una escala de brillantez (es decir, de intensidad percibida), la diferencia entre las intensidades 0.10 y 0.11 y la que existe entre las intensidades de 0.50 y 0.55 son iguales. Por lo tanto, los niveles de intensidad se deben espaciar logarítmicamente y no linealmente para lograr pasos iguales en la brillantez.

Para encontrar 256 intensidades, comenzando por la menor obtenible hasta la intensidad máxima de 1.0, con cada intensidad  $r$  veces mayor que la anterior, se emplean las siguientes relaciones:

$$I_0 = I_0, I_1 = rI_0, I_2 = rI_1 = r^2I_0, I_3 = rI_2 = r^3I_0, \dots, I_{255} = r^{255}I_0 = 1. \quad (11.1)$$

Por lo tanto,

$$r = (1/I_0)^{1/255}, I_j = r^j I_0 = (1/I_0)^{j/255} I_0 = I_0^{(255-j)/255} \quad \text{para } 0 \leq j \leq 255, \quad (11.2)$$

y, en forma general, para  $n + 1$  intensidades,

$$r = (1/I_0)^{1/n}, I_j = I_0^{(n-j)/n} \quad \text{para } 0 \leq j \leq n. \quad (11.3)$$

Con sólo cuatro intensidades ( $n = 3$ ) e  $I_0$  de  $\frac{1}{8}$  (un valor muy grande y poco realista, elegido sólo como ejemplo), la ecuación (11.3) nos dice que  $r = 2$  y se generan valores de intensidad de  $\frac{1}{8}, \frac{1}{4}, \frac{1}{2}$  y 1.

La mínima intensidad  $I_0$  que se puede obtener en un CRT está entre  $\frac{1}{200}$  y  $\frac{1}{4}$  de la intensidad máxima en 1.0. Por lo tanto, los valores típicos de  $I_0$  se hallan entre 0.005 y 0.025. El mínimo no es cero, debido al reflejo de la luz en el fósforo del CRT. La razón entre las intensidades máxima y mínima se denomina **intervalo dinámico**. Podemos hallar el valor exacto para un CRT específico presentando un cuadrado de color blanco sobre un campo negro y midiendo las dos intensidades con un fotómetro. Esta medición se lleva a cabo en una habitación totalmente a oscuras, de manera que la luz ambiental reflejada no afecte las intensidades. Con  $I_0$  de 0.02, correspondiente a un intervalo dinámico de 50, la ecuación (11.2) produce  $r = 1.0154595\dots$ , y las primeras y las dos últimas de las 256 intensidades de la ecuación (11.1) son 0.0200, 0.0203, 0.0206, 0.0209, 0.0213, 0.0216, ..., 0.9848, 1.0000.

La presentación correcta en un CRT de las intensidades definidas por la ecuación (11.1) es un proceso difícil, y registrarlas en película fotográfica es incluso más difícil, por las no linealidades en el CRT y la película. Estas dificultades se pueden superar usando una técnica conocida como **corrección gama**, que comprende cargar la tabla de consulta de un dispositivo de barrido con valores compensatorios. Los detalles de este procedimiento se presentan en [FOLE90].

Una pregunta obvia es: “¿Cuántas intensidades son suficientes?” Por “suficientes”, queremos decir el número necesario para representar una imagen en blanco y negro de tono continuo, de manera que la reproducción parezca continua. Esta apariencia se logra cuando la razón  $r$  es 1.01 o menor (por debajo de esta razón, el ojo no puede distinguir entre las intensidades  $I_j$  e  $I_{j+1}$ ) [WYSZ82, pág. 569]. Por lo tanto, para hallar el valor apropiado de  $n$ , el número de niveles de intensidad, asignamos  $r = 1.01$  en la ecuación (11.3):

$$r = (1/I_0)^{1/n} \quad \text{o} \quad 1.01 = (1/I_0)^{1/n}. \quad (11.4)$$

Al despejar  $n$  obtenemos

$$n = \log_{1.01}(1/I_0), \quad (11.5)$$

donde  $1/I_0$  es el intervalo dinámico del dispositivo.

En la tabla 11.1 se presenta el intervalo dinámico  $1/I_0$  para varios medios de presentación y la  $n$  correspondiente, que es el número de niveles de intensidad necesarios para mantener  $r = 1.01$  y al mismo tiempo utilizar todo el intervalo dinámico. Estos valores son teóricos y suponen procesos perfectos de reproducción. En la práctica, una pequeña borrosidad ocasionada por el ensanchamiento

**Tabla 11.1**

Intervalo dinámico ( $1/l_0$ ) y número de intensidades requeridas  $n = \log_{1.01} (1/l_0)$  para varios medios de presentación.

Medio de presentación	Intervalo dinámico típico	Número de intensidades
CRT	50-200	400-530
Impresiones fotográficas	100	465
Diapositivas fotográficas	1000	700
Papel revestido impreso en B/N*	100	465
Papel revestido impreso en color	50	400
Papel periódico impreso en B/N	10	234

\*B/N = blanco y negro

de la tinta y pequeñas cantidades de ruido aleatorio en la reproducción reduce notablemente la  $n$  en los medios impresos. Por ejemplo, en la figura 11.1 se presenta una fotografía de tono continuo y en la figura 11.2 se reproduce la misma fotografía con 4 y 32 niveles de intensidad. Con cuatro niveles, las transiciones o contornos entre un nivel y otro son bastante evidentes, ya que la razón entre intensidades sucesivas es mucho mayor que el valor ideal de 1.01. Con 32 niveles de intensidad apenas pueden detectarse los contornos, y para este tipo de imágenes desaparecerían con 64 niveles. Esta observación nos sugiere que 64 niveles de intensidad constituyen el mínimo absoluto que se requiere para una impresión libre de contornos de imágenes en blanco y negro de tono continuo sobre papel como el que se utiliza en este libro. Sin embargo, para un CRT ajustado en una habitación perfectamente oscura, el aumento en el intervalo dinámico requiere la utilización de más niveles.



**Figura 11.1** Fotografía de tono continuo.



**Figura 11.2** Efecto de los niveles de intensidad sobre la reproducción de imágenes. (a) Fotografía de tono continuo reproducida con cuatro niveles de intensidad. (b) Fotografía de tono continuo reproducida con 32 niveles de intensidad. (Cortesía de Alan Paeth, University of Waterloo Computer Graphics Lab.)

### 11.1.2 Aproximación por medios tonos

Muchas pantallas y dispositivos de impresión son de dos niveles (sólo producen dos niveles de intensidad), e incluso las pantallas de barrido de 2 o 3 bits por pixel producen menos niveles de intensidad de los que deseamos. ¿Cómo podemos ampliar el intervalo de intensidades disponibles? La respuesta se encuentra en la **integración espacial** que llevan a cabo nuestros ojos. Si vemos un área pequeña desde una distancia suficientemente grande, nuestros ojos promedian el detalle fino dentro del área pequeña y registran únicamente la intensidad global del área.

Este fenómeno se aprovecha en la impresión de fotografías en blanco y negro de diarios, revistas y libros, usando una técnica llamada **medios tonos** (o **punteado ordenado por puntos agrupados**, en la graficación por computador). Cada unidad pequeña de resolución se imprime con un círculo de tinta negra cuya área es proporcional a la negrura  $1 - I$  (donde  $I$  = intensidad) del área de la fotografía original. En la figura 11.3 se presenta parte del patrón de medios tonos, muy amplificado. Observe que el patrón forma un ángulo de  $45^\circ$  con respecto a la horizontal, llamado **ángulo de pantalla**. Los medios tonos de diarios utilizan de 60 a 80 áreas de tamaño y forma variables [ULIC87] por pulgada, mientras que los medios tonos en revistas y libros emplean de 110 a 200 por pulgada.

Los dispositivos de salida gráfica pueden aproximar los círculos de área variable de la reproducción con medios tonos. Por ejemplo, se puede utilizar un



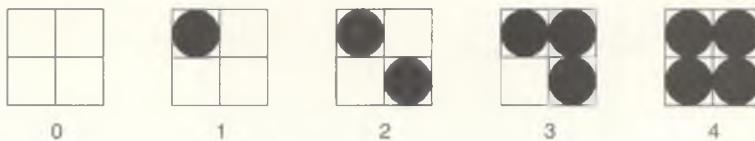
**Figura 11.3** La utilización de medios tonos expande el número de intensidades disponibles para los medios con intervalo dinámico pequeño. En este patrón ampliado de medios tonos podemos ver que los tamaños de los puntos varían inversamente con la intensidad de la fotografía original (véase la Fig. 11.1). (Cortesía de Alan Paeth, University of Waterloo Computer Graphics Lab.)

área de  $2 \times 2$  pixeles en una pantalla de dos niveles para producir cinco niveles de intensidad diferentes, a expensas de reducir la resolución espacial sobre cada eje. Los patrones presentados en la figura 11.4 pueden emplearse para rellenar las áreas de  $2 \times 2$  con el número de pixeles *encendidos* que sea proporcional a la intensidad deseada. En la figura 11.5 se presenta un rostro digitalizado como arreglo de imagen de 351 por 351 y presentado con patrones de  $2 \times 2$ .

Un grupo de  $n \times n$  pixeles de dos niveles puede proporcionar  $n^2 + 1$  niveles de intensidad. En términos generales, existe una conciliación entre la resolución espacial y la resolución de intensidad. Si usamos un patrón de  $3 \times 3$ , reduce en un tercio la resolución espacial en cada eje, pero se proporcionan 9 niveles de intensidad. Por supuesto, las opciones de conciliación están limitadas por nuestra agudeza visual (aproximadamente un minuto de arco con iluminación normal), la distancia a la cual se observa la imagen y la resolución en píxeles por pulgada del dispositivo gráfico.

La aproximación por medios tonos no está limitada a los dispositivos a dos niveles. Considere una pantalla con dos bits por pixel y, por lo tanto, cuatro niveles de intensidad. Podemos aprovechar la técnica de medios tonos para aumentar el número de niveles de intensidad. Si se emplea un patrón de  $2 \times 2$ , obtiene un total de cuatro pixeles disponibles, cada uno de los cuales puede mirar tres valores además del negro; con esto podemos presentar  $4 \times 3 + 1 = 13$  intensidades.

Las técnicas que hemos presentado hasta ahora suponen que el arreglo de la imagen que se presenta es menor que el arreglo de pixeles del dispositivo de presentación, de manera que pueden emplearse varios pixeles de presentación



**Figura 11.4** Cinco niveles de intensidad aproximados con cuatro patrones de punteado de  $2 \times 2$ .

para un pixel de la imagen. ¿Qué sucede si los arreglos de la imagen y del dispositivo de presentación son del mismo tamaño? Un método es usar la **difusión de errores**, una técnica desarrollada por Floyd y Steinberg [FLOY75]. Los resultados visuales que se obtienen al aplicar la difusión de errores generalmente son satisfactorios. El error (es decir, la diferencia entre el valor exacto del pixel y el valor aproximado que se presenta) se suma a los valores de los cuatro pixeles del arreglo de la imagen que se encuentran a la derecha y debajo del pixel en cuestión:  $\frac{7}{16}$  del error al pixel de la derecha,  $\frac{3}{16}$  al pixel debajo y a la izquierda,  $\frac{5}{16}$  al pixel inmediatamente debajo y  $\frac{1}{16}$  al pixel debajo y a la derecha. Esta estrategia tiene el efecto de distribuir o difundir el error sobre varios pixeles en el arreglo de la imagen. La figura 11.6 se creó con este método.

Si tenemos una imagen  $S$  que se presentará en la matriz de intensidad  $I$ , se calculan los valores para los pixeles modificados en  $S$  y los valores presentados en  $I$  en orden de línea de barrido, en forma descendente a partir de la línea de barrido superior:

```

K = aproximar(S[x][y]); /*Aproximar S a la intensidad más cercana que pueda presentarse */
I[x][y] = K; /*Dibujar el pixel en (x, y)*/
error = S[x][y] - K; /*Término de error. Debe ser de tipo float */

/* Paso 1: distribuir 7/16 del error al pixel de la derecha, en (x + 1, y) */
S[x + 1][y] += 7 * error/16;

/* Paso 2: distribuir 3/16 del error al pixel debajo y a la izquierda */
S[x - 1][y - 1] += 3 * error/16;

/* Paso 3: distribuir 5/16 del error al pixel de abajo */
S[x][y - 1] += 5 * error/16;

/* Paso 4: distribuir 1/16 del error al pixel debajo y a la derecha */
S[x + 1][y - 1] += error/16;

```

Para evitar la introducción de artefactos visuales en la imagen presentada, debemos asegurarnos de que los cuatro errores sumen exactamente *uno*; no se pueden permitir errores de redondeo. Esta restricción se puede cumplir calculando el término de error del paso 4 como *error* menos los términos de error de los tres pasos anteriores. La función *aproximar* devuelve el valor de intensidad



**Figura 11.5** Fotografía de tono continuo, digitalizada a una resolución de  $351 \times 351$  y presentada con los patrones de  $2 \times 2$  de la figura 11.4. (Cortesía de Alan Paeth, University of Waterloo Computer Graphics Lab.)

presentable más cercano al valor real del pixel. En el caso de una pantalla a dos niveles, el valor de  $S$  se redondea a cero o a uno.

Podemos obtener resultados aún mejores si alternamos el barrido de izquierda a derecha y de derecha a izquierda; en un barrido de derecha a izquierda se invierten las direcciones izquierda-derecha para los errores en los pasos 2 y 4. Para conocer los detalles de éste y otros métodos de difusión de error véase [ULIC87]; en [KNUT87] se analizan otros métodos.



**Figura 11.6** Fotografía de tono continuo reproducida con la difusión de error de Floyd-Steinberg. (Cortesía de Alan Paeth, University of Waterloo Computer Graphics Lab.)

## 11.2 Luz cromática

Las sensaciones visuales ocasionadas por la luz con color son mucho más ricas que las producidas por la luz acromática. Los análisis de la percepción de colores generalmente comprenden tres cantidades, conocidas como tinte, saturación y claridad. El **tinte** distingue entre colores como rojo, verde, morado y amarillo. La **saturación** se refiere a la distancia entre un color y un gris de la misma intensidad. El rojo tiene mucha saturación; el rosa tiene poca; el azul real es muy saturado; el azul cielo tiene relativamente poca saturación. Los colores pastel son poco saturados; los colores sin saturación incluyen más luz blanca que los vivos colores saturados. La **claridad** incorpora el concepto acromático de la intensidad percibida de un objeto reflejante. La **brillantez**, un cuarto término, se emplea en lugar de la **claridad** para hacer referencia a la intensidad percibida de un objeto autoluminoso (es decir, que emite luz en lugar de reflejarla), como una bombilla, el sol o un CRT.

Es necesario especificar y medir colores si vamos a usarlos de manera precisa en la graficación por computador. En el caso de la luz reflejada, estas tareas se llevan a cabo comparando visualmente una muestra del color desconocido con un conjunto de muestras *estándar*. Los colores desconocido y muestra deben observarse bajo una fuente luminosa estándar, ya que el color percibido de una superficie depende de la superficie y de la luz bajo la cual se observe. El popular sistema de clasificación de colores de Munsell incluye conjuntos de colores estándar [MUNS76] organizados en un espacio tridimensional de tinte, valor (lo que hemos definido como claridad) y croma (saturación). Cada color recibe un nombre y se ordena de manera que tenga una *distancia* percibida en el espacio de colores igual a la de sus vecinos (según lo establecido por varios observadores). [KELL76] presenta un análisis exhaustivo de las muestras estándar, diagramas que ilustran el espacio de Munsell y tablas de los nombres de colores.

En la industria de la impresión y en la profesión de diseño gráfico, los colores suelen especificarse de acuerdo con su equivalencia con muestras impresas de colores, como las proporcionadas por el PANTONE MATCHING SYSTEM® [PANT91].

Los artistas muchas veces especifican el color como distintos tintes, matices y tonos de pigmentos fuertemente saturados o puros. Se obtiene una **tinta** al añadir pigmento blanco a un pigmento puro, con lo cual se reduce la saturación. Un **matiz** se obtiene al añadir pigmento negro a un pigmento puro, con lo cual disminuye la claridad. Un **tono** es la consecuencia de añadir pigmentos blanco y negro a un pigmento puro. Todos estos pasos producen colores diferentes con el mismo tinte pero con variaciones en la saturación y la claridad. Si se mezclan únicamente pigmentos blanco y negro, se crean grises. En la figura 11.7 se presenta la relación entre tintas, matices y tonos. El porcentaje de los pigmentos que tienen que mezclarse para igualar un color se puede usar como especificación del color. El sistema de clasificación de Ostwald [OSTW31] es similar al modelo de tintas, matices y tonos de un artista.

### 11.2.1 Psicofísica

Los métodos de Munsell y de mezclado de colores de los artistas son subjetivos y dependen del juicio del observador humano, la iluminación, el tamaño de la muestra, el color circundante y la claridad global del ambiente. Necesitamos una forma objetiva y cuantitativa para especificar colores; para ello, pasamos a la rama de la física conocida como **colorimetría**. Algunos de los términos importantes en la colorimetría son la longitud de onda dominante, la pureza de excitación y la luminancia.

La **longitud de onda dominante** es la longitud de onda del color que "vemos" al observar la luz, y corresponde al concepto de percepción del **tinte**; la **pureza de excitación** corresponde a la saturación del color; la **luminancia** es la cantidad o intensidad de la luz. La pureza de excitación de una luz con color es la proporción entre la luz pura de la longitud de onda dominante y la luz blanca necesaria para definir el color. Un color totalmente puro está 100 por ciento saturado y por lo tanto no contiene luz blanca, mientras que las mezclas de color puro y luz blanca tienen saturaciones entre 0 y 100 por ciento. La luz blanca, y por ende los grises, tienen saturación del 0 por ciento, sin contenido de color de ninguna longitud de onda dominante. La correspondencia entre estos términos de percepción y de colorimetría es como sigue:

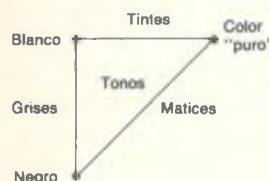


Figura 11.7

Tintes, tonos y matices.

#### Término de percepción

Tinte

Saturación

Claridad (objetos reflejantes)

Brillantez (objetos autoluminosos)

#### Colorimetría

Longitud de onda dominante

Pureza de excitación

Luminancia

Luminancia

Básicamente, la luz es energía electromagnética en la porción de longitud de onda de 400 a 700 nm del espectro que se percibe como los colores del violeta al rojo, pasando por índigo, azul, verde, amarillo y naranja. La cantidad de energía presente en cada longitud de onda está representada por una distribución energéticapectral  $P(\lambda)$ , como se ilustra en la figura 11.8. La distribución

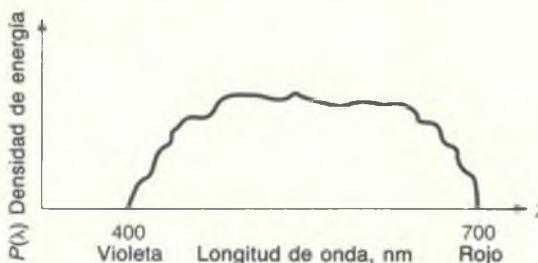


Figura 11.8 Distribución energéticapectral  $P(\lambda)$  típica de una luz.

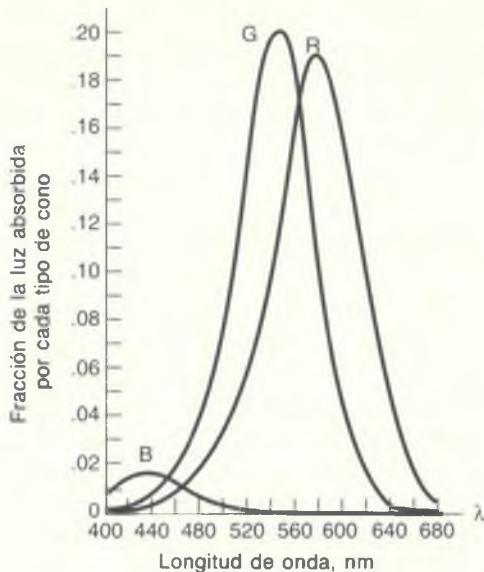


Figura 11.9 Funciones de respuesta espectral de los tres tipos de conos en la retina humana.

presenta una infinidad de números, uno para cada longitud de onda en el espectro visible (en la práctica, esta distribución se representa con gran cantidad de puntos de muestra en el espectro, según lo medido por un espectrorradiómetro). Por fortuna, podemos describir el efecto visual de cualquier distribución espectral de manera mucho más concisa con el triple [longitud de onda dominante, pureza de excitación, luminancia]. Esto implica que varias distribuciones diferentes de energía espectral producen el mismo color: se “ven” iguales. Por ende, la relación entre las distribuciones esenciales y los colores es de muchas a uno.

¿Cómo se relaciona este análisis con los puntos de fósforo rojo, verde y azul en un CRT en color? ¿Cómo se relaciona con la teoría de tres estímulos de la percepción de colores, que se basa en la hipótesis de que la retina tiene tres tipos de sensores de color (llamados conos) con sensibilidad máxima a las luces rojas, verdes o azules? Los experimentos basados en esta hipótesis producen las funciones de respuesta espectral de la figura 11.9. La respuesta máxima al azul está a alrededor de 440 nm; la del verde a unos 545 nm; la del rojo a unos 580 nm. (Los términos *verde* y *rojo* son un poco engañosos en este contexto, ya que los picos en 545 nm y 580 nm en realidad se encuentran en el intervalo del amarillo.) Las curvas sugieren que la respuesta del ojo a la luz azul es más débil que su respuesta al rojo o al verde.

En la figura 11.10 se presenta la función de eficiencia luminosa —la respuesta del ojo a una luz de luminancia constante— conforme varía la longitud de onda dominante: nuestra sensibilidad máxima es a la luz verde-amarillo con

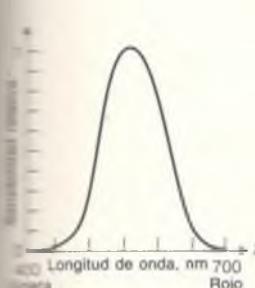


Figura 11.10  
Función de eficiencia  
luminosa para el ojo  
humano.

longitud de onda de alrededor de 550 nm. Hay evidencia experimental de esta curva es la suma de las tres curvas que se presentan en la figura 11.9.

La teoría de tres estímulos es intuitivamente muy atractiva porque corresponde de manera general al concepto de que los colores se pueden especificar con sumas positivamente ponderadas de rojo, verde y azul (los llamados colores primarios). Este concepto casi es verdadero: las tres funciones de equivalencia de colores de la figura 11.11 muestran las cantidades de luz roja, verde y azul que se requieren para que un observador promedio iguale un color de luminancia constante, para todos los valores de una longitud de onda dominante en el espectro visible.

Un valor negativo en la figura 11.11 indica que no podemos igualar el color con la suma de los primarios. Sin embargo, si se agrega uno de los primarios a la muestra de color, ésta se puede igualar con una mezcla de los otros dos primarios. De esa manera, los valores negativos en la figura 11.11 indican qué primario añadió el primario al color que se iguala. La necesidad de valores negativos quiere decir que no sea válido el concepto de mezclar el rojo, el verde y el azul para obtener los demás colores; por el contrario, se puede igualar una gran variedad muy extensa de colores usando cantidades positivas de rojo, verde y azul. Si fuera así, un CRT en color no funcionaría! Sin embargo, sí significa que no es posible producir ciertos colores con mezclas RGB (*red, green, blue*; rojo, verde y azul) y por consiguiente no pueden presentarse en un CRT ordinario.

El ojo humano puede distinguir entre cientos de miles de colores distintos en el espacio de colores, cuando observadores diferentes juzgan colores y tintos que se encuentran uno al lado del otro y determinan si los colores

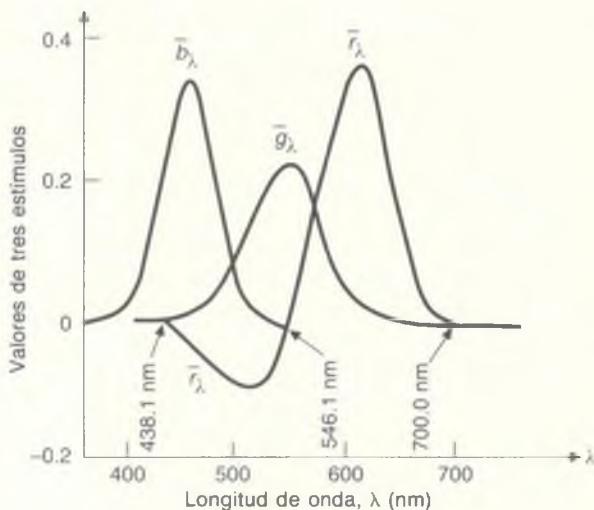


Figura 11.11 Funciones de equivalencia de colores, donde se muestran las cantidades de los tres colores primarios necesarios para igualar las longitudes de onda del espectro visible.

iguales o distintos. Cuando los colores sólo difieren en cuanto a tinte, la longitud de onda entre colores cuya diferencia es apenas perceptible varía de más de 10 nm en los extremos del espectro a menos de 2 nm alrededor de 480 nm (azul) y 580 nm (amarillo) [BEDF58]. Sin embargo, con la excepción de los extremos del espectro, la mayoría de los tintes distinguibles se encuentra a unos 4 nm. En conjunto, se pueden distinguir uno 128 tintes completamente saturados.

El ojo es menos sensible a los cambios de tinte de la luz menos saturada, mientras que la sensibilidad a los cambios en saturación para tinte y claridad fijos es mayor en los extremos del espectro visible, donde existen unos 23 pasos distinguibles.

### 11.2.2 Diagrama de cromaticidad CIE

La equivalencia y, por ende, la definición de una luz de color usando una mezcla de tres colores primarios fijos es una estrategia deseable para la especificación de colores, pero la necesidad de ponderaciones negativas que propone la figura 11.11 complica el asunto. En 1931, la *Commission Internationale de L'Éclairage* (CIE, Comisión Internacional de Iluminación) definió tres colores primarios estándar, llamados X, Y y Z, para reemplazar al rojo, verde y azul en este proceso de equivalencia. En la figura 11.12 se presentan las tres funciones correspondientes de equivalencia de colores,  $\bar{x}_\lambda$ ,  $\bar{y}_\lambda$  y  $\bar{z}_\lambda$ . Los colores primarios se pueden usar para igualar todos los colores que podemos ver, usando únicamen-

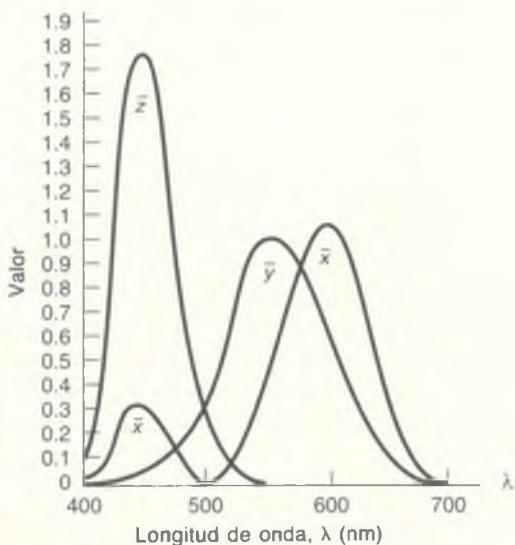
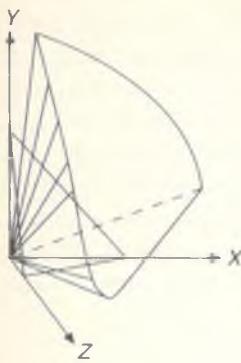


Figura 11.12 Funciones de equivalencia de colores  $\bar{x}_\lambda$ ,  $\bar{y}_\lambda$  y  $\bar{z}_\lambda$  para los colores primarios X, Y y Z CIE 1931.

**Figura 11.13**

El cono de los colores visibles en el espacio de colores CIE, indicado por las líneas que parten del origen. Se muestra el plano  $X + Y + Z = 1$ . (Cortesía de Gary Meyer, Program of Computer Graphics, Cornell University, 1978.)

te ponderaciones positivas. El primario  $\mathbf{Y}$  se definió intencionalmente para que tuviera la función de equivalencia de colores  $y_\lambda$ , que equivale exactamente a la función de eficiencia luminosa de la figura 11.10. Observe que  $x_\lambda$ ,  $y_\lambda$  y  $z_\lambda$  no son las distribuciones espectrales de los colores  $\mathbf{X}$ ,  $\mathbf{Y}$  y  $\mathbf{Z}$ , así como las curvas en la figura 11.11 no son las distribuciones espectrales del rojo, el verde y el azul. Se trata únicamente de funciones auxiliares que se usan para calcular las cantidades de  $\mathbf{X}$ ,  $\mathbf{Y}$  y  $\mathbf{Z}$  que hay que mezclar para generar una distribuciónpectral de cualquier color visible.

Las cantidades de los colores primarios  $\mathbf{X}$ ,  $\mathbf{Y}$  y  $\mathbf{Z}$  necesarias para igualar un color con distribución energéticapectral  $P(\lambda)$  son

$$X = k \int P(\lambda) \bar{x}_\lambda d\lambda, \quad Y = k \int P(\lambda) \bar{y}_\lambda d\lambda, \quad Z = k \int P(\lambda) \bar{z}_\lambda d\lambda. \quad (11.6)$$

En el caso de objetos autoluminosos como un CRT,  $k$  es 680 lumens por watt. En el caso de objetos reflejantes,  $k$  suele seleccionarse de manera que un blanco brillante tenga valor  $Y$  de 100; así, todos los valores de  $Y$  estarán en un intervalo de 0 a 100.

En la figura 11.13 se presenta el volumen en forma de cono del espacio XYZ que contiene los colores visibles. El volumen se extiende del origen al octante positivo y termina en la línea suave que limita el cono.

Sean  $(X, Y, Z)$  las ponderaciones aplicadas a los primarios CIE para igualar un color  $\mathbf{C}$ , determinados con la ecuación (11.16). Entonces,  $\mathbf{C} = X \mathbf{X} + Y \mathbf{Y} + Z \mathbf{Z}$ . Los valores de **cromaticidad** (que dependen únicamente de la longitud de onda dominante y de la saturación, y que son independientes de la cantidad de energía luminosa) se definen normalizando con respecto a  $X + Y + Z$ , que puede considerarse como la cantidad total de energía luminosa:

$$x = \frac{X}{(X + Y + Z)}, \quad y = \frac{Y}{(X + Y + Z)}, \quad z = \frac{Z}{(X + Y + Z)}. \quad (11.7)$$

Observe que  $x + y + z = 1$ . Es decir,  $x$ ,  $y$  y  $z$  se encuentran en el plano  $(X + Y + Z = 1)$  de la figura 11.13. La ilustración en color 14 muestra el plano  $X + Y + Z = 1$  como parte del espacio CIE y también una vista ortográfica del plano junto con una proyección del plano sobre el plano  $(X, Y)$ . Esta última proyección es el diagrama de cromaticidad CIE.

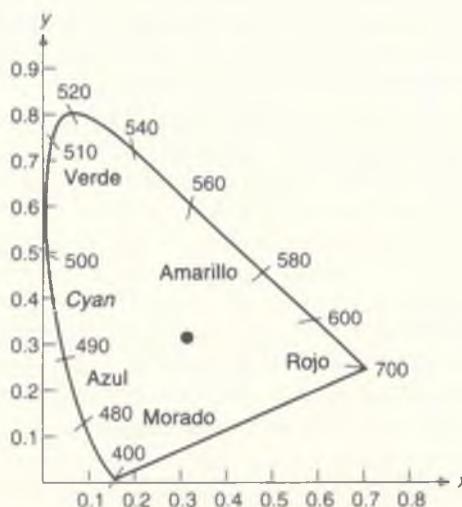
Si especificamos  $x$  y  $y$ , entonces  $z$  se determina como  $z = 1 - x - y$ . Sin embargo, no podemos obtener  $X$ ,  $Y$  y  $Z$  a partir de  $x$  y  $y$ . Para ello necesitaremos un elemento de información adicional, por lo general  $Y$ , que contiene información de luminancia. Si tenemos  $(x, y, Y)$ , la transformación a la  $(X, Y, Z)$  correspondiente es

$$X = \frac{x}{y} Y, \quad Y = Y, \quad Z = \frac{1-x-y}{y} Y. \quad (11.8)$$

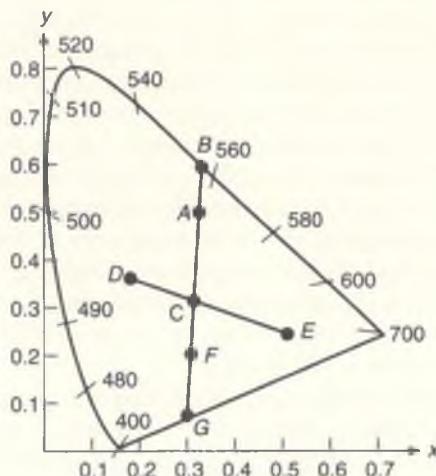
Los valores de cromaticidad sólo dependen de la longitud de onda dominante y de la saturación, y son independientes de la cantidad de energía lumino-

sa. Graficando  $x$  y  $y$  para todos los colores visibles, se obtiene el diagrama de cromaticidad CIE presentado en la figura 11.14, que es la proyección sobre el plano ( $X$ ,  $Y$ ) del plano ( $X + Y + Z = 1$ ) de la figura 11.13. El interior y la frontera de la región en forma de herradura representan todos los valores de cromaticidad posibles. (Todos los colores perceptibles con la misma cromaticidad pero diferente luminancia corresponden al mismo punto en esta región.) Los colores 100 por ciento espectralmente puros se encuentran en la parte curva de la frontera. Una luz blanca estándar, que se aproxima a la luz solar, se define de manera formal como una fuente luminosa **C iluminante**, marcada con el punto central. Se encuentra cerca del lugar donde  $x = y = z = \frac{1}{3}$ , aunque no precisamente allí. La C iluminante se definió mediante la especificación de una distribución de potenciapectral cercana a la luz diurna a una temperatura de color correlacionado de 6774° kelvin.

El diagrama de cromaticidad CIE es muy útil. En primer lugar, nos permite medir la longitud de onda dominante y la pureza de excitación de cualquier color igualando el color a una mezcla de los tres colores primarios CIE. Suponga ahora que el color igualado se encuentra en el punto *A* de la figura 11.15. Al sumar dos colores, el color resultante está en algún lugar de la línea recta del diagrama de cromaticidad que conecta los dos colores que se suman. Por lo tanto, el color *A* se puede considerar como una mezcla de luz blanca “estándar” (C iluminante) y la luz espectral pura en el punto *B*. Así, *B* define la longitud de onda dominante. La razón entre la longitud *AC* y la longitud *BC*, expresada como porcentaje, es la pureza de excitación de *A*. Cuanto más cerca esté *A* de *C*, incluye más luz blanca y es menos pura.



**Figura 11.14** Diagrama de cromaticidad CIE. Las longitudes de onda alrededor de la periferia se indican en nanómetros. El punto marca la posición de la C iluminante.



**Figura 11.15** Colores en el diagrama de cromaticidad. La longitud de onda dominante del color *A* es la del color *B*. Los colores *D* y *E* son complementarios. La longitud de onda dominante del color *F* se define como el complemento de la longitud de onda dominante del color *A*.

El diagrama de cromaticidad elimina la luminancia, de manera que se incluyen las sensaciones de color relacionadas con ella. Por ejemplo, no aparece el marrón, que es una cromaticidad naranja-rojo con muy poca luminancia relativa al área que la rodea. Por ello es importante recordar que el diagrama de cromaticidad no es una paleta completa de los colores. En el espacio (*X*, *Y*, *Z*) hay infinidad de planos, cada uno de los cuales se proyecta sobre el diagrama de cromaticidad y pierde información de luminancia al hacerlo. Todos los colores que se encuentran en estos planos son diferentes.

Los colores **complementarios** son aquellos que se pueden mezclar para producir luz blanca (como *D* y *E* en la figura 11.15). Algunos colores (como *F* en la figura 11.15) no se pueden definir con una longitud de onda dominante, y por tanto se les denomina **no espectrales**. En este caso se dice que la longitud de onda dominante es el complemento de la longitud de onda en la cual la línea que pasa por *F* y *C* interseca la parte en forma de herradura de la curva en el punto *C* y se designa con una “c” (en este caso, unos 555 nm c). La pureza de excitación se define con la razón de las longitudes (en este ejemplo, *CF* a *CG*). Los colores que deben expresarse usando longitudes de onda dominantes complementarias son los morados y magentas, que aparecen en la parte inferior del diagrama CIE.

Otra aplicación del diagrama de cromaticidad CIE es la definición de **gamas de colores**, o intervalos de colores, que muestran el efecto de la adición de colores. Podemos sumar dos colores, digamos *I* y *J* en la figura 11.16, para producir cualquier color que esté sobre la línea que los conecta, variando las cantidades relativas que se suman de los dos colores. Se puede utilizar un tercer color (*K* véase la Fig. 11.16) con diversas mezclas de *I* y *J* para producir la gama de todos los colores en el triángulo *IJK*, de nuevo variando las cantidades relativas.

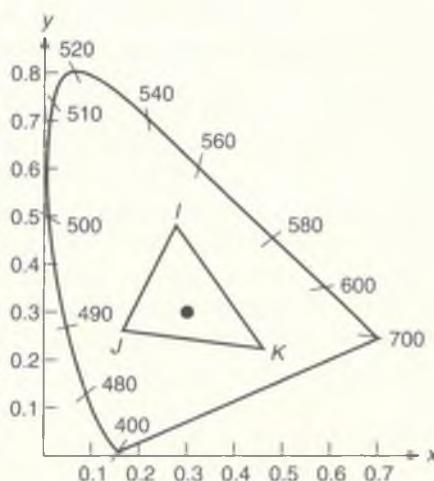
La forma del diagrama muestra por qué no se pueden mezclar aditivamente el rojo, el verde y el azul visibles para igualar todos los colores: no existe ningún triángulo cuyos vértices se encuentren dentro del área visible y que abarque completamente esta área.

El diagrama de cromaticidad también se utiliza para comparar las gamas disponibles en varios dispositivos de pantalla e impresión en color. En la ilustración en color 15 se muestran las gamas de un monitor de televisión, una película fotográfica y una impresión en color. La pequeñez de la gama de impresión con respecto a la gama del monitor en color sugiere que, si hay que reproducir fielmente en forma impresa las imágenes observadas en el monitor, es necesario reducir la gama de colores utilizada en éste. De lo contrario, sería imposible efectuar la reproducción exacta. Sin embargo, si el objetivo es obtener una reproducción agradable pero no exacta, no tienen tanta importancia las pequeñas diferencias en las gamas de colores. En [HALL89] puede hallar un análisis de la compresión de gamas de colores.

Con base en estos antecedentes acerca del color, pasemos ahora al color en la graficación por computador.

## 11.3 Modelos de colores para gráficos de trama

Un **modelo de colores** es una especificación de un sistema tridimensional de coordenadas de colores con un subconjunto visible en el sistema de coordenadas



**Figura 11.16** Mezcla de colores. Podemos crear todos los colores en la línea  $IJ$  mezclando los colores  $I$  y  $J$ ; podemos crear todos los colores en el triángulo  $IJK$  mezclando los colores  $I$ ,  $J$  y  $K$ .

donde se encuentran todos los colores de una gama específica. Por ejemplo, el modelo de colores RGB es el subconjunto de cubo unidad del sistema de coordenadas cartesianas tridimensionales.

El propósito de un modelo de colores es permitir la especificación práctica de colores dentro de una gama. Nuestro principal interés es la gama para los monitores CRT a color, definida por los primarios RGB (*red, green, blue*; rojo, verde y azul) en la ilustración en color 15. Como puede observarse en dicha ilustración, una gama de colores es un subconjunto de todas las cromaticidades visibles. Por lo tanto, no es posible utilizar un modelo de colores para especificar todos los colores visibles.

Tres modelos de colores orientados al hardware son el RGB, usado con monitores CRT en color; el YIQ, el sistema para la transmisión de TV en color; y CMY (*cyan, magenta, yellow*; cyan, magenta y amarillo) usado en ciertos dispositivos de impresión en color. Por desgracia, ninguno de estos modelos es fácil de usar, ya que no se relacionan en forma directa con los conceptos intuitivos de tinte, saturación y brillantez del color. Por lo tanto, se han desarrollado otros modelos cuya finalidad es la sencillez de uso. Varios de estos modelos se analizan en [GSPC79; JOBL78; MEYE80; SMIT78]. Nosotros sólo analizaremos uno: el modelo HSV (en ocasiones conocido como modelo HSB).

Cada modelo tiene una forma de convertirse a otra especificación. Veremos cómo convertir entre RGB y los modelos HSV y CMY, y entre RGB y YIQ. En [FOLE90] se incluyen otros algoritmos de conversión.

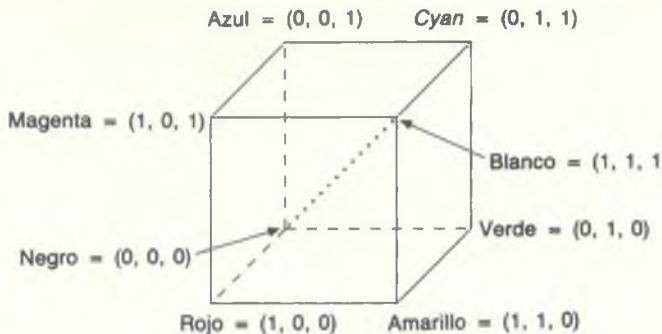
### 11.3.1 Modelo de colores RGB

El modelo de colores RGB utilizado en los monitores CRT en color y en los gráficos de barrido en color emplea un sistema de coordenadas cartesianas. Los colores primarios del modelo RGB son **aditivos**; es decir, para producir el resultado se suman las contribuciones individuales de cada primario, como se propone en la ilustración en color 16. El subconjunto de interés es el cubo unidad presentado en la figura 11.17. La diagonal principal del cubo, con cantidades iguales de cada primario, representa los niveles de gris: el negro es (0, 0, 0) y el blanco es (1, 1, 1).

La gama de colores cubierta por el modelo RGB está definida por las cromaticidades de los fósforos de un CRT. Dos CRT con fósforos diferentes abarcarán gamas distintas. Para convertir los colores especificados en la gama de un CRT a la gama de otro se emplean las transformaciones del espacio de colores RGB de cada monitor al espacio de colores ( $X, Y, Z$ ). Véase [FOLE90] para conocer los detalles.

### 11.3.2 Modelo de colores CMY

El *cyan*, el magenta y el amarillo (denotado con Y) son los complementos del rojo, el verde y el azul, respectivamente. Cuando se emplean como filtros para formar luz blanca, se denominan primarios **sustractivos**. El subconjunto del sistema de coordenadas cartesianas para el modelo CMY es el mismo que para el



**Figura 11.17** Cubo RGB. Los grises se encuentran en la diagonal principal punteada.

RGB, excepto que en el origen está el blanco (luz total) en lugar del negro (ausencia de luz). Los colores se especifican con lo que se quita o resta a la luz blanca, en lugar de lo que se agrega al negro.

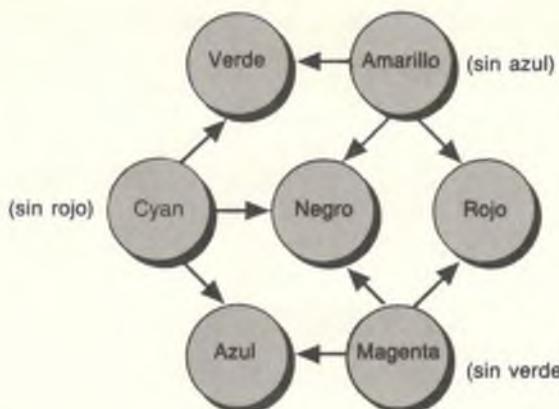
Es importante conocer el modelo CMY cuando se tienen que utilizar dispositivos de impresión que depositan pigmentos de color sobre papel, como los graficadores electrostáticos y de chorro de tinta. Cuando una superficie se cubre con tinta *cyan*, no se refleja luz roja. El *cyan* resta el rojo a la luz blanca reflejada, la cual es la suma del rojo, el verde y el azul. Por lo tanto, en función de los primarios aditivos, el *cyan* es el blanco menos el rojo, o sea, el azul más el verde. En forma similar, el magenta absorbe el verde, de manera que es rojo más azul; el amarillo absorbe el azul, de manera que es rojo más verde. Una superficie cubierta con *cyan* y amarillo absorbe el rojo y el azul, con lo cual sólo se refleja el verde a partir de la luz blanca iluminante. Una superficie con *cyan*, amarillo y magenta absorbe el rojo, el verde y el azul, por lo cual es negra. Estas relaciones, presentadas en forma de diagrama en la figura 11.18, se pueden observar en la ilustración en color 17 y se representan con la siguiente ecuación:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (11.9)$$

El vector columna unidad es la representación RGB para el blanco y la representación CMY para el negro.

La conversión de RGB a CMY es entonces

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix} \quad (11.10)$$



**Figura 11.18** Colores primarios sustractivos (cyan, magenta y amarillo) y sus mezclas. Por ejemplo, el cyan y el amarillo se combinan para formar el verde.

Estas transformaciones directas se pueden utilizar para convertir los ocho colores que se pueden obtener con las combinaciones binarias de rojo, verde y azul a los ocho colores que se obtienen con combinaciones binarias de cyan, magenta y amarillo. Esta conversión es relevante para las impresoras a color a chorro de tinta y xerográficas.

Otro modelo de colores, CMYK, usa el negro (denotado con K) como cuarto color. El modelo CMYK se emplea en el proceso de impresión de cuatro colores de las prensas y otros dispositivos de impresión. Dada una especificación CMY, se usa el negro en lugar de cantidades iguales de C, M y Y, de acuerdo con las siguientes relaciones:

$$K = \min(C, M, Y)$$

$$C = C - K$$

$$M = M - K$$

$$Y = Y - K$$

(11.33)

Este tema se trata con mayor detalle en [STON88].

### 11.3.3 Modelo de colores YIQ

El modelo YIQ se emplea en la transmisión de televisión comercial en Estados Unidos y por lo tanto se encuentra estrechamente relacionado con los gráficos de barrido en color. El modelo YIQ es una recodificación de RGB para obtener mayor eficiencia de transmisión y para tener compatibilidad descendente con la televisión en blanco y negro. La señal registrada se transmite usando el estándar NTSC (National Television System Committee) [PRIT77].

El componente  $Y$  de YIQ no corresponde al amarillo, sino a la luminancia; se define como el primario Y CIE. En un televisor en blanco y negro sólo se presenta el componente  $Y$  de la señal. La cromaticidad está codificada en  $I$  y  $Q$ . El modelo YIQ utiliza un sistema tridimensional de coordenadas cartesianas, donde el subconjunto visible es un poliedro convexo que corresponde al cubo RGB.

La correspondencia RGB-YIQ se define como sigue:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.528 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (11.12)$$

Las cantidades en la primera fila reflejan la importancia relativa del verde y el rojo, así como la poca importancia relativa del azul en la brillantez. La inversa de la matriz RGB a YIQ se usa para la conversión YIQ a RGB.

La especificación de colores con el modelo YIQ resuelve un problema potencial con el material que se prepara para la transmisión televisiva: dos colores presentados lado a lado en un monitor a color aparecerán diferentes, pero al convertirse a YIQ y verse en un monitor monocromático, pueden parecer iguales. Este problema se puede evitar especificando los colores con distintos valores  $Y$  en el espacio de colores del modelo YIQ (es decir, ajustando sólo el valor  $Y$  para eliminar la ambigüedad).

El modelo YIQ aprovecha dos propiedades útiles de nuestro sistema visual. Primero, el sistema es más sensible a cambios en la luminancia que a cambios en el tinte o la saturación; es decir, nuestra capacidad para discernir espacialmente la información de colores es más débil que nuestra capacidad para discriminar espacialmente información monocromática. Esta observación sugiere que deben usarse más bits de ancho de banda para representar  $Y$  que para la representación de  $I$  y  $Q$ , a fin de proporcionar mayor resolución en  $Y$ . Segundo, los objetos que cubren una parte muy pequeña de nuestro campo visual producen una sensación de color limitada, que puede especificarse adecuadamente con una y no con dos dimensiones de color. Este hecho sugiere que la señal  $I$  o  $Q$  puede tener menor ancho de banda que la otra. La codificación NTSC de YIQ a una señal de transmisión usa estas propiedades para maximizar la cantidad de información transmitida en un ancho de banda fijo: se asigna 4 MHz a  $Y$ , 1.5 MHz a  $I$  y 0.6 MHz a  $Q$ . En [SMIT78; PRIT77] puede hallar más detalles acerca del modelo YIQ.

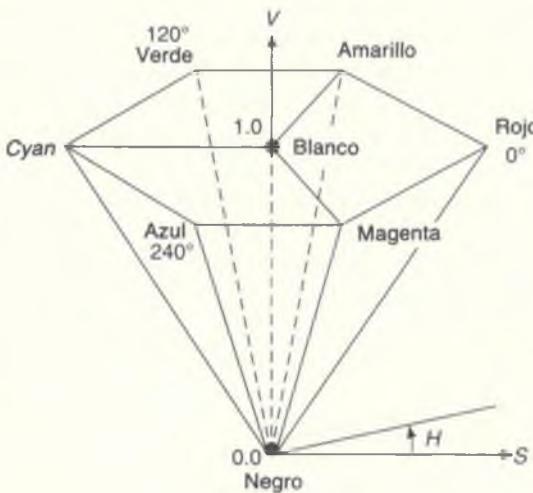
#### 11.3.4 Modelo de colores HSV

Los modelos RGB, CMY y YIQ están orientados al hardware. En cambio, el modelo HSV (*hue, saturation, value; tinte, saturación, valor*) de Smith [SMIT78] (también conocido como modelo HSB, donde  $B$  representa la brillantez) está orientado al usuario y se basa en el atractivo intuitivo del modelo de los

artistas: tinta, matiz y tono. El sistema de coordenadas es cilíndrico, y el conjunto del espacio donde está definido el modelo es un cono hexagonal o pirámide de seis lados, como se ilustra en la figura 11.19. La parte superior del cono hexagonal corresponde a  $V = 1$ , que contiene los colores relativamente brillantes. Sin embargo, *no* todos los colores del plano  $V = 1$  se perciben de la misma brillantez.

El tinte, o  $H$  (*hue*), se mide con el ángulo alrededor del eje vertical, con el rojo en  $0^\circ$  y el verde en  $120^\circ$ , etcétera (véase la Fig. 11.19). En el modelo HSV los colores complementarios están opuestos  $180^\circ$  entre sí. El valor de  $S$  es la razón que varía de 0 en la línea central (eje  $V$ ) a 1 en los lados triangulares del cono hexagonal. La saturación se mide con respecto a la gama de colores representada por el modelo, la cual, por supuesto, constituye un subconjunto de todo el diagrama de cromaticidad CIE. Por lo tanto, una saturación del 100 por ciento en el modelo es menor que el 100 por ciento de pureza de excitación.

El cono hexagonal tiene una unidad de altura en  $V$ , con el ápice en el origen. El punto en el ápice es negro y su coordenada  $V$  es cero. En este punto, los valores de  $H$  y  $S$  son irrelevantes. El punto en  $S = 0$ ,  $V = 1$  es blanco. Los valores intermedios de  $V$  para  $S = 0$  (en la línea central) son los grises. Cuando  $S = 0$ , el valor de  $H$  es irrelevante (llamado, por convención INDEFINIDO). Cuando  $S$  es distinto de cero,  $H$  es relevante. Por ejemplo, el rojo puro está en  $H = 0$ ,  $S = 1$ ,  $V = 1$ . De hecho, cualquier color con  $V = 1$ ,  $S = 1$  equivale a un pigmento puro que usa un artista como punto de partida para mezclar colores. La adición de pigmento blanco corresponde a la reducción de  $S$  (sin alterar  $V$ ). Creamos matices manteniendo  $S = 1$  y reduciendo  $V$ . Los tonos se crean reduciendo  $H$ .



**Figura 11.19** Modelo de colores HSV de un cono hexagonal. El plano  $V = 1$  contiene los planos  $R = 1$ ,  $G = 1$  y  $B = 1$  del modelo RGB en las regiones indicadas.



**Figura 11.20**  
Cubo de colores RGB visto a lo largo de la diagonal principal. Las aristas visibles del cubo son sólidas; las aristas invisibles están punteadas.

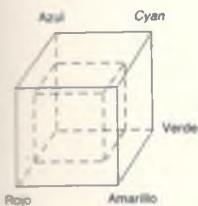
ciendo tanto  $S$  como  $V$ . Por supuesto, un cambio en  $H$  corresponde a la selección del color puro con el cual se comienza. Por lo tanto,  $H$ ,  $S$  y  $V$  corresponden a conceptos del sistema de colores de un artista y no son exactamente iguales a los términos presentados en la sección 11.2.

La parte superior del cono hexagonal HSV corresponde a la proyección que podemos ver a lo largo de la diagonal principal del cubo de colores RGB, del blanco hacia el negro, como se ilustra en la figura 11.20. El cubo RGB tiene subcubos, como se muestra en la figura 11.21. Cada subcubo, al observarse por su diagonal principal, es como el hexágono de la figura 11.20, excepto que es más pequeño. Cada plano de  $V$  constante en el espacio HSV corresponde a una de estas vistas de un subcubo en el espacio RGB. La diagonal principal del espacio RGB se convierte en el eje  $V$  del espacio HSV. Por lo tanto, podemos ver de manera intuitiva la correspondencia entre RGB y HSV. Los algoritmos de los programas 11.1 y 11.2 definen la correspondencia de manera precisa, ofreciendo conversiones de un método a otro.

**Programa 11.1**  
Algoritmo para convertir  
del espacio RGB al  
espacio en color HSV.

```
void RGB_a_HSV (float r, float g, float b, float *h, float *s, float *v)
{
    /* Datos proporcionados: r, g, b, cada uno en [0, 1].
       Resultado deseado: h en [0, 360], s y v en [0, 1]; si s = 0, entonces h = INDEFINIDO, que
       es alguna constante definida con un valor fuera del intervalo [0, 360] */
    float máx, min, delta;

    máx = MÁX (r, g, b);
    min = MÍN (r, g, b);
    *v = máx;                                /* Este es el valor v */
    /* Calcular ahora la saturación, s. */
    if (máx != 0.0)
        *s = (máx - min) / máx;                /* s es la saturación */
    else
        *s = 0.0;                               /* La saturación es 0 si el rojo, el verde y el azul son iguales a 0 */
    if (*s == 0.0) {
        *h = INDEFINIDO;
        return;
    }
    /* Caso cromático: saturación distinta de 0; determinar tinte */
    delta = máx - min;
    if (r == máx)                                /* El color resultante está entre el magenta y el cyan */
        *h = (g - b) / delta;                    /* El color resultante está entre el amarillo y el magenta */
    else if (g == máx)
        *h = 2.0 + (b - r) / delta;              /* El color resultante está entre el cyan y el amarillo */
    else if (b == máx)
        *h = 4.0 + (r - g) / delta;
    *h *= 60.0;                                  /* Convertir tinte a grados */
    if (*h < 0.0)
        *h += 360.0;                            /* Asegurar que el tinte no sea negativo */
}
```



**Figura 11.21**  
Cubo RGB y un subcubo.

**Programa 11.2**

*Algoritmo para convertir  
del espacio de colores  
HSV al espacio RGB.*

```

void HSV_a_RGB (float *r, float *g, float *b, float h, float s, float v)
{
/* Datos proporcionados: h en [0, 360] o INDEFINIDO, s y v en [0, 1].
   Resultado deseado: r, g, b, cada uno en [0, 1]. */
  float f, p, q, t;
  int i;

  if (s == 0.0) { /* El color está sobre la línea central de blanco y negro */
    if (h != INDEFINIDO) /* Color acromático: no hay tinte */
      error (); /* Por nuestra convención, hay error si s = 0 y h tiene valor. */
      return;

    /*r = v;
    *g = v;
    *b = v;
    return;

  /* Color cromático: s ≠ 0, por lo que hay tinte */
  if (h == 360.0) /* 360° equivale a 0° */
    h = 0.0;
  h /= 60.0; /* h está ahora en [0, 6] */
  i = floor(h); /* floor devuelve el mayor entero ≤ h */
  f = h - i; /* f es la parte fraccionaria de h */

  p = v * (1 - s);
  q = v * (1 - s * f);
  t = v * (1 - s * (1 - f));

  switch (i) {
    case 0:
      *r = v;
      *g = t;
      *b = p;
      break;

    case 1:
      *r = q;
      *g = v;
      *b = p;
      break;

    case 2:
      *r = p;
      *g = v;
      *b = t;
      break;

    case 3:
      *r = p;
      *g = q;
      break;
  
```

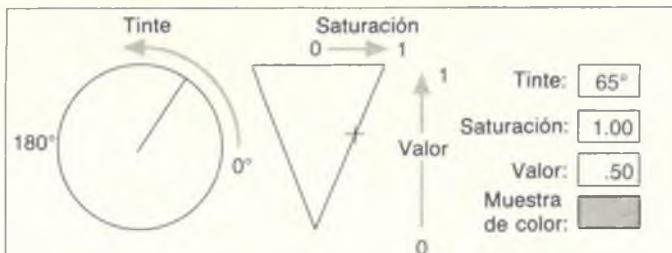
```
*b = v;  
break;  
  
case 4:  
    *r = t;  
    *g = p;  
    *b = v;  
    break;  
  
case 5:  
    *r = v;  
    *g = p;  
    *b = q;  
    break;
```

### 11.3.5 Especificación interactiva del color

Muchos programas de aplicación permiten al usuario especificar los colores de áreas, líneas, texto, etcétera. Si sólo se proporciona un pequeño conjunto de colores, la selección desde un menú de los colores disponibles es apropiada. Sin embargo, ¿qué sucede si el conjunto de colores es mayor que lo que puede presentarse razonablemente en un menú?

Las opciones básicas son usar nombres para los colores, especificar las coordenadas numéricas del color en el espacio de colores (ya sea tecleando los valores o usando controles deslizantes) o interactuar directamente con una representación visual del espacio de colores. La asignación de nombres por lo general no es satisfactoria, ya que resulta ambigua y subjetiva (azul marino claro con un toque de verde) y también es la antítesis de la interacción gráfica. Por otra parte, [BERK82] describe CNS, un esquema de nomenclatura de colores bastante bien definido que usa términos como amarillo verdoso, verde-amarillo y verde amarillento para distinguir tres tintes entre el verde y el amarillo. En un experimento, los usuarios de CNS pudieron especificar colores con mayor precisión que los usuarios que registraron coordenadas numéricas en el espacio RGB o HSV.

La especificación de coordenadas se puede efectuar con controles deslizantes, usando cualquiera de los modelos de colores. Si el usuario comprende la forma en que cada dimensión afecta al color, obtendrá buenos resultados con esta técnica. Quizás el mejor método interactivo de especificación de coordenadas sea permitir que el usuario interactúe directamente con una representación del espacio de colores, como se muestra en la figura 11.22. La línea en el círculo (que representa al plano  $V = 1$ ) se puede arrastrar alrededor del círculo para determinar la porción del volumen HSV que se presentará en el triángulo. El cursor en el triángulo se puede mover para especificar la saturación y el valor. Al mover la línea o el cursor cambian de valor las lecturas numéricas. Cuando el usuario teclea directamente valores nuevos en estas lecturas, cambia la posi-



**Figura 11.22** Una forma conveniente de especificar los colores en el espacio HSV. La saturación y el valor se indican con el cursor en el área triangular; el tinte con la línea en el área circular. El usuario puede mover los indicadores de línea y cursor en los diagramas, con lo cual se actualizan las lecturas numéricas. Alternativamente, el usuario puede teclear nuevos valores, con lo cual cambiarán los indicadores. También podrían agregarse controles deslizantes para  $H$ ,  $S$  y  $V$ , proporcionando al usuario un control preciso sobre una dimensión a la vez, sin tener que teclear valores.

ción de la línea y el cursor. El recuadro de muestra del color presenta el color seleccionado. Sin embargo, la forma en que una persona percibe un color se ve afectada por los colores circundantes y el tamaño de las áreas coloreadas; por consiguiente, es probable que la percepción del color en el área de realimentación difiera de la percepción del color en la pantalla. Es importante que el usuario también vea la pantalla al ajustar los colores.

### 11.3.6 Interpolación en el espacio de colores

La interpolación de colores es necesaria al menos en tres situaciones: para el sombreado de Gouraud (Sec. 14.2.4), para la eliminación de artefactos de discretización (Sec. 3.14) y para el mezclado de dos imágenes, por ejemplo, en el caso de una secuencia de presentación por desvanecimiento (*fade-in*, *fade-out*). Los resultados de la interpolación dependen del modelo de colores que se usa para la interpolación; debemos tener cuidado y seleccionar un modelo apropiado.

Si la conversión de un modelo de colores a otro transforma una línea recta (que representa la trayectoria de interpolación) en un modelo de colores a otra línea recta en el otro modelo, entonces los resultados de la interpolación lineal en ambos modelos serán iguales. Esta situación se aplica a los modelos de colores RGB, CMY, YIQ y CIE, los cuales están relacionados por transformaciones afines simples. Sin embargo, una línea recta en el modelo RGB generalmente no se transforma en una línea recta en el modelo HSV. En la ilustración en color 18 se muestra el resultado de la interpolación lineal entre los mismos dos colores en los espacios HSV, RGB y YIQ. Considere la interpolación entre rojo y verde. En RGB, rojo = (1, 0, 0) y verde = (0, 1, 0). Su interpolación (con ambas ponderaciones iguales a 0.5 por conveniencia) es (0.5, 0.5, 0). Al aplicar a este resultado el algoritmo RGB\_a\_HSV (Prog. 11.1) se obtiene (60°, 1, 0.5). Ahora, si representamos el rojo y el verde en HSV, tenemos (0°, 1, 1) y (120°,

1, 1), pero si interpolamos con ponderaciones iguales en HSV, tenemos  $(60^\circ, 1, 1)$ ; por lo tanto, el valor difiere en 0.5 con respecto a la misma interpolación en RGB.

Como segundo ejemplo, considere la interpolación del rojo y el *cyan* en los modelos RGB y HSV. En RGB, comenzamos con (1, 0, 0) y (0, 1, 1) respectivamente, e interpolamos a (0.5, 0.5, 0.5), que en HSV se interpreta como (INDEFINIDO, 0, 0.5). En HSV, el rojo y el *cyan* son  $(0^\circ, 1, 1)$  y  $(180^\circ, 1, 1)$ . Al interpolar obtenemos  $(90^\circ, 1, 1)$ ; se ha incluido un nuevo tinte con valor y saturación máximos, mientras que el resultado *correcto* de la combinación de cantidades iguales de colores complementarios es un valor de gris. Una vez más, se obtienen resultados distintos al interpolar y luego transformar que si primero se transforma y después se interpola.

Para el sombreado de Gouraud se puede emplear cualquiera de los modelos, ya que los dos interpolantes usualmente están tan cerca que las trayectorias de interpolación entre los colores también son próximas. Al mezclar dos imágenes —como en una secuencia de desvanecimiento o en la eliminación de artefactos de discretización—, los colores pueden ser distantes y es apropiado usar un modelo aditivo, como RGB. Por otra parte, si el objetivo es interpolar entre dos colores de tinte (o saturación) fijo y mantener este tinte (o saturación) fijo para todos los colores interpolados, entonces es preferible usar HSV.

## 11.4 Utilización del color en la graficación por computador

Utilizamos el color por estética, para establecer un ambiente o estado de ánimo, por cuestiones de realismo, como realce, para identificar la relación entre áreas y para codificar. Si tenemos cuidado, el color se puede emplear efectivamente para todos estos fines. Además, los usuarios tienden a preferir el color, aunque no exista ninguna evidencia cuantitativa de que su empleo mejore el rendimiento.

Si el color se utiliza de manera descuidada, la presentación puede ser menos atractiva o útil que la correspondiente presentación monocromática. En un experimento, la inclusión de un color sin sentido redujo el rendimiento de los usuarios a una tercera parte de lo que era sin color [KREB79]. El color debe emplearse de manera conservadora. Cualquier aplicación decorativa del color debe depender de la utilización funcional, de manera que el color no pueda interpretarse incorrectamente como algo que tiene un significado subyacente. Por lo tanto, la utilización del color, como los demás aspectos de la interfaz usuario-computador, debe ensayarse con usuarios reales para identificar y remediar los problemas. Por supuesto, las personas tienen distintas preferencias, por lo cual una práctica común es ofrecer valores por omisión con base en las reglas de utilización de colores y proporcionar un medio que permita a los usuarios cambiar estos valores por omisión. Una estrategia conservadora para seleccionar colores es diseñar primero para una presentación monocromática, a fin de asegurar que la utilización de color sea exclusivamente redundante. Con esta opción se evitan los problemas que tienen ciertos usuarios con deficiencias en la

percepción de colores, y permite además que la aplicación se ejecute en una pantalla monocromática. Las opciones de colores que se utilizan en administradores de ventanas, como las que se muestran en las ilustraciones en color 9 y 10, muchas veces son conservadoras. El color no se usa como un código único para el estado del botón, el elemento seleccionado en un menú, etcétera.

Se han escrito muchos libros acerca del empleo del color con fines estéticos, incluyendo [BIRR61]; aquí enunciaremos sólo algunas de las reglas más sencillas que ayudan a producir armonía de colores. La regla fundamental de la estética de colores es seleccionar éstos de acuerdo con algún método, generalmente haciendo un recorrido por una trayectoria suave en un modelo de colores o restringiendo los colores a planos o conos hexagonales en un espacio de colores. Esta pauta puede especificar colores de claridad o valor constante. Así misma, la mejor forma de espaciar los colores es con distancias *perceptivas* iguales, que no es lo mismo que usar incrementos de espaciado igual para las coordenadas, lo que puede ser difícil de implantar. Recuerde también que la interpolación lineal (como en el sombreado de Gouraud) entre dos colores produce distintos resultados en espacios de colores diferentes (véase el Ejer. 11.6 y la Ilust. color 18).

La selección aleatoria de diferentes tintes y saturaciones pocas veces produce buenos resultados. Alvy Ray Smith realizó un experimento informal en el cual una malla de  $16 \times 16$  se llenó con colores generados en forma aleatoria. Como era de esperar, la malla no resultó muy atractiva. Sin embargo, la apariencia mejoró notablemente al ordenar los 256 colores de acuerdo con sus valores *H*, *S* y *V* y presentarlos en forma ordenada en la malla.

Algunos casos más específicos de estas reglas revelan que si un diagrama contiene únicamente unos cuantos colores, entonces el complemento de uno de los colores debe usarse como fondo. En el caso de una imagen que contiene muchos colores diferentes, debe usarse un fondo neutro (gris) ya que es armonioso y no resalta. Si dos colores adyacentes no son muy armoniosos, se puede utilizar un delgado borde negro para separarlos. Esta forma de utilizar bordes también es efectiva en el canal visual acromático (blanco y negro), ya que el contorno negro facilita la detección de formas. Algunas de estas reglas están codificadas en ACE (*A Color Expert*, Experto en colores), un sistema experto para seleccionar colores que se usarán en interfaces con los usuarios [MEIER88]. En términos generales, es conveniente minimizar el número de colores que se usan (excepto para el sombreado de imágenes realistas).

El color también se puede usar para codificar información, como se muestra en la ilustración en color 20. Sin embargo, es necesario hacer ciertas advertencias. En primer lugar, los códigos de colores pueden tener significados planeados. Por ejemplo, si presentamos las ganancias de la compañía A en rojo y las de la compañía B en verde, esto puede dar a entender que la compañía A tiene problemas financieros, ya que hay una tendencia a asociar el rojo con los déficit financieros. Los colores brillantes y saturados se destacan más que los apagados y pálidos, por lo que pueden dar un énfasis no intencionado. Los elementos que tengan el mismo color en una presentación pueden considerarse relacionados por el mismo código de color, aunque no lo estén.

Este último problema se presenta con frecuencia cuando se usa el color para agrupar elementos de menú y para distinguir elementos de presentación, como son capas diferentes en una tablilla de circuitos impresos o en una pastilla VLSI; por ejemplo, los usuarios tienden a relacionar los elementos presentados en verde con los elementos de menú del mismo color. Esta tendencia es una de las razones por las cuales debe limitarse la utilización de color en elementos de interfaz con el usuario, como menús, recuadros de diálogo y bordes de ventanas. (Otra razón es dejar el mayor número posible de colores libres para que puedan ser usados por el programa de aplicación.)

Varias de las reglas de utilización de colores se basan en aspectos fisiológicos, no estéticos. Por ejemplo, el ojo es más sensible a la variación espacial de la intensidad que a la variación en cromaticidad, de manera que las líneas, el texto y otros detalles finos deben variar con respecto al fondo no sólo en cromaticidad, sino también en brillo (intensidad percibida), sobre todo en el caso de colores que contienen azul, ya que son relativamente pocos los conos en el ojo sensibles al azul. Por ello, no estará bien definida la orilla entre dos áreas coloreadas con la misma brillantez, que sólo difieren en cuanto a la cantidad de azul. Por otra parte, los conos sensibles al azul tienen mayor distribución sobre la retina que los conos sensibles al rojo y al verde, de manera que nuestra visión periférica es mejor para el azul; es por esto que las luces de las patrullas de policía son ahora azules y no rojas.

El azul y el negro difieren muy poco en lo que se refiere a brillantez, por lo que no son una combinación muy buena. En forma parecida, es difícil distinguir el amarillo del blanco, ya que ambos colores son brillantes. En la ilustración en color 10 se muestra el uso eficaz del amarillo para realzar texto negro sobre fondo blanco. El amarillo contrasta con el texto negro y también sobresale. Así mismo, el realce amarillo no es tan llamativo como un realce en negro con texto inverso (es decir, el texto realzado en blanco sobre un realce negro), de uso común en pantallas monocromáticas.

El texto blanco sobre fondo azul proporciona un buen contraste, menos severo que el blanco sobre negro. Es conveniente evitar los rojos y verdes con poca saturación y luminancia, ya que estos colores son los que confunden las personas con daltonismo, la forma más común de deficiencia en la percepción de colores. Meyer y Greenberg describen formas efectivas de seleccionar colores para personas que no pueden distinguir entre colores [MEYE88].

El ojo humano no puede distinguir el color de objetos muy pequeños, como ya mencionamos al analizar el modelo de colores YIQ NTSC, de manera que la codificación por colores no debe aplicarse a objetos pequeños. Específicamente, la determinación del color de objetos con subtensión menor que 20 a 40 minutos de arco es susceptible de errores [BISH60, HAEU76]. Un objeto de 0.25 cm, visto a unos 60 cm (una distancia de observación típica) subtende menos que este arco, que corresponde a unos 7 pixeles de altura en una pantalla de 1024 líneas y altura vertical de 15 pulgadas. Es obvio que resulta difícil discernir el color de un pixel (véase el Ejer. 11.10).

El color percibido de un área coloreada es afectado por el color del área que la rodea; este efecto es muy problemático si se usan colores para codificar información. El efecto se minimiza cuando las áreas circundantes son un matiz de gris o un color poco saturado.

El color de un área puede afectar el tamaño percibido del área. Cleveland y McGill descubrieron que un cuadrado rojo se percibe de mayor tamaño que uno verde del mismo tamaño [CLEV83]. Este efecto puede ocasionar que el observador asigne mayor importancia al cuadrado rojo que al verde.

Si un usuario observa detenidamente durante varios segundos un área de color muy saturado y luego dirige su mirada a otra parte, verá un remanente de la imagen. Este efecto es desconcertante y cansa la vista. Por lo tanto, no es recomendable utilizar grandes áreas de colores saturados. Además, las áreas vastas de colores distintos pueden dar la apariencia de estar a distancias diferentes del observador, ya que el índice de refracción de la luz depende de la longitud de onda. El ojo cambia su enfoque cuando la visión del observador pasa de un área coloreada a otra, y este cambio de enfoque da la impresión de variaciones en la profundidad. El rojo y el azul, que se encuentran en extremos opuestos del espectro, tienen el mayor efecto de disparidad de profundidad: el rojo aparece más cercano y el azul más lejano. Por consiguiente, no se recomienda usar el azul para objetos en primer plano si el fondo es rojo; no hay problema a la inversa.

Por todos estos peligros y problemas del empleo de colores, ¿le sorprende ahora que una de las primeras reglas que establecimos fue usar el color en forma conservadora?

## RESUMEN

La importancia del color en la graficación por computador seguirá aumentando conforme los monitores en color y los dispositivos de impresión en color constituyan la norma en muchas aplicaciones. En este capítulo hemos presentado los conceptos del color más relevantes para la graficación por computador; si desea más información, consulte la vasta literatura sobre el color, como [BILL81; BOYN79; GREG66; HUNT87; JUDD75; WYSZ82]. En [FROM84; MARC84; MEIE88; MURC85] puede encontrar más información relacionada con los aspectos artísticos y estéticos de la utilización del color en la graficación por computador. Los difíciles problemas de calibrar monitores con precisión e igualar los colores que aparecen en pantalla con los impresos se analizan en [COWA88; STON88].

## Ejercicios

11.1 Obtenga una ecuación para el número de intensidades que se pueden presentar con patrones de pixeles de  $m \times m$ , donde cada pixel tiene  $w$  bits.

11.2 Escriba un algoritmo para presentar un arreglo de pixeles en un dispositivo de salida de dos niveles. Las entradas del algoritmo son un arreglo de  $m$  de intensidades de pixeles, con  $w$  bits por pixel, y una matriz de  $n \times n$ .

secuencia de crecimiento. Suponga que el dispositivo de salida tiene resolución de  $m \times n \times m \times n$ .

11.3 Escriba un algoritmo para presentar un polígono relleno en un dispositivo de dos niveles usando un patrón de relleno de  $n \times n$ .

11.4 Cuando se usan ciertos patrones para llenar un polígono que se presenta en una pantalla de trama entrelazada, todos los bits *encendidos* caen en las líneas de barrido pares o impares, lo que ocasiona un poco de parpadeo. Revise el algoritmo del ejercicio 11.3 para permutar las filas del patrón de  $n \times n$  de manera que las duplicaciones alternas del patrón alternen la utilización de las líneas de barrido pares e impares. En la figura 11.23 se presentan los resultados que se obtienen al usar el nivel de intensidad 1 de la figura 11.4 con y sin esta alternación.

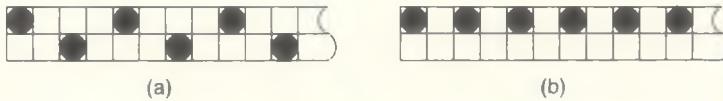
11.5 Grafique el lugar correspondiente a los puntos de los valores de luminancia constante 0.25, 0.50 y 0.75, definidos por  $Y = 0.299R + 0.587G + 0.114B$ , en el cubo RGB y el cono hexagonal HSV.

11.6 Exprese, en función de  $R$ ,  $G$  y  $B$ , la  $I$  de YIQ y la  $V$  de HSV. Note que  $I$  y  $V$  no son lo mismo.

11.7 Analice el diseño de una pantalla de barrido que usa HSV, en lugar de RGB, como especificación de colores.

11.8 Reescriba el algoritmo de conversión HSV a RGB para que sea más eficaz. Reemplace los enunciados de asignación para  $p$ ,  $q$  y  $t$  con  $vs = v * s$ ;  $vsf = vs * f$ ;  $P = v - vs$ ;  $q = v - vsf$ ;  $t = p + vsf$ . Suponga también que  $R$ ,  $G$  y  $B$  se encuentran en el intervalo  $[0, 255]$  y vea cuántos cálculos se pueden convertir a enteros.

11.9 Escriba un programa que presente, lado a lado, dos mallas de  $16 \times 16$ . Llene cada malla con colores. La malla de la izquierda tendrá 256 colores elegidos al azar del espacio de colores HSV (éste se crea usando un generador de números aleatorios para elegir uno de 10 valores igualmente espaciados de  $H$ ,  $S$



**Figura 11.23**

Resultados obtenidos cuando el nivel de intensidad 1 de la figura 11.4 se usa de dos maneras: (a) con alternancia (los pixeles intensificados están en ambas líneas de rastreo) y (b) sin alternancia (todos los pixeles intensificados están en la misma línea de rastreo).

y  $V$ ). La malla de la derecha contiene los mismos 256 colores, ordenados acuerdo con  $H$ ,  $S$  y  $V$ . Experimente con los resultados, variando  $H$ ,  $S$  y  $V$  como clave primaria para el ordenamiento.

11.10 Escriba un programa para presentar sobre un fondo gris pequeños cuadrados con colores naranja, rojo, verde, azul, *cyan*, magenta y amarillo. Cada cuadrado está separado de los otros y su tamaño es de  $n \times n$  pixeles, donde  $n$  es una variable de entrada. ¿Cuál debe ser el tamaño de  $n$  para que el usuario pueda determinar sin ambigüedades los colores de cada cuadrado, a una distancia de 60 y 120 cm? ¿Cuál debe ser la relación entre los dos valores de  $n$ ? ¿Qué efecto, si lo hay, tienen los distintos colores de fondo en este resultado?

11.11 Calcule el número de bits de precisión de la tabla de consulta necesaria para almacenar 256 niveles de intensidad para intervalos de intensidad dinámica de 50, 100 y 200.

11.12 Escriba un programa para interpolar linealmente entre dos colores en RGB y HSV. Acepte los dos colores como entrada, permitiendo que se especifiquen en cualquiera de los modelos.

En los capítulos anteriores analizamos técnicas gráficas que comprenden primitivas simples en dos y tres dimensiones. Las imágenes que produjimos, como las casas alambradas del capítulo 6, representan objetos que en la vida real son mucho más complejos tanto en estructura como en apariencia. En este capítulo presentaremos una aplicación cada vez más importante de la graficación por computador: la creación de imágenes realistas de escenas tridimensionales.

¿Qué es una imagen *realista*? El sentido en que se puede afirmar que una imagen —ya sea pintada, fotografiada o generada por computador— es *realista* constituye el tema de muchos debates entre investigadores [HAGE85]. Aquí usaremos el término de manera bastante amplia para referirnos a una imagen que captura muchos de los efectos de la luz que interactúa con los objetos reales. De esta manera, tratamos las imágenes realistas como un continuo y hablamos libremente de las imágenes y las técnicas utilizadas para crearlas, como *más* o *menos* realistas. En un extremo del continuo están ejemplos de lo que a menudo se denomina **realismo fotográfico** (o **fotorrealismo**). Estas imágenes tratan de sintetizar el campo de las intensidades de luz que se enfocarían en el plano de la película de una cámara orientada a los objetos. Conforme nos acercamos al otro extremo del continuo, hallamos imágenes que presentan cada vez menos de los aspectos visuales que analizaremos.

Hay que tener presente que una imagen más realista no siempre es la más deseable o útil. Si el objetivo final de una imagen es transmitir información, entonces una imagen libre de las complicaciones de las sombras y los reflejos puede ser mejor que una con realismo fotográfico. Además, en muchas de las aplicaciones de las técnicas que se describen en los capítulos siguientes, se altera

la realidad para lograr efectos estéticos o cumplir con las expectativas de observador cándido. Estas técnicas se aplican por las mismas razones que películas de ciencia-ficción presentan sonidos de explosiones en el espacio exterior, algo que es imposible en el vacío. Por ejemplo, al presentar Urano en la ilustración en color 23, Blinn proyectó una luz adicional en el lado nocturno del planeta y aumentó el contraste para que todas las características fueran visibles simultáneamente; el lado nocturno habría estado a oscuras en caso contrario. ¿Si nos tomamos libertades con la física podemos obtener imágenes atractivas memorables y útiles?

La creación de imágenes realistas comprende varias etapas que se tratan con detalle en los capítulos siguientes. Aunque estas etapas muchas veces se consideran como parte de un ducto conceptual, puede variar el orden de su ejecución, como veremos, dependiendo de los algoritmos que se empleen. La primera etapa genera modelos de los objetos, usando los métodos analizados en los capítulos 9 y 10. Despues se selecciona una especificación de visualización (como se desarrolló en el capítulo 6) y las condiciones de iluminación. Luego se determinan las superficies visibles al observador usando los algoritmos presentados en el capítulo 13. El color que se asigna a cada pixel de la proyección de una superficie visible es una función de la luz reflejada y transmitida por los objetos y se determina con los métodos analizados en el capítulo 14. La imagen resultante se puede combinar con otras generadas previamente (p. ej., para reutilizar fondo complejo) usando técnicas de composición. Por último, si estamos produciendo una secuencia animada, debemos definir cambios variables en el tiempo de los modelos, la iluminación y las especificaciones de visualización. El proceso de creación de imágenes a partir de modelos con frecuencia se denomina **generación de imágenes (rendering)**. El término **generación de barrido (rasterización)** también se usa para referirse específicamente a aquellos pasos que comprenden la determinación de valores de píxeles a partir de primitivas geométricas.

En este capítulo se presenta la generación de imágenes realistas desde varias perspectivas. En primer lugar veremos algunas de las aplicaciones de las imágenes realistas. Después examinaremos, en una secuencia aproximadamente histórica, una serie de técnicas que hacen posible la creación de imágenes cada vez más realistas. Cada técnica se ilustra con la imagen de una escena estándar aplicando a ella la nueva técnica. Por último, concluimos con sugerencias acerca de cómo estudiar los capítulos siguientes.

## 12.1 ¿Por qué el realismo?

La creación de imágenes realistas es una meta importante en campos como la simulación, el diseño, el entretenimiento y la publicidad, la investigación, la educación, y el mando y control.

Los sistemas de simulación presentan imágenes que no sólo son realistas, sino que también cambian en forma dinámica. Por ejemplo, un simulador de vuelo presenta la vista que se vería desde la cabina de un avión en movimiento. Para producir el efecto de movimiento, el sistema genera y presenta una vista nueva, ligeramente distinta, varias veces por segundo. Se han utilizado simuladores para entrenar a pilotos de naves espaciales, aeroplanos, barcos y, en fechas más recientes, conductores de automóviles.

Los diseñadores de objetos tridimensionales como automóviles, aviones y edificios quieren saber cómo se verán sus diseños preliminares. Con frecuencia, la creación de imágenes realistas generadas por computador es una forma más fácil, menos costosa y más efectiva de ver los resultados preliminares, en lugar de construir modelos y prototipos; además, permite considerar diseños alternativos. Si el trabajo de diseño también está basado en un computador, es probable que exista una descripción digital del objeto que pueda usarse para crear las imágenes. En teoría, el diseñador puede también interactuar con la imagen presentada para modificar el diseño. Se han desarrollado sistemas de diseño de automóviles para determinar cómo se verá un vehículo en diversas condiciones de iluminación. Con frecuencia, los gráficos realistas se acoplan con programas que analizan otros aspectos del objeto que se diseña, como sus propiedades de masa o su respuesta a la tensión.

Las imágenes generadas por computador se usan mucho en el mundo del entretenimiento, tanto en las caricaturas animadas tradicionales como en imágenes realistas y surrealistas para logotipos, anuncios y películas de ciencia-ficción. Las caricaturas animadas generadas por computador pueden imitar la animación tradicional, pero también pueden trascender las técnicas manuales introduciendo movimientos más complicados e imágenes más ricas o reales. Algunas imágenes realistas complejas se pueden producir a menor costo que si se filmaran a partir de modelos físicos de los objetos. Se han generado otras imágenes que habrían sido muy difíciles o imposibles de montar con modelos reales. El hardware y el software de propósito especial utilizados en el entretenimiento incluyen elaborados sistemas de pintura y sistemas en tiempo real para generar efectos especiales y combinar imágenes. Conforme mejora la tecnología, los video-juegos caseros y de centros de entretenimiento generan imágenes cada vez más realistas.

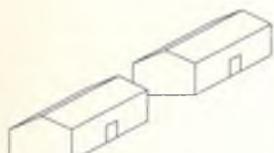
Las imágenes realistas se están convirtiendo en una herramienta indispensable para la investigación y la educación. Un ejemplo especialmente importante es el uso de gráficos en el modelado molecular, como se muestra en la ilustración en color 22. Es interesante ver cómo se amplía el concepto del realismo en este caso: las representaciones realistas no corresponden a átomos "reales", sino a modelos volumétricos de esferas y barras que permiten crear con ellos estructuras mayores que las que serían factibles con modelos físicos y que además permiten efectos especiales, como enlaces vibratorios animados y cambios de color que correspondan a reacciones. En una escala macroscópica, los filmes realizados en JPL muestran misiones de sondas espaciales de la NASA, como se presenta en la ilustración en color 23.

Otra aplicación de las imágenes realistas es en las áreas de mando y control donde el usuario debe estar informado acerca del complejo proceso que representa la imagen y debe ser capaz de controlarlo. A diferencia de las simulaciones, que pretenden imitar lo que vería y sentiría el usuario en la situación simulada, las aplicaciones de mando y control muchas veces generan pantallas simbólicas que destacan ciertos datos y suprimen otros para ayudar en la toma de decisiones.

## 12.2 Dificultades fundamentales

Una dificultad fundamental para lograr el realismo visual total es la complejidad del mundo real. Observe la riqueza de su entorno. Hay muchas texturas superficies, sutiles gradientes de colores, sombras, reflejos y pequeñas irregularidades en los objetos circundantes. Piense en los patrones de una tela arrugada, la textura de la piel, el cabello revuelto, marcas de pisadas en el piso y la pintura cuarteadas en la pared. Todos se combinan para crear una experiencia visual real. El costo computacional de simular estos efectos puede ser muy elevado; algunas de las imágenes que se presentan en las ilustraciones en color requieren varios minutos o incluso horas para generarse en computadores muy poderosos.

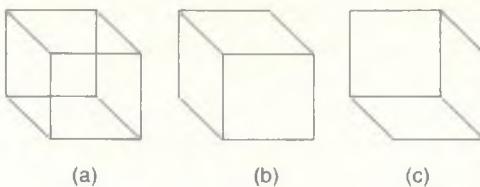
Un subobjetivo que es más fácil de alcanzar en la búsqueda del realismo es proporcionar información suficiente para permitir que el observador comprenda las relaciones espaciales tridimensionales entre varios objetos. Este subobjetivo se puede lograr a un costo mucho menor y es un requisito común en CAD y en muchas otras áreas de aplicación. Aunque las imágenes muy realistas muestran las relaciones espaciales tridimensionales, por lo general muestran mucha más información. Por ejemplo, en la figura 12.1, un sencillo dibujo de líneas basta para convencernos de que un edificio está parcialmente detrás del otro. No es necesario presentar las superficies de los edificios rellenadas con tejas o ladrillos ni las sombras producidas por los edificios. De hecho, en algunos contextos estos detalles adicionales pueden distraer la atención del observador respecto a otra información más importante.



**Figura 12.1**  
Dibujo lineal de dos casas.

Un problema constante en la representación de relaciones espaciales es que la mayoría de los dispositivos son bidimensionales. Por lo tanto, es necesario proyectar los objetos tridimensionales a dos dimensiones, lo que implica una pérdida considerable de información que en ocasiones puede causar ambigüedades en la imagen. Se pueden aplicar algunas de las técnicas presentadas en este capítulo para devolver información del tipo que normalmente existe en nuestro ambiente visual, de manera que nuestros mecanismos de percepción de profundidad resuelvan correctamente las ambigüedades.

Considere la ilusión del cubo de Necker que se presenta en la figura 12.2. Es una proyección bidimensional de un cubo; no sabemos si representa el cubo de la parte (b) o el de la parte (c) de la misma figura. De hecho, el usuario percibe

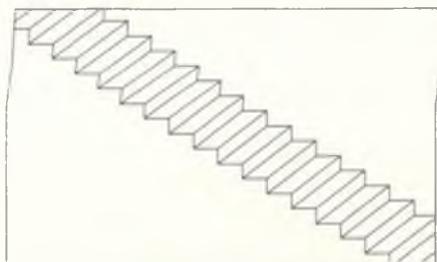


**Figura 12.2** Ilusión del cubo de Necker. ¿Está el cubo orientado como el que se presenta en (b) o como el que aparece en (c)?

“alternar” fácilmente entre las opciones, ya que la figura 12.2(a) no contiene información visual suficiente para una interpretación no ambigua.

Si los observadores conocen más acerca del objeto que se presenta, más fácilmente pueden formar lo que Gregory denomina una **hipótesis del objeto** [GREG70]. En la figura 12.3 se presenta la ilusión de escalera de Schröder: ¿estamos viendo hacia abajo o hacia arriba por la escalera? Lo más probable es que elijamos la primera interpretación, quizás porque es más usual que veamos una escalera a nuestros pies que por encima de nuestra cabeza, de manera que *conocemos* más acerca de las escaleras vistas desde arriba. Sin embargo, con un poco de imaginación podemos visualizar la interpretación alternativa de la figura. No obstante, para la mayoría de los observadores la situación se invierte al parpadear, y una vez más la escalera aparece como una vista desde arriba. Por supuesto, esta ambigüedad se resolvería con contexto adicional, por ejemplo una persona de pie en los escalones.

En las secciones siguientes listaremos algunos de los pasos en el camino a las imágenes realistas. Este camino en realidad ha sido un conjunto de senderos retorcidos en lugar de una ruta recta, pero lo hemos linealizado por cuestiones de sencillez, ofreciendo una introducción puramente descriptiva al tratamiento detallado de los capítulos subsecuentes. Primero mencionaremos las técnicas aplicables a dibujos de línea estáticos. Estos métodos se centran en formas de



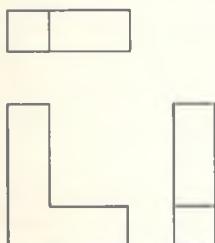
**Figura 12.3** Ilusión de la escalera de Schröder. ¿Se observa la escalera desde arriba o desde abajo?

presentar en una pantalla bidimensional las relaciones espaciales tridimensionales entre varios objetos. Después vienen las técnicas para las imágenes sombreadas, posibles gracias al hardware gráfico de barrido, que hacen hincapié en la interacción de los objetos y la luz. Luego se analizan los temas del aumento de la complejidad del modelo y la dinámica, aplicable tanto a imágenes de líneas y sombreadas. Finalmente, vemos las posibilidades de las imágenes tridimensionales verdaderas, los avances en el hardware de presentación y el lugar que ocupará en el futuro la generación de imágenes en el contexto de una síntesis interactiva total del ambiente.

## 12.3 Técnicas de generación para dibujos de línea

En esta sección nos centraremos en un subobjetivo del realismo: presentar relaciones espaciales tridimensionales en una superficie bidimensional. Para ello sirven las proyecciones geométricas planas definidas en el capítulo 6.

### 12.3.1 Vistas ortográficas múltiples



**Figura 12.4**

Proyecciones ortográficas frontal, superior y lateral de la letra de bloque "L".

Las proyecciones más fáciles de crear son las ortográficas paralelas, como vistas de planta y elevación, donde el plano de proyección es perpendicular al eje principal. Como se descarta la información de profundidad, es común mostrar juntas las vistas de planta y elevación, como ocurre con las vistas superior, frontal y lateral de una letra "L" de bloque de la figura 12.4. Este dibujo en particular no es difícil de comprender; sin embargo, la comprensión de los dibujos de ciertas piezas manufacturadas muy complicadas a partir de este tipo de vistas puede requerir muchas horas de estudio. El entrenamiento y la experiencia afinan nuestros poderes interpretativos, sin lugar a dudas, y la familiaridad con los tipos de objetos representados acelera nuestra formulación de la hipótesis correcta del objeto. Sin embargo, escenas tan complicadas como nuestra *cena estándar* de la ilustración en color 24 pueden ser muy confusas si sólo muestran con tres de estas proyecciones. Aunque es posible localizar sin ambigüedades un punto a partir de tres proyecciones ortográficas perpendiculares, los puntos y líneas múltiples se pueden ocultar al proyectarse de esta manera.

### 12.3.2 Proyecciones de perspectiva

En las proyecciones de perspectiva, el tamaño de un objeto se escala en proporción inversa a su distancia del observador. La proyección de perspectiva de un cubo que se presenta en la figura 12.5 refleja este escalamiento. Sin embargo, persisten las ambigüedades; la proyección podría ser el marco de un cuadro, la proyección paralela de una pirámide truncada o la proyección de perspectiva de un paralelepípedo con dos caras iguales. Si nuestra hipótesis del objeto es una pirámide truncada, el cuadrado más pequeño representa la cara más cercana al

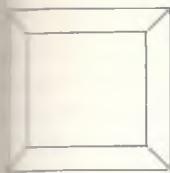


Figura 12.5  
Proyección de perspectiva  
de una casa.

observador; si la hipótesis del objeto es un cubo o un paralelepípedo rectangular, el cuadrado más pequeño corresponde a la cara que está más lejos del observador.

Nuestra interpretación de las proyecciones de perspectiva muchas veces se basa en la suposición de que un objeto más pequeño está más lejos. En la figura 12.6, podríamos suponer que la casa mayor está más cerca del observador. Sin embargo, la casa que parece de mayor tamaño (una mansión, quizás) posiblemente esté más lejos que la que parece más pequeña (una choza, por ejemplo), por lo menos mientras no tengamos otros indicadores, como árboles y ventanas. Cuando el observador sabe que los objetos proyectados tienen varias líneas paralelas, la perspectiva ayuda a transmitir la noción de profundidad, ya que las líneas paralelas parecen converger en sus puntos de fuga. Esta convergencia puede ser incluso un indicador de profundidad más poderoso que el efecto de reducción de tamaño. En la ilustración a color 25 se muestra una proyección de perspectiva de nuestra escena estándar.

### 12.3.3 Indicadores de profundidad

La profundidad (distancia) de un objeto se puede representar con la intensidad de la imagen: las partes de los objetos que deben aparecer más lejos del observador se presentan con menor intensidad. Este efecto se conoce como **indicación de profundidad**. La indicación de profundidad aprovecha el hecho de que los objetos más distantes se ven más tenues que los cercanos, sobre todo si hay bruma. Estos efectos pueden ser tan convincentes que los artistas se refieren a los cambios en intensidad (así como los cambios en textura, agudeza y color) utilizados para representar la distancia como **perspectiva aérea**. De esta manera, la indicación de profundidad se puede considerar como una versión simplificada de los efectos de la atenuación atmosférica.

En las pantallas vectoriales, la indicación de profundidad se implanta interpolando la intensidad del haz por el vector como función de sus coordenadas  $z$  inicial y final. Los sistemas gráficos en color generalmente utilizan la técnica para apoyar la interpolación entre el color de una primitiva y el color indicador de profundidad especificado por el usuario, que usualmente es el color del fondo.

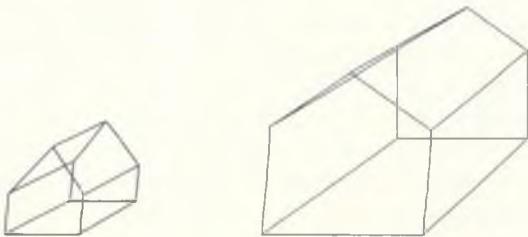


Figura 12.6 Proyección de perspectiva de dos casas.

Para limitar el efecto a una gama limitada de profundidades, PHIGS PLUS permite al usuario especificar los planos anterior y posterior de indicación de profundidad, entre los cuales se lleva a cabo la indicación. Un factor de escala separado, relacionado con cada plano, indica las proporciones del color original y el color indicador de profundidad que se usarán frente al plano anterior y detrás del plano posterior. El color de los puntos entre los planos se interpola linealmente entre estos dos valores. La resolución de intensidades del ojo es menor que su resolución espacial, de manera que la indicación de profundidad es útil para representar correctamente pequeñas diferencias en la distancia. Sin embargo, es bastante efectiva para representar grandes diferencias o como indicador exagerado para representar diferencias pequeñas.

#### 12.3.4 Recortes de profundidad

Se puede proporcionar más información acerca de la profundidad con los **recortes de profundidad**. El plano de recorte posterior se coloca de manera que corta los objetos que se presentan. De esta manera, el usuario sabe que los objetos parcialmente recortados se presentan así debido al plano de recorte. También se puede usar un plano de recorte anterior. Si permitimos la variación dinámica de uno o ambos planos, el sistema puede transmitir mayor información de profundidad al observador. El recorte de profundidad de plano posterior se puede considerar como un caso especial de la indicación de profundidad: en la indicación normal de profundidad, la intensidad es una función regular de  $z$ ; en el recorte de profundidad, es una función incremental. Una técnica relacionada con el recorte de profundidad es realizar todos los puntos en un objeto intersecado por un plano. Esta técnica es muy efectiva cuando el plano cortante se mueve dinámicamente por el objeto; incluso se ha usado para ilustrar la profundidad en una cuarta dimensión [BANC77].

#### 12.3.5 Textura

Se pueden aplicar a un objeto texturas de vector simples, conocidas como **cuadriculado**. Estas texturas siguen la forma de un objeto y la delinean de manera más clara. La texturización de un conjunto de caras que de otra manera serían idénticas puede dejar en claro una proyección parcialmente ambigua. La texturización es muy útil en las proyecciones de perspectiva, ya que añade más líneas cuya convergencia y recorte frontal pueden servir como indicadores de profundidad.

#### 12.3.6 Color

El color se puede emplear simbólicamente para distinguir un objeto de otros asignando un color distinto a cada objeto en la escena. Así mismo, el color puede usarse en dibujos de línea para proporcionar información adicional. Por ejemplo, el color de cada vector de un objeto se puede determinar interpolando los colores que codifican las temperaturas en los puntos extremos del vector.

### 12.3.7 Determinación de líneas visibles

La última técnica de dibujo de líneas que mencionaremos es la **determinación de líneas visibles** o **eliminación de líneas ocultas**, cuyo resultado es la presentación únicamente de las líneas o partes de línea que están visibles (es decir, no oscurecidas). Sólo las superficies, acotadas por aristas (líneas), pueden oscurecer otras líneas. De esta manera, los objetos que bloquearán a otros deben modelarse como colecciones de superficies o como sólidos.

En la ilustración en color 26 se muestra la utilidad de la eliminación de líneas ocultas (también usando el color en la forma descrita en la sección 12.3.6). Como las vistas con eliminación de líneas ocultas esconden *toda* la estructura interna de los objetos opacos, no siempre son la manera más efectiva de presentar relaciones de profundidad. Las vistas con eliminación de líneas ocultas transmiten menos información de profundidad que las vistas explotadas y de corte. Una solución puede ser la presentación de las líneas ocultas como líneas punteadas.

## 12.4 Técnicas de generación para imágenes sombreadas

Las técnicas que mencionamos en la sección 12.3 se pueden emplear para crear dibujos de línea en pantallas vectoriales y de barrido. Las técnicas que presentaremos en esta sección aprovechan la capacidad de los dispositivos de barrido para presentar áreas sombreadas. Al generar imágenes para pantallas de barrido, se presentan problemas por la malla relativamente áspera de pixeles sobre la cual deben reproducirse contornos y sombreados suaves. La manera más sencilla de generar imágenes sombreadas caen víctimas de los artefactos de discretización, los cuales se presentaron por primera vez en la sección 3.14. Debido a la función principal que tiene la eliminación de artefactos de discretización en la producción de imágenes de alta calidad, todas las imágenes creadas en esta sección incorporan eliminación de artefactos de discretización.

### 12.4.1 Determinación de superficies visibles

Por analogía con la determinación de líneas visibles, la **determinación de superficies visibles** o **eliminación de superficies ocultas** comprende la presentación sólo de las partes de las superficies que están visibles para el observador. Como hemos visto, muchas veces es posible comprender los dibujos de línea más simples sin la determinación de líneas visibles. Cuando sólo hay unas cuantas líneas, las que están enfrente no obstruyen nuestra vista de las que están detrás. Por otra parte, si en los gráficos de barrido las superficies se generan como **áreas opacas**, la determinación de superficies visibles es indispensable para que la imagen tenga sentido. En la ilustración en color 27 se muestra un ejemplo en el cual todas las caras de un objeto están pintadas con el mismo color.

### 12.4.2 Iluminación y sombreado

Un problema con la ilustración en color 27 es que cada objeto aparece como una silueta plana. Nuestro siguiente paso hacia el realismo es sombrear las superficies visibles. A final de cuentas, la apariencia de cada superficie debe depender de los tipos de fuentes luminosas que la iluminan, sus propiedades (color, textura, reflectancia) y su posición y orientación con respecto a las fuentes luminosas, el observador y otras superficies.

En muchos ambientes visuales reales, llega de todas direcciones una cantidad considerable de *luz ambiental*. La luz ambiental es el tipo de fuente lumínosa más sencillo de modelar, ya que en un modelo de iluminación sencillo se supone que produce una iluminación constante para todas las superficies, sin importar su posición u orientación. Sin embargo, si sólo se emplea iluminación ambiental se obtienen imágenes poco realistas, ya que son pocos los ambientes reales iluminados exclusivamente con luz ambiental uniforme. La ilustración en color 27 es un ejemplo de una imagen sombreada de esta manera.

Una *fuente puntual*, cuyos rayos emanan de un solo punto, puede apropiar una pequeña bombilla incandescente. Se puede emplear una *fuente direccional*, donde todos los rayos provienen de la misma dirección, para representar el sol lejano aproximándolo como una fuente puntual infinitamente distante. Para el modelado de estas fuentes se requiere trabajo adicional, ya que su efecto depende de la orientación de la superficie. Si ésta es *normal* (perpendicular) a los rayos luminosos incidentes, estará iluminada de manera brillante; cuantos más oblicua sea la superficie a los rayos luminosos, menor será su iluminación. Esta variación en la iluminación es, por supuesto, un indicador poderoso de la estructura tridimensional de un objeto. Por último, una *fuente distribuida o extendida*, cuya área superficial emite luz, como un banco de luces fluorescentes, es incluso más compleja de modelar ya que la luz no proviene de una sola dirección ni de un solo punto. En la ilustración en color 28 se presenta el efecto de la iluminación de nuestra escena con fuentes luminosas ambientales y puntuales, así como del sombreado de cada polígono en forma individual.

### 12.4.3 Sombreado interpolado

El **sombreado interpolado** es una técnica con la cual se calcula información de sombreado para cada vértice de polígono y se interpola a través de los polígonos para determinar el sombreado en cada pixel. Este método es muy eficaz cuando se pretende que el objeto poligonal aproxime una superficie curva. En este caso, la información de sombreado que se calcula en cada vértice puede basarse en la orientación real de la superficie en ese punto y utilizarse para todos los polígonos que comparten ese vértice. La interpolación entre estos valores en un polígono aproxima los cambios suaves en el sombreado que ocurren en una superficie curva, no plana.

Incluso los objetos que se suponen poliédricos, no curvos, pueden aprovechar el sombreado interpolado ya que la información de sombreado que se calcula para cada vértice de un polígono puede variar, aunque generalmente

mucho menos que con un objeto curvo. Cuando se calcula la información de sombreado para un objeto poliédrico verdadero, el valor que se determina para un vértice del polígono sólo se usa para ese polígono y no para los demás que comparten el vértice. En la ilustración en color 29 se muestra el sombreado de Gouraud, un tipo de sombreado interpolado que se analiza en la sección 14.2.

#### 12.4.4 Propiedades materiales

El realismo es mayor si se consideran las **propiedades materiales** de cada objeto al determinar el sombreado. Algunos materiales son opacos y dispersan la luz reflejada de igual forma en todas las direcciones, como una tiza; otros son brillantes y reflejan la luz sólo en ciertas direcciones relativas al observador y a la fuente luminosa, como un espejo. En la ilustración en color 31 se puede observar cómo se vería nuestra escena si algunos objetos se modelaran como brillantes. Aquí se usó el sombreado de Phong, un método de sombreado interpolado más preciso (Sec. 14.2).

#### 12.4.5 Modelado de superficies curvas

Aunque el sombreado interpolado mejora en forma considerable la apariencia de una imagen, la geometría del objeto sigue siendo poligonal. La ilustración en color 32 usa modelos de objetos que incluyen superficies curvas. Para cada pixel de la imagen se calcula toda la información de sombreado.

#### 12.4.6 Iluminación y sombreado mejorados

Una de las razones más importantes por las cuales la apariencia de la mayoría de las imágenes graficadas por computador es “irreal”, es la imposibilidad de modelar con precisión las diversas formas en que la luz interactúa con los objetos. En la ilustración en color 33 se emplean mejores modelos de iluminación. En la sección 14.1.7 se analizan los avances en el diseño de modelos de iluminación eficaces y físicamente correctos.

#### 12.4.7 Textura

La textura de los objetos no sólo proporciona más indicación de la profundidad, como vimos en la sección 12.3.5, sino que además puede imitar el detalle de la superficie de objetos reales. En la ilustración en color 35 se presentan varias formas de simular una textura, las cuales van desde la variación del color de la superficie (como se hizo con la pelota con patrones) hasta la deformación de la geometría de la superficie (como se hizo con el toro estriado y el cono arrugado).

#### 12.4.8 Sombras

Podemos aumentar el realismo reproduciendo las sombras proyectadas por un objeto sobre otro. Observe que ésta es la primera técnica que hemos visto en la

cual la apariencia de las superficies visibles de un objeto se ve afectada por otros objetos. En la ilustración en color 35 se muestran las sombras proyectadas por la lámpara en la parte posterior de la escena. Las sombras aumentan el realismo y ofrecen mayor indicación de la profundidad: si el objeto *A* proyecta una sombra sobre la superficie *B*, entonces sabemos que *A* está entre *B* y una fuente de luz directa o reflejada. Una fuente luminosa puntual proyecta sombras bien definidas, ya que es totalmente visible o invisible desde cualquier punto. Una fuente luminosa extendida proyecta sombras suaves, ya que hay una transición regular desde los puntos que ven la totalidad de la fuente lumínosa hasta los que no la observan, pasando por aquellos que sólo ven parte de la fuente.

#### 12.4.9 Transparencia y reflexión

Hasta ahora sólo hemos tratado superficies opacas. Las superficies transparentes también pueden ser útiles en la elaboración de imágenes. Los modelos de transparencia más sencillos no incluyen la refracción de la luz que pasa por un sólido transparente. Sin embargo, la carencia de refracción puede ser una ventaja si la transparencia se usa más para revelar la geometría interna de un objeto que para simular la realidad. Los modelos más complejos incluyen refracción, translucencia difusa y atenuación de la luz con la distancia. En forma parecida, un modelo de reflexión luminosa puede simular los reflejos bien definidos de un espejo perfecto que refleja otro objeto o las reflexiones difusas de una superficie menos pulida. En la ilustración en color 36 se muestra el efecto de la reflexión en el piso y en la tetera. En la ilustración en color 41 se muestra la transparencia.

Como sucede en el modelado de sombras, para modelar la transparencia o la reflexión se requiere el conocimiento de otras superficies además de la que sombrean. Así mismo, la transparencia refractiva es el primer efecto que hemos mencionado en el cual se requiere que los objetos realmente se modelen como sólidos y no simplemente como superficies. Tenemos que conocer algo acerca de los materiales por los cuales pasa un rayo luminoso y la distancia que viaja para poder modelar correctamente su refracción.

#### 12.4.10 Modelos de cámara mejorados

Todas las imágenes que hemos presentado hasta el momento se basan en un modelo de cámara con un lente de agujero de alfiler y un obturador infinitamente rápido. Todos los objetos están bien enfocados y representan el mundo en un instante a la vez. Es posible modelar con mayor precisión la forma en que vemos (nosotros y las cámaras) el mundo. Por ejemplo, modelando las propiedades focales de los lentes se pueden producir imágenes, como la ilustración en color 37, que muestran la **profundidad de campo**: algunas partes del objeto están enfocadas, mientras que las partes más cercanas y lejanas se encuentran fuera de foco. Consulte los detalles al respecto en [POTM82]. Otras técnicas permiten usar efectos especiales, como lentes de ojo de pescado. La falta

efectos de profundidad de campo es en parte responsable de la apariencia surrealista de las primeras imágenes generadas por computador.

Los objetos en movimiento se ven distintos de los estacionarios en una fotografía tomada con una cámara fija o de cine. Como el obturador está abierto durante un periodo finito, las partes visibles de los objetos en movimiento aparecen borrosas en el plano de la película. Este efecto, conocido como **borrosidad de movimiento**, se puede simular de manera muy convincente [KORE83]. La borrosidad de movimiento no sólo captura el efecto del movimiento en imágenes fijas, sino además es muy importante en la producción de animaciones de alta calidad, como se describe en el capítulo 21 de [FOLE90].

## 12.5 Modelos de objetos mejorados

Independientemente de la tecnología de generación de imágenes que se utilice, la búsqueda de realismo se ha centrado, en parte, en formas de construir modelos estáticos y dinámicos más convincentes. Algunos investigadores han desarrollado modelos para tipos de objetos especiales, como gases, olas, montañas y árboles (p. ej., véanse las Ilustr. en color 11, 12 y 13). Las técnicas para producir estos objetos se basan en fractales, métodos gramaticales y sistemas de partículas. Otros investigadores se han centrado en el modelado avanzado usando *splines*, modelos por procedimientos, generación de volúmenes, modelado físico y modelado humano. Otro tema importante es la automatización del posicionamiento de gran número de objetos, como árboles en un bosque, que sería muy tedioso como tarea manual. Witkin y Kass [WITK88] describen un método automático de colocación que aplicaron a la animación de un modelo de *Luxo Jr.* Sin embargo, las imágenes de *Luxo Jr.* que aparecen en la portada de este libro corresponden a una animación de Pixar [PIXA86] cuya producción no utilizó métodos automatizados para la ubicación. Algunas de estas técnicas se analizan en la sección 9.5, y se puede hallar un tratamiento más detallado en el capítulo 20 de [FOLE90].

## 12.6 Dinámica y animación

### 12.6.1 El valor del movimiento

Por **dinámica** entendemos los cambios que se distribuyen en una secuencia de imágenes, incluyendo cambios en la posición, el tamaño, las propiedades materiales, la iluminación y la especificación de visualización; de hecho, se trata de cualquier cambio en una parte de la escena o en las técnicas que se apliquen a

ella. Los beneficios de la dinámica se pueden examinar en forma independiente del avance hacia imágenes estáticas más elaboradas.

Quizás la forma más popular de la dinámica sea la dinámica de movimiento, que puede abarcar desde transformaciones simples realizadas bajo el control del usuario, hasta animaciones complejas. El movimiento ha sido parte importante de la graficación por computador desde que nació el campo. En los primeros días del lento hardware de gráficos de barrido, la capacidad de movimiento era uno de los puntos de venta más competitivos de los sistemas gráficos vectoriales. Si se presenta en sucesión rápida una serie de proyecciones del mismo objeto, cada una desde un punto de vista ligeramente distinto alrededor del objeto, parecerá que éste rota. Al integrar la información de las vistas, el observador crea una hipótesis del objeto.

Por ejemplo, una proyección de perspectiva de un cubo en rotación proporciona varios tipos de información. Hay una serie de proyecciones diferentes, las cuales son en sí útiles. Esta información es complementada por el efecto de movimiento, donde la velocidad lineal máxima de los puntos cercanos al centro de rotación es menor que la de los puntos distantes del centro. Esta diferencia puede ayudar a esclarecer la distancia relativa de un punto al centro de rotación. Además, la variación de tamaño de varias partes del cubo al variar su distancia en la proyección de perspectiva ofrece indicaciones adicionales acerca de las relaciones de profundidad. El movimiento es aún más poderoso si se encuentra bajo el control interactivo del usuario. Los observadores pueden formular una hipótesis del objeto con mayor rapidez usando la transformación selectiva de un objeto.

En contraste con la utilización de transformaciones simples para clarificar modelos complejos, algunos modelos sorprendentemente sencillos son muy convincentes si se mueven de manera realista. Por ejemplo, unos cuantos puntos colocados en lugares clave de un modelo humano pueden proporcionar una ilusión convincente de una persona en movimiento si se mueven de manera natural. Los puntos en sí no parecen una persona, pero informan al observador que hay una persona presente. También es un hecho conocido que los objetos en movimiento se pueden generar con menor detalle que el necesario para presentar objetos estáticos, ya que es más difícil para el observador comprender los detalles si el objeto está en movimiento. Por ejemplo, los televidentes muchas veces se sorprenden al descubrir lo malo y granuloso que es un cuadro de televisión.

## 12.6.2 Animación

*Animar* es, literalmente, dar vida. Aunque la gente piensa en la animación como un sinónimo de movimiento, abarca todos los cambios que tienen un efecto visual. Así, incluye la posición variable en el tiempo (**dinámica de movimiento**, la forma, el color, la transparencia, la estructura y la textura de un objeto (**dinámica de actualización**), así como los cambios en iluminación y posición, orientación y enfoque de la cámara, e incluso variaciones en la técnica de generación de la imagen.

La animación se usa mucho en la industria del entretenimiento y también se utiliza en la educación; en aplicaciones industriales como sistemas de control, simuladores de vuelo y pantallas de navegación en aeronaves; y en investigación científica. Las aplicaciones científicas de la graficación por computador, sobre todo la animación, se han agrupado bajo el nombre de **visualización científica**. Sin embargo, la visualización es mucho más que la simple aplicación de gráficos a la ciencia y la ingeniería; puede comprender otras disciplinas, como el procesamiento de señales, la geometría computacional y la teoría de bases de datos. Con frecuencia, las animaciones en la visualización científica se generan a partir de simulaciones de fenómenos científicos. Los resultados de las simulaciones pueden ser grandes bases de datos que representan datos bidimensionales o tridimensionales (p. ej., en el caso de simulaciones de flujo de fluidos); estos datos se convierten en imágenes que constituyen la animación. En el otro extremo, la simulación puede generar posiciones y localidades de objetos físicos, los cuales se deben generar en alguna forma para producir la animación. Esto sucede, por ejemplo, en las simulaciones químicas, donde la simulación puede generar las posiciones y orientaciones de los átomos en una reacción, pero la animación puede presentar una vista de esfera y barra para cada molécula, o bien esferas traslapantes con sombreado suave que representan a cada átomo. En algunos casos, el programa de simulación contendrá un lenguaje de animación incorporado, de manera que los procesos de simulación y animación sean simultáneos.

Si algún aspecto de la animación cambia con demasiada rapidez con respecto al número de cuadros de animación presentados por segundo, entonces ocurre el **artefacto temporal de discretización**. Como ejemplos de este tipo tenemos las ruedas de carro que parecen girar hacia atrás y el movimiento brusco de objetos que se mueven a través de un gran campo visual en poco tiempo. La cinta de video se presenta a 30 cuadros por segundo (*fps. frames per second*) y la velocidad típica de la película fotográfica es de 24 cuadros por segundo; ambas proporcionan resultados adecuados para la mayoría de las aplicaciones. Por supuesto, para aprovechar estas tasas hay que generar una nueva imagen para cada cuadro de cinta de video o de película. En cambio, si el animador registra cada imagen en dos cuadros de cinta de video, el resultado efectivo será de 15 cuadros por segundo y el movimiento se volverá más brusco.

La animación tradicional (o sea, la animación no producida por computador) es una disciplina aparte; aquí analizaremos sólo algunos de sus aspectos. Sin embargo, nuestro enfoque será un resumen de los conceptos básicos de la animación basada en computador. Comenzaremos por analizar la animación convencional y la forma en que se han usado computadores para asistir en su creación. Después pasaremos a la animación producida principalmente por un computador. Como gran parte de ésta es animación tridimensional, no se pueden aplicar directamente muchas de las técnicas de la animación de caracteres bidimensionales. Además, el control del curso de una animación es mucho más difícil cuando el animador no la dibuja en forma directa: muchas veces es más difícil describir *cómo* hacer algo que realizar la acción. Por ello, después de mencionar los lenguajes de animación, examinaremos varias técnicas de control de animaciones. Concluiremos con el análisis de algunas reglas generales para la animación y los problemas específicos que ésta presenta.

**Animación convencional y asistida por computador.** Una animación convencional se crea con una secuencia bastante fija. La historia de la animación se escribe (o quizás sólo se concibe) y se elabora en forma de **historieta** (*Storyboard*). Una historieta es una animación en forma general, una secuencia de alto nivel de bosquejos que muestran la estructura y las ideas de la animación. Después se graba la banda sonora (si la hay), se produce una disposición detallada (con dibujo para cada escena de la animación) y se lee la banda sonora, es decir, se registran en orden los instantes en que ocurren los sonidos significativos. Despues se correlacionan la disposición detallada y la banda sonora. El siguiente paso consiste en dibujar ciertos **cuadros clave**, (*keyframes*) éstos son los cuadros donde las entidades que se animan están en posiciones extremas o caracteristicas, a partir de las cuales se pueden inferir sus posiciones intermedias. Se crean los cuadros intermedios (proceso llamado *inbetweening* en inglés) y se produce un filme de prueba (**una prueba a lápiz**). Los cuadros de la prueba a lápiz se transfieren a hojas de película de acetato (*cels*), ya sea copiando manualmente la tinta o por fotocopiado directo a las hojas. Las hojas de acetato se colorean o pintan y se montan en la secuencia correcta; después se filman. Este tipo de animación se conoce como **animación de cuadros clave**, por la utilización de cuadros clave e intermedios. El nombre también se aplica a los sistemas basados en computador que imitan este proceso.

Muchas etapas de la animación convencional parecen adaptarse idealmente a la ayuda del computador, sobre todo la creación de cuadros intermedios y el coloreado, que pueden efectuarse con una técnica de relleno por semáforos [SMIT79]. Sin embargo, antes de usar el computador hay que digitalizar los dibujos. La digitalización se puede realizar con rastreo óptico, trazando los dibujos con una tableta de datos o produciendo los dibujos originales con un programa de dibujo. Es probable que los dibujos tengan que someterse a postprocesamiento (p. ej., filtrado) para limpiar pequeñas fallas que se presenten durante el proceso de entrada (sobre todo con el rastreo óptico) y para vizar un poco los contornos.

**Lenguajes de animación.** Se han desarrollado muchos lenguajes para describir la animación, desde notaciones independientes hasta paquetes de procedimientos que se emplean con lenguajes convencionales (véase el Cap. 21 de [FOLLE90]). Algunos lenguajes de animación se combinan con lenguajes de modelado, para que las descripciones de los objetos en una animación y las modificaciones de dichos objetos se lleven a cabo al mismo tiempo.

**Métodos de control de animación.** El control de una animación es bastante dependiente del lenguaje que se usa para describirla, y la mayoría de los mecanismos de control pueden adaptarse para ser usados con varios tipos de lenguajes. Los mecanismos de control de animación van del control explícito total —en el cual el animador describe explícitamente la posición y los atributos de cada objeto en la escena por medio de traslaciones, rotaciones y otros operadores de modificación de posición y atributos— hasta el altamente automati-

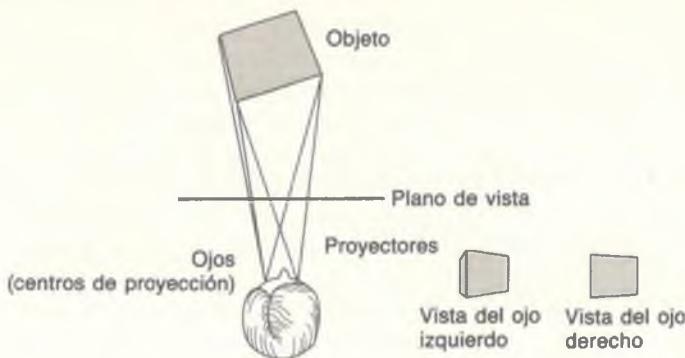
zado control que ofrecen los sistemas basados en conocimiento, que toman descripciones de alto nivel de una animación (“que el personaje salga caminando de la habitación”) y generan los controles explícitos que realizan los cambios necesarios para producir la animación. En [FOLE90] se describen varias de las técnicas más recientes de control de animación.

**Reglas básicas de la animación.** La animación tradicional de personajes pasó de ser una forma de arte a una industria en Walt Disney Studio entre 1925 y finales de la década de 1930. En un principio, la animación implicaba poco más que dibujar una secuencia de caricaturas, una colección de imágenes estáticas que, en conjunto, formaban una imagen animada. Conforme se desarrollaron las técnicas de animación, surgieron ciertos principios básicos que se convirtieron en las reglas fundamentales para la animación de personajes y que siguen usándose en la actualidad [LAYB79; LASS87]. A pesar de tener sus orígenes en la animación de personajes de caricatura, muchos de estos principios también son aplicables a las animaciones tridimensionales realistas. Estas reglas, junto con su aplicación en la animación de personajes tridimensionales, se estudian en [LASS87]. Sin embargo, es importante reconocer que estas reglas no son absolutas. Así como mucho del arte moderno se ha alejado de las reglas tradicionales del dibujo, en la actualidad muchos animadores se han alejado de las reglas tradicionales de la animación, muchas veces con resultados excelentes (véase, por ejemplo, [LEAF74; LEAF77]). Entre las reglas se encuentran la *compresión y estiramiento (squash and stretch)* que indica las propiedades físicas de un objeto por medio de distorsiones de su forma; movimiento de *entrada lenta/salida lenta (slow-in/slow-out)* para proporcionar transiciones suaves; y la *puesta en escena (staging)* apropiada, o selección de una vista que proyecte la mayor cantidad de información acerca de los eventos que tienen lugar en la animación.

**Problemas particulares de la animación.** Así como al pasar de los gráficos bidimensionales a los tridimensionales se presentaron nuevos problemas y retos, el cambio de tres dimensiones a cuatro (la adición de la dimensión temporal) también presenta problemas especiales. Uno de estos problemas es el *artefacto temporal de discretización*. Así como los problemas de artefactos de discretización en los gráficos bidimensionales y tridimensionales se resuelven parcialmente con un aumento en la resolución de la pantalla, los problemas de artefacto temporal en la animación se resuelven en parte aumentando la resolución temporal. Por supuesto, otro aspecto de la solución bidimensional es la eliminación del artefacto de discretización; la solución correspondiente en tres dimensiones es la eliminación del artefacto temporal.

## 12.7 Estereoóptica

Todas las técnicas que hemos analizado hasta este momento presentan la misma imagen a los dos ojos del observador. Ahora lleve a cabo el siguiente experi-



**Figura 12.7** Disparidad binocular.

mento: vea la superficie de su escritorio o mesa primero con un ojo y luego con el otro. Las dos vistas difieren ligeramente porque los ojos están separados entre sí unos cuantos centímetros, como se ilustra en la figura 12.7. La *disparidad binocular* causada por esta separación proporciona un poderoso indicador de profundidad llamado **estereoscóptica** o **visión estéreo**. Nuestro cerebro combina los dos imágenes separadas en una sola que se interpreta como tridimensional. Las dos imágenes se conocen como *par estéreo*; los pares estéreo se utilizan en populares visores estéreo al iniciar el siglo y se emplean en la actualidad en un juguete común llamado *View-Master*. En la ilustración en color 22 se muestra un par estéreo de una molécula. Se pueden combinar las dos imágenes para formar una tridimensional viendo el par de manera que cada ojo sólo vea una imagen; esto se puede hacer colocando por ejemplo un pedazo de papel perpendicularmente entre las dos imágenes. Algunas personas pueden ver el efecto sin necesidad del trozo de papel, y pocas personas no pueden ver el efecto.

Existen otras técnicas para proporcionar imágenes distintas a cada ojo, entre las cuales se incluyen anteojos con filtros polarizantes y la holografía. Algunas de estas técnicas hacen posible la creación de imágenes verdaderamente tridimensionales que ocupan espacio en lugar de ser proyectadas sobre un plano. Estos presentadores pueden ofrecer un indicador adicional de profundidad tridimensional: los objetos cercanos están en realidad más cerca, como en la vida real, de manera que los ojos del observador enfocan de manera distinta los diversos objetos, dependiendo de su proximidad. Las matemáticas de la proyección estéreo se describen en el ejercicio 6.17.

## 12.8 Pantallas mejoradas

Además de las mejoras en el software que se usa para diseñar y generar imágenes de objetos, el perfeccionamiento de las pantallas ha aumentado la ilusión

realidad. La historia de la graficación por computador es, en parte, la de una mejora constante en la calidad visual lograda por los dispositivos de presentación. No obstante, la gama de colores de un monitor moderno y su intervalo de intensidad dinámica son pequeños subconjuntos de lo que podemos ver. ¡Falta mucho por hacer antes de que la imagen en la pantalla tenga la misma claridad y contraste que una fotografía profesional bien impresa! La resolución limitada de las pantallas hace imposible la reproducción del detalle muy fino. Los artefactos como un patrón visible del fósforo, el reflejo de la pantalla, la distorsión geométrica y el efecto estroboscópico del parpadeo de la tasa de cuadros son recordatorios siempre presentes de que estamos viendo una pantalla. El tamaño relativamente pequeño de la pantalla, comparado con nuestro campo visual, también nos ayuda a recordar que la pantalla es una ventana al mundo y no el mundo en sí.

## 12.9 Interacción con nuestros otros sentidos

Quizás el último paso hacia el realismo sea la integración de las imágenes realistas con la información presentada a nuestros otros sentidos. La graficación por computador tiene un largo historial de programas que se apoyan en diversos dispositivos de entrada para permitir la interacción con el usuario. Los simuladores de vuelo son un ejemplo actual del acoplamiento de los gráficos con sonidos de máquinas y movimientos realistas, todo lo cual se ofrece en una cabina simulada para crear un ambiente completo. Utilizando un simulador en forma de casco, que revisa el movimiento de la cabeza, es posible lograr otro indicador de profundidad tridimensional importante, llamado *paralaje de movimiento de cabeza*: cuando el usuario mueve su cabeza de lado a lado, quizás tratando de ver más de un objeto parcialmente oculto, la vista cambia como sucedería en la vida real. Otras actividades en el área de pantallas de casco se centran en la exploración de *mundos virtuales*, como el interior de moléculas o edificios que aún no se han construido [CHUN89].

Varios juegos de video actuales tienen como característica principal un automóvil o un avión que controla el jugador, que se mueve en sincronía con una simulación que incluye imágenes y sonidos reales o sintetizados y retroalimentación. Esta forma de usar modalidades de entrada y salida adicionales marca el camino para los sistemas del futuro que permitirán inmersiones totales de todos los sentidos, incluyendo el oído, el tacto, el gusto y el olfato.

## RESUMEN

En este capítulo proporcionamos una introducción de alto nivel a las técnicas utilizadas para producir imágenes realistas. En los capítulos siguientes veremos con detalle cómo implantar estas técnicas. Hay cuatro preguntas clave que usted debe tener en cuenta al leer acerca de los algoritmos que se presentan en capítulos subsecuentes:

1. *¿El algoritmo es de propósito general o especial?* Algunas técnicas funcionan mejor sólo en ciertas circunstancias; otras están diseñadas para ser más generales. Por ejemplo, algunos algoritmos suponen que todos los objetos son poliedros convexos y obtienen parte de su velocidad y su relativa sencillez de esta suposición.
2. *¿Cuál es el rendimiento espacio-temporal del algoritmo?* ¿Cómo es afectado el algoritmo por factores como el tamaño o la complejidad de la base de datos, o la resolución a la cual se genera la imagen?
3. *¿Cuán convincentes son los efectos que se generan?* Por ejemplo, ¿se modela correctamente la refracción, se ve bien sólo en ciertos casos o no se modela de ninguna manera? ¿Pueden añadirse efectos adicionales, como sombras o reflexión especular? ¿Serán convincentes? Al sacrificar la precisión con la cual se genera un objeto pueden lograrse mejoras notables en los requisitos espaciales o temporales de un programa.
4. *¿Es apropiado el algoritmo, dado el propósito para el cual se crea la imagen?* La filosofía que sirve de base para muchas de las imágenes en los capítulos siguientes se puede resumir como: "Si se ve bien, ¡hágalo!" Esta pauta se puede interpretar de dos maneras. Puede usarse un algoritmo simple o rápido si produce efectos atractivos, aunque no pueda hallar ninguna justificación en las leyes de la física. Por otra parte, se puede emplear un algoritmo muy costoso si es la única forma conocida para generar ciertos efectos.

## Ejercicios

12.1 Suponga que tiene un sistema de graficación con el cual puede dibujar en tiempo real cualquiera de las ilustraciones en color que aparecen como referencia en este capítulo. Considere varias áreas de aplicación con las cuales esté (o vaya a estar) familiarizado. En cada área, liste aquellos efectos que serían las más útiles y los que tendrían menos utilidad.

12.2 Muestre que no puede inferir la dirección de rotación a partir de proyecciones ortográficas de un cubo alambrado, monocromático y giratorio. Explique cómo podrían ayudar otras técnicas para que la dirección de rotación sea clara sin necesidad de cambiar la proyección.

Dado un conjunto de objetos tridimensionales y una especificación de vista, queremos determinar cuáles son las líneas o superficies visibles de los objetos, ya sea desde el centro de proyección (para proyecciones de perspectiva) o a lo largo de la dirección de proyección (para proyecciones paralelas), de manera que podamos presentar únicamente las líneas o superficies visibles. Este proceso se conoce como **determinación de líneas visibles** o de **superficies visibles**, o como **eliminación de líneas ocultas** o **superficies ocultas**. En la determinación de líneas visibles se supone que las líneas son las aristas de superficies opacas que pueden oscurecer las aristas de otras superficies que se encuentran más lejos del observador. Por lo tanto, haremos referencia al proceso general como **determinación de superficies visibles**.

Aunque el enunciado de esta idea fundamental es sencillo, su implantación requiere gran poder de procesamiento y por ende comprende grandes cantidades de tiempo en máquinas convencionales. Estos requisitos han alentado el desarrollo de varios algoritmos cuidadosamente estructurados para la determinación de superficies visibles; muchos de ellos se describen en este capítulo. Además, se han diseñado varias arquitecturas de propósito especial para tratar este problema, algunas de las cuales se analizan en el capítulo 18 de [FOLE90]. La necesidad de esta atención se puede ver a partir del análisis de dos métodos fundamentales para resolver el problema. En ambos casos se puede pensar en el objeto como algo que comprende uno o más polígonos (o más superficies complejas).

El primer método consiste en determinar cuál de los  $n$  objetos es visible en cada pixel de la imagen. El seudocódigo para este método tiene el siguiente aspecto:

```
for (cada pixel en la imagen) {
    determinar el objeto más cercano al observador que es atravesado por el proyector a través del pixel;
    dibujar el pixel con el color apropiado;
}
```

La forma más burda y directa de hacer esto con un pixel requiere la revisión de los  $n$  objetos para determinar cuál es el más cercano al observador a lo largo del proyector que pasa por el pixel. En el caso de  $p$  pixeles, este esfuerzo es proporcional a  $np$ , donde  $p$  es superior al millón en una pantalla de alta resolución.

El segundo método es comparar los objetos directamente entre sí, eliminando objetos enteros o porciones de ellos que no sean visibles. En seudocódigo, esto es

```
for (cada objeto en el mundo) {
    determinar aquellas partes del objeto cuya vista no está obstruida por otras partes del mismo objeto o por otro objeto;
    dibujar esas partes con el color apropiado;
}
```

Esto se puede hacer en forma sencilla comparando cada uno de los  $n$  objetos consigo mismo y con los otros, y descartando las porciones invisibles. En este caso el esfuerzo computacional es proporcional a  $n^2$ . Aunque este segundo método podría parecer superior si  $n < p$ , sus pasos individuales son generalmente más complejos y consumen más tiempo, como veremos, de manera que en la mayoría de los casos es más lento y difícil de implantar.

Nos referiremos a estos métodos prototípicos como algoritmos de **precisión de la imagen** y de **precisión del objeto**, respectivamente. Los algoritmos de precisión de la imagen suelen llevarse a cabo con la resolución del dispositivo de presentación y determinan la visibilidad en cada pixel. Los algoritmos de precisión del objeto se efectúan con la precisión de la definición del objeto y, por consiguiente, determinan la visibilidad de cada objeto.<sup>1</sup> Como los cálculos de

<sup>1</sup> Los términos *espacio de imagen* y *espacio de objeto*, popularizados por Sutherland, Sproull y Schumacker [SUTH74a] también se utilizan para establecer la misma distinción. Por desgracia, estos términos también se han usado en forma bastante distinta en la graficación por computadora. Por ejemplo, el término *espacio de imagen* se ha usado para hacer referencia a objetos después de la transformación de perspectiva [CATM75] o después de la proyección sobre el plano de visión [GILO78], pero con su precisión original. Para evitar confusiones, hemos optado por nuestros términos ligeramente modificados. Nos referimos de manera explícita a la proyección o transformación de perspectiva de un objeto, cuando sea necesario, y reservamos los términos *precisión de la imagen* y *precisión del objeto* para indicar la precisión con la cual se llevan a cabo los cálculos. Por ejemplo, la intersección de dos proyecciones de objetos en el plano de vista en una operación de precisión de objeto si se mantiene la precisión del objeto original.

precisión del objeto se efectúan sin tener en cuenta la resolución de una pantalla en particular, deben seguirse con un paso en el cual los objetos se dibujen con la resolución deseada. Sólo hay que repetir este paso final si se modifica el tamaño de la imagen, por ejemplo para cubrir un número diferente de pixeles en una pantalla de trama. Esto se debe a que la geometría de la proyección de cada objeto visible está representada con la resolución completa del objeto en la base de datos. En cambio, considere la ampliación de una imagen creada con un algoritmo de precisión de imagen. Como los cálculos de superficies visibles se efectuaron con la resolución original menor, deben efectuarse de nuevo si queremos presentar detalles adicionales. Por ello, los algoritmos de precisión de la imagen presentan artefactos de discretización durante el cálculo de la visibilidad, algo que no ocurre con los algoritmos de precisión del objeto.

Los algoritmos de precisión del objeto se desarrollaron inicialmente para sistemas gráficos vectoriales. En estos dispositivos, la forma más natural de lograr la eliminación de aristas ocultas era por medio de la conversión de la lista inicial de líneas a otra lista donde se eliminaran las líneas totalmente ocultas por otras superficies y donde las líneas parcialmente ocultas se recortaran a uno o más segmentos de línea visibles. Todo el procesamiento se llevaba a cabo con la precisión de la lista original y el resultado era una lista con el mismo formato. En cambio, los algoritmos de precisión de la imagen se escribieron originalmente para dispositivos de barrido, de manera que aprovecharan el número relativamente pequeño de pixeles para los cuales había que efectuar los cálculos de visibilidad. Esta división era de fácil comprensión. Las pantallas vectoriales tenían un gran espacio de direcciones ( $4096 \times 4096$ , incluso en los sistemas más antiguos) y límites muy estrictos con respecto al número de líneas y objetos que podían presentarse. Por otra parte, las pantallas de barrido tenían un espacio de direcciones limitado ( $256 \times 256$  en los primeros sistemas) y la capacidad para presentar una cantidad de objetos potencialmente ilimitada. Los algoritmos más recientes muchas veces combinan cálculos de precisión de imagen y de objeto, usando los cálculos de precisión del objeto por cuestiones de precisión y los de precisión de la imagen por cuestiones de velocidad.

En este capítulo presentaremos primero varios temas relacionados con la eficiencia de los algoritmos generales de superficies visibles. Después presentaremos los métodos principales para determinar superficies visibles.

### 13.1 Técnicas para algoritmos eficientes de superficies visibles

La formulación de los algoritmos típicos de precisión de la imagen y de precisión del objeto puede requerir varias operaciones que quizás sean muy costosas. Estas operaciones incluyen la determinación de un proyector y un objeto o de dos proyecciones de objetos, si se intersecan o no y dónde se intersecan. Entonces, para cada conjunto de intersecciones es necesario calcular el objeto más

cercano al observador y por ende visible. A fin de minimizar el tiempo que requiere la creación de una imagen, tenemos que organizar los algoritmos de superficies visibles de manera que las operaciones más costosas se lleven a cabo con la mayor eficiencia y la menor frecuencia posible. En las secciones siguientes se describen varias formas generales de hacerlo.

### 13.1.1 Coherencia

Sutherland, Sproull y Schumacker [SUTH74a] señalan cómo los algoritmos de superficies visibles pueden aprovechar la **coherencia**, es decir, el grado al cual partes de un ambiente o su proyección exhiben similitudes locales. Por lo general los ambientes contienen objetos cuyas propiedades varían suavemente de una parte a otra. De hecho, son las discontinuidades menos frecuentes en las propiedades (como la profundidad, el color y la textura) y los efectos que producen lo que nos permite distinguir un objeto de otro. Aprovechamos la coherencia al reutilizar los cálculos realizados para una parte del ambiente o de la imagen en otras partes cercanas, ya sea sin efectuar cambios o con cambios incrementales que son más eficientes que el recálculo de toda la información. Se han identificado varios tipos de coherencia [SUTH74a], los cuales listamos a continuación y mencionaremos más adelante:

- **Coherencia de objetos.** Si un objeto está completamente separado de otros, quizás sólo haya que efectuar comparaciones entre los dos objetos, no entre sus caras o aristas componentes. Por ejemplo, si todas las partes del objeto *A* están más lejos del observador que todas las partes del objeto *B*, no hay que comparar ninguna de las caras del objeto *A* con las de *B* para determinar si oscurecen las caras de *B*.
- **Coherencia de caras.** Las propiedades de las superficies generalmente varían suavemente sobre una cara, permitiendo la modificación incremental de los cálculos de una parte de la cara para aplicarlos a partes adyacentes. En algunos modelos se puede garantizar que las caras no serán interpenetrantes.
- **Coherencia de aristas.** Una arista puede cambiar de visibilidad sólo cuando cruza detrás de una arista visible o penetra en una cara visible.
- **Coherencia de aristas implicadas.** Si una cara plana penetra en otra, su línea de intersección (la arista implicada) se puede determinar a partir de los puntos de intersección.
- **Coherencia de líneas de barrido.** El conjunto de tramos de objetos visibles determinado para una linea de barrido de una imagen generalmente difiere muy poco del conjunto en la línea anterior.
- **Coherencia de áreas.** Un grupo de píxeles adyacentes muchas veces está abierto por la misma cara visible. Un caso especial de la coherencia de áreas

es la **coherencia de tramos**, que se refiere a la visibilidad de una cara en un tramo de pixeles adyacentes de una línea de barrido.

- **Coherencia de profundidad.** Las partes adyacentes de la misma superficie generalmente están próximas en cuanto a profundidad, mientras que las superficies distintas en el mismo lugar de la pantalla suelen tener mayor separación de profundidad. Una vez que se ha calculado la profundidad en un punto de la superficie, la profundidad de los puntos en el resto de la superficie se puede determinar con una simple ecuación de diferencia.
- **Coherencia de cuadros.** Las imágenes del mismo ambiente en dos puntos sucesivos en el tiempo casi siempre serán bastante similares, a pesar de varios cambios pequeños en los objetos y en el punto de vista. Los cálculos efectuados para una imagen se pueden reutilizar con la siguiente en una secuencia.

### 13.1.2 Transformación de perspectiva

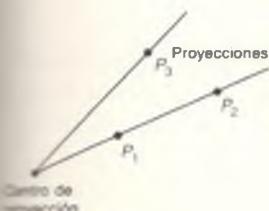


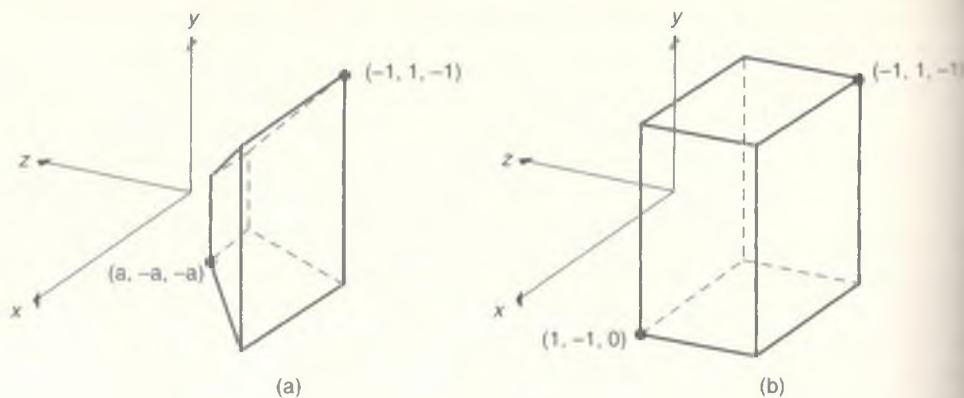
Figura 13.1

Si dos puntos  $P_1$  y  $P_2$  están en el mismo proyector, el más cercano oscurece al otro; esto no sucede en el caso contrario (p. ej.,  $P_1$  no oscurece a  $P_3$ ).

Es obvio que la determinación de superficies visibles se debe efectuar en el espacio tridimensional antes de realizar las proyecciones a dos dimensiones que destruyen la información necesaria para las comparaciones de profundidad. No obstante el tipo de proyección que se elija, la comparación de profundidad básica en un punto generalmente se puede reducir a la siguiente pregunta: dados los puntos  $P_1 = (x_1, y_1, z_1)$  y  $P_2 = (x_2, y_2, z_2)$ , ¿oscurece alguno de ellos al otro? La siguiente pregunta es equivalente: ¿Están  $P_1$  y  $P_2$  en el mismo proyector (véase la Fig. 13.1)? Si la respuesta es sí, se comparan  $z_1$  y  $z_2$  para determinar cuál es el punto más cercano al observador. Si la respuesta es no, ningún punto puede oscurecer al otro.

Las comparaciones de profundidad suelen realizarse después de aplicar la transformación de normalización (Cap. 6), de manera que los proyectores estén paralelos al eje  $z$  en las proyecciones paralelas o emanen del origen en las proyecciones de perspectiva. En el caso de una proyección paralela, los puntos estarán en el mismo proyector si  $x_1 = x_2$  y  $y_1 = y_2$ . En una proyección de perspectiva, por desgracia tenemos que efectuar cuatro divisiones para determinar si  $x_1/z_1 = x_2/z_2$  y  $y_1/z_1 = y_2/z_2$ , en cuyo caso los puntos se encuentran en el mismo proyector, como se muestra en la figura 13.1. Así mismo, si más tarde se compara  $P_1$  con un punto  $P_3$ , se requieren dos divisiones más.

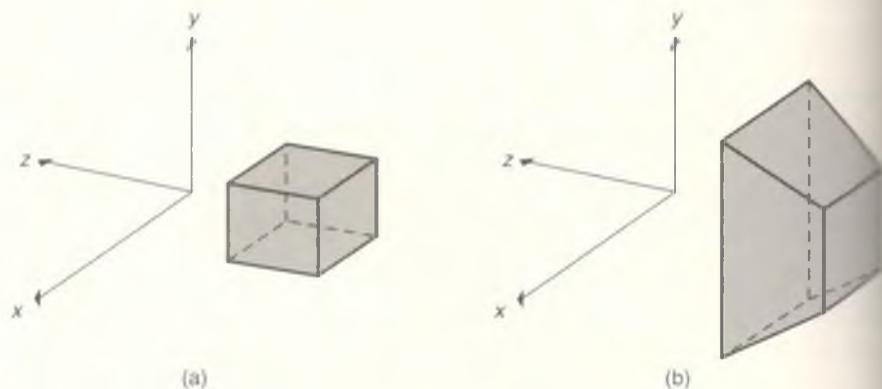
Se pueden evitar divisiones innecesarias transformando primero un objeto tridimensional en un sistema de coordenadas tridimensionales de la pantalla, de manera que la proyección paralela del objeto transformado sea igual que la proyección de perspectiva del objeto sin transformar. Así, la prueba para determinar si un punto oscurece a otro es la misma que en las proyecciones paralelas. Esta transformación de perspectiva distorsiona los objetos y mueve el centro de proyección al infinito en el eje  $z$  positivo, con lo cual los proyectores se convierten en paralelos. En la figura 13.2 se muestra el efecto de esta transformación sobre el volumen de vista en perspectiva; en la figura 13.3 se presenta la forma como se distorsiona un cubo con la transformación.



**Figura 13.2** Volumen de vista en perspectiva normalizado (a) antes y (b) después de la transformación de perspectiva.

La esencia de esta transformación es que conserva la profundidad relativa de los planos rectas y las líneas, al mismo tiempo que lleva a cabo el recorte frontal de perspectiva. La división que logra el recorte frontal se efectúa una vez por punto y no cada vez que se comparan dos puntos. La matriz

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1+z_{\min}} & \frac{-z_{\min}}{1+z_{\min}} \\ 0 & 0 & -1 & 0 \end{bmatrix}, \quad z_{\min} \neq -1 \quad (13.1)$$



**Figura 13.3** Un cubo (a) antes y (b) después de la transformación de perspectiva.

transforma el volumen de vista en perspectiva normalizado al paralelepípedo rectangular acotado por

$$-1 \leq x \leq 1, \quad -1 \leq y \leq 1, \quad -1 \leq z \leq 0. \quad (13.2)$$

Los recortes se pueden realizar con respecto al volumen de vista normalizado de pirámide truncada antes de aplicar  $M$ , pero en este caso los resultados del recorte deben multiplicarse por  $M$ . Una alternativa más atractiva es incorporar  $M$  a la perspectiva normalizando la transformación  $N_{\text{per}}$  del capítulo 6, de manera que sólo se requiera una multiplicación de matrices y luego se recorte en coordenadas homogéneas antes de efectuar la división. Si llamamos  $(X, Y, Z, W)$  a los resultados de la multiplicación, entonces, para  $W > 0$ , los límites del recorte serán

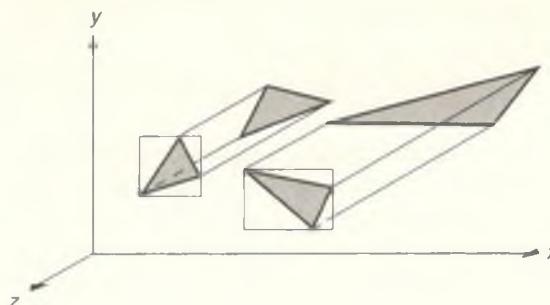
$$-W \leq X \leq W, \quad -W \leq Y \leq W, \quad -W \leq Z \leq 0. \quad (13.3)$$

Estos límites se derivan de la ecuación (13.2) reemplazando  $x, y$  y  $z$  por  $X/W, Y/W$  y  $Z/W$ , respectivamente, para reflejar el hecho de que  $x, y$  y  $z$  en la ecuación (13.2) son el resultado de una división entre  $W$ . Después del recorte, se divide entre  $W$  para obtener  $(x_p, y_p, z_p)$ . Observe que  $M$  supone que el volumen de vista está en el medio espacio  $z$  negativo. Sin embargo, por cuestiones de notación, en nuestros ejemplos usaremos valores positivos decrecientes de  $z$  en lugar de valores negativos decrecientes para indicar el aumento en la distancia con respecto al observador. En cambio, muchos sistemas de graficación transforman su mundo de mano derecha a un sistema de coordenadas de vista de mano izquierda, en el cual los valores crecientes positivos de  $z$  corresponden a un aumento en la distancia con respecto al observador.

Ahora podemos continuar con la determinación de superficies visibles sin ser molestados por las complicaciones sugeridas por la figura 13.1. Por supuesto, si se especifica una proyección paralela, la transformación de perspectiva de  $M$  no es necesaria, ya que la transformación de normalización  $N_{\text{par}}$  para proyecciones paralelas hace que los proyectores sean paralelos al eje  $z$ .

### 13.1.3 Extensiones y volúmenes acotantes

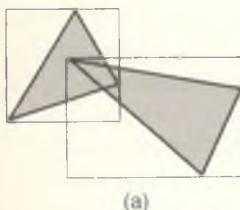
Las extensiones de pantalla, presentadas en el capítulo 3 como una manera de evitar recortes innecesarios, también son de uso común para evitar comparaciones innecesarias entre objetos o sus proyecciones. En la figura 13.4 se presentan dos objetos (polígonos tridimensionales en este caso), sus proyecciones y las extensiones rectangulares verticales de pantalla que rodean a sus proyecciones. Se supone que los objetos han sido transformados con la matriz  $M$  de transformación de perspectiva de la sección 13.1.2. Por lo tanto, en el caso de polígonos, la proyección ortográfica sobre el plano  $(x, y)$  se realiza trivialmente ignorando las coordenadas  $z$  de cada vértice. En la figura 13.4 no se sobreponen las extensiones, de manera que no hay que evaluar las proyecciones para determinar si se



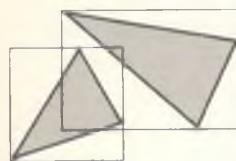
**Figura 13.4** Dos objetos, sus proyecciones sobre el plano ( $x, y$ ) y las extensiones que rodean a las proyecciones.

sobreponen. Si las extensiones se sobreponen ocurre uno de los dos casos presentados en la figura 13.5: se sobreponen también las proyecciones, como en la parte (a), o no lo hacen, como en la parte (b). En ambos casos hay que efectuar más comparaciones para determinar si las proyecciones se sobreponen. En la parte (b), las comparaciones establecerán que no hay intersección entre las proyecciones; en cierta forma, la superposición de extensiones era una falsa alarma. Las pruebas con extensiones ofrecen entonces un servicio similar al de las pruebas de rechazo trivial en los recortes.

Las pruebas con extensiones rectangulares también se conocen como pruebas de **caja acotante**. Las extensiones se pueden usar como en el capítulo 7 para rodear objetos, en lugar de sus proyecciones; en este caso, las extensiones se convierten en sólidos y también se conocen como **volumenes acotantes**. Alternativamente, las extensiones se pueden emplear para acotar una sola dimensión, por ejemplo para determinar si dos objetos se sobreponen o no en  $z$ . En la figura 13.6 se muestra la utilización de extensiones en esta situación; aquí, una extensión es el volumen infinito acotado por los valores mínimo y máximo de  $z$  para cada objeto. No hay sobreposición en  $z$  si



(a)



(b)

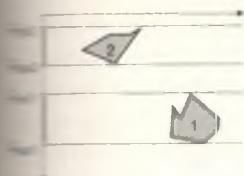
**Figura 13.5**

Extensiones que acotan proyecciones de objetos. (a) Las extensiones y las proyecciones se sobreponen. (b) Las extensiones se sobreponen, pero no las proyecciones.

$$z_{\max 2} < z_{\min 1} \quad \text{o} \quad z_{\max 1} < z_{\min 2}.$$

La comparación con cotas mínimas y máximas en una o más dimensiones se conoce también como prueba **minmax**. Al comparar extensiones minmax, la parte más complicada del trabajo es encontrar la extensión. En el caso de polígonos (u otros objetos que están totalmente contenidos en la envoltura convexa de un conjunto de puntos de definición), una extensión se puede calcular iterando a través de una lista de coordenadas de puntos y registrando los valores máximos y mínimos para cada coordenada.

Las extensiones y los volúmenes acotantes no sólo se usan para comparar dos objetos o sus proyecciones, sino también para determinar si un proyecto interseca un objeto. Esto implica calcular la intersección de un punto con



**Figura 13.6**  
Utilización de extensiones  
dimensionales para  
minar si se  
encuentran objetos.

proyección bidimensional o un vector con un objeto tridimensional, como se describe en la sección 13.4.

Aunque hasta ahora sólo hemos analizado las extensiones mínmax, son posibles otros volúmenes acotantes. ¿Cuál es el mejor que se puede usar? No será ninguna sorpresa saber que la respuesta depende del trabajo de realizar las pruebas con el propio volumen acotante y de lo bien que el volumen proteja al objeto encerrado de las pruebas que no producen una intersección. Weghorst, Hooper y Greenberg [WEGH84] tratan la selección de volumen acotante como un asunto de minimización de la función de costo total  $T$  de la prueba para determinar la intersección de un objeto. Esto se puede expresar como

$$T = bB + oO, \quad (13.5)$$

donde  $b$  es el número de veces que se evalúa el volumen acotante para la intersección,  $B$  es el costo de realizar una prueba de intersección con el volumen acotante,  $o$  es el número de veces que se prueba el objeto para la intersección (el número de veces que en realidad se interseca el volumen acotante) y  $O$  es el costo de realizar una prueba de intersección con el objeto.

Como la prueba de intersección del objeto sólo se lleva a cabo cuando realmente se interseca el volumen acotante,  $a \leq b$ . Aunque  $O$  y  $b$  son constantes para un objeto y un conjunto de pruebas que se realizarán,  $B$  y  $o$  varían como función de la forma y el tamaño del volumen acotante. Un volumen acotante más apretado, que minimiza  $o$ , generalmente se asocia con una  $B$  mayor. La efectividad de un volumen acotante también puede depender de la orientación del objeto o el tipo de objetos con los cuales se intersecará el objeto en cuestión.

### 13.1.4 Eliminación de caras posteriores

Si un objeto es aproximado con un poliedro sólido, sus caras poligonales encierran completamente su volumen. Suponga que todos los polígonos se han definido de manera que las normales a sus superficies apunten hacia el exterior del poliedro. Si el plano de recorte frontal no expone el interior de ningún poliedro, aquellos polígonos cuyas normales a la superficie apuntan alejándose del observador se hallarán en una parte del poliedro cuya visibilidad está completamente bloqueada por otros polígonos más cercanos, como se observa en la figura 13.7. Estas **caras posteriores** invisibles de los polígonos se pueden eliminar del procesamiento ulterior, técnica conocida como **eliminación de caras posteriores** (*back-face culling*). Por analogía, los polígonos que no están orientados hacia atrás se denominan **caras anteriores**.

En las coordenadas del ojo, un polígono posterior se puede identificar con el producto punto no negativo que forma la normal a su superficie con el vector del centro de proyección a cualquier punto del polígono. (En un sentido estricto, el producto punto es positivo para un polígono de cara posterior; un producto punto igual a cero indica un polígono que se observa en una arista.) Suponiendo que se haya efectuado la transformación de perspectiva o que se

desea una proyección ortográfica sobre el plano  $(x, y)$ , la dirección de proyección es  $(0, 0, -1)$ . En este caso, la prueba de producto punto se reduce a la selección de un polígono como posterior únicamente si la normal a su superficie tiene coordenada  $z$  negativa. Si el ambiente consiste en un solo poliedro convexo, la eliminación de caras posteriores es el único cálculo que hay que efectuar. En caso contrario, puede haber polígonos anteriores, como  $C$  y  $E$  en la figura 13.7, que se encuentren parcial o totalmente oscurecidos.

Si todos los poliedros tienen caras frontales faltantes o recortadas, o si los polígonos no forman parte de los poliedros, aún es posible dar un tratamiento especial a los polígonos posteriores. Si no se desea la eliminación, la estrategia más sencilla es tratar un polígono posterior como si fuera anterior, invirtiendo su normal a la dirección opuesta. En PHIGS PLUS, el usuario puede especificar un conjunto completamente separado de propiedades para cada lado de una superficie.

Observe que un proyector que pasa por un poliedro interseca el mismo número de polígonos anteriores y posteriores. Por lo tanto, un punto en la proyección de un poliedro cae en las proyecciones del mismo número de polígonos posteriores que de polígonos anteriores. Por consiguiente, la eliminación de caras posteriores reduce a la mitad el número de polígonos que tendrán que considerarse para cada pixel en los algoritmos de precisión de imagen para superficies visibles. En promedio, aproximadamente la mitad de los polígonos de un poliedro son posteriores. Entonces, la eliminación de caras posteriores también reduce aproximadamente a la mitad el número de polígonos que tiene que considerar el resto de un algoritmo de precisión del objeto para superficies visibles. (Sin embargo, observe que esto sólo es verdadero en promedio. Por ejemplo, la base de una pirámide puede ser el único polígono anterior o posterior del objeto.)

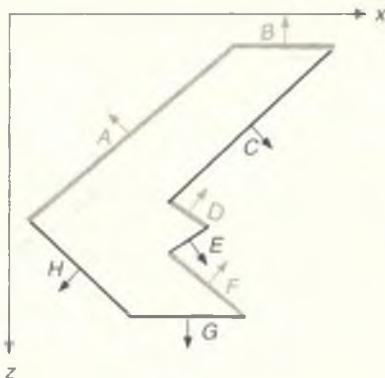


Figura 13.7

Eliminación de caras posteriores. Se eliminan los polígonos posteriores ( $A, B, D, F$ ), presentados en gris, mientras que los polígonos anteriores ( $C, E, G, H$ ) se conservan.

### 13.1.5 Partición espacial

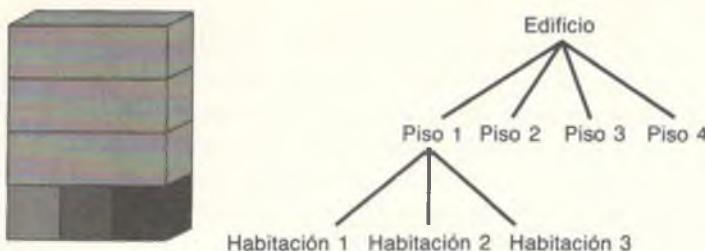
La **partición espacial** (también conocida como **subdivisión espacial**) nos permite dividir un problema grande en otros más pequeños. El método básico consiste en asignar, como paso de preprocesamiento, objetos o sus proyecciones a grupos espacialmente coherentes. Por ejemplo, podemos dividir el plano de proyección con una malla rectangular bidimensional regular y determinar en qué espacios de la malla está la proyección de cada objeto. Sólo hay que comparar las proyecciones para determinar si se sobreponen con otras proyecciones que caen dentro de sus cuadros de malla. Esta técnica es usada por [ENCA72; MAHN73; FRAN80; HEDG82]. La partición espacial también puede usarse para imponer una malla tridimensional regular a los objetos del ambiente. Así, el proceso de determinación de los objetos que intersecan un proyector se puede acelerar si se determina primero cuáles son las particiones que interseca el proyector para luego evaluar únicamente los objetos que se encuentren dentro de estas particiones (Sec. 13.4).

Si los objetos que se presentan están distribuidos de manera desigual en el espacio, puede ser más eficiente usar la **partición adaptable**, en la cual varía el tamaño de cada partición. Un método para la partición adaptable es subdividir recursivamente el espacio hasta que se cumpla un criterio de terminación para cada partición. Por ejemplo, la subdivisión puede concluir cuando haya menos que cierto número máximo de objetos en una partición [TAMM82]. Las estructuras de datos de árboles de cuadrantes, octantes y BSP de la sección 10.6 son especialmente atractivas para este fin.

### 13.1.6 Jerarquía

Como vimos en el capítulo 7, las jerarquías pueden ser útiles para relacionar la estructura y el movimiento de objetos diferentes. Un modelo jerárquico anidado, donde cada hijo se considera parte del padre, también puede servir para restringir el número de comparaciones entre objetos que requiere un algoritmo de superficies visibles [CLAR76; RUBI80; WEGH84]. Un objeto en un nivel de la jerarquía puede servir como extensión para sus hijos si éstos se encuentran totalmente contenidos en dicho nivel, como se muestra en la figura 13.8. En este caso, si no se intersecan dos objetos en la jerarquía, no es necesario evaluar los objetos de menor nivel de uno para ver si intersecan los del otro. Así mismo, hay que probar un proyector con los hijos de un objeto sólo si el proyector penetra el objeto en la jerarquía. Esta forma de utilizar la jerarquía es un ejemplo importante de la coherencia de objetos.

*En lo que resta de este capítulo analizaremos la rica variedad de algoritmos desarrollados para la determinación de superficies visibles. Aquí nos centraremos en el cálculo de cuáles partes de la superficie de un objeto son visibles, dejando la determinación del color de la superficie para el capítulo 14. Al describir cada algoritmo haremos hincapié en su aplicación a los polígonos, pero indicaremos cuándo se puede generalizar para manejar otros objetos.*



**Figura 13.8** Es posible usar la jerarquía para restringir el número de comparaciones de objetos. Sólo si un proyector interseca el edificio y el piso es necesario evaluar si hay intersección con las habitaciones 1 a 3.

## 13.2 Algoritmo de z-buffer (memoria de profundidad)

El algoritmo de precisión de imagen de **z-buffer** o **memoria de profundidad**, desarrollado por Catmull [CATM74], es uno de los algoritmos de superficies visibles más fáciles de implantar en software o en hardware. Requiere que se tenga no sólo una memoria gráfica  $F$  donde se almacenen los valores de los colores, sino también una **memoria de profundidad Z**, con el mismo número de entradas, donde se almacene un valor  $z$  para cada pixel. La memoria  $z$  se asigna inicialmente a cero, que representa el valor  $z$  en el plano de recorte posterior, mientras que la memoria gráfica se asigna igual al color de fondo. El mayor valor que se puede almacenar en la memoria de profundidad  $z$  representa la  $z$  del plano de recorte anterior. Los polígonos se discretizan a la memoria gráfica en un orden arbitrario. Durante el proceso de discretización, si el punto del polígono que se discretiza en  $(x, y)$  está más cerca o a igual distancia del observador que el punto cuyo color y profundidad están actualmente en las memorias, entonces el color y la profundidad del nuevo punto sustituyen a los valores anteriores. El seudocódigo para el algoritmo de memoria de profundidad  $z$  se presenta en el programa 13.1. Los procedimientos *escribir\_pixel* y *leer\_pixel* presentados en el capítulo 3 se complementan aquí con procedimientos *leer\_Z* y *escribir\_Z* que leen y escriben la memoria de profundidad.

### Programa 13.1

Seudocódigo para el algoritmo de memoria de profundidad  $z$ .

```
void memoria_Z
int pz;                                /* La z del polígono en las coordenadas de pixel (x, y) */
{
    for(y = 0; y < YMÁX; y++){
        for(x = 0; x < XMÁX; x++) {
            escribir_pixel (x, y, valor_fondo);
            escribir_Z (x, y, 0);
            for (polígono : polígonos_en_zona) {
                if (intersección(x, y, polígono)) {
                    if (polígono.z <= pz) {
                        pz = polígono.z;
                        escribir_pixel (x, y, polígono.color);
                    }
                }
            }
        }
    }
}
```

```

    } }

    for (cada polígono) {
        for (cada pixel en la proyección del polígono) {
            pz >= valor z del polígono en las coordenadas de pixel (x, y);
            if (pz >= leer_Z (x, y)) {           /* El punto nuevo no está más lejos */
                escribir_Z (x, y, pz);
                escribir_pixel (x, y, color del polígono en las coordenadas de pixel (x, y));
            }
        }
    }
}

```

No se requiere ningún ordenamiento previo ni comparaciones objeto-objeto. El proceso completo no es más que una búsqueda en cada conjunto de pares  $\{Z_i(x, y), F_i(x, y)\}$  de  $x$  y  $y$  fijas, para hallar la  $Z_i$  mayor. La memoria de profundidad  $z$  y la memoria gráfica registran la información relacionada con la mayor  $z$  detectada hasta el momento para cada  $(x, y)$ . Así, los polígonos aparecen en la pantalla en el orden en que se procesan. Cada polígono se puede discretizar a las memorias una línea de barrido a la vez, como se describe en la sección 3.5. En la figura 13.9 se muestra la adición de dos polígonos a la imagen. El matiz de cada pixel se indica con su color; su  $z$  con un número.

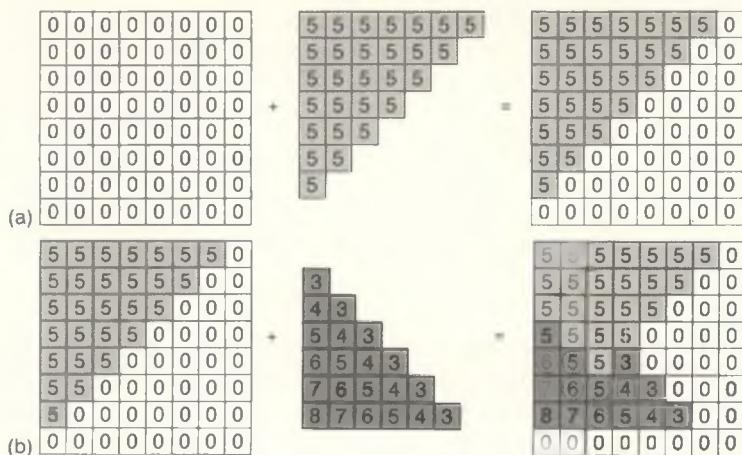
Si nos remitimos a nuestro análisis de la coherencia de profundidad, podemos simplificar el cálculo de  $z$  para cada punto en una línea de barrido aprovechando el hecho de que un polígono es plano. Normalmente, para calcular  $z$  tendríamos que resolver la ecuación de plano  $Ax + By + Cz + D = 0$  para la variable  $z$ :

$$z = \frac{-D - Ax - By}{C}. \quad (13.6)$$

Ahora, si en  $(x, y)$  la ecuación (13.6) se evalúa como  $z_1$ , entonces en  $(x + \Delta x, y)$  el valor de  $z$  es

$$z_1 - \frac{A}{C} (\Delta x). \quad (13.7)$$

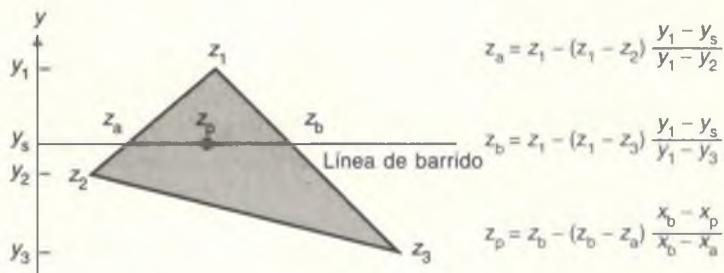
Sólo se requiere una resta para calcular  $z(x + 1, y)$  si tenemos  $z(x, y)$ , ya que el cociente  $A/C$  es constante y  $\Delta x = 1$ . Se puede efectuar un cálculo incremental similar para determinar el primer valor de  $z$  en la siguiente línea de rastreo, reduciendo cada  $\Delta y$  en  $B/C$ . Alternativamente, si no se ha determinado la superficie o si el polígono no es plano (véase la Sec. 9.1.2), podemos determinar  $z(x, y)$  interpolando las coordenadas  $z$  de los vértices del polígono a lo largo de pares de aristas y después por las líneas de rastreo, como se muestra en la figura 13.10. Aquí también se pueden utilizar cálculos incrementales. Observe que no es necesario calcular el color en un pixel si no se satisface la condición que determina su visibilidad. Por lo tanto, si el cálculo del sombreado consume mucho tiempo, se puede obtener mayor eficiencia si se lleva a cabo un ordenamiento



**Figura 13.9** Memoria de profundidad. El matiz de un pixel se indica con su color; su valor  $z$  aparece como un número. (a) Adición de un polígono de  $z$  constante a la memoria  $z$  vacía. (b) Adición de otro polígono que interseca el primero.

general de los objetos por profundidades, de anterior a posterior, para presentar primero los objetos más cercanos. El algoritmo de memoria de profundidad no requiere que los objetos sean polígonos. De hecho, uno de sus aspectos más atractivos es que se puede usar para generar la imagen de cualquier objeto si es posible determinar un matiz y un valor  $z$  para cada punto de su proyección; no hay que escribir algoritmos de intersección explícitos.

El algoritmo de  $z$ -buffer lleva a cabo ordenamientos de raíz en  $x$  y  $y$ , por lo cual no se requieren comparaciones, y su ordenamiento en  $z$  sólo requiere una comparación por pixel en cada polígono que lo contiene. El tiempo necesario para los cálculos de superficie visible tiende a ser independiente del número de polígonos en los objetos, ya que, en promedio, el número de pixeles cubiertos



**Figura 13.10** Interpolación de valores  $z$  en aristas de polígonos y líneas de rastreo.  $z_a$  se interpola entre  $z_1$  y  $z_2$ ;  $z_b$  entre  $z_1$  y  $z_3$ ;  $z_p$  entre  $z_a$  y  $z_b$ .

por cada polígono disminuye al aumentar el número de polígonos en el volumen de vista. Por lo tanto, el tamaño medio de cada conjunto de pares que se revisan tiende a permanecer fijo. Por supuesto, también hay que considerar el tiempo adicional de discretización ocasionado por los polígonos adicionales.

Aunque el algoritmo de *z-buffer* requiere mucho espacio para la memoria *z*, es fácil de implantar. Si la disponibilidad de memoria es un problema, la imagen se puede discretizar por tiras, de manera que sólo se requiera memoria de profundidad suficiente para la tira que se procesa, a expensas de efectuar varias pasadas por los objetos. Gracias a la sencillez de la memoria de profundidad y la carencia de estructuras de datos adicionales, la reducción en el costo de la memoria ha inspirado varias implantaciones en hardware y firmware de la memoria de profundidad. Sin embargo, como este algoritmo opera con precisión de imagen, está sujeto a artefactos de discretización. El algoritmo de *A-buffer* [CARP84] trata este problema usando una aproximación discreta para el muestreo de área no ponderada.

La memoria de profundidad a menudo se planta en hardware con valores enteros de 16 a 32 bits, pero las implantaciones en software (y algunas en hardware) pueden usar valores de punto flotante. Aunque una memoria de profundidad de 16 bits ofrece un intervalo adecuado para muchas aplicaciones de CAD/CAM, 16 bits no tienen la suficiente precisión para representar ambientes en los cuales los objetos definidos con posición milimétrica están ubicados a un kilómetro de distancia. Para empeorar la situación, al usar una proyección de perspectiva la compresión de los valores *z* distantes que resultan de la división de perspectiva tiene un serio efecto en el ordenamiento por profundidad y en las intersecciones de objetos distantes. Dos puntos que se transformarían a valores *z* enteros diferentes si estuvieran cerca del plano de vista podrían transformarse al mismo valor *z* si estuvieran lejanos (véase el Ejer. 13.9 y [HUGH89]).

La precisión finita de la memoria de profundidad *z* es responsable de otro problema de artefacto de discretización. Los algoritmos de discretización suelen generar dos conjuntos diferentes de pixeles al dibujar la parte común de dos aristas colineales que comienzan en puntos extremos diferentes. Algunos de los pixeles compartidos por las aristas generadas también pueden recibir valores *z* ligeramente distintos debido a las imprecisiones numéricas al realizar la interpolación *z*. Este efecto es más notorio en las aristas compartidas de las caras de un poliedro. Algunos de los pixeles visibles en una arista pueden ser parte de un polígono, mientras que los demás provienen del vecino del polígono. El problema se puede resolver insertando vértices adicionales para asegurar que los vértices ocurran en los mismos puntos a lo largo de la parte común de dos aristas colineales.

Podemos aprovechar la memoria de profundidad incluso después de generar las imágenes. Como es la única estructura de datos utilizada propiamente por el algoritmo de superficies visibles, se puede almacenar junto con la imagen y luego usarse para combinar otros objetos cuya *z* se pueda calcular. El algoritmo también se puede codificar para que el contenido de la memoria de profun-

didad no sea modificado al generar objetos seleccionados. Si la memoria de profundidad se enmascara de esta manera, un objeto se puede escribir en un conjunto separado de planos de superposición con las superficies ocultas correctamente eliminadas (si el objeto es una función de un solo valor de  $x$  y  $y$ ) y luego borrarse sin afectar el contenido de la memoria de profundidad. De esta manera, un objeto sencillo, como una malla reglada, se puede mover por la imagen en  $x$ ,  $y$  y  $z$  para servir como *cursor tridimensional* que oscurezca y oscurecido por los objetos en el ambiente. Se pueden crear vistas recortadas haciendo que las escrituras en la memoria de profundidad y en la memoria gráfica dependan de si el valor  $z$  se encuentra detrás de un plano de recorte. Si los objetos que se presentan tienen un solo valor  $z$  para cada  $(x, y)$ , el contenido de la memoria de profundidad  $z$  también se puede usar para calcular área y volumen.

Rossignac y Requicha [ROSS86] analizan cómo adaptar el algoritmo *z-buffer* para manejar objetos definidos por geometría de sólidos constructivos. Cada pixel en la proyección de una superficie se escribe sólo si está más cerca de  $z$  y sobre un objeto CSG construido a partir de la superficie. En lugar de almacenar en cada pixel solamente el punto con  $z$  más cercana, Atherton propone almacenar una lista de todos los puntos, ordenada por  $z$  y acompañada por la identidad de cada superficie, para formar una memoria de objetos [ATHE83]. Una etapa de postprocesamiento determina cómo se presenta la imagen. Procesando la lista de cada pixel se pueden obtener varios efectos, como transparencia, recortes y operaciones de conjuntos booleanos, sin tener que volver a discretizar los objetos.

### 13.3 Algoritmos de línea de barrido

Los **algoritmos de línea de barrido**, desarrollados originalmente por Wyllie, Romney, Evans y Erdahl [WYLI67], Bouknight [BOUK70a; BOUK70B] y Watkins [WATK70], operan con precisión de imagen para crear una imagen de línea de barrido a la vez. El método básico es una extensión del algoritmo de discretización de polígonos analizado en la sección 3.5, por lo que se emplean varias formas de coherencia, incluyendo la de línea de barrido y la de aristas. La diferencia es que no se trata un solo polígono, sino un conjunto de polígonos. El primer paso es crear una **tabla de aristas** para todas las aristas no horizontales de los polígonos proyectados sobre el plano de vista. Como antes, ignoran las aristas horizontales. Las entradas en la tabla de aristas se clasifican en cubetas con base en la coordenada  $y$  menor de cada arista; los datos en las cubetas se ordenan en forma ascendente de acuerdo con la coordenada  $x$  de su punto extremo inferior. Cada entrada contiene lo siguiente:

1. La coordenada  $x$  del extremo con menor coordenada  $y$ .
2. La coordenada  $y$  del otro extremo de la arista.

3. El incremento  $x$ ,  $\Delta x$ , que se usa para pasar de una línea de rastreo a la siguiente ( $\Delta x$  es la pendiente inversa de la arista).
4. El número de identificación del polígono, que indica el polígono al cual pertenece la arista.

También se requiere una **tabla de polígonos** que contenga al menos la siguiente información para cada polígono, además de su identificador:

1. Los coeficientes de la ecuación de plano.
2. La información de sombreado o color para el polígono.
3. Una bandera booleana de entrada-salida, con valor inicial *falso*, que se usa en el procesamiento de línea de barrido.

En la figura 13.11 se muestra la proyección de dos triángulos sobre el plano  $(x, y)$ ; las aristas ocultas se presentan como líneas discontinuas. La tabla de aristas clasificada para esta figura contiene entradas para  $AB$ ,  $AC$ ,  $FD$ ,  $FE$ ,  $CB$  y  $DE$ . La tabla de polígonos contiene entradas para  $ABC$  y  $DEF$ .

Aquí también es necesaria la **tabla de aristas activas** (AET) utilizada en la sección 3.5. Siempre se mantiene en orden creciente de  $x$ . En la figura 13.12 se presentan las entradas de las tablas de aristas, de polígonos y de aristas activas. Cuando el algoritmo ha avanzado hasta la línea de rastreo  $y = \alpha$ , la AET contiene  $AB$  y  $AC$ , en ese orden. Las aristas se procesan de izquierda a derecha. Para procesar  $AB$ , primero se invierte la bandera de entrada-salida del polígono  $ABC$ . En este caso, la bandera será *verdadera*; por lo tanto, el barrido está ahora *dentro* del polígono y por ende éste debe considerarse. Ahora, como el barri-

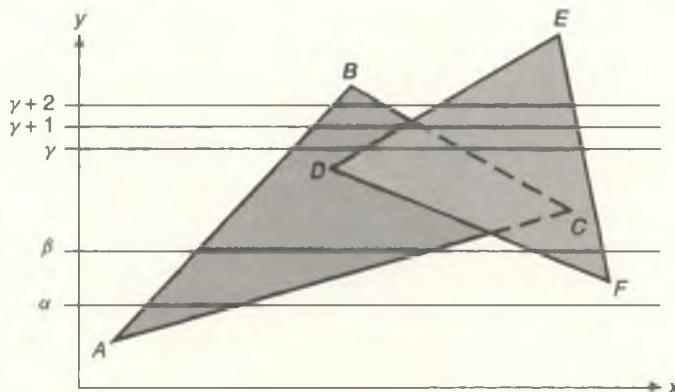


Figura 13.11 Dos polígonos procesados por un algoritmo de línea de barrido.

Entrada de la tabla de aristas	$x$	$y_{\text{mín}}$	$\Delta x$	ID		Contenido de la AET
					→	Línea de barrido Entradas
	$\alpha$					$AB \quad AC$
	$\beta$					$AB \quad AC \quad FD \quad FE$
Entrada de la tabla de polígonos	ID	Ecuación de plano	Información de sombreado	Entrada-salida		$\gamma, \gamma + 1 \quad AB \quad DE \quad CB \quad FE$
						$\gamma + 2 \quad AB \quad CB \quad DE \quad FE$

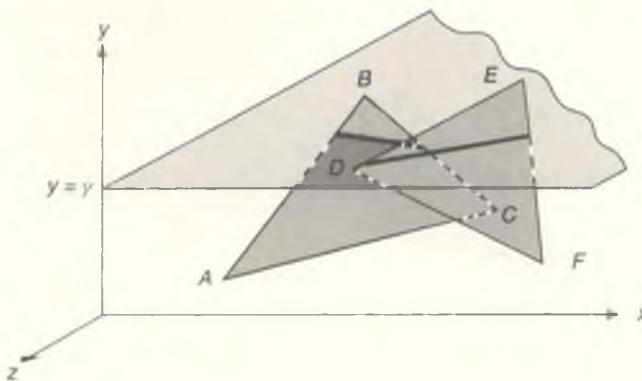
Figura 13.12 Tabla de aristas, tabla de polígonos y AET para el algoritmo de línea de barrido.

do está *dentro* de un solo polígono ( $ABC$ ), debe ser visible, de manera que el sombreado de  $ABC$  se aplica al *tramo* de la arista  $AB$  a la siguiente arista en la AET, la arista  $AC$ . Éste es un ejemplo de la coherencia de tramos. Al llegar a esta arista, la bandera de  $ABC$  se invierte a falso, de manera que el rastreo ya no está *dentro* de ningún polígono. Así mismo, como  $AC$  es la última arista de la AET, termina el procesamiento de la línea de barrido. La AET se actualiza a partir de la tabla de aristas y se ordena una vez más con respecto a  $x$ , ya que pueden haberse cruzado algunas de sus aristas, y se procesa la siguiente línea de barrido.

Cuando se detecta la línea de barrido  $y = \beta$ , la AET ordenada es  $AB, AC, FD$  y  $FE$ . El procesamiento continúa casi como antes. Hay dos polígonos en la línea de rastreo, pero el barrido sólo está *dentro* de un polígono en un instante determinado.

Para la línea de barrido  $y = \gamma$  las cosas son más interesantes. Al ingresar  $ABC$  su bandera se convierte en *verdadero*. La sombra de  $ABC$  se usa para el tramo hasta la siguiente arista,  $DE$ . Al llegar a este punto, la bandera de  $DEF$  también se convierte en *verdadero*, de manera que el barrido se halla *dentro* de los dos polígonos. (Es conveniente mantener una lista explícita de los polígonos cuya bandera de entrada-salida sea *verdadero*, y también llevar un recuento de la cantidad de polígonos que hay en la lista.) Ahora debemos decidir si  $ABC$  o  $DEF$  está más cerca del observador, lo cual se determina evaluando las ecuaciones de plano de ambos polígonos para  $z$  en  $y = \gamma$  y con  $x$  igual a la intersección de  $y = \gamma$  con la arista  $DE$ . Este valor de  $x$  está en la entrada de la AET para  $DE$ . En nuestro ejemplo,  $DEF$  tiene  $z$  mayor y por ende es visible. Por lo tanto, suponiendo que no son polígonos interpenetrantes, el sombreado de  $DEF$  se aplica para el tramo a la arista  $CB$ , en cuyo punto la bandera de  $ABC$  se convierte en *falso* y el rastreo una vez más está *dentro* de un solo polígono,  $DEF$ , cuyo sombreado se sigue usando hasta la arista  $FE$ . En la figura 13.13 se muestra la relación entre los dos polígonos y el plano  $y = \gamma$ ; las dos líneas gruesas son las intersecciones de los polígonos con el plano.

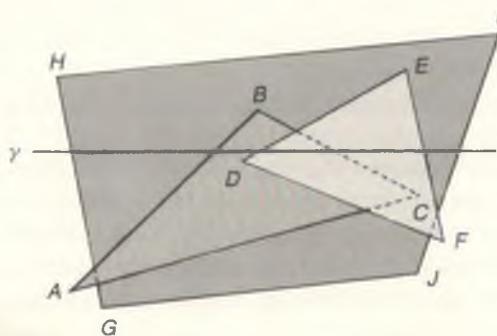
Suponga que hay un polígono de gran tamaño  $GHIJ$  detrás de  $ABC$  y  $DEF$ , como en la figura 13.14. Entonces, cuando la línea de barrido  $y = \gamma$  llega a la arista  $CB$ , el rastreo aún está *dentro* de los polígonos  $DEF$  y  $GHIJ$ , de manera que



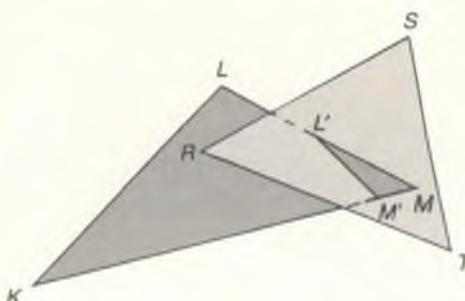
**Figura 13.13** Intersecciones de los polígonos  $ABC$  y  $DEF$  con el plano  $y = \gamma$ .

que se realizan de nuevo los cálculos de profundidad. Sin embargo, podemos evitar estos cálculos si suponemos que ninguno de los polígonos penetra en otro. Esta suposición quiere decir que, cuando el barrido sale de  $ABC$ , la relación de profundidad entre  $DEF$  y  $GHIJ$  no puede cambiar, por lo cual  $DEF$  sigue estando enfrente. Por consiguiente, los cálculos de profundidad no son necesarios cuando el barrido sale de un polígono oscurecido, y se requieren únicamente al salir de un polígono oscurecedor.

Para usar este algoritmo adecuadamente con polígonos penetrantes, como se ilustra en la figura 13.15, se divide  $KLM$  en  $KLL'M'$  y  $L'MM'$ , introduciendo la *arista falsa*  $M'L'$ . Alternativamente, se puede modificar el algoritmo para encontrar el punto de penetración en una línea de barrido cuando se procesa ésta.



**Figura 13.14** Tres polígonos no penetrantes. No es preciso realizar cálculos de profundidad cuando la línea de barrido  $\gamma$  sale del polígono tapado  $ABC$ , ya que los polígonos no penetrantes mantienen su orden  $z$  relativo.



**Figura 13.15** El polígono  $KLM$  perfora el polígono  $RST$  en la línea  $L'M'$ .

Otra variante de este algoritmo usa la *coherencia de profundidad*. Suponiendo que los polígonos no se penetren, Romney notó que, si las mismas aristas están en la AET para una línea de rastreo y la línea de rastreo inmediatamente anterior, y que si están en el mismo orden, no han ocurrido cambios en las relaciones de profundidad en ninguna parte de la línea de rastreo y por ende no se requieren nuevos cálculos de profundidad [ROMN69]. El centro de tramos visibles en la línea de barrido anterior define los tramos en la actual. Esto se aplica a las líneas de barrido  $y = \gamma$  y  $y = \gamma + 1$  de la figura 13.11, en las cuales los tramos de  $AB$  a  $DE$  y de  $DE$  a  $FE$  son visibles. Sin embargo, la coherencia de profundidad en esta figura se pierde al pasar de  $y = \gamma + 1$  a  $y = \gamma + 2$ , ya que las aristas  $DE$  y  $CB$  cambian de orden en la AET (una situación que debe tener presente el algoritmo). Por lo tanto, cambian los tramos visibles y, en este caso, se convierten en  $AB$  a  $CB$  y  $DE$  a  $FE$ . Hamlin Gear [HAML77] muestran cómo se puede mantener la coherencia de profundidad incluso si las artistas cambian de orden en la AET.

Aún no hemos visto cómo tratar el fondo. La manera más sencilla consiste en asignar el color de fondo como valor inicial de la memoria gráfica, para que el algoritmo sólo tenga que procesar las líneas de rastreo que intersecan aristas. Otra manera de hacerlo es incluir en la definición de las escenas un polígono de tamaño suficiente y más atrás que los demás, que sea paralelo al plano de proyección y que tenga el sombreado deseado. Una última alternativa es modificar el algoritmo para colocar el color de fondo explícitamente en la memoria gráfica cuando el rastreo no esté *dentro* de ningún polígono.

Aunque los algoritmos que hemos presentado hasta ahora sólo tratan polígonos, el método de línea de barrido se ha utilizado considerablemente para superficies más generales, como se describe en la sección 13.5.3. Para ello, la tabla de aristas y la AET se sustituyen con una **tabla de superficies** y una **tabla de superficies activas**, ordenadas de acuerdo con las extensiones  $(x, y)$  de las superficies. Cuando una superficie se mueve de la tabla de superficies a la tabla de superficies activas, se puede efectuar procesamiento adicional. Por ejemplo,

la superficie se puede descomponer en un conjunto de polígonos aproximantes, los cuales se descartarían cuando el rastreo salga de la extensión y de la superficie; con esto se elimina la necesidad de mantener los datos de superficies durante todo el proceso de generación de la imagen. El seudocódigo para este algoritmo general de línea de barrido se presenta en el programa 13.2. Atherton [ATHE83] analiza un algoritmo de línea de barrido que genera objetos poligonales combinados usando las operaciones regularizadas de conjuntos booleanos de la geometría sólida constructiva.

**Programa 13.2**  
**Seudocódigo para un**  
**algoritmo general de línea**  
**de barrido.**

```

añadir superficies a la tabla de superficies;
asignar valores iniciales a la tabla de superficies activas;

for (cada línea de barrido) {
    actualizar tabla de superficies activas;

    for (cada pixel en la línea de barrido) {
        determinar las superficies en la tabla de superficies activas que se proyectan al pixel;
        encontrar la más cercana de estas superficies;
        determinar el matiz de la superficie más cercana en el pixel;
    }
}

```

Un método de línea de barrido muy atractivo por su sencillez utiliza una memoria de profundidad para resolver el problema de superficies visibles [MYER75]. Para cada línea de barrido se limpian una memoria gráfica para una sola línea de barrido y una memoria de profundidad, las cuales se usan para acumular los tramos. Como sólo se requiere almacenamiento para una línea de rastreo, es fácil acomodar imágenes de muy alta resolución.

## 13.4 Trazo de rayos en superficies visibles

La **traza de rayos**, también conocido como **lanzamiento de rayos**, determina la visibilidad de las superficies trazando rayos luminosos imaginarios del ojo del observador a los objetos en la escena.<sup>2</sup> Éste es precisamente el algoritmo prototípico de precisión de imagen que se analizó al iniciar el capítulo. Se seleccionan un centro de proyección (el ojo del observador) y una ventana en un plano de vista arbitrario. La ventana se puede considerar como una malla regular cuyos elementos corresponden a los pixeles con la resolución deseada. Entonces, para

<sup>2</sup> Aunque *traza de rayos* y *lanzamiento de rayos* muchas veces se usan como sinónimos, en ocasiones el término *lanzamiento de rayos* se usa únicamente para referirse al algoritmo de superficies visibles de esta sección, mientras que *traza de rayos* se reserva para el algoritmo recursivo de la sección 14.7.

cada pixel en la ventana, se dispara un **rayo ocular** que va del centro de proyección a la escena, a través del centro del pixel, como se muestra en la figura 13.16. El color del pixel se asigna igual al del objeto en el punto de intersección más cercano. El seudocódigo para este sencillo trazador de rayos se presenta en el programa 13.3.

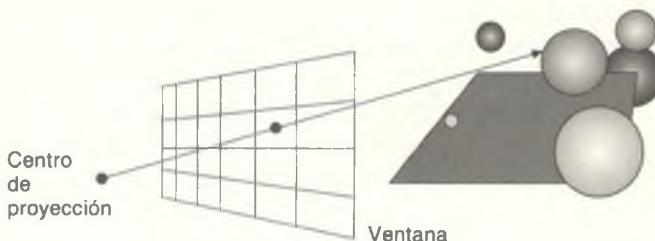
**Programa 13.3**  
Seudocódigo para un trazador de rayos sencillo.

```

seleccionar el centro de proyección y una ventana en el plano de vista;
for (cada línea de rastreo en la imagen) {
    for (cada pixel en la línea de rastreo) {
        determinar el rayo del centro de proyección a través del pixel;
        for (cada objeto en la escena) {
            if (se interseca el objeto y es el más cercano considerado hasta ahora)
                registrar intersección y nombre del objeto;
        }
        asignar el color del pixel igual al de la intersección del objeto más cercano;
    }
}

```

La traza de rayos fue desarrollado por primera vez por Appel [APPE68] y por Goldstein y Nagel [MAGI68; GOLD71]. Appel utilizó una malla de rayos abierta para determinar el sombreado, incluyendo si un punto estaba en la sombra. Goldstein y Nagel usaron originalmente su algoritmo para simular la trayectoria de proyectiles balísticos y partículas nucleares; sólo después lo aplicaron a los gráficos. Appel fue el primero en usar la traza de rayos para sombras, mientras que Goldstein y Nagel fueron los pioneros de la utilización de la traza de rayos para evaluar operaciones de conjuntos booleanos. Whitted [WHIT80] y Kajiya [KAY79a] extendieron la traza de rayos para manejar la refracción y la reflexión especulares. Analizaremos las sombras, la reflexión y la refracción —efectos por los que mejor se conoce la traza de rayos— en la sección 14.7, donde describiremos un algoritmo recursivo completo de traza de rayos que integra la determinación de superficies visibles y el sombreado. Por el momento, sólo usaremos la traza de rayos como algoritmo de superficies visibles.



**Figura 13.16** Se dispara un rayo del centro de proyección a través de cada pixel de la ventana, para determinar el objeto intersecado más cercano.

### 13.4.1 Cálculo de intersecciones

La base de toda traza de rayos es la tarea de determinar la intersección de un rayo y un objeto. Para ello se emplea la misma representación paramétrica de vectores que se analizó en el capítulo 3. Cada punto  $(x, y, z)$  en el rayo de  $(x_0, y_0, z_0)$  a  $(x_1, y_1, z_1)$  está definido por un valor  $t$  tal que

$$x = x_0 + t(x_1 - x_0), \quad y = y_0 + t(y_1 - y_0), \quad z = z_0 + t(z_1 - z_0). \quad (13.8)$$

Por conveniencia, definimos  $\Delta x$ ,  $\Delta y$  y  $\Delta z$  de manera que

$$\Delta x = x_1 - x_0, \quad \Delta y = y_1 - y_0, \quad \Delta z = z_1 - z_0. \quad (13.9)$$

Así,

$$x = x_0 + t\Delta x, \quad y = y_0 + t\Delta y, \quad z = z_0 + t\Delta z. \quad (13.10)$$

Si  $(x_0, y_0, z_0)$  es el centro de proyección y  $(x_1, y_1, z_1)$  es el centro de un pixel en la ventana,  $t$  varía entre 0 y 1 entre estos puntos. Los valores negativos de  $t$  representan puntos detrás del centro de proyección, mientras que los valores de  $t$  mayores que 1 corresponden a puntos en el lado de la ventana que está más lejos del centro de proyección. Necesitamos encontrar una representación para cada tipo de objeto que nos permita determinar  $t$  en la intersección del objeto con el rayo. Uno de los objetos más fáciles para hacerle esto es la esfera, ¡lo cual nos indica el por qué de la pléthora de esferas que se observa en las imágenes de rastreo de rayos! La esfera con centro  $(a, b, c)$  y radio  $r$  se puede representar con la ecuación

$$(x - a)^2 + (y - b)^2 + (z - c)^2 = r^2. \quad (13.11)$$

La intersección se determina expandiendo la ecuación (13.11) y restando los valores de  $x$ ,  $y$  y  $z$  a la ecuación (13.10), para obtener

$$x^2 - 2ax + a^2 + y^2 - 2by + b^2 + z^2 - 2cz + c^2 = r^2, \quad (13.12)$$

$$(x_0 + t\Delta x)^2 - 2a(x_0 + t\Delta x) + a^2 + (y_0 + t\Delta y)^2 - 2b(y_0 + t\Delta y) + b^2 + (z_0 + t\Delta z)^2 - 2c(z_0 + t\Delta z) + c^2 = r^2, \quad (13.13)$$

$$+ (x_0^2 + 2x_0\Delta xt + \Delta x^2t^2 - 2ax_0 - 2a\Delta xt + a^2$$

$$+ y_0^2 + 2y_0\Delta yt + \Delta y^2t^2 - 2by_0 - 2b\Delta yt + b^2$$

$$+ z_0^2 + 2z_0\Delta zt + \Delta z^2t^2 - 2cz_0 - 2c\Delta zt + c^2 = r^2.$$

Al agrupar los términos se obtiene

$$(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2 + 2t[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)] \quad (13.15)$$

$$+ (x_0^2 - 2ax_0 + a^2 + y_0^2 - 2by_0 + b^2 + z_0^2 - 2cz_0 + c^2) - r^2 = 0,$$

$$(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2 + 2t[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)] \quad (13.16)$$

$$+ (x_0 - a)^2 + (y_0 - b)^2 + (z_0 - c)^2 - r^2 = 0.$$

La ecuación (13.16) es cuadrática con respecto a  $t$ , con coeficientes que se presan en su totalidad con constantes derivadas de las ecuaciones de la esfera y el rayo, de manera que se puede resolver usando la fórmula cuadrática. Si no hay raíces reales, el rayo y la esfera no se intersecan; si hay una raíz real, el rayo roza la esfera. En caso contrario, las dos raíces son los puntos de intersección con la esfera; el punto que genera la menor  $t$  positiva es el más cercano. También es conveniente normalizar el rayo para que la distancia de  $(x_0, y_0, z_0)$  a  $(x_1, y_1, z_1)$  sea 1. Con esto se obtiene un valor de  $t$  que mide distancias en unidades de coordenadas mundiales y simplifica el cálculo de intersecciones, ya que el coeficiente de  $t^2$  en la ecuación (13.16) es 1. Podemos obtener en forma similar la intersección de un rayo con las superficies cuádricas generales presentadas en el capítulo 9.

Como veremos en el capítulo 14, hay que determinar la normal a la superficie en el punto de intersección para poder sombrear la superficie. Esta tarea es muy sencilla en el caso de una esfera, ya que la normal (no normalizada) es el vector del centro al punto de intersección. La esfera con centro  $(a, b, c)$  tiene una normal a la superficie  $((x - a)/r, (y - b)/r, (z - c)/r)$  en el punto de intersección  $(x, y, z)$ .

Es un poco más difícil encontrar la intersección de un rayo con un polígono. Podemos determinar dónde interseca el rayo al polígono verificando primero si el rayo interseca el plano del polígono y si el punto de intersección está dentro del polígono. Como la ecuación de un plano es

$$Ax + By + Cz + D = 0, \quad (13.17)$$

al sustituir con la ecuación (13.10) se obtiene

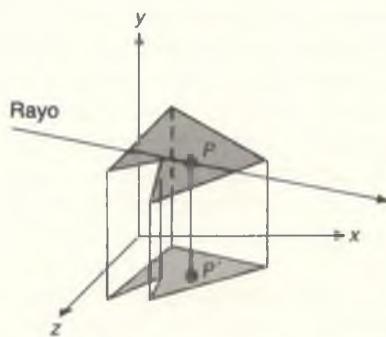
$$A(x_0 + t\Delta x) + B(y_0 + t\Delta y) + C(z_0 + t\Delta z) + D = 0, \quad (13.18)$$

$$t(A\Delta x + B\Delta y + C\Delta z) + (Ax_0 + By_0 + Cz_0 + D) = 0, \quad (13.19)$$

$$t = -\frac{(Ax_0 + By_0 + Cz_0 + D)}{[A\Delta x + B\Delta y + C\Delta z]}. \quad (13.20)$$

Si el denominador de la ecuación (13.20) es 0, entonces el rayo y el plano son paralelos y no se intersecan. Una forma sencilla de determinar si el punto de intersección cae dentro del polígono es proyectar el polígono y el punto ortográficamente a uno de los tres planos que definen el sistema de coordenadas, como se muestra en la figura 13.17. Para obtener los resultados más precisos, debemos seleccionar el eje que produzca la proyección más larga. Esto corresponde a la coordenada cuyo coeficiente en la ecuación de plano del polígono tiene el mayor valor absoluto. La proyección ortográfica se lleva a cabo eliminando esta coordenada de los vértices del polígono y del punto. La verificación de si el punto está contenido en el polígono se puede realizar entonces totalmente en dos dimensiones, usando el algoritmo de punto en polígono presentado en la sección 7.11.2.

Como en el algoritmo de *z-buffer*, la traza de rayos tiene el atractivo de que la única operación de intersección que se realiza es la del proyector con un objeto. No es necesario determinar la intersección directa de dos objetos en la escena. El algoritmo de *z-buffer* aproxima un objeto como un conjunto de valores *z* sobre los proyectores que lo intersecan. La traza de rayos aproxima objetos como el conjunto de las intersecciones sobre cada proyector que interseca la escena. Podemos extender el algoritmo de *z-buffer* para manejar un nuevo tipo de objeto si escribimos una rutina de discretización y de cálculo de *z*. Así mismo, podemos extender la traza de rayos de superficies visibles para manejar un nuevo tipo de objeto si escribimos una rutina de intersección de rayos para él. En ambos casos, también es necesario escribir una rutina que calcule las normales de las superficies para el sombreado. Se han desarrollado algoritmos de intersección y de normales a superficies para superficies algebraicas [HANR83] y paramétricas [KAJI82; SEDE84; TOTH85; JOY86]. Estos algoritmos se estudian en [HAIN89; HANR89].



**Figura 13.17** Determinación de si un rayo interseca un polígono. El polígono y el punto de intersección  $P$  del rayo con el plano del polígono se proyectan sobre uno de los tres planos que definen el sistema de coordenadas. Después se evalúa el punto proyectado  $P'$  para determinar si está contenido en el polígono proyectado.

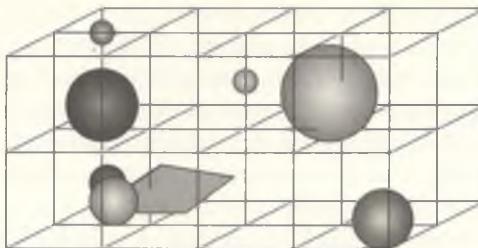
### 13.4.2 Consideraciones de eficiencia para la traza de rayos en superficies visibles

En cada pixel, el algoritmo de memoria de profundidad  $z$  sólo calcula la información de aquellos objetos que se proyectan al pixel, aprovechando la coherencia. En cambio, la versión sencilla pero costosa del algoritmo de traza de rayos en superficies visibles que analizamos interseca cada uno de los rayos del pixel con cada uno de los objetos en la escena. Por lo tanto, una imagen de 1024x1024 de 100 objetos requeriría 100 millones de cálculos de intersección. No es sorprendente saber que Whitted encontró que un 75 a 95 por ciento del tiempo en su sistema se dedicaba a la rutina de intersección para escenas típicas [WHIT80]. Por lo tanto, las estrategias para mejorar la eficiencia del rastreo de rayos en superficies visibles que analizaremos a continuación tratan de acelerar los cálculos de las intersecciones individuales o evitarlos por completo. Como veremos en la sección 14.7, los rastreadores de rayos recursivos rastrean rayos adicionales de los puntos de intersección para determinar el matiz de un pixel. Por ende, varias de las técnicas desarrolladas en la sección 13.1, como la transformación de perspectiva y la eliminación de caras posteriores, no tienen utilidad general, ya que no todos los rayos emanan del mismo centro de proyección.

**Optimización del cálculo de intersecciones.** Muchos de los términos en las ecuaciones de intersección objeto-rayo contienen expresiones que son constantes para toda una imagen o para un rayo en particular. Éstas se pueden calcular por adelantado, al igual que, por ejemplo, la proyección ortográfica de un polígono sobre un plano. Si se tiene cuidado y visión matemática, es posible desarrollar métodos de intersección más rápidos; incluso se puede mejorar la sencilla fórmula de intersección para una esfera que se proporcionó en la sección 13.4.1 [HAIN89]. Si transformamos los rayos para que caigan sobre el eje  $z$ , se puede aplicar la misma transformación a cada objeto candidato, para que cualquier intersección ocurra en  $x = y = 0$ . Este paso simplifica el cálculo de la intersección y permite determinar el objeto más cercano con un ordenamiento basado en  $z$ . Después se puede usar una transformación inversa y transformar de vuelta el punto de intersección para los cálculos de sombreado.

Los volúmenes acotantes representan una forma muy atractiva de reducir la cantidad de tiempo para el cálculo de intersecciones. Un objeto cuyas pruebas de intersección requieren mucho tiempo se puede encerrar en un volumen acotante cuya prueba de intersección sea menos costosa, como una esfera [WHIT80], un elipsoide [BOUV85] o un sólido rectangular [RUBI80; TOTH85]. No es necesario evaluar el objeto si el rayo no interseca el volumen acotante.

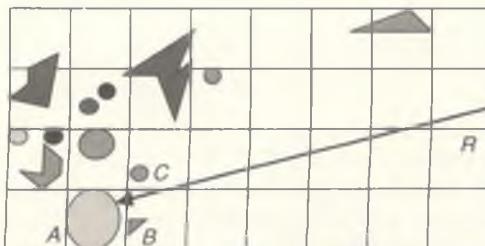
**Jerarquías.** Aunque los volúmenes acotantes en sí no determinan el orden ni la frecuencia de las pruebas de intersección, se pueden organizar en jerarquías dadas con objetos en las hojas y nodos internos que acotan a sus hijos [RUBI80; WEGH84; KAY86]. Se garantiza que un volumen hijo no intersecará un rayo si su padre no lo hace. Por lo tanto, si las pruebas de intersección comienzan en la raíz, es posible rechazar trivialmente muchas ramas de la jerarquía (y por tanto muchos objetos).



**Figura 13.18** La escena se divide en una malla regular con volúmenes de igual tamaño.

**Partición espacial.** Las jerarquías de volúmenes acotantes organizan los objetos en forma ascendente; en cambio, la partición espacial subdivisiona el espacio en forma descendente. Primero se calcula la caja acotante de la escena. En un método, la caja acotante se divide en una malla regular con extensiones de igual tamaño, como en la figura 13.18. Cada partición se asocia a una lista de objetos que contiene, ya sea en forma total o parcial. Las listas se completan asignando cada objeto a una o más particiones que lo contengan. Ahora, como se muestra en dos dimensiones en la figura 13.19, hay que intersecar un rayo sólo con los objetos contenidos en las particiones por las cuales pasa. Además, se pueden revisar las particiones en el orden en que el rayo pasa por ellas; así, tan pronto como se detecta una partición en la cual existe una intersección, no hay que inspeccionar más particiones. Observe que hay que considerar todos los objetos restantes en la partición, para determinar a cuál corresponde la intersección más cercana. Como las particiones siguen una malla regular, las particiones sucesivas a lo largo de un rayo se pueden calcular usando una versión tridimensional del algoritmo de dibujo de líneas analizado en la sección 3.2.2, modificado para listar cada partición por la cual pasa el rayo [FUJI85; AMAN87].

Si un rayo interseca un objeto en una partición, también es necesario revisar si la intersección misma cae en la partición; es posible que la intersección detec-



**Figura 13.19** Partición espacial. El rayo  $R$  sólo tiene que intersecar los objetos  $A$ ,  $B$  y  $C$ , ya que las otras particiones por las que pasa están vacías.

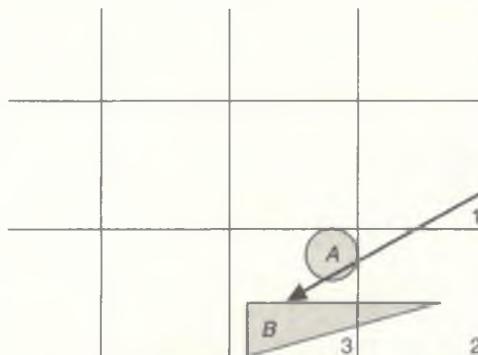
tada esté más adelante en el rayo y que exista otro objeto con intersección cercana. Por ejemplo, el objeto *B* de la figura 13.20 es intersecado en la partición 3 aunque se detectó en la partición 2. Es necesario seguir el recorrido de las particiones hasta encontrar una intersección en la partición que se esté recorriendo, en este caso *A* en la partición 3. Para evitar el recálculo de la intersección de un rayo con un objeto que está en varias particiones, la primera vez que se detecte el objeto se pueden colocar en memoria el punto de intersección, identificador de rayo y el objeto.

Dippé y Swensen [DIPP84] analizan un algoritmo de subdivisión adaptable que produce porciones de distinto tamaño. Otro método adaptable de subdivisión espacial divide la escena usando un árbol de octantes [GLAS84]. En este caso, el algoritmo de detección de vecinos presentado en la sección 10.6.3 puede servir para determinar las particiones sucesivas a lo largo de un rayo [SME89b]. Los árboles de octantes y otras particiones espaciales jerárquicas pueden considerar como casos especiales de una jerarquía en la cual se garantiza que los hijos de los nodos no se intersequen entre sí. Como estos métodos permiten la subdivisión adaptable, la decisión de subdividir aún más una partición puede depender del número de objetos en la subdivisión o del costo de interesar los objetos. Esta situación es una gran ventaja en ambientes heterogéneos con distribución no uniforme.

## 13.5 Otros métodos

### 13.5.1 Algoritmos de prioridad de listas

Los algoritmos de prioridad de listas determinan un ordenamiento de visibilidad para los objetos, asegurando que se produzca una imagen correcta si los ob-



**Figura 13.20** Un objeto puede estar intersecado en un elemento de voz distinto del actual.

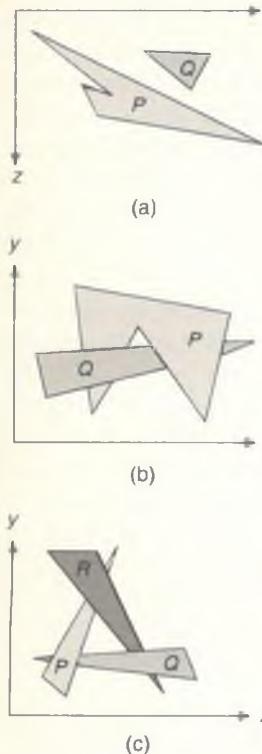
se generan en dicho orden. Por ejemplo, si ningún objeto se sobrepone a otro en  $z$ , sólo hay que ordenar los objetos en forma ascendente de acuerdo con  $z$  y luego generarlos en ese orden. Los objetos más lejanos estarán oscurecidos por los más cercanos, ya que los pixeles de los polígonos más próximos sobreescreiben los de los más distantes. Si los objetos se sobreponen en  $z$ , aún es posible determinar el orden correcto, como en la figura 13.21(a). Si los objetos se sobreponen cíclicamente, como en las figuras 13.21(b) y (c), o son interpenetrantes, no hay un orden correcto. En estos casos será necesario dividir uno o más objetos para que sea posible el ordenamiento lineal.

Los algoritmos de prioridad de listas son híbridos que combinan operaciones tanto de precisión de la imagen como de precisión del objeto. Las comparaciones de profundidad y la división de objetos se llevan a cabo con precisión de objeto. Sólo la discretización, que se basa en la capacidad del dispositivo gráfico para sobreescribir los pixeles de los objetos previamente dibujados, se realiza con precisión de imagen. Sin embargo, como la lista de objetos ordenados se crea con precisión de objeto, se puede volver a presentar correctamente a cualquier resolución. Como veremos más adelante, los algoritmos de prioridad de listas difieren en cuanto a la forma de determinar el orden, así como en lo referente a cuáles objetos se dividen y cuándo ocurre la división. No es necesario que el ordenamiento sea con base en  $z$ , se pueden dividir objetos que no se sobrepongan cíclicamente ni sean interpenetrantes, y la división se puede efectuar en forma independiente de la posición del observador.

**Algoritmo de ordenamiento por profundidad.** La idea básica del **algoritmo de ordenamiento por profundidad**, desarrollado por Newell, Newell y Sancha [NEW72], es pintar los polígonos en la memoria gráfica en orden decreciente de la distancia al punto de observación. Se realizan tres pasos conceptuales:

1. Ordenar todos los polígonos de acuerdo con la menor (más lejana) coordenada  $z$  de cada uno.
2. Resolver las ambigüedades que puede ocasionar el ordenamiento cuando las extensiones  $z$  de los polígonos se sobreponen, dividiendo polígonos de ser necesario.
3. Discretizar cada polígono en orden ascendente de la menor coordenada  $z$  (es decir, de atrás hacia adelante).

Considere la utilización de la prioridad explícita, como la que se relaciona con las vistas en SPHIGS. La prioridad explícita ocupa el lugar del valor mínimo de  $z$ , y no puede haber ambigüedades con respecto a la profundidad porque cada prioridad se considera correspondiente a un plano distinto de  $z$  constante. Esta versión simplificada del algoritmo de ordenamiento por profundidad también se conoce como **algoritmo del pintor**, haciendo referencia a cómo un pintor pintaría los objetos cercanos encima de los lejanos. Los ambientes cuyos ob-



**Figura 13.21**  
Algunos casos en los cuales se sobreponen las extensiones  $z$  de los polígonos.

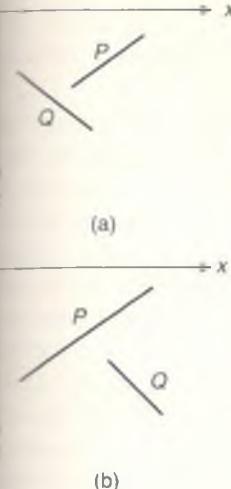
tos existen en un plano de  $z$  constante, como la distribución VLSI, la cartografía y la administración de ventanas, se consideran de  $2\frac{1}{2}$  dimensiones y se pueden manejar correctamente con el algoritmo del pintor. Este algoritmo se puede aplicar a una escena en la cual los polígonos no estén incorporados en un plano de  $z$  constante, ordenando los polígonos de acuerdo con su coordenada  $z$  mínima o por la coordenada  $z$  de su centroide, e ignorando el paso 2. Se pueden construir algunas escenas con este enfoque, pero por lo general no produce ordenamiento correcto.

En la figura 13.21 se presentan algunos tipos de ambigüedades que deben resolverse como parte del paso 2. ¿Cómo se hace esto? Llámemos  $P$  al polígono que actualmente se encuentra en el extremo lejano de la lista ordenada de polígonos. Antes de discretizar este polígono a la memoria gráfica, debe compararse con cada polígono  $Q$  cuya extensión  $z$  se sobreponga a la extensión  $z$  de  $P$ , para demostrar que  $P$  no puede oscurecer a  $Q$  y que por lo tanto se puede esconder  $P$  antes que  $Q$ . Se realizan hasta cinco pruebas, en orden creciente de complejidad. En cuanto una de ellas tiene éxito, se ha demostrado que  $P$  no oscurece a  $Q$  y se prueba el siguiente polígono  $Q$  que se sobreponga a  $P$  en  $z$ . Si aprueban todos estos polígonos, se discretiza  $P$  y el siguiente polígono en la lista se convierte en la nueva  $P$ . Las cinco pruebas son las siguientes:

1. ¿No se sobreponen las extensiones  $x$  de los polígonos?
2. ¿No se sobreponen las extensiones  $y$  de los polígonos?
3. ¿Está  $P$  totalmente del lado opuesto del plano de  $Q$  desde el punto de observación? [Esto no sucede en la figura 13.21(a), pero sí en la figura 13.22(a).]
4. ¿Está  $Q$  totalmente en el mismo lado del plano de  $P$  desde el punto de observación? [Esto no sucede en la figura 13.21(a), pero sí en la figura 13.22(b).]
5. ¿No se sobreponen las proyecciones de los polígonos en el plano ( $x, y$ )? (Esto se puede determinar comparando las aristas de un polígono con las aristas del otro.)

Si fracasan las cinco pruebas, podemos suponer por el momento que  $P$  en realidad oscurece a  $Q$  y por lo tanto probamos si es posible discretizar  $Q$  antes que  $P$ . No es necesario repetir las pruebas 1, 2 y 5, pero se emplean nuevas versiones de las pruebas 3 y 4, invirtiendo los polígonos:

- 3'. ¿Está  $Q$  totalmente del lado opuesto del plano de  $P$  desde el punto de observación?
- 4'. ¿Está  $P$  totalmente en el mismo lado del plano de  $Q$  desde el punto de observación?

**Figura 13.22**

Possibles orientaciones de polígonos. (a) La prueba 3 es verdadera. (b) La prueba 3 es falsa, pero la prueba 4 es verdadera.

En el caso de la figura 13.21(a), la prueba 3' tiene éxito. Por lo tanto, movemos  $Q$  al final de la lista y se convierte en la nueva  $P$ . Sin embargo, en el caso de la figura 13.21(b) las pruebas no son concluyentes; de hecho, no existe ningún orden con el cual se puedan discretizar correctamente  $P$  y  $Q$ . Por el contrario, tenemos que dividir  $P$  o  $Q$  en el plano del otro (véase la sección 3.11 acerca del recorte de polígonos, usando la arista de corte como plano de corte). Se descarta el polígono original, no dividido, se introducen sus partes en la lista en el orden  $z$  correcto y continúa el algoritmo.

En la figura 13.21(c) se muestra un caso más sutil. Es posible que  $P$ ,  $Q$  y  $R$  estén orientados de manera que siempre se pueda mover uno de los polígonos al final de la lista para colocarlo en el orden correcto con respecto a uno de los dos polígonos restantes, pero no ambos. El resultado de esto sería un ciclo infinito. Para evitar la formación de lazos hay que modificar la estrategia, marcando cada polígono que se mueva al final de la lista. Así, si las cinco pruebas fracasan y está marcado el polígono  $Q$  actual, no se llevan a cabo las pruebas 3' y 4', sino que se divide  $P$  o  $Q$  (como si hubieran fracasado las pruebas 3' y 4') y se vuelven a insertar las partes.

**Árboles binarios para partición de espacio.** El algoritmo de árbol binario para partición de espacio (*BSP binary space partitioning*) fue desarrollado por Fuchs, Kedem y Naylor [FUCH80; FUCH83] con base en el trabajo de Schumacker [SCHU69]. El algoritmo de árbol BSP es muy eficiente para calcular las relaciones de visibilidad entre un grupo estático de polígonos tridimensionales vistos desde un punto arbitrario. Hace un compromiso entre un paso inicial de preprocessamiento que consume mucho tiempo y espacio, y un algoritmo de presentación lineal que se ejecuta siempre que se desea una nueva especificación de visualización. De esta manera, el algoritmo es muy apropiado para aplicaciones en las cuales cambia el punto de observación pero no los objetos.

El algoritmo de árbol BSP se basa en la observación de que un polígono se discretizará correctamente (es decir, no se sobrepondrá ni se le sobrepondrán de manera incorrecta otros polígonos) si se discretizan primero todos los polígonos que están al otro lado, con respecto al observador, seguidos por el polígono en cuestión y finalmente por los polígonos que están del mismo lado que el observador. Necesitamos asegurar que esta situación se aplique a todos los polígonos.

Con el algoritmo es fácil determinar un orden correcto para la discretización, construyendo un árbol binario de polígonos, el **árbol BSP**. La raíz del árbol BSP es un polígono seleccionado entre aquellos que se presentarán; el algoritmo funciona correctamente sin importar cuál se seleccione. El polígono raíz sirve para dividir el ambiente en dos medios espacios. Un medio espacio contiene los polígonos restantes frente al polígono raíz, con respecto a la normal a su superficie; el otro contiene los polígonos que se encuentran detrás del polígono raíz. Cualquier polígono que se encuentre a ambos lados del plano del polígono raíz es dividido por el plano y sus partes anterior y posterior se asignan al medio espacio apropiado. Un polígono del medio espacio anterior del polígono raíz y otro del medio espacio posterior se convierten en los hijos anterior

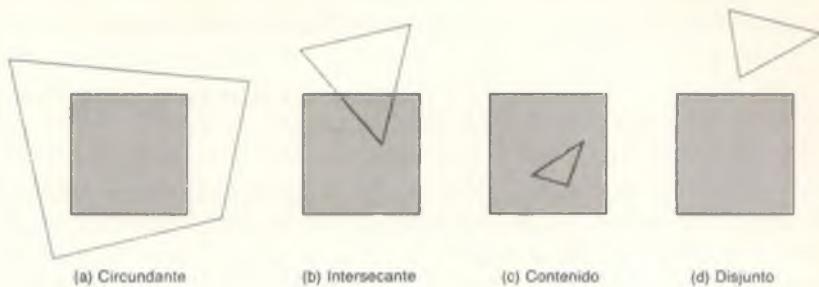
y posterior, y cada uno de estos hijos se usa recursivamente para dividir en forma similar los demás polígonos en su medio espacio. El algoritmo termina cuando cada nodo contiene un solo polígono.

De manera sorprendente, el árbol BSP se puede recorrer usando un recorrido en-orden modificado para producir una lista de polígonos correctamente ordenada por prioridad para un punto de observación arbitrario. Consideremos el polígono raíz. Divide los polígonos restantes en dos conjuntos, cada uno de los cuales cae completamente en un lado del plano de raíz. De esta manera, el algoritmo sólo tiene que garantizar que los conjuntos se presenten en el orden relativo correcto, para asegurar que los polígonos de un conjunto no interfieran con los del otro y que el polígono raíz se dibuje adecuadamente y en el orden correcto con respecto a los demás. Si el observador se halla en el medio espacio anterior del polígono raíz, el algoritmo deberá dibujar primero todos los polígonos del medio espacio posterior (los que serían oscurecidos por la raíz), luego el polígono raíz y finalmente todos los polígonos en el medio espacio anterior (los que podrían oscurecer a la raíz). De manera alternativa, si el observador se encuentra está en el medio espacio posterior del polígono raíz, el algoritmo debe dibujar primero todos los polígonos en el medio espacio anterior de la raíz, después el polígono raíz y por último los polígonos en el medio espacio posterior. Si el polígono se observa sobre la arista, cualquier orden de presentación es suficiente. La eliminación de caras posteriores se puede lograr no presentando los polígonos si el ojo está en su medio espacio posterior. El algoritmo procesa recursivamente cada uno de los hijos de la raíz. El pseudocódigo para la fase de construcción del árbol y la fase de presentación aparece en [FOLE90].

Al igual que el algoritmo de ordenamiento por profundidad, el algoritmo de árbol BSP lleva a cabo la intersección y el ordenamiento totalmente con precisión de objeto y se apoya en la capacidad de sobreescritura en precisión de imagen del dispositivo de barrido. Sin embargo, a diferencia del algoritmo de ordenamiento por profundidad, lleva a cabo toda la división de polígonos en un paso de preprocessamiento que sólo debe repetirse cuando cambia el ambiente. Observe que puede ocurrir más división de polígonos que en el algoritmo de ordenamiento por profundidad.

Los algoritmos de prioridad de listas permiten usar discretizadores de polígonos en hardware que suelen ser mucho más rápidos que los que revisan la z en cada pixel. Los algoritmos de ordenamiento por profundidad y de árbol BSP presentan los polígonos en orden posterior-anterior, quizás oscureciendo después los más distantes. Por ello, como en el algoritmo de *z-buffer*, los cálculos de sombreado pueden realizarse más de una vez por pixel. Como alternativa, los polígonos se pueden presentar en orden anterior-posterior y sólo escribir un pixel de un polígono si no se ha escrito antes.

Si se emplea un algoritmo de prioridad de listas para la eliminación de líneas ocultas, hay que prestar atención especial a las nuevas aristas creadas en el proceso de subdivisión. Si estas aristas se discretizan como las aristas del polígono original, aparecerán en la imagen como artefactos indeseados y por lo tanto se deben marcar con banderas para que no se discreticen.



**Figura 13.23** Cuatro relaciones de proyecciones de polígonos a un elemento de área: (a) circundante, (b) intersecante, (c) contenido y (d) disjunto.

### 13.5.2 Algoritmos de subdivisión de área

Todos los **algoritmos de subdivisión de área** siguen la estrategia “divide y vencerás” de partición espacial en el plano de proyección. Se examina un área de la imagen proyectada; si es fácil decidir cuáles son los polígonos visibles en el área, éstos se dibujan. En caso contrario, el área se subdivide en áreas más pequeñas a las cuales se aplica recursivamente la lógica de decisión. A medida que las áreas se van haciendo más pequeñas, son menos los polígonos que se sobreponen en cada área, y a final de cuentas es posible tomar una decisión. Esta estrategia aprovecha la coherencia de áreas, ya que las áreas suficientemente pequeñas de una imagen estarán contenidas a lo sumo en un solo polígono visible.

**Algoritmo de Warnock.** El algoritmo de subdivisión de área desarrollado por Warnock [WARN69] subdivide cada área en cuatro cuadrados iguales. En cada etapa del proceso de subdivisión recursiva, la proyección de cada polígono tiene una de cuatro relaciones con el área de interés (véase la Fig. 13.23):

1. **Polígonos circundantes**, que contienen toda el área de interés (sombreada) (Fig. 13.23a).
2. **Polígonos intersecantes**, que intersecan el área (Fig. 13.23b).
3. **Polígonos contenidos**, que están totalmente dentro del área (Fig. 13.23c).
4. **Polígonos disjuntos**, que están completamente fuera del área (Fig. 13.23d).

Es obvio que los polígonos disjuntos no tienen influencia sobre el área de interés. También es irrelevante la parte de un polígono intersecante que está fuera del área, mientras que la parte de un polígono intersecante dentro del área equivale a un polígono contenido y por ende se puede tratar como tal.

La decisión acerca de un área se puede tomar fácilmente en cuatro casos, de manera que ya no se tenga que subdividir más:

1. Todos los polígonos son disjuntos con respecto al área. El color de fondo se puede dibujar en el área.
2. Sólo hay un polígono intersecante o contenido. Primero se rellena el área con el color de fondo y después se discretiza la parte del polígono contenido en el área.
3. Hay un solo polígono circundante, pero no polígonos intersecantes ni contenidos. El área se rellena con el color del polígono circundante.
4. Hay más de un polígono intersecante, contenido o circundante, pero uno de ellos es un polígono circundante que se encuentra enfrente de los demás polígonos. La determinación de si un polígono circundante está enfrente lleva a cabo calculando las coordenadas  $z$  de los planos de todos los polígonos circundantes, intersecantes y contenidos en los cuatro vértices del área; si hay un polígono circundante cuyas cuatro coordenadas  $z$  de vértices son mayores (más cercanas al punto de observación) que las de cualquier otro polígono, se puede llenar toda el área con el color del polígono circundante.

Los casos 1, 2 y 3 son fáciles de comprender. El caso 4 se ilustra con mayor detalle en la figura 13.24. En la parte (a), las cuatro intersecciones del polígono circundante están más cerca del punto de observación (el cual está en infinito en el eje  $+z$ ) que cualquiera de las otras intersecciones. Por consiguiente, toda



**Figura 13.24** Dos ejemplos del caso 4 en la subdivisión recursiva. (a) El polígono circundante es el más cercano en todas las esquinas del área de interés. (b) El polígono intersecante es el más cercano en el lado izquierdo del área de interés. El símbolo  $\times$  marca la intersección del plano del polígono circundante;  $\circ$  marca la intersección del plano del polígono intersecante;  $*$  marca la intersección del plano del polígono contenido.

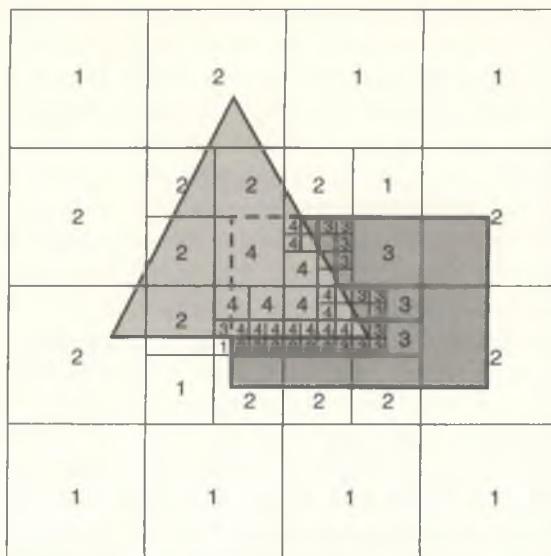
área se rellena con el color del polígono circundante. En la parte (b) no se puede tomar ninguna decisión, aunque el polígono circundante parece estar enfrente del polígono intersecante, ya que en la parte izquierda el plano del polígono intersecante se halla enfrente del plano del polígono circundante. Observe que el algoritmo de ordenamiento por profundidad acepta este caso sin subdivisiones adicionales si el polígono intersecante está totalmente en el lado del polígono circundante que se encuentra más lejos del punto de observación. Sin embargo, el algoritmo de Warnock siempre subdivide el área para simplificar el problema. Después de la subdivisión, sólo hay que reexaminar los polígonos contenidos e intersecantes: los polígonos circundantes y disjuntos del área original son polígonos circundantes y disjuntos de cada área subdividida.

Hasta ahora, el algoritmo ha operado con precisión de objetos, con la excepción de la discretización del fondo y los polígonos recortados en los cuatro casos. Sin embargo, estas operaciones de discretización de precisión de imagen se pueden reemplazar con operaciones de precisión de objeto que producen una representación precisa de las superficies visibles: un cuadrado del tamaño del área (casos 1, 3 y 4) o un polígono recortado con respecto al área, así como su complemento booleano con respecto al área, que represente la parte visible del fondo (caso 2). ¿Qué ocurre con los casos que no corresponden a estos cuatro? Un método consiste en detener la subdivisión cuando se alcanza la resolución del dispositivo de presentación. De esta manera, en una pantalla de barrido de  $1024 \times 1024$  se requieren 10 niveles de subdivisión como máximo. Si después de este número máximo de subdivisiones no se ha presentado ninguno de los casos 1 a 4, se calcula la profundidad de todos los polígonos relevantes en el centro de esta área indivisible de tamaño de un pixel. El polígono con la coordenada  $z$  más cercana define el sombreado del área. Alternativamente, para la eliminación de artefactos de discretización se pueden emplear varios niveles adicionales de subdivisión para determinar el color de un pixel, ponderando el color de cada una de las áreas con tamaño de subpixel de acuerdo con su tamaño. Estas operaciones de precisión de imagen, efectuadas cuando un área no es uno de los casos sencillos, son las que hacen que este método sea de precisión de imagen.

En la figura 13.25 se observa una escena sencilla y las subdivisiones necesarias para la presentación de dicha pantalla. El número en cada área subdividida corresponde a uno de los cuatro casos; en las áreas no numeradas no es válido ninguno de los cuatro casos. Compare este método con la partición espacial bidimensional efectuada por los árboles de cuadrantes (Sec. 10.6.3).

### 13.5.3 Algoritmos para superficies curvas

Todos los algoritmos presentados hasta ahora, con la excepción de el de *z-buffer*, se han descrito únicamente para objetos definidos por caras poligonales. Los objetos como las superficies curvas del capítulo 9 se deben aproximar primero con varias facetas pequeñas antes de poder emplear las versiones poligonales de cualquiera de estos algoritmos. Aunque se puede llevar a cabo esta aproximación, muchas veces es preferible discretizar directamente las superficies curvas,



**Figura 13.25** Subdivisión de un área en cuadrados.

eliminando artefactos poligonales y evitando el almacenamiento adicional requiere la aproximación poligonal.

Las superficies cuádricas, analizadas en la sección 9.4, constituyen una opción común en la graficación por computador. Weiss [WEIS66], Woon [WOON71], Mahl [MAHL72], Levin [LEVI76] y Sarraga [SARR83] han desarrollado varios algoritmos para cuádricas. Todos estos algoritmos encuentran las intersecciones de dos cuádricas, generando una ecuación de cuarto orden en  $x$ ,  $y$  y  $z$  cuyas raíces deben determinarse en forma numérica. Levin reduce esto a un problema de segundo orden parametrizando las curvas de intersección. Las esferas, un caso especial de las cuádricas, son más fáciles de manipular y tienen un interés especial porque las moléculas con frecuencia se presentan como colecciones de esferas coloreadas (véase la Ilust. en color 22). Se han desarrollado varios algoritmos de presentación molecular [KNOW77; STAU78; MAXPORT79; FRAN81; MAX84]. En la sección 13.4 se analiza cómo generar estas usando rastreo de rayos.

Se puede obtener incluso más flexibilidad con las superficies de spline paramétricas presentadas en el capítulo 9, ya que son más generales y permiten la continuidad de tangentes en fronteras de parches. Catmull [CATM74; CATM75] desarrolló el primer algoritmo de presentación para bicúbicas. En el espíritu del algoritmo de Warnock, el parche se subdivide recursivamente en cuartos para obtener cuatro parches, hasta que su proyección no cubra más que un pixel. Un algoritmo de *z-buffer* determina si el parche es visible en este pixel. Si lo es, se calcula un matiz para él y se coloca en la memoria gráfica. El seudocódigo

digo para este algoritmo se presenta en el programa 13.4. Como la revisión del tamaño del parche curvo requiere mucho tiempo, se puede utilizar en cambio un cuadrilátero definido por los vértices del parche.

**Programa 13.4**  
Seudocódigo para el  
algoritmo de subdivisión  
recursiva de Catmull.

```

for (cada parche) {
    meter el parche en la pila;
    while (pila con datos) {
        sacar parche de la pila;
        if (parche cubre ≤ 1 pixel) {
            if (pixel del parche más cercano en z)
                determinar matiz y dibujar;
        }
        else {
            subdividir parche en 4 subparches;
            meter subparches en la pila;
        }
    }
}

```

Otro método se basa en la subdivisión adaptable de cada parche bicúbico hasta que el parche subdividido esté dentro de cierta tolerancia de ser plano. Esta tolerancia depende de la resolución del dispositivo de presentación y de la orientación del área que se subdivide con respecto al plano de proyección, de manera que se eliminen las subdivisiones innecesarias. El parche sólo tiene que subdividirse en una dirección si ya es lo suficientemente plano en la otra. Una vez que se ha subdividido lo suficiente, un parche se puede tratar como cuadrilátero. Las pequeñas áreas poligonales definidas por los cuatro vértices de cada parche son procesadas por un algoritmo de línea de barrido, permitiendo la combinación de superficies poligonales y bicúbicas. Lane y Carpenter [LANE80] y Clark [CLAR79] han desarrollado algoritmos que utilizan esta idea básica; se analizan en [FOLE90].

## RESUMEN

Sutherland, Sproull y Schumacker [SUTH74a] hacen hincapié en que la base de la determinación de superficies visibles es el ordenamiento. Hemos visto muchos ejemplos de ordenamiento y búsquedas en los algoritmos, y el ordenamiento eficiente es vital para una determinación eficiente de superficies visibles. Es igualmente importante evitar más ordenamiento del que en realidad se requiere, objetivo que se cumple aprovechando la coherencia. Por ejemplo, los algoritmos de línea de barrido utilizan coherencia de líneas de barrido para eliminar la necesidad de efectuar un ordenamiento completo con respecto a *x* para cada línea de barrido.

Los algoritmos se pueden clasificar de acuerdo con su forma de ordenamiento. El algoritmo de ordenamiento por profundidad ordena con base en *z* y luego con base en *x* y *y* (usando las extensiones en las pruebas 1 y 2); por lo tanto, se le conoce como algoritmo *zxy*. Los algoritmos de línea de barrido or-

denan con respecto a  $y$  (con ordenamiento de cubeta), después con respecto a  $z$  (inicialmente con un ordenamiento por inserción y después con uno de burbuja) al procesar cada línea de rastreo) y por último realizan una búsqueda en  $z$  del polígono más cercano al punto de observación; por lo tanto, también hay algoritmos  $yzx$ . El algoritmo de Warnock lleva a cabo un ordenamiento paralelo con respecto a  $x$  y  $y$ , y después busca en  $z$ , por lo que es un algoritmo  $(xy)z$  (el ordenamiento usando una combinación de dimensiones se indica con paréntesis). El algoritmo de memoria de profundidad  $z$  no lleva a cabo ningún ordenamiento explícito y sólo busca en  $z$ ; se le conoce como algoritmo  $(xyz)$ .

Sancha ha planteado que la base del ordenamiento no tiene importancia: hay ningún beneficio intrínseco en ordenar primero con respecto a un eje en particular que con respecto a otro, ya que, por lo menos en principio, el objeto medio tiene la misma complejidad en las tres dimensiones [SUTH74a]. Por otra parte, una escena gráfica, como un escenario de Hollywood, se puede construir de manera que se vea mejor desde un punto en particular, y esto puede implicar mayor complejidad a lo largo de un eje que a lo largo de otro. Incluso si suponemos una complejidad de objetos aproximadamente simétrica, "no todos los algoritmos tienen la misma eficiencia: difieren en cuanto a la eficacia en el empleo de la coherencia para evitar el ordenamiento y otros cálculos, así como en la forma de usar los compromisos tiempo-espacio. Los resultados presentados en [SUTH74a, tabla VII], que comparan el desempeño estimado de cuatro de los algoritmos básicos que hemos presentado, se resumen en la tabla 13.1. Los autores proponen que, como se trata sólo de cálculos, deben ignorarse las diferencias pequeñas, pero que "nos sentimos en la libertad de hacer comparaciones de orden de magnitud entre los algoritmos para aprender algo acerca de la eficiencia de los diversos métodos" [SUTH74a, pág. 52].

El algoritmo de ordenamiento por profundidad es eficiente para un número pequeño de polígonos porque las sencillas pruebas de superposición casi siempre bastan para determinar si se puede discretizar un polígono. Si hay más polígonos, se requieren pruebas más complejas con mayor frecuencia y es probable que se requiera la subdivisión de polígonos. El algoritmo de  $z$ -buffer tiene

Tabla 13.1

Desempeño relativo calculado de cuatro algoritmos para la determinación de superficies visibles

Algoritmo	100	2500	6000
Ordenamiento por profundidad	1*	10	20
$z$ -buffer	54	54	54
Línea de barrido	5	21	21
Subdivisión de área de Warnock	11	64	64

\* Las entradas están normalizadas, de manera que este caso es la unidad.

desempeño constante ya que, al aumentar el número de polígonos en una escena, se reduce el número de pixeles cubiertos por un solo polígono. Por otra parte, sus requisitos de memoria son muy elevados. Las pruebas individuales y los cálculos requeridos en el algoritmo de subdivisión de área de Warnock son relativamente complejos, y por ende dicho algoritmo generalmente es más lento que los otros métodos. Además de estos cálculos informales, se han realizado trabajos en lo referente a la formalización del problema de superficies visibles y el análisis de su complejidad computacional [GILO78; FOUR88; FIUM89]. Por ejemplo, Fiume [FIUM89] demuestra que los algoritmos de precisión de objeto para superficies visibles tienen una cota inferior que es peor que la del ordenamiento.

En términos generales, es difícil comparar los algoritmos de superficies visibles porque no todos calculan la misma información con igual precisión. Por ejemplo, hemos analizado algoritmos que restringen los tipos de objetos, las relaciones entre objetos e incluso los tipos de proyecciones que se permiten. Como veremos en el capítulo siguiente, en la elección de un algoritmo de superficies visibles también influye el tipo de sombreado que se desea. Si se utiliza un procedimiento de sombreado muy costoso, es mejor elegir un algoritmo de superficies visibles que sombre únicamente partes visibles de los objetos, como un algoritmo de línea de barrido. En este caso sería bastante mala la elección del algoritmo de ordenamiento por profundidad, ya que dibuja por completo todos los objetos. Cuando es importante el rendimiento interactivo, por lo general se emplean los métodos de *z-buffer*. Por otra parte, el algoritmo de árbol BSP puede generar rápidamente nuevas vistas de un ambiente estático, pero requiere procesamiento adicional cuando cambia el ambiente. Los algoritmos de línea de barrido permiten resoluciones muy altas porque las estructuras de datos tienen que representar versiones completamente elaboradas sólo de las primitivas que afectan a la línea que se procesa. Como sucede con cualquier algoritmo, el tiempo que se invierte en la implantación del algoritmo y la facilidad con la cual se puede modificar (p. ej., para acomodar nuevas primitivas) también son factores importantes.

Una cuestión fundamental al implantar un algoritmo de superficies visibles es el tipo de apoyo de hardware que está disponible. Si se dispone de una máquina paralela, debemos reconocer que, en cada lugar donde un algoritmo aproveche la coherencia, éste depende de los resultados de los cálculos previos. El aprovechamiento del paralelismo quizás implique ignorar otras formas de coherencia que podrían ser útiles. La traza de rayos ha sido un candidato muy popular para la implantación paralela ya que, en su forma más simple, cada pixel se calcula de manera independiente.

## Ejercicios

13.1 Demuestre que la transformación  $M$  en la sección 13.1.2 conserva (a) las líneas rectas, (b) los planos y (c) las relaciones de profundidad.

13.2 Dado un plano  $Ax + By + Cz + D = 0$ , aplique  $M$  de la sección 13.1.2 para hallar los nuevos coeficientes de la ecuación del plano.

13.3 ¿Cómo se puede extender un algoritmo de línea de barrido para tratar polígonos con aristas compartidas? ¿Debe la arista compartida representarse una vez, como compartida, o dos veces, una para cada polígono que acota, sin llevar un registro de si se trata de una arista compartida? Cuando se evalúa la profundidad de dos polígonos en su arista común, las profundidades serán, por supuesto, iguales. ¿Cuál polígono se debe declarar como visible si el rastreo intergresa a ambos?

13.4 Explique cómo se manejarían los polígonos penetrantes en los algoritmos de *z-buffer*, de ordenamiento por profundidad, de Warnock y de árbol BSP. ¿Constituyen un caso especial que debe tratarse en forma explícita o puede manejarlos el algoritmo básico?

13.5 ¿Cómo se pueden adaptar los algoritmos mencionados en el ejercicio 13.4 para trabajar con polígonos que contienen agujeros?

13.6 Una de las ventajas del algoritmo de *z-buffer* es que las primitivas se pueden presentar en cualquier orden. ¿Significa esto que dos imágenes creadas al enviar primitivas en orden diferente tendrán valores idénticos en sus memorias de profundidad y en sus memorias gráficas? Explique su respuesta.

13.7 Considere la combinación de dos imágenes de tamaño idéntico, representadas por el contenido de sus memorias de profundidad y gráfica. Si conocemos la  $z_{\min}$  y la  $z_{\max}$  de cada imagen y los valores de  $z$  a los que correspondían originalmente, ¿puede combinar las imágenes adecuadamente? ¿Se requiere información adicional?

13.8 En la sección 13.2 se mencionan los problemas de compresión de profundidad ocasionados al generar una proyección de perspectiva usando una memoria de profundidad entera. Elija una especificación de visualización de perspectiva y un pequeño número de puntos de objetos. Muestre cómo, en la transformación de perspectiva, dos puntos cercanos al centro de proyección responden a valores distintos de  $z$ , mientras que dos puntos alejados la misma distancia entre sí pero más lejos del centro de proyección, corresponden a un solo valor de  $z$ .

13.9 a. Suponga que el volumen de vista  $V$  tiene planos de recorte anterior y posterior a las distancias  $F$  y  $B$  (ambas positivas) respectivamente, en VRP, medidas en la dirección DOP. Suponga que la distancia del COP al VRP, medida por la DOP, es  $w$  y que el plano de recorte anterior se halla entre el VRP y el COP, y que además el VRP está entre el COP y el plano de recorte posterior (como se ilustra en la figura 6.16). Defina  $f = w - F$  y  $b = w + B$ , de manera que  $f$  sea la distancia del COP al plano anterior y  $b$  la distancia del COP al plano posterior. Haga esto de nuevo con un volumen de vista  $V'$  y defina  $f'$  y  $b'$  en forma similar. Despues de la transformación al volumen de vista canónico de perspectiva,

tiva, el plano de recorte posterior de  $V$  pasa a  $z = -1$  y el plano anterior a  $z = A$ . Así mismo, para el volumen  $V'$ , el plano anterior pasará a  $z = A'$ . Muestre que si  $f/b = f'/b'$ , entonces  $A = A'$  y viceversa. En resumen, el intervalo de valores de  $z$  después de la transformación al volumen de vista canónico sólo depende de la razón entre las distancias del COP a los planos de recorte anterior y posterior.

- b. En la parte (a) se muestra que, al considerar el efecto de la perspectiva, sólo hay que tener presente la razón entre las distancias a los planos anterior y posterior (desde el COP). Por lo tanto, basta estudiar el volumen de vista canónico con diversos valores de la distancia al plano frontal. Suponga entonces que tenemos un volumen de vista canónico de perspectiva, con plano de recorte anterior  $z = A$  y plano de recorte posterior  $z = -1$  y que los transformamos, por medio de la transformación de perspectiva, al volumen de vista paralela entre  $z = 0$  y  $z = -1$ . Escriba la fórmula para la coordenada  $z$  transformada en función de la coordenada  $z$  original (su respuesta dependerá de  $A$ , por supuesto). Suponga que los valores  $z$  transformados en el volumen de vista paralela se multiplican por  $2^n$  y se redondean a enteros (es decir, se establece una correspondencia con una memoria de profundidad  $z$  entera). Encuentre dos valores de  $z$  que estén lo más distantes posible, pero que, con la transformación, correspondan al mismo entero (su respuesta dependerá de  $n$  y de  $A$ ).
- c. Suponga que quiere crear una imagen en la cual la razón entre  $f$  y  $b$  sea  $R$  y donde los objetos que estén separados una distancia superior a  $Q$  (en  $z$ ) deban corresponder a valores diferentes en la memoria de profundidad. Use su trabajo de la parte (b) y escriba una fórmula para el número de bits que requieren la memoria de profundidad  $z$ .

13.10 Al realizar la traza de rayos, por lo general sólo hay que calcular si un rayo interseca una extensión, no cuáles son los puntos precisos de la intersección. Complete la ecuación de intersección rayo-esfera (Ec. 13.16) usando una fórmula cuadrática, y muestre cómo se puede simplificar para determinar únicamente si el rayo y la esfera se intersecan.

13.11 La traza de rayos también puede ser útil para determinar las propiedades de masa de los objetos a través de la integración numérica. El conjunto completo de intersecciones de un rayo con un objeto indica la porción total del rayo que se encuentra dentro del objeto. Muestre cómo se puede calcular el volumen de un objeto lanzando un arreglo regular de rayos paralelos a través del objeto.

13.12 Obtenga la intersección de un rayo y una superficie cuádrica. Modifique el método utilizado para obtener la intersección de un rayo y una esfera en las

ecuaciones (13.12) a (13.15), para manejar la definición de una cuádrica que presenta en la sección 9.4.

13.13 Implante uno de los algoritmos de superficies visibles de polígonos presentados en este capítulo, como el algoritmo de *z-buffer* o el algoritmo de *línea* de barrido.

13.14 Implante la traza de rayos simple para esferas y polígonos (elija uno de los modelos de iluminación de la sección 14.1). Mejore el rendimiento del programa usando la partición espacial o jerarquías de volúmenes acotantes.

En este capítulo analizamos cómo sombrear superficies con base en la posición, la orientación y las características de las superficies y las fuentes luminosas que las iluminan. Desarrollaremos varios **modelos de iluminación** que expresen los factores que determinan el color de una superficie en un punto específico. Los modelos de iluminación también se conocen como **modelos de alumbrado** o **modelos de sombreado**. Sin embargo, aquí reservaremos el término **modelo de sombreado** para el esquema más general dentro del cual se encuentra el modelo de iluminación. El modelo de sombreado determina cuándo se aplica el modelo de iluminación y qué argumentos recibe. Por ejemplo, algunos modelos de sombreado invocan un modelo de iluminación para cada pixel de la imagen, mientras que otros invocan un modelo de iluminación únicamente para ciertos pixeles y sombrean los demás por interpolación.

Cuando comparamos la precisión con la cual se realizan los cálculos de superficies visibles en el capítulo anterior, establecimos la diferencia entre los algoritmos que usan la geometría real del objeto y los que usan aproximaciones poliédricas, así como entre los algoritmos de precisión de la imagen y los de precisión del objeto. Sin embargo, en todos los casos el criterio para determinar la visibilidad directa de un objeto en un pixel es si algo está entre el objeto y el observador a lo largo del proyector que pasa por el pixel. En cambio, la interacción entre luces y las superficies es mucho más compleja. Los investigadores gráficos muchas veces han deducido las reglas que sirven de base a la óptica y la radiación térmica, ya sea para simplificar los cálculos o porque no se conocían modelos más precisos en la comunidad gráfica. Por lo tanto, muchos de los modelos de iluminación y sombreado que se han usado tradicionalmente en la

graficación por computador incluyen muchos “trucos” y simplificaciones que no tienen bases teóricas firmes, pero que en la práctica funcionan bien. En la primera parte de este capítulo se abarcan estos modelos sencillos, que siguen siendo de uso común porque pueden producir resultados útiles y atractivos con la menor cantidad posible de cálculos.

Comenzamos, en la sección 14.1, con un análisis de los modelos de iluminación simples que consideran un punto individual en una superficie y las fuentes luminosas que lo iluminan directamente. Primero desarrollaremos modelos de iluminación para superficies y luces monocromáticas, y luego mostraremos cómo se pueden generalizar los cálculos para manejar los sistemas de color analizados en el capítulo 11. En la sección 14.2 se describen los modelos de sombreado más comunes que se emplean con estos modelos de iluminación. En la sección 14.3 se expanden estos modelos para simular superficies texturizadas.

Para el modelado de refracción, reflexión y sombras se requieren cálculos adicionales similares a los cálculos para la eliminación de superficies ocultas, con los cuales muchas veces se integran. De hecho, estos efectos ocurren porque algunas de las *superficies ocultas* en realidad no están ocultas: se ven a través de otros objetos, son reflejadas o proyectan sombras sobre la superficie que se sombra. En las secciones 14.4 y 14.5 se analiza la forma de modelar algunos de estos efectos.

En las secciones 14.6 a 14.8 se describen los **modelos de iluminación global** que pretenden considerar el intercambio de luz entre todas las superficies: traza de rayos recursiva y métodos de radiosidad. La traza de rayos recursiva extiende el algoritmo de traza de rayos para superficies visibles presentado en el capítulo anterior para entrelazar la determinación de la visibilidad, la iluminación y el sombreado en cada pixel. Los métodos de radiosidad modelan el equilibrio de energía en un sistema de superficies; determinan la iluminación en un conjunto de puntos de muestra del ambiente en una forma independiente de la vista, antes de determinar las superficies visibles a partir del punto de observación descrito. Un tratamiento más detallado de varios de los modelos de iluminación y sombreado que presentamos se puede hallar en [GLAS89; HALL89].

Finalmente, en la sección 14.9 se ven varios ductos gráficos que integran las técnicas de generación de barrido analizadas en este capítulo y en los anteriores. Se examinan formas de implantar estas capacidades y producir sistemas que sean eficientes y extensibles.

## 14.1 Modelos de iluminación

### 14.1.1 Luz ambiental

Quizás el modelo de iluminación más sencillo sea el que se usa implícitamente en los primeros capítulos del libro: cada objeto se presenta con una intensidad

intrínseca. Podemos considerar este modelo, que no tiene una fuente de luz externa, como la descripción de un mundo ligeramente irreal de objetos no reflejantes y autoluminosos. Cada objeto aparece como una silueta monocromática, a menos que sus partes individuales, como los polígonos de un poliedro, reciban matices diferentes al crear el objeto. Este efecto se muestra en la ilustración en color 27.

Un modelo de iluminación se puede expresar con una **ecuación de iluminación** de variables asociadas con el punto en el objeto que se sombra. La ecuación de iluminación que expresa este sencillo modelo es

$$I = k_i, \quad (14.1)$$

donde  $I$  es la intensidad resultante y el coeficiente  $k_i$  es la intensidad intrínseca del objeto. Como esta ecuación de iluminación no contiene términos que dependan de la posición del punto que se sombra, se puede evaluar una vez para cada objeto. El proceso de evaluación de la ecuación de iluminación en uno o más puntos de un objeto se conoce como **iluminación** del objeto.

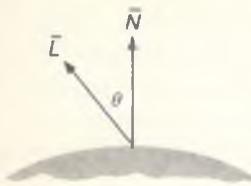
Imagine ahora que en lugar de autoluminosisdad hay una fuente luminosa difusa, no direccional, producto de reflexiones múltiples de la luz en las superficies presentes en el ambiente. Esto se conoce como **luz ambiental**. Si suponemos que la luz ambiental afecta de la misma forma a todas las superficies desde todas las direcciones, nuestra ecuación de iluminación se convierte en

$$I = I_a k_a. \quad (14.2)$$

$I_a$  es la intensidad de la luz ambiental, que se supone constante para todos los objetos. La cantidad de luz ambiental reflejada por la superficie de un objeto está determinada por  $k_a$ , el **coeficiente de reflexión ambiental**, que varía entre 0 y 1. El coeficiente de reflexión ambiental es una **propiedad material**. Al igual que las demás propiedades materiales que analizaremos, se puede considerar como la caracterización del material que conforma la superficie. Como sucede con otras propiedades, el coeficiente de reflexión ambiental es una conveniencia empírica y no corresponde directamente a ninguna propiedad física de los materiales reales. Además, la luz ambiental no es de mucho interés. Como veremos más adelante, se usa para considerar todas las formas complejas en que la luz puede llegar a un objeto y que no son tratadas de otra forma por la ecuación de iluminación. En la ilustración en color 27 también se muestra la iluminación con luz ambiental.

### 14.1.2 Reflexión difusa

Aunque los objetos iluminados por luz ambiental se iluminan en forma más o menos brillante en proporción directa a la intensidad ambiental, se siguen iluminando de manera uniforme en todas sus superficies. Considere ahora la iluminación de un objeto a partir de una **fuente luminosa puntual**, cuyos rayos

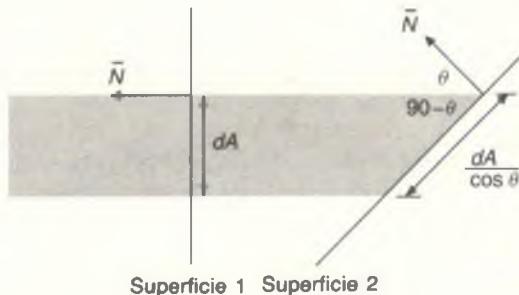


**Figura 14.1**  
Reflexión difusa.

emanan uniformemente en todas direcciones a partir de un solo punto. La brillantez del objeto varía de una parte a otra, dependiendo de la dirección y la distancia con respecto a la fuente luminosa.

**Reflexión lambertiana.** Las superficies opacas y mates, como la tiza, exhiben reflexión difusa, también conocida como **reflexión lambertiana**. Estas superficies aparecen con la misma brillantez desde todos los ángulos de observación porque reflejan la luz con igual intensidad en todas direcciones. Para una superficie, la brillantez depende únicamente del ángulo  $\theta$  entre la dirección  $L$  a la fuente luminosa y la normal a la superficie  $N$  de la figura 14.1. Examinemos por qué ocurre esto. Entran en juego dos factores. En primer lugar, en la figura 14.2 se ilustra que un haz que intercepta una superficie cubre un área cuyo tamaño es inversamente proporcional al coseno del ángulo  $\theta$  entre el haz y  $N$ . Si el haz tiene un área diferencial  $dA$  de sección transversal infinitesimalmente pequeña, entonces el haz intercepta un área  $dA/\cos \theta$  en la superficie. Entonces para un haz luminoso incidente, la cantidad de energía luminosa que cae en  $dA$  es proporcional a  $\cos \theta$ . Esto se aplica a cualquier superficie, independientemente de su material.

En segundo lugar debemos considerar la cantidad de luz que ve el observador. Las superficies lambertianas tienen la propiedad, muchas veces conocida como ley de Lambert, de que la cantidad de luz que reflejan de un área diferencial unidad  $dA$  hacia el observador es directamente proporcional al coseno del ángulo entre la dirección al observador y  $N$ . Como la cantidad de área de la superficie que se observa es inversamente proporcional al coseno del ángulo, estos dos factores se cancelan. Por ejemplo, al aumentar el ángulo de vista, el observador ve más área de la superficie, pero la cantidad de luz reflejada en un ángulo por área unidad de la superficie es proporcionalmente menor. Por tanto, para las superficies lambertianas, la cantidad de luz que ve el observador es independiente de la dirección de éste y sólo es proporcional a  $\cos \theta$ , el ángulo de incidencia de la luz.



**Figura 14.2** Un haz (mostrado en sección transversal bidimensional) de área de sección transversal infinitesimal  $dA$ , en un ángulo de incidencia  $\theta$ , intercepta un área de  $dA/\cos \theta$ .

La ecuación de iluminación difusa es

$$I = I_p k_d \cos \theta. \quad (14.3)$$

$I_p$  es la intensidad de la fuente luminosa puntual; el **coeficiente de reflexión difusa**  $k_d$  del material es una constante entre 0 y 1 que varía de un material a otro. El ángulo  $\theta$  debe estar entre  $0^\circ$  y  $90^\circ$  para que la luz tenga efecto directo en el punto que se sombra. Esto quiere decir que estamos tratando la superficie como **autooclayente**, de manera que la luz lanzada desde atrás de un punto en la superficie no lo ilumina. En lugar de incluir aquí y en las ecuaciones siguientes un término  $\max(\cos \theta, 0)$  en forma explícita, supondremos que  $\theta$  se halla dentro del intervalo permitido. Si se desea iluminar superficies autooclayentes, se puede usar  $\text{abs}(\cos \theta)$  para invertir las normales a sus superficies. Esto hace que ambos lados de la superficie se traten de la misma manera, como si la superficie estuviera iluminada por dos luces opuestas.

Suponiendo que los vectores  $\bar{N}$  y  $\bar{L}$  se normalizaron (véase la Sec. 5.1), podemos reescribir la ecuación (14.3) usando el producto punto:

$$I = I_p k_d (\bar{N} \cdot \bar{L}). \quad (14.4)$$

La normal a la superficie  $\bar{N}$  se puede calcular usando los métodos analizados en el capítulo 9. Si se calculan previamente las normales a los polígonos y se transforman con la misma matriz utilizada para los vértices de los polígonos, es importante no efectuar transformaciones de modelado que no sean rígidas, como sesgos o escalamiento diferencial; estas transformaciones no conservan los ángulos y ocasionan que algunas normales ya no sean perpendiculares a sus polígonos. En la sección 5.7 se describe el método adecuado para transformar las normales cuando los objetos se someten a transformaciones arbitrarias. En cualquier caso, es necesario evaluar la ecuación de iluminación en el sistema de coordenadas de mundo (o en cualquier sistema de coordenadas isométrico), ya que las transformaciones de normalización y de perspectiva modificarán el valor de  $\theta$ .

Si una fuente luminosa puntual se encuentra a distancia suficiente de los objetos que se sombrean, formará en esencia el mismo ángulo con todas las superficies que comparten la misma normal. En este caso, la luz se conoce como **fuente luminosa direccional** y  $\bar{L}$  es una constante para la fuente luminosa.

En la figura 14.3 se muestra una serie de fotografías de una esfera iluminada por una sola fuente puntual. El modelo de sombreado calculó la intensidad en cada pixel en el cual la esfera era visible usando el modelo de iluminación de la ecuación (14.4). Los objetos iluminados de esta manera se ven duros, como al iluminar con una linterna un objeto en una habitación a oscuras. Por lo tanto, es común añadir un término de ambiente para obtener una ecuación de iluminación más realista:

$$I = I_a k_a + I_p k_d (\bar{N} \cdot \bar{L}). \quad (14.5)$$

Se utilizó la ecuación (14.5) para producir la figura 14.4.



**Figura 14.3** Esferas sombreadas usando un modelo de reflexión difusa (Ec. 14.4). De izquierda a derecha,  $k_d = 0.4, 0.55, 0.7, 0.85, 1.0$ . (Por David Kurlander, Columbia University.)

**Atenuación de fuente luminosa.** Si las proyecciones de dos superficies paralelas de idéntico material, iluminadas desde el ojo, se sobreponen en una imagen, la ecuación (14.5) no distinguirá dónde termina una superficie y comienza la otra, sin importar cuán diferente sea su distancia a la fuente luminosa. Para ello, introduce un factor de atenuación de fuente luminosa,  $f_{att}$ , para obtener

$$I = I_a k_a + f_{att} I_p k_d (\bar{N} \cdot \bar{L}). \quad (14.5)$$

Una elección obvia de  $f_{att}$  tiene en cuenta el hecho de que la energía de una fuente luminosa puntual que llega a una parte de una superficie disminuye en una proporción inversa al cuadrado de  $d_L$ , la distancia que viaja la luz de la fuente puntual a la superficie. En este caso,

$$f_{att} = \frac{1}{d_L^2}. \quad (14.6)$$

Sin embargo, en la práctica esto no siempre funciona muy bien. Si la luz es muy lejos,  $1/d_L^2$  no varía mucho; si está muy cerca, varía considerablemente, ocasionando sombras muy variables en superficies con el mismo ángulo  $\theta$  entre  $\bar{N}$  y  $\bar{L}$ . Aunque este comportamiento es correcto para una fuente luminosa p

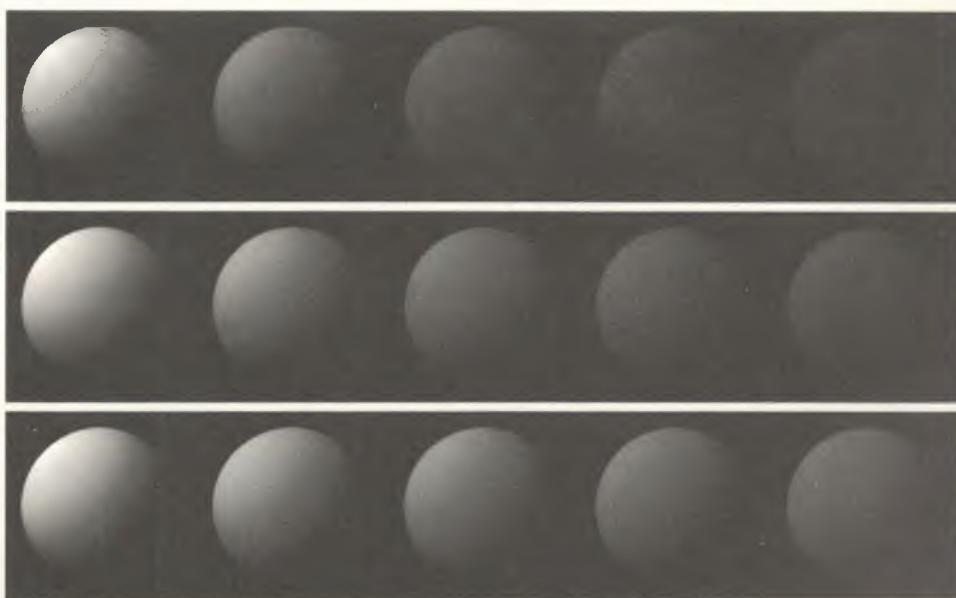


**Figura 14.4** Esferas sombreadas usando reflexión ambiental y difusa (Ec. 14.5). Para todas las esferas  $I_a = I_p = 1.0$ ,  $k_d = 0.4$ , de izquierda a derecha,  $k_a = 0.0, 0.15, 0.30, 0.45, 0.60$ . (Por David Kurlander, Columbia University.)

tual, los objetos que vemos en la vida real generalmente no están iluminados por fuente puntuales y no se sombrean con los sencillos modelos de iluminación de la graficación por computador. Para complicar el asunto, los primeros investigadores gráficos usaban con frecuencia una fuente luminosa puntual ubicada exactamente en el ojo. Esperaban que  $f_{\text{att}}$  dedujera algunos de los efectos de la atenuación atmosférica entre el observador y el objeto (véase la Sec. 14.1.3), así como la reducción en la densidad energética de la luz al objeto. Una forma útil, que permite mayor gama de efectos que la sencilla atenuación de ley cuadrada, es

$$f_{\text{att}} = \min \left( \frac{1}{c_1 + c_2 d_L + c_3 d_L^2}, 1 \right). \quad (14.8)$$

Aquí,  $c_1$ ,  $c_2$  y  $c_3$  son constantes definidas por el usuario relacionadas con la fuente luminosa. La constante  $c_1$  evita que el denominador sea demasiado pequeño cuando la luz esté cerca, y la expresión se limita a un máximo de 1 para asegurar que siempre se atenúa. En la figura 14.5 se emplea este modelo de iluminación con diferentes constantes, para mostrar una gama de efectos.



**Figura 14.5** Esferas sombreadas usando reflexión ambiental y difusa con un término de atenuación de fuente luminosa (Ecs. 14.6 y 14.8). Para todas las esferas,  $I_a = I_p = 1.0$ ,  $k_s = 0.1$ ,  $k_d = 0.9$ . De izquierda a derecha, la distancia de la esfera a la fuente luminosa es 1.0, 1.375, 1.75, 2.125, 2.5. Fila superior:  $c_1 = c_2 = 0.0$ ,  $c_3 = 1.0 (1/d_L^3)$ . Fila central:  $c_1 = c_2 = 0.25$ ,  $c_3 = 0.5$ . Fila inferior:  $c_1 = 0.0$ ,  $c_2 = 1.0$ ,  $c_3 = 0.0 (1/d_L)$ . (Por David Kurlander, Columbia University.)

**Luces y superficies coloreadas.** Hasta ahora hemos descrito luces y superficies monocromáticas. Las luces y superficies coloreadas se tratan en forma similar escribiendo ecuaciones distintas para cada componente del modelo de color. El color difuso de un objeto se representa con un valor de  $O_d$  para cada componente. Por ejemplo, el triple ( $O_{dR}$ ,  $O_{dG}$ ,  $O_{dB}$ ) define los componentes difusos rojo, verde y azul en el sistema de colores RGB. En este caso, los tres componentes primarios de la luz iluminadora,  $I_{pR}$ ,  $I_{pG}$  e  $I_{pB}$  se reflejan en proporción a  $k_d O_{dR}$ ,  $k_d O_{dG}$  y  $k_d O_{dB}$ , respectivamente. Por lo tanto, para el componente rojo,

$$I_R = I_{aR} k_a O_{dR} + f_{att} I_{pR} k_d O_{dR} (\bar{N} \cdot \bar{L}). \quad (14.9)$$

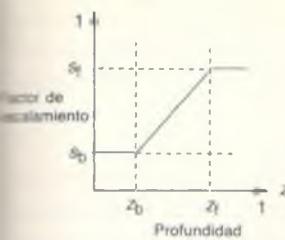
Se emplean ecuaciones similares para  $I_G$  e  $I_B$ , los componentes verde y azul. La utilización de un solo componente para escalar una expresión en cada una de las ecuaciones permite al usuario controlar la cantidad de reflexión ambiental o difusa sin alterar las proporciones de sus componentes. Una formulación alternativa, más compacta pero menos sencilla de controlar, utiliza un coeficiente distinto para cada componente, por ejemplo sustituyendo  $k_{aR}$  en lugar de  $k_a O_{dR}$  y  $k_{dR}$  en lugar de  $k_d O_{dR}$ .

Aquí hacemos una suposición de simplificación: que un modelo de colores de tres componentes puede modelar totalmente la interacción de la luz con los objetos. Esta suposición es errónea, pero resulta fácil de implantar y muchas veces produce imágenes aceptables. En teoría, la ecuación de iluminación debe evaluarse en forma continua en toda la gamapectral que se modela; en la práctica, se evalúa para varias muestras espectrales discretas. En lugar de limitarnos a un modelo de color en particular, indicamos explícitamente los términos en una ecuación de iluminación que dependen de la longitud de onda, colocando un subíndice  $\lambda$ . De esta manera, la ecuación (14.9) se convierte en

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{att} I_{p\lambda} k_d O_{d\lambda} (\bar{N} \cdot \bar{L}). \quad (14.10)$$

### 14.1.3 Atenuación atmosférica

Para simular la atenuación atmosférica entre el objeto y el observador, muchos sistemas proporcionan **indicaciones de profundidad**. En esta técnica, que tiene su origen en el hardware de gráficos vectoriales, los objetos más distantes se generan con menor intensidad que los cercanos. El estándar PHIGS PLUS recomienda un método de indicación de profundidad que también hace posible deducir el desplazamiento en colores ocasionado por la atmósfera. Se definen en NPC planos de referencia anterior y posterior para la indicación de profundidad; cada uno de estos planos está asociado a un factor de escalamiento,  $s_a$  y  $s_b$ , respectivamente, entre 0 y 1. Los factores de escalamiento determinan la combinación de la intensidad original con la de un color indicador de profundidad,  $I_{dc\lambda}$ . El objetivo es modificar una  $I_\lambda$  previamente calculada para producir un valor indicador de profundidad  $I'_\lambda$  que se presenta. Si se tiene  $z_0$ , la coordenada

**Figura 14.6**

Cálculo del factor de escalamiento para la atenuación atmosférica.

$z$  del objeto, se obtiene un factor de escalamiento  $s_o$  que se usará para interpolar entre  $I_\lambda$  e  $I_{dc\lambda}$  y determinar

$$I'_\lambda = s_o I_\lambda + (1 - s_o) I_{dc\lambda}. \quad (14.11)$$

Si  $z_o$  está enfrente de la coordenada  $z_i$  del plano anterior de referencia de indicación de profundidad, entonces  $s_o = s_f$ . Si  $z_o$  está detrás de la coordenada  $z_b$  del plano posterior de referencia de indicación de profundidad, entonces  $s_o = s_b$ . Finalmente, si  $z_o$  se halla entre los dos planos, entonces

$$s_o = s_b + \frac{(z_o - z_b)(s_f - s_b)}{z_f - z_b}. \quad (14.12)$$

La relación entre  $s_o$  y  $z_o$  se presenta en la figura 14.6. En la figura 14.7 se muestran varias esferas sombreadas con indicación de profundidad. Para no complicar más las ecuaciones, ignoramos la indicación de profundidad al desarrollar más el modelo de iluminación.

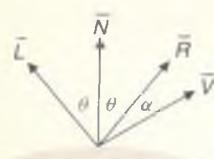
#### 14.1.4 Reflexión especular

La reflexión especular se puede observar en cualquier superficie brillante. Ilumine una manzana con una luz blanca brillante: el punto de máximo brillo (*highlight*) es ocasionado por la reflexión especular, mientras que la luz reflejada del resto de la manzana es el resultado de la reflexión difusa. Observe también que, en el punto de máximo brillo, la manzana no aparece roja, sino blanca, el color de la luz incidente. Los objetos como las manzanas enceradas o los plásticos brillantes tienen superficie transparente; los plásticos, por ejemplo, generalmente están compuestos por partículas de pigmento incorporadas en un material transparente. La luz que se refleja especularmente de la superficie incolora tiene un color muy similar al de la fuente luminosa.

Ahora mueva su cabeza y note cómo se mueve el punto de máximo brillo. Esto sucede porque las superficies brillantes reflejan la luz en forma desigual en distintas direcciones: en una superficie perfectamente brillante, como un espejo

**Figura 14.7**

Esferas sombreadas usando indicación de profundidad (Ecs. 14.5, 14.11 y 14.12). La distancia con respecto a la luz es constante. Para todas las esferas,  $I_a = I_p = 1.0$ ,  $k_a = 0.1$ ,  $k_d = 0.9$ ,  $Z_i = 1.0$ ,  $Z_b = 0.0$ ,  $s_f = 1.0$ ,  $s_b = 0.1$ , radio = 0.09. De izquierda a derecha, la  $z$  al frente de la esfera es 1.0, 0.77, 0.55, 0.32, 0.09. (Por David Kurlander, Columbia University.)



**Figura 14.8**  
Reflexión especular.

perfecto, la luz se refleja sólo en la dirección de reflexión  $\bar{R}$ , que es  $\bar{L}$  invertida con respecto a  $\bar{N}$ . De esta manera, el observador puede ver luz reflejada especialmente en un espejo sólo si el ángulo  $\alpha$  de la figura 14.8 es cero;  $\alpha$  es el ángulo entre  $\bar{R}$  y la dirección  $\bar{V}$  al punto de observación.

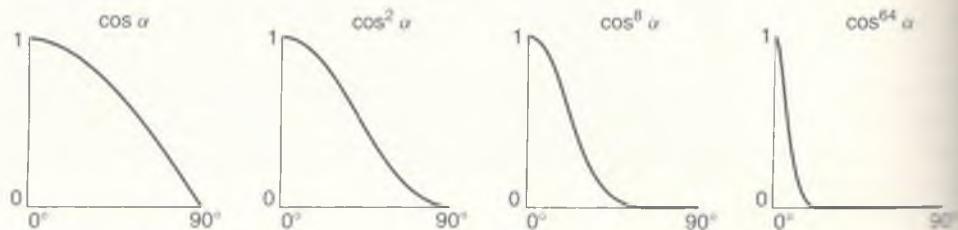
**El modelo de iluminación de Phong.** Phong Bui-Tuong [BUI75] desarrolló un popular modelo de iluminación para reflectores imperfectos, como la manzana. El modelo supone que la máxima reflectancia especular ocurre cuando  $\alpha$  es cero y decrece rápidamente conforme aumenta  $\alpha$ . Esta caída rápida es aproximada por  $\cos^n \alpha$ , donde  $n$  es el **exponente de reflexión especular** del material. Los valores típicos de  $n$  varían entre uno y varios cientos, dependiendo del material de superficie que se simule. Un valor de 1 proporciona una caída amplia, suave, mientras que los valores más elevados simulan un punto de máximo brillo definido y enfocado (Fig. 14.9). Para un reflector perfecto,  $n$  sería infinito. Como antes, tratamos un valor negativo de  $\cos \alpha$  como cero. El modelo de iluminación de Phong se basa en el trabajo anterior de investigadores como Warnock [WARN69], quienes usaron un término  $\cos^n \theta$  para modelar la reflexión especular con la luz en el punto de observación. Sin embargo, Phong fue el primero en tener en cuenta observadores y luces en posiciones arbitrarias.

La cantidad de luz incidente que se refleja especularmente depende del ángulo de incidencia  $\theta$ . Si  $W(\theta)$  es la fracción de luz reflejada especularmente, el modelo de Phong es

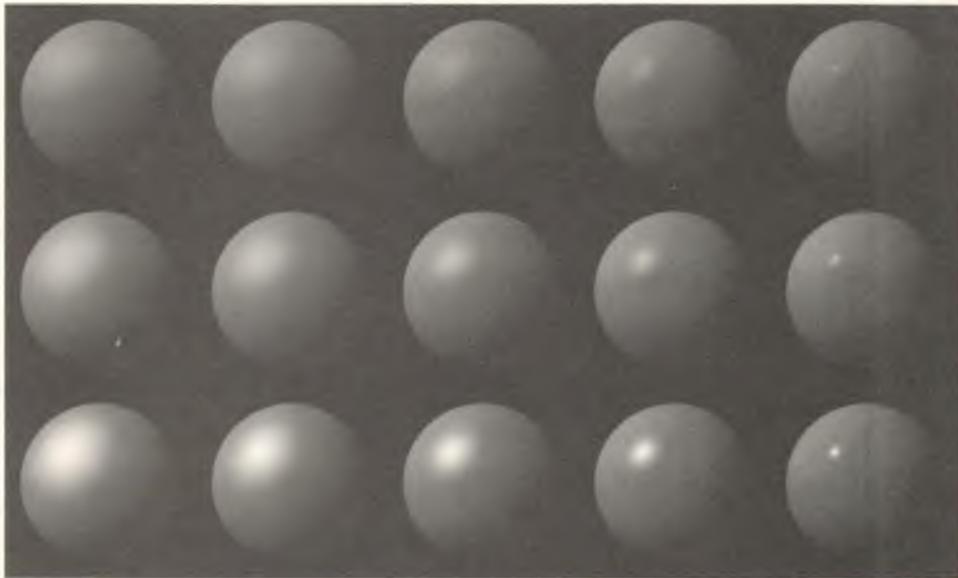
$$I_\lambda = I_{a\lambda} k_s O_{d\lambda} + f_{att} I_{p\lambda} [k_d O_{d\lambda} \cos \theta + W(\theta) \cos^n \alpha]. \quad (14.13)$$

Si se normalizan la dirección de reflexión  $\bar{R}$  y la dirección al punto de observación  $\bar{V}$ , entonces  $\cos \alpha = \bar{R} \cdot \bar{V}$ . Además,  $W(\theta)$  suele asignarse igual a una constante  $k_s$ , el **coeficiente de reflexión especular** del material, que varía entre 0 y 1. El valor de  $k_s$  se selecciona experimentalmente para producir resultados atractivos desde el punto de vista estético. Así, la ecuación (14.13) se puede reescribir como

$$I_\lambda = I_{a\lambda} k_s O_{d\lambda} + f_{att} I_{p\lambda} [k_d O_{d\lambda} (\bar{N} \cdot \bar{L}) + k_s (\bar{R} \cdot \bar{V})^n]. \quad (14.14)$$



**Figura 14.9** Valores diferentes de  $\cos^n \alpha$  usados en el modelo de iluminación de Phong.



**Figura 14.10** Esferas sombreadas usando el modelo de iluminación de Phong (Ec. 14.14) y distintos valores de  $k_s$  y  $n$ . Para todas las esferas,  $I_a = I_p = 1.0$ ,  $k_a = 0.1$ ,  $k_d = 0.45$ . De izquierda a derecha,  $n = 3.0, 5.0, 10.0, 27.0, 200.0$ . De arriba hacia abajo,  $k_s = 0.1, 0.25, 0.5$ . (Por David Kurlander, Columbia University.)

Observe que el color en el componente especular del modelo de iluminación de Phong *no* depende de ninguna propiedad material; por lo tanto, este modelo produce buenos resultados al modelar reflexiones especulares de superficies plásticas. Como veremos en la sección 14.1.7, la reflexión especular es afectada por las propiedades de la superficie y, por lo general, puede tener un color distinto al de la reflexión difusa cuando la superficie está compuesta por varios materiales. Podemos tener en cuenta este efecto con una primera aproximación modificando la ecuación (14.14) para obtener

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{att} I_{p\lambda} [k_d O_{d\lambda} (\bar{N} \cdot \bar{L}) + k_s O_{s\lambda} (\bar{R} \cdot \bar{V})^n], \quad (14.15)$$

donde  $O_{s\lambda}$  es el **color especular** del objeto. En la figura 14.10 se presenta una esfera iluminada usando distintos valores de  $k_s$  y  $n$  en la ecuación (14.14).

**Cálculo del vector de reflexión.** Para el cálculo de  $\bar{R}$  se requiere reflejar  $\bar{L}$  sobre  $\bar{N}$ . Como se muestra en la figura 14.11, esto se puede lograr con un poco de geometría. Como  $\bar{N}$  y  $\bar{L}$  están normalizados, la proyección de  $\bar{L}$  sobre  $\bar{N}$  es  $\bar{N} \cos\theta$ . Observe que  $\bar{R} = \bar{N} \cos\theta + \bar{S}$ , donde  $|\bar{S}|$  es  $\sin\theta$ . Sin embargo, por resta de vectores y triángulos congruentes,  $\bar{S}$  es simplemente  $\bar{N} \cos\theta - \bar{L}$ . Por lo tanto,

**Figura 14.11** Cálculo del vector de reflexión.

$\bar{R} = 2\bar{N}\cos\theta - \bar{L}$ . Al sustituir  $\bar{N} \cdot \bar{L}$  en lugar de  $\cos\theta$  y  $\bar{R} \cdot \bar{V}$  en lugar de  $\cos\alpha$  se obtiene

$$\bar{R} = 2\bar{N}(\bar{N} \cdot \bar{L}) - \bar{L}, \quad (14.16)$$

$$\bar{R} \cdot \bar{V} = (2\bar{N}(\bar{N} \cdot \bar{L}) - \bar{L}) \cdot \bar{V}. \quad (14.17)$$

Si la fuente luminosa está en el infinito,  $\bar{N} \cdot \bar{L}$  es constante para un polígono, mientras que  $\bar{R} \cdot \bar{V}$  varía a través del polígono. En el caso de superficies curvas o de una fuente luminosa que no está en el infinito, tanto  $\bar{N} \cdot \bar{L}$  como  $\bar{R} \cdot \bar{V}$  varían a través de la superficie.

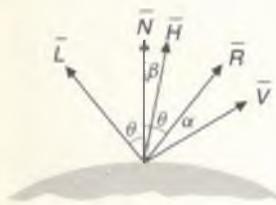


Figura 14.12

$H$ , el vector intermedio, está a la mitad entre la dirección de la fuente luminosa y el observador.

**El vector intermedio.** Una formulación alternativa del modelo de iluminación de Phong usa el **vector intermedio**  $\bar{H}$ , llamado así porque su dirección es intermedia entre las direcciones de la fuente luminosa y el observador, como se ilustra en la figura 14.12.  $\bar{H}$  también se conoce como la dirección de rayos máximos. Si la superficie se orientara de manera que su normal estuviera en la misma dirección que  $\bar{H}$ , el observador vería el punto de máximo brillo especular ya que  $\bar{R}$  y  $\bar{V}$  también apuntarían en la misma dirección. El nuevo término de reflexión especular se puede expresar como  $(\bar{N} \cdot \bar{H})^n$ , donde  $\bar{H} = (\bar{L} + \bar{V}) / |\bar{L} + \bar{V}|$ . Cuando la fuente luminosa y el observador están en infinito, se obtiene una ventaja computacional al usar  $\bar{N} \cdot \bar{H}$ , ya que  $\bar{H}$  es constante. Observe que el ángulo entre  $\bar{N}$  y  $\bar{H}$ , no es igual a  $\alpha$ , el ángulo entre  $\bar{R}$  y  $\bar{V}$ , de manera que el mismo exponente especular  $n$  produce resultados diferentes en ambas formulaciones (véase el Ejer. 14.1). Aunque la utilización de un término  $\cos^n$  permite generar superficies brillantes reconocibles, hay que recordar que se basa en una observación empírica, no en un modelo teórico del proceso de reflexión especular.

### 14.1.5 Mejora del modelo de fuente luminosa puntual



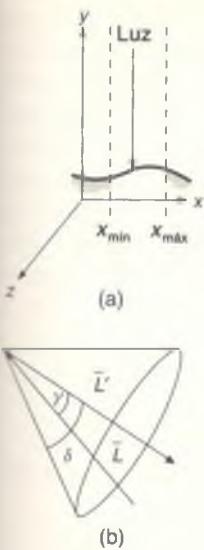
Figura 14.13

Modelo de iluminación de Warn. Una luz se modela como la reflexión especular de un solo punto iluminado por una fuente luminosa puntual.

Las fuentes luminosas reales no radian de igual manera en todas las direcciones. Warn [WARN83] ha desarrollado controles de iluminación fáciles de implementar que pueden añadirse a cualquier ecuación para modelar parte de la direccionalidad de las luces usadas por los fotógrafos. En el modelo de Phong, una fuente luminosa puntual sólo tiene intensidad y posición. En el modelo de Warn, una luz  $L'$  se modela con un punto en una superficie reflejante especular hipotética, como se muestra en la figura 14.13. Esta superficie es iluminada por una fuente luminosa puntual  $L'$  en la dirección  $\bar{L}'$ . Suponga que  $\bar{L}'$  es normal a la superficie reflejante hipotética. Entonces podemos usar la ecuación de iluminación de Phong para determinar la intensidad de  $L$  en un punto del objeto como función del ángulo  $\gamma$  entre  $\bar{L}$  y  $\bar{L}'$ . Si suponemos también que el reflector sólo refleja luz especular y que su coeficiente especular es 1, la intensidad de la luz en un punto del objeto es

$$I_{L'} \cos^p \gamma, \quad (14.18)$$

donde  $I_{L'}$  es la intensidad de la fuente luminosa puntual hipotética,  $p$  es el exponente especular del reflector y  $\gamma$  es el ángulo entre  $-\bar{L}$  y la normal a la super-



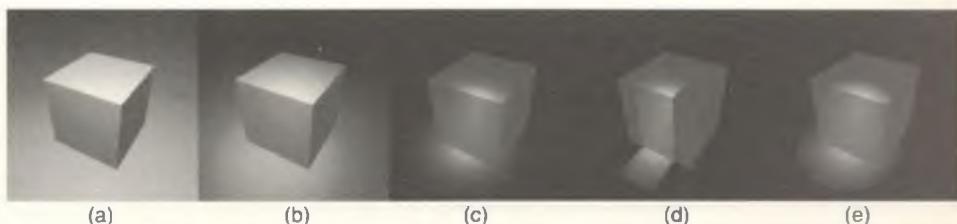
**Figura 14.14**  
Utilización de (a) aletas y  
(b) conos.

ficie hipotética,  $\bar{L}'$ , que es la dirección a  $L'$ . La ecuación (14.18) modela una fuente luminosa dirigida simétrica cuyo eje de simetría es  $\bar{L}'$ , que puede considerarse como la dirección a la cual apunta la luz. Usando productos punto, la ecuación (14.18) se puede escribir como

$$I_{L\lambda}(-\bar{L} \cdot \bar{L}')^p. \quad (14.19)$$

Una vez más, un producto punto negativo se considera cero. La ecuación (14.19) puede reemplazar entonces a la intensidad de la fuente luminosa  $I_{p\lambda}$  en la formulación de la ecuación (14.15) o cualquier otra ecuación de iluminación. Cuanto mayor sea el valor de  $p$ , más se concentrará la luz sobre  $\bar{L}'$ . Por lo tanto, un gran valor de  $p$  puede simular un reflector altamente direccional, mientras que un valor pequeño de  $p$  puede simular un faro más difuso. Si  $p$  es 0, la luz actúa como una fuente puntual uniformemente radiante. En la figura 14.15(a-c) se muestran los efectos de diversos valores de  $p$ .

Para restringir el efecto de una luz a un área limitada de la escena, Warn implantó **aletas y conos**. Las aletas, modeladas de manera muy general con base en las “puertas de granero” que se encuentran en las luces fotográficas profesionales, restringen el efecto de una luz a un intervalo designado en coordenadas mundiales  $x$ ,  $y$  y  $z$ . Cada luz tiene seis aletas, que corresponden a valores mínimos y máximos especificados por el usuario en cada coordenada. Al determinar la sombra de un punto, el modelo de iluminación se evalúa para una luz únicamente si las coordenadas del punto están dentro del intervalo especificado por las coordenadas mínimas y máximas de las aletas encendidas. Por ejemplo, si  $\bar{L}'$  es paralela al eje  $x$ , las aletas  $x$  y  $z$  pueden restringir considerablemente los efectos de la luz, en forma muy similar a las puertas en una luz fotográfica. En la figura 14.14(a) se muestra la utilización de aletas  $x$  en esta situación. Las aletas y también se pueden usar para restringir la luz en una forma que no tiene equivalente físico, permitiendo que sólo se iluminen los objetos dentro de un intervalo especificado de distancias con respecto a la luz. En la figura 14.15(d), el cubo se alinea con el sistema de coordenadas, de manera que dos pares de aletas pueden producir los efectos que se muestran.



**Figura 14.15** Cubo y plano iluminados usando controles de iluminación de Warn. (a) Fuente puntual de radiación uniforme ( $0 \leq p \leq 0$ ). (b)  $p = 4$ . (c)  $p = 32$ . (d) Aletas. (e) Cono con  $\delta = 18^\circ$ . (Por David Kurlander, Columbia University.)

Warn hace posible la creación de un reflector muy bien delineado si se usa un cono cuyo ápice es la fuente luminosa y cuyo eje está sobre  $\vec{L}'$ . Como se muestra en la figura 14.14(b), se puede usar un cono con ángulo de generación de  $\delta$  para limitar los efectos de la fuente luminosa, evaluando el modelo de iluminación sólo si  $\gamma < \delta$  (o cuando  $\cos \gamma > \cos \delta$ , puesto que  $\cos \gamma$  ya ha sido calculado). El modelo de iluminación de PHIGS PLUS incluye el término  $\cos \gamma$  de Warn y el ángulo de cono  $\delta$ . En la figura 14.15(e) se muestra la forma de usar un cono para limitar la luz de la figura 14.15(c). En la ilustración en color 21 se presenta un automóvil generado con controles de iluminación de Warn.

#### 14.1.6 Fuentes luminosas múltiples

Si hay  $m$  fuentes luminosas, se suman los términos de cada una de ellas:

$$I_\lambda = I_{\text{a}\lambda} k_a O_{\text{d}\lambda} + \sum_{1 \leq i \leq m} f_{\text{att},i} I_{\text{p}\lambda,i} [k_d O_{\text{d}\lambda} (\vec{N} \cdot \vec{L}_i) + k_s O_{\text{s}\lambda} (\vec{R}_i \cdot \vec{V})^n]. \quad (14.20)$$

La sumatoria alberga una nueva posibilidad de error, ya que ahora  $I_\lambda$  puede exceder el valor máximo que se presenta para un pixel. (Aunque esto también puede ocurrir con una sola luz, es fácil evitar la situación con una elección apropiada de  $f_{\text{att}}$  y del material.) Hay varios métodos para evitar el sobreflujo. El más sencillo es limitar cada  $I_\lambda$  en forma individual a su valor máximo. En otro método se consideran juntos todos los valores  $I_\lambda$  de un pixel. Si al menos uno de ellos es demasiado grande, cada uno se divide entre el valor mayor para mantener el tinte y la saturación a expensas del valor. Si es posible calcular todos los valores de los pixeles antes de la presentación, se pueden aplicar transformaciones de procesamiento a toda la imagen para que los valores queden dentro del intervalo deseado. Hall [HALL89] analiza los pros y contras de éstas y otras técnicas.

#### 14.1.7 Modelos de iluminación físicos

Los modelos de iluminación analizados en las secciones anteriores son en gran parte el resultado de un enfoque práctico, de sentido común, con respecto a los gráficos. Aunque las ecuaciones utilizadas aproximan algunas de las formas en que la luz interactúa con los objetos, no tienen una base física. En esta sección veremos los modelos de iluminación físicos, basándonos en parte en el trabajo de Cook y Torrance [COOK82].

Hasta ahora hemos usado la palabra *intensidad* sin definirla, refiriéndonos informalmente a la intensidad de una fuente luminosa, de un punto en una superficie o de un pixel. Es hora de formalizar nuestros términos presentando la terminología radiométrica utilizada en el estudio de la radiación térmica, que es la base de nuestra comprensión de la interacción entre la luz y los objetos [NICO77; SPARR78; SIEG81; IES87]. Comenzaremos por el *flujo*, que es la tasa con la cual se emite la energía luminosa, y se mide en watts ( $W$ ). Para referirnos

a la cantidad de flujo emitida o recibida en una dirección determinada, se requiere el concepto de **ángulo sólido**, que es el ángulo en el ápice de un cono. El ángulo sólido se mide en función del área sobre una esfera interceptada por un cono cuyo ápice está en el centro de una esfera. Un **estereoradián** (sr) es el ángulo sólido de uno de estos conos que intercepta un área igual al cuadrado del radio  $r$  de la esfera. Si un punto está en una superficie, nos interesa el hemisferio sobre él. Como el área de una esfera es  $4\pi r^2$ , hay  $4\pi r^2/2r^2 = 2\pi$  sr en un hemisferio. Imagine la proyección de la forma de un objeto a un hemisferio centrado alrededor de un punto en la superficie que sirve como centro de proyección. El ángulo  $\omega$  subtendido por el objeto es el área en el hemisferio ocupada por la proyección, dividida entre el cuadrado del radio del hemisferio (la división elimina la dependencia del tamaño del hemisferio).

La **intensidad radiante** es el flujo que se radia a un ángulo sólido unidad en una dirección específica y se mide en W/sr. Cuando hemos usado la palabra **intensidad** haciendo referencia a una fuente puntual, nos hemos referido a su intensidad radiante.

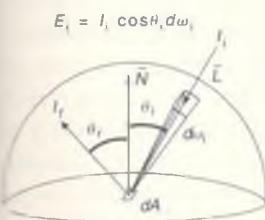
La **radiancia** es la intensidad radiante por área unidad de superficie escorzada frontalmente y se mide en W/(sr · m<sup>2</sup>). El **área de superficie recortada frontalmente**, también conocida como **área de superficie proyectada**, se refiere a la proyección de la superficie sobre el plano perpendicular a la dirección de la radiación. El área de superficie escorzada frontalmente se determina multiplicando el área de la superficie por  $\cos \theta_r$ , donde  $\theta_r$  es el ángulo de la luz radiada con respecto a la normal a la superficie. Un ángulo sólido pequeño  $d\omega$  se puede aproximar como el área de superficie escorzada frontalmente, dividida entre el cuadrado de la distancia del objeto al punto donde se calcula el ángulo sólido. Cuando empleamos la palabra **intensidad** haciendo referencia a una superficie, nos estamos refiriendo a su radiancia. Por último, la **irradiancia**, también conocida como **densidad de flujo**, es el flujo incidente por unidad de área de superficie (recortada frontalmente) y se mide en W/m<sup>2</sup>.

En los gráficos nos interesa la relación entre la luz que incide sobre una superficie y la luz que se refleja y transmite en ella. Considere la figura 14.16. La irradiancia de la luz incidente es

$$E_i = I_i(\bar{N} \cdot \bar{L}) d\omega_i,$$

donde  $I_i$  es la radiancia de la luz incidente y  $\bar{N} \cdot \bar{L}$  es  $\cos \theta_i$ . Como la irradiancia se expresa por unidad de área, mientras que la radiancia se expresa por unidad de área escorzada frontalmente, se multiplica por  $\bar{N} \cdot \bar{L}$  para convertirla al equivalente por unidad de área no recortada frontalmente.

No es suficiente considerar sólo  $I_i$  (la radiancia incidente) al determinar  $I_r$  (la radiancia reflejada); hay que considerar en su lugar  $E_i$  (la irradiancia incidente). Por ejemplo, un haz incidente que tiene la misma intensidad radiante (W/sr) que otro haz, pero con mayor ángulo sólido, tiene una  $E_i$  proporcionalmente mayor y hace que la superficie aparezca proporcionalmente más brillante. La razón entre la radiancia reflejada (intensidad) en una dirección y la



**Figura 14.16**  
Radiancia reflejada e  
radiancia incidente.

irradiancia incidente (densidad de flujo) responsable de ella en otra dirección, se conoce como **reflectividad bidireccional**,  $\rho$ , que es una función de las direcciones de incidencia y reflexión,

$$\rho = \frac{I_r}{E_i}$$

Como hemos visto, en la graficación por computador es convencional considerar la reflectividad bidireccional como compuesta por componentes difusas y especulares. Por lo tanto,

$$\rho = k_d \rho_d + k_s \rho_s,$$

donde  $\rho_d$  y  $\rho_s$  son, respectivamente, las reflectividades bidireccionales especular y difusa, mientras que  $k_d$  y  $k_s$  son respectivamente, los coeficientes de reflexión difusa y especular presentados antes en este capítulo.

El modelo de superficie de Torrance-Sparrow [TORR66; TORR67], desarrollado por físicos, es un modelo físico de una superficie reflejante. Blinn fue el primero en adaptar el modelo de Torrance-Sparrow a la graficación por computador, proporcionando detalles matemáticos y comparándolo con el modelo de Phong en [BLIN77a], Cook y Torrance [COOK82] fueron los primeros en aproximar la composición espectral de la luz reflejada en una implantación del modelo.

En el modelo de Torrance-Sparrow, se supone que la superficie es una colección isotrópica de facetas microscópicas planas, cada una de ellas un reflector perfectamente liso. La geometría y distribución de estas **microfacetas** y la dirección de la luz (que se supone emana de una fuente infinitamente distante, de manera que todos los rayos son paralelos) determinan la intensidad y la dirección de la reflexión especular como función de  $I_p$  (la intensidad de la fuente luminosa puntual),  $\bar{N}$ ,  $\bar{L}$  y  $\bar{V}$ . Las mediciones experimentales indican que hay muy buena correspondencia entre la reflexión real y la reflexión pronosticada por este modelo [TORR67].

Para el componente especular de la reflectividad bidireccional, Cook y Torrance usan

$$\rho_s = \frac{F_\lambda}{\pi} \frac{DG}{(\bar{N} \cdot \bar{V})(\bar{N} \cdot \bar{L})}, \quad (14.21)$$

donde  $D$  es una función de distribución de las orientaciones de las microfacetas.  $G$  es el **factor de atenuación geométrica**, que representa los efectos de enmascaramiento y sombreado de las microfacetas entre sí y  $F_\lambda$  es el término de Fresnel calculado usando la ecuación de Fresnel, la cual, para la reflexión especular, relaciona la luz incidente con la luz reflejada para la superficie lisa de cada microfaceta. El término  $\pi$  en el denominador supuestamente debe considerar la aspera de la superficie (pero véase [JOY88, págs. 227-230] para una descripción general de la obtención de esta ecuación). El término  $\bar{N} \cdot \bar{V}$  hace la ecuación

ción proporcional al área de la superficie (y por ende al número de microfacetas) que ve el observador en una porción unidad de área de superficie recortada frontalmente, mientras que el término  $\bar{N} \cdot \bar{L}$  hace que la ecuación sea proporcional al área de superficie que la luz ve en una porción unidad de área de superficie recortada frontalmente. Los detalles de los términos que constituyen la ecuación (14.21) se incluyen en la sección 16.7 de [FOLE90]; aquí sólo presentan resultados.

En la ilustración en color 40 se presentan dos jarrones de cobre generados con el modelo de Cook-Torrance; ambos usan la reflectancia bidireccional del cobre para el término difuso. El primer jarrón modela el término especular usando la reflectancia de un espejo de vinilo y representa resultados similares a los que se obtienen con el modelo original de iluminación de Phong de la ecuación (14.14). El segundo jarrón modela el término especular con la reflectancia de un espejo de cobre. Observe cómo, al considerar la dependencia del color de realce especular tanto en el ángulo de incidencia como en el material de la superficie, se produce una imagen más convincente de una superficie metálica.

En términos generales, los componentes ambientales, difusos y especulares son del mismo color que el material para dielectricos y conductores. Los objetos compuestos, como los plásticos, usualmente tienen componentes difusos y especulares de colores diferentes. Los metales suelen presentar poca reflexión difusa y tienen un color de componente especular que varía entre el del metal y el de la fuente luminosa conforme  $\theta_i$  se aproxima a  $90^\circ$ . Esta observación sugiere una aproximación al modelo de Cook-Torrance que usa la ecuación (14.15) con  $O_{s\lambda}$  elegido a partir de la interpolación en una tabla de consulta basada en  $\theta_i$ .

Se han realizado varias mejoras y generalizaciones en el modelo de iluminación de Cook-Torrance (p. ej., véase [KAJI85; CABR87; WOLF90]). El trabajo reciente efectuado por He *et al.* [HE92] demuestra un método rápido y preciso para aplicar estos modelos físicos. Es interesante señalar que [HE92] es una publicación multimedia que permite al lector explorar de manera interactiva el efecto de varios de los términos en la ecuación (14.21).

## 14.2 Modelos de sombreado para polígonos

Debe ser evidente que podemos sombrear cualquier superficie calculando la normal a la superficie en cada punto visible y aplicando el modelo de iluminación deseado en dicho punto. Por desgracia, este método burdo para el sombreado es muy costoso. En esta sección describiremos modelos de sombreado más eficientes para superficies definidas por polígonos y mallas poligonales.

### 14.2.1 Sombreado constante

El modelo de sombreado más sencillo para un polígono es el **sombreado constante**, también conocido como **sombreado facetado** o **sombreado plano**. Este

método aplica un modelo de iluminación una vez para determinar un solo valor de intensidad que después se usa para sombrear todo un polígono. En esencia se muestrea el valor de la ecuación de iluminación una vez para cada polígono y se mantiene el valor en todo el polígono para reconstruir su sombra. ~~Ese~~ método es válido si son verdaderas varias suposiciones:

1. La fuente luminosa está en infinito, de manera que  $\bar{N} \cdot \bar{L}$  es constante en toda la cara del polígono.
2. El observador está en infinito, de manera que  $\bar{N} \cdot \bar{V}$  es constante en toda la cara del polígono.
3. El polígono representa la superficie real que se modela y no es una ~~aproximación~~ a una superficie curva.

Si se usa un algoritmo de superficies visibles que produzca una lista de polígonos, como uno de los algoritmos de prioridad de listas, el sombreado constante puede aprovechar la ubicua primitiva de polígono bidimensional de un solo color.

Cuando ninguna de las dos primeras suposiciones es incorrecta, entonces, vamos a usar sombreado constante, es necesario un método para determinar solo valor para  $\bar{L}$  y  $\bar{V}$ . Por ejemplo, los valores se pueden calcular para el centro del polígono o para su primer vértice. Por supuesto, el sombreado constante produce las variaciones en sombreado a través del polígono que deben ocurrir en esta situación.

### 14.2.2 Sombreado interpolado

Como alternativa a la evaluación de la ecuación de iluminación en cada punto del polígono, Wylie, Romney, Evans y Erdahl [WYLI67] promovieron el uso del **sombreado interpolado**, en el cual la información de sombreado se interpola linealmente sobre un triángulo a partir de valores determinados para sus vértices. Gouraud [GOUR71] generalizó esta técnica para polígonos arbitrarios. ~~Ese~~ método es muy fácil de usar en un algoritmo de línea de barrido que interpola el valor  $z$  en un tramo a partir de valores  $z$  interpolados que se calculan para los puntos extremos del tramo.

Es posible usar una ecuación de diferencia, como la que se desarrolló en la sección 13.2, para aumentar la eficiencia en la determinación del valor  $z$  en cada pixel. Aunque la interpolación  $z$  es físicamente correcta (suponiendo que el polígono es plano), observe que el sombreado interpolado no lo es, ya que sólo approxima la evaluación del modelo de iluminación en cada punto del polígono.

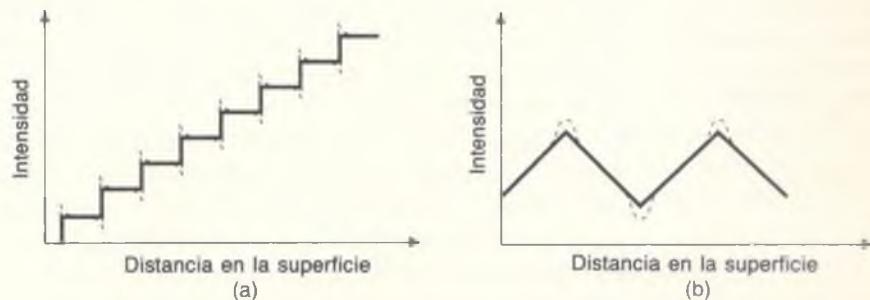
Nuestra suposición final, de que el polígono representa correctamente la superficie que se modela, es incorrecta en la mayoría de los casos y tiene un efecto más sustancial en la imagen resultante que la falla de las otras dos suposiciones. Muchos objetos son curvos, no poliédricos, pero al representarse como una

malla poligonal se pueden aprovechar los eficientes algoritmos de superficies visibles poligonales. A continuación analizaremos cómo generar una malla poligonal para que sea lo más parecida posible a una superficie curva.

### 14.2.3 Sombreado de malla poligonal

Suponga que queremos aproximar una superficie curva con una malla poligonal. Si cada faceta poligonal de la malla se sombra individualmente, será fácil distinguirla de los vecinos cuya orientación sea diferente, produciendo una apariencia “facetada”, como se muestra en la ilustración en color 28. Esto es cierto si los polígonos se generan usando sombreado constante, sombreado interpolado o incluso cálculos de iluminación por pixel, ya que dos polígonos adyacentes de orientación diferente tienen distintas intensidades en sus fronteras. La solución simple de usar una malla más fina es, sorprendentemente, poco efectiva, porque la diferencia percibida en el sombreado de facetas adyacentes se acentúa por el efecto de banda de Mach (descubierto por Mach en 1865 y descrito con detalle en [RATL72]), que exagera el cambio de intensidad en cualquier arista donde haya una discontinuidad en la magnitud o pendiente de intensidad. En la frontera entre dos facetas, la faceta oscura se ve más oscura y la clara se ve más clara. En la figura 14.17 se muestran los cambios reales y los percibidos en la intensidad de una superficie para dos casos separados.

El efecto de banda de Mach es ocasionado por la **inhibición lateral** de los receptores del ojo. Cuanto más luz reciba un receptor, más inhibe la respuesta de los receptores adyacentes. La respuesta de un receptor a la luz es inhibida por sus receptores adyacentes en relación inversa a la distancia al receptor adyacente. Los receptores que están directamente en el lado más brillante de un cambio de intensidad tienen mayor respuesta que los que están en el lado más brillante pero más lejos de la arista, ya que reciben menos inhibición de sus vecinos en el lado oscuro. En forma similar, los receptores inmediatos al lado oscuro de un cambio de intensidad tienen menor respuesta que los que están más lejos en el



**Figura 14.17** Dos ejemplos de las intensidades reales y percibidas en el efecto de la banda de Mach. Las líneas punteadas son la intensidad percibida; las líneas sólidas son la intensidad real.

área oscura, ya que reciben más inhibición de sus vecinos del lado brillante. El efecto de la banda de Mach es bastante evidente en la ilustración en color 29 sobre todo entre polígonos adyacentes de color similar.

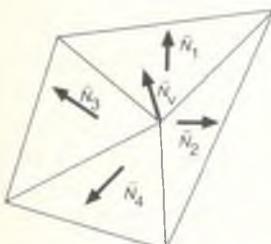
Los modelos de sombreado de polígonos que hemos descrito determinan la sombra de un polígono en forma individual. Dos modelos de sombreado básicos para mallas poligonales aprovechan la información proporcionada por polígonos adyacentes para simular una superficie suave. Estos modelos se conocen, en orden de mayor complejidad (y efecto realista), como sombreado Gouraud y sombreado de Phong, en honor de los investigadores que los desarrollaron. Las estaciones de trabajo de graficación tridimensional que se usan en la actualidad por lo general permiten emplear uno o ambos métodos a través de una combinación de hardware y firmware.

#### 14.2.4 Sombreado de Gouraud

El **sombreado de Gouraud** [GOUR71], también llamado **sombreado de interpolación de intensidad** o **sombreado de interpolación de color**, elimina las discontinuidades de intensidad. La ilustración en color 29 usa sombreado de Gouraud. Aunque gran parte del efecto de banda de Mach de la ilustración en color 28 ya no es visible en la ilustración en color 29, las tiras brillantes en los objetos como el toro y el cono son bandas de Mach ocasionadas por cambios rápidos, aunque no discontinuos, en la pendiente de la curva de intensidad; el sombreado Gouraud no elimina por completo estos cambios de intensidad.

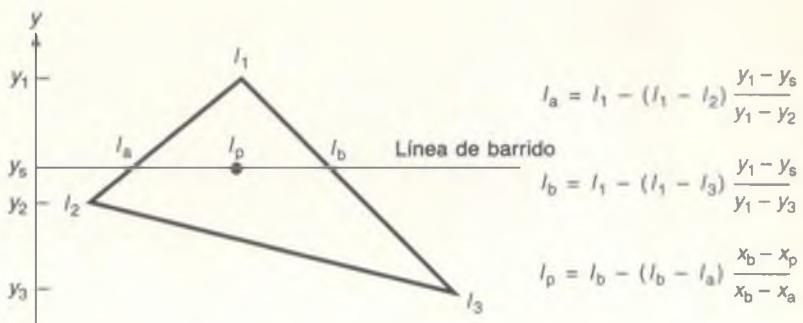
El sombreado de Gouraud extiende el concepto del sombreado interpolado aplicado a polígonos individuales, interpolando los valores de iluminación de los vértices de polígonos que toman en cuenta la superficie que se aproxima. El proceso de sombreado de Gouraud requiere que se conozca la normal a cada vértice de la malla poligonal. Gouraud pudo calcular estas **normales a vértices** directamente de la descripción analítica de la superficie. Alternativamente, si las normales a los vértices no se almacenan con la malla y no se pueden determinar directamente de la superficie real, Gouraud propone que se aproximen promediando las normales a la superficie de todas las facetas poligonales que componen cada vértice (Fig. 14.18). Si una arista será visible (como la unión entre la ala y el cuerpo de un avión), se encuentran dos normales a vértices, una para cada lado de la arista, promediando por separado las normales de los polígonos a cada lado de la arista.

El siguiente paso en el sombreado de Gouraud es encontrar **intensidades de vértices** usando las normales a los vértices con cualquier modelo de iluminación que se deseé. Finalmente, cada polígono se sombra por interpolación lineal de las intensidades de los vértices a lo largo de cada arista y luego entre las aristas de cada línea de barro (Fig. 14.19), de la misma forma que se describió para la interpolación de valores  $z$  en la sección 13.2. El término *sombreado de Gouraud* muchas veces se generaliza para hacer referencia al sombreado de interpolación de intensidad incluso de un polígono aislado, o a la interpolación de colores arbitrarios relacionados con vértices de polígonos.



**Figura 14.18**

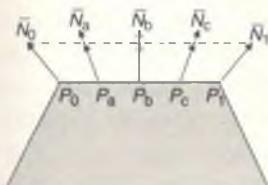
Las normales a la superficie de un polígono normalizado se pueden promediar para obtener las normales a los vértices. La normal promediada  $N$ , es  $\sum_{i=1}^n N_i / |\sum_{i=1}^n N_i|$ .



**Figura 14.19** Interpolación de intensidad en aristas de polígonos y líneas de rastreo.

La interpolación a lo largo de aristas también se puede integrar fácilmente con el algoritmo de línea de barrido para superficies visibles de la sección 13.3. Para cada arista, se almacenan la intensidad inicial y el cambio de intensidad de cada componente de color por cada cambio unidad en  $y$ . Un tramo visible en una línea de rastreo se rellena interpolando los valores de intensidad de las dos aristas que acotan el tramo. Como en todos los algoritmos de interpolación lineal, se puede usar una ecuación de diferencia para aumentar la eficiencia.

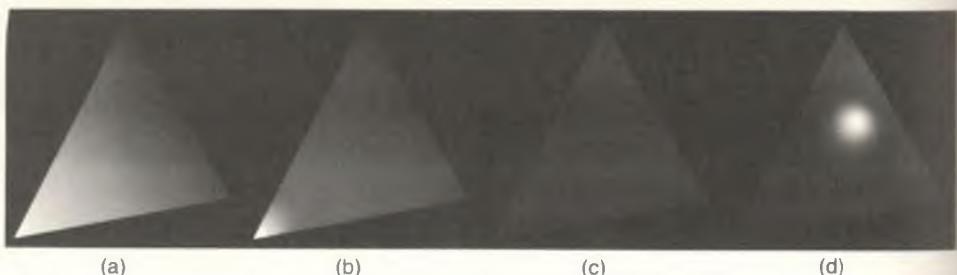
#### 14.2.5 Sombreado de Phong



**Figura 14.20**  
Interpolación de vectores normales (basado en [BUIT75]).

El sombreado de Phong [BUIT75], también conocido como **sombreado de interpolación de vector normal**, interpola el vector normal a la superficie,  $\bar{N}$ , en lugar de la intensidad. La interpolación ocurre en un tramo de polígono sobre una línea de rastreo, entre las normales inicial y final del tramo. A su vez, estas normales se interpolan en las aristas de los polígonos a partir de normales a vértices que se calculan, de ser necesario, como en el sombreado de Gouraud. La interpolación a lo largo de aristas se puede efectuar, una vez más, con cálculos incrementales, aumentando los tres componentes del vector normal de una línea de rastreo a otra. En cada pixel de una línea de rastreo, la normal interpolada se normaliza y se establece la correspondencia de regreso al sistema de coordenadas mundiales o a uno isométrico, donde se calcula una nueva intensidad usando cualquier modelo de iluminación. En la figura 14.20 se muestran dos normales a aristas y las normales interpoladas a partir de ellas, antes y después de la normalización.

Las ilustraciones en color 30 y 31 se generaron usando sombreados de Gouraud y de Phong, respectivamente, y una ecuación de iluminación con término de reflectancia especular. El sombreado de Phong produce mejoras notables con respecto al sombreado de Gouraud cuando se usan estos modelos de iluminación, ya que los puntos de máximo brillo se producen de manera más fiel,



**Figura 14.21** Modelo de iluminación de reflexión especular usado con sombreado de Gouraud y de Phong. El punto de máximo brillo cae en el vértice izquierdo: (a) sombreado de Gouraud, (b) sombreado de Phong. El punto de máximo brillo cae en el interior del polígono: (c) sombreado de Gouraud, (d) sombreado de Phong. (Por David Kurlander, Columbia University.)

como se ilustra en la figura 14.21. Considere lo que sucede si la  $n$  en el término de iluminación  $\cos^n \alpha$  de Phong es grande y uno de los vértices tiene  $\alpha$  muy pequeña, pero sus vértices adyacentes tienen  $\alpha$  muy grande. La intensidad asociada al vértice con  $\alpha$  pequeña será apropiada para un punto de máximo brillo, mientras que los otros vértices no tendrán intensidades de punto de máximo brillo. Si se emplea el sombreado de Gouraud, la intensidad en el polígono se interpola linealmente entre la intensidad de punto de máximo brillo y las intensidades menores de los vértices adyacentes, distribuyendo el punto de máximo brillo por el polígono (Fig. 14.21a). Compare esto con la rápida caída en la intensidad de punto de máximo brillo que se calcula al usar normales interpoladas linealmente para calcular el término  $\cos^n \alpha$  en cada pixel (Fig. 14.21b). Además, si un punto de máximo brillo no cae sobre un vértice, es posible que el sombreado de Gouraud lo omita (Fig. 14.21c), ya que ningún punto interior puede ser más brillante que el vértice de mayor brillo a partir del cual se interpola. En cambio, el sombreado de Phong permite localizar realces en el interior de un polígono (Fig. 14.21d). Compare los puntos de máximo brillo en la penumbra de las ilustraciones en color 30 y 31.

Incluso con un modelo de iluminación que no considera la reflectancia especular, los resultados de la interpolación de vectores normales son generalmente superiores a los que se obtienen con interpolación de intensidad, ya que se usa una aproximación a la normal en cada punto. Esto reduce los problemas de banda de Mach en la mayoría de los casos, pero aumenta considerablemente el costo del sombreado en una implantación directa, ya que es necesario normalizar la normal interpolada cada vez que se use en un modelo de iluminación diferente. Duff [DUFF79] ha desarrollado una combinación de ecuaciones de diferencia y consulta de tabla para acelerar el cálculo. Bishop y Weimer [BISH86] proporcionan una excelente aproximación al sombreado de Phong usando un desarrollo de serie de Taylor que ofrece mayores aumentos en la velocidad del sombreado.

#### 14.2.6 Problemas con el sombreado interpolado

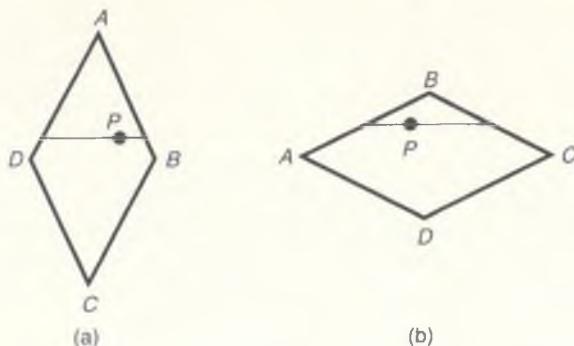
Todos estos modelos de sombreado interpolado tienen algunos problemas comunes, algunos de los cuales describiremos a continuación.

**Silueta poligonal.** No importa cuán buena sea la aproximación de un modelo de sombreado interpolado con respecto al sombreado real de una superficie curva, la orilla de silueta de la malla es evidentemente poligonal. Podemos mejorar esta situación dividiendo la superficie en un número mayor de polígonos más pequeños, pero con un aumento correspondiente en costo.

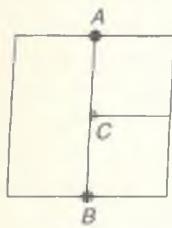
**Distorsión de perspectiva.** Se introducen anomalías porque la interpolación se lleva a cabo después de la transformación de perspectiva en el sistema tridimensional de coordenadas de pantalla, en lugar de hacerlo en el sistema de coordenadas de mundo. Por ejemplo, la interpolación lineal hace que la información de sombreado de la figura 14.19 aumente en una cantidad constante de una línea de barrido a otra sobre cada arista. Considere lo que sucede si el vértice 1 está más distante que el vértice 2. Por el escorzo frontal de perspectiva, la diferencia de una línea de barrido a otra en el valor  $z$  no transformado a lo largo de una arista aumenta en la dirección de la coordenada más lejana. Por lo tanto, si  $y_s = (y_1 + y_2)/2$ , entonces  $I_s = I_1 + I_2)/2$ , pero  $z_s$  no será igual a  $(z_1 + z_2)/2$ . Este problema también se puede reducir usando un mayor número de polígonos más pequeños. La reducción en el tamaño de los polígonos aumenta el número de puntos en los cuales se muestrea la información para la interpolación, lo cual aumenta la precisión del sombreado.

**Dependencia de la orientación.** Los resultados de los modelos de sombreado por interpolación no son independientes de la orientación del polígono proyectado. Como los valores se interpolan entre vértices y a lo largo de líneas de rastreo horizontales, los resultados pueden variar al rotar el polígono (véase Fig. 14.22). Este efecto es muy obvio cuando la orientación cambia lentamente entre cuadros sucesivos de una animación. También puede ocurrir un problema similar en la determinación de superficies visibles cuando se interpola el valor  $z$  en cada punto a partir de los valores  $z$  asignados a cada vértice. Ambos problemas se pueden resolver descomponiendo los polígonos en triángulos (véase el Ejer. 14.2). Alternativamente, Duff [DUFF79] propone métodos de interpolación independientes de la rotación, aunque costosos, que resuelven el problema sin tener que efectuar la descomposición.

**Problemas en vértices compartidos.** Pueden ocurrir discontinuidades de sombreado cuando dos polígonos adyacentes no comparten un vértice que está sobre su arista común. Considere los tres polígonos de la figura 14.23, donde el vértice  $C$  es compartido por los dos polígonos a la derecha pero no por el polígono de gran tamaño a la izquierda. La información de sombreado que se determina directamente en  $C$  para los polígonos a la derecha generalmente será



**Figura 14.22** Los valores interpolados obtenidos para un punto  $P$  en el mismo polígono con distinta orientación varían de (a) a (b).  $P$  interpola  $A$ ,  $B$  y  $D$  en (a) y  $A$ ,  $B$  y  $C$  en (b).

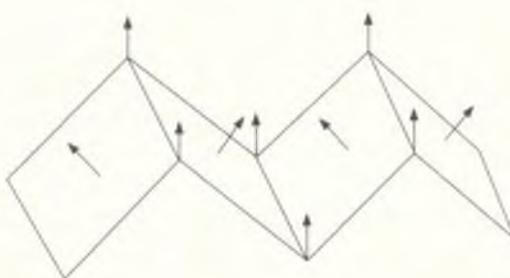


**Figura 14.23**

El vértice  $C$  es compartido por los dos polígonos a la derecha, pero no por el polígono rectangular de mayor tamaño que está a la izquierda.

distinta de la información interpolada en ese punto a partir de los valores en  $A$  y  $B$  para el polígono de la izquierda. Como resultado, habrá una discontinuidad en el sombreado, la cual se puede eliminar insertando en el polígono de la izquierda un vértice adicional que comparta la información de sombreado de  $C$ . Podemos preprocesar una base de datos poligonal estática para eliminar este problema; por otra parte, si los polígonos se van a dividir sobre la marcha (e.g., usando el algoritmo de árbol BSP para superficies visibles), se pueden realizar tareas de control adicionales para introducir un nuevo vértice en una arista que comparta una arista dividida.

**Normales a vértices no representativas.** Las normales a vértices calculadas siempre representan de manera adecuada la geometría de la superficie. Por ejemplo, si calculamos las normales a los vértices promediando las normales a las superficies que comparten un vértice, todas las normales en la figura 14.24 serán paralelas entre sí, con lo cual hay muy poca o ninguna variación en



**Figura 14.24** Problemas con el cálculo de normales a vértices. Todas las normales a los vértices son paralelas.

sombreado si la fuente luminosa está distante. Este problema se resolverá subdividiendo los polígonos antes de calcular las normales a los vértices.

Estos problemas han originado mucho trabajo para producir algoritmos que manejen en forma directa superficies curvas, pero los polígonos son bastante más rápidos (y fáciles) de procesar y por lo tanto siguen siendo la base de la mayoría de los sistemas de generación de imágenes.

## 14.3 Detalle de superficie

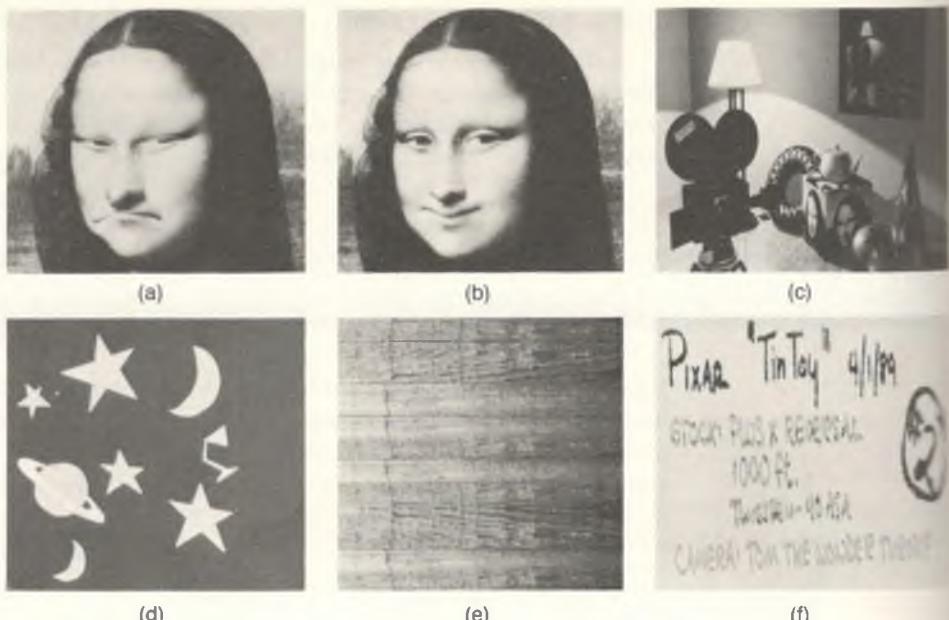
La aplicación de cualquiera de los modelos de sombreado que hemos descrito hasta ahora a superficies planas o bicúbicas produce superficies suaves y uniformes, muy diferentes de la mayoría de las superficies que vemos y sentimos. A continuación analizaremos varios métodos desarrollados para simular este detalle faltante de las superficies.

### 14.3.1 Polígonos de detalle de superficie

El método más sencillo añade un detalle grueso a través de *polígonos de detalle de superficie*, para mostrar características (como puertas, ventanas y letras) en un polígono base (como el costado de un edificio). Cada polígono de detalle de superficie es coplanar con su polígono base y se marca de manera que no tenga que compararse con otros polígonos durante la determinación de superficies visibles. Cuando se sombra el polígono base, los polígonos de detalle de superficie y sus propiedades materiales tienen precedencia sobre las partes que cubren en el polígono base.

### 14.3.2 Correspondencia de texturas

Conforme el detalle se hace más fino y complicado, deja de ser práctico el modelado explícito con polígonos u otras primitivas geométricas. Una alternativa es establecer una correspondencia entre una imagen, ya sea digitalizada o sintetizada, y una superficie, técnica desarrollada por Catmull [CATM74] y refinada por Blinn y Newell [BLIN76]. Este método se conoce como **correspondencia de textura** o **correspondencia de patrones**; la imagen se conoce como **mapa de textura** y sus elementos individuales con frecuencia se denominan **elementos de textura** (o **texoles**). El mapa de textura rectangular reside en su propio espacio de coordenadas de textura ( $u, v$ ). Como alternativa, la textura puede estar definida por un procedimiento. La ilustración en color 34 muestra varios ejemplos de correspondencia de texturas, usando las texturas mostradas en la figura 14.25. En cada pixel generado, los elementos de textura seleccionados se usan para sustituir o escalar una o más de las propiedades materiales de la superficie, como sus componentes de color difuso. Con frecuencia un pixel estará cubierto



**Figura 14.25** Texturas utilizadas para crear la ilustración en color 34. (a) Mona Lisa enojada. (b) Mona Lisa sonriente. (c) Cuadro. (d) Sombrero de hechicero. (e) Piso. (f) Etiqueta de película. (Copyright © 1990, Pixar. Imágenes generadas por Thomas Williams y H. B. Siegel usando el software PhotoRealistic RenderMan™ de Pixar.)

por varios elementos de textura. Para evitar problemas de artefactos de discretización, es necesario considerar todos los elementos de textura relevantes.

Como se muestra en la figura 14.26, la correspondencia de texturas se pude lograr en dos pasos. Una estrategia sencilla comienza estableciendo la correspondencia entre las cuatro esquinas del pixel y la superficie. En el caso de un parche bicúbico, esta correspondencia define naturalmente un conjunto de puntos en el espacio de coordenadas  $(s, t)$  de la superficie. Después se establece la correspondencia entre los puntos esquina del pixel en el espacio de coordenadas  $(s, t)$  de la superficie y el espacio de coordenadas  $(u, v)$  de la textura. Los cuatro puntos  $(u, v)$  en el mapa de textura definen un cuadrilátero que aproxima la forma más compleja a la cual podría corresponder en realidad el pixel, debido a la curvatura de la superficie. Se calcula un valor para cada pixel sumando todos los elementos de textura que caigan dentro del cuadrilátero, ponderando cada uno por una fracción del elemento de textura que está dentro del cuadrilatero. Si un punto transformado en el espacio  $(u, v)$  cae fuera del mapa de textura, se puede considerar que éste se duplica, como los patrones de la sección 2.1.3. En lugar de usar siempre la correspondencia de identidad entre  $(s, t)$  y  $(u, v)$ , se puede definir una correspondencia entre las cuatro esquinas del rectángulo 0 a 1 en  $(s, t)$  y un cuadrilátero en  $(u, v)$ . Cuando la superficie es un polígono,



**Figura 14.26** Correspondencia de textura entre un pixel y la superficie y el mapa de textura.

es común asignar coordenadas de mapa de textura directamente a sus vértices. Como hemos visto, los valores linealmente interpolantes a través de polígonos arbitrarios dependen de la orientación, por lo cual los polígonos se pueden descomponer primero en triángulos. Sin embargo, incluso después de la triangulación, la interpolación lineal ocasionará distorsión en el caso de la proyección de perspectiva. Esta distorsión será más notoria que la que se produce al interpolar otra información de sombreado, ya que las características de textura no se escorzarán frontalmente de manera correcta. Podemos obtener una solución aproximada para este problema descomponiendo los polígonos en otros más pequeños, o también una solución exacta, a mayor costo, realizando la división de perspectiva durante la interpolación. Heckbert [HECK86b] proporciona un estudio muy detallado de los métodos de correspondencia de textura.

### 14.3.3 Correspondencia de protuberancias

La correspondencia de texturas afecta el sombreado de una superficie, pero ésta sigue apareciendo geométricamente suave. Si el mapa de textura es una fotografía de una superficie áspera, la superficie que se sombra no se verá bien, ya que la dirección hacia la fuente luminosa usada para crear el mapa de textura generalmente es diferente de la dirección hacia la fuente luminosa que ilumina la superficie. Blinn [BLIN78b] desarrolló una forma de proporcionar la apariencia de una geometría de superficie modificada que evita el modelado geométrico explícito. Su estrategia implica perturbar la normal a la superficie antes de usarla en el modelo de iluminación, como una ligera asperidad en una superficie perturbaría la normal a la superficie. Este método se conoce como **correspondencia de protuberancias** y se basa en la correspondencia de texturas.

Un **mapa de protuberancias** es un arreglo de desplazamientos, cada uno de los cuales se puede usar para simular el desplazamiento de un punto en la superficie a una posición ligeramente arriba o abajo de la posición real del punto. Los resultados de la correspondencia de protuberancias pueden ser muy convin-

centes. Los observadores muchas veces no notan que la textura de un objeto no afecta las orillas de su silueta. Las ilustraciones en color 38 y 39 presentan dos ejemplos de correspondencia de protuberancias.

#### 14.3.4 Otros métodos

Aunque la correspondencia bidimensional puede ser efectiva en muchas situaciones, en ocasiones no produce resultados convincentes. Las texturas muchas veces sacan a relucir su origen bidimensional cuando se establece una correspondencia entre ellas y superficies curvas, y también se presentan problemas en las *uniones* de la textura. Por ejemplo, cuando se establece la correspondencia entre una textura de madera y la superficie de un objeto curvo, el objeto parecerá pintado con la textura. Peachey [PEAC85] y Perlin [PERL85] han investigado el empleo de texturas sólidas para la generación apropiada de objetos *tallados* en madera o mármol. En este método, descrito en el capítulo 20 de [FOLE90], la textura es una función tridimensional de su posición en el objeto.

También se puede establecer la correspondencia de otras propiedades de las superficies. Por ejemplo, Cook ha implantado una **correspondencia de desplazamiento** con la cual se desplaza la superficie real, no sólo las normales a ella [COOK84]; este proceso, que debe efectuarse antes de la determinación de superficies visibles, se usó para modificar las superficies del cono y el toro de la ilustración en color 35. En la sección 9.5.1 se analiza la utilización de fractales para crear geometrías muy detalladas a partir de una descripción geométrica inicial simple.

Hasta ahora hemos supuesto tácitamente que el proceso de sombreado de un punto en un objeto no es afectado por el resto del objeto ni por cualquier otro. Sin embargo, un objeto podría de hecho estar bajo la sombra de otro objeto que se encontrara entre el objeto y una fuente luminosa; podría transmitir luz, permitiendo ver otro objeto a través de él; o podría reflejar otros objetos, permitiendo que se viera en él otro objeto. En las secciones siguientes describiremos cómo modelar algunos de estos efectos.

### 14.4 Sombras

Los algoritmos de superficies visibles determinan cuáles son las superficies que pueden verse desde el punto de observación; los algoritmos de sombras determinan cuáles son las superficies que se pueden “ver” desde la fuente luminosa. Por lo tanto, los algoritmos de superficies visibles y de sombras son en esencia los mismos. Las superficies que son visibles desde la fuente luminosa no están bajo sombra; las que no son visibles desde la fuente están bajo sombra. Cuando hay varias fuentes luminosas, hay que clasificar una superficie con respecto a cada una de ellas.

Aquí consideraremos los algoritmos de sombra para las fuentes luminosas puntuales; las fuentes luminosas extendidas se analizan en [FOLE90], capítulo 16. La visibilidad desde una fuente luminosa puntual es, como la visibilidad desde el punto de observación, un aspecto absoluto: todo o nada. Cuando un punto en la superficie no se puede ver desde una fuente luminosa, el cálculo de iluminación se debe ajustar para considerarlo. La adición de sombras a la ecuación de iluminación genera

$$I_{\lambda} = I_{a\lambda} k_a O_{d\lambda} + \sum_{1 \leq i \leq m} S_i f_{alt_i} I_{p\lambda_i} [k_d O_{d\lambda} (\bar{N} \cdot \bar{L}_i) + k_s O_{sh} (\bar{R}_i \cdot \bar{V})^n], \quad (14.22)$$

donde

$$S_i = \begin{cases} 0, & \text{si la luz } i \text{ está bloqueada en este punto;} \\ 1, & \text{si la luz } i \text{ no está bloqueada en este punto.} \end{cases}$$

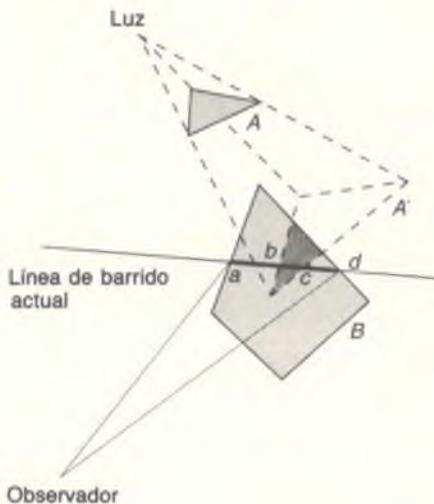
Observe que las áreas en la sombra de todas las fuentes luminosas puntuales continúan iluminadas por la luz ambiental.

Aunque el cálculo de sombras requiere calcular la visibilidad desde la fuente luminosa, como hemos señalado, también es posible generar sombras “falsas” sin llevar a cabo pruebas de visibilidad. Estas sombras se pueden crear con eficiencia transformando cada objeto en su proyección poligonal desde una fuente luminosa puntual a un plano de suelo designado, sin recortar el polígono transformado a la superficie que sombreá ni revisar si está bloqueado por otras superficies [BLIN88]. Estas sombras se tratan después como polígonos de detalle de superficie. En el caso general, en el cual estas sombras falsas no son adecuadas, pueden aplicarse varios métodos de generación de sombras. Podríamos realizar primero todo el procesamiento de sombras, combinarlo de varias maneras con el procesamiento de superficies visibles o incluso efectuarlo después del procesamiento de superficies visibles. Aquí veremos dos clases de algoritmos de sombras. Más adelante, en las secciones 14.7 y 14.8, analizaremos cómo se manejan las sombras en los métodos de iluminación global. En [FOLE90] se estudian otros algoritmos de sombras. Para simplificar las explicaciones supondremos que todos los objetos son polígonos a menos que se especifique lo contrario.

#### 14.4.1 Generación de sombras por línea de barrido

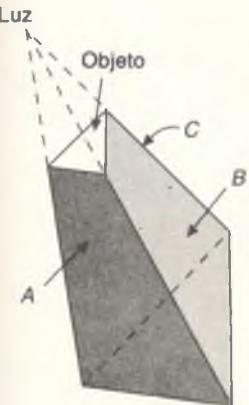
Uno de los métodos más antiguos para generar sombras es aumentar un algoritmo de línea de barrido para combinar los procesamientos de sombras y de superficies visibles [APPE68; BOUK70b]. Usando la fuente luminosa como centro de proyección, las aristas de los polígonos que posiblemente produzcan sombras se proyectan sobre los polígonos que intersecan la línea de barrido actual. Cuando el barrido cruza una de estas aristas de sombra, se modifican en forma correspondiente los colores de los pixeles de la imagen.

Una implantación burda de este algoritmo debe calcular las  $n(n - 1)$  proyecciones de cada polígono sobre los demás polígonos. Bouknight y Kelley [BOUK70b] usan en cambio un astuto paso de preprocessamiento en el cual todos los polígonos se proyectan sobre una esfera que rodea a la fuente luminosa, usando la fuente como centro de proyección. Se pueden eliminar los pares de proyecciones cuyas extensiones no se sobrepongan, y se pueden identificar varios casos especiales más para limitar el número de pares de polígonos que deben ser considerados por el resto del algoritmo. Los autores calculan entonces la proyección de cada polígono desde la fuente luminosa al plano de cada uno de los polígonos que han determinado puede sombrear, como se muestra en la figura 14.27. Cada una de estas proyecciones de polígonos que sombrean tiene información relacionada con el hecho de que el polígono produce o quizás recibe sombras. Mientras el barrido regular del algoritmo de línea de barrido lleva el control de cuáles son las aristas de polígonos regulares que se cruzan, un barrido de sombras paralelo, reparado lleva el control de cuáles son las proyecciones poligonales que causan sombras que cruzan el barrido de sombra y, por consiguiente, *dentro* de cuáles proyecciones de polígonos que causan sombra se encuentra el barrido de sombra en ese momento. Al calcular el matiz de un tramo, estará bajo sombra si el barrido de sombra está *dentro* de una de las proyecciones de sombra dirigidas al plano del polígono. Por lo tanto, el tramo *bc* de la figura 14.27 está bajo sombra, mientras que los tramos *ac* y *cd* no lo están. Observe que el algoritmo no tiene que recortar analíticamente las proyecciones de los polígonos que causan sombra con respecto a los polígonos que se sombrean.



**Figura 14.27** Algoritmo de sombra por línea de barido que usa la estrategia de Bouknight y Kelley. El polígono *A* proyecta la sombra *A'* sobre el plano de *B*.

### 14.4.2 Volúmenes de sombra

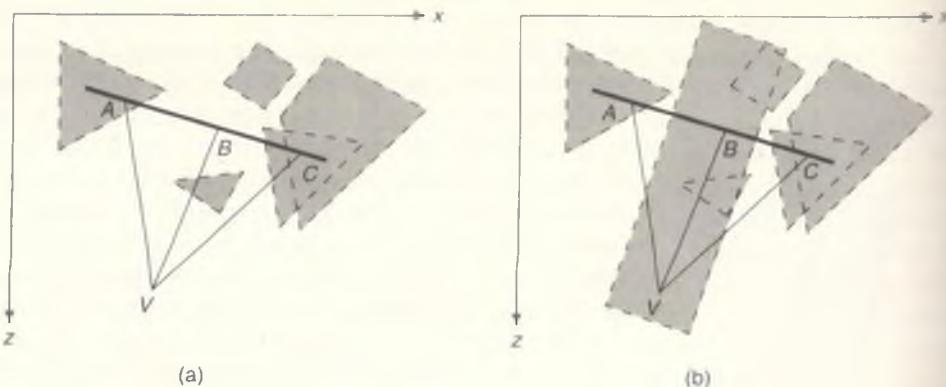


**Figura 14.28**  
Un volumen de sombra es definido por una fuente luminosa y un objeto.

Crow [CROW77] describe cómo generar sombras creando para cada objeto un **volumen de sombra** que bloquea al objeto de la fuente luminosa. Un volumen de sombra se define con la fuente luminosa y un objeto, y está acotado por un conjunto de **polígonos de sombra** invisibles. Como se muestra en la figura 14.28, hay un polígono de sombra cuadrilátero para cada arista de silueta del objeto con respecto a la fuente luminosa. Tres lados de un polígono de sombra se definen a partir de una arista de silueta del objeto y las dos líneas que emanan de la fuente luminosa y pasan por los puntos extremos de la arista. Cada polígono de sombra tiene una normal que apunta fuera del volumen de sombra. Los volúmenes de sombra se generan únicamente para los polígonos de cara a la luz. En la implantación descrita por Bergeron [BERG86a], el volumen de sombra —y por ende cada uno de sus polígonos de sombra— termina en un extremo con el polígono objeto original y en el otro con una copia escalada del polígono objeto cuya normal se ha invertido. Esta copia escalada se coloca a una distancia de la luz detrás de la cual su densidad energética atenuada se supone insignificante. Podemos considerar esta distancia como la **esfera de influencia** de la luz. Cualquier punto fuera de esta esfera de influencia se encuentra de hecho en la sombra y no requiere procesamiento adicional de sombras. Es más, no es necesario generar un volumen de sombra para cualquier objeto que se encuentre totalmente fuera de la esfera de influencia. Este método se puede generalizar para aplicarse a fuentes radientes no uniformes mediante la consideración de una **región de influencia**, por ejemplo eliminando los objetos fuera de las aletas y conos de una luz. El volumen de sombra también se puede recortar con respecto al volumen de vista si éste se conoce por adelantado. El algoritmo también trata los polígonos terminales como polígonos de sombra.

Los polígonos de sombra no se generan por sí mismos, sino que se usan para determinar si otros polígonos están bajo sombra. Con respecto al observador, un polígono de sombra de cara frontal (polígono *A* o *B* en la Fig. 14.28) hace que los objetos detrás de él estén bajo sombra; un polígono de sombra de cara posterior (polígono *C*) cancela el efecto de uno de cara anterior. Considere un vector del punto de observación *V* a un punto en el objeto. El punto está bajo sombra si el vector interseca más polígonos de sombra de cara anterior que de cara posterior. De esta manera, los puntos *A* y *C* de la figura 14.29(a) se encuentran bajo sombra. Éste es el único caso en el cual un punto está bajo sombra cuando *V* no lo está; por lo tanto, el punto *B* está iluminado. Si *V* está bajo sombra, hay un caso más en el cual el punto está bajo sombra: cuando no se han detectado todos los polígonos de sombra de cara posterior de los polígonos de objeto que sombrean al ojo. Así, los puntos *A*, *B* y *C* de la figura 14.29(b) se encuentran bajo sombra, aun cuando el vector de *V* a *B* interseque el mismo número de polígonos de sombra de cara anterior y posterior que en la parte (a).

Podemos calcular si un punto está bajo sombra si asignamos a cada polígono de sombra de cara anterior (con respecto al observador) un valor de +1 y a cada polígono de sombra de cara posterior un valor de -1. Se asigna inicialmen-



**Figura 14.29** Determinación de si un punto está bajo sombra para un observador en  $V$ . Las líneas punteadas definen volúmenes de sombra (en gris). (a)  $V$  no está bajo sombra. Los puntos  $A$  y  $C$  están bajo sombra; el punto  $B$  está iluminado. (b)  $V$  está bajo sombra. Los puntos  $A$ ,  $B$  y  $C$  están bajo sombra.

te un contador con el número de volúmenes de sombra que contiene el ojo y se incrementa con los valores asociados a todos los polígonos de sombra entre el ojo y el punto en el objeto. El punto se hallará bajo sombra si el contador es positivo. El número de volúmenes de sombra que contiene el ojo se calcula una sola vez para cada punto de observación, tomando el negativo de la suma de los valores de todos los polígonos de sombra interceptados por un proyector arbitrario del ojo al infinito.

Las fuentes luminosas múltiples se pueden manejar construyendo un conjunto aparte de volúmenes de sombra para cada fuente luminosa, marcando los polígonos del volumen de sombra con su identificador de fuente luminosa y manteniendo un contador distinto para cada fuente luminosa. Brotman y Badler [BROT84] han implantado una versión de memoria de profundidad  $z$  para el algoritmo de volumen de sombra, mientras que Bergeron [BERG86a] analiza una implantación de línea de barrido que maneja con eficiencia objetos poliedricos arbitrarios que contienen polígonos no planares. Chin y Feiner [CHIN89] describen un algoritmo de precisión de objeto que construye un solo volumen de sombra para un ambiente poligonal, utilizando la representación de árbol BSP para modelado de sólidos que presentamos en la sección 10.6.4.

## 14.5 Transparencia

Así como las superficies pueden tener reflexión especular y difusa, aquellas que transmiten la luz pueden ser transparentes o translúcidas. Usualmente podemos

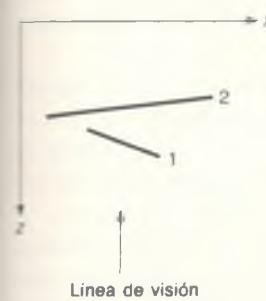
ver con claridad a través de los materiales **transparentes**, como el vidrio, aunque en términos generales los rayos se refractan (doblan). La transmisión difusa se presenta en los materiales **translúcidos**, como el vidrio esmerilado. Los rayos que pasan por los materiales translúcidos son desordenados por irregularidades internas o superficiales, y por ello los objetos se ven borrosos a través de los materiales translúcidos.

#### 14.5.1 Transparencia no refractiva

El método más sencillo para modelar la transparencia ignora la refracción, de manera que los rayos no se doblan al pasar por la superficie. De esta manera, lo que está visible en la línea de visión a través de una superficie transparente también está ubicado geométricamente en la línea de visión. Aunque la transparencia sin refracción no es realista, con frecuencia puede ser un efecto más útil que la refracción. Por ejemplo, puede proporcionar una vista sin distorsión a través de una superficie. Como indicamos antes, el realismo fotográfico total no siempre es el objetivo al formar imágenes.

Por lo general se han usado dos métodos diferentes para aproximar la forma en que se combinan los colores de dos objetos cuando uno de ellos es observado a través del otro. Haremos referencia a estos métodos como transparencia **interpolada** y transparencia **filtrada**.

**Transparencia interpolada.** Considere lo que sucede cuando el polígono transparente 1 se encuentra entre el observador y el polígono opaco 2, como se ilustra en la figura 14.30. La **transparencia interpolada** determina el matiz de un pixel en la intersección de las proyecciones de los dos polígonos interpolando linealmente los matices individuales que se calculan para los dos polígonos:



**Figura 14.30**  
Sección transversal de dos polígonos.

$$I_{\lambda} = (1 - k_{t_1})I_{\lambda_1} + k_{t_1}I_{\lambda_2}. \quad (14.23)$$

El **coeficiente de transmisión**  $k_{t_1}$  mide la **transparencia** del polígono 1 y se halla entre 0 y 1. Cuando  $k_{t_1}$  es 0, el polígono es opaco y no transmite luz; si  $k_{t_1}$  es 1, el polígono es perfectamente transparente y no contribuye en absoluto a la intensidad  $I_{\lambda}$ . El valor  $1 - k_{t_1}$  se denomina **opacidad** del polígono. La transparencia interpolada se puede considerar como el modelado de un polígono que consiste en una fina malla de material opaco a través del cual se pueden ver otros objetos;  $k_{t_1}$  es la fracción de la superficie de la malla a través de la cual se puede ver. Un polígono totalmente transparente procesado de esta manera no tendrá reflexión especular. Para obtener un efecto más realista, podemos interpolar sólo los componentes ambiental y difuso del polígono 1 con el matiz total del polígono 2, para luego añadir el componente especular del polígono 1 [KAY79b].

Otro método, conocido como **transparencia de mosquitero**, implanta literalmente una malla generando sólo algunos de los pixeles relacionados con la proyección del objeto transparente. Los bits de orden bajo de la dirección ( $x, y$ )

de un pixel se usan como índice en una mascarailla de bits de transparencia. El pixel se escribe si el bit indizado es 1; en caso contrario, se suprime y queda visible el polígono más próximo en ese pixel.

**Transparencia filtrada.** La transparencia filtrada trata un polígono como un filtro transparente que deja pasar selectivamente diferentes longitudes de onda; se puede modelar como

$$I_\lambda = I_{\lambda_1} + k_t O_{t\lambda} I_{\lambda_2}, \quad (14.2)$$

donde  $O_{t\lambda}$  es el color de transparencia del polígono 1. Un filtro coloreado se puede modelar eligiendo un valor distinto de  $O_{t\lambda}$  para cada  $\lambda$ . En la transparencia interpolada o filtrada, si hay polígonos adicionales enfrente de estos polígonos, el cálculo se invoca recursivamente para los polígonos en orden posterior-anterior, usando cada vez como  $I_{\lambda_1}$  el valor  $I_\lambda$  previamente calculado.

**Implantación de la transparencia.** Podemos adaptar fácilmente varios algoritmos de superficies visibles para incorporar la transparencia, incluyendo los algoritmos de línea de barrido y de prioridad de listas. En los algoritmos de prioridad de listas, se lee el color de un pixel que será cubierto por un polígono transparente y se usa en el modelo de iluminación al discretizar el polígono.

La mayoría de los sistemas basados en *z-buffer* apoyan el uso de la transparencia de mosquitero porque permite combinar objetos transparentes con objetos opacos y dibujarlos en cualquier orden. Es más difícil añadir al algoritmo de *z-buffer* efectos de transparencia que usen las ecuaciones (14.23) o (14.24), ya que los polígonos se generan en el orden en que se detectan. Imagine la generación de varios polígonos transparentes superpuestos seguidos por uno opaco. Nos gustaría colocar el polígono opaco detrás de los polígonos transparentes apropiados. Por desgracia, el *z-buffer* no almacena la información necesaria para determinar cuáles polígonos transparentes están frente al opaco ni el orden relativo de los polígonos. Una estrategia sencilla, aunque incorrecta, es generar al final los polígonos transparentes, combinando sus colores con los que ya se encuentran en la memoria gráfica pero sin modificar el *z-buffer* sin embargo, no se tiene en cuenta la profundidad relativa cuando se sobreponen dos polígonos transparentes.

Kay y Greenberg [KAY79b] implantaron una aproximación útil al aumento en atenuación que ocurre cerca de la silueta de las superficies curvas delgadas, donde la luz pasa por más material. Ellos definen  $k_t$  en términos de una función no lineal del componente  $z$  de la normal a la superficie después de la transformación de perspectiva,

$$k_t = k_{t_{\min}} + (k_{t_{\max}} - k_{t_{\min}})(1 - (1 - z_N)^m),$$

donde  $k_{t_{\min}}$  y  $k_{t_{\max}}$  son las transparencias mínima y máxima del objeto,  $z_N$  es el componente  $z$  de la normal a la superficie normalizada en el punto para el cual

se calcula  $k_t$ , y  $m$  es un factor de potencia (por lo general 2 o 3). Una  $m$  mayor modela una superficie más delgada. Este nuevo valor de  $k_t$  se puede usar como  $k_t$ , ya sea en la ecuación (14.23) o en la (14.24).

### 14.5.2 Transparencia refractiva

La transparencia refractiva es mucho más difícil de modelar que la no refractiva, ya que las líneas de visión geométrica y óptica son diferentes. Si se considera la refracción en la figura 14.31, el objeto  $A$  es visible a través del objeto transparente en la línea de visión que se muestra; si se ignora la refracción, el objeto  $B$  es visible. La relación entre el ángulo de incidencia  $\theta_i$  y el ángulo de refracción  $\theta_r$  está indicada por la ley de Snell

$$\frac{\sin \theta_i}{\sin \theta_r} = \frac{\eta_{iz}}{\eta_{ix}}, \quad (14.25)$$

donde  $\eta_{ix}$  y  $\eta_{iz}$  son los índices de refracción de los materiales por los cuales pasa la luz. El índice de refracción de un material es la razón entre la velocidad de la luz en el vacío y la velocidad de la luz en el material. Este índice varía de acuerdo con la longitud de onda de la luz e incluso con la temperatura. Un vacío tiene índice de refracción de 1.0, como la atmósfera con una aproximación cercana; todos los materiales tienen valores más altos. La dependencia de la longitud de onda que tiene el índice de refracción se presenta en muchos casos de refracción como dispersión: el familiar, pero difícil de modelar, fenómeno de la luz refractada que se distribuye en su espectro [THOM86; MUSG89].

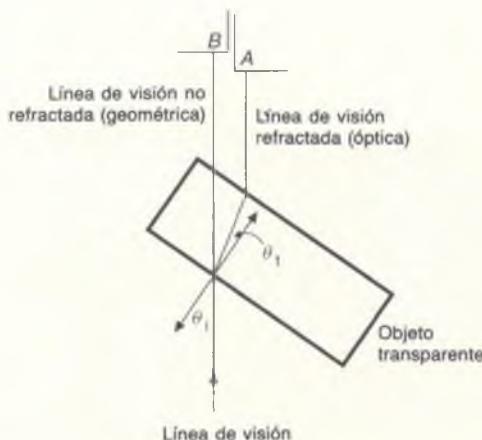


Figura 14.31 Refracción.

**Cálculo del vector de refracción.** El vector unidad en la dirección de la refracción,  $\bar{T}$ , se puede calcular como

$$\bar{T} = \operatorname{sen} \theta_i \bar{M} - \cos \theta_i \bar{N}, \quad (14.26)$$

donde  $\bar{M}$  es un vector unidad perpendicular a  $\bar{N}$  en el plano del rayo incidente  $\bar{I}$  y  $\bar{N}$  [HECK84] (Fig. 14.32). Recordando el uso de  $\bar{S}$  en el cálculo del vector de reflexión  $\bar{R}$  en la sección 14.1.4, vemos que  $\bar{M} = (\bar{N} \cos \theta_i - \bar{I}) / \operatorname{sen} \theta_i$ . Por sustitución,

$$\bar{T} = \frac{\operatorname{sen} \theta_i}{\operatorname{sen} \theta_i} (\bar{N} \cos \theta_i - \bar{I}) - \cos \theta_i \bar{N}. \quad (14.27)$$

Si hacemos que  $\theta_{r\lambda} = \eta_{r\lambda} / \eta_{i\lambda} = \operatorname{sen} \theta_i / \operatorname{sen} \theta_i$ , después de acomodar los términos se obtiene

$$\bar{T} = (\eta_{r\lambda} \cos \theta_i - \cos \theta_i) \bar{N} - \eta_{r\lambda} \bar{I}. \quad (14.28)$$

Observe que  $\cos \theta_i$  es  $\bar{N} \cdot \bar{I}$  y que  $\cos \theta_i$  se puede calcular como

$$\cos \theta_i = \sqrt{1 - \operatorname{sen}^2 \theta_i} = \sqrt{1 - \eta_{r\lambda}^2 \operatorname{sen}^2 \theta_i} = \sqrt{1 - \eta_{r\lambda}^2 (1 - (\bar{N} \cdot \bar{I})^2)}. \quad (14.29)$$

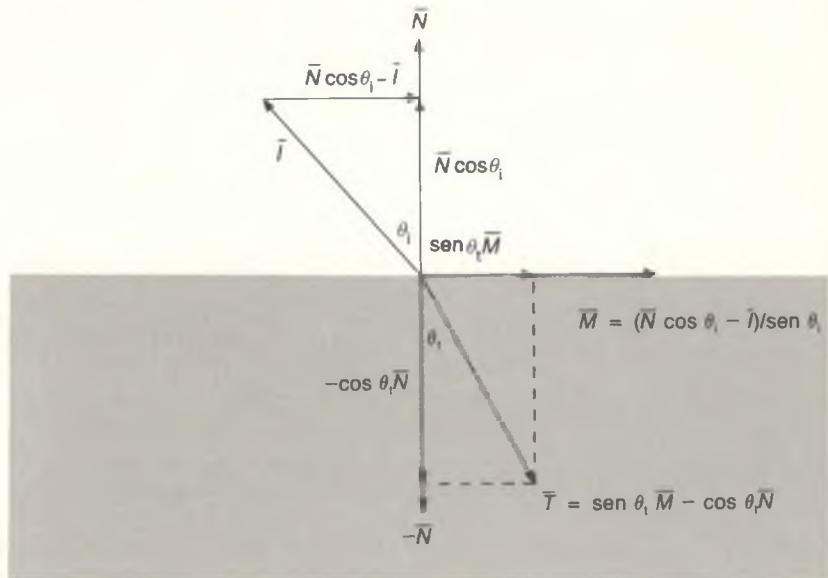


Figura 14.32 Cálculo del vector de refracción.

Por lo tanto,

$$\bar{T} = \left( \eta_{r\lambda}(\bar{N} \cdot \bar{I}) - \sqrt{1 - \eta_{r\lambda}^2(1 - (\bar{N} \cdot \bar{I})^2)} \right) \bar{N} - \eta_{t\lambda} \bar{I}. \quad (14.30)$$

**Reflexión interna total.** Cuando la luz pasa de un medio a otro cuyo índice de refracción es menor, el ángulo  $\theta_t$  del rayo transmitido es mayor que el ángulo  $\theta_i$ . Si  $\theta_t$  aumenta lo suficiente, entonces  $\theta_t$  excede de  $90^\circ$  y el rayo se refleja desde la interfaz entre los medios, en lugar de transmitirse. Este fenómeno se conoce como **reflexión interna total**, y el ángulo  $\theta_c$  más pequeño donde ocurre se denomina **ángulo crítico**. Usted puede observar la reflexión interna total tratando de ver su mano a través de una pecera llena. Cuando el ángulo de observación es mayor que el ángulo crítico, las únicas partes visibles de su mano serán aquellas que estén presionadas firmemente contra la pecera, sin una capa de aire intermedia (cuyo índice de refracción es menor que el del vidrio o del agua). El ángulo crítico es el valor de  $\theta_i$  en el cual  $\sin \theta_t$  es 1. Si se asigna  $\sin \theta_t$  igual a 1 en la ecuación (14.25), podemos ver que el ángulo crítico es  $\sin^{-1}(\eta_o/\eta_d)$ . La reflexión interna total ocurre cuando la raíz cuadrada en la ecuación (14.30) es imaginaria.

En la sección 14.7 se analiza la utilización de la ley de Snell para modelar la transparencia con rastreo de rayos.

## 14.6 Algoritmos de iluminación global

Un modelo de iluminación calcula el color en un punto en función de la luz emitida en forma directa por las fuentes luminosas, y de la luz que llega al punto después de la reflexión y la transmisión en su superficie y en otras. Esta luz transmitida y reflejada en forma indirecta se conoce a menudo como **iluminación global**. En cambio, la **iluminación local** es la luz que viene directamente de las fuentes luminosas al punto que se sombra. Hasta ahora hemos modelado la iluminación global con un término de iluminación ambiental que se ha mantenido constante para todos los puntos en los objetos. No depende de las posiciones del objeto ni del observador, como tampoco de la presencia o ausencia de objetos cercanos que podrían bloquear la luz ambiental. Así mismo, hemos visto algunos efectos limitados de iluminación global que pueden efectuarse con sombras y transparencia.

Una gran parte de la luz en los ambientes de mundo real no proviene de fuentes luminosas directas. Se han usado dos clases de algoritmos para generar imágenes que subrayen las contribuciones de la iluminación global. En la sección 14.7 se analizan las extensiones al algoritmo de traza de rayos para superficies visibles que combinan la determinación de superficies visibles y el sombreado para representar sombras, reflexión y refracción. Así, la transmisión

y la reflexión especular global complementan la iluminación local especular, difusa y ambiental calculada para una superficie. En cambio, los métodos de radiosidad presentados en la sección 14.8 separan totalmente el sombreado y la determinación de superficies visibles, modelando todas las interacciones del ambiente con las fuentes luminosas, primero en una etapa independiente de la vista, para después calcular una o más imágenes para los puntos de observación deseados usando algoritmos convencionales de superficies visibles y sombreado por interpolación.

La diferencia entre los algoritmos dependientes de la vista, como la traza de rayos, y los independientes, como la radiosidad, es muy importante. Los algoritmos dependientes de la vista discretizan el plano de vista para determinar puntos en los cuales se evaluará la ecuación de iluminación, con base en la dirección de observación. Por otra parte, los algoritmos independientes de la vista discretizan el ambiente y lo procesan para proporcionar información suficiente que permita evaluar la ecuación de iluminación en cualquier punto y desde cualquier dirección de observación. Los algoritmos dependientes de la vista son adecuados para manejar fenómenos especulares altamente dependientes de la posición del observador, pero estos algoritmos pueden requerir llevar a cabo trabajo adicional al modelar fenómenos difusos que cambian poco en grandes áreas de una imagen o entre imágenes formadas por distintos puntos de observación. En cambio, los algoritmos independientes de la vista modelan eficiencia fenómenos difusos, pero requieren cantidades exorbitantes de almacenamiento para disponer de información suficiente acerca de los fenómenos especulares.

A final de cuentas, todas estas estrategias tratan de resolver lo que Kajiya [KAIJ86] denominó **ecuación de generación**, que expresa la luz transferida de un punto a otro en función de la intensidad de la luz emitida del primer punto al segundo y la intensidad de la luz emitida desde todos los demás puntos que llega al primero y es reflejada del primero al segundo. La luz transferida de cada uno de estos puntos al primero se expresa, a su vez, recursivamente con la ecuación de generación. Kajiya presenta la ecuación de generación como

$$I(x, x') = g(x, x') \left[ \epsilon(x, x') + \int_S \rho(x, x', x'') I(x', x'') dx'' \right]. \quad (14.3)$$

donde  $x$ ,  $x'$  y  $x''$  son puntos en el ambiente;  $I(x, x')$  se relaciona con la intensidad que pasa de  $x'$  a  $x$ ;  $g(x, x')$  es un término geométrico que es cero cuando  $x$  y  $x'$  están ocultos uno con respecto al otro y  $1/r^2$  cuando son visibles, donde  $r$  es la distancia entre ellos; y  $\epsilon(x, x')$  se relaciona con la intensidad de la luz que se emite de  $x'$  a  $x$ . La evaluación inicial de  $g(x, x')\epsilon(x, x')$  para  $x$  en el punto de observación logra la determinación de superficies visibles en la esfera alrededor de  $x$ . La integral abarca todos los puntos en todas las superficies  $S$ .  $\rho(x, x', x'')$  se relaciona con la intensidad de la luz reflejada (incluyendo la reflexión tanto especular como difusa) de  $x''$  a  $x$  desde la superficie en  $x'$ . Así, la ecuación de generación establece que la luz de  $x'$  que llega a  $x$  consiste en la luz emitida por

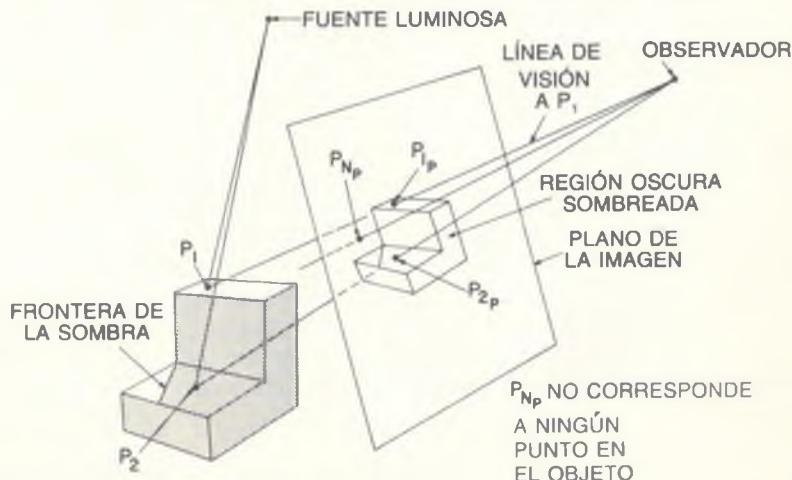
la propia  $x'$  y la luz dispersada por  $x'$  a todas las demás superficies, las cuales emiten luz y dispersan recursivamente luz de otras superficies.

Como veremos, el éxito de una estrategia para resolver la ecuación de generación depende en gran parte de cómo maneje los términos restantes y la recursión, qué combinaciones de reflectividad difusa y especular permita, y cuán bien se modelen las relaciones de visibilidad entre superficies.

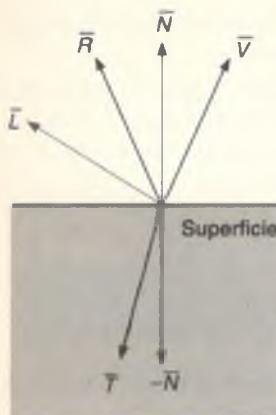
## 14.7 Trazo de rayos recursiva

En esta sección extenderemos el algoritmo básico de trazo de rayos de la sección 13.4 para manejar sombras, reflexión y refracción. Este sencillo algoritmo determina el color de un pixel en la intersección más cercana de un rayo ocular con un objeto, usando cualquiera de los modelos de iluminación previamente descritos. Para calcular sombras, se lanza un rayo adicional del punto de intersección a cada una de las fuentes luminosas. Esto se ilustra en la figura 14.33 para una fuente luminosa única, reproducida de un artículo de Appel [APPE68], el primero publicado acerca de la traza de rayos para la graficación por computador. Si uno de estos **rayos de sombra** interseca un objeto, dicho objeto está bajo sombra en ese punto y el algoritmo de sombras ignora la contribución de la fuente luminosa del rayo de sombra.

El modelo de iluminación desarrollado por Whitted [WHIT80] y Kay [KAY79a] extiende fundamentalmente la traza de rayos para incluir la reflexión especular y la transparencia refractiva. La ilustración en color 41 es una de las primeras imágenes generadas con estos efectos. Además de los rayos de sombra,



**Figura 14.33** Determinación de si un punto en un objeto está bajo sombra. (Cortesía de Arthur Appel, IBM T. J. Watson Research Center.)



**Figura 14.34**  
Los rayos de reflexión, refracción y sombra se generan desde un punto de intersección.

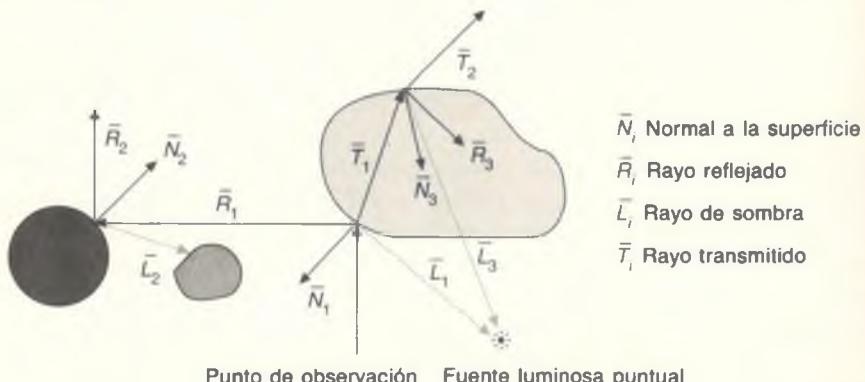
el algoritmo recursivo de traza de rayos de Whitted genera condicionalmente **rayos de reflexión** y **rayos de refracción** desde el punto de intersección, como se muestra en la figura 14.34. Los rayos de sombra, reflexión y refracción también se conocen como **rayos secundarios**, para distinguirlos de los **rayos primarios** que parten del ojo. Si el objeto es especularmente reflejante, un rayo de reflexión se refleja por la normal a la superficie en la dirección de  $\bar{R}$ , que puede calcularse de la manera descrita en la sección 14.1.4. Si el objeto es transparente y no ocurre la reflexión interna total, se envía un rayo de refracción al objeto a lo largo de  $\bar{T}$  con un ángulo determinado por la ley de Snell, descrita en la sección 14.5.2. (Observe que su rayo incidente puede tener una orientación opuesta a las mencionadas en esas secciones).

A su vez, cada uno de estos rayos de reflexión y refracción puede generar recursivamente rayos de sombra, reflexión y refracción, como se presenta en la figura 14.35. Estos rayos forman un **árbol de rayos** como el de la figura 14.36. En el algoritmo de Whitted, una rama termina si los rayos reflejados y refractados no intersecan un objeto, si se ha alcanzado una profundidad máxima establecida por el usuario o si el sistema agota su espacio de almacenamiento. El árbol se evalúa en forma ascendente, y se calcula la intensidad de cada nodo como función de las intensidades de sus hijos.

Podemos representar la ecuación de iluminación de Whitted como

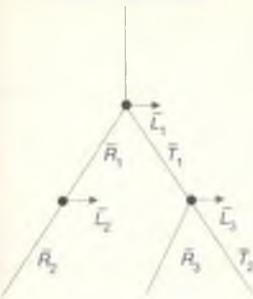
$$I_{\lambda} = I_{a\lambda} k_a O_{d\lambda} + \sum_{1 \leq i \leq m} S_i f_{att, i} p_{\lambda, i} [k_d O_{d\lambda} (\bar{N} \cdot \bar{L}_i) + k_s (\bar{N} \cdot \bar{H}_i)^n] + k_t I_{t\lambda} + k_l I_{l\lambda}, \quad (14.32)$$

donde  $I_{r\lambda}$  es la intensidad del rayo reflejado,  $k_t$  es el **coeficiente de transmisión** con valor entre 0 y 1, e  $I_{l\lambda}$  es la intensidad del rayo transmitido refractado. Los valores de  $I_{r\lambda}$  e  $I_{l\lambda}$  se determinan evaluando recursivamente la ecuación (14.32) en la superficie más cercana que intersecan los rayos reflejados y transmitidos. Para aproximar la atenuación con la distancia, Whitted multiplicó la  $I_{\lambda}$  calculada para cada rayo por el inverso de la distancia que ha viajado el rayo. En lugar



**Figura 14.35** Rayos que generan recursivamente otros rayos.

Punto de observación

**Figura 14.36**

Árbol de rayos para la figura 14.35.

de tratar  $S_i$  como una función de incremento, como en la ecuación (14.22), también la hizo una función continua de la  $k_i$  de los objetos intersecados por el rayo de sombra, de manera que un objeto transparente oscurece menos luz que uno opaco en aquellos puntos que sombrean.

El programa 14.1 presenta el seudocódigo para un sencillo trazador de rayos recursivo. *Traza\_RR* determina la intersección más cercana entre el rayo y un objeto, y llama a *sombra\_RR* para determinar la sombra en dicho punto. En primer lugar, *sombra\_RR* determina el color ambiental de la intersección. Después se genera un rayo de sombra para cada luz en el lado de la superficie que se sombrean a fin de determinar su contribución al color. Un objeto opaco bloquea totalmente la luz, mientras que uno transparente escala la contribución luminosa. Si no estamos en un nivel muy profundo del árbol de rayos, se efectúan llamadas recursivas a *traza\_RR* para manejar los rayos de reflexión de los objetos reflejantes y los rayos de refracción de los objetos transparentes. Como se requieren los índices de refracción de los dos medios para determinar la dirección del rayo de refracción, con cada rayo se puede incluir el índice de refracción del material por el cual viaja el rayo. *Traza\_RR* conserva el árbol de rayos únicamente el tiempo suficiente para determinar el color del pixel actual. Si es posible mantener los árboles de rayos para toda la imagen, se pueden alterar las propiedades superficiales y volver a calcular con relativa rapidez una nueva imagen, con sólo evaluar de nuevo los árboles. Sequin y Smyrl [SEQU89] presentan técnicas que minimizan el tiempo y el espacio necesarios para procesar y almacenar árboles de rayos.

**Programa 14.1**  
Seudocódigo para una sencilla traza de rayos recursiva sin eliminación de artefactos de discretización.

```

seleccionar el centro de proyección y la ventana en el plano de vista;
for (cada línea de barrido en la imagen) {
    for (cada pixel en la línea de barrido) {
        determinar rayo del centro de proyección a través del pixel;
        pixel = traza_RR (rayo, 1);
    }
}

/* Intersecar rayo con los objetos y calcular la sombra en la intersección más cercana. */
/* La profundidad es la profundidad actual en el árbol de rayos. */

color_RR traza_RR (rayo_RR, int profundidad)
{
    determinar la intersección más cercana de rayo con un objeto;
    if (objeto intersecado) {
        calcular normal en la intersección;
        return sombra_RR (objeto intersecado más cercano, rayo, intersección, normal,
profundidad);
    }
    else
        return VALOR_FONDO;
}
/* Calcular sombra en el punto del objeto, trazando rayos para sombras, reflexión y refracción. */

```

```

color__RR sombra__RR (
    objeto__RR objeto,
    rayo__RR rayo,
    punto__RR punto,
    normal__RR normal,
    int profundidad)
{
    color__RR color; /* Color del rayo */
    rayo__RR rayo_r, rayo_t, rayo_s;
    color__RR color_r, color_t; /* Rayos reflejado, refractado y de sombra */

    color = término de ambiente;
    for(cada luz){
        rayo_s = rayo desde punto a la luz;
        if(el producto punto de normal y la dirección a la luz es positivo) {
            calcular cuánta luz es bloqueada por superficie opacas y transparentes, y usarla
            para escalar los términos difusos y especulares antes de añadirlos a color;
        }
    }
    if(profundidad < profundidad_máx){ /* Regresar si la profundidad es excesiva */
        if(objeto es reflejante){
            rayo_r = rayo en la dirección de reflexión desde punto;
            color_r = rastreo__RR (rayo_r, profundidad + 1);
            escalar color_r por el coeficiente especular y añadir a color;
        }
        if(objeto es transparente){
            rayo_t = rayo en la dirección de refracción desde punto;
            if(no ocurre la reflexión interna total){
                color_t = rastreo__RR (rayo_t, profundidad + 1);
                escalar color_t por el coeficiente de transmisión y añadir a color;
            }
        }
    }
    return color; /* Devolver color del rayo. */
}

```

En la figura 14.35 se presenta un problema básico con la forma en que la traza de rayos modela la refracción: el rayo de sombra  $\bar{L}_3$ , no se refracta en su trayectoria hacia la luz. De hecho, si simplemente refractáramos  $\bar{L}_3$  de su dirección actual al punto donde sale del objeto grande, no terminaría en la fuente luminosa. Además, al determinar las trayectorias de los rayos refractados se usa un sólo índice de refracción para cada rayo.

La traza de rayos es muy susceptible de presentar problemas ocasionados por la limitada precisión numérica. Estos problemas se presentan al calcular objetos que intersecan rayos secundarios. Después de haber calculado las coordenadas  $x$ ,  $y$  y  $z$  del punto de intersección de un objeto visible a un rayo ocular, éstas se usan para definir el punto de partida del rayo secundario cuyo parámetro  $t$  debemos determinar (Sec. 13.4.1). Si el objeto que acabamos de intersecar

interseca el nuevo rayo, con frecuencia tendrá una  $t$  muy pequeña, distinta de cero, debido a las limitaciones de la precisión numérica. Si no consideramos esta situación, esta falsa intersección puede ocasionar problemas visuales. Por ejemplo, si el rayo fuera de sombra, se consideraría que el objeto bloquea la luz que recibe, produciendo manchas de superficie incorrectamente “autosombreada”. Una forma sencilla de resolver este problema de los rayos de sombra es tratar como caso especial el objeto del cual se genera un rayo secundario, para que las pruebas de intersección no se apliquen a él. Es obvio que, esto no funcionará si se utilizan objetos que en realidad podrían oscurecerse a sí mismos o si los rayos transmitidos tienen que pasar por el objeto y reflejarse en su interior. Una solución más general es calcular  $\text{abs}(t)$  para la intersección, comparar el resultado con un pequeño valor de tolerancia e ignorarlo si está por debajo de la tolerancia.

El artículo que Whitted presentó en *SIGGRAPH '79* [WHIT80] y los filmes que creó usando el algoritmo allí descrito iniciaron un renacimiento del interés por la traza de rayos. La traza de rayos recursiva permite producir una pléthora de efectos impresionantes, como sombras, reflexión especular y transparencia refractiva, que eran difíciles o imposibles de obtener antes. Además, es bastante fácil implantar un trazador de rayos sencillo. Por consiguiente, se ha dirigido mucho esfuerzo a mejorar tanto la eficiencia del algoritmo como la calidad de las imágenes. Si desea conocer más detalles sobre el tema, consulte la sección 16.12 de [FOLE90] y [GLAS89].

## 14.8 Métodos de radiosidad

Aunque la traza de rayos lleva a cabo un trabajo excelente en el modelado de la reflexión especular y la transparencia refractiva sin dispersión, sigue usando un término de iluminación ambiental no direccional para considerar todas las otras contribuciones luminosas. Los métodos basados en los modelos de ingeniería térmica para la emisión y la reflexión de la radiación eliminan la necesidad de usar el término de iluminación ambiental al proporcionar un tratamiento más preciso de la reflexión entre objetos. Estos algoritmos, presentados originalmente por Goral, Torrance, Greenberg y Battaile [GORA84] y por Nishita y Nakamae [NISH85], suponen la conservación de la energía luminosa en un ambiente cerrado. Toda la energía emitida o reflejada por cada una de las superficies es considerada por su reflexión o absorción en las demás. La tasa con la cual la energía parte de una superficie, conocida como **radiosidad**, es la suma de las tasas de emisión de energía de la superficie y de la reflexión o transmisión de dicha energía en esa u otras superficies. Por lo tanto, los métodos que calculan las radiosidades de las superficies en un ambiente se han denominado **métodos de radiosidad**. A diferencia de los algoritmos convencionales para la generación

de imágenes, los métodos de radiosidad primero determinan todas las interacciones luminosas en un ambiente en forma independiente de la vista. Después se generan una o más vistas, con sólo el tiempo de procesamiento adicional necesario para determinar superficies visibles y el sombreado por interpolación.

#### 14.8.1 Ecuación de radiosidad

En los algoritmos de sombreado que consideramos previamente, las fuentes luminosas siempre se han tratado en forma separada de las superficies que iluminan. En cambio, los métodos de radiosidad permiten que cualquier superficie emita luz; por ende, todas las fuentes luminosas se modelan de manera inherente como áreas. Imagine dividir el ambiente en un número finito  $n$  de parches discretos, cada uno de los cuales se supone con tamaño finito, que emiten y reflejan luz uniformemente en toda su área. Si consideramos los parches como emisores y reflectores difusos lambertianos, entonces, para la superficie  $i$ ,

$$B_i = E_i + \rho_i \sum_{1 \leq j \leq n} B_j F_{j-i} \frac{A_j}{A_i}. \quad (14.33)$$

$B_i$  y  $B_j$  son las radiosidades de los parches  $i$  y  $j$ , medidos en energía/unidad de tiempo/unidad de área (es decir,  $\text{W/m}^2$ ).  $E_i$  es la tasa con la cual se emite luz del parche  $i$  y tiene las mismas unidades que la radiosidad.  $\rho_i$  es la reflectividad del parche  $i$  y no tiene dimensión.  $F_{j-i}$  es el **factor de forma** o **factor de configuración**, sin dimensiones, que especifica la fracción de la energía que parte de todo el parche  $j$  y que llega a todo el parche  $i$ , considerando la forma y la orientación relativa de ambos parches, así como la presencia de cualquier parche obstructor.  $A_i$  y  $A_j$  son las áreas de los parches  $i$  y  $j$ .

La ecuación (14.33) establece que la energía que parte de un área unidad de superficie es la suma de la luz emitida más la luz reflejada. La luz reflejada calcula escalando la suma de la luz incidente por la reflectividad. A su vez, la luz incidente es la suma de la luz que parte de toda el área de cada parche en el ambiente, escalada por la fracción de la luz que llega a un área unidad del parche receptor.  $B_j F_{j-i}$  es la cantidad de luz que parte de un área unidad de  $A_j$  y que llega a toda el área  $A_i$ . Por lo tanto, es necesario multiplicar por la razón de áreas  $A_j/A_i$  para determinar la luz que parte de toda el área  $A_j$  y que llega a una área unidad de  $A_i$ .

Por fortuna, hay una relación de reciprocidad sencilla entre los factores de forma en los ambientes difusos:

$$A_j F_{j-i} = A_i F_{i-j}. \quad (14.34)$$

Por lo tanto, podemos simplificar la ecuación (14.33) para obtener

$$B_i = E_i + \rho_i \sum_{1 \leq j \leq n} B_j F_{i-j}. \quad (14.35)$$

Reordenando los términos,

$$B_i - \rho_i \sum_{1 \leq j \leq n} B_j F_{i-j} = E_i . \quad (14.36)$$

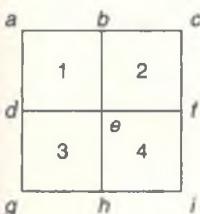
Por lo tanto, la interacción de la luz entre parches en el ambiente se puede especificar como un conjunto de ecuaciones simultáneas:

$$\begin{bmatrix} 1 - \rho_1 F_{1-1} & -\rho_1 F_{1-2} & \cdots & -\rho_1 F_{1-n} \\ -\rho_2 F_{2-1} & 1 - \rho_2 F_{2-2} & \cdots & -\rho_2 F_{2-n} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ -\rho_n F_{n-1} & -\rho_n F_{n-2} & \cdots & 1 - \rho_n F_{n-n} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ \vdots \\ E_n \end{bmatrix} . \quad (14.37)$$

Observe que es necesario considerar la contribución de un parche a su propia energía reflejada (p. ej., puede ser cóncavo); por consiguiente, de manera general, cada término en la diagonal no es simplemente 1. Hay que resolver la ecuación (14.37) para cada banda de longitudes de onda consideradas en el modelo de iluminación, ya que  $\rho_i$  y  $E_i$  dependen de la longitud de onda. Sin embargo, los factores de forma son independientes de la longitud de onda y constituyen únicamente una función de la geometría, por lo cual no hay que recalcularlos si cambia la iluminación o la reflectividad de la superficie.

La ecuación (14.37) se puede resolver con la iteración de Gauss-Seidel [PRES88], generando una radiosidad para cada parche. Después se pueden generar los parches desde cualquier punto de observación que se desee, usando un algoritmo convencional de superficies visibles; el conjunto de radiosidades calculado para las bandas de longitud de onda de cada parche corresponde a las intensidades del parche. En lugar de usar el sombreado facetado, podemos calcular las radiosidades de vértices a partir de las radiosidades de los parches, permitiendo así el sombreado por interpolación de intensidades.

Cohen y Greenberg [COHE85] proponen el siguiente método para determinar las radiosidades de los vértices. Si un vértice es interior a una superficie, se le asigna el promedio de las radiosidades de los parches que lo comparten. Si está en una arista, entonces se encuentra el vértice interior  $v$  más cercano. Al promediar la radiosidad de un vértice de arista con  $B_v$ , el resultado debe ser el promedio de las radiosidades de los parches que comparten el vértice de arista. Considere los parches en la figura 14.37. La radiosidad de un vértice interior  $e$  es  $B_e = (B_1 + B_2 + B_3 + B_4)/4$ . La radiosidad de un vértice de arista  $b$  se calcula hallando su vértice interior más cercano,  $e$ , y observando que  $b$  es compartido por los parches 1 y 2. Entonces, para determinar  $B_b$  usamos la definición anterior:  $(B_b + B_e)/2 = (B_1 + B_2)/2$ . Al despejar  $B_b$  se obtiene  $B_b = B_1 + B_2 - B_e$ . El vértice interior más cercano a  $a$  también es  $e$  y  $a$  sólo forma parte del



**Figura 14.37**  
Cálculo de radiosidades de vértices a partir de radiosidades de parches.

parche 1. Por lo tanto, como  $(B_a + B_e)/2 = B_1$ , se obtiene  $B_a = 2B_1 - B_e$ . Las radiosidades de otros vértices se calculan en forma similar.

El primer método de radiosidad fue implantado por Goral *et al.* [GOR-RA84], quienes usaron integrales de contorno para calcular factores de forma exactos de ambientes convexos con superficies no ocluidas, como se muestra en la ilustración en color 43. Observe los efectos correctos de *contaminación de colores* ocasionados por la reflexión difusa entre superficies adyacentes, visibles tanto en el modelo como en la imagen generada. Las superficies difusas están coloreadas ligeramente por los colores de las otras superficies difusas que reflejan. Sin embargo, para que los métodos de radiosidad fueran prácticos, primero tuvieron que desarrollarse métodos para calcular los factores de forma entre superficies ocluidas.

#### 14.8.2 Cálculo de factores de forma

Cohen y Greenberg [COHE85] adaptaron un algoritmo de precisión de imágenes para superficies visibles a fin de aproximar con eficiencia los factores de forma para superficies ocluidas. Considere los dos parches presentados en la figura 14.38. El factor de forma de un área diferencial  $dA_i$  a un área diferencial  $dA_j$  es

$$dF_{di-dj} = \frac{\cos \theta_i \cos \theta_j}{\pi r^2} H_{ij} dA_j. \quad (14.38)$$

Para el rayo entre las áreas diferenciales  $dA_i$  y  $dA_j$  de la figura 14.38,  $\theta_i$  es el ángulo que forma el rayo con la normal a  $A_i$ ,  $\theta_j$  es el ángulo que forma con la normal a  $A_j$  y  $r$  es la longitud del rayo.  $H_{ij}$  es 1 o 0, dependiendo de si  $dA_j$  es visible desde  $dA_i$ . Para determinar  $F_{di-j}$ , el factor de forma del área diferencial  $dA_i$  al área finita  $A_j$ , tenemos que integrar en el área del parche  $j$ . De esta manera,

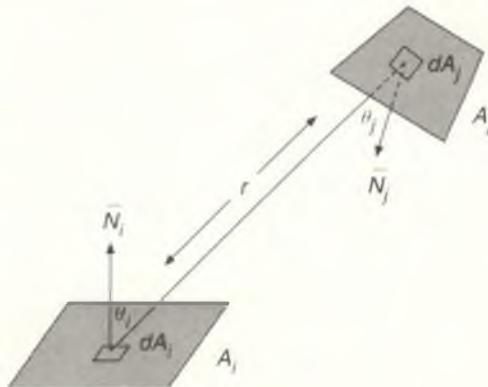


Figura 14.38 Cálculo del factor de forma entre un parche y un área diferencial.

$$F_{di-j} = \int_{A_j} \frac{\cos \theta_i \cos \theta_j}{\pi r^2} H_{ij} dA_j . \quad (14.39)$$

Finalmente, el factor de forma de  $A_i$  a  $A_j$  es el área media de la ecuación (14.39) sobre el parche  $i$ :

$$F_{i-j} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos \theta_i \cos \theta_j}{\pi r^2} H_{ij} dA_j dA_i . \quad (14.40)$$

Si suponemos que el punto central en un parche tipifica los demás puntos del parche, entonces  $F_{i-j}$  se puede aproximar con  $F_{di-j}$  calculado para  $dA_i$  en el centro del parche  $i$ .

Nusselt ha mostrado [SIEG81] que el cálculo de  $F_{di-j}$  equivale a proyectar las partes de  $A_j$  visibles desde  $dA_i$  sobre un hemisferio unidad centrado con respecto a  $dA_i$ , para luego proyectar esta área ortográficamente sobre la base circular unidad del hemisferio y dividir entre el área del círculo (Fig. 14.39). La proyección sobre el hemisferio unidad considera el término  $\cos \theta_j / r^2$  en la ecuación (14.39), la proyección sobre la base corresponde a una multiplicación por

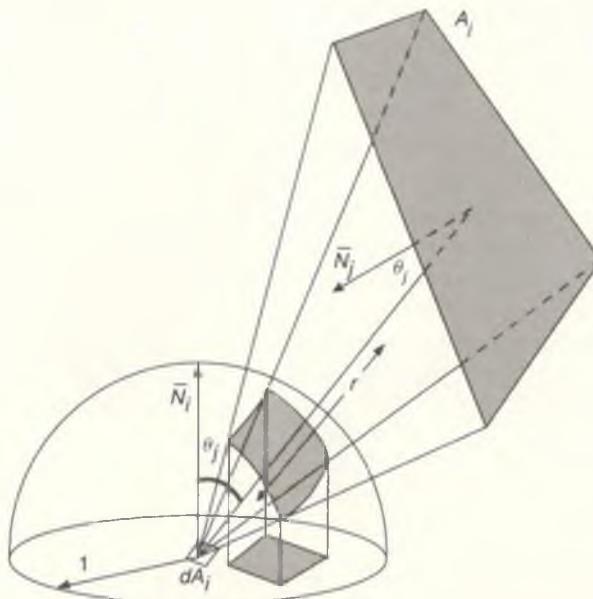


Figura 14.39

Determinación del factor de forma entre un área diferencial y un parche usando el método de Nusselt. El factor de forma es la razón entre el área proyectada sobre la base del hemisferio y el área total de la base. (Basado en [SIEG81].)

$\cos \theta_i$ , y la división entre el área del círculo unidad considera el término  $\pi$  en el denominador.

En lugar de proyectar analíticamente cada  $A_i$  sobre un hemisferio, Cohen y Greenberg desarrollaron un eficaz algoritmo de precisión de imagen que proyecta sobre la mitad superior de un cubo centrado con respecto a  $dA_i$ , con la parte superior del cubo paralela a la superficie (Fig. 14.40). Cada cara de este hemicubo se divide en varias celdas cuadradas de igual tamaño (las resoluciones utilizadas en este libro varían entre  $50 \times 50$  y varios cientos en una cara). Los demás parches se recortan con respecto al tronco del volumen de vista definido por el centro del cubo y cada una de sus cinco caras superiores; después, cada uno de los parches recortados se proyecta sobre la cara apropiada del hemicubo. Se usa un algoritmo de *buffer de elementos* [WEGH84] que registra la identidad del parche intersecante más cercano a cada celda. Estos buffers de elementos se pueden calcular aplicando el algoritmo de *z-buffer* a cada lado del hemicubo, y registrando en cada celda el identificador del parche más cercano y no el matiz. Cada una de las celdas del hemicubo está relacionada con un factor de forma delta basado en su posición. Para un parche  $j$ , podemos aproximar  $F_{di-j}$  sumando los valores de los factores de forma delta relacionados con todas las celdas del hemicubo que contengan el identificador del parche  $j$ . Como gran parte de los cálculos que se realizan con el hemicubo comprende el cálculo de memorias de elementos, es posible aprovechar el hardware existente para la memoria de profundidad  $z$ . Por otra parte, el hemicubo es susceptible a artefactos de discretización, ya que usa operaciones de precisión de imagen.

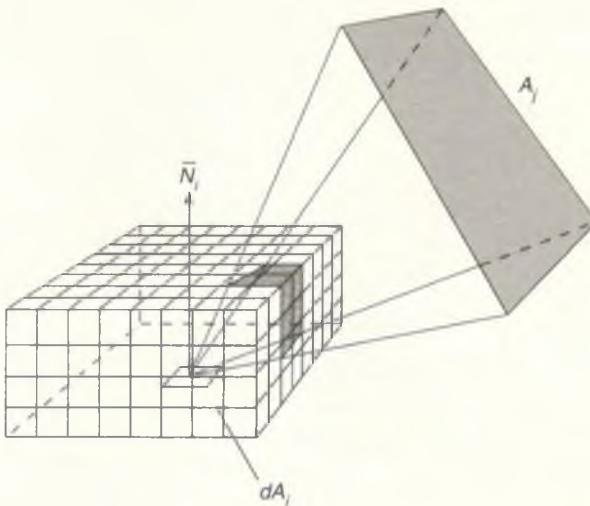


Figura 14.40 El hemicubo es la mitad superior de un cubo centrado con respecto al parche. (Basado en [COHE85].)

### 14.8.3 Refinamiento progresivo

Debido a los elevados costos que requiere la ejecución del algoritmo de radiosidad que hemos descrito hasta ahora, tiene sentido preguntarnos si es posible aproximar los resultados del algoritmo en forma incremental. ¿Podemos producir una imagen útil, aunque quizás inexacta, que pueda refinarse sucesivamente para obtener mayor precisión si se asigna más tiempo? El método de radiosidad descrito en las secciones anteriores no nos permite hacerlo, por dos razones. En primer lugar, la iteración de Gauss-Seidel debe llevarse a cabo antes de disponer de un cálculo de las radiosidades de los parches. Segundo, los factores de forma se calculan entre todos los parches al iniciar y deben almacenarse durante todos los cálculos, algo que requiere  $O(n^2)$  tiempo y espacio. Cohen, Chen, Wallace y Greenberg [COHE88] han desarrollado un algoritmo de radiosidad por refinamiento progresivo que trata estas dos cuestiones.

Consideré el método que se ha descrito hasta ahora. Al evaluar la  $i$ -ésima fila de la ecuación (14.37) se obtiene un cálculo de la radiosidad del parche  $i$ ,  $B_i$ , expresada en la ecuación (14.35), que se basa en los cálculos de las radiosidades de los otros parches. Cada término de la sumatoria en la ecuación (14.35) representa el efecto del parche  $j$  sobre la radiosidad del parche  $i$ :

$$B_i \text{ debida a } B_j = \rho_j B_j F_{i-j}, \quad \text{para toda } j. \quad (14.41)$$

Así, este método *colecta* la luz del resto del ambiente. En cambio, la estrategia de refinamiento progresivo *dispara* al ambiente la radiosidad de un parche. Una forma directa de hacerlo es modificando la ecuación (14.41) para obtener

$$B_j \text{ debida a } B_i = \rho_i B_i F_{j-i}, \quad \text{para toda } j. \quad (14.42)$$

Con base en un cálculo de  $B_i$ , podemos determinar la contribución del parche  $j$  al resto del ambiente evaluando la ecuación (14.42) para cada parche  $j$ . Por desgracia, para esto hay que conocer  $F_{j-i}$  para cada  $j$ , un valor que se determina con un hemicubo diferente. Esto impone considerables requisitos adicionales de tiempo y espacio, al igual que el método original. Sin embargo, podemos usar la relación de reciprocidad de la ecuación (14.34) para reescribir la ecuación (14.42) como

$$B_j \text{ debida a } B_i = \rho_j B_j F_{i-j} \frac{A_i}{A_j}, \quad \text{para toda } j. \quad (14.43)$$

La evaluación de esta ecuación para cada  $j$  sólo requiere los factores de forma calculados usando un hemicubo centrado con respecto al parche  $i$ . Si es posible calcular rápidamente los factores de forma del parche  $i$  (p. ej., usando hardware de memoria de profundidad  $z$ ), se pueden descartar en cuanto se hayan calculado las radiosidades disparadas del parche  $i$ . Así, sólo es necesario calcular y almacenar un hemicubo y sus factores de forma en un instante determinado.

Tan pronto como se ha disparado la radiosidad de un parche, se selecciona otro. Se puede seleccionar un parche para que dispare de nuevo después de haberle disparado luz de otros parches. Por lo tanto, no se dispara la radiosidad total estimada del parche  $i$ , sino  $\Delta B_i$ , la cantidad de radiosidad que ha recibido el parche  $i$  desde la última vez que disparó. El algoritmo itera hasta alcanzar la tolerancia deseada. En lugar de elegir parches en orden aleatorio, tiene sentido seleccionar el parche que tendrá mayor impacto. Éste es el parche que tiene más energía para radiar. Como la radiosidad se mide por área unidad, se escoge un parche  $i$  que tenga mayor  $\Delta B_i A_i$ . Inicialmente,  $B_i = \Delta B_i = E_i$  para todos los parches, lo cual sólo será distinto de cero en el caso de fuentes luminosas. El seudocódigo para una sola iteración se presenta en el programa 14.2.

**Programa 14.2**  
Seudocódigo para  
disparar radiosidad desde  
un parche.

```

seleccionar parche  $i$ ;
calcular  $F_{i-j}$  para cada parche  $j$ ;
for (cada parche) {
    Δ Radiosidad =  $\rho_j \Delta B_i F_{i-j} A_i / A_j$ ;
     $\Delta B_i + = \Delta$  Radiosidad;
     $B_i + = \Delta$  Radiosidad;
}
 $\Delta B_i = 0$ ;
```

Cada ejecución del seudocódigo del programa 14.2 ocasionará que otro parche dispare al ambiente la radiosidad que no ha disparado aún. De esta manera, las únicas superficies que se iluminan después de la primera ejecución son aquellas que constituyen fuentes luminosas y las que están iluminadas directamente por el primer parche cuya radiosidad se disparó. Si se genera una imagen nueva al final de cada ejecución, la primera estará relativamente oscura y las siguientes serán progresivamente más brillantes. Podemos añadir un término de ambiente a las radiosidades para que las primeras imágenes sean más útiles. Con cada pasada adicional por el lazo se reduce el término de ambiente, hasta que desaparezca. La ilustración en color 44, generada usando un término de ambiente, muestra las etapas en la creación de una imagen después de 1, 2, 24 y 100 iteraciones.

## 14.9 Ducto de generación (*rendering*)

Ahora que hemos visto diferentes maneras de llevar a cabo la determinación de superficies visibles, la iluminación y el sombreado, repasaremos la forma en que estos procesos encajan en el ducto estándar de graficación presentado en el ca-

pítulo 7. Para simplificar supondremos ambientes poligonales, a menos que se especifique lo contrario. En el capítulo 18 de [FOLE90] se proporciona un análisis más detallado de cómo se pueden implantar estos ductos en hardware.

### 14.9.1 Ductos de iluminación local

**Z-buffer y sombreado de Gouraud.** Quizá la modificación más directa al ducto se presenta en un sistema que utiliza el algoritmo de *z-buffer* para superficies visibles en la generación de polígonos con sombreado de Gouraud, como se muestra en la figura 14.41. El algoritmo de *z-buffer* tiene la ventaja de que las primitivas se le pueden presentar en cualquier orden. Por lo tanto, las primitivas se obtienen recorriendo la base de datos, como antes, y transformándolas al sistema de coordenadas mundiales mediante la transformación de modelado.

Las primitivas pueden estar asociadas con normales a superficies especificadas al construir el modelo. Como el paso de iluminación requerirá la utilización de las normales a las superficie, es importante recordar que estas normales se deben transformar en forma correcta. Así mismo, no podemos ignorar las normales almacenadas y tratar de recalcular posteriormente las nuevas usando los vértices correctamente transformados. Las normales definidas con los objetos pueden representar la verdadera geometría de la superficie o bien especificar efectos de mezclado de superficie definidos por los usuarios, en lugar de ser simplemente promedios de las normales de las caras compartidas en la aproximación de malla poligonal.

Nuestro siguiente paso es eliminar las primitivas que caen totalmente fuera de la ventana y efectuar la eliminación de caras posteriores. Esta fase de rechazo trivial suele llevarse a cabo en este momento porque queremos eliminar el procesamiento innecesario en el paso de iluminación que sigue. Ahora, como estamos usando el sombreado de Gouraud, la ecuación de iluminación se evalúa en cada vértice. Esta operación debe efectuarse en el sistema de coordenadas de mundo (o en cualquier sistema de coordenadas isométrico) antes de la transformación de vista (que puede incluir transformaciones de sesgo y de perspectiva), para conservar el ángulo y la distancia correctos de cada luz a las superficies. Si



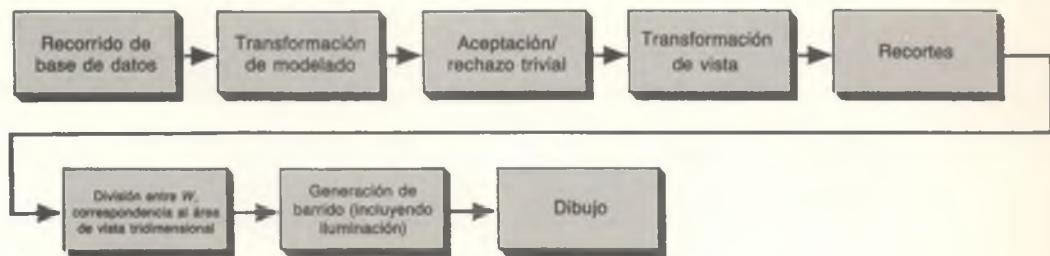
Figura 14.41 Ducto de generación para *z-buffer* y el sombreado de Gouraud.

no se proporcionan normales a los vértices con los objetos, se pueden calcular inmediatamente antes de iluminar los vértices. La eliminación y la iluminación muchas veces se llevan a cabo en un sistema de coordenadas de iluminación que es una transformación de cuerpo rígido de las coordenadas mundiales (p. ej., VRC cuando la matriz de orientación de vista se crea con las utilerías estándar de PHIGS).

El siguiente paso es transformar los objetos a NPC usando la transformación de visualización y recortarlos con respecto al volumen de vista. Se lleva a cabo la división entre  $W$  y se establece la correspondencia entre los objetos y el área de vista. Si un objeto se recorta parcialmente, hay que calcular los valores de intensidad correctos para los vértices creados durante los recortes. En este punto, la primitiva recortada se somete al algoritmo *z-buffer* que lleva a cabo la generación de barrido, combinando la discretización con la interpolación necesaria para calcular el valor  $z$  y los valores de intensidad de color necesarios para cada pixel. Si se determina que un pixel será visible, es posible modificar adicionalmente sus valores de intensidad y color de acuerdo con la indicación de profundidad (Ec. 14.11), que no se presenta aquí.

Aunque este ducto puede parecer bastante directo, hay muchos asuntos nuevos que deben tratarse para proporcionar una implantación eficiente y correcta. Por ejemplo, considere los problemas que se presentan al manejar superficies curvas, como parches de *B-spline*, que deben teselarse. El teselado debe ocurrir después de la transformación a un sistema de coordenadas donde pueda determinarse el tamaño de la pantalla. Esto permite determinar de manera adaptable el tamaño del teselado y limita la cantidad de datos que se transforman. Por otra parte, las primitivas teseladas deben iluminarse en un sistema de coordenadas isométrico a las coordenadas mundiales. Abi-Ezzi [ABIE89] estudia estos temas y propone una formulación del ducto más eficiente, aunque más compleja, que incorpora ciclos de realimentación. Este nuevo ducto utiliza un sistema de coordenadas de iluminación que es una transformación isométrica (es decir, rígida o euclídea) de las coordenadas de mundo, pero con una proximidad computacional a DC que permite tomar eficientemente las decisiones acerca del teselado.

**Z-buffer y sombreado de Phong.** Es necesario modificar este sencillo ducto para el sombreado de Phong, como se muestra en la figura 14.42. Como el sombreado de Phong interpola normales a las superficies en lugar de intensidades, los vértices no pueden iluminarse en las primeras etapas del ducto. En cambio, hay que recortar cada objeto (con las normales correctamente interpoladas que se han creado para cada vértice nuevo), transformarlo con la transformación de visualización y luego pasarlo al algoritmo de *z-buffer*. Finalmente, se lleva a cabo la iluminación con las normales a superficies interpoladas que se obtienen durante la discretización. De esta manera, para evaluar la ecuación de iluminación hay que establecer un mapeo invertido de cada punto y su normal a un sistema de coordenadas que sea isométrico a las coordenadas mundiales.



**Figura 14.42** Ducto de generación para memoria de profundidad z y sombreado de Phong.

**Algoritmo de prioridad de listas y sombreado de Phong.** Al utilizar un algoritmo de prioridad de listas, las primitivas obtenidas del recorrido y procesadas por la transformación de modelado se insertan en una base de datos separada, por ejemplo un árbol BSP, como parte de la determinación preliminar de superficies visibles. En la figura 14.43 se presenta el ducto para el algoritmo de árbol BSP, cuya determinación preliminar de superficies visibles es independiente de la vista. Como señalamos en el capítulo 7, el programa de aplicación y el paquete gráfico pueden mantener bases de datos separadas. Aquí podemos ver que la generación puede requerir incluso otra base de datos. Como en este caso los polígonos se dividen, hay que determinar la información de sombreado correcta para los vértices nuevos que se crean. Ahora es posible recorrer la base de datos de generación para devolver las primitivas en el orden correcto, de atrás hacia adelante. Por supuesto, el procesamiento adicional que se requiere para construir esta base de datos se puede aplicar a la creación de imágenes múltiples. Por lo tanto, lo hemos presentado como un ducto separado cuya salida es la nueva base de datos. Las primitivas extraídas de las bases de datos de generación se recortan y normalizan para presentarlas a las demás etapas del ducto. Estas etapas se estructuran de manera muy similar a las del ducto del algoritmo de *z-buffer*, excepto que el único proceso de superficies visibles que requieren es para garantizar que cada polígono sobreescriba de manera correcta cualquier polígono previamente discretizado que interseque.



**Figura 14.43** Ducto de generación para el algoritmo de prioridad de listas y sombreado de Phong.

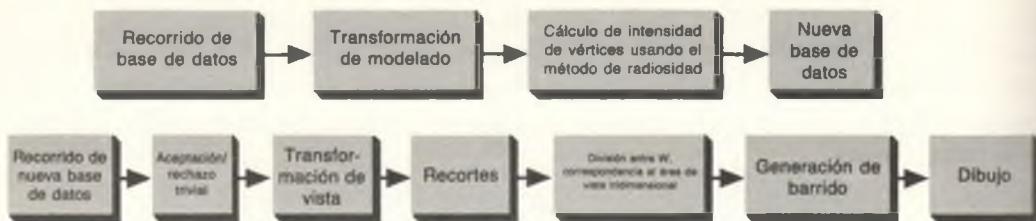


Figura 14.44 Ducto de generación para la radiosidad y el sombreado de Gouraud.

#### 14.9.2 Ductos de iluminación global

Hasta ahora hemos ignorado la iluminación global. Como señalamos antes, la incorporación de efectos de iluminación global requiere información acerca de las relaciones geométricas entre el objeto que se genera y los demás objetos en el mundo. En el caso de sombras, que sólo dependen de la posición de la fuente luminosa y no de la posición del observador, el preprocesamiento del ambiente para añadir sombras poligonales de detalle de superficie permite usar un ducto que de otra manera sería convencional.

**Radiosidad.** Los algoritmos de radiosidad constituyen un interesante ejemplo de cómo aprovechar el ducto convencional para lograr efectos de iluminación global. Los algoritmos de la sección 14.8 procesan objetos y les asignan un conjunto de intensidades de vértices independientes de la vista. Estos objetos se pueden presentar a una versión modificada del ducto para el algoritmo de *z-buffer* y sombreado de Gouraud, ilustrado en la figura 14.44, que elimina la etapa de iluminación.

**Rastreo de rayos.** Por último, consideraremos la traza de rayos, cuyo ducto, presentado en la figura 14.45, es el más sencillo porque los objetos visibles en cada pixel y su iluminación se determinan por completo en coordenadas de mundo. Una vez que se han obtenido los objetos de la base de datos y se han transformado usando la transformación de vista, se cargan en la base de datos de coordenadas mundiales del trazador de rayos, la cual se implanta cuidadosamente para apoyar el cálculo eficiente de la intersección de rayos.



Figura 14.45 Ducto de generación para el rastreo de rayos.

### 14.9.3 Refinamiento progresivo

Una modificación interesante a los ductos que hemos analizado se sirve del hecho de que la imagen es vista durante un tiempo finito. En lugar de tratar de generar de una vez una versión final de la imagen, se puede generar primero la imagen de manera burda para luego refinarla progresivamente y mejorarla. Por ejemplo, la primera imagen se puede generar con modelos de objetos simples, sombreado sencillo y sin eliminación de artefactos de discretización. Conforme el usuario observa la imagen, pueden dedicarse ciclos inactivos a mejorar su calidad [FORR85]. Si hay alguna pauta que dictamine lo que debe hacerse a continuación, el refinamiento puede efectuarse de manera adaptable. Bergman, Fuchs, Grant y Spach [BERG86b] han desarrollado un sistema de este tipo que usa diversas estrategias heurísticas para determinar cómo debe invertir su tiempo. Por ejemplo, a un objeto se le aplica el sombreado de Gouraud en lugar del sombreado constante únicamente si el intervalo de intensidades de vértices excede de cierto valor límite. Los algoritmos de traza de rayos [PAIN89] y de radiosidad [COHE88] son apropiados para el refinamiento progresivo.

## RESUMEN

En este capítulo vimos muchos modelos de iluminación, algunos inspirados principalmente por la necesidad de la eficiencia y otros que tratan de considerar los aspectos físicos de la interacción entre las superficies y la luz. Vimos cómo se puede usar la interpolación en los modelos de sombreado, tanto para minimizar el número de puntos donde se evalúa la ecuación de iluminación como para permitir la aproximación de superficies curvas con mallas poligonales. Comparamos los métodos de iluminación local que consideran en forma aislada cada punto de la superficie y las luces que iluminan en forma directa cada punto, con las estrategias globales que consideran la refracción y la reflexión de otros objetos en el ambiente.

Como hemos recalcado en todo este capítulo, la amplia gama de algoritmos de iluminación y sombreado origina una diversidad correspondiente en las imágenes que se pueden producir para la misma escena usando la misma especificación de vista. La decisión acerca de cuál es el algoritmo que se debe usar depende de muchos factores, incluyendo los fines para los cuales se genera la imagen. Aunque muchas veces se sacrifica el fotorrealismo en aras de la eficiencia, los adelantos en los algoritmos y el hardware pronto harán realidad las implantaciones en tiempo real de modelos de iluminación global físicamente correctos. Sin embargo, cuando la eficiencia deje de ser un factor primordial, en algunas ocasiones preferiremos generar imágenes sin textura, sombras ni refracción, ya que en algunos casos ésta será la mejor forma de comunicar la información deseada al observador.

## Ejercicios

- 14.1 (a) Describa la diferencia en apariencia que esperaría entre un modelo de iluminación de Phong que usa  $(\vec{N} \cdot \vec{H})^\alpha$  y otro que usa  $(\vec{R} \cdot \vec{V})^\alpha$ . (b) Demuestre que  $\alpha = 2\beta$  cuando todos los vectores de la figura 14.12 son coplanares. (c) Demuestre que esta relación *no* es verdadera en términos generales.

14.2 Demuestre que los resultados de la interpolación de información de vértices en las aristas y líneas de rastreo de un polígono son independientes de la orientación en el caso de triángulos.

14.3 Suponga que hay tres polígonos, *A*, *B* y *C*, que intersecan el mismo proyector en orden de distancia creciente con respecto al observador. Demuestre que, en términos generales, si los polígonos *A* y *B* son transparentes, el color calculado para un pixel en la intersección de las proyecciones dependerá de si la ecuación (14.23) se evalúa con los polígonos *A* y *B* considerados como los polígonos 1 y 2 o como los polígonos 2 y 1.

14.4 Considere la utilización de la correspondencia de texturas para modificar o reemplazar distintas propiedades materiales. Liste los efectos que se pueden producir estableciendo la correspondencia de propiedades en forma única o combinada.

14.5 ¿Qué otros efectos de iluminación cree usted que pueden generalizar las aletas y los conos de Warn?

14.6 Implante un sencillo trazador de rayos recursivo basado en el material de las secciones 13.4 y 14.7.

14.7 Explique por qué la iluminación debe efectuarse antes que los recortes en el ducto de la figura 14.41.

14.8 Implante un sistema de pruebas para experimentar con los modelos de iluminación local. Almacene una imagen que contenga, para cada pixel, su índice de superficie visible en una tabla de propiedades materiales, la normal a la superficie, la distancia al observador y la distancia de un vector normalizado a una o más fuentes luminosas. Permita que el usuario modifique la ecuación de iluminación, la intensidad y el color de las luces, y las propiedades de las superficies. Genere la superficie cada vez que se haga un cambio. Use la ecuación (14.20) con atenuación de fuente lumínosa (Ec. 14.8) e indicación de profundidad (Ec. 14.11).

# Apéndice: Versión original en inglés de las codificaciones de SRGP y PHIGS

## A.1 Programas del capítulo 2

*Programa 2.2  
(pág. 35):*

```
SRGP_setFillStyle (BITMAP_PATTERN_OPAQUE);
SRGP_setFillBitmapPattern (BRICK_BIT_PATTERN);          /* Patrón de ladrillos */
SRGP_fillPolygon (3, triangle_coords);                  /* a */

SRGP_setFillBitmapPattern (MEDIUM_GRAY_BIT_PATTERN);   /* Gris al 50 por ciento */
SRGP_fillEllipseArc (ellipseArcRect, 60.0, 290);       /* b */

SRGP_setFillBitmapPattern (DIAGONAL_BIT_PATTERN);
SRGP_fillRectangle (opaqueFilledRect);                  /* c */

SRGP_setFillStyle (BITMAP_PATTERN_TRANSPARENT);
SRGP_fillRectangle (transparentFilledRect);             /* d */

SRGP_setFillStyle (SOLID);
SRGPSetColor (COLOR_WHITE);
SRGP_fillEllipse (circleRect);                          /* e */

Programa 2.3  
(pág. 41):
```

```
void MakeQuitButton (rectangle buttonRect)
{
    point centerOfButton, textOrigin;
    int width, height, descent;
```

```

SRGP_setFillStyle (SOLID);
SRGPSetColor (COLOR_WHITE);
SRGP_fillRectangle (buttonRect);
SRGPSetColor (COLOR_BLACK);
SRGP_setLineWidth (2);
SRGP_Rectangle (buttonRect);
SRGP_inquireTextExtent ("quit", &width, &height, &descent);
centerOfButton.x = (buttonRect.bottomLeft.x + buttonRect.topRight.x)/2;
centerOfButton.y = (buttonRect.bottomLeft.y + buttonRect.topRight.y)/2;
textOrigin.x = centerOfButton.x - (width/2);
textOrigin.y = centerOfButton.y - (height/2);
SRGP_text (textOrigin, "quit");
}

```

*Programa 2.5*  
(pág. 48):

*establecer atributos de color y patrones y el tamaño del pincel en halfBrushHeight y halfBrushWidth*

```
SRGP_setInputMode (LOCATOR, SAMPLE);
```

```
/* Primero muestrear hasta que se oprima el botón */
do {
```

```
    SRGP_sampleLocator (&locMeasure);
```

```
} while (locMeasure.buttonChord[0] == UP);
```

*/\* Ciclo de pintura:*

*Colocar continuamente el pincel y muestrear hasta que se libere el botón \*/*

```
do {
```

```
    rect = SRGP_defRectangle (locMeasure.position.x - halfBrushWidth,
        locMeasure.position.y - halfBrushHeight,
        locMeasure.position.x + halfBrushWidth,
        locMeasure.position.y + halfBrushHeight);
```

```
    SRGP_fillRectangle (rect);
```

```
    SRGP_sampleLocator (&locMeasure);
```

```
} while (locMeasure.buttonChord[0] == DOWN);
```

*Programa 2.6*  
(pág. 50):

```
# define KEYMEASURE_SIZE 80
```

```
SRGP_setInputMode (KEYBOARD, EVENT); /* Suponga que sólo hay un teclado activo */
```

```
SRGP_setKeyboardProcessingMode (EDIT);
```

```
punto = SRGP_defPoint (100, 100);
```

```
SRGP_text (pt, "Especifique uno o más archivos que desee eliminar; oprima Retorno para salir\n");
```

*/\* ciclo principal del evento \*/*

```
do {
```

```
    inputDev = SRGP_waitEvent (INDEFINITE);
```

```
    SRGP_getKeyboard (measure, KEYMEASURE_SIZE);
```

```
    if (strcoll (measure, ""))
```

```
        DeleteFile (measure);
```

```
        /* DeleteFile lleva a cabo la confirmación, etc. */
```

```
}
```

```
while (strcoll (measure, ""));
```

**Programa 2.7**

(pág. 51):

```
# define QUIT 0
crear el botón de salida en la pantalla;
SRGP_setLocatorButtonMask (LEFT_BUTTON_MASK);
SRGP_setInputMode (LOCATOR, EVENT);/* Suponga que sólo está activo el localizador */
/* ciclo principal del evento */
terminate = FALSE;
do {
    inputDev = SRGP_waitEvent (INDEFINITE);
    SRGP_getLocator (&measure);
    if (measure.buttonChord[SALIDA] == DOWN) {
        if PickedQuitButton (measure.position) terminate = TRUE;
        else
            SRGP_marker (measure.position);
    }
}
while (!terminate);
```

**Programa 2.8**

(pág. 52):

```
# define PLACE_BUTTON 0
# define QUIT_BUTTON 2

generar distribución inicial de la pantalla;
SRGP_setInputMode (KEYBOARD, EVENT);
SRGP_setKeyboardProcessingMode (RAW);
SRGP_setInputMode (LOCATOR, EVENT);
SRGP_setLocatorButtonMask (LEFT_BUTTON_MASK · RIGHT_BUTTON_MASK);
/* Ignorar el segundo botón */
/* Ciclo principal del evento */
terminate = FALSE;
do {
    device = SRGP_waitEvent (INDEFINITE);
    switch (device) {
        case KEYBOARD:
            SRGP_getKeyboard (keyMeasure, lbuf);
            terminate = (keyMeasure[0] == 'q' · (keyMeasure_tecla[0] == 'Q'));
            break;
        case LOCATOR:
            SRGP_getLocator (locMeasure);
            switch (locMeasure.buttonOfMostRecentTransition) {
                case PLACE_BUTTON:
                    if ((locMeasure.buttonChord[PLACE_BUTTON] == DOWN)
                        && InDrawingArea (lockMeasure.position))
                        SRGP_marker (locMeasure.position);
                    break;
                case QUIT_BUTTON:
                    terminate = TRUE;
                    break;
            } /* case del botón */
    } /* case del localizador */
} /* case del dispositivo */
}
while (!terminate);
```

*Programa 2.9  
(pág. 54):*

```
void HighLevelInteractionHandler (locatorMeasure measureOfLocator)
{
    if GEOM__pointInRect (measureOfLocator.position, menuBarExtent) {
        /* Averiguar si el usuario seleccionó un encabezado de menú y cuál fue;
           después presentar el cuerpo del menú */
        menuID = CorrelateMenuBar (measureOfLocator.position);
        if (menuID>0) {
            chosenItemIndex = PerformPullownMenuItemInteraction (menuID);
            if (chosenItemIndex>0)
                PerformActionChosenFromMenu (menuID, chosenItemIndex);
        }
    } else /* El usuario seleccionó dentro del área de dibujo; detectar qué fue y responder */
    {
        objectID = CorrelateDrawingArea (measureOfLocator.position);
        if (objectID>0) ProcessObject (objectID);
    }
}
```

*Programa 2.11  
(pág. 63):*

```
/* Este fragmento de código copia una imagen de cuerpo de menú a la pantalla, en la posición
   almacenada en el registro del cuerpo */

/* Guardar el ID del lienzo activo, que no tiene que ser la pantalla */
saveCanvasID = SRGP__inquireActiveCanvas ( );

/* Guardar el valor de atributo del rectángulo de recorte del lienzo de pantalla */
SRGP__useCanvas (SCREEN_CANVAS);
saveClipRectangle = SRGP__inquireClipRectangle ( );

/* Asignar temporalmente el rectángulo de recorte para permitir la escritura de toda la pantalla */
SRGP__setClipRectangle (SCREEN_EXTENT);

/* Copiar el cuerpo del menú de su lienzo al área correspondiente debajo del encabezado en la
   barra de menú */
SRGP__copyPixel (menuCanvasID, menuBodyExtent, menuBodyScreenExtent.lLeft);

/* Restaurar atributos de la pantalla y el lienzo activo */
SRGP__setClipRectangle (saveClipRectangle);
SRGP__useCanvas (saveCanvasID);
```

## A.2 Programas del capítulo 3

*Programa 3.11  
(pág. 135):*

```
void SRGP__characterText (point origin, char stringToPrint, fontCacheDescriptor fontInfo)
/* Origin es donde se colocará el carácter en el lienzo actual */
{
    rectangle fontCacheRectangle;
    char charToPrint;
```

```

int i;
charLocation *fp;

/* El origin especificado por la aplicación es la línea base y no incluye el asta descendente */
origin.y -= fontInfo.descenderHeight;

for (i = 0; i < strlen(stringToPrint); i++) {
    charToPrint = stringToPrint[i];
    fp = &fontInfo.locationTable[charToPrint];
    /* Encontrar la región rectangular en la memoria caché donde está el carácter */
    SRGP_defPoint (fp->leftX, 0, fontCacheRectangle.bottomLeft);
    SRGP_defPoint (fp->leftX, + fp->width - 1, fontInfo.totalHeight - 1,
    fontCacheRectangle.topRight);
    SRGP_copyPixel (fontInfo.cache, fontCacheRectangle, origin);
    /* Actualizar el origin para avanzar después del nuevo carácter más el espacio entre caracteres */
    origin.x += fp->width + interCharacterSpacing;
}

```

## A.3 Programas del capítulo 7

*Programa 7.1  
(pág. 286):*

```

polyLine:
    void SPH_setLineStyle (style CONTINUOUS / DASHED / DOTTED / DOT_DASHED);
    void SPH_setLineWidthScaleFactor (double scaleFactor);
    void SPH_setLineColor (int colorIndex);

```

relleno de áreas y poliedros:

```

void SPH_setInteriorColor (int colorIndex);
void SPH_setEdgeFlag (flag EDGE_VISIBLE / EDGE_INVISIBLE);
void SPH_setEdgeStyle (style CONTINUOUS / DASHED / DOTTED / DOT_DASHED);
void SPH_setEdgeWidthScaleFactor (double scaleFactor);
void SPH_setEdgeColor (int colorIndex);

```

*polyMarker:*

```

void SPH_setMarkerStyle (style MARKER_CIRCLE / MARKER_SQUARE / ...);
void SPH_setMarkerSizeScaleFactor (double scaleFactor);
void SPH_setMarkerColor (int colorIndex);

```

*texto:*

```

void SPH_setTextFont (int fontIndex);
void SPH_setTextColor (int colorIndex);

```

*Programa 7.2  
(pág. 290):*

```

/* Para establecer el sistema de coordenadas de referencia de visualización UVN */
void SPH_evaluateViewOrientationMatrix (point_3D viewRefPoint,
vector_3D viewPlaneNormal, vector_3D viewUpVector,
matrix_4x4 *voMatrix);

```

```

/* Para establecer el volumen de vista y describir como corresponde al espacio NPC */
void SPH_evaluateViewMappingMatrix (
/* Primero se seleccionan las fronteras del plano de vista */
    double umin, double umax, double vmin, double vmax,
    int projectionType,/* PARALLEL / PERSPECTIVE */
    point_3D projectionReferencePoint/* En VRC */
    double frontPlaneDistance, double backPlaneDistance, /* Planos de recorte */
/* Despues se especifica el área de vista NPC */
    double NPCvp_minX, double NPCvp_maxX,
    double NPCvp_minY, double NPCvp_maxY,
    double NPCvp_minZ, double NPCvp_maxZ,
    matrix_4x4 *vmMatrix);

/* definir las constantes necesarias para usarse en el programa */
/* definir la geometría necesaria para la habitación, los objetos y el robot */

main (argc, argv)
int argc;
char **argv;
{

/* establecer vistas iniciales, colores y parámetros de SPHIGS */

/* Prepararse para la entrada del evento */
SPH_setInputMode (KEYBOARD, EVENT);
SPH_setKeyboardProcessingMode (RAW);

SPH_setInputMode (LOCATOR, EVENT);
SPH_setLocatorButtonMask (LEFT_BUTTON_MASK);

ShowWorld (OUR_ROOM)                                /* presentar la vista inicial de la habitación */
terminate = NO;
do {
    whichdev = SPH_waitEvent (INDEFINITE);
    switch (whichdev) {
        case KEYBOARD:
            SPH_getKeyboard (keymeasure, KEYMEASURE_SIZE);
            if (keymeasure[0] == 'Q' || keymeasure[0] == 'q')
                terminate = YES;
            break;
        case LOCATOR: {
            SPH_getLocator (&curr_locmeasure);
            /* si se oprimió el botón, se revisa si hay una selección */
            if (curr_locmeasure.button_chord[LEFT_BUTTON] == DOWN) {
                SPH_pickCorrelate (curr_locmeasure.position,
                                    curr_locmeasure.view_index, &pick_info);
                if (pick_info.pickLevel > 2) /* objeto en segundo nivel */
                    if (pick_info.path[1].structureID == OBJECT_SET)
                        MakeAndRunPlan (pick_info.path[1].pickID);
            }
        }
    }
}

```

```
        }
        break;
    }
}
while (!terminate);
SPH_end ();
}

void MakeAndRunPlan (int object_id)
{
    double x, z;

    x = objects_info[object_id].x;
    z = objects_info[object_id].z;

    /* movimiento para alinear el robot con el objeto en x */
    MoveRobot (x, Z_WAIT);
    /* encarar el objeto y aproximarse a él */
    SpinRobot (NORTH, CLOCKWISE);
    MoveRobot (x, z + DIST_OF_APPROACH);

    /* tomar el objeto */
    LowerArm ( );
    Grab ( );
    Pickup (object_id);
    RaiseArm ( );
    /* objeto sujetado. */

    /* llevarse el objeto */
    SpinRobot (EAST, COUNTERCLOCKWISE);
    SpinRobot (SOUTH, COUNTERCLOCKWISE);
    MoveRobot (x, Z_HALL);
    SpinRobot (EAST, CLOCKWISE);
    MoveRobot (X_OFFSCREEN, Z_HALL);

    /* soltar el objeto */
    LowerArm ( );
    UnGrab ( );
    LetGo ( );
    RaiseArm ( );

    /* regresar al área de espera */
    SpinRobot (WEST, COUNTERCLOCKWISE);
    MoveRobot (X_WAIT, Z_HALL);
    SpinRobot (NORTH, COUNTERCLOCKWISE);
    MoveRobot (X_WAIT, Z_WAIT);
    SpinRobot (EAST, COUNTERCLOCKWISE);
}
```

```

void PickUp (int id)
/* Elimina el objeto indicado de la habitación y lo agrega
   a la mano (brazo) del robot */
{
    matrix temp_matrix; /* matriz de transformación */

    SPH_openStructure (OBJECT_SET);
    SPH_setElementPointer (0);
    SPH_moveElementPointerToLabel (objects_info[ID].label);
    SPH_offsetElementPointer (1);
    SPH_deleteElement ( );
    SPH_executeStructure (NULL_OBJECT);
    SPH_closeStructure ( );

    SPH_openStructure (ARM);
    SPH_setElementPointer (0);
    SPH_moveElementPointerToLabel (OUR_ARM_DRAW_ROD);
    SPH_offsetElementPointer (1);
    SPH_deleteElement ( );
    SPH_executeStructure (objects_info[ID].struct_id);
    SPH_closeStructure ( );
/* mostrar el cuadro: */
    SPH regenerateScreen ( );
}

```

## A.4 Programas del capítulo 9

*Programa sin número  
de las págs. 384-386:*

```

#include "srpg.h"
#include <stdio.h>

#define KEYMEASURE_SIZE 80
#define WINDOW_SIZE 400
#define NÚM_STEPS 20

void DrawCurve (point *ControlPoints, int n)
{
    int i;
    float t, delta;
    point CurvePoints[n];

    CurvePoints[0].x = ControlPoints[0].x;
    CurvePoints[0].y = ControlPoints[0].y;
    delta = 1.0 / n;
    /* Las curvas de Bézier interpolan el primero */
    /* y el último de los puntos de control */
    /* La curva se aproximará con n puntos */
    /* t está en el intervalo 0.0 a 1.0 */
    for (i = 1; i <= n; i++) {
        t = i * delta;

```

```

CurvePoints[i].x = ControlPoints[0].x * (1.0 - t) * (1.0 - t) * (1.0 - t)
+ ControlPoints[1].x * 3.0 * t * (1.0 - t) * (1.0 - t)
+ ControlPoints[2].x * 3.0 * t * t * (1.0 - t)
+ ControlPoints[3].x * 3.0 * t * t * t;

CurvePoints[i].y = ControlPoints[0].y * (1.0 - t) * (1.0 - t) * (1.0 - t)
+ ControlPoints[1].y * 3.0 * t * (1.0 - t) * (1.0 - t)
+ ControlPoints[2].y * 3.0 * t * t * (1.0 - t)
+ ControlPoints[3].y * 3.0 * t * t * t;
}

SRGP_polyLine (n + 1, CurvePoints); /* Dibujar la curva completa */
}

main ( )
{
    locator_measure locMeasure, pastlocMeasure;
    char keyMeasure[KEYMEASURE_SIZE];
    int device;
    int numCtl;
    boolean terminate;
    rectangle screen;
    point ControlPoints[4];

    SRGP_begin ("Curvas de Bézier", WINDOW_SIZE, WINDOW_SIZE, 1, FALSE);
    SRGP_setLocatorEchoType (CURSOR);
    SRGP_setLocatorButtonMask (LEFT_BUTTON_MASK · RIGHT_BUTTON_MASK);
    pastlocMeasure.position = SRGP_defPoint (-1, -1);
                                /* Valor inicial arbitrario para la posición */
    SRGP_setLocatorMeasure (pastlocMeasure.position);
    SRGP_setKeyboardProcessingMode (RAW);
    SRGP_setInputMode (LOCATOR, EVENT);
                                /* Se activan el localizador (ratón) y el teclado */
    SRGP_setInputMode (KEYBOARD, EVENT);
    screen = SRGP_defRectangle (0, 0, WINDOW_SIZE - 1, WINDOW_SIZE - 1);

/* Lazo principal de eventos */
    terminate = FALSE;
    do {
        device = SRGP_waitEvent (INDEFINITE);
        switch (device){
            case KEYBOARD: {
                SRGP_getKeyboard (keyMeasure, KEYMEASURE_SIZE);
                switch (keyMeasure [0]){
                    case 'q':/* Salir del programa */
                        terminate = TRUE;
                        break;
                    case 'c':/* Limpiar la ventana */
                        SRGPSetColor (0);
                        SRGP_fillRectangle (screen);
                        SRGPSetColor (1);
                }
            }
        }
    } while (!terminate);
}

```

```

        break;
    }
    break;
}
/* case del teclado */
case LOCATOR: {
    SRGB_getLocator (&locMeasure);
    switch (locMeasure.buttonOfMostRecentTransition) {
    case LEFT_BUTTON: /* Definición de los puntos de control restantes */
        if ((locMeasure.buttonChord[LEFT_BUTTON] == DOWN) &
            pastlocMeasure.position.x > 0) {
            SRGB_setLocatorEchoRubberAnchor (locMeasure.position);
            SRGB_line (pastlocMeasure.position, locMeasure.position);
            pastlocMeasure = locMeasure;
            ControlPoints[numCtl] = locMeasure.position;
            numCtl++;
        if (numCtl == 4) {
            SRGB_setLineStyle (CONTINUOUS); /* Para dibujar la curva */
            SRGB_setLineWidth (2);
            DrawCurve (ControlPoints, NUM_STEPS);
            pastlocMeasure.position.x = 1;
            SRGB_setLocatorEchoType (CURSOR);
        }
        break;
    case RIGHT_BUTTON: /* Iniciar nuevo conjunto de puntos de control */
        SRGB_setLocatorEchoRubberAnchor (locMeasure.position);
        pastlocMeasure = locMeasure;
        SRGB_setLocatorEchoType (RUBBER_LINE);
        SRGB_setLineStyle (DASHED); /* Para dibujar el polígono de control */
        SRGB_setLineWidth (1);
        ControlPoints[0] = locMeasure.position;
        numCtl = 1;
        break;
    }
}
/* manejo de botones */
/* case del localizador */
/* switch de dispositivo */
} while (!terminate);
SRGP_end ( );
}

```

# Bibliografía

A continuación se presenta una extensa bibliografía acerca de la graficación por computador. Además de ser una lista de las referencias que aparecen en los capítulos del libro, también es una sección que conviene repasar. Con sólo ver los títulos de los libros y artículos podrá tener una buena idea por dónde ha pasado la investigación en este campo y hacia dónde se dirige.

Varias de las revistas aparecen con frecuencia como referencias, por lo que las hemos abreviado. De éstas, las más importantes son *ACM SIGGRAPH Conference Proceedings*, publicada cada año como edición de *Computer Graphics*, y *ACM Transactions on Graphics*. Estas dos fuentes comprenden más de una tercera parte de la bibliografía.

## Abreviaturas

<i>ACM TOG</i>	<i>Association for Computing Machinery, Transactions on Graphics</i>
<i>CACM</i>	<i>Communications of the ACM</i>
<i>CG &amp; A</i>	<i>IEEE Computer Graphics and Applications</i>
<i>CGIP</i>	<i>Computer Graphics and Image Processing</i>
<i>CVGIP</i>	<i>Computer Vision, Graphics and Image Processing (antes CGIP)</i>
<i>FJCC</i>	<i>Proceedings of the Fall Joint Computer Conference</i>
<i>JACM</i>	<i>Journal of the ACM</i>
<i>NCC</i>	<i>Proceedings of the National Computer Conference</i>
<i>SJCC</i>	<i>Proceedings of the Spring Joint Computer Conference</i>
<i>SIGGRAPH 76</i>	<i>Proceedings of SIGGRAPH '76 (Filadelfia, Pennsylvania, 14-16 de julio de 1976). En Computer Graphics, 10(2), verano de 1976, ACM SIGGRAPH, Nueva York.</i>
<i>SIGGRAPH 77</i>	<i>Proceedings of SIGGRAPH '77 (San Jose, California, 20-22 de julio de 1977). En Computer Graphics, 11(2), verano de 1977, ACM SIGGRAPH, Nueva York.</i>

- SIGGRAPH 78** *Proceedings of SIGGRAPH '78 (Atlanta, Georgia, 23-25 de agosto de 1978)*. En *Computer Graphics*, 12(3), agosto de 1978, ACM SIGGRAPH, Nueva York.
- SIGGRAPH 79** *Proceedings of SIGGRAPH '79 (Chicago, Illinois, 8-10 de agosto de 1979)*. En *Computer Graphics*, 13(2), agosto de 1979, ACM SIGGRAPH, Nueva York.
- SIGGRAPH 80** *Proceedings of SIGGRAPH '80 (Seattle, Washington, 14-18 de julio de 1980)*. En *Computer Graphics*, 14(3), julio de 1980, ACM SIGGRAPH, Nueva York.
- SIGGRAPH 81** *Proceedings of SIGGRAPH '81 (Dallas, Texas, 3-7 de agosto de 1981)*. En *Computer Graphics*, 15(3), agosto de 1981, ACM SIGGRAPH, Nueva York.
- SIGGRAPH 82** *Proceedings of SIGGRAPH '82 (Boston, Massachusetts, 26-30 de julio de 1982)*. En *Computer Graphics*, 16(3), julio de 1982, ACM SIGGRAPH, Nueva York.
- SIGGRAPH 83** *Proceedings of SIGGRAPH '83 (Detroit, Michigan, 25-29 de julio de 1983)*. En *Computer Graphics*, 17(3), julio de 1983, ACM SIGGRAPH, Nueva York.
- SIGGRAPH 84** *Proceedings of SIGGRAPH '84 (Minneapolis, Minnesota, 23-27 de julio de 1984)*. En *Computer Graphics*, 18(3), julio de 1984, ACM SIGGRAPH, Nueva York.
- SIGGRAPH 85** *Proceedings of SIGGRAPH '85 (San Francisco, California, 22-26 de julio de 1985)*. En *Computer Graphics*, 19(3), julio de 1985, ACM SIGGRAPH, Nueva York.
- SIGGRAPH 86** *Proceedings of SIGGRAPH '86 (Dallas, Texas, 18-22 de agosto de 1986)*. En *Computer Graphics*, 20(4), agosto de 1986, ACM SIGGRAPH, Nueva York.
- SIGGRAPH 87** *Proceedings of SIGGRAPH '87 (Anaheim, California, 27-31 de julio de 1987)*. En *Computer Graphics*, 21(4), julio de 1987, ACM SIGGRAPH, Nueva York.
- SIGGRAPH 88** *Proceedings of SIGGRAPH '88 (Atlanta, Georgia, 1-5 de agosto de 1988)*. En *Computer Graphics*, 22(4), agosto de 1988, ACM SIGGRAPH, Nueva York.
- SIGGRAPH 89** *Proceedings of SIGGRAPH '89 (Boston, Massachusetts, 31 de julio-4 de agosto de 1989)*. En *Computer Graphics*, 23(3), julio de 1989, ACM SIGGRAPH, Nueva York.

- ABIE89** Abi-Ezzi, S. S., *The Graphical Processing of B-Splines in a Highly Dynamic Environment*, tesis doctoral, Rensselaer Polytechnic Institute, Troy, Nueva York, mayo de 1989.
- ADOB85** Adobe Systems, Inc., *PostScript Language Reference Manual*, Addison-Wesley, Reading, Massachusetts, 1985.
- AMAN87** Amanatides, J. y Woo, A., "A Fast Voxel Traversal Algorithm for Ray Tracing", en Kunii, T. L., editor, *Computer Graphics International 87, London, October 1987*, Springer, Tokio, 1987, págs. 149-155.
- ANSI85** ANSI (American National Standards Institute), *American National Standard for Information Processing Systems—Computer Graphics—Graphical*

- Kernel System (GKS) Functional Description*, ANSI X3.124-1985, ANSI, Nueva York, 1985.
- ANSI88 ANSI (American National Standards Institute), *American National Standard for Information Processing Systems—Programmer's Hierarchical Interactive Graphics System (PHIGS) Functional Description, Archive File Format, Clear-Text Encoding of Archive File*, ANSI X3.144-1988, ANSI, Nueva York, 1988.
- APPE68 Appel, A. "Some Techniques for Shading Machine Renderings of Solids", *SJCC*, 1968, págs. 37-45.
- APPL85 Apple Computer, Inc., *Inside Macintosh*, Addison-Wesley, Reading, Massachusetts, 1985.
- APT85 Apt, C., "Perfecting the Picture", *IEEE Spectrum*, 22(7), julio de 1985, págs. 60-66.
- ATHE81 Atherton, P. R., "A Method of Interactive Visualization of CAD Surface Models on a Color Video Display", *SIGGRAPH 81*, págs. 279-287.
- ATHE83 Atherton, P. R., "A Scan-Line Hideen Surface Removal Procedure for Constructive Solid Geometry", *SIGGRAPH 83*, págs. 73-82.
- BAEC87 Baecker, R. y Buxton, B., *Readings in Human-Computer Interaction*, Morgan Kaufmann, Los Altos, California, 1987.
- BALD85 Baldauf, D., "The Workhorse CRT: New Life", *IEEE Spectrum*, 22(7), julio de 1985, págs. 67-73.
- BANC77 Banchoff, T. F. y Strauss, C. M., *The Hypercube: Proyections and Slicings*, película, International Film Bureau, 1977.
- BANC83 Banchoff, T. y Wermer, J., *Linear Algebra Through Geometry*, Springer-Verlag, Nueva York, 1983.
- BARS83 Barsky, B. y Beatty, J., "Local Control of Bias and Tension in Beta-Splines", *ACM TOG*, 2(2), abril de 1983, págs. 109-134.
- BARS88 Barksy, B., *Computer Graphics and Geometric Modeling Using Beta-Splines*, Springer-Verlag, Nueva York, 1988.
- BART87 Bartels, R., Beatty, J. y Barsky, B., *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann, Los Altos, California, 1987.
- BASS90 Bass, C., Capsimalis, M., Jeske, N., Look, A., Sheppard, S. y Wiegand, G., *HOOPS Graphics System User Guides*, Ithaca Software, 1990.
- BAUM74 Baumgart, B. G., *Geometric Modeling for Computer Vision*, tesis doctoral, Informe AIM-249, STAN-CS-74-463, Computer Science Department, Stanford University, Palo Alto, California, octubre de 1974.
- BAUM75 Baumgart, B. G., "A Polyhedron Representation for Computer Vision", *NCC 75*, págs. 589-596.
- BAYE73 Bayer, B. E., "An Optimum Method for Two-Level Rendition of Continuous-Tone Pictures", en *Conference Record of the International Conference on Communications*, 1973, págs. 26-11-26-15.
- BEAT82 Beatty, J. C. y Booth, K. S., editores., *Tutorial: Computer Graphics*, 2a. ed., IEEE Comp. Soc. Press, Silver Spring, Maryland, 1982.
- BEDF58 Bedford, R. y G. Wyszecki, "Wavelength Discrimination for Point Sources", *Journal of the Optical Society of America*, 48, 1958, págs. 129-ss.
- BERG86a Bergeron, P., "A General Version of Crow's Shadow Volumes", *CG & A*, 6(9), septiembre de 1986, págs. 17-28.

- BERG86b Bergman, L., Fuchs, H., Grant, E. y Spach, S., "Image Rendering by Adaptive Refinement", *SIGGRAPH 86*, págs. 29-37.
- BERK82 Berk, T., Brownston, L. y Kaufman, A., "A New Color-Naming System for Graphics Languages", *CG & A*, 2(3), mayo de 1982, págs. 37-44.
- BEZI70 Bézier, P., *Emploi des Machines à Commande Numérique*, Mason et Cie, París, 1970. Traducido por Forrest, A. R. y Pankhurst, A. F. como Bézier, P., *Numerical Control—Mathematics and Applications*, Wiley, Londres, 1972.
- BEZI74 Bézier, P., "Mathematical and Practical Possibilities of UNISURF", en Barnhill, R. E. y Riesenfeld, R. F., editores, *Computer Aided Geometric Design*, Academic Press, Nueva York, 1974.
- BILL81 Billmeyer, F. y Saltzman, M., *Principles of Color Technology*, 2a. ed., Wiley, Nueva York, 1981.
- BINF71 Binford, T., en *Visual Perception by Computer, Proceedings of the IEEE Conference on Systems and Control*, Miami, Florida, diciembre de 1971.
- BIRR61 Birren, R., *Creative Color*, Van Nostrand Reinhold, Nueva York, 1961.
- BISH60 Bishop, A. y Crook, M., *Absolute Identification of Color for Targets Presented Against White and Colored Backgrounds*. Informe WADD TR 60-611, Wright Air Development Division, Wright Patterson AFB, Dayton, Ohio, 1960.
- BISH86 Bishop, G. y Weimer, D. M., "Fast Phong Shading", *SIGGRAPH 86*, págs. 103-106.
- BLIN76 Blinn, J. F. y Newell, M. E., "Texture and Reflection in Computer Generated Images", *CACM*, 19(10), octubre de 1976, págs. 542-547. También en BEAT82, págs. 456-461.
- BLIN77a Blinn, J. F., "Models of Light Reflection for Computer Synthesized Pictures", *SIGGRAPH 77*, págs. 192-198. También en FREE80, págs. 316-322.
- BLIN77b Blinn, J. F., "A Homogeneous Formulation for Lines in 3-Space", *SIGGRAPH 77*, págs. 237-241.
- BLIN78a Blinn, J. F. y Newell, M. E., "Clipping Using Homogeneous Coordinates", *SIGGRAPH 78*, págs. 245-251.
- BLIN78b Blinn, J. F., "Simulation of Wrinkled Surfaces", *SIGGRAPH 78*, págs. 286-292.
- BLIN88 Blinn, J. F., "Me and My (Fake) Shadow", *CG & A*, 9(1), enero de 1988, págs. 82-86.
- BOUK70a Bouknight, W. J., "A Procedure for Generation of Three-Dimensional Half-Toned Computer Graphics Presentations", *CACM*, 13(9), septiembre de 1970, págs. 527-536. También en FREE80, págs. 292-301.
- BOUK70b Bouknight, W. J. y Kelly, K. C., "An Algorithm for Producing Half-Tone Computer Graphics Presentations with Shadows and Movable Light Sources", *SJCC*, AFIPS Press, Montvale, Nueva Jersey, 1970, págs. 1-10.
- BOUV85 Bouville, C., "Bounding Ellipsoids for Ray-Fractal Intersection", *SIGGRAPH 85*, págs. 45-52.
- BOYN79 Boynton, R. M., *Human Color Vision*, Holt, Rinehart, and Winston, Nueva York, 1979.
- BOYS82 Boyse, J. W. y Gilchrist, J. E., GMSolid: "Interactive Modeling for Design and Analysis of Solids", *CG & A*, 2(2), marzo de 1982, págs. 27-40.
- BÖHM80 Böhm, W., "Inserting New Knots into B-spline Curves", *Computer Aided Design*, 12(4), julio de 1980, págs. 199-201.

- BÖHM84 Böhm, W., Farin, G. y Kahmann, J., "A Survey of Curve and Surface Methods in CAGD", *Computer Aided Geometric Design*, 1(1), julio de 1984, págs. 1-60.
- BRAI78 Braid, I. C., Hillyard, R. C. y Stroud, I. A., *Stepwise Construction of Polyhedra in Geometric Modeling*, CAD Group Document no. 100, Cambridge University, Cambridge, Inglaterra, 1978. También en Brodlie, K. W., editor, *Mathematical Methods in Computer Graphics and Design*, Academic Press, Nueva York, 1980, págs. 123-141.
- BRES65 Bresenham, J. E., "Algorithms for Computer Control of a Digital Plotter", *IBM Systems Journal*, 4(1), 1965, págs. 25-30.
- BRES77 Bresenham, J. E., "A Linear Algorithm for Incremental Digital Display of Circular Arcs", *Communications of the ACM*, 20(2), febrero de 1977, págs. 100-106.
- BRES83 Bresenham, J. E., Grice, D. G. y Pi, S. C., "Bi-Directional Display of Circular Arcs", Patente de E.U.A. 4,371,933, primero de febrero de 1983.
- BRIT78 Britton, E., Lipscomb, J. y Pique, M., "Making Nested Rotations Convenient for the User", *SIGGRAPH 78*, págs. 222-227.
- BROT84 Brotman, L. S. y Badler, N. I., "Generating Soft Shadows with a Depth Buffer Algorithm", *CG & A*, 4(10), octubre de 1984, págs. 5-12.
- BUIT75 Bui-Tuong, Phong, Illumination for Computer Generated Pictures, *CACM*, 18(6), junio de 1975, págs. 311-317. También en BEAT82, págs. 449-455.
- CABR87 Cabral, D., Max, N. y Springmeyer, R., "Bidirectional Reflection Functions from Surface Bump Maps", *SIGGRAPH 87*, págs. 273-281.
- CACM84 "An Interview with Andries van Dam", *CACM*, 27(7), agosto de 1984.
- CARD83 Card, S., Moran, T. y Newell, A., *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, Nueva Jersey, 1983.
- CALR78 Carlbom, I. y Paciorek, J., "Planar Geometric Projections and Viewing Transformations", *Computing Surveys*, 10(4), diciembre de 1978, págs. 465-502.
- CARP84 Carpenter, L., "The A-buffer, an Antialiased Hidden Surface Method", *SIGGRAPH 84*, 103-108.
- CATM74 Catmull, E., *A Subdivision Algorithm for Computer Display of Curved Surfaces*, tesis doctoral, Informe UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt Lake City, Utah, diciembre de 1974.
- CATM75 Catmull, E., "Computer Display of Curved Surfaces", en *Proc. IEEE Conf. on Computer Graphics, Pattern Recognition and Data Structures*, mayo de 1975. También en FREE80, págs. 309-315.
- CHAP72 Chapanis, A. y Kinkade, R., "Design of Controls", en Van Cott, H. y Kinkade, R., editores, *Human Engineering Guide to Equipment Design*, U. S. Government Printing Office, 1972.
- CHAS81 Chasen, S. H., "Historical Highlights of Interactive Computer Graphics", *Mechanical Engineering*, 103, ASME, noviembre de 1981, págs. 32-41.
- CHIN89 Chin, N. y Feiner, S., "Near Real-Time Shadow Generation Using BSP Trees", *SIGGRAPH 89*, págs. 99-106.
- CHIN90 Chin, N., *Near Real-Time Object-Precision Shadow Generation Using BSP Trees*, tesis de maestría, Department of Computer Science, Columbia University, Nueva York, 1990.

- CHUN89 Chung, J. C. et al., "Exploring Virtual Worlds with Head-Mounted Displays", *Proc. SPIE Meeting on Non-Holographic True 3-Dimensional Display Technologies*, 1083, Los Angeles, 15-20 de enero de 1989.
- CLAR76 Clark, J. H., "Hierarchical Geometric Models for Visible Surface Algorithms", *CACM*, 19(10), octubre de 1976, págs. 547-554. También en BEAT82, págs. 296-303.
- CLAR79 Clark, J., "A Fast Scan-Line Algorithm for Rendering Parametric Surfaces", resumen en *SIGGRAPH 79*, pág. 174. También en Whitted, T. y Cook, R., editores, *Image Rendering Tricks, Course Notes 16 for SIGGRAPH 86*, Dallas, Texas, agosto de 1986. También en JOY88, págs. 88-93.
- CLEV83 Cleveland, W. y McGill, R., "A Color-Caused Optical Illusion on a Statistical Graph", *The American Statistician*, 37(2), mayo de 1983, págs. 101-105.
- COHE85 Cohen, M. F. y Greenberg, D. P., "The Hemi-Cube: A Radiosity Solution for Complex Environments", *SIGGRAPH 85*, págs. 31-40.
- COHE88 Cohen, M. F., Chen, S. E., Wallace, J. R. y Greenberg, D. P., "A Progressive Refinement Approach to Fast Radiosity Image Generation", *SIGGRAPH 88*, págs. 75-84.
- CONR85 Conrac Corporation, *Raster Graphics Handbook*, 2a. ed., Van Nostrand Reinhold, Nueva York, 1985.
- COOK82 Cook, R. y Torrance, K., "A Reflectance Model for Computer Graphics", *ACM TOG*, 1(1), enero de 1982, págs. 7-24.
- COOK84 Cook, R. L., "Shade Trees", *SIGGRAPH 84*, págs. 223-231.
- COWA83 Cowan, W., "An Inexpensive Scheme for Calibration of a Colour Monitor in Terms of CIE Standard Coordinates", *SIGGRAPH 83*, págs. 315-321.
- CROW77 Crow, F. C., "Shadow Algorithms for Computer Graphics", *SIGGRAPH 77*, págs. 242-247. También en BEAT82, págs. 442-448.
- CROW87 Crow, F. C., "The Origins of the Teapot", *CG & A*, 7(1), enero de 1987, págs. 8-19.
- CYRU78 Cyrus, M. y Beck, J., "Generalized Two- and Three-Dimensional Clipping", *Computers and Graphics*, 3(1), 1978, págs. 23-28.
- DEBO78 de Boor, C., *A Practical Guide to Splines*, "Applied Mathematical Sciences", vol. 27, Springer-Verlag, Nueva York, 1978.
- DIPP84 Dippé, M. y Swensen, J., "An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis", *SIGGRAPH 84*, págs. 149-158.
- DUFF79 Duff, T., "Smoothly Shaded Renderings of Polyhedral Objects on Raster Displays", *SIGGRAPH 79*, págs. 270-275.
- DURB88 Durbeck, R. y Sherr, S., editores, *Output Hardcopy Devices*, Academic Press, Nueva York, 1988.
- DUVA90 Duvanenko, V., W. E. Robbins y R. S. Gyurcsik, "Improved Line Segment Clipping", *Dr. Dobb's Journal*, julio de 1990, págs. 36-45, 98-100.
- DVOR43 Dvorák, A., "There Is a Better Typewriter Keyboard", *National Business Education Quarterly*, 12(2), 1943, págs. 51-58.
- ENCA72 Encarnação, J. y Giloi, W., "PRADIS—An Advanced Programming System for 3-D-Display", *SJCC*, AFIPS Press, Montvale, Nueva Jersey, 1972, págs. 985-998.
- ENGE68 Englebart, D. C. y English, W. K., *A Research Center for Augmenting Human Intellect*, FJCC, Thompson Books, Washington, D. C., 1968, págs. 395.

- FAUX79 Faux, I. D. y Pratt, M. J., *Computational Geometry for Design and Manufacture*, Wiley, Nueva York, 1979.
- FITT54 Fitts, P., "The Information Capacity of the Human Motor System in Controlling Amplitude of Motion", *Journal of Experimental Psychology*, 47(6), junio de 1954, págs. 381-391.
- FIUM89 Fiume, E. L., *The Mathematical Structure of Raster Graphics*, Academic Press, San Diego, 1989.
- FLOY75 Floyd, R. y Steinberg, L., "An Adaptive Algorithm for Spatial Gray Scale", en *Society for Information Display 1975 Symposium Digest of Technical Papers*, 1975, pág. 36.
- FOLE82 Foley, J. y van Dam, A., *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, Massachusetts, 1982.
- FOLE84 Foley, J., Wallace, V. y Chan, P., The Human Factors of Computer Graphics Interaction Techniques", *CG & A*, 4(11), noviembre de 1984, 13-48.
- FOLE87 Foley, J., "Interfaces for Advanced Computing", *Scientific American*, 257(4), octubre de 1987, págs. 126-135.
- FOLE90 Foley, J., van Dam, A., Feiner, S. y Hughes, J., *Computer Graphics: Principles and Practice, Second Edition*, Addison-Wesley, Reading, Massachusetts, 1990.
- FORR79 Forrest, A. R., "On the Rendering of Surfaces", *SIGGRAPH 79*, págs. 253-259.
- FORR80 Forrest, A. R., "The Twisted Cubic Curve: A Computer-Aided Geometric Design Approach", *Computer Aided Design*, 12(4), julio de 1980, págs. 165-172.
- FORR85 Forrest, A. R., "Antialiasing in Practice", en Earnshaw, R. A., editor, *Fundamental Algorithms for Computer Graphics*, NATO ASI Series F: Computer and Systems Sciences, vol. 17, Springer-Verlag, Nueva York, 1985, págs. 113-134.
- FOUR82 Fournier, A., Fussell, D. y Carpenter, L., "Computer Rendering of Stochastic Models", *CACM*, 25(6), junio de 1982, págs. 371-384.
- FOUR88 Fournier, A. y Fussell, D., "On the Power of the Frame Buffer", *ACM TOG*, 7(2), abril de 1988, págs. 103-128.
- FRAN80 Franklin, W. R., "A Linear Time Exact Hidden Surface Algorithm", *SIGGRAPH 80*, págs. 117-123.
- FRAN81 Franklin, W. R., "An Exact Hidden Sphere Algorithm that Operates in Linear Time", *CGIP*, 15(4), abril de 1981, págs. 364-379.
- FREE80 Freeman, H., editor, *Tutorial and Selected Readings in Interactive Computer Graphics*, IEEE Comp. Soc. Press, Silver Spring, Maryland, 1980.
- FROM84 Fromme, F., "Improving Color CAD Systems for Users: Some Suggestions from Human Factors Studies", *IEEE Design and Test of Computers*, 1(1), febrero de 1984, págs. 18-27.
- FUCH80 Fuchs, H., Kedem, Z. M. y Naylor, B. F., "On Visible Surface Generation by A Priori Tree Structures", *SIGGRAPH 80*, págs. 124-133.
- FUCH83 Fuchs, H., Abram, G. D. y Grant, E. D., "Near Real-Time Shaded Display of Rigid Objects", *SIGGRAPH 83*, págs. 65-72.
- FUJI85 Fujimura, K. y Kunii, T. L., "A Hierarchical Space Indexing Method", en Kunii, T. L., editor, *Computer Graphics: Visual Technology and Art*,

- Proceedings of Computer Graphics Tokyo '85 Conference*, Springer-Verlag, 1985, págs. 21-34.
- GILO78 Giloi, W. K., *Interactive Computer Graphics—Data Structures, Algorithms, Languages*, Prentice-Hall, Englewood Cliffs, Nueva Jersey, 1978.
- GLAS84 Glassner, A. S., "Space Subdivision for Fast Ray Tracing", *CG & A*, 4(10), octubre de 1984, págs. 15-22.
- GLAS89 Glassner, A. S., editor, *An Introduction to Ray Tracing*, Academic Press, Londres, 1989.
- GOLD71 Goldstein, R. A. y Nagel, R., "3-D Visual Simulation", *Simulation*, 16(1), enero de 1971, págs. 25-31.
- GOLD83 Goldberg, A. y Robson, D., *SmallTalk 80: The Language and Its Implementation*, Addison-Wesley, Reading, Massachusetts, 1983.
- GORA84 Goral, C. M., Torrance, K. E., Greenberg, D. P. y Battaile, B., "Modeling the Interaction of Light Between Diffuse Surfaces", *SIGGRAPH 84*, págs. 213-222.
- GOSS88 Gossard, D., Zuffante, R. y Sakurai, H., "Representing Dimensions, Tolerances, and Features in MCAE Systems", *CG & A*, 8(2), marzo de 1988, págs. 51-59.
- GOUR71 Gouraud, H., "Continuous Shading of Curved Surfaces", *IEEE Trans. on Computers*, C-20(6), junio de 1971, págs. 623-629. También en FREE80, págs. 302-308.
- GREE87 Greenstein, J. y Arnaud, L., "Human Factors Aspects of Manual Computer Input Devices", en Salvendy, G., editor, *Handbook of Human Factors*, Wiley, Nueva York, 1987, págs. 1450-1489.
- GREG66 Gregory, R. L., *Eye and Brain—The Psychology of Seeing*, McGraw-Hill, Nueva York, 1966.
- GREG70 Gregory, R. L., *The Intelligent Eye*, McGraw-Hill, Londres, 1970.
- GSPC77 Graphics Standards Planning Committee, "Status Report of the Graphics Standards Planning Committee", *Computer Graphics*, 11, 1977.
- GSPC79 Graphics Standards Planning Committee, "Status Report of the Graphics Standards Planning Committee", *Computer Graphics*, 13(3), agosto de 1979.
- HAEU76 Haeusing, M., "Color Coding of Information on Electronic Displays", en *Proceedings of the Sixth Congress of the International Ergonomics Association*, 1976, págs. 210-217.
- HAGE86 Hagen, M., *Varieties of Realism*, Cambridge University Press, Cambridge, Inglaterra, 1986.
- HAIN89 Haines, E., "Essential Ray Tracing Algorithms", en Glassner, A. S., editor, *An Introduction to Ray Tracing*, Academic Press, Londres, 1989, págs. 33-77.
- HALL89 Hall, R., *Illumination and Color in Computer Generated Imagery*, Springer-Verlag, Nueva York, 1989.
- HAML77 Hamlin, G., Jr. y Gear, C. W., "Raster-Scan Hidden Surface Algorithm Techniques", *SIGGRAPH 77*, págs. 206-213. También en FREE80, págs. 264-271.
- HANR83 Hanrahan, P., "Ray Tracing Algebraic Surfaces", *SIGGRAPH 83*, págs. 83-90.
- HANR89 Hanrahan, P., "A Survey of Ray-Surface Intersection Algorithms", en Glassner, A. S., editor, *An Introduction to Ray Tracing*, Academic Press, Londres, 1989, págs. 79-119.

- HE92 He, X., Heynen, P., Phillips, R., Torrance, K., Salesin, D. y Greenberg, D., "A Fast and Accurate Light Reflection Model", *SIGGRAPH 92*, págs. 253-254.
- HECK84 Heckbert, P. S. y Hanrahan, P., "Beam Tracing Polygonal Objects", *SIGGRAPH 84*, págs. 119-127.
- HECK86a Heckbert, P. S., "Filtering by Repeated Integration", *SIGGRAPH 86*, págs. 315-321.
- HECK86b Heckbert, P. S., "Survey of Texture Mapping", *CG & A*, 6(11), noviembre de 1986, págs. 56-67.
- HEDG82 Hedgley, D. R., Jr., *A General Solution to the Hidden-Line Problem*, NASA Reference Publication 1085, NASA Scientific and Technical Information Branch, 1982.
- HERO76 Herot, C., "Graphical Input Through Machine Recognition of Sketches", *SIGGRAPH 76*, págs. 97-102.
- HIRS70 Hirsch, R., "Effects of Standard vs. Alphabetical Keyboard Formats on Typing Performance", *Journal of Applied Psychology*, 54, diciembre de 1970, págs. 484-490.
- HODG85 Hodges, L. y McAllister, D., "Stereo and Alternating-Pair Techniques of Display of Computer-Generated Images", *CG & A*, 5(9), septiembre de 1985, págs. 38-45.
- HOFF61 Hoffman, K. y Kunze, R., *Linear Algebra*, Prentice-Hall, Englewood Cliffs, Nueva Jersey, 1961.
- HUGH89 Hughes, J., *Integer and Floating-Point Z-Buffer Resolution*, informe técnico, Department of Computer Science, Brown University, Providence, Rhode Island, 1989.
- HUNT78 Hunter, G. M., *Efficient Computation and Data Structures for Graphics*, tesis doctoral, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, Nueva Jersey, 1978.
- HUNT79 Hunter, G. M. y Steiglitz, K., "Operations on Images Using Quad Trees", *IEEE Trans. Pattern Anal. Mach. Intell.*, 1(2), abril de 1979, págs. 145-153.
- HUNT87 Hunt, R. W., *The Reproduction of Colour*, 4a. ed., Fountain Press, Tolworth, Inglaterra, 1987.
- HUTC86 Hutchins, E., Hollan, J. y Norman, D., "Direct Manipulation Interfaces", en Norman, D. y Draper, S., editores, *User Centered System Design*, Erlbaum, Hillsdale, Nueva Jersey, 1986, págs. 87-124.
- IES87 Illuminating Engineering Society, Nomenclature Committee, *ANSI/IES RP-16-1986: American National Standard: Nomenclature and Definitions for Illuminating Engineering*, Illuminating Engineering Society of North America, Nueva York, 1987.
- INGA81 Ingalls, D., "The SmallTalk Graphics Kernel", *BYTE*, 6(8), agosto de 1981.
- INTE85 Interaction Systems, Inc., *TK-1000 Touch System*, Interaction Systems, Inc., Newtonville, Massachusetts, 1985.
- INTE88 International Standards Organization, *International Standard Information Processing Systems—Computer Graphics—Graphical Kernel System for Three Dimensions (GKS-3D) Functional Description*, ISO Document Number 8805:1988(E), American National Standards Institute, Nueva York, 1988.
- JACK64 Jacks, E., "A Laboratory for the Study of Man-Machine Communication", en *FJCC 64*, AFIPS, Montvale, Nueva Jersey, 1964, págs. 343-350.

- JACK80 Jackins, C. y Tanimoto, S. L., "Oct-Trees and Their Use in Representing Three-Dimensional Objects", *CGIP*, 14(3), noviembre de 1980, págs. 249-270.
- JARV76a Jarvis, J. F., Judice, C. N. y Ninke, W. H., "A Survey of Techniques for the Image Display of Continuous Tone Pictures on Bilevel Displays", *CGIP*, 5(1), marzo de 1976, págs. 13-40.
- JARV76b Jarvis, J. F. y Roberts, C. S., "A New Technique for Displaying Continuous Tone Images on a Bilevel Display", *IEEE Trans.*, COMM-24(8), agosto de 1976, págs. 891-898.
- JOBL78 Joblove, G. H. y Greenberg, D., "Color Spaces for Computer Graphics", *SIGGRAPH 78*, págs. 20-27.
- JOY86 Joy, K. I. y Bhetanabhotla, M. N., "Ray Tracing Parametric Surface Patches Utilizing Numerical Techniques and Ray Coherence", *SIGGRAPH 86*, págs. 279-285.
- JOY88 Joy, K., Grant, C., Max, N. y Hatfield, L., *Tutorial: Computer Graphics: Image Synthesis*, IEEE Computer Society, Washington, D.C., 1988.
- JUDD75 Judd, D. y Wyszecki, G., *Color in Business, Science, and Industry*, Wiley, Nueva York, 1975.
- JUDI74 Judice, J. N., Jarvis, J. F. y Ninke, W., "Using Ordered Dither to Display Continuous Tone Pictures on an AC Plasma Panel", *Proceedings of the Society for Information Display*, Q4 1974, págs. 161-169.
- KAJI82 Kajiya, J. T., "Ray Tracing Parametric Patches", *SIGGRAPH 82*, págs. 245-254.
- KAJI83 Kajiya, J., "New Techniques for Ray Tracing Procedurally Defined Objects", *SIGGRAPH 83*, págs. 91-102.
- KAJI85 Kajiya, J.T., "Anisotropic Reflection Models", *SIGGRAPH 85*, págs. 15-21.
- KAJI86 Kajiya, J.T., "The Rendering Equation", *SIGGRAPH 86*, págs. 143-150.
- KAPP85 Kappel, M. R., "An Ellipse-Drawing Algorithm for Raster Displays", en Earnshaw, R., editor, *Fundamental Algorithms for Computer Graphics*, NATO ASI Series, Springer-Verlag, Berlín, 1985, págs. 257-280.
- KAY79a Kay, D. S., *Transparency, Refraction and Ray Tracing for Computer Synthesized Images*, tesis de maestría, Program of Computer Graphics, Cornell University, Ithaca, Nueva York, enero de 1979.
- KAY79b Kay, D. S. y Greenberg, D., "Transparency for Computer Synthesized Images", *SIGGRAPH 79*, págs. 158-164.
- KAY86 Kay, T. L. y Kajiya, J.T., "Ray Tracing Complex Scenes", *SIGGRAPH 86*, págs. 269-278.
- KELL76 Kelly, K. y Judd, D., *COLOR—Universal Language and Dictionary of Names*, National Bureau of Standards Spec. Publ. 440, 003-003-01705-1, U.S. Government Printing Office, Washington, D.C., 1976.
- KLIN71 Klinger, A., "Patterns and Search Statistics", in Rustagi, J., editor, *Optimizing Methods in Statistics*, Academic Press, Nueva York, 1971, págs. 303-337.
- KNOW77 Knowlton, K. y Cherry, L., "ATOMS-A Three-D Opaque Molecule System for Color Pictures of Space-Filling or Ball-and-Stick Models", *Computers and Chemistry*, 1, 1977, págs. 161-166.
- KNUT87 Knuth, D., "Digital Halftones by Dot Diffusion", *ACM TOG*, 6(4), octubre de 1987, págs. 245-273.

- KORE83 Korein, J. y Badler, N., "Temporal Anti-Aliasing in Computer Generated Animation", *SIGGRAPH 83*, págs. 377-388.
- KREB79 Krebs, M. y Wolf, J., "Design Principles for the Use of Color in Displays", *Proceedings of the Society for Information Display*, 20, 1979, págs. 10-15.
- KRUE83 Krueger, M., *Artificial Reality*, Addison-Wesley, Reading, Massachusetts, 1983.
- LAID86 Laidlaw, D. H., Trumbore, W. B. y Hughes, J. F., "Constructive Solid Geometry for Polyhedral Objects", *SIGGRAPH 86*, págs. 161-170.
- LAND85 Landauer, T. y Nachbar, D., "Selection from Alphabetic and Numeric Menu Trees Using a Touch-Sensitive Screen: Breadth, Depth, and Width", en *Proceedings CHI '85 Human Factors in Computing Systems Conference*, ACM, Nueva York, 1985, págs. 73-78.
- LANE80 Lane, J., Carpenter, L., Whitted, T. y Blinn, J., "Scan Line Methods for Displaying Parametrically Defined Surfaces", *CACM*, 23(1), enero de 1980, págs. 23-34. También en BEAT82, págs. 468-479.
- LASS87 Lasseter, J., "Principles of Traditional Animation Applied to 3D Computer Animation", *SIGGRAPH 87*, págs. 35-44.
- LAYB79 Laybourne, K., *The Animation Book*, Crown, Nueva York, 1979.
- LEAF74 Leaf, C., *The Owl Who Married a Goose*, filme, National Film Board of Canada, 1974.
- LEAF77 Leaf, C., *The Metamorphosis of Mr. Samsa*, filme, National Film Board of Canada, 1977.
- LEV176 Levin, J., "A Parametric Algorithm for Drawing Pictures of Solid Objects Composed of Quadric Surfaces", *CACM*, 19(10), octubre de 1976, págs. 555-563.
- LIAN84 Liang, Y-D. y Barsky, B., "A New Concept and Method for Line Clipping", *ACM TOG*, 3(1), enero de 1984, págs. 1-22.
- LIND68 Lindenmayer, A., "Mathematical Models for Cellular Interactions in Development, Parts I and II", *J. Theor. Biol.*, 18, 1968, págs. 280-315.
- LINT89 Linton, M., Vlissides, J. y Calder, P., "Composing User Interfaces with InterViews", *IEEE Computer*, 22(2), febrero de 1989, págs. 8-22.
- MACH78 Machover, C., "A Brief Personal History of Computer Graphics", *Computer*, 11(11), noviembre de 1978, págs. 38-45.
- MAGI68 Mathematical Applications Group, Inc., "3-D Simulated Graphics Offered by Service Bureau", *Datamation*, 13(1), febrero de 1968, págs. 69.
- MAHL72 Mahl, R., "Visible Surface Algorithms for Quadric Patches", *IEEE Trans. on Computers*, C-21(1), enero de 1972, págs. 1-4.
- MAHN73 Mahnkopf, R. y Encarnaçāo, J. L., *FLAVISA—Hidden Line Algorithm for Displaying Spatial Constructs Given by Point Sets*, Technischer Bericht Nr. 148, Heinrich Hertz Institut, Berlín, 1973.
- MAMM89 Mammen, A., "Transparency and Antialiasing Algorithms Implemented with the Virtual Pixel Maps Technique", *CG & A*, 9(4), julio de 1989, págs. 43-55.
- MAND82 Mandelbrot, B., Technical Correspondence, *CACM*, 25(8), agosto de 1982, págs. 581-583.
- MANT88 Mäntyl, M., *Introduction to Solid Modeling*, Computer Science Press, Rockville, Maryland, 1988.

- MARC82 Marcus, A., "Color: A Tool for Computer Graphics Communication", en Greenberg, D., A. Marcus, A. Schmidt y V. Gorter, *The Computer Image*, Addison-Wesley, Reading, Massachusetts, 1982, págs. 76-90.
- MARK80 Markowsky, G. y Wesley, M. A., "Fleshing Out Wire Frames", *IBM Journal of Research and Development*, 24(5), septiembre de 1980, págs. 582-597.
- MARS85 Marsden, J. y Weinstein, A., *Calculus I, II, and III*, 2a. ed. Springer Verlag, Nueva York, 1985.
- MAX79 Max, N. L., "ATOMLLL: ATOMS with Shading and Highlights", *SIGGRAPH 79*, págs. 165-173.
- MAX84 Max, N. L., "Atoms with Transparency and Shadows", *CVGIP*, 27(1), julio de 1984, págs. 46-63.
- MAXW46 Maxwell, E. A., *Methods of Plane Projective Geometry Based on the Use of General Homogeneous Coordinates*, Cambridge University Press, Cambridge, Inglaterra, 1946.
- MAXW51 Maxwell, E. A., *General Homogeneous Coordinates in Space of Three Dimensions*, Cambridge University Press, Cambridge, Inglaterra, 1951.
- MAYH90 Mayhew, D., *Principles and Guidelines in User Interface Design*, Prentice-Hall, Englewood Cliffs, Nueva Jersey, 1990.
- MEAG80 Meagher, D., *OcTree Encoding: A New Technique for the Representation, Manipulation, and Display of Arbitrary 3-D Objects by Computer*, informe técnico IPL-TR-80-111, Image Processing Laboratory, Rensselaer Polytechnic Institute, Troy, Nueva York, octubre de 1980.
- MEAG82 Meagher, D., "Geometric Modeling Using OcTree Encoding", *CGIP*, 19(2), junio de 1982, págs. 129-147.
- MEIE88 Meier, B., "ACE: A Color Expert System for User Interface Design", en *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, ACM, Nueva York, 1988, págs. 117-128.
- MEYE80 Meyer, G. W. y Greenberg, D. P., "Perceptual Color Spaces for Computer Graphics", *SIGGRAPH 80*, págs. 254-261.
- MEYE88 Meyer, G. y Greenberg, D., "Color-defective Vision and Computer Graphic Displays", *CG & A*, 8(5), septiembre de 1988, págs. 28-40.
- MICH71 Michaels, S., "QWERTY Versus Alphabetical Keyboards as a Function of Typing Skill", *Human Factors*, 13(5), octubre de 1971, págs. 419-426.
- MICR89 Microsoft Corporation, *Presentation Manager*, Microsoft Corporation, Bellevue, Washington, 1989.
- MILL87 Miller, J. R., "Geometric Approaches to Nonplanar Quadric Surface Intersection Curves", *ACM TOG*, 6(4), octubre de 1987, págs. 274-307.
- MILL89 Miller, J. R., "Architectural Issues in Solid Modelers", *CG & A*, 9(5), septiembre de 1989, págs. 72-87.
- MORT85 Mortenson, M., *Geometric Modeling*, Wiley, Nueva York, 1985.
- MUNS76 Munsell Color Company, *Book of Color*, Munsell Color Company, Baltimore, Maryland, 1976.
- MURC85 Murch, G., "Using Color Effectively: Designing to Human Specifications", *Technical Communications*, Q4 1985, Tektronix Corporation, Beaverton, Oregon, 14-20.
- MUSG89 Musgrave, F.K., "Prisms and Rainbows: A Dispersion Model for Computer Graphics", en *Proceedings of Graphics Interface '89*, Londres, Ontario, 19-23 de junio de 1989, págs. 227-234.

- MYER68 Myer, T. y Sutherland, I., "On the Design of Display Processors", *CACM*, 11 (6), junio de 1968, págs. 410-414.
- MYER75 Myers, A. J., *An Efficient Visible Surface Program*, Informe a National Science Foundation, Computer Graphics Research Group, Ohio State University, Columbus, Ohio, julio de 1975.
- NAYL90 Naylor, B. F., "Binary Space Partitioning Trees as an Alternative Representation of Polytopes", *CAD*, 22(4), May 1990, págs. 250-253.
- NEID93 Neider, J., Davis, T. y Woo, M., *OpenGL Programming Guide*, Addison-Wesley, Reading, Massachusetts, 1993.
- NEWE72 Newell, M. E., Newell, R. G. y Sancha, T. L., "A Solution to the Hidden Surface Problem", en *Proceedings of the ACM National Conference 1972*, págs. 443-450. También en FREE80, págs. 236-243.
- NICH87 Nicholl, T. M., Lee, D. T. y Nicholl, R. A., "An Efficient New Algorithm for 2-D Line Clipping: Its Development and Analysis", *SIGGRAPH 87*, págs. 253-262.
- NICO77 Nicodemus, F. E., Richmond, J. C., Hsia, J. J., Ginsberg, I. W. y Limperis, T., *Geometrical Considerations and Nomenclature for Reflectance*, NBS Monograph 160, U. S. Department of Commerce, Washington D.C., octubre de 1977.
- NISH85 Nishita, T. y Nakamae, E., "Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection", *SIGGRAPH 85*, págs. 23-30.
- NOLL67 Noll, M., "A Computer Technique for Displaying N-dimensional Hyper-objects", *CACM*, 10(8), agosto de 1967, págs. 469-473.
- NORM88 Norman, D., *The Psychology of Everyday Things*, Basic Books, Nueva York, 1988.
- OPEN89 Open Software Foundation, *OSF/MOTIF™ Manual*, Open Software Foundation, Cambridge, Massachusetts, 1989.
- OSTW31 Ostwald, W., *Colour Science*, Winsor & Winsor, Londres, 1931.
- PAIN89 Painter, J. y Sloan, K., "Antialiased Ray Tracing by Adaptive Progressive Refinement", *SIGGRAPH 89*, págs. 281-288.
- PALA88 Palay, A., Hansen, W., Kazar, M., Sherman, M., Wadlow, M., Neendorffer, T., Stern, Z., Bader, M. y Peters, T., "The Andrew Toolkit: An Overview", en *Proceedings 1988 Winter USENIX*, febrero de 1988, págs. 9-21.
- PANT91 Pantone, Inc., *PANTONE Color Formula Guide 1000*, © Pantone, Inc., 1991.
- PEAC85 Peachey, D. R., "Solid Texturing of Complex Surfaces", *SIGGRAPH 85*, págs. 279-286.
- PEIT86 Peitgen, H.-O. y Richter, P. H., *The Beauty of Fractals: Images of Complex Dynamical Systems*, Springer-Verlag, Berlín, 1986.
- PERL85 Perlin, K., "An Image Synthesizer", *SIGGRAPH 85*, págs. 287-296.
- PERR85 Perry, T. y Wallach, P., "Computer Displays: New Choices, New Tradeoffs", *IEEE Spectrum*, 22(7), julio de 1985, págs. 52-59.
- PHIG88 PHIGS+ Committee, Andries van Dam, presidente, "PHIGS+ Functional Description, Revision 3.0", *Computer Graphics*, 22(3), julio de 1988, págs. 125-218.
- PHIG92 Programmer's Hierarchical Interactive Graphics System (PHIGS) Part 4-Plus Lumière und Surfaces (PHIGS PLUS), ISO/IEC 9592-4:1992(E).

- PHIL91** Phillips, R. L., "MediaView: A General Multimedia Digital Publishing System", *CACM*, 34(7), julio de 1991, págs. 75-83.
- PHIL92** Phillips, R. L., "Opportunities for Multimedia in Education", en Cunningham, S. y Hubbald, R., *Interactive Learning Through Visualization*, Springer-Verlag, Berlín, 1992, págs. 25-35.
- PITT67** Pitteway, M. L. V., "Algorithm for Drawing Ellipses or Hyperbolae with a Digital Plotter", *Computer J.*, 10(3), noviembre de 1967, págs. 282-289.
- PIXA86** Pixar Corporation, *Luxo, Jr.*, filme, Pixar Corporation, San Rafael, California, 1986.
- PIXA88** Pixar Corporation, *The RenderMan Interface*, versión 3.0, Pixar Corporation, San Rafael, California, mayo de 1988.
- PORT79** Porter, T., "The Shaded Surface Display of Large Molecules", *SIGGRAPH 79*, págs. 234-236.
- POTM82** Potmesil, M. y Chakravarty, I., "Synthetic Image Generation with a Lens and Aperture Camera Model", *ACM TOG*, 1(2), abril de 1982, págs. 85-108.
- PRAT84** Pratt, M., "Solid Modeling and the Interface Between Design and Manufacture", *CG & A*, 4(7), julio de 1984, págs. 52-59.
- PREP85** Preparata, F. P. y Shamos, M. I., *Computational Geometry: An Introduction*, Springer-Verlag, Nueva York, 1985.
- PRES88** Press, W. H., Flannery, B. P., Teukolsky, S. A. y Vetterling, W. T., *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge, Inglaterra, 1988.
- PRIN71** Prince, D., *Interactive Graphics for Computer Aided Design*, Addison-Wesley, Reading, Massachusetts, 1971.
- PRIT77** Pritchard, D.H., "U.S. Color Television Fundamentals—A Review", *IEEE Transactions on Consumer Electronics*, CE-23(4), noviembre de 1977, págs. 467-478.
- PRUS88** Prusinkiewicz, P., Lindenmayer, A. y Hanan, J., "Developmental Models of Herbaceous Plants for Computer Imagery Purposes", *SIGGRAPH 88*, págs. 141-150.
- PUTN86** Putnam, L. K. y Subrahmanyam, P. A., "Boolean Operations on  $n$ -Dimensional Objects", *CG & A*, 6(6), junio de 1986, págs. 43-51.
- RATL72** Ratliff, F., "Contour and Contrast", *Scientific American*, 226(6), junio de 1972, págs. 91-101. También en *BEAT82*, págs. 364-375.
- REDD78** Reddy, D. y Rubin, S., *Representation of Three-Dimensional Objects*, CMU-CS-78-113, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1978.
- REFF88** de Reffye, P., Edelin, C., Franon, J., Jaeger, M. y Puech, C., "Plant Models Faithful to Botanical Structure and Development", *SIGGRAPH 88*, págs. 151-158.
- REQU77** Requicha, A. A. G., *Mathematical Models of Rigid Solids*, memorándum técnico 28, Production Automation Project, University of Rochester, Rochester, Nueva York, 1977.
- REQU80** Requicha, A. A. G., "Representations for Rigid Solids: Theory, Methods, and Systems", *ACM Computing Surveys*, 12(4), diciembre de 1980, págs. 437-464.
- REQU84** Requicha, A. A. G., "Representation of Tolerances in Solid Modeling: Issues and Alternative Approaches", en Pickett, M. y Boyse, J., editores, *Solid Modeling by Computers*, Plenum Press, Nueva York, 1984, págs. 3-22.

- REQU85 Requicha, A. A. G. y Voelcker, H. B., "Boolean Operations in Solid Modeling: Boundary Evaluation and Merging Algorithms", *Proc. IEEE*, 73(1), enero de 1985, págs. 30-44.
- ROBE65 Roberts, L. G., *Homogeneous Matrix Representations and Manipulation of N-Dimensional Constructs*, documento MS 1405, Lincoln Laboratory, MIT, Cambridge, Massachusetts, 1965.
- ROMN69 Romney, G. W., Watkins, G. S. y Evans, D. C., "Real Time Display of Computer Generated Half-Tone Perspective Pictures", en *Proceedings 1968 IFIP Congress*, North Holland Publishing Co., 1969, págs. 973-978.
- ROSE85 Rose, C., Hacker, B., Anders, R., Wittney, K., Metzler, M., Chernicoff, S., Espinosa, C., Averill, A., Davis, B. y Howard, B., *Inside Macintosh*, I, Addison-Wesley, Reading, Massachusetts, 1985, págs. I-35-I-213.
- ROSS86 Rossignac, J. R. y Requicha, A. A. G., "Depth-Buffering Display Techniques for Constructive Solid Geometry", *CG & A*, 6(9), septiembre de 1986, págs. 29-39.
- RUBE84 Rubenstein, R. y Hersh, H., *The Human Factor—Designing Computer Systems for People*, Digital Press, Burlington, Massachusetts, 1984.
- RUBI80 Rubin, S. M. y Whitted, T., "A 3-Dimensional Representation for Fast Rendering of Complex Scenes", *SIGGRAPH 80*, págs. 110-116.
- SALV87 Salvendy, G., editor, *Handbook of Human Factors*, Wiley, Nueva York, 1987.
- SAME84 Samet, H., "The Quadtree and Related Hierarchical Data Structures", *ACM Comp. Surv.*, 16(2), junio de 1984, págs. 187-260.
- SAME89a Samet, H., "Neighbor Finding in Images Represented by Octrees", *CVGIP*, 46(3), junio de 1989, págs. 367-386.
- SAME89b Samet, H. y Webber, R., "Hierarchical Data Structures and Algorithms for Computer Graphics, Part II: Applications", *CG & A*, 8(4), julio de 1988, págs. 59-75.
- SAME90a Samet, H., *Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, Massachusetts, 1990.
- SAME90b Samet, H., *Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS*, Addison-Wesley, Reading, Massachusetts, 1990.
- SARR83 Sarraga, R. F., "Algebraic Methods for Intersections of Quadric Surfaces in GMSOLID", *CVGIP*, 22(2), mayo de 1983, págs. 222-238.
- SCHE88 Scheifler, R. W., Gettys, J. y Newman, R., *X Window System*, Digital Press, 1988.
- SCHM86 Schmucker, K., "MacApp: An Application Framework", *Byte*, 11(8), agosto de 1986, págs. 189-193.
- SCHN90 Schneider, P., "An Algorithm for Automatically Fitting Digitized Curves", *Graphics Gems*, Glassner, A., editor, Academic Press, 1990, págs. 612-626.
- SCHU69 Schumacker, R., Brand, B., Gilliland, M. y Sharp, W., *Study for Applying Computer-Generated Images to Visual Simulation*, informe técnico AFHRL-TR-69-14, NTIS AD700375, U.S. Air Force Human Resources Lab., Air Force Systems Command, Brooks AFB, Texas, septiembre de 1969.
- SCHW82 Schweitzer, D. y Cobb, E., "Scanline Rendering of Parametric Surfaces", *SIGGRAPH 82*, págs. 265-271.

- SEDE84 Sederberg, T. W. y Anderson, D. C., "Ray Tracing of Steiner Patches", *SIGGRAPH 84*, págs. 159-164.
- SEQU89 Séquin, C. H. y Smyrl, E. K., "Parameterized Ray Tracing", *SIGGRAPH 89*, págs. 307-314.
- SHER93 Sherr, S., *Electronic Displays, Second Edition*, Wiley, Nueva York, 1993.
- SHNE86 Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley, Reading, Massachusetts, 1986.
- SIEG81 Siegel, R. y Howell, J., *Thermal Radiation Heat Transfer*, 2a. ed., Hemisphere, Washington, D. C., 1981.
- SMIT78 Smith, A. R., "Color Gamut Transform Pairs", *SIGGRAPH 78*, págs. 12-19.
- SMIT79 Smith, A. R., "Tint Fill", *SIGGRAPH 79*, págs. 276-283.
- SMIT84 Smith, A. R., "Plants, Fractals and Formal Languages", *SIGGRAPH 84*, págs. 1-10.
- SNOW83 Snowberry, K., Parkinson, S. y Sisson, N., "Computer Display Menus", *Ergonomics*, 26(7), julio de 1983, págs. 699-712.
- SPAR78 Sparrow, E. M. y Cess, R. D., *Radiation Heat Transfer*, Hemisphere, Washington, D. C., 1978.
- SPRO82 Sproull, R. F., "Using Program Transformations to Derive Line-Drawing Algorithms", *ACM TOG*, 1(4), octubre de 1982, págs. 259-273.
- STAU78 Staudhammer, J., "On Display of Space Filling Atomic Models in Real Time", *SIGGRAPH 78*, págs. 167-172.
- STON88 Stone, M., Cowan, W. y Beatty, J., "Color Gamut Mapping and the Printing of Digital Color Images", *ACM TOG*, 7(3), octubre de 1988, págs. 249-292.
- STOV82 Stover, H., "True Three-Dimensional Display of Computer Data", en *Proceedings of SPIE*, 367, agosto de 1982, págs. 141-144.
- STRO91 Stroustrup, B., *The C + + Programming Language*, 2a. ed., Addison-Wesley, Reading, Massachusetts, 1991.
- SUN89 Sun Microsystems, *OPEN LOOK Graphical User Interface*, Sun Microsystems, Mountain View, California, 1989.
- SUTH63 Sutherland, I. E., "Sketchpad: A Man-Machine Graphical Communication System", en SJCC, Spartan Books, Baltimore, Maryland, 1963.
- SUTH74a Sutherland, I. E., Sproull, R. F. y Schumacker, R. A., "A Characterization of Ten Hidden-Surface Algorithms", *ACM Computing Surveys*, 6(1), marzo de 1974, págs. 1-55. También en BEAT82, págs. 387-441.
- SUTH74b Sutherland, I. E. y Hodgman, G. W., "Reentrant Polygon Clipping", *CACM*, 17(1), enero de 1974, págs. 32-42.
- TAMM82 Tarminnen, M. y Sulonen, R., "The EXCELL Method for Efficient Geometric Access to Data", en *Proc. 19th ACM IEEE Design Automation Conf.*, Las Vegas, 14-16 de junio de 1982, págs. 345-351.
- TANN85 Tannas, L. Jr., editor, *Flat-Panel Displays and CRTs*, Van Nostrand Reinhold, Nueva York, 1985.
- THIB87 Thibault, W. C. y Naylor, B. F., "Set Operations on Polyhedra Using Binary Space Partitioning Trees", *SIGGRAPH 87*, págs. 153-162.
- THOM86 Thomas, S. W., "Dispersive Refraction in Ray Tracing", *The Visual Computer*, 2(1), enero de 1986, págs. 3-8.

- TILL83 Tiller, W., "Rational B-Splines for Curve and Surface Representation", *CG & A*, 3(6), septiembre de 1983, págs. 61-69.
- TILO80 Tilove, R. B., "Set Membership Classification: A Unified Approach to Geometric Intersection Problems", *IEEE Trans. on Computers*, C-29(10), octubre de 1980, págs. 847-883.
- TORR66 Torrance, K. E., Sparrow, E. M. y Birkebak, R. C., "Polarization, Directional Distribution, and Off-Specular Peak Phenomena in Light Reflected from Roughened Surfaces", *J. Opt. Soc. Am.*, 56(7), julio de 1966, págs. 916-925.
- TORR67 Torrance, K. y Sparrow, E. M., "Theory for Off-Specular Reflection from Roughened Surfaces", *J. Opt. Soc. Am.*, 57(9), septiembre de 1967, págs. 1105-1114.
- TOTH85 Toth, D. L., "On Ray Tracing Parametric Surfaces", *SIGGRAPH 85*, págs. 171-179.
- TURN84 Turner, J.A., *A Set-Operation Algorithm for Two and Three-Dimensional Geometric Objects*, Architecture and Planning Research Laboratory, College of Architecture, University of Michigan, Ann Arbor, Michigan, agosto de 1984.
- ULIC87 Ulichney, R., *Digital Halftoning*, MIT Press, Cambridge, Massachusetts, 1987.
- VANA84 Van Aken, J. R., "An Efficient Ellipse-Drawing Algorithm", *CG & A*, 4(9), septiembre de 1984, págs. 24-35.
- VANA85 Van Aken, J. R. y Novak, M., "Curve-Drawing Algorithms for Raster Displays", *ACM TOG*, 4(2), abril de 1985, págs. 147-169.
- VOSS87 Voss, R., "Fractals in Nature: Characterization, Measurement, and Simulation", en *Course Notes 15 for SIGGRAPH 87*, Anaheim, California, julio de 1987.
- WACO93 Wacom Technology Corporation, *Wacom Cordless Digitizer Specification Sheet*, Wacom Technology Corporation, Vancouver, Washington, 1993.
- WARE87 Ware, C. y Mikaelian, J., "An Evaluation of an Eye Tracker as a Device for Computer Input", en *Proceedings of CHI + GI 1987*, ACM, Nueva York, págs. 183-188.
- WARN69 Warnock, J., *A Hidden-Surface Algorithm for Computer Generated Half-Tone Pictures*, informe técnico TR 4-15, NTIS AD-753 671, Computer Science Department, University of Utah, Salt Lake City, Utah, junio de 1969.
- WARN83 Warn, D. R., "Lighting Controls for Synthetic Images", *SIGGRAPH 83*, págs. 13-21.
- WATK70 Watkins, G. S., *A Real Time Visible Surface Algorithm*, tesis doctoral, informe técnico UTEC-CSc-70-101, NTIS AD-762 004, Computer Science Department, University of Utah, Salt Lake City, Utah, junio de 1970.
- WEGH84 Weghorst, H., Hooper, G. y Greenberg, D. P., "Improved Computational Methods for Ray Tracing", *ACM TOG*, 3(1), enero de 1984, págs. 52-69.
- WEIL77 Weiler, K. y Atherton, P., Hidden Surface Removal Using Polygon Area Sorting, *SIGGRAPH 77*, págs. 214-222.
- WEIS66 Weiss, R. A., "BE VISION, A Package of IBM 7090 FORTRAN Programs to Draw Orthographic Views of Combinations of Plane and Quadric Surfaces", *JACM*, 13(2), abril de 1966, págs. 194-204. También en *FREE80*, págs. 203-213.

- WESL81 Wesley, M. A. y Markowsky, G., "Fleshing Out Projections", *IBM Journal of Research and Development*, 25(6), noviembre de 1981, págs. 934-954.
- WHIT80 Whitted, T., "An Improved Illumination Model for Shaded Display", *CACM*, 23(6), junio de 1980, págs. 343-349.
- WHIT84 Whitton, M., "Memory Design for Raster Graphics Displays", *CG & A*, 4(3), marzo de 1984, págs. 48-65.
- WITK88 Witkin, A. y Kass, M., "Spacetime Constraints", *SIGGRAPH 88*, págs. 159-168.
- WOLF87 Wolf, C. y Morel-Samuels, R., "The Use of Hand-Drawn Gestures for Text Editing", *International Journal of Man-Machine Studies*, 27(1), julio de 1987, págs. 91-102.
- WOLF90 Wolff, L. y Kurlander, D., "Ray Tracing with Polarization Parameters", *CG & A*, 10(6), noviembre de 1990, págs. 44-55.
- WOLF91 Wolfram, S., *Mathematica: A System for Doing Mathematics by Computer*, 2a. ed., Addison-Wesley, Reading, Massachusetts, 1991.
- WOO85 Woo, T., "A Combinatorial Analysis of Boundary Data Structure Schemata", *CG & A*, 5(3), marzo de 1985, págs. 19-27.
- WOON71 Woon, P. Y. y Freeman, H., "A Procedure for Generating Visible-Line Projections of Solids Bounded by Quadric Surfaces", en *IFIP 1971*. North-Holland Pub. Co., Amsterdam, 1971, págs. 1120-1125. También en FREE80, págs. 230-235.
- WU87 Wu, X. y Rokne, J. G., "Double-Step Incremental Generation of Lines and Circles", *CVGIP*, (37), 1987, págs. 331-334.
- WYL167 Wylie, C., Romney, G. W., Evans, D. C. y Erdahl, A. C., "Halftone Perspective Drawings by Computer", *FJCC 67*, Thompson Books, Washington, D. C., 1967, págs. 49-58.
- WYSZ82 Wyszecki, G. y Stiles, W., *Color Science: Concepts and Methods, Quantitative Data and Formulae*, 2a. ed., Wiley, Nueva York, 1982.
- WYVI88 Wyvill, B., "The Great Train Robbery", *ACM SIGGRAPH 88 Electronic Theater and Video Review*, 26, 1988.
- WYVI90 Wyvill, B., "Symmetric Double Step Line Algorithm", *Graphics Gems*, A. Glassner, editor, Academic Press, 1990, págs. 101-104.
- ZDON90 Zdonik, S. B. y Maier, D., *Readings in Object-Oriented Database Systems*, Morgan Kaufmann, San Mateo, California, 1990.
- ZIMM87 Zimmerman, T., Lanier, J., Blanchard, C., Bryson, S. y Harvill, Y., "A Hand Gesture Interface Device", en *Proceedings of the CHI + GI 1987 Conference*, ACM, Nueva York, págs. 189-192.

# Vocabulario técnico bilingüe\*

<b>2-variedades</b>	
2-manifolds	
<b>accionador</b>	
trigger	
<b>aceptación trivial</b>	
trivial acceptance	
<b>acercamiento/alejamiento</b>	
zoom	
<b>acorde</b>	
chord	
<b>actualización de pantallas por lotes</b>	
batch screen updates	
<b>administrador de ventanas</b>	
window manager	
<b>ajuste de curvas</b>	
curve fitting	
<b>ajuste fino</b>	
tweaking	
<b>alambrado</b>	
wireframe	
<b>aletas</b>	
flaps	
<b>algoritmo de línea de barrido</b>	
scan-line algorithm	
<b>algoritmo de ordenamiento por profundidad</b>	
depth-sort algorithm	
<b>algoritmo de recorte de líneas</b>	
line-clipping algorithm	
<b>algoritmo de recorte de polígonos</b>	
polygon-clipping algorithm	
<b>algoritmo de z-buffer (memoria de profundidad z)</b>	
z-buffer algorithm	
<b>algoritmo del pintor</b>	
painter's algorithm	
<b>algoritmos de prioridad de listas</b>	
list-priority algorithms	
<b>algoritmos de subdivisión de área</b>	
area subdivision algorithms	
<b>alineación por la derecha</b>	
right justification	
<b>alineación</b>	
alignment	
<b>almacenamiento central de estructuras (CSS)</b>	
central structure storage	
<b>almacenamiento de lista de presentación</b>	
display list storage	

\* Ante la falta de un lenguaje estandarizado en castellano para las ciencias de la computación, se ha elaborado el presente vocabulario con la traducción que hemos dado en este libro a los principales términos de la versión original en inglés. Esta labor se verá compensada por el servicio que pueda prestar al lector. (*N. del E.*)

- almacenamiento local de segmentos**  
segment storage, local
- alumbrado**  
lighting
- analizador diferencial digital**  
digital differential analyzer
- ancho de banda**  
bandwidth
- ancla**  
anchor
- ángulo de pantalla**  
screen angle
- ángulo sólido**  
solid angle
- animación**  
animation
- aplicación**  
application
- árbol binario para partición de espacio (BSP)**  
binary space-partitioning (BSP) tree
- árbol de cuadrantes**  
quadtree
- árbol de octantes**  
octree
- árboles**  
trees
- área de superficie escorizada**  
foreshortened surface area
- área de vista**  
viewport
- armonía de colores**  
color harmony
- artefacto de discretización temporal**  
temporal aliasing
- artefacto de discretización**  
aliasing
- asideros para la interacción con el usuario**  
handles for user interaction
- atenuación atmosférica**  
atmospheric attenuation
- atributos**  
attributes
- autocompletación de mandatos**  
autocomplete, command
- autoocclusión**  
self-occlusion
- autosimilitud**  
self-similarity
- bandas de Mach**  
Mach bands
- barrido**  
sweep, scan
- barrido aleatorio**  
random scan
- barrido entrelazado**  
interlaced scan
- bloque de ejemplares**  
instance block
- bobinas de desviación**  
deflection coils
- bola rastreadora**  
trackball
- borrosidad de movimiento**  
motion blur
- brillantez**  
brightness
- B-splines no uniformes, no racionales**  
nonuniform, nonrational B-splines
- buffer de elementos**  
item buffer
- C iluminante**  
illuminant C
- caja acotante**  
bounding box
- cálculo de tramos**  
span calculation
- cámara sintética**  
synthetic camera
- campo de valor**  
value field
- cañón de electrones**  
electron gun
- capacidad de direccionamiento**  
addressability
- carácter descendente**  
descender, character
- carácter**  
character
- centro de proyección (COP)**  
center of projection (COP)
- centro de ventana (CW)**  
center of window (CW)
- cilindro generalizado**  
generalized cylinder
- claridad**  
lightness
- codificación de colores**  
color coding
- codificación**  
coding
- coeficiente de reflexión ambiental**  
ambient reflection coefficient
- coeficiente de transmisión**  
transmission coefficient

- coherencia de aristas**  
edge coherence
- coherencia**  
coherence
- cola de eventos**  
event queue
- color no espectral**  
nonspectral color
- colores complementarios**  
complementary colors
- colores primarios aditivos**  
additive color primaries
- colores primarios CIE**  
CIE primaries
- colores primarios sustractivos**  
subtractive color primaries
- colorimetría**  
colorimetry
- combinación lineal**  
linear combination
- compatibilidad estímulo-respuesta (E-R)**  
stimulus-response (S-R) compatibility
- composición de transformación**  
transformation composition
- composición**  
composition
- compresión y estiramiento**  
squash and stretch
- conectividad**  
connectivity
- conjunto de Julia-Fatou**  
Julia-Fatou set
- conjunto de Mandelbrot**  
Mandelbrot set
- conjunto de nombres**  
name set
- conjunto de opciones**  
choice set
- conjuntos de herramientas para interacción**  
interaction toolkits
- comutador de pie**  
foot switch
- comutatividad**  
commutativity
- conos**  
cones
- conos receptores en el ojo**  
cone receptors in eye
- continuidad geométrica**  
geometric continuity
- continuidad paramétrica**  
parametric continuity
- contorneado**  
contouring
- control de animación**  
animation control
- control local**  
local control
- controlador de vídeo**  
video controller
- controles de iluminación de Warn**  
Warn lighting controls
- convergencia de un haz de electrones**  
convergence of electron beam
- conversión entre modelos de colores**  
conversion between color models
- coordenadas de cromaticidad**  
chromaticity coordinates
- coordenadas de proyección normalizadas (NPC)**  
normalized projection coordinates (NPC)
- coordenadas homogéneas**  
homogeneous coordinates
- coordinación visual-motriz**  
eye-hand coordination
- copo de nieve de Von Koch**  
Von Koch snowflake
- coprocesador de pantalla**  
display coprocessor
- corrección gama**  
gamma correction
- correlación**  
correlation
- correspondencia de colores**  
color map
- correspondencia de patrones**  
pattern mapping
- correspondencia de texturas**  
texture mapping
- correspondencia ventana-área de vista**  
window-to-viewport mapping
- creación de cuadros intermedios**  
inbetweening
- croma**  
chroma
- cromaticidad**  
chromaticity
- CRT de dos niveles**  
bilevel CRT
- cuadro clave**  
key-frame
- cursor de texto**  
text cursor
- curva cúbica**  
cubic curve
- curvas cúbicas paramétricas**  
parametric cubic curves

<b>curvas de Bézier</b>	Bézier curves
<b>curvas de Hermite</b>	Hermite curves
<b>curvas de spline B</b>	B-spline curves
<b>curvas de B-spline uniformes, no racionales</b>	uniform nonrational B-spline curves
<b>curvas polinomiales paramétricas</b>	parametric polynomial curves
<b>deficiencia en la percepción de colores</b>	color deficient
<b>descomposición en celdas</b>	cell decomposition
<b>desechar caras posteriores</b>	back-face culling
<b>desplazamiento</b>	scroll
<b>detalle de superficie</b>	surface detail
<b>determinación de la visibilidad</b>	visibility determination
<b>determinación de líneas ocultas</b>	hidden-line determination
<b>determinación de líneas visibles</b>	visible-line determination
<b>determinación de superficies visibles</b>	visible-surface determination
<b>determinante</b>	determinant
<b>diagrama de cromaticidad</b>	chromaticity diagram
<b>diagrama de transición de estado</b>	state-transition diagram
<b>dibujo de líneas</b>	line drawing
<b>diferencia apenas perceptible en colores</b>	just-noticeable color difference
<b>diferencias de segundo orden</b>	second-order differences
<b>difusión de errores</b>	error diffusion
<b>digitalizar</b>	digitize
<b>digitalizador</b>	scanner
<b>dimensión fractal</b>	fractal dimension
<b>diodo emisor de luz (LED)</b>	light-emitting diode (LED)
<b>dirección de proyección (DOP)</b>	direction of projection (DOP)
<b>dirección de reflexión</b>	direction of reflection
<b>direccionalamiento de pantalla por matriz</b>	matrix addressing of display
<b>dirigido por eventos</b>	event-driven
<b>discretización</b>	scan conversion
<b>diseño asistido por computador (CAD)</b>	computer-aided design (CAD)
<b>disparidad binocular</b>	binocular disparity
<b>dispersión</b>	dispersion
<b>dispositivo acoplado por carga (CCD)</b>	charge-coupled device (CCD)
<b>dispositivo lógico de entrada</b>	logical input device
<b>dispositivo lógico de opciones</b>	choice logical device
<b>dispositivo lógico localizador</b>	locator logical device
<b>dispositivo lógico valuador</b>	valuator logical device
<b>dispositivos de entrada</b>	input devices
<b>dispositivos de presentación por barrido</b>	display devices, raster
<b>distancia al plano anterior</b>	front distance
<b>distancia al plano posterior</b>	back distance
<b>distancia entre puntos</b>	interdot distance
<b>distribución de energía</b>	energy distribution
<b>distribución energética espectral</b>	spectral energy distribution
<b>ducto de entrada</b>	input pipeline
<b>ducto de generación de imágenes</b>	rendering pipeline
<b>ducto de salida</b>	output pipeline
<b>ecuación de generación de imágenes</b>	rendering equation
<b>ecuaciones implícitas</b>	implicit equations
<b>ejemplar</b>	instance
<b>ejemplares de primitivas</b>	primitive instancing

<b>elemento de textura</b>	<b>familia tipográfica (fuente)</b>
texel	font
<b>elemento de volumen</b>	<b>fantasma</b>
voxel	sprite
<b>eliminación de artefacto de discretización</b>	<b>faro</b>
antialiasing	floodlight
<b>eliminación de superficies ocultas</b>	<b>filtro de caja</b>
hidden-surface elimination	box filter
<b>embaldosado</b>	<b>filtro de cono</b>
tiling	cone filter
<b>enfoque</b>	<b>flujo luminoso</b>
focus	flux
<b>entrada lenta/salida lenta</b>	<b>fondo</b>
slow-in/slow-out	background
<b>entrada</b>	<b>fósforo</b>
input	phosphor
<b>envolvente convexa</b>	<b>fotorrealismo</b>
convex hull	photorealism
<b>ergonomía</b>	<b>frecuencia de fusión</b>
ergonomics	fusion frequency
<b>escalamiento de imágenes</b>	<b>fuente luminosa puntual</b>
image scaling	point light source
<b>escalamiento</b>	<b>función de eficiencia luminosa</b>
scaling	luminous efficiency function
<b>escalonamiento</b>	<b>función de plantilla</b>
staircasing	template function
<b>escorzo de perspectiva</b>	<b>función de ponderación</b>
perspective foreshortening	weighting function
<b>escritura de rectángulo</b>	<b>funciones de equivalencia de colores</b>
rectangle write	color matching functions
<b>espaciado entre caracteres</b>	<b>funciones de mezclado</b>
kerning	blending functions
<b>espacio de direcciones</b>	<b>funciones de respuesta espectral</b>
address space	spectral-response functions
<b>estación de trabajo gráfica</b>	<b>funciones explícitas</b>
graphics workstation	explicit functions
<b>estereoscóptica</b>	<b>gama dinámica</b>
stereopsis	dynamic range
<b>estereoradián</b>	<b>gamas de colores</b>
steradian	color gamuts
<b>estilo de pincel</b>	<b>generación de imágenes</b>
pen style	rendering
<b>etiqueta</b>	<b>generación de trama</b>
label	rasterization
<b>evento</b>	<b>generador de barrido de trama</b>
event	raster-scan generator
<b>extensión de texto</b>	<b>género de un poliedro</b>
text-extent	genus of a polyhedron
<b>extensión</b>	<b>geometría sólida constructiva (CSG)</b>
extent	constructive solid geometry (CSG)
<b>extensiones geométricas</b>	<b>grabadora de película fotográfica</b>
geometric extents	film recorder
<b>factores humanos</b>	
human factors	

<b>graficador</b>	irradiancia de imágenes
plotter	image irradiance
<b>gráficos</b>	<b>jerarquía de imágenes</b>
graphics	picture hierarchy
<b>gráficos de modo inmediato</b>	<b>lanzamiento de rayos</b>
immediate-mode graphics	ray casting
<b>gráficos de modo retenido</b>	<b>lenguajes gráficos</b>
retained-mode graphics	graphical languages
<b>graftales</b>	<b>lienzo</b>
graftals	canvas
<b>gramáticas L</b>	<b>línea base para texto</b>
L-grammars	text baseline
<b>“Haz lo que quiero decir” (DWIM)</b>	<b>línea elástica</b>
Do what I mean (DWIM)	rubberband
<b>hipótesis del objeto</b>	<b>líneas de trama</b>
object hypothesis	raster lines
<b>huella</b>	<b>líneas gruesas</b>
footprint	thick lines
<b>ilusión de escalera de Schröder</b>	<b>localizador</b>
Schröder stairway illusion	locator
<b>ilusión de rueda de carreta</b>	<b>longitud de onda dominante</b>
wagon wheel illusion	dominant wavelength
<b>ilusión del cubo de Necker</b>	<b>longitud de un vector</b>
Necker cube illusion	length, of a vector
<b>implantación de proyección</b>	<b>luminancia</b>
projection implementation	luminance
<b>impresora</b>	<b>luz acromática</b>
printer	achromatic light
<b>independencia del dispositivo</b>	<b>luz ambiental</b>
device-independence	ambient light
<b>indicador de profundidad</b>	<b>luz reflejada</b>
depth cueing	reflected light
<b>inhibición lateral</b>	<b>maestro</b>
lateral inhibition	master
<b>integración espacial</b>	<b>malla cuadrilateral</b>
spatial integration	quadrilateral mesh
<b>intensidad percibida de la luz</b>	<b>malla de control</b>
perceived intensity of light	control grid
<b>interacción “oprime y arrastre”</b>	<b>malla poligonal</b>
click and drag interaction	polygon mesh
<b>interacción dirigida por muestras</b>	<b>manejo de interacción</b>
sample-driven interaction	interaction handling
<b>interfaz con el usuario</b>	<b>manipulación directa</b>
user interface	direct manipulation
<b>interpolación de colores</b>	<b>manojo de atributos</b>
color interpolation	attribute bundle
<b>intersección externa</b>	<b>manufactura asistida por computador (CAM)</b>
intersection, external	computer-aided manufacturing (CAM)
<b>inversa</b>	<b>mapa de bits</b>
inverse	bitmap
<b>irradiancia</b>	<b>mapa de pixeles</b>
irradiance	pixmap

**marca**  
marker  
**mascarilla de botón**  
button mask  
**mascarilla de tensión plana**  
flat tension mask  
**mascarillas**  
masks  
**matiz**  
shade  
**matriz base**  
basis matrix  
**matriz de proyección**  
projection matrix  
**matriz de transformación global**  
global transformation matrix  
**matriz de transformación local**  
local transformation matrix  
**matriz de transformación**  
transformation matrix  
**matriz geométrica**  
geometry matrix  
**matriz ortogonal especial**  
special orthogonal matrix  
**medida**  
measure  
**medios tonos**  
halftoning  
**memoria caché de familias tipográficas**  
font cache  
**memoria de acceso aleatorio para vídeo**  
video random-access memory  
**memoria de objetos**  
object buffer  
**memoria doble**  
double-buffering  
**memoria gráfica**  
frame buffer  
**menú descendente**  
 pulldown menu  
**metaarchivo**  
metafile  
**métodos de radiosidad**  
radiosity methods  
**métodos incrementales**  
incremental methods  
**mezclado de video**  
video mixing  
**microfacetas**  
microfacets  
**modelado de objetos**  
object modeling  
**modelado de sólidos**  
solid modeling

**modelado de superficies**  
surface modeling  
**modelado geométrico**  
geometric modeling  
**modelado molecular**  
molecular modeling  
**modelo de colores HSV de cono hexagonal**  
hexcone HSV color model  
**modelos de colores**  
color models  
**modelos fractales**  
fractal models  
**modelos gramaticales**  
grammar-based models  
**modo de escritura**  
write mode  
**modo de procesamiento**  
processing mode  
**monocromático**  
monochrome  
**muestra**  
sample  
**muestreo de área**  
area sampling  
**mundo virtual**  
virtual world  
**normal a la superficie**  
surface normal  
**nudo**  
knot  
**numeración de ocupación espacial**  
spatial occupancy enumeration  
**objeto autoluminoso**  
self-luminous object  
**objeto estandarizado**  
standardized object  
**opacidad**  
opacity  
**operación de barrido**  
raster operation, RasterOp  
**operaciones booleanas regularizadas de conjuntos**  
regularized Boolean set operations  
**operaciones de visualización**  
viewing operations  
**operadores de Euler**  
Euler operators  
**palanca de mandos**  
joystick  
**paleta**  
palette

<b>pantalla caligráfica</b>	calligraphic display
<b>pantalla de cristal líquido (LCD)</b>	liquid-crystal display (LCD)
<b>pantalla de dos niveles</b>	bilevel display
<b>pantalla de plasma</b>	plasma panel display
<b>pantalla electroluminiscente (EL)</b>	electroluminescent (EL) display
<b>pantalla montada en la cabeza</b>	head-mounted display
<b>pantalla sensible al tacto</b>	touch panel
<b>paquete de gráficos de barrido</b>	raster graphics package
<b> parche de superficie</b>	surface patch
<b>parches de superficie polinomial paramétrica de dos variables</b>	parametric bivariate polynomial surface patches
<b>parpadeo</b>	flicker
<b>partición espacial</b>	spatial partitioning
<b>paso</b>	pitch
<b>patrón</b>	pattern
<b>patrón de medios tonos</b>	halftone pattern
<b>persistencia</b>	persistence
<b>perspectiva aérea</b>	aerial perspective
<b>pintura</b>	painting
<b>plano de proyección</b>	projection plane
<b>plano de recorte anterior</b>	hither clipping plane
<b>plano de recorte posterior</b>	yon clipping plane
<b>plantilla</b>	template
<b>poda</b>	pruning
<b>polígonos de astilla</b>	sliver polygons
<b>polilínea</b>	polyline
<b>polinomio continuo por trozos</b>	piecewise continuous polynomial
<b>posición actual (CP)</b>	current position (CP)
<b>precisión de imagen</b>	image-precision
<b>precisión del objeto</b>	object-precision
<b>primitivas de salida</b>	output primitives
<b>prioridad de dibujo</b>	display priority
<b>procesador de imágenes de barrido</b>	raster image processor
<b>proceso de vista</b>	viewing process
<b>producto punto</b>	dot product
<b>profundidad de campo</b>	depth of field
<b>programa de dibujo</b>	display program
<b>propiedades materiales</b>	material properties
<b>proyección de perspectiva</b>	perspective projection
<b>proyección geométrica plana</b>	planar geometric projection
<b>proyección paralela</b>	parallel projection
<b>puesta en escena</b>	staging
<b>punteo agrupado ordenado por puntos</b>	clustered dot-ordered dither
<b>punteo ordenado</b>	ordered dither
<b>punto de evaluación</b>	point of evaluation
<b>punto de máximo brillo</b>	highlight
<b>punto de referencia de proyección (PRP)</b>	projection reference point (PRP)
<b>puntos de control múltiples</b>	multiple control points
<b>puntos de fuga</b>	vanishing points
<b>pureza de excitación</b>	excitation purity
<b>radiancia</b>	radiance
<b>radiosidad</b>	radiosity
<b>raíz</b>	root

<b>ratón</b>	mouse
<b>rayo</b>	ray
<b>realidad artificial</b>	artificial reality
<b>realimentación</b>	feedback
<b>rechazo trivial</b>	trivial rejection
<b>reconocimiento de voz</b>	voice recognition
<b>reconocimiento del habla</b>	speech recognition
<b>recorrido de dibujo</b>	display traversal
<b>recorte de polígonos</b>	polygon clipping
<b>recorte de profundidad</b>	depth clipping
<b>recorte por códigos de región</b>	outcode clipping
<b>rectángulo de recorte</b>	clip rectangle
<b>recuadro de diálogo</b>	dialogue box
<b>red de estructuras</b>	structure network
<b>refinamiento progresivo</b>	progressive refinement
<b>reflectividad bidireccional</b>	bidirectional reflectivity
<b>reflector</b>	spotlight
<b>reflexión de Lambert</b>	Lambertian reflection
<b>reflexión difusa</b>	diffuse reflection
<b>reflexión especular</b>	specular reflection
<b>reflexión interna total</b>	total internal reflection
<b>refrescamiento</b>	refresh
<b>regla de paridad impar</b>	odd-parity rule
<b>regla del paralelogramo</b>	parallelogram rule
<b>reglas de utilización de colores</b>	color usage rules
<b>rellenado con patrones</b>	pattern filling
<b>reparación de daños</b>	damage repair
<b>representación B</b>	B-rep
<b>representación de frontera</b>	boundary representation
<b>representación de modelado de sólidos</b>	solid modeling representation
<b>representación de sistemas de coordenadas</b>	coordinate-system representation
<b>representación paramétrica</b>	parametric representation
<b>representaciones de la partición espacial</b>	spatial partitioning representations
<b>resolución espacial</b>	spatial resolution
<b>restricciones</b>	constraints
<b>retícula cúbica</b>	cuberille
<b>retorno vertical</b>	vertical retrace
<b>revisión de región</b>	region checks
<b>rueda de reencarnación</b>	wheel of reincarnation
<b>saturación</b>	saturation
<b>segmentos</b>	segments
<b>selección</b>	pick
<b>selección de tarea de interacción</b>	select interaction task
<b>selección jerárquica de objetos</b>	hierarchical object selection
<b>selección por menús jerárquicos</b>	hierarchical menu selection
<b>serrado</b>	jaggies
<b>sesgo</b>	shear
<b>seudorealismo</b>	pseudorealism
<b>simulador de vuelo</b>	flight simulator
<b>sistema de clasificación de colores</b>	color-order system
<b>sistema de coordenadas</b>	coordinate system
<b>sistema de coordenadas de mundo</b>	world-coordinate system
<b>sistema de proyección de válvulas de luz</b>	light valve projection system

<b>sistema gráfico</b>	graphical system	<b>tarea de interacción para rotación</b>
graphics system	rotation interaction task	
<b>sistemas multimedia</b>	multimedia systems	<b>tarea espacial</b>
multimedia	spatial task	
<b>sombreado</b>	shading	<b>tasa de refrescamiento</b>
shading	refresh rate	
<b>sondeo</b>	polling	<b>tecla de función</b>
polling	function key	
<b>soporte del filtro</b>	support, of filter	<b>tecleo adelantado</b>
support, of filter	typeahead	
<b>spline cúbica natural</b>	natural cubic spline	<b>teclado</b>
natural cubic spline	keyboard	
<b>splines</b>	splines	<b>teclas aceleradoras</b>
splines	accelerator keys	
<b>subdivisión adaptable</b>	adaptive subdivision	<b>técnica de interacción de botón de radio</b>
adaptive subdivision	radio button interaction technique	
<b>subdivisión espacial</b>	spatial subdivision	<b>técnica de interacción para reconocimiento de patrones</b>
spatial subdivision	pattern recognition interaction technique	
<b>superficie bicúbica</b>	bicubic surface	<b>técnicas de interacción</b>
bicubic surface	interaction techniques	
<b>superficie cuádrica</b>	quadric surface	<b>tecnología de grupos</b>
quadric surface	group technology	
<b>superficies curvas</b>	curved surfaces	<b>teoría de tres estímulos</b>
curved surfaces	tristimulus theory	
<b>superficies de spline B</b>	B-spline surfaces	<b>término de Fresnel</b>
B-spline surfaces	Fresnel term	
<b>superficies paramétricas</b>	parametric surfaces	<b>tijereteo</b>
parametric surfaces	scissoring	
<b>tabla de aristas activas (AET)</b>	active-edge table (AET)	<b>tinta</b>
active-edge table (AET)	tint	
<b>tabla de aristas</b>	edge table	<b>tinte</b>
edge table	hue	
<b>tabla de colores</b>	color table	<b>tipo de letra</b>
color table	typeface	
<b>tabla de consulta</b>	look-up table (LUT)	<b>tono</b>
look-up table (LUT)	tone	
<b>tabla de polígonos</b>	polygon table	<b>transformación de modelado</b>
polygon table	modeling transformation	
<b>tabla de superficies</b>	surface table	<b>transformación de perspectiva</b>
surface table	perspective transformation	
<b>tableta de datos</b>	data tablet	<b>transformación normalizada</b>
data tablet	normalized transformation	
<b>tamaño de punto</b>	dot size, spot size	<b>transparencia</b>
dot size, spot size	transparency	
<b>tarea de interacción compuesta</b>	composite interaction task	<b>transportabilidad</b>
composite interaction task	portability	
<b>tarea de interacción lingüística</b>	linguistic interaction task	<b>transpuesta</b>
linguistic interaction task	transpose	
<b>tarea de interacción para cuantificación</b>	quantify interaction task	<b>traslación</b>
quantify interaction task	translation	
<b>tarea de interacción para posicionamiento</b>	positioning interaction task	<b>traza de rayos</b>
positioning interaction task	ray tracing	

<b>trazo</b>	tangent vector
stroke	
<b>tubo de rayos catódicos (CRT)</b>	vectorización
cathode-ray tube (CRT)	vectorize
<b>unidad de procesamiento de pantalla</b>	velocidad paramétrica
display processing unit	parametric velocity
<b>uso adelantado del ratón</b>	ventana
mouseahead	window
<b>valores de tres estímulos</b>	vértices fantasma
tristimulus values	phantom vertices
<b>variable de decisión</b>	<b>video compuesto</b>
decision variable	composite video
<b>vector de refracción</b>	<b>visibilidad</b>
refraction vector	visibility
<b>vector intermedio</b>	<b>vista</b>
halfway vector	view
<b>vector tangente</b>	<b>visualización científica</b>
	scientific visualization
	<b>volumen de vista</b>
	view volume



# Índice de materias

2-variedades, 427  
gráficos, 11  
sistema, 10  
 $\alpha$  (ángulo entre  $\bar{R}$  y  $\bar{V}$ ), 552  
 $\eta_{i\lambda}$ ,  $\eta_{i\Omega}$  (índices de refracción), 575  
 $\theta$  (ángulo entre  $\bar{L}$  y  $\bar{N}$ ), 544  
 $\theta_i$  (ángulo de incidencia), 544, 575  
 $\theta_r$  (ángulo de refracción), 575  
 $\rho$  (reflectividad bidireccional), 556  
 $\rho_d$  (reflectividad bidireccional difusa), 556  
 $\rho_s$  (reflectividad bidireccional especular), 556  
accionador de evento, 44-47  
ACE, 474  
aceptación trivial, 119  
acercamiento/alejamiento, 172  
ACM, 14  
acorde, estado de botón de localizador, 47  
actualización de pantallas por lotes, 319  
ajuste fino, 439  
alambrado, 418, 442-443  
generación de imágenes, 314  
alesas, 553  
algoritmos de prioridad de listas, 526, 530,  
véase también algoritmo de  
ordenamiento por profundidad, árbol  
binario para partición de espacio  
alineación  
para patrones, 108

alineación por la derecha, 129  
almacenamiento central de estructuras  
(CSS), 281  
almacenamiento de lista de presentación, 171  
alumbrado, véase iluminación  
American National Standards Institute,  
véase ANSI  
analizador diferencial digital, 83  
ancho de banda, 158, 175  
ancla, de un patrón, 108  
ángulo de pantalla, impresión, 451  
ángulo sólido, 555  
animación, 6, 492-494  
control de, 494  
convencional, 494  
cuadros clave, 494  
lenguajes gráficos, 493  
personajes de caricaturas, 495  
puesta en escena de la, 495  
reglas básicas de la, 495  
animación, control de  
cuadros clave, 494  
explícito, 494  
ANSI (American National Standards  
Institute), 14, 271  
aplicación  
base de datos de la, 18  
modelo de, 18

- programa de, 17
- árbol binario para partición de espacio (BSP)
  - algoritmo de sombra, 572
  - determinación de superficies visibles, 532
  - operaciones regularizadas de conjuntos booleanos, 429
  - para modelado de sólidos, 436-438
- árbol de cuadrantes, 509, 533
  - detección de vecinos en un, 435
- árbol de octantes
  - detección de vecinos en un, 436
  - operaciones regularizadas de conjuntos booleanos, 434
  - rotación en un, 435
- árboles, modelado de, 408, 411-414
- arco elíptico, 31-32
- área de superficie escorizada frontalmente, 55
- área de vista, 203, 234-235, 231-233
  - SPHIGS, 264-266, 289-292
- aristas activas, tabla de (AET), 105, 515
- aristas, coherencia de, 98, 103
- aristas, tabla de, 514
- arreglo bidimensional de bits, véase mapa de bits
- arreglo bidimensional de pixeles, véase mapa de pixeles
- artefacto de discretización, 137, 487, 495, 501, 513, véase también eliminación de artefacto de discretización
  - artefacto, 12
  - polígonos astilla, 103
  - temporal, 493, 495
- artefacto de discretización temporal, véase artefacto de discretización, temporal
- asideros para la interacción con el usuario, 360
- atenuación atmosférica, 548
- atributos, 32
  - de objetos, 18
  - herencia, 310
  - no geométricos, 286, 313
  - primitivas de salida, 32-39
  - SRGP, 32-34
- autocompletación de mandatos, 348
- autooclusión, 545
- autosimilitud, 405
  
- B** (radiosidad), 548
- B-spline**, curvas de, véase *splines*, curvas de *B-spline*
- B-spline**, superficies de, véase *splines*, superficies de *B-spline*
  
- B-splines** no uniformes, no racionales, 391-394
- barrido (*scan*), 9
  - pantalla de, 167-174, véase también tubo de rayos catódicos
- barrido (*sweep*), 424-426, 440-442
  - general, 425
  - rotacional, 425
  - traslacional, 425
- barrido aleatorio, 11
- barrido de trama, generador de, 164
- barrido entrelazado, 174
- barrido, generación de, 480, véase también discretización, sombreado
- barrido, líneas de, 10
- barrido, operación de (*RasterOp*), 149, véase también BitBlt, PixBlt, modo de escritura
- barrido, paquete de gráficos, 15, 25-26, 57, 69
- barrido, procesador de imágenes de, 70
- Bernstein, polinomios de, 382
- Bézier, curvas de, véase *splines*, curvas de Bézier
- biblioteca de subrutinas gráficas, 19
- BitBlt, 77, véase también CopyPixel
  - implantación de, 136
- bitmap**, véase mapa de bits
- bobinas de desviación, 156
- bola rastreadora, 179, 339, 357
- borrosidad de movimiento, 491
- botón de radio, técnica de interacción, 352
- Bresenham, discretización de círculos, véase discretización, círculo de punto medio
- Bresenham, discretización de líneas, véase discretización, línea de punto medio
- brillantez de la luz, 455
- BSP, árbol, véase árbol binario para partición de espacio
- buffer** de elementos, 588
  
- C iluminante, 461
- CAD, véase diseño asistido por computador
- caja acotante, 506, 525, véase también extensión
- caja, filtro de, véase filtro de caja
- CAM, véase manufactura asistida por computador
- cámara
  - sintética, 193, 223, 288-289
  - visualización, 267
- campo de valor, 107
- cañón de electrones, 162, 167

- carácter (caracteres)
  - alineación de, 134
  - anchura de, 39, 134
  - cursivas, 39, 135
  - descendente, 40, 134
  - familia tipográfica, 39, 135
  - línea base, 40
  - negritas, 135
  - reconocimiento de, 353
  - romano, 135
  - tipo de letra, 132
- cartografía, 3
- centro de proyección (COP), 224, 231
- centro de ventana (CW), 231
- CIE (Commission Internationale de l'Éclairage), 460, 462, 472
- CIE, colores primarios, 462
- CIE, diagrama de cromaticidad, 459, 461, 463
- CIE, modelo de color, 474
- cilindro generalizado, véase barrido
- círculos, véase discretización de círculos
- claridad, color, 455
- CMY, modelo de color, 464
- CMYK, modelo de color, 466
- codificación visual, 360
- códigos de región, recorte por, 119-123
- Cohen-Sutherland, algoritmo de recorte de
  - líneas de, véase recortes, algoritmo de
  - línea de Cohen-Sutherland
- coherencia
  - de áreas, 502
  - de aristas, 98, 502
  - de aristas implicadas, 502
  - de caras, 502
  - de cuadros, 503
  - de línea de rastreo, 98, 502
  - de objetos, 502
  - de profundidades, 503
  - de tramos, 516
  - espacial, 98
- color, véase longitud de onda dominante, tinte, luminancia
- color no espectral, 462
- colores, armonía de, 474
- colores, codificación de, 474
- colores complementarios, 462, 464, 466
- colores, correspondencia de, véase video, tabla de consulta
- colores, deficiencia en la percepción de, 475
- colores, funciones de equivalencia de, 458-459
- colores, gamas de, 462-464
- colores, interpolación de, 472
- colores, modelos de, 464, véase también CIE, CMY, HSV, RGB, YIQ
- colores primarios aditivos, 464
- colores primarios sustractivos, 464
- colores, reglas de utilización de, 472
- colores, tabla de, 34, véase también video, tabla de consulta
- colorimetría, 456
- combinación lineal, 188
- Commission Internationale de l'Éclairage, véase CIE
- composición, véase composición de transformaciones
- compresión y estiramiento, 495
- conectividad, 18
- conjunto
  - abierto, 420
  - cerrado, 420
  - cerradura, 420
  - interior, 420
  - puntos de frontera, 420
  - regular, 420
  - regularización, 420
- conjunto de nombres, PHIGS, 328
- conjunto de opciones, 346, 349-353
- comutador de pie, 342
- comutatividad de operaciones matriciales, 187, 203
- cono hexagonal, modelo de colores HSV, 468-469
- conos, 553
- conos receptores en el ojo, 457
- continuidad, véase continuidad geométrica, continuidad paramétrica
- continuidad geométrica, 374-376
- continuidad paramétrica, 374-375
- contornos, intensidad de los, 449-451
- control local, curvas cúbicas, 387
- convergencia de un haz de electrones, 157
- conversión entre modelos de colores, 463-472
- Cook-Torrance, véase iluminación de Cook-Torrance
- coordenadas de proyección normalizadas (NPC), 234
- coordenadas homogéneas, 196
- coordenadas, sistemas de
  - $(u, v, n)$ , 230
  - $(u, v, Vpn)$ , 231, 267
  - de barrido, 267
  - de cámara, 267
  - de dispositivo, 267
  - de dispositivo local, 267
  - de dispositivo normalizado, 267
  - del problema, 267
  - de la aplicación, 68, 267
  - de mano derecha, 207, 231

- de mano izquierda, 267
- de modelado, 266
- de mundo, 203, 266
- de objeto, 266
- de pantalla, 267, 356
- de proyección normalizada, 267
- de referencia de vista (VRC), 230
- en SRGP, 61-63
- locales, 266
- oculares, 223, 267
- vector de vista arriba (VUP), 230
- coordinación visual-motriz, 339
- CopyPixel**, 110
- Core Graphics System**, 14, 267
- correlación, véase selección, correlación de croma, color, 455
- cromaticidad, color, 455, 460-463, 469
- cromaticidad, coordenadas de, 460-463
- cromaticidad, diagrama de, véase CIE, diagrama de cromaticidad
- CRT**, véase tubo de rayos catódicos
- CRT de dos niveles**, véase pantalla de dos niveles
- CRT parpadeo**, enfoque, 9, 157
- CRT paso**, de mascarailla de sombra, 159
- CSG**, véase geometría sólida constructiva
- cuadro clave, 494
- cuadros intermedios, creación de, 494
- cuantificación, tarea de interacción para, véase interacción, tareas de cursivas, véase carácter, cursivas
- cursor de texto, 50
- cursor, tridimensional, 514
- curva cúbica, 365
- curvas, véase también *splines*
  - cúbicas paramétricas, 371-396
  - polinomiales paramétricas, 364
- curvas, ajuste de, 394-395, véase también *splines*, curvas de Bézier, ajuste de datos con
- curvas cúbicas paramétricas, véase curvas, cúbicas paramétricas
- curvas de *spline B* uniformes, no racionales, 387-393
- curvas polinomiales paramétricas, véase curvas, polinomiales paramétricas
- Cyrus-Beck**, algoritmo de recorte de líneas de, 123
  
- D** (función de distribución de microfacetas), 556
- DataGlove**, 343
- DDA**, véase analizador diferencial digital
  
- descendente, carácter, véase carácter descendente
- descomposición en celdas, 431
- desechar caras posteriores, 507-508, 524, 530, 591
- desplazamiento, 172, 308
- determinante, véase matriz, determinante de una
- diagrama de transición de estado, 348
- diálogo, recuadro de, 357
- dibujo
  - lista de, 171
  - primitivas de, 9
- dibujo de círculos elásticos, 358
- dibujo de líneas elásticas, 358
- dibujo de rectángulos elásticos, 358
- dibujo, recorrido de, 287
- herencia de atributos, 310, 313
- implantación del, 329
- optimización del, 330
- transformación de modelado, 306
- transformación de visualización, 287-292
- diferencia apenas perceptible en colores, 459
- diferencias de segundo orden, discretización de círculo, 96
- digitalización, 394
- digitalizador, 181-183
- dimensión fractal, 405
- dinámica, 492
  - de actualización, 16, 315
  - del modelado geométrico, 300
  - de movimiento, 16, 315
- diodo emisor de luz (LED), 343-345
- dirección de proyección (DOP), 225, 226, 231
- dirección de reflexión (*R*), 550
- direccionalismo, capacidad de, 151
- discretización, 12, véase también barrido, generación de
  - cadenas de texto, 135-136
  - caracteres, 132-136
  - círculo de punto medio, 93-97
  - de círculos, 92-97
  - de líneas, 9, 81, 92
  - de líneas incrementales, 82-84
  - de primitivas gruesas, 112
  - de primitivas huecas, 91-92
  - línea de punto medio, 84-92
  - líneas con eliminación de artefacto de discretización, 137-144
  - mallada de triángulos, 106
  - rectángulo, 98
  - trapezoide, 106
  - triángulo, 106

- diseño asistido por computador (CAD), 4, 18, 247  
disparidad binocular, 496-497  
dispersión, 575  
dispositivo acoplado por carga (CCD), 181  
dispositivo, independencia del  
  gráficos, 13  
  interfaz, 76  
dispositivo lógico de entrada, 177, 338  
dispositivo lógico de opciones, 44, 177, 341  
dispositivo lógico valuador, 44, 177, 241  
dispositivos de entrada, 177, 337-338, véase  
  también interacción, tareas de, 337, 338  
  dispositivo lógico de entrada  
dispositivos de presentación por barrido, 25  
distancia al plano anterior ( $F$ ), 232-234, 252  
distancia al plano posterior ( $B$ ), 232  
distancia entre puntos, 151  
distribución energética espectral, 456-457  
 $d_1$  (distancia de la fuente puntual a la  
  superficie), 546  
ducto de entrada, 77  
ducto de salida, 80, véase también  
  presentación
- eco, véase retroalimentación  
ecuaciones implícitas, 371  
edición, de red de estructuras, 272, 283,  
  315-320  
eficiencia luminosa, función de, 457  
 $E_i$  (irradiancia incidente), 55  
ejemplar, jerarquía de objetos, 278  
ejemplares, bloque de, 317-319  
elemento de textura, 565  
elemento de volumen, 431  
elemento, red de estructuras, 281, 283, 316  
eliminación del artefacto de discretización,  
  12, 76, 81, 90, 137, 144, 419, véase  
  también muestreo de área  
  temporal, 493, 495  
embaldosado, 291  
energía, distribución de, véase distribución  
  energética espectral  
energía electromagnética, 456  
energía luminosa, 457  
enfoque, CRT, 156-158  
entrada, véase interacción, manejo de  
  entrada lenta/salida lenta, 495  
envoltura convexa, 383, 389, 393  
ergonomía, véase factores humanos  
escalamiento, véase también escalamiento de  
  imágenes  
  bidimensional, 193, 198, 199  
diferencial, 194  
tridimensional, 207  
escalonamiento, 137  
escritorio  
  metáfora de, 337  
  ventanas, 8  
espaciado entre caracteres, 135  
espacio de direcciones, 164, 174  
  único, 173-174  
espectro, 457  
estación de trabajo, 150  
estación de trabajo gráfica, véase estación de  
  trabajo  
estándares  
  paquetes gráficos, 272  
estéreo  
  par, 355, 496  
estereoscopía, 496  
estereorradián, 555  
estímulo-respuesta (E-R), compatibilidad, 356  
estructura(s)  
  correlación de selección, 320  
  edición de, 315  
  ejemplo de, 300-305  
  elisión, 331  
  jerarquía de, 298-305  
  referencia a la, 331  
  transformación de modelado, 292-298, 304,  
    330  
estructuras, 14  
estructuras, red de, 288, 290  
etiqueta, elemento de estructura, 316  
Euler, fórmula de, 427-429  
Euler, operadores de, 429  
evento, 20  
  modo de entrada de SRGP, 43  
eventos, cola de, 45  
eventos, dirigido (dirigida) por  
  interacción, 44-45  
  lazo, 19  
extensión, 505  
  de texto, 41  
  mínmax, 506-507  
  objeto tridimensional, 330, 336  
extensión de texto, 134  
extensiones geométricas, véase extensión
- $F_\lambda$  (término de Fresnel), 556  
factores humanos, 42  
familia tipográfica, véase carácter  
familias tipográficas, memoria caché de, 133  
fantasma, 176  
faro, 554  
 $f_{att}$  (factor de atenuación de fuente  
  luminosa), 546

- fenómenos naturales, 404  
 filtro  
     de caja, 141  
     de cono, 142  
     realce/invisibilidad, 328  
     soporte del, 141  
 filtro de cono, 142  
 Fitts, ley de, 352  
 Floyd-Steinberg, difusión de error de, 453  
 flujo, 555, véase también irradiancia  
 fluorescencia, de fósforo, 157  
 fondo, atributo de color, 37-38  
 fosforescencia, 157  
 fósforo, 156-158, 457, 464  
 fotómetro, 449  
 fotorealismo, 479-482  
     generación de imágenes, 22  
 fractal, 405; véase también dimensión fractal  
 frecuencia de fusión, crítica, 157  
 Fresnel, término de, véase  $F_\lambda$   
 frontera, representación de, 426-427, 429,  
     438-442  
     operaciones regularizadas de conjuntos  
         booleanos, 419, 422, 424, 438  
 fuente luminosa, véase también iluminación  
     aletas, 553  
     atenuación de, 546  
     conos, 553  
     de color, 548  
     direccional, 488, 545  
     distribuida, 488  
     extendida, 488  
     factor de atenuación ( $f_{\text{att}}$ ) de, 546  
     faro, 554  
     puntual, 488, 543  
     reflector, 554  
     sobreflujo, 555  
     Warn, controles de, 552  
 función de filtrado, 141  
 función de plantilla, modelado geométrico,  
     297  
 función de ponderación, 140  
 funciones de mezclado, curvas, 376-378, 382,  
     387-393  
 funciones explícitas, 371  
  
 $G$  (factor de atenuación geométrica), 556  
 gama, corrección, 449  
 generación de imágenes, 480  
 generación de imágenes, ducto de, 590  
     iluminación global, 594  
     iluminación local, 591  
     memoria de profundidad  $z$ , 592  
     prioridad de listas, 593  
  
 rastreo de rayos, 594  
     sombreado de Gouraud, 592  
     sombreado de Phong, 593  
 generación de imágenes, ecuación, 578  
 generación de imágenes, SPHIGS, véase  
     también recorrido de presentación  
     tipos, 313-315  
 género de un poliedro, 428  
 geometría sólida constructiva (CSG), 438, 458  
 GKS-3D, 14  
 Gouraud, véase sombreado de Gouraud  
 grabadora de película fotográfica, 149, 154  
 grados, rectangulares, 31  
 graficador  
     de cama plana, 152  
     de chorro de tinta, 153  
     de escritorio, 152  
     de tambor, 152  
 gráficos de modo inmediato, 280  
 gráficos de modo retenido, véase SPHIGS  
 gráficos vectoriales, 11  
 graftales, 410  
 grosor, 113  
  
 $H$  (vector intermedio), 552  
 "Haz lo que quiero decir" (DWIM), 348  
 Hermite, curvas de, 376, véase también  
     splines  
 Hermite, superficies de, 397  
 hipótesis del objeto, 483  
 HOOPS, 15, 271, 333-334  
 HSB, modelo de colores, véase HSV,  
     modelo de colores  
 HSV, modelo de colores, 467  
 huella, 113  
     dispositivo de interacción, 339  
  
 $I_{\text{dx}}$  (color de indicador de profundidad), 548  
 $I_i$  (radiancia incidente), 555  
 iluminación, véase también fuente luminosa  
     de Cook-Torrance, 554  
     de Phong, 550  
     de Torrance-Sparrow, 556  
     ecuación de, 543  
     global, 542, 577  
     local, 577  
     modelo de, 542  
     modelos físicos de, 554  
 imagen, precisión de, véase superficies  
     visibles, determinación de  
 imágenes, escalamiento de, 70  
 imágenes, irradiancia de, 555  
 imágenes, jerarquía de, véase jerarquía de  
     objetos

- impresora  
de chorro de tinta, 153  
de matriz de puntos, 152  
de transferencia de tintes por sublimación, 154  
de transferencia térmica, 153  
láser, 152
- índic平 de profundidad, 485, 497, 548
- inhibición lateral, 559
- integración espacial, 451
- intensidad  
de una línea como función de la pendiente, 91  
luminancia, 448  
radiante, 489  
resolución de, 452
- interacción, véase dispositivo lógico de entrada
- interacción compuesta, tarea de, véase interacción, tareas de
- interacción, conjuntos de herramientas para, 361-362
- interacción dirigida por muestras, 44-47
- interacción lingüística, tarea de, 346
- interacción, manejo de, 20-21, 42, 53  
en SPHIGS, 320-324  
en SRGP, 53-54
- muestreo contra evento, 44-46
- interacción “opríma y arrastre”, 359
- interacción, tareas de, 338, 345-357  
compuestas, 357-361  
cuantificación, 338, 353  
posicionamiento, 338, 345  
rotación tridimensional, 356  
selección, 338, 346, 349  
texto, 338, 353  
tridimensionales, 354-355
- interacción, técnicas de, 338, 345-347, 361  
especificación de colores, 471
- interfaz con el usuario, 2, 443
- interior, véase regla de paridad impar, algoritmos de relleno
- International Standards Organization (Organización Internacional de Estándares), véase ISO
- interpolación, véase también sombreado color, 472
- intersección externa, 123
- InterViews, 362
- intervalo dinámico, intensidad, 449, 450
- inversa, véase matriz, inversa de una
- $I_p$  (intensidad de fuente puntual luminosa), 545
- $I_r$  (radiancia reflejada), 555
- irradiancia, 555
- ISO (International Standards Organization), 14
- jerarquía, 509, 524, véase también recorrido de presentación.
- JPL, 481
- Julia-Fatou, conjunto de, 406
- $k_a$  (coeficiente de reflexión ambiental), 543
- $k_d$  (coeficiente de reflexión difusa), 544
- $k_i$  (color intrínseco), 543
- $k_s$  (coeficiente de reflexión specular), 550
- $k_t$  (coeficiente de transmisión), 573
- $L$ , gramáticas, 410
- $L$  (vector a luz  $L$ ), 544
- Lambert, ley de, 544
- LCD, véase pantalla de cristal líquido
- lenguajes gráficos, véase también animación, lenguajes gráficos
- Liang-Barsky, algoritmo de recorte de líneas de, véase recortes, algoritmo de línea de Liang-Barsky
- lienzo, 57  
contexto de, 58  
de pantalla, 58  
estado de, 58  
fuera de pantalla, 79
- línea, véase también discretización  
estilo, 32-34
- línea base, para texto, 40
- línea de rastreo, algoritmo de, 105, 514, 516, 517, 535-536, 569
- operaciones regularizadas de conjuntos booleanos, 519
- líneas, dibujo de, 9
- líneas gruesas, véase discretización de primitivas gruesas
- líneas ocultas, determinación de, véase líneas visibles, determinación de
- líneas visibles, determinación de, 487, 499
- localizador, 20
- localizador, dispositivo lógico, 177  
absoluto, 339  
bidimensional, 43-55  
continuo, 340  
directo, 340  
discreto, 340  
indirecto, 340  
relativo, 339  
tridimensional, 320, 342-345
- localizador, mascarilla de botón del, 50-51
- longitud de onda dominante, color, 456, 459, 460-463

- longitud de un vector, 188
- luminancia, color, 448, 456
- Luxo, Jr., 491
- luz, véase luz ambiental, reflexión difusa, iluminación, etcétera
- luz acromática, 447
- luz ambiental, 488, 542-543
- luz, intensidad percibida, 448
- luz reflejada, 455
  
- Mach, bandas de, 559
- Macintosh, 352-354, 362
- maestro, jerarquía de objetos, 278
- malla cuadrilateral, véase malla poligonal
- malla de control de CRT, 157
- malla poligonal
  - consistencia de la, 364-368
- Mandelbrot, conjunto de, 405-409
- manipulación directa, 9
  - apuntar y presionar, 9
  - interfaces con el usuario, 338
- manojo de atributos, PHIGS, 329
- manufactura asistida por computador (CAM), 8
- mapa de bits, 8, 75; véase también mapa de pixeles
  - caracteres de, 133
  - fuera de pantalla, véase lienzo, fuera de pantalla
  - gráficos de, 8
  - patrón de, 35-38
- mapa de pixeles, 11, 75, 163, 169, 176; véase también lienzo, fuera de pantalla
- marca, primitiva de salida, 29
- mascarilla de tensión plana, CRT, 457-458
- mascarillas, 61
- matiz, color, 455
- matriz
  - determinante de una, 190
  - identidad, 190
  - inversa de una, 191
  - multiplicación de una, 190, 191
  - transpuesta de una, 191
- matriz base, 375-376, 381, 389, 415
- matriz de transformación global, 307
- matriz de transformación local, 294
- matriz, direccionamiento de pantalla, 161
- matriz geométrica, 375-377, 381, 388, 396-400
- matriz ortogonal especial, 199, 208
- medida, dispositivos lógicos de entrada, 44, 47-53
- medios tonos, 451-452
  
- memoria de acceso aleatorio para vídeo, véase vídeo, RAM para
- memoria de objetos, 514
- memoria de profundidad z, algoritmo de, 510-514, 534
- memoria doble, 173, 330
- memoria gráfica, 11, 163-167
- menú
  - cuerpo de, 53
  - de barra, 53
  - encabezados de, 53
- menú descendente, véase menús, descendentes
- menús
  - aparición de, 351
  - de ventana, 351-352
  - descendentes, 351-352
  - desprendibles, 351
  - estáticos, 352
  - jerárquicos, 349
- menús jerárquicos, selección por, 350
- metaarchivo, 329
- métodos incrementales, 80
- microfacetas, 556
- Microsoft, 352, 353
- modalidad, 32
- modelado, 273
- modelado, véase modelado geométrico
- modelado de objetos, véase modelado geométrico
- modelado geométrico, véase también estructuras, jerarquía de
  - interactivo, 305
  - jerarquía de objetos, 276
- modelado molecular, 481
- modelado, transformación de, véase transformación de modelado
- modelos fractales, 405-410
- modelos gramaticales, 410-411
- modo de escritura, 63
- modo de procesamiento, teclado, 49
- modo, dispositivos de entrada, 47
- muestra, 20
- muestra, modo de entrada SRGP, 47-48
- muestreo de área, 137-142
  - no ponderada, 137
  - ponderada, 140
- multimedia, sistemas, 4
- mundo virtual, 497, véase también pantalla montada en la cabeza
- Munsell, sistema de clasificación de colores de, 455
  
- $n$  (exponente de reflexión especular), 550
- $N$  (normal a la superficie), 544

- NASA, 481  
 National Television System Committee (NTSC), 175-176, 413  
 Necker, ilusión del cubo de, 483  
 negritas, véase carácter  
 Newell-Newell-Sancha, algoritmo de, véase algoritmo de ordenamiento por profundidad  
 NeXT, 310, 312  
 Nicholl-Lee-Nicholl, algoritmo de, véase recortes, algoritmo de línea de Nicholl-Lee-Nicholl  
 normal  
     a un plano, 209, 368  
     a una superficie, 591  
     a una superficie bicúbica, 400  
     a una superficie cuádrica, 403  
 NPC, véase coordenadas de proyección normalizadas  
 NTSC, véase National Television System Committee  
 nudo, curva cúbica, 387  
 NURBS, véase *splines* B no uniformes, racionales  
  
 objeto autoluminoso, 461  
 objeto estandarizado, 292  
 objeto, precisión del, véase determinación de superficies visibles  
 ocupación espacial, enumeración de, 431, 440  
 $O_{\alpha}$  (color difuso de objeto), 548  
 ojo, 452, 507, 519  
 opacidad, 573  
 OPEN LOOK, 362  
 Open Software Foundation (OSF), 362  
 OpenGL, 15, 271, 333-334  
 operación de barrido (*RasterOp*), 63-66, 170, véase también barrido, operación de  
 operaciones booleanas regularizadas de conjuntos, 419-423, 424, 438, véase también geometría sólida constructiva en comparación con las operaciones booleanas ordinarias de conjuntos, 421  
 para árboles binarios para partición de espacio, 436  
 para árboles de octantes, 434  
 para barridos, 425  
 para representaciones de fronteras, 429  
 traza de rayos, 519  
 ordenamiento por profundidad, algoritmo de, 527, 535-536  
 $O_{\sigma\alpha}$  (color especular de objeto), 551  
 Ostwald, sistema de clasificación de colores, 455  
  
 PAL, estándar de televisión, 175  
 palanca de mandos, 179, 339  
 paleta, 42  
 pantalla caligráfica, 9  
 pantalla, controlador de, 9, 77-78  
 pantalla, coprocesador de, véase procesadores de pantalla gráfica  
 pantalla de cristal líquido (LCD), 149, 160  
 pantalla de dos niveles, 11, 452-453  
 pantalla de plasma, 161  
 pantalla electroluminiscente (EL), 161  
 pantalla monocromática, véase pantalla de dos niveles  
 pantalla montada en la cabeza, 343  
 pantalla sensible al tacto, 180  
 PANTONE MATCHING SYSTEM, 455  
 paquete de subrutinas gráficas, 19, 271  
 paralelogramo, regla del, 188  
 parches de superficie polinomial paramétrica de dos variables, 365, véase también *splines*  
 paridad impar, regla de, 35  
 partición espacial, 509, 525-526 adaptable, 509  
 partición espacial, representaciones de la, 430-438  
 patrón, atributo de salida, 36-39  
 patrón de medios tonos, 61  
 patrones, correspondencia de, véase superficie, detalle de  
 patrones, relleno con, véase relleno, com patrones  
 patrones, técnica de interacción para reconocimiento de, 353  
 pels, 8  
 persistencia, fósforo, 157  
 perspectiva aérea, 485  
 perspectiva, proyección de  
     de dos puntos, 227, 238, 244  
     de tres puntos, 227, 246  
     de un punto, 226, 236-238, 245  
 perspectiva, escorzo de, 224  
 perspectiva, transformación de, véase transformación, perspectiva  
 PHIGS, 14, véase también SPHIGS  
 PHIGS Plus, 15  
 Phong, véase iluminación de Phong y sombreado de Phong  
 pincel  
     forma del, 112  
     orientación del, 112  
 pincel, estilo de, 113  
 pintor, algoritmo del, 527  
 pintura, implantación, 46

- Pitteway, 84  
 Pixar, 267, 433  
 PixBlt, 163, 176, véase también BitBlt  
 pixel(es), 8  
     duplicación de, 70, 113  
     geometría de, 140  
 pixmap, véase mapa de pixeles  
 plano  
     ecuación de, 191, 209, 368-370  
 plano de recorte cercano, véase plano de recorte, anterior  
 plano de recorte lejano, véase plano de recorte, posterior  
 plano de recorte  
     anterior, 232-234, 252  
     posterior, 232, 242-243, 252, 258  
 plantilla, 185  
 poda, 331  
 Polhemus, digitalizador tridimensional, 342-345  
 poliedro  
     simple, 427  
     SPHIGS, 285  
 polígono  
     prueba interior, 35  
 polígonos de astilla, 103  
 polígonos, recorte de, véase recortes, de polígonos  
 polígonos, tabla de, 515  
 polilínea, 27  
 polinomio continuo por trozos, 364, 371, 376  
 posición actual (CP), 169-170  
 posicionamiento, tarea de interacción de, véase interacción, tareas de  
 PostScript, 68, 153, 395  
 potenciómetro, 341, 354  
 primarios, colores, 464, 465, 466  
 primitivas, véase primitivas de salida  
 primitivas de salida  
     gráficos de barrido de, 26-32  
     modelado geométrico de, 284  
     reespecificación de, 68  
 primitivas, generación de ejemplares de, 423  
 prioridad de presentación, 291  
 procesadores de pantalla gráfica, 163  
 producto punto, 188  
 profundidad de campo, 490  
 programa de dibujo, véase dibujo, lista de propiedades materiales, 489  
 proyección, 222  
     de perspectiva, 224  
     paralela, 224  
     paralela axonométrica, 228  
     paralela isométrica, 228  
 paralela oblicua, 228  
 paralela ortográfica, 222, 227  
 planta, 230  
 proyección geométrica plana, 224  
 proyección, implantación de paralela, 245-253  
     perspectiva, 253-258  
 proyección, matriz de ortográfica, 245  
     perspectiva, 243-245  
 proyección paralela  
     frontal, 227, 229, 241  
     lateral, 227, 228-229  
     oblicua, 198, 200, 201  
     ortográfica, 227, 228  
     planta, 230  
     superior, 230, 229, 242  
 proyección, plano de, 224  
 proyección, punto de referencia de (PRP), 231  
 proyector, 224  
 puesta en escena, véase animación, puesta en escena  
 punteo agrupado ordenado por puntos, 451  
 puntero ordenado, 451  
 punto de evaluación, 94  
 punto de máximo brillo, en modelos geométricos, 329  
 punto, tamaño de, 151  
 puntos de control múltiples, curvas, 388-391, 394  
 puntos de fuga, 225  
 puntual, fuente luminosa, véase fuente luminosa puntual  
 pureza, véase pureza de excitación  
 pureza de excitación, color, 456  
 QuickDraw, 15, 112  
 R (dirección de la reflexión), 550  
 radiancia, 555  
 radiosidad (*B*), 583  
 radiosidad, métodos de  
     contaminación de colores, 586  
     disparo, 589  
     ecuación de radiosidad, 584  
     factor de forma ( $F_{ij}$ ), 584  
     factores de forma triangular, 587  
     hemicubo, 588  
     iteración de Gauss-Seidel, 585  
     memoria de profundidad  $z$ , 591  
     radiosidades de vértices, 585  
     recopilación, 589  
     refinamiento progresivo, 589

- relación de reciprocidad de factor de forma, 584
- término de ambiente, 589
- raíz, red de estructuras, 288
- ratón, 9, 178, 338-341, 346, 351-352, 355, véase también localizador mecánico, 178
- óptico, 178
- ratón, uso adelantado del, 47
- rayo, ojo, 520
- rayos, lanzamiento de, 429, 510, véase también rayos, traza de en operaciones de conjuntos booleanos b-rep, 429
- rayos, traza de, 459-465, 579-583
- árbol de rayos, 580
- árboles de octantes, 526
- cálculo de intersecciones, 521-523, 524
- eficiencia, 524-526
- esferas, 521-523
- jerarquía, 524
- normal a la superficie, 522
- partición espacial, 525
- polígonos, 522-523
- problemas de precisión numérica, 582
- propiedades de masa, 539
- rayos de reflexión, 580
- rayos de refracción, 580
- rayos de sombra, 579
- rayos primarios, 580
- rayos secundarios, 580
- recursiva, 581
- sombras, 579
- volumen acotante, 524
- realidad artificial, véase mundo virtual
- realimentación, 54-55
- realimentación, dispositivos de entrada, 54
- realismo, véase fotorealismo
- rechazo trivial, 120
- reconocimiento del habla, véase reconocimiento de voz
- recorrido, véase presentación, recorrido de recorte de profundidad, 486
- recortes, 79
  - analíticos, 114
  - cadenas de texto, 134
  - Cohen-Sutherland, algoritmo de línea de, 118-123
  - Cohen-Sutherland, algoritmo de línea tridimensional de, 258-260
  - de caracteres, 132
  - de Cyrus-Beck tridimensionales, 258
  - de gráficos de barrido bidimensionales, 60 de líneas, 116-128
- de polígonos, 128-132
- de primitivas bidimensionales en un mundo de barrido de trama, 114-132
- de puntos extremos, 117
- en coordenadas homogéneas, 260
- Liang-Barsky, algoritmo de línea de, 127, 143
- Nicholl-Lee-Nicholl, algoritmo de línea de, 127, 143
- Sutherland-Hodgman, algoritmo de recorte de polígonos de, 128-132
- tridimensionales, 223, 267, 503-505, 510, 539
- rectángulo de recorte, 60
- rectángulo, escritura de, 109
- reemplazo, modo de escritura de, 109, 137
- refinamiento progresivo, 595, véase también radiosidad
- reflectividad bidireccional
  - difusa, 556
  - especular, 556
- reflector, 554
- reflexión, 490, véase también reflexión difusa, dirección de reflexión, reflexión especular
- reflexión ambiental ( $k_a$ ), coeficiente de, 5
- reflexión difusa, 543
  - coeficiente ( $k_d$ ) de, 545
- reflexión especular, 549-551, 573
  - coeficiente ( $k_s$ ) de, 550
  - exponente ( $n$ ) de, 550
- reflexión interna total, 577
- reflexión lambertiana, 544
- refracción, 490
- refracción, vector ( $\vec{r}$ ) de, 576
- refrescamiento, 9, 156-157, véase también recorrido de presentación
- refrescamiento, tasa de, 157, 167
- rellenado
  - algoritmos de, 144
  - con patrones, 108
  - de polígonos, 99
  - de rectángulos, 98
- RenderMan, 267
- reparación de daños, 39
- representación B, véase representación de frontera
- representación paramétrica, en recorte de líneas, 123
- resolución, 151, 154, 158, 159
- resolución espacial, 452
- respuesta espectral, funciones de, 457
- restricciones, en el dibujo de líneas, 358
- retícula cúbica, 432

- retorno vertical, 165
- revisión de región, recortes, 118
- RGB, modelo de colores, 464
- RIP, véase barrido, procesador de imágenes de rotación
  - bidimensional, 201
  - tridimensional, 207
- rotación, tarea de interacción, véase interacción, tareas de
- rueda de carreta, ilusión de, 493
- rueda de reencarnación, 171
- saturación, color, 455-456, 460, 464, 469, 473
- Schröder, ilusión de escalera de, 483
- SECAM, estándar de televisión, 175
- segmentos, almacenamiento local de, 172
- segmentos, en GKS, 14
- selección
  - correlación de, 53-54
  - dispositivo lógico, 177
  - dispositivo lógico de entrada, 43
  - identificador de, 322
  - implantación de correlación de, 322
  - jerarquía de correlación de objetos, 323
  - por nombre, 347
  - por señalamiento, 348
  - punto de, 119
  - ventana de, 119
- selección de tarea de interacción, véase interacción, tareas de
- selección jerárquica de objetos, 348-349
- serrado, 12, 137, véase también eliminación de artefacto de discretización
- sesgo
  - bidimensional, 200
  - tridimensional, 209
- seudorealismo, generación de imágenes, véase fotorrealismo
- S*, (coeficiente de sombra), 569
- SIGGRAPH, 14
- símbolo, 2, 185, 209, 347, 359
- simulación, 6
- simulador de vuelo, 16
- sistema de coordenadas de mundo, 68
- sistema de proyección de válvulas de luz, 162
- sistema gráfico
  - transformación de entrada, 17
  - transformación de salida, 17
- sistema vectorial, 9
- sistemas de coordenadas, representación de, 81
- Sketchpad, 8
- Snell, ley de, 577
- sólidos, modelado de, 417-445, véase también modelado geométrico
  - características del, 444
  - clasificación de puntos, 437
  - interfaz con el usuario, 443
  - objetos tolerados, 444
  - solidez, 443
- sólidos, representación de modelado de comparación, 440-442
- conversión, 441
- modelo evaluado, 441
- modelo no evaluado, 441
- sombras, 489, 569
  - árbol binario para partición de espacio de volumen de sombra, 572
  - esfera de influencia, 571
  - falsas, 569
  - línea de barrido, 569
  - polígono de sombra, 571
  - región de influencia, 571
  - volumen de sombra, 571
- sombreado, 488, véase también barrido, generación de
  - constante (plano, con faceta), 557
  - de Gouraud (interpolación de intensidad), 488, 560
  - de malla poligonal, 559
  - de Phong (interpolación normal), 561
  - interpolado, 488, 558
  - modelo de, 541
  - problemas con el sombreado interpolado, 563-565
- sondeo, véase muestreo
- soporte del filtro, 141
- SPHIGS, 15, 272-335, véase también estructura
  - atributos de salida en, 286
  - gráficos de modo retenido en, 280
  - manejo de interacciones en, 320-324
  - modelado de objetos en, 292-302
  - primitivas de salida, 284-287
- SPHIGS, actualización de pantalla, véase visualización de operaciones
- spline* cúbica natural, 387
- splines*
  - B-splines racionales, no uniformes (NURBS), 394
  - curvas de Bézier, 375
  - curvas de Bézier, ajuste de datos con, 392-394
  - curvas de B-spline, 387
  - curvas de B-spline no uniformes, no racionales, 391
  - curvas de B-spline uniformes, no racionales, 387

- segmentos de curva polinomial cúbica no uniforme, racional, 393
- superficies de Hermite, 397
- superficies de B-spline, 400
- usadas para caracteres, 135
- SRGP, 15**
  - atributos de salida en, 32-39
  - control de memoria gráfica en, 57-67
  - manejo de interacción en, 42-47
  - primitivas de salida en, 26-32
- SRGPcopyPixel, 79, 134, 136**
  - implantación de, 136
- Storyboard, 494**
- subdivisión adaptable, véase partición espacial
- subdivisión de área, algoritmo de Warnock de, 433
- subdivisión de área, algoritmos de, 531
- subdivisión espacial, véase partición espacial
- superficie bicubica, 365, 396
- superficie cuadrática, 365, 403
- superficie, detalle de, 568
  - correspondencia de desplazamiento, 568
  - correspondencia de protuberancias, 567
  - correspondencia de texturas, 565-566
  - polígonos de detalle de superficie, 565
  - textura de sólidos, 568
- superficie, normal a la, 507, 523, 591
  - en operaciones de conjuntos booleanos, 421, 422, véase también, normal a la superficie
- superficie, parche de, véase también superficies curvas
  - algoritmo de presentación por subdivisión recursiva de Catmull, 534
- superficies activas, tabla de, 518
- superficies bicubicas, dibujo de, 401-403
- superficies curvas, véase también superficie, parche de
  - métodos de presentación de, 401-403
  - teselado de, 592
- superficies, modelado de, 364
- superficies ocultas, eliminación de, véase superficies visibles, determinación de
- superficies paramétricas, 364, 396
- superficies, tabla de, 518
- superficies visibles, determinación de, 487, 499, 503, 505, 519, 535, véase también algoritmos de subdivisión de área, algoritmo de ordenamiento por profundidad, algoritmos de prioridad de listas, algoritmo de línea de rastreo, algoritmo de Warnock, algoritmo de z-buffer
  - eficiencia, 524
  - ordenamiento, 535
  - precisión de imágenes, 500, 508, 510, 519, 527, 530
  - precisión de objetos, 500, 508, 527, 533
  - superficies curvas, 533-534
- Sutherland-Hodgman, algoritmo de recorte de polígonos, véase recortes, algoritmo de polígonos de Sutherland-Hodgman**
- T (vector de refracción), 576**
- tabla de consulta (LUT), véase video, tabla de consulta de tableta
  - de datos, 177
  - lápiz, 177
  - resistiva, 178
  - sónica, 178
- tableta de datos, véase tableta
- tamaño de punto, véase punto, tamaño de tarea espacial, 353
- tecla de función, 181, 341, 353
- teclado, 338, 341
  - alfanumérico, 180
  - codificado, 181
  - dispositivo lógico de entrada, 43, 49, 177
  - no codificado, 181
- teclas aceleradoras, 351
- tecleo adelantado, 46
- tecnología de grupos, 424
- tetera, 363, 400, 416
- textura, 486, 489
- texturas, correspondencia de, véase superficie, detalle de
- tijereteo, 80, 115
- tinta, color, 455, 468
- tinte, colores, 455, 464, 467, 469-470
- tipo de letra, véase carácter, tipo de letra
- tono, color, 455
- Torrance-Sparrow, véase iluminación, de Torrance-Sparrow**
- tramos, cálculo de, 107
- transformación, 185-220
  - afín, 199, 418
  - de cuerpo rígido, 199
  - de modelado, 266
  - de un sistema de coordenadas, 215
  - de vectores nominales, 214
- en modelado de objetos, 280, 292-297, 305
- geométrica, 152
- normalización de la, 247
- perspectiva, 261
- ventana-área de vista, 203-205
- visualización en SPHIGS, 287, 291

- transformación, composición de, 198, 201, 210  
 transformación, matriz de, 196, 201  
 transformación normalizada  
   paralela, 247-253  
   perspectiva, 253-257  
 transmisión, coeficiente ( $k_i$ ) de, 573, 580  
 transparencia, 490  
   ángulo crítico, 577  
   aproximación a  $k_i$  cambiante, 574  
   de mosquitero, 573  
   filtrada, 573  
   implantaciones con memoria de profundidad  $z$ , 574  
   interpolada, 573  
   no refractiva, 573  
   reflexión interna total, 577  
   refractiva, 575  
 transportabilidad, 13, 271  
   programas de aplicación, 289  
 transpuesta, véase matriz, transpuesta de una  
 traslación  
   bidimensional, 193  
   tridimensional, 207  
 traza de rayos  
   de datos, 334  
   de objetos, 348  
   limitaciones en el modelado, 331  
   modelado de objetos, 275-279  
 trazo, 9  
   dispositivo lógico de entrada, 43  
 tres estímulos, teoría del color, 457-459  
 tres estímulos, valores de, 458  
 tubo de rayos catódicos (CRT), 449, 450,  
   véase también mascarilla de tensión  
   plana, parpadeo, enfoque  
 mascarilla de sombra, 158  
 monocromático, 156  
 triángulo de precisión en línea, 159  
 triángulo-triángulo, 159  
 unidad de procesamiento de pantalla, véase controlador de pantalla  
 $V$  (dirección al punto de observación), 550  
 variable de decisión  
   para discretización de círculos, 95  
   para discretización de líneas, 85  
 vector intermedio ( $H$ ), 552  
 vector tangente, 373-381  
 vectores  
   ángulo entre, 189  
   normalización de, 189  
   proyección de, 189  
 vectorización, 182  
 velocidad paramétrica, 374  
 ventana  
   coordenadas mundiales, 203, 223, 231  
   de selección, véase selección, ventana de ventana-área de vista, correspondencia, 203  
 ventanas, administrador de, 8  
 vértices fantasma, curvas cúbicas, 391  
 vídeo compuesto, 176  
 vídeo, controlador de, 10, 164-167, 174-176  
 vídeo, mezclado de, 176  
 vídeo, RAM (VRAM) para, 166  
 vídeo, tabla de consulta de, 166, 450  
 visibilidad, determinación de la, véase barridos, generación de  
 visibilidad, en modelo geométrico, 328  
 vista, 19, 229  
   coordenada de referencia (VRC) de, 230  
   matriz de correspondencia de, 235  
   matriz de orientación de, 234-235  
   normal al plano (VPN) de, 230  
   plano de, 230-235, 237-240, 241-242, 246, 253  
   punto de referencia (VRP) de, 230  
   vector (VUP) de, 230  
 vista, volumen de, 224, 231, 242, 247  
   canónico, 247, 260  
 vista, operaciones de, 246  
   en SPHIGS, 288-292  
 vista, proceso de, 221-223  
 visual  
   agudeza, 452  
   continuidad, 351  
   realismo, 481  
 visualización científica, 6, 16, 493  
 Von Koch, copo de nieve de, 405  
 voxel, 431  
 voz, reconocimiento de, 342  
 Warn, controles de iluminación de, 552  
 Warnock, algoritmo de, 531, 537, 550  
 Whitted, 520, 524  
 X Window System, 15  
 YIQ, modelo de colores, 464, 466  
 z-buffer, algoritmo de, 510-514, 534

# Paquetes de software para graficación SRGP y SPHIGS

## Requisitos técnicos:

**Estaciones de trabajo con UNIX y sistema X Window:** X11 versión R4; un compilador de ANSI C (gcc es recomendable); ya sea BSD UNIX v4.3 o UNIX System V.

**Macintosh:** Cualquier modelo de Macintosh (color o monocromático) con un mínimo de 1 megabyte en RAM; se requieren 2 megabytes en RAM para ejecutar el depurador; System Tools v5.0 o posterior; System Software v6.0 o posterior; THINK C v5.0 de Symantec. Es muy recomendable disponer de un disco duro o acceso a THINK C vía Apple Talk.

**Microsoft Windows para IBM PC:** Cualquier computador de la familia IBM PC con microprocesador 80286 o superior, con un mínimo de 1 megabyte en RAM (memoria convencional y extendida combinadas); adaptador monocromático Hercules o, de preferencia, monitor a color EGA o uno mejor; al menos una unidad de disco flexible y disco duro; MS-DOS v3.1 o posterior (se recomienda v5.0 o posterior); Microsoft Mouse o algún dispositivo apuntador compatible; compilador Microsoft de C/C++ v6.0 o el Turbo C v2.0 de Borland; Microsoft Windows v3.1; Microsoft Software Development Kit para Windows.

## Cómo ordenar:

El público en general puede comprar este software de Addison-Wesley, en versión para IBM PC o para Macintosh, al precio que se indica a continuación:

**Versión para Macintosh**  
0-201-60956-8 US\$15.00

**Versión para IBM PC**  
0-201-60957-6 US\$15.00

**Estaciones de trabajo con UNIX y sistema X Window:** Para obtener información sobre cómo adquirir el software vía ftp, envíe un mensaje por correo electrónico a "graphtext@cs.brown.edu" con el mensaje "Software-Distribution" en el renglón de asunto (*Subject*).

Para ordenar, fotocopie y complete la forma siguiente, incluya un cheque en dólares por \$15.00 (que incluye envío por correo aéreo) a nombre de Addison-Wesley Publishing Company, y envíelo a:

Addison-Wesley Publishing Company  
Attn: International Order Department  
Reading, MA 001867-9984

Please send me the following version of SRGP y SPHIGS:

Macintosh (60956)       Windows (60957)

I enclose check for US\$15.00

NAME \_\_\_\_\_

Nombre \_\_\_\_\_

COMPANY \_\_\_\_\_

Compañía \_\_\_\_\_

ADDRESS \_\_\_\_\_

Dirección \_\_\_\_\_

CITY \_\_\_\_\_

Ciudad \_\_\_\_\_

COUNTRY \_\_\_\_\_

Pais \_\_\_\_\_

ZIP \_\_\_\_\_

Código Postal \_\_\_\_\_