TEMA 7 - REPRESENTACIÓN DE SÓLIDOS

Al igual que un conjunto de líneas y curvas 2D no tiene necesariamente que describir un área cerrada, un conjunto de planos y superficies 3D no tiene por qué representar un volumen cerrado. Sin embargo en muchas aplicaciones es importante considerar a un objeto como un volumen perfectamente cerrado, y poder distinguir entre interior, exterior y superficie del objeto 3D. Así por ejemplo, en las aplicaciones de CAD/CAM en donde disponemos de un objeto modelado según su descripción geométrica, podemos calcular infinidad de operaciones sobre él antes de haberlo construido físicamente: puede determinarse si un objeto se solapa o intersecta con otro (un brazo robot puede esquivar los objetos del entorno), calcular volúmenes y centros de masa (en el diseño de un coche, éste debe ser lo más aerodinámico posible), calcular la resistencia a la presión o a la temperatura (mediante el método de elementos finitos), etc. Todas estas aplicaciones son ejemplos de **modelado de sólidos**.

En el mundo real todo está construido por átomos, los cuales se juntan formando moléculas, y así sucesivamente hasta formar objetos tales como manzanas, coches o este libro. Lo ideal sería que en el ordenador pudiera disponer igualmente de un elemento mínimo similar al átomo con el que construir todos los objetos posibles. Desafortunadamente esto no es así. La memoria del ordenador sólamente puede almacenar elementos binarios que representan números o símbolos, los cuales pueden emplearse para representar coordenadas, ecuaciones, tangentes, propiedades físicas, etc. Por lo tanto puede afirmase que no existe un esquema de modelado universal que permita construir todos los objetos presentes en la realidad (ni tampoco los abstractos).

Por ejemplo, un simple cubo se compone de 8 vértices, 12 aristas y 6 caras. Quizás la mejor forma de representación sería almacenar las coordenadas cartesianas de los 8 vértices, con los cuales podríamos reconstruir las aristas y caras. Pero ¿podríamos usar la misma técnica para representar una esfera? En realidad sí, pero deberíamos usar un enorme conjunto de vértices para poder representarla lo más aproximada posible. Una mejor técnica de representación sería usar su ecuación $x^2 + y^2 + z^2 < r^2$, donde incluso podríamos saber qué puntos pertenecen al interior, al exterior o a la superficie de la esfera. Por otro lado, ¿qué hay de objetos más complejos como el fuego, una planta, o una nube? La geometría convencional no puede con ellos, y se necesitan técnicas más complejas como sistemas de partículas o fractales.

Finalmente, consideremos el modelo de la famosa tetera de Utah, el cual tiene un cuerpo, una tapa, un asa y un caño por donde saldría el té. Pero si examinamos el interior de su geometría descubrimos que ¡no tiene agujero! Sin embargo, la pregunta es ¿realmente importa? Si estamos trabajando con una herramienta CAD y el objetivo final es construir la tetera, por supuesto que sí. Seguramente deberíamos poder calcular su peso, su centro de gravedad, el área de su superficie y su momento de inercia, por lo que es esencial disponer de una descripción lo más exacta posible. Pero si por el contrario estamos desarrollando una película de dibujos y la tetera es un elemento más del escenario, entonces no nos interesaría tanta exactitud. Resumientdo, esquemas de representación hay muchos, y cuál elijamos dependerá de la aplicación con la que trabajemos y del tipo de objeto.

7.1 MODELOS DE ALAMBRE

El método más simple de construir un objeto 3D es hacerlo a partir de una colección de líneas que representen los bordes rectos que identifiquen la geometría del objeto en cuestión. Tal método se conoce como modelo de alambre (**wire-frame**) ya que al dibujarlo en pantalla parece que está construido por trozos rectos de alambre. Por ejemplo, la representación de una mesa sería algo parecido a la figura 1:

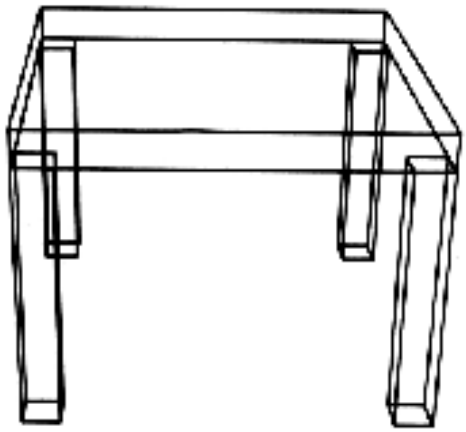

Figure 1

Debido a que la única información de que disponemos del objeto es el conjunto de aristas, es imposible para los programas eliminar las líneas que quedan ocultas por alguna superficie, cosa que ocurre en la realidad. Esto es porque no disponemos de datos referentes a las caras del objeto, por lo que la imagen resultante puede considerarse como una vista transparente del

objeto. Esta propiedad puede ser útil en algunas ocasiones. Por ejemplo, en las aplicaciones de diseño arquitectónico, como AutoCad, cuando se tienen muchos objetos en pantalla simultáneamente se hace muy lento trabajar debido a la complejidad de cada objeto, por lo que en la fase de interacción con el usuario, la vista en la pantalla muestra a todos los objetos en wire-frame, lo cual es mucho más rápido, y sólo en la imagen final es cuando mostramos los objetos completamente.

Si intentáramos mejorar la estructura de datos añadiendo información de cómo las aristas están conectadas para formar superficies, estaríamos hablando ya de otros modelos más completos: los modelos poligonales.

7.2 MALLAS POLIGONALES (MESHES)

Las mallas poligonales o meshes fueron el primer sistema de representación de objetos, ya que era el más evidente y práctico para las máquinas existentes cuando comenzaron las primeras aplicaciones gráficas. Todavía hoy en día se siguen usando, por supuesto, salvo en aquellas aplicaciones que precisen mayor exactitud tanto en el modelado como en los cálculos posteriores (áreas, volúmenes, etc.). Vamos a definir por malla poligonal a un conjunto de aristas, vértices y polígonos conectados donde cada arista es compartida al menos 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, y un vértices es compartido al menos por dos aristas.

Un mesh puede representarse de varias maneras, cada una con sus ventajas y desventajas. Es tarea del programador de la aplicación el elegir la representación más apropiada, o incluso es posible elegir varias: una para almacenamiento, otra para uso interno de la aplicación, otra para que el usuario interactúe con ella, etc. Las operaciones típicas que pueden hacerse sobre un mesh son encontrar todas las aristas incidentes a un vértice, encontrar los polígonos que comparten una arista o un vértice, encontrar los vértices conectados por una arista, encontrar las aristas de un polígono, visualizar el mesh, e identificar errores en la representación (como una arista o un vértice perdido). En general, mientras más explícita sea la representación, más rápidas serán las operaciones, pero más espacio de almacenamiento van a requerir.

7.2.1 Tipos de Representación

a) Representación explícita. Cada polígono viene representado por una lista de coordenadas de vértices:

$$P_i = ((x_1, y_1, z_1), (x_2, y_2, z_2), ..., (x_n, y_n, z_n))$$

Los vértices se almacenan en el mismo orden que si recorriéramos el polígono a traves de su contorno. Existe una arista entre cada dos vértices consecutivos del vector, y una arista más que conecta el último vértice con el primero.

Para un simple polígono, esta representación es bastante eficiente, pero para un mesh se pierde mucho espacio, ya que las coordenadas de los vértices compartidos se hallan duplicadas. Además, otra desventaja es que no hay representación explícita de las aristas y los vértices compartidos. Por ejemplo, para arrastar un vértice y todas sus aristas incidentes de forma interactiva, debemos primero encontrar todos los polígonos que comparten dicho vértice.

Para visualizar el mesh necesitaremos transformar cada vértice y recortar cada arista de cada polígono. Si dibujamos las aristas, aquellas que estén compartidas se dibujarán dos veces, lo cual puede causar problemas, aparte de tardar el doble de tiempo.

b) Punteros a una lista de vértices. Cada vértice del mesh se almacena una sola vez, en una lista de vértices:

$$V = ((x_1, y_1, z_1), (x_2, y_2, z_2), ..., (x_n, y_n, z_n))$$

Cada polígono se define como una lista de índices o punteros a la lista de vectores. Así, un polígono formado por los vértices 3,5,6 y 10 vendría representado por el vector

$$P_i = (3, 5, 7, 10)$$

Esta representación, que puede verse en la figura 1, tiene varias ventajas sobre el tipo anterior. Ya que cada vértice se almacena sólo una vez se ahorra mucho espacio. Por otro lado, la modificación de las coordenadas de los vértices se hace de forma directa, sin repercutir en la información de los polígonos. Sin embargo, aún sigue siendo complicado el encontrar los polígonos que comparten una arista, y las aristas compartidas continúan dibujándose dos veces.

c) Punteros a una lista de aristas. Seguimos teniendo una lista V de vértices, y además vamos a representar cada arista por separado como un nuevo vector, en donde se indica cuáles son sus dos vértices extremos y a qué dos polígonos pertenece:

$$P_i = (A_1, A_2, ..., A_n); \quad A_j = (V_1, V_2, P_1, P_2)$$

Cuando una arista pertenece a un solo polígono (en los contornos del mesh) P_2 se pone a nulo. Un ejemplo puede verse en la figura 2.

Para visualizar el mesh simplemente hay que dibujar todas las aristas, en lugar de dibujar los polígono, evitando así las transformaciones redundantes.

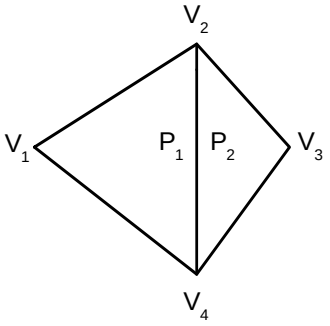

Figure 2

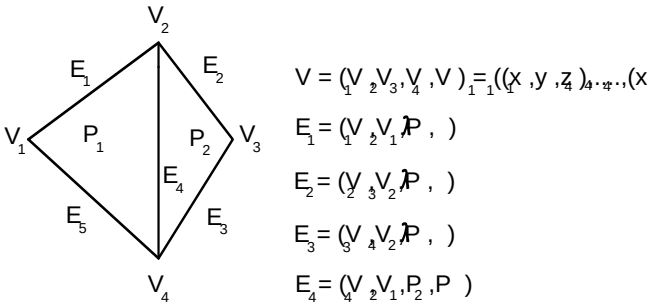

Figure 3

En ninguna de estas tres representaciones es fácil determinar qué aristas son incidentes a un vértice, pues habría que inspeccionar todas las aristas. Por supuesto que podemos añadir información que explícitamente indique tales relaciones, pero a cambio de pagar más coste de almacenamiento.

7.2.2 Consistencia de las representaciones

Los meshes suelen generarse interactivamente, a menudo de forma automática como resultado de digitalizaciones, por lo cual es inevitable encontrar errores en la representación. Por lo tanto se hace necesario asegurarse que los polígonos se encuentran todos cerrados, que todas las aristas son usadas al menos una vez, y que cada vértice es referenciado por al menos dos aris-

tas. En algunas aplicaciones incluso se exige que el mesh esté completamente conectado (cualquier vértice puede alcanzarse desde cualquier otro viajando por las aristas), que sea topológicamente plano (las relaciones binarias en los vértices definidas por las aristas puede representarse por un grafo planar) y que no contenga agujeros. De las tres representaciones anteriores, el esquema de punteros a aristas es el más sencillo de chequear, ya que contiene la máxima información.

7.2.3 La ecuación del plano

Cuando trabajamos con cada polígono a veces se hace necesario disponer de la ecuación del plano sobre el que se encuentra. En algunos casos dicha ecuación es conocida implícitamente debido al método de construcción interactivo que se ha usado para definir el polígono. Pero si no se conoce, siempre podemos usar las coordenadas de los tres vértices para encontrar el plano. La ecuación de un plano se define como:

$$ax + by + cz + d = 0$$

Los coeficientes a, b, c definen la normal al plano, $N = (a, b, c)$. Esta normal es fácil de calcular si disponemos de tres puntos, P_1, P_2, P_3, pertenecientes al plano. Sólo hay que evaluar el producto vectorial

$$N = \overrightarrow{P_1P_2} \times \overrightarrow{P_1P_3}$$

Si el producto es cero significará que los tres puntos son colineales y que los tres puntos no definen un plano, por lo que necesitaremos un vértice más. Una vez calculados los coeficientes a, b, c sólo queda calcular d, para lo cual evaluaremos la ecuación del plano en cualquiera de los tres puntos dados y la despejaremos.

Si hay más de tres vértices en el polígono, éstos pueden no ser coplanares, lo cual puede acarrear numerosos problemas. En estos casos es mejor usar otra técnica diferente para encontrar los coeficientes que mejor aproximan el plano a todos los vértices. Una vez calculados podemos determinar entonces una medida de lo "no-plano" que es nuestro polígono, en base a calcular las distancias perpendiculares al plano de cada vértice. La distancia D de un punto a un plano viene dada por la expresión

$$D = \frac{ax + by + cz + d}{\sqrt{a^2 + b^2 + c^2}}$$

Esta distancia puede ser positiva o negativa dependiendo de a qué lado del plano se encuentra el punto. Si el vértice se encuentra sobre el plano, $D = 0$.

7.2.4 Representación de objetos más complejos

En un caso más general, aquellas caras del objeto que no sean planas sino superficies curvas serán aproximadas también como nuevos meshes, como se ve en la figura 2.

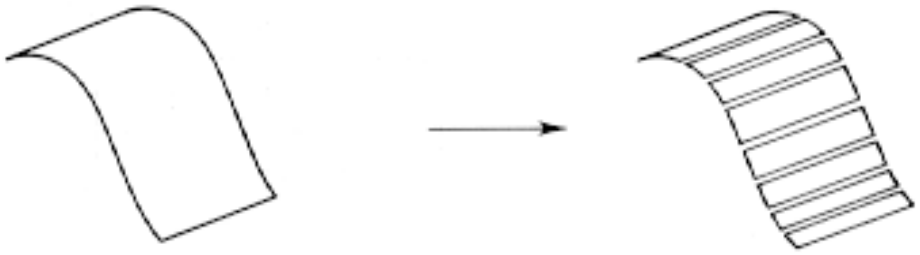

Figure 4

Este tipo de representación también recibe el nombre de **representación de fronteras** (boundary representation o **B-rep**) ya que es en realidad una descripción topológica y geométrica de la frontera o superficie del objeto. Es un esquema muy utilizado en Informática Gráfica, entre otras razones porque el modelado de objetos poligonales es simple de realizar. Sin embargo existen ciertas dificultades prácticas. La exactitud del modelo, es decir, la diferencia entre la representación facetada y la superficie real curva del objeto suele ser arbitraria. Para lograr una buena calidad en la imagen, el tamaño de los polígonos individuales debe depender de la curvatura espacial local. Donde la curvatura cambie rápidamente se necesitarán más polígonos por unidad de área de la superficie. Este factor debe ser tenido en cuenta a la hora de construir los polígonos.

La representación poligonal no sólamente se usa como estructura de datos al modelar, sno también como forma intermedia de otros muchas estructuras más complejas. Esto es así debido a que esta representación ha sido la técnica tradicional durante muchos años, y la mayoría de los algoritmos desarrollados en el campo de los gráficos trabajan con esta representación. Por ejemplo, los algoritmos de iluminación de polígonos han sido tan optimizados que algunos tarjetas gráficas de ordenador ya los llevan implementados en hardware. Por lo tanto, si disponemos de un objeto definido mediante patches bicúbicos como los que veíamos en el capítulo anterior, una buena estrategia de visualización sería convertirlos a mallas poligonales y mandarlos directamente al hardware.

En el caso más sencillo, un mesh poligonal es una estructura de datos que consiste en una lista de polígonos representados por las coordenadas (x, y, z)

de sus vértices. Ya vimos en el capítulo anterior los distintos subtipos de representación posibles. Además de esta información podemos almacenar como parte de la representación del objeto otra información geométrica que pueda ser usada en procesos de cálculo posteriores. Por ejemplo, ya hemos dicho que para muchos procesos es necesario conocer las normales a un punto de la superficie. Si las calculásemos todas inicialmente y las almacenáramos en la propia estructura de datos, los cálculos posteriores serían más rápido. Por supuesto, la contrapartida es que nuestro modelo requeriría más espacio de almacenamiento.

Otra idea conveniente puede ser ordenar los poligonos en una estructura jerárquica, como en la figura 3, en donde los polígonos se agrupan en superficies, y éstas en objetos. De esta manera, un cilindro posee tres superficies: dos caras planas arriba y abajo y una superficie curva. La razón para esta agrupación es que así podemos distinguir entre aristas que son parte de la aproximación (las aristas entre rectángulos adyacentes sobre la superficie curva del cilindro) y aristas que existen en la realidad. Esto puede ser muy útil por ejemplo para los algoritmos de iluminación a la hora de sombrear sin cambios abruptos la pared del cilindro.

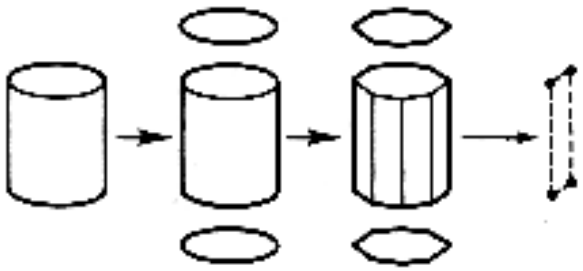

Figure 5

La característica importante de esta representación es que los polígonos son entidades independientes entre sí y por lo tanto pueden ser tratados individualmente. Es decir, calcular el área de la superficie de un objeto sería tan fácil como calcular el área individual de cada polígono y sumarlas todas.

7.2.5 Modelado de objetos poligonales

Aunque el mesh poligonal es la forma de representación más común en informática gráfica, el modelado aunque es simple es también tedioso. La popularidad de esta representación ha sido debida a la facilidad para modelar, la aparición de técnicas de iluminación para objetos poligonales, y el hecho

importante de que no existe restricción alguna de la forma o complejidad del objeto que se está modelando.

Las estrategias de modelado son principalmente de fuerza bruta. Una vez tenemos una aproximación inicial del objeto a modelar, interactivamente vamos moviendo los vértices y estirando y encogiendo los polígonos hasta lograr el objeto final, bien de forma ordenada mediante código o macros, o bien de forma arbitraria. Para poder conseguir la aproximación inicial hay varias estrategias dependiendo del objeto y del material de que dispongamos:

a) **Modelado manual**. La forma más fácil de modelar un objeto real es manualmente mediante un digitalizador 3D. El operador dispone de una especie de lápiz con el que va tocando la superficie del objeto. El ordenador detecta la coordenada (x, y, z) de la punta del lápiz en el espacio y almacena ese punto. El usuario usa su experiencia y su juicio para colocar los puntos sobre el objeto donde quiere que vayan los vértices de los polígonos, aumentando la densidad en las zonas de alta curvatura. Una vez en el ordenador todos esos puntos se genera una red que forma la superficie del objeto. Donde las líneas curvas de la red intersecten se define la posición de los vértices de los polígonos.

b) **Generación automática**. Un dispositivo capaz de crear una malla poligonal de alta resolución de un objeto real es un scanner laser 3D. Se coloca el objeto sobre una tabla rotatoria delante del rayo. La tabla además se mueve verticalmente. Cuando empieza el proceso, el rayo va detectando un conjunto de contornos (la intersección del objeto con un conjunto de planos paralelos muy cercanos entre sí) a diferentes alturas, midiendo la distancia a la superficie del objeto. Un algoritmo de "skinning" (skin = piel) va procesando cada par de contornos convirtiéndo los datos de la superficie en un alto número de polígonos. En la figura 4 se aprecia la evolución de este algoritmo, y también un caso concreto: una cabeza de estatua poligonizada con esta estrategia usando 400.000 poligonos.

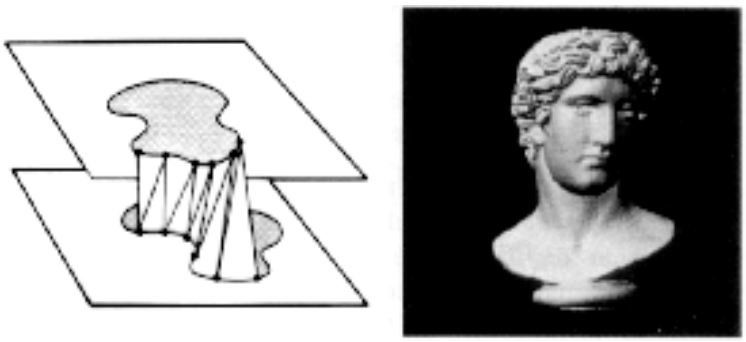

Figure 6

c) **Modelado matemático**. Está claro que cuando disponemos de la descripción matemática del objeto a modelar, lo más fácil es usar un algoritmo que vaya evaluando la ecuación en distintos puntos y considerando cada resultado como el vértice de un futuro polígono. La resolución de dichos polígonos se controla también directamente por el algoritmo de generación, por lo que habrá que tener en cuenta en qué zonas deben evaluarse mayor cantidad de puntos.

d) **Extruding**. Ésta es una de las técnicas más rápidas para construir objetos con forma de paralelepípedos. Partiendo de un polígono, el cual consideraremos que representa un corte transversal de nuestro objeto, lo desplazamos a lo largo de un eje de coordenadas y ya tenemos el objeto, como el que aparece en la figura 5.

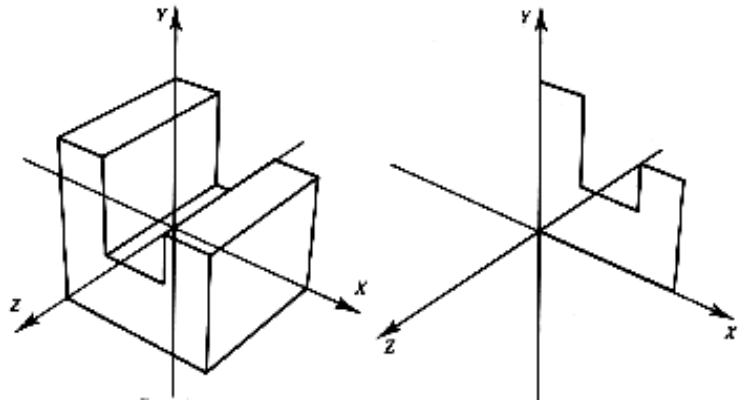

Figure 7

La tarea de construir la superficie corre a cargo del programa correspondiente que la realiza de forma automática. Todo lo que se requiere conocer es simplemente las coordenadas de los vértices del polígono que representa la sección transversal, y la longitud de la extrusión.

Un desarrollo de esta técnica consiste en construir no sólamente un polígono al principio y otro al final, sino una familia entera de polígonos a diferentes distancias discretas, a la vez que el polígono va siendo rotado alrededor del eje. Esto produce objetos "twisted" (¿retorcidos?) como el que se ve en la figura 6.

Y todavía podemos seguir complicándolo, haciendo que incluso el propio polígono vaya cambiando de forma a medida que nos vamos moviendo. Idealmente, un buen programa debería permitir al usuario especificar las secciones inicial y final (pudiendo ser distintas entre sí) y un ángulo de rotación para aplicar a la sección en cada etapa.

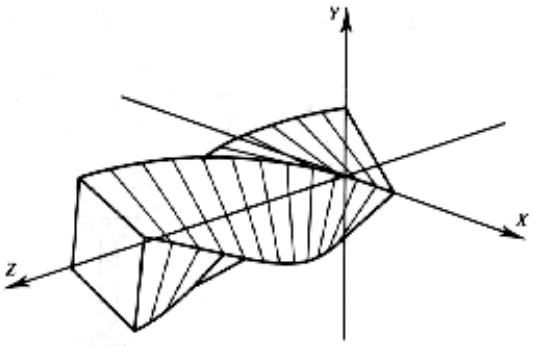

Figure 8

e) **Sweeping**. Es una extensión del extruding. Si en esta técnica íbamos moviendo el polígono por uno de los ejes, ahora vamos a moverlo a lo largo de una curva arbitraria en el espacio (por ejemplo cualquier curva cúbica paramétrica). De esta manera podemos construir objetos aún más variados, como el que muestra la figura 7.

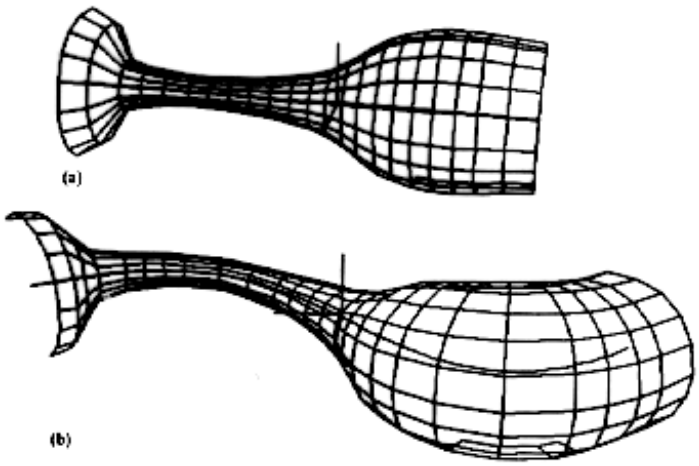

Figure 9

Sin embargo existen ligeros problemas que no aparecían en el extruding. Consideremos el simple caso de mover una sección transversal constante a lo largo de la curva $Q(t)$. Para ello habrá que definir intervalos en la curva en los cuales ir colocando el polígono, y luego ir uniendo vértice a vértice con el polígono anterior mediante líneas rectas. Las cuestiones que se plantean son: ¿qué intervalos debo escoger?, ¿cuál es la orientación del polígono a medida que se mueve por la curva?

En cuanto a la primera pregunta, dividir el parámetro t en intervalos

iguales no tiene por qué dar los mejores resultados. Además, intervalos iguales en t no representan necesariamente intervalos iguales en la curva[1]. Lo ideal es que los intervalos dependan de la curvatura de cada zona de la curva. Si ésta es alta debe existir un número alto de polígonos para representar con detalle dicha zona. La forma más directa de hacer esto es usar el algoritmo de subdivisión de la curva hasta que consigamos tramos planos, y colocar un polígono entre cada dos tramos.

En cuanto a la segunda pregunta, necesitamos definir un sistema de referencia que viaje por la curva, y que mantenga el polígono siempre perpendicular a ésta. Para lograr esto necesitamos definir tres vectores ortogonales entre sí que formen los ejes de coordenadas. Uno de ellos será la tangente en t a la curva, y otro indicará en todo momento la vertical con respecto a la curva. El tercer vector es simplemente el producto vectorial de los dos anteriores, tal y como se muestra en la figura 8.

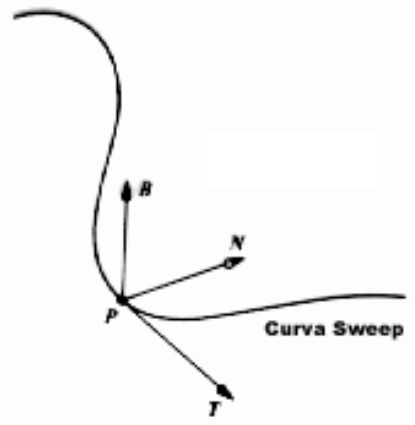

Figure 10

f) **Superficies de Revolución**. Esta técnica es la idónea para construir objetos tales como vasos, esferas, botellas, y en general aquellos que tengan simetría con respecto a un eje central. Para generar dicho objeto sólo es necesario un polígono que representará el contorno 2D de un lado de la figura. Dicho polígono es rotado 360^o alrededor de un eje hasta lograr una superficie cerrada que represente al objeto. En la figura 9 puede verse un ejempo de construcción de una copa de vino:Fíjense que la superficie final se ha construido en base a polígonos, y el número de ellos depende del número de vértices existentes en la curva original del contorno, y del incremento del ángulo en cada etapa de la construcción.

Al igual que ocurría con las dos técnicas anteriores, las superficies de rev-

[1]Recordemos que t podía considerarse como la velocidad a lo largo de la curva. Valores de t igualmente distanciados no implican puntos de la curva igualmente distanciados.

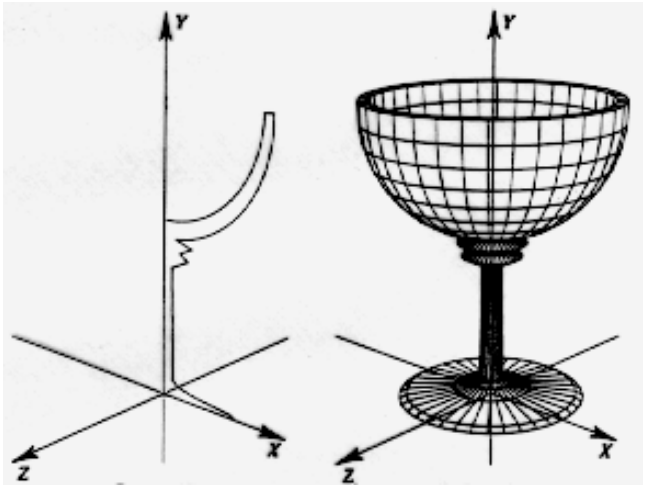

Figure 11

olución puede construirse automáticamente. El usuario solo debe especificar la curva que representa el contorno, el número de incrementos durante la rotación, y el eje sobre el cual se va a hacer la construcción. Tal programa sólo fabricaría objetos simétricos, aunque podrían crearse mejoras, como por ejemplo que en la etapa de construcción, la curva que identifica el contorno fuera variando durante la rotación, o incluso que el propio eje de rotación fuera moviéndose a lo largo de una curva arbitraria. El límite está en la imaginación.

7.3 PATCHES BICÚBICOS

Si consideramos la representación anterior mediante una malla de polígonos, y sustituimos cada uno de ellos por un patch bicúbico de los que veíamos en el tema anterior, estaremos hablando entonces de una red de patches (**patch mesh**). Cada patch es una superficie curva $Q(s,t)$ donde $0 \leq s,t \leq 1$, los cuales mantienen algún tipo de continuidad entre sus patches vecinos.

Para la mayoría de las aplicaciones, el modelar o construir una estructura de datos que represente un objeto tridimensional es más difícil si se usan patches bicúbicos que usando representación poligonal, ya que usando un digitilizador 3D o un scanner con el software apropiado ya hemos visto que podemos generar rápidamente gran cantidad de polígonos que representen objetos muy complejos. Cuando trabajamos con patches, suponiendo que éstos sean de Bezier, necesitaremos 16 puntos de control para cada patch, y además debemos mantener la integridad de la representación, en cuanto a mantener los requisitos de continuidad entre patches. Por todo ello, la descripción de una malla de patches suele generarse de forma semi-automática, como veremos en la siguiente sección.

Pero a pesar de esta desventaja, la diferencia en calidad es la que garantiza su uso extendido en las aplicaciones de CAD. La representación es sencilla, y usando el software apropiado para poder modificar la posición de los puntos de control podemos ajustar la forma del objeto que estemos modelando. Esto puede verse en la figura 10.

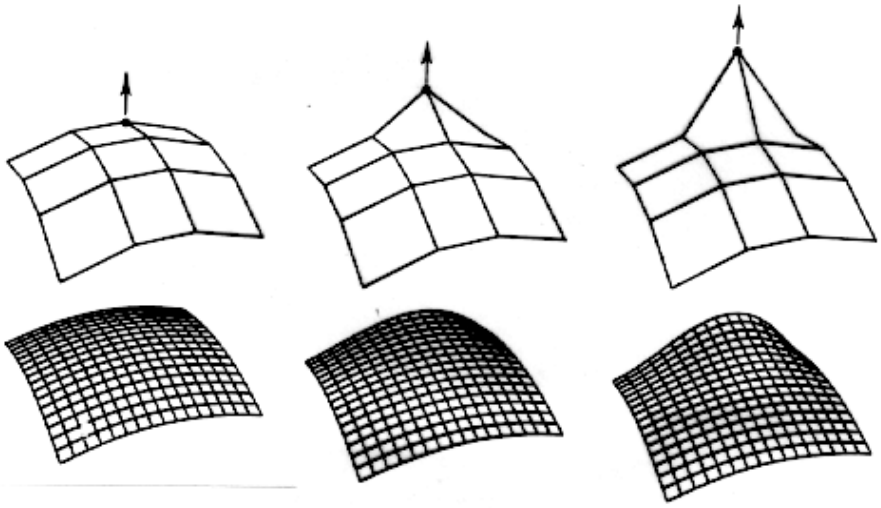

Figure 12

Otra gran ventaja es que al tener una descripción analítica exacta de la superficie, las propiedades físicas tales como masa, volumen, área y momentos de inercia pueden extraerse de dicha descripción. Esta propiedad es totalmente aprovechada por las aplicaciones CAD. Por último, hay que mencionar que la representación de objetos de alta resolución por medio de polígonos necesita un alto coste de almacenamiento y de tiempo de cómputo. De hecho, en escenas con muchos objetos complejos casi debe ser tratado como una base de datos. Por el contrario, usando patches bicúbicos obtenemos una representación mucho más exacta con mucho menos coste de memoria.

Veamos un ejemplo donde se comparan ambas estructuras de datos, y por supuesto tomaremos como objeto modelo la famosa tetera de Utah. En la figura 11a) se han dibujado varias líneas con s y t constante para cada patch. El objeto está compuesto de 32 patches de Bezier, y se puede ver sombreado y con línea gruesa un patch concreto. En la figura 11b) se ven los puntos de control en wire-frame (la zona sombreada se corresponde con los puntos de control del patch sombreado). En la figura 11c) se muestra un wire-frame de los bordes de cada patch.

Como para cada patch son necesarios 16 puntos de control, para almacenar los 32 patches necesitaremos $32 \times 16 = 512$ puntos de control, aunque en

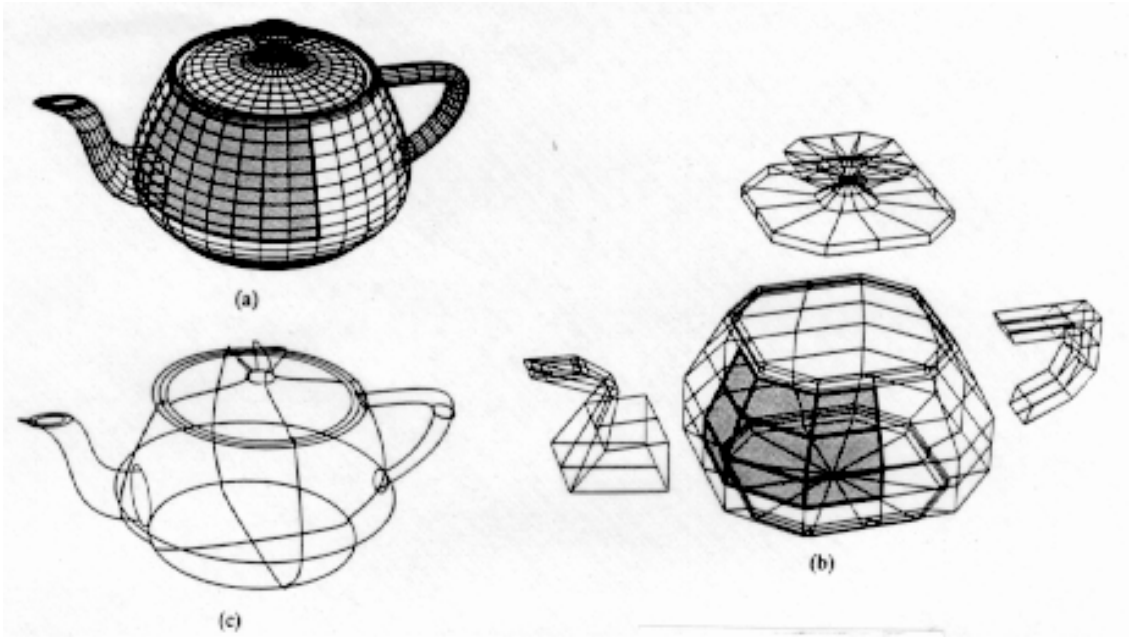

Figure 13

realidad son menos, ya que la mayoría de los patches comparten 12 puntos de control con sus patches vecinos. En este caso han sido necesarios 306 puntos de control. Cada punto requiere 3 coordenadas, donde cada una es un número real (4 bytes). Esto nos lleva a que necesitamos $306 \times 3 = 918$ números reales (3'6 Kb).

Ahora veamos lo que supondría mantener el mismo objeto mediante mallas poligonales. En primer lugar, una resolución equiparable visualmente a la del dibujo sería sustituir cada patch por 64 polígonos, y digo visualmente porque recordemos que usando patches bicúbicos podemos aumentar la resolución cuando queramos (los polígonos una vez elegidos son fijos) y además realizar cálculos exactos sobre la geometría del objeto. Pero bueno, pues aún así, requeriríamos $64 \times 32 = 2048$ polígonos, que a cuatro vértices por polígono nos salen $2048 \times 4 = 8192$ puntos, de 3 coordenadas cada uno: $8192 \times 3 = 24576$ números reales (96 Kb). Es decir, una representación "inexacta" de la misma tetera requeriría $96/3.6 = 26.6$ veces más de memoria.

Por si fuera poco aún existe otra ventaja adicional. Supongamos que queremos obtener una imagen de una cierta escena en la que hay numerosos objetos, entre ellos la tetera, y ésta queda bastante lejos del observador, por lo que sólo se proyectará sobre una pequeña área sobre la pantalla. Usando un modelo poligonal realizaremos un cierto coste computacional para dibujar el objeto en pantalla, y dicho coste será independiente del tamaño que éste ocupe en la imagen final. Pero si estamos usando una red de patches, y

además usamos un esquema de subdivisión que vaya convirtiendo los patches a polígonos hasta que éstos sean planos o hasta que su proyección se halle contenida en un único pixel de la pantalla, el coste en ese caso será mucho menor. Sólo hay que ver que el esquema poligonal estaría sobreescribiendo 2048 polígonos diferentes sobre los mismos pixels.

7.3.1 Modelado de objetos mediante patches bicúbicos

Existen dos estrategias prácticas de modelado que pueden usarse para obtener una aproximación inicial de patches, a la que luego habra que ajustarle sus puntos de control.

a) **Surface Fitting**. (¿ajustado de superficie?) Es básicamente un método particular de realizar interpolación de superficie. Se dispone de un conjunto de puntos que descansan sobre la superficie del objeto que queremos representar, y el objetivo es producir una descripción de los patches a partir de dichos puntos. De forma similar a cómo ajustamos una línea a través de un conjunto de puntos del plano, ajustaremos una superficie a través de un conjunto de puntos del espacio 3D. Así esta técnica puede usarse para modelar objetos de los que conocemos ya algunos puntos, bien sea porque el objeto es abstracto, o porque los hemos recogido previamente con un digitalizador.

Por supuesto, la diferencia entre esta técnica y obtener una representación poligonal con dichos puntos es que obtenemos una superficie continua a partir de los puntos. Sin embargo, hay que tener en cuenta el detalle que supone que en el caso de un objeto real, la superficie definida por los patches no coincida exactamente con la superficie de la cual se extrajeron los puntos digitalizados. La exactitud de la representación va a depender del número de puntos iniciales, así como de la extensión espacial de los patches en la red.

El principio del proceso es el siguiente: partimos de un conjunto de puntos 3D. El siguiente paso es ajustar una curva a través de los puntos en dos direcciones paramétricas perpendiculares, s y t. Esta red de curvas se divide en un conjunto de cuadriláteros curvilíneos, los cuales forman los bordes de los patches individuales. En cada cuadrilátero se crean los puntos de control necesarios para cada patch, y finalmente se construye la red de patches. En la figura 12 se ilustra el proceso seguido, así como un ejemplo en el que a partir de los puntos digitalizados de una cara se ha construido primero la red de patches, y luego se ha sombreado.

b) **Cross-sectional sweeping**. (¿?) De la misma forma que realizábamos un sweeping de un polígono alrededor de una curva arbitraria para generar un objeto polígonal, en este caso el polígono se convierte en una nueva curva cúbica. Esta curva, que representaría la sección transversal de la superficie a

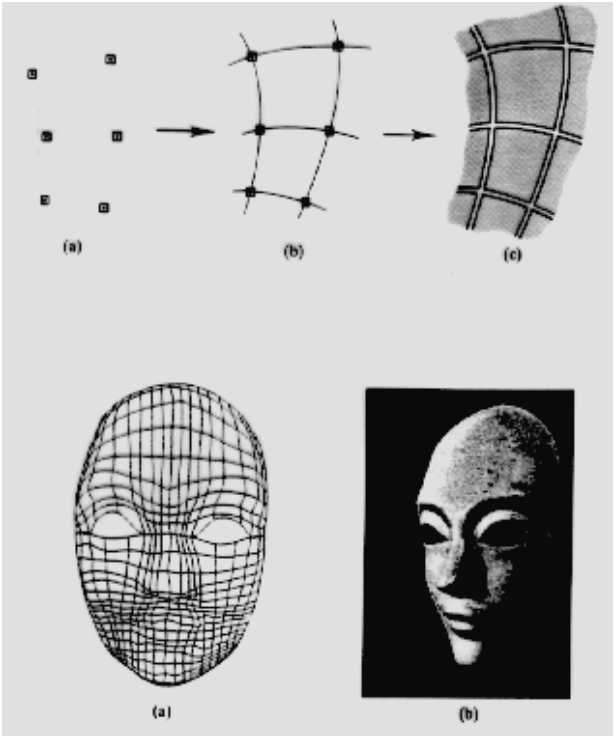

Figure 14

modelar, se va colocando a intervalos apropiados a lo largo de la curva que indica la trayectoria (curva de sweep), usando las mismas técnicas que comentábamos con anterioridad. En la figura 13 se ilustra este proceso, donde pueden verse dos curvas seccionales colocadas en dos posiciones consecutivas a lo largo de la curva de sweep. A partir de los dos extremos de estos dos segmentos generamos otras dos curvas. Estas cuatro curvas forman los bordes de un único patch, y de ellas obtenemos la descripción necesaria para dicho patch. Luego se siguen colocando más curvas seccionales.

7.3.2 Superficies Ruled (Ruled Surfaces)

Este tipo de superficies está a mitad de camino entre los patches cúbicos y los polígonos. En una de las direcciones parámetricas seguimos teniendo segmentos curvos, pero en la otra dirección sólo tenemos rectas, como se ve en la figura 14. Un ejemplo pueden ser las paredes de los cilindros y los conos. Un método conveniente para crear tales superficies es usar una aproximación paramétrica para las dos curvas, e ir interpolando linealmente el espacio entre ambas. Es decir, si $Q_1(t)$ y $Q_2(t)$ representan las dos curvas, siendo $0 \leq t \leq 1$, la superficie vendría dada por

$$Q(s,t) = (1-s)Q_1(t) + sQ_2(t), \text{ siendo } 0 \leq t, s \leq 1$$

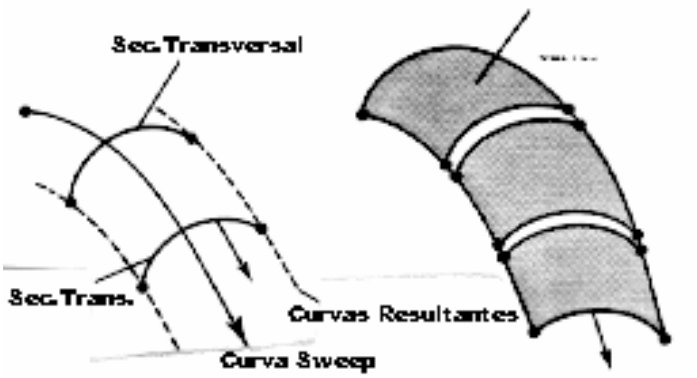

Figure 15

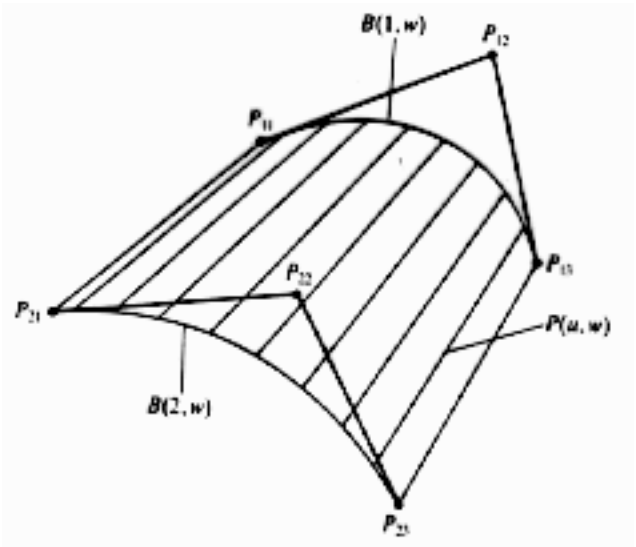

Figure 16

En realidad, las curvas de la dirección t no tienen por qué ser curvas cúbicas. Alguna de ellas podría ser una recta, o una semicircunferencia, y seguirían siendo superficies de este tipo. Incluso ambas podrían ser rectas, en cuyo caso recibirían el nombre de **superficies bilineales**.

7.4 GEOMETRÍA SÓLIDA CONSTRUCTIVA (CSG)

Los dos modelos anteriores, mallas de polígonos y de patches, pertenecen a la categoría de los esquemas de representación de fronteras. En estos esquemas la construcción de los objetos 3D se hace a partir de una descripción

de su superficie. Pero existen otra categoría de esquemas, conocidas como esquemas de **representación volumétrica**, que como su propio nombre indica describen el volumen interno al objeto, pudiendo por tanto almacenar propiedades del interior, tales como peso, presión interna, o incluso composición química.

La geometría sólida constructiva (CSG en inglés) es un tipo de representación volumétrica que facilita en gran medida la interactividad en el modelado de sólidos. La idea es llegar a construir los objetos mediante una combinación adecuada de volúmenes básicos elementales, conocidos como **primitivas geométricas**, las cuales pueden ser esferas, conos, cilindros, cubos, etc. Estas primitivas se combinan usando un conjunto de operadores booleanos y una serie de transformaciones lineales.

Cada objeto se almacena como un árbol. Los nodos hoja contienen a las primitivas, y los nodos no terminales almacenan los operadores booleanos o las transformaciones lineales que deben aplicarse. Debido a que la representación no sólamente define la forma del objeto sino además todas las etapas necesarias para su construcción, el realizar modificaciones a los objetos es muy sencillo. Por ejemplo, incrementar el diámetro de un agujero que atraviese un sólido rectangular implica una sencilla modificación (aumentar el radio de la primitiva cilindro). Esto constrasta con la representación por fronteras, donde efectuar la misma alteración no es nada trivial.

El conjunto de operadores booleanos que se usa para la representación también es usado como técnica del interface de usuario. Éste puede especificar primitivas y combinarlas usando estos operadores. La representación del objeto es una "grabación" de las operaciones que interactivamente ha ido ejecutando el usuario. Es por esto por lo que se dice que el modelado y la representación no están separados: el modelado se convierte en la representación. Veamos un ejemplo que demuestre tal afirmación, con la figura 15.

Existen distintos tipos de operaciones booleanas. En la figura 15 pueden verse las tres operaciones más frecuentes entre sólidos: la **unión**, que incluye todos los puntos que pertenezcan al interior de cualquiera de los objetos, la **substracción**, que elimina los puntos del primer objeto que pertenecen también al segundo, y la **intersección**, que sólamente mantiene los puntos pertenecientes a ambos objetos. Así, con estos tres pasos hemos creado el objeto de la figura de una forma super sencilla. Imaginen crear el mismo objeto usando una representación poligonal (o incluso alámbrica). Claro está que el tipo de objetos ideal para esta técnica son aquellos que poseen una geometría inherente clara, como por ejemplo todo tipo de piezas mecánicas. Las aplicaciones CAD/CAM son sus clientes más allegados.

En la figura 16 puede verse una representación que refleja la construcción

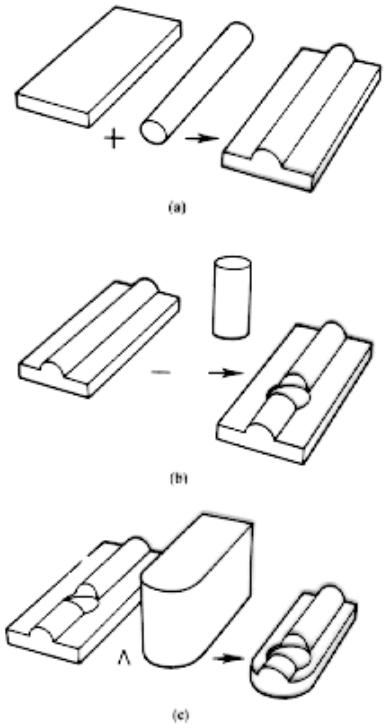

Figure 17

de un objeto simple. Aparecen tres primitivas en las hojas del árbol: dos cajas y un cilindro. Las cajas se combinan mediante una unión y luego se genera un agujero en medio definiendo un cilindro y luego restando. Obsérvese como las dos cajas de los nodos hojas son diferentes entre sí. ¿Significa esto que son dos primitivas diferentes? No, existe una única primitiva para las cajas, que puede ser un cubo unidad inicialmente. Cualquier otra caja, sea ésta más ancha o menos alta, se generará a partir de dicha primitiva mediante la transformación lineal correspondiente.

La figura 17 muestra dos ejemplos en los que puede apreciarse la verdadera potencia de esta técnica. En el primero de ellos, dos piezas desarrolladas por separado se combinan para formar la configuración deseada mediante un operador unión seguido por un operador de substracción. El segundo ejemplo muestra un objeto complejo construido a partir de la unión de varios cilindros, los cuales han sido restados a una esfera. Este segundo objeto es casi imposible de describir de forma tan exacta con ningún otro método.

Pero no todo son ventajas. Un problema práctico es el tiempo de cálculo requerido para producir una imagen del modelo. Otro problema es la limitación de las operaciones disponibles para crear y modificar el sólido. Las operaciones booleanas son globales, afectan al sólido completo. Una operación local, como una modificación detallada en una cara de un objeto

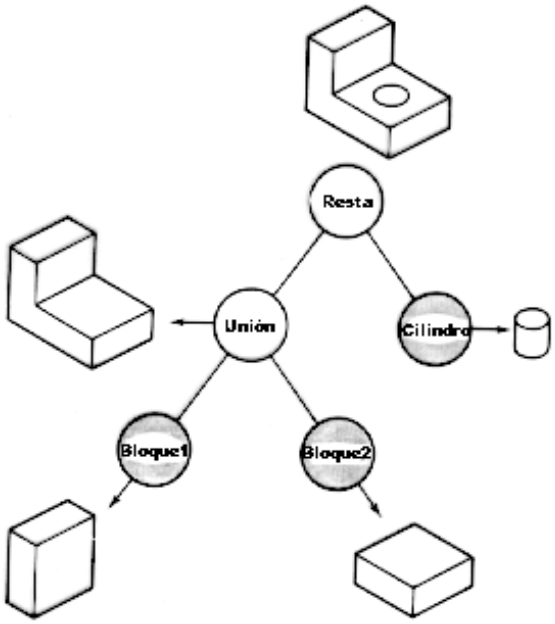

Figure 18

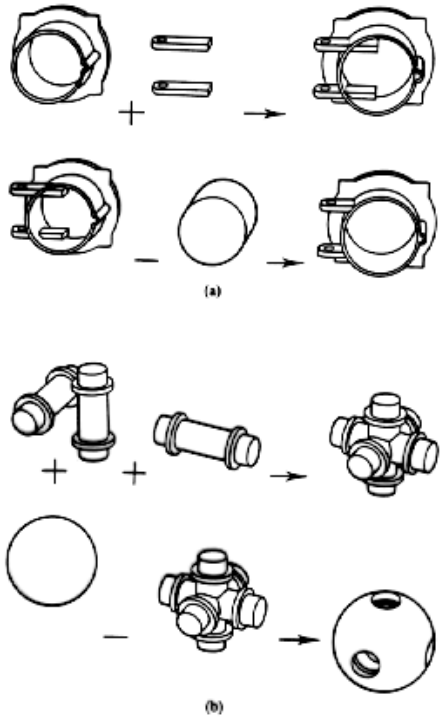

Figure 19

complejo no es fácilmente implementable usando el conjunto de operadores booleanos. Este hecho ha provocado que muchas aplicaciones usen B-rep internamente para la representación de los objetos. Es decir, aunque en la fase de construcción vayamos creando el árbol de operaciones, finalmente lo tranformamos en una malla poligonal para crear la imagen o almacenar la estructura de datos.

Una ventaja más: el esquema CSG permite de forma fácil examinar el interior de un objeto modelado con esta técnica. Esto se logra realizando la intersección del objeto con un plano de corte que lo divida en dos, con lo que podemos ver con detalle cualquier corte transversal. Esto se usa mucho en publicidad o en vídeos explicativos donde se quiera mostrar con detalle el interior de algún objeto.

Pero para poder lograr toda esta potencia, se necesita implementar algún mecanismo en el ordenador (lógicamente) que soporte la matemática y la geometría que lleva inherente esta técnica. Básicamente, se suele almacenar un sistema de inecuaciones para cada primitiva que indica para un punto de coordenadas (x, y, z) si se encuentra dentro o fuera del objeto. Por ejemplo, para una esfera con centro en el punto (x_c, y_c, z_c) y radio r es tán fácil como

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 \leq r^2$$

Para una caja serían necesarias una inecuación para cada una de las 6 caras, que indicara si un punto está por el lado de fuera o por el de dentro del plano que contiene a la cara. Un punto estaría dentro de la caja si verificase las seis inecuaciones. Con esta información, la aplicación de los operadores booleanos van combinando los sistemas de inecuaciones hasta llegar al objeto final.

7.5 TÉCNICAS DE SUBDIVISIÓN ESPACIAL

Este tipo de técnicas son métodos que consideran el espacio completo del entorno del objeto. Dicho espacio es subdividido en zonas o celdas adjuntas y no intersectantes, y cada uno de esas zonas o celdas es etiquetada con el objeto al que pertenece (si es que pertenece a alguno). Suele usarse como estructura de datos secundaria o auxiliar. Por ejemplo, generar la imagen de un objeto representado por un árbol CSG directamente no es tarea fácil. Una alternativa válida puede ser convertir el árbol a un esquema de subdivisión espacial y generar la imagen a partir de éste último.

Dentro de este tipo de técnicas existen varios modelos.

7.5.1 Descomposición en celdas

Es una de las formas más generales de subdivisión espacial. Se define un conjunto de celdas primitivas básicas, con las cuales se forma el objeto (es como jugar al Lego). Estas celdas no pueden solaparse, y el único operador booleano permitido es la unión. La estructura de datos que representa al objeto es simplemente un array de celdas (cada celda con su posición y tamaño). Un ejemplo puede verse en la figura 18.

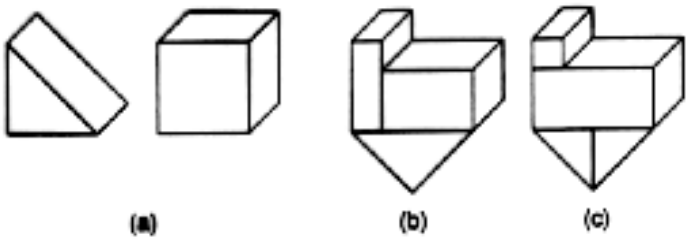

Figure 20

7.5.2 Enumeración de la ocupación espacial

Es un caso especial de descomposición en celdas, en la cual el objeto se descompone en celdas idénticas alineadas según una malla regular. Estas celdas reciben el nombre de **voxels**, por su similitud con los pixels. El proceso es muy parecido. Representar mediante pixels un círculo en la pantalla es tan fácil como discretizar la pantalla (dividir toda la pantalla en una malla de pequeños cuadraditos) e indicar para cada pixel si pertenece o no al interior del círculo (por ejemplo los que pertenezcan se pintarán de un color y los que no de otro). De forma similar se realiza el proceso 3D: el espacio alrededor del objeto es discretizado en una malla 3D de pequeños cubitos, y cada cubito es etiquetado indicando si pertenece o no al interior del objeto. La figura 19 muestra un donut representado con esta técnica.

El voxel típico suele ser un cubo, pero no es el único tipo de primitiva que podemos usar. A diferencia de la técnica anterior de descomposición en celdas, sólo existe un tipo de primitiva, y tanto la posición como el tamaño de cada una de ellas es único e invariante, y vendrá definido por la resolución de la malla. Para cada voxel solamente podemos indicar ausencia o existencia de algún objeto de forma booleana. Es decir, o pertenece en su totalidad o no pertenece. No vale decir que la mitad derecha del voxel pertenece al objeto y la otra mitad no. Es por eso que en la figura anterior se nota el efecto escalera o aliasing en la frontera del objeto. La única manera de

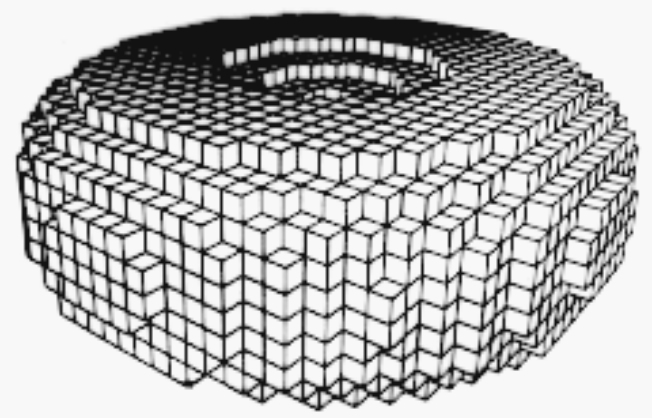

Figure 21

evitarlo es aumentando la resolución de la malla de voxels, haciendo éstos más pequeños. Para representar un objeto sólo hay que decidir qué voxels pertenecen al objeto y cuáles no. De esta manera el objeto vendrá especificado por un array de voxels ocupados.

Usando esta técnica es fácil saber si un punto del espacio pertenece o no a un objeto, o determinar cuándo dos objetos son adyacentes. Esta propiedad es explotada en la mayoría de las aplicaciones biomédicas para representar datos volumétricos obtenidos desde tomografías (TACs). La gran ventaja es que, al contrario que ocurría con los esquemas B-rep, tengo información sobre el interior de objeto. Por ejemplo, en la figura anterior del donut, los voxels que caen en el interior del donut no se ven, pero sí están almacenados en la estructura de datos. Por ello, en una aplicación médica puede tenerse por ejemplo un brazo modelado con esta técnica, y almacenar en cada voxel del brazo no sólo su pertenencia a éste, sino además a qué tipo de tejido pertenece (hueso, nervio, vena, etc.), con lo que el médico usuario del sistema podría visualizar en pantalla sólamente el tipo de tejido que deseara.

Entre sus desventajas está la ya comentada de la prohibición de una ocupación "parcial" dentro de un mismo voxel. Esto provoca que los únicos sólidos que pueden ser representados con total exactitud son aquellos cuyas caras sean paralelas a las caras de los voxels, y cuyos vértices coincidan exactamente con la malla. Al igual que ocurre con los pixels en un bitmap, las celdas pueden ser tan pequeñas como se quiera para aumentar la exactitud de la representación, a costa de un aumento importante de memoria, ya que para representar un objeto con n voxels de resolución en cada dimensión necesitaremos almacenar n^3 voxels.

7.5.3 Octrees

Los octrees son una variación jerárquica del esquema anterior, que intenta evitar el excesivo coste de almacenamiento. La idea es describir de forma jerárquica mediante un arbol octal la distribución de los voxels, en lugar de hacerlo mediante una lista exhaustiva de ellos. Está basado en la misma estructura que los quadtrees, empleados en el tratamiento de imágenes. Veamos primero en qué consisten los quadtrees y luego veamos su extensión a 3D. Supongamos que tenemos la imagen de la figura 20, y queremos obtener una representación de ella por medio de un quadtree. Se supone que la imagen representa una escena bidimensional a nivel de pixels.

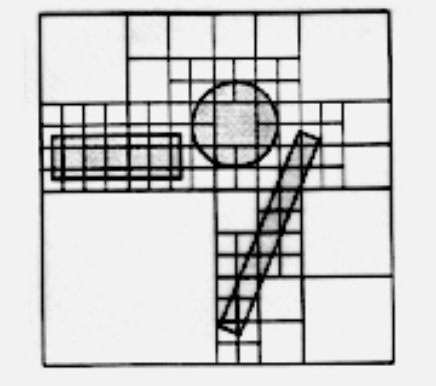

Figure 22

Comenzamos a crear el árbol considerando una región cuadrada que represente el espacio total de la escena (en 3D sería un cubo). Esta región vendrá representada por el nodo raíz del árbol. Como la región se encuentra ocupada por objetos en su interior, la subdividimos en cuatro subregiones, representadas por los cuatro nodos hijos en el árbol (en 3D serían 8 hijos). A continuación cualquiera de las subregiones que se encuentre ocupada por objetos se va subdividiendo recursivamente, hasta que el tamaño de la subregión se corresponda con la resolución máxima permitida (pixels en 2D y voxels en 3D). De esta manera, al final del proceso tendremos un árbol que me representa la totalidad de la escena, como se ve en la figura 21. El orden en el que se almacenan los hijos debe ser establecido previamente.

Existen dos tipos de nodos hoja en el árbol. Unos corresponderán a subregiones que no contienen a ningún objeto, mientras que otras corresponderán a voxels de mínimo tamaño que se encuentran ocupados por parte de algún objeto. Obsérvese que en el caso 2D, los objetos se representan por su frontera, y el espacio de su interior se cuenta como espacio no ocupado. En el caso 3D, los objetos se representan por su superficie, y sólo podremos subdividir

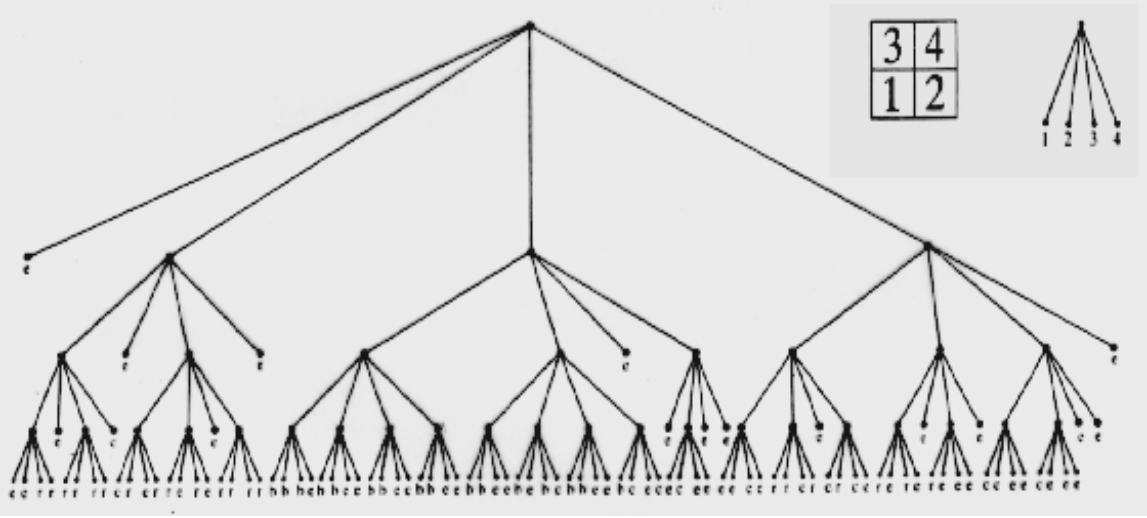

Figure 23

regiones que contengan parte de la superficie.

Una vez visto el quadtree, la definición para el octree es automática. El nodo raíz representa el volumen total de la escena. Cada nodo se divide en ocho subregiones (octantes), y recursivamente se van subdividiendo como en el caso 2D hasta que el volumen al que representan esté vacío o se llegue al tamaño mínimo para un voxel. Existen dos formas diferentes para representar una escena mediante un octree. La primera es tal y como acabamos de ver, obteniendo una representación completa de los objetos en la escena. El conjunto de voxels ocupados por el objeto constituye la representación del objeto. Esta forma viene muy bien para escenas simples con pocos objetos.

Sin embargo, para una escena compleja, se requiere un alto coste computacional para descomponer el espacio ocupado en un largo número de voxels, así como un enorme volumen de memoria. Una alternativa común es usar una estructura de datos estándar para representar a los objetos, y usar el octree exclusivamente para representar la distribución de los objetos dentro de la escena. En este caso, un nodo hoja que representase una región ocupada indicaría un puntero a la estructura de datos de cualquier objeto que intersectase la región. Esta alternativa se muestra en la figura 22, aunque usando quadtrees para verlo más fácil.

En este caso, el proceso de subdivisión finaliza cuando la región ocupe sólamente un objeto. Una región representada por un nodo hoja no tiene por qué estar completamente ocupada por el objeto asociado a ella. La forma del objeto en el interior de la región vendrá descrita por su representación en la estructura de datos correspondiente, la cual vendrá compuesta por polígonos o patches, en el caso de estar trabajando con modelos de superficie para

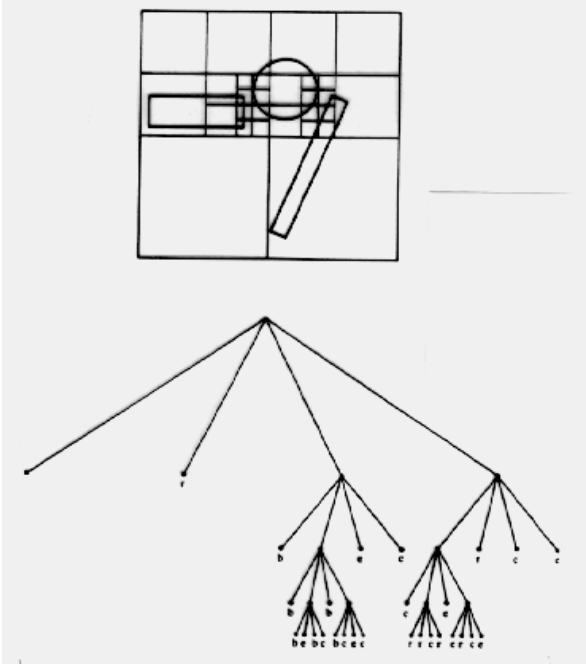

Figure 24

los objetos. En general, una región ocupada representada por un nodo hoja podrá intersectar con varios polígonos, en cuyo caso vendrá representada por una lista de punteros a las estructuras de datos de los objetos.

7.5.4 Árboles BSP

Este tipo de árboles siguen siendo en esencia octrees, pero la estructura de links en el árbol es ligeramente diferente. Las siglas BSP vienen del inglés (como siempre) y significan partición espacial binaria. Es decir, en cada subdivisión dividimos la región en dos mitades, por lo que el número de hijos de cada nodo siempre es dos (los que tengan hijos). En la figura 23 se muestra un ejemplo, aunque de nuevo en 2D donde es más fácil de ver.

En dicha figura puede verse la representación para un solo nivel de subdivisión usando un quadtree, y la misma representación, que necesita dos niveles, usando un árbol BSP. En el primer nivel se ha dividido a lo largo del eje x, obteniendo la mitad superior y la inferior de la región inicial. En el segundo final, cada subregión se vuelve a subdividir por la mitad, pero esta vez a lo largo del eje y. La extensión a 3D es directa: haría falta un nivel más para subdividir a lo largo del eje z. Cada nodo no terminal del árbol representa un plano de corte que divide el espacio ocupado en dos mitades. Un nodo hoja representa una región que no ha sido subdividida y que con-

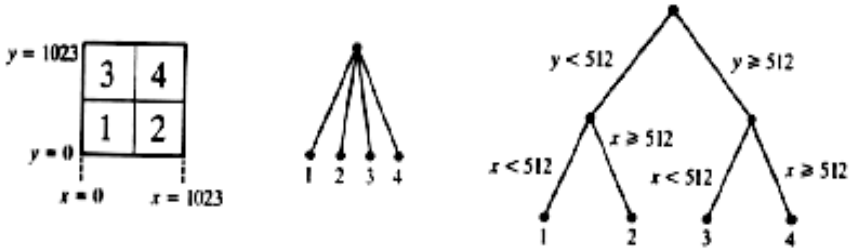

Figure 25

tiene punteros a las estructuras de datos que representan a los objetos que intersectan la región.

Subdivisión adaptativa

Cuando usamos un árbol BSP para representar una subdivisión del espacio en celdas cúbicas, en realidad no hay prácticamente diferencia con el hecho de usar un octree. Sin embargo, la verdadera utilidad de los árboles BSP es cuando no exigimos que las subdivisiones sean siempre en celdas cúbicas, es decir, que cada región se subdividad siempre en dos mitades iguales. De hecho, la idea original de este tipo de árboles consideraba el hecho de poder dividir el espacio usando planos de corte con cualquier orientación. Por ejemplo, existen aplicaciones que eligen los planos de corte para que en cada subdivisión separen dos objetos entre sí, dejando uno a un lado y otro al otro, con lo que el número de subdivisiones, y por tanto de niveles del árbol, es mucho más pequeño.

Por otro lado, normalmente los objetos de la escena no se encuentran uniformemente distribuidos dentro del espacio. Más aún cuando dichos objetos son los patches o polígonos que aproximan las superficies de los objetos reales. Un objeto real vendrá representado por un largo conjunto de patches conectados en el espacio, y existirán regiones extensas de espacio vacío entre los objetos. Cuando elijamos la posición para los planos en función de la distribución de los objetos en la escena, diremos que hemos aplicado subdivisión adaptativa para simplificar el árbol BSP.

En la figura 24 podemos ver un ejemplo de esta subdivisión frente a la subdivisión normal. Supongamos la región que se muestra conteniendo 16 objetos, etiquetados desde a hasta p. Estos objetos se encuentran distribuidos de forma no uniforme en el interior de la región. En la figura a) vemos el BSP tradicional, de forma similar a como lo haría un quadtree. La máxima profundidad de este árbol es 8, que además es la longitud máxima de búsqueda requerida para identificar a qué región pertenece un punto determinado. En la figura b) se ha usado subdivisión adaptativa. En cada paso, se ha elegido

como plano de corte aquél que divide la región de tal forma que a cada lado del plano existan el mismo número de objetos. Esto produce un árbol BSP más balanceado en el que la lóngitud máxima de búsqueda se ha reducido a 4.

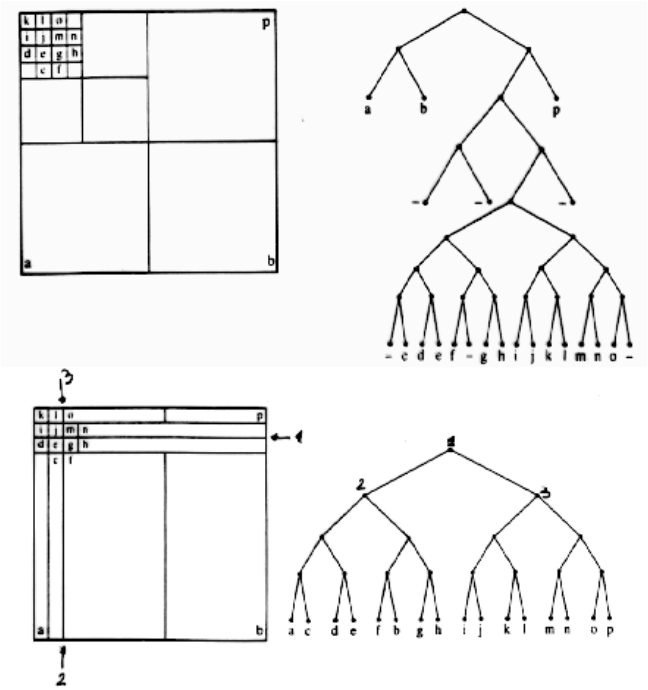

Figure 26