

CAPÍTULO 1

Visão Geral

- 1.1.** Computação Gráfica, Arte e Matemática
- 1.2.** Origens da Computação Gráfica
 - 1.2.1.** Escala Temporal
- 1.3.** Áreas da Computação Gráfica
 - 1.3.1.** O Mercado da Computação Gráfica
- 1.4.** Percepção Tridimensional
 - 1.4.1.** Informações monoculares
 - 1.4.1.1.** Perspectiva
 - 1.4.1.2.** Conhecimento prévio do objeto
 - 1.4.1.3.** Oclusão
 - 1.4.1.4.** Densidade das texturas
 - 1.4.1.5.** Variação da reflexão da luz
 - 1.4.1.6.** Sombras e sombreamentos
 - 1.4.2.** Informações visuais óculo-motoras
 - 1.4.2.1.** Acomodação
 - 1.4.2.2.** Convergência
 - 1.4.3.** Informações visuais estereoscópicas
- 1.5.** Representação Vetorial e Matricial de imagens
- 1.6.** Arquitetura de Hardware
 - 1.6.1.** Dispositivos Gráficos de Entrada
 - 1.6.2.** Dispositivos Gráficos de Saída
- 1.7.** OpenGL

O primeiro experimento matemático foi artístico: a música. Segundo a história, foi o próprio Pitágoras quem descobriu que as notas musicais se relacionavam com as razões do comprimento da corda do instrumento musical que as produziam ao vibrar (Maor, 1987). Como matemática, filosofia e música tinha a mesma importância no mundo Grego, nesta descoberta foram vistos sinais de que tudo no universo obedeceria às mesmas leis da música, ou seja, a razão. Assim, os números racionais dominaram a visão grega do mundo como o pensamento racional dominou sua filosofia. Na verdade, a palavra grega para razão é “logos” da qual se originou a atual lógica.

1.1. COMPUTAÇÃO GRÁFICA, ARTE E MATEMÁTICA

A computação gráfica é matemática e arte. É uma ferramenta de concepção de arte, assim como o piano ou o pincel. Esta ferramenta proporciona um maior poder de abstração, ajudando na criação de imagens complexas e em muitos casos não imaginadas. A relação entre luz, tempo e movimento constitui a base desta que poderia ser classificada como uma arte tecnológica. A computação gráfica pode ser encarada como uma ferramenta não convencional que permite ao artista transcender das técnicas tradicionais de desenho ou modelagem. Imagens que exigiriam do artista o uso de uma técnica apurada de desenho podem ser geradas mais facilmente com o auxílio de softwares. As ilustrações que usam técnicas de radiossidade ou *caustic* são belos exemplos. Nesses casos, as representações das sombras são extremamente difíceis de serem desenhadas à mão. Com o uso de software, o artista precisa ter apenas a idéia e deixar a parte complexa por conta da máquina. Contudo, esses softwares exigem um certo nível de conhecimento e treinamento que forçarão os artistas a uma complementação do estudo das técnicas de desenho tradicional, com a teoria da computação gráfica e matemática.

A computação gráfica proporciona um novo impulso ao artista, abrindo novos horizontes, fornecendo meios para se fazer um novo tipo de arte. Uma questão que surgiu após o relacionamento da computação gráfica com as artes foi a definição do seu verdadeiro papel na criação. Ela é apenas uma ferramenta do artista ou ela é responsável pela obra em si?

Consultando o dicionário Aurélio temos os seguintes significados da palavra arte: *palavra originária do latim, que significa: “saber; habilidade”. Capacidade que tem o homem de pôr em prática uma idéia, valendo-se da faculdade de dominar a matéria. Capacidade criadora do artista de expressar ou transmitir sensações ou sentimentos.*

Para melhor interpretação consultamos novamente o dicionário Aurélio e vimos o significado da palavra criar. *Criar* significa: “dar existência a, tirar do nada, dar origem, formar, produzir, inventar, imaginar”.

Segundo a ISO – (*International Organization for Standardization*), a definição de *computação gráfica* é: “um conjunto de ferramentas e técnicas para converter dados para ou de um dispositivo gráfico através do computador”.

A computação gráfica vista como ferramenta indicaria que temos um artista responsável pela arte gerada. Mesmo as imagens geradas a partir de equações podem ser consideradas arte, se essas equações forem fruto da criatividade e da capacidade do descobridor que manifesta sua habilidade e originalidade inventiva. Então vamos aproveitar para desmistificar uma questão. A matemática pode parecer um monte de números aglomerados em equações que se destinam apenas à construção de objetos concretos, mas isso não é verdade. Segundo Steve Hawking em seu livro *The Large Scale Structure of Space-Time*, “a matemática é a linguagem do homem com a natureza” e é exatamente aí que entram os computadores. A habilidade de simular a natureza em computadores tem sido objeto de atenção e curiosidade de toda a comunidade científica. Os fractais certamente são os melhores exemplos, compondo imagens intrigantes de realismo impressionante.

Talvez seja melhor notar que a relação entre a arte e a computação gráfica é simbiótica, uma interagindo com a outra, fazendo com que as duas evoluam de forma conjunta. A cada evolução da computação gráfica, podem ser abertos novos campos para as artes e vice-versa.

1.2. ORIGENS DA COMPUTAÇÃO GRÁFICA

Conhecer a origem é saber se posicionar na escala da evolução, descobrindo as necessidades, motivos e personalidades que alavancaram o desenvolvimento, para só, então, se projetar para um futuro real e imaginário.

Parece existir um consenso entre os pesquisadores de que o primeiro computador a possuir recursos gráficos de visualização de dados numéricos foi o *Whirlwind I*, desenvolvido pelo MIT. Esse equipamento foi desenvolvido, em 1950, com finalidades acadêmicas e militares. Em 1955, o comando de defesa aérea dos Estados Unidos desenvolveu um sistema de monitoramento e controle de vôos (*SAGE – Semi-Automatic Ground Environment*) utilizando o *Whirlwind I* como plataforma. O sistema convertia as informações capturadas pelo radar em imagem de um tubo de raios catódicos (na época, uma invenção recente), no qual o usuário podia apontar com uma caneta ótica para os pontos suspeitos.

Em 1959, surgiu o termo *Computer Graphics*, criado por Verne Hudson, enquanto o mesmo coordenava um projeto para a Boeing de simulação de fatores humanos em aviões.

Em 1962, surgiu uma das mais importantes publicações da computação gráfica de todos os tempos, a tese de Ivan Sutherland (*Sketchpad – A Man-Machine Graphical Communication System*), introduzindo as estruturas de dados para o armazenamento de hierarquias construídas através da replicação de componentes básicos, bem como as técnicas de interação que usavam o teclado e a caneta ótica para desenhar, apontar e escolher alternativas. Essa publicação chamou a atenção das indústrias automobilísticas e aeroespaciais americanas. Os conceitos de estruturação de

dados, bem como o núcleo da noção de computação gráfica interativa, levaram a General Motors a desenvolver em 1965 o precursor dos programas de CAD (Computer Aided Design). Logo depois, diversas outras grandes corporações americanas seguiram esse exemplo sendo que, no final da década de 1960, praticamente toda a indústria automobilística e aeroespacial utilizava softwares de CAD.

Na década de 1970, vários pesquisadores desenvolveram novas técnicas e algoritmos que são utilizados até hoje, tais como os métodos de sombreamento e o algoritmo de z-buffer. Nessa mesma época, surgiu a tecnologia dos circuitos integrados permitindo o barateamento das máquinas e o lançamento, em 1975, do primeiro computador com interface visual, o predecessor do Macintosh. Outros fatos importantes dessa década foram: o reconhecimento da computação gráfica como área específica da ciência da computação, o surgimento dos congressos específicos em computação gráfica (SIGGRAPH), a publicação do primeiro livro sobre computação gráfica interativa e o lançamento em 1977 do livro *Fractals: Form, Chance and Dimension*, onde o autor, Benoît Mandelbrot, matemático e, na época, pesquisador da IBM, conseguiu mostrar com imagens geradas em computador a incrível complexidade das equações fractais.

Em janeiro de 1980, a *Scientific American* publicou uma extraordinária imagem chamada Plume 2, a primeira imagem de uma erupção vulcânica no espaço, na lua Jovian tirada pela nave espacial Voyager 1. A foto era ao mesmo tempo um marco e um triunfo para a computação gráfica e mais especificamente o processamento de imagens, onde a imagem do telescópio recebeu um processamento utilizando técnicas da computação gráfica para permitir a visualização da erupção.

Imagens de satélite e de explorações interplanetárias são grandes usuárias das técnicas de processamento de imagem, que numericamente manipulam os pixels das imagens para reduzir ruídos, melhorar o contraste ou produzir um efeito desejado, como, por exemplo, salientar um aspecto em particular que mereça maior atenção.

A década de 1980 viu surgir diversas técnicas novas de iluminação global como o *ray-tracing* (em 1980) e a radiosidade (em 1984), aproximando as imagens geradas por computador do fotorrealismo. Outro fato marcante dessa década foi a estranha criação, em 1987, da cabeça falante Max Headroom, utilizada em um programa de TV da Inglaterra para simular expressões faciais humanas e apresentar o programa.

A década de 1990 marcou o amadurecimento da computação gráfica com imagens impressionantes como no filme *Jurassic Park*, em 1993. O filme marca a perfeição do fotorrealismo, nas cenas de movimentos dos dinossauros. Não poderíamos deixar de citar o filme *Terminator 2* com a utilização de um personagem computadorizado, e *Toy Story*, o primeiro longa metragem 3D, em 1995. Na área de sistemas, surge a linguagem de programação Open GL em 1992 e as primeiras placas gráficas para PC da NVIDIA, em 1999.

Com a virada para o ano 2000, a plataforma mais comum para atividades em computação deixa de ser as estações SGJ passando para PC. Em 2001, são lançados diversos sucessos de bilheteria, como *Shrek* (Dreamworks), com novos métodos de

síntese e animação de personagens e *Final Fantasy*, o triunfo da modelagem de personagens 3D; também não poderíamos deixar de citar *Matrix Reloaded*, com personagens virtuais sendo usados, dentre outras coisas, para cenas de risco.

1.2.1. Escala Temporal

A escala temporal nos ajuda a identificar oportunidades e direções de investigação e aplicação. Algumas das fundações que merecem destaque são:

Euclides [300-250a.C.] – desenvolveu toda a geometria que norteou seu desenvolvimento até o século XVIII.

Brunelleschi [1377-1446] – arquiteto e escultor italiano que usou de forma criativa a noção de percepção visual, e criou em 1425 a perspectiva.

Descartes [1596-1650] – matemático e filósofo francês que formulou a geometria analítica e os sistemas de coordenadas 2D e 3D.

Euler [1707-1783] – o mais produtivo matemático do século XVIII, que, entre outros, criou o conceito de senos, tangentes, a expressão que relaciona o número de vértices, arestas e faces de poliedros etc.

Monge [1746-1818] – matemático francês que desenvolveu a geometria descritiva como um ramo da geometria.

Sylvester [1814-1897] – matemático inglês que inventou as matrizes e a notação matricial, uma das ferramentas mais comuns da computação gráfica.

Hermite [1822-1901] – matemático francês que provou a transcendência do número e (usado como base para os logaritmos naturais) desenvolveu funções elípticas e curvas, entre outros.

Continuando nossa escala temporal, podemos identificar os aspectos de mudança que são considerados marcos da investigação científica e suas principais aplicações nas indústrias e na sociedade.

- Em 1885, iniciou-se o desenvolvimento da tecnologia do tubo de raios catódicos;
- Em 1927, a indústria cinematográfica define o padrão de 24 imagens/segundo;
- Em 1930, P. e W. Mauchly constroem o primeiro computador chamado ENIAC;
- Em 1938, Valensi propõe o tubo de raios catódicos colorido;
- Em 1947, os Bell Labs inventam o transistor;
- Em 1950, Laposky cria as primeiras obras de arte com bases tecnológicas, usando um efeito de um osciloscópio;

- Em 1955, surge o sistema Sage de monitoramento aéreo;
- Em 1956, o MIT constrói o primeiro computador totalmente transistorizado;
- Em 1959, surge o termo *Computer Graphics*, criado por L. Hudson da Boeing;
- No final da década de 1950, as universidades e empresas americanas, como a Boeing, começam a usar computadores para testar idéias e novas aplicações;
- Em 1960, é lançado o primeiro computador comercial DEC PDP-1;
- Em 1961, no MIT é criado o primeiro jogo de computador (*Spacewars*) para o computador DEC PDP-1;
- Em 1963, Sutherland apresenta um sistema de desenho interativo de primitivas gráficas 2D baseado em caneta luminosa;
- Em 1963, Englebart inventa o dispositivo de interação “mouse”;
- Em 1963, Zajac produz nos laboratórios da Bell o primeiro filme gerado por computador (imagens formadas de linhas e texto);
- Em 1963, surge o primeiro sistema comercial de CAD (*DAC-1*);
- Em 1963, Coons inventa a teoria de representação de superfícies curvas através de “retalhos” baseados em aproximações polinomiais;
- Em 1965, Roberts cria um algoritmo de remoção de partes invisíveis de segmentos de reta e introduz a noção de coordenadas homogêneas na representação geométrica de objetos;
- Em 1966, é lançado no mercado o primeiro console caseiro de jogos Odissey;
- Em 1966, surge a primeira empresa de produção computacional de animações e efeitos especiais, a MAGI;
- Em 1967, Rougelet cria o primeiro simulador de voo interativo da NASA;
- Em 1968, é fundada a Intel;
- Em 1969, a MAGI produz para a IBM o primeiro comercial baseado em técnicas de computação gráfica;
- Em 1969, é criado entre os grupos da ACM o Special Interest Group on Graphics SIGGRAPH;
- Em 1969, nasce a ARPANET, rede percussora da Internet;
- Em 1969, nos laboratórios da Bell, é construída a primeira matriz de pixels (cada pixel representado por 3 bits);
- Em 1972, A. Kay, no Xerox PARC, produz o computador gráfico Alto;
- Em 1972, Bushnell funda a empresa ATARI;
- Em 1973, Metcalf desenvolve a tecnologia Ethernet e é editado o primeiro livro que aborda detalhadamente os algoritmos e métodos da computação gráfica (autores Newman e Sproull);
- Em 1977, a Academia de Artes e Ciências Cinematográficas de Hollywood cria a categoria de Oscar de Efeitos Especiais;

- Em 1979, G. Lucas contrata Catmull, Ray Smith e outros para uma nova empresa denominada Lucas Film;
- Em 1974, Catmull desenvolve o algoritmo Z-Buffer;

A partir do algoritmo de Z-Buffer e com o lançamento do PC no início da década de 1980, surge uma infinidade de aplicações e filmes baseados em computador. O mercado da computação gráfica atinge seu estágio de maturidade apresentando um grande crescimento com produções realistas e técnicas avançadas de iluminação e modelagem. São exploradas outras possibilidades de geometrias além do espaço tridimensional, que são utilizadas com uma frequência cada vez maior pelas pessoas que trabalham com artes, computação e visualização científica.

1.3. ÁREAS

A computação gráfica atualmente é uma área que engloba, para melhor descrição didática, pelo menos três grandes subáreas: a **Síntese de Imagens**, o **Processamento de Imagens** e a **Análise de Imagens**.

A **Síntese de Imagens** considera a criação sintética das imagens, ou seja, as representações visuais de objetos criados pelo computador a partir das especificações geométricas e visuais de seus componentes. Pode também ser descrita como **Visualização Científica ou Computacional**, principalmente quando se preocupa com a **representação gráfica** da informação, de forma a facilitar o entendimento de conjuntos de dados de alta complexidade, como, por exemplo, os dados de dinâmica dos fluidos, ou simulações espaciais.

O **Processamento de Imagens** considera o processamento das imagens na forma digital e suas transformações, por exemplo, para melhorar ou realçar suas características visuais.

A **Análise de Imagens** considera as imagens digitais e as analisa para obtenção de características desejadas, como, por exemplo, a especificação dos componentes de uma imagem a partir de sua representação visual.

1.4. MERCADO

Se você puder imaginar algo, isso pode ser gerado com a computação gráfica. A computação gráfica está a um passo de um mundo novo, repleto de aplicações, ainda desconhecidas, e muitas oportunidades de trabalho para designers, modeladores, animadores, iluminadores e programadores.

Toda essa realidade impulsiona a computação gráfica para o desenvolvimento de diversas aplicações 3D, hoje restritas a jogos. Para isso, foram criadas em diversos países, inclusive no Brasil, ferramentas SDK (*Software Development Kit*) capazes de simular fenômenos físicos, facilitar a criação de cenários, entre outros infinitos recursos que podem ser criados por qualquer pessoa e acoplados à engrenagem do SDK sob a forma de *plug-in*.

Algumas das aplicações mais evidentes no momento são os mercados em realidade virtual e *walkthrough* para projetos arquitetônicos. Na comunidade científica, é consenso que em breve os ambientes 3D modificarão os atuais sistemas operacionais, os banco de dados e todos os componentes de interface que deverão ser recriados para esse novo mundo.

Programar para esse novo ambiente pode parecer um tanto hostil, exigindo dos programadores o conhecimento tanto de bibliotecas gráficas como *OpenGL* quanto da teoria da computação gráfica. Para designers, esse novo ambiente exigirá uma noção dos conceitos da computação gráfica e conhecimento de técnicas de modelagem, utilizando-se poucos polígonos ou operações *booleanas* e mapeamento com limitações no tamanho das imagens.

Diversos projetos já estão em desenvolvimento no Brasil, onde a mão-de-obra especializada é praticamente inexistente. A teoria da computação gráfica pode impulsionar sua carreira!

No mercado atual, a computação gráfica está presente em diversos segmentos, alguns dos quais são resumidos neste quadro:

| | |
|------------------------|--|
| Arte | Efeitos especiais, modelagens criativas, esculturas e pinturas |
| Medicina | Exames, diagnósticos, estudo, planejamento de procedimentos |
| Arquitetura | Perspectivas, projetos de interiores e paisagismo |
| Engenharia | Em todas as suas áreas (mecânica, civil, aeronáutica etc.) |
| Geografia | Cartografia, GIS, georreferenciamento, previsão de colheitas |
| Meteorologia | Previsão do tempo, reconhecimento de poluição |
| Astronomia | Tratamento de imagens, modelagem de superfícies |
| Marketing | Efeitos especiais, tratamento de imagens, projetos de criação |
| Segurança Pública | Definição de estratégias, treinamento, reconhecimento |
| Indústria | Treinamento, controle de qualidade, projetos |
| Turismo | Visitas virtuais, mapas, divulgação e reservas |
| Moda | Padronagem, estamparias, criação, modelagens, gradeamentos |
| Lazer | Jogos, efeitos em filmes, desenhos animados, propaganda |
| Processamento de Dados | Interface, projeto de sistemas, mineração de dados |
| Psicologia | Terapias de fobia e dor, reabilitação |
| Educação | Aprendizado, desenvolvimento motor, reabilitação |

Além dessas aplicações que citamos, ainda existe uma série de fenômenos que só podem ser vistos com o auxílio da computação gráfica. Há algum tempo, era prati-

camente impossível estudar certos tipos de comportamentos que fugiam à percepção humana. Com o advento da computação gráfica e o aperfeiçoamento de técnicas e métodos computacionais, vários desses fenômenos podem agora ser visualizados, modelados e estudados.

Muito antes de se conseguir visualizar um segmento de DNA humano, seu estudo já era possível através da computação gráfica. Em 1983, foi possível visualizar o vírus da AIDS e simular o seu comportamento. Através de dados científicos, foi criada uma imagem sintética que representava o vírus de forma adequada aos propósitos do estudo.

Em um outro extremo, a computação gráfica nos permite estudar elementos que possuem extrema complexidade como os Buracos Negros, objetos extremamente maciços que aprisionam tudo o que está próximo deles, inclusive a luz; ou objetos só realizáveis em dimensões superiores à quarta, como a garrafa de Klein; ou ainda as imagens adquiridas por sinais de galáxias distantes.

Quando a imagem real não é suficiente ou mesmo inviável, a imagem sintética toma o seu lugar. Por exemplo, podemos corrigir imperfeições causadas por ruídos, falta de luz ou distorções nas imagens transmitidas por satélites, visualizar uma nuvem radioativa, como a de Chernobyl, ou enxergar o que ocorre no interior dos profundos poços de petróleo. A imagem sintética pode mesmo transformar qualquer dado em imagem, como os sinais de radar ou calor dos corpos no interior de um prédio. Esse tipo de “visão” só pode ser realizada com o auxílio da computação gráfica.

A área médica encontra na computação gráfica uma poderosa aliada. É possível simular o corpo humano e obter conclusões a partir disso. Utilizando uma combinação de dados, como os de ressonância magnética, ultra-som, ou tomográficos, é possível reconstituir tridimensionalmente qualquer parte do corpo, focalizando seus elementos e possíveis doenças ou distúrbios. Na meteorologia, ciclones, tufões, deslocamentos de massas de ar, além do estudo do aquecimento global e da camada de ozônio, podem ser representados para estudos e previsões.

1.5. PERCEPÇÃO TRIDIMENSIONAL

Entender a forma como percebemos a profundidade em imagens bidimensionais servirá tanto para evitarmos erros na confecção da imagem, como para possibilitar uma interação amigável com objetos em ambientes virtuais. Outro motivo relevante para o entendimento está na limitação tecnológica que nos fará usuários, por ainda muito tempo, de telas de computador que, mesmo que estejam projetando uma visão estéreo, serão sempre imagens bidimensionais.

A percepção de “espacialidade” de uma imagem pode ser vista como a capacidade que o ser humano tem de distinguir a forma, as cores, a textura e a relação espacial existente entre os objetos de uma porção do mundo real.

Segundo Stuart (1996), há três categorias de estímulos visuais usados pelo cérebro para formar uma imagem 3D: informações monoculares, informações óculo-motoras e informações estereoscópicas.

1.5.1. Informações Monoculares

As **informações monoculares** são inerentes à imagem formada na retina, são também chamadas de *static depth cues* (informações estáticas de profundidade) [Auskalnis, 1992] ou *pictorial depth cues* (informações de profundidade na imagem) [Mckenna, 1992]. Entre as informações monoculares podemos citar a noção de perspectiva linear, o conhecimento prévio do objeto, a oclusão, a densidade das texturas, a variação da reflexão da luz e as sombras.

1.5.1.1. Perspectiva

A noção de perspectiva linear, ou, ao longo do livro, simplesmente perspectiva, é resultado da aparente diminuição dos tamanhos e das distâncias entre os objetos, à medida que o observador se distancia destes. Atualmente, esse recurso é largamente usado para expressar cenas tridimensionais em superfícies planas (papel, monitor de vídeo etc.).

Considerada como a mais importante descoberta no campo das artes, a perspectiva (descoberta por Filippo Brunelleschi em 1425) foi amplamente utilizada no final do século XV por artistas italianos do renascimento para a criação de obras realísticas [Maor, 1987], e foi levada à perfeição por Leonardo da Vinci em *A Ceia*. Essa obra trouxe a possibilidade de retratar a realidade com profundidade e máxima semelhança, permitindo alterações sutis para reforçar conceitos filosóficos e religiosos. Na época, eram usadas máquinas de fazer perspectiva, verdadeiras parafernâlias complexas para calcular as distâncias, tentando aproximar ao máximo o desenho na tela ao que o olho do artista vê.

Somente no início do século XIX, após inúmeros métodos e máquinas de cálculo da perspectiva, descobriu-se que a representação gráfica da perspectiva é feita como se somente um olho estivesse vendo a cena. Descobriu-se então que os olhos não vêem em perspectiva, mas por um fenômeno estereoscópico. Foi novamente a arte que marcou uma nova era para as imagens tridimensionais. Desta vez pelas mãos de um mestre francês Cézanne, que em suas pinturas produzia a representação do volume e da profundidade não pelos recursos geométricos da perspectiva, mas com a variação das tonalidades das cores e as direções das pinceladas.

Outra grande descoberta no século XIX foi a fotografia, tornando possível, pela primeira vez, gravar e reproduzir imagens da realidade por processos óticos e não dons artísticos. Isso fez com que a imagem saísse do isolamento do mundo artístico, tornando-a popular e uma das maiores fontes de informação da sociedade moderna.

1.5.1.2. Conhecimento Prévio do Objeto

O conhecimento prévio do tamanho de um objeto serve tanto para determinar a distância absoluta a partir do observador, quanto as distâncias relativas entre os objetos. Além disso, quando há dois ou mais objetos no mesmo campo de visão, e o observador tem noção de seus tamanhos reais, o tamanho aparente serve para determinar qual deles está mais próximo ou mais distante.

Quando olhamos o cubo à esquerda na Figura 1.1, conseguimos de imediato estabelecer uma noção da sua profundidade. Quando olhamos a imagem à direita na Figura 1.1, um galho de árvore, nosso cérebro não consegue perceber as profundidades. Essa limitação se deve ao fato de o objeto existir na natureza em uma infinidade de formas criando a ausência de uma referência para nos auxiliar na percepção. Esse problema pode causar uma grande limitação em sistemas de realidade virtual ou jogos quando interagimos com objetos nunca vistos, e pode causar percepções errôneas de tamanho e posição de objetos na cena.

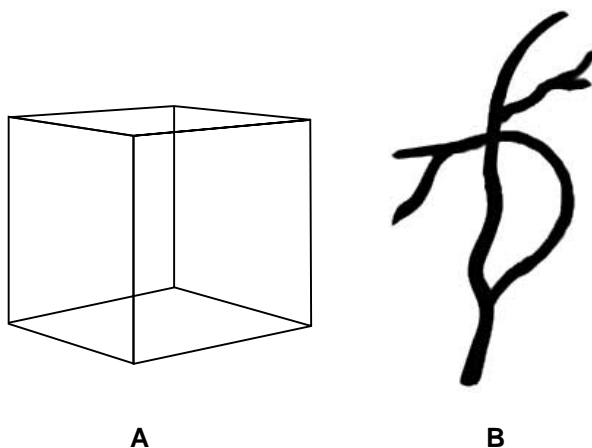


FIGURA 1.1. Limite da percepção de profundidade.

1.5.1.3. Oclusão

A oclusão é responsável pela informação da posição relativa dos objetos. Este fenômeno, também chamado de interposição ou interrupção de contorno, é descrito com a obstrução da visão de um objeto por um outro que está mais próximo do observador e sobre uma mesma direção de visão. Assim, quando um objeto A obscurece um objeto B, o cérebro sabe que o objeto A está mais próximo do que o objeto B.

1.5.1.4. Densidade das Texturas

Conhecido também como “gradiente de texturas”, este fenômeno visual baseia-se no fato de que muitos objetos possuem em sua aparência algum tipo de padrão

com uma certa regularidade. Nesse caso, à medida que os padrões aparecem mais densos e menos detalhados, mais distantes estarão do observador [Stevens, 1979]. As texturas também auxiliam na percepção do movimento, como, por exemplo, se girarmos uma esfera sem textura, nosso sistema de visão terá alguma dificuldade para perceber o seu movimento.

1.5.1.5. Variação da Reflexão da Luz

A mudança na intensidade da luz refletida ao longo de uma superfície de um objeto fornece informações sobre a forma e a curvatura da superfície desse objeto. Se não for gerada uma variação na cor dos pontos da superfície, a identificação do objeto pode se tornar difícil.

1.5.1.6. Sombras e Sombreamentos

Este efeito é útil na determinação da posição de um objeto em relação a um piso colocado abaixo deste, ou na definição relativa entre objetos. [Gibson, 1950].

1.5.2. Informações Visuais Óculo-motoras

Estas informações são fornecidas pelo movimento dos olhos, produzidos pelos dois conjuntos de músculos do globo ocular. Um desses conjuntos é formado por seis músculos, que se inserem ao redor do globo ocular, o circundam e o movem fornecendo informações do grau de contração para o cérebro humano. O segundo conjunto é responsável por focar os raios luminosos na retina (fundo do olho), mudando a curvatura da lente que fica atrás da íris, denominada cristalino. Há dois tipos de informações nessa categoria: acomodação e a convergência.

1.5.2.1. Acomodação

No processo de acomodação, os músculos ciliares dos olhos relaxam ou contraem para mudar o formato do cristalino (as lentes dos olhos), com o objetivo de alterar o foco dos objetos projetados na retina em função da distância desses objetos do observador.

1.5.2.2. Convergência

A convergência considera o grau de rotação dos olhos ao longo do eixo de visão (quando um objeto é focado) para obter informações a respeito da posição e da distância.

1.5.3. Informações Visuais Estereoscópicas

Como os olhos estão posicionados em lugares diferentes, cada um vê uma imagem de forma diferente. Essa diferença é chamada disparidade binocular. O cérebro usa essas diferenças para obter a distância relativa dos objetos. Segundo Mckenna (1992),

a estereoscopia é útil na noção da distância de objetos colocados até 10 metros do observador.

Em 1960, Morton Heilig patenteou a invenção *Stereoscopic TV Apparatus for Individual Use*, hoje mais conhecida como *Head-Mounted Display* ou *Virtual Reality Display*. Na época, sua invenção destinava-se apenas ao uso de imagens de televisão totalmente passivas, não-imersivas e sem possibilidade de interação. Porém, até hoje vemos sua invenção sendo utilizada em parques de diversão, como os da Disney, para um passeio passivo e simulado. [Bryson, 1993].

No momento, possuímos uma diversidade grande de dispositivos, todos voltados para nossa habilidade de perceber a profundidade com pares de imagens em estéreo. Porém ainda não está claro o quanto nossa percepção de profundidade depende puramente das disparidades geométricas ou quanto isso está relacionado à familiaridade de objetos já conhecidos por nós.

1.6. REPRESENTAÇÃO VETORIAL E MATRICIAL DE IMAGENS

Um vetor é basicamente um segmento de reta orientado. Podemos pensar em um vetor 2D, V , como uma seta que vai da origem do sistema de coordenadas, para o ponto (x,y) , tendo assim uma direção, um sentido e um comprimento especificado. Podemos então calcular o comprimento do vetor pela fórmula:

$$|V| = \sqrt{x^2 + y^2}$$

Se pensarmos em um vetor 3D, definido da origem ao ponto (x,y,z) , seu comprimento seria:

$$|V| = \sqrt{x^2 + y^2 + z^2}$$

Uma matriz é um arranjo (*array*) de elementos em duas direções. Quando trabalhamos, primeiro definimos quantos elementos existem em cada direção, uma matriz 4×4 , por exemplo, é do tipo:

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 |

A **representação vetorial** das imagens é principalmente empregada, em computação gráfica, para a definição e modelagem dos objetos sintéticos que serão representados pela imagem.

Na representação vetorial das imagens, são usados como elementos básicos os pontos, as linhas, as curvas, as superfícies tridimensionais ou mesmo os sólidos que descrevem os elementos, que formam as imagens sinteticamente no computador. Esses elementos são denominados **primitivas vetoriais** da imagem. As primitivas vetoriais são associadas a um conjunto de **atributos** que define sua aparência e a um conjunto de dados que define sua **geometria** (pontos de controle). Para esclarecer melhor, vamos considerar alguns exemplos. Dois elementos facilmente caracterizados como vetoriais, pela noção de vetores já discutida são os pontos e linhas retas. A cada elemento de um conjunto de pontos associa-se uma posição, que pode ser representada por suas coordenadas (geometria), e uma cor, que será como esses pontos aparecerão na tela (atributos). No caso de um conjunto de linhas retas, cada uma pode ser definida pelas coordenadas de seus pontos extremos (geometria) e sua cor, espessura, ou ainda se aparecerá pontilhada ou tracejada (atributos).

A descrição matricial é típica das imagens digitalizadas capturadas por scanners ou utilizadas nos vídeos. É a forma de descrição principal na análise e no processamento de imagens. Em computação gráfica sintética, surgem nos processos de finalização (*ray tracing*, *z-buffers*).

Na **representação matricial**, a imagem é descrita por um conjunto de células em um arranjo espacial bidimensional, uma matriz. Cada célula representa os **pixels** (ou pontos) da imagem matricial. Os objetos são formados usando adequadamente esses pixels. A Figura 1.2 explica melhor as formas de descrição de imagens matricial. Essa é a representação usualmente empregada para formar a imagem nas memórias e telas dos computadores e na maioria dos dispositivos de saída gráficos (impressoras e vídeos).

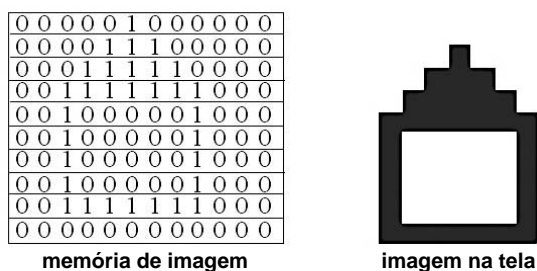


FIGURA 1.2. Descrição de imagens matriciais por conjunto de pixels.

1.7. ARQUITETURA DE SISTEMAS

Os sistemas para computação gráfica precisam de alguns dispositivos gráficos de entrada e saída (*In/Out* ou *I/O*) ligados a um computador. Assim, dispositivos gráficos são elementos críticos de um sistema de computação gráfica. Através dele interagimos com o sistema na busca de uma extensão dos limites de nosso corpo e uma melhor comunicação com a máquina. Ao contrário do software,

que encontra nas novas arquiteturas de CPUs e memórias, um ambiente adequado para sua evolução contínua, o hardware enfrenta diversos obstáculos para sua evolução. Um dos principais obstáculos está no preço ocasionado pelo elevado custo de desenvolvimento que, apesar de freqüentes quedas, ainda está muito aquém da realidade do mercado consumidor. Outro grande obstáculo está no peso e tamanho desses componentes que devem ser reduzidos para que possam ser facilmente utilizados.

1.7.1. Dispositivos Gráficos de Entrada

Os dispositivos de entrada são componentes eletrônicos que permitem a movimentação e interação com os sistemas. A cada dia surge um novo dispositivo com novas propostas ergonômicas, recursos adicionais que agilizam a tarefa de interação ou simplesmente reduzem a quantidade de fios em sua mesa. Dentre os dispositivos mais usados na computação gráfica podemos citar:

Teclado: Basicamente podemos definir um teclado como um conjunto de teclas associadas a um código que corresponde ao caractere ou função. Diversos dispositivos de teclado foram inventados ao longo de décadas, porém o mais usado é o teclado QWERTY. É irônico pensar que esse teclado foi inventado para a redução da velocidade do digitador e, como consequência, causar menores danos à sua saúde.

Mouse: Os mouses atuais utilizados por profissionais da computação gráfica são compostos por sensores óticos e processadores digitais para escanear a superfície sob o mouse sem a bola de rolagem. Enviam 1.500 sinais por segundo para rastrear com segurança o menor movimento possível.

Joysticks: São alavancas de comando que determinam a direção e velocidade do cursor na tela. São usados geralmente nos jogos de videogame, estações de realidade virtual e estações industriais de controle de robôs.

Tablet: Os *tablets* são extensões dos monitores sensíveis ao toque. Fruto de anos de pesquisas sobre como os profissionais realmente trabalham, os novos tablets são calibrados com perfeição para ler com absoluta precisão os movimentos da caneta, que opera com 1.024 níveis de sensibilidade à pressão. Estes são traduzidos em curvas suaves, transições graduais e controles precisos do traço. Um software incluso nos pacotes dos hardwares de *tablet* possibilita o reconhecimento da escrita.

Mesa Digitalizadora: Dispositivo vetorial que consiste de uma mesa e de um apontador. A cada vez que o usuário toca a mesa com o apontador é informado ao compu-

tador a coordenada deste ponto da mesa. Existem diversos trabalhos em andamento para a substituição deste periférico por sistemas mais baratos com câmeras digitais e softwares de reconhecimento de padrões.

Dispositivos de Entrada 3D

Com a popularização dos sistemas 3D (na maioria dos jogos), e o barateamento dos componentes eletrônicos, começaram a surgir uma diversidade de dispositivos de entrada 3D. Em qualquer casa de jogos eletrônicos, e não raramente em lojas de produtos de informática, podemos ver simuladores de esqui, skate entre outros. Basicamente, esses dispositivos permitem a movimentação e interação dentro de um espaço 3D qualquer.

Digitalizador Tridimensional: Trata-se de um dispositivo vetorial e consiste em uma espécie de braço mecânico com um sensor de toque na ponta. A cada vez que o sensor atinge um ponto na superfície de um objeto, a coordenada deste ponto em relação a um ponto referencial (origem) é transmitida ao computador.

Scanners Tridimensionais: Existem diversas tecnologias de scanners disponíveis no mercado. As mais baratas utilizam câmeras digitais acopladas a uma mesa especial que fornece as coordenadas para os sistemas. Esta tecnologia quase sempre requer a intervenção de modeladores para o acabamento das peças. A tecnologia de scanners a laser, de alto custo, é sem dúvida a tecnologia de dispositivos de entrada que vem atraindo mais atenção no mundo. Suas aplicações são grandes e muitas delas ainda estão por se descobrir.

Luvas: Luvas para interação 3D são dispositivos que, através de sensores, detectam e medem as flexões e pressões dos dedos. Os sensores podem ser mecânicos, ópticos ou híbridos.

Capacetes: Existem diversos tipos de capacetes para visualização de Realidade Virtual disponíveis no mercado. A principal característica desses equipamentos é que podem ser: *Estereoscópicos* ou *monoscópicos* (isto é, usados com uma ou duas cenas); *Binoculares* ou *monoculares* (um ou os dois olhos são estimulados); *Opacos* ou *translúcidos* (substituem ou complementam a realidade).

3D Controllers: São dispositivos para interatividade com ambientes 3D capazes de tornar o ambiente participativo, seguindo os movimentos executados pelo usuário. Trabalham tanto em cima de uma mesa como no ar, pois possuem um giroscópio que tem comunicação por rádio com o computador. São capazes de medir a velocidade e a força que estão sendo aplicadas pelo usuário.

Roupa de RV: A roupa para Realidade Virtual (ou data suits) é uma indumentária que permite a interação com o mundo virtual. A comunicação pode ser realizada de várias maneiras, sendo que o acompanhamento óptico de marcadores vem sendo o mais utilizado. Essas roupas são usadas para gerar informações do movimento humano, a partir daí surge uma infinidade de aplicações para animações, esporte, desenvolvimento de produtos, medicina etc.

1.7.2. Dispositivos Gráficos de Saída

É possível classificar os dispositivos de saída em duas principais categorias, segundo a forma pela qual as imagens são geradas (veja seção anterior de descrição vetorial e matricial de imagens): vetoriais e matriciais. Os dispositivos vetoriais conseguem traçar segmentos de reta perfeitos entre dois pontos. Os dispositivos matriciais apenas conseguem traçar pontos, ou seja, segmentos de reta são traçados como seqüências de pontos próximos, são entretanto, bastante adequados para desenhar áreas cheias e sombras, onde os vetoriais mostram deficiência.

Impressoras de Jato de Tinta: São equipamentos matriciais com cabeçote que ejetam tinta sobre o papel. Podem utilizar tintas de várias cores e chegar a níveis altos de realismo na imagem impressa.

Impressoras Laser: São as que têm melhor qualidade. Um feixe de raio laser varre uma chapa em processo ótico parecido com o do cabeçote de uma impressora, o bombardeio do feixe deixa a chapa carregada com uma carga eletrostática. Por efeito da atração elétrica, uma tintura (toner) adere à chapa e por pressão e aquecimento é fixada no papel formando a imagem.

Impressoras Térmicas: São equipamentos silenciosos, com boa resolução, podem trabalhar com ampla gama de cores. As impressoras térmicas precisam utilizar um papel termo-sensível especial.

Plotters: São dispositivos vetoriais e eletromecânicos que, de uma forma geral, produzem o desenho pelo movimento de uma caneta na superfície do papel. Existem dois tipos, em um, o papel permanece fixo e a caneta produz desenhos sobre o mesmo pela combinação de movimentos horizontais e verticais. No outro tipo, o desenho é produzido pela combinação dos movimentos do papel e da caneta.

Monitores: A maioria dos modelos atuais se baseia na tecnologia de tubos de raios catódicos (CRT – *Catode Ray Tube*), já madura e capaz de oferecer uma boa relação custo/benefício, para produzir imagens de qualidade em computadores pessoais. Mas dentro de poucos anos, encontrar um monitor CRT em uma loja poderá ser

quase impossível. Em várias partes do mundo, já é difícil encontrar um modelo CRT nas lojas. A Apple Computer, por exemplo, aboliu os monitores CRT de seus sistemas. Os fabricantes NEC Mitsubishi e Hitachi também já deixaram de produzir modelos CRT. A queda drástica nos preços dos monitores LCD (*Liquid Cristal Displays*), seu pouco peso e espessura são as principais causas dessa derrocada. Porém, os CRTs têm ainda uma vantagem substancial em relação aos LCDs no que diz respeito ao brilho e à versatilidade em resolução.

Monitores CRT: Até há pouco tempo era praticamente o único tipo de vídeo utilizado. A resolução máxima com a qual um monitor CRT pode trabalhar depende de sua habilidade física em focar o feixe de elétrons sobre os pontos de fósforo. Os monitores CRT são compostos por um canhão que gera um feixe de elétrons. Um aquecedor é utilizado para liberar elétrons de um catodo, razão pela qual os monitores demoram um pouco para apresentar a primeira imagem depois de ligado. Esses elétrons são atraídos por anodos (cargas positivas) próximos à parte da frente do monitor (Figura 1.3). O feixe de elétrons percorre um caminho da esquerda para a direita e de cima para baixo, orientado por diversos componentes chamados bobinas defletoras. Ao atingir a extremidade direita da tela, o feixe é desligado para retornar à extrema esquerda da linha inferior e, quando atinge a extremidade de baixo, também é desativado para retornar novamente à primeira linha. Esse processo é chamado de **varredura**. Aumentando ou diminuindo a intensidade do feixe, consegue-se controlar o brilho dos pontos de fósforo da tela para gerar a imagem. A velocidade com que o feixe percorre toda a tela é chamada de taxa de renovação (*refresh rate*) ou frequência de varredura vertical. O padrão antigo para monitores determinava que a **taxa de renovação** ideal era de 60 Hz, mas um novo modelo desenvolvido pela VESA (*Video Electronics Standards Association*) recomenda a frequência de 75 Hz para monitores trabalhando

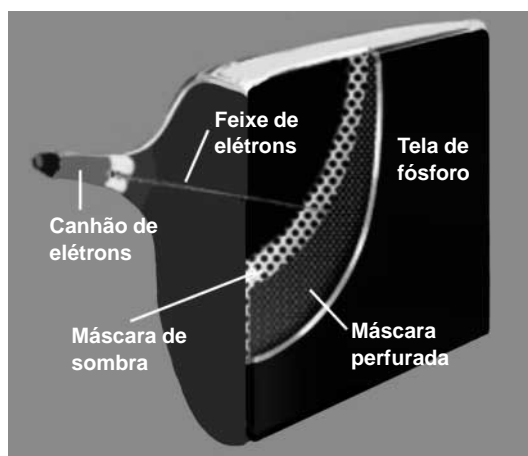


FIGURA 1.3. *Tubo de Raio Catódico.*

com resolução de 640 por 480 pixels ou maior. Quanto maior a taxa de renovação, menos sensível é o fenômeno de cintilação (*flicker*). Para oferecer maior resolução, sem que o custo do monitor se elevasse muito, foi criada a técnica de **entrelaçamento**. Nos monitores **entrelaçados**, o canhão de elétrons renova apenas metade das linhas em uma passada (por exemplo, apenas as linhas ímpares em um passo e, no seguinte, as linhas pares). Como apenas parte das linhas é refeita por vez, é possível apresentar o dobro de linhas por ciclo de renovação, aumentando conseqüentemente a resolução vertical oferecida pelo monitor. Em outras palavras, os modelos entrelaçados podem oferecer a mesma resolução que um não-entrelaçado, mas a um custo menor. A desvantagem dessa técnica fica por conta do tempo de resposta menor, o que pode ser crítico em aplicações de animação e vídeo, e do possível efeito de cintilação.

Monitores CRT Coloridos: A cor da luz emitida vai depender do fósforo usado. Os monitores monocromáticos, mais simples, produzem imagens na cor verde, branco ou âmbar e, durante muito tempo, foram os únicos a oferecer custo acessível para o usuário de computadores de mesa. No Brasil, os monitores coloridos tornaram-se populares no início da década de 1990. Esses modelos usam o padrão RGB (*Red*, *Green* e *Blue*), um sistema de representação de todas as cores com base no vermelho, verde e azul (Capítulo 5). Para gerar qualquer cor do espectro, os monitores coloridos precisam de três sinais separados, que vão sensibilizar, respectivamente, os pontos de fósforo das três cores primárias, suficientemente pequenos para parecer ao olho humano como um único ponto de luz.

Os monitores CRT coloridos empregam uma das três técnicas descritas a seguir para mesclar os trios de fósforo, cada uma com suas características específicas. São elas:

- **Shadow mask:** Insere uma fina folha de metal perfurada entre a tela e o canhão de elétrons; dessa forma, miram-se os respectivos feixes das três cores primárias em um mesmo orifício na placa, que direcionará a formação do ponto colorido na tela.
- **Aperture grille:** Insere uma grade de fios entre a tela e os canhões de elétrons para realizar a mesma tarefa da *shadow mask*.
- **Slot mask:** Usa também uma *shadow mask*, mas com orifícios mais compridos e finos. Pode ser considerada uma técnica intermediária entre as duas anteriores.

Além da resolução máxima, outro fator que influencia na qualidade final da imagem gerada pelo monitor é o chamado *dot pitch*, que especifica a distância entre dois pontos de fósforo da mesma cor em trios RGB adjacentes. Esse valor é uma medida que também deve ser levada em conta para determinar a qualidade de um monitor. De nada adianta uma tela grande, com alta resolução, ou seja, muitos pontos RGB, se esses pontos estiverem muito distantes entre si, gerando uma imagem reticulada. Assim, quanto menor o *dot pitch*, mais pontos por polegada terá o monitor e maior

sua capacidade de resolução máxima. O *dot pitch* é medido em milímetros, e os valores encontrados nos modelos de monitores mais comuns são de 0,28 mm ou menores. Existe uma diferença substancial no modo como o *dot pitch* é medido entre os diversos fabricantes de monitores. Por essa razão, tais valores não devem ser comparados diretamente. Algumas vezes, o *dot pitch* é medido como a distância entre dois pontos de fósforo da mesma cor em tríades adjacentes, na diagonal. Diversos fabricantes fazem a medida na horizontal, para fornecer um valor mais palpável ao consumidor. Há ainda quem meça a distância entre os orifícios na máscara e não entre os pontos de fósforo. Em monitores que se baseiam no *aperture grill*, a medição é feita entre duas tiras de fósforo da mesma cor.

Resolução: Um conceito estreitamente ligado ao tamanho da tela dos monitores é a resolução, que descreve a quantidade de informação que o equipamento pode apresentar em um determinado instante. A resolução é medida em **pixels** (palavra criada a partir da expressão em inglês *picture element*), uma unidade básica da imagem que pode ser controlada individualmente e que contém informações sobre cores e intensidade (brilho). O pixel deve ser encarado como uma unidade lógica e não física. O tamanho físico de um pixel vai depender de como a resolução da tela foi configurada. Assim, se estiver sendo usada a resolução máxima, um pixel será equivalente a uma unidade física do monitor. Porém, para resoluções menores do que a máxima, um pixel será composto por mais de um ponto físico da tela.

A resolução pode ser descrita pelo número de pixels (pontos individuais de uma imagem) apresentados na tela de um monitor, expressos nos eixos horizontal e vertical. A forma de uma imagem na tela depende da resolução e do tamanho do monitor. A mesma resolução produzirá uma imagem de melhor qualidade, menos reticulada, em um monitor menor, e perderá gradualmente a forma, à medida que forem usados modelos maiores. Isso acontece porque o mesmo número de pixels terá de ser espalhado por uma área maior da tela. O padrão atual de resolução adotado por grande parte das páginas na Web é de 800 por 600 pontos, para citar um exemplo conhecido pela maioria dos usuários de computador. Já programas baseados em janelas (windows) costumam adotar resoluções de 1.024 por 768 pixels. A principal vantagem de usar resoluções maiores é a redução da necessidade de ampliar a imagem (usando recursos de zoom), pois mais informação da imagem é apresentada de uma só vez.

Existe uma relação estreita entre a resolução usada e o tamanho do monitor. Resoluções muito altas em uma tela pequena podem resultar em problemas, pois alguns pacotes de software usam texto limitado a um número fixo de pontos. Desse modo, quando apresentado em telas pequenas configuradas com alta resolução, o texto aparecerá muito reduzido para propiciar uma leitura confortável. Por outro lado, aumentar simplesmente o tamanho do monitor nem sempre é a solução indicada. Se a resolução não puder acompanhar o aumento da tela, o resultado será caracteres e imagens mais reticulados.

A relação entre o tamanho da tela e a resolução padrão (default) é mostrada nesta tabela:

| Medida nominal | Resolução recomendada |
|----------------|-----------------------|
| 14" | 800 × 600 |
| 15" | 800 × 600 |
| 17" | 1.024 × 768 |
| 19" | 1.280 × 1.024 |
| 21" | 1.600 × 1.200 |

Monitores LCD: Os monitores de cristal líquido (*LCD – Liquid Crystal Display*) não possuem um canhão de elétrons, o que lhes confere uma série de vantagens e algumas desvantagens. Em substituição ao tubo de raios catódicos, existe um sistema de células contendo cristal líquido e filtros coloridos polarizados. As moléculas do cristal líquido (substância descoberta em 1888) expostas a campos elétricos são alinhadas (fenômeno da polarização). Estas moléculas de cristal líquido ficam entre duas camadas de filtros formados por um conjunto de finíssimos fios dispostos em paralelo. Cada filtro tem os fios em posição perpendicular ao outro, levando as moléculas a formarem uma coluna “torcida” para passagem dos raios luminosos. A fonte de luz fluorescente que deverá passar pelo cristal líquido é denominada *backlight*. O objetivo é fazer com que esses raios luminosos passem pelas células contendo cristal líquido. Quando é aplicado um campo elétrico aos cristais, vemos as moléculas arranjadas no sentido vertical, permitindo que os raios de luz passem por elas até encontrarem o segundo filtro. Como estão em posições defasadas um ao outro, os raios luminosos não passarão e não há geração de imagem. Ao contrário, sem tensão aplicada sobre o cristal líquido, os raios luminosos passarão pelo primeiro filtro, sofrendo realinhamento na coluna torcida de cristal líquido contida nas pequenas células, alinhadas na tela, até atingir o segundo filtro, gerando a imagem.

Em monitores LCD monocromáticos o pixel é formado por uma célula, enquanto nos monitores LCD policromáticos (coloridos), cada pixel é formado por três células de cristal. Cada uma dessas células tem um filtro vermelho, verde ou azul, barrando a entrada da luz. Passando pelas células com as cores filtradas, a luz gerada pelo *backlight* resultará na imagem com as cores vistas na tela de um monitor LCD.

As células fazem o papel dos pixels nos monitores do tipo CRT, e um pixel pode ser formado por uma ou mais células. No entanto, essas células não podem variar em suas dimensões. Nos monitores CRT, podemos aumentar ou diminuir, sempre proporcionalmente, o número de linhas e colunas, aumentando ou diminuindo o

número de pixels. Nos monitores tipo LCD, isso não é tão simples, pois cada célula vem com dimensões predefinidas. Ao alterarmos a resolução da imagem, provocamos a distorção da mesma. Isso ocorre porque não é possível fazer um aumento proporcional (altura e largura) da tela.

Monitores LCD de Matriz Passiva – Bastante utilizados inicialmente em notebooks monocromáticos ou não, hoje esta tecnologia é empregada na fabricação de telas para celulares, handhelds, notepads etc. Nessa tecnologia, a tela é formada por uma grade de fios condutores. Cada interseção desses fios forma um pixel, e a corrente necessária para a polarização será transmitida por esses fios. O controle do brilho dos pixels é obtido aplicando-se uma corrente elétrica forçando os cristais a se realinharem, ou seja, haverá alteração na direção dos raios luminosos. Esse processo é repetido linha a linha de pixels, da parte superior à inferior da tela do monitor LCD até a corrente específica chegar à célula específica. Essa técnica tem o inconveniente de produzir cintilação, o que levou à utilização de um tipo de cristal com baixo tempo de resposta às diferentes correntes aplicadas sobre os cristais. Quer dizer que há uma demora maior que a desejada para o realinhamento dos cristais. Essa demora causa menor nitidez na geração de imagens, e se percebe mais na execução de arquivos de vídeo ou games. Outra desvantagem é o efeito *crosstalk*, causado pela interferência do campo magnético de uma célula sobre células de cristal líquido vizinhas, alterando a imagem que deveria ser gerada. Daí surgiu a tecnologia *Dual Scan Twisted Nematic* – DSTN, para amenizar esse efeito negativo dos monitores de matriz passiva. Nela a tela é dividida em uma parte superior e outra inferior, com varreduras independentes. Para compensar essas dificuldades, alguns monitores de matriz passiva dispõem do recurso de endereçamento simultâneo para duas linhas diferentes.

Monitores LCD de Matriz Ativa: Atualmente os novos monitores LCD utilizam a tecnologia de matriz ativa, na qual um transistor especial – TFT *Thin-Film-Transistor* – alimenta cada célula individualmente. Neste caso, cada célula recebe uma corrente elétrica, inferior à utilizada nos monitores de matriz passiva, independente das outras. Essa é a solução completa para os problemas existentes ao gerarmos imagens em sequência (filmes, games etc.) além do *crosstalking*. Como resultado, têm-se imagens mais nítidas com cores mais intensas e ausência da cintilação. Outra vantagem é a possibilidade de fabricação de telas com mais de 20". Em compensação, a fabricação desses monitores implica em algumas dificuldades. As telas de LCD são montadas sobre um substrato de vidro com um único chip (conjunto de transistores). Levando em conta que uma tela preparada para atingir resolução de 800 x 600 pixels pode conter mais de 6 milhões de transistores, as chances de haver transistores defeituosos não é pequena. A verdade é que toda tela LCD de matriz ativa tem defeitos, e para descobri-los basta preenchê-la com fundo branco ou preto. Os pontos que mais se destacarem são os defeituosos.

Monitores *See-through*: Os monitores ditos *see-through* são aqueles em que é possível enxergar através do monitor. Esse tipo de aparato é muito usado em situações onde se precisa sobrepor uma imagem real a uma imagem gerada por computador ou vice-versa. É constituído de monitores do tipo LCD pois essa tecnologia permite que se veja através do monitor. O maior problema desse tipo de monitor é a pouca intensidade da imagem exibida, o que muitas vezes prejudica a aplicação. Recentemente, esses monitores vêm sendo usados para visão noturna e aproveitamento de espaço em automóveis e veículos de combate. Uma forma alternativa à implementação desse monitor é utilizar um monitor convencional do tipo CRT, filmar a imagem do ambiente real (com uma pequena câmera presa à cabeça do usuário) e sobrepor esta imagem à imagem gerada pelo computador. Esse mecanismo permite uma melhoria significativa na qualidade da imagem mostrada, mas nem sempre será a solução dependendo do tipo de aplicação.

Displays de Retina: O *Human Interface Technology Lab*, da Universidade de Washington, desenvolveu um tipo de monitor onde o equipamento é um laser que exibe as imagens diretamente na retina do usuário. Por enquanto, o equipamento ainda é muito grande e com uma resolução máxima de 1000x1000 pontos monocromáticos. Esse equipamento tem despertado o interesse da comunidade científica por diferentes propósitos. Primeiro, porque ele permite que portadores de alguns tipos de doença da retina possam enxergar, já que o equipamento força a entrada da luz para dentro do olho. Segundo, porque a redução do seu custo poderia determinar o fim dos monitores.

Head Mouted Displays: Também conhecidos como “óculos de realidade virtual” ou “capacetes de realidade virtual”, os HMDs (*Head Mouted Displays*) operam exibindo em duas pequenas telas (uma para cada olho) imagens de uma cena virtual. Os HMDs são construídos, normalmente, usando dois tipos de monitores: CRT ou LCD. Os monitores de CRT, em função da avançada tecnologia disponível nesta área, podem exibir imagens de alta resolução com uma qualidade de cor excelente, mesmo em pequenas dimensões. Entretanto, são relativamente pesados, volumosos e colocam altas voltagens muito próximas à cabeça do usuário. Os monitores LCD, por sua vez, são leves e podem ser usados com pequenas voltagens. Entretanto, devido às limitações tecnológicas, a resolução disponível em monitores pequenos ainda é baixa. Acoplados aos HMDs, em geral existem sistemas de rastreamento da posição da cabeça, a fim de permitir que se atualize as imagens do mundo virtual de acordo com a direção para onde o usuário está olhando.

Stereo Glasses ou Shutter Glasses: Os *Stereo Glasses* são uma extensão dos monitores “*see-through*”. Útil em aplicações onde várias pessoas precisam observar a mesma imagem estéreo, como visualização científica, cirurgias e passeios virtuais em par-

ques de diversão, estes dispositivos, bem mais baratos do que os HMDs, buscam gerar imagens a partir de uma tela convencional de computador. A idéia é colocar nos usuários pares de óculos com lentes de cristal líquido capazes de bloquear a visão de um dos olhos quando necessário. A idéia básica desses dispositivos consiste em exibir na tela a imagem correspondente à do olho esquerdo e bloquear a visão do olho direito escurecendo a tela de cristal líquido, a seguir faz-se o contrário, ou seja, exibe-se a imagem do olho direito e bloqueia-se a visão do esquerdo.

Cave: Os primeiros registros de simulações realizados pelo homem são da época das cavernas. Na pré-história, os guerreiros desenhavam nas cavernas cenas de batalhas e caça de animais. Usando uma tocha, iluminavam a seqüência e passos para demonstrar às crianças e aos jovens como seria na prática. Daí vem o nome Cave (caverna) ou, mais recentemente, *Surround-Screen Project-Based*. Esses dispositivos usam a idéia de colocar o usuário em uma sala com paredes que são na verdade telas para projeções de imagens.

Placas Aceleradoras de Vídeo: Os monitores interpretam sinais analógicos para apresentar imagens na tela. Para isso, o processador existente na placa de vídeo precisa transformar os sinais digitais em analógicos antes de enviá-los ao monitor. Nem todo o processamento de imagens é realizado pelo processador de vídeo. Parte desse trabalho é realizada pelo processador principal, mas quanto mais poderoso o processador de vídeo, menos sobrecarregado fica o processador principal, ficando disponível para efetuar outras tarefas. Processar imagens é, basicamente, fazer cálculos. Quanto mais complexa uma imagem, maior o número de pontos que devem ser criados, ocorrendo o mesmo se desejarmos melhores resoluções de imagem. As placas aceleradoras 3D de uso profissional são normalmente otimizadas para trabalhar com OpenGL, DirectX e alguns softwares de modelagem. A indicação para uso profissional desse tipo de placa também foi reforçada pela Microsoft que implementou a compatibilidade nativa para OpenGL desde o Windows NT e 2000.

AGP: A porta AGP (*Accelerated Graphics Port*) segue como padrão mínimo para boas placas gráficas. É usada para conectar placas gráficas diretamente para a CPU e para a memória principal. O modo AGP permite que as placas gráficas se comuniquem com a CPU e com a memória principal a taxas de dados de cinco a oito vezes mais rápido do que o barramento PCI.

1.8. OPENGL

OpenGL (*Open Graphical Library*) pode ser definida como uma interface de software (API – *Aplication Program Interface*) para aceleração da programação de dispositi-

vos gráficos, com aproximadamente 120 comandos para especificação de objetos e operações necessárias para a produção de aplicações gráficas interativas 3D. Na verdade, podemos classificá-la como uma biblioteca de rotinas gráficas para modelagem 2D ou 3D, extremamente portátil e rápida, possibilitando a criação de gráficos 3D com excelente qualidade visual e rapidez, uma vez que usa algoritmos bem desenvolvidos e otimizados pela Silicon Graphics. Justamente pela sua portabilidade, não possui funções para gerenciamento de janelas, interação com o usuário ou arquivos de entrada/saída. Cada ambiente, como, por exemplo, o Microsoft Windows, possui suas próprias funções para esses propósitos.

OpenGL não é uma linguagem de programação como C, C++ ou Java, é uma poderosa e sofisticada API (Application Programming Interface) ou biblioteca de códigos, para desenvolvimento de aplicações gráficas 3D em tempo real, seguindo a convenção de chamada de bibliotecas da linguagem de programação C. Isso significa que programas escritos nessa linguagem podem facilmente chamar as funções, tanto porque estas foram escritas em C, como por ser fornecido um conjunto de funções C intermediárias que chamam funções escritas em *Assembler* ou outra linguagem de programação. Porém, podemos utilizar também as linguagens Ada, C++, Fortran, Python, Perl, Java, Delphi (visite o site <http://www.glsce-ne.org/>), Visual Basic (visite o site <http://is6.pacific.net.hk/~edx/>) e outras. No caso de Java, existem várias bibliotecas de classes e plugins para browsers que ligam as chamadas dos *Applets* com .dlls do OpenGL. Na verdade, esses plugins não executarão OpenGL no nível da máquina de Java e sim no nível de .dlls de OpenGL. Normalmente se diz que um programa é baseado em OpenGL ou é uma aplicação OpenGL, o que significa ser escrito em alguma linguagem de programação que faz chamadas a uma ou mais de suas bibliotecas.

Os principais fabricantes de PC e supercomputadores como SGI, Cray Research, Compaq, Fujitsu, Hewlett-Packard, Hitachi, IBM, Intel, Microsoft, Mitsubishi, NEC, Samsung, Siemens-Nixdorf, Sony e Sun Microsystems adotaram OpenGL como uma estratégia de padrão aberto para hardware gráfico. Suas aplicações variam de ferramentas CAD a jogos e imagens médicas ou programas de modelagem usados para criar efeitos especiais para televisão e cinema (como em *Jurassic Park* e *Star Wars*). Dentre os programas de modelagem, podemos citar 3D MAX, Bryce, Character Studio, Lightware, Lightscape, Maya, Rhino 3D, TruSpace e muitos outros. Dentre os jogos, temos Quake, Half-Life, MDK2, Baldurs Gate, Decent, Madden, NFL 2001 etc.

A biblioteca OpenGL vai além do desenho de primitivas gráficas, tais como linhas e polígonos, dando suporte a iluminação, sombreamento, mapeamento de textura, transparência, animação, gerenciamento de eventos de entrada por teclado e mouse. A biblioteca GLU (que faz parte da implementação OpenGL) possui várias funções para modelagem, tais como superfícies quadráticas, e curvas e superfícies NURBS (*Non Uniform Rational B-Splines*). Além disso, a biblio-

teca OpenGL executa transformações de translação, escala e rotação, através da multiplicação de matrizes com transformações cumulativas, ou seja, umas sobre as outras.

Apesar de OpenGL ser uma biblioteca de programação “padrão”, existem muitas implementações dessa biblioteca, por exemplo, para Windows e Linux. Das implementações para PC, as mais conhecidas são as da Silicon Graphics e da Microsoft. No caso da implementação da Microsoft, são fornecidas com suas ferramentas de programação as bibliotecas `opengl32.lib` (OpenGL) e `glu32.lib` (utilitários OpenGL). Estas, se não constarem em seu computador, podem ser baixadas gratuitamente junto com sua excelente documentação no site oficial em <http://www.opengl.org>. A biblioteca `glut.lib` e sua respectiva documentação, que será usada neste livro, pode ser baixada direta e gratuitamente do site do autor em <http://reality.sgi.com/opengl/glut3/glut3.html>.

Entre suas características estão:

- **Aceleração de Hardware:** Aceleração do processamento geométrico, luzes, *real-time*, transformações e *render*.
- **Efeitos 3D em Real-Time:** Real-time fog, anti-aliasing, volume shadows, bump mapping, motion blur, transparência, reflexões, texturas 3D, volume rendering e outras.
- **Suporte a inovações futuras de software e hardware:** Um mecanismo de extensão permite configurações para novos hardwares e softwares.
- **Estabilidade:** Estações avançadas e supercomputadores vêm utilizando suas bibliotecas desde 1992.
- **Escalável:** Suas aplicações API podem ser executadas de pequenos aparelhos eletrônicos até supercomputadores com utilização máxima dos recursos disponíveis.

Integração do OpenGL com Windows95/98/2000/XP

De forma semelhante a outras APIs do Windows, os comandos do OpenGL são disponibilizados através de bibliotecas dinâmicas, conhecidas como DLLs (*Dynamic Link Library*) e seus respectivos arquivos header e library. Os arquivos header são incluídos no código-fonte enquanto os arquivos library devem ser incluídos no projeto. A tabela a seguir caracteriza esses arquivos.

| | | | |
|--------------------|---------|--------------|---|
| OPENG32.DLL | gl.h | opengl32.lib | Contém as funções padrão do OpenGL, definidas pelo OpenGL Architecture Review Board e são caracterizadas pelo prefixo gl . |
| GLU32.DLL | glu.h | glu32.lib | A biblioteca de utilitários contém funções com o prefixo glu para desenho de esferas, cubos, discos, cilindros etc. |
| GLAUX.DLL | glaux.h | glaux.lib | Os comandos da chamada biblioteca auxiliar são caracterizados pelo prefixo aux e, embora não sejam realmente parte da especificação OpenGL, permitem desenvolver aplicações simples, independente de plataforma e sistema operacional. |
| GLUT.DLL | glut.h | glut.lib | OpenGL Utility Toolkit (Conjunto de Ferramentas Utilitárias do OpenGL): é um sistema de gerenciamento de janelas independente do sistema operacional escrito por Mark Kilgard. Todas as suas rotinas começam com o prefixo glut . |

0 Primeiro Programa

Para entender o funcionamento das bibliotecas vejamos um programa simples em C. No CD que acompanha o livro você encontrará exemplos de programas escritos em VB (Basic) e Delphi (Pascal).

```
#include <GL/glut.h> // inclusão da biblioteca
#include <stdlib.h> // inclusão da biblioteca

// Declarações
void init(void);
void display(void);
void keyboard(unsigned char key, int x, int y);

int main(int argc, char** argv){
    glutInit(&argc, argv); // Inicializa o GLUT e processa qualquer parâmetro passado
                           // pela linha de comandos. Deve ser chamada antes de qualquer
                           // outra rotina GLUT.

    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); // Indica se vamos usar cores no
                                                    // modo RGBA ou Indexado. Também especificamos se usaremos um ou
                                                    // dois buffers para a geração das imagens e se vamos associar buffers
                                                    // de profundidade, estêncil e/ou acumulador à janela que estamos usando.
```

```

glutInitWindowSize (150, 150); // Especifica as dimensões da janela em pixels.
glutInitWindowPosition (100, 100); // Especifica a coordenada superior esquerda da
                                     janela
glutCreateWindow ("Desenha uma linha"); // Cria a janela e devolve um identificador
                                     único para a janela. Até que o comando glutMainLoop seja
                                     chamado, a janela não será mostrada.

init( );
glutDisplayFunc(display); // Toda vez que o GLUT determinar que a janela tem de
                             ser desenhada, ele chamará a função aqui determinada.

glutKeyboardFunc(keyboard); // Determinam as funções que usaremos para ler o
                             teclado e o mouse respectivamente.
glutMainLoop( ); // É o último comando que chamamos. Ele faz com que todas as
                             janelas criadas sejam mostradas. Uma vez que entramos neste loop,
                             só saímos quando o programa se encerra.

return 0;
}

void init(void){
glClearColor(1.0, 1.0, 1.0, 1.0); // Fornece os valores para limpeza do buffer de cor no
                                     modo RGBA
glOrtho (0, 256, 0, 256, - 1 ,1); // Seleciona o modo de projeção Ortogonal.
}

void display(void){
    int i;
    glClear(GL_COLOR_BUFFER_BIT); \\ Limpa toda a Janela para a cor do comando
                                     glClearColor
    glColor3f (0.0, 0.0, 0.0); \\ Seleciona a cor preta para a linha
    glBegin(GL_LINES);
    glVertex2i(20,100); glVertex2i(100,20); \\ Fornece as coordenadas dos pontos
                                     iniciais e finais

    glEnd( );
    glFlush( ); \\ Este comando permite a execução em ambiente de rede
}

\\ A rotina a seguir termina o programa com a tecla Esc
void keyboard(unsigned char key, int x, int y){
    switch (key) {
        case 27:
            exit(0);
            break;
    }
}

```

Sintaxe de Comando

Tomando como exemplo o comando `glColor3f`, podemos observar que o prefixo `gl` representa a biblioteca `gl`, e o sufixo `3f` significa que a função possui três valores de

ponto flutuante como parâmetro. Resumindo, todas as funções OpenGL possuem o seguinte formato:

| Sufixo | Tipo de Dados | Correspondente em C | Tipo de dados OpenGL |
|--------|-------------------------|---------------------|----------------------------|
| b | 8-bit integer | signed char | Glbyte |
| s | 16-bit integer | short | Glshort |
| i | 32-bit integer | long | GLint, GLsizei |
| f | 32-bit floating-point | float | GLfloat, GLclampf |
| d | 64-bit floating-point | double | GLdouble, GLclampd |
| ub | 8-bit unsigned integer | unsigned char | GLubyte, GLboolean |
| us | 16-bit unsigned integer | unsigned short | Glushort |
| ui | 32-bit unsigned integer | unsigned long | GLuint, GLenum, GLbitfield |

RESUMO

Neste capítulo, apresentou-se uma visão geral da computação gráfica, sua história e aplicações. Detalhes de como percebemos e podemos apresentar imagens foram discutidos. Também foram vistos detalhes ligados aos equipamentos e a biblioteca OpenGL.

CAPÍTULO 2

Transformações Geométricas no Plano e no Espaço

- 2.1.** Matrizes em Computação Gráfica
- 2.2.** Pontos, Vetores e Matrizes
- 2.3.** Aritmética de Vetores e Matrizes
- 2.4.** Sistemas de Coordenadas
 - 2.4.1.** Sistema de Referência do Universo
 - 2.4.2.** Sistema de Referência do Objeto
 - 2.4.3.** Sistema de Referência Normalizado
 - 2.4.4.** Sistema de Referência do Dispositivo
 - 2.4.5.** Transformações entre Sistemas de Coordenadas
- 2.5.** Transformações em Pontos e Objetos
 - 2.5.1.** Transformação de Translação
 - 2.5.2.** Transformação de Escala
 - 2.5.3.** Transformação de Rotação
 - 2.5.4.** Transformação de Reflexão, Espelhamento ou Flip
 - 2.5.5.** Transformação de Cisalhamento
- 2.6.** Coordenadas Homogêneas
- 2.7.** Projeções Geométricas
 - 2.7.1.** Classificação das Projeções Geométricas
 - 2.7.2.** Projeções Paralelas Ortográficas
 - 2.7.3.** Projeções Paralelas Axométricas
 - 2.7.4.** Projeção Perspectiva ou Cônica

2.8. Especificação dos pontos de fuga

2.9. Câmera Virtual

2.10. Transformações Geométricas com OpenGL

2.10.1. Transformação de Translação

2.10.2. Transformação de Escala

2.10.3. Transformação de Rotação

2.10.4. Matrizes de transformação

2.10.5. Armazenando as transformações na matriz

2.10.6. Alterando a matriz de transformação

2.10.7. Montando transformações hierárquicas

2.10.8. Desfazendo o vínculo de hierarquia

2.10.9. Matriz genérica de projeção

2.10.10. Projeção paralela ortogonal

2.10.11. Projeção em perspectiva

2.10.12. Posição da câmera

Transformações geométricas são operações que podem ser utilizadas visando a alteração de algumas características como posição, orientação, forma ou tamanho do objeto a ser desenhado.

2.1. MATRIZES EM COMPUTAÇÃO GRÁFICA

Todas as transformações geométricas podem ser representadas na forma de equações. O problema é que manipulações de objetos gráficos normalmente envolvem muitas operações de aritmética simples. As matrizes são muito usadas nessas manipulações porque são mais fáceis de usar e entender do que as equações algébricas, o que explica por que programadores e engenheiros as usam extensivamente.

As matrizes são parecidas com o modelo organizacional da memória dos computadores. Suas representações se relacionam diretamente com estas estruturas de armazenamento, (Seção 1.6), facilitando o trabalho dos programadores e permitindo maior velocidade para aplicações críticas como jogos e aplicações em realidade virtual. É devido a esse fato que os computadores com “facilidades vetoriais” têm sido muito usados junto a aplicações de computação gráfica.

Devido ao padrão de coordenadas usualmente adotado para representação de pontos no plano (x,y) e no espaço tridimensional (x,y,z), pode ser conveniente manipular esses pontos em matrizes quadradas de 2x2 ou 3x3 elementos. Através de matrizes e de sua multiplicação, podemos representar todas as transformações lineares 2D e 3D. Várias transformações podem ser combinadas resultando em uma única matriz denominada matriz de transformação.

2.2. PONTOS, VETORES E MATRIZES

Dado um sistema de coordenadas, é possível definir pontos e objetos neste sistema pelas suas coordenadas. Nos espaços bidimensionais ou nos objetos planos, duas coordenadas caracterizam um ponto. Para objetos tridimensionais ou pontos no espaço, três coordenadas são necessárias para definir seu posicionamento. Assim, dado um sistema de coordenadas, cada ponto pode ser associado às suas coordenadas no sistema. Por exemplo:

$$A = [2,3] = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \text{ e } B = [1,1] = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

A convenção usada é que ao definir um ponto, usa-se a sua distância em relação a cada um dos eixos do sistema de coordenadas. Essas representações também podem ser chamadas de vetores linhas, ou vetores colunas, respectivamente. São ainda chamadas de arranjos (arrays) ou matrizes. Assim, a forma mais simples de matriz é o vetor linha com dois ou mais elementos colocados lado a lado e envolvidos por colchetes, por exemplo, o vetor bidimensional [x y] ou o vetor tridimensional

semelhantes podem ser definidas com matrizes, mas só fará sentido falar em soma ou subtração de vetores ou matrizes se os dois operandos tiverem a mesma dimensão. Outra operação muito comum é a multiplicação dos elementos dos vetores e matrizes por um valor constante. Essa operação é chamada multiplicação por um escalar. Por exemplo:

$$\frac{1}{2} \times \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A transposta de um vetor ou matriz é o vetor ou matriz resultante da troca dos valores de suas linhas e colunas. Assim $[2 \ 3]$ e $\begin{bmatrix} 3 \\ 2 \end{bmatrix}$ são vetores transpostos um do outro. Para simbolizar o transposto de um vetor ou a transposta de uma matriz, usa-se um T superescrito. Assim:

$$[2 \ 3]^T = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

Matrizes também podem ser multiplicadas entre si, desde que o número de colunas da primeira seja igual ao número de linhas da segunda, na operação. O resultado será uma matriz com o número de linhas da primeira e o número de colunas da segunda. O valor de cada elemento da matriz resultante é dado pela soma dos produtos dos elementos das linhas da primeira pelos elementos da coluna da segunda, de mesma ordenação, exemplificando:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 7 & 6 \\ 5 & 0 \end{bmatrix} = \begin{bmatrix} 1 \times 7 + 2 \times 5 & 1 \times 6 + 2 \times 0 \\ 3 \times 7 + 4 \times 5 & 3 \times 6 + 4 \times 0 \end{bmatrix} = \begin{bmatrix} 17 & 6 \\ 41 & 18 \end{bmatrix}$$

A regra de multiplicação torna algumas operações impossíveis de serem realizadas. Por exemplo, se desejarmos multiplicar os seguintes vetores:

$$V_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \text{ e } V_2 = [3 \ 4]$$

verificaremos que só podem ser multiplicados na ordem $V_2 \times V_1$ e não como $V_1 \times V_2$. Repare que essa multiplicação resultará em uma matriz 1×1 , ou seja, em um número. Outras vezes, as transportas podem auxiliar a multiplicação de um elemento por outro, por exemplo: se $M_1 = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix}$, então será impossível realizar a multiplicação $V_1 \times M_1$ a menos que se use o transporte de V_1 na operação.

Assim $V_1^T M_1$ resulta $[1 \ 2] \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix} = [-1 \ -4]$.

Uma propriedade interessante é que $(A B)^T = B^T A^T$. Lembre-se de usá-la sempre que comparar resultados de diferentes fontes da literatura, principalmente vetores linhas ou vetores colunas nas multiplicações. O uso da transporta torna sempre possível a multiplicação de dois vetores. Se for feita a multiplicação de forma que o resultado seja um número, ela é denominada de produto interno ou produto escalar de dois vetores, tem muitas aplicações importantes e é simbolizada por um “•”. Assim podemos dizer que o produto interno dos vetores V_1 e V_2 é o número (ou escalar) 11, ou seja:

$$V_1 \bullet V_2 = 1 * 3 + 2 * 4$$

2.4. SISTEMAS DE COORDENADAS

Podemos utilizar diferentes sistemas de coordenadas para descrever os objetos modelados em um sistema 2D. O sistema de coordenadas serve para nos dar uma referência em termos de medidas do tamanho e posição dos objetos dentro de nossa área de trabalho. A Figura 2.1 mostra três sistemas de coordenadas. No sistema de coordenadas polares (ao centro da figura), as coordenadas são descritas por raio e ângulo: (r, ϕ) . No sistema de coordenadas esférico (à esquerda), as coordenadas são descritas por raio e dois ângulos. Nos sistemas de coordenadas cilíndricas (à direita), as coordenadas são descritas por raio, ângulo e um comprimento. Os dois sistemas das extremidades são 3D.

Um determinado sistema de coordenadas é denominado de Sistema de Referência se for um sistema de coordenadas cartesianas para alguma finalidade específica. Ao definirmos um sistema de coordenadas de referência, devemos especificar dois aspectos principais: a unidade de referência básica e os limites extremos dos valores aceitos para descrever os objetos. Alguns sistemas, como os mostrados no esquema a seguir, recebem uma denominação especial, são os Sistema de Referência do Uni-

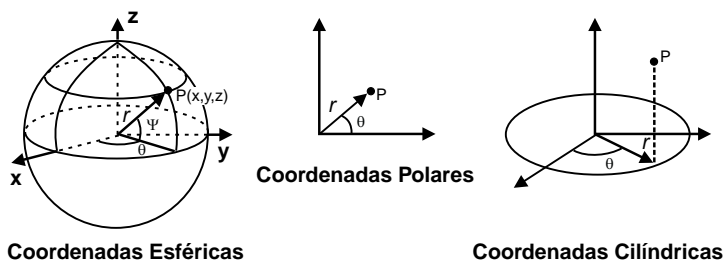
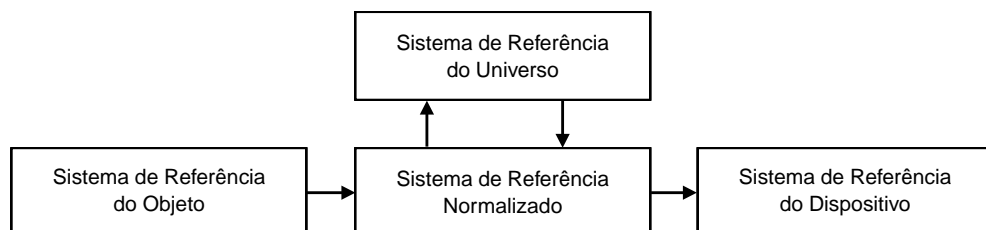


FIGURA 2.1. Diferentes sistemas de coordenadas.

verso (SRU); Sistema de Referência do Objeto (SRO); Sistema de Referência Normalizado (SRN); e Sistema de Referência do Dispositivo (SRD), vejamos o que eles significam.



2.4.1. Sistema de Referência do Universo (SRU)

É chamado de coordenadas do universo, ou do mundo, o sistema de referência utilizado para descrever os objetos em termos das coordenadas utilizadas pelo usuário em determinada aplicação. Sendo assim, cada tipo de aplicação especifica o seu universo de trabalho próprio, por exemplo, para sistemas de CAD de arquitetura, o universo poderá ser em metros ou centímetros; e para um CAD de mecânica de precisão, o universo provavelmente estará em milímetros ou nanômetros. Em outros casos, o melhor sistema nem mesmo é cartesiano, para localizações de aviação (por exemplo nos sistemas de radar) coordenadas polares (Figura 2.1) são mais indicadas. Cada um destes sistemas tem seus limites extremos (coordenadas mínimas e máximas do universo).

2.4.2. Sistema de Referência do Objeto (SRO)

Neste sistema de referência fazemos com que cada objeto seja um miniuniverso individual, ou seja, cada objeto tem suas particularidades descritas em função de seu sistema, muitas vezes coincidindo o centro do sistema de coordenadas com o seu centro de gravidade. Na modelagem de sólidos, este centro é conhecido como pivô (Seção 4.1).

2.4.3. Sistema de Referência Normalizado (SRN)

Esse sistema trabalha com as coordenadas normalizadas, isso é com valores entre 0 e 1 onde $0 \leq x \leq 1$ e $0 \leq y \leq 1$, sendo x e y as coordenadas horizontais e verticais possíveis. O Sistema de Referência Normalizado serve como um sistema de referência intermediário entre o SRU e o SRD. Sua principal aplicação é tornar a geração das imagens independente do dispositivo, pois as coordenadas do universo são convertidas para um sistema de coordenadas padrão normalizado.

2.4.4. Sistema de Referência do Dispositivo (SRD)

Utiliza coordenadas que podem ser fornecidas diretamente para um dado dispositivo de saída específico. Por exemplo, em um vídeo esses valores podem ser o número máximo de pixels que podem ser acesos (640×480, 800×600 etc.) ou podem indicar a resolução especificada em determinada configuração do sistema operacional, por exemplo 800×600×TrueColor(32bits) para vídeos ou, no caso de um scanner, a resolução máxima estabelecida ou de captura. Assim, nos hardwares, o sistema de coordenadas depende geralmente da resolução possível e da configuração definida pelo usuário entre um conjunto de configurações possíveis.

2.4.5. Transformações entre Sistemas de Coordenadas

Aplicações gráficas freqüentemente requerem a transformação de descrições de objetos de um sistema de coordenadas para outro. Muitas vezes, o objeto é descrito em um sistema de coordenadas não-cartesiano (como as coordenadas polares mostradas na Figura 2.1 ou elípticas), e precisa ser convertido para o sistema de coordenadas Cartesianas. Em aplicações de animação e modelagem, objetos individuais são definidos em seu próprio sistema de coordenadas, e as coordenadas locais devem ser transformadas para posicionar os objetos no sistema de coordenadas global da cena.

2.5. TRANSFORMAÇÕES EM PONTOS E OBJETOS

A habilidade de representar um objeto em várias posições no espaço é fundamental para compreender sua forma. A possibilidade de submetê-lo a diversas transformações é importante em diversas aplicações da computação gráfica [Rogers, 1990]. As operações lineares de rotação e translação de objetos são chamadas operações de corpos rígidos. A seguir veremos algumas transformações em 2D e 3D.

2.5.1. Transformação de Translação

Transladar significa movimentar o objeto. Transladamos um objeto transladando todos os seus pontos, como mostrado na Figura 2.2. É possível efetuar a translação de pontos no plano (x,y) adicionando quantidades às suas coordenadas. Assim, cada ponto em (x,y) pode ser movido por T_x unidades em relação ao eixo x , e por T_y unidades em relação ao eixo y . Logo, a nova posição do ponto (x,y) passa a ser (x',y') , que pode ser escrito como:

$$\begin{aligned}x' &= x + T_x \\ y' &= y + T_y\end{aligned}$$

Repare que, se o ponto for representado na forma de um vetor, $P=(x,y)$, a translação de um ponto pode ser obtida pela adição de um vetor de deslocamento à posição atual do ponto: $P' = P + T = [x' \ y'] = [x \ y] + [T_x \ T_y]$.

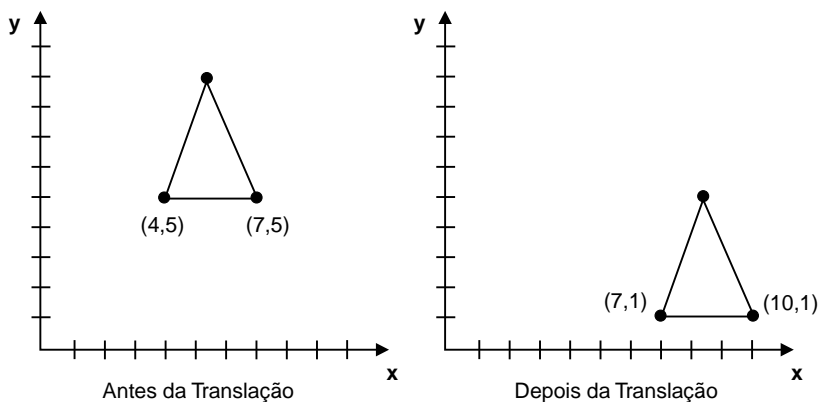


FIGURA 2.2. Translação de um triângulo de três unidades na horizontal e -4 na vertical. Repare que se teria o mesmo efeito transladando a origem do sistema de coordenadas para o ponto $(-3, 4)$ na primeira figura.

O mesmo ocorre se o ponto P for definido em 3D pelas coordenadas (x,y,z) . Pontos definidos em um espaço podem ser movimentados pela adição ou subtração de valores de translação às suas coordenadas. Assim, por exemplo, o ponto P definido por (x,y,z) pode ser reposicionado pelo uso de fatores de translação, como se segue:

$$\begin{aligned}x' &= x + Tx \\y' &= y + Ty \\z' &= z + Tz\end{aligned}$$

T_x , T_y , e T_z são valores de translação e definem um vetor. P' será definido por x' , y' e z' , que indicam novo posicionamento do ponto P , ou seja, suas novas coordenadas.

Se utilizarmos a notação matricial, essa translação se apresentará da seguinte forma:

$$[x' \ y' \ z'] = [x \ y \ z] + [T_x \ T_y \ T_z]$$

e pode ser descrita pela soma de dois vetores (ou matrizes): o vetor de coordenadas iniciais do ponto e o vetor de translação.

Os objetos são descritos pelos seus pontos. Assim, o triângulo da Figura 2.2 é definido pelos seus pontos extremos. A translação de um objeto formado por P pontos no espaço 3D é definida em função do vetor deslocamento T definido por (T_x, T_y, T_z) , aplicado a todos os seus pontos resultando em um objeto formado por pontos transformados P' :

$$P' = P + T = [x' \ y' \ z'] = [x \ y \ z] + [T_x \ T_y \ T_z]$$

ou seja, para transladar um objeto, alteramos todos os seus pontos pelo mesmo vetor T , o que nem sempre será a forma mais eficiente. No caso de uma linha, podemos realizar a operação somente em seus pontos limites e sobre estes redesenhar a nova linha. Isso também é verdade para as alterações de Escala e Rotação, que veremos a seguir.

2.5.2. Transformação de Escala

Escalonar significa mudar as dimensões de escala. A Figura 2.3 mostra essa transformação. Para fazer com que uma imagem definida por um conjunto de pontos mude de tamanho, teremos de multiplicar os valores de suas coordenadas por um fator de escala. Transformar um objeto por alguma operação nada mais é do que fazer essa operação com todos os seus pontos. Nesse caso, cada um dos vetores de suas coordenadas são multiplicados por fatores de escala. Estes fatores de escala em 2D podem, por exemplo, ser S_x e S_y :

$$x' = x \cdot S_x \quad y' = y \cdot S_y$$

Essa operação pode ser representada na forma matricial:

$$[x \ y] \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

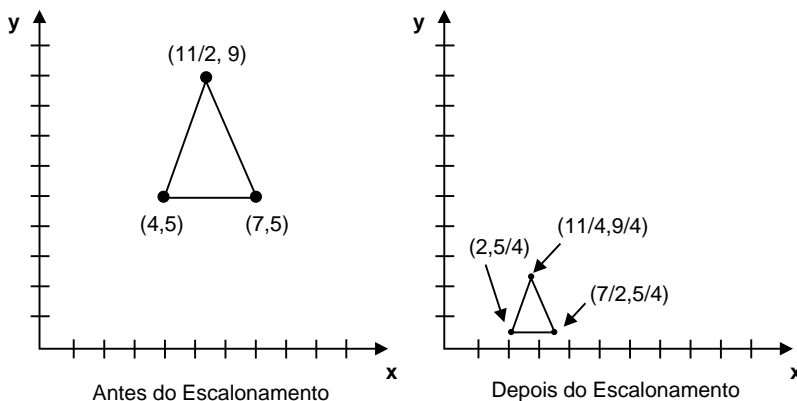


FIGURA 2.3. A mesma figura antes e depois de uma mudança de escala genérica, de $1/2$ na horizontal e $1/4$ na vertical. Repare que esse mesmo efeito relativo seria conseguido mudando a escala do sistema de eixos para uma outra que fosse o dobro da primeira na horizontal e quatro vezes maior na vertical.

A mudança de escala de um ponto de um objeto no espaço tridimensional pode ser obtida pela multiplicação de três fatores de escala ao ponto. A operação de mudança de escala pode ser descrita pela multiplicação das coordenadas do ponto por uma matriz diagonal cujos valores dos elementos não-nulos sejam os fatores de escala. Assim, no caso 3D tem-se:

$$[x' \ y' \ z'] = [x \ y \ z] \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{bmatrix} = [xS_x \ yS_y \ zS_z]$$

É importante lembrar de que, se o objeto não estiver definido em relação a origem, essa operação de multiplicação de suas coordenadas por uma matriz também fará com que o objeto translate. Veja o que ocorre com os triângulos das Figuras 2.3 e 2.4. Nessas figuras, os triângulos foram multiplicados respectivamente pelas matrizes:

$$\begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{4} \end{bmatrix} \quad \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$$

Um último ponto é que a transformação de escala não é uma alteração de corpo rígido já que geralmente deforma o corpo transformado (como os triângulos das Figuras 2.3 e 2.4).

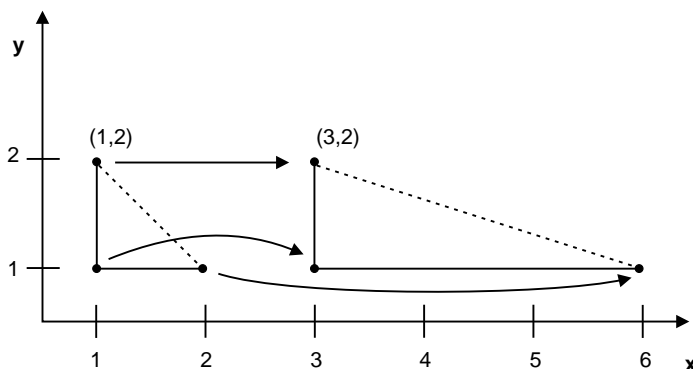


FIGURA 2.4. Mudança de escala de objetos não na origem é acompanhada por uma translação. Se os fatores de escala não forem iguais, o objeto também se deforma.

2.5.3. Transformação de Rotação

Rotacionar significa girar. A Figura 2.5 mostra a rotação de um ponto P em torno da origem, passando para a posição P' .

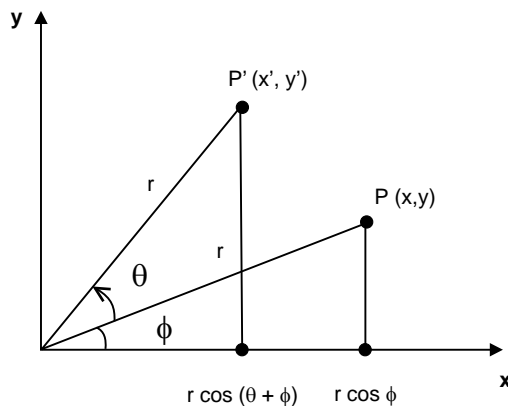


FIGURA 2.5. Rotação de um ponto P em torno da origem, passando para a posição P' . Repare que se chegaria a esse mesmo ponto através de uma rotação de $-\theta$ no sistema de eixos XY .

Se um ponto de coordenada (x, y) , distante $r = (x^2 + y^2)^{1/2}$ da origem do sistema de coordenadas, for rotacionado de um ângulo θ em torno da origem, suas coordenadas, que antes eram definidas como: $x = r \cdot \cos(\phi)$, $y = r \cdot \sin(\phi)$, passam a ser descritas como (x', y') dadas por:

$$\begin{aligned} x' &= r \cdot \cos(\theta + \phi) = r \cdot \cos\phi \cdot \cos\theta - r \cdot \sin\phi \cdot \sin\theta \\ y' &= r \cdot \sin(\theta + \phi) = r \cdot \sin\phi \cdot \cos\theta + r \cdot \cos\phi \cdot \sin\theta \end{aligned}$$

isso equivale às expressões:

$$\begin{aligned} x' &= x \cos(\theta) - y \sin(\theta) \\ y' &= y \cos(\theta) + x \sin(\theta) \end{aligned}$$

Essas expressões podem ser descritas pela multiplicação do vetor de coordenadas do ponto $(x \ y)$ pela matriz:

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

Essa matriz é denominada matriz de rotação no plano xy por um ângulo θ . No caso de o objeto não estar definido na origem do sistema de coordenadas, a multiplicação de suas coordenadas por uma matriz de rotação também resulta em uma translação, como mostrado na Figura 2.6.

Para alterar a orientação de um objeto em torno de um certo ponto realizando uma combinação da rotação com a translação, é necessário, antes de aplicar a rotação de um ângulo θ no plano das coordenadas em torno de um ponto, realizar uma

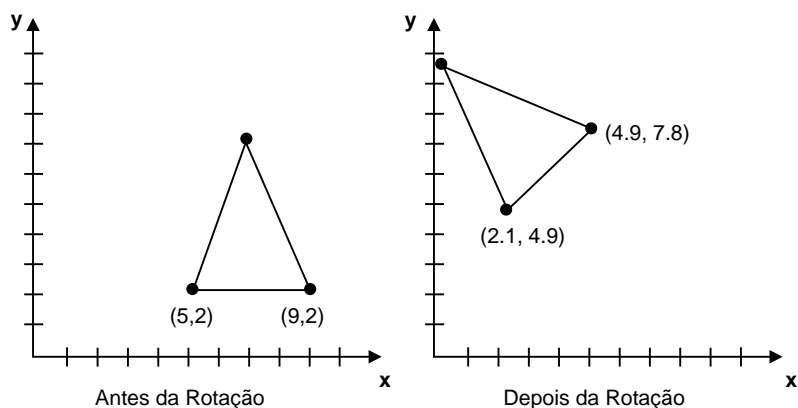


FIGURA 2.6. A multiplicação das coordenadas por uma matriz de rotação pode resultar em uma translação.

translação para localizar esse ponto na origem do sistema, aplicando a rotação desejada e, então, uma translação inversa para localizar o dado ponto na origem. A Figura 2.7 mostra isso.

Esse mesmo procedimento pode ser usado para alterar a escala de um objeto em torno de um certo ponto. Na realidade, diversos efeitos podem ser combinados de maneira análoga. Por exemplo, podemos realizar uma combinação da rotação e mudança de escala em torno de um ponto, combinando essas operações com a translação. Isto é, antes de aplicar a rotação de um ângulo θ e a mudança de escala, ambas em torno de um mesmo ponto, usamos uma translação para localizar esse ponto na origem do sistema, aplicamos a rotação e a mudança de escala desejadas e, então, usamos uma translação inversa para localizar o dado ponto na origem.

Parece óbvio, mas é bom frisar que qualquer alteração de corpo rígido de um objeto pode ser realizada por uma alteração inversa no sistema de coordenadas. Assim, girar em um ângulo α um objeto equivale a girar seu sistema de coordenadas $-\alpha$. Transladar um objeto de T_x, T_y equivale a transladar seu sistema de coordena-

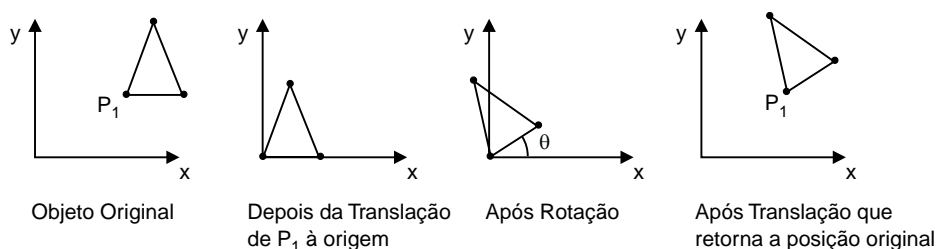


FIGURA 2.7. Processo de alteração da orientação de um objeto em torno de um certo ponto, que não na origem.

das de $-Tx$, $-Ty$. E modificar um objeto uniformemente por uma escala E , equivale a multiplicar a escala do sistema de eixos por $1/E$.

A rotação de objetos tridimensionais permite que vejamos o objeto de diferentes posições e ângulos. A rotação de um objeto 3D é mais simples de ser realizada individualmente sobre cada um dos eixos usando os denominados ângulos de Euler, que definiremos a seguir. Cada uma dessas rotações pode ser obtida a partir da análise de operações realizadas (nos planos xy , yz e zx) (2D).

Antes de generalizar as operações no plano (2D) para o espaço (3D), precisamos comentar sobre os sistemas 3D positivos ou negativos, pois isso vai afetar toda a forma de representação daqui para frente. Os sistemas de coordenadas com três eixos ortogonais podem ser descritos por diferentes posições dos eixos. Mesmo que se considere o eixo x como o horizontal e o eixo y como o vertical, o eixo z pode ser considerado apontando em duas direções. A direção positiva será a que obedecer a denominada *regra da “mão direita”* de ordenação dos eixos. Podemos descrever essa regra da seguinte forma: posicione sua mão direita aberta, na direção do primeiro eixo, vá girando a mão de modo que ela aponte para o segundo eixo, afaste o dedo dos demais dedos e veja se ele aponta no sentido do terceiro eixo. Se isso ocorrer, significa que as três direções formam um sistema de eixos positivo. Assim, o sistema de eixos à esquerda da Figura 2.8 é positivo e o da direita é negativo. Se forem usados os eixos x como no sentido horizontal da tela e o y como vertical, o eixo z deve apontar para fora da tela do computador, para termos um sistema de eixos positivos, como mostrado na Figura 2.9.

Rotações são usadas de duas maneiras em computação gráfica. Pode-se pensar em girar objetos no espaço em um certo ângulo ou rotacionar o próprio espaço com o ângulo (Figura 2.10). Um exemplo clássico da segunda opção ocorre nos simuladores de voo onde o piloto manipula a sua visão do espaço movimentando a cena. Essa segunda forma também facilita a representação das rotações como uma combinação de matrizes.

Os ângulos de Euler facilitam uma definição precisa das rotações em relação a um sistema de eixos. Esses ângulos definem a rotação em um plano pelo giro em

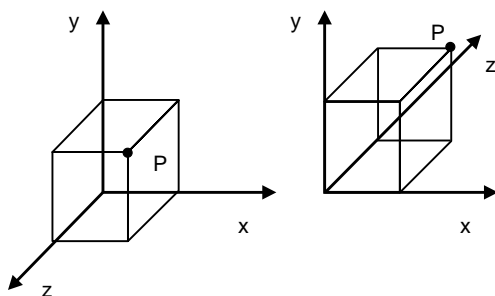


FIGURA 2.8. Sistemas de eixos positivos e negativos.

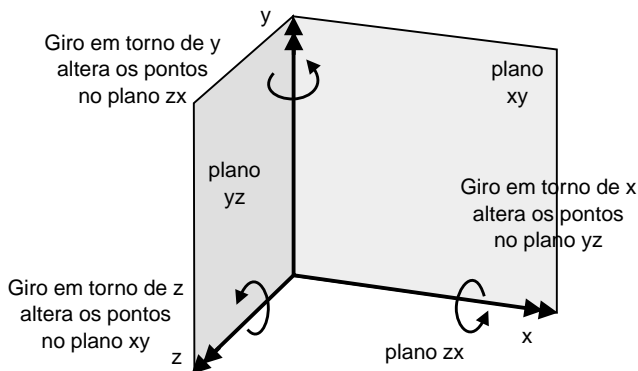


FIGURA 2.9. Definição dos três ângulos de Euler em relação aos eixos x , y e z .

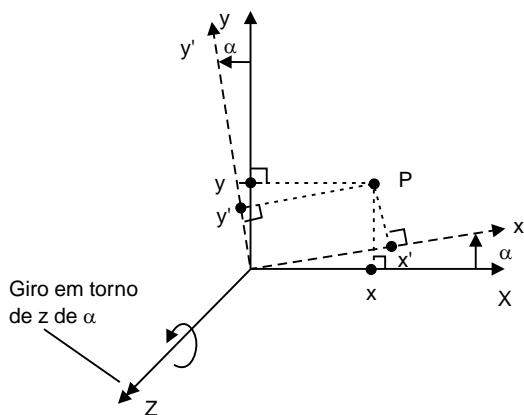


FIGURA 2.10. Ângulo de Euler em torno do eixo z . Só os pontos do plano xy são alterados. Pode-se observar nesta figura uma outra forma de entender a rotação: girar objetos no espaço de um certo ângulo ou rodar o próprio sistema de eixos com o ângulo em sentido oposto.

torno de um vetor normal a esse plano, e são muito usados na Mecânica (na descrição do movimento de peças e partes de máquinas) e na Física (para o posicionamento de objetos) [Kane, 1983].

Assim considerando o sistema de eixos mostrado na Figura 2.9, podemos definir três ângulos de Euler em relação aos eixos x , y e z , respectivamente. Um ângulo que define o giro em torno do eixo x para pontos no plano yz , outro ângulo que define o giro em torno do eixo y , para pontos no plano xz e um último ângulo que define o giro em torno do eixo z , para pontos no plano xy . O sentido positivo desses ângulos é definido pelo sentido de rotação dos dedos na regra da mão direita, quando o seu dedo aponta no sentido positivo de cada um dos eixos. Ao longo do resto desta obra, ao nos referirmos a ângulos em torno de uma direção ou eixo, é a este conceito de

ângulo de Euler (ou ângulo em torno de um eixo) que estaremos implicitamente nos referindo.

A rotação de um ponto no espaço tridimensional pode ser obtida pela multiplicação dos ângulos de rotação em torno dos eixos ao ponto. Esses ângulos são descritos em relação à direção dos três eixos, ou seja, da mesma maneira como descrevemos um ponto pelas suas três coordenadas nas direções do sistema de eixos usado: (β, δ, α) . Também definiremos uma rotação genérica pelos seus componentes nas três direções dos sistemas de eixos.

Em 3D, em vez de uma matriz de rotação, teremos três possíveis matrizes de rotação, uma para cada eixo. Mas essas matrizes não são simples extensão do caso 2D para o 3D. A forma da matriz de rotação é dependente do eixo sobre o qual se efetua a rotação.

A rotação ao redor do eixo z é produzida por uma matriz idêntica ao caso 2D apenas estendida para 3×3 . O eixo z permanece inalterado e os outros dois (x e y) giram no sentido positivo trigonometricamente, ou seja, anti-horário para quem olha o plano xy do eixo z (ou ainda positivo pela regra da mão direita). Assim, um giro de α graus em torno do eixo z muda as coordenadas de um ponto de $[x, y, z]$ para $[x', y', z']$ dados por:

$$[x' \ y' \ z'] = [x \ y \ z] * \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotação ao redor do eixo x, ou no plano yz, deixa o eixo x inalterado, enquanto as demais coordenadas são alteradas em função do ângulo de giro em torno do eixo x. Assim, um giro de β graus em torno do eixo x, muda as coordenadas de um ponto de $[x, y, z]$ para $[x', y', z']$ dados por:

$$[x' \ y' \ z'] = [x \ y \ z] * \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\beta) & \sin(\beta) \\ 0 & -\sin(\beta) & \cos(\beta) \end{bmatrix}$$

Rotação ao redor do eixo y, ou no plano zx, mantém inalterado o eixo y, enquanto os demais mudam em função do ângulo de giro do objeto (δ) ou do sistema de eixos ($-\delta$). Se δ for o ângulo em torno do eixo y, as coordenadas de um ponto serão modificadas por:

$$[x' \ y' \ z'] = [x \ y \ z] * \begin{bmatrix} \cos(\delta) & 0 & -\sin(\delta) \\ 0 & 1 & 0 \\ \sin(\delta) & 0 & \cos(\delta) \end{bmatrix}$$

Uma observação interessante é que essas matrizes são todas ortogonais e normalizadas, o que é denominado de **ortonormais**. Isto é, cada uma das suas colunas (ou

linhas), se consideradas como vetores teriam comprimento 1 (normalizadas) e seriam vetores mutualmente ortogonais, isto é, cujos produtos internos ou escalares resultariam em zero. Uma propriedade muito útil de matrizes ortonormais é que suas **inversas** são simplesmente suas **transpostas**.

A **inversa** de uma matriz M é a matriz denotada por M^{-1} , que ao ser multiplicada por M produz uma matriz identidade de mesma dimensão. Assim $I = M \cdot M^{-1}$ onde I representa a matriz identidade. No caso de matrizes de rotação como as anteriores, temos ainda que $R^{-1} = R^T$.

Outro ponto importante é que ao definirmos a rotação tridimensional por combinação de rotações definidas através de ângulos de Euler, teremos uma posição que é dependente da ordem de definição das rotações. Experimente dar três rotações em um objeto qualquer (uma caixinha, um disquete etc.) de 90° em torno dos seus eixos duas vezes, em ordens diferentes, e você verá a mesma face do objeto como resultado. Ou seja, os ângulos de Euler não definem as rotações de objetos de maneira **comutativa** (ou sem importar a ordem). Se isso for importante, você deve usar outra forma de definição de ângulos espaciais como os ângulos de Rodrigues, por exemplo [Kane, 1983].

O processo de **combinar** duas ou mais matrizes é chamado **concatenação** e é executado multiplicando as matrizes antes de aplicá-las aos pontos. Esse processo é especialmente produtivo quando se deseja aplicar muitas operações seguidas como escala e rotações em um conjunto de pontos. Deve-se, no entanto, ter em mente que a ordem de aplicação das transformações afeta o produto final. A multiplicação de matrizes não é necessariamente **comutativa**. Assim, se o ponto for girado 10° em torno de x , 20° em torno de y e 30° em torno de z , a matriz de rotação final será o resultado das três matrizes de rotação multiplicadas nessa ordem, ou seja, o produto das matrizes:

$$[x' \ y' \ z'] = [x \ y \ z] \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos 10^\circ & \sin 10^\circ \\ 0 & -\sin 10^\circ & \cos 10^\circ \end{bmatrix} \begin{bmatrix} \cos 20^\circ & 0 & -\sin 20^\circ \\ 0 & 1 & 0 \\ \sin 20^\circ & 0 & \cos 20^\circ \end{bmatrix} \begin{bmatrix} \cos 30^\circ & -\sin 30^\circ & 0 \\ \sin 30^\circ & \cos 30^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A **concatenação** também é muito útil para alterar a escala ao mesmo tempo em que roda objetos em torno de eixos específicos, e pode ser usada sempre que as transformações forem descritas por matrizes.

2.5.4. Transformação de Reflexão

A transformação de reflexão em torno de um eixo, ou espelhamento (ou flip), aplicada a um objeto, produz um novo objeto que é como se o objeto anterior fosse visto reproduzido por um espelho, posicionado no eixo em torno do qual se faz o espelhamento. No caso de uma reflexão 2D, o espelho pode ser considerado sobre o eixo

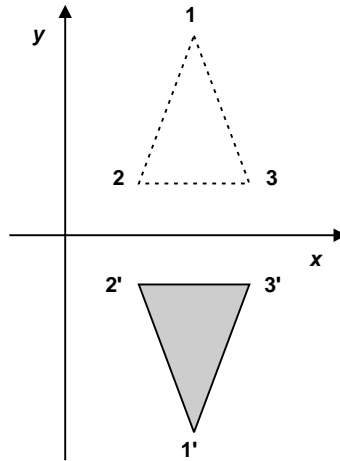


FIGURA 2.11. Reflexão de um objeto em torno do eixo x .

vertical ou horizontal, como mostrado na Figura 2.11. No caso de objetos 3D, a reflexão pode ser em torno de qualquer um dos três planos. Por exemplo, é possível aplicar uma reflexão em torno do plano xz (Figura 2.9), invertendo as coordenadas y , ou usando a seguinte matriz de transformação:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Essa transformação mantém as coordenadas x e z do objeto inalteradas, mas “inverte” os valores das coordenadas y , alterando a orientação espacial do objeto.

Podemos também definir uma reflexão em torno de dois eixos, por exemplo, xy . Nesse caso, a reflexão é feita em torno da origem do sistema de coordenadas, “invertendo” ambas as coordenadas x e y . A matriz dessa transformação é dada por:

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A Figura 2.12 exemplifica o resultado de uma reflexão em torno dos eixos x e y para uma figura no espaço bidimensional.

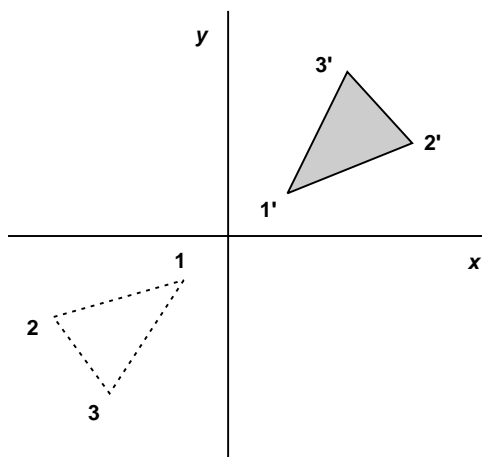


FIGURA 2.12. Reflexão de um objeto em torno da origem dos eixos xy .

2.5.5. Transformação de Cisalhamento

Cisalhamento (*Shearing ou Skew*) é uma transformação que distorce o formato de um objeto. Nela aplica-se um deslocamento aos valores das coordenadas x , y ou z do objeto proporcional ao valor das outras coordenadas de cada ponto transformado. Uma distorção na direção x , proporcional a coordenada y , pode ser produzida com a seguinte matriz de transformação:

$$\begin{bmatrix} 1 & 0 & 0 \\ S & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Nesse exemplo, as coordenadas do objeto são transformadas da seguinte maneira:

$$x' = x + S \cdot y, y' = y \text{ e } z' = z,$$

onde S é um valor fixo qualquer de modo que, se um cubo unitário for transformado por essa operação, passará a ter a forma mostrada na Figura 2.13, se $S=1$.

Qualquer número real pode ser usado como parâmetro, assim como é possível fazer a distorção em qualquer direção, por exemplo:

$$\begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & b & 1 \end{bmatrix}$$

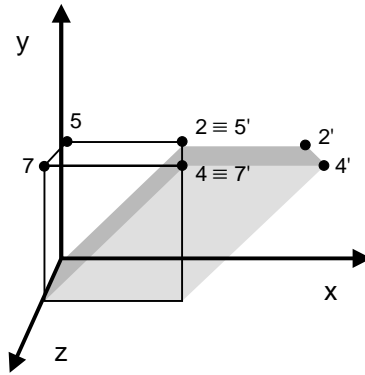


FIGURA 2.13. Efeito de cisalhamento (skew) em um cubo unitário.

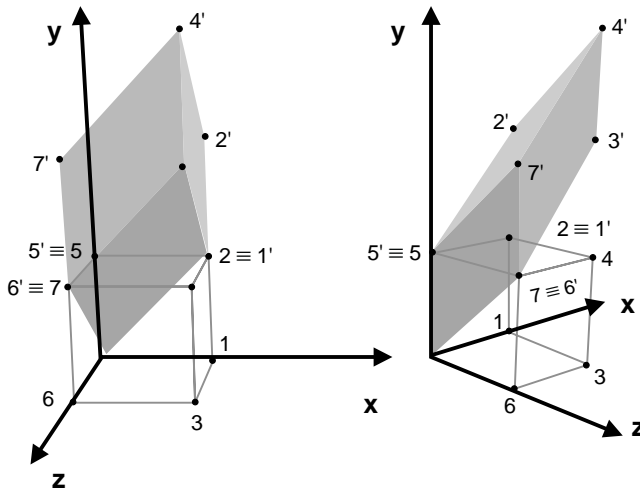


FIGURA 2.14. Duas vistas do mesmo cisalhamento na direção y em função das coordenadas x e y de cada ponto.

distorcerá um objeto ao longo da direção y de forma proporcional às coordenadas x e z de cada um dos seus pontos. Se nessa matriz $a=b=1$, o efeito produzido no cubo unitário, por essa transformação, seria o que mostra a Figura 2.14.

2.6. COORDENADAS HOMOGÊNEAS

As operações de cisalhamento, reflexão, rotação e escala podem ser facilmente executadas com o uso de matrizes. Assim, diversas operações podem ser concatenadas numa única matriz pela multiplicação prévia. As operações de translação ainda têm de ser conduzidas em separado, uma vez que sua aplicação depende de uma soma ou uma subtração vetorial.

Com o objetivo de otimizar a aplicação dessas operações, podemos usar um sistema de coordenadas denominado coordenadas homogêneas. Quando tratamos de representar um ponto no espaço 3D, no sistema cartesiano, fazemos uso das coordenadas (x,y,z) que posicionam o ponto no espaço em relação ao centro de coordenadas. O sistema de coordenadas homogêneas utiliza quatro valores para expressar um ponto P que será descrito por (x',y',z',M) . A transformação do sistema homogêneo para o cartesiano se dá pela seguinte relação: $(x,y,z)=(x'/M, y'/M, z'/M)$. Dizemos que dois conjuntos de coordenadas homogêneas, (x,y,z,M) e (x',y',z',M') , representam o mesmo ponto se, e somente se, um é múltiplo do outro. Assim, $(2,3,4,6)$ e $(4,6,8,12)$ são o mesmo ponto com diferentes representações. Isto é, cada ponto do espaço pode ter uma representação em uma infinidade de coordenadas homogêneas.

Os pontos onde $M=0$ estão, por definição, fora do espaço dimensional. Assim, M pode ter qualquer valor desde que não seja zero. No entanto, o uso do zero, apenas com o significado de algo que **tenda** a zero (ou seja, como um **símbolo** bem esclarecido), pode ser uma maneira conveniente de representar pontos no **infinito** [Mäntylä, 1988].

O uso de coordenadas homogêneas é importante para permitir a representação de reais por inteiros, por exemplo, $(1,2,1,1000)$ evita o uso de casas decimais e vírgulas. É também usado para evitar problemas ocasionados pela representação de números muito grandes: $(1, 2, 1, 1/1000000)$.

Quando $M=1$, a transformação entre os espaços é direta de modo que, $(x,y,1)$, no sistema homogêneo, tem os mesmos valores no espaço cartesiano 2D: (x,y) . O mesmo ocorrendo com as matrizes já definidas, como a de rotação, por exemplo, que pode ser escrita em coordenadas homogêneas como:

$$\text{Matriz de rotação} \rightarrow \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Para as matrizes de mudança de escala e rotação, a passagem da forma cartesiana normal para a homogênea é direta, assim em 2D teremos:

$$\text{Matriz de escala} \rightarrow \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Em 3D, (x,y,z) é $(x,y,z,1)$ no sistema homogêneo (divisão por 1), de modo que em 3D (lembre-se do comentado no final da Seção 2.3):

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

As coordenadas homogêneas permitem escrever as translações como matrizes de transformação. Isso faz as transformações geométricas ficarem uniformizadas pelo cálculo matricial e podem ser combinadas por concatenação (multiplicação) de matrizes.

A aplicação da translação em 3D por multiplicação de matrizes pode, nesse sistema de coordenadas, ser escrita na seguinte forma:

$$[x' \ y' \ z' \ 1] = [x \ y \ z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

2.7. PROJEÇÕES GEOMÉTRICAS

A transformação do plano 3D em 2D é estudada desde o século XIX, quando Gaspard Monge conceituou a geometria descritiva. Hoje, essa transformação é facilmente realizada por alguns softwares gráficos. Parâmetros como distância e altura podem ser determinados pelo usuário; o ângulo de visão (ou de tomada da cena) é preestabelecido pelo software. Nesses softwares, esses detalhes são interpretados de forma tridimensional, previamente conhecidos e manipulados de modo a serem projetados de maneira realista no plano do vídeo ou no papel.

Projeções permitem a visualização bidimensional de objetos tridimensionais. Para gerar a imagem de um objeto 3D, precisamos converter as coordenadas 3D em coordenadas 2D, que correspondam a uma visão do objeto de uma posição específica. Esse processo é chamado de projeção e pode ser observado na Figura 2.15.

Sendo a projeção de um objeto sua representação gráfica em um plano, e tendo os objetos três dimensões, para sua representação em um plano bidimensional, de-

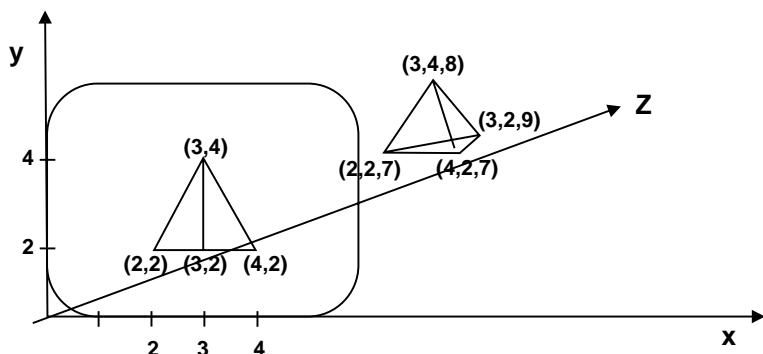


FIGURA 2.15. *Projeção das coordenadas de pontos do espaço para o plano.*

vem ser considerados os elementos básicos: plano de projeção, raio projetante e centro de projeção.

O plano de projeção é a superfície onde será projetado o objeto, ou seja, onde ele será representado em 2D, como mostra a Figura 2.16. Os raios de projeção são as retas que passam pelos pontos do objeto e pelo centro de projeção; na Figura 2.16 é representada pela reta que une o ponto P à origem do sistema de eixos. Centro de projeção é o ponto fixo de onde os raios de projeção partem. Um ponto se projeta no plano de projeção quando o raio projetante intercepta o plano de projeção, como mostrado na Figura 2.16. Todos os pontos visíveis do objeto devem ser projetados no plano de projeção.

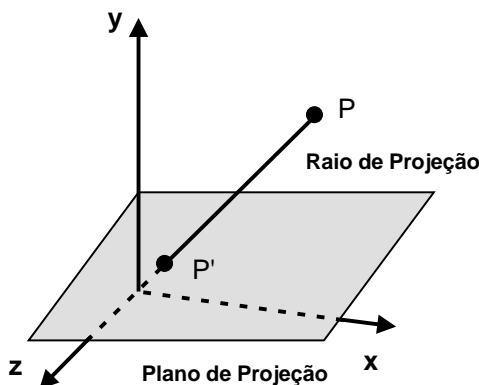


FIGURA 2.16. Exemplo da projeção de um ponto em um plano. O plano de projeção é a superfície onde os pontos do objeto serão projetados.

2.7.1. Classificação das Projeções Geométricas

As projeções geométricas são classificadas, conforme o organograma da Figura 2.17. As classificações dependem das relações entre o centro de projeção, o plano de projeção (onde o objeto aparece como 2D) e as direções das linhas ou raios de projeção.

Nas projeções paralelas, o centro de projeção é localizado no infinito, e todas as linhas de projeção são paralelas entre si (como é possível perceber nas Figuras 2.18 e 2.19). Nas projeções paralelas ortográficas, as linhas de projeção são paralelas entre si e perpendiculares ao plano de projeção (como a mostrada na Figura 2.18). As projeções oblíquas são produzidas por um conjunto de linhas de projeção inclinadas em relação ao plano de projeção de qualquer ângulo (como mostra a Figura 2.19).

A forma geral de definição de matrizes de projeção oblíquas usa um vetor unitário e sua projeção. As projeções oblíquas podem ser produzidas com ângulos de linhas de projeções diferentes em relação ao plano de projeção (Figura 2.19). Quando as linhas de projeção fazem um ângulo de 45 graus com o plano de projeção, os pontos

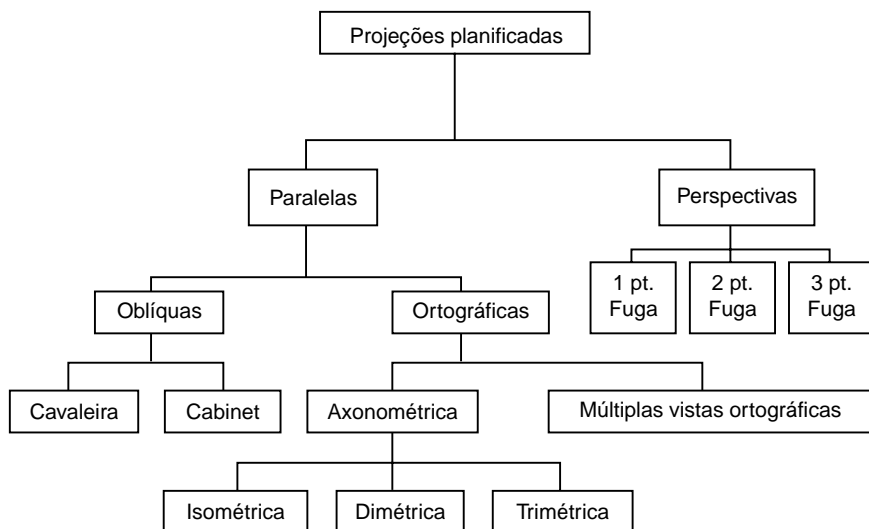


FIGURA 2.17. Classificação das projeções geométricas.

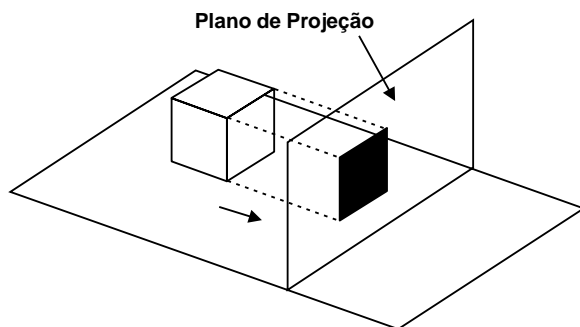


FIGURA 2.18. Projeção paralela ortográfica de um cubo em um plano.

projetados preservam sua medida original nas direções não-paralelas ao plano de projeção. Essa projeção oblíqua é chamada de cavaleira ou cavalier (Figura 2.20).

Não é importante, nesta classificação, o ângulo com que a direção não-paralela ao plano de projeção aparecerá na imagem projetada, assim ambas as imagens da Figura 2.20 são de projeções oblíquas cavaleiras, embora a profundidade do cubo apareça em ângulos diferentes em cada uma delas.

O outro tipo de projeção oblíqua é a paralela cabinet, que faz um ângulo específico com o plano de projeção, de modo a reproduzir objetos com uma dimensão de metade do tamanho original. Somente a face do objeto, paralela ao plano de projeção, permanece com o seu tamanho sem distorção (ou com a verdadeira grandeza). Esse ângulo é tal que tenha $\tan = 2$, ou seja, é de aproximadamente $63,4^\circ$ (Figura 2.21).

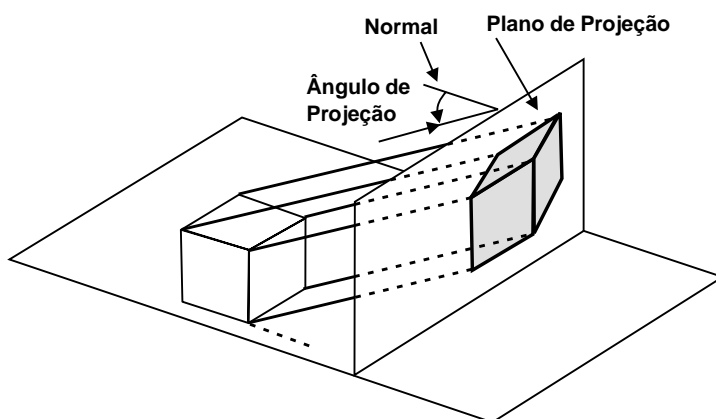


FIGURA 2.19. *Projeção paralela oblíqua de um cubo em um plano.*

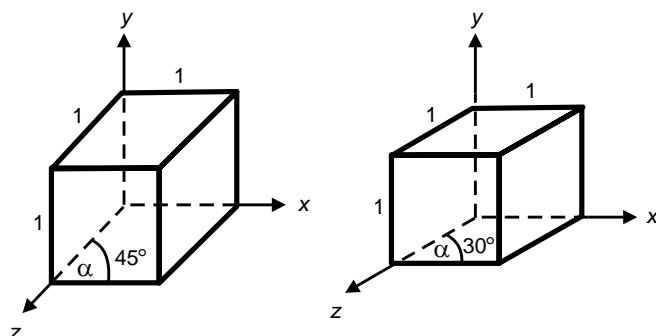


FIGURA 2.20. *Projeção paralela oblíqua cavaleira, onde o eixo não-paralelo ao plano de projeção aparece com ângulos de 45° e 30° .*

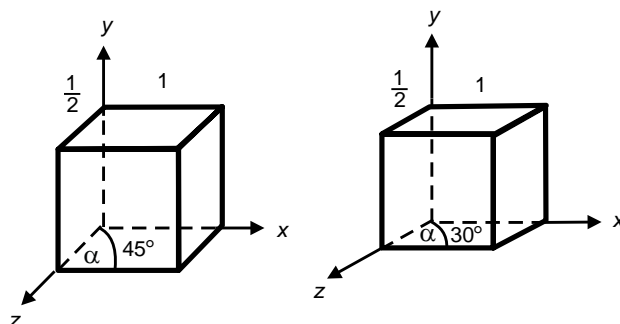


FIGURA 2.21. *Projeção paralela oblíqua cabinet, onde o eixo não-paralelo ao plano de projeção aparece com ângulos de 45° e 30° .*

A característica principal das classificações das projeções ortográficas é a direção que o plano de projeção faz com as faces principais do objeto a ser projetado. Nas múltiplas vistas ortográficas, o plano de projeção aparece paralelo aos planos principais (que representam as faces do objeto). Essas projeções mostram assim o objeto visto do topo (planta baixa), de frente e de lado (elevação). Nas projeções ortográficas (como mostra a Figura 2.18), se os planos principais do objeto forem paralelos ao plano de projeção, as faces do objeto perpendiculares ao plano de projeção não são vistas.

Dependendo de como as medidas do objeto aparecem no plano de projeção, recebem denominações especiais: isométrica, dimétrica ou trimétrica. Das projeções axonométricas, a mais comum e bastante utilizada em Engenharia é a projeção isométrica, as projeções dimétrica e trimétrica não são muito usuais. Na isométrica, o plano de projeção está posicionado em relação aos planos do objeto de maneira tal que os três eixos do objeto parecerão ter a mesma mudança nas métricas. Assim, se o objeto for um cubo, seus três lados parecerão continuar tendo a mesma medida quando projetados. Na projeção isométrica, os ângulos no plano de projeção entre os eixos principais projetados são iguais entre si, e iguais a 120° (Figura 2.22).

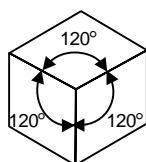


FIGURA 2.22. *Projeção paralela isométrica com ângulos no plano de projeção iguais entre si.*

As projeções definidas por raios de projeções paralelas são tradicionalmente usadas em Engenharia e desenhos técnicos. Em alguns casos, elas preservam as verdadeiras dimensões do objeto, mas não produzem uma imagem realista, por não tornarem menores as medidas mais distantes do observador (Seção 1.5). As perspectivas ou projeções cônicas são bastante mais realísticas na representação de objetos. Nas perspectivas, todos os raios de projeção partem do centro de projeção e interceptam o plano de projeção com diferentes ângulos. Nessas, os raios projetores não são paralelos, e as classificações se baseiam no número de pontos de fuga da imagem projetada, ou pontos onde as retas paralelas da imagem projetada parecerão convergir.

2.7.2. Projeções Paralelas Ortográficas

A característica principal das classificações nas projeções ortográficas é a direção que o plano de projeção faz com as faces principais do objeto a ser projetado. Nas diversas vistas ortográficas, o plano de projeção aparece paralelo aos planos princi-

pais (que representam as faces do objeto). Nesse caso, se o objeto tiver faces a 90° como no caso de cubos ou paralelepípedos, uma das faces simplesmente deixará de ser vista. Essas projeções mostram assim o objeto visto do topo (planta baixa), de frente e de lado (elevação).

Se um objeto estiver posicionado no espaço com seus eixos principais paralelos aos eixos do sistema de coordenadas, e quisermos suas projeções ortográficas em relação ao plano xy (ou $z=0$), a matriz em coordenadas homogêneas que produz o objeto projetado é:

$$[x' \ y' \ z' \ 1] = [x \ y \ z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Se, em vez de projetá-lo no plano $z=0$, for escolhido outro plano qualquer $z=T_z$, paralelo a este, a matriz de projeção pode ser obtida compondo uma matriz de translação com a matriz anterior, de modo que obteremos o objeto projetado após projetar cada um dos seus pontos, ou melhor multiplicá-los por:

$$[x' \ y' \ z' \ 1] = [x \ y \ z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & T_z & 1 \end{bmatrix}$$

Do mesmo modo, teremos as demais projeções ortográficas em planos paralelos aos demais planos principais do objeto. Assim, para projeções nas direções dos eixos y e z, ou seja, para planos paralelos ao plano x, $x=T_x$ teremos:

$$[x' \ y' \ z' \ 1] = [x \ y \ z \ 1] \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & 0 & 0 & 1 \end{bmatrix}$$

Para projeções paralelas aos eixos x e z, ou seja, em planos paralelos a y ($y=T_y$), teremos:

$$[x' \ y' \ z' \ 1] = [x \ y \ z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & T_y & 0 & 1 \end{bmatrix}$$

2.7.3. Projeções Paralelas Axométricas

Nas projeções axométricas, os planos do objeto são inclinados com relação ao plano de projeção estabelecendo alguma relação entre as medidas dos diversos eixos na forma projetada do objeto. Dependendo de como as medidas do objeto aparecem no plano de projeção, recebem denominações especiais: isométrica, dimétrica ou trimétrica.

Na isométrica, o plano de projeção está posicionado em relação aos planos do objeto de maneira tal que os três eixos do objeto parecerão ter a mesma mudança nas métricas, como (*iso*= *mesmo*, *métrica*= *medida*) o próprio nome indica. Assim, se o objeto for um cubo, seus três lados parecerão continuar tendo a mesma medida quando projetados. Na projeção isométrica, os ângulos das direções principais no plano de projeção (isto é, no objeto depois de projetado) parecem estar defasados entre si em 120° (Figura 2.22).

Para obter uma projeção isométrica usando métodos computacionais, uma série de rotações e translações é executada no objeto. Após essas transformações, uma projeção ortográfica, geralmente no plano vertical $z=0$, é efetuada, com a condição de que as linhas paralelas tenham o mesmo fator de redução. Esse fator é a relação entre o objeto projetado e a sua verdadeira grandeza. Assim, supondo que se tenha uma rotação ao redor do eixo y de um ângulo (δ) e ao redor do eixo x de (β), seguidas de uma projeção no plano $z=0$, podemos combinar estas matrizes em coordenadas homogêneas multiplicando as matrizes:

$$[x \ y \ z \ 1] \begin{bmatrix} \cos \delta & 0 & -\sin \delta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \delta & 0 & \cos \delta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \beta & \sin \beta & 0 \\ 0 & -\sin \beta & \cos \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

o que resulta na matriz:

$$[x \ y \ z \ 1] \begin{bmatrix} \cos \delta & \sin \delta \sin \beta & 0 & 0 \\ 0 & \cos \beta & 0 & 0 \\ \sin \delta & -\cos \delta \sin \beta & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x' \ y' \ z' \ 1]$$

Para obter os ângulos δ e β que resultam em uma projeção isométrica, poderemos verificar o comportamento de três vetores unitários na direção dos eixos x , y e z . Utilizando assim os vetores $(1 \ 0 \ 0 \ 1)$, $(0 \ 1 \ 0 \ 1)$ e $(0 \ 0 \ 1 \ 1)$, observamos que ao serem transformados pela matriz anterior, resultam respectivamente nos vetores: $(\cos \delta \ \sin \delta \sin \beta \ 0 \ 1)$, $(0 \ \cos \beta \ 0 \ 1)$ e $(\sin \delta \ -\cos \delta \sin \beta \ 0 \ 1)$. Esses vetores têm seus comprimentos iguais a: $(\cos^2 \delta + \sin^2 \beta \sin^2 \delta)^{1/2}$; $(\cos^2 \beta)^{1/2}$; e $(\sin^2 \delta + \sin^2 \beta \cos^2 \delta)^{1/2}$. Como

esses vetores devem ter todos as mesmas medidas depois de transformados (pela definição básica da projeção isométrica), igualando os valores de $x' = y'$ e $x' = z'$ obtém-se $\text{sen}^2\beta = 1/3$ e $\text{sen}^2\delta = 1/2$. O que resulta em ângulos $\beta \cong 35,26^\circ$ e $\delta = 45^\circ$. Assim a matriz de projeção isométrica é obtida substituindo estes valores na matriz anterior:

$$[x \ y \ z \ 1] \begin{bmatrix} 0,707 & 0,408 & 0 & 0 \\ 0 & 0,816 & 0 & 0 \\ 0,707 & -0,408 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x' \ y' \ z' \ 1]$$

Nas projeções dimétricas, em vez dos três eixos sofrerem as mesmas mudanças de escala, apenas dois eixos terão a mesma redução. Nesse caso, o posicionamento em relação ao plano de projeção não é único. Por exemplo, se for desejado que haja redução idêntica das medidas nos eixos x e y , obtém-se uma relação entre os ângulos β e δ igualando x' a y' . Desse modo, dado um ângulo β , o ângulo δ deve ser tal que: $\text{tag } \beta = \text{sen} \delta$. Com os valores dos ângulos β e δ , substituídos na mesma matriz, obtém-se a matriz que produz o objeto projetado segundo a forma desejada.

Nas projeções trimétricas, cada eixo sofrerá uma transformação de escala própria. Nesse caso, se for desejado a formulação de uma transformação específica, por exemplo, a redução de metade do valor em uma dada direção, chega-se a matriz desejada utilizando a mesma idéia básica já desenvolvida a partir dos vetores inicialmente unitários.

2.7.4. Projeção Perspectiva ou Cônica

A representação do espaço tridimensional (3D), da forma vista pelo olho humano, no plano bidimensional (2D), foi uma das mais importantes descobertas no mundo das artes introduzindo realismo nas pinturas e desenhos. A análise geométrica espacial e visual da imagem faz parte do cotidiano profissional do arquiteto e do artista. Ela é usada, em alguns casos, para proporcionar o posicionamento, exprimir o movimento ou caracterizar o trabalho de um artista, entre outros. Na arquitetura, a análise é determinante na distribuição espacial.

A partir de análises visuais, o arquiteto italiano Brunelleschi (1377-1446), descobriu a Perspectiva na busca de soluções geométricas para a construção da cúpula da Catedral de Florença. Após essa descoberta, outros artistas passaram a usar a perspectiva nas suas representações. A projeção perspectiva, ao contrário da projeção paralela, produz uma imagem realista, porém não pode reproduzir suas verdadeiras medidas (Figura 1.1). A projeção perspectiva é uma transformação dentro do espaço tridimensional e suas projeções representam a cena vista de um ponto de observação a uma distância finita. Nela, o centro de projeção está a uma certa distância da cena, enquanto nas projeções paralelas ele está no infinito.

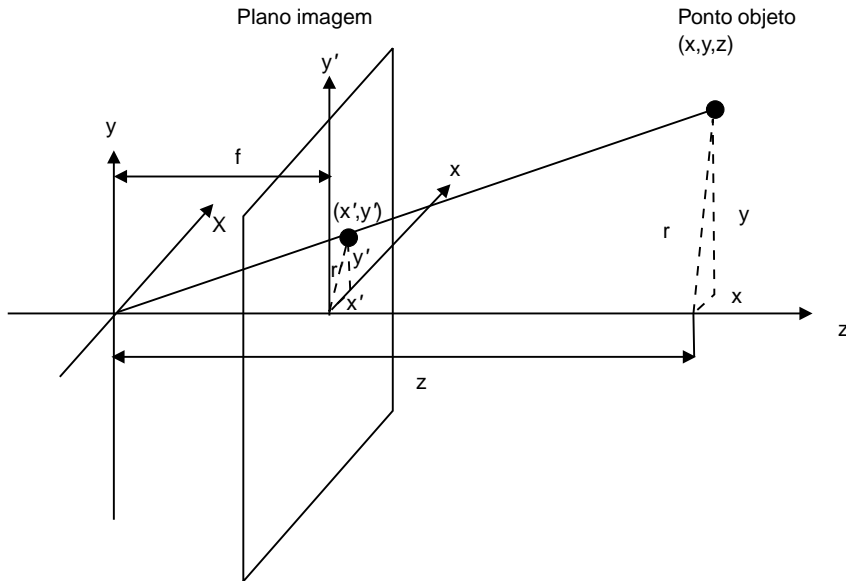


FIGURA 2.23. As coordenadas dos pontos projetados em perspectivas são obtidas pela interseção dos raios projetores com o plano de projeção.

Se for suposto que o centro de projeção coincide com a origem do sistema de eixos e se desejar a projeção de um ponto qualquer do espaço de coordenadas (x,y,z) em um plano $z=f$, como mostrado na Figura 2.23, temos que as coordenadas do ponto projetado, (x',y',z') , por semelhança de triângulos são:

$$\begin{aligned} r^2 &= x^2 + y^2 \\ r'^2 &= x'^2 + y'^2 \end{aligned}$$

por semelhança de triângulos, são:

$$\frac{f}{z} = \frac{r'}{r}$$

onde:

$$\begin{aligned} r' &= x' + y' \\ r &= x + y \end{aligned}$$

Considerando o triângulo formado pelas coordenadas x e y e a distância r , bem como o triângulo formado pelas coordenadas (x',y') e a perpendicular r' por semelhança, tem-se:

$$\frac{x'}{x} = \frac{y'}{y} = \frac{r'}{r}$$

Combinando essas expressões temos:

$$\frac{x'}{x} = \frac{f}{z} \quad \text{e} \quad \frac{y'}{y} = \frac{f}{z}$$

A posição do ponto $(x', y', z'=f)$ no plano da imagem é dado pelas seguintes equações.

$$x' = \frac{f}{z} x \quad y' = \frac{f}{z} y$$

Assim sendo, as coordenadas do ponto projetado dependem da sua própria distância, z , ao plano de projeção. Repare que a relação (f/z) anterior representa um fator de escala para as transformações entre as coordenadas. Isso explica porque cada ponto do objeto em perspectiva parece reduzido por um fator de escala próprio.

Se transladarmos o sistema de eixos de f_z unidades na direção do eixo z , o centro de projeção passará para a coordenada $(0,0,-f_z)$ e o plano de projeção será $z=0$. Nesse caso, todas as coordenadas do ponto projetado, menos a coordenada z (que agora passará a ser $z=0$), permanecerão inalteradas, e serão obtidas pelas relações anteriores. Assim, podemos descrever as expressões anteriores na forma matricial:

$$[x \ y \ z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{-1}{f_z} \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x' \ y' \ z' \ 1]$$

Essa matriz pode ser vista como a concatenação da matriz de transformação pela vista em perspectiva a partir de um ponto com uma matriz de projeção ortográfica no plano $z=0$:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{-1}{f_z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Se quisermos que o centro de projeção esteja em uma posição qualquer (f_x, f_y, f_z) e não sobre um dos eixos, podemos transladar o ponto para a origem, usar a matriz de projeção e depois transladar novamente a posição desejada. Ou seja, usar a solução de concatenar matrizes. As expressões a seguir mostram como se faz para ter a matriz definida para centro de projeção em (f_x, f_y, f_z) .

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -f_x & -f_y & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{-1}{f_z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ f_x & f_y & 0 & 1 \end{bmatrix}$$

As matrizes de projeção em perspectiva são obtidas usando a coluna da matriz genérica 4x4 correspondente às coordenadas homogêneas. Quando um dos elementos dessa submatriz 3x1 for diferente de zero, o objeto transformado sofre uma transformação perspectiva. Nesse caso, como usamos uma matriz de projeção com centro sobre o eixo z, apenas um desses elementos deve ter valor diferente de zero. Exatamente o correspondente à coordenada z. Se o centro de projeção for localizado ao longo do eixo x, e o plano de projeção for o plano x=0, a matriz de projeção de perspectiva de um ponto será:

$$[x \ y \ z \ 1] \begin{bmatrix} 0 & 0 & 0 & \frac{-1}{f_x} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x' \ y' \ z' \ 1]$$

Do mesmo modo, se o centro de projeção for localizado ao longo do eixo y, e o plano de projeção for o plano y=0, a matriz de projeção de perspectiva de um ponto será:

$$[x \ y \ z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{-1}{f_y} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x' \ y' \ z' \ 1]$$

Para obter as matrizes de perspectivas de dois ou três pontos, podemos definir adequadamente os elementos da matriz 4x4 e depois multiplicá-los adequadamente por uma matriz de projeção ortográfica. Assim, matrizes de projeção perspectiva em dois pontos localizados nos eixos x e z, ou x e y são definidas como:

$$\begin{bmatrix} 1 & 0 & 0 & \frac{-1}{f_x} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{-1}{f_z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & \frac{-1}{f_x} \\ 0 & 1 & 0 & \frac{-1}{f_y} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (11)$$

Para a definição de matrizes em perspectivas de três pontos, podemos pensar em diversas concatenações de matrizes, como as anteriores, seguidas de uma projeção ortográfica no plano desejado [Gardan, 1986]. Desse modo, uma matriz de perspectiva em três pontos é definida como:

$$\begin{bmatrix} 1 & 0 & 0 & \frac{-1}{f_x} \\ 0 & 1 & 0 & \frac{-1}{f_y} \\ 0 & 0 & 1 & \frac{-1}{f_z} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.8. ESPECIFICAÇÃO DOS PONTOS DE FUGA

Os desenhos em perspectiva são caracterizados pela mudança do comprimento e pelos pontos de fuga. O primeiro é uma ilusão que nos mostra objetos cada vez menores à medida que sua distância do centro de projeção aumenta (Figura 1.1). Pontos de fuga também são uma ilusão, neste caso de que conjuntos de linhas paralelas (não-paralelas ao plano de projeção) convergem para um ponto, denominado de fuga (Figura 2.24).

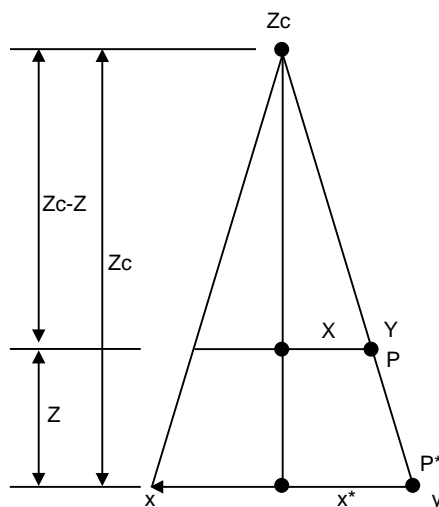


FIGURA 2.24. Exemplos de uma projeção em perspectiva com 2 e 3 pontos de fuga.

Denominam-se pontos de fuga principais os que aparentam uma interseção entre um conjunto de retas paralelas com um dos eixos principais x , y ou z . O número de pontos de fuga principais é determinado pelo número de eixos principais interceptados pelo plano de projeção. Assim, se o plano de projeção intercepta apenas o eixo z , somente o eixo z possui um ponto de fuga principal, pois linhas paralelas aos eixos x e y são também paralelas ao plano de projeção, e dessa forma não ocorre a ilusão de convergência.

Projeções Cônicas ou Perspectivas são categorizadas pelo número de pontos de fuga principais, ou seja, o número de eixos que o plano de projeção intercepta, como descrito na Figura 2.17. Na seção anterior, as matrizes de projeção foram obtidas a partir da especificação dos centros de projeção e não dos pontos de fuga. Para estabelecer a localização desses pontos e sua relação com a forma de cada uma das matrizes, vamos considerar a transformação pela matriz perspectiva de um ponto localizado no infinito na direção z. Esse ponto pode ser escrito em coordenadas homogêneas como $[0, 0, 1, 0]$. O número zero na posição da coordenada homogênea leva um ponto na forma cartesiana $[x, y, z]$ para o infinito [Gardan, 1986]. Aplicando a transformação em perspectiva a este ponto teremos:

$$[0 \ 0 \ 1 \ 0] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & f_z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & -1/f_z \end{bmatrix}$$

Usando a transformação homogênea, esse ponto se torna $[0, 0, -f_z, 1]$, que define o ponto de fuga, ou o ponto onde as retas se encontrarão no eixo z. Procedimento idêntico pode ser usado para obtenção dos pontos de fuga nas outras direções, ou para determinação dos dois ou mesmo três pontos de fuga que as matrizes das expressões anteriores representam. Assim, para a última matriz da Seção 2.7.4 tem-se três pontos de fuga:

$$\begin{aligned} &[-f_x, 0, 0, 1], \text{ sobre o eixo } x, \\ &[0, -f_y, 0, 1], \text{ sobre o eixo } y, \text{ e} \\ &[0, 0, -f_z, 1] \text{ sobre o eixo } z. \end{aligned}$$

Projeções perspectivas com um e dois pontos de fuga (quando um ou dois eixos principais são interceptados pelo plano de projeção) são mais comumente usadas em arquitetura e desenho publicitário. Já as projeções com três pontos de fuga são bem menos utilizadas, pois adicionam um certo surrealismo à cena, talvez só ocorram ao se visualizar a esquina de um prédio altíssimo, quando até mesmo as linhas na direção de sua altura deixarão de ter a aparência de paralelas.

2.9. CÂMERA VIRTUAL

A fotografia que se obtém com uma máquina fotográfica real é uma projeção da cena em um plano, que corresponde ao filme. Da mesma forma que no mundo real, a imagem que se obtém da cena sintética depende de vários fatores que determinam como esta é projetada em um plano para formar a imagem 2D exibida em algum dispositivo, como, por exemplo, o vídeo. Ao gerar imagens de cenas 3D em computa-

ção gráfica, é comum fazermos uma analogia com uma máquina fotográfica. Nessa analogia, imaginamos um observador que, posicionado em um ponto de observação, vê a cena através das lentes de uma câmera virtual que pode ser posicionada de forma a obter a imagem da cena, e onde pode-se definir, além da posição da câmera, sua orientação e foco, o tipo de projeção usada e a posição dos planos que limitam a visibilidade da cena, os chamados *clipping planes*.

A posição e o ponto focal da câmera definem, respectivamente, onde a câmera está e para onde está apontando. O vetor que vai da posição da câmera ao ponto focal é denominado direção de projeção. O plano de imagem, que é o plano no qual a cena será projetada, está posicionado no ponto focal e, na maioria dos casos, é considerado perpendicular ao vetor de direção de projeção. A Figura 2.25 mostra esses elementos. A orientação da câmera é controlada pela sua posição (x, y, z) , seu ponto focal (ponto D, na Figura 2.25) e pelo vetor que indica o “lado de cima” da cena 3D, denominado de *view up*. Esses parâmetros definem a câmera.

O tipo de projeção usado controla como a cena é mostrada no plano de imagem. Na projeção paralela, o processo de mapeamento é feito pelos raios de projeção paralelos e assume-se que todos os raios que atingem a câmera são paralelos à direção de projeção. Na projeção perspectiva, todos os raios convergem para um ponto comum, denominado ponto de observação ou centro da projeção. Nesse caso, é importante determinar o ângulo de visão da câmera.

Os planos de recorte anterior e posterior interceptam a direção de projeção (Figura 2.25) e são, geralmente, perpendiculares a ela. Os planos de recorte são usados para eliminar partes da cena que estão muito próximas ou muito distantes da câmera, de forma que as que estão na área interior aos planos de recorte serão visíveis.

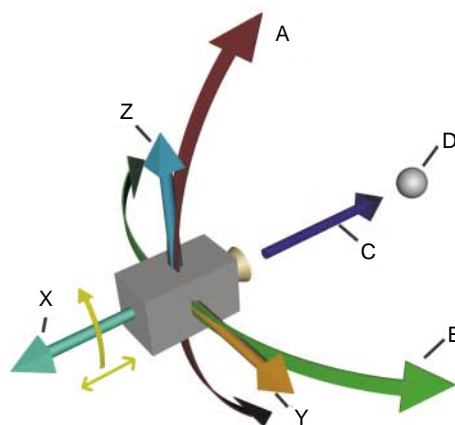


FIGURA 2.25. Coordenadas da posição da câmera, e seus 7 graus de liberdade: localização no espaço (x,y,z) , ângulos de rotação em torno de cada um dos eixos (setas curvas) e foco.

Em geral, os planos de recorte são perpendiculares à direção de projeção. Suas posições podem ser especificadas usando o chamado intervalo de visibilidade (denominado geralmente de *clipping range*) da câmera.

2.10. TRANSFORMAÇÕES GEOMÉTRICAS COM OpenGL

Os vértices que definem as primitivas geométricas são definidos no OpenGL em um sistema de eixos ortogonais orientados segundo a regra da mão direita. O eixo x será horizontal e orientado da esquerda para a direita, enquanto o eixo y será vertical, orientado de baixo para cima.

2.10.1. Transformação de Translação

O comando para a translação é `glTranslate(TYPE x, TYPE y, TYPE z)` e tem como parâmetros as distâncias em cada um dos eixos coordenados. Por exemplo, para mover o ponto de visão ao longo do eixo z de 5 unidades, usa-se:

```
glTranslatef(0.0f,0.0f,-5.0f);
```

A sequência de comandos para mover todos os pontos de um objeto, é dada pelas linhas:

```
DesenhaObjeto( );    // Desenha o objeto na posição nas coordenadas originais
glTranslatef(10,10,10);
DesenhaObjeto( );    // Desenha o objeto deslocado de 10 unidades em cada eixo
```

2.10.2. Transformação de Escala

O comando que permite a mudança das dimensões de um modelo é `glScale(TYPE x, TYPE y, TYPE z)` e muda as escalas relativas a cada eixo principal. O comando `glScale` é relativamente simples mas a utilização do fator de escala não-uniforme afetará os objetos desenhados. Por exemplo, para triplicar a altura de um objeto usa-se:

```
glScalef(1.0f,3.0f,1.0f);
```

2.10.3. Transformação de Rotação

O comando para a rotação é `glRotate(TYPE angle, TYPE x, TYPE y, TYPE z)` e tem como parâmetro o ângulo de rotação e as coordenadas de um vetor que determina o eixo de rotação (ângulos de Euler). Para a rotação ser feita em torno de um dos eixos principais, deve-se definir x, y e z apropriadamente como os vetores unitários nas direções destes eixos. Por exemplo, a rotação do ponto de visão de 30° em torno do eixo x pode ser definida como:

```
glRotatef(30.0f,1.0f,0.0f,0.0f);
```

2.10.4. Matrizes de Transformação

O OpenGL mantém três matrizes de transformação **ModelView**, **Projection** e **View-Point**, que são usadas para transformar um ponto qualquer dado em um ponto da janela de visualização. Cada ponto especificado é multiplicado por essas três matrizes.

O comando **glMatrixMode** deverá ser utilizado antes dos comandos que especificam as formas de projeção, que são definidas com os comandos: **glOrtho**, **glFrustum** ou **gluPerspective**.

Essas matrizes podem ser modificadas por uma série de comandos como **glTranslate**, **glRotate** ou **glScale**, entre outros. A matriz que será modificada é definida pelo comando **glMatrixMode** executado com uma das constantes **GL_MODELVIEW**, **GL_PROJECTION** ou **GL_TEXTURE**, como mostra o seguinte exemplo:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity( );
gluPerspective(30.0,width/height,1.0,10.0); //
glMatrixMode(GL_MODELVIEW);
glLoadIdentity( );
glTranslatef(0.0f,0.0f,-5.0f);
glRotatef(30.0,1.0,0.0,0.0);
```

As transformações geométricas aplicadas usando comandos OpenGL do tipo **glTranslate**, **glRotate** ou **glScale**, são sempre armazenadas em uma matriz chamada **MODELVIEW**. A cada chamada de uma dessas funções, o OpenGL cria uma matriz de transformação específica para a função e a seguir multiplica essa matriz pela matriz **MODELVIEW** atual.

Por exemplo, na chamada da função **glTranslatef** $(-3, 2, -8)$ é criada a seguinte matriz:

| | | | |
|-------|------|-------|------|
| 1.00 | 0.00 | 0.00 | 0.00 |
| 0.00 | 1.00 | 0.00 | 0.00 |
| 0.00 | 0.00 | 1.00 | 0.00 |
| -3.00 | 2.00 | -8.00 | 1.00 |

Note que na última linha da matriz são colocados os valores passados como parâmetros para a função. No momento de exibir um objeto, o que o OpenGL faz é a multiplicação dessa matriz pelos vértices do objeto, obtendo assim a posição final do objeto.

2.10.5. Armazenando as Transformações na Matriz

Para armazenar as transformações geométricas de um objeto em uma matriz, deve-se chamar a função **glGetFloatv** imediatamente após serem efetuadas todas as transformações do objeto.

```

glPushMatrix( );
    glLoadIdentity( );
    glTranslatef (PosObj.X, PosObj.Y, PosObj.Z);
    // a próxima linha guarda a matriz de transformação do objeto para um futuro uso
    glGetFloatv(GL_MODELVIEW_MATRIX, &MatObj[0][0]);
glPopMatrix( );

```

2.10.6. Alterando a Matriz de Transformação

Uma vez que se tem a matriz de transformação de um objeto, é possível aplicar essas transformações multiplicando-se essa matriz pela matriz atual. Isso é feito antes de desenhar um objeto da seguinte forma:

```

glColor3f(0.0f,0.0f,0.6f);
glPushMatrix( );
    glMultMatrixf(&MatObj[0][0]);
    DesenhaCubo( );
glPopMatrix( );

```

2.10.7. Montando Transformações Hierárquicas

A montagem de uma hierarquia de transformações serve para que se possa definir um objeto como filho de outro. Nesse caso, o objeto filho sofrerá todas as transformações geométricas aplicadas a seu pai. Para tanto, deve-se apenas desenhar o objeto filho logo após o desenho do pai. O exemplo a seguir ilustra esse procedimento.

```

glPushMatrix( );
    glMultMatrixf(&Mat_Pai[0][0]); // aplica as transformações do pai
    DesenhaCubo( );
    GLfloat MatInv[4][4];
        TemFilho = true;
    Se (TemFilho)
    se vínculo = false
    // DESFAZ TRANSFORMAÇÕES DO PAI
    // calcula a inversa da matriz do Pai
    CopyMatrix(MatInv_Pai, Mat_Pai);
    M_invert(MatInv_Pai);
    // Multiplica a Matriz do filho pela inversa da matriz do pai
    M_mult(MatObj, MatInv_Pai); // MatObj = MatObj * MatInv_Pai;
    vínculo = true;
    senão
        DesenhaObjeto( ); // desenha Filho
glPopMatrix( );

```

No momento em que o objeto filho for desenhado pela primeira vez após o pai, a matriz do filho deve desfazer as transformações existentes no pai naquele momento. Para tanto, deve-se multiplicar a matriz do filho pela **inversa da matriz** do pai. Note que isso deve ser feito somente no instante em que se estabelece o vínculo entre pai e filho. Isso porque só se deve desfazer as transformações do pai que existem no momento do estabelecimento do vínculo. A partir desse momento, se uma nova transformação for aplicada ao pai, ela deverá ser também aplicada ao filho.

2.10.8. Desfazendo o Vínculo de Hierarquia

Para desfazer o vínculo de hierarquia e deixar o objeto no mesmo lugar, deve-se multiplicar sua matriz pela matriz atual de seu pai. Se isso não for feito, ocorrerá uma translação indesejada no objeto filho pois este deixará de sofrer as translações do pai após o término do vínculo.

```
TemFilho = FALSE;  
M_mult(MatObj, MatPai); // MatObj = MatObj * Mat_Pai
```

2.10.9. Matriz Genérica de Projeção

Em OpenGL, para especificar sua própria matriz de projeção, é necessário declarar uma matriz de 16 elementos (4×4) da seguinte maneira:

```
GLfloat mprojecao [16]; // declaração de uma matriz 4x4
```

Para inserir os valores na matriz deve-se utilizar o seguinte comando de atribuição:

```
mprojecao[ posição ] = valor a ser inserido;
```

Para inserir, por exemplo, os valores correspondentes a primeira coluna da matriz (valores que são constantes), temos:

```
mprojecao[0] = 1; // inserindo 1 em A0  
mprojecao[1] = 0; // inserindo 0 em A1  
mprojecao[2] = 0; // inserindo 0 em A2  
mprojecao[3] = 0; // inserindo 0 em A3
```

Depois de todos os valores inseridos na matriz, ela já está pronta para ser utilizada. Para carregar essa matriz, deve-se utilizar o seguinte método:

```
glLoadMatriz(mprojecao);
```

2.10.10. Projeção Paralela Ortogonal

A projeção ortogonal é selecionada pelo comando `glOrtho` e utiliza seis parâmetros que definem um volume de visão.

```
glOrtho(left, right, bottom, top, near, far);
```

onde: near é a menor distância desejada para um objeto visível. Se a distância entre um objeto e o observador for menor do que este número, este objeto não será visível; e

far é a maior distância desejada para um objeto visível. Se a distância entre um objeto e o observador for maior do que este número, este objeto não será visível.

2.10.11. Projeção em Perspectiva

A projeção em perspectiva é selecionada com o comando **glFrustrum**, que também utiliza seis valores para definir o volume de visão em forma de tronco de pirâmide (frustum).

```
glFrustrum(left,right,bottom,top,near,far);
```

Pode ser usado também o comando **gluPerspective**, da biblioteca de utilitários.

```
gluPerspective(angle,aspect,near,far);
```

onde: angle é o ângulo, em graus, na direção y (usada para determinar a “altura” do volume de visualização);

aspect é a razão de aspecto que determina a área de visualização na direção x, e seu valor é a razão entre x (largura) e y (altura);

2.10.12. Posição da Câmera

Através dos seus argumentos, é possível indicar a posição da câmera e para onde ela está direcionada, utilizando:

```
void gluLookAt( GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz );
```

onde: eyex, eyey e eyez são usados para definir as coordenadas x, y e z, respectivamente, da posição da câmera (ou observador). Na Figura 2.25, x, y e z correspondem a eyex, eyey e eyez e D é o ponto focal. B e A são o movimento nos eixos x, y e z que mantém a distância da câmera em relação ao ponto focal e C é a direção da projeção.

centerx, centery e centerz são usados para definir as coordenadas x, y e z, respectivamente, da posição do alvo, isto é, para onde o observador está olhan-

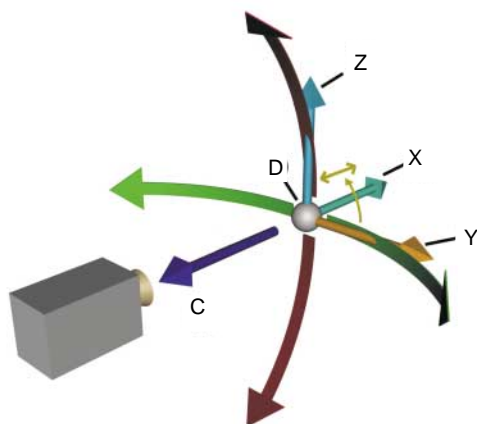


FIGURA 2.26. *Coordenadas da posição do alvo.*

do (normalmente, o centro da cena). Na Figura 2.26, x , y e z correspondem a $centerx$, $centery$ e $centerz$; D é o ponto focal e C é a normal do plano de visão.

upx , upy e upz são as coordenadas x , y e z , que estabelecem o vetor up (indica o “lado para cima” de uma cena 3D).

RESUMO

Neste segundo capítulo discutimos a transformação de objetos e pontos: como eles podem ser modificados por transformações (de escala, rotação, translação e outros) e como podem ser representados em planos mesmo sendo tridimensionais. Para uma adequada compreensão da teoria relacionada foram representados diversos sistemas de coordenados, as formas de utilizar vetores e matrizes e as coordenadas homogêneas.

CAPÍTULO 3

Curvas e Superfícies

3.1. Representação de Curvas

3.1.1. Conjunto de Pontos

3.1.2. Representação Analítica

3.1.3. Formas não-Paramétricas de Representar Curvas

3.1.4. Formas Paramétricas de Representar Curvas

3.1.5. Curvas Paramétricas de Terceira Ordem

3.1.6. Hermite

3.1.7. Bézier

3.1.8. Splines

3.1.8.1. Splines Uniformes e Periódicas

3.1.8.2. Splines Não-Periódicas

3.1.8.3. Splines Não-Uniformes

3.1.8.4. Desenvolvimento da Formulação Genérica de B-Splines

3.1.8.5. Catmull-Rom Splines

3.1.9. Curvas Racionais

3.2. Superfícies

3.2.1. Superfícies de Revolução

3.2.2. Superfícies Geradas por Deslocamento

3.2.3. Superfícies Geradas por Interpolação Bi-linear

3.2.4. Interpolações Trilineares

3.2.5. Superfícies de Formas Livres

3.2.6. Superfícies Paramétricas Bicúbicas

3.2.7. Superfícies de Hermite

3.2.8. Superfícies de Bézier

3.2.9. Superfícies de B-Spline

3.2.10. Normais a Superfícies

3.2.11. Superfícies Racionais

3.2.12. NURBS

3.2.12.1. NURBS em OpenGL

3.2.13. Subdivisão de Superfícies com Catmull-Clark
e NURMS

Curvas e superfícies desempenham um papel importante em diversas áreas tanto na criação de objetos sintéticos quanto na visualização de fenômenos científicos. Na modelagem geométrica em computação gráfica, as curvas são a base, tanto da geração de formas simples, como círculos e elipses, quanto na criação de projetos complexos como automóveis, navios ou aeronaves (onde são referidas como formas livres).

Representar uma curva como uma sucessão de linhas retas pode ser suficiente para várias aplicações. No entanto, curvas e superfícies complexas demandam uma maneira mais eficiente de representação.

Definir uma curva que passe por um conjunto determinado de pontos é (matematicamente) um problema distinto da procura da melhor curva para representar um conjunto determinado de pontos. Usaremos o termo geração de curva para ambos os casos pois, freqüentemente, os dois problemas acontecem de forma combinada.

3.1. REPRESENTAÇÃO DE CURVAS

3.1.1. Conjunto de Pontos

Segmentos de curva ou de retas, mesmo que pequenos, têm sempre (geometricamente) infinitos pontos. Representa-se uma curva por um conjunto finito de pontos geralmente quando estes são medidos experimentalmente, por exemplo: $\{(-2,8),(-1,3),(0,0),(1,-1),(2,0),(3,3),(4,8)\}$. Nessa representação, cada par (x,y) representa um ponto no plano, e o conjunto inteiro a curva. Essa representação de pontos em um sistema de coordenadas foi proposta inicialmente em 1637 por René Descartes (1596-1650). Os grupos (x,y) são chamados de coordenadas cartesianas do ponto e representam as distâncias do ponto até as duas linhas perpendiculares aos eixos (Figura 3.1). Essa forma de representação é muito utilizada na visualização de medições experimentais.

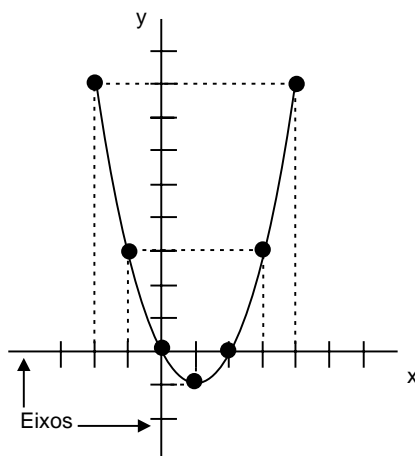


FIGURA 3.1. Pontos na coordenada cartesiana.

Na representação por conjunto de pontos, a curva pode ser gerada pelo uso de um número grande de pontos ou pela conexão deles por segmentos adequados. Se o número de pontos for pequeno em uma determinada resolução, ter-se idéia de curva contínua, é possível então (para melhorar a representação) uni-los por segmentos de retas como mostrado na Figura 3.2A. Para representação de curvas acentuadas, a aproximação por segmentos de reta pode não ser satisfatória (Figura 3.2A). Nesse caso, temos de aumentar o número de pontos nessa região (Figura 3.2B) obtendo mais pontos ou, se isso não for possível, gerando mais pontos por interpolação ou aproximação.

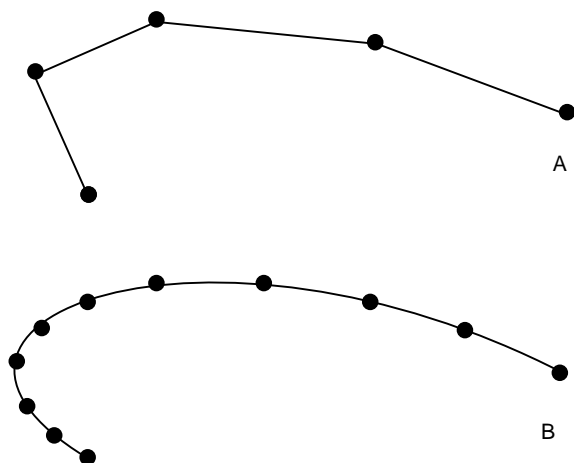


FIGURA 3.2. Representações por conjunto de pontos.

3.1.2. Representação Analítica

A representação analítica utiliza uma ou mais equações e apresenta várias vantagens em relação à representação por conjunto de pontos. É mais precisa, compacta, não requer área de armazenamento e facilita o cálculo de novos pontos (se isso for necessário devido à mudança de escala ou para melhor representação). Os pontos adicionais, a serem incluídos, são sempre exatos e não aproximações. Nessa forma de representação, é mais simples calcular propriedades da curva como área, inclinação, curvatura e outras do que na representação por pontos. Se forem conhecidos apenas os pontos da curva, e não sua expressão analítica, essas propriedades precisam ser calculadas por métodos numéricos. A representação analítica também apresenta mais simplicidade para ser redesenhada quando sujeita a transformações como mudança de escala, rotação, projeções e outras.

As formas analíticas ou por equações, de representar curvas, podem usar ou não parâmetros, sendo então denominadas paramétricas ou não-paramétricas. As for-

mas não-paramétricas podem ser ainda explícitas ou implícitas. A seguir, veremos em detalhes cada uma dessas formas.

3.1.3. Formas Não-paramétricas de Representar Curvas

Na forma não-paramétrica, como diz o nome, não são usados parâmetros e y é dado como uma função de x e vice-versa.

$$y = f_x(x) \text{ ou } x = f_y(y)$$

Nesta representação, a equação de um quarto de círculo de raio 10 é dada por:

$$y = \sqrt{10^2 - x^2} \text{ ou } x = \sqrt{10^2 - y^2}$$

e a equação de uma reta fica:

$$y = 2x - 1 \text{ ou } x = \frac{1}{2}(y+1)$$

para a representação de curvas 3D, basta acrescentar a coordenada $z(y)$ ou $z(x)$.

A forma não-paramétrica explícita de representar uma curva é dada por uma equação do tipo $y = f(x)$, onde os pares $(x, y = f(x))$ representam os pontos da curva. Por exemplo, a equação genérica explícita de uma parábola é dada por:

$$y = ax^2 + bx + c$$

sendo a , b e c números quaisquer. Com $a=1$, $b=-2$ e $c=0$, essa equação pode gerar a curva da Figura 3.1.

Nessa representação, chama-se ordem ou grau da curva, a potência de maior valor a que x for elevado. Os polinômios são a classe de funções matemáticas muito adequadas a essa representação explícita de curvas. O grau do polinômio corresponde à ordem ou grau da curva. Por exemplo:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_2 x^2 + a_1 x^1 + a_0$$

onde n são inteiros positivos, e a_0, a_1, \dots, a_n são números reais.

Polinômios são geralmente usados para representar curvas, pois são muito fáceis de combinar, derivar, integrar ou avaliar seu valor em algum ponto.

Dessa forma não-paramétrica, obtém-se um valor de y para cada valor de x dado. Conseqüentemente, se a curva tiver valores múltiplos (muitos valores de y para cada x), como um círculo, não pode ser representada explicitamente. Essa limitação não existe no caso de representações implícitas que são dadas por expressões na forma:

$$f(x,y) = 0$$

Por exemplo, a equação implícita de uma curva do segundo grau (ou ordem 2) genérica é:

$$ax^2 + bxy + cy^2 + dx + ey + f = 0$$

Essa expressão representa a variedade de curvas planas denominadas seções cônicas, muito importantes na Geometria Euclidiana. Essas curvas, no total cinco, são chamadas seções cônicas porque são obtidas pelo corte de um cone por um plano, resultando em um círculo, uma elipse, uma parábola, uma hipérbole ou uma reta (figura 3.3). Cada um desses tipos surge dependendo da direção em que o plano está em relação ao cone. Na equação, esses tipos de curvas surgem atribuindo-se diferentes valores às constantes “a”, “b”, “c”, “d”, “e” e “f”. O círculo e a elipse tem comprimentos finitos enquanto as demais podem se estender infinitamente. Os gregos já o utilizavam e ainda são muito usados em diversas aplicações como na astrologia para estudar o movimento dos corpos celestes (planetas, satélites, cometas etc.).

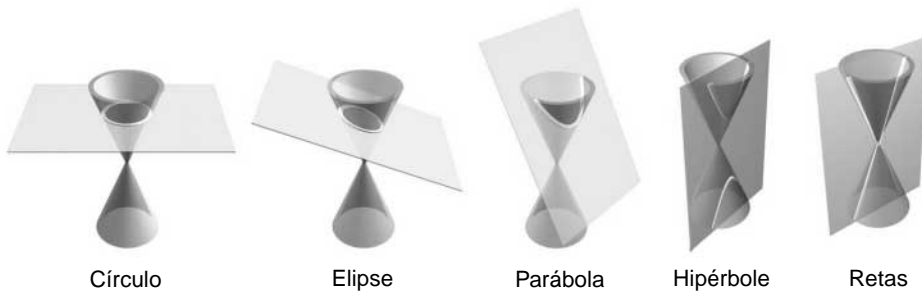


FIGURA 3.3. *Os cinco tipos de seções cônicas.*

A representação de uma curva 3D, na forma implícita, é descrita pela interseção de duas superfícies na forma:

$$\begin{cases} F_1(x,y,z) = 0 \\ F_2(x,y,z) = 0 \end{cases}$$

que devem ser resolvidas simultaneamente para a determinação dos pontos da curva. Esse processo é lento e, por isso, a forma implícita geralmente só é usada para a representação 2D. A grande vantagem da forma implícita é que com ela é muito fácil verificar se um dado ponto pertence ou não a curva, o que é útil para muitas aplicações de modelagem geométrica, onde precisa-se conhecer pontos interiores, exteriores e na fronteira dos objetos.

Chama-se implícitação a conversão de uma dada curva para essa forma. Por exemplo, a forma implícita da curva que descreve um círculo de raio r é:

$$x^2 + y^2 - r^2 = 0$$

e da mesma reta anterior (no início desta seção) é:

$$2x - y - 1 = 0$$

As formas explícita e implícita são dependentes do sistema de coordenadas, cuja escolha afeta a facilidade de uso. Se os pontos de uma curva forem calculados a partir de incrementos uniformes em x ou y podem não ficar distribuídos uniformemente ao longo da curva. Isso afeta a qualidade e a precisão da representação da curva. Essas limitações são superadas pelo uso de representações paramétricas.

3.1.4. Formas Paramétricas de Representar Curvas

Na forma paramétrica, usa-se um parâmetro (t , θ etc.) para definir as coordenadas dos pontos da curva. Por exemplo: a equação de um quarto de círculo de raio $r=10$ pode ser descrita como:

$$\begin{aligned}x &= 10 \cos \theta = f_x(\theta) \\y &= 10 \sin \theta = f_y(\theta)\end{aligned}$$

e a equação da reta da seção anterior pode ser representada por:

$$\begin{aligned}x &= t+1 = f_x(t) \\y &= 2t+1 = f_y(t)\end{aligned}$$

Os parâmetros usados nessa representação podem ser quaisquer símbolos.

Na forma paramétrica, cada coordenada de um ponto em uma curva é representada como uma função de um único parâmetro, sendo que a posição de um ponto na curva é fixada pelo valor do parâmetro. Para uma curva 2D que usa t como parâmetro, as coordenadas cartesianas de cada ponto são funções desse parâmetro t .

$$\begin{aligned}x &= x(t) \\y &= y(t)\end{aligned}$$

A posição do ponto na curva é, portanto, dada por:

$$P(t) = (x(t), y(t))$$

Nota-se, então, que é mais simples ter uma representação em intervalos de comprimentos constantes ao longo da curva. É também muito mais fácil o cálculo de diversas características úteis como, por exemplo, a derivada em relação ao parâmetro t , ou o vetor tangente da curva

$$P'(t) = (x'(t), y'(t))$$

A inclinação da curva (ou sua diferencial) é dada por:

$$\frac{dy}{dx} = \frac{dy / dt}{dx / dt} = \frac{y'(t)}{x'(t)}$$

Fica também fácil calcular a área que a curva faz com os eixos x e y , que é obtida das integrais em x e y .

A forma paramétrica permite representar curvas fechadas e com valores múltiplos.

Uma vez que um ponto na curva é especificado por um único valor de parâmetro, a forma paramétrica é independente do sistema de coordenadas. Os extremos e o comprimento da curva são fixos pelo intervalo de variação do parâmetro, frequentemente normalizado para $0 \leq t \leq 1$, por conveniência (isto é, como valores mínimos 0 e máximos 1). Como essas curvas são independentes do sistema de coordenadas, elas são facilmente manipuladas usando as transformações geométricas.

Uma grande vantagem da representação paramétrica é que ela pode representar curvas espaciais com a mesma facilidade que representa curvas no plano. Passar de 2D para 3D, nesta representação, corresponde apenas a incluir mais uma coordenada pela função: $z = z(t)$. Sendo um ponto da curva espacial definido quando suas três coordenadas forem conhecidas:

$$P(t) = (x(t), y(t), z(t))$$

Mas, como cada forma de representação é mais apropriada a determinada característica, será muitas vezes necessário trocar as formas de representação. A forma não-paramétrica pode ser obtida eliminando-se o parâmetro para obter uma única equação em x e y . Determinar um ponto em uma curva, i.e., determinar o valor de y , dado x , é trivial no caso da representação não-paramétrica explícita. No caso da paramétrica, é necessário obter o valor do parâmetro t , a partir de x , e a seguir usar este valor para obter y . Para equações paramétricas mais complexas, uma técnica iterativa pode ser mais conveniente.

Alguns exemplos de descrição de cônicas nas formas paramétrica e não-paramétrica implícita são dados na tabela a seguir. Não usamos as formas explícitas, pois elas não são representáveis em todos os valores.

| Cônica | Forma Paramétrica | Forma Implícita |
|-----------|--|---|
| Elipse | $x = a \cos \theta$ $y = b \sin \theta$ | $\frac{x^2}{a^2} + \frac{y^2}{b^2} - 1 = 0$ |
| Parábola | $x = at^2, y = 2at$ | $y^2 - 4ax = 0$ |
| Hipérbole | $x = a \cosh \theta$ $y = b \sinh \theta$ | $\frac{x^2}{a^2} - \frac{y^2}{b^2} - 1 = 0$ |

Nessa tabela \cosh e \sinh representam o coseno e o seno hiperbólico do ângulo θ .

3.1.5. Curvas Paramétricas de Terceira Ordem

Algumas curvas não podem ser facilmente descritas por expressões analíticas em toda sua extensão. Nesses casos, usam-se geralmente descrições pela união de diversas curvas. Para que diversas curvas sejam adequadamente transformadas em uma curva única, muitas vezes será necessário que a curva resultante da união tenha curvatura contínua. Para que isso ocorra, cada um dos pedaços deve também ter continuidade até a segunda derivada, é por essa razão que geralmente são usadas curvas representadas por polinômios cúbicos.

Conhecidas pela grande maioria dos usuários da computação gráfica, as curvas paramétricas de terceira ordem de Hermite, Bézier e Splines, são geradas por um polinômio cúbico e pela definição de um conjunto determinado de pontos de controle.

3.1.6. Hermite

O uso de polinômios de terceira ordem para ajuste de curvas foi extensamente descrito pelo matemático francês Charles Hermite (1822-1901). Hermite é também conhecido pelas várias entidades matemáticas que levam seu nome: polinômios de Hermite, equações diferenciais de Hermite, fórmulas de interpolação de Hermite e matrizes Hermitianas. Hermite construiu também a teoria geral das funções elípticas a partir dos trabalhos de Cauchy.

A formulação de Hermite é básica para o entendimento dos demais polinômios de ajuste de curvas. Para gerar uma curva de Hermite, você precisa de quatro fatores. Entre os quatro, dois são pontos que chamaremos de P1 e P2. Eles descrevem os pontos inicial e final da curva. Os outros dois são os vetores T1 e T2 que descrevem as tangentes e seus pesos na curva em P1 e P2, ou seja, T1 indica como a curva deixa o ponto P1, e T2 como encontra o ponto P2 (Figura 3.4). Esses quatro fatores de controle têm participação na composição da geometria da curva de Hermite, conforme mostrado na Figura 3.5.

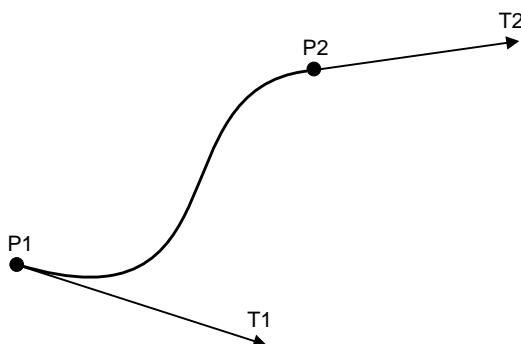


FIGURA 3.4. Elementos da curva de Hermite.

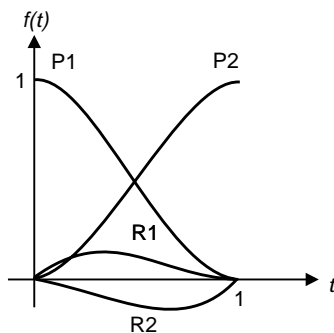


FIGURA 3.5. Composição da geometria da curva de Hermite.

É interessante lembrar de que os vetores têm quatro propriedades básicas: módulo, direção, sentido e ponto de aplicação. Quando Hermite usou esse tipo de controle, para a forma de chegada e saída da curva, ele ampliou o controle em relação ao uso somente das tangentes, pois assim teria no máximo a direção e o sentido. O módulo dos vetores funciona como um peso que muda completamente a curva.

Essa definição confere à curva uma grande versatilidade, permitindo, em um dado momento, uma forma suave e homogênea, e em outro, formas mais bruscas, podendo apresentar ruptura e formar “loops”. Além disso, quando os pontos e vetores estão sobre a mesma reta, ela assume a forma de uma linha reta.

Em um dado instante (t) qualquer, as coordenadas dos pontos da curva de Hermite resultam dos fatores de controle ($P1$, $P2$, $T1$, $T2$) ponderados. As curvas de ponderação da geometria de Hermite são definidas por polinômios de terceira ordem. A forma geral desses são:

$$\begin{aligned}x(t) &= P_x = a_x t^3 + b_x t^2 + c_x t + d_x \\y(t) &= P_y = a_y t^3 + b_y t^2 + c_y t + d_y \\z(t) &= P_z = a_z t^3 + b_z t^2 + c_z t + d_z\end{aligned}$$

Definir uma curva de Hermite consiste na determinação dos valores de (a , b , c , d) para os valores ($P1$, $P2$, $T1$, $T2$) dados. Para ($t = 0$), devemos ter as coordenadas do ponto inicial, ou seja,

$$(P1_x, P1_y, P1_z) = (x(0), y(0), z(0))$$

$$\begin{aligned}P1_x &= a_x 0^3 + b_x 0^2 + c_x 0 + d_x \\P1_y &= a_y 0^3 + b_y 0^2 + c_y 0 + d_y \\P1_z &= a_z 0^3 + b_z 0^2 + c_z 0 + d_z\end{aligned}$$

ou seja, as coordenadas do ponto inicial definem o parâmetro d em termos de suas coordenadas (d_x , d_y e d_z) no polinômio de Hermite.

Antes de prosseguirmos encontrando os demais parâmetros, repare que a equação do 3º grau que descreve o polinômio pode ser descrita de forma mais elegante através da multiplicação de matrizes:

$$x(t) = [t^3 \ t^2 \ t^1 \ 1] \begin{bmatrix} a_x \\ b_x \\ c_x \\ d_x \end{bmatrix} = T \ C_x$$

$$y(t) = [t^3 \ t^2 \ t^1 \ 1] \begin{bmatrix} a_y \\ b_y \\ c_y \\ d_y \end{bmatrix} = T \ C_y$$

$$z(t) = [t^3 \ t^2 \ t^1 \ 1] \begin{bmatrix} a_z \\ b_z \\ c_z \\ d_z \end{bmatrix} = T \ C_z$$

A primeira matriz descreve as potências do parâmetro t , e a segunda considera os coeficientes da curva de Hermite que se deseja definir. Como essas matrizes só têm uma linha e uma coluna são também chamadas de matrizes ou vetores linha e coluna, respectivamente.

Usando essa notação matricial, poderemos reescrever as expressões anteriores, que descrevem como achar o coeficiente “ d ” da curva de Hermite:

$$P1_x = [0 \ 0 \ 0 \ 1] \ C_x$$

$$P1_y = [0 \ 0 \ 0 \ 1] \ C_y$$

$$P1_z = [0 \ 0 \ 0 \ 1] \ C_z$$

Como a curva deve acabar no ponto $P2$, ou seja, o polinômio no valor final de t deve satisfazer as coordenadas desse ponto e o valor final do parâmetro t pode ser qualquer um, usa-se normalizar a curva, ou seja, supor que este será 1. Assim para $t = 1$, a equação da curva $x(t)$ deve resultar na coordenada x de $P2$:

$$x(1) = a_x + b_x + c_x + d_x$$

ou forma matricial:

$$x(1) = [1 \ 1 \ 1 \ 1] \ C_x$$

o mesmo ocorrendo para as demais coordenadas

$$\begin{aligned}y(1) &= [1 \ 1 \ 1 \ 1] C_y \\z(1) &= [1 \ 1 \ 1 \ 1] C_z\end{aligned}$$

Devemos agora satisfazer as condições vetoriais da curva. Como a direção do vetor está ligada à tangente da curva, usaremos:

$$\begin{aligned}x'(t) &= P'_x = 3a_x t^2 + 2b_x t + c_x \\y'(t) &= P'_y = 3a_y t^2 + 2b_y t + c_y \\z'(t) &= P'_z = 3a_z t^2 + 2b_z t + c_z\end{aligned}$$

que na forma matricial podem ser escritas como:

$$\begin{aligned}x'(t) &= P'_x = [3t^2 \ 2t \ 1 \ 0] C_x \\y'(t) &= P'_y = [3t^2 \ 2t \ 1 \ 0] C_y \\z'(t) &= P'_z = [3t^2 \ 2t \ 1 \ 0] C_z\end{aligned}$$

assim, a condição de a curva em $t = 0$ ser o vetor T1 resulta na definição dos parâmetros c_x , c_y e c_z da curva. Na forma matricial, podemos escrever:

$$\begin{aligned}x'(0) &= T1_x = [0 \ 0 \ 1 \ 0] C_x = c_x \\y'(0) &= T1_y = [0 \ 0 \ 1 \ 0] C_y = c_y \\z'(0) &= T1_z = [0 \ 0 \ 1 \ 0] C_z = c_z\end{aligned}$$

A última condição é que em $t = 1$ o vetor T2 defina a tangente da curva. Essa condição se expressa matricialmente como:

$$\begin{aligned}x'(1) &= T2_x = [3 \ 2 \ 1 \ 0] C_x \\y'(1) &= T2_y = [3 \ 2 \ 1 \ 0] C_y \\z'(1) &= T2_z = [3 \ 2 \ 1 \ 0] C_z\end{aligned}$$

Unindo as quatro condições P1, P2, T1 e T2, podemos unificar as expressões independentemente de representarem os eixos x , y ou z :

$$\begin{bmatrix} P1 \\ P2 \\ T1 \\ T2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} C_x = H^{-1} C_x$$

Como os valores de C_x , C_y e C_z são desconhecidos, podemos representar melhor a expressão anterior como:

$$C_x = HH^{-1}C_x = H \begin{bmatrix} P1 \\ P2 \\ T1 \\ T2 \end{bmatrix}_x$$

onde H é a função que multiplicada por H^{-1} produz a matriz identidade. É fácil verificar que essa matriz é:

$$H = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

como essa matriz independe da direção x, y ou z, temos:

$$x(t) = TC_x = TH \begin{bmatrix} P1 \\ P2 \\ T1 \\ T2 \end{bmatrix}_x = THG_x$$

$$y(t) = TC_y = TH \begin{bmatrix} P1 \\ P2 \\ T1 \\ T2 \end{bmatrix}_y = THG_y$$

$$z(t) = TC_z = TH \begin{bmatrix} P1 \\ P2 \\ T1 \\ T2 \end{bmatrix}_z = THG_z$$

As condições geométricas que definem uma dada curva de Hermite são frequentemente denominadas de vetores G_x , G_y e G_z , ou em uma forma única como matriz G_h :

$$G_h = \begin{bmatrix} P1_x & P1_y & P1_z \\ P2_x & P2_y & P2_z \\ T1_x & T1_y & T1_z \\ T2_x & T2_y & T2_z \end{bmatrix}$$

Assim, dada uma condição geométrica G_h , a curva de Hermite definida por ela fica perfeitamente definida pela expressão:

$$P(t) = T H G_h$$

Como T e H são constantes na representação de Hermite, a forma mais simples de representar essas curvas será através das chamadas funções interpolantes de Hermite:

$$P(t) = \begin{pmatrix} (2t^3 - 3t^2 + 1), (-2t^3 + 3t^2), (t^3 - 2t^2 + t), (t^3 - t^2) \end{pmatrix} \begin{bmatrix} P1 \\ P2 \\ T1 \\ T2 \end{bmatrix}$$

Estas funções produzem uma curva que é o resultado da combinação (blending function) das quatro propriedades geométricas mostradas na Figura 3.5.

É interessante lembrar que essas funções interpoladoras são completamente diferentes dos polinômios de Hermite, que solucionam a equação diferencial de Hermite:

$$y'' - 2xy' + 2ny = 0, n = 0, 1, 2, \dots$$

cuja solução é generalizada pela fórmula de Rodrigue:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} (e^{-x^2})$$

A curva de Hermite é a que possibilita o maior controle entre as demais de terceiro grau usadas em computação gráfica. A direção das retas tangentes utilizadas na geração da curva de Hermite permite introduzir modificações significativas na curva gerada. A Figura 3.6 ilustra os diferentes resultados obtidos pela simples alteração na direção da tangente inicial da curva de Hermite.

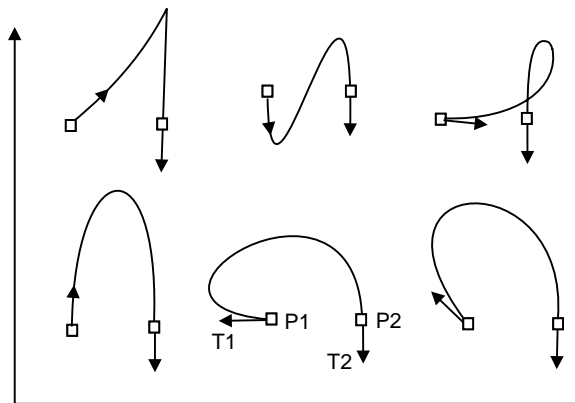


FIGURA 3.6. Diferentes resultados obtidos pela alteração na direção da tangente inicial da curva de Hermite.

3.1.7. Bézier

A curva de Bézier foi desenvolvida por Pierre Bézier durante seus trabalhos em projetos de automóveis para a Renault francesa no início da década de 1960. Bézier baseou sua curva nos princípios descritos por Hermite, com a diferença básica que para a determinação das tangentes em P1 e P2 utiliza pontos e não vetores. A grande maioria dos softwares de computação gráfica, disponíveis no mercado, utiliza o conceito da curva de Bézier. Entre eles, encontramos o Adobe Illustrator, O Corel Draw, o Auto CAD, o Paint Shop Pro, 3D MAX.

É importante notar que essa formulação matemática foi desenvolvida na mesma época e independentemente por Casteljau da Citroën, mas como os trabalhos de Bézier foram publicados antes, seu nome ficou mais conhecido. Essa curva, por não usar o conceito de vetor e apenas pontos, é mais facilmente entendida pelos usuários em geral.

Para ajuste por um polinômio de grau n , a curva de Bézier pode ser gerada por 3, 4, até $n + 1$ pontos de controle. Geralmente, em computação gráfica se utiliza a curva de Bézier em sua forma cúbica, necessitando então de quatro pontos de controle. A curva de Bézier cúbica passa pelo primeiro e pelo último ponto de controle e utiliza os outros dois para construir sua tangente (Figura 3.7).

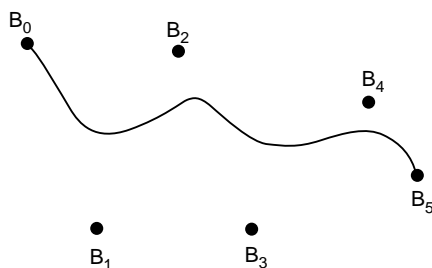


FIGURA 3.7. Pontos de controle da curva de Bézier.

A curva paramétrica de Bézier é definida como:

$$P(t) = \sum_{i=0}^n B_i J_{n,i}(t), \quad 0 \leq t \leq 1$$

Onde B_i representa cada um dos $n+1$ pontos de controle considerados, e $J_{n,i}(t)$ são as funções que combinam a influência de todos os pontos (blending functions) [Faux, 79]. Essas funções são descritas pelos polinômios de Bernstein como:

$$J_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-1}$$

onde n é o grau dos polinômios e

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}$$

sendo $i = 0, 1, \dots, n$ são os coeficientes binomiais. Essas funções $J_{n,i}(t)$ devem satisfazer as condições: $J_{n,i}(t) \geq 0$ para todo i entre 0 e 1, isto é $0 \leq t \leq 1$ e também:

$$\sum_{i=0}^n J_{n,i}(t) = 1, \quad 0 \leq t \leq 1$$

Essa última condição é chamada “propriedade normalizante” e força a curva gerada a ficar inteiramente dentro da figura convexa (convex hull) definida pelos pontos extremos e de controle (Figuras 3.8 e 3.9).

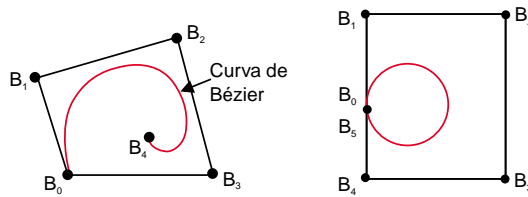


FIGURA 3.8. A curva de Bézier é sempre interior ao polígono convexo definido pelos extremos de seus pontos de controle. Curva de Bézier fechada pode ser obtida pela coincidência dos pontos de controle inicial e final.

FIGURA 3.9. Funções de combinação ou Blending functions de Bézier cúbicas ($n = 3$)

Os polinômios de Bernstein, usados como funções de combinação (blending function) nas curvas de Bézier, aproximam os pontos de controle por um único polinômio. O grau da forma final resultante depende do número de pontos de controle usados. Movendo-se a posição de um só ponto, toda a forma da curva se modifica. Uma curva com essa característica é dita ter **controle global**. Essa característica pode ser muito negativa quando um usuário deseja fazer ajustes finos na forma final do desenho. Para dar mais flexibilidade ao usuário, é necessário aumentar bastante o número de pontos de controle (Figura 3.10).

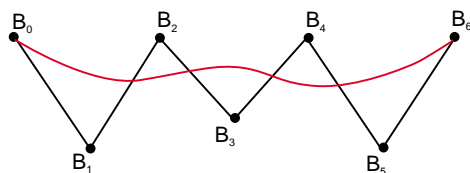


FIGURA 3.10. Adicionando pontos de controle a curva de Bézier.

Quando muitos pontos de controle são usados, as expressões podem se tornar complexas, pois resultarão em polinômios de graus maiores. Uma alternativa simples, quando forem necessários muitos pontos de controle, é a conexão de vários segmentos de curvas de graus menores. Nesse caso, para que as duas curvas tenham a mesma inclinação no ponto de união, ou seja, derivadas contínuas, deve-se fazer com que tenham três pontos em linha reta. Isto é, o ponto imediatamente antes do final de uma das curvas deve estar sobre a mesma linha reta dos pontos de controle (coincidentes) e do ponto imediatamente depois do início da curva seguinte (Figura 3.11).

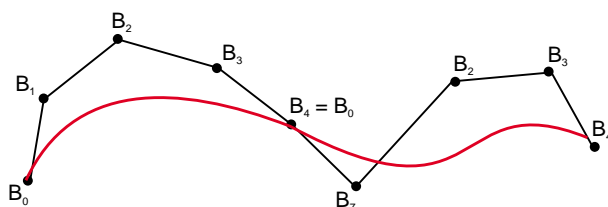


FIGURA 3.11. Conexão de vários segmentos de curvas de graus menores para simplificação da expressão.

Em geral, uma forma complexa é mais facilmente modelada por várias curvas que são conectadas em seus pontos extremos. Ao criar as junções, o projetista, em geral, deseja controlar a continuidade nos pontos de junção. Continuidade de ordem 0 significa que as duas curvas se encontram; continuidade de primeira ordem exige que as curvas tenham tangentes comuns no ponto de junção, e continuidade de segunda ordem exige que as curvaturas sejam as mesmas. Para representar essas continuidades usa-se a notação C^0 , C^1 , C^2 etc. Essa simbologia usa a letra C maiúscula com um número superescrito. A forma mais simples de continuidade C^0 assegura que uma curva ou a união de curvas não terá descontinuidade. O nível seguinte de continuidade C^1 indica que a inclinação ou sua derivada primeira da curva é constante em todos os pontos. A continuidade C^2 implica em continuidade na derivada segunda da curva e assim por diante (Figura 3.12).

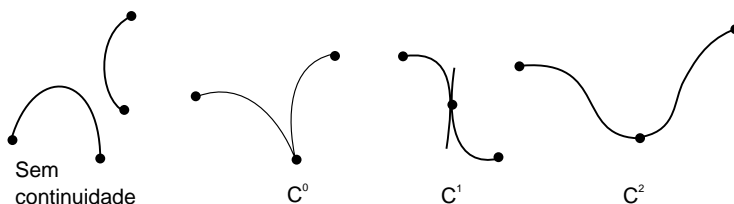


FIGURA 3.12. Níveis de continuidade na união de duas curvas.

Como exemplo do uso das expressões que descrevem as curvas de Bézier, vamos considerar o caso de três pontos de controle: B_0 , B_1 e B_2 . Nesse caso, os polinômios terão grau $n = 2$. Expandindo as expressões anteriores, teremos:

$$P(t) = B_0 J_{2,0}(t) + B_1 J_{2,1}(t) + B_2 J_{2,2}(t)$$

As três funções de combinação serão:

$$J_{2,0} = \frac{2!}{0!2!} t^0 (1-t)^2 = (1-t)^2 = 1 - 2t + t^2$$

$$J_{2,1} = \frac{2!}{1!1!} t^1 (1-t)^1 = 2t(1-t) = 2t - 2t^2$$

$$J_{2,2} = \frac{2!}{2!0!} t^2 (1-t)^0 = t^2$$

Substituindo esses valores na equação anterior teremos:

$$P(t) = (1-t)^2 B_0 + 2t(1-t) B_1 + t^2 B_2$$

Na forma matricial podemos escrever:

$$P(t) = \begin{bmatrix} (1-t)^2 & 2t(1-t) & t^2 \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \\ B_2 \end{bmatrix}$$

ou, ainda, separando o parâmetro t em uma matriz linha de potências:

$$P(t) = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \\ B_2 \end{bmatrix}$$

de uma maneira mais compacta podemos escrever:

$$P(t) = T \cdot M_B \cdot G_B$$

onde, de forma semelhante à curva de Hermite, o vetor T representa as potências do parâmetro, M_B são os coeficientes da matriz de Bézier, e G_B representa as condições geométricas. Um desenvolvimento semelhante para quatro pontos de controle levaria à expressão:

$$P(t) = (1-t)^3 B_0 + 3t(1-t)^2 B_1 + 3t^2(1-t) B_2 + t^3 B_3$$

ou na forma matricial:

$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{bmatrix}$$

As matrizes específicas para valores pequenos de n ($n = 3, 4$) são de particular interesse. Para qualquer valor de n , a matriz $[M_B]$ é simétrica em relação à diagonal principal e o canto triangular inferior direito contém apenas zeros.

A formulação de Bézier apresenta, além das já comentadas, outras propriedades interessantes. As funções base são reais. A forma da curva geralmente acompanha a forma do polígono de definição (na verdade é uma versão “suavizada” da forma do polígono). Assim, para desenhar uma curva, basta definir o polígono e depois ajustar os pontos que forem necessários para aproximar melhor a forma desejada. Isso torna a formulação adequada para um sistema interativo. Um projetista experiente consegue obter a forma desejada depois de dois ou três interações com o sistema usado.

Em situações práticas, em geral é desejável ter controle direto sobre os pontos extremos da curva. O primeiro e o último pontos da curva gerada coincidem com o primeiro e o último pontos do polígono de definição. Os vetores tangentes nos extremos da curva têm a mesma direção que o primeiro e o último segmentos do polígono de definição, respectivamente.

A curva está contida no fecho convexo do polígono. Uma consequência simples desse fato é que um polígono plano sempre gera uma curva plana. A curva exibe a propriedade da variação decrescente. Isso significa, basicamente, que a curva não oscila em relação a qualquer linha reta com mais frequência que o seu polígono de definição. Algumas representações matemáticas têm a tendência de amplificar, em vez de suavizar, quaisquer irregularidades de formato esboçadas pelos pontos de definição, enquanto outras, como as curvas de Bézier, sempre suavizam os pontos de controle. Assim, a curva nunca cruza uma linha reta arbitrária mais vezes que a sequência de segmentos que conectam os pontos de controle.

A curva de Bézier é invariante sob transformações afins. Transformações afins estão disponíveis em qualquer sistema de CAD, pois é essencial reposicionar, escalar ou girar os objetos. Essa propriedade garante que os dois procedimentos a seguir produzem os mesmos resultados: a) primeiro calcula-se um ponto na curva, e depois aplica-se uma transformação afim; e b) primeiro, aplica-se uma transformação afim ao polígono de definição, e depois gera-se a curva.

3.1.8. Splines

Existem diversas formas de introduzir a teoria dessas curvas e muito já se tem escrito sobre elas desde o seu desenvolvimento em 1967 por Schoenberg.

O nome Spline faz alusão ao termo da língua inglesa utilizado para denominar a régua flexível usada em desenho para gerar curvas livres suaves, de classe C^2 , isto é, com curvaturas contínuas. A expressão matemática que descreve essa régua é denominada Spline Cúbica Natural. Nessa expressão, as alterações em qualquer um dos pontos de controle provocam alterações em toda a curva. Esse comportamento, como já mencionado nas seções anteriores não é apropriado para aplicações de ajuste de curvas interativas.

A B-Spline é uma “versão” da Spline Natural, com controle local, isto é, as alterações nos pontos de controle da B-Spline apenas se propagam para os vizinhos mais próximos. A função B-Spline não passa pelos pontos de controle (Figura 3.13). Outra característica básica é que ela pode ser gerada para qualquer número de pontos de controle e grau de polinômio, ou seja, o grau do polinômio pode ser selecionado de maneira independente do número de pontos de controle. No entanto, é claro que o grau i de continuidade C^i depende da ordem dos polinômios usados nas funções de base.

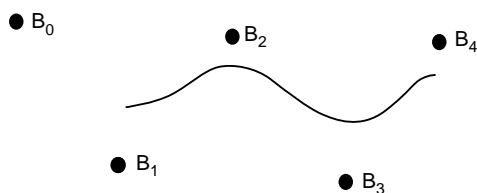


FIGURA 3.13. A função B-Spline não passa pelos pontos de controle.

Nas aplicações que usam curvas de forma livre para o projeto de modelos, curvatura contínua é geralmente um fator importante e por isso B-Splines cúbicas são preferencialmente usadas.

A forma geral da curva B-Spline é bastante semelhante a da curva de Bézier. Um conjunto de funções $N_{i,k}(t)$ combina o efeito dos pontos de controle B_i para gerar a curva:

$$P(t) = \sum_{i=0}^n B_i N_{i,k}(t)$$

As diferenças fundamentais entre ambas são as funções $N_{i,k}(t)$ ($i = 0, 1, \dots, n$) usadas. O parâmetro k controla a ordem de continuidade da curva, e n o número de pontos de controle usados. O parâmetro t também pode ter maior gama de variação do que nas curvas anteriores. Assim $N_{i,k}$ representa as funções de grau $(k-1)$ (ordem do polinômio) e curvas de continuidade C^{k-2} .

Cada uma das funções $N_{i,k}(t)$ é definida de maneira recursiva pelas equações:

$$N_{i,1}(t) = \begin{cases} 1 & \text{para } t_i \leq t \leq t_{i+1} \\ 0 & \text{nos demais intervalos} \end{cases}$$

$$N_{i,k}(t) = \left(\frac{t - t_i}{t_{i+k-1} - t_i} \right) N_{i,k-1}(t) + \left(\frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} \right) N_{i+1,k-1}(t)$$

Como o denominador pode se tornar zero, usa-se a convenção: $0/0 = 0$. Essa formulação requer a escolha de um conjunto de valores t_i chamados nós, que se relacionam ao parâmetro t . As únicas restrições impostas a esses nós são que:

- estejam em ordem não decrescente, ou seja, os valores dos elementos t_i devem satisfazer a relação $t_i \leq t_{i+1}$;
- um mesmo valor não deve aparecer mais que k vezes, ou seja, não pode surgir mais vezes que a ordem da Spline usada. Esses valores de nós idênticos são referidos como nós múltiplos, ou nós em multiplicidade. Os valores dos nós influenciam bastante as funções de combinação $N_{i,k}(t)$ e, portanto, a curva gerada.

Como as curvas de Bézier, as Splines satisfazem a propriedade de envoltória convexa (convex hull property) já comentada. Satisfazem também a propriedade normalizante já que:

$$\sum_{i=0}^n N_{i,k}(t) = 1$$

Em uma curva B-Spline, o número de pontos de controle $(n+1)$, o grau $(k-1)$ e o número de nós estão relacionados. Supondo que esses nós sejam $t_0, t_1, t_2, \dots, t_m$, essas características se relacionam pela expressão: $m = n + k$

Os nós são geralmente apresentados como vetores ou matrizes linhas $[t_0, t_1, t_2, \dots, t_{n+k}]$ e podem ser classificados como: “uniformes e periódicos”, “uniformes e não-periódicos” e “não-uniformes”

Como os nós influenciam toda a forma da curva B-Spline, usa-se geralmente essa mesma classificação para a curva representada por eles.

3.1.8.1. Splines Uniformes e Periódicas

Uma Spline é uniforme se o vetor de nós for uniforme. Um vetor de nós é dito ser uniforme quando é definido em intervalos iguais, isto é, $t_i - t_{i-1} = t_{i+1} - t_i = \Delta t$ para todos os intervalos. Por exemplo, $[-2 \ 0 \ 2 \ 4 \ 6]$ com $t = 2$. Em muitas aplicações práticas, a seqüência de nós começa no zero e é normalizada como, por exemplo, $[0 \ \frac{1}{2} \ \frac{3}{4} \ 1]$ com $\Delta t = \frac{1}{4}$. Muitas vezes cada segmento da B-Spline é localmente normalizado, como o caso do próximo desenvolvimento. (Seção 3.1.8.4).

Vetores de nós uniformes são também periódicos, ou seja, a função B-Spline se translada para cada segmento. A influência de cada função base é limitada a k intervalos. Por exemplo, se uma função cúbica for usada ela se expandirá em quatro intervalos na geração de uma B-Spline cúbica uniforme.

Uma curva B-Spline periódica não passa pelos pontos de controle inicial e final, a menos que seja de ordem 2, ou seja, uma reta. Na figura 3.14, tem-se exemplos de B-splines periódicas de grau 1, 2 e 3 (linear, cúbica e quadrática) calculadas com os mesmos pontos de controle.

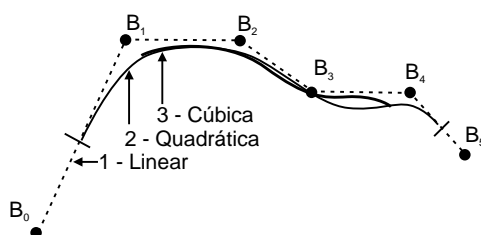


FIGURA 3.14. B-Splines uniformes de diversos graus com B_i s iguais.

Na forma linear ($k=2$), a curva coincide com os pontos de controle. Para $k=3$, a curva de grau dois (ou quadrática) começa no ponto médio da reta que une os dois primeiros pontos de controle e termina no ponto médio da reta que une os dois últimos pontos de controle. Com o aumento do grau, o parâmetro t diminui seus limites da variação, como pode ser observado claramente na Figura 3.14 pela redução do comprimento da curva gerada. Isso é representado pela expressão $(k-1) \leq t \leq n+1$.

3.1.8.2. Splines Não-periódicas

A Spline é não-periódica se o vetor de nós for não-periódico. Um vetor de nós é não-periódico se tem nós de valores repetidos nos extremos com multiplicidade igual à ordem k , e nós internos igualmente espaçados. Por exemplo, se o polígono de controle tiver $n=4$ pontos de controle (B_0, B_1, B_2 e B_3), a relação entre a ordem de continuidade da curva k e o número de nós $m=n+k$ faz com que tenha-se, como possíveis vetores de nós, as combinações:

| Ordem(k) | Nº de nós(m) | Vetor de nós não-periódicos |
|----------|--------------|-----------------------------|
| 2 | 6 | 0 0 1 2 3 3 |
| 3 | 7 | 0 0 0 1 2 2 2 |
| 4 | 8 | 0 0 0 0 1 1 1 1 |

A Figura 3.15 mostra essas curvas. As expressões $t_i = 0$ para $i < k$, $t_i = i - k + 1$ para $k \leq i \leq n$ e $t_i = n - k + 2$ para $i > n$ devem ser satisfeitas para um nó t_i em um vetor de nós não-periódicos que inicie em t_0 .

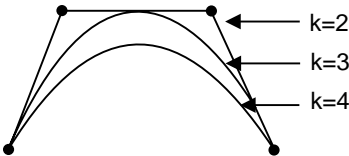


FIGURA 3.15. B-Splines não-periódicos de diversos graus.

Nesse caso, as curvas não diminuem seus limites como ocorria no caso anterior de vetores de nós periódicos, eles sempre iniciam no primeiro ponto e terminam no último.

3.1.8.3. Splines Não-uniformes

Os vetores de nós são ditos não-uniformes se forem não-periódicos e não tiverem nós múltiplos nas extremidades ou nós internos com mesmo espaçamento (ambas condições necessárias para a classificação de Splines periódicas). Por exemplo, os nós $[0.0 \ 0.2 \ 0.4 \ 0.5 \ 0.6 \ 0.8 \ 1.0]$ têm espaçamentos desiguais e são, por isso, não-uniformes, gerando B-Splines não-uniformes. Também a sequência $[0 \ 1 \ 2 \ 2 \ 3 \ 4]$ representa vetores de nós não-uniformes devido à multiplicidade de nós internos. Embora nós com espaçamento uniforme sejam mais simples, há vantagens no uso de espaçamento não-uniforme para controle mais preciso de formas. A Figura 3.16 mostra exemplos de duas B-Splines não-uniformes, ambas com $k=4$, mas com diferentes vetores de nós.

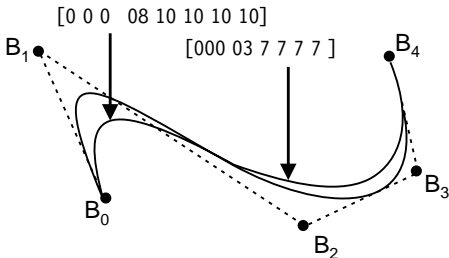


FIGURA 3.16. B-Spline cúbica não-uniforme.

3.1.8.4. Desenvolvimento da Formulação Genérica de B-Splines

A fórmula geral apresentada no início da descrição das curvas B-Splines será usada aqui para demonstrar o desenvolvimento de qualquer tipo desejado dessas curvas, segundo as classificações descritas nas seções anteriores.

Para gerar interpolações lineares, tem-se $k=2$, e a curva passa a ser descrita pelas funções:

$$N_{i,2}(t) = \begin{cases} \frac{(t-t_i)}{(t_{i+2}-t_i)} & \text{se } t_i \leq t \leq t_{i+1} \end{cases}$$

e

$$N_{i,2}(t) = \begin{cases} \frac{(t_{i+2}-t)}{(t_{i+2}-t_{i+1})} & \text{se } t_{i+1} \leq t \leq t_{i+2} \end{cases}$$

Dependendo do vetor de nós escolhido, é possível ter curvas uniformes e periódicas, não-periódicas ou não-uniformes. Se o desejado for uma B-Spline periódica definida a intervalos iguais de 1 a partir de 0(zero), teremos

$$N_{i,2}(t) = \begin{cases} t & \text{para } 0 \leq t \leq 1 \\ 2-t & \text{se } 1 \leq t \leq 2 \end{cases}$$

Para gerar interpolações quadráticas, tem-se $k=3$ e as funções são definidas recursivamente como:

$$N_{i,3}(t) = \frac{(t-t_i)}{(t_{i+2}-t_i)} N_{i,2}(t) + \frac{(t_{i+3}-t)}{(t_{i+3}-t_{i+1})} N_{i+1,2}(t)$$

onde o valor de $N_{i+1,2}(t)$ pode ser obtido da expressão anterior. Assim, se forem usados intervalos iguais de t a partir de zero para o vetor de nós, tem-se:

$$N_{i,4}(t) = \begin{cases} \frac{1}{2}t^2 & \text{se } 0 \leq t < 1 \\ \frac{3}{4} - \left(t - \frac{3}{2}\right)^2 & \text{se } 1 \leq t < 2 \\ \frac{1}{2}(3-t)^2 & \text{se } 2 \leq t \leq 3 \end{cases}$$

As funções de interpolação cúbica ($k=4$) para o mesmo conjunto de nós (periódicos e uniformes) serão:

$$N_{i,4}(t) = \begin{cases} \frac{1}{6}t & 0 \leq t \leq 1 \\ \frac{2}{3} - \frac{1}{2}(t-2)^3 - (t-2)^2 & 1 \leq t \leq 2 \\ \frac{2}{3} - \frac{1}{2}(t-2)^3 - (t-2)^2 & 2 \leq t \leq 3 \\ \frac{1}{6}(4-t)^3 & 3 \leq t \leq 4 \end{cases}$$

Do mesmo modo, é possível recursivamente gerar qualquer tipo de B-Spline, não-periódica ou não-uniforme apenas escolhendo adequadamente os vetores nós.

Assim como Hermite e Bézier, um ajustador B-Spline, utilizando a forma matricial com os parâmetros separados das matrizes B-Spline, descreve as curvas de mistura de sua geometria e os pontos de controle. Se essa forma for usada para a expressão da spline cúbica anterior teremos:

$$PC(t) = [t^3 \ t^2 \ t \ 1] M_s \begin{bmatrix} B_{i-1} \\ B_i \\ B_{i+1} \\ B_{i+2} \end{bmatrix}$$

no caso de t ser substituído por $(t+i)$ em cada intervalo, sendo ainda:

$$M_s = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

Os ajustadores de B-Spline abrigam também uma outra característica interessante. Para gerar uma curva convexa, com o mesmo grau de continuidade do restante dos segmentos, basta repetir os três primeiros pontos ao final da sequência de pontos de controle.

3.1.8.5. Catmull-Rom Splines

A Catmull-Rom Spline é uma interpolação local das curvas Spline desenvolvida para ser aplicada em computação gráfica. Seu uso inicial era no projeto de curvas e superfícies, e foi recentemente usada em várias outras aplicações. Uma característica importante da Catmull-Rom Spline é que a curva gerada passa através de todos os pontos de controle, o que não é comum para uma Spline.

Para calcular um ponto na curva, são necessários dois pontos, um antes e outro depois do ponto desejado, como os pontos P1 e P2 na Figura 3.17. A posição do ponto a ser interpolado é especificada pelo parâmetro t , que indica a posição do novo

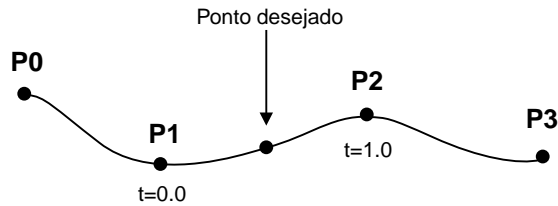


FIGURA 3.17. Elementos usados na interpolação local por uma Catmull-Rom Spline.

ponto em relação a sua distância dos outros dois. Quanto mais próximo de zero, mais próximo estará o novo ponto de P1. Quanto mais próximo de um, mais próximo o novo ponto estará de P2.

Dado os pontos de controle P_0 , P_1 , P_2 e P_3 , e o valor do parâmetro t , a localização do ponto pode ser calculada, assumindo um espaçamento uniforme entre os pontos de controle, como:

$$q(t) = \frac{1}{2} \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} * \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} * \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

A Spline de Catmull-Rom tem as seguintes características:

- a curva gerada passa por todos os pontos de controle;
- a curva gerada é contínua de classe 1, C^1 , ou seja, não existe descontinuidade nas tangentes (Figura 3.18). Além disso, o vetor tangente em um ponto P_i é paralelo a linha que une os pontos P_{i-1} e P_{i+1} ;
- a curva gerada não é contínua de classe 2, C^2 , isto é, a curvatura do segmento gerado não é constante. A segunda derivativa, em vez de ser contínua, apresenta uma curvatura que varia linearmente;
- os pontos no segmento quando podem estar fora da figura convexa definida pelos pontos de controle (convexhull).

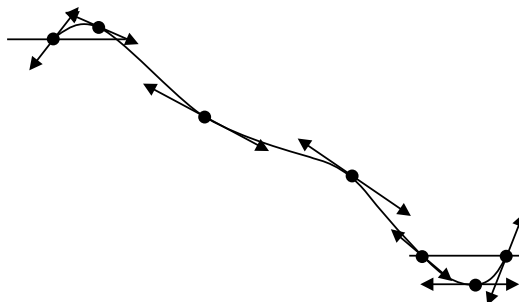


FIGURA 3.18. A Spline contínua sem descontinuidade na direção da tangente.

3.1.9. Curvas Racionais

Quando aprendemos o que eram os números reais, nos foi ensinado antes os conceitos de números racionais e irracionais. Como o conceito de números inteiros é bastante simples, o conceito de racionais veio a partir da idéia de fração ou divisão de dois inteiros. Assim, os números racionais são todos os que podem ser escritos como a razão de dois números inteiros. Como $0,1 = 1/10$ ou $0,98765 = 98765/100000$.

As curvas vistas até aqui foram descritas como polinômios. As curvas racionais (de maneira análoga aos números racionais) são descritas como a razão de dois polinômios. Essas curvas são importantes, pois têm a propriedade de serem invariantes a transformações de projeção.

Tanto as curvas de Bézier quanto as B-Splines possuem forma racional ou forma inteira. A forma inteira foi vista nas seções anteriores. A tabela seguinte mostra as expressões correspondentes na forma racional.

| | Forma Inteira | Forma Racional |
|----------|-------------------------------|---|
| Bézier | $\sum_{i=0}^n B_i J_{n,i}(t)$ | $\frac{\sum_{i=0}^n w_i B_i J_{n,i}(t)}{\sum_{i=0}^n w_i J_{n,i}(t)}$ |
| B-Spline | $\sum_{i=0}^n B_i N_{i,k}(t)$ | $\frac{\sum_{i=0}^n w_i B_i N_{i,k}(t)}{\sum_{i=0}^n w_i N_{i,k}(t)}$ |

Essas curvas sob projeção perspectiva continuam sendo racionais. Isso não ocorre com a forma inteira dessas curvas. O ponto básico da invariância à perspectiva das curvas racionais são as coordenadas homogêneas. Essas coordenadas representam um ponto no espaço 3D (x,y,z) como um elemento do espaço de quatro dimensões homogêneo (wx,wy,wz,w), sendo w um valor maior que zero. Esse valor w é a coordenada homogênea e também denominado peso (Capítulo 2, seção 2.6).

A representação de uma curva na forma racional começa com sua representação no espaço homogêneo. Vamos exemplificar considerando a formulação da curva de Bézier em coordenadas homogêneas.

$$P^w(t) = \sum_{i=0}^n B_i^w J_{n,i}(t) \quad 0 \leq t \leq 1$$

Nessa expressão, os valores $P^w(t)$ indicam os pontos da curva no espaço homogêneo 4D, ou seja, $(wx(t), wy(t), wz(t), w)$; B_i^w representam os pontos de controle no espaço homogêneo 4D, e $J_{n,i}(t)$ são as funções de Bézier padrão de combinação da influência dos pontos de controle.

Os pontos de controle no espaço 3D são obtidos pela projeção de B_i^w , ou seja, são obtidos pela divisão das três primeiras coordenadas pela coordenada homogênea w_i . A projeção dos pontos de controle B_i^w do espaço 4D para o espaço 3D é dada por:

$$B_i = B_i^w / w_i$$

Desse modo $B_i^w = w_i B_i$. De maneira análoga, os pontos gerados pela curva são projetados do espaço 4D para o 3D. O resultado é a curva de Bézier racional.

$$P(t) = P^w(t) = \frac{\sum_{i=0}^n w_i B_i J_{n,i}(t)}{\sum_{i=0}^n w_i J_{n,i}(t)}$$

Se todos os pesos w_i da expressão racional anterior forem iguais a 1, a expressão se reverte para a forma anterior inteira. Os pesos w_i , portanto, adicionam um novo grau de liberdade na curva a ser criada. Todas as propriedades das formas inteiras são também válidas e se aplicam às formas racionais. A única restrição é que a coordenada homogênea seja positiva, isto é $w_i \geq 0$. Por exemplo, se todos os pesos se mantiverem fixos e abaixo de um peso w_i , um aumento no valor desse peso empurrará a curva na direção do ponto de controle B_i . A Figura 3.19 mostra como a curva racional pode mudar a sua forma quando todos os parâmetros se mantiverem fixos abaixo do peso relacionado ao ponto fixo B_3 .

As curvas racionais têm se tornado muito populares nos diversos sistemas de CAD, tanto as Bézier quanto todas as formas de Splines (uniformes-periódicos, não-periódicos e não-uniformes). A forma mais comumente encontrada são as B-Splines não-uniformes racionais, comumente chamadas pela forma abreviada NURB, porque essa representação inclui todas as formas possíveis de Bézier e B-Splines.

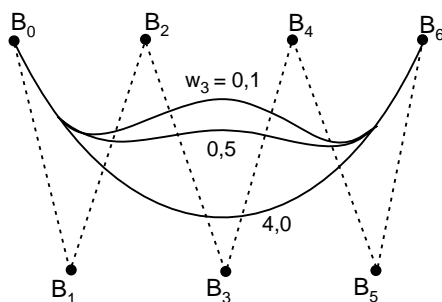


FIGURA 3.19. Variando um dos pesos de uma curva racional.

3.2. SUPERFÍCIES

As superfícies têm um papel muito importante na computação gráfica. De uma maneira geral, as superfícies são uma generalização das curvas.

Uma superfície (como uma curva) pode ser gerada por famílias de conjuntos de pontos; ter representação analítica; explícita ou implícita; paramétrica ou não-paramétrica. Podemos ainda interpolar, ajustar ou aproximar superfícies a partir de pontos. Essa forma de geração de objetos por seus contornos é muito importante na modelagem geométrica e de certa forma será revista no próximo capítulo.

A Figura 3.20 mostra algumas superfícies muito conhecidas e as equações que as geraram. Essas superfícies não estão na forma paramétrica, cada ponto sobre elas é uma função de suas coordenadas (x,y,z) .

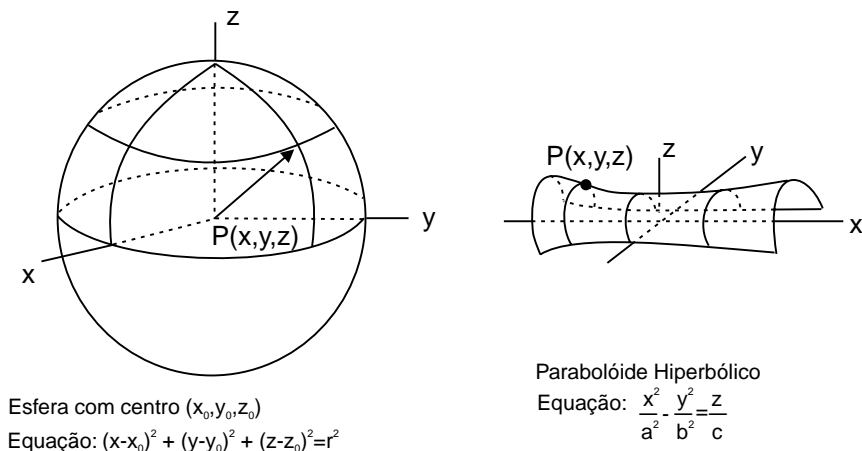


FIGURA 3.20. Superfícies não-paramétricas: os pontos são uma função de suas coordenadas.

3.2.1. Superfícies de Revolução

A rotação de uma curva plana em torno de um eixo produz a família mais conhecida de superfícies. Assim, um segmento de reta girando de 360° em torno do eixo z , como mostrado na Figura 3.21, produz uma superfície cônica. Curvas, ângulos e eixos de rotações diferentes produzem várias formas de superfícies de revolução.

Cada ponto da superfície de revolução da Figura 3.21 é uma função de dois parâmetros: o ângulo de rotação θ , e uma posição na curva t a ser rotacionada. Um ponto do segmento da curva no plano $x=0$ será representado como $[0, y(t), z(t)]$, e quando rotacionado de um ângulo θ em torno do eixo z , teremos $[y(t) \sin \theta, y(t) \cos \theta, z(t)]$.

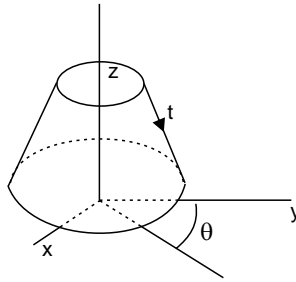


FIGURA 3.21. Um segmento de reta girando 360° em torno do eixo z produz uma superfície cônica.

De maneira geral tem-se que um ponto da superfície de revolução é descrito como $P(t, \theta)$. Superfícies de revolução podem ser obtidas por qualquer tipo de curva, mesmo as fechadas como elipses e círculos, ou as geradas por qualquer um dos métodos descritos nas seções anteriores (Cônica, Hermite, Bézier, Splines etc.).

3.2.2. Superfícies Geradas por Deslocamento

Translações e deslocamentos genéricos de curvas produzem diversas formas de superfícies. Essa forma de geração é denominada “sweeping” (varredura). A geração por rotação, vista na seção anterior, pode ser considerada um caso particular de sweeping, no qual o deslocamento é uma rotação. Mais precisamente, sweeping é o procedimento de gerar uma superfície através do movimento de uma curva ou figura plana ao longo de um caminho.

O movimento que a operação de sweeping fará pode ser descrito tanto por uma simples linha reta quanto por curvas complexas. Se um ponto da curva for descrito como $C(t)$ e um ponto do movimento que a curva fará for $M(s)$, um ponto de superfície gerada será:

$$P(t, s) = C(t) \times M(s)$$

onde \times indica o produto cartesiano das duas curvas $M(s)$ e $C(t)$. A Figura 3.22 mostra algumas superfícies geradas por sweeping.

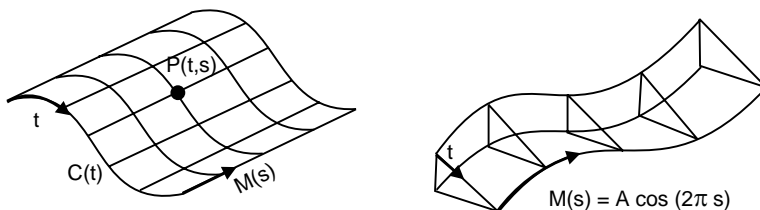


FIGURA 3.22. Superfícies geradas por sweeping.

3.2.3. Superfícies Geradas por Interpolação Bilinear

A geração de superfícies a partir da expressão da curva ou dos pontos que descrevem seus limites é uma das formas mais úteis, muito empregada nas construções navais, aeroespaciais e na análise numérica, onde há necessidade de discretização de domínios (elementos finitos ou de contorno).

Essas formas são paramétricas e começam pela definição da forma como os parâmetros representarão a superfície. A forma mais simples é considerar o espaço dos parâmetros representados por uma área unitária limitada pelos pontos (0,0);(0,1) (1,0) e (1,1). Essa área pode ser vista como o produto cartesiano dos dois eixos normalizados ortogonais, mostrados na Figura 3.23, de modo que qualquer ponto do interior seja definido univocamente. Se você deseja gerar uma superfície a partir de quatro pontos: A,B,C e D, deve associar esses pontos aos limites do espaço dos parâmetros (0,0), (0,1) (1,0) (1,1) e gerar o interior empregando duas interpolações lineares sucessivas. Na primeira interpolação, serão geradas as retas AD e BC, que correspondem aos limites com parâmetro $u=0$ e $u=1$. Qualquer ponto E sobre a reta AD será definido como:

$$E = (1 - v) A + v D,$$

de modo que se $v=0$, E é o próprio ponto A, e se $v=1$, E corresponde ao ponto D (Figura 3.23). Do mesmo modo, os pontos F sobre a reta BC serão obtidos pela interpolação linear das coordenadas de B e C:

$$F = (1 - v) B + v C.$$

Com os pontos E e F é possível gerar o interior da superfície a partir de outra interpolação linear, usando agora o parâmetro u :

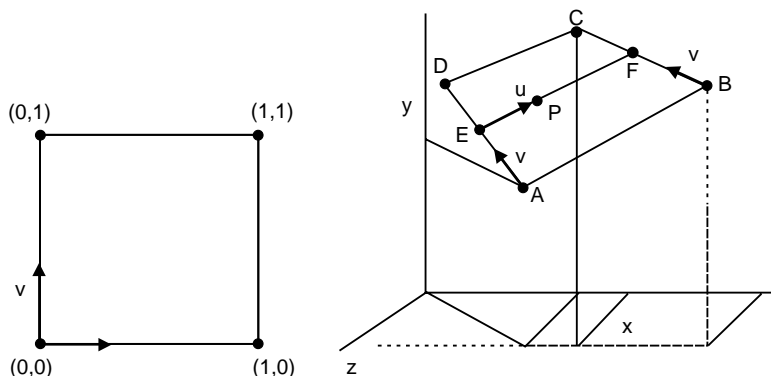


FIGURA 3.23. Parâmetros u , v e geração de superfícies por quatro pontos limites.

$$P(u,v) = (1 - u) E + u F$$

As expressões anteriores podem ser reunidas, resultando em duas interpolações lineares ou uma interpolação bilinear:

$$P(u,v) = (1 - u) (1 - v) A + (1 - u) v D + u (1 - v) B + u v C$$

É fácil verificar que, se a superfície gerada é um plano então: os quatro pontos devem ser coplanares (estarem sobre um mesmo plano) e as fronteiras devem estar sempre limitadas por segmentos de reta.

Se as fronteiras (ou limites) forem definidas por curvas, e não por retas, a forma mais simples de gerar o interior é usar a expressão dessas curvas para gerar os pontos E e F. Essa é a idéia de geração de superfícies denominada *lofting*, usada desde a antiguidade na construção de caravelas, naus, embarcações e navios. Nela, as curvas dos limites opostos, nas direções u ou v dos parâmetros, são usadas para a geração da superfície. A Figura 3.24 mostra os chamados *loftings* verticais(v) e horizontais(u).

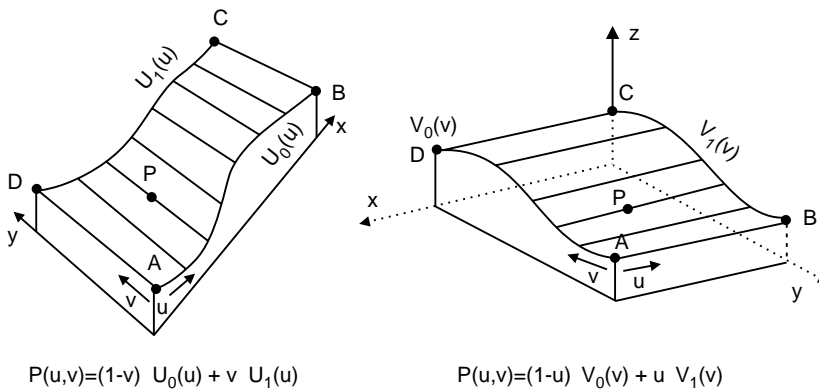


FIGURA 3.24. Loftings verticais e horizontais.

A interpolação resultante é linear a partir das curvas das fronteiras. Por isso, as outras fronteiras limitantes serão retas.

No caso de o limite ser quatro curvas, a interpolação de Coons(1974) resolve esse problema. O resultado obtido, ou seja, a superfície gerada é conhecida na literatura como *retalho de Coon*, *interpolação de Coon* ou *Coons Patches*. Esse método consiste na soma das superfícies geradas pelos loftings verticais e horizontais anteriores, subtraídas da superfície gerada pela interpolação bilinear dos quatro pontos expressos por A, B, C e D. Sendo esses representados por qualquer uma das duas curvas limites que passam por eles, isto é:

$$\begin{aligned}
 A &= U_0(0) = V_0(0) \\
 B &= U_0(1) = V_1(0) \\
 C &= U_1(1) = V_1(1) \\
 D &= U_1(0) = V_0(1)
 \end{aligned}$$

3.2.4. Interpolações Trilineares

As interpolações trilineares são úteis no caso da definição da superfície por três curvas de fronteira. Nessa interpolação, um ponto do interior é definido por três parâmetros u, v e w , cada um com valores entre 0 e 1 (Figura 3.25). Como as superfícies são elementos 2D, ou seja, sempre podem ser descritas com apenas dois parâmetros, obviamente há a restrição adicional de que $w + v + u = 1$ em qualquer ponto.

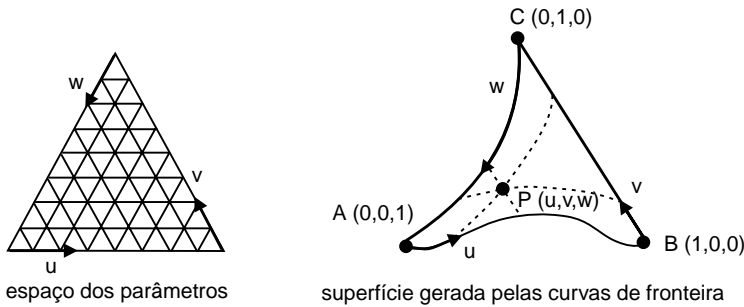


FIGURA 3.25. Interpolações trilineares.

A geração de superfícies a partir de suas fronteiras permite diversas possibilidades de restrições adicionais muito úteis às análises numéricas, como a passagem por pontos ou curvas interiores específicas; a combinação de duas curvas para a formação de um contorno único; a utilização de contornos definidos por curvas fechadas (a Figura 3.26 mostra essa possibilidade); a redução de uma das várias curvas a pontos para discretizar a malha gerada, e diversos outros.

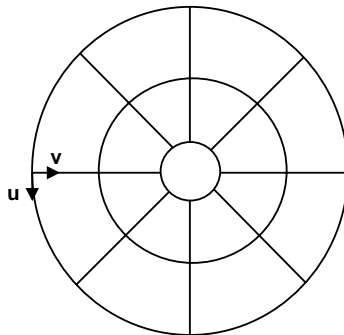


FIGURA 3.26. Restrições adicionais úteis às análises numéricas.

As generalizações mais importantes dessas formas de geração de superfícies pelos seus contornos são os mapeamentos transfinitos. Eles possibilitam a utilização de limites discretos de qualquer número para a definição do contorno, o que os tornam aplicáveis às formas experimentais ao mesmo tempo em que generalizam qualquer forma de descrição das fronteiras.

3.2.5. Superfícies de Formas Livres

A idéia básica dos retalhos (patches), usados para construir uma superfície gerada pelas interpolações da seção anterior, pode ser generalizada para uso com qualquer uma das formas de descrição de superfícies por pontos de controle utilizadas nas seções iniciais deste capítulo, quando a geração de curvas (Hermite, Bézier, Splines ou Racionais) foi desenvolvida.

Superfícies podem então ser geradas pela combinação de pedaços gerados por essas curvas como na confecção de uma “colcha de retalhos” (Figura 3.27). Essa forma é conhecida como geração de superfícies por formas livres.

3.2.6. Superfícies Paramétricas Bicúbicas

São curvas quadrilaterais com representação similar às superfícies poliedrais (Figura 3.27A) com a diferença de serem formadas por superfícies originárias de duas curvas cúbicas (Figura 3.27B). Cada pedaço da malha é definido por uma fórmula matemática (semelhante a da Seção 3.23) que indica sua posição e forma no espaço tridimensional. Essa forma de representação permite obter uma infinidade de formas alterando somente as especificações matemáticas ou os pontos de controle.

As superfícies paramétricas bicúbicas são definidas como:

$$P(u, v) = \sum_{i=0}^3 \sum_{j=0}^3 P_{ij} B_i(u) B_j(v)$$

Como os objetos são descritos por formulações matemáticas podemos extrair com precisão diversas propriedades como massa, volume e área.

Quando alteramos as formas de determinados pedaços, podemos encontrar algumas dificuldades para manter a suavidade em relação aos pedaços vizinhos. Ou-

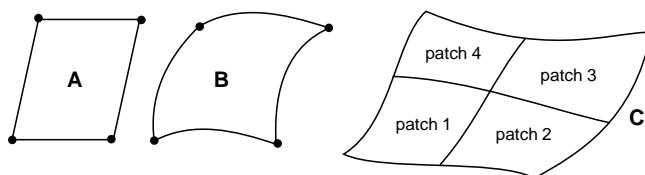


FIGURA 3.27. Superfícies Paramétricas Bicúbicas.

tro problema desta forma de representação é o seu alto custo de tempo para geração de uma visualização realística (render) e memória para representação de objetos complexos (veja o Capítulo 7).

3.2.7. Superfícies de Hermite

A superfície bicúbica de Hermite é uma extensão da formulação da curva de Hermite. As curvas de contorno são definidas pelas expressões de Hermite e pelo interior gerado pelas funções de mistura (blending function). Dois parâmetros são necessários, ambos variando entre 0 e 1. Se esses parâmetros forem chamados de s e t , a superfície bicúbica pode ser escrita como:

$$P(s, t) = S H G_h H^T T^T$$

onde $S = [s^3 s^2 s 1]$, $T = [t^3 t^2 t 1]$, (o índice T indica transporte das matrizes e vetores), H é a matriz de Hermite usada na geração de curvas (Seção 3.1.6) e G_h as condições geométricas que definirão a superfície de Hermite. Essa matriz deve ser representada pelos quatro pontos limites, suas derivadas em relação a s e t e suas derivadas cruzadas. Isto é,

$$G_h = \begin{bmatrix} P(0,0) & P(0,1) & \frac{\partial P}{\partial t}(0,0) & \frac{\partial P}{\partial t}(0,1) \\ P(1,0) & P(1,1) & \frac{\partial P}{\partial t}(1,0) & \frac{\partial P}{\partial t}(1,1) \\ \frac{\partial P}{\partial s}(0,0) & \frac{\partial P}{\partial s}(0,1) & \frac{\partial^2 P}{\partial s \partial t}(0,0) & \frac{\partial^2 P}{\partial s \partial t}(0,1) \\ \frac{\partial P}{\partial s}(1,0) & \frac{\partial P}{\partial s}(1,1) & \frac{\partial^2 P}{\partial s \partial t}(1,0) & \frac{\partial^2 P}{\partial s \partial t}(1,1) \end{bmatrix}$$

Variações da superfície são obtidas trocando os pontos (representados na submatriz superior esquerda), os vetores tangentes ou as derivadas cruzadas (submatriz inferior a direita). A curva pode ser simplificada anulando as derivadas cruzadas. Essa variação é chamada de superfície Ferguson – patch ou F-patch (Faulx, 1987).

3.2.8. Superfícies de Bézier

A equação para os patches de Bézier é, como os de Hermite, uma extensão direta das curvas de Bézier, sendo mais simples de criar e mais intuitivamente modificáveis que aquelas (da Seção 3.27). Um ponto qualquer da superfície pode ser obtido pela expressão:

$$P(s, t) = \sum_{i=0}^n \sum_{j=0}^m B_{i,j} J_{i,n}(s) J_{j,m}(t) \quad 0 \leq s, t \leq 1$$

onde, como no caso das curvas de Bézier, $B_{i,j}$ define o vértice de controle da superfície e $J_{i,n}(s), J_{j,m}(t)$ são as funções de Bernstein nas direções s e t respectivamente (Seção 3.1.7). As funções não precisam ter o mesmo grau nas duas direções, podendo ser cúbicas na direção s e quadráticas na direção t , por exemplo. Os pontos dos quatro cantos da superfície gerada coincidem com os quatro pontos de controle. Na forma matricial, a equação anterior pode ser escrita como:

$$P(s,t) = S M_B G_B M_B^T T^T,$$

onde os vetores S e T têm o significado usual M_B é a matriz de Bezier e G_B os pontos de controle.

Se a superfície de Bézier a ser gerada for definida por dois polinômios de grau 3, teremos as chamadas bicúbicas de Bézier, que podem ser escritas como:

$$P(s,t) = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} M_B G_B M_B^T \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

sendo

$$M_B = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

e os pontos de controle representados pela matriz:

$$G_B = \begin{bmatrix} P_{0,0} & P_{0,1} & P_{0,2} & P_{0,3} \\ P_{1,0} & P_{1,1} & P_{1,2} & P_{1,3} \\ P_{2,0} & P_{2,1} & P_{2,2} & P_{2,3} \\ P_{3,0} & P_{3,1} & P_{3,2} & P_{3,3} \end{bmatrix}$$

Para representar uma superfície Bézier bicúbica, os dezesseis pontos de controle, que a definem mostrados na Figura 3.28 devem ser especificados.

3.2.9. Superfícies B-Spline

As superfícies B-Spline, como as anteriores, são uma extensão das curvas de B-Spline e, podem ser representadas pela expressão:

$$P(s,t) = \sum_{i=0}^n \sum_{j=0}^m B_{i,j} N_{i,k}(s) N_{j,l}(t)$$

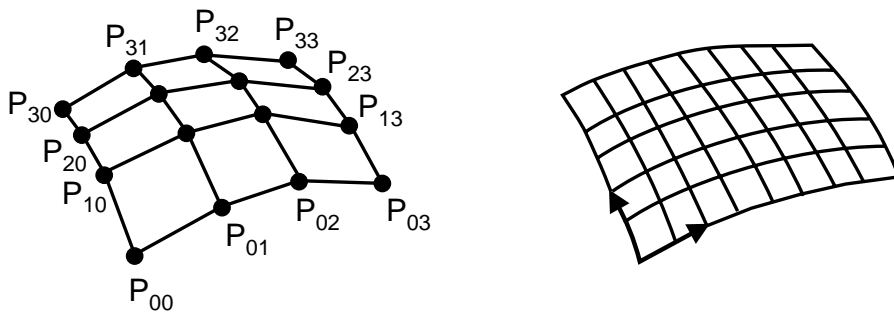


FIGURA 3.28. Os dezesseis pontos de controle de um patch bicúbico de Bézier.

onde $N_{i,k}(s)$ e $N_{j,l}(t)$ são as funções de B-Spline definidas na seção 3.18 de curvas B-Spline, e $B_{i,j}$ são os pontos de controle. Os vetores de nós nas duas direções de parametrização podem ser classificados como periódicos uniformes, não-periódicos ou não-uniformes, como no caso das curvas. As superfícies B-Splines periódicas uniformes são geradas usando vetores de nós uniformes. Como no caso das curvas B-Splines periódicas, essas superfícies também podem ser expressas em formulação matricial que, no caso das bicúbicas, torna a forma:

$$P(s,t) = S M_s G_s M_s^T T^T$$

onde os vetores S e T têm o significado usual, G_s representa a matriz formada pelos dezesseis pontos de controle e as matrizes M_s são as mesmas já usadas para curvas:

$$M_s = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

3.2.10. Normais a Superfícies

Normais a superfícies são importantes para a geração realística de sombreamentos (capítulo 7), cálculo de trajetórias em jogos, detecção de interferências em robótica e para diversos cálculos na modelagem de objetos.

Essas normais, no caso de superfícies bicúbicas, são fáceis de se obter. Todas as expressões na forma matricial têm tangentes na superfície $P(s,t)$ dadas pelas derivadas parciais

$$\frac{\partial}{\partial s}(s,t) = [3s^2 \ 2s \ 1 \ 0] M G M^T T^T$$

e

$$\frac{\partial}{\partial s}(s, t) = S M G M^T \begin{bmatrix} 3t^2 \\ 2t \\ 1 \\ 0 \end{bmatrix}$$

de modo que, para calcular o valor do vetor tangente em um ponto, basta substituir nas expressões anteriores as coordenadas s e t do ponto onde deseja-se conhecer a tangente e as matrizes do tipo da curva desejada. Por exemplo, para as cúbicas de Hermite, as tangentes no ponto $s = \frac{1}{2}$ e $t = 1$ serão definidas pelos vetores

$$P'_s = (14; 9,25; 12)$$

$$P'_t = (11,375; 11,25; 10)$$

3.2.11. Superfícies Racionais

O processo de geração de superfícies (seção 3.1.9) racionais também é uma extensão do processo usado para gerar as curvas racionais, aplicando os mesmos conceitos de coordenadas homogêneas. A tabela que segue compara as expressões.

| $P(s, t) =$ | Forma Inteira | Forma Racional |
|-----------------|---|---|
| Bézier | $\sum_{i=0}^n \sum_{j=0}^m B_{i,j} J_{i,n}(s) J_{j,m}(t)$ | $\frac{\sum_{i=0}^n \sum_{j=0}^m w_{i,j} B_{i,j} J_{i,n}(s) J_{j,m}(t)}{\sum_{i=0}^n \sum_{j=0}^m w_{i,j} J_{i,n}(s) J_{j,m}(t)}$ |
| B-Spline | $\sum_{i=0}^n \sum_{j=0}^m B_{i,j} N_{i,k}(s) N_{j,l}(t)$ | $\frac{\sum_{i=0}^n \sum_{j=0}^m w_{i,j} N_{i,k}(s) N_{j,l}(t) B_{i,j}}{\sum_{i=0}^n \sum_{j=0}^m w_{i,j} N_{i,k}(s) N_{j,l}(t)}$ |

Onde B_i são os pontos de controle, $w_{i,j}$ são os pesos de cada ponto de controle (coordenada homogênea) e $N_{i,k}(s)$, $N_{j,l}(t)$, $J_{i,n}(s)$ e $J_{j,m}(t)$ são as funções de interpolação usadas.

No caso das superfícies de B-Spline, o vetor de nós usado pode ser periódico e uniforme; não-periódico ou não-uniforme, podendo ser usados vetores de nós diferentes nas direções s e t , como no caso das curvas onde os pesos $w_{i,j}$ dão um grau de liberdade adicional para a forma da superfície a ser obtida. A maioria das proprieda-

des das superfícies não-rationais estende-se à sua forma similar racional. Por exemplo, se rotações e translações forem aplicadas aos pontos de controle, a superfície será interior ao convex hull do poliedro que a envolve.

3.2.12. NURBS

O termo NURBS é a abreviatura de Non-Uniform Rational B-Splines Surfaces, ou seja, é uma B-Spline racional (originária da razão de polinômios). Non-Uniform significa que a influência da extensão de um controle de vértice não precisa ser a intervalos iguais do parâmetro t , podendo variar (o que é muito bom na modelagem de superfícies irregulares). Rational significa que a equação usada para representar a curva ou superfície é expressa pela razão de dois polinômios. A forma Rational (como comentado na seção anterior) fornece um modelo melhor de algumas importantes superfícies especialmente para formas cônicas e esféricas.

As superfícies NURBS não existem no mundo do desenho tradicional. Elas foram criadas especialmente para a modelagem em três dimensões no computador. As NURBS são construídas matematicamente e representam formas de um espaço 3D. O uso de NURBS oferece uma forma matemática comum tanto para a análise quanto para a geração de formas livres. Ela provê uma flexibilidade adicional para projetar uma grande variedade de formas e pode ser avaliada de maneira razoavelmente rápida por algoritmos numericamente estáveis e precisos.

Essas superfícies são obtidas com o uso de matrizes $B_{i,j}$ de nós não-uniformes e são uma das formas de representação mais usadas em projetos de engenharia, principalmente por englobar todas as outras formas de representação.

A forma geral das B-Splines racionais, descritas na seção anterior, é também usada nas formas não-uniformes. A única restrição é (como o nome indica) que os vetores de nós sejam não-uniformes. Ela pode fazer com que todas as demais formas de representação sejam casos particulares dela, bastando seguir as seguintes restrições:

| Tipo De Curva | Restrição |
|------------------------|--|
| B-Spline não- racional | Fixar todos os pesos $w_{i,j}=1$ |
| Bézier racional | Se o número de pontos de controle é igual à ordem em cada direção dos parâmetros e não existirem nós interiores duplicados |
| Bézier não- racional | As mesmas das racionais, além de fixar que o peso seja $w_{i,j}=1$ |

Além disso, as NURBS têm a habilidade de representar superfícies quadráticas.

A característica não-uniforme das NURBS nos leva a um importante ponto. Por serem geradas matematicamente na forma homogênea, NURBS possuem um parâmetro adicional em 3D. Especificamente, um “array” de valores chamados nós especifica a influência de sua extensão para cada vértice da curva ou superfície. Os nós são invisíveis

A função polinomial das curvas NURBs é definida como:

$$P(t) = P^w(t) = \frac{\sum_{i=0}^n w_i B_i N_{n,i}(t)}{\sum_{i=0}^n w_i N_{n,i}(t)}$$

Continuidade e grau estão relacionados. O grau 3 pode gerar curvas de continuidade C^2 . Esse é o motivo pelo qual curvas de maior grau não são necessárias para modelagem de NURBs. Curvas de graus maiores são também menos estáveis numericamente, o que as torna não recomendáveis.

Existem três pontos de controles no ápice da esquerda e dois no ápice da direita nas curvas da Figura 3.29. Movendo um dos controles de vértices para longe do outro, o nível de continuidade da curva será aumentado. A multiplicidade também é aplicada quando os pontos de controle são fundidos, gerando a criação de curvas refinadas ou quebradas. Refinar uma curva significa adicionar mais pontos de controle à curva. (Figura 3.29). Quando uma curva é refinada, sugerimos reparametrizá-la. A reparametrização ajusta os parâmetros para que a curva se comporte adequadamente quando em edição.

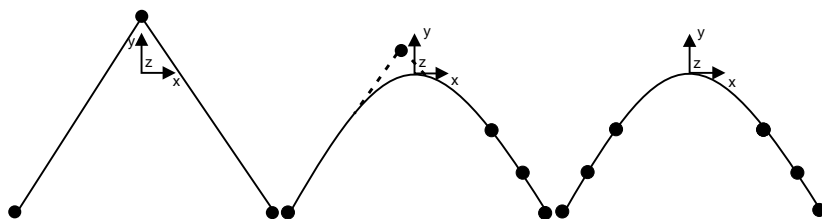


FIGURA 3.29. Adicionando controles de vértices à curva.

Os reparametrizadores mais úteis são os de comprimento de curva e os uniformes. No primeiro caso, os espaços entre os nós são reparametrizados com base na raiz quadrada do comprimento de cada segmento da curva. No segundo caso, ocorre a reparametrização uniforme dos espaços entre os nós.

3.2.12.1. NURBS em OpenGL

Antes de chamarmos qualquer função NURBS, temos de dizer ao OpenGL que criaremos um NURB. Isso pode ser feito pela chamada de função

```
nurbSurface = gluNewNurbsRenderer( );
```

Cada chamada a essa função retorna um ponteiro para a `GLUnurbsObj`, que deve ser usado com as demais funções NURBS. Dessa maneira, podemos criar quantos patches NURBS desejarmos ajustando suas propriedades individualmente.

Para ajustarmos as propriedades de um NURBS, chamamos a função `gluNurbsProperty()`.

```
gluNurbsProperty( GLUnurbsObj *nurb, GLenum property, GLfloat value );
```

As propriedades devem ser uma das que seguem:

GLU_SAMPLING_TOLERANCE: especifica o comprimento máximo em pixel dos polígonos que compõem a malha. Quanto menor, mais suave parecerá a malha, porém demandará mais tempo para o render. O valor padrão é de 50 pixels.

GLU_DISPLAY_MODE: define como um NURBS deve ser renderizado entre as seguintes opções: `GLU_FILL`, `GLU_OUTLINE_POLYGON`, ou `GLU_OUTLINE_PATCH`

GLU_CULLING: este é um valor booleano. `GL_TRUE` significa que uma superfície NURBS deve ser descartada se os seus pontos de controle estiverem fora da janela de visualização. O padrão é `GL_FALSE`.

GLU_AUTO_LOAD_MATRIX: este também é um valor booleano. `GL_TRUE` significa que um NURBS utiliza as matrizes de projeção corrente, modelview e viewport. `GL_FALSE` requer a especificação de matrizes através da função `gluLoadSamplingMatrices()`.

Agora que ajustamos as propriedades basta desenhar a curva através da função:

```
gluNurbsSurface( GLUnurbsObj *nurb, GLint uKnotCount, GLfloat *uKnot,
                 GLint vKnotCount, GLfloat *vKnot, GLint uStride,
                 GLint vStride, GLfloat *ctrlArray, GLint uOrder,
                 GLint vOrder, GLenum type );
```

- *nurb* é o ponteiro do objeto NURBS
- *uKnotCount* especifica o número de nós na direção paramétrica *u*
- *uKnot* especifica um array de nós crescente na direção *u*
- *vKnotCount* especifica o número de nós na direção paramétrica *v*
- *vKnot* especifica um array de nós crescente na direção *v*
- *uStride* especifica o equilíbrio entre pontos de controles sucessivos na direção paramétrica *u* em *ctrlArray*
- *vStride* especifica o equilíbrio entre pontos de controles sucessivos na direção paramétrica *v* em *ctrlArray*
- *uOrder* especifica a ordem da superfície NURBS na direção *u*
- *vOrder* especifica a ordem da superfície NURBS na direção *v*
- *type* especifica o tipo de superfície, que poderá ser `GL_MAP2_VERTEX_3` or `GL_MAP2_COLOR_4`

Se desejar desenhar uma linha curva, em vez de uma superfície, você poderá chamar a função `gluNurbsCurve()` que possui os mesmos parâmetros, menos a direção *v*.

3.2.13. Superfícies NURMS

Foram Catmull e Clark que primeiro notaram que as regras de subdivisão podiam ser estendidas para incluir malhas de topologia arbitrária. Em 1978, através da publicação de *Recursively generated B-spline surfaces on arbitrary topological surfaces*, eles mostraram que a superfície limite é localmente uma B-spline bicúbica uniforme, exceto em alguns pontos na superfície que eles chamaram pontos extraordinários.

Essa técnica de subdivisão fez surgir as ferramentas de modelagem NURMS (Non-Uniform Rational Mesh Smooth). As superfícies NURMS são mais fáceis e rápidas de gerar e alterar do que as superfícies NURBS e, por isso, vêm ganhando adeptos quando o assunto é modelagem de personagens ou modelagem de objetos de contornos suaves. A Figura 3.30 demonstra a aplicação de NURMS em um modelo de face de poucos polígonos (low-poly model), observe os contornos do queixo. Outro exemplo clássico para rápida e fácil geração de personagens é demonstrado na Figura 3.31, onde as NURMS são aplicadas a um modelo do corpo de um personagem com poucos polígonos.

Utilizando a subdivisão de superfícies B-spline bicúbicas uniformes, Catmull e Clark seguiram a metodologia de Doo e Sabin e notaram que a regra de subdivisão expressa para a superfície B-spline cúbica não só trabalha para malhas retangulares, mas também pode ser estendida para malhas de uma topologia arbitrária. Essa extensão pode ser realizada generalizando a definição de um ponto da superfície, mo-

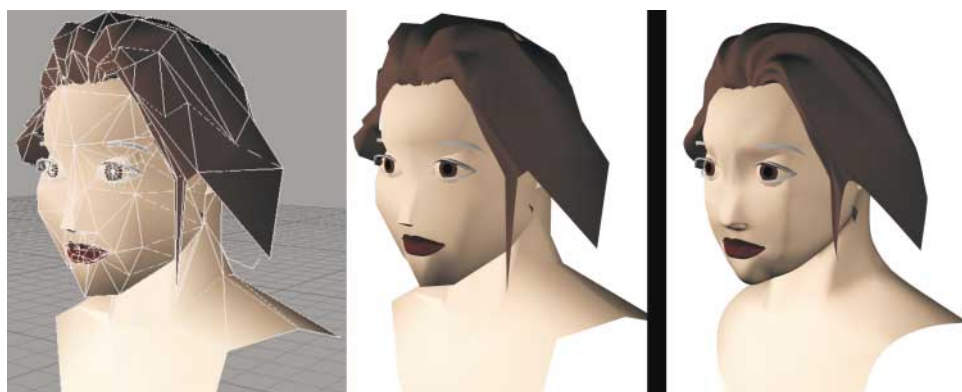


FIGURA 3.30. NURMS aplicado em um modelo de face de poucos polígonos.

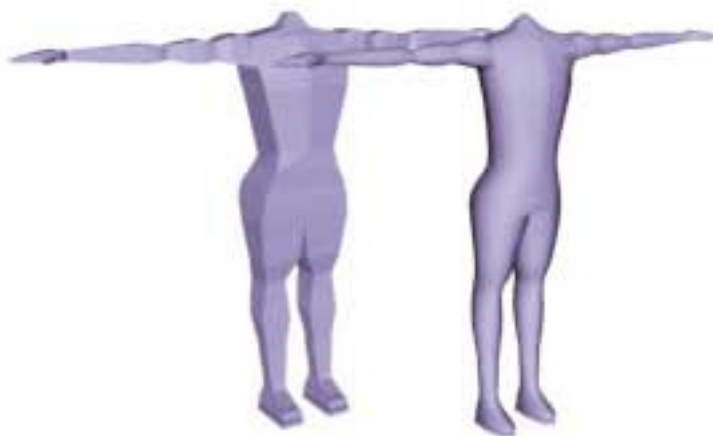


FIGURA 3.31. NURMS aplicado a um modelo de corpo de poucos polígonos.

dificando o método para calcular novos pontos nos lados da malha poligonal e especificando um método para reconectar os pontos da malha.

O procedimento de refinamento pode ser exemplificado com o uso de uma malha formada por quatro triângulos, como mostra a Figura 3.32:

- Primeiro, construímos os pontos médios das superfícies. Esses pontos são calculados com a média aritmética das coordenadas dos pontos que compõem os vértices de cada triângulo. Esses pontos são mostrados na Figura 3.33, pela letra F. Podemos dizer que o cálculo de F_1 , por exemplo, é obtido de:

$$F_1 = G_1 + G_2 + G_0 / 3,$$

e do mesmo modo se obtém os demais F_2, F_3, F_4 .

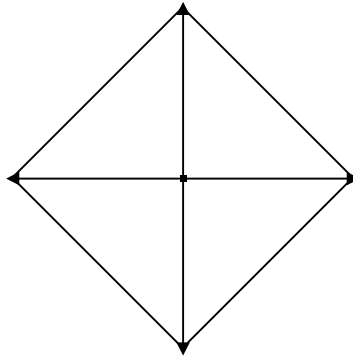


FIGURA 3.32. A malha de quatro triângulos.

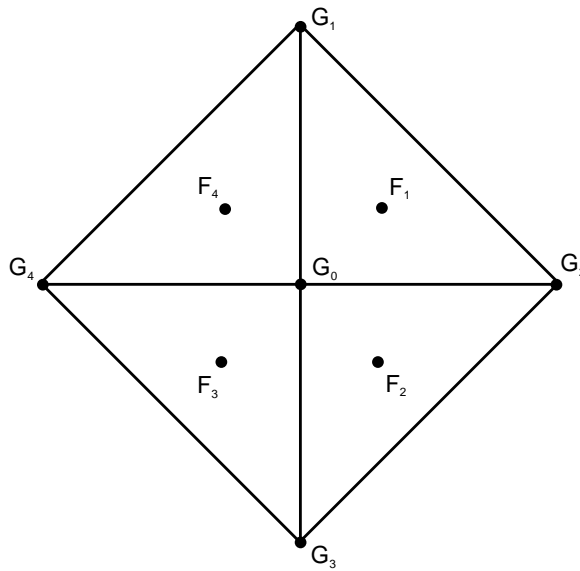


FIGURA 3.33. Determinando os pontos médios das superfícies triangulares.

- Agora, construímos os novos pontos para subdivisão da malha, que são calculados pela média dos quatro pontos formados pelos dois novos pontos (F) de lados adjacentes e os dois pontos dos vértices (G) do triângulo original. Os novos pontos são identificados pela letra E na Figura 3.34. Dessa maneira, E₁ será o resultado da média dos pontos F₁, F₄, G₁ e G₀. Do mesmo modo, determina-se os demais E₂, E₃, E₄;
- Os novos pontos de vértice são calculados pela expressão:

$$\frac{Q}{n} + \frac{2R}{n} + \frac{S(n-3)}{n}$$

onde Q é a média dos novos pontos (F) de todas as faces adjacentes à malha original, R é a média dos pontos centrais (E) de todas as extremidades adjacentes originais, S é o ponto de vértice original e n é o número de vértices da face em consideração. Este ponto, pelo menos para esse exemplo em duas dimensões, é idêntico ao centro do diamante. Este ponto está representado na Figura 3.35 pela letra V .

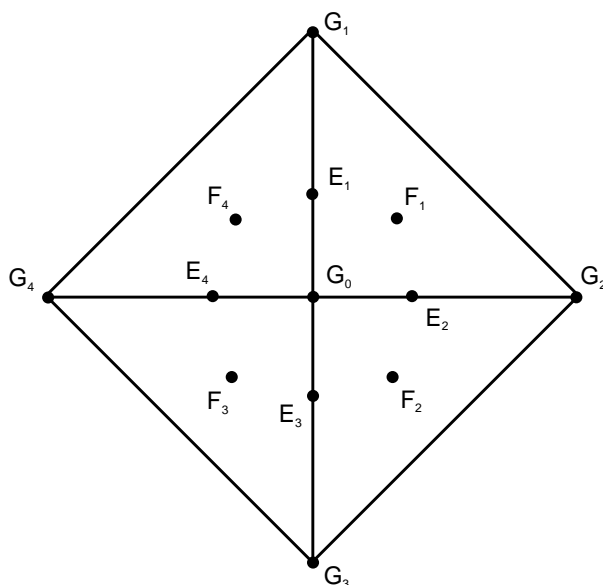


FIGURA 3.34. Determinando os pontos de extremidade.

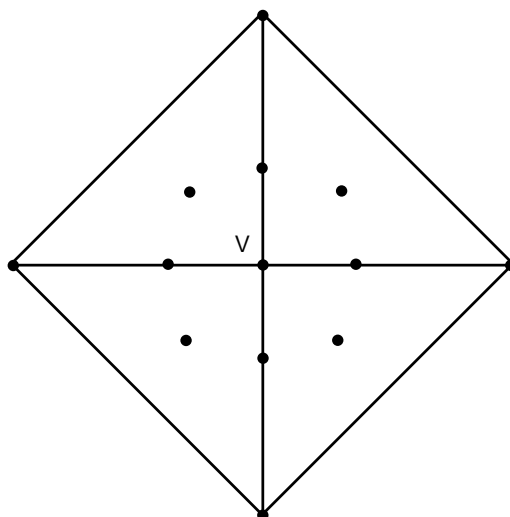


FIGURA 3.35. Determinando os novos pontos de vértice.

- Finalmente, conectamos as extremidades aos pontos que geramos: primeiro conectando os novos pontos das faces (pontos E e F), e então conectando os novos pontos de vértice aos pontos das faces originais, neste caso, conectamos todos os pontos E ao G_0 (Figura 3.36).

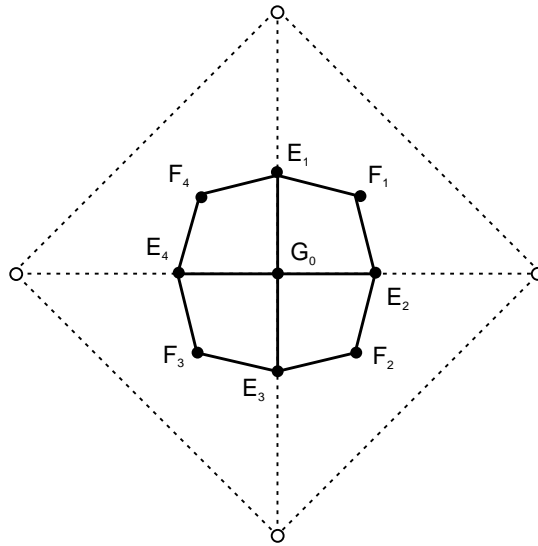


FIGURA 3.36. Conectando as extremidades dos novos pontos.

A Figura 3.37 exemplifica a inclusão de malhas na topologia de um objeto 3D com a técnica de refinamento de Catmull-Clark.

Outras implementações de modelagem de superfícies por subdivisão podem ter características diferentes. Alguns produtos como o *Pixar Render Man* dão suporte a subdivisão de superfícies usando NURBS. Algumas engrenagens 3D de jogos e visualizadores 3D para Internet suportam subdivisão de superfície em *real-time*.

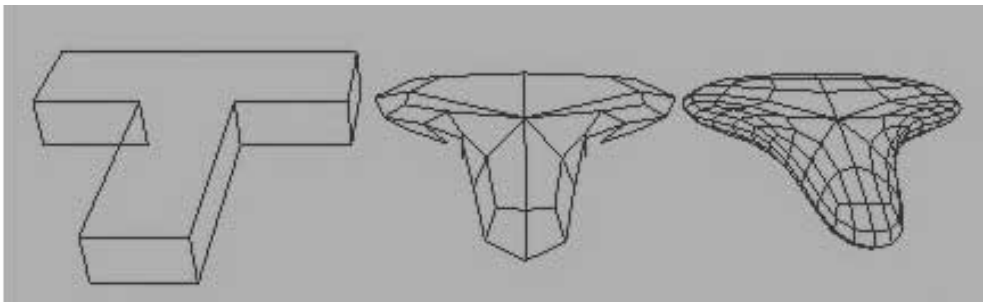


FIGURA 3.37. Inclusão de malhas na topologia de um objeto 3D com a técnica de Catmull-Clark.

CAPÍTULO 4

Representação e Modelagem

- 4.1. Pivô
- 4.2. Sólidos uni, bi e tridimensionais
- 4.3. Sólidos realizáveis
- 4.4. Formas de representação de objetos
 - 4.4.1. Representação Aramada (Wire Frame)
 - 4.4.2. Representação por Faces (ou Superfícies Limitantes)
 - 4.4.3. Representação por Faces Poligonais
 - 4.4.4. Fórmula de Euler
 - 4.4.5. Operadores de Euler
 - 4.4.6. Estrutura de dados baseada em vértices
 - 4.4.7. Estrutura de dados baseada em arestas
 - 4.4.8. Estrutura de dados Winged-Edge e Half Winged-Edge
 - 4.4.9. Utilização de Operações Booleanas com B-rep
- 4.5. Técnicas de Modelagem Geométrica
 - 4.5.1. Instanciamento de Primitivas
 - 4.5.2. Objetos de Combinação
 - 4.5.3. Operações Booleanas e Booleanas Regularizadas
 - 4.5.4. Geometria Sólida Construtiva (*CSG-Constructive Solid Geometry*)
 - 4.5.5. Connect
 - 4.5.6. Modelagem por Varredura (*Sweep*) ou Deslizamento
 - 4.5.7. Modelagem por Seções Transversais
 - 4.5.8. Modelagem por Superfícies
- 4.6. Modificadores
 - 4.6.1. Bend ou Curvatura
 - 4.6.2. Lattice ou Malha
 - 4.6.3. MeshSmooth ou Suavização
 - 4.6.4. Optimize

- 4.6.5. Bevel
- 4.6.6. Melt
- 4.6.7. Skew
- 4.6.8. Squeeze ou Stretch
- 4.6.9. Taper
- 4.6.10. Twist
- 4.6.11. Diaplace ou Deslocamento
- 4.6.12. Space Warps
- 4.7. Interfaces para Modelagem Geométrica
 - 4.7.1. Tape (Fita de Medida)
 - 4.7.2. Protractor (Transferidor)
 - 4.7.3. Compass (Bússola)
 - 4.7.4. Array (Vetor)
- 4.8. Modelagem Geométrica com OpenGL
 - 4.8.1. Desenhando um Ponto
 - 4.8.2. Desenhando uma Linha
 - 4.8.3. Desenhando um Polígono
 - 4.8.4. Primitivas
 - 4.8.5. Objetos Sólidos
 - 4.8.6. Wireframe
- 4.9. Modelagem pelo Número de Ouro
 - 4.9.1. A Seqüência de Fibonacci
- 4.10. Modelagem Fractal
 - 4.10.1. Os Objetos de Mandelbrot
- 4.11. Shape from X – Reconstrução Tridimensional
 - 4.11.1. Implementação
- 4.12. Sistemas de Partículas
 - 4.12.1. Distribuição de partículas no sistema
 - 4.12.2. Noção de centro de massa
 - 4.12.3. Velocidade do centro de massa
 - 4.12.4. Quantidade de movimento
 - 4.12.5. Aceleração do centro de massa
 - 4.12.6. Movimento de uma partícula em relação ao seu centro de massa
 - 4.12.7. Choque
 - 4.12.8. Quantidade de movimento de um sistema de duas partículas
 - 4.12.9. Energia de um sistema de duas partículas
 - 4.12.10. Conceito de Sistemas de Partículas
 - 4.12.11. Renderizando as Partículas
 - 4.12.12. Características dos Sistemas de Partículas

Quando modelamos um objeto, devemos sempre observar as limitações resultantes das técnicas disponíveis. As atuais técnicas de modelagem, apesar de eficientes para determinadas situações, podem representar um problema para outras. Se tomarmos um objeto complexo, por exemplo, uma cabeça humana, veremos que este poderá ser inviável para utilização em aplicações de tempo real (como Games ou Realidade Virtual). Uma cabeça com riqueza de detalhes terá no mínimo 300.000 polígonos. Podemos ainda citar diversas outras dificuldades relacionadas com a representação de um rosto: como as expressões faciais realistas ou custo de renderização.

Tornar-se um bom modelador poderá garantir o sucesso profissional de quem deseja trabalhar com computação gráfica. Na verdade, modelar é um assunto que exige muita intuição, conhecimento e experiência. A modelagem está presente em quase todas as aplicações: da medicina (para diagnósticos e ensino) à indústria (para garantir a precisão dos projetos), no entretenimento (para criar personagens e cenários virtuais) e em uma infinidade de outras aplicações.

4.1. PIVÔ

Antes de iniciarmos os estudos sobre modelagem, precisamos conhecer uma peça importante do assunto: o Pivô, ou usando a terminologia da seção 2.4 o sistema de referência do objeto. (Figura 4.1).

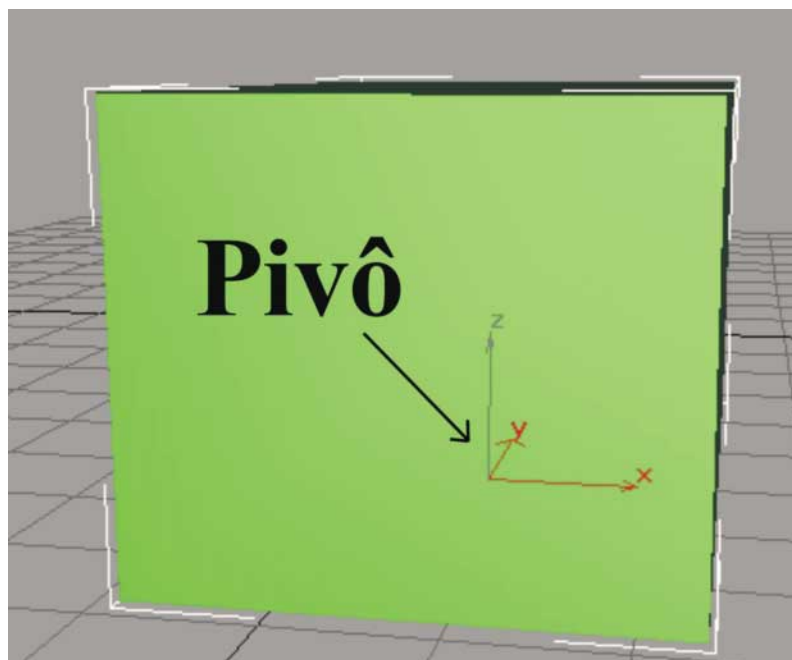


FIGURA 4.1. O ponto pivô.

Todos os objetos têm um pivô (e um sistema de coordenadas próprio). Você pode pensar sobre o pivô como o centro do objeto ou o centro de coordenadas locais. Quando modelamos um objeto, usando um software, podemos observar que o pivô é criado automaticamente pelo sistema. Nesse momento, é importante verificar se a escolha feita pelo sistema será adequada para as modificações que aplicaremos. Muitas vezes será necessário alterar a posição padrão do sistema para realizar uma modificação qualquer no objeto. O pivô de um objeto é usado em funções de transformação (Seção 2.5) como centralizar, rotacionar e dimensionar; para determinar a posição de atuação de um modificador (como veremos mais à frente) e para definir a relação de transformação para os elementos conectados. Agora que conhecemos o pivô, podemos iniciar os estudos de modelagem pelos sólidos.

4.2. SÓLIDOS UNI, BI E TRIDIMENSIONAIS

O termo modelagem de sólidos designa um conjunto de teorias, técnicas e sistemas que permitem criar um sólido com suas propriedades geométricas. Mas antes vejamos se está bem clara a idéia do que é um sólido.

Algo é considerado um sólido se tem uma forma própria. A modelagem de líquidos, gases, materiais flexíveis ou de coisas que não tenham forma própria (roupas, tecidos, plásticos, gel e outros) é também necessária e representável em computação gráfica. Muitos modelos representativos desses comportamentos complexos ainda estão em estágio de desenvolvimento e pesquisa.

Uma definição mais precisa de sólido é encontrada em uma das ótimas referências sobre modelagem de sólidos, o livro do pesquisador finlandês Martti Mäntylä: *An introduction to solid modeling*, publicado pela *Computer Science Press* em 1988:

Definição: Um sólido é um subconjunto *fechado e limitado* do espaço Euclidiano tridimensional: E^3

Essa definição esclarece alguns pontos muito importantes sobre um sólido. O mais importante talvez é que, associada à idéia de sólido está a idéia de ser algo essencialmente **tridimensional** (abreviadamente: **3D**). Assim, embora em muitos casos o objeto em estudo possa ser considerado unidimensional, como um fio, ou bidimensional como folhas de metal ou papel, será sempre algo essencialmente parte de nosso mundo físico palpável e, portanto, **3D**. Poderemos considerá-lo bidimensional (**2D**) se uma das dimensões não for considerada, por nela nada ocorrer ou por ser desprezível (for 100 vezes menor que qualquer uma das outras dimensões, por exemplo, que concentram todas as informações sobre o objeto). Do mesmo modo, o sólido pode ser considerado unidimensional (**1D**) se só uma das dimensões concentrar as informações ou as demais forem desprezíveis face a essa (uma delas for 100 vezes maior que qualquer uma das outras dimensões, por exemplo).

Outro ponto importante é ser um elemento **Euclidiano**. Algo que obedece às regras da geometria formulada pelo matemático grego Euclides, que viveu na Alexandria no século III A.C., ou seja, obedece à geometria que estamos acostumados a tratar desde que entramos na escola e que é regida por todo um conceito de regras rigorosamente lógicas e bem definidas, formuladas na obra de Euclides: *Elementos de Geometria*. Nessa obra, é construído tudo o que é, até hoje, usado no ensino da geometria, partindo de definições simples e do célebre *postulado* ou *axioma* de Euclides: *por um ponto pode-se traçar uma única paralela a uma reta*. Essa afirmação nunca pôde ser provada como teorema, embora isso tenha sido tentado por um grande número de matemáticos ilustres no decorrer dos séculos. O próprio Carl Friedrich Gauss (1777 – 1855) voltou sua atenção a esse postulado e tentou prová-lo, falhando como todos os demais que o tentaram. No entanto, Gauss, com a idade de 15 anos, mostrou que trocando um ou mais dos axiomas euclidianos toda uma *estrutura completamente diferente das lógicas de geometria surgiria*, ou seja, como axioma a afirmação pode ser posta em dúvida e mesmo eliminada. Pela sua substituição surgiram as **geometrias não-euclidianas** [Maor, 1987].

Foi apenas no século XIX que a geometria de Euclides deixou de ser verdade absoluta e surgiram as primeiras geometrias não-euclidianas desenvolvidas por Lobatchevski (Nicolai Ivanovitch Lobachevski, matemático russo: 1793-1856) e Riemann (Georg Friedrich Bernhard Riemann, matemático alemão: 1826-1866). Atualmente, existem diversas outras geometrias, cada uma com aplicações específicas, como, por exemplo, a geometria Fractal, que é muito importante para a modelagem em computação gráfica de formas da natureza, como plantas, montanhas, nuvens, e outras, como comentaremos no final deste capítulo [Falconer, 1990].

Há ainda dois pontos a serem comentados na definição de sólido citada, as palavras: **fechado** e **limitado**. Vamos ficar com os conceitos intuitivos dessas afirmações (embora, pelo menos, o conceito de fechado seja bem familiar se você tiver ainda lembrança da teoria de conjuntos). Um conjunto, um corpo ou qualquer objeto é **fechado** se tiver todos os seus pontos de contorno, assim como uma pele que limita o interior do corpo humano do mundo exterior. **Limitado** está associado à idéia de não ter alguma dimensão infinita. Infinito no sentido de realmente não ter fim, e não apenas de ser muito grande.

4.3. SÓLIDOS REALIZÁVEIS

Os modeladores de sólidos devem gerar objetos que mesmo que não sejam reais possam ser construídos ou virem a existir no nosso mundo euclidiano do dia a dia, e não simples desenhos. Um sólido é considerado realizável ou válido se satisfizer às seguintes propriedades:

- **Rigidez:** o objeto deve possuir **forma invariante** se for movido de um lugar para outro. Mais rigorosamente, isso é expresso dizendo que deve ser invari-

ante sobre **transformações de corpo rígido**. São elas: a rotação, a translação e a mudança dos sistemas de coordenadas, usadas para descrever suas coordenadas. Ou seja, ter a **propriedade de rigidez** significa que o sólido deve ser independente da sua localização e orientação no espaço;

- **Finitude**: o objeto deve ter dimensões finitas e ser contido em uma porção finita do espaço. Essa propriedade está relacionada ao conceito de limitado descrito anteriormente na definição de sólido;
- **Homogeneidade**: o objeto deve ter as mesmas propriedades em todos os seus pontos interiores;
- **Determinismo dos limites**: deve ser possível descrever o limite, o interior e o exterior do objeto. Essa propriedade está relacionada ao conceito de fechado descrito anteriormente na definição de sólido;
- **Descritibilidade**: o objeto deve poder ser descrito através de um número finito de propriedades físicas, químicas, biológicas etc.;
- **Fechamento sobre operações**: o resultado de operações geométricas realizadas em objetos válidos deve ser ainda um objeto válido.

Os sólidos podem ser representados por diversas formulações. As propriedades desejáveis das formas de representação são:

- **Validade**: o modelo deve representar somente sólidos válidos;
- **Unicidade**: cada sólido válido deve ter apenas um modelo;
- **Não ambigüidade**: cada modelo deve corresponder a apenas um sólido válido;
- **Compleitude**: o modelo deve ser completo, ou seja, conter uma variedade de informações sobre as quais as várias funções possam ser executadas. Deve ser possível obter informações suficientes a partir do modelo, mesmo após eventuais transformações por funções que operam sobre o mesmo;
- **Concisão**: o modelo deve ocupar o menor espaço de memória possível;
- **Simplicidade**: deve ser possível criar o modelo de uma forma simples e direta, sem que nenhuma característica especial de hardware seja exigida;
- **Eficiência**: as operações devem ser de fácil aplicação e apresentar respostas rápidas;
- **Fechamento sobre operações**: as operações de descrição e manipulação devem preservar a validade do modelo.

4.4. FORMAS DE REPRESENTAÇÃO DE OBJETOS

Cada método de representação tem suas vantagens e desvantagens. Uma solução ideal poderá ser uma **forma híbrida**, isto é, uma mistura de alguns desses métodos.

4.4.1. Representação Aramada (Wire Frame)

Nesta representação, os objetos são descritos por um conjunto de arestas que define as bordas do objeto. O método nada mais é do que uma extensão 3D do método de representação de objetos 2D por contornos.

A principal vantagem dessa técnica é a sua velocidade na exibição dos modelos, sendo necessário apenas exibir um conjunto de linhas. Porém, há sérios inconvenientes nos usos da representação aramada. Um dos inconvenientes é o fato de gerar uma representação ambígua com margem para várias interpretações. A Figura 4.2 exemplifica esse problema. Em “A” tem-se a representação aramada e em “B” e “C”, duas possíveis interpretações. O problema não reside propriamente no fato de que a simples exibição das linhas gera ambigüidades, mas na constatação de que o modelo não fornece informações precisas para que estas sejam eliminadas (no exemplo, seria necessário remover as arestas da parte traseira ou dianteira do objeto). Outra inconveniência está na dificuldade de realizar certas operações como a determinação de massa ou volume. No sentido restrito da modelagem de sólidos, essa técnica não tem como garantir que o objeto desenhado seja um sólido válido, e geralmente nem é considerada uma técnica de modelagem de sólidos.

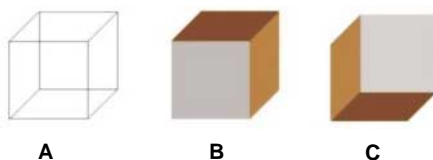


FIGURA 4.2. Representação aramada em A e suas possíveis interpretações em B e C.

4.4.2. Representação por Faces (ou Superfícies Limitantes)

Esta representação de sólidos usa suas superfícies limites para descrever seus contornos. Essas superfícies são supostas fechadas e orientáveis. Orientáveis, nesse caso, significa que é possível distinguir entre dois lados da superfície, de modo que um esteja no interior e o outro no exterior do sólido. Essa técnica consiste em definir um modelo através de um conjunto de superfícies que delimita a região fechada do espaço que define o interior do modelo. As superfícies que limitam a região recebem o nome de “faces”. Essa técnica é também denominada *Boundary Representation* ou *B-rep*, consiste na descrição de objetos pelos seus contornos, ou seja, suas faces, arestas e vértices. Pode ser restrita a formas definidas (cúbicas, elipsóides, esféricas), aproximadas por superfícies planas ou pode ser descrita por faces representadas por retalhos de superfícies curvas.

A representação de sólidos por faces pode também ser vista como uma simples extensão da modelagem 2D por contornos. Como no caso 2D, onde há pelo menos duas formas de armazenamento das arestas, em 3D as faces podem ser representa-

das através de uma ou mais listas explícitas de vértices contendo a **topologia** e a **geometria** da face. A primeira se encarrega de fazer as relações entre os elementos, por exemplo: aresta a_i é limitada pelos vértices V_l e V_m e faces F_n e F_o . A segunda define as posições dos elementos no espaço, pela determinação da sua forma geométrica (a aresta a_i é uma semi-reta, ou um arco de círculo, por exemplo, e as coordenadas do vértice V_l do modelo são 10,20,30).

A representação pelos seus limites é a forma mais encontrada na modelagem de sólidos em geral. Muitos sistemas que a usam se limitam, no entanto, a representação de sólidos que tenham contornos formados por superfícies que sejam “variedades de dimensão 2” ou *2-manifold*.

Manifold é uma palavra que em inglês arcaico quer dizer algo que tem muitas partes, muitos elementos ou formas. Mas matematicamente é um termo que indica um conjunto de objetos que tem determinadas propriedades topológicas. Por definição, um objeto é um *manifold* de dimensão 2 ou uma *variedade de dimensão 2* se cada um dos pontos de seu contorno tiver uma vizinhança que seja **topologicamente equivalente** a um disco plano.

Uma definição formal simples é encontrada em Mäntylä (1988):

Definição: Um 2-manifold é um espaço topológico, onde cada ponto tem uma vizinhança topologicamente equivalente a um disco do espaço Euclidiano bidimensional, E^2 .

Um objeto é topologicamente equivalente a outro se é possível encontrar uma função *biunívoca* (*inversível*) e contínua que *mapeie* os pontos do contorno de um no outro. Obviamente, a melhor forma de entender a noção de equivalência topológica é por uma analogia física. Supondo que o objeto seja feito de um material extremamente elástico (como as membranas das bolas de encher de aniversário), todas as transformações que mantêm as vizinhanças das superfícies são modificações topologicamente equivalentes. Isso é, se você mudar as formas da geometria construída nesse material elástico (puxando e esticando), até conseguir transformá-lo em outra forma, terá apenas transformações geométricas e não topológicas, ou seja, todas as transformações que você puder fazer em uma membrana elástica sem rasgá-la ou colar um ponto a outro, podem ser entendidas como equivalentes físicos de transformações contínuas e biunívocas (ou inversíveis). Essas operações não mudam as *relações de vizinhança* entre os pontos da região, ou seja, não mudam a *topologia* da região.

Assim, em um objeto com superfícies que sejam variedades de dimensão 2 (*2-manifold*), todos os seus pontos têm uma vizinhança topologicamente equivalente a um disco. Uma forma de ter uma visualização disso é imaginar que você tem uma etiqueta adesiva na forma de um círculo e deseja colá-la no objeto (analogia da etiqueta adesiva). Se você puder colá-la em qualquer ponto da superfície desse objeto en-

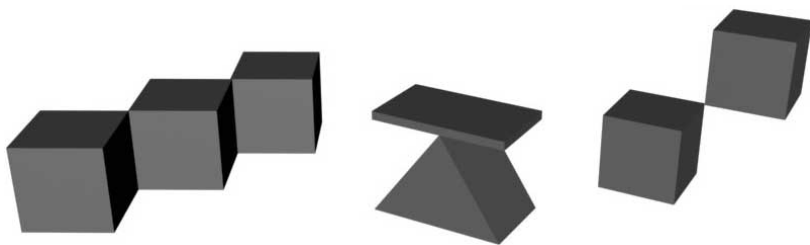


FIGURA 4.3. Exemplo de 3 formas simples que não são variedades de dimensão 2 se forem consideradas como objetos únicos. A da esquerda por ter arestas que limitam 4 faces (de modo que para termos a vizinhança de um ponto precisaríamos de pelo menos duas etiquetas adesivas), a do centro por ter uma aresta que limita três faces, e a da direita pelo vértice comum às duas formas dos cubos.

tão ele pode ser considerado uma variedade de dimensão 2. A Figura 4.3 mostra alguns objetos simples que não são 2-manifolds.

Outra forma talvez mais simples, mas menos intuitiva de definir uma variedade de dimensão 2, seria dizer que são as formas que têm planos tangentes em todos os seus pontos, ou pelo menos, só não tem plano tangente em um número finito de pontos.

4.4.3. Representação por Faces Poligonais

Polígonos (das palavras gregas *polys*, que significa muitos, e *gonon*, que significa ângulo) são figuras planas fechadas formadas por muitos segmentos de retas e muitos ângulos. O polígono mais simples é o triângulo. Mas é possível construir polígonos com qualquer número de lados. Um polígono é chamado **regular** se tiver ângulos (e, portanto, lados) iguais. A técnica de **representação por faces poligonais** pode ser considerada um caso particular da anterior.

Chama-se *tessellation* a cobertura de uma área plana, como um piso, por repetições sem fim de uma forma sem deixar vazios (como fazer calçadas com pedras portuguesas). Ela oferece grandes possibilidades de criação [Glassner, 1998]. O artista holandês Maurits Cornelis Escher (1898-1972) soube usá-la com inteligência na criação de belas obras de arte (seção 1.1).

Uma questão interessante é: se quisermos cobrir uma área com polígonos regulares, quais são os polígonos que nos permitem fazer isso? Você já pensou sobre isso? Embora existam polígonos regulares com qualquer número de lados, só poderemos usar triângulos equiláteros, quadrados e hexágonos (Figura 4.4) para fazer *tessellation* (ou *tiling*) de um plano por formas regulares (essas são as únicas formas cujos ângulos internos multiplicados por um número inteiro resulta em um múltiplo de 360 graus.)

Quase todos os softwares de modelagem e *real-time rendering* (jogos e realidade virtual) utilizam a representação por faces triangulares. Essa representação é muito utili-



FIGURA 4.4. Tiling de uma superfície por triângulos, quadrados e hexágonos.

zada porque usa menos memória, menor tempo de *render* e se adapta a qualquer tipo de contorno. Além de ser automaticamente implementada por funções de OpenGL ou Java 3D.

Uma classe importante de objetos usados na representação de sólidos são os **poliédros** (das palavras gregas *polys*, que significa muitos, e *hedra*, que significa lados), ou seja, objetos compostos por muitos “lados”. Como os objetos da Figura 4.5.

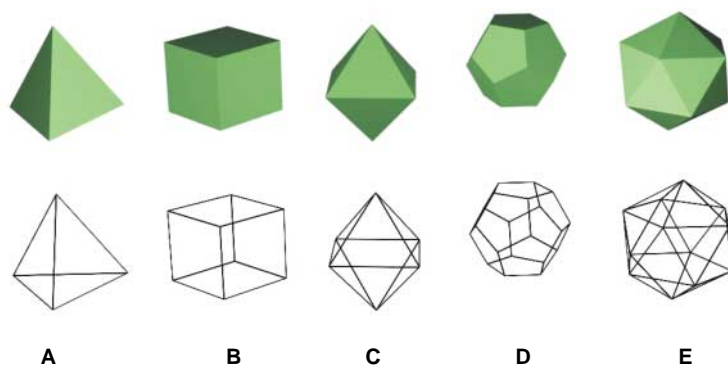


FIGURA 4.5. Os 5 tipos possíveis de poliédros regulares.

Mais precisamente, um **poliedro** é um sólido que é limitado por um conjunto de polígonos cujos lados (chamados arestas do poliedro) pertencem a um número par de polígonos. Para os poliedros, um objeto ser uma **variedade de dimensão 2** ou um **2-manifold** significa simplesmente que o número de polígonos que compartilham uma aresta deve ser 2. Ou, em outras palavras, se as arestas do objeto poliedral forem formadas pelos lados de dois polígonos, então esse objeto é uma variedade de dimensão 2 ou um **2-manifold**.

Se o poliedro for *topologicamente equivalente* a uma esfera, é classificado como um **poliedro simples**. Assim um poliedro simples é o que pode ser *deformável* em uma esfera, ou seja, não é equivalente topologicamente a um toro ou outras figuras com furos. E é composto de só um bloco (componente) ou uma única parte. Se um

objeto tem um único componente, o número dos furos que trespassam o objeto é denominado de *genus*, G , do objeto. Assim, um poliedro simples é um objeto de *genus* 0. A Figura 4.6 ilustra essa definição de *genus*.

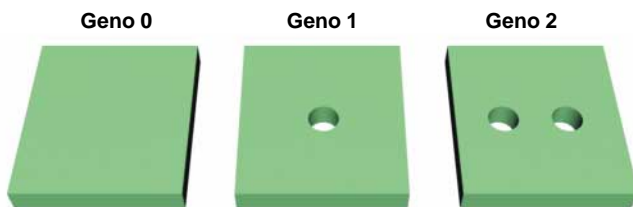


FIGURA 4.6. Exemplos de diversos *genus*.

4.4.3.1. Fórmula de Euler

A B-rep de poliedros simples, sem furos, satisfaz a fórmula proposta por Leonhard Euler em 1752:

$$V - A + F = 2$$

onde V representa o número de vértices do objeto, A é o número de arestas e F o número de faces. Euler era um matemático suíço, nasceu em 1707 na Basileia e morreu em 1783 em São Petersburgo. Foi aluno de Bernouilli, e a despeito de uma cegueira progressiva, produziu mais de 900 trabalhos de matemática, mecânica, astronomia, ótica, ciência naval e música [Struik, 1967]. Essa fórmula expressa uma relação invariante entre os componentes topológicos de um sólido simples realizáveis, que já era utilizada pelos gregos, embora de maneira intuitiva, sem uma prova definitiva.

A fórmula de Euler é facilmente verificada para objetos formados de faces planas. É também com facilidade incorporada nas implementações da formulação por contornos, pois sempre é simples o conhecimento do número de componentes das listas de vértices, arestas e faces.

Satisfazer a fórmula de Euler é uma condição necessária para um objeto ser um poliedro simples, mas não suficiente. Os poliedros são regulares se forem constituídos por faces e ângulos iguais. Embora existam infinitos polígonos regulares, há apenas cinco poliedros regulares, ou seja, só podem ser construídos poliedros regulares com 4, 6, 8, 12 ou 20 faces: o tetraedro, o hexaedro (cubo), o octaedro, o dodecaedro e o icosaedro (Figura 4.5). Todos muito importantes em cristalografia, pois os átomos dos cristais são arranjados segundo essas formas regulares. Por exemplo, os átomos do diamante ocupam os vértices de um tetraedro regular, o que lhe confere o alto grau de dureza. A possibilidade de existência de apenas cinco formas regulares já era conhecida desde a antiguidade grega, onde poliedros regulares eram conhecidos como sólidos Platônicos, e aos quais se atribuíam propriedades místicas.

A fórmula de Euler é aplicável mesmo a objetos que não tenham faces planas como o cilindro mostrado na Figura 4.7. Neste caso, o conceito de faces deve ser estendido para considerar toda as superfícies; as arestas devem ser entendidas como os limites entre as faces e os vértices definidos pelos limites das arestas. Assim, um cilindro pode ser considerado formado por: dois vértices, três arestas e três faces (veja a Figura 4.7). Uma esfera pode ser entendida a partir de um caso limite do cilindro, quando as faces planas e as arestas que as limitavam desaparecem e a aresta antes reta, vai se deformando até formar uma semicircunferência limitada pelos dois vértices, de modo que a figura passa a ser formada por uma única face, só uma aresta e dois vértices.

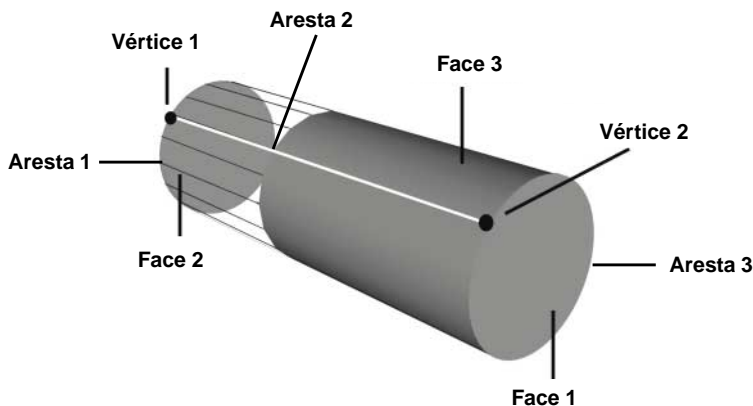


FIGURA 4.7. A fórmula de Euler é aplicável mesmo a objetos que não tenham faces planas como o cilindro.

Essa fórmula deve ser usada com cuidado também em outros casos simples. Por exemplo, como uma esfera é topologicamente equivalente a um elipsóide e esta a uma pirâmide e um cone, todos esses objetos devem satisfazê-la.

Para variedades de dimensão 2, que tenham furos, a fórmula de Euler é generalizada para fórmula de Euler-Poincaré:

$$V - A + F - H = 2 (C - G)$$

onde os primeiros três elementos têm os mesmos significados anteriores, e H é o número de loops internos fechados (ou quaisquer buracos) nas faces, C é o número de partes ou componentes separáveis do objeto, e G é o número de furos que trespassam o objeto (ou genus).

4.4.3.2. Operadores de Euler

As informações topológicas e geométricas devem ser armazenadas na estrutura de dados usada pelas B-rep. Ao se criar um objeto, deve-se, a partir de sua geometria,

assegurar a validade de sua topologia, essas informações servem como uma “cola” que mantém as informações geométricas consistentes. Assim, operações de mudança da geometria, pelo deslocamento da posição de um vértice, alteração do comprimento de uma aresta, ou mudança da área de uma face, são feitas facilmente sem alterar a consistência do sólido. Essas operações são chamadas de *tweaking operations* (operações de arrasto).

As operações de alteração topológicas são mais complexas. Para assegurar a validade topológica do modelo B-rep, alguns operadores especiais são usados para criar e manipular as entidades topológicas. Eles são chamados de operadores de Euler, e se aplicam a objetos que satisfazem a fórmula de Euler-Poincaré.

Esses operadores foram introduzidos por Baumgart em 1974, e operam nos objetos pela adição ou remoção dos elementos topológicos (vértices, arestas, faces) de modo a transformá-los em outros que também satisfaçam a fórmula de Euler-Poincaré. A prova de que qualquer B-rep válida pode ser construída por um número finito de operadores de Euler foi apresentada por Mäntylä em 1988.

4.4.3.3. Estrutura de Dados Baseada em Vértices

A estrutura de dados usada para armazenamento das informações é muito importante, pois permite mais rapidez e consistência nas operações geométricas e até a realização de operações topológicas (como subdivisão, criação ou agrupamento de faces, arestas e vértices). Uma boa discussão sobre elas é encontrada na obra de Mäntylä, já citada. Os modelos mais simples são para objetos de faces poligonais e se baseiam na definição da topologia das faces planas e arestas retas por uma lista de vértices, e uma lista de vértices que define a geometria dos objetos por suas coordenadas. Esta é a estrutura do denominado *modelo baseado em vértices*. Cuja única exigência é que a descrição das faces seja consistente: os vértices limites das faces descritos sempre no mesmo sentido horário (ou anti-horário) do exterior do objeto, para todas as faces. Nas tabelas abaixo tem-se um exemplo desta estrutura de dados, para os vértices do cubo mostrado na Figura 4.8A.

| Vértices | Coordenadas |
|----------|-------------|
| A | (0,0,0) |
| B | (1,0,0) |
| C | (1,1,0) |
| D | (0,1,0) |
| E | (0,0,1) |
| F | (1,0,1) |
| G | (1,1,1) |
| H | (0,1,1) |

| Faces | Vértices |
|-------|----------|
| F1 | EFBA |
| F2 | GFEH |
| F3 | CBFG |
| F4 | DABC |
| F5 | HEAD |
| F6 | DCGH |

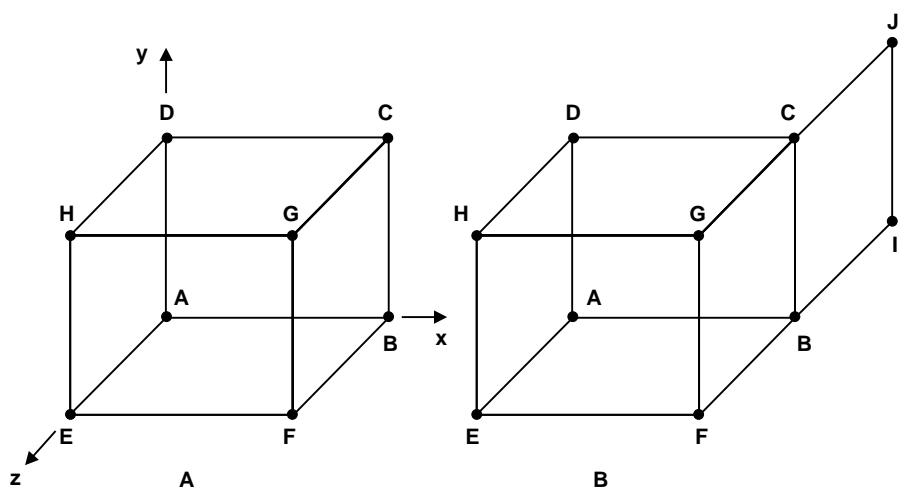


FIGURA 4.8. Exemplificação as diferentes estruturas de dados.

4.4.3.4. Estrutura de Dados Baseada em Arestas

No caso de uso de faces curvas no modelo B-rep, é útil uma estrutura de dados que permita a descrição das faces através de uma seqüência de arestas fechadas, ou de um loop de arestas. Essa descrição é denominada de estrutura de dados baseada em arestas. Nela, a ordem das arestas nas faces também deve ter sempre a mesma orientação e as próprias arestas são consideradas orientadas.

Na estrutura de dados baseada em arestas além das listas de coordenadas de vértices e definição das faces, tem-se uma lista que identifica cada aresta e seus vértices limitantes. A estrutura completa para o cubo da Figura 4.8A é composta das 3 tabelas que seguem.

| Vértices | Coordenadas |
|----------|-------------|
| A | (0,0,0) |
| B | (1,0,0) |
| C | (1,1,0) |
| D | (0,1,0) |
| E | (0,0,1) |
| F | (1,0,1) |
| G | (1,1,1) |
| H | (0,1,1) |

| Faces | Arestas |
|-------|----------------|
| F1 | A1 A2 A3 A4 |
| F2 | A9 A6 A1 A5 |
| F3 | A6 A10 A7 A2 |
| F4 | A7 A11 A8 A3 |
| F5 | A12 A5 A4 A8 |
| F6 | A9 A12 A11 A10 |

| Aresta | Vértices | Aresta | Vértices |
|--------|----------|--------|----------|
| A1 | EF | A7 | BC |
| A2 | FB | A8 | AD |
| A3 | BA | A9 | HG |
| A4 | AE | A10 | GC |
| A5 | EH | A11 | CD |
| A6 | FG | A12 | DH |

4.4.3.5. Estrutura de Dados Winged-Edge e Half Winged-Edge

Introduzida por Baumgart, a estrutura de dados Winged-Edge foi proposta para melhorar os algoritmos de remoção de superfícies escondidas e sombreado. Nesta estrutura, as informações das faces vizinhas e das próximas arestas são adicionadas aos dados das arestas.

Sabendo que cada aresta pertence a duas faces, podemos dizer que duas novas arestas podem descrever estas faces. Tomando como exemplo o cubo da Figura 4.8 A, podemos dizer que a aresta composta pelos vértices “EF” pertence a face EFBA e GFEH. Nesse caso, as arestas FB e HE podem representar essas duas faces. Por convenção, o identificador de sentido horário ncw (next clockwise) e o identificador de sentido anti-horário nccw (next counterclockwise) indicam as arestas que pertencem às faces vizinhas e sua orientação em relação à aresta comum as duas faces, neste exemplo “EF”. Devido a esta orientação consistente das faces cada aresta só ocorre uma vez no sentido positivo e uma vez no negativo.

A estrutura de dados Winged-Edge para o cubo da Figura 4.8 A será a mostrada nas tabelas que seguem:

| Vértices | Coordenadas |
|----------|-------------|
| A | (0,0,0) |
| B | (1,0,0) |
| C | (1,1,0) |
| D | (0,1,0) |
| E | (0,0,1) |
| F | (1,0,1) |
| G | (1,1,1) |
| H | (0,1,1) |

| Aresta | Vértices | Vértice Inicial | Vértice Final | ncw | nccw |
|--------|----------|-----------------|---------------|-----|------|
| A1 | EF | E | F | A2 | A5 |
| A2 | FB | F | B | A3 | A6 |
| A3 | BA | B | A | A4 | A7 |
| A4 | AE | A | E | A1 | A8 |
| A5 | EH | E | H | A9 | A4 |
| A6 | FG | F | G | A10 | A1 |
| A7 | BC | B | C | A11 | A2 |
| A8 | AD | A | D | A12 | A3 |
| A9 | HG | H | G | A6 | A12 |
| A10 | GC | G | C | A7 | A9 |
| A11 | CD | C | D | A8 | A10 |
| A12 | DH | D | H | A5 | A11 |

| Face | Primeira Aresta | Sinal |
|------|-----------------|-------|
| F1 | A1 | + |
| F2 | A9 | + |
| F3 | A6 | + |
| F4 | A7 | + |
| F5 | A12 | + |
| F6 | A9 | - |

4.4.3.6. Utilização de Operações Booleanas com B-rep.

As B-rep não têm representação única. Muitos sistemas são capazes de modelar apenas objetos B-rep que sejam variedades de dimensão 2. Esses, no entanto, não são fechados em relação às operações booleanas, isso quer dizer que nem sempre Operações Booleanas realizadas como variedades de dimensão 2 resultam em objetos que sejam variedades de dimensão 2. Weiler descreveu em 1988 um modelador que levanta essa restrição e pode ser usado para modelar até objetos non-manifold.

Um objeto 2-manifold, (veja seção 4.4.2) ou que pode ser considerado uma variedade de dimensão 2, também pode ser descrito como aquele que tem planos tangentes em todos os seus pontos de superfície, ou pelo menos só não tem plano tangente em um número finito de pontos. Um objeto 1-manifold, ou que pode ser considerado uma variedade de dimensão 1, é um objeto unidimensional que tem uma reta tangente em todos os seus pontos, ou pelo menos só não tem tangente em um número finito de pontos. Um objeto nonmanifold é o que não tem qualquer tangente.

Em termos de vizinhança, um objeto é um manifold de dimensão 2, ou uma variedade de dimensão 2, se cada um dos pontos de seu contorno tiver uma vizinhança que seja topologicamente equivalente a um disco plano. Um objeto unidimensional é um manifold de dimensão 1, ou uma variedade de dimensão 1, se cada um dos pontos de seu contorno tiver uma vizinhança que seja topologicamente equivalente a um intervalo na reta.

4.4.4. Representação por Enumeração da Ocupação Espacial

Esta representação decompõe o sólido em “pedaços” (Figura 4.9). Como se estivéssemos fazendo uma *tessellation* do espaço. Um aspecto interessante é: se quisermos fazer um “tile” do espaço por sólidos regulares (os poliedros vistos anteriormente), dos cinco tipos de sólidos regulares existentes (Figura 4.5) apenas um permite o preenchimento total do espaço por repetições infinitas dele mesmo: o cubo. Assim, na representação por Enumeração da Ocupação Espacial, o espaço é subdividido em cubos que são chamados de *voxels*, formando uma grade tridimensional. Isso pode ser considerado uma generalização da descrição de uma imagem 2D através de uma matriz de zeros e uns. Através das regiões que o objeto ocupa nesse espaço, podemos obter diversas informações de suas propriedades.

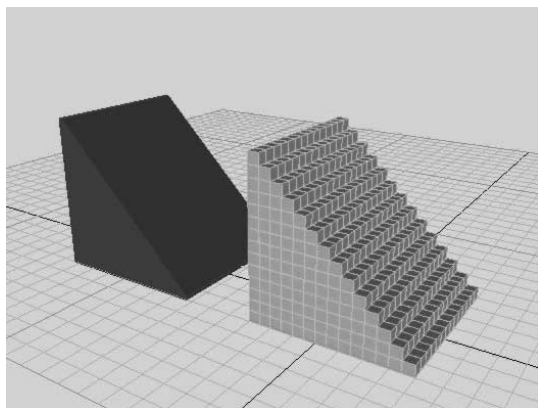


FIGURA 4.9. Exemplificando a representação por ocupação espacial: o sólido a ser descrito é representado por elementos de volume chamados voxels.

Essa representação tem algumas características interessantes:

- Para determinar se um dado ponto pertence ou não ao sólido, basta verificar se o ponto pertence a algum dos *voxels*;
- É fácil determinar se dois objetos se interferem;
- É fácil a realização de operações booleanas, como união, diferença e intersecção entre sólidos;

- É fácil a obtenção da propriedade de massa e volume do objeto, bastando saber o volume de uma das partes e multiplicar pelo número total de divisões ocupadas.

A desvantagem dessa representação é que na representação de objetos complexos e muito detalhados é necessária uma grande quantidade de memória disponível. A modelagem por voxel foi usada em alguns jogos, mas foi abandonada devido ao seu alto custo para armazenagem e representação realística (render).

4.4.5. Representação por Decomposição do Espaço em Octrees

Essa representação pode ser considerada um caso particular de Subdivisão Espacial. A técnica de representação por octree (ou árvore com oito filhos) envolve o objeto por um cubo, que, em seguida, é dividido em oito cubos menores de igual tamanho (octantes). Cada um destes é então classificado em:

- **Cheio**, caso o objeto ocupe todo o cubo em classificação;
- **Vazio**, caso o objeto não ocupe nenhuma parte do cubo; ou
- **Cheio-Vazio**, caso o objeto ocupe parte do cubo.

Quando um octante for classificado em “Cheio-Vazio”, ele é novamente dividido em oito partes iguais e o processo de classificação é refeito para as novas partes. Este algoritmo repete-se até que todos os objetos pertençam às duas primeiras classes (Figura 4.10). Neste caso, os voxels passam a ser “cubos” (paralelepípedos) de dimensões variáveis.

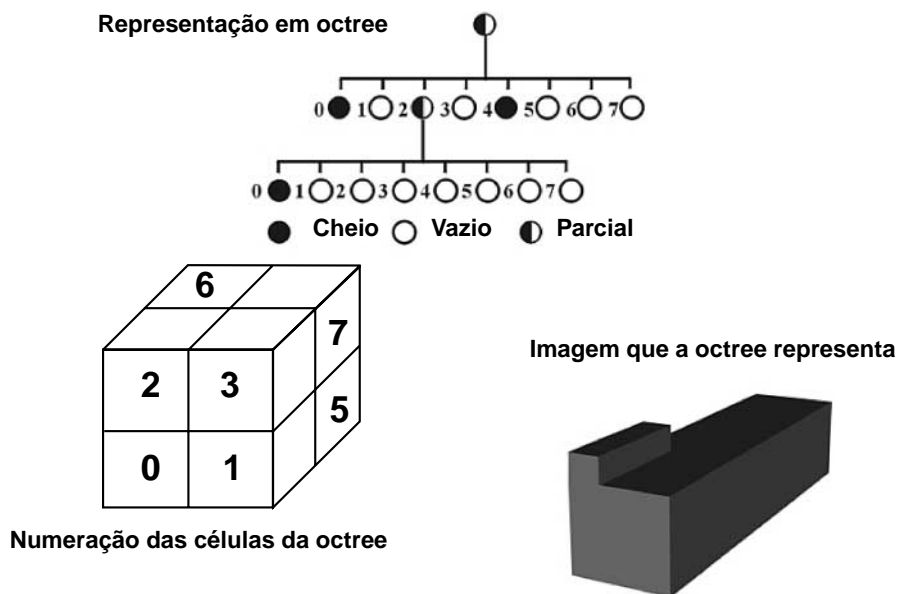


FIGURA 4.10. Representação por Octree.

Essa representação facilita as operações de união e interseção. No caso da união de dois objetos representados por octree, se em determinado nível um deles estiver **cheio**, o resultado da operação será um objeto cheio nesse nível, não importando qual a forma em que o mesmo nível do outro esteja. No caso da interseção de dois objetos representados por octree, se em determinado nível um deles estiver **vazio**, o resultado da operação será um objeto vazio nesse nível, não importando qual a forma em que o mesmo nível do outro esteja. Operação de rotação de 90 graus em torno de qualquer um dos três eixos é também facilmente implementada diretamente na estrutura da octree.

4.4.6. Representação por Decomposição do Espaço em Quadrees

Para o armazenamento de objetos 2D ou 3D com espessura constante, usa-se as Quadrees. Essas estruturas de armazenamento dividem o plano onde está o objeto em quatro partes iguais e classificam cada parte da mesma forma semelhante as *octrees* (Figura 4.11).

Essa representação tem as mesmas vantagens e desvantagens da Enumeração Espacial e das octrees, além de permitirem uma representação mais detalhada que as octrees, com um gasto menor de memória.

4.4.7. Quadrees e Octrees Lineares

Embora pareça à primeira vista necessário o uso de ponteiros nas estruturas de árvore octree e quadtree, é também possível o uso de uma notação sem ponteiros.

Uma notação de octree e quadtree sem ponteiros e na forma de um endereço até cada nó do objeto é chamada linear. Na notação linear, cada nó completamente cheio é representado como uma seqüência de dígitos que representam seus endereços através da árvore. Existem tantos dígitos quanto níveis de subdivisão da árvore. Nós cheios que não estão no nível mais baixo podem ser representados por um dígito adicional indicando o fim do endereço como “X”, por exemplo. Cada nó completamente cheio é representado como uma seqüência que identifica sua posição de 0 a 3, no caso das quadrees, ou de 0 a 7 para as octrees. Cada endereço terá tantos dígitos quantos forem os níveis usados na representação. Nós que não necessitam de níveis adicionais recebem o caractere especial.

A representação linear da Figura 4.11 tem, no máximo, 4 níveis de divisão, de modo que os endereços terão no máximo quatro dígitos. Se for considerada a numeração dos quadrantes como:

| | |
|---|---|
| 0 | 1 |
| 2 | 3 |

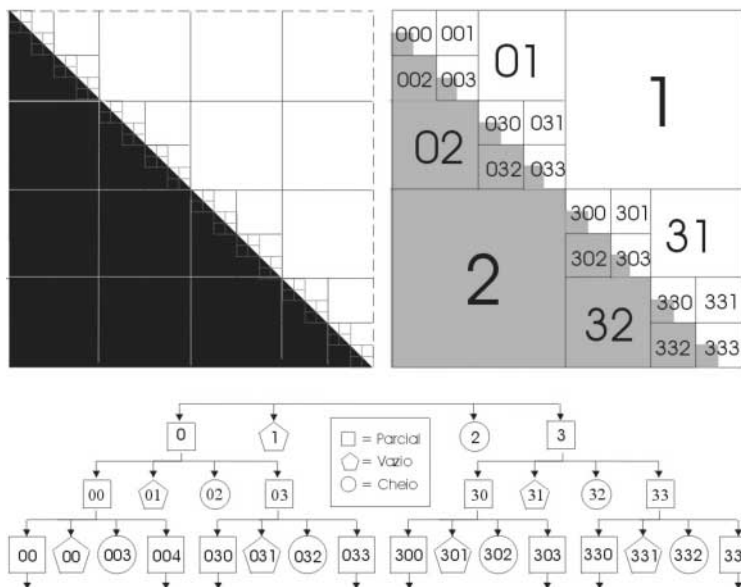


FIGURA 4.11. Representação por Quadrees do sólido da Figura 4.9.

O endereço 1 é vazio em todos os níveis de divisão, de modo que não haverá nenhum endereço iniciado por 1. O oposto ocorre com o endereço 2, que já está completamente cheio na primeira divisão da figura em quatro quadrantes, isso é indicado por 2XXX. Já os endereços 0 e 3 têm maior nível de divisão. A primeira posição totalmente cheia do endereço 0 é 0002. A figura inteira pode ser representada como:

0002 002X 0032 X 02X 0302 032X 0332 2X 3002 302X 3032 32X 3302 332X 3332

A representação linear da árvore octree da Figura 4.10 seria:

0X X 20X X 4X X X X

Nessa notação, todas as operações realizadas com a forma usual podem também ser implementadas.

Outra notação linear é a Depth-First ou DF, que é obtida através de uma verificação prévia na árvore [Noborio et al., 1988]. Nesta, o símbolo “(” representa um nó parcialmente cheio, e um incremento de um nível na árvore, e o símbolo “)” representa o decremento de um nível na árvore. Os símbolos 0 e 1 representam nós cheios e vazios (ou vice-versa). Essa notação é útil para um armazenamento condensado da octree ou quadtree. Para a Figura 4.11 essa notação seria:

((((0010)01(0010)01((0010)01(0010)))01(((0010)01(0010)01((0010)01(0010))))

4.4.8. Quadtrees e Octrees Híbridas

Diversos pesquisadores desenvolveram formas de combinar as octrees e as B-reps, de modo a manter a geometria e a consistência das representações. Uma dessas formas híbridas são as Polygonal Map octrees ou PM octrees, onde os nós folhas finais podem ser de cinco tipos: além dos dois normais (cheio, vazio) são possíveis três outras classes de parcialmente cheio: vértices, arestas e faces.

4.4.9. Representação por Partição Binária do Espaço (Binary Space-Partitioning – BSP)

As estruturas de árvores BSP, que representam uma partição binária do espaço, foram inicialmente propostas para solucionar os problemas de superfícies ocultas em aplicações onde o observador móvel desloca-se por uma cena estática. Um exemplo clássico dessa situação ocorre nos simuladores de voo e games [Watt e Policarpo, 2003].

As árvores BSP dividem o espaço em pares de subespaços, cada um separado do outro por planos de orientação e posição arbitrária (Figura 4.12). Cada nó não terminal na árvore BSP representa um único plano que divide o espaço ocupado em dois. Um nó terminal representa uma região que não será mais subdividida e contém ponteiros para representações da estrutura de dados dos objetos cruzando aquela região. Cada nó interno da árvore BSP é associado a um plano e tem dois filhos, um para cada lado do plano. Assumindo que as normais da Figura 4.12 apontam para fora do objeto, o filho da esquerda estará dentro do plano, enquanto o da direita estará fora do plano.

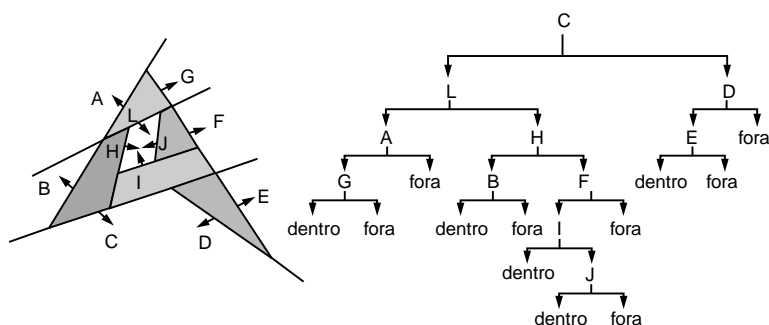


FIGURA 4.12. Uma figura 2D e sua representação em árvore binária.

Como as octrees e as quadrees, essa representação também pode ter uma forma linear de descrição dos objetos representados.

4.4.10. Representação Implícita

Em muitas situações, as expressões matemáticas implícitas dos sólidos modelados podem ser armazenadas e usadas para a modelagem. Podendo mesmo depois serem combinadas com outras formas de representação, como, por exemplo, para gerar novos objetos por combinações, por união de objetos (CSG) ou serem usadas em representação pelo contorno (B-rep).

Essa forma tira proveito do fato de que muitas formas Euclidianas têm equações perfeitamente conhecidas e simples (veja capítulo 3 de superfícies e curvas). Assim, se os sólidos forem esferas, elipsóides, cilindros elípticos e diversos outros, usar as equações implícitas dessas formas é uma forma simples de armazená-los, operá-los e mesmo renderizá-los eficientemente.

Assim, para representar um cilindro elíptico, pode-se usar sua equação:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} \leq 1$$

onde a e b são os raios nas direções x e y da base elíptica. Se $a=b=r$ o sólido torna-se um cilindro circular de raio r .

E para representar um elipsóide, tem-se:

$$\frac{(x - x_0)^2}{a^2} + \frac{(y - y_0)^2}{b^2} + \frac{(z - z_0)^2}{c^2} \leq 1$$

onde (x_0, y_0, z_0) é a posição do centro, a , b e c são os raios nas direções x , y e z do elipsóide. Se $a=b=c=r$ o sólido torna-se uma esfera de raio r .

4.5. TÉCNICAS DE MODELAGEM GEOMÉTRICA

Basicamente, podemos dividir as técnicas de modelagem em três formas: modelagem manual, automática ou matemática.

O método matemático de modelagem usa uma descrição matemática e algoritmos para gerar um objeto. Exemplos desse método são a modelagem por varredura e a modelagem fractal, descritas a seguir. O método matemático vem sendo usado para modelar efeitos naturais como turbulência ou proliferação de organismos microscópicos.

A modelagem automática é sem dúvida a mais sofisticada, porém, a mais rápida e poderosa. Através de equipamentos especiais como scanners 3D, podemos obter o modelo tridimensional de quase tudo. Esse método tem encontrado diversas novas aplicações para a computação gráfica, como o projeto T-Rex, onde o modelo tridimensional de uma auto-estrada é usado para inspeções. Outro exemplo do uso desse método é o projeto Michelangelo (*The Digital Michelangelo Project: 3D Scanning of Large Statues* – <http://graphics.stanford.edu/papers/dmich-sigoo>), onde busca-se fazer um arquivo tridimensional das obras desse gênio da renascença.

A modelagem manual é sem dúvida o método mais fácil, barato e antigo que utiliza basicamente as medidas de um modelo real e a intuição do modelador. A forma manual foi inicialmente usada pela indústria automobilística e aeronáutica para a concepção e teste de novos modelos. A seguir descrevemos uma série de técnicas de modelagem manual.

4.5.1. Instanciamento de Primitivas

Em instanciamento de primitivas, o sistema de modelagem define um conjunto de formas sólidas primitivas que são relevantes à área de aplicação. Esta técnica de modelagem cria novos objetos através do posicionamento de objetos por transformações geométricas (mudanças de escala, rotação, translação etc.) ou pelo uso de primitivas parametrizáveis. No primeiro caso, Figura 4.13, uma cadeira pode ser modelada pela justaposição de paralelepípedos. No segundo caso, cria-se um conjunto de peças geralmente complexas e de uso comum para um determinado fim, por exemplo, engrenagens ou parafusos, e, a partir dessas primitivas, podemos criar uma infinidade de variações desses objetos com apenas alguns comandos para alteração de seus parâmetros (Figura 4.14), como mudança de alturas e diâmetros.

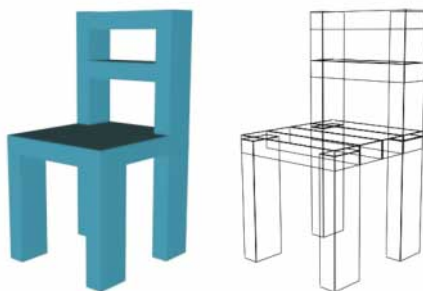


FIGURA 4.13. Modelagem pela justaposição de paralelepípedos.

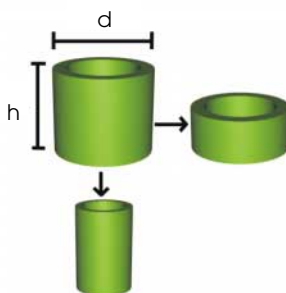


FIGURA 4.14. Alteração dos parâmetros de primitivas instanciadas.

4.5.2. Combinação de Objetos

A habilidade de combinar objetos para criar outros é sem dúvida o método mais intuitivo e popular. Essa combinação é na sua forma mais simples feita por justaposição ou colagem de formas como mostrado na Figura 4.13.

As operações booleanas de união (soma), interseção e diferença, que são denotados por: \cup , \cap e $-$, são outra forma simples de combinar objetos. Entretanto, aplicar operações booleanas ordinárias entre duas representações de sólidos **pode não gerar uma outra representação válida** como sólido, como podemos ver nos casos (B), (C) e (D) da Figura 4.15, onde a interseção ordinária entre dois cubos pode gerar: (A) um sólido (forma válida), (B) um plano (não válida), (C) uma linha (não válida), (D) um ponto (não válida), ou (E) um conjunto vazio.

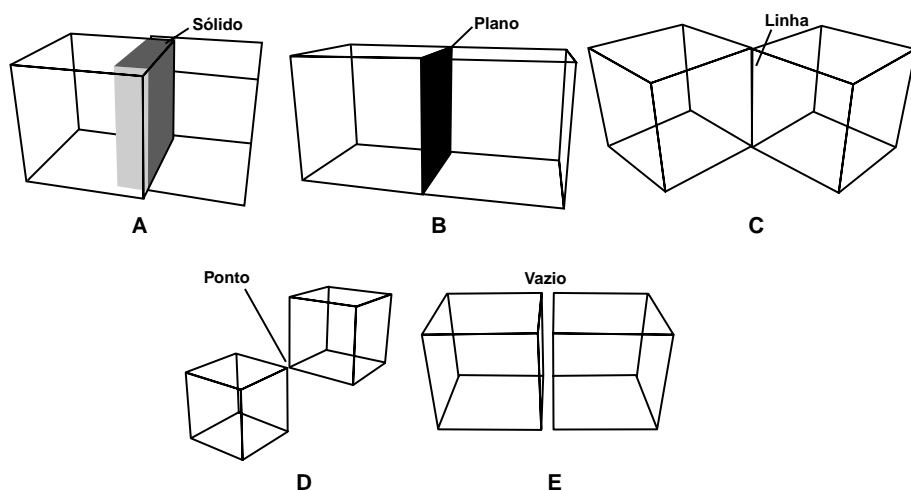


FIGURA 4.15. Resultados das operações booleanas em dois cubos.

4.5.3. Operações Booleanas e Booleanas Regularizadas

Aplicar um conjunto de operações booleanas comuns a dois objetos sólidos, como já comentado, nem sempre gera um objeto sólido. Por exemplo, a interseção entre dois cubos: se eles se “penetram” mutuamente resulta um sólido; se eles compartilham apenas uma face, o resultado é um plano; se compartilham apenas uma aresta, o resultado é uma reta; se compartilham apenas um dos vértices, o resultado é um ponto. Na Figura 4.15.B, os vértices da interseção dos cubos ocupam a mesma posição no espaço resultando em um plano. Para solucionar o problema, foi criado o conceito de Operações Booleanas Regularizadas, cuja notação é \cup^* , \cap^* e $-^*$.

As operações desse tipo são realizadas em um tipo de objeto chamado objeto regularizado utilizando operadores booleanos como união, diferença e interseção para formar outros objetos também regularizados.

FIGURA 4.16. Regularizando um objeto. (A) Objeto qualquer com pontos interiores (em cinza), pontos de contorno que fazem parte do objeto (em preto) e outros pontos (em cinza e preto). (B) Interior do objeto: retira-se todos os pontos de contorno. (C) Regularização do objeto é o fechamento de seu interior.

Um objeto qualquer, regularizado ou não, pode ser definido como um conjunto de pontos divididos em pontos do interior e pontos do contorno. Pontos do contorno são aqueles cuja distância entre o objeto e o seu complemento é zero, enquanto pontos do interior são todos os outros pontos. Um objeto regularizado é obtido pelo o fechamento do conjunto de seus pontos interiores. Isto é, são regularizados os objetos que sempre têm todos os pontos de seu contorno. A Figura 4.16 mostra isso.

Novamente uma definição formal simples é encontrada em Mäntylä (1988):

Definição: A regularização de um conjunto de pontos A é denotada por $r(A)$ e definida como o fechamento do interior de A . Diz-se que A é um conjunto regular se $r(A)=A$.

As operações regularizadas são definidas pelos operadores booleanos usuais, como mostrado na tabela a seguir, onde A e B são dois objetos quaisquer; o subscrito “i” indica os pontos do interior do conjunto, “c” indica os pontos do contorno, “igual” se refere ao caso dos interiores estarem no mesmo lado dos contornos em análise e “difer” o caso de estarem em lados opostos.

| Set | $A \cup B^*$ | $A \cap B^*$ | $A - B^*$ |
|----------------|--------------|--------------|-----------|
| $A_i \cap B_i$ | • | • | |
| $A_i - B$ | • | | • |
| $B_i - A$ | • | | |

| Set | $A \cup B^*$ | $A \cap B^*$ | $A - B^*$ |
|----------------------|--------------|--------------|-----------|
| $A_c \cap B_i$ | | • | |
| $B_c \cap A_i$ | | • | • |
| $A_c - B$ | • | | • |
| $B_c - A$ | • | | |
| $A_c \cap B_c$ igual | • | • | |
| $A_c \cap B_c$ difer | | | • |

4.5.4. Geometria Sólida Construtiva (CSG-Constructive Solid Geometry)

O método de modelagem por CSG usa um esquema de representação de sólidos através de operações booleanas ou combinações de objetos sólidos a partir de operações de conjuntos (união, interseção e diferença).

Nesse caso, o objeto é armazenado como uma árvore de operadores (nós) e primitivas simples. Alguns nós representam operadores booleanos, e outros representam translação, rotação, escala etc.

No exemplo da Figura 4.17, é possível observar a realização de duas operações para a construção de objetos ou superfície. Essa técnica é muito utilizada tanto por designers como por engenheiros para modelagem de objetos ou peças mecânicas.

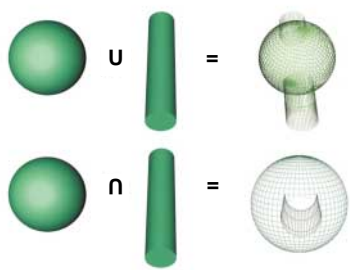


FIGURA 4.17. Construção de objetos por operações booleanas.

4.5.5. Connect

Connect é uma forma de criar novos sólidos a partir de sólidos já existentes combinados por uma conexão. A operação de combinação Connect deixa você conectar dois ou mais objetos preenchendo espaços em suas superfícies. Para fazer isso, você apaga faces em cada objeto para criar um ou mais vazios em suas superfícies, posionando-os de forma que as faces dos vazios fiquem de frente uma para outra. A criação por connect não funciona bem com objetos criados por NURBs, pois eles se convertem em

muitas superfícies separadas em vez de uma superfície única. No exemplo da Figura 4.18, o encosto da cadeira é criado (a partir da Figura 4.13) usando connect.



FIGURA 4.18. Encosto da cadeira recriado com o objeto de combinação Connect.

4.5.6. Modelagem por Varredura (Sweep) ou Deslizamento

A representação por varredura cria objetos baseados na noção de que uma curva C1, quando deslocada no espaço, ao longo de uma trajetória dada por uma outra curva C2, descreve uma superfície que pode ser usada para definir um sólido. A curva C1 recebe o nome de Contorno ou Geratriz e a C2 tem o nome de Caminho ou Diretriz. Na forma mais geral é possível ainda variar a orientação relativa entre as curvas durante o processo [Watt e Policorpo, 1988].

4.5.6.1. Varredura Translacional (Extrusão)

Um objeto “O” definido por varredura translacional é obtido pela translação por uma distância D, de uma superfície C, ao longo de um vetor V. A Figura 4.19 exemplifica a criação de um objeto com essa técnica (veja também a seção 3.2.2). A varredura translacional de um retângulo gera um paralelepípedo, de uma circunferência gera um cilindro. Podemos ainda aplicar a varredura translacional em faces de um objeto

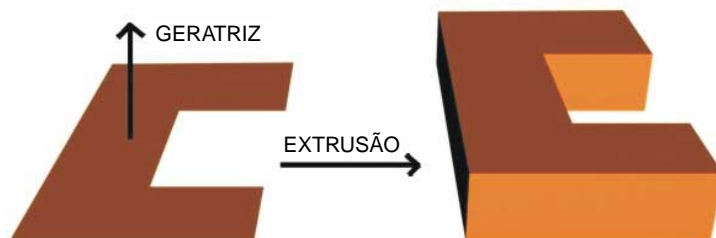


FIGURA 4.19. Modelagem por varredura translacional.

3D. Apesar de ser considerada uma técnica de modelagem, a varredura translacional é considerada por diversos autores e sistemas como um modificador (seção 4.6).

4.5.6.2. Varredura Rotacional

Neste tipo de modelagem por varredura, a superfície do objeto é descrita por uma supercúria ou curva que gira em torno de um eixo (seção 3.2.1). Na Figura 4.20 temos, à esquerda, uma curva C e, à direita, dois objetos gerados pela rotação de C em torno de dois eixos verticais diferentes. Diversas são as formas possíveis de criação usando esse método. Alguns sistemas de modelagem atribuem o nome *Lathe* (torno mecânico) a esse método.



FIGURA 4.20. Modelagem por Varredura Rotacional.

4.5.7. Modelagem por Seções Transversais

Esta técnica permite gerar sólidos por reconstrução através de cortes. A ideia básica é interpolar os dados (seção 3.2.3) das seções transversais do objeto que se deseja modelar. As fatias podem ser obtidas pelas leituras de cortes do objeto ou por scanners em processos como tomografia, ultrassom e ressonância magnética [Duncan e Ayache, 2000]. Essa técnica de modelagem foi utilizada no maior projeto de modelagem do corpo humano, o projeto Visible Human. Para realizar esse projeto, o corpo de um homem real foi fatiado e digitalizado. A partir das imagens, o modelo tridimensional foi reconstituído e atlas 3D de anatomia foram gerados (<http://www.spl.harvard.edu> [Treinish e Silver, 1996]).

4.5.8. Modelagem pela Geração de Superfícies

O princípio deste processo é mostrado na Figura 4.21. Iniciamos colocando alguns pontos no espaço (A). O próximo passo será colocar uma curva ligando os pontos nas direções x e y de um espaço bidimensional (B). As curvas são então particiona-

das em seções de quadriláteros curvilíneos (patches) (C) e preenchidas. Podemos então alterar a forma do objeto com simples deslocamentos dos pontos (A).

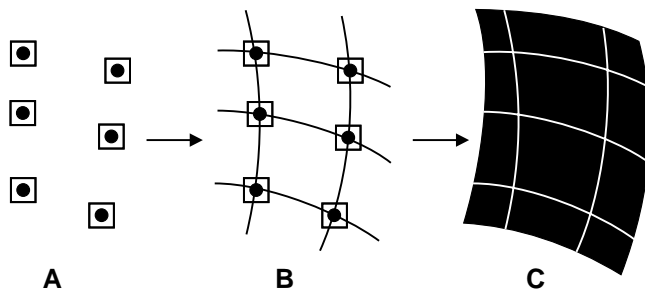


FIGURA 4.21. Operador de Malha Lattice.

Esse método (que não deixa de ser uma forma de B-rep) é muito utilizado quando desejamos modelar objetos com formas arredondadas e complexas, como partes do corpo humano ou animal, veículos ou peças. Obviamente a relação entre o número de pontos necessários para definir e controlar a superfície gerada depende da formulação de geração de superfície usada (seção 3.2.5).

4.6. MODIFICADORES

Modificadores, como o próprio nome diz, modificam a estrutura geométrica do objeto. Todos os sistemas de modelagem utilizam esses recursos para facilitar e agilizar a modelagem (e a animação). Sobre eles, é importante saber que:

- É possível aplicar um número ilimitado de modificadores para um objeto ou parte de um objeto;
- É possível alterar os parâmetros de modificação para realizar uma animação;
- Os modificadores podem ser retirados e todas as suas mudanças para o objeto desaparecem;
- É possível reposicionar e copiar modificadores para outros objetos usando controles específicos;
- A sequência em que você faz modificações é importante. Cada modificação afeta aquela que vem depois. Duas modificações em um mesmo objeto aplicadas em ordem inversa geram resultados diferentes.

Todos os sistemas de modelagem possuem modificadores para auxiliar na tarefa de modelagem e animação. Seria quase impossível ou muito custoso (tempo é dinheiro) realizar determinadas alterações na geometria do objeto sem o uso deles. Os modificadores que iremos mencionar podem ser considerados como genéricos para todos os sistemas 3D.

4.6.1. Bend

O modificador Bend deixa você curvar ou fletir a seleção corrente até 360 graus sobre um eixo único e em várias direções (Figura 4.22), produzindo uma curva uniforme na geometria do objeto.



FIGURA 4.22. O modificador de curvatura Bend.

4.6.2. Lattice

O modificador Lattice converte os segmentos ou extremidades de um objeto em uma estruturas de barras. É usado para criar geometria estrutural baseada na topologia ou como um método alternativo para alcançar um efeito de wireframe do contorno (Figura 4.23).

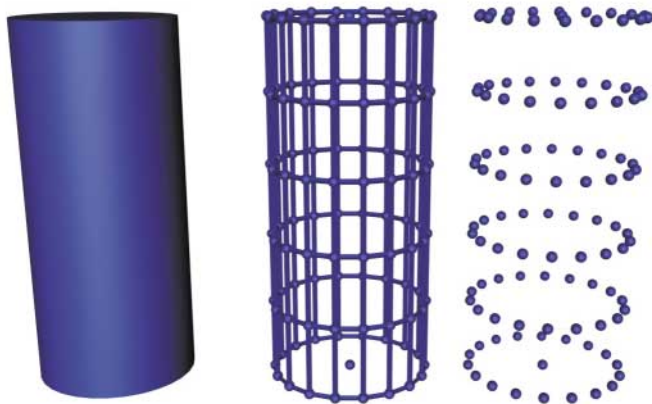


FIGURA 4.23. O modificador Lattice.

4.6.3. MeshSmooth

Um dos modificadores mais interessantes, o meshsmooth, é baseado nas regras de subdivisão que suavizam a geometria do objeto adicionando faces em cantos e ao longo de extremidades. Em alguns sistemas, esse modificador é também chamado de NURMs onde recebe recursos adicionais para facilitar a modelagem orgânica ou de objetos com extremidades suaves (seção 3.2.13), como a bola da Figura 4.24 (a bola à direita recebeu o modificador). Quanto maior for o número de faces, maior será o efeito de suavização. Esse modificador pode não ser recomendável para modelagem de objetos que serão usados em sistemas real-time rendering, como os jogos ou aplicações em realidade virtual, pois aumentam o número de polígonos (uma alternativa neste caso pode ser o uso de Bump Map – seção 7.4).

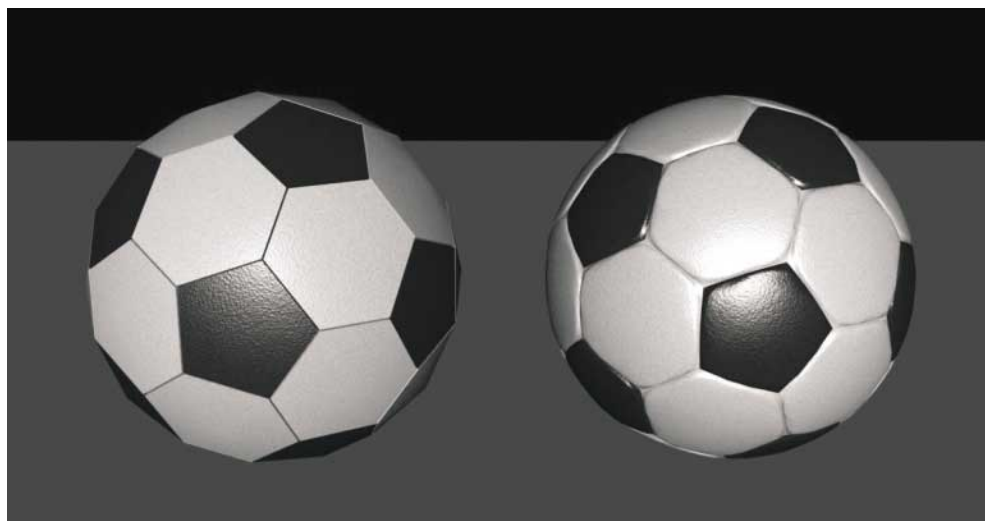


FIGURA 4.24. Suavização da geometria com o modificador MeshSmooth.

4.6.4. Optimize

O modificador Optimize deixa você reduzir progressivamente (em regiões de pouca curvatura por exemplo) o número de faces e vértices em um objeto buscando reduzir o tempo de render, uma vez que o número de polígonos será menor (Figura 4.25). Esse modificador é quase obrigatório para modelagem de objetos que serão usados em sistemas real-time rendering.

4.6.5. Bevel

Este modificador realiza a extrusão de objetos 2D para 3D e aplica um arredondamento nos cantos das extremidades. Pode ser considerado como uma extensão da

técnica de extrusão. O uso comum desse modificador é para criar texto 3D e logos, mas você pode aplicar este modificador em qualquer forma (Figura 4.26).

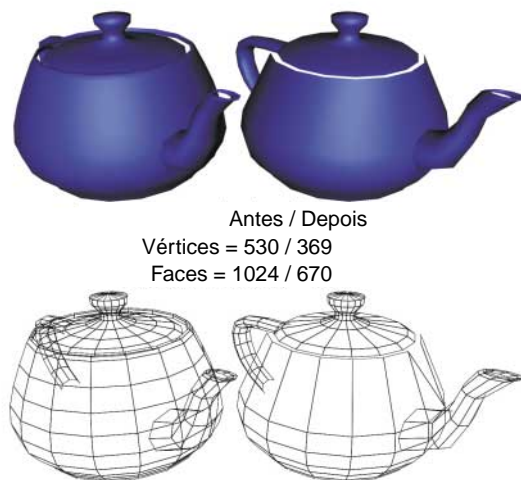


FIGURA 4.25. Resultado da aplicação do modificador Optimize.



FIGURA 4.26. Comparação entre os modificadores Extrude e Bevel.

4.6.6. Melt

O modificador Melt deixa você aplicar um efeito de “derretimento” realista a todos os tipos de objetos, inclusive objetos NURBS. Os efeitos de derretimento podem ser feitos de várias formas que por serem transparentes aos usuários são referenciadas como: seguir a forma de gelo, plástico, vidro ou geléia (Figura 4.27).

4.6.7. Skew

O modificador Skew em objetos 3D tem o mesmo efeito do operador skew 2D (seção 2.5.5). Deixa você produzir um deslocamento uniforme em qualquer parte da geometria do objeto. Você pode controlar a quantidade e direção da distorção em quaisquer dos três eixos (Figura 4.28).



FIGURA 4.27. Operador Melt aplicado em um objeto.

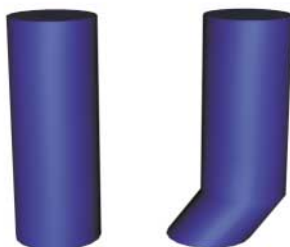


FIGURA 4.28. O modificador de cisalhamento Skew.

4.6.8. Squeeze e Stretch

O modificador Squeeze deixa você aplicar um efeito de apertar ou espremer de modo que os vértices mais próximos do ponto de pivô do objeto são reposicionados para dentro. O Stretch produz um estiramento no objeto, simula o tradicional efeito de “espreme-e-estica” em animação (Figura 4.29).

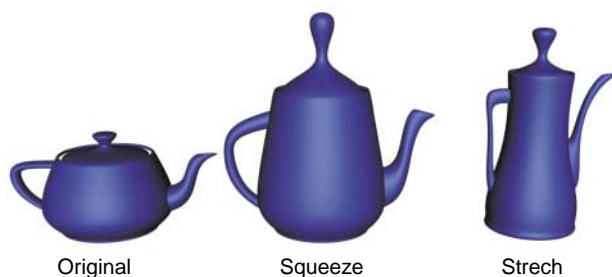


FIGURA 4.29. O efeito de “espreme-e-estica” com os modificadores Squeeze e Stretch.

4.6.9. Taper

O modificador Taper produz um contorno mais ou menos afilado, ajustando as escalas dos pontos terminais da geometria do objeto (Figura 4.30).



FIGURA 4.30. *Redução e aumento de espessura com o modificador Taper.*

4.6.10. Twist

O modificador Twist produz uma torção ou retorcido na geometria do objeto. Você pode controlar o ângulo da torção em quaisquer das três direções. Você também pode limitar a torção para uma seção da geometria (Figura 4.31).

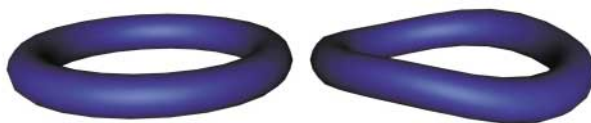


FIGURA 4.31. *Torção na geometria do objeto com o modificador Twist.*

4.6.11. Displace

Esse modificador atua como um “campo de força” para empurrar e alterar a geometria do objeto. Podemos utilizar um bitmap para moldar uma superfície (Figura 4.32) ou ajustar os parâmetros do modificador para produzir imagens parecidas com pegadas nos solos.



FIGURA 4.32. *O modificador Displace desloca uma superfície utilizando um bitmap.*

4.6.12. Space Warps

Space Warps são objetos não renderizáveis que afetam a aparência de outros objetos. Eles criam “campos de força” que deformam outros objetos, criando o efeito de empenamento, ondulações, explosões, vento, colisões e assim por diante.

Quando você cria um objeto Space Warps nos sistemas de modelagem, visores mostram uma representação em wireframe para que o usuário possa ter controle. Os sistemas de modelagem disponibilizam uma grande variedade desses objetos que são muito úteis na animação.

Os Space Warps podem trabalhar em conjunto com os sistemas de partículas e servir para propósitos especiais em simulações dinâmicas.

4.7. INTERFACES PARA MODELAGEM GEOMÉTRICA

As interfaces dos sistemas de modelagem são independentes da representação interna utilizada. Você pode alternar entre as diferentes formas de representação buscando o resultado pretendido. A interface pode fornecer diversas maneiras de definir o objeto dentro da mesma representação. Por exemplo, você pode definir um cubo arrastando o mouse para definir a largura, o comprimento e a altura; ou por teclado, informando a largura e o comprimento e em seguida aplicar uma extrusão para definir a altura. A primeira maneira pode ser mais rápida, enquanto a segunda pode ser mais precisa ou fácil de posicionar na cena.

Um dos problemas está na limitação tecnológica que nos fará usuários, por ainda muito tempo, de telas de computador que, mesmo colocadas lado a lado para possibilitar uma visão estéreo, será a projeção de duas imagens bidimensionais. Muitos sistemas combatem alguns desses problemas fornecendo várias janelas de exibição, que permitem ao usuário ver o objeto simultaneamente de várias posições e perspectivas.

Um bom sistema de modelagem deve possuir ferramentas básicas de auxílio a geração de objetos. Essas ferramentas são geralmente chamadas de *Helpers* e fornecem informações relevantes para uma modelagem precisa ou auxiliam na criação eficiente de objetos. Um bom sistema deve conter, no mínimo, as seguintes ferramentas:

4.7.1. Tape (Fita de Medida)

A fita provê na tela uma régua auxiliar, no sistema métrico desejado (metro, polegada etc.), para determinar e configurar distâncias.

4.7.2. Protractor (Transferidor)

O transferidor permite medir o ângulo entre dois objetos em uma cena.

4.7.3. Compass (Bússola)

A bússola é um objeto que indica o Norte, Sul, Leste, e Oeste da cena. Uma bússola é parte de um sistema de luz solar. Em um sistema de luz solar, a orientação da bússola indica a orientação da cena, relativa ao caminho do sol, sendo importante para o seu realismo.

4.7.4. Array(Arranjo)

O array cria um arranjo (vetor ou matriz) de objetos com uma, duas ou três dimensões a partir de um objeto selecionado. Nesse caso, o termo dimensão se refere à ordenação dos objetos. Por exemplo, uma linha de cinco objetos é um vetor de dimensão um, embora saibamos que ocupará um espaço tridimensional na cena. Um vetor de objetos que tem cinco linhas por três colunas é um array bidimensional, e um vetor de objetos que tem cinco linhas por três colunas por dois níveis é um array tridimensional. O array é uma poderosa ferramenta que auxilia na criação e posição de objetos complexos. Dentre os exemplos de seu uso podemos citar a criação de escadas, pétalas de flores ou hélices (Figura 4.33).

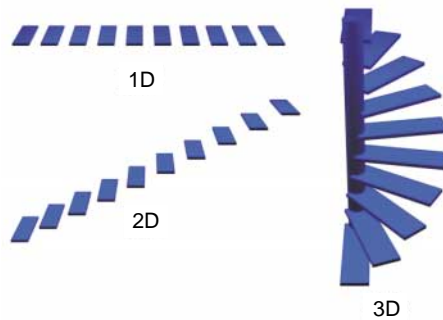


FIGURA 4.33. Arrays de várias dimensões.

4.8. MODELAGEM GEOMÉTRICA COM OPENG

4.8.1. Desenhando Pontos

Os pontos podem ter tamanhos específicos baseados no número de pixels, tendo como padrão o valor 1.

```
void glPointSize(GLfloat tamanho),
```

Sendo definido pelas suas coordenadas:

```
glBegin(GL_POINTS);  
    glVertex3d(0.0, 0.0, 0.0);  
    glVertex3d(100.0, 0.0, 0.0);  
    glVertex3d(100.0, -100.0, 0.0);  
    glVertex3d(0.0, -100.0, 0.0);  
    glVertex3d(50.0, -50.0, 100.0);  
glEnd( );
```

4.8.2. Desenhando uma Linha

As linhas podem ser apresentadas como setas, tracejadas, pontos e outras variações destas.

GL_LINES

4.8.3. Desenhando um Polígono

Os polígonos podem ser representados por pontos, segmentos de retas, sólidos, texturas, entre outras.

GL_POLYGON

4.8.4. Primitivas

Além das anteriores as seguintes primitivas estão disponíveis em OpenGL:

- GL_LINES: exibe uma linha a cada dois comandos *glVertex*;
- GL_LINE_STRIP: exibe uma sequência de linhas conectando os pontos definidos por *glVertex*;
- GL_LINE_LOOP: exibe uma sequência de linhas conectando os pontos definidos por *glVertex* e, ao final, liga o primeiro ao último ponto;
- GL_POLYGON: exibe um polígono convexo preenchido, definido por uma sequência de chamadas a *glVertex*;
- GL_TRIANGLES: exibe um triângulo preenchido a cada três pontos definidos por *glVertex*;
- GL_TRIANGLE_STRIP: exibe uma sequência de triângulos baseados no trio de vértices *v0*, *v1*, *v2*, depois, *v2*, *v1*, *v3*, então, *v2*, *v3*, *v4* e assim por diante;
- GL_TRIANGLE_FAN: exibe uma sequência de triângulos conectados baseados no trio de vértices *v0*, *v1*, *v2*, depois, *v0*, *v2*, *v3*, então, *v0*, *v3*, *v4* e assim por diante;
- GL_QUADS: exibe um quadrado preenchido conectando cada quatro pontos definidos por *glVertex*;
- GL_QUAD_STRIP: exibe uma sequência de quadriláteros conectados a cada quatro vértices; primeiro *v0*, *v1*, *v3*, *v2*, depois, *v2*, *v3*, *v5*, *v4*, então, *v4*, *v5*, *v7*, *v6*, e assim por diante.

4.8.5. Objetos Sólidos

A biblioteca OpenGL possui uma série de sólidos que podem ser criados diretamente através dos seguintes comandos:

```

glutSolidTeapot(GLdouble size); // Desenha uma Chaleira da Figura 4.29
glutSolidCube(GLdouble size);   // Desenha um Cubo
glutSolidSphere(GLdouble radius, GLint slices, GLint stacks); // Desenha uma Esfera
glutSolidCone(GLdouble radius, GLdouble height, GLint slices, GLint stacks);
glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings);
glutSolidIcosahedron(void);
glutSolidOctahedron(void);
glutSolidTetrahedron(void);
glutSolidDodecahedron(GLdouble radius);

```

Os parâmetros **slices** e **stacks**, que aparecem no protótipo de algumas funções, representam os números de subdivisões em torno do eixo z (como se fossem linhas longitudinais) e o número de subdivisões ao longo do eixo z (como se fossem linhas de latitude). Os parâmetros **rings** e **nsides** correspondem, respectivamente, ao número de seções que serão usadas para formar o torus, e ao número de subdivisões para cada seção. O parâmetro **size** indica um diâmetro aproximado.

4.8.6. Wireframe

Os objetos das primitivas podem ser desenhados também em wireframe substituindo a palavra **Solid** por **Wire** no nome dos comandos, por exemplo:

```

glutWireTeapot(GLdouble size); // Desenha uma Chaleira em wireframe

```

4.9. MODELAGEM PELO NÚMERO DE OURO

A modelagem pelo número de ouro nos remete à Grécia antiga, mais precisamente ao século V a.C. Nas ruas dessa avançada civilização, eram discutidos conceitos de filosofia, matemática e ciência. Os gregos adoravam o teatro, as esculturas, a arquitetura e qualquer forma de manifestação da arte. Um exemplo dessa adoração foi a construção, no centro de Atenas, do Partenon, mais conhecido como Templo das Virgens (Figura 4.34). Na época, o líder do partido democrático Péricles, contratou o maior escultor da Grécia para desenvolver e acompanhar a construção em louvor à deusa da cidade, Atena Partenos. O escultor utilizou em sua fachada um modelo de medida onde o lado maior dividido pelo lado menor é igual à divisão entre o lado menor e a diferença entre o lado maior e o menor (Figura 4.34). Os gregos consideravam essa proporção a forma ideal de beleza e harmonia e deram a ela o nome de proporção áurea ou proporção de ouro.

No século XV, o movimento renascentista buscou retomar os valores estéticos da Grécia antiga. Na época, Leonardo Da Vinci (1542-1519) apresentou o famoso “O Homem Vitruviano” (Figura 4.35), onde podemos ver a proporção de ouro relacionada com a estrutura ideal do corpo humano. Segundo ele, o umbigo deve dividir o corpo segundo a seção áurea, ou seja, o resultado da divisão da altura total pela altu-

ra do umbigo deve ser o número de ouro. O estudo de Da Vinci recebeu este nome por ser baseado no tratado “De architecture” do arquiteto romano do século I a.C. Marcus Vitruvius Pollio, o único tratado de arquitetura que restou da antiguidade que considerava, em 10 livros, de métodos de construção e decoração a planificação de cidades e suprimento de água, nas proporções do que chamava de segmento áureo, que seria o retângulo perfeito.

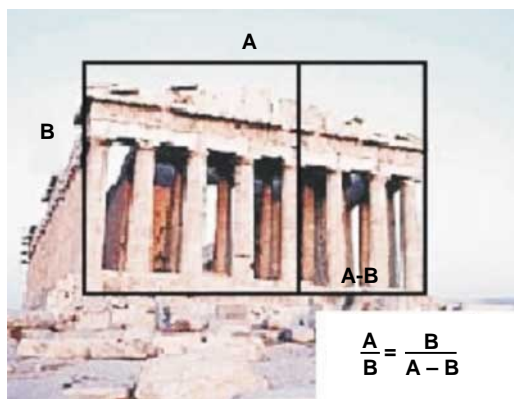


FIGURA 4.34. Partenon na acrópole de Athenas e a proporção áurea.

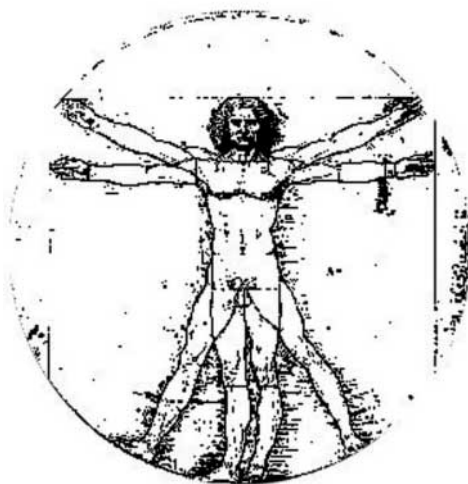


FIGURA 4.35. As proporções do corpo humano segundo o cânon Vitruviano desenho de Da Vinci (Academina, Veneza).

Ainda hoje, artistas de todo o mundo utilizam essa proporção em suas obras, na maioria das vezes intuitivamente, porém podemos perceber o número de ouro em uma infinidade de obras de arte e monumentos arquitetônicos. No Brasil, pintores

como Portinari fizeram grande uso dessa proporção intencionalmente. Na obra “O Café”, Portinari utilizou a proporção para atrair a atenção do observador para um ponto desejado da cena.

A proporção áurea, ou número de ouro, é ensinada nas escolas de arte, porém é difícil explicar o que a faz tão atraente e harmoniosa. Na verdade, a proporção trouxe uma forma do artista controlar e garantir a harmonia e perfeição de sua criação. De forma surpreendente esse número aparece no arranjo de diversas formas de plantas [Maor, 1988].

Curiosos em como obter a proporção áurea? Ela é dada pela relação mostrada na Figura 4.34. Se você substituir A por 1 e B por x, terá a equação $1/x = x/(1 - x)$; que também pode ser escrita como $x^2 + x - 1 = 0$ e cuja raiz positiva é $x = (\sqrt{5} - 1)/2$. Assim a razão áurea ou número de ouro é $1/x$. Este número como $\sqrt{2}$ ou π é um número irracional, ou seja um número cuja expansão decimal é infinita e nunca se repete. Como $\sqrt{2}$, \emptyset é um irracional algébrico (originário de um polinômio com coeficientes inteiros). Outros irracionais como π e e são chamados de transcendentais. O número de ouro foi batizado com a letra grega “ \emptyset ” e vale aproximadamente 1,61803...

4.9.1. A Sequência de Fibonacci

Há muito tempo que as manifestações geométricas na natureza vêm intrigando muitas pessoas. Na regularidade do crescimento das árvores, nas proporções do corpo humano e dos animais, na frequência do nascimento de animais, na forma de conchas, na regularidade do girassol ou na constituição hexagonal dos favos de abelhas.

Leonardo Fibonacci, (ou Leonardo de Pisa (1180-1250), entre muitos outros feitos, escreveu o “Liber Abaci” (Livro do Ábaco), onde apresentava a forma indo-arábica de números que permitia maior rapidez nas operações feitas através da numeração decimal quando comparadas às operações com números romanos utilizados na época pela sociedade européia. Fibonacci (filho de negociantes, aprendeu matemática no norte da África e assimilou em suas viagens os conhecimentos árabes) escreveu: “Impossível algo subsistir se não for devidamente proporcional à sua necessidade”.

Fibonacci enunciou o seguinte problema: “Se eu tiver um casal de coelhos que gera um novo casal ao fim de dois meses e depois um novo casal todos os meses (os quais geram novos casais nas mesmas condições), quantos casais de coelhos terei ao fim de n meses?” A resposta é dada pela série 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233...(Figura 4.36), designada geralmente como Números de Fibonacci ou Sequência de Fibonacci. Essa série tem a particularidade de à medida que se avança no número de casais, a razão entre um valor e o seu antecessor se aproxima do número de ouro. Ex: $21/13 = 1,6153...$

A descoberta da relação entre a sequência de Fibonacci e o número de ouro desencadeou uma série de pesquisas em busca de seqüências e padrões matemáticos

na natureza. As descobertas permitem hoje que biólogos utilizem os padrões para organizar e reconhecer espécies. Dentre as descobertas relacionadas com o número de ouro, estão exemplos como a da concha do Nautilus (Figura 4.37). À medida que o molusco vai crescendo, ele vai construindo uma nova câmara para morar e cada nova câmara é maior que a anterior na proporção do número de ouro.



FIGURA 4.36. A solução do problema de Fibonacci.



FIGURA 4.37. À medida que o Nautilus cresce, uma nova câmara é construída em relação a anterior, na proporção do número de ouro.

Podemos citar diversos outros exemplos como a disposição das sementes de um girassol, das escamas de peixe, da distribuição de galhos em árvores ou das cores na natureza. O número de ouro pode aparecer ainda nas poesias, em pirâmides no Egito, no artesanato indígena ou onde mais você desejar. Trata-se de uma poderosa chave para modelagem de objetos naturais e, como se disse anteriormente, uma forma segura de o artista controlar e garantir a harmonia e a perfeição de sua criação.

4.10. MODELAGEM FRACTAL

No decurso da história humana, a nossa crescente percepção do mundo natural foi atribuída a um universo com um número cada vez maior de dimensões (seção 1.2.1). Há dois mil anos, os Gregos mostraram que o universo tinha três dimensões, com

base nos sentidos e nos princípios básicos da geometria Euclidiana, formalizada por Euclides a partir de axiomas, onde todos os objetos são dotados de comprimento, largura e altura.

Na geometria Euclidiana uma linha (como um fio de lã) tem principalmente comprimento, portanto, é um objeto unidimensional. Um plano (como a folha de papel), que possui dois comprimentos é um objeto bidimensional, e um sólido (como um cubo ou livro), tem comprimento, largura e altura sendo definido como tridimensional (Figura 4.38).

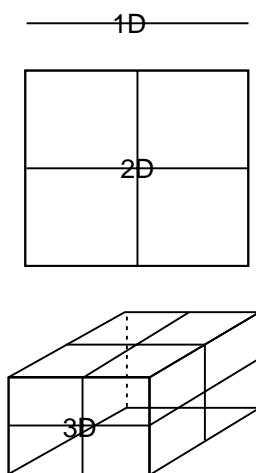


FIGURA 4.38. *Objetos de 1, 2 ou 3 dimensões.*

Desse modo, os matemáticos do tempo de Euclides concordavam com a noção do senso comum de que o universo possui três dimensões. A geometria Cartesiana, de René Descartes auxiliou a Euclidiana nessa definição passando a definir a dimensionalidade de um objeto pelo número de coordenadas necessárias para a sua descrição.

No ano de 1854, um jovem matemático alemão, Bernhard Riemann (1826-1866), aluno de Gauss, anunciou uma nova extensão da geometria de Euclides e da Geometria Analítica de Descartes. Ele defendia que as retas paralelas se encontram em um ponto: o pólo. Ao criar essa nova definição de dimensão, Riemann conseguiu descrever melhor as coisas relacionadas à latitude e à longitude que ocorrem nas coordenadas geográficas de nosso planeta [Struik, 1967].

Benoit Mandelbrot (1924-...), em 1975, consolidou e interpretou os trabalhos dispersos de muitos matemáticos que o antecederam no estudo das dimensões não-inteiras. Mandelbrot mostra que é possível definir uma dimensão fracionária, tal como dimensão $1/3$ ou dimensão $3/4$, por exemplo.

Mandelbrot também afirmou que as dimensões fracionárias típicas da paisagem terrestre diferem das de Marte. Esse estudo foi feito com base em fotos do planeta vermelho obtidas através da NASA. Daqui conclui-se que a Terra ronda a dimensão

2,1, enquanto Marte atinge cerca de 2,4, o que significa que a paisagem de Marte é muito mais “dentada” do que a da Terra [Voss, 1988].

As indústrias cinematográficas utilizaram essas informações e o auxílio dos computadores para conseguirem criar paisagens com qualquer dimensão fracionária particular, o que facilitou a criação de paisagens com aspectos estranhos, tão utilizadas em filmes de ficção científica, como em *Star Trek* [Mandelbrot, 1988].

Os trabalhos de Mandelbrot tiveram como ponto de partida a definição de “dimensão” apresentada em 1919 por Felix Hausdorff, (cerca de sessenta anos após ter sido apresentada a definição de Riemann) [Falconar, 1990].

A geometria dos fractais (palavra derivada do Latim, “fractus” ou do adjetivo “frangere”, que significa “quebrar”) apresenta estruturas geométricas de grande complexidade e beleza, ligadas às formas da natureza, ao desenvolvimento da vida, à própria compreensão do universo ou às simples expressões matemáticas. São imagens de objetos abstratos que possuem o caráter de onipresença por terem as características do todo infinitamente multiplicadas dentro de cada parte [Peitgen e Richter, 1986].

Os fractais podem apresentar uma infinidade de formas diferentes, contudo, existem duas características muito freqüentes nesta geometria: auto-semelhança e complexidade em qualquer escala de observação [Mandelbrot, 1977].

Essa geometria, nada convencional, tem raízes remontando ao século XIX e algumas indicações nesse sentido vêm de muito antes, na Índia, China e Grécia, quando Pitágoras fracionou aritmeticamente uma corda para produzir as notas e suas variações musicais (Capítulo 1).

Porém somente há poucos anos essa geometria vem se consolidando com o desenvolvimento dos computadores e o auxílio de novas teorias nas áreas da física, biologia, astronomia, matemática e outras ciências.

Os fractais constituíram certamente uma surpresa e até mesmo um abalo para muitos. De repente, viram-se confrontados com técnicas e imagens que, se por um lado eram altamente sugestivas, por outro, não conseguiam ser justificadas nem englobadas em situações anteriormente conhecidas. Assim, ao mesmo tempo em que uns, com a ajuda do computador, tentavam encontrar sentido nos resultados que obtinham, outros esforçavam-se por produzir definições e demonstrações em termos matemáticos tradicionais.

Distante do rigor e do formalismo matemático, é possível definir Fractais, como nos ensinam alguns estudiosos da área: “Objetos que apresentam auto-semelhança e complexidade infinita, ou seja, têm sempre cópias aproximadas de si; mesmo em seu interior.” (Figura 4.39)

A propriedade de auto-similaridade é o ponto central da geometria fractal e está associada ao conceito de dimensão. Na Figura 4.39, a curva de flocos de neve de Kock, é construída a partir de uma reta dividida em 3 partes iguais cuja parte central é substituído por 2 outros pedaços de mesmo comprimento, em um procedimento repetido infinitamente.

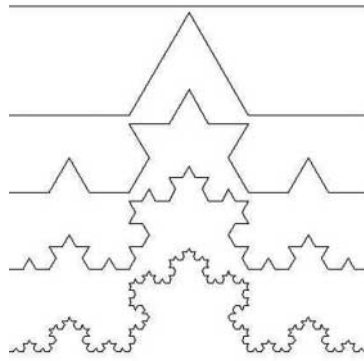


FIGURA 4.39. *Os fractais têm sempre cópias aproximadas de si.*

A Geometria Fractal pode ser utilizada para descrever diversos fenômenos da natureza, onde não podem ser utilizadas as geometrias tradicionais. “Nuvens não são esferas, montanhas não são cones, continentes não são círculos, um latido não é contínuo e nem o raio viaja em linha reta.” [Mandelbrot, 1977]. Outros exemplos são: gelo, neve, cinzas, flores, plantas, tecidos ou frutas [Pruxinkiwicz, 1990]. A Figura 4.40 ilustra o uso de fractais na concepção de água e nuvens.



FIGURA 4.40. *Água e nuvens criadas com fractais.*

O processo de gerar montanhas fractais é semelhante ao processo de geração da curva de Kock [Figura 4.39] com um fator adicional randômico que realiza as irregularidades da superfície. Nesse caso, as superfícies com uma dimensão D maior aparentam ser mais rugosas [Saupe, 1988].

O fenômeno de auto-similaridade pode ser observado na natureza onde pedras parecem com cascalhos, galhos com troncos e assim por diante.

4.10.1. O Conjunto de Mandelbrot

Considerados como um dos objetos mais complexos da matemática, o conjunto de Mandelbrot possui a particularidade de combinar os aspectos de auto-similaridade em qualquer escala do resultado do processo iterativo (Figura 4.41). Esse conjunto é gerado pelo estudo das iterações $x_{K+1} \rightarrow x_K^2 + c$, onde x e c são números complexos. Um número complexo é um número que tem duas partes, uma real e outra imaginária, ou seja, tem a forma $x = x_r + x_i * i$, onde $i = \sqrt{-1}$; são exemplos de números complexos: $c = 2 + 3i$ ou $d = -4 + 1$. Através da variação dos números complexos c , podemos gerar os diversos conjuntos de Julia. Assim os conjuntos de Julia dependem do valor de c [Peigen, 1988].

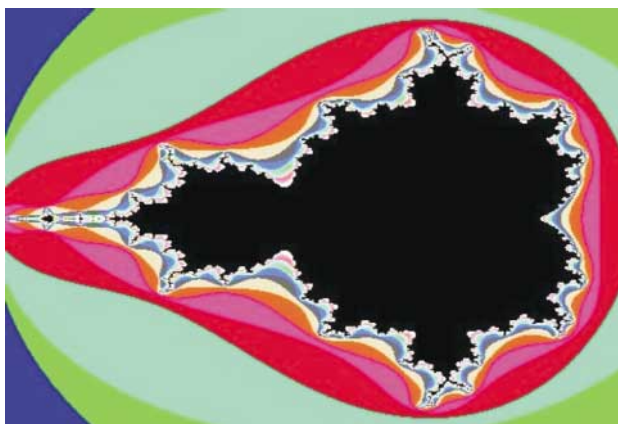


FIGURA 4.41. Os conjunto de Mandelbrot.

Podemos encontrar a aplicação dos fractais na música, no reconhecimento de padrões, na medicina, na engenharia, na meteorologia, na geociência e muito mais.

A aplicação dos fractais é cada vez maior, principalmente pela sua associação com os sistemas dinâmicos e caóticos, constituindo uma ferramenta científica de enorme alcance que ainda está em seus primeiros passos. No entanto, muito contribui para a sua divulgação suas imagens que, no mínimo, podem ser consideradas intrigantes e bizarras [Peitgen e Richter, 1986].

4.11. RECONSTRUÇÃO TRIDIMENSIONAL

A representação do espaço tridimensional (3D), no plano bidimensional (2D), introduziu realismo nas pinturas e desenhos e foi uma das mais importantes desco-

bertas no mundo das artes: a perspectiva (seção 2,7.4). Quando a fotografia foi inventada, a possibilidade de registro em 2D de cenas 3D foi popularizada (passando a ser feita por máquinas e não mais por dons artísticos ou estudos de técnicas de desenho).

A imagem de uma máquina fotográfica, assim como a projeção perspectiva, possibilita o registro de uma cena 3D em 2D com o realismo da visão humana. Uma imagem fotográfica e uma projeção perspectiva contêm informações análogas. O que se espera da imagem fotográfica de objetos é que ela corresponde à imagem dos objetos vista por olhos humanos.

Fotografias e vídeo constituem meios, tradicionais e amplamente difundidos, de registrar uma cena de forma instantânea. No entanto, no processo de projeção do mundo 3D em imagens 2D, parte da informação contida na cena, como a informação de profundidade, é perdida. Uma única imagem representando as mudanças de intensidade de luz e geometria local de uma cena não contém, por si só, informação suficiente para reconstrução da cena. Assim, uma questão importante é a recuperação, a partir de imagens de intensidade bidimensionais, das propriedades tridimensionais geométricas e físicas das superfícies representadas nessas imagens [Pentland, 1990].

A perda de informação que ocorre ao se projetar em um plano a cena tridimensional geralmente faz com que a solução do problema inverso não seja única (ambigüidade), ou seja inexistente devido à oclusão de informações. Como consequência, precisa contar com suposições (restrições) sobre o mundo físico para eliminar a ambigüidade na sua reconstrução. Essas suposições, de acordo com o número de vistas usadas na análise da cena, podem se basear em técnicas monoculares, binoculares ou de múltiplas vistas [Noborio et al, 1988].

Visão monocular é o termo usado para técnicas de visão computacional que utilizam apenas um ponto de vista. Os métodos de aquisição de forma dependem da informação disponível. Uma só imagem não é suficiente para fornecer informação completa de profundidade, mas distâncias a objetos, ou profundidades podem ser percebidas monoscopicamente com base no tamanho relativo de objetos ou de suas texturas (seção 1.5).

Pessoas com visão normal têm visão binocular, e é a partir desta visão estereoscópica que o ser humano inconscientemente avalia profundidades ou julga distâncias [Hads, 1992].

A reconstrução tridimensional, através de técnicas de computação visual e computação gráfica, pode recriar espaços virtuosos (VE) a partir de imagens planas, e ser um ponto de partida para um número enorme de aplicações: reconstrução de edificações, planejamento de estratégias militares, cenários para jogos tridimensionais, sistemas interativos de realidade virtual em tempo real, sistemas para visão de robôs, transformação de desenhos 2D ou filmes para 3D, e muitas outras ainda por se criar [Stytz et al, 1996]. Atualmente, os maiores usuários das

técnicas de reconstrução as utilizam para processamento de imagens médicas, microscopia, geologia e indústria aeroespacial [Rhods, 1997].

Para reconstrução do objeto é importante conhecer as coordenadas tridimensionais (3D) de seus pontos (ou pelo menos dos pontos mais importantes do objeto), e recuperar dados perdidos no processo de projeção/fotografia, que transforma os dados 3D em 2D. Diferentes técnicas empregadas nos sistemas de visão computacional tentam obter esta terceira coordenada.

Os sistemas de *shape from stereo* usam duas ou mais imagens de um ponto de vista, com uma pequena diferença de iluminação, para calcular a profundidade. Podemos ainda utilizar duas ou mais imagens tiradas de uma câmera em movimento [Giachetti e Torre, 1996].

A teoria geométrica da visão estéreo, que utiliza a obtenção de coordenadas tridimensionais de objetos a partir de pares de imagens, depende da capacidade de se resolver o problema da correspondência, ou seja, determinar os elementos do par de imagens que são projeções ao mesmo elemento no mundo 3D. Não existe solução geral para o problema da correspondência devido à existência de candidatos ambíguos para combinação, ou mesmo da inexistência de correspondentes devido à oclusão. Conseqüentemente, todo método estéreo usa várias suposições sobre a geometria da imagem e/ou os objetos na cena para reduzir o número de ambigüidades.

Na busca de soluções para dificuldades encontradas nos métodos estéreos, também foram desenvolvidas técnicas de múltiplas vistas. O multiple-baseline stereo foi desenvolvido por Okutomi e Kanade em 1991, e é base de diversas variações. O estéreo fotométrico baseado em disparidades, agrega características dos processos de estéreo fotométrico e de estereoscopia.

Nos sistemas *shape from motion*, um objeto em movimento pode produzir as disparidades de iluminação em uma seqüência de imagens capturadas por uma câmera estática.

Nos sistemas *shape from texture*, propostos inicialmente por Kender em 1979, as variações no tamanho, na forma e na densidade das texturas da imagem 2D fornecem pistas para a recuperação das informações 3D.

Os sistemas *shape from focus* exploram as propriedades das câmeras, onde objetos em uma determinada distância aparecem focados e outros, em distâncias maiores, ficam mais embaçados quanto maior sua distância do ponto focal.

Nos sistemas *shape from shading*, introduzidos em 1977 por Horn, a idéia básica se concentra na variação da intensidade no plano da imagem para calcular a profundidade, considerando o modelo de iluminação de Phong e Gouraud.

As técnicas de *shape from shading* têm sido alvo de muitos estudos, mas sua integração com outras técnicas de reconstrução 3D, como motion, textura e estéreo, por exemplo, é recente. A importância da fusão de técnicas está na possibilidade de explorar os pontos fortes de cada uma das técnicas envolvidas. Sistemas que utilizam a fusão de técnicas de reconstrução de superfícies ainda não estão disponíveis

comercialmente no mercado, dado que essas tentativas são recentes e, portanto, com poucos resultados práticos disponíveis.

A habilidade de representar um objeto no espaço tridimensional é fundamental para visualizar, compreender e analisar detalhes do objeto [Wanger et al, 1992]. Girar, transladar e projetar o objeto é fundamental para diversas aplicações da simples compressão de sua forma a mais sofisticada aplicação de realidade virtual.

A técnica Shape from Boxes é o resultado de um estudo realizado junto a Secretaria de Segurança Pública do Estado do Rio de Janeiro (SSP-RJ). Para a SSP-RJ, a inclusão de um sistema de treinamento em realidade virtual deveria acompanhar as necessidades imediatistas de planejamento das operações do Comando de Operações Especiais, o COPE.

Essa metodologia foi então publicada no Simpósio Brasileiro de Computação Gráfica (SIBGRAPHI 2002) onde apresentamos um método para retirar as informações 3D do objeto a partir de uma foto (Figura 4.42) e pontos marcados na imagem, como mostram as linhas na Figura 4.43. A Figura 4.44 mostra a reconstituição vista de um outro ângulo. A reconstituição permitirá visualizar espaços entre as construções, determinar campos e alcances visuais, distâncias, medidas, inclusão de ícones e objetos para planejamento estratégico.



FIGURA 4.42. Imagem aérea original.

O sistema trabalha em conjunto com a engrenagem Fly3D, permitindo simulações de colisão, passeios virtuais, treinamento em redes, perseguições com armas de fogo e todos os outros recursos de simulação física da engrenagem. O modelo 3D pode ser exportado para o Max, onde ajustes finos de iluminação e modelagem podem ser feitos.



FIGURA 4.43. *Determinação de pontos estratégicos para a reconstrução.*



FIGURA 4.44. *Reconstituição vista de um outro ângulo.*

O sistema permite ainda a escolha da textura do solo, iluminação virtual, inclusão de qualquer objeto 3DS na cena, empilhamento de objetos reconstruídos e vãos virtuais pelas construções através de um observador móvel.

4.12. SISTEMAS DE PARTÍCULAS

A história dos sistemas de partículas iniciou em 1983 quando William T. Reeves publicou seu paper “Particle Systems – A Technique for Modeling a Class of Fuzzy Objects”. Nesse paper, Reeves explica como chegou ao paradigma de um sistema de partícula para um projeto de efeitos especiais do filme *Star Trek II: A Ira de Khan*. Ele mostrou que, aplicando algumas das leis fundamentais da mecânica de Newton para uma coleção virtual de partículas no computador, era possível criar elementos. Até então, a computação gráfica consistia principalmente de formas cria-

das por polígonos e vértices fixos. A metodologia de Reeves permitiu a criação de objetos que não tinham extremidades discretas. Esse novo paradigma permitiu a criação de efeitos como neve, chuva, fogo, fogos de artifício e nuvens [Loke et al, 1992].

Antes de Reeves, os sistemas de partículas foram estudados pela mecânica das colisões. Na segunda metade do século XVII, o estudo das colisões sofreu um grande desenvolvimento. Nomes como os de Robert Hooke, Christian Huyghens e Christopher Wren, contribuíram para esse desenvolvimento, levando a cabo trabalhos de pesquisas baseados em experimentações, cujo objetivo era interpretar corretamente o mecanismo das colisões.

O segundo avanço importante, em sistemas de partículas, veio mais uma vez de Reeves, em um paper intitulado, *Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems*. Reeves melhorou o algoritmo do sistema de partícula para criar objetos mais complexos como árvores, gramas, e arbustos. Para modelar esses objetos, o comportamento dos sistemas de partículas precisava ser expandido. A maior mudança, foi a implementação de um processo de modelagem estocástico.

Os trabalhos posteriores em sistemas de partículas consistem em restringir ou modificar o comportamento geral das partículas de Reeves. Por exemplo, Teng-See Loke usou os sistemas de partículas, junto com as leis de Newton, para produzir fogos de artifício. Craig Reynolds usou os sistemas de partículas para modelar o comportamento de bandos de pássaros. Richard Szeliski achou um caminho para criar sistemas de partícula que têm uma superfície de orientação. A partir dessa técnica, foi possível criar objetos que eram originados dos sistemas de partículas. Alex Pang usou os sistemas para pintar, dissecar e modificar objetos tridimensionais.

Os sistemas de partículas são poderosas ferramentas usadas para modelar água, nuvens, fogo, fumaça, explosões, fluidos em geral, neve, árvores etc. (Figura 4.45). Na animação, contribuem na modelagem de multidões, exércitos, manadas etc. A Figura 4.46 exemplifica o uso dos sistemas de partículas para criar e controlar o comportamento de cardumes. Na Figura 4.47, foram utilizados dois emissores, um para dar o efeito explosão, fumaça e fogo e o outro para simular o lançamento dos destroços do tanque.

4.12.1. Distribuição de Partículas no Sistema

Nos sistemas de partículas, distingui-se duas formas diferentes de distribuição (de um ponto de vista macroscópico): contínuos e discretos. A primeira caracteriza-se por apresentar uma distribuição contínua de porções de matéria que é representada por um volume elementar dv e uma massa elementar dm . Essa distribuição caracteriza-se pela sua massa específica em cada ponto $r = dm/dv$. O desgaste de um lápis e uma borracha são dois exemplos de uma distribuição contínua de partículas. A dis-

tribuição descontínua é o caso, por exemplo, de duas bolas de bilhar que se chocam sobre a mesa. Essa distribuição é caracterizada em cada instante pelas massas e posições das partículas que as constituem. É nela que vamos centrar a nossa atenção (Figura 4.48). Esta caracterização é muito semelhante à forma como se considera o modelo físico dos corpos em estudo. Modelos contínuos ou discretos aqui, como na física, são semelhantes ao lápis e à borracha. Eles poderiam ser considerados modelos discretos se o objetivo da análise for, por exemplo, representar não o seu desgaste com o uso e sim modelar a trajetória de colisão de ambos ao serem jogados sobre uma mesa.



FIGURA 4.45. Sistema de Partículas simulando água em colisão com as pedras.

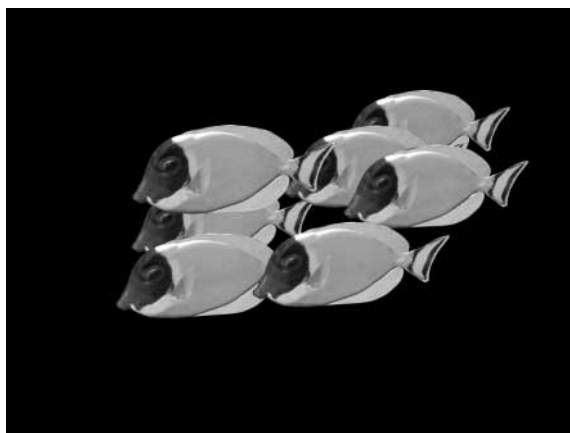


FIGURA 4.46. Modelagem de cardumes com partículas.



FIGURA 4.47. Dois emissores com simulação de fumaça, fogo, explosão e lançamento de destroços.

4.12.2. Centro de Massa

O centro de massa é o ponto, onde se supõe concentrada toda a massa M do sistema, e cujo vetor posicional em relação a uma origem fixa e arbitrária é R_{cm} (Figura 4.48).

$$R_{cm} = \frac{\sum m_k \cdot r_k}{M}$$

$$M = \sum m_k$$

onde m_k é a massa de cada partícula do sistema e r_k é o seu vetor posicional em relação a uma origem fixa e arbitrária.

4.12.3. Velocidade do Centro de Massa

A velocidade do centro de massa é obtida derivando em relação ao tempo a expressão do vetor posicional do centro de massa R_{cm} .

$$\frac{dR_{cm}}{dt} = \frac{1}{M} \sum m_k \cdot \frac{dr_k}{dt} = \frac{1}{M} \sum m_k \cdot v_k$$

$$V_{cm} = \frac{1}{M} \sum m_k \cdot v_k$$

4.12.4. Quantidade de Movimento

A quantidade de movimento de uma partícula de massa m_k , que se move com velocidade v_k , é $p_k = m_k \cdot v_k$ modo que:

$$V_{cm} = \frac{1}{M} \sum m_k \cdot v_k = \frac{\sum p_k}{M} = \frac{P}{M}$$

$$P = M V_{cm}$$

A quantidade de movimento do sistema é a quantidade de movimento do seu centro de massa.

4.12.5. Aceleração do Centro de Massa

Derivando V_{cm} em relação ao tempo, obtém-se a aceleração do centro de massa do sistema de partículas:

$$\frac{dV_{cm}}{dt} = \frac{1}{M} \sum m_k \frac{dv_k}{dt} = \frac{\sum m_k \cdot a_k}{M}$$

considerando que $f_k = m_k \cdot a_k$ representa a resultante de todas as forças que atuam sobre a partícula k , e que $\sum f_k$ representa a resultante de todas as forças que atuam sobre todas as partículas do sistema, tem-se:

$$\sum f_k = M A_{cm}$$

Essas forças podem ser de dois tipos:

Forças exteriores (f_e) – exercidas por partículas que não pertencem ao sistema, sobre as partículas do sistema; e

Forças interiores (f_i) – aquelas que as partículas do sistema exercem umas sobre as outras, e cuja resultante é sempre nula (lei da ação e reação).

Seja F_e a resultante de todas as forças exteriores que atuam sobre todas as partículas do sistema, a equação anterior reduz-se a:

$$F_e = M A_{cm}$$

Recordando que a quantidade de movimento de uma partícula de massa m_k , que se move com velocidade v_k , é $p_k = m_k \cdot v_k$, podemos escrever:

$$M A_{cm} = \sum m_k \cdot \frac{dv_k}{dt} = \sum \frac{dp_k}{dt} = \frac{dP}{dt} = F_e$$

$$F_e = \frac{dP}{dt}$$

A derivada em relação ao tempo da quantidade de movimento de um sistema de partícula é igual à resultante das forças exteriores que atuam sobre o sistema. Con-

siderando às equações $F_e = M A_{cm}$ e $F_e = dP/dt$, podemos concluir que, para cada sistema de partículas pode se encontrar um ponto (o seu centro de massa) que se move como se fosse um ponto material onde estivesse concentrada toda a massa do sistema e sobre o qual atuassem todas as forças exteriores que exercem a sua ação sobre o sistema.

Se a resultante das forças exteriores que atuam sobre o sistema for nula, a sua quantidade de movimento permanece inalterada, isso é $F_e = dP/dt = 0$, ou seja, $P =$ constante.

Assim é constante a quantidade de movimento de um sistema de partículas isolado (sobre o qual não atuam forças exteriores, ou atuam forças de resultante nula).

4.12.6. Movimento em Relação ao Centro de Massa

Considerando a Figura 4.48, onde r_k é vetor posicional da partícula k em relação ao referencial S ; r_{kcm} o vetor posicional da partícula k em relação ao centro de massa do sistema, e R_{cm} o vetor posicional do centro de massa do sistema em relação ao referencial S . Podemos observar que:

$$r_k = r_{kcm} + R_{cm}$$

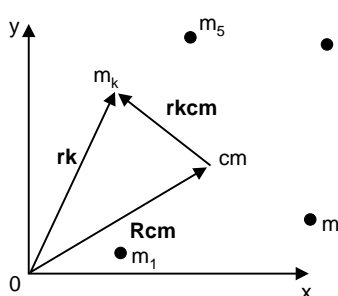
$$\frac{dr_k}{dt} = \frac{dr_{kcm}}{dt} + \frac{dR_{cm}}{dt}$$


FIGURA 4.48. Um sistema de partículas.

4.12.7. Choque

De um modo geral, dizemos que ocorre choque ou colisão de dois corpos, quando eles entram em contato, por exemplo, o choque de duas bolas de bilhar. No entanto, não tem de ser necessariamente assim. Se duas partículas se aproximam, a certa altura faz-se sentir a ação de uma sobre a outra (lei de atração universal). A força que cada partícula exerce sobre a outra, altera a trajetória desta, a sua quantidade de movimento e a sua energia. Desde que as partículas se aproximem o suficiente para que seja detectável a interação mútua, pode-se dizer que houve interferência.

4.12.8. Quantidade de Movimento de um Sistema de Duas Partículas

Se durante o choque, apenas intervierem forças interiores, a quantidade de movimento do sistema mantém-se inalterada, antes e depois do choque.

Se $p_1 = m_1 v_1$, $p_2 = m_2 v_2$ forem as quantidades de movimento de duas partículas antes do choque e, $p_1 = m_1 v_1'$, $p_2 = m_2 v_2'$ as quantidades de movimento das duas partículas depois do choque, pelo teorema de conservação da quantidade de movimento; tem-se:

$$\begin{aligned} p_1 + p_2 &= p_1' + p_2' \\ m_1 v_1 + m_2 v_2 &= m_1 v_1' + m_2 v_2' \end{aligned}$$

4.12.9. Energia de um Sistema de Duas Partículas

Se durante o choque apenas atuarem sobre cada partícula forças interiores, a conservação da energia do sistema considerado traduz-se pela igualdade:

$$\left(\frac{m_1 v_1'^2}{2} + \frac{m_2 v_2'^2}{2} \right) - \left(\frac{m_1 v_1^2}{2} + \frac{m_2 v_2^2}{2} \right) = E_p(1,2) - E_p'(1,2)$$

onde $\frac{1}{2} m_1 v_1'^2$, $\frac{1}{2} m_2 v_2'^2$ e $\frac{1}{2} m_1 v_1^2$, $\frac{1}{2} m_2 v_2^2$ representam respectivamente, as energias cinéticas das duas partículas antes e depois do choque; e $E_p(1,2)$ e $E_p'(1,2)$ designam as energias potenciais internas do sistema constituído pelas duas partículas, antes e depois do choque. Essas energias poderão ser diferentes, já que dependem da configuração do sistema, que pode alterar-se durante o choque, chamando de a esta variação, tem-se:

$$Q = \left(\frac{1}{2} m_1 v_1'^2 + \frac{1}{2} m_2 v_2'^2 \right) - \left(\frac{1}{2} m_1 v_1^2 + \frac{1}{2} m_2 v_2^2 \right)$$

Se $Q = 0$, o choque é perfeitamente elástico, neste caso a energia cinética do sistema é a mesma antes e depois do choque.

4.12.10. Conceituando de Sistemas de Partículas

O termo sistemas de partículas pode ser usado para descrever técnicas de modelagem, rendering e animação. De fato, a definição de um sistema de partícula parece depender da aplicação para a qual se destina, porém, os sistemas de partículas devem obedecer aos seguintes critérios:

- **Coleção de Partículas** – Um sistema de partículas é composto de uma ou mais partículas individuais. Cada uma dessas partículas tem atributos que direta ou indiretamente afetam o comportamento da partícula ou, em última instância, como e onde a partícula é posicionada. Frequentemente, partículas são primitivas gráficas como pontos ou linhas. Os sistemas de partículas tam-

bém são usados para representar uma dinâmica de grupo complexa, como um aglomerado de pássaros.

- **Atributos Estocásticos** – Uma outra característica comum de todos os sistemas de partículas é a introdução de algum tipo de efeito randômico. Esses efeitos podem ser usados para controlar os atributos das partículas como posição, velocidade e cor. Normalmente, o efeito randômico é controlado por um tipo de limite estocástico predefinido, como variações, ou tipo de distribuição.

Os sistemas de partículas são um exemplo de modelagem procedural estocástica, e possuem algumas características:

- os sistemas complexos podem ser criados com pequeno esforço humano; e
- o nível de detalhe pode estar facilmente ajustado. Por exemplo, se um objeto emissor de partículas está longe da câmera, então ele pode ser modelado com poucos detalhes (poucas partículas), mas se estiver perto da câmera, então ele pode ser modelado com riqueza de detalhes (muitas partículas).

As partículas não têm superfícies bem definidas e nem são objetos rígidos, isto é, podem ser dinâmicas e fluidas. Os sistemas de partículas diferem em três modos da representação usual em para síntese de imagem:

- Um objeto não é representado por um conjunto de primitivas de superfície, nem por polígonos ou malhas, mas como primitivas de partículas que definem seu volume.
- Um sistema de partícula não é uma entidade estática, suas partículas mudam de forma e movimento. Novas partículas podem ser criadas, e antigas destruídas.
- Um objeto representado por um sistema de partícula não tem sua forma e aspecto completamente especificados. Processos estocásticos podem ser usados para criar e mudar a forma e aparência do objeto.

Cada partícula tem como atributos sua posição; velocidade, cor, transparência, tamanho, forma e tempo de vida.

Um sistema de partícula possui uma série de parâmetros que controla a posição inicial da partícula. Esses são:

1. A origem da partícula: x , y e z ;
2. Dois ângulos de rotação que definem sua orientação;
3. Um gerador de formas (shape) que define a forma da região em torno da origem da partícula com que novas partículas são gerados, por exemplo, uma esfera de raio R . O gerador de formas descreve a direção inicial das novas partículas, por exemplo, para uma esfera, as partículas sairiam da origem em to-

das as direções. Para uma forma plana, por exemplo, um círculo no plano x-y, as partículas subiriam e se afastariam do plano.

A velocidade de uma partícula pode ser dada por:

$$\text{Velocidade} = \text{VelocidadeMédia} + \text{Rand}() \times \text{VariaçãodaVelocidade}.$$

onde a função $\text{Rand}()$ é responsável pela introdução de uma aleatoriedade no sistema.

A cor, e o brilho são dos atributos mais importantes, representáveis por:

$$\text{Cor} = \text{CorMédia}(R,G,B) + \text{Rand}() \times \text{VariaçãodaCor}(R,G,B)$$

A transparência pode ser dada por:

$$\text{Transparência} = \text{TransparênciaMédia}(R,G,B) + \text{Rand}() \times \text{VariaçãodaTransparência}(R,G,B)$$

O tamanho representa quanto maior ou menor a partícula será quando comparada a um tamanho médio:

$$\text{Tamanho} = \text{TamanhoMédio} + \text{Rand}() \times \text{VariaçãodoTamanho}$$

Esse parâmetro também se relaciona com a forma de cada partícula.

Tempo de Vida

Cada partícula segue três fases distintas: geração, dinâmica e extinção.

- **Geração** – As partículas no sistema são geradas randomicamente dentro de um local predeterminado chamado objeto emissor. O objeto emissor pode ter sua forma alterada com o passar do tempo. Cada um dos parâmetros mencionados anteriormente recebe um valor inicial. Esses valores iniciais podem ser fixos ou podem ser determinados por um processo estocástico.
- **Dinâmica de partícula** – Os atributos de cada uma das partículas podem variar com o passar do tempo. Por exemplo, a cor de uma partícula em uma explosão pode ficar mais escura conforme ela se afasta do centro da explosão, indicando que está se apagando. Em geral, cada um dos atributos de uma partícula pode ser especificado por uma equação paramétrica considerando o tempo como parâmetro. Os atributos da partícula podem ser funções do tempo ou de outros atributos da partícula. Por exemplo, a posição que partícula vai estar dependente da posição prévia da partícula, da sua velocidade e do tempo.

- **Extinção** – Cada partícula tem dois atributos que lidam com o seu tempo de existência: idade e tempo de vida. A idade é o tempo que a partícula tem de existência (medida em quadros), esse valor será sempre inicializado com 0 quando a partícula for criada. O tempo de vida é a quantia máxima de tempo que a partícula pode viver (medida em quadros). Quando a partícula atinge sua idade máxima, ela é destruída. Além desse fator, outros critérios podem contribuir para a destruição prematura da partícula:
- **Fora dos limites** – Se uma partícula sair da área de visualização e não reentrar, então não existe nenhuma razão para manter a partícula ativa.
- **Atingindo o Solo** – Podemos assumir que as partículas que atingem o solo resfriam-se e não podem ser mais vistas.
- **Algum atributo alcança um limite** – Por exemplo, se a cor da partícula está tão próxima do preto que não possa mais ser vista, então ela pode ser considerada destruída.

4.12.11. Renderizando as Partículas

Quando temos de renderizar milhares de partículas, algumas suposições têm de ser feitas para simplificar o processo. Primeiro, cada partícula é renderizada por uma primitiva gráfica. Partículas que ocupam os mesmos pixels na imagem são aditivas, a cor de um pixel é simplesmente a soma dos valores de cor de todas as outras. Por causa dessa suposição, não será necessária a utilização de algoritmos de superfícies escondidas, as partículas são simplesmente renderizadas em ordem. Além disso, partículas podem obscurecer outras partículas, podem ser transparentes e podem lançar sombras em outras partículas ou primitivas convencionais.

Uma segunda aproximação é que as partículas são fontes de luz, que se combinam de acordo com sua cor e valores de opacidade (propriedade que também elimina o problema de superfícies escondidas).

4.12.12. Características dos Sistemas de Partículas

Os sistemas de partículas podem resultar em grandes quantidades de código e uma diminuição na velocidade de render (frame rate). Grande parte desse problema é causado por estruturas de dados complexas e o conseqüente, gerenciamento da memória.

Uma das coisas mais importantes é que os sistemas de partículas aumentam o número de polígonos visíveis por quadros (frames). Cada partícula provavelmente necessita de quatro vértices e dois triângulos. Desse modo, por exemplo, com 2.000 partículas de floco de neve visíveis em uma cena, estamos adicionando 4.000 triângulos visíveis (isso não quer dizer que os sistemas de partículas não possam ser usa-

dos para aplicações em real-time). Como a maioria das partículas se movimenta, não há como pré-calcular o buffer dos vértices, devendo ser atualizado a cada quadro. Para otimizar esse processo, será necessário evitar uma série de processos. Por exemplo, se uma partícula morre, em vez de retirá-la da memória, podemos simplesmente mudar o indicador, que passará a indicar sua morte. Esse procedimento, além de ser mais rápido, poderá manter o sistema de partículas vivo.

Outra característica comum dos sistemas de partículas é a capacidade de estabelecer um elo com objetos da cena, por exemplo, utilizando um emissor de partículas na ponta de um cigarro, esse emissor deve acompanhar o cigarro caso a cabeça do fumante se movimente.

RESUMO

Neste capítulo viu-se diversas formas de modelagens sintéticas. Este é um campo onde muitas pesquisas ainda são necessárias em especial das formas que não tem geometria definida.

CAPÍTULO 5

Cores

- 5.1.** Sistema Visual Humano
- 5.2.** Descrição da Cor da Luz
- 5.3.** As Ondas Eletromagnéticas
- 5.4.** Sistemas de Cores Primárias
- 5.5.** Sistemas de Cores Aditivas
- 5.6.** Sistemas de Cores Subtrativas
- 5.7.** Famílias de Espaços de Cor
 - 5.7.1.** Modelo fisiológico
 - 5.7.2.** Modelo baseado em medidas físicas
 - 5.7.3.** Modelo de sensações oponentes
 - 5.7.4.** Modelo psicofísico
- 5.8.** O Modelo RGB
- 5.9.** O Modelo CMYK
- 5.10.** Espaço de percepção subjetiva
- 5.11.** O Modelo HSV
- 5.12.** O Modelo HLS
- 5.13.** O Modelo YIQ
- 5.14.** Transformação dos Espaços de Cor
 - 5.14.1.** De RGB para XYZ
 - 5.14.2.** De RGB para Eixos-Oponentes
 - 5.14.3.** De RGB para HSV
- 5.15.** Uso de Cores nas Imagens

- 5.16.** Descrição das Cores
- 5.17.** Histograma de Cores
- 5.18.** Espaço de Cores Oponentes
- 5.19.** Ilusões Relacionadas às Cores
- 5.20.** Problemas com Cores na Computação
- 5.21.** Cores em OpenGL
 - 5.21.1** Transparência

A cor exerce uma ação tríplice: a de impressionar, a de expressar e a de construir. A cor é vista quando impressiona a retina. É sentida e transmitida como uma emoção. É construtiva, pois tendo um significado próprio, possui valor de símbolo, podendo assim, construir uma linguagem que comunique uma idéia.

O uso da cor na computação gráfica apresenta vários aspectos interessantes: melhora a legibilidade da informação, possibilita gerar imagens realistas, permite indicar mecanismos de segurança, permite focar a atenção do observador, permite passar emoções e muitos mais. Enfim, o uso de cores torna o processo de comunicação mais eficiente.

O conjunto de técnicas que permite definir e comparar cores é chamado de colorimetria. Esta técnica estuda e quantifica como o sistema visual humano percebe a cor. Também chamamos colorimetria os métodos de análise espectrofotométricos de absorção da luz.

As técnicas de colorimetria têm como referência o chamado observador padrão ou observador médio, que é determinado a partir de experimentos. A colorimetria baseia-se na premissa de que qualquer cor pode sempre ser definida por três parâmetros: a intensidade, a tonalidade cromática e a saturação. O primeiro parâmetro mede a luminância (intensidade luminosa) da superfície examinada, se a superfície for emissora ou refletora de luz podemos também o chamar de brilho ou claridade. A tonalidade cromática caracteriza o comprimento de onda dominante da cor, sendo também chamado matiz. O último parâmetro, saturação, mede a pureza da cor, isso é, o quanto ela é saturada de um só tom. No caso de cores originárias de objetos emissores (luzes), como a mistura de todas as cores puras resulta em uma luz branca, é possível dizer que a saturação caracteriza a quantidade de branco da cor.

Os aparelhos que permitem a determinação das componentes ou coordenadas tricromáticas de um estímulo de cor, sob condições de iluminação e observação definidas, são chamados colorímetros.

5.1. SISTEMA VISUAL HUMANO

A retina do olho humano contém dois tipos de células que detectam a luz e a transformam em impulsos nervosos. Devido à sua forma, elas são chamadas de **cones e bastonetes** (Figura 5.1). Os **cones** são aproximadamente seis a sete milhões e estão concentrados principalmente no **centro da retina na região chamada fóvea** (Figura 5.2). Eles são sensíveis a alto nível de iluminação e responsáveis pela **percepção das cores**. A retina é a área do olho que recebe a luz, transforma em sinais nervosos e transmite para o cérebro através dos nervos óticos. Os **bastonetes** são aproximadamente 125 milhões em cada olho e estão concentrados na **periferia da retina**. São sensíveis a baixo nível de iluminação, distinguem os tons de cinza e são responsáveis pela visão periférica. Os bastonetes são cerca de 100 vezes mais **sensíveis à luz** que os cones (isto é, podem ser sensibilizados por uma pequena quantidade de energia luminosa) mas não



FIGURA 5.1. *As células cone e bastonete.*

têm a capacidade de distinguir cores. Os animais em geral possuem uma quantidade muito maior do que os humanos dando-lhes uma visão noturna maior.

A retina possui um tempo de saturação; assim, os fenômenos visuais observados dependem do tempo de exposição da retina e da intensidade luminosa (Figura 5.2). No fundo do olho, existe uma região denominada de ponto cego, onde as células nervosas da retina se ligam ao nervo ótico. É por esse nervo que as impressões visuais são transmitidas ao cérebro. Na parte externa da retina, existe uma fina camada de cor vermelha escura chamada coróide, cujos vasos sanguíneos alimentam as diversas camadas do olho. A coróide apresenta células de pigmento escuro, cuja função é enfraquecer a intensidade de luz que chega aos cones e bastonetes.

O olho usa substâncias químicas fotossensíveis existentes nas células da retina para transformar a luz em impulsos nervosos enviados ao cérebro. Cada bastonete contém milhões de moléculas de um pigmento sensível à luz chamado rodopsina (ou púrpura visual). Quando a luz incide sobre uma molécula de rodopsina, ela gera um minúsculo sinal elétrico. Esses sinais são acumulados até serem suficientes para desencadear uma mensagem nas células nervosas da retina. A rodopsina é produzida à noite, num processo que utiliza a vitamina A. Durante o dia, a rodopsina é gradualmente consumida pela visão. Por isso, a falta de vitamina A pode levar a deficiência visual em condições de pouca luz, conhecida como cegueira noturna. Uma luz brilhante incidindo durante muito tempo (como na neve ou deserto) nas células fotossensíveis da retina, “gastam” os pigmentos visuais fazendo-os deixar de reagir à luz; após algum tempo a pessoa não consegue enxergar (a chamada “cegueira da neve”). Os cones são menores, mais espessos e reagem à luz produzindo pigmentos quatro vezes mais rápido do que os bastonetes. A adaptação gradativa do olho à escuridão deve-se à lenta passagem do pigmento escuro para o fundo da retina. Quando passamos imediatamente de um ambiente escuro para o claro, nossa visão também necessitará de um tempo até que possamos ver todas as cores, porém esse tempo será menor do que a reação dos bastonetes.

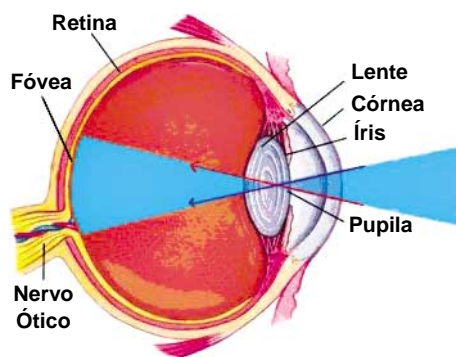


FIGURA 5.2. O sistema visual humano.

A cegueira às cores, que ocorre em 1 a cada 40 mil pessoas, e mais especificamente em cerca de 8% dos homens e apenas 0,5% das mulheres, é devido ao não funcionamento ou falta de um tipo de cone. A forma mais comum de deficiência à visão colorida é chamada daltonismo. O daltonismo pode se manifestar em um, dois ou três tipos de receptores. Os tricromatas são daltônicos que possuem os três tipos de pigmentos, mas que os utilizam em proporções diferentes das pessoas normais e das outras pessoas com o mesmo defeito. Os dicromatas percebem as cores defeituosamente porque combinam apenas dois tipos. Os monocromatas percebem apenas graduações de claro e de escuro, pois sua estimulação visual se baseia em um único tipo de célula. A ausência dos cones responsáveis pelo registro das cores vermelha e verde leva a uma forma de daltonismo chamada anomalia.

Para detectar a forma de um objeto colorido, o observador deve fixar o conjunto de receptores de seu olho nos contornos do objeto. O contorno é o elemento básico na percepção da forma e pode ser criado a partir de áreas adjacentes que diferem em brilho, cor ou ambos. Na Figura 5.3A, o topo do cilindro possui uma extremidade que se confunde com a faixa. Na Figura 5.3B, os elementos da extremidade foram retirados impossibilitando a compreensão dos elementos que compõem a cena.

De acordo com a teoria tricromática ou dos três estímulos, de Thomas Young, (1773-1829) a retina é formada por três tipos de fotopigmentos capazes de receber e transmitir três sensações distintas. Um dos grupos de fotopigmentos é mais sensível aos comprimentos de onda curtos; esses pigmentos não se sensibilizam por comprimentos de onda que excedam o valor de $5.29 \times 10^{-7} \text{m}$; possuem pico de resposta próximo a $4.45 \times 10^{-7} \text{m}$, motivo pelo qual são popularmente conhecidos como fotopigmentos azuis. Eles permitem ver azuis e violetas.

Os outros dois grupos são mais sensíveis aos comprimentos de onda de $5.35 \times 10^{-7} \text{m}$ e $5.75 \times 10^{-7} \text{m}$, mas esses, ao contrário dos azuis, são sensíveis também a todos os demais comprimentos. Assim há uma complementação das cores que vemos. Devido a sua maior sensibilidade para os comprimentos citados, eles são cha-

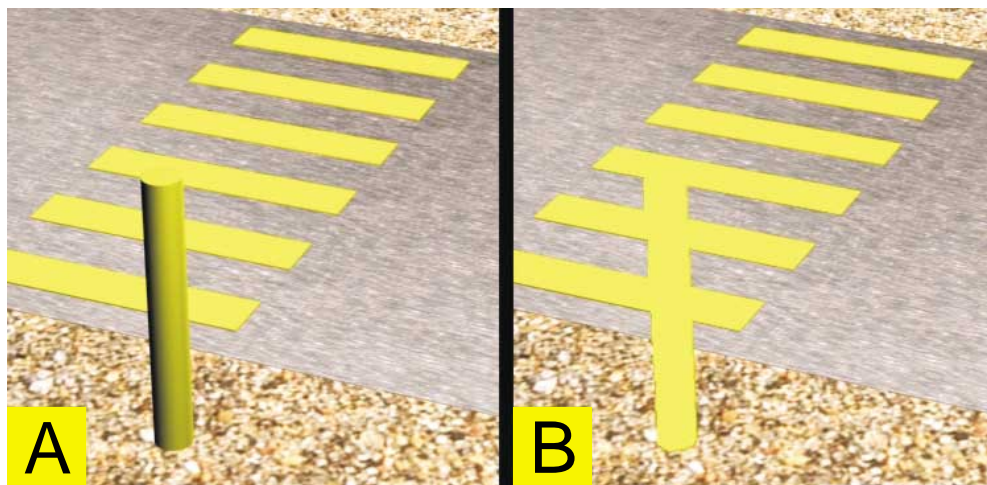


FIGURA 5.3. A extremidade como elemento básico na percepção da forma.

cados de fotopigmentos verdes e vermelhos, respectivamente: a denominação de vermelho não é muito apropriada já que $5.75 \times 10^{-7} \text{m}$ se aproxima mais do amarelo.

A área que enxergamos é chamada campo visual. Em cada olho existem campos visuais separados para os vários tipos de cones e bastonetes, pois essas células têm disposição diferente na retina. O campo visual dos bastonetes é maior, ainda que fraco, no centro da visão, pois no centro da retina existem poucas células desse tipo. O campo visual dos cones verdes é o menor de todos, porque eles se localizam muito próximos e no centro da retina. Assim, só temos visão exata das cores quando olhamos de frente para um objeto e a sua imagem incide na região central da retina (fóvea). Os bastonetes espalhados pelo resto da retina vêem principalmente movimentos em forma de cinza. Essa é a chamada visão periférica.

Young, no século XIX, através de seus experimentos com superposição de luzes, mostrou que **todas as cores do espectro visível podiam ser representadas como uma soma de três cores primárias**. Ele concluiu que isso era consequência, não das características do raio luminoso, mas da composição do sistema visual humano. Ele pressupôs que o raio luminoso era transportado para o cérebro através de três diferentes tipos de impulsos nervosos, que transportavam, respectivamente, o vermelho, o verde e o azul-violeta.

O físico alemão **Hermann von Helmholtz** (1821-1894) prosseguiu nos estudos da teoria de Young e propôs que o **olho continha apenas três tipos de receptores de cor**, que respondiam mais fortemente aos comprimentos de onda **vermelho (R), verde (G) e azul-violeta (B)**. Ele deduziu, ainda, que cada tipo de receptor deveria possuir grande sensibilidade à incidência luminosa, porém, com diferentes pontos máximos. A 480 nm a média da resposta R:G:B seria 1:5:9, dando uma sensação azul-ciano. Enquanto a 570 nm a média seria 7:7:2 que daria uma sensação de amarelo. A percepção da cor, portanto, seria determinada pela média das três respostas (Figura 5.4).

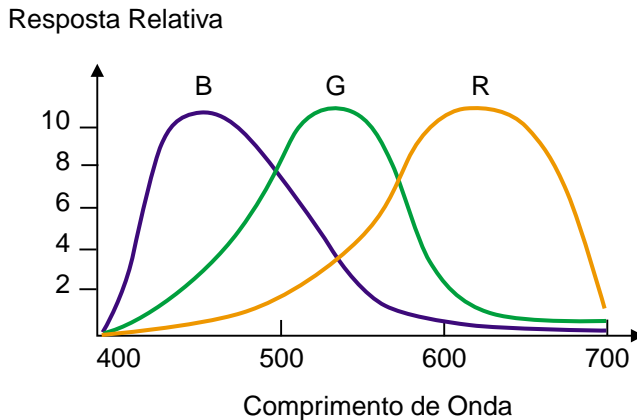


FIGURA 5.4. A percepção da cor pode ser determinada pela média das três respostas RGB.

5.2. DESCRIÇÃO DA COR DE UMA LUZ

O que vemos como cores em uma luz são, na verdade, diferenças de comprimento de onda. Muitas cores podem ser geradas por um único comprimento de onda, como as luzes vermelha e verde puras. Outras cores só podem ser produzidas por luzes com vários comprimentos de onda, como roxo ou rosa. É possível descrever os comprimentos de onda contidos numa cor através de sua curva espectral, que é um gráfico onde os comprimentos de onda estão no eixo horizontal e a intensidade de luz no eixo vertical. **A luz que tem toda sua energia concentrada em um único comprimento de onda é chamada de pura.** É o caso da luz na Figura 5.5.A. Cores que são produzidas por um único comprimento de onda têm toda sua intensidade concentrada num único ponto do eixo horizontal (Figura 5.5.A), enquanto as outras cores possuem uma mistura de comprimentos de onda, como mostra a Figura 5.5.B.

Um problema na descrição das cores é que muitas densidades espectrais podem ser vistas como a mesma cor pelo olho humano. Distribuições espectrais diferentes que produzem a mesma sensação de cor são chamadas metâmeros. Isso torna difícil a descrição de cores através de curvas espectrais. Para contornar esse proble-

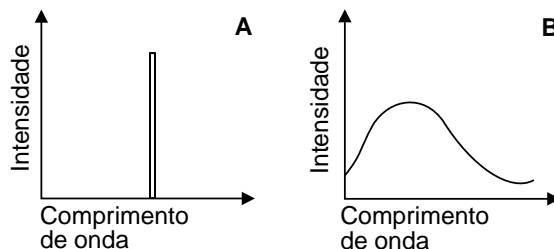


FIGURA 5.5. Comprimento de onda da luz pura em A e comprimento de uma luz misturada em B.

ma, uma forma mais concisa de especificar cada cor é a caracterização na curva espectral dos três parâmetros da colorimetria já comentados: matiz, saturação e brilho. O matiz de uma cor (comprimento de luz predominante) aparece como a posição do eixo horizontal onde a curva de densidade espectral tem mais intensidade. A saturação é a proporção da cor que não se manifesta como uma constante em todos os comprimentos de onda. Para dessaturar a cor, é necessário apenas adicionar luz branca, que é composta de todos os comprimentos de onda. Uma cor dessaturada, como o rosa-claro, é descrita com matiz (vermelho) misturado com uma certa porcentagem de branco. Finalmente, o brilho é a área embaixo da curva de densidade espectral, que determina o quão brilhante a cor resultante será. A Figura 5.6 mostra uma curva fictícia onde esses elementos podem ser caracterizados. Neste gráfico, quanto menor o “ i_b ”, mais saturada a cor. A matiz corresponde ao ponto do eixo horizontal onde ocorre a intensidade máxima em “ i_m ”.

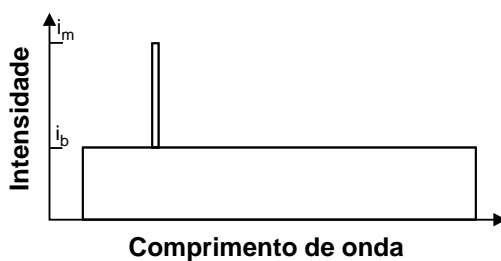


FIGURA 5.6. A área embaixo da curva de densidade espectral determinando o quão brilhante a cor resultante será.

As diferentes cores, ou espectros luminosos, que podem ser percebidos pelo sistema visual humano correspondem a uma pequena faixa de frequências do espectro eletromagnético, que inclui as ondas de rádio, microondas, os raios infravermelhos, os raios ultravioleta, os raios X e os raios gama, como mostrado na Figura 5.7.

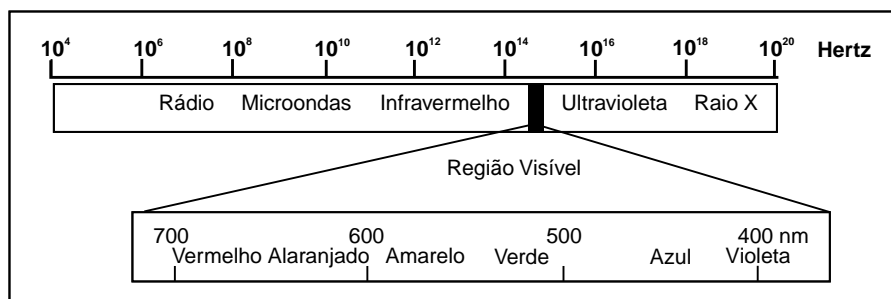


FIGURA 5.7. Frequências do espectro eletromagnético.

A frequência mais baixa do espectro visível corresponde à cor vermelha (4.3×10^{14} Hertz) e a mais alta à cor violeta (7.5×10^{14} Hertz). Os valores de frequência intermediários correspondem às cores que passam pelo alaranjado e amarelo e por todas as outras cores, até chegar nos verdes e azuis.

Os comprimentos de onda maiores possuem distâncias focais maiores e, conseqüentemente, requerem maior curvatura da lente do olho para serem focalizados, (a cor vermelha possui a maior distância focal e a azul, a menor). As utilizações simultâneas de cores localizadas em extremos opostos do espectro fazem com que a lente altere o seu formato constantemente, causando cansaço no olho.

5.3. ONDAS ELETROMAGNÉTICAS

A luz é uma onda eletromagnética que permite ver o mundo ao redor. As ondas eletromagnéticas cobrem um intervalo bem amplo de frequência (Figura 5.7), ou comprimentos de onda. Chama-se frequência o número de ciclos da onda por segundo. Comprimento de onda é a distância percorrida entre dois ciclos. Assim, o comprimento de uma onda é igual à sua velocidade de propagação em um meio (no ar 300.000 km/s, na água 225.000 km/s) dividido pela sua frequência. Dependendo de suas frequências, são classificadas como ondas de rádio, microondas, infravermelho, luz visível, raios ultravioletas, raios X e raios Gama.

Desde a antiguidade, a luz vem sendo investigada, porém fenômenos luminosos inexplicados ainda existem. Na verdade, a luz é uma das diversas formas que a energia se manifesta.

Até o século XVII, a luz era definida como: *a radiação que pode ser percebida pelos olhos humanos* [Halliday e Resnick, 1976]. Após experiências sobre as propriedades dos raios infravermelhos e ultravioletas (invisíveis), a ciência passou a considerá-los como luz, em virtude da semelhança do comportamento de suas regiões do espectro com a faixa visível.

Os babilônios já sabiam muito sobre a propagação da luz, mas foi a Escola de Platão que primeiro teorizou esse conhecimento, criando a base da Ótica Geométrica (parte da Física que estuda a luz se deslocando em linha reta, podendo ser representável por raios).

Modificações substanciais no estudo da luz somente ocorreram dois mil anos mais tarde, com os trabalhos de René Descartes e principalmente de Isaac Newton, no século XVIII, quando então surgiu a Ótica Física ou Ondulatória. Os modelos clássicos usados para descrever os fenômenos luminosos são denominados corpuscular e ondulatório.

O modelo corpuscular considera que a luz é formada por pequenos corpúsculos (fótons) que se propagam em grande velocidade e em linha reta em todas as direções. Essa teoria explica fenômenos como o da propagação e da reflexão da luz, mas deixa outros sem explicação.

No modelo ondulatório, a luz parte de uma fonte luminosa em movimento ondulatório e não precisa de nenhum meio material para se propagar, ou seja, trata-se de uma onda eletromagnética. Essa teoria permite explicar a propagação, a reflexão, a refração e a difração entre outros fenômenos luminosos. Atualmente, o modelo onda-corpúsculo é o mais aceito. Ele engloba os dois modelos clássicos e foi apresentado nas primeiras décadas do século XX pelo cientista Francis Louis Broglie (1892-1987). Segundo Broglie, a luz tem ambos os comportamentos conforme se observa o fenômeno.

A teoria de Newton também está baseada na emissão corpuscular. Newton fez incidir a luz do sol sobre um prisma e observou que ela se decompunha em um arco-íris de cores, que ia do azul ao vermelho. Ele foi capaz de combinar cores para formar outras cores distintas, concluindo que eram necessárias sete cores para representar todas as cores visíveis.

Posteriormente, ao levantarem-se os princípios da teoria ondulatória de Huygens, Young e Fresnel, acreditou-se ser falha aquela teoria, fato acentuado quando Maxwell e Hertz demonstraram ser a luz uma radiação eletromagnética [Feynman et al, 1967]. No entanto, os trabalhos do físico alemão Max Planck, realizados no início do século, mostraram evidências de que a luz é emitida e absorvida em porções de energia perfeitamente definidas, denominadas de “fótons”, partículas que apresentam um “*quanta*” de energia.

Novos aspectos do fenômeno foram revelados, conciliando as teorias de Newton e Planck com as teorias de Maxwell e Hertz. Surgiu uma nova visão: a luz pode assumir tanto um comportamento ondulatório como um comportamento corpuscular.

As diferentes sensações que uma luz pura produz no olho humano, e que produzem as cores, dependem da frequência ou comprimento de onda dessa luz. O olho humano é sensível à radiação eletromagnética na faixa de 400 a 700 nanômetros (Figura 5.4), chamada espectro visível, dentro da qual estão localizadas as chamadas sete cores visíveis, distinguidas por seus respectivos comprimentos de onda.

Espectro Visível

| | | | | | | |
|---------|---------|---------|---------|---------|---------|----------|
| Violeta | Azul | Ciano | Verde | Amarelo | Laranja | Vermelho |
| 380-450 | 540-480 | 480-490 | 490-560 | 560-580 | 580-600 | 600-700 |

Comprimentos de onda em nanômetros

As cores mostradas na tabela acima correspondem ao leque de cores que aparece no arco-íris. A cor vermelha é a que possui o maior comprimento de onda e a violeta, o menor. Essas cores aparecem sempre nessa ordem também quando uma luz branca é decomposta ao atravessar um prisma de cristal. Esse fenômeno é conhecido com dispersão da luz e foi descoberto por Isaac Newton em 1666. Ele mostrou

que a luz branca é a mistura de todas as luzes coloridas do espectro. A cor dos objetos que não emitem luz é percebida pela energia que ela reflete, ou seja, quando a luz incide sobre um objeto, uma parte é absorvida pela própria superfície e a outra é refletida chegando aos nossos olhos.

5.4. CORES PRIMÁRIAS

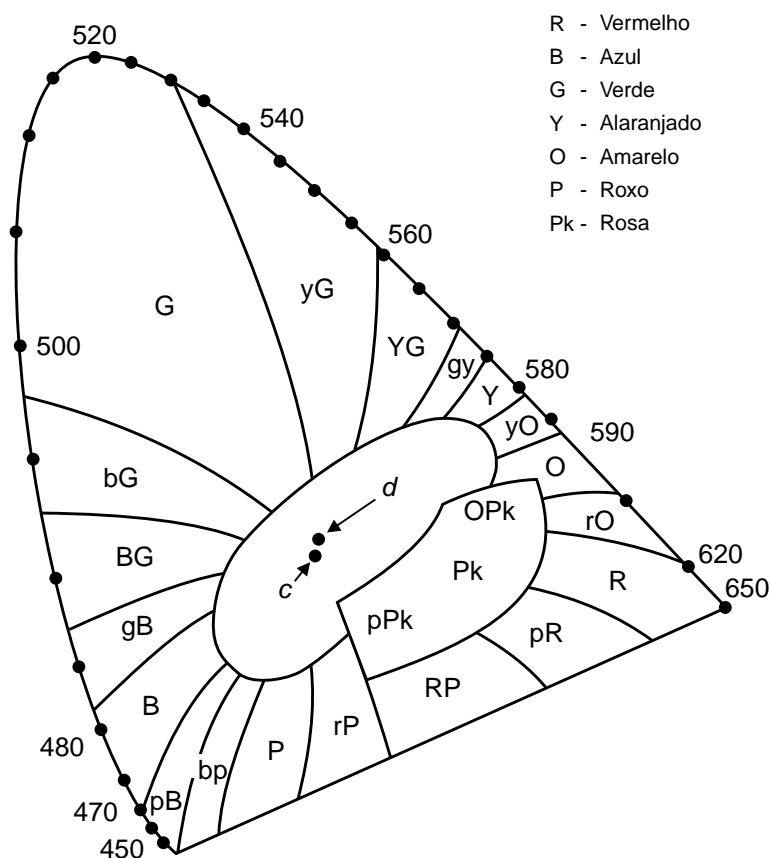
As cores primárias são as cores básicas que podem ser usadas para produzir outras cores. As cores podem ser produzidas a partir de uma combinação das primárias, ou então, da composição de suas combinações. Não existe um conjunto finito de cores primárias visíveis que produza realmente todas as cores, mas sabe-se que uma grande parte delas pode ser produzida realmente a partir de três primárias escolhidas das extremidades e centro do espectro de luzes visíveis como, por exemplo, as luzes vermelha, azul e verde, conhecidas como primárias RGB.

A razão pela qual se usam três primárias é pelo fato de os olhos humanos possuírem três tipos de sensores coloridos diferentes, sensíveis a diferentes partes do espectro de luz visível, como já foi comentado. Eles são os chamados fotopigmentos azul, vermelho e verde. Os picos das respostas desses fotopigmentos se localizam nos comprimentos de onda das luzes azul, verde e amarelo e respondem às sensações luminosas em todo o resto do espectro das radiações visíveis (Figura 5.6).

Um sistema de cores é um modelo que explica as propriedades ou o comportamento das cores num contexto particular. Não existe um sistema que explique todos os aspectos relacionados à cor. Por isso, são utilizados sistemas diferentes para ajudar a descrever as diferentes características das cores e sua percepção pelo ser humano. Existem vários sistemas de cores, sendo aqui apresentados apenas alguns dos principais: o XYZ, o RGB, o HSV e o HLS.

O universo de cores que podem ser reproduzidas por um sistema é chamado de espaço de cores (*color space* ou *color gamut*). Um espaço de cor é um método formal, necessário para se quantificar as sensações visuais das cores, que podem assim ser precisamente especificadas. A introdução de uma representação matemática no processo de especificação de cor gera muitos benefícios já que permite a especificação de uma cor através de um sistema de coordenadas geralmente cartesiano.

Para definir todas as cores visíveis através de radiações primárias, a *Commission Internationale de l'Éclairage-CIE* (Comissão Internacional de Iluminação) definiu três primárias supersaturadas que podem ser combinadas para formarem todas as cores. Embora essas primárias supersaturadas não possuam representação física, elas podem ser usadas para produzir um gráfico com todas as outras cores. Alguns instrumentos que podem medir as especificações de cores CIE, a partir do mundo real, foram desenvolvidos. Uma cópia do gráfico de cores CIE aparece na Figura 5.8.



No gráfico CIE duas cores com iguais distâncias no gráfico nem sempre representam iguais distâncias na percepção [Hill et al, 1997]. Apesar disso, esse gráfico auxilia a vários cálculos. Cada um dos pontos ao longo da borda externa da curva, excluindo os da parte reta, é composto de luzes formadas de um único comprimento de onda. Os dois pontos nomeados próximos ao centro do diagrama, c e d, representam dois tipos muito parecidos de branco. Criando-se uma reta que comece em um dos pontos brancos escolhidos, passe por um outro ponto que represente uma cor e cruze a borda do gráfico, é possível determinar o comprimento de onda primário (matiz) da cor que se localiza neste ponto. A saturação da cor é a razão entre a distância do ponto à borda do gráfico e a distância do ponto à cor branca. E também, se uma linha é desenhada entre quaisquer duas cores do gráfico, todas as cores da linha entre elas podem ser criadas combinando-se essas duas cores na proporção da distância ao ponto desejado. Os sistemas de cores podem ser aditivos ou subtrativos [Wyszecki, Stiles, 1982].

5.5. SISTEMAS DE CORES ADITIVAS

É o sistema usado nos monitores de vídeo e televisões, no qual, a cor é gerada pela mistura de vários comprimentos de onda luminosa provocando uma sensação de cor quando atinge e sensibiliza o olho.

As cores primárias aditivas são: **vermelho, verde e azul**. No processo aditivo, o **preto é gerado pela ausência de qualquer cor**, indicando que nenhuma luz está sendo transmitida; o **branco é a mistura de todas elas, o que indica que uma quantidade máxima de vermelho, verde e azul está sendo transmitida** (Figura 5.9).

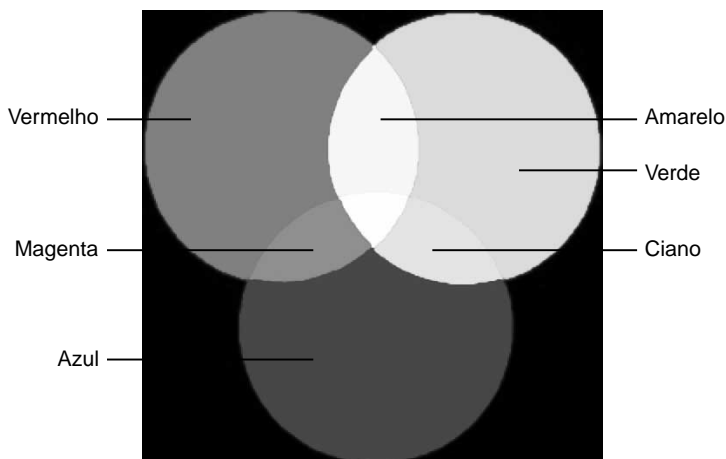


FIGURA 5.9. Processo aditivo das cores primárias (veja também nas ilustrações coloridas).

Em uma imagem colorida, a representação da cor C de cada pixel da imagem pode ser obtida matematicamente por:

$$C = r.R + g.G + b.B$$

Onde R , G e B são as três cores primárias e r , g e b são os coeficientes de mistura correspondentes a cada uma das intensidades associadas a cada um dos canais RGB. Esses coeficientes de mistura podem ser números reais ou inteiros. A primeira forma é mais utilizada para transformação entre espaços de cor, ou consultas em tabelas. A segunda, valores inteiros, é utilizada nas aquisições de imagens digitais e em sua armazenagem na forma de arquivos de imagens. Dessa forma, a cor C de cada pixel da imagem pode ser plotada no espaço de cores RGB usando-se os coeficientes de mistura (r , g , b) como coordenadas. Uma cor, portanto, pode ser considerada como um ponto em um espaço tridimensional (modelada como um subconjunto do espaço R^3 ou N^3), onde cada cor possui uma coordenada (r , g , b).

5.6. SISTEMAS DE CORES SUBTRATIVAS

É o processo usado nas impressoras e pinturas. Uma pintura é diferente de um monitor que, por ser uma fonte de luz, pode criar cores. As cores primárias para objetos sem luz própria são: **magenta, amarelo e ciano**; são cores primárias subtrativas, pois **seu efeito é subtrair, isto é, absorver alguma cor da luz branca**. Quando a luz branca atinge um objeto, ela é parcialmente absorvida pelo objeto. A parte que não é absorvida é refletida, e eventualmente atinge o olho humano, determinando assim a cor do refletida. O processo subtrativo altera a cor através de uma diminuição da luz incidente dos comprimentos que são absorvidos.

No processo subtrativo, o **branco corresponde a ausência de qualquer cor** e o **preto é a presença de todas** (Figura 5.10).

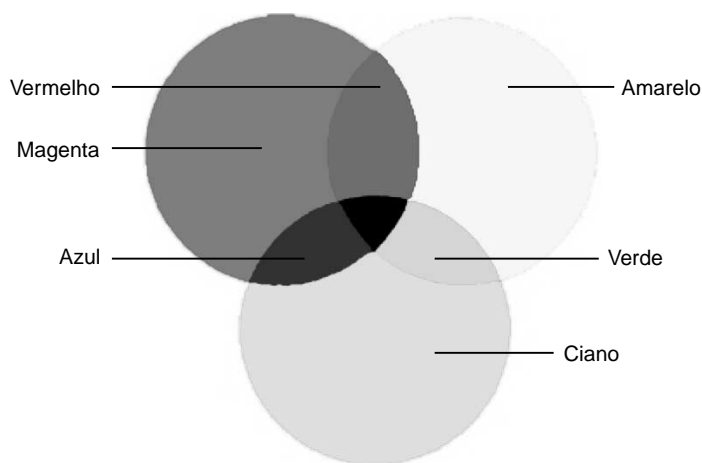


FIGURA 5.10. Processo subtrativo das cores secundárias (veja também nas ilustrações à cores deste livro).

Neste processo uma cor é vista como:

Ciano: se absorve a componente vermelha da luz branca refletida; a luz branca é a soma das cores azul, verde e vermelho; assim, em termos de cores aditivas, ciano é a soma de verde e azul.

Magenta: se retira a componente verde da luz branca, sendo assim, a soma das cores aditivas vermelho e azul.

Amarelo: se subtrai a componente azul da luz branca refletida; é a soma das cores aditivas verde e vermelho.

Podemos ainda obter outros tipos de cores através da combinação das cores primárias. As cores secundárias, por exemplo, são obtidas pela combinação das primá-

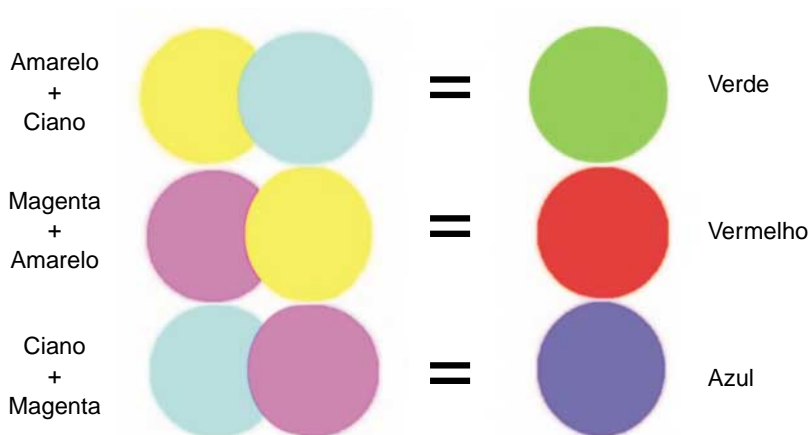


FIGURA 5.11. *Processo de obtenção das cores secundárias (veja nas páginas em cores).*

rias, duas a duas, em proporções iguais (Figura 5.11). As terciárias podem ser obtidas pela combinação de duas primárias em proporções diferentes. A cor marrom, por exemplo, é uma cor terciária obtida da mistura das três primárias podendo ser obtida com a mistura do amarelo ou vermelho alaranjado com um pouco de preto.

5.7. FAMÍLIAS DE ESPAÇOS DE COR

Na literatura sobre visão da cor, quatro famílias de modelos podem ser identificadas: modelo fisiológico; modelo baseado em medidas físicas; modelo de sensações oponentes e o modelo psicofísico.

5.7.1. Modelo Fisiológico

Os modelos baseados nos estudos fisiológicos da visão humana têm como fundamento a existência de três tipos de sensores (cones) encontrados nas células da retina humana, com sensibilidades máximas correspondentes às cores vermelho, verde e azul. Exemplo desses são os modelos RGB usados em monitores de vídeo (CRT) colorido.

5.7.2. Modelo Baseado em Medidas Físicas

Os modelos baseados nas medidas físicas da reflectância espectral são chamados modelos colorimétricos. Utilizam filtros das três cores primárias (vermelho, verde e azul) para composição das cores avaliadas por um observador padrão médio e medidas fotométricas. Exemplo desse modelo é o XYZ (usado no diagrama de cromaticidade do CIE).

5.7.3. Modelo de Sensações Oponentes

O modelo de sensações de cores oponentes foi descoberto a partir de experiências sobre o processo de percepção visual nas camadas mais altas do cérebro, resultante da interação de pares de células oponentes vermelho-verde, azul-amarelo e preto-branco (*red-green*, *blue-yellow*, *black-white*).

5.7.4. Modelo Psicofísico

Modelos de cores fundamentados em reações psicológicas e psicofísicas são oriundos de informações obtidas pelo ponto de vista impressionista da cor (modelos de Munsell e Ostwald) ou de modo experimental (família dos modelos de cor HSB ou de espaço uniforme OSA).

5.8. O MODELO RGB

O modelo RGB possui como primárias as cores aditivas vermelhas (R), verdes (G) e azuis (B), este modelo baseia-se na sensibilidade do olho, e usa um sistema de coordenadas cartesianas R, G, B, cujo subespaço de interesse é o cubo unitário mostrado na Figura 5.12.

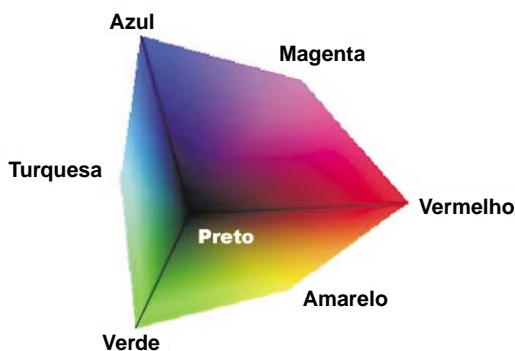


FIGURA 5.12. Subespaço do modelo RGB a partir dos eixos XYZ.

As cores primárias RGB são aditivas. A diagonal principal do cubo, que vai do preto ao branco, possui quantidades iguais de cores primárias e representa a escala de cinza. Cada ponto colorido, dentro dos limites do cubo, pode ser representado por (R, G, B), onde os valores de R, G e B variam entre zero e um, valor máximo.

A resposta do olho aos estímulos espectrais não é linear e, por isso, algumas cores não podem ser reproduzidas pela sobreposição das três primárias. Isso significa que algumas cores existentes na natureza não podem ser mostradas nesse sistema. Por isso, um fenômeno natural colorido como, por exemplo, formação de rochas, não pode ser reproduzido com precisão.

5.9. O MODELO CMYK

O modelo CMYK é um modelo complementar ao modelo RGB, porém destinado a produtos e dispositivos não emissores de luz tais como impressoras. A composição da cor ocorre similarmente ao modelo RGB, porém emprega as cores complementares Ciano (C), Magenta (M), Amarelo (Y) e Preto (K). As cores complementares atuam na luz incidente subtraindo desta as componentes RGB criando assim cores equivalentes às produzidas por dispositivos emissores de luz. Por esse motivo o modelo CMYK é conhecido como um modelo subtrativo onde, além das subtrações efetuadas pelas cores ciano, magenta e amarelo (comentadas na seção 5.6) tem-se o

Preto que subtrai todos os componentes.

Embora complementares, os modelos RGB e CMYK não produzem os mesmos resultados visuais, ou seja, não existe a transposição exata e precisa de cores de um modelo para outro. Existem cores de um modelo que simplesmente não podem ser expressas pelo modelo complementar, enquanto algumas cores encontradas na natureza simplesmente não podem ser expressas por nenhum dos dois modelos (cores metálicas, cores fluorescentes etc.).

Apesar de suas limitações, o par RGB/CMYK é o modelo de cores atualmente mais popular para processamento digital de imagens, principalmente em função da tecnologia dos tubos de imagens atuais que utilizam três canhões de elétrons (RGB) para a composição de imagens coloridas no vídeo, e das impressoras a jato de tinta, laser ou térmicas que utilizam tecnologia CMYK herdada da indústria fotográfica.

5.10. ESPAÇOS DE PERCEPÇÃO SUBJETIVA

Esta forma de modelagem de um espaço de cor é conhecida dos estudos de Munsell e Ostwald. O sistema de Munsell é baseado na concepção de intervalos iguais de percepção visual, onde qualquer cor pode ser definida como um ponto em um espaço de cor tridimensional. Os atributos desse sistema são: o matiz (*H – Hue*), o brilho (*V – Value*) e a saturação (*C – Chroma*) percebidos, cuja notação simbólica é H V C.

O matiz (*Hue*) é representado (Figura 5.13) em uma ordem natural de cores na forma circular: vermelho, amarelo, verde, azul e púrpura, consideradas por Munsell como as principais. Misturando-se cores adjacentes nessa série, obtém-se uma variação contínua entre as cores. A saturação (*Chroma*) é o grau de distância de uma cor qualquer a cor neutra localizada no centro do círculo. O brilho (*Value*), representado pelo eixo perpendicular ao círculo, indica a intensidade de luz refletida de uma cor. O sólido de cor representando o espaço de Munsell é mostrado na Figura 5.14.



FIGURA 5.13. Cores circulares de Munsell (veja nas páginas coloridas).

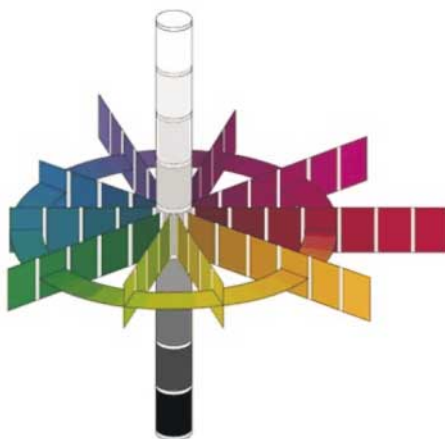


FIGURA 5.14. O espaço de Munsell (veja nas páginas em cores).

Os sistemas de cor de Munsell e Ostwald representam as primeiras tentativas (1915) de organizar a percepção da cor em um espaço. São baseados mais na *subjetividade* da observação do que em medidas diretas ou em experiências controladas da percepção. O sistema de ordenação de cor de Munsell e derivados, como o sistema Pantone e Scottch, continuam sendo muito usados na indústria, não obstante as tentativas de substituí-los pelos modelos colorimétricos.

Os sistemas Munsell e Ostwald são baseados na reflexão (subtrativos) das cores. Embora existam métodos de conversão para outros espaços de cores, estes sistemas não costumam ser usados em trabalhos que utilizem adição de cores. A forma geral desses espaços é a representação por coordenadas cilíndricas com três dimensões correspondendo aproximadamente às variáveis: brilho (eixo vertical), matiz (deslocamento angular em relação a uma determinada cor) e saturação (distância ao eixo vertical), semelhantes ao esquema do sólido de cor de Munsell (Figura 5.14).

Outros modelos, resultantes de medidas experimentais, com características dimensionais semelhantes ao espaço Munsell, mas com configurações diferentes, são muito utilizados em trabalhos de computação visual e gráfica como os sistemas HSV (H – cor, S – saturação, V – intensidade) e HLS (H – cor, L – quantidade de luz, S – saturação) [Dubuisson e Gupta, 2000].

5.11. O MODELO HSV

O modelo HSV (*Hue, Saturation, Value*), orientado ao usuário, foi desenvolvido em 1978 por Alvey Ray Smith que se baseou na maneira com que o artista descreve e mistura as cores. O sistema de coordenadas é um cilindro, e o subconjunto dentro do qual o modelo é definido é um hexágono, ou uma pirâmide de seis lados, como mostrado na Figura 5.15.

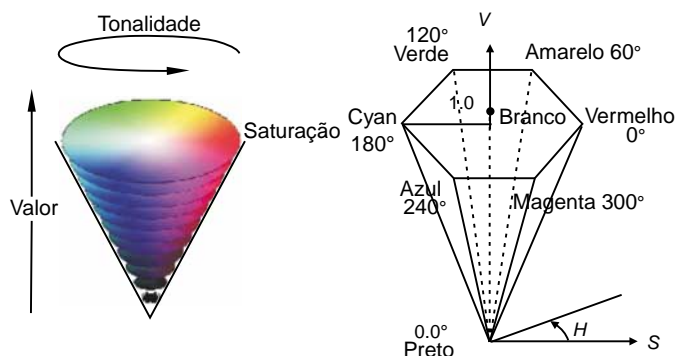


FIGURA 5.15. O modelo HSV (veja também em cores).

O eixo V representa a escala de tons de cinza, onde o ponto $V = 0$ e $S = 0$ corresponde ao preto, e o ponto $V = 1$ e $S = 0$ ao branco. No topo do hexágono encontram-se as cores de maior intensidade.

O matiz, ou H, é dado pelo ângulo ao redor do eixo vertical, sendo o vermelho igual a 0° , o verde 120° e assim por diante. O valor da saturação varia de 0, na linha central (eixo V), até 1, nos lados do hexágono. As cores cujos valores de V e S são igualmente 1, assemelham-se aos pigmentos primitivos usados por artistas.

O sistema HSV utiliza descrições de cor que são mais intuitivas do que as combinações de um conjunto de cores primárias e, por isso, é mais adequado para ser usado na especificação de cores em nível de interface com o usuário.

Os parâmetros de cor utilizados nesse sistema são a tonalidade (*hue*), a saturação (*saturation*) e a luminância (*value*). As várias tonalidades estão representadas na parte superior do cone, a saturação é medida ao longo do eixo horizontal e a luminância é medida ao longo do eixo vertical, que passa pelo centro do cone.

5.12. O MODELO HSL

O modelo HLS (*Hue, Lightness, Saturation*), é um modelo alternativo para o HSV, desenvolvido por *Gerald Murch*, da *Tektronix*, mais ou menos na mesma época. É fácil de ser utilizado, sendo definido por um hexágono duplo, conforme mostrado na Figura 5.16.

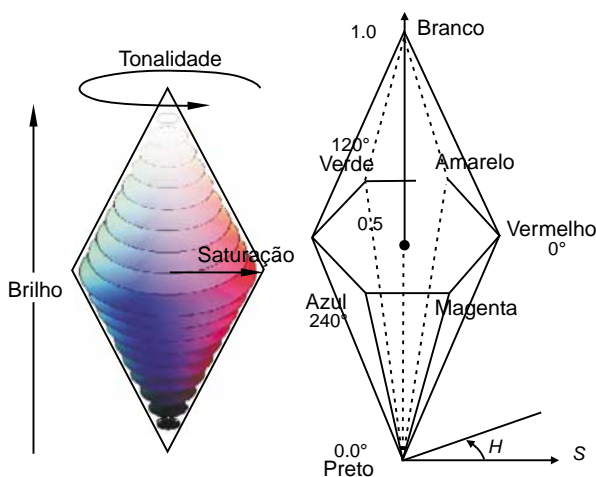


FIGURA 5.16. O modelo HLS (veja páginas em cores).

A escala de “cinzas” encontra-se em $S = 0$; nesse modelo, os matizes com máxima saturação possuem $L = 0.5$, e como no modelo HSV, são definidos na extremidade do hexágono

Vantagens dos Modelos HSV e HLS

Os dois últimos modelos tem as características de:

- Simplicidade e facilidade de implementação; e
- Popularidade entre os programadores de computação gráfica.

Desvantagens dos Modelos HSV e HLS

Quando representados na tela as coordenadas são definidas diretamente em termos dos sinais do monitor RGB, o que faz com que a cor produzida dependa das características do fósforo do monitor. Ao se trocar de equipamento, é possível obter resultados indesejados, uma vez que uma cor não é exatamente a mesma em diferentes dispositivos. Além desta característica presente em qualquer representação em vídeos CRT (seção 1.7), para os sistemas HSV e HSL observa-se ainda que:

- Nenhum dos eixos é uniforme; uma alteração em qualquer coordenada resulta em uma mudança aparentemente variável, de acordo com a localização no espaço de cor. O ângulo de matiz é notavelmente o menos uniforme, com mudanças lentas próximas aos ângulos referentes às primárias RGB e mudanças rápidas entre eles. Isso dificulta a tarefa do operador em precisar o efeito na cor devido a um ajuste dos controles HSV ou HLS.
- Os três eixos não são de fato independentes um do outro. Por exemplo, à medida que se locomove ao redor do plano de cor H-S, a claridade (lightness) aparente no monitor sofrerá mudanças, mesmo que os valores das componentes V (ou L) permaneçam constantes. A maior discrepância ocorre entre o verde e o azul, que tipicamente difere na luminância por um fator próximo a cinco. Já uma mudança em V (ou L) produz normalmente uma alteração na saturação aparente, mesmo que o componente S permaneça numericamente constante.

5.13. O MODELO YIQ

O modelo YIQ é usado na transmissão de televisão a cores nos Estados Unidos. Esse modelo foi projetado para separar a cromaticidade da luminância. Ele foi implementado como solução para compatibilizar os sinais das televisões coloridas com os das preto e branco. O canal Y contém informações de luminância (suficiente para televisões preto e branca) enquanto os canais I e Q transmitem informações de cor. Uma televisão colorida utiliza esses três canais e converte as informações de volta para R, G, e B. Essa conversão acarreta uma variação de cor por tipo de fósforo do monitor, isso significa que, diferentes monitores podem apresentar uma variação nas cores. O espaço YIQ é utilizado pelo padrão americano NTSC (*National Television Standard Committee*).

5.14. ESPAÇO DE CORES OPONENTES

O modelo de cor mais conhecido, por pares de cores oponentes, é originário da teoria do processo oponente descrito nos trabalhos de Hurvich e Jameson [Kayser e Bayton, 1996]. Essa teoria remonta aos estudos da teoria de visão da cor de Hering

(1878), na qual cores oponentes (*opponent hues*) se cancelam quando superpostas. O modelo usa para representação um sistema cartesiano, onde o amarelo e o azul são diametralmente opostos em um eixo, e o verde e o vermelho opostos em outro eixo, formando um plano com o eixo branco e preto perpendicular a esse plano.

Os sinais visuais tricromáticos enviados pelos cones alcançam um estágio neural subsequente (LGN), onde se desenvolvem principalmente dois tipos de processamento da cor: (1) processos espectralmente oponentes no qual *vermelho* versus *verde* e *amarelo* versus *azul* ocorrem e (2) processos espectralmente não-opponentes, *branco* versus *negro*.

As experiências demonstram vários fenômenos visuais que não são adequadamente explicados pela teoria tricromática. Exemplos de tais fenômenos são os efeitos notados pela visão após uma observação prolongada de uma das cores oponentes. A retirada do estímulo amarelo, após a sua observação durante algum tempo, provoca o aparecimento de uma sensação visual da cor azul.

Hurvich e Jameson elaboraram um método experimental de cancelamento de cores (*hue cancellation*) com a finalidade de avaliar psicofisicamente a natureza dos processos oponentes da visão colorida [Feynman et al, 1967]. A teoria de cores oponentes considera que os sinais provindos dos três tipos de cones (R,G,B) são combinados em um segundo estágio do processamento neural da visão da cor, para alimentar três canais de cores oponentes. Um canal branco-preto (BW) ou acromático é obtido pela soma dos sinais vindos dos cones R, G e B. Um canal vermelho-verde ($RG = R - G$) é excitado com sinais vindos dos cones R e inibido pelos sinais vindos dos cones G, e o terceiro canal amarelo-azul ($YB = B - (R + G)$) que é excitado pelos sinais vindos dos cones B e inibido por uma combinação de sinais vindos dos cones R e G.

Hurvich e Jameson conseguiram com esse método registrar as quantidades relativas de cada uma das quatro cores básicas presentes em qualquer estímulo espectral. Verificaram que o vermelho e o verde não são vistos simultaneamente no mesmo lugar, ao mesmo tempo; quando misturados produzem o amarelo; não há o verde-avermelhado (*reddish green*). Analogamente, também o azul e o amarelo não são vistos simultaneamente; quando misturados produzem o branco; não há azul-amarelado (*yellowish blue*). O vermelho e o verde se cancelam mutuamente; da mesma forma, o amarelo e o azul. Se a partir de uma mistura verde-azulada (*bluish green*), por exemplo, se quiser obter o verde, é necessário adicionar-se o amarelo puro de modo a cancelar o azul.

A quantidade necessária da cor para cancelamento foi considerada como um indicador do grau da cor cancelada. Estes dados foram usados para produzir as *curvas de processamento das cores oponentes*, a Figura 5.17 ilustra essas curvas.

Para se obter o vermelho puro, é necessário misturar um pouco de azul (em um comprimento de onda, por exemplo, 700 nm) de modo a cancelar o amarelo que ocorre nessa faixa. Uma cor só é perfeitamente pura quando somente ela é percebida.

da. Uma cor pura somente ocorre onde o valor cromático das outras duas é zero. No gráfico da Figura 5.17, os valores positivos e negativos são arbitrariamente assinalados para representar a natureza do processo das cores oponentes.

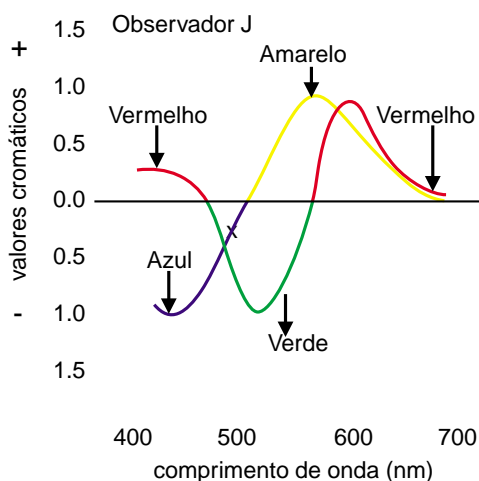


FIGURA 5.17. Curvas de processamento das cores oponentes.

Os valores cromáticos representam as quantidades de cores vistas em diferentes comprimentos de ondas. Por exemplo, no comprimento de onda 450 nm, o valor cromático azul é aproximadamente $-1,0$ e o valor da cor vermelha próximo a $0,2$. Esses valores indicam que a cor percebida seria azul com um pouco de vermelho. Essa cor é conhecida como violeta ou púrpura. Já em 550nm, há uma grande quantidade de amarelo e verde indicando que na faixa em torno dessa cor apareceria um amarelo-verde-claro, que é o que a maioria das pessoas vê.

5.15. TRANSFORMAÇÃO ENTRE ESPAÇOS DE COR

Há uma grande variedade de espaços de cor, cada um deles com vantagens e desvantagens quanto à capacidade de representação de cores distintas e suas composições.

Geralmente, as imagens coloridas são capturadas por scanners ou câmeras que as armazenam em forma de pontos com a informação das cores representadas pelos canais RGB. Esse espaço seria o candidato natural, por ser *completo*, para ser usado em pesquisas por similaridade, caso fosse perceptualmente *uniforme*. Como espaço completo, o RGB é ponto de partida para transformações para outros espaços completos que serão sujeitos a avaliação quanto ao processamento por semelhança [Conci e Castro, 2002]. Sendo o RGB um espaço contínuo, em trabalhos computacionais pode ser amostrado em um grande número de valores discretos, imagens

discretizadas em 256 níveis por canal podem reproduzir aproximadamente até 16,7 milhões de cores distintas.

Dada a impraticabilidade de se trabalhar com tão grande quantidade de cores, deve-se reduzir o número de cores possíveis do espaço, de 16,7 milhões a um número menor de cores, para que as imagens possam ser mais facilmente manipuladas.

As transformações a partir do RGB podem ser lineares ou não. A linearidade permite um retorno direto ao RGB o que é interessante em muitas aplicações. Por exemplo: os espaços YIQ (padrão NTSC), YUV (padrão PAL e SECAM), YCrCb (padrão JPEG e MPEG) e o de cores oponentes (OPP) são lineares. Já espaços como HSV, CIE-LAB, CIE-LUV são gerados por transformações não-lineares. As transformações de espaço de cor descritas a seguir são do RGB para: (1) XYZ; (2) OPP – Eixos-Oponentes; (3) HSV – Hue-Saturation-Value.

A transformação do espaço de cor é a primeira parte do processo para gerar as representações nos vários espaços.

5.15.1. De RGB para XYZ

Com o desenvolvimento de vários estudos sobre a percepção das cores (seção 5.1), constatou-se que o olho humano é sensível a três cores primárias (vermelho, verde e azul) que, combinadas formam todas as cores do espectro visível.

A (CIE – *Commission Internationale de l'Éclairage*) estabeleceu um padrão para representar as cores visíveis, denominado XYZ, formado por três coeficientes que representam as porcentagens de vermelho, verde e azul existentes em uma cor.

A partir dessa padronização, vários sistemas de cores foram criados para representar os subconjuntos de cores (*color gamut*) pertencentes ao espaço de cores XYZ. Esses sistemas utilizam pelo menos três componentes para representar uma cor (Ex: Sistema HSB – matiz, saturação e brilho), tomando como base as componentes das três cores primárias e transformando-as em um novo sistema de coordenadas de cores.

Dois sistemas padrão foram criados para definir as componentes das cores vermelho, verde e azul. O primeiro padrão foi criado pelo (NTSC – *National Television Standards Committee*) (para utilização em sistemas de televisão) e o outro padrão foi definido pelo CIE.

Os dois se relacionam através da equação:

$$\begin{bmatrix} R_N \\ G_N \\ B_N \end{bmatrix} = \begin{bmatrix} 0.842 & 0.156 & 0.091 \\ -0.129 & 1.319 & -0.203 \\ 0.006 & -0.069 & 0.897 \end{bmatrix} \begin{bmatrix} R_{CIE} \\ G_{CIE} \\ B_{CIE} \end{bmatrix}$$

onde: R_{CIE} , G_{CIE} e B_{CIE} – são as componentes das cores vermelho, verde e azul do padrão CIE; e

R_N , G_N e B_N – são correspondentes do padrão NTSC.

O padrão NTSC deu origem ao sistema RGB, utilizado para representar o *color gamut* dos monitores de vídeo, cuja distribuição no espaço XYZ é apresentada na Figura 5.18.

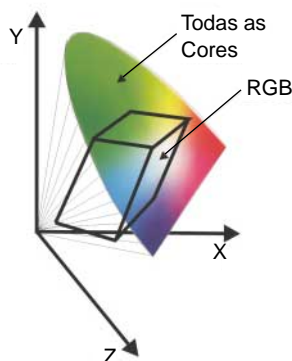


FIGURA 5.18. Distribuição no espaço XYZ do sistema RGB

A conversão do sistema RGB para o sistema XYZ é obtida aplicando a equação:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.607 & 0.174 & 0.201 \\ 0.299 & 0.587 & 0.114 \\ 0.000 & 0.066 & 1.117 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Como o sistema RGB está, diretamente, relacionado ao sistema XYZ, os demais sistemas de cores possuem suas conversões para o sistema XYZ através de uma conversão intermediária para o sistema RGB.

5.15.2. De RGB para Eixos-Oponentes (OPP)

O modelo de cores oponentes explica uma grande quantidade dos fenômenos psicofísicos da percepção da cor. Conforme já mencionado, há evidências de que o sistema de visão dos seres humanos usa um modelo de espaço no qual as respostas aos estímulos das células cones (R,G,B) existentes na retina sejam convertidas, num estágio cerebral mais avançado, para uma combinação de cores oponentes, em um sistema cartesiano, onde as novas coordenadas são chamadas de *rg*, *by* e *wb*. Esse espaço não é uniforme [Kaiser e Bayton, 1996]. O espaço OPP é obtido pela seguinte transformação linear do RGB:

$$\begin{bmatrix} rg \\ by \\ wb \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 \\ -1 & -1 & 2 \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

O eixo *wb* representa o canal de luminância e os outros dois as crominâncias.

5.15.3. De RGB para HSV

O espaço de cor *HSV* é uma forma natural e intuitiva para formação das cores, assim como o *CIE-LAB* e *CIE-LUV*. A transformação do RGB para o *HSV* é geralmente feita na forma não é linear, mas tem a vantagem de que o processo de reversão ao RGB é facilmente realizado.

Para fazer a transformação, usa-se o seguinte algoritmo:

```
//Primeiro calcula-se os valores máximo e mínimo:
max = máximo(R,G,B), min = mínimo(R,G,B)
//depois os valores de saturação e brilho:
V = max, S = (max - min) / max
//aí passa-se a calcular as cores ou H:
if S = 0 /* H irrelevante */
else
  R1 = (max-R) / (max-min)
  G1 = (max-G) / (max-min)
  B1 = (max-B) / (max-min)
  If R1 = max and G1 = min, H = 5 + B1
  else if R1 = max and G1 not = min, H = 1 - G1
  else if G1 = max and B1 = min, H = 1 + R1
  else if G1 = max and B1 not = min H = 3 - B1
  else if B1 = max and R1 = min H = 3 + G1
  else H = 5 - R1
//(converte-se H em graus)
H = H*60
//usa-se H variando de 0 a 360, S e V variando entre 0 e 1.
```

5.16. USO DE CORES NAS IMAGENS

A cor, elemento fundamental em qualquer processo de comunicação, merece uma atenção especial. É um componente com grande influência no dia a dia de uma pessoa, interferindo nos sentidos, emoções e intelecto; pode, portanto, ser usada deliberadamente para atingir objetivos específicos. O uso apropriado de cores pode resultar em uma rápida e correta assimilação da informação.

Como muitas vezes age no inconsciente de uma pessoa, é um parâmetro que deve ser muito bem compreendido; seu uso incorreto pode trazer resultados indesejáveis que nem chegam a ser notados [Levkowitz e Herman, 1992].

O uso de cores em imagens permite:

- Mostrar as coisas conforme são vistas na natureza;
- Representar associações simbólicas;
- Chamar e direcionar a atenção;
- Enfatizar alguns aspectos sociais;
- Determinar um estado de espírito;
- Tornar uma imagem mais fácil de ser memorizada.

Por se tratar de um recurso tão poderoso, o uso de cor deve ser feito com cautela. Ao preparar um trabalho, devemos observar que:

- Uma escolha não adequada de cores pode interferir na legibilidade da imagem;
- As cores podem apresentar características distintas em condições diferentes;
- As cores devem ser selecionadas de modo a não causarem fadiga nos olhos do usuário e nem deixá-lo confuso;
- Se elementos da imagem são agrupados com as mesmas cores; deve-se tomar um cuidado para não agrupar elementos que não possuem nenhuma relação entre si de modo a não induzir o usuário a conclusões erradas.

No que diz respeito aos fatores humanos, associações climáticas podem facilmente ser identificadas na preferência por certas cores. De um modo geral, as pessoas de lugares tropicais gostam mais de cores saturadas e com brilho; já os moradores de regiões mais temperadas possuem uma tendência ao uso de cores sombrias. Isso se deve ao fato de serem essas as cores que elas estão mais acostumadas a ver no seu habitat natural. Outro exemplo de associação que reflete aspectos culturais é a cor branca: no ocidente, ela é associada à pureza e à alegria, sendo muito usada por noivas no dia de seu casamento. No oriente, é a cor da morte e da dor, sendo o vermelho a cor convencional para o vestido de noiva.

5.16.1. Branco

A cor branca é a chave para a comparação de cores; é usada para determinar o conteúdo espectral de uma luz. É a cor que possui a maior leveza, e atrair a atenção em um fundo escuro. Uma imagem com o fundo branco fornece a máxima legibilidade para um objeto escuro, mas seu intenso brilho pode causar problemas ao se olhar para ela por um período prolongado; áreas extensas de branco geralmente resultam em um brilho que dilui as cores exibidas. Devemos evitá-la também nos cantos das

imagens devido à grande sensibilidade do olho a intensidade de luz no campo visual periférico. Para a reprodução de imagens, recomenda-se uma margem estreita (com uma largura próxima a 5 mm) de modo a estabelecer a referência branca para um melhor julgamento da aparência da cor. Em nossa cultura a cor branca está associada a: neve, pureza, inocência, paz, leveza, limpeza, frio, hospital, vulnerabilidade, palidez fúnebre, rendição e esterilidade.

5.16.2. Preto

A cor preta age como um estimulante para as demais cores e se harmoniza bem com todas elas. Apesar de muitas conotações emocionais negativas, sua conformidade faz dela um padrão para várias situações: traje para noite, roupa de nazistas e fascistas, grupo de adolescentes.

Objetos brilhantes aparentam ser mais escuros do que objetos foscos e as superfícies pretas geralmente se tornam mais negras à medida que o nível de iluminação aumenta. Contrastes simultâneos fazem com que um contorno preto torne áreas coloridas mais claras e amplas.

Nos dispositivos de vídeo, o preto é a cor natural de fundo (nenhum sinal no monitor ou cor no pixel). Ela fornece um bom contraste com as cores brilhantes e torna-se mais legível quando em contraste com fundos claros.

A cor preta está associada a: noite, carvão, poder, estabilidade, formalidade, solidez, medo, vazio, morte, segredos, anonimato e maldição.

5.16.3. Cinza

O cinza é uma cor que reduz as conotações emocionais. Combina bem com todas as cores, as quais, por sua vez, apresentam seu colorido máximo quando contrastando com cinza escuro.

Pode ser obtida através da mistura de branco e preto ou de cores complementares; pode também ser gerada através da mistura de pontos adjacentes pretos e brancos, como no caso de um monitor de vídeo, o que provoca um tom de cinza mais vibrante do que quando se trata de um cinza uniforme.

Por ser uma resultante da mistura entre o preto e o branco, a cor cinza não possui uma associação específica.

5.16.4. Vermelho

É a cor com o maior impacto emocional universal, provavelmente devido à sua associação com o sangue e o fogo, portanto, com a guerra. Seu significado simbólico varia com diferentes culturas: na Inglaterra simboliza a realeza; nos Estados Unidos denota perigo, na China representa revolução; na Índia, casamento.

A cor vermelha é uma das três cores primárias usadas nos dispositivos de vídeo RGB. Mostra-se muito eficiente quando usada nas imagens para sinalizar algum perigo ou chamar a atenção, como, por exemplo, bordas vermelhas nos sinais de advertência são rapidamente percebidas. Mas seu uso deve ser evitado em áreas amplas ou para a cor de fundo; trata-se de uma cor dominante e agressiva que chama muito a atenção.

A cor vermelha está associada a: vitória, paixão, amor, força, energia, sexualidade, sangue, guerra, fogo, perigo, raiva e satã.

5.16.5. Amarelo

O amarelo é uma cor incandescente com grande qualidade acolhedora. Sua associação imediata com o sol faz com que ela simbolize a vida e o calor.

Nos terminais RGB, essa cor é produzida pela combinação de vermelho e verde e, em geral, preferencialmente de baixa saturação. Por ser a mais clara de todas as matizes, é um bom indicador de atividade: daí o seu uso em faixas de obstrução policial.

Não deve ser usada como cor de texto, a não ser com fundo preto ou azul escuro.

A cor amarela está associada a: sol, verão, serenidade, ouro, colheita, inovação, covardia, traição, ciúmes, risco, doença e loucura.

5.16.6. Verde

O verde é uma cor intimamente ligada à natureza; é a cor da vegetação e se tornou a marca dos movimentos ambientalistas. A idéia tradicional de que a plantação significa uma certa estabilidade resultou na associação do verde com o sentimento de segurança. Acredita-se que ambientes com um tom de verde-claro promovam um estado de paz na mente mas, verde em excesso pode resultar em uma aparência doentia.

O olho humano é mais sensível aos comprimentos de onda próximos ao amarelo-verde. Assim, essa é a cor mais visível das três cores primárias dos terminais RGB, sendo muito propícia quando se deseja passar rapidamente uma informação.

A cor verde está associada a: vegetação, natureza, primavera, fertilidade, esperança, segurança, decadência, inexperiência, inveja, ganância, fuga da realidade e má sorte.

5.16.7. Azul

Por ser a cor do céu e do mar, o azul sugere ar, espaço e profundidade. É uma cor fria e suave, a mais tranqüila de todas; tem uma grande capacidade de relaxar e tranqüilizar as pessoas. Fornece um bom fundo para cores vividas. É a cor preferida pela moda, em qualquer circunstância social. Simboliza autoridade e espiritualidade, sendo a cor mais amplamente usada nas bandeiras nacionais, mostrando assim, o desejo de unidade e estabilidade.

É uma das três cores primárias dos terminais de vídeo. Como o olho humano é menos sensível aos comprimentos de onda do azul, o azul é uma cor difícil de ser focalizada e de se obter um bom contraste; não deve assim ser usado para texto nem detalhes finos. É uma excelente cor para o fundo, principalmente porque pode-se tirar proveito de sua qualidade de expansão e profundidade infinita.

A cor azul está associada a: céu, mar, espiritualidade, estabilidade, paz, unidade, frio, depressão, melancolia, obscenidade, mistério, conservadorismo.

5.17. HISTOGRAMA DE CORES

A partir de um espaço de cor, determinamos as características de cor de uma imagem com o auxílio de histogramas [Conci e Castro, 1999]. Um histograma de cor de uma imagem caracteriza a sua distribuição de cores. Cada ponto da imagem pode ser representado como um vetor 3D no espaço de cor usado, o RGB por exemplo. Nesse caso, a imagem terá três histogramas, um para cada canal de cor. Definir esses histogramas consiste em primeiro capturar na imagem, em cada pixel, a intensidade de cor no formato RGB, onde o valor de cada ponto da imagem $I[x,y]$ é expresso por um vetor:

$$(V_c) = (I_r[x,y], I_g[x,y], I_b[x,y]), \quad x \in [1,X], y \in [1,Y],$$

onde X e Y são a largura e a altura da imagem. Depois, conta-se o número de ocorrências de cada intensidade para cada canal, o gráfico desses números será o histograma do canal [Han, Ma, 2002].

Na forma mais simples, esses histogramas podem ser 1D. Neste caso, o processo para se obter um histograma de cor consiste em definir as diversas cores presentes na imagem e contar quantos pixels há de cada uma delas. Para isso, primeiro deve-se conhecer a quantidade n , de cores presentes na imagem. Esse mesmo valor, n , será o número de elementos do histograma. Depois, contabiliza-se o número de pontos de cada cor. O gráfico desses números será um histograma 1D com n células, onde cada uma representa uma proporção dos pontos com aquela cor presentes na imagem:

$$h[m_i] = \frac{1}{XY} \sum_{x=1}^{X-1} \sum_{y=1}^{Y-1} f(x,y), \quad f(x,y) = \begin{cases} 1 & \text{se a cor de } (x,y) = m_i \\ 0 & \text{se a cor de } (x,y) \neq m_i \end{cases}, \quad m_i = 1, 2, \dots, n$$

onde X, Y são a largura e a altura da imagem em pixels, m_i é uma coordenada horizontal do histograma, correspondendo à célula de contagem dos pontos de cor m_i e $h[m_i]$ corresponde ao número de pontos de cada cor m_i dividido pelo número total de pixels da imagem.

5.18. ILUSÕES RELACIONADAS ÀS CORES

Aberrações cromáticas fazem com que comprimentos de onda distintos sejam focalizados em diferentes planos dentro do olho. Para a maior parte das pessoas, as cores vermelhas parecem se encontrar mais próximas do observador do que as cores azuis.

Em pintura, existe uma técnica chamada *perspectiva aerial*, na qual se pinta objetos distantes (como montanhas) de azulado devido à absorção da cor amarela ou vermelha pela umidade atmosférica. Tal técnica pode ser aplicada a interfaces: o uso de um azul não-saturado mostra-se uma boa escolha para uma cor de fundo de tela (em decoração, tal cor provoca uma impressão da maior altura ou profundidade de uma sala), permitindo que os demais elementos da tela apareçam com mais destaque.

5.19. PROBLEMAS COM CORES NA COMPUTAÇÃO

Observando o formato do gráfico de cores CIE (Figura 5.8), é evidente que não é possível representar todas as cores do gráfico através da combinação das três cores básicas dos monitores: vermelho, verde e azul. Se as 3 cores aditivas utilizadas como base para combinação de cor forem marcadas neste gráfico, tem-se um triângulo cujo interior define todas as cores que podem ser reproduzidas a partir destas. Essa forma que define as cores possíveis de serem reproduzidas é chamada gamut. Ocorre variação das cores representáveis por duas impressoras diferentes e entre monitores diferentes devido às diferenças na seleção das cores primárias usados e dos métodos de reprodução de cores. Até mesmo em um mesmo dispositivos, as cores podem parecer diferentes devido aos erros de calibragem, desgaste do fósforo diferenças entre as tintas, qualidade dos papéis, e a iluminação do ambiente.

Para reduzir esses problemas existem sistemas de cores que usam amostras (swatches) calibradas para definir cores. Um desses é o sistema Pantone. Quando usam esses tipos de sistema, as aplicações têm a necessidade de saber o número específico de cada amostra (swatch), e a impressora é responsável em reproduzir a cor do swatch da melhor maneira possível. Observando-se o swatch é mais fácil determinar a cor final do que olhando na tela. Alguns sistemas tentam produzir cores Pantone, que parecem melhores nos monitores, usando uma tabela de consulta que define características, como marca e modelo do monitor, juntamente com especificações das condições de iluminação local.

Embora isso funcione bem para o design gráfico de objetos, não acontece o mesmo com imagens bitmap que foram escaneadas ou renderizadas por programas 3D, pois ainda não existe uma maneira realmente eficiente de mapear cores no formato Pantone. Apesar disso, alguns sistemas, como o ColorSync da Apple, possuem perfis de configuração para monitores, scanners e impressoras. O sistema então tenta simular e corrigir as cores, em cada passo, da melhor maneira possível, modificando os dados que são recebidos de scanners, que são enviados para impressoras, ou

que são mostrados nos monitores. Esses sistemas são um avanço na área de produções com alta definição de cores, e estão agora se tornando mais acessíveis para outros usuários (o ColorSync é gratuito para computadores Apple Power Macintosh e Macintosh).

5.20. CORES EM OPENGL

O OpenGL permite também representar informações de transparência, o que é chamado de canal alfa (seção 6.5). Cores são definidas através do comando `glColor` e seus parâmetros são as intensidades das cores RGBA (vermelho, verde, azul e o canal alfa). As cores variam entre 0,0 e 1,0. Sendo para o branco todos os valores iguais a 1 e para o preto todos os valores iguais a 0. A linha que segue produz uma combinação de amarelo vivo com 25% da cor do fundo.

```
glColor4f(1.0, 1.0, 0.0,0.75)
```

Exemplos de outras cores, usando RGB são:

| | |
|----------|---|
| Vermelho | <code>glColor3f(1.0f,0.0f,0.0f);</code> |
| Verde | <code>glColor3f(0.0f,1.0f,0.0f);</code> |
| Azul | <code>glColor3f(0.0f,0.0f,1.0f);</code> |
| Branco | <code>glColor3f(1.0f,1.0f,1.0f);</code> |
| Preto | <code>glColor3f(0.0f,0.0f,0.0f);</code> |

5.20.1. Transparência

Para exibir objetos transparentes em OpenGL, deve-se utilizar as *Funções de Blend* (ou de mistura). Essas funções misturam a cor do objeto que já está na tela com a cor do objeto que está sendo desenhado num dado momento.

Em primeiro lugar, é necessário *ativar as funções* de BLEND. Para tanto, em OpenGL usa-se:

```
glEnable(GL_BLEND);
```

Utilizando a quarta componente de cor, o canal alfa, podemos modificar a transparência de um objeto. Esse quarto parâmetro pode variar entre 0 e 1, sendo 0 um objeto totalmente transparente e 1 um objeto opaco. Para habilitar o processamento dessa quarta componente usamos:

```
glEnable(GL_ALFA_TEST)
```

As funções de BLEND definem como é feita a mistura entre as cores do objeto que está sendo desenhado e as imagens que já estão na tela. Essa mistura é feita a partir de uma *média ponderada* entre o ponto da tela e o ponto do objeto que está sendo desenhado. A função OpenGL para definir esses pesos é “glBlendFunc”.

Essa função recebe dois parâmetros: o primeiro define o peso da cor do novo ponto, e o segundo o peso da cor do ponto que já está na tela. Esses pesos, em OpenGL, serão sempre uma função do nível de transparência do objeto, ou seja, de seu valor alfa. Por exemplo, a chamada:

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Define que a nova cor do ponto na tela será:

$$\text{NovaCorNaTela} = \text{CorDoObjeto} * \text{AlfaDoObjeto} + \text{CorAntigaNaTela} * (1 - \text{AlfaDoObjeto});$$

Nesse caso, as constantes têm o seguinte significado:

GL_SRC_ALPHA: define que o peso da cor do objeto que está sendo desenhado é o próprio valor alfa de sua cor;

GL_ONE_MINUS_SRC_ALPHA: define que o peso da cor que já está na tela é de $(1 - \text{alfa})$, onde alfa é o nível de transparência do objeto que está sendo desenhado.

Exemplo:

```
void display( void )
{
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    // Desabilita o BLEND para o primeiro objeto
    glDisable(GL_BLEND);
    glPushMatrix( );
    glTranslatef(0,0,-1);
    glRotatef ( xrot, 1.0, 0.0, 0.0 );
    glRotatef ( yrot, 0.0, 1.0, 0.0 );
    glRotatef ( zrot, 0.0, 0.0, 1.0 );
    // habilita remoção de faces traseiras
    glEnable(GL_CULL_FACE);
    glCullFace(GL_BACK);
    DrawCube(0.5);
    glPopMatrix( );
    // Habilita o BLEND para o segundo objeto
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

```
// NovaCorNaTela = CorDoObjeto * AlfaDoObjeto + CorAntigaNaTela * (1-AlfaDoObjeto)
DesenhaVidro(AlfaVidro);
xrot+=0.3f;
yrot+=0.2f;
zrot+=0.4f;
glutSwapBuffers( );
}
```

RESUMO

O mecanismo da visão colorida e da sensação de cores é muito complexo. Em computação gráfica a reprodução precisa das cores e diversos outros aspectos a elas relacionados representa um vasto campo de pesquisas. Ao assunto, esse capítulo apenas fornece uma leve introdução. Como leitura adicional veja “Color Correction for an Image Sequence” de P. Pham e G Pringle, IEEE Comp. Graph. and Appl. vol. 15 nº 3, 1995, pp 38-42.

Para um fascinante estudo dos problemas de percepção e distinção entre cor, como uma característica da luz e como uma propriedade percebida nos objetos leia “Color and Perception: the work of Edwinhand in the light of current concepts”, de M. H. Wilson e R. W. Brocklebank, Contemporary Physics, dezembro de 1961.

CAPÍTULO 6

Animação

6.1. Histórico

6.2. Aplicações da Animação

6.3. Animação por Computador

6.4. Formas de Animação

6.4.1. Animação por Quadro-Chave (Keyframe)

6.4.2. Animação por Script

6.4.3. Animação Procedural

6.4.4. Animação Representacional

6.4.5. Animação Estocástica

6.4.6. Animação Straight Ahead

6.4.7. Animação Pose-to-Pose

6.4.8. Animação por Comportamento ou Comportamental

6.4.9. Animação Track Based

6.5. Canal Alpha

6.6. Composição

6.6.1. Filmagem da Cena Real

6.6.2. Reconstrução da Cena em 3D

6.7. Captura de Movimento

6.7.1. Rotoscopia

6.7.2. Sistemas de Captura

6.7.2.1. Ótico

6.7.2.2. Mecânico

6.13. A Animação no Processo de Aprendizado

Animação: do Latim. *Animare*, dar vida, movimento, coragem, entusiasmo, alma.

A transcrição de um dicionário acima, não só define como também descreve os passos para produção de animação em computação gráfica. Primeiro tentamos desenhar ou esculpir, depois capturar os movimentos e, por último, retratar o espírito da criatura ou cena a ser animada.

6.1. HISTÓRICO

Os primeiros registros de animação datam de 2000 a.C. quando os egípcios pintavam nas paredes seqüências de lutas e cenas de adoração.

Somente em 1828, a animação pôde ser explicada pelo entendimento do princípio fundamental do olho humano: a persistência da visão. Esse princípio foi demonstrado por Paul Roget Frenchman, com a invenção do thaumatrope, um disco colocado em uma haste. Thaumá é uma palavra grega que significa magia e deu origem a thaumaturgy, a arte de fazer mágica ou arte miraculosa. No português, tau-maturgo é aquele que faz milagres. De um lado do disco do thaumatrope, Frenchman pintou um pássaro e do outro uma gaiola vazia. Quando o disco girava, o pássaro aparecia dentro da gaiola. Isso deu origem a diversos dispositivos rotativos como o Zoetrope e os “abajures” que giram movidos pelo ar quente das lâmpadas, projetando sombras animadas na parede.

Um dos artefatos mais curiosos e em uso até hoje são os Flipbooks. Os flipbooks são pequenos livros, ou cantos de livros que são usados para simular uma animação desenhando folha a folha. A animação ganha vida ao se folhar com rapidez essas páginas. Parece uma brincadeira de criança, mas em alguns países existe uma legião de fãs e compradores de histórias complexas neste tipo de animação.

A maior invenção no ramo da animação foi a câmera fotográfica. Quando inventadas, essas câmeras eram utilizadas somente para imagens estáticas. Pouco ou quase nada se sabia sobre como representar uma história a partir de imagens. No final de 1890, Meis introduziu os primeiros conceitos e truques para contar histórias e dar vida a coisas inanimadas, hoje conhecida como animação antropomórfica. Diversos outros fatos marcaram a evolução da animação em camadas e cores, mas ninguém marcou mais a história da animação do que Walter Elias Disney (1901-1966).

Walt Disney, como gostava de ser chamado, não só construiu novas técnicas e estúdios, como transformou a animação em uma forma de arte. Disney, foi premiado com 31 vezes com o Oscar, criou os storyboards, os animatics (pilotos de animação), planos de câmera, desenvolveu o uso de cores e som. Ele também promoveu a idéia que a mente do personagem era a força motriz da ação e que uma chave para movimento animado era a análise do movimento real.

A animação hoje possui um mercado bem delineado com um público de faixa etária bem diversificada. Segundo o canal Cartoon Network, um terço dos seus telespectadores estão na faixa de 18-34 anos. Parece curioso, mas o mercado de animações para adultos segue em pleno desenvolvimento.

6.2. APLICAÇÕES DA ANIMAÇÃO

A animação está presente em praticamente todas as aplicações da computação gráfica. Geralmente estamos acostumados a associar a animação com filmes ou propagandas, mas ela está em todos os segmentos. Na engenharia, por exemplo, é utilizada para diferentes tipos de verificações e visualizações. Na medicina, permite visualizar e entender os movimentos e órgãos do corpo humano. No ensino auxilia na criação de multimídias didáticos.

6.3. ANIMAÇÃO POR COMPUTADOR

Embora Mondelbrat descreva clips gerados por galaxias fractais em 1972, [Mondelbrat, 1988], é mais usual se dizer que a história da animação por computador iniciou-se em 1974 por René Jodoin, com o primeiro filme “Hung”. Esse filme foi concebido utilizando a técnica de interpolação de objetos. Filmes de circuito comercial que marcaram a história são Star Trek II (1977, que usou em poucos minutos), Tron (1982, que usou como parte integrante do filme), o Exterminador do Futuro II, Jurassic Park, O Segredo do Abismo, e mais recentemente a série Matrix e os desenhos animados 3D de Toy Story da Pixar aos novos produtos resultantes da união Pixar-Disney.

Os “cartoons” podem ser produzidos a partir de desenhos ou modelos físicos. No primeiro tipo, os artistas criam uma sucessão de quadros de desenhos, que são então combinados em um filme. No segundo método, é usado um modelo físico, por exemplo um boneco de massa, onde cada movimento é registrado como uma fotografia quadro a quadro.

A animação por computador pode ser produzida por uma grande diversidade de métodos. Uma das maiores dificuldades do animador é escolher qual desses métodos aplicará. Neste momento, a experiência e intimidade com a ferramenta podem ser decisivas. Enquanto em alguns casos os sistemas funcionam, em outros não. Na falta de experiência, testar será sua única saída.

Basicamente, podemos dividir a animação por computador em duas categorias: computer-assisted animation (animação assistida por computador) e computer generated animation (animação gerada por computador). No primeiro caso, o animador faz os quadros-chave e o computador se encarrega de gerar os quadros intermediários. O segundo apresenta um grupo maior de técnicas subdivididas nos seguintes grupos: técnicas de baixo nível (low level techniques – técnicas que ajudam o animador na especificação precisa do movimento), e técnicas de alto nível (high level techniques – técnicas usadas para descrever como se comporta o personagem durante a animação). Nesse caso, alto e baixo níveis estão associados ao nível de abstração. Por exemplo, se tiver um sistema que permita o controle do movimento de um personagem, como uma macrofunção que se resuma no comando “caminhe cuidadosamente observando cada detalhe”, você tem um sistema de alto nível. De

forma oposta, se você tiver de gerar todos os quadros correspondentes a cada movimento do andar do personagem, temos um sistema de baixo nível.

As de baixo nível usam técnicas como as do algoritmo de interpolação inbetweening, que ajudam o animador a detalhar os movimentos sempre que este possuir claramente as idéias do movimento pretendido.

Para alguns pode parecer óbvio e até mesmo obrigatório o controle de uma animação, mas isso não é verdade. O número de variantes em uma animação pode ser tão grande que seria impossível imaginar a seqüência exata do movimento. Um bom exemplo ocorre quando jogamos uma bola de borracha entre quatro paredes. Seria impossível imaginar a trajetória dessa bola.

As técnicas de alto nível são geralmente algoritmos ou modelos usados para gerar um movimento a partir de regras. Esses modelos/algoritmos são sofisticados programas computacionais capazes de simular forças físicas e realizar animações complexas.

Qualquer que seja a opção da técnica, sempre demandará um bom esforço por parte do animador. Porém, como via de regra, as técnicas de baixo nível sempre demandarão um maior esforço do que as alto nível. Na prática, os animadores usarão ambas as técnicas, ou seja, sempre que as técnicas de alto nível não produzirem o movimento esperado, será necessário recuar para as técnicas de baixo nível.

6.4. FORMAS DE ANIMAÇÃO

A classificação das formas de animação varia de acordo com os aspectos considerados. Além da que aqui usamos métodos de controle do movimento ou a interface com os atores podem ser usados na classificação das técnicas de animação [Magenat-Thalmann e Thalmann, 1991]

6.4.1. Animação por Quadro-Chave (Keyframe)

A animação por quadro-chave é um processo para criação de animações pelo qual os objetos são posicionados nos quadros críticos. Um quadro-chave (keyframe) é qualquer quadro de uma animação onde supostamente ocorre um evento específico importante. Os quadros localizados entre os quadros-chave são chamados de intermediários. Esse processo derivado da animação tradicional foi implementado em todos os sistemas de animação por computador. Os quadros intermediários são gerados automaticamente a partir dos quadros-chave (por interpolação) [Faley et al, 1993].

6.4.2. Animação por Script

Uma das mais poderosas ferramentas de animação é, sem dúvida, a animação por script. Um script é uma seqüência de instruções, em uma linguagem interpretável pelo sistema, para controle dos objetos e suas respectivas propriedades de animação, textura e comportamento. Um fato interessante e ao mesmo tempo polêmico, é

que muitas das recentes animações foram produzidas por pessoas da computação e não por artistas. Esse fato deve-se a intimidade que o pessoal da ciência da computação têm com as linguagens de programação.

O sistema de Scripting é a forma mais antiga de controle do movimento. Uma destas é conhecida como ASAS (*Actor Script Animation Language*), que tem uma sintaxe semelhante ao LISP. A ASAS introduziu o conceito de ator, isto é, um elemento complexo que tem suas próprias regras de animação. Os atores podem comunicar-se entre si através de mensagens e, então, sincronizar seus movimentos. Isso é semelhante ao comportamento de objetos em linguagens orientadas ao objeto.

Vamos então ilustrar um pequeno script para o Max:

```
b = GeoSphere ( ); s = sphere ( )
animate on
(
  at time 0   (move b [-100, 0, 0]; scale s [1, 1, 0.25])
  at time 35  move b [0, 100, 0]
  at time 100 (move b [200, 0, 0]; scale s [1, 1, 3])
)
```

Esse script anima uma geoesfera em torno de uma esfera alterando a escala da esfera.

6.4.3. Animação Procedural

As linguagens de script são linguagens de programação que não necessitam de compilação e geralmente são hospedeiras de determinados sistemas. Isso quer dizer que cada sistema pode e geralmente utiliza a sua própria linguagem.

A animação procedural utiliza o modelo de linguagem de programação por procedimentos, incluindo a orientação por objetos e não possui uma relação direta com um determinado sistema. As linguagens procedurais são aquelas em que os operadores são executados em uma certa ordem, para atender a uma solicitação ou atualização de dados. Um bom exemplo desse tipo de animação são os jogos [Wott, Policorpo, 2003].

Na verdade, a animação procedural pode ser um pouco mais complexa, mas certamente muito mais poderosa. Alguns cientistas encaram esse tipo de animação como uma forma científica de animação.

Existe uma infinidade de movimentos físicos, como o fogo ou as ondas, que podem ser representados por uma equação ou fórmula assim como os físicos e matemáticos o fariam. Até então, a representação do movimento tem sido quase que totalmente realizada pela mão de artistas. Se pensarmos no fogo, veremos que será impossível para um artista descrever a sequência de animação de milhões de partículas incandescentes. Na verdade, não só será difícil desenhar ou modelar como também o observar.

A animação procedural consiste basicamente em modelos matemáticos implementados em linguagens de programação para simulação de forças físicas (gravidade, por exemplo). As melhores aplicações desse tipo implementam soluções para a simulação da dinâmica de fluidos, movimento de roupas, cor dos, modos [Barzel, 1977] e de alguns animais.

6.4.4. Animação Representacional

Esta técnica permite um objeto mudar sua forma, se movimentar e andar durante a animação [Koe Badler, 1996]. Existem três subcategorias. A primeira é a animação de objetos articulados, isto é, objetos complexos compostos de segmentos rígidos conectados [Van de Panne, 1996]. O segundo é a animação de objetos suaves usada para deformar e animar a deformação de objetos, por exemplo, pele acima de um corpo, músculos faciais, tecidos e formas elásticas [Wilhelms, 1997]. O terceiro é o morphing que é a transformação de uma forma em outra bastante diferente [Zhong, 1996].

6.4.5. Animação Estocástica

Este tipo de animação utiliza o processo estocástico ou randômico para controlar grupos de objetos, como nos sistemas de partículas.

6.4.6. Animação Straight Ahead

Straight Ahead é uma forma de animação convencional, na qual o animador literalmente desenha quadro a quadro na forma seqüencial, partindo de um quadro inicial. Esse tipo de animação permite que novas idéias sejam aplicadas conforme a seqüência se desenvolve. Geralmente é usado para seqüências de espontaneidade e descontração.

6.4.7. Animação Pose-to-Pose

Na animação pose-to-pose, o animador define alguns quadros-chave, que definem a animação, e desenha depois os quadros intermediários. Esse tipo de técnica é melhor empregada onde a posição e o tempo são importantes.

6.4.8. Animação Comportamental

A animação comportamental ou por comportamento é aquela em que o animador descreve um conjunto de regras para a maneira como um ou mais objetos da cena reagirão com o ambiente. Um exemplo desse tipo é o sistema de partículas quando usado para multidões, bandos ou grupos de animais.

6.4.9. Animação Track Based

Track Based é uma generalização da animação por quadros-chave. Enquanto nos sistemas de quadros-chave todos os parâmetros devem ser especificados, nos sistemas track based, somente os parâmetros que controlam as interpolações devem ser especificados e em qualquer ponto do tempo.

6.5. CANAL ALPHA

Em computação gráfica, uma parte de cada pixel é reservada para informação referente à transparência. Uma imagem de um canal alpha é essencialmente uma silhueta em preto e branco dos objetos na cena (Figura 6.1), onde o preto representa as partes totalmente transparentes da imagem e o branco, as partes totalmente opacas. Os tons de cinza intermediários indicam transparência parcial. Sistemas de 32-bits contêm quatro canais, ou seja, são formados por quatro tipos de imagens. Sendo três canais de 8-bits (24-bits), ou seja, um canal para os tons vermelhos (Red), um para os tons verdes (Green) e 1 para os tons azuis (Blue), formando, assim, o sistema RGB. Cada um dos canais RGB pode ser entendido como imagens em tons de cinza, indicando o quanto de cada cor tem na imagem. Quanto mais alto o tom, mais daquela cor terá o pixel. O quarto canal também tem 8-bits e é chamado de canal alpha, indicando a transparência da imagem em relação a um fundo.



FIGURA 6.1. A imagem e o seu canal alpha em relação ao fundo.

O canal alpha é, na verdade, um tipo de máscara, especificando como as cores de cada pixel devem ser mescladas com outros pixels quando estes estiverem sobrepostos. É possível alterar o canal alpha de maneira a indicar as partes que você deseja que sejam transparentes ou não. Quando você estiver criando uma imagem de um objeto em um programa 3D e optar pelo formato de 32-bits, o programa cria automaticamente o canal alpha do objeto em 3D.

Se quiser mesclar a imagem de uma pessoa real com a de um cenário real, você tem de criar um canal alpha que defina como esta pessoa vai ser inserida na imagem. Para isto será necessário filmar a pessoa em frente a uma tela azul, verde ou mesmo um fundo preto. Com programas de edição de imagem ou de composição (como Photoshop, Paint Shop, Adobe After Effects etc.), é possível retirar a cor definida como chave (croma-key) do fundo e criar um canal alpha.

6.6. COMPOSIÇÃO

A composição é um conjunto de técnicas para unir os diferentes elementos renderizados em computador com elementos reais obtidos no set de filmagem. Tudo isso é possível através do canal alpha, como foi explicado. Na composição, é possível ajustar cores das imagens, adicionar a granulação típica de filme (*film grain*), trocar texturas dos objetos 3D, ajustar a iluminação virtual etc.

Esse processo pode ser dividido em duas etapas, que representam a captura da cena e sua posterior reconstrução.

6.6.1. Filmagem da Cena Real

Durante a filmagem da cena real, deve-se proceder como se o personagem 3D estivesse no local. Deve-se simular todas as interações do personagem virtual com o ambiente real. É preciso coletar o máximo de informações do ambiente de filmagem para que seja possível encaixar o personagem 3D no ambiente real. Assim 3 aspectos devem ser considerados:

- Coletar todas as informações do ambiente quanto a iluminação, posição da câmera, altura da câmera, tipo e tamanho da lente da câmera (distância focal). A Figura 6.2.B mostra uma bola branca que servirá para o animador estimar a intensidade, cor e direção da luz no ambiente de filmagem. Isso servirá para ajudar a criar e posicionar as luzes virtuais no ambiente 3D.
- Adicionar marcadores no ambiente de filmagem (Figura 6.2). Esses marcadores permitem a inserção correta dos personagens 3D na cena. A Figura 6.2.A mostra uma vara com faixas alternadas de cores diferentes apresentando medidas definidas de comprimento (por exemplo, de 15 em 15 cm). Assim, o animador pode usar esta cena como referência para saber quão alto o personagem deverá aparecer. No filme Parque dos Dinossauros foram usadas bolas de golfe e de ping-pong para marcar o chão durante a filmagem. As bolas eram enterradas parcialmente no chão, de forma a aparecer somente a sua metade.
- Interação dos elementos 3D virtuais com o elemento real. Nas cenas em que os personagens 3D interagem com objetos reais, devemos ter os elementos de

interações previamente filmados. Por exemplo, se um dinossauro vai a um lago beber água, devemos gravar a cena real de forma que a água comece a mexer assim que a boca dele chega a ela. Essas interações podem ser feitas no próprio ambiente de filmagem ou em estúdio com uso de croma key. Neste ponto, a criatividade será sua grande aliada.



FIGURA 6.2. Exemplo de marcadores para composição.

6.6.2. Reconstrução da Cena em 3D

De posse das filmagens reais será preciso então criar uma versão em 3D sintética deste cenário e daqueles objetos com os quais o personagem 3D interage diretamente. Os elementos da cena real serviram de base para criação dos correspondentes sintéticos.

- Com o conhecimento do tamanho e da distância da bola em relação à câmera (Figura 6.2), é possível criar bolas em 3D do mesmo tamanho das reais e posicioná-las de acordo com a cena real e assim definir os planos 3D.
- Para a inserção dos personagens na proporção correta usam-se os marcadores de vara com medida conhecida. Assim, se o objeto de referência tem dois metros e o personagem 3D tem quatro metros, o personagem precisa parecer estar duas vezes maior que a vara. Para isso, a vara é recriada em 3D e posicionada de acordo com os dados da cena filmada com a vara.
- Simular a iluminação da cena real: todas as fontes de luz do ambiente de filmagem devem receber marcadores. O sol ou qualquer outra fonte de iluminação principal (aquela que representa a sombra predominante) pode ser simulada com auxílio da bola da Figura 6.2.B. As lâmpadas virtuais devem ter propriedades similares às das lâmpadas reais simulando as cores, as intensidades, os focos etc.

6.7. CAPTURA DE MOVIMENTO

A análise de movimento por meios automáticos é de grande importância para diversas áreas da medicina, esporte, e reconhecimento de atividades humanas. No início do século XX, técnicas de captura de movimento foram usadas na animação tradicional em 2D (celulóide) pela Disney, para adequada representação dos desenhos animados.

Um sistema de captura de movimento é aquele em que se permite levar os movimentos de um ator real para um ator virtual. Os motivos para o uso desses processos são a busca do realismo, redução de risco em cenas perigosas, barateamento da produção e análise de movimento e produtos.

6.7.1. Rotoscopia

Derivada da técnica de animação tradicional, a rotoscopia consiste na utilização de um filme ou vídeo de um personagem real como base para a animação. Essa técnica foi usada tradicionalmente para elaborar desenhos animados baseados nos movimentos de animais ou pessoas. Disney, por exemplo, utilizou essa técnica para capturar os movimentos do personagem Pluto. Na época, um televisor era fixado ao tampo de uma mesa e o desenhista capturava o movimento quadro a quadro desenhando em papel manteiga. Hoje os principais sistemas de animação possuem diversos recursos para automação da captura do movimento por rotoscopia.

Apesar de existirem técnicas melhores e de estar sendo utilizada há várias décadas, a rotoscopia também foi utilizada para captura de movimento em grandes sucessos recentes do cinema, como o Exterminador do Futuro II (Figura 6.3).

Na década de 1980, o tipo de captura de movimento usado, era uma extensão da rotoscopia, em que os movimentos dos atores eram filmados ao mesmo tempo em mais de um ponto de vista. Marcadores eram colocados na pessoa e então manualmente codificados nos correspondentes pontos do espaço 3D. Esse processo ficou conhecido como fotogrametria.

6.7.2. Sistemas de Captura

No fim da década de 1980, a captura de movimento que hoje conhecemos começou a aparecer. Os sistemas de captura de movimento evoluíram muito nas últimas décadas com algoritmos capazes de reconhecer os marcadores no corpo da pessoa. Atualmente, existem basicamente quatro tipos: ótico, mecânico, magnético e acústico.

6.7.2.1. Ótico

É o tipo mais usado atualmente. Esses sistemas podem oferecer ao ator mais liberdade de movimentos uma vez que não precisam de cabos ligados ao corpo. Eles utilizam pequenas bolas ou discos refletivos que são fixadas ao corpo do ator. Podem

ser fixados nas articulações ou nos membros, mas não nas juntas. Precisam de pelo menos três câmeras de vídeo, cada uma equipada com uma fonte de luz, geralmente infravermelha, dirigida para iluminar a área de visão da câmera. As bolas ou discos refletem a luz na direção da câmera e um computador capta e sincroniza a imagem. Os dados das posições x , y , z de cada marcador são aplicados a um sistema de cinemática inversa para animar um esqueleto 3D.

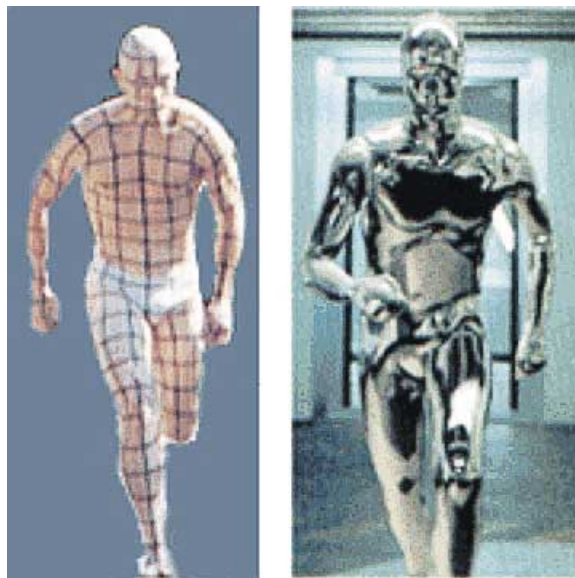


FIGURA 6.3. *Captura de movimento com a técnica de rotosopia.*

Uma desvantagem desse sistema é a facilidade com que o corpo do ator obstrui a reflexão das bolas. Esse problema pode ser contornado usando-se mais câmeras.

6.7.2.2. Mecânico

É um dos tipos mais antigos de captura de movimento e ainda hoje é o segundo mais usado. Possui alavancas que transformam movimentos rotacionais em dados de computador em tempo real. É baseado em um conjunto de armações que devem ser aderidas ao corpo do ator. As armações são conectadas entre si usando uma série de codificadores rotacionais e lineares (potenciômetros). Esses codificadores são conectados a uma interface que lê todos os codificadores ao mesmo tempo. Para captura facial, os tipos mais usados são os mecânicos e ópticos.

6.7.2.3. Magnético

O terceiro tipo mais em uso, captura o movimento por um transmissor magnético central e um conjunto de receptores que são colocados em várias partes do corpo.

Tem a vantagem de não sobrepor os marcadores, mas a desvantagem de utilizar vários cabos fixados ao corpo do ator, atrapalhando sua atuação e podendo ser afetados por qualquer objeto de metal na área de captura.

6.7.2.4. Acústico

Este tipo utiliza marcadores, que são transmissores de áudio, capazes de calcular a distância pelo tempo de resposta do sinal. Não é muito utilizado e apresenta as mesmas vantagens e desvantagens do magnético.

A captura de movimento não se limita a capturar apenas os movimentos humanos. Para algumas cenas de filmes como Parque dos Dinossauros, foram usados bonecos na forma de dinossauros.

6.7.3. Cartoon Motion Capture

Essa técnica de captura de movimento de desenhos animados 2D foi recentemente proposta pela Universidade de Stanford. Basicamente, a técnica se parece com a de morphing.

Utilizando as técnicas de rotoscopia e composição, o animador posiciona alguns pontos de controle no entorno do personagem 2D. A partir dos pontos de controle, o sistema cria uma animação com o canal alpha, que contém a silhueta do personagem. O animador, então, define uma série de quadros-chave para a animação, como os criados na animação pose-to-pose, reposicionando os pontos de controle com referência a posição inicial. Para o sistema, a variação da posição dos pontos será a informação transformada em dados de captura de movimento, que poderá ser usada futuramente para animar outros personagens. Essa técnica ainda não está disponível comercialmente, mas com certeza será introduzida como fonte de captura de movimentos.

6.8. ANIMAÇÃO DE PERSONAGENS 3D

Durante a fase de animação, os animadores criam a animação primária e depois a secundária. A animação primária consiste em animar as articulações e movimentos principais, como o caminhar. A animação secundária consiste em animar os pequenos movimentos que dão realismo, como movimento de músculos e gordura, respiração, piscar de olhos etc.

Para representar criaturas animadas, é conveniente armazenar as relações entre cada porção móvel diferente dos elementos geométricos que compõem o personagem. Tomando como exemplo o homem, as partes responsáveis pela deformação e movimento da malha do personagem 3D recebem o nome de seus correspondentes com a estrutura real do corpo humano. Os ossos (bones) são elementos individuais de estruturação definidos a partir do comprimento e vetor de direção, estabelecendo a ligação entre duas articulações. As articulações (joint) são os pontos pivôs de

rotação que estabelecem a conexão entre os ossos podendo apresentar até três eixos de rotação. O esqueleto é o conjunto formado por todos os ossos e articulações componentes da estrutura de deformação. Além da parte óssea, temos os músculos flexores que além de limitar o movimento do esqueleto, deformam a malha.

O aspecto mais importante a ser levado em consideração para a síntese de movimento em computador é o domínio do software a ser utilizado.

Independente do programa de síntese de movimento empregado, um software de animação tridimensional gerencia as relações entre as diversas camadas que compõem um personagem, sendo o mediador da interação entre o animador e o personagem virtual criado, possibilitando o controle e o refino dos movimentos, dos gestos e das ações realizadas pelos personagens e objetos animados.

Para o controle de movimento em personagens tridimensionais criados em computador, podemos identificar três camadas primárias para a sua construção:

- Uma primeira camada composta pela estrutura formada pelos ossos, responsável por receber a entrada de dados sobre rotação e translação da estrutura do personagem;
- Uma segunda camada que define a distorção do modelo geométrico, que apresenta os controles de deformações quando o personagem, por exemplo, senta ou fala.
- Uma terceira camada que descreve a superfície e a aparência da geometria que compõem o personagem ou objeto (skin ou malha). Ao contrário das demais, esta será vista como a forma final do personagem.

Na prática, o processo de animação 3D ocorre na seguinte sequência. Baseado na história, é feito o desenho em 2D do personagem demonstrando suas expressões. Em seguida, ocorrem simultaneamente a modelagem 3D do personagem e a construção do esqueleto. A malha do personagem é então fixada no esqueleto e a animação é desenvolvida.

6.8.1. Cinemática

Diversas aplicações precisam representar o complexo movimento humano e animal. Métodos tradicionais de animação baseados em quadros-chave não trazem a realidade desses movimentos, o que proporcionou a criação de inúmeras técnicas para a resolução desse problema. Dentre as mais conhecidas estão a Scripting Animation, Procedural Animation, Forward Kinematics (cinemática direta) e Inverse Kinematics (cinemática inversa). A captura de movimento é atualmente a melhor fonte de dados de movimentos para a cinemática.

As funções cinemáticas são usadas para diferentes fins. Na robótica, elas são usadas para calcular a trajetória de um braço mecânico baseado na posição de um objeto; nos jogos, são usadas para dar realidade aos movimentos e para guiar, por exemplo, o braço

de um jogador a um objeto; e, por fim, nos trajes de captura de movimento, são utilizadas para minimizar o uso de dispositivos de posição e orientação fixados à roupa.

Um corpo humano pode ser modelado com um conjunto de 19 articulações conectadas por membros (Figura 6.4) e dispostas numa estrutura hierárquica (Figura 6.5). Se o grau de liberdade dessas articulações for conhecido ou especificado, a posição final do membro pode ser facilmente computada.

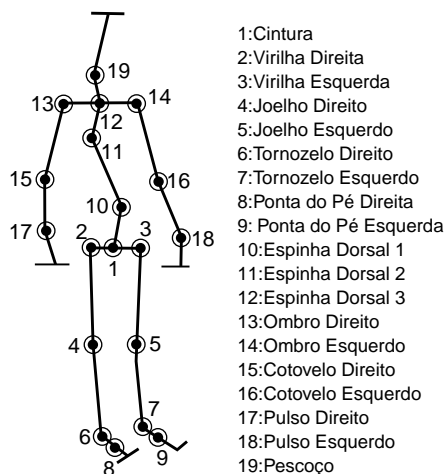


FIGURA 6.4. Conjunto das articulações humanas.

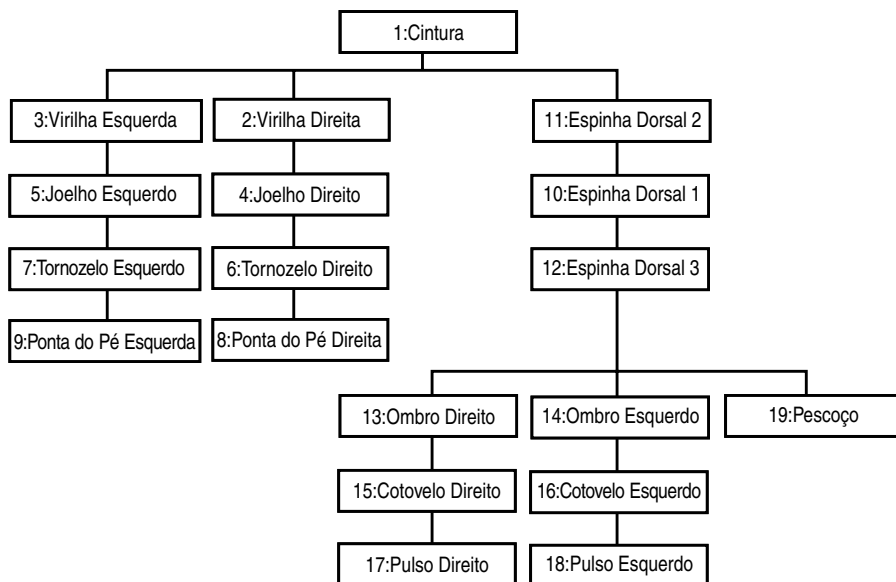


FIGURA 6.5. Estrutura hierárquica do conjunto das articulações humanas.

6.8.1.1. Cinemática Direta

A estrutura da Cinemática Direta (Forward Kinematics-FK) é a mais básica de todas as estruturas. Quando é criada uma árvore de ossos ou objetos vinculados, define-se uma hierarquia FK. Os ossos FK são “filhos” do anterior e assumem as propriedades de posicionamento, translação e rotação do osso anterior.

FK usa a metodologia de hierarquia top-down, onde a posição das articulações pais interfere nas articulações filhas. A função que descreve a forward kinematics se torna cada vez mais complexa com o aumento de articulações, podendo ser impossível revertê-la.

FK é uma boa introdução para o entendimento de uma estrutura básica de árvore e como ela funciona, mas se torna muito difícil para o animador executar o trabalho, porque todos os ossos “filhos” ficam dependentes de qualquer animação que se for fazer no osso “pai”. Uma vantagem no uso de FK é que você poderá fazer uso de arcos para o controle de quebras nos movimentos das juntas.

Quando se trabalha com FK geralmente são usadas malhas segmentadas, como uma marionete, por exemplo. Uma malha segmentada é um grupo de peças conectadas nas juntas ligadas formando uma hierarquia. Os esqueletos FK são geralmente usados em cenas com pouco movimento, como uma narrativa, por exemplo, ou uma cena em que o personagem está falando e não se movimentando. Isso na verdade não é uma regra. A escolha entre as estruturas FK ou IK depende, é claro, do animador e de como ele imagina, na sua criação, os movimentos dos personagens.

6.8.1.2. Cinemática Inversa

Já a Cinemática Inversa (Inverse Kinematics) utiliza a metodologia de hierarquia inversa, na qual as posições das articulações filhas interferem nas articulações pais. A função que a descreve calculará a posição e orientação das articulações e membros pais a partir da posição final da articulação filha da hierarquia, ou seja, com a estrutura IK, um osso do pé, na sua posição final definirá como os outros ossos do joelho e quadril, deverão girar e se posicionar, tornando a animação de um andar bem mais convincente.

A estrutura IK tem uma difícil configuração, mas pode ser compensadora se criada e observada corretamente. Em um ciclo de um andar, por exemplo, o animador poderá poupar um bom tempo do trabalho, pois essa estrutura facilitará esse tipo de movimento.

Comparando-se as duas técnicas, podemos perceber que a IK possibilita a utilização de movimentos mais complexos com mais facilidade do que a FK. Porém, em contrapartida, a IK faz com que o movimento seja definido por uma função, o que pode resultar em movimentos indesejados. Concluimos, então, que, em muitos casos, a solução ideal poderá ser a utilização de ambas as técnicas para o mesmo personagem.

6.8.2. Ossos

A estrutura de ossos criada para ser a base de deformação do personagem articulado é utilizada para o estabelecimento de vínculo geométrico entre os diferentes elementos possíveis de serem animados. Essa estrutura, composta por segmentos conectados por articulações de forma hierárquica, é a base para todas as deformações que serão aplicadas para a alteração do aspecto visual das superfícies.

Fisiologicamente, não podemos fixar um eixo ou centro de rotação para uma articulação, pois são executados simultaneamente movimentos de rotação e deslizamento nas áreas de contato dos ossos. Para a animação voltada para o entretenimento, podemos ignorar as pequenas translações ocorridas e criar um modelo de articulação ideal, que faça uso apenas de coordenadas angulares para registrar o movimento dos parâmetros de rotação.

Os ossos podem ser também elásticos, ou seja, variar de tamanho durante a animação, permitindo simular movimentos de criaturas que, na verdade, não possuem um esqueleto (como minhocas) ou criaturas típicas de desenho animado.

6.8.3. Articulações

São muito importantes para possibilitar posturas e gestos. Lee e Kunii apresentam um interessante estudo sobre as mãos em [Lee, Kunii, 1995]. As articulações mais utilizadas na criação de personagens animados são as juntas de revolução e esférica, descritas a seguir:

6.8.3.1. Junta de Revolução

É o tipo de junção mais utilizada e o termo “revolução” se deve ao fato de que o extremo de um osso é rotacionado no extremo do outro osso, componente da cadeia cinemática. Por obter rotações apenas em um simples eixo, a junta de revolução tem apenas um grau de liberdade.

Existem três formas de juntas de revolução:

R1: Dobradiça (hinge), é a forma mais comum e simples de articulação, onde um osso (A) gira em um eixo perpendicular a outro osso (B), como as articulações encontradas no cotovelo e no joelho. O ponto localizado no extremo final do osso A realiza um movimento circular com centro de rotação no extremo final do osso B.

R2: O eixo de rotação é paralelo aos dois ossos (A e B). O ponto extremo de B não pode mudar sua posição no espaço, mas pode girar longitudinalmente em relação ao osso A. Podemos criar um modelo simplificado para o movimento do pulso (ao girar a palma da mão para cima e para baixo) com esse tipo de junta.

R3: É uma variação de R2; o eixo de rotação permanece como na forma anterior, mas o osso B muda de posição e é colocado perpendicularmente em relação ao osso A. Dessa forma, o extremo final de B rotaciona ao redor do ponto final do osso A (como ao realizar todos os movimentos possíveis do pulso).

6.8.3.2. Junta Esférica

Este tipo de articulação implementa o conceito de junção *ball-and-socket*, onde uma esfera está livre para executar qualquer movimento de rotação enquanto estiver segura por um encaixe.

6.8.3.3. Grau de Liberdade

As possibilidades de movimento, do mecanismo de articulação no espaço tridimensional, correspondem ao número de parâmetros cinemáticos independentes permitidos, e são chamados de grau de liberdade (*DOF – Degree of Freedom*). Quanto maior o número de DOFs, maior a liberdade de movimentação da estrutura e configurações possíveis para o estabelecimento de poses.

O grau de liberdade está diretamente relacionado aos ângulos de rotação em torno dos eixos x , y e z (Capítulo 2). Podemos limitar o grau de liberdade para um ou dois eixos. O pulso humano é um bom exemplo de grau de liberdade em dois eixos: se observarmos os movimentos de nosso pulso, podemos rotacioná-lo de cima para baixo (e vice-versa) e de um lado para o outro, ou seja, em dois eixos. Nosso joelho é um bom exemplo de grau de liberdade em um eixo, ou seja, de cima para baixo.

Podemos também limitar o movimento pelo ângulo máximo possível entre as articulações. Se movimentarmos nosso joelho para trás veremos que ele percorre um ângulo máximo de 180° .

Observando qualquer criatura viva, vemos que quando a articulação se rotaciona próxima ao limite máximo há uma redução da velocidade, isso tem o nome de amortecimento. Esse fenômeno é ao mesmo tempo a representação do limite da musculatura e da capacidade de rotação, em um determinado eixo, de uma junta.

Neste caso, podemos controlar os atributos de amortecimento de duas formas: alcance e resistência (Figura 6.6).

O alcance refere-se aos limites máximos e mínimos das articulações e pode ser controlado pelo ângulo máximo de rotação.

A resistência está relacionada à capacidade elástica dos músculos e pode ser controlada por um atributo de resistência que determina o alcance.

6.8.4. Esqueleto

O conjunto formado pela ligação entre os ossos e articulações é denominado esqueleto. O esqueleto é organizado também segundo um modelo hierárquico, onde as

transformações ocorridas no topo da hierarquia são herdadas pelos segmentos inferiores. O nível mais alto de uma estrutura hierárquica é chamado origem ou raiz (root); cada origem de uma composição pode ter um ou mais filhos (child). Nesse arranjo, o “pai” (parent) de um osso é o segmento imediatamente acima na cadeia cinemática, que controlará o movimento de todos os ossos localizados abaixo. Dessa forma, apenas o acúmulo das transformações geométricas entre os sistemas de coordenadas das articulações, ou seja, ao movimentar o osso do braço, toda a estrutura inferior – antebraço, punho, mão e dedos – acompanhará esse movimento.

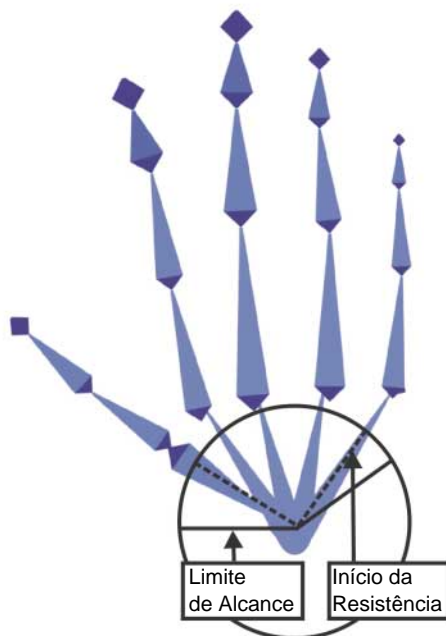


FIGURA 6.6. *Atributos de amortecimento da articulação da mão.*

Simulando uma estrutura biomecânica real, o esqueleto tem a função de receber as informações de translação e rotação simples, para que sejam posteriormente aplicadas na geometria original. Outros parâmetros também armazenados pelo esqueleto são:

- A estrutura hierárquica de toda a ramificação dos ossos: descrição do estado de cada osso em relação aos segmentos estabelecidos pela vinculação hierárquica através da definição dos manipuladores.
- Os parâmetros das articulações: matrizes de transformação com os parâmetros cinemáticos relativos ao comprimento, posição, orientação, distância relativa, ângulo formado entre os segmentos adjacentes e rotação longitudinal (*twist*).

- Os limites de rotação e atributos físicos: restrições arbitrárias concedidas às articulações para restringir o movimento nos eixos de rotação. Podem ser atribuídos valores de resistência oferecida pela articulação ao movimento e aos limites para o ângulo de rotação possível para uma junta.

Um esqueleto para animação não precisa ser a cópia exata do esqueleto real do personagem. Um esqueleto pode ser composto por uma ferramenta especial de animação [Monheit e Badler, 1991].

Geralmente, iniciamos a construção do esqueleto a partir das pernas dando as proporções dos ossos à malha do personagem. Quando pensamos em um esqueleto real, pensamos primeiro nos ossos e depois nas articulações, que permitirão o movimento. Na animação por computador, precisamos primeiro focar nas articulações e depois no relacionamento hierárquico indicado pelos ossos (Figura 6.7).

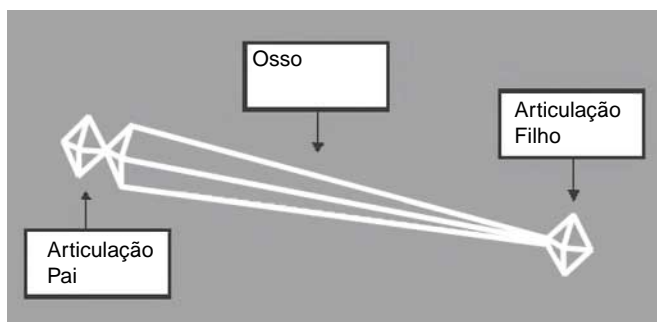


FIGURA 6.7. *Relacionamento hierárquico indicado pela forma triangular do osso.*

Observe que o osso tem um formato cônico com lados de polígonos triangulares cujas extremidades são usadas para definir as hierarquias. Logo, a junta na parte grossa estará hierarquicamente superior a junta na parte fina, ou seja, será o pai da mais fina. Quando o pai se move, o filho também se move, o inverso não é verdadeiro. Qualquer série simples de articulações conectadas por ossos é chamada de cadeia de articulação. A articulação mais alta na hierarquia da cadeia de articulação é chamada de articulação pai.

Um membro consiste em uma ou mais cadeias de articulação que ramificam-se em uma árvore de estrutura (Figura 6.8).

A articulação mais alta na hierarquia desse esqueleto é chamada de articulação raiz; quando a articulação raiz se move ou gira, tudo deve mover ou girar com ela.

Existem dois caminhos básicos para posicionar uma cadeia de articulação: as (já comentadas) cinemática direta e cinemática inversa.

Com a cinemática direta, temos de especificar as rotações de cada articulação individualmente, a partir da articulação pai e em todas as articulações hierarquicamente abaixo. Essa abordagem é excelente para criar movimentos detalhados de arco.



FIGURA 6.8. *Árvore de estrutura de uma cadeia de articulação.*

Com a cinemática inversa, tudo o que você terá de fazer é dizer à articulação mais baixa, na hierarquia da cadeia de articulação, onde será sua nova posição, todas as articulações acima automaticamente girarão. A cinemática inversa oferece um caminho muito intuitivo para posicionar uma cadeia de articulação porque habilita uma meta dirigida de posicionamento.

6.8.4.1. Controladores IK

Para posicionar uma cadeia de articulação com a cinemática inversa, será necessário adicionar uma ferramenta especial para o esqueleto. Essa ferramenta não possui um nome específico, chamaremos de Controladores IK.

Um controlador começa na articulação pai e pode terminar em qualquer articulação abaixo dela. Por exemplo, para cada braço você pode criar um controlador que controle a cadeia de articulação que começa na articulação do ombro e finaliza na articulação do pulso (Figura 6.9).

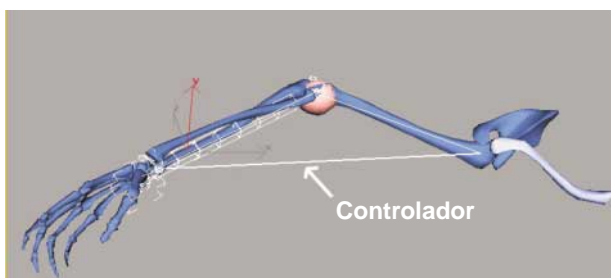


FIGURA 6.9. *Controlador IK iniciando na articulação do ombro e finalizando na articulação do pulso.*

Os controladores não só facilitam o processo de posicionamento como também desenvolvem um importante papel na solução dos movimentos entre os quadros-chave.

A arquitetura de um controlador é composta por uma série de elementos e definições como mostra a Figura 6.10.

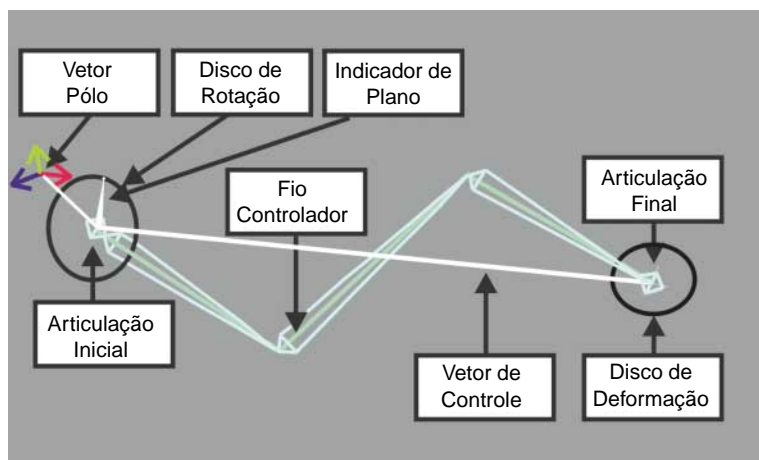


FIGURA 6.10. Elementos da arquitetura de um controlador IK.

Articulação Inicial (Start Joint): é onde começa o controlador IK. Deve ser a primeira articulação na hierarquia da cadeia de articulação, por exemplo, a articulação raiz.

Articulação Final (End Joint): é onde termina o controlador IK. Deve ser a última articulação na hierarquia da cadeia de articulações.

Fio Controlador (Handle Wire): é uma linha que percorre todos os ossos indo da articulação inicial até a final.

End Effector: é o fim do controlador IK e está localizado na articulação final.

Objetivo (Goal): indica onde o controlador deve estar. Você pode movê-lo para qualquer ponto da cena. O controlador usará essa posição para determinar o movimento.

Disco de Rotação (Rotation Disc): está localizado na articulação inicial e indica como a cadeia de articulações pode rotacionar.

Disco de Deformação (Twist Disc): está localizado na articulação final e é usado como ferramenta para trançar a cadeia de articulações.

Indicador de Plano (Plane Indicator): indica a orientação do plano da cadeia de articulações, que é o grau de deformação na cadeia de articulação em relação ao plano de referência. Pode ser imaginado como a reflexão do plano da cadeia de articulação no disco de rotação.

Plano de Referência (Reference Plane): quando há a rotação e a deformação da cadeia de articulações, o plano deve rotacionar em relação a algum outro plano, para que o grau de deformação possa ser medido. Para isso, o plano da cadeia de articulação rotaciona em relação ao plano de referência. O plano de referência é definido pelo Vetor de Controle e pelo Vetor Pólo:

- **Vetor de Controle (Handle Vector):** é a linha que vai da articulação inicial até o End Effector. É usado para indicar em qual articulação o controlador IK deve iniciar e acabar.
- **Vetor Pólo (Pole Vector):** o vetor pólo inicia na articulação inicial e pode terminar em qualquer articulação. O propósito do vetor pólo é definir o plano de referência.

6.8.4.2. Ciclo de Animação

O ciclo de animação é uma poderosa técnica derivada da animação tradicional. Essa técnica é baseada na observação do movimento contínuo no processo de caminhar, correr ou qualquer movimento que se repita.

Desenvolver um ciclo de animação não-linear envolve duas poses-chave onde a primeira e a última poses do ciclo são iguais. No caso da caminhada, por exemplo, as poses-chave intermediárias do ciclo são quando um dos dois pés está plantado no chão (para corrida quando os dois estão no ar) e quando um pé está acima do outro.

6.8.4.3. Sistemas de Animação IK

Uma das formas mais fáceis de controlar uma animação de personagem é usando um sistema de animação IK. Esses sistemas permitem a inserção automática de modelos de controladores IK pré-fabricados com formas variáveis, por exemplo, bípede, quadrúpede, com ou sem rabo etc. Esses modelos podem ser facilmente deformados para adaptarem-se à escala da malha do personagem (Figura 6.11).

As diferenças entre o sistema de animação IK e as estruturas ósseas são descritas na tabela que segue:

| Estrutura Óssea | Sistemas de Animação IK |
|--|---|
| você cria a sua estrutura de ossos | o sistema cria uma estrutura para você |
| controle de transformações ilimitadas | o sistema limita o movimento e as rotações |
| acesso a todas as funções do sistema de modelagem | interface limitada pelo sistema de modelagem |
| difícil de aplicar arquivos de movimentos (motion capture) | fácil de aplicar arquivos de movimentos |
| permite animar qualquer tipo de objeto | permite somente animar objetos com aspecto humano ou animal |
| animação manual | animação automática com funções avançadas para multidões. |

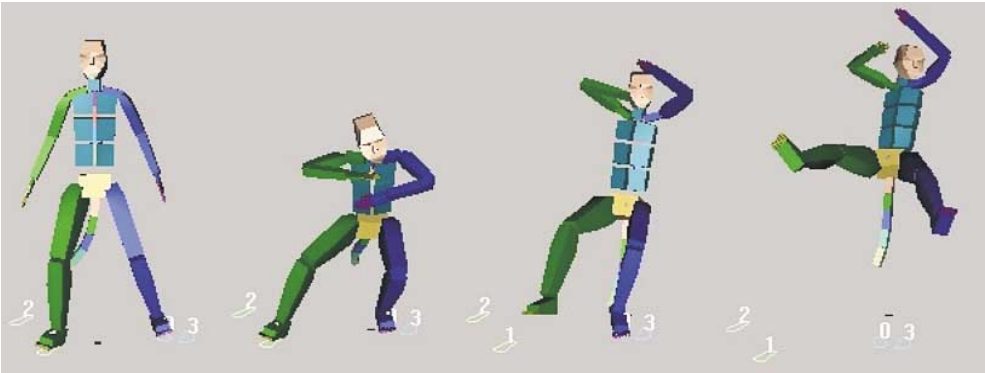


FIGURA 6.11. Um salto para trás usando o sistema de animação IK.

Como podemos ver, a estrutura óssea é mais versátil do que os sistemas de animação IK. Em compensação, o sistema IK disponibiliza em seus arquivos uma variedade enorme de ciclos de animação captados em sofisticados equipamentos de motion capture com controles dos limites de movimento predefinidos.

6.8.5. Músculo Flexor

Para animar as deformações de pele, será necessário usar ferramentas de deformação especiais chamadas flexores. Os flexores são ferramentas de alto nível para usar com peles (malha) e esqueletos. O efeito provocado na malha dependerá da posição do esqueleto.

Os flexores podem ser divididos em: reticulados (lattice), esculturais (sculpt) ou de grupo (cluster).

Um flexor reticulado influencia a malha em volta das articulações ou os ossos das articulações. Esse flexor enrugua e alisa a malha e dá definição muscular em volta do osso.

Um flexor escultural influencia a malha em volta das articulações ou os ossos das articulações criando saliências e inclinações. A Figura 6.12 demonstra a formação da junta do antebraço e do bíceps utilizando o flexor. As deformações anatômicas provocadas pelos flexores podem ainda simular o efeito de juntas do joelho ou cotovelo. Um flexor de grupo controla os pontos da malha em volta das articulações com variações percentuais de influência.

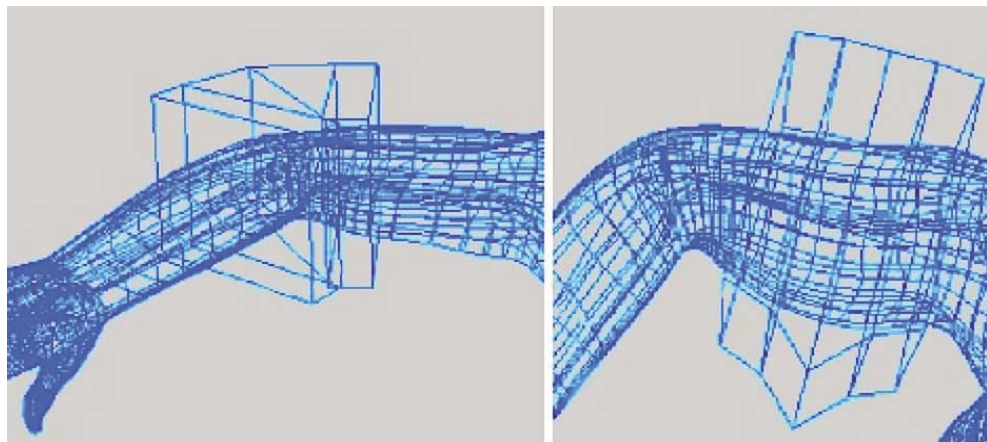


FIGURA 6.12. Formação da junta do antebraço e do bíceps utilizando o flexor.

O tempo que você gasta para construir um esqueleto e criar os flexores é bem gasto. Uma boa construção será recompensada na hora de animar. No exemplo da Figura 6.13, a imagem à direita mostra o número de controladores para animar o dinossauro da imagem à esquerda.

A estrutura de animação desse dinossauro usou 138 ossos, 34 músculos, 84 dummies ou helpers, 10 controladores IK e 18 pontos de referência

6.8.6. Cabelos e Pêlos

A animação de pêlos é importante na geração realista de personagens humanos ou animais [Hzinen e Walter, 2001]. O mesmo processo pode ser aplicado na animação de gramas ou qualquer outro tipo de objeto parecido. Contudo, existem diversos problemas relacionados à quantidade de pêlos que não raramente ultrapassam a casa dos duzentos mil. Para o filme *Um Lobisomem Americano em Paris*, foram renderizados cerca de 3,5 milhões de pêlos no corpo do Lobisomem.

As técnicas de renderização de cabelos/pêlos são bem recentes. Em 1988, por exemplo, Miller propôs a geração de cabelos pela modelagem de triângulos. No ano

seguinte, Kajiya e Kay propuseram a utilização de mapeamento tridimensional. Em 1992, Watanabe e Suenaga apresentam um modelo considerando cada fio formado por sequência de prismos triangulares [Watanabe e Suenago, 1992].

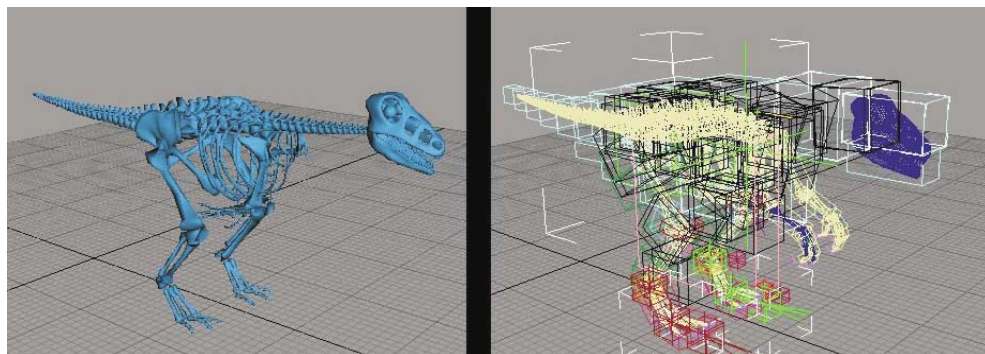


FIGURA 6.13. A complexa estrutura de animação de um dinossauro.

Atualmente, os sistemas de animação de pêlos/cabelos são geralmente baseados na interpolação entre cabelos guias. Os cabelos guias são splines, ou seja, o animador pode posicionar dois pontos para definir as curvas que serão o cabelo guia. Os cabelos relacionados com esses tomarão a forma dessa curva. Devido ao formato variado da pele, um bom modelo deverá ter cerca de 200 cabelos guias no corpo.

Outras questões relevantes são os parâmetros relacionados ao cabelo como espessura, comprimento, cor da raiz a ponta, forma (liso, encaracolado), especularidade, influência da gravidade, inércia etc. Para um maior realismo, cada cabelo deve ser capaz de receber e projetar sombras e ainda interagir com outros cabelos ao se movimentar com o vento ou como personagem, o que dá volume e determina a direção na colisão entre fios.

6.8.7. Animação Facial

Animação facial refere-se à criação de objetos tridimensionais assemelhados a faces, capazes de simular expressões, fala e movimentos característicos da face. Esse campo tem sido objeto de pesquisa constante nos últimos tempos, e pode ser encontrado em interfaces sofisticadas de homem-computador, jogos interativos, multimídias, realidade virtual etc. As tecnologias de suporte incluem síntese da fala e inteligência artificial [Hong et al. 2002].

A expressão facial humana foi assunto de investigação científica por mais de 100 anos. Os esforços em relação à expressão facial por computador começaram há aproximadamente 25 anos. É interessante que a maioria das técnicas atualmente empregadas envolvam os princípios desenvolvidos na comunidade científica, em alguns casos, várias décadas atrás.

O mais antigo trabalho com computadores baseado em expressões faciais data do início da década de 1970. Em 1971, Chernoff propôs o uso de faces bidimensionais como forma de representar dados dimensionais. A primeira animação tridimensional foi criada por Parke em 1972. Em 1973, Gilleson desenvolveu um sistema interativo para montar e editar imagens faciais a partir do desenho de linha. E em 1974, Parke desenvolveu um modelo facial tridimensional parametrizado.

Na década de 1980, foram desenvolvidos os primeiros modelos de controle de músculo da face e técnicas de caricatura facial. Em 1985, a película animada do curta “Tony de Peltrie” foi um marco para a animação facial. Pela primeira vez, foi mostrado que a expressão facial e a animação da fala eram fundamentais para se contar uma história. Nesse período também foram desenvolvidos modelos de novos músculos por Waters, e um modelo abstrato da ação do músculo por Magnenat-Thalmann. Na década de 1990, a animação facial atingiu o realismo com o filme “Toy Story” desenvolvido pelo estúdio Pixar.

Quando falamos em animação facial não podemos deixar de mencionar o filme Shrek; no qual os personagens são capazes de expressar os diálogos e as emoções através de um complexo sistema de animação facial desenvolvido na PDI/DreamWorks. Utilizando ferramentas especiais chamadas *shapers* (basicamente um processo de camadas que deforma a superfície a partir do seu interior), os animadores puderam realizar sofisticados movimentos aplicando camadas de ossos, músculos, gordura, pele, cabelos e roupas que interagem entre si. Essencialmente, o crânio do personagem é formado no computador e coberto com uma camada de músculos que simulam os verdadeiros da face. Depois, tudo é coberto com a pele e programado para responder às manipulações dos músculos, como aconteceria num rosto humano, completado por rugas, marcas de expressão e outras imperfeições. Centenas de controles estão ligados no rosto como se fossem nervos humanos permitindo aos animadores simularem os fonemas da fala para uma perfeita sincronia labial. No filme, há também avanços na criação de ambientes complexos e orgânicos; de roupas que se movimentam, dobram e reagem à luz como tecidos de verdade; do fogo e fluidos com viscosidades diferentes obtidas através do uso do premiado Fluid Animation System (FLU), da PDI/DreamWorks. Existem diversos métodos de animação facial, dentre os quais podemos citar:

6.8.7.1. Sincronização Labial

O processo de animação facial em 3D segue a linha do modelo de animação 2D, onde o animador deve desenhar todas as posições labiais da cena e modificá-las tentando uma sincronização labial (Lip Sink) da fala.

6.8.7.2. Seqüência de Texturas

Neste método, a animação é conseguida através da troca do mapeamento para cada posição da boca. Para isso, é necessário um arquivo de posição ou textura para cada mudança de fonema [Matheus et al, 2002].

6.8.7.3. Morphing

Neste método, são modeladas faces para cada fonema e expressão. O software, então, faz a interpolação das faces criando a sincronização. A técnica também é muito útil para caracterizar o envelhecimento do personagem [Rowland e Perrett, 1995].

6.8.7.4. Esqueleto

Como descrevemos anteriormente, a idéia de articulações simulando esqueleto pode ser usado para animar qualquer coisa, inclusive a face.

6.8.7.5. Free Form Deformation

Para criar as diversas posições da boca, utilizamos uma ferramenta de deformação sobre a malha do rosto, que funciona basicamente da mesma maneira que o esqueleto.

6.8.7.6. Weighted Morphing

Da mesma maneira que o Morphing, são modeladas as posições da face para cada grupo de músculos faciais, fonemas ou expressões. Essas posições chamados VISEMES são aplicadas como deformação à face inicial (neutra) e podem ser mixadas em diferentes proporções propiciando uma diversidade de poses. Normalmente, esse é o método predileto dos animadores, devido à sua rapidez e facilidade de criação de novas posições.

Independente da escolha do método, os fonemas da língua portuguesa podem ser formados por no mínimo doze posições, onde teremos cinco para as vogais (A, E, I, O e U) e mais sete para o grupo de sons (B-P-M, C-K-G-Q, D-N-T-L, F-V, R, S-Z, J-X-CH). Para isso, temos de quebrar o áudio das frases identificando os picos de fonemas e seu tempo em quadros.

Além da definição da quantidade de posições dos fonemas, devemos definir faces para os sentimentos que o personagem terá durante a animação (felicidade, tristeza, surpresa, medo, ódio, desgosto etc.).

Em alguns casos, dependendo do realismo desejado, você poderá optar pela retirada de alguns fonemas deixando somente aqueles que apresentam uma variação maior entre os picos.

É importante observar alguns aspectos da fala humana, por exemplo, a coarticulação, ou seja sempre que iniciamos uma fala, nossa boca antecipa a forma de emis-

são do próximo fonema. Procure estudar e observar os sotaques e expressões regionais. Animadores freqüentemente utilizam espelhos e demonstrações com o próprio corpo para captar a essência do movimento [Zahng e Cohen, 2002].

6.9. ANIMAÇÃO DE SUPERFÍCIES DEFORMÁVEIS

Nos últimos anos, temos percebido um crescente interesse no desenvolvimento de modelos de animação baseados na física. Engrenagens capazes de simular colisões, gravidade, entre outros fenômenos, passaram a ser distribuídas em conjunto com os browsers mais usados de navegação na Internet. Inicialmente, a animação era realizada apenas com objetos rígidos, mais tarde surgiram os objetos articulados e, recentemente, aqueles ditos flexíveis ou deformáveis.

A animação de personagens ou objetos deformáveis exige que se possua um modelo geométrico que permita a mudança de forma ao longo do tempo. Em computação gráfica, isso pode ser definido através de modelos físicos de curvas, superfícies ou mesmo sólidos deformáveis.

O objetivo principal na simulação de modelos deformáveis é produzir movimentos fisicamente realistas. Exemplos incluem a simulação da musculatura do corpo humano, a fim de representar realisticamente a pele de um personagem falando; a simulação do fluxo de líquidos viscosos, como, por exemplo, derramamento de óleos ou ainda a simulação de tecidos, parte essencial de qualquer personagem e produto para a indústria de vestuário [Ng e Grimsdale, 1996].

O modelo geométrico para simular uma superfície deformável consiste em uma malha representada por uma matriz de pontos M . Cada elemento M_{ij} dessa matriz contém uma posição xyz correspondente a um ponto da superfície, em um espaço tridimensional. Considera-se ainda r como a distância entre os elementos da matriz M já definida.

Para simular fisicamente uma malha flexível, podemos usar o conceito de mola elásticas (ou que obedecem a lei de Hooke). Esse modelo baseia-se fundamentalmente na aplicação de forças sobre os diversos pontos de massa da malha, gerando novas posições para o equilíbrio dos mesmos.

Considerando a aplicação de apenas três forças distintas: da gravidade, de elasticidade e de curvatura (ou torção), obtemos uma força resultante em cada ponto da malha que pode ser calculada da seguinte forma:

$$F_{\text{resultante}} = F_{\text{gravidade}} + F_{\text{elasticidade}} + F_{\text{curvatura}} + F_{\text{restrição}}$$

Isso, no entanto, não quer dizer que não possam aplicar outras forças ao modelo. Por exemplo, para incluir a simulação da força do vento, ou qualquer restrição basta acrescentar este elemento na soma acima.

A força da gravidade pode ser calculada para cada ponto da malha, da seguinte forma:

$$F_{\text{gravidade}} = g \cdot m$$

onde g representa a aceleração da gravidade e m a sua massa do ponto.

A força de elasticidade em um ponto P_{ij} , da matriz M_{ij} , supõe a sua conexão aos seus oito vizinhos, através de molas elásticas (Figura 6.14). Sabendo-se que a força exercida por uma mola elástica que obedeça a lei de Hooke sobre um ponto de massa m é:

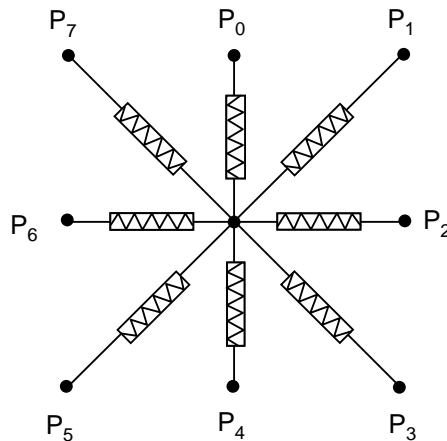


FIGURA 6.14. A força de elasticidade em um ponto P_{ij} , representado pela conexão aos seus oito vizinhos, através de molas elásticas.

$$F_{\text{Mola}} = -k_m \cdot (P - P_r)$$

onde k_m é a constante elástica da mola (ou melhor a constante elástica modificada para considerar deslocamentos e não deformação), P representa as coordenadas do ponto em questão e P_r as coordenadas do ponto em repouso. Podemos então definir a força de elasticidade como sendo o somatório das forças exercidas pelas molas, no ponto P .

$$F_{\text{elasticidade}} = \sum F_{\text{Mola}}$$

A força de curvatura ou torção indica o grau com que a superfície se curva e torce. Essa força é calculada também em função dos oito vizinhos de um ponto. A simulação desse efeito pode ser concebida colocando molas angulares entre os vizinhos do ponto P , da seguinte forma: uma mola no ângulo formado entre os vértices P_0 , P e P_4 ; entre P_1 , P e P_5 ; entre P_2 , P e P_6 e entre P_3 , P e P_7 . A colocação dessas molas é feita de forma que estejam na posição de repouso quando o ângulo formado entre os três vértices for igual a 180° . Além disso, considera-se que as molas em questão

possuam um coeficiente de elasticidade parametrizável, denominado k_c . A alteração que a força exercida por essas molas possa vir a causar na posição do ponto P_{ij} é indicada por uma função $f(i, j)$.

Devemos ainda considerar as restrições físicas do modelo ou a força de restrição, (observando a existência de pontos da superfície que não se moverão) e a detecção de colisão de superfícies deformáveis com objetos sólidos. No caso de haver qualquer tipo de restrição sobre um ponto, é calculada uma força tal que leve o ponto a não se mover ou a se mover obedecendo a restrição em questão.

A colisão da superfície deformável com corpos rígidos é calculada como uma função $f(i, j)$ que verifica para cada ponto da malha se este está dentro do sólido, fora ou na superfície deste. No caso de estar fora ou sobre a superfície, a força de restrição é nula. Entretanto, se o ponto da superfície estiver dentro do sólido, deve ser aplicada uma força que leve o ponto para a superfície.

O algoritmo utilizado na geração dos novos pontos da malha, após cada intervalo de tempo especificado (Δt), é:

Para um tempo igual a t , inicialmente igual a 0 e variando de Δt :

- Calcular a força resultante em cada ponto da malha flexível, através dos métodos já descritos;
- Calcular a aceleração de cada ponto através da fórmula $F = m a$, onde m é a massa e a , a aceleração;
- Com base nessa aceleração, encontrar a velocidade dos pontos através da fórmula: $V = V_a + a \Delta t$, onde V_a representa a velocidade no momento anterior.

Dada a velocidade, encontra-se a nova posição do ponto, através da fórmula:

$P = P_a + V \Delta t$, onde P_a é a posição anterior do ponto e V , a sua velocidade.

Novos desafios a esse algoritmo podem adicionar características inelásticas ao modelo, considerando a deformação definitiva ou fratura das molas que unem os pontos da malha, e pode-se estender a aplicação para objetos tridimensionais e não somente superfícies [Eberhardt et. al, 1996].

6.10. PRODUÇÃO DE ANIMAÇÃO

A produção de uma animação é baseada na habilidade lógica, e criativa, combinando os conhecimentos técnicos do processo de animação com estilos individuais, experiência e instinto.

Na indústria de animação, não existe nenhuma definição única do que um animador faz. Geralmente, a tarefa muda conforme o projeto. Pode-se descobrir que para uma determinada ação suas habilidades e conhecimentos sejam melhores ou piores.

A produção de uma animação tipicamente segue os seguintes passos:

1. Uma história é inventada;
2. Os personagens da história são criados ganhando personalidades e comportamentos;
3. Desenvolve-se um storyboard. No storyboard desenha-se cada tomada de cena na ordem que o filme ocorre. O storyboard fornece uma estrutura e um plano global para a animação ou filme, permitindo ao diretor e sua equipe ter uma idéia clara de onde cada tomada se encaixa no filme. Inclui-se uma breve explicação da tomada;
4. Com base no storyboard são criados os quadros-chave;
5. Para testar os movimentos faz-se o animatic. O animatic é um rascunho animado que vem sendo largamente usado por grandes produtoras graças a popularização dos softwares de animação que agilizam e barateiam o processo;
6. Finalmente, são criados os chamados inbetweening, quadros entre os quadros-chave;

A animação por computador segue todos esses passos com a diferença, que algumas etapas não são mais desenhados a mão.

6.11. PRINCÍPIOS DA ANIMAÇÃO

As animações realistas devem seguir alguns princípios básicos. Esses princípios alertam para observações dos fenômenos físicos e propriedades dos materiais envolvidos na animação. Seguem sete princípios, mas suas observações podem contribuir para novos itens nesta lista.

Achatar e Esticar: todo objeto possui sua rigidez, elasticidade, amortecimento e massa que devem ser consideradas no processo de animação. Esticar e achatar ajuda a acentuar o movimento.

Antecipação: os movimentos devem ser antecipados, por exemplo, se você vai dar um soco o braço deve ir para trás e só então para frente.

Momento: as ações devem estar de acordo com o peso, tamanho e personalidade do personagem.

Ações secundárias: as ações secundárias fornecem um suporte para a ação principal fornecendo reações de efeito físico.

Slow in & Slow out: as ações iniciam-se e terminam lentamente. Esse conceito é devido às leis da inércia e gravidade e pode ser observado no ciclo de uma bola pulando. Esse conceito também é válido para animações de esqueletos.

Arco: quase nada se movimenta em linha reta. Se você observar atentamente notará que terá de fazer um grande esforço para andar em linha reta.

Direção: a direção é a noção que a audiência deve ter para não se perder na história. Este ponto pode se complicar quando voltamos ou avançamos no tempo em uma história.

6.12. ANIMAÇÃO DE CENA

Uma cena é um evento ou interação única entre personagens, acontecendo durante um período de tempo e em um lugar único, que move a história adiante na direção de um clímax e resolução, um evento, um período de tempo, um lugar.

Quando elaboramos uma cena precisamos considerar:

- Ficar fora de clichês e soluções fáceis.
- Fazer o inesperado.
- Usar obstáculos para complicar e fazer reversões inesperadas.
- O que esta cena realizará? As cenas podem ter um ponto principal e diversos pontos secundários. Cada informação realmente importante de enredo em seu roteiro provavelmente exige uma cena separada.
- Quem está na cena?
- Onde e quando será realizada a cena?
- Quem é o protagonista da cena? (Normalmente é uma pessoa ou animal, mas pode ser um objeto inanimado ou até um ato de natureza.)
- O que os personagens querem? Qual será sua atitude?
- Quem está colocando obstáculos? Por quê? O que este personagem quer?
- Os personagens estão conhecendo o problema ou ele é desconhecido?
- Os personagens estão se referindo direta ou indiretamente ao que querem?
- Onde está a tensão ou o conflito?
- O que acontece nesta cena? Os personagens estão mais próximos ou mais distantes do seu objetivo inicial da cena?
- Como a cena revela o caráter e a motivação pelo comportamento dos protagonistas?
- A cena tem um catalisador ou incidente estimulador no princípio?
- A cena está criando um clímax? Como será o clímax, mais engraçado ou mais dramático?

Uma cena geralmente possui um começo, um meio e um fim. Retire qualquer exposição desnecessária. Adicione alguma comédia. Se o público for criança, opte por cenas rápidas com muitas ações. Lembre-se de que as crianças perdem a atenção rapidamente.

6.13. ANIMAÇÃO NO PROCESSO DE APRENDIZADO

A habilidade de concepção, criação e a abstração tem papel fundamental na solução de problemas reais. Muitas pessoas são capazes de resolver problemas se estes forem apresentados em uma forma mais concreta, mas quando estes lhe são apresentados de forma abstrata, sentem enorme dificuldade. Esta deficiência muitas vezes indica que o processo de aprendizagem aconteceu apenas em nível mecânico, no qual o aluno é capaz de aplicar construções e procedimentos, porém sem domínio conceitual dos mesmos. Essa situação o impedirá de resolver problemas que exijam adaptação das técnicas aprendidas ou sua aplicação em outro contexto. Pior ainda, por serem resultado de memorização temporária, esses conhecimentos serão perdidos em prazo muito curto.

Animações, que transformem gradualmente imagens realistas de objetos em representações mais abstratas e vice-versa, são meios auxiliares poderosos para o aprendizado. Essas animações devem fazer parte de metodologias específicas para auxiliar a compreensão de conceitos sejam abstratos que difíceis de serem entendidos.

A transferência de um contexto para o outro é também melhorada pelo uso de animações que ajude os alunos a representar os problemas em níveis mais altos de abstração. Como é o caso da representação de formas de objetos reais por modelos geométricos.

Assim, é interessante usar a capacidade da animação para fixar e transferir o conhecimento de um contexto para outro, fazendo com que os alunos percebam que o conhecimento adquirido pode ser útil em novas situações, isto é, para auxiliar a generalizar e abstrair.

RESUMO

A animação é em muitos casos o objetivo final dos estudos da computação gráfica. Este capítulo não tem a pretensão de ser mais do que uma introdução inicial ao assunto. Dependendo de seu interesse específico na área pode ser muito difícil ou bastante fácil encontrar trabalhos que o auxiliem. Talvez a melhor dica que poderíamos fornecer neste sentido seria visitar os sites dos 13 mais atuantes grupos de animação que cujos endereços se encontram nas referências bibliográficas no final do livro.

CAPÍTULO 7

Realismo Visual e Iluminação

7.1. Rendering

7.1.1. Fases do Processo de Realismo Visual

7.1.2. Realismo por Passadas

7.1.3. Acabamentos não-fotográficos

7.2. Rasterização

7.2.1. Algoritmo de Bresenham para traçado de linhas

7.2.2. Rasterização de Polígonos

7.2.3. Preenchimento de polígonos por scanline

7.2.4. Remoção de Linhas e Superfícies Escondidas

7.3. Iluminação

7.3.1. Tipo de Emissores

7.3.2. Reflexões

7.3.3. Refração

7.3.4. Transparência

7.3.5. Sombreamento (Shading)

7.3.6. Sombras

7.3.7. Modelo de Iluminação Global

7.3.8. Técnicas de Iluminação

7.4. Texturas

7.4.1. Mapas Procedurais

7.4.2. UVW Map

7.4.3. Texture Map

7.4.4. Environment Map ou Mapa de Reflexão

7.4.5. Bump Map

7.4.6. Light Map

7.4.7. Mip-Mapping

7.5. Hiper-Realismo

7.5.1. High Dynamic Range Images (HDRI)

7.5.2. Atenuação Atmosférica

7.5.3. Area Light e Soft Shadow

7.5.4. Sub-Surface Light Scattering

7.5.5. Depth of Field (DOF) ou Profundidade de Campo

7.5.6. Motion Blur / Desfoque por Movimento

7.5.7. Film Grain

7.5.8. Lens Flare

7.5.9. Glow

7.6. Realismo e Iluminação em OpenGL

7.6.1. Z-buffer

7.6.2. Hidden-Surface Removal

7.6.3. Algoritmo de Recorte (Culling)

7.6.4. Iluminação

7.6.5. Texturas

Criar imagens sintéticas realistas é o objetivo final da computação gráfica. Basicamente, podemos definir o realismo visual como as técnicas de tratamento computacional aplicadas aos objetos sintéticos gerados (por modelagem de sólidos, partículas, fractais ou qualquer técnica de geração) com objetivo de criar-lhes uma imagem sintética, o mais próximo da realidade que se teria se eles fossem construídos e filmados.

O realismo é fundamental nas simulações, no entretenimento, na educação e muitas outras aplicações. Como exemplo de aplicação prática, temos a utilização do realismo pelas construtoras, para mostrar aos compradores de um apartamento, por exemplo, como será o ambiente de uma sala, sua iluminação, a vista que se terá de suas janelas, o seu exterior, as possibilidades de decoração, ou seja, para que se conheça a realidade que se poderá ter antes de ela ser realmente construída. Você já deve ter se deparado com esse exemplo ao entrar em uma página de classificados para comprar um imóvel, ou nos sites de construtoras.

No entretenimento, são muitos citados pelas mídias os cenários de filmes e programas. Certamente você já sentiu nos jogos a importância de um cenário ou de um personagem mais real. Na educação, um ótimo exemplo são suas utilizações nas escolas de medicina e nos grandes hospitais. Atualmente, a reconstrução realística tridimensional do corpo humano auxilia no planejamento de procedimentos cirúrgicos (*PLS* ou *Labs Surgical Planning*) e no ensino de anatomia. Ela possibilita a visualização dos órgãos (seu interior, exterior, vistas em fatias e inter-relações) o estudo dos diversos sistemas (por exemplo digestivo, circulatório, nervoso), a visualização em cortes ou camadas estruturais como ossos, músculos, órgãos. Uma ótima idéia aqui é visitar as páginas da Internet, onde podem ser vistos (usados e até obtidos gratuitamente) os atlas de anatomia resultantes do projeto Visible Man and Woman e os diversos softwares como o 3D-Slicer (visual.ic.uff.br/biomedicalp.html).

Podemos considerar o realismo em duas etapas: a estática e a dinâmica. A dinâmica está relacionada ao movimento da cena e seus personagens. Sendo introduzida no capítulo referente à animação. Neste capítulo, descreveremos as técnicas e teorias necessárias para gerar objetos e cenas estáticas com realismo fotográfico.

7.1. RENDERING

A computação gráfica trata da síntese de imagens através do computador. Sintetizar uma imagem não é mostrá-la no computador digitalmente a partir da captura de algo existente. Isso é tratado no Processamento de Imagens. Sintetizar uma imagem (uma cena ou um objeto) é criá-la em termos da definição dos dados dos objetos que a compõem. Isso se faz a partir da geometria da cena, das informações sobre os materiais de que são feitos os objetos (suas cores e suas texturas), das condições de iluminação ambiente; e da posição de observação da cena. Nesse processo de criação sintética, é denominado *rendering* a fase de Introdução nas cenas, do realismo fotográfico.

Se você procurar uma tradução no dicionário para *rendering* achará: (1) uma interpretação de um drama ou uma composição musical; (2) uma tradução; ou (3) uma representação de um edifício, interior etc., executado em perspectiva. Ou seja, não existe uma tradução fora de nossa área adequada para essa palavra, como não existe uma tradução literal para diversos outros termos técnicos que nos contextos onde são usados têm significado próprio. Basicamente, podemos descrevê-la por “realismo visual”. Podemos interpretar o processo de *rendering* como o de converter dados em uma imagem realística ou simplesmente sintetizar um objeto ou cena até ter-se deles uma aparência de algo real e não de formas inteiramente criadas no computador.

7.1.1. Fases do Processo de Realismo Visual

Este processo envolve sete fases distintas. Nem todas usadas em todas as aplicações. O mais usual é que um trabalho seja apresentado em um nível de realismo adequado ao seu uso. Principalmente porque ao aumentar-se o realismo de uma cena aumenta-se também seu tempo de processamento e custo de geração.

Na primeira fase, tem-se a **construção do modelo** que conterá todas as informações necessárias para o processo de realismo visual. Essa primeira fase consiste na utilização de alguma técnica de modelagem. A segunda fase consiste em aplicar transformações lineares ao modelo de modo que ele tenha **aparência tridimensional** nos diversos dispositivos de visualização (geralmente bidimensionais). Essa fase consiste, então, na utilização de projeções e perspectivas adequadas. A fase seguinte considera a **eliminação de polígonos ou faces escondidas** devido à posição relativa entre os objetos da cena e o observador (*culling back-faces*). São com essas técnicas que iniciaremos nosso estudo de realismo visual nas próximas seções. Muitas delas usam os dados tridimensionais dos objetos e devem ser refeitas a cada mudança de ponto de observação ou forma de projeção.

Na quarta fase, são desconsideradas as partes das cenas que não serão mostradas, são os chamados “**recortes**”. Essa fase, até o fim do capítulo, ficará bem entendida, mas agora talvez fique melhor explicada por uma analogia. Imagine que você esteja vendo uma paisagem de sua janela. Mesmo que você tenha uma infinidade de coisas para ver, algumas para sua visão será limitada. Por exemplo, o que estiver atrás de paredes e não de vidros transparentes, detalhes muito longe (para serem bem percebidos) ou coisas muito pequenas (que não merecem ou conseguem chamar sua atenção). Assim, para que a cena fique parecendo “real”, muitos dados presentes no modelo vão deixar de ser mostrados, embora estejam descritos (pelos dados da cena). Isso é chamado de *clipping* (e nem se dê ao trabalho de procurar uma tradução no dicionário para isso, como *rendering*, o importante é o seu sentido no contexto de nossa área).

A quinta fase se preocupa em converter a representação tridimensional para pixels. Seja lá qual for o sistema de coordenadas que se esteja usando, os dados serão

levados para um conjunto de coordenadas do dispositivo em que será mostrado. Essa conversão de coordenadas leva os dados do modelo para o mundo digital. Linhas e áreas contínuas serão transformadas em conjuntos de pixels. Esse processo é denominado de **rasterization** e iremos aqui denominá-lo usando um anglicismo: **rasterização**. Usamos isso no sentido de usar um termo técnico global, que você identificará facilmente ao ler trabalhos escritos em qualquer língua.

A sexta fase é de certa maneira uma continuação da terceira, pois trata também da eliminação de partes de um objeto que devem ser removidas. Só que agora essas partes são devidas à interferência dos diversos objetos presentes na cena, onde, devido à sua posição relativa, pode ocorrer que uns fiquem na frente de partes de outros. Muitos autores englobam essas duas partes no que se chama **tratamento de partes escondidas** (*hidden*). Preferimos seguir a linha dos que usam separá-las, pois a maioria das eliminações de partes de objetos é feita em coordenadas do dispositivo (depois de o processo passar pela quinta fase) enquanto a eliminação de faces inteiras (*culling*) usa as coordenadas tridimensionais do objeto.

A última fase trata de colorir cada pixel individualmente, usando um esquema incremental ou interpolador de sombreamento. Nessa fase, o realismo “fotográfico” começa a ser realmente tratado e percebido. E para que seja feito adequadamente, deve-se levar em conta as luzes presentes na cena, suas intensidades e direções (em relação aos objetos). Também devem ser consideradas todas as características das superfícies representadas: transparência, brilho, reflexão e textura. Ainda devem ser consideradas as sombras que os diversos objetos fazem entre si e nas superfícies em que se apóiam. O nível de realismo dessa fase pode ser tão sofisticado quanto a aplicação precisar, sendo necessário o uso de modelos físicos (de representação da luz e sua interação com os materiais) para um tratamento adequado.

7.1.2. Realismo por Passadas

Pelo já comentado, deve ter ficado claro que a construção de uma cena realística é um processo incremental. Isso ocorre tanto no nível conceitual discutido anteriormente, quanto no nível das técnicas e softwares disponíveis para a realização das diversas fases do processo. Essa forma de criação de cenas realísticas é denominada de realismo por passadas.

Assim, a renderização por passadas é a forma utilizada pela grande maioria dos sistemas para geração de cenas realísticas. Esse processo permite que, os atributos de uma cena sejam renderizados separadamente da sua geração ou modelagem e, em muitos casos que, diversas técnicas e softwares participem do processo de inclusão do grau de realismo desejado à cena.

Uma série de bons motivos tornou a renderização por passadas o padrão da computação gráfica atual. Um desses é a **economia de memória**, pois com essa técnica não precisam ser colocados todos os objetos de uma vez para o render. Isso torna possível a renderização de cenas complexas em um PC comum. Outro é a **facilida-**

de da introdução de modificações, isso é, se precisarmos alterar somente alguns elementos de uma cena como letras, sombras ou cores, não será necessário perder ou refazer todo o processo de renderização anterior. Ele pode ser reiniciado ou alterado a partir do ponto desejado.

Um outro motivo que torna mais eficiente essa forma de renderização é a possibilidade de **maior utilização das imagens estáticas**. Por exemplo, se já temos em uma animação uma cena observada a partir de um ponto estático (como um filme com câmera fixa), por exemplo um quarto com um personagem andando, não será necessário renderizar todo o quarto a cada passo do personagem; só o personagem será modificado a cada quadro.

Essa técnica permite também a *mixagem* com objetos ou texturas reais obtidas por captura, ou seja, o render por passadas permite uma **integração** com imagens fotográficas, acrescentando sombras, texturas complexas ou elementos do mundo real (como rostos) à cena. É possível a **reciclagem** de objetos e cenas geradas anteriormente. Uma boa técnica é a reutilização de objetos em diferentes momentos ou em cenas complexas. Esses objetos não necessitarão ser completamente renderizados, na **renderização por passadas**, poderão apenas ter suas formas de iluminação alteradas para se integrarem ao ambiente da nova cena.

O render por passadas, às vezes, elimina a necessidade do cálculo de anti-aliasing, substituindo-o por um leve blur (borramento). Os **sistemas de partículas** também utilizam as passadas para inclusão de efeitos especiais.

Além dos citados, poderíamos citar ainda diversos outros efeitos como, por exemplo *bump map*, *depth offield* (simulação do foco das máquinas fotográficas e filmadoras) ou *glows* (incandescência), que são introduzidos na última passada e geralmente são inclusões 2D na imagem final.

Agora que já entendemos a importância desta forma de criação de realismo em computação gráfica, vamos entender como o processo é feito em nível de criação da cena, ou seja, como são os tipos de passada nos diversos sistemas disponíveis.

A primeira passada ou **passada principal**, também chamada de *diffuse pass* ou *color pass*, corresponde ao momento em que se atribui cores aos objetos, ou seja, nela estão incluídos o modelo, a iluminação ambiente, e o mapa de cores. Não estão incluídas formas de realismo mais complexas como as reflexões em superfícies espelhadas, os brilhos e as sombras que se projetam dos objetos devido às luzes direcionais.

A passada seguinte inclui a luminosidade direcional, o brilho e as ênfases da cena. Denominada de **highlight pass**, ou também chamada de *specular pass*, ela considera os brilhos especulares dos objetos. Essa passada ocorre devido aos efeitos de pelo menos uma luz direcional na cena. O resultado é uma imagem com os objetos bem iluminados sobre um fundo sem sombra.

A passada da reflexão inclui as **reflexões** dos objetos podendo substituir ou complementar a passada anterior. Nela, serão considerados os espelhos do cenário, os assoalhos encerados, os espelhos d'água, e outras superfícies que refletem as imagens.

A passada de **iluminação** inclui os pontos de luz (lâmpadas, velas) e suas interações na cena. Em vez de utilizarmos a passada **principal** para renderizar toda a iluminação de uma só vez, podemos utilizar esse processo para mostrar a influência de uma luz (ou grupo) em um determinado elemento. Isso permite a inclusão de diversos efeitos interessantes à cena, como a idéia de luzes sendo progressivamente acendidas ou apagadas, a inclusão do efeito de faróis passando, de feixes de raios sendo disparados, de fogos de artifício, entre outros.

A passada seguinte inclui as **sombras**, ou seja, mostra a localização das sombras projetadas pelos objetos no solo ou nos outros objetos da cena.

A passada de **efeitos especiais** inclui efeitos como explosões, curvaturas de lentes de aumento ou distorção, nuvens de fumaça, a visão através de água ou exaustão de turbinas. Esses efeitos, nem sempre presentes em uma cena, são renderizados em separado nesta fase.

A passada de **profundidade** inclui na imagem informações para incluir a noção de profundidade à cena. Diferente de uma imagem real, onde os detalhes mais longe vão perdendo a nitidez, (pois se tornam fora de foco), em uma imagem sintética (gerada pela computação gráfica) todos os dados permaneceram em cena a menos que essa noção seja introduzida artificialmente. Isso é tratado nesta passada e pode ser feito através da inclusão de um filtro de *blurring*, por exemplo.

7.1.3. Acabamentos Não-fotográficos

Os renders realísticos, comentados na seção anterior, tentam fazer uma imagem sintética indistinguível de uma fotografia. Mas em contrapartida, os acabamentos não-realísticos, ou **non-photorealistic rendering** (NPR), também chamados de estilizados (*stylistic rendering*), possuem uma variedade muito grande de aplicações. Um exemplo é a simulação das pinceladas de pinturas reais (Figura 7.1), efeitos criativos imitando serigrafias, ou o estilo de expressão de um artista.



FIGURA 7.1. Render NPR, simulação das pinceladas de pinturas (veja também em cores).

Outra aplicação dos renders NPRs é a criação de imagens similares as do desenho tradicional, onde somente os detalhes significantes para a particularidade da aplicação da imagem são mostrados. Por exemplo, uma fotografia perfeita do motor de um carro pode ser excelente para realizar a venda para o consumidor. Porém, se fôssemos utilizar essa imagem para ilustrar uma mudança significativa no modelo, ela seria muito complexa e de impressão cara. E ainda poderia não deixar claro algum detalhe específico que se desejasse salientar.

Dentre as técnicas dos renders NPRs que mais chamam a atenção, podemos citar a *Toon Shading*, mostrada na Figura 7.2. Essa técnica tenta simular as imagens de desenhos animados (menos complexas e por isso de mais fácil entendimento pelas crianças) assumindo um clima de fantasia.

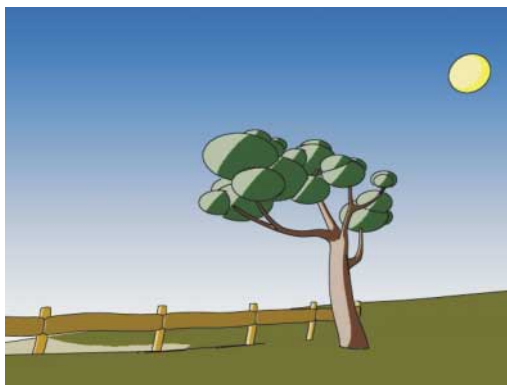


FIGURA 7.2. *Render NPR, Toon Shading (veja também em cores).*

O estilo *toon shading* vem sendo usado pela computação gráfica, desde o início da década de 1990 para integrar modelos tridimensionais com células de animação bidimensional. O toon render (ou *cartoon render*) ganhou vários adeptos pelo mundo devido a sua potencialidade em comunicar emoções e à facilidade para geração de suas imagens, comparada a outros estilos de NPR.

7.2. RASTERIZAÇÃO

Rasterização é o processo de conversão da representação vetorial para a matricial. Ela permite realizar a conversão de um desenho tridimensional qualquer em uma representação inteira possível de ser armazenada na memória (de vídeo ou impressão) de um dispositivo raster. A Figura 7.3 ilustra a rasterização de uma reta.

Grande parte dos dispositivos de entrada e saída, tal como filmadoras digitais, scanners, vídeos e impressoras, usam uma tecnologia matricial, também denominada tecnologia *raster*. Esses dispositivos possuem uma memória na qual é composta a imagem a ser posteriormente exibida no dispositivo.

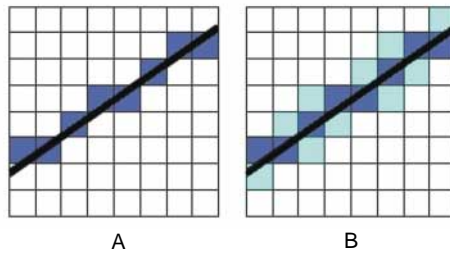


FIGURA 7.3. Conversão da representação de uma reta na forma vetorial para a matricial. Em B, é incluído um tratamento de anti-aliasing.

Um vídeo *raster* é composto de uma memória (Figura 1.2) onde estão armazenadas as informações que descrevem a imagem. Essa memória de vídeo é uma área de armazenamento onde cada posição indica quando um determinado pixel na tela deve estar apagado ou aceso e em qual cor.

A Figura 7.4 mostra um exemplo no qual a cor é especificada na memória de vídeo na forma conhecida como tabela de cores. Um circuito de hardware especial faz a leitura dessa memória e aciona a forma usada pelo hardware para acender o pixel. Se o vídeo usar a tecnologia de tubo de raios catódicos (CRT), por exemplo, o canhão de elétrons será sensibilizado de forma a compor na tela o mesmo desenho composto na memória de vídeo (Figura 1.3).

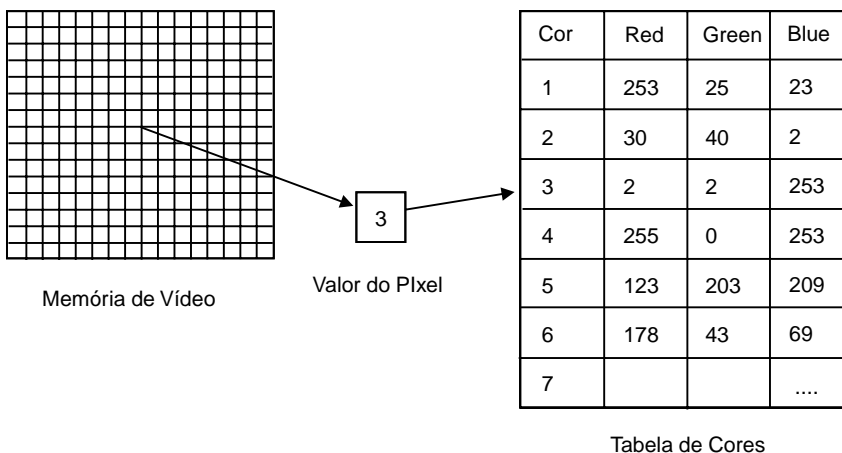


FIGURA 7.4. Memória de vídeo e uso de tabela (paleta) de cores.

7.2.1. Rasterização de retas

Normalmente, gráficos são definidos através de primitivas geométricas como: pontos, retas, círculos, textos etc. Pode parecer simples traçar uma reta no vídeo,

mas, no entanto, esse tipo de rotina não é tão simples quanto parece. Ao tentarmos desenhar uma reta no vídeo, devemos nos lembrar de que essa reta não poderá ser desenhada da mesma forma que a desenhamos em uma folha de papel, ou seja, nem sempre terá uma forma reta perfeita. Ela será desenhada pelos pixels que puderem ser acessos no dispositivo de visualização utilizado, através de uma aproximação a ser obtida com a utilização do quadriculado formado pelos pixels (Figura 7.3).

Dependendo da inclinação traçada, podemos obter uma linha com uma aparência serrilhada. Isso é denominado *aliasing* e é devido às quebras de continuidade impostas pela malha de pontos. Essa quebra das linhas tende a ser muito mais aparente à medida que os pontos apresentados na tela forem de maior tamanho (*dot pitch*) ou que o dispositivo possuir menor resolução (números possíveis de pixels nas duas direções). Esses serrilhados podem ser melhorados através da aplicação de algoritmos de anti-serrilhamento conhecidos como algoritmos de *anti-aliasing*.

Os algoritmos de *anti-aliasing* buscam tentar “enganar” o olho humano. Eles geralmente conseguem isso fazendo as bordas de um desenho ficarem um pouco “borradas”, ou melhor, menos nítidas. Geralmente usam uma cor intermediária entre a cor da linha e a cor do fundo, obtendo assim a suavização do contraste entre as duas cores. Dessa maneira, tem-se uma linha que terá uma aparência mais perfeita para uma pessoa que a observe a uma certa distância. A Figura 7.3.B ilustra essa técnica. Os esquemas representados em A e B nesta figura, permitem uma comparação entre as duas linhas, uma sem e a outra com o tratamento *anti-aliasing*.

Sendo a tela gráfica uma matriz de pontos, é impossível traçar uma linha direta de um ponto a outro. Sendo assim, alguns pontos da tela deverão ser selecionados para representar a reta que se deseja desenhar. Esta seleção é feita pelos algoritmos de rasterização.

7.2.1.1. Algoritmo de Bresenham para Traçado de Linhas

O algoritmo de Bresenham para a rasterização ou linhas considera como dados de entrada a localização de dois pixels (x_1, y_1) e (x_2, y_2) da reta a ser rasterizada.

Para cada ponto a ser traçado, o algoritmo verifica sua distância entre a posição do próximo pixel e a localização da reta no grid. Apenas o sinal do erro é analisado. O algoritmo é mostrado a seguir:

```
x = x1
y = y1
Δx = x2 - x1
Δy = y2 - y1
m = Δy/Δx
e = m - 1/2
for i = 1 to Δx do
    desenhaPonto(x,y)
    while e ≥ 0 do
```



```

        y = y + 1
        e = e - 1
    end while
    x = x + 1
    e = e + m
end for

```

onde e é uma medida do “erro”. A cada iteração, $e = e + \Delta x / \Delta y$. Quando o erro assume um valor positivo, é necessário reduzi-lo, subtraindo “1” do seu valor. A Figura 7.5 serve como base de entendimento do algoritmo. O algoritmo de Bresenham pode ser melhorado se a divisão por Δx for eliminada, passando a utilizar somente aritmética inteira. O novo “erro” será agora: $e = 2e\Delta x$.

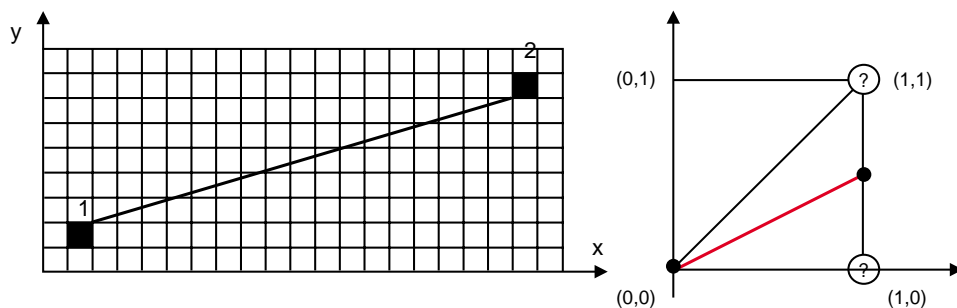


FIGURA 7.5. Base de entendimento do algoritmo de Bresenham.

7.2.2. Rasterização de Polígonos

Os polígonos são geralmente representados na memória pela lista de seus vértices, que são pontos tridimensionais. São projetados na tela dos dispositivos (e transformado em dados 2D inteiros) pela projeção de cada um desses vértices. Os lados 2D resultam da conexão dos vértices por linhas virtuais. Os polígonos são rasterizados, então, primeiro rasterizando todos os seus lados. Isso é feito calculando a interseção de cada uma das linhas horizontais do vídeo, chamadas *scan-lines*, com as linhas virtuais dos lados. A Figura 7.6.A mostra a rasterização dos lados de um polígono, é possível ver as linhas virtuais conectando os vértices e o seu contorno rasterizado.

A forma usual de calcular essas interseções é referida como (*DDA – Digital Differential Analyser*). Sendo a equação da linha entre dois pontos $(x_1; y_1)$ e $(x_2; y_2)$ dada por:

$$y = y_1 + \frac{\Delta y}{\Delta x} x \text{ ou } x = x_1 + \frac{\Delta x}{\Delta y} y$$

onde: $\Delta x = x_2 - x_1$ e $\Delta y = y_2 - y_1$.

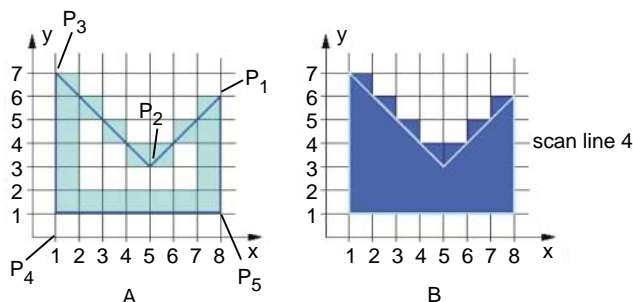


FIGURA 7.6. Rasterização dos lados de um polígono (A) e preenchimento de polígonos por scanline(B).

Para implementar um algoritmo DDA simples, o maior dos valores de Δx ou Δy é escolhido como unidade de rasterização resultando no seguinte algoritmo:

```

Se  $\text{abs}(x_2 - x_1) \geq \text{abs}(y_2 - y_1)$  então
  Tamanho =  $\text{abs}(x_2 - x_1)$ 
else
  Tamanho =  $\text{abs}(y_2 - y_1)$ 
end if
{seleciona o maior dos valores entre  $\Delta x$  e  $\Delta y$  como unidade rasterização}
 $\Delta x = (x_2 - x_1) / \text{Tamanho}$ 
 $\Delta y = (y_2 - y_1) / \text{Tamanho}$ 
i = 1
Enquanto i ≤ Tamanho faça
  DesenhaPonto(Arredonda(x), Arredonda(y)) {Arredonda: valor arredondado de x e y}
  x = x +  $\Delta x$ 
  y = y +  $\Delta y$ 
  i = i + 1
Fim Enquanto
  
```

O problema principal desse algoritmo é que ele utiliza aritmética de ponto flutuante e um arredondamento.

7.2.3. Preenchimento de Polígonos por Scan Line

Exceto nas bordas, pixels adjacentes em um polígono possuem as mesmas características. Essa propriedade é chamada coerência espacial. Assim, os pixels de uma dada linha (*scan line*) variam somente nas bordas do polígono.

O processo de determinar quais pixels serão desenhados no preenchimento é chamado conversão de varredura (*scan conversion*), e mostrado na Figura 7.6.B. A *scan line 4*, por exemplo, pode ser dividida nas regiões $x < 1$ (fora do polígono), $1 \leq x \leq 4$ (dentro do polígono), $4 < x < 6$ (fora do polígono), $6 \leq x \leq 8$ (dentro do polígono) e $x > 8$ (fora do polígono).

A determinação dos pontos de interseção não é feita necessariamente da esquerda para a direita. Caso o polígono seja definido pela lista de vértices P1, P2, P3, P4 e P5, a seqüência das interseções será 8, 6, 4, 1. Será necessário então ordenar a lista obtida, ou seja, 1, 4, 6, 8. As interseções podem ser consideradas em pares. Pixels contidos no intervalo formado por esses pares são desenhados na cor do polígono.

7.2.4. Remoção de Linhas e Superfícies Escondidas

A solução eficiente de problemas de visibilidade é o principal passo do processo de criação de cenas realísticas. Esse problema lida freqüentemente com a determinação da visibilidade de linhas e superfícies. Na descrição das fases do processo de realismo visual no início deste capítulo, essa fase foi denominada de **eliminação de polígonos ou faces escondidas** (*culling back-faces*).

A Figura 4.2A exibe um cubo com todas as linhas e faces visíveis. Embora existam doze arestas no objeto, apenas nove deveriam ser exibidas para criar a impressão de um objeto sólido Figura 4.2.B e C. O problema também pode ser expresso em termos de superfícies ocultas, considerando as faces do cubo Figura 4.50. Quando apresentamos um objeto em perspectiva nem todas as suas faces estarão visíveis. Por exemplo, sabemos que um cubo tem seis faces, mas dependendo do ponto de vista do observador, ele poderá ter uma, duas ou, no máximo, três faces visíveis (Figura 7.7).

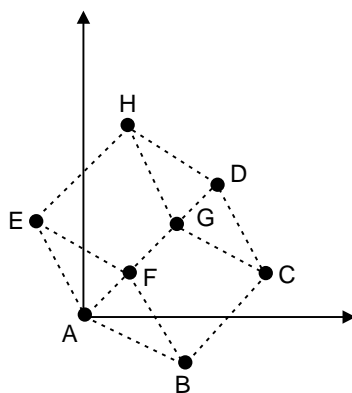


FIGURA 7.7. Um cubo dependendo do ponto de vista pode ter 2 faces visíveis.

O uso apropriado de técnicas de projeção e de eliminação de superfícies e linhas não-visíveis auxilia no objetivo de criar resultados mais realistas. É preciso considerar a posição relativa entre os objetos da cena e o observador, tridimensionalmente, para essa consideração de que partes serão invisíveis por ele.

Para exibir objetos complexos, é necessário descrevê-los internamente no sistema. Comumente são empregadas duas soluções, uma primeira opção é a represen-

tação por um subconjunto de retas e curvas, extraídas dos contornos do objeto, de maneira a representá-lo. A descrição matemática dessas retas e curvas é normalmente usada para gerar o objeto na memória do sistema e sua exibição. Quando o objeto é gerado por essa representação interna, a fase de realismo considerado é o tratamento **das linhas** (sejam elas retas ou curvas) **escondidas**.

Uma segunda solução propõe a representação do sólido através de uma série de faces ou superfícies conectadas apropriadamente. Uma aproximação interna simples pode ser utilizada para descrever cada face ou superfície, e a exibição do objeto é produzida como o agregado dessas faces ou superfícies. Quando o objeto é gerado por essa representação interna, a fase de realismo considerada é a eliminação **das superfícies** (sejam elas planares ou não) **escondidas**.

Freqüentemente na descrição de superfícies utilizamos faces planas triangulares, porque a triangulação de uma superfície qualquer é um processo relativamente simples. A principal desvantagem dessa representação é a necessidade de um grande número de planos para produzir na visualização a impressão de uma superfície suave.

Se o objeto for representado na tela pelas suas linhas (na forma chamada de representação em *wire-frame* Figura 4.2), usualmente, há três formas de se representar as linhas não-visíveis: tracejadas e da mesma cor das visíveis (Figura 7.8); tracejadas, com cor diferente das visíveis; ou simplesmente não traçá-las (Figura 7.9). Normalmente não são traçadas as linhas não-visíveis, visto que o traçado delas pode prejudicar a visualização do objeto na sua forma final.

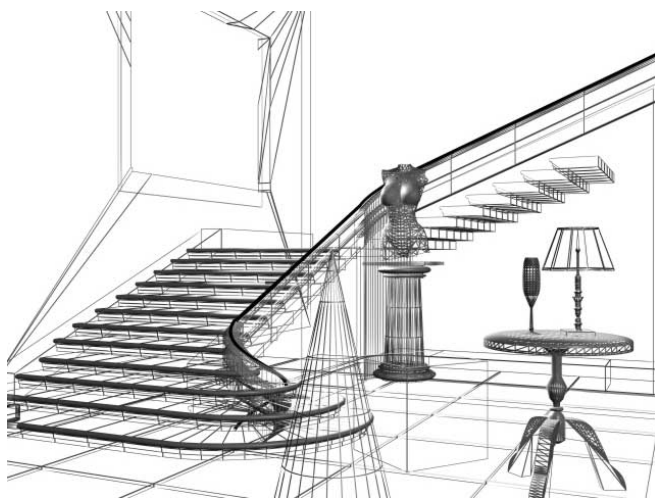


FIGURA 7.8. Representação dos objetos em *wire-frame*.

São consideradas visíveis todas as linhas de contorno das faces do objeto que possam ser vistas a partir de um determinado ponto de vista, ou seja, quando não houver barreiras entre o observador e a linha de contorno do objeto. As linhas invi-

síveis são aquelas que se encontram por trás de uma face opaca ou de qualquer outro obstáculo do próprio objeto.



FIGURA 7.9. *Remoção das linhas não-visíveis.*

É importante uma forma de identificação de linhas, arestas ou superfícies visíveis e não-visíveis, principalmente para que seja possível decidir se a mesma deverá ser traçada ou não, visando principalmente conservar a fidelidade, resolução e agilidade da representação. Em computação gráfica, esses conceitos são muito importantes, principalmente, porque para o computador não faz diferença se as arestas ou superfícies são visíveis ou não.

Os diferentes algoritmos deverão realizar as seguintes tarefas:

- Ler as coordenadas tridimensionais do objeto, e armazená-las em forma de matriz.
- Localizar no espaço 3D a posição do observador, através da qual definirá os parâmetros de visibilidade.
- Calcular o vetor normal 3D de cada face do objeto.
- Calcular o vetor da linha de visibilidade para cada face do objeto. Esse vetor é definido pela ligação de algum ponto da face ao observador.
- Realizar o teste de visibilidade. Isso é feito verificando a magnitude do ângulo formado pela normal, a face em consideração e a linha de visibilidade. Esse é o ângulo identificado com a letra grega beta: β , na Figura 7.10. Se o valor absoluto do ângulo β estiver entre 90° e 180° , a superfície estará invisível. A superfície estará visível se esse ângulo estiver entre -90° e 90° .

A Figura 7.10 exemplifica o teste de visibilidade através do ângulo em um dos lados de um cubo. O teste de visibilidade é o ponto central do **algoritmo de culling**.

- Definir os vértices (ou cantos) das faces do objeto e armazená-los de forma matricial (raster).
- Verificar os vértices (ou cantos) visíveis, com seus respectivos posicionamentos.
- Traçar as arestas das faces visíveis, que revelarão o objeto como enxergado de um determinado ponto de vista. Se desejado, é possível traçar também as linhas não-visíveis (tracejadas ou não) naquele ponto de vista.

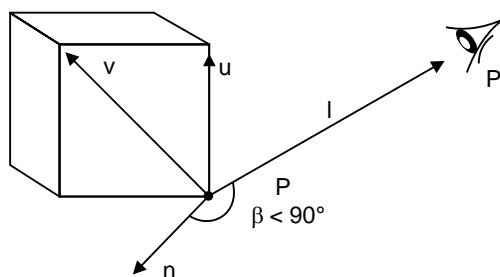


FIGURA 7.10. Vetores de orientação (u e v), vetor normal (n) e de visibilidade (l).

O desenvolvimento dos algoritmos de remoção das faces ocultas ao observador, (*HSR – Hidden Surface Removal*), como maneira de melhorar o entendimento da cena iniciou-se na década de 1970. Muitos algoritmos foram desenvolvidos e caíram no esquecimento. Existem diversas maneiras de se resolver o problema de superfícies ocultas. Algumas são simples mas requerem grande quantidade de memória. Outras usam menos memória mas demandam maior tempo de computação. Outros métodos são rápidos, demandam pouca memória e são simples, porém sua utilidade é limitada a alguns casos especiais. O avanço tecnológico e o barateamento das memórias possibilitaram o armazenamento de quantidades enormes de informação a baixo custo. Assim, as técnicas de determinação de superfícies ocultas baseadas no uso intensivo de memória estão ganhando popularidade. Sem dúvida, dessas a que mais chama a atenção é a denominada *Z-buffer*, presente na grande maioria dos softwares de modelagem e real-time rendering. Como veremos, a escolha de um bom algoritmo de *HSR* pode representar o sucesso ou o fracasso de um sistema. A relevância desse problema não reside apenas na sua complexidade, mas na necessidade de uma solução eficiente que permita a apresentação do objeto em movimento quando uma animação for necessária.

Dentre os algoritmos ainda em uso podemos citar:

- Algoritmo de Visibilidade por Prioridade ou Algoritmo do Pintor;
- Algoritmo de Eliminação de Faces Ocultas pelo Cálculo da Normal;
- Algoritmo Z-Buffer.

7.2.4.1. Algoritmo de Visibilidade por Prioridade

Usando uma linha simplificada de raciocínio como: *se um objeto “A” bloqueia a visão de um objeto “B” e ambos os objetos encontram-se na mesma linha de visão do observador, então o objeto “B” está mais distante do observador que o objeto “A”*, é possível criar um algoritmo que calcule a distância dos objetos ao observador, e que dê prioridade a visualização dos objetos mais próximos ao observador.

O **algoritmo de Visibilidade por Prioridade** baseia-se nessa idéia. Recebe muitas vezes o nome ilustrativo de **algoritmo do pintor** por simular a forma como um *pintor* faria para tratar a visibilidade de uma *tela*, que tivesse *pintando a óleo*, onde os detalhes mais na frente são acrescentados encobrindo os objetos posteriores. Os pontos fundamentais do **algoritmo de Visibilidade por Prioridade** são:

- Calcula-se a distância ao observador de todas as faces poligonais da cena (que chamaremos de D);
- Ordenam-se todos os polígonos pelo valor da sua distância ao observador, D ;
- Resolvem-se as ambigüidades nos casos em que as distâncias ao observador (D) de dois polígonos forem iguais (verificando se ocupam as mesmas posições rasterizadas ou não);
- Desenham-se primeiro os polígonos que tiverem mais distantes do observador (ou seja, os que tiverem maior valor de D).

Nesse processo, deve-se calcular a distância de cada superfície ao observador e ordenar os valores obtidos numa tabela de prioridades. Os objetos mais distantes devem então ser apresentados na tela antes dos objetos mais próximos, por ordem decrescente de distância, de maneira que os objetos mais próximos ao observador sejam “sobrescritos” aos objetos mais distantes, “bloqueando” a sua visualização. Existem diversas variações dessa técnica em função da maneira pela qual se escolhe ou se calcula a distância de um objeto ao plano de projeção ou ao observador. A aproximação mais simples para esse cálculo é usar a distância do *centróide*, ou *centro de gravidade*, da superfície em análise de visibilidade. Essa distância D , considerando-se o observador na origem (0,0,0) do sistema de coordenadas 3D, é obtida por:

$$D = \sqrt{x^2 + y^2 + z^2}$$

onde x, y, z representam as *coordenadas do centróide da face*. Se o observador não estiver na origem dos sistemas de coordenadas, é possível usar a mesma expressão desde que x, y, z representem as *diferenças* entre as coordenadas do ponto onde está o observador e as coordenadas correspondentes do centróide da face em análise. Como operações envolvendo raízes demandam maior quantidade de cálculos para serem realizados, e o importante nesta análise são as posições relativas e não a distância real, a forma que usa a soma das diferenças das coordenadas em módulo é normalmente usada:

$$D = |x| + |y| + |z|$$

Aqui, e em outros pontos deste capítulo, faremos menção a polígonos *convexos* e *não-convexos*. Vejamos, então, qual seu significado. Considerando um único polígono, este é dito ser convexo, se em qualquer posição que se trace um segmento de reta por dois pontos distintos dele não houver interseção com o seu contorno. Caso contrário, havendo interseção, é chamado de *não-convexo*, ou seja, se um segmento de reta unindo pontos do polígono estiver sempre no seu interior ele será convexo, e se isso não acontecer será não-convexo. A Figura 7.11 exemplifica essas duas definições.

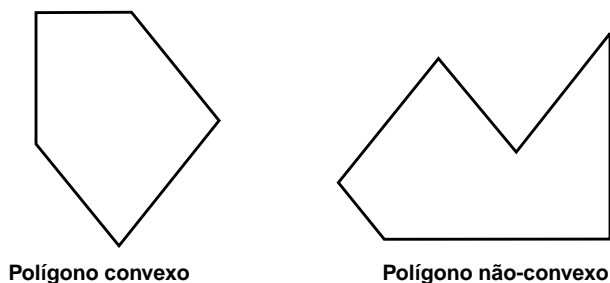


FIGURA 7.11. Exemplo de polígonos convexos e não-convexos.

A Figura 7.12 apresenta polígonos com formas simples que tornam complexa a identificação de qual estará na frente. Como a visibilidade por prioridades baseia-se na suposição de que uma superfície qualquer sempre domina o seu plano de visibilidade, ou seja, se “A” bloqueia “B” então “B” não pode bloquear “A”, o que só é sempre verdadeiro para polígonos convexos, algumas incorreções podem ocorrer. Polígonos não-convexos ou disposições “exóticas” podem bloquear uns aos outros. A utilização de um polígono de contorno pouco usual pode acarretar erro no algoritmo de determinação de visibilidade. Na Figura 7.12.B fica difícil determinar qual superfície aparecerá, mas nesse caso é devido ao uso apenas do centróide das áreas para identificação das distâncias.

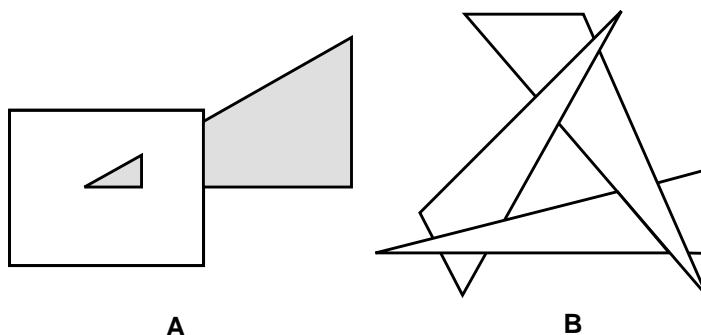


FIGURA 7.12. *Polígonos com formas simples mas que dificultam identificação de qual estará na frente.*

A solução para o problema da determinação de visibilidade nos casos de polígonos não-convexos é usar vários pontos dos polígonos na comparação das distâncias. Ou seja, dividi-los em muitos polígonos convexos. Já no caso de disposições “exóticos”, apesar da aparente simplicidade, as dificuldades que aparecem na resolução das ambigüidades são maiores. Isso geralmente ocorre quando as faces não têm todos os seus pontos com mesma distância ao observador, e estão dispostas de modo que partes de alguma superfície sejam obscurecidas por partes de outra. Isto é, problemas podem ocorrer quando partes de objetos mais distantes obscurecem partes de objetos mais próximos. Nesse caso, será necessário uma série de testes para determinar como os dois polígonos necessitam ser subdivididos e reordenados. Essa bateria de testes, para a resolução das ambigüidades, torna esse algoritmo difícil de ser implementado e muito lento conforme o número de polígonos aumentar.

7.2.4.2. Algoritmo de Eliminação de Faces Ocultas pelo Cálculo da Normal

Uma outra característica das superfícies, que não a distância ao observador, pode ser usada para determinar a sua visibilidade: o ângulo que a sua normal faz com a direção de observação. Se observarmos diretamente uma superfície plana (um cartão, por exemplo) verificaremos que não podemos observar o seu lado oposto. Apenas vê-se a face do plano da frente, não importa qual sua distância ao observador. Se você girar essa superfície de modo a ver a outra face, alguma hora ela será visível apenas como um segmento de reta. Nessa posição em que a superfície é vista como uma reta, sua normal estará perpendicular (90°) à direção dos seus olhos. Assim podemos dizer, através dessa experiência simples, que uma superfície só é visível se sua normal estiver fazendo um ângulo entre -90° e $+90^\circ$ com o observador.

Essa característica é particularmente aplicável nas representações de superfícies visíveis de poliedros convexos fechados, sendo a base do **teste de visibilidade da**

normal (ou *Robert's visibility test*). Como já comentamos anteriormente neste capítulo, esse teste é feito verificando a magnitude do ângulo formado pela normal, da face em consideração, e a linha de visão. Se o valor absoluto deste estiver entre 0° e 90° , então a superfície estará visível. A superfície estará invisível se este ângulo for menor que -90° ou maior que 90° . A Figura 7.10 exemplifica esse teste de visibilidade através do ângulo de um dos lados de um cubo.

Para realização do teste de visibilidade consideraremos então os dois vetores de orientação (u e v) associados a cada uma das faces ou superfícies, o vetor normal (n) de cada uma dessas faces ou superfícies e o vetor da linha de visibilidade (l), que é o vetor que se encontra sobre a reta que vai desde o olho do observador até o ponto de encontro de dois vetores (u e v) localizados sobre a face em estudo.

A verificação de visibilidade é então realizada mediante o cálculo do ângulo (β) entre o vetor normal (n) de cada uma das faces do objeto em relação ao vetor da linha de visibilidade (l). O vetor normal n é perpendicular à face em estudo, pode ser calculado através do produto vetorial de dois vetores contidos na face do objeto: ($u \times v$).

Da álgebra linear, sabemos que o *produto escalar*, \cdot , relaciona dois vetores e o ângulo entre eles, pela seguinte equação:

$$\bar{n} \cdot \bar{l} = |\bar{n}| \cdot |\bar{l}| \cdot \cos \beta,$$

sendo então: $\beta = \arccos \left(\frac{\bar{n} \cdot \bar{l}}{|\bar{n}| \cdot |\bar{l}|} \right)$

onde: $|n|$ corresponde à magnitude ou “norma” do vetor normal da face, n ;
 $|l|$ corresponde à magnitude ou “norma” do vetor da linha de visibilidade, l , que liga a face ao olho do observador; e
 β o ângulo entre os vetores \bar{n} e \bar{l} .

Se o ângulo (β) entre o vetor normal (n) e o vetor da linha de visibilidade (l) se encontrar entre 0° e 90° , então $n \cdot l > 0$, a superfície é visível e pode ser traçada, disponibilizada na tela ou impressa. Se o ângulo (β) estiver entre 90° e 180° , então $n \cdot l < 0$, a superfície estará por trás de alguma outra e, portanto, invisível e indisponível para apresentação ou impressão, ou caso queira exibi-la ela será diferente das visíveis, por exemplo, tracejada ou hachuriada. Na Figura 7.10 tem-se uma representação esquemática dos vetores citados.

Pelo exposto, é interessante observar que se $|\beta| < 90^\circ$ então $n \cdot l > 0$, ou seja, não é preciso considerar a expressão inteira no cálculo da visibilidade, mas apenas o sinal do produto interno de n e l . Resumindo, uma superfície será considerada possivelmente visível quando o produto interno do vetor entre ela e o observador, l , e a normal do plano definido por u e v for positivo.

Assim, a visibilidade de um objeto formado por várias superfícies planas, um cubo por exemplo, pode ser determinada facilmente pela determinação do ângulo

que as normais das suas diversas faces fazem com a direção de observação. Para a determinação da normal de cada face, primeiro deve-se obter dois vetores (u e v) do plano. Esses podem ser obtidos a partir de três pontos quaisquer, não alinhados, contidos no plano. A ordem em que esses pontos são utilizados no cálculo da normal do plano é relevante. Neste exemplo, usaremos a regra de orientar esses pontos sempre no sentido *anti-horário*, conforme ilustrado pela Figura 7.13, para determinação da normal do plano, definido, por P1, P2 e P3, para fora do objeto 3D.

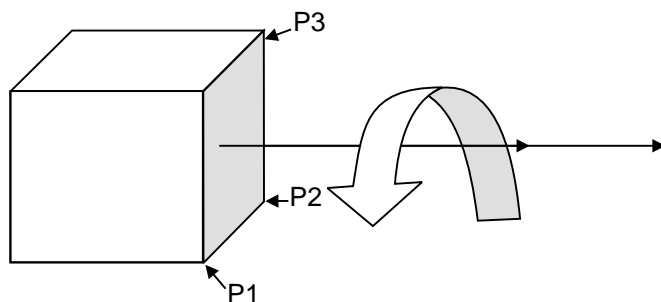


FIGURA 7.13. Determinação da ordem de definição no sentido anti-horário dos pontos P1, P2 e P3 para a definição da normal à superfície.

Assim, o vetor u é definido ligando-se P2 a P3, o vetor v é definido ligando-se P2 a P1, (Figura 7.10) e o vetor l é definido ligando-se P2 a P (posição onde está o observador). A normal é obtida do produto vetorial de u e v , e a resposta da fase ser visível ou não está associada ao produto interno da normal e l , que é então dado pela expressão a seguir em termos das coordenadas de P, P1, P2 e P3. A equação da normal de um plano definido por três pontos, [P1, P2, P3], pode ser obtida pelas seguintes expressões:

$$\begin{aligned}n_x &= (P3_y - P2_y) * (P1_z - P2_z) - (P1_y - P2_y) * (P3_z - P2_z); \\n_y &= (P3_z - P2_z) * (P1_x - P2_x) - (P1_z - P2_z) * (P3_x - P2_x); \\n_z &= (P3_x - P2_x) * (P1_y - P2_y) - (P1_x - P2_x) * (P3_y - P2_y);\end{aligned}$$

Onde: (n_x , n_y , n_z) representa a normal do plano.

Usando as coordenadas do observador, definidas por P_x , P_y , P_z , as coordenadas do vetor l serão: $l_x = (P_x - P2_x)$, $l_y = (P_y - P2_y)$, $l_z = (P_z - P2_z)$; de modo que o produto interno será definido como:

$$n \cdot l = n_x * (P_x - P2_x) + n_y * (P_y - P2_y) + n_z * (P_z - P2_z)$$

E o teste de visibilidade se reduz a verificar o resultado dessa expressão. De modo que: se $n \cdot l = 0$, então o observador está perpendicular ao plano. Se $n \cdot l < 0$, então o plano é visível ao observador. Se $n \cdot l > 0$, então o plano é invisível ao observador.

A partir dessas informações básicas, é possível identificar as faces visíveis e não-visíveis de um objeto, considerando o ponto de vista, e tomar decisões quanto a traçá-las ou não. No caso de linhas escondidas e não faces, a idéia é: são consideradas visíveis as arestas das faces que forem visíveis. De modo que o problema de identificação de linhas invisíveis é uma simples extensão do problema de identificação de faces.

Assim, o algoritmo de definição de visibilidade pela normal pode ser descrito pelas seguintes tarefas:

- Ler as coordenadas do objeto no espaço tridimensional, considerando um ponto de referência e armazená-las em forma de matriz;
- Localizar no espaço a posição do observador, através da qual definirá os parâmetros de visibilidade;
- Calcular o vetor normal de cada face do objeto;
- Calcular o vetor da linha de visibilidade para cada face do objeto;
- Realizar o teste de visibilidade calculando o produto escalar entre os dois vetores:

Se $n \cdot l > 0$, a face estará visível

Se $n \cdot l < 0$, a face estará invisível

- Definir os cantos das faces do objeto e armazená-los de forma matricial;
- Verificar os cantos visíveis, com seus respectivos posicionamentos;
- Traçar as arestas das faces visíveis, que revelarão o objeto como observado de um determinado ponto de vista. Se desejável, é possível traçar também as linhas não-visíveis naquele ponto de vista.

A eliminação de faces ocultas pelo método definido nesta seção, por si só, não constitui uma técnica completa para determinação de superfícies ocultas, uma vez que sua aplicação é limitada, como já dissemos, apenas aos poliedros convexos fechados. No entanto, a aplicação dessa técnica tem a capacidade de, em muitos casos, eliminar grande parte das superfícies a serem consideradas para cálculos de sombreamento e definição da cor das superfícies (como veremos a seguir). Como o tempo de processamento de muitas rotinas de determinação de faces ocultas cresce exponencialmente em função do número de faces (principalmente devido às rotinas de ordenação), a aplicação dessa técnica como um “pré-filtro” costuma trazer uma redução no tempo de processamento de aproximadamente 75%, em muitos processos de criação de cenas com realismo.

7.2.4.3. Algoritmo Z-Buffer

O algoritmo *z-buffer*, desenvolvido inicialmente por Catmull [Catmull, 74], é um dos algoritmos de determinação de visibilidade de superfícies mais simples de se implementar, tanto em *software* como em *hardware*, e hoje é o mais popular dentre

os algoritmos de HSR. Porém, apresenta alto custo de memória e processamento. Requer a alocação de até dois *buffers*, ou matrizes, em memória, com dimensões idênticas à tela de apresentação, normalmente denominados *buffers* de Imagem e de Profundidade.

O uso do *buffer* de Imagem, é opcional, devendo possuir a mesma profundidade de cor da tela de apresentação final. Utiliza-se o *buffer* de Imagem como “rascunho” durante os cálculos de visibilidade dos objetos. O *buffer* de Profundidade destina-se a armazenar a distância de cada pixel, da tela rascunho fictícia inicial, ao plano de projeção, sendo também chamado de *z-Buffer*. Daí o nome do método. Os *z-Buffers* são geralmente implementados em hardware com inteiros de 16 a 32-bits, mas as implementações por softwares devem usar valores de ponto flutuante, para evitar problemas de *aliasing*.

Antes do cálculo e da projeção dos objetos do espaço no plano de projeção, inicializa-se o *buffer* de Profundidade ou *z-Buffer* com os mais altos valores possíveis, representando o valor de distância máxima (todos os demais serão menores) de um ponto ao plano de projeção. O *buffer* de Imagem (ou de rascunho) deve ser inicializado com o valor das cores de fundo da imagem.

Os polígonos que compõem ou representam os objetos no espaço 3D são então projetados na tela de rascunho, pixel a pixel. Para cada novo pixel projetado na tela de rascunho é efetuado o cálculo de distância em relação ao plano de projeção. Se essa distância for inferior à distância armazenada no *z-Buffer*, para aquela coordenada, então os valores de cor desse pixel substituem os valores anteriormente armazenados na tela de rascunho e essa nova profundidade é armazenada no *z-Buffer* para aquela posição. Se os valores de distância forem maiores que os presentes no *z-Buffer*, então esse pixel está sendo “bloqueado” por um ponto, previamente calculado, que já se encontra mais próximo do plano de projeção.

Z-Buffer consiste em usar as coordenadas raster de todos os pontos. Devemos manter a coordenada *z* de todos os pontos que colocarmos na tela (raster) em um enorme array. Quando formos colocar outro ponto na tela no mesmo lugar em coordenadas raster, verificamos se ele está mais perto do observador do que o atual, se não estiver, descartamos o ponto. Como você pode ver, a única ação necessária é computar o valor de *z* para cada ponto.

A idéia básica é:

- Criar e inicializar com a cor de fundo um array bidimensional, que conterá a informação de cada pixel da tela;
- Inicializar o array com o valor da máxima profundidade;
- Achar a coordenada *z* para cada ponto do polígono;
- Testar a profundidade *z* de ponto de cada superfície para determinar a mais próxima do observador;
- Atualizar o valor nos arrays se *z* estiver mais próximo do observador.

O pseudocódigo a seguir ilustra bem esse processo depois das inicializações.

```

Para cada polígono P da cena
  Para cada pixel (x, y) de um polígono P
    computar z_depth na posição x, y
    se z_depth < z_buffer (x, y) então
      defina_pixel (x, y, color)
      troque o valor : z_buffer (x, y) = z_depth

```

A grande vantagem desse algoritmo é que ele sempre funciona e é fácil de implementar. Sua maior desvantagem é a alteração freqüente no valor dos pixels e, como consequência, a obrigatoriedade do cálculo da cor. Outra grande desvantagem desse algoritmo é que, como somente um valor de intensidade e profundidade é armazenado para cada pixel, os problemas relativos ao serrilhamento (*aliasing*) são de difícil solução. Outros problemas relacionados ao uso demasiado da memória vêm sendo relevados pela queda do seu preço e aumento da velocidade de processamento.

7.2.4.3.1. Z-Buffer em Projeções Paralelas

Quando utiliza-se o z-Buffer, nenhum procedimento de pré-ordenação é necessário e todo o processo se resume a sucessivas operações de cálculo de projeção e cálculos de profundidade. Utilizando os mesmos três pontos de uma face usados na seção anterior (Figura 7.13), temos que a forma geral da equação do plano:

$$Ax + By + Cz + D = 0$$

pode ser definida a partir dos três pontos, [P1, P2, P3] pelas seguintes expressões:

$$\begin{aligned}
 A &= P1_y (P2_z - P3_z) + P2_y (P3_z - P1_z) + P3_y (P1_z - P2_z); \\
 B &= P1_x (P3_z - P2_z) + P2_x (P2_z - P3_z) + P3_x (P2_z - P1_z) + P2_x (P2_z - P2_z); \\
 C &= P1_x (P2_y - P3_y) + P2_x (P3_y - P2_y) + P3_x (P1_y - P2_y); \\
 D &= -P1_x A - P1_y B - P1_z C;
 \end{aligned}$$

Assumindo que se use uma projeção paralela no plano XY, temos que a componente Z (profundidade) de um ponto qualquer no plano $Ax + By + Cz + D = 0$ pode ser obtida de:

$$z = \frac{-D - Ax - By}{C}$$

Assim, se já avaliamos z em (x, y), o próximo valor de z1, para o mesmo y, neste plano $Ax + By + Cz + D = 0$, correspondente a $(x1 = x + \Delta x, y)$, será:

$$z_1 = \frac{-D - Ax_1 - By}{C}$$

Assim:

$$\Delta z = z_1 - z = -\frac{A}{C}(\Delta x)$$

A aplicação prática dessa proporção entre as profundidades acontece diretamente para o preenchimento dos polígonos, como ilustra a Figura 7.14, limitando o cálculo completo da profundidade apenas ao primeiro (z_a) e ao último (z_b) ponto da linha de preenchimento (ou *scan line*). Apenas uma subtração é necessária para o cálculo das demais profundidades da *scan line* pois A/C é constante e $\Delta x = 1$. Um cálculo incremental semelhante Δy pode ser utilizado para obter-se os valores de Z_a e Z_b da próxima *scan line*. Observe também que a cor de um pixel não precisa ser calculada se a condição de visibilidade não for satisfeita, poupando cálculos desnecessários. O algoritmo de *z-Buffer* também não requer que os objetos sejam polígonos ou impõe qualquer restrição à forma. Nenhum algoritmo explícito de cálculo de interseção precisa ser utilizado. No entanto, a aplicação direta do *z-Buffer* com projeções em perspectiva não é possível utilizando essa técnica, uma vez que essa projeção “distorce” as proporções lineares do desenho, invalidando a regra de proporcionalidade de primeira ordem já analisada.

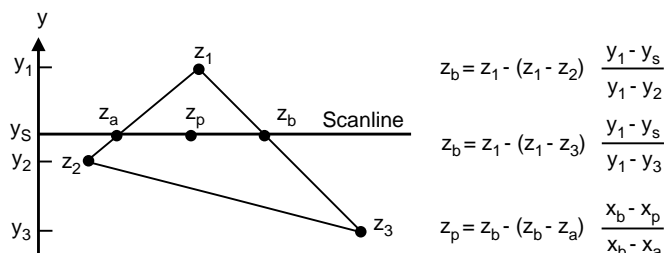


FIGURA 7.14. Determinação de Z_p pela interpolação de Z_a e Z_b para rápida geração da *scan line*. [FOLEY, 93]

7.2.4.3.2. Z-Buffer em Perspectivas

A aplicação de *z-Buffers* em apresentações em perspectiva demanda uma quantidade muito maior de cálculos, sendo necessário o cálculo individual e completo da profundidade para cada pixel projetado na tela. Com o objetivo de reduzir o número dessas operações, a determinação exata do pixel no espaço para o qual a profundidade deve ser calculada é fundamental.

O processo de determinação dos pontos no espaço que geram uma projeção na tela inicia com o cálculo das arestas projetadas de uma face em função de seus vértices. Em perspectiva, a projeção de uma reta no plano é sempre uma reta. Para cada um dos pontos projetados da reta, na tela de rascunho, será necessário calcular a

respectiva profundidade. Para isso, é necessário o cálculo inverso do ponto projetado em função da equação da reta que a originou com o objetivo de determinar precisamente qual o ponto da reta no espaço que gerou a projeção. Para as extremidades da reta, esses pontos são evidentemente os seus pontos delimitadores, no entanto, em função da “distorção” imposta pela projeção em perspectiva, a determinação dos pontos intermediários segue uma complicada regra de proporções de triângulos em função dos ângulos de visada. É recomendável que superfícies complexas sejam decompostas em triângulos e que estes sejam preenchidos com o uso de *scan lines*. Assim sendo, todos os objetos serão descritos como uma sucessão de *scan lines* para as quais se efetua o cálculo da profundidade de seus pixels componentes. A aplicação da scan line segue com o cálculo da projeção inversa, a partir das coordenadas projetadas na tela e da equação de reta, para a determinação do ponto no espaço 3D que originou aquela projeção na tela. De posse do ponto de origem no espaço, aplica-se então a equação padrão para cálculo de profundidade, conforme abordado anteriormente, e procede-se à determinação de visibilidade pelo conteúdo de z-Buffer. O cálculo da projeção inversa é feito em função da evolução da scan line usando a coordenada de x na tela, e da esquerda para a direita, com o objetivo de garantir que não ocorrerão “brancos” no preenchimento das linhas e, para reduzir os efeitos de *aliasing* devidos à precisão finita das matrizes de profundidade.

7.3. ILUMINAÇÃO

A iluminação é um aspecto fundamental em qualquer composição. Primeiramente, vamos entender alguns aspectos técnicos da iluminação natural. Ao nosso redor existe uma série de formas de energia física conhecidas, em seu conjunto, como radiações eletromagnéticas. Algumas características dessas formas de energia estão relacionadas a serem um fenômeno ondulatório.

Analisando um fenômeno ondulatório mais conhecido do nosso dia a dia, como as ondas do mar, podemos dizer que uma onda será completa quando estiver determinada por uma crista positiva e uma crista negativa consecutiva (também chamada período). A distância entre duas cristas iguais é denominada comprimento de onda e geralmente identificada pela letra grega Lambda, como mostrado na Figura 7.15.

Uma radiação (onda) eletromagnética consiste em um campo elétrico e um campo magnético, variáveis com o tempo e com a posição. Um perpendicular ao outro, e ambos perpendiculares à direção de propagação, conforme ilustrado na Figura 7.15. A propagação dessas ondas no vácuo (e no ar) se faz com a velocidade de aproximadamente 300.000 km/s. As ondas eletromagnéticas são caracterizadas por diversas outras grandezas físicas além do já definido comprimento de onda. As principais são a frequência e a energia.

A frequência de uma onda é o número de vibrações por unidade de tempo. O comprimento de onda também pode ser definido como a distância percorrida por

uma onda no intervalo de tempo igual a um período, ou seja, o comprimento de onda é igual a velocidade de propagação da onda no meio multiplicada pela sua frequência. A faixa de comprimentos de onda ou as frequências possíveis de uma radiação eletromagnética são ilimitadas. Atualmente, é possível gerar ou medir radiações em uma faixa que varia em frequência de 1 a 10^{24} Hz (Hertz), ou em comprimentos de onda com intervalo de valores entre 10^{-16} m e 10^{+16} m. Essa extensa faixa define o espectro eletromagnético manipulável pelo homem atualmente.

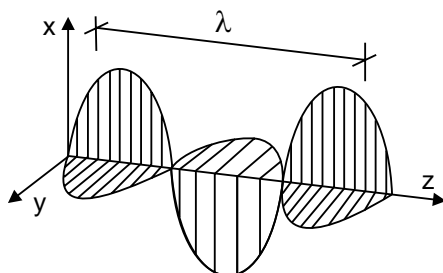


FIGURA 7.15. Em um fenômeno ondulatório distância entre duas cristas iguais é denominada comprimento de onda.

Maxwell mostrou que a luz é uma componente do espectro eletromagnético. Dependendo de seu comprimento de onda (ou frequência), uma radiação eletromagnética pode excitar ou não a visão humana. Quando excita, denominamos de luz visível, ou seja, luz é a radiação que pode ser percebida pelos nossos órgãos visuais. O espectro de luz visível é compreendido no intervalo de 380 a 750 milimicrons (lembrando que $1\mu=10^{-6}\text{m}$), ocupa uma faixa muito estreita do espectro total de radiações eletromagnéticas. A luz visível pode apresentar-se de cores diferentes (veja capítulo de cores), dependendo da frequência. Por ordem crescente de frequência, as cores são: vermelho, alaranjado, amarelo, verde, azul, anil e violeta.

A luz é uma formas de radiações eletromagnéticas. Ela se desloca em linha reta, transportada por uma onda que determina suas características físicas pelo comprimento de onda e frequência. Considerando a energia emitida, podemos dizer que na propagação da luz (em todas as direções a partir da fonte), sua intensidade diminui em relação inversa ao quadrado da distância à fonte de luz.

Esse conceito é muito importante para focalizar os pontos que devem ser corretamente iluminados. Outro fato é que a energia não se perde, se transforma. A luz pode sofrer diferentes processos de transformação. Por exemplo, uma intensa luz branca, caindo sobre uma superfície preta, será quase totalmente absorvida e convertida em calor, que é outra forma de energia com diferentes comprimentos de onda.

A computação gráfica refere-se frequentemente às regras da ótica e à física das radiações para explicar a interação da luz nos objetos. No entanto, a complexidade ou

a inexistência de modelos completos e abrangentes levam os programadores a simplificações do processo computacional. Em consequência, muitos modelos de iluminação atualmente em uso em computação gráfica contêm simplificações sem nenhum fundamento teórico mas que alcançam um bom resultado prático.

7.3.1. Tipos de Emissores

Retornando para o modelo de iluminação digital, temos que neste modelo todo objeto em uma cena é potencialmente uma fonte de luz. A luz pode ser *emitida* ou *refletida* de objetos. Geralmente, fazemos uma distinção entre emissores de luz e refletores de luz. Os emissores são as fontes de luz (lâmpadas, velas, fogo, sol, estrelas), e os refletores são normalmente os objetos que serão coloridos de maneira realística (*renderizados*). As fontes de luz são caracterizadas por suas intensidades e frequências (ou comprimento de onda) enquanto os refletores são caracterizados pelas propriedades de suas superfícies como cor, material e polimento.

A escolha do tipo a ser usado para representar uma fonte ou um emissor de luz depende diretamente do ambiente da cena. Enquanto cenas externas utilizam uma fonte natural como fonte principal de iluminação (sol, estrelas ou lua), cenas internas geralmente utilizam diversas luzes artificiais que podem ter ou não mesma intensidade. Podemos, então, classificar os tipos de emissores como natural, ambiente ou artificial.

7.3.1.1. Emissor Natural

Os emissores naturais tentam simular as duas principais fontes de luz da natureza: o Sol e a Lua. Para propósitos práticos, a luz solar tem raios paralelos vindo de uma direção única. A direção e o ângulo desses raios variam dependendo da hora do dia, da latitude da região e da estação do ano.

A luz solar combina diversos comprimentos de onda de modo que geralmente é definida como branca, mas pode ter algumas variações. Em dia de céu claro, a cor da luz solar é um amarelo-pálido, com valores em RGB em torno de (250, 255, 175). Em dia nublado, a luz solar pode atingir uma coloração azulada, tendendo para cinza, em dias tempestuosos. No amanhecer e no pôr-do-sol, a cor pode tender para laranja ou vermelha. Outra alteração ocorre pela presença de partículas no ar, que podem alterar a cor da luz solar para laranja ou marrom.

A luz solar (ou da Lua) é uma luz direcional. Se for necessário para determinada aplicação (por exemplo, para mostrar a sombra produzida pelo sol do amanhecer ao entardecer), pode ser animada com razoável precisão, seguindo as coordenadas geográficas do local, o movimento do sol em torno da Terra e o ângulo que faz com a posição dada. Você deve considerar, se todo realismo for necessário, além da posição, a data, o horário, e a orientação em relação aos pontos cardinais (norte, sul, leste, oeste, sudoeste, noroeste etc.). As sombras projetadas assim podem representar com precisão o passar das horas.

7.3.1.2. Luz Ambiente

Uma luz é considerada como ambiente quando não tem uma direção possível de ser observável. Esse modelo simula a iluminação geral vinda da reflexão da luz em muitas superfícies difusas. Essa luz determina o nível de iluminação das superfícies no ambiente. É usada para cenas de exterior, quando a iluminação do céu produz uma distribuição da luz refletida (céu nublado). Uma técnica muito comum para melhorar o realismo em cenas externas é alterar a cor da luz ambiente para complementar a luz principal.

7.3.1.3. Emissores de Luz Artificiais

As luzes artificiais são todas aquelas que não são naturais. Em computação gráfica, geralmente simulam as lâmpadas convencionais. Dentre as quais podemos citar as: Luzes Fotométricas, Luz Omni; Luz Direcional e os Refletores.

7.3.1.3.1. Luzes Fotométricas

Com as luzes fotométricas, a atuação da iluminação depende diretamente da intensidade de energia, estando relacionada diretamente com as unidades fotométricas. Nesse tipo de iluminação, a fonte de energia pode ser direcionada para um ponto e ser representada nas formas: pontual (uma única lâmpada), linear (lâmpadas com determinado comprimento como as fluorescentes convencionais) ou áreas iluminadas (painéis luminosos). A Figura 7.16, usada também para demonstrar a sombra nesse tipo de luz, exemplifica uma luz fotométrica linear. Essas luzes devem representar de forma realista a intensidade de energia e a influência do tipo de lâmpada. As lâmpadas podem ser do tipo incandescente, fluorescente, alógena, mercúrio, sódio ou um tipo definido pelo usuário. A intensidade da iluminação é medida pela quantidade de Lumens ou Candelas e podem ser obtidas através de tabelas ou especificações dos fabricantes de lâmpadas reais. As especificações podem ser utilizadas para simulações e teste de iluminação devendo seguir os padrões *IES (Illuminating Engineering Society)*, *CIBSE* ou *LTLI*.

7.3.1.3.2. Luz Omni

Uma luz omni lança os raios em todas as direções de uma origem única podendo criar sombras e projeções (Figura 7.17).

7.3.1.3.3. Luz Direcional (Direct Light)

Raios de luzes direcionais paralelos iluminam, em uma direção única, como o sol faz na superfície da Terra (Figura 7.18). As luzes direcionais são principalmente usadas para simular luz solar. Você pode ajustar a cor da luz, posição e girar no espaço 3D.

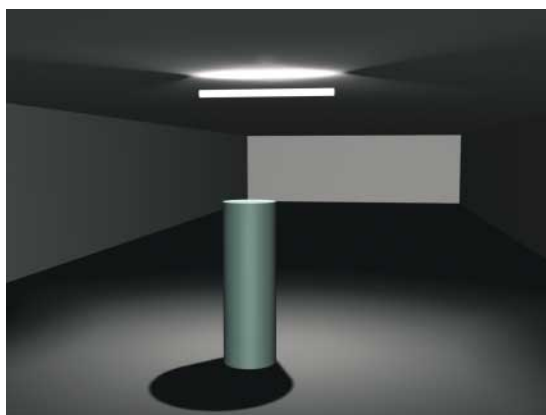


FIGURA 7.16. Exemplo de uma luz fotométrica e seu sombreamento.

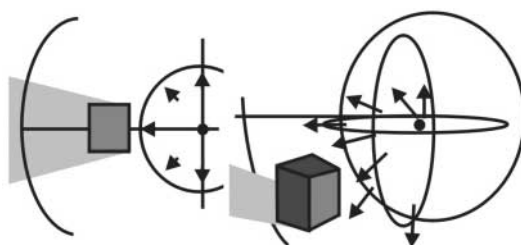


FIGURA 7.17. Trajetória dos raios de luz de uma lâmpada Omni.

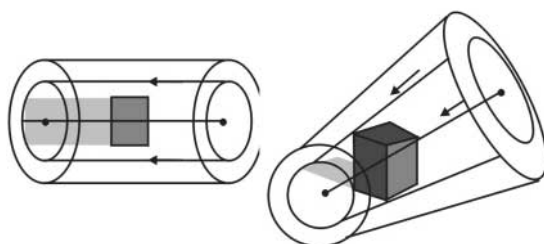


FIGURA 7.18. Trajetória dos raios de luz de uma lâmpada Direcional.

7.3.1.3.4. Luz Refletora (Spot Light)

Um refletor lança um raio de luz focado levemente como uma lanterna ou um farol (Figura 7.19). Esse tipo de emissor pode ser mapeado para produzir o efeito da luz como o utilizado no filme *Batman* (Figura 7.20).

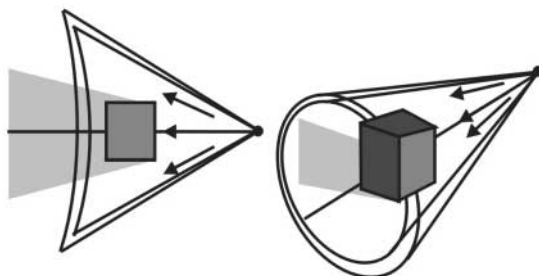


FIGURA 7.19. *Trajetória dos raios de luz de uma lâmpada Refletora.*

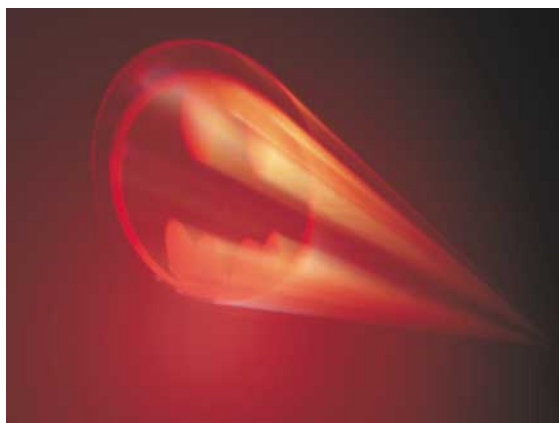


FIGURA 7.20. *Mapeamento de um bitmap na lâmpada Refletora (spot light).*

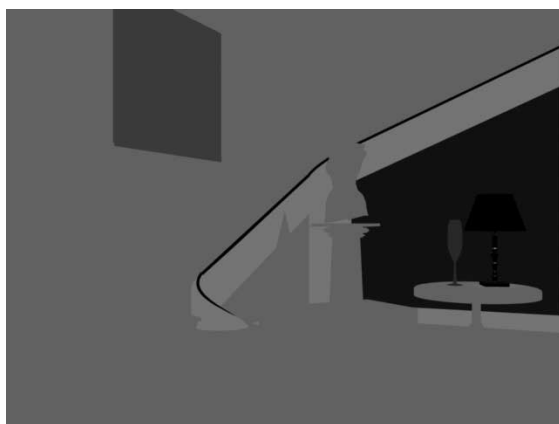


FIGURA 7.21. *Reflexão Ambiente aplicada em uma cena.*

7.3.2. Reflexões

Em computação gráfica, a manipulação da luz assume um papel fundamental no aspecto realístico da apresentação. Os efeitos da luz sobre as superfícies e seus materiais, o obscurecimento de superfícies em função de sua posição, orientação e características da luz são, portanto, peças-chave.

A maioria dos objetos ao nosso redor não emite luz própria, pelo contrário, reflete a radiação neles incidente em diferentes comprimentos de onda. A reflexão se deve à interação molecular entre a radiação incidente e o material que compõe a superfície dos objetos.

7.3.2.1. Reflexão Ambiente

A reflexão ambiente atinge as superfícies igualmente em todas as direções, a partir de uma fonte de luz difusa não-direcional. Em ambientes reais, há superfícies que não são iluminadas diretamente mas também não são completamente escuras. Luzes geradas por reflexão em outras superfícies servem para iluminar essas áreas.

O modelo de iluminação mais simples, que é utilizado na geração de imagens digitais, usa a própria cor do objeto atenuada por um fator de iluminação ambiental (Figura 7.21). Esse fator é fruto do cálculo das reflexões múltiplas da luz nas muitas superfícies presentes no ambiente. Estas múltiplas reflexões combinadas dão origem ao fenômeno definido como luz ambiente. Ao assumir-se que a luz ambiente afeta igualmente todas as superfícies de uma cena em todas as direções, pode-se representá-la como:

$$I = I_a r_a$$

onde I é a intensidade da componente de luz do ambiente na superfície em estudo, I_a é a intensidade da luz ambiente, assumida como constante para todos os objetos. A quantidade de luz ambiente refletida pela superfície de um objeto é determinada através do parâmetro r_a ou coeficiente de reflexão da iluminação ambiente da superfície, que varia de 0 a 1. O coeficiente de reflexão ambiente é uma propriedade do material que compõe a superfície do objeto que se está sombreando e dos pigmentos que a cobrem. A Figura 7.22 mostra um objeto que é sombreado segundo esse modelo, percebe-se que todos os pontos do objeto refletem a luz com a mesma intensidade.

É possível imaginar esse modelo como não contendo nenhuma fonte de luz pontual. O coeficiente de reflexão ambiente é uma conveniência empírica e não corresponde diretamente a qualquer propriedade física real dos materiais. Além disso, a luz ambiente é apenas uma entre outras componentes luminosas a serem consideradas na composição de um modelo mais adequado de sombreamento.



FIGURA 7.22. Reflexão Ambiente aplicada em um objeto.

7.3.2.2. Reflexão Difusa

Uma superfície é definida como um difusor perfeito se ela é capaz de refletir a radiação incidente igualmente em todas as direções. Isso significa que a quantidade de radiação visível (luz) refletida e percebida pelo olho humano não depende da posição do observador.

Superfícies foscas apresentam um tipo de reflexão conhecido como *quase Lambertiana*. Esse tipo de comportamento tem esse nome em homenagem a Jean-Henri Lambert (1728-1777), matemático e físico francês a quem devemos, entre outras coisas, a demonstração da irracionalidade do π (Pi) em 1768, teoremas sobre a trajetória geométrica dos cometas e que estabeleceu os fundamentos da fotometria. Superfícies *Lambertianas* apresentam brilho uniforme, sem reflexão especular, em todos os ângulos de observação de sua superfície. A variação da intensidade da luz refletida é proporcional ao co-seno do ângulo de incidência da luz na superfície. Assim, pode-se obter a intensidade da reflexão difusa como:

$$I = I_d r_d \cos \theta$$

onde I é a intensidade da componente difusa da superfície em estudo, I_d representa a intensidade da fonte de luz direcional presente, r_d representa a reflectividade de luz difusa da superfície e θ é o ângulo que a direção do feixe de luz faz com a superfície, como mostrado na Figura 7.23.

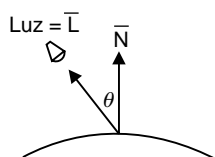


FIGURA 7.23. Reflexão Difusa.

A reflexão difusa faz o objeto refletir a luz que recebe uniformemente em todas as direções. As superfícies sombreadas segundo esse modelo não vão mais aparecer com mesma intensidade se tiverem diferentes ângulos em relação a fonte de luz. A Figura 7.24 mostra um objeto que é sombreado segundo esse modelo, percebe-se que agora é possível identificar a curvatura das superfícies do objeto, já que cada ponto reflete a luz de forma variável com a direção que sua normal faz com a direção da luz.



FIGURA 7.24. Reflexão Difusa aplicada em um objeto.

Assim, um modelo mais completo de iluminação deve considerar ambas, a luz ambiente e a direção de iluminação:

$$I = I_a r_a + I_d r_d \cos \theta$$

Mas esse modelo ainda tem diversas deficiências como: não modelar bem diversos objetos em cena com mesma orientação mas com distâncias diversas do observador (Figura 7.25); como planos paralelos mas a distâncias diferentes. Como a intensidade da luz refletida é inversamente proporcional à distância, um modelo que considera a distância entre os objetos e o observador, d , é:

$$I = I_a \cdot r_a + I_d r_d \cos \theta / (d+k)$$

onde k é uma constante da cena, usada para evitar algumas ambigüidades, como divisão por números muito pequenos e grande diferença entre objetos muito próximos. Outro bom modelo considera essa influência função do quadrado da distância:

$$I = I_a \cdot r_a + I_d r_d \cos \theta / (d+k)^2$$

ou como um fator de atenuação, f_{at} , cujo valor máximo seja 1, resultando:

$$I = I_a \cdot r_a + f_{at} I_d r_d \cos \theta$$

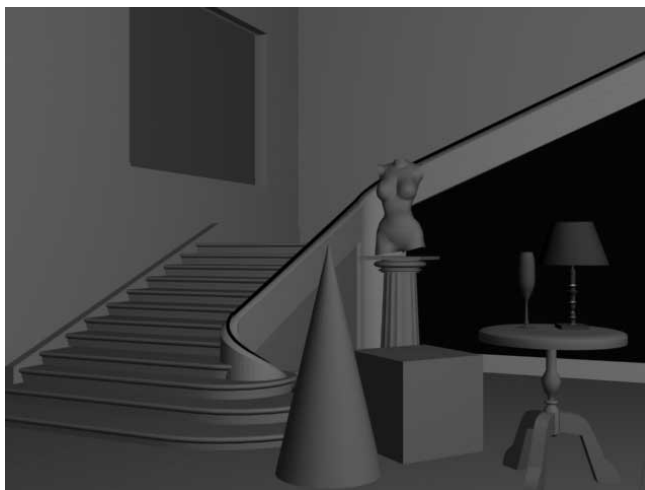


FIGURA 7.25. *Reflexão Difusa aplicada em uma cena.*

Na reflexão difusa, cada superfície tem uma reflexão característica, que determina quanto da luz será refletida. A quantidade de luz refletida não depende do ângulo de visão da superfície, mas sim de sua orientação em relação a direção de luz. Se a fonte de luz direcional for pontual, a intensidade da reflexão difusa varia de ponto para ponto e é melhor caracterizada pela variação do ângulo entre a normal da superfície e a fonte de luz:

$$I = I_d r_d (\overline{u_e \cdot u_n})$$

onde I , I_d e r_d têm os mesmos significados das expressões anteriores, mas u_e representa um vetor unitário na direção da fonte de luz e u_n , o vetor normal unitário da superfície em estudo.

7.3.2.3. Reflexão Especular

Superfícies brilhantes, polidas ou lustras apresentam variações drásticas de intensidade da luz refletida em determinados ângulos de observação. Tal efeito é conhecido por reflexão especular e é um fator importante a ser considerado na geração de imagens sintéticas por computador.

A reflexão especular é a componente responsável pelo brilho (*highlight*) da luz no objeto. Nesse tipo de reflexão, o fóton não interage com os pigmentos da superfície deixando a cor da luz refletida igual à cor original da luz incidente. Basicamente, determinados pontos da superfície atuam como espelho refletindo a luz incidente sem atenuações (Figuras 7.26 e 7.27).



FIGURA 7.26. Reflexão Especular aplicada em um objeto.

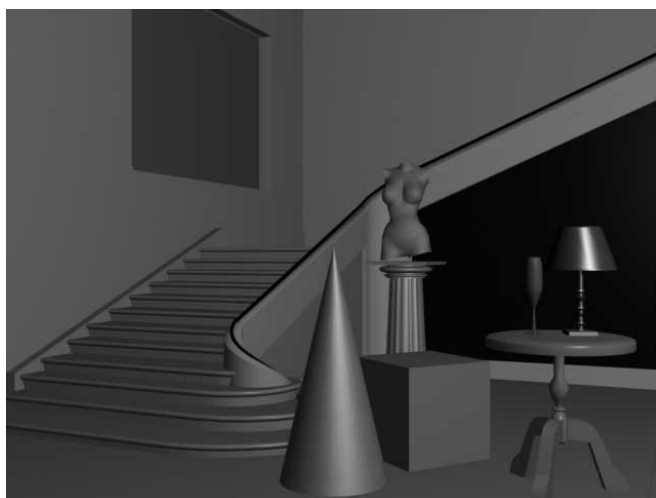


FIGURA 7.27. Reflexão Especular aplicada em uma cena.

Existem vários modelos de iluminação diferentes que expressam e controlam os fatores que determinam a cor de uma superfície em função de um determinado conjunto de luzes. Alguns modelos de iluminação controlam o efeito da luz para cada pixel na imagem, enquanto outros consideram um modelo de iluminação para alguns poucos pixels e aplicam interpoladores para o cálculo dos demais. Dentre os modelos que calculam a contribuição da componente especular estão o de Phong, baseado em um modelo de fonte de luz pontual, e o modelo de Cook-Torrance, que considera a energia incidente sobre o objeto. Recentes melhoramentos no modelo de reflexão especular permitem refletir luzes em formas e superfícies variadas.

O modelo de Phong para luz especular é o mais usado. Esse modelo considera a reflexão especular como uma função do ângulo que a direção de reflexão faz com o observador. Ele é função da cor apenas da fonte de luz. Os modelos anteriores (para a luz ambiente e a luz direcional) eram função da cor dos objetos a serem modela-

dos. Geralmente, usam-se três canais em um determinado espaço de cores. Isto é, a intensidade I seria decomposta em três valores (por exemplo Hue, Saturation e Brightness, ou Red, Green e Blue). Sendo descrito por:

$$I_l = I_{al} \cdot r_{al} + f_{at} I_{dl} (r_{dl} \cos \theta) + f_{at} I_d r_s \cos^n \alpha$$

onde I deve ser cada um dos canais separadamente, por exemplo $I = R, G, B$ ou $I = H, S, V$. Para evitar o uso de muitos símbolos e expressões grandes demais, assumiremos que está claro na exposição que outros tipos, menos a luz especular, devem considerar os três componentes, e escreveremos simplesmente:

$$I = I_a \cdot r_a + f_{at} I_d (r_d \cos \theta + r_s \cos^n \alpha)$$

Repare que agora temos mais uma constante, r_s , de luz especular e mais um ângulo envolvido, α . Esse ângulo representa a direção de observação, como mostrado na Figura 7.28. A potência n usada nesse modelo geralmente é uma variável dependente da superfície ser mais ou menos especular, variando de 1 a 200. Como no caso anterior, podem ser usados os vetores unitários nas direções de reflexão, R , e de visão, V : u_r e u_v . A equação assim pode ser escrita em termos do produto interno entre os dois vetores.

$$I = I_a \cdot r_a + f_{at} I_d (r_d (u_e \cdot u_n) + r_s (u_r \cdot u_v)^n)$$

A potência n simula o fato de as superfícies com brilho apresentarem variações drásticas na intensidade da luz refletida à medida que o ângulo de observação se afasta do ângulo de reflexão, conforme ilustrado pela Figura 7.28. O caso limite de brilho especular é um espelho perfeito, onde só são refletidas as cenas vistas na direção de reflexão. Esse é um fato importante a ser considerado na geração de imagens sintéticas por computador, que contiverem espelhos na cena.

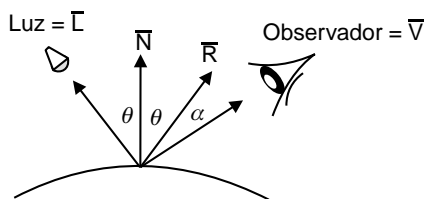


FIGURA 7.28. Reflexão Especular.

O modelo de iluminação engloba os efeitos de múltiplas fontes de luz, em uma única expressão, fazendo um somatório de efeitos quando mais de uma fonte de luz é considerada:

$$I = I_a \cdot r_a + \sum_{j=1}^J I_{dj} (r_d (u_{ej} \cdot u_n) + r_s (u_{rj} \cdot u_v)^n) / (d_j + k_j)$$

ou

$$I = I_a \cdot r_a + \sum_{j=1}^J f_{atj} I_{dj} (r_a (u_{ej} \cdot u_n) + r_s (u_{rj} \cdot u_v)^n)$$

onde:

- I = intensidade do ponto amostrado nos canais cor;
- I_a = intensidade da iluminação ambiente global;
- r_a = coeficiente de reflexão da iluminação ambiente do material;
- I_d = representa a intensidade da j -ésima fonte de luz direcional presente;
- r_d = coeficiente de reflexão difusa do material;
- u_n = vetor unitário normal à superfície do objeto no ponto;
- u_{ej} = vetor na direção da j -ésima fonte de luz;
- J = número das fontes de luz que iluminam o ponto, consideradas no somatório;
- r_s = coeficiente de reflexão especular do material;
- u_r = vetor unitário na direção de reflexão da j -ésima fonte de luz;
- u_v = vetor unitário na direção de observação;
- n = expoente do polimento da superfície;
- d_j = distância da j -ésima fonte de luz;
- f_{atj} = coeficiente de atenuação da j -ésima fonte de luz;
- k_j = constante de iluminação da j -ésima fonte de luz; e

Os dois últimos elementos, d_j e k_j , podem ser substituídos, por f_{atj} . A cor de cada pixel é determinada individualmente através do cálculo da luz (I) que chega até o ponto de observação usando cada uma das componentes do modelo de cor considerado, o RGB por exemplo.

Uma alternativa ao modelo de luz especular de Phong é o uso do vetor de *intensidade de luz especular máxima* ou *vetor de caminho médio*, H , esse vetor é definido usando a direção da fonte de luz, L , e de visualização, V , como:

$$\bar{H} = \frac{\bar{L} + \bar{V}}{|\bar{L} + \bar{V}|} \text{ ou } \bar{H} = (\bar{L} + \bar{V})/2 \text{ se } |\bar{L}| = |\bar{V}| = 1$$

Se a superfície a ser iluminada tiver a normal na mesma direção que H , o observador verá o máximo brilho especular, pois R e V estarão na mesma direção. Esse mo-

delo então substitui $(u_r \cdot u_v)^n$ por $(u_n \cdot u_h)^n$. Isto é, em vez dos vetores unitários nas direções de reflexão, R, e de visão, V: u_r e u_v , usa-se os vetores unitários nas direções da normal, N, e o vetor H: u_n e u_h .

Quando a fonte de luz e o observador estiverem no infinito, essa substituição oferece vantagens computacionais, pois H terá direção constante. Esse modelo é considerado em algoritmos de iluminação global, como o ray tracing que faz essa determinação através do modelo de iluminação de Whitted por essa referência.

7.3.3. Refração

Um feixe de luz incidente sobre uma superfície líquida é refletido e desviado ao penetrar na água. Esse desvio é chamado de refração. Os raios de luz incidentes, refletidos e refratados estão contidos em um mesmo plano. O cálculo da refração leva em consideração a *Lei da Refração*, que determina que um raio de luz sempre sofre um desvio na sua trajetória quando transitar por espaços com densidades diferentes. Além do desvio, é necessário considerar uma atenuação na energia de propagação do raio de luz, uma vez que nenhum material é perfeitamente transparente e que essa transparência é frequentemente diferente em materiais de densidades diferentes (vidro, ar, água etc.).

A *Lei da Reflexão* já era empregada por Euclides. A *Lei da Refração* foi descoberta experimentalmente em 1620 pelo astrônomo e matemático neerlandês Willebrod Snell (1580-1626) e deduzida da primitiva teoria corpuscular da luz por René Descartes (1596-1650). É conhecida como lei de Snell ou como lei de Descartes (na França), ou ainda lei de Snell-Descartes. As leis da *Reflexão* e da *Refração* podem ser derivadas das *equações de Maxwell* (ou equações fundamentais do eletromagnetismo), o que significa que são válidas em todas as regiões do espectro eletromagnético e não só para a luz visível. O cálculo da refração é uma aplicação direta da Lei de Snell-Descartes. A Figura 7.29 e a equação a seguir ilustram a alteração no curso de um raio de luz que passa para um meio de densidade diferente. Os ângulos de incidência θ_i , de reflexão (também igual a) θ_r e de refração θ_t são medidos entre a normal à superfície e o raio correspondente.

$$\frac{\sin \theta_i}{\sin \theta_r} = n_{21}$$

Nesta equação, n_{21} é uma constante, chamada índice de refração do meio 2 em relação ao meio 1.

Segue uma tabela com índices de refração (IR) para vários materiais transparentes comuns, em relação ao vácuo. Os índices em relação ao ar (com exceção naturalmente do próprio ar) são praticamente idênticos:

| Material | IR |
|---|--------|
| Ar (em temperatura e pressão padrão ou STP) | 1,0003 |
| Água | 1,33 |
| Álcool etílico | 1,36 |
| Vidro | 1,66 |
| Plástico | 1,51 |
| Vidro Denso | 1,52 |
| Sal | 1,53 |
| Quartzo | 1,46 |
| Cristal | 1,58 |
| Diamante | 2,42 |

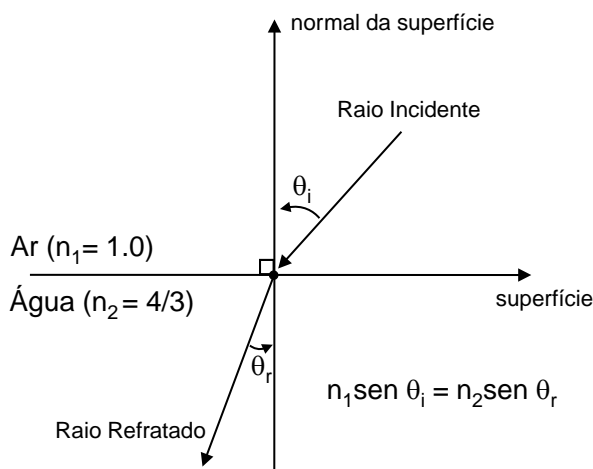


FIGURA 7.29. Geometria do efeito de refração usada na Lei de Snell-Descartes. [WATKINS, 92]

7.3.4. Transparência

O modelo de iluminação apresentado anteriormente considera apenas objetos e superfícies opacas. Entretanto, nem todos os materiais utilizados para confeccionar objetos são opacos: alguns, como o vidro e a água, transmitem luz. Quando um raio de luz passa de um meio para outro, normalmente ocorre o fenômeno de refração. Como já citamos, a intensidade com que o raio de luz é desviado na refração é determinada pela lei de Snell. Essa lei recebe esse nome em homenagem ao seu formulador, o astrônomo e matemático Willebrord Snell van Royen.

Nenhum material transmite toda a luz incidente. Na maioria dos casos, uma parte da luz é refletida ou absorvida. Por analogia com a reflexão especular e difusa, a luz pode ser transmitida especular ou difusamente. Materiais transparentes transmitem a luz especularmente, enquanto materiais que transmitem a luz difusamente têm aparência fosca ou translúcida.

As implementações mais simples do efeito de transparência ignoram a refração e suas consequências. Efeitos simples de transparência podem ser diretamente incorporados ao processo de tonalização. A idéia é primeiro identificar todas as superfícies transparentes. Quando uma superfície visível é transparente, uma combinação linear das duas superfícies mais próximas é armazenada no *frame buffer* para ser mostrada. A intensidade é então,

$$I = t I_1 + (1-t) I_2, 0 \leq t \leq 1$$

onde, I_1 é a superfície visível, I_2 é a superfície imediatamente atrás da superfície visível, e t é o fator de transparência para I_1 . Se I_2 também é transparente, o algoritmo é aplicado recursivamente até encontrar uma superfície opaca ou o fundo da cena. Essa aproximação linear não é adequada para superfícies curvas, ou objetos complexos que espalham a luz, como nuvens etc.

Para modelar objetos transparentes, dois modelos são considerados:

Transparência sem Refração – este modelo não considera a influência da refração, ou seja, não há o aparente deslocamento dos objetos localizados atrás da superfície, devido à refração. Entre as formas desse modelo estão a *interpolada*, que considera a opacidade dos objetos, e a *filtrada*, que considera a cor transparente do objeto.

Transparência com Refração – este modelo, como o nome sugere, considera a refração, e os objetos localizados atrás do objeto atingido pela luz não se encontram na sua verdadeira posição, devido às diferenças entre os meios por onde a luz passa.

Efeitos de transparência podem ser emulados usando algoritmos que rasterizam superfícies em ordem de profundidade invertida. Ao rasterizar uma superfície transparente, faz-se uma combinação de cores, permitindo que partes da imagem anteriormente renderizada possam ser vistas através da nova superfície.

7.3.5. Sombreamento (Shading)

Em Computação Gráfica, a manipulação da luz assume um papel fundamental no aspecto realístico da apresentação. Os efeitos da luz sobre as superfícies e seus materiais, o obscurecimento de superfícies em função de sua posição, orientação e características da luz são, portanto, peças-chave.

Existem vários modelos de iluminação diferentes que expressam e controlam os fatores que determinam a cor de uma superfície em função de um determinado conjunto de luzes. Alguns modelos de iluminação controlam o efeito da luz para cada

pixel na imagem enquanto outros consideram um modelo de iluminação para alguns poucos pixels e aplicam interpoladores para o cálculo dos demais.

A Computação Gráfica refere-se frequentemente às regras da física ótica e à física das radiações para explicar a interação da luz nos objetos. No entanto, a complexidade ou a inexistência de modelos completos e abrangentes levam os programadores a simplificações do processo computacional. Em consequência, muitos modelos de iluminação atualmente em uso em computação gráfica contêm simplificações sem nenhum fundamento teórico, mas que alcançam um bom resultado prático. A primeira técnica de sombreado não constante para polígonos genéricos foi desenvolvido por Gouraud em 1971 (Watt e Policarpo). Em 1975, Phong Bui-Tuong melhorou a técnica de Gouraud e tornou-a mais adequada para os sistemas de computação gráfica 3D. Mesmo nas novas técnicas de iluminação global, como ray tracing e radiossidade, o modelo Phong ainda é o mais usado no render dos softwares atuais.

Existe aqui um ponto que geralmente causa uma confusão. Modelo de sombreado não é o mesmo que modelo de iluminação. O processo de se aplicar um modelo de iluminação para vários pontos de uma superfície é chamado sombreado.

Existem duas considerações separadas para o sombreado dos polígonos. Primeiro temos de considerar como calcular a luz refletida em qualquer ponto da superfície do objeto, chamamos isso de modelo local de reflexão. Depois, temos de calcular a intensidade da luz no pixel para o qual o polígono se projeta, chamamos isso de algoritmos de sombreado.

7.3.5.1. Modelo de Sombreamento Constante

No modelo de sombreado constante, aplica-se o cálculo da componente de luz refletida apenas uma vez por superfície plana da imagem, determinando-se um único valor de cor e intensidade da luz refletida que é utilizado para o preenchimento de toda a superfície. Essa técnica é chamada também de *flat shading*, *faceted shading* ou *constant shading* e é usada mais frequentemente em primitivas poligonais. Essa aproximação somente é aceitável se for possível supor que:

1. A fonte de luz localiza-se no infinito, fazendo com que o ângulo de incidência de cada um dos raios de luz que compõem o feixe de luz incidente possua o mesmo ângulo ao longo de toda a superfície plana (Figura 7.30 e 7.31);
2. O observador localiza-se no infinito, assim os raios de luz que compõem o feixe de luz refletida da superfície plana e que atingem o observador têm o mesmo ângulo (Figura 7.31);
3. As superfícies são representações de objetos realmente formados por faces planas e não apenas aproximados por faces planas (Figura 7.30).

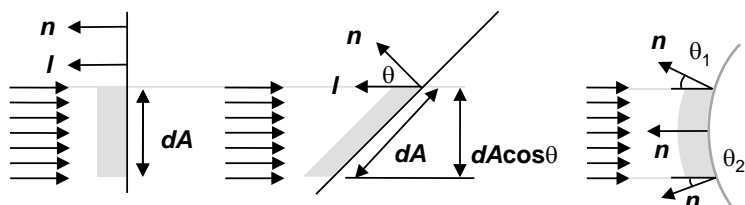


FIGURA 7.30. Feixe de raios incidentes paralelos nos diversos pontos da superfície iluminada, tem a mesma direção de reflexão apenas se essa superfície for plana.

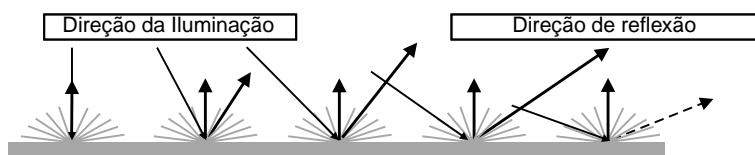


FIGURA 7.31. Se a fonte de luz e o observador não estiverem no infinito, os raios de luz refletidos não têm todos a mesma direção, mesmo nas superfícies planas.

Se essas suposições não forem razoavelmente satisfeitas, então o modelo de sombreamento constante não deve ser aplicado, pois gerará resultados pouco satisfatórios. Esse método de sombreamento é o mais simples e somente um cálculo de iluminação é aplicado para cada objeto em cena. A imagem gerada, por consequência, não aparenta um realismo satisfatório (Figura 7.32).



FIGURA 7.32. Modelo de Sombreamento Constante não apresenta um realismo satisfatório.

Uma solução para esse problema seria calcular a normal à superfície em cada ponto do polígono e aplicar o modelo de iluminação desejado naquele ponto. Assim, conseguiríamos sombrear todo polígono. Só que isso tem um custo maior computacionalmente. Então, para resolver esse problema de maneira mais eficiente, surgiram vários modelos de sombreamento interpolado para superfícies defi-

nidas por polígonos. Os mais conhecidos desses métodos recebem o nome de seus desenvolvedores técnicas de sombreados por interpolação de Gouraud e de Phong.

7.3.5.2. Sombreamento de Gouraud

Uma alternativa à técnica de sombreado constante foi proposta por [Henri. Gouraud, 71]. O método de Gouraud aplica a iluminação em um subconjunto de pontos da superfície e interpola a intensidade dos pontos restantes na superfície. Usualmente, os vértices de cada uma das faces poligonais da superfície plana que representa o objeto são utilizados para o cálculo da luz refletida aplicando-se, então, interpoladores lineares para o cálculo da luz nos demais pontos da superfície. No caso de uma malha triangular, normalmente o modelo de iluminação é aplicado em cada vértice do triângulo e os tons no interior do triângulo são linearmente interpolados destes valores de vértice.

Freqüentemente, as normais de vértice não estão disponíveis, nesse caso, devemos aproximar a normal dos vértices calculando a média das normais das faces adjacentes. Cada normal é um vetor n_i , ou seja, representável pelas suas coordenadas como (n_x, n_y, n_z) , essas normais podem ser normalizadas, isto é, modificadas, para cada uma delas ter comprimento unitário, se forem divididas pelos seus comprimentos:

$$\overline{n_i} = \frac{n_i}{|n_x| + |n_y| + |n_z|}$$

$$n_v = \sum_{i=1}^k \frac{\overline{n_i}}{k}$$

onde k é o número total de faces que formam o vértice, n_v será o valor da normal no vértice, e n_i cada uma das normais das faces, que chegam aos vértice i , devendo serem usadas normalizadas. A Figura 7.33 ilustra o cálculo da componente da luz refletida em diversos pontos de cada face plana de um poliedro 3D e os elementos dessa fórmula (nesta figura $k=4$).

Esse método é particularmente adequado a algoritmos de apresentação ou cálculo de visibilidade que utilizem *scan line* para o preenchimento de faces como o *z-Buffer*. Algoritmos de scan line são usualmente empregados na determinação de visibilidade por *z-Buffer* e a interpolação das componentes de luz refletidas pode ser feita durante a geração da imagem no buffer de imagem ou na própria tela. A Figura 7.34 ilustra a projeção de um poliedro 3D sendo criada no buffer de imagem, onde a intensidade da componente de luz refletida em Ia é resultado da interpolação linear das luzes refletidas de I1 e I2, e a intensidade em Ib é resultado da interpolação linear de I1 e I4. A intensidade ao longo da scan line Is pode ser calculada durante o cálculo do *z-Buffer* sendo o resultado da interpolação linear de Ia e Ib.

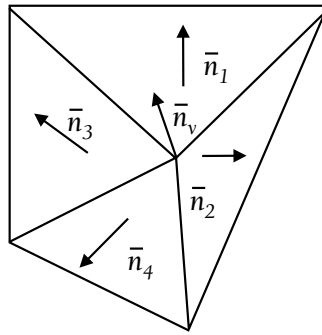


FIGURA 7.33. Cálculo da componente de luz refletida em diversos pontos de um poliedro. $\overline{n_y} = \frac{1}{4}(\overline{n_1} + \overline{n_2} + \overline{n_3} + \overline{n_4})$, [FOLEY, 93].

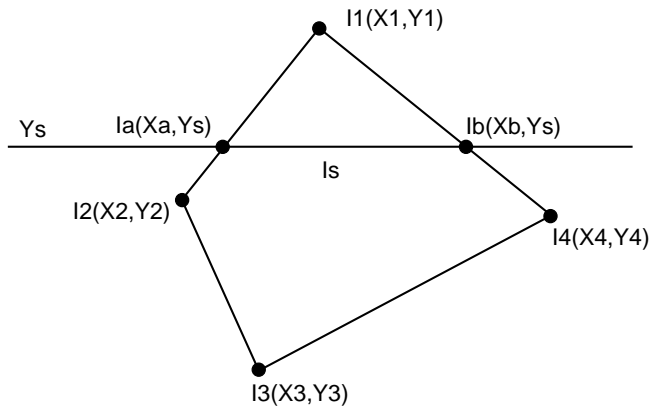


FIGURA 7.34. Cálculo da intensidade da luz refletida pela técnica de Gouraud durante o “scan line”. [XICHUN JENNIFER GUO, 96]

O sombreamento proposto por Gouraud é também conhecido *como intensity interpolation shading* ou *color interpolation shading*, pois interpola linearmente a intensidade da luz, eliminando a descontinuidade de cor ao longo dos polígonos normalmente utilizados para representar superfícies complexas. Essas descontinuidades de cor são naturalmente acentuadas pelo efeito de bandas de Mach (*Mach Banding*), descoberto por Ernst Mach, físico e filósofo austríaco (cujo trabalho em filosofia teve grande influência na relatividade restrita de Einstein), em 1865. O efeito de Mach representa uma característica da visão humana: o aumento na percepção da transição de tons causado por características de atenuação dos receptores de luz na retina. O efeito de Mach causa um aparente aumento da descontinuidade nas transições de intensidade de luz (Figura 7.35. Muito embora a técnica de Gouraud

atenue grandemente esse efeito, ela não o elimina completamente, pois o efeito também está associado à mudança da direção do degradê, como mostrado na Figura 7.36.



FIGURA 7.35. Efeito de Bandas de Mach com a variação da intensidade do tom. Cada um dos 10 trechos tem exatamente o mesmo tom, embora próximo a mudança da intensidade o lado mais claro pareça mais claro e o lado mais escuro pareça mais escuro, do que no restante do trecho de mesmo tom.



FIGURA 7.36. Efeito de Bandas de Mach com a variação da direção do degradê do tom. Cada um dos três trechos tem exatamente o mesmo nível de variação do tom, de claro para escuro e inversamente do escuro para o claro, embora próximo à mudança da região, a intensidade da variação parece maior que no centro do trecho.

Para obter maior eficiência no cálculo da luz refletida, é possível usar uma equação incremental como a apresentada para projeções paralelas com z-Buffer. Quando se utiliza a projeção em perspectiva ou quando a superfície representada pelo polígono plano não é no modelo real uma figura plana, esse modelo não pode ser aplicado com resultados satisfatórios. A solução de Gouraud também mostra-se pouco eficiente na apresentação das reflexões especulares quando essas ocorrem distantes dos pontos selecionados para o cálculo efetivo da componente de luz refletida.

7.3.5.3. O Modelo de Phong

Outra técnica bastante difundida para o cálculo da componente de luz refletida foi proposta por Bui-Tuong Phong (1942-1975). [PHONG, 75], propôs a interpolação linear dos vetores normais para o cálculo do sombreado, com o posterior cálculo da iluminação. Ao contrário do modelo de Gouraud, que interpola as intensidades da luz refletida, o modelo de Phong interpola a variação do ângulo de incidência do feixe de luz na superfície, possibilitando a determinação de pontos de reflexão especular afastados das extremidades dos polígonos. O Modelo de Phong conside-

ra, desse modo, que a normal da superfície é interpolada linearmente ao longo dos pontos das fases. Phong assume uma normal para cada vértice. O modelo de iluminação é aplicado em todos os pontos da superfície que estiver sendo sombreada, sendo a normal em cada ponto o resultado da interpolação linear das normais dos vértices. A Figura 7.37 ilustra o uso das normais em \bar{n}_0 e \bar{n}_1 , para fazer a interpolação linear dos pontos interiores P_a , P_b , e P_c .

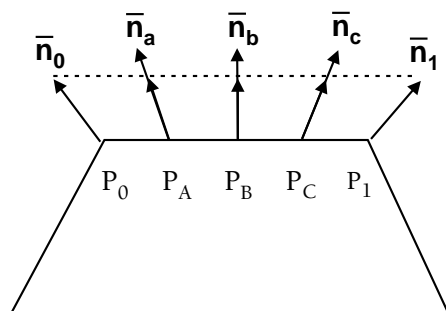


FIGURA 7.37. Interpolação dos ângulos de incidência dos pontos do interior através da consideração das normais das extremidades em P_0 e P_1 em uma superfície plana [FOLEY, 93].

O método proposto por Phong, como o método proposto por Gouraud, adequa-se muito bem aos algoritmos de scan line usados geralmente para determinação da visibilidade por z-Buffer. A interpolação do ângulo de incidência da luz refletida pode ser feita durante a geração da imagem no buffer ou na própria tela. A Figura 7.38 ilustra a projeção de um poliedro 3D sendo criada no buffer de imagem, onde o ângulo de incidência da luz em n_a é resultado da interpolação linear de n_1 e n_2 , e o ângulo em n_b é resultado da interpolação linear de n_1 e n_4 . A intensidade ao longo da scan line n_s pode ser calculada durante o cálculo do z-Buffer sendo o resultado da interpolação linear de n_a e n_b .

Para obter maior eficiência no cálculo do ângulo de incidência do feixe de luz, é possível usar uma equação incremental como a apresentada para projeções paralelas com z-Buffer. Quando se utiliza a projeção em perspectiva, esse modelo pode ser aplicado com resultados satisfatórios. Uma grande vantagem da solução de interpolação de Phong está na real redução do efeito de bandas de Mach, já que não usa as intensidades de luz. Os resultados obtidos são geralmente bem melhores que o de Gouraud devido à sua capacidade de apresentar com realismo pontos de grande variação da intensidade luminosa, como os que ocorrem com o uso de luzes pontuais intensas (*spotlights* e *highlights*) na cena ou nos casos de reflexão especular, mesmo quando afastados dos pontos de amostra utilizados na interpolação.

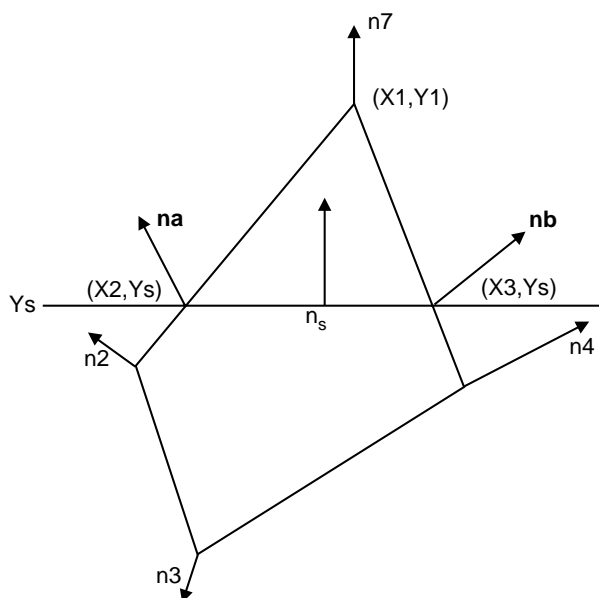


FIGURA 7.37. Geometria do cálculo do ângulo da luz refletida pela técnica de Phong durante o processo de “scan line”. Xichun Jennifer Guo, 1996.

Esse modelo de interpolação é também conhecido como *normal-vector interpolation shading*. E não deve ser confundido com o modelo de luz especular do mesmo autor. No modelo de luz de Phong, descrito anteriormente neste capítulo, a luz é composta das três componentes: reflexão difusa (*diffuse*), reflexão especular (*specular*), e ambiente (*ambient*). Essas três componentes são combinadas de modo a determinar a iluminação ou cor de um ponto ou de um polígono. O algoritmo de Phong para interpolação, apesar de menos implementado em hardware do que o modelo de Gouraud devido à sua maior complexidade, permite a obtenção de imagens mais realistas pois emula melhor os efeitos locais da luz (como as reflexões que podemos observar em superfícies brilhantes).

7.3.6. Sombras

As sombras são de fundamental importância para aumentar o realismo das imagens. Elas estabelecem diversos níveis de realismo em uma imagem, evitando que os objetos pareçam estar flutuando no ar. A análise realística das sombras se caracteriza pela presença de duas regiões. Uma região de sombra (ou umbra em Latim), onde a intensidade luminosa devido a uma certa fonte é nula (Figuras 7.17 a 7.19, e uma região de penumbra, onde a intensidade luminosa varia de zero até a intensidade de luz do ambiente (Figura 7.16). Se no ambiente só houver uma única luz pontual, a região de penumbra pode ser anulada. A seguir daremos dois exemplos de algoritmos de geração de sombra. São eles: o de *Volume de Sombra* e o de *Sombra Projetada*.

7.3.6.1. Volume de Sombra

É um algoritmo geométrico que foi introduzido por [FRANK CROW, 77]. Chamamos de volume de sombra de um objeto em relação a uma fonte de luz (L) pontual, o conjunto dos pontos do espaço da imagem que não são visíveis por um observador se ele estiver colocado na posição da fonte, isso é, em L. Ele é obtido pela criação de uma pirâmide de não iluminação ou sombra.

Essa pirâmide tem ápice na fonte de luz e nos lados formados pelos prolongamentos dos raios luminosos a partir da fonte até os contornos do objeto cujo volume da sombra se está definindo. Nesse algoritmo, ao fazer o cálculo da intensidade luminosa em um ponto, é verificado se o ponto está dentro do volume de sombra de cada objeto, de modo a fazer a atenuação necessária no cálculo da intensidade.

7.3.6.2. Sombra Projetada

É também um algoritmo geométrico. Só que é mais amplamente difundido entre os usuários de sistemas gráficos. Ele consiste em modelar a região de sombra a partir da projeção do objeto cuja sombra se deseja definir. Nessa projeção, o ponto de vista assume o lugar da fonte de luz e o plano de projeção é considerado como superfície plana onde se deseja conhecer a sombra do objeto. Esse método funciona bem, e é eficiente no caso em que a sombra se projeta em uma única superfície plana (piso, parede, face plana de outro objeto), mas se torna complexo à medida que o número de superfícies planas onde a sombra se estende aumenta.

7.3.6.3. Algoritmo de Atherton-Weiler-Greeberg

Este algoritmo cria um modelo constituído por recortes dos objetos do cenário que não são visíveis da fonte de luz. Ao rasterizar e tonalizar um polígono, leva-se em consideração se ele produz sombra ou não. A principal restrição desse algoritmo é que só se aplica em cenas constituídas por objetos poligonais [ATHERTON-WEILER-GREEBERG, 78].

7.3.6.4. Algoritmo de Mapeamento

Proposto por Lance William em 1978, o algoritmo de mapeamento trata as sombras como um problema de visibilidade a partir das fontes de luz. Essa visibilidade é resolvida usando um método semelhante ao Z-Buffer. Ao calcular a iluminação da cena, os dados de Z-Buffer são usados para determinar as sombras.

7.3.6.5. Algoritmo de Appel

Proposto em 1968 por Appel e posteriormente por Bouknight e Kelley em 1970, este algoritmo traça um raio de sombra para cada fonte de luz durante o processo de

rasterização, testando se há uma obstrução. Em caso positivo, é determinada a parte da linha de varredura afetada pela sombra e os relacionamentos entre polígonos que causam sombra um nos outros. A implementação desse algoritmo só é utilizável para modelos poligonais.

7.3.6.6. Sombreamento Anisotrópico

Os prefixos gregos *is(o)* e *anis(o)* querem dizer igual e desigual, respectivamente. Diz-se que um objeto, um material ou um modelo é *isotrópico* quando tem as mesmas propriedades ou características físicas em todas as direções. Esse conceito já foi considerado na modelagem de sólidos, onde geralmente supomos que os objetos a serem modelados têm as mesmas características em todas as direções. Assim, algo é anisotrópico se suas propriedades dependem da direção de observação.

O sombreamento anisotrópico é um tipo de shading que prolonga reflexões e brilhos em uma dada direção, geralmente na direção perpendicular às fibras ou ranhuras de uma superfície (Figura 7.39), e não é uniforme em todas as direções como os sombreamentos isotrópicos usuais, isto é, todos os comentados até aqui. Como há muito mais curvaturas ao redor dos fios que ao longo dos fios, a luz é refletida pelos cabelos anisotropicamente, produzindo um brilho na direção perpendicular ao comprimento dos fios. Aço escovado, discos de vinil, CDs e madeiras também possuem essa característica, devendo ser renderizados com *shading anisotrópico*, isso é com mais brilhos na direção perpendicular às ranhuras. Esse recurso está implementado nas melhores placas gráficas do mercado e por isso pode ser usado em real-time rendering.



FIGURA 7.39. Exemplos de sombreamento anisotrópico.

7.3.7. Modelos de Iluminação Global

Cada uma das técnicas já abordadas para a geração de imagens realistas em computador possui o seu conjunto de vantagens e desvantagens. No entanto, entre as téc-

nicas abordadas até aqui, os modelos de iluminação global oferecem melhores resultados na simulação de uma cena real. Podemos chamar de iluminação global, as técnicas que consideram todos os feixes de luz emitidos por fontes indiretas. O céu, por exemplo, reflete a luz do sol com uma cor azulada, esses raios de luz azulados também contribuirão para modificar o restante da cena. Essa consideração não pode ser feita nos modelos de iluminação anteriores, que por considerarem todos os objetos da cena como possíveis emissores de luz, precisam executar muito mais cálculos que os anteriores.

Com a melhora da velocidade de processamento das máquinas, diminuí o tempo de cálculo das técnicas de iluminação global, que estão sendo cada dia mais usadas e, por isso, de domínio obrigatório para os profissionais de CG. Essas técnicas não representam o fim das demais, ficando a cargo dos artistas julgar qual o melhor método de iluminação para determinado tipo de trabalho. Atualmente, temos no mercado uma diversidade enorme de softwares habilitados a trabalhar com iluminação global. Dentre eles podemos citar o *Brazil*, *Mental Ray*, *Vray* ou *Final Render*. Nas seções que seguem descrevemos as principais idéias e os métodos que as usam.

7.3.8. Ray Tracing

A técnica de *Ray tracing* é também usada simplesmente como um algoritmo de determinação da visibilidade dos elementos de cena. Nesse caso, é geralmente denominada de *Ray-casting*. É principalmente conhecida pelas possibilidades de inclusão na cena de sombras, reflexão, refração, texturas e não apenas o sombreado das superfícies visíveis.

Os primeiros estudos sobre *Ray tracing* são da década de 1960. O *Ray tracing* foi desenvolvido em 1968 como um algoritmo para simulação de trajetórias de projéteis balísticos e partículas nucleares. A Apple foi a primeira a apresentá-lo como uma ferramenta para o cálculo de sombras em Computação Gráfica. Na época, os computadores eram lentos demais para possibilitar o uso dessa técnica, e as aproximações descritas anteriormente eram mais usadas.

À medida que os computadores foram ficando mais poderosos, notou-se que seria interessante voltar atrás e implementar os modelos de iluminação global. A partir daí, o algoritmo de *Ray Tracing* foi estendido e implementado. Sua implementação inicial ocorreu em 1980 quando já era possível criar imagens com sombras, reflexões, transparência e refrações. Em 1984, o algoritmo de *Ray Tracing* sofreu modificações possibilitando efeitos de penumbra, *motion blur*, *depth of field* (veja seção hiper-realismo) entre outros.

Hoje em dia, *Ray tracing* é uma das mais populares e poderosas técnicas de síntese de imagens de fácil implementação. *Ray tracing* possibilita a representação de cenas complexas com muitos objetos e muitos efeitos (como sombras e vários tipos de reflexões). O princípio do *Ray tracing* é simular a geometria ótica envolvida no trajeto de alguns raios de luz que viajam pelo espaço da cena. Por motivos

computacionais, o modelo utilizado é o contrário do que realmente acontece quando vemos uma cena. Isto é, normalmente um raio de luz originário no objeto chega aos nossos olhos. Nessa técnica, supõe-se que um raio originário de nossos olhos chegue até o objeto que se quer renderizar (Figura 7.40). Pelas leis da ótica, essa reversão não provoca alteração alguma na geometria envolvida. Devido à essa inversão ser tão comum, raramente se usa o adjetivo reverso para *Ray tracing*. Se desejarmos, podemos classificar o *Ray tracing* como reverso (usado na computação) ou direto (o algoritmo que segue o modelo físico real).

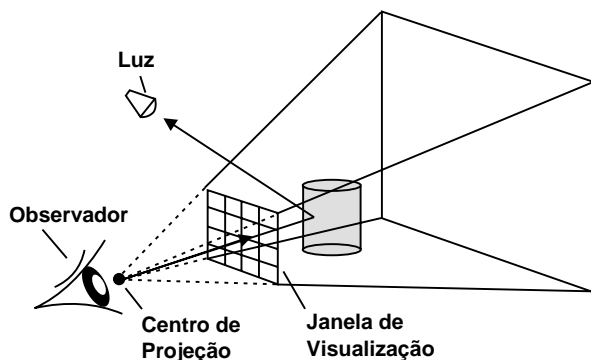


FIGURA 7.40. Geometria do “ray tracing”: repare o ângulo de visão da cena pelo observador, a região de influência de cada pixel, simbolizada pela grade retangular, o ponto da cena real que corresponde a esse pixel e a fonte de luz que originou o raio.

Na realidade, uma ou mais fontes de luz iluminam uma cena com um número infinito de raios de luz. Apenas uma pequena parte desses raios é refletida dos objetos da cena e atinge o olho do observador. Como apenas esses raios nos interessam, podemos traçá-los de volta para o objeto e deste para cada fonte de luz. Inserindo uma tela em algum ponto entre o observador e a cena, traça-se um raio do olho do observador até cada pixel da tela e deste para qualquer objeto que a fonte de luz ilumine. O resultado final será que as cores e intensidades que serão associadas à tela produzirão uma imagem como uma fotografia.

O algoritmo de Ray tracing considera os seguintes pontos:

- Os raios são disparados de forma sistemática, de modo que cada um deles corresponda a um pixel na tela.
- Após o disparo, o raio percorre o espaço podendo atingir um objeto ou sair da cena.
- Se atingir algum objeto, o ponto de intersecção é calculado. As contribuições das fontes de luz para cada ponto, levando em conta a sombra de outros objetos, também são calculadas.

- Se o objeto for opaco, a soma dessas contribuições será a intensidade luminosa total naquele ponto.
- Caso contrário, as contribuições devidas aos reflexos e refrações, serão também computadas. O pixel correspondente pode, então, ser exibido.
- Se não houver interseção, o pixel terá a cor de fundo.

O cálculo do ray tracing clássico é bastante simples e é constituído das seguintes tarefas efetuadas para cada pixel da tela:

1. Trace um “raio” a partir do observador até a cena a ser representada através de um pixel da tela;
2. Determine qual o primeiro objeto a interceptar esse raio;
3. Calcule a cor ambiente da superfície do objeto no ponto de interseção baseado nas características do objeto e na luz ambiente;
4. Se a superfície do objeto for reflexiva, calcule um novo raio a partir do ponto de interseção e na “direção de reflexão”;
5. Se a superfície do objeto for transparente, calcule um novo raio a partir do ponto de interseção.
6. Considere a cor de todos os objetos interceptados pelo raio até sair da cena ou atingir uma fonte de luz, e use esse valor para determinar a cor do pixel e se há sombras.

O algoritmo de Ray Tracing, então, pode ser descrito como:

Considerando um centro de projeção, no plano de visão

Para (cada linha horizontal de varredura da imagem – *scan line*)

```

{ Para (cada pixel da linha de varredura)
    { determinar raio que vai do centro de projeção ao pixel
      Para (cada objeto da cena)
        { Se (objeto for interceptado pelo raio &&
           é a interseção mais próxima até agora)
          registrar interseção e o objeto interceptado
        }
      atribuir ao pixel a cor do objeto da interseção mais próxima
    }
  }

```

O objetivo da técnica de ray tracing é simular a propagação da luz no ambiente avaliando a sua interação com os objetos que o compõem e considerando a interação da luz com as suas superfícies. Com o objetivo de limitar a grande quantidade de feixes de luz necessária para compor uma cena com razoável grau de realismo, o ray tracing tira proveito da característica física dos controladores gráficos de com-

putador e da maneira como os pixels da tela, quando colocados lado a lado, são capazes de compor uma cena com elevado grau de realismo. Assim sendo, a técnica de ray tracing concentra o seu esforço computacional no estudo dos raios de luz que efetivamente alcançam o observador através da “malha” de pixels da tela.

O algoritmo de ray tracing não leva em consideração apenas a cor intrínseca de cada objeto, mas também os efeitos de reflexão e refração da luz provocados por esse objeto. A geração ou determinação de imagens refletidas é feita pelo somatório das componentes de cor obtidas por cada interação do raio com um objeto ou fonte de luz. Para cada pixel da tela, é traçado um raio de “luz” a partir do observador, e é estudada a sua interação com os objetos em cena com o objetivo de determinar sua componente RGB. A Figura 7.40 ilustra o trajeto considerado por essa técnica onde o raio de luz incidente no observador, através da consideração de um pixel da tela como uma “janela”, é traçado de volta à fonte percorrendo o caminho inverso, em termos de reflexões, até a fonte de luz que o originou.

O ray tracing deve considerar os raios refletidos toda vez que o coeficiente de reflexão de uma superfície for diferente de zero. O coeficiente de reflexão varia entre 0 e 1, determinando que quantidade de energia do raio de luz deve ser considerada como absorvida pelo objeto em questão, compondo uma soma ponderada das componentes de cor para o pixel na tela. Um espelho possui um coeficiente de reflexão próximo de 1, ou seja, nessa superfície todos os raios incidentes devem ser refletidos com o mesmo ângulo de incidência em relação à direção da reta normal à superfície. Além do coeficiente de reflexão, as superfícies também apresentam um coeficiente de refração que expressa a maneira pela qual a luz passa através de um meio para outro.

Basicamente, o que o algoritmo de ray tracing faz é calcular a interseção de uma semi-reta com todos os objetos da cena. A esta semi-reta damos o nome de raio. Como mencionamos anteriormente, só será necessário o cálculo dos raios que atingem o observador. Para isso, teremos de calcular os pontos de interseção do raio com os objetos da cena.

7.3.8.1. Cálculo de Interseções e Normais

A tarefa principal do ray tracing consiste no cálculo da interseção de um raio com o objeto. Para essa tarefa, utiliza-se normalmente a representação paramétrica de um vetor ou reta. Cada ponto (x, y, z) ao longo de um raio com origem no ponto (x_0, y_0, z_0) e direção do ponto (x_0, y_0, z_0) para o ponto (x_1, y_1, z_1) é definido em função do parâmetro t , (com valores no intervalo $[0,1]$ pelas equações paramétricas da reta):

$$x = x_0 + t(x_1 - x_0);$$

$$y = y_0 + t(y_1 - y_0);$$

$$z = z_0 + t(z_1 - z_0);$$

$$x = x_0 + t\Delta x; \Delta x = x_1 - x_0$$

$$y = y_0 + t\Delta y; \Delta y = y_1 - y_0$$

$$z = z_0 + t\Delta z; \Delta z = z_1 - z_0$$

Se (x_0, y_0, z_0) for considerado o centro de projeção, ou o olho do observador, na Figura 7.40, e (x_1, y_1, z_1) for o centro de um pixel na “janela” da mesma Figura 7.40, então t varia de 0 a 1 entre esses pontos. Valores negativos de t representam pontos atrás do centro de projeção, enquanto valores de t maiores que 1 correspondem a pontos depois da janela, isso é mais distante do centro de projeção. Assim precisamos achar uma representação para cada tipo de objeto que nos possibilite determinar t , isto é, o valor do parâmetro que define o ponto de interseção do objeto real com o raio.

Necessita-se, assim, de uma representação que permita determinar a interseção de um raio com os objetos da cena, em função de t , de forma eficiente. A forma das superfícies do objeto é que determinará a maneira como o cálculo das interseções pode ser otimizado. Uma das formas de superfície dos objetos mais simples para esse propósito é a esfera, talvez por isso seja tão usada como exemplo de cenas renderizadas por essa técnica. A superfície da esfera é definida pela posição das coordenadas do seu centro (x_c, y_c, z_c) e pelo comprimento de seu raio r , podendo ser representada pelo conjunto de pontos cujas coordenadas x, y e z satisfazem a expressão:

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = r^2;$$

ou

$$x^2 - 2 x_c x + x_c^2 + y^2 - 2 y_c y + y_c^2 + z^2 - 2 z_c z + z_c^2 = r^2$$

Aplicando-se a equação paramétrica da reta (anterior) em substituição a x, y e z temos:

$$\begin{aligned} (x_0 + t(x_1 - x_0))^2 - 2 x_c(x_0 + t(x_1 - x_0)) + x_c^2 + \\ (y_0 + t(y_1 - y_0))^2 - 2 y_c(y_0 + t(y_1 - y_0)) + y_c^2 + \\ (z_0 + t(z_1 - z_0))^2 - 2 z_c(z_0 + t(z_1 - z_0)) + z_c^2 = r^2 \end{aligned}$$

Essa equação de segundo grau (como qualquer outra deste grau) pode ter um, dois ou nenhum valor numérico pertencente ao conjunto dos números reais que a solucione, o que é chamado de *raiz da equação*. Caso não possua raízes, então o raio não cruza a esfera; caso possua apenas uma raiz, então o raio tangencia a esfera no ponto definido pela raiz (t); caso possua duas raízes, então o raio cruza a esfera e a raiz de menor valor determina o ponto de “entrada” se estiver originalmente fora na esfera e a raiz de maior valor determina o ponto de saída.

Outro elemento muito necessário na renderização é a normal à superfície do objeto em um certo ponto. Determinar essa normal é também muito simples para as superfícies esféricas. Para uma esfera de raio r e centro (x_c, y_c, z_c) , a normal em um ponto da esfera de coordenadas (x, y, z) é descrita pelo vetor:

$$1/r (x - x_c, y - y_c, z - z_c)$$

A interseção com polígonos, isto é, objetos com faces poligonais, também pode ser facilmente obtida, embora alguns cálculos adicionais sejam necessários. A equação genérica do plano que contém um polígono pode ser definida pela equação geral do plano, como:

$$Ax + By + Cz + D = 0;$$

Aplicando-se a equação paramétrica da reta (anterior) em substituição a x , y e z , temos:

$$A(x_0 + t(x_1 - x_0)) + B(y_0 + t(y_1 - y_0)) + C(z_0 + t(z_1 - z_0)) + D = 0;$$

De modo que, resolvendo a equação tem-se:

$$t = (A x_0 + B y_0 + C z_0 + D) / (A (x_1 - x_0) + B(y_1 - y_0) + C(z_1 - z_0));$$

Essa é uma expressão do primeiro grau e matematicamente tem uma ou nenhuma solução. Se o denominador (parte de baixo) da divisão for igual a zero, então a equação não tem solução, isso fisicamente significa que o raio de luz é paralelo ao plano.

Determinar a normal à superfície do objeto plano, para a renderização, é também direto, pois em qualquer ponto do plano ela será definida simplesmente pelo vetor (A,B,C) . Resta saber se o ponto de interseção do raio com o plano, que contém o polígono, está contido no polígono. Isso pode ser facilmente obtido em duas dimensões, ou seja, utiliza-se a projeção ortogonal do polígono em um dos plano de projeção, como mostra a Figura 7.41. Por razões de precisão, recomenda-se a utilização do plano no qual a projeção ortogonal do polígono seja maior.

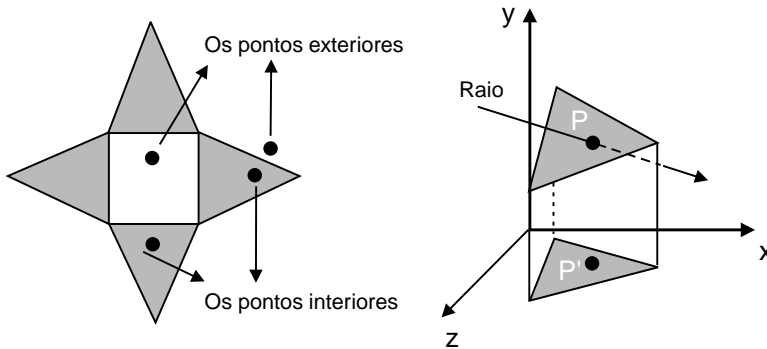


FIGURA 7.41. Técnicas de determinação da posição de um ponto em relação a um polígono qualquer e ponto de interseção de um raio com um plano.

Considerando que polígonos podem ser não-convexos, precisa-se de uma regra para especificar quais regiões são interiores, e quais são exteriores. Para determinar se um ponto está contido ou não em um polígono, traça-se um raio a partir do ponto em uma direção qualquer, mas que não passe por nenhum dos vértices. Se esse raio cruzar a borda do polígono um número ímpar de vezes, então ele está no interior do polígono. Essa técnica pode usar a *scan line* já discutida no z-Buffer. A Figura 7.41 ilustra as técnicas de determinação da posição de um ponto com relação a um polígono qualquer; que são fundamentais para um algoritmo de *ray tracing* eficiente.

Além de ser um algoritmo bastante simples e elegante para geração de imagens fotorrealísticas, o algoritmo de ray tracing é utilizado em várias outras aplicações, entre elas:

- Cálculo de volume, massa, centróide, centro de massa, área de superfície, momentos e produtos de inércia de sólidos em aplicações CAD;
- Auxílio no projeto das indústrias óticas e de lentes;
- Cálculo dos fatores de forma em algoritmos de radiação e transferência de calor;
- Conversões entre formas de armazenamento de sólidos;
- Análise de antenas, sonares e radares;
- Modelagem molecular, sismologia, processamento de imagens e sinais;
- Base para “benchmark” de computadores.

Todo processo de síntese de imagens resume-se em determinar a cor de cada pixel de uma imagem. O algoritmo de ray tracing geralmente faz essa determinação através do modelo de iluminação proposto por Whitted [Whitted, 1980] que engloba todos os efeitos da luz, em uma única expressão, fazendo um somatório de efeitos quando mais de uma fonte de luz é considerada. Esse modelo é representado por esta equação:

$$I = I_a \cdot r_a + \sum_j f_{atj} I_d (r_d (u_{ej} \cdot u_n) + r_s (u_n \cdot u_{hj})^n + I_t r_t + I_r r_r)$$

Onde:

- I = intensidade do ponto de interseção no plano de imagem nos canais (RGB);
 I_a = intensidade da iluminação ambiente global;
 r_a = coeficiente de reflexão da iluminação ambiente do material;
 f_{atj} = coeficiente de atenuação da j -ésima fonte de luz;
 I_d = representa a intensidade da fonte de luz direcional presente;
 r_d = coeficiente de reflexão difusa do material;
 u_{ej} = vetor na direção da j -ésima fonte de luz;

- u_n = vetor unitário normal à superfície do objeto;
- J = número das fontes de luz que iluminam o ponto consideradas no somatório;
- r_t = coeficiente de transmissão do material;
- I_t = intensidade do raio transmitido por refração;
- r_r = coeficiente de reflexão do material;
- I_r = intensidade do raio refletido;
- r_s = coeficiente de reflexão especular do material;
- $u_{h,j}$ = vetor unitário na direção de máxima intensidade de reflexão, H da j -ésima fonte de luz;
- n = expoente que representa o polimento da superfície;

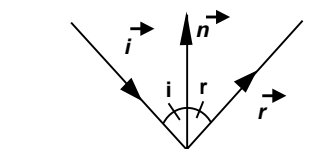
A cor de cada pixel é determinada individualmente através do cálculo da luz (I) que chega até o ponto de observação. Essa luz, segundo o modelo de Whitted, tem duas componentes: local e global.

A componente local é constituída pela parcela de iluminação que, partindo das fontes de luz do cenário, atinge diretamente uma superfície visível ao observador. É composta por duas parcelas, difusa e especular, representadas, respectivamente, pelo segundo e terceiro termos da equação anterior.

A componente global é representada pelos demais termos da equação e é o grande diferencial do ray tracing. Essa componente é responsável pelos efeitos de reflexão e refração exibidos na imagem. O cálculo da componente global é obtido através da luz que chega ao ponto de interseção do raio pelas direções de reflexão (I_r) e refração (I_t).

O cálculo do parâmetro I_r é feito através do lançamento de um raio na direção de reflexão dada pela equação:

$$-r = i - 2(n \cdot i)i$$



Essa equação está baseada nas propriedades de que os raios incidentes e refletidos, formam com a normal da superfície do objeto ângulos de incidência e reflexão iguais. Considera ainda que os vetores normais n , e de incidência, i , tenham comprimento unitário.

Fisicamente, existem infinitos raios de luz, partindo de cada fonte de luz em uma cena. Simular todos os raios seria uma tarefa impossível para um PC. Por outro lado, apenas os raios que são refletidos por objetos contribuem para a formação da imagem vista pelo observador. Logo, somente estes deverão ser simulados.

Determinados todos os pontos de interseção do raio com os objetos da cena, podemos determinar qual o objeto mais próximo que será usado no cálculo da normal à superfície deste, para uso na equação de iluminação (descrita anteriormente). Essa equação contém somatórias que consideram apenas as fontes que iluminam o ponto de interseção e não provocam sombras. Precisamos, então, determinar quais fontes iluminam o ponto interceptado. Essa determinação é feita lançando-se novos raios (raios de sombra) que partam do ponto de interseção em direção às fontes de luz. A interseção de qualquer objeto por um raio de sombra indica que a fonte de luz provoca sombra no ponto considerado.

Nesta etapa, o algoritmo já pode calcular a componente local da luz que determina a cor de um dado pixel.

Podemos observar que o algoritmo de ray tracing é bastante simples pois seu núcleo é constituído por uma rotina recursiva de lançamento de raios, que forma a chamada “árvore de raios”.

7.3.7.1.2. Ray Tracing Parametrizado

A técnica de ray tracing parametrizado foi introduzida formalmente em 1989 por Séquim e Smyryl. A idéia desse algoritmo é armazenar as árvores de raios de cada pixel e fazer uma reavaliação da equação correspondente à cor de cada pixel para cada mudança nos parâmetros ópticos obtendo-se uma nova imagem ajustada.

7.3.7.1.3. Monte Carlo Forward Ray Tracing

O algoritmo de ray tracing Monte Carlo, ao contrário de todos os outros algoritmos, é baseado no método direto (forward). Isso significa que os raios partem da fonte de luz.

Em poucas palavras, o método pode ser descrito assim. Inicialmente, escolhemos randomicamente uma fonte de luz, usando maior probabilidade para as fontes de maior intensidade. No caso de fontes tubulares, como lâmpadas fluorescentes (onde a luz parte de todo o seu extenso cilindro), um ponto em sua superfície é escolhido randomicamente. Uma direção de raio é então escolhida randomicamente de acordo como a distribuição espacial da intensidade da fonte de luz.

O raio é então seguido até a interseção com algum elemento da cena (ou até ele deixar o espaço da cena). Quando ocorre a interseção, devemos verificar o tipo de evento provocado, ou seja, se foi uma luz especular, uma reflexão difusa, uma refração, se a luz foi absorvida etc. A probabilidade de cada evento pode ser relacionada pelos coeficientes de reflexão, transmissão ou absorção da superfície. Se a absorção total for escolhida, o raio não precisa mais ser seguido. Para os outros casos, uma nova direção é escolhida deterministicamente, no caso da luz refletida especular, ou estocasticamente, no caso de dispersão difusa.

Nesse algoritmo, a energia do raio não muda durante o processo de rastreamento do raio (ray tracing). Cada evento que poderia mudar as propriedades da energia, como uma absorção parcial, passa a ser tratado na forma probabilística. Durante o processo, se um raio intercepta um triângulo (no caso dos objetos serem formados por conjuntos de triângulos) ou a malha da superfície, os elementos do mapeamento de iluminação (light map) são modificados para somar a energia trazida pelo raio.

O algoritmo de Monte Carlo trata as interseções dos raios com os pontos da malha da superfície, necessitando somente das propriedades da superfície para aquele ponto. A radiosidade trata a interseção dos raios considerando cada elemento da malha como um todo. Como resultado, a radiosidade fica limitada no cálculo das variações das propriedades da superfície dentro dos elementos da malha. Na prática, observamos que os algoritmos determinísticos podem produzir erros na textura quando usados na reflexão de superfícies curvas especulares.

7.3.7.2. Trabalhando com Ray Tracing

Com o ray tracing, é possível fazer duas reproduções com excelentes resultados: reflexos e refrações. Quando você estiver recriando a aparência de metal ou vidro a partir de um material padrão, quase nenhuma outra técnica superará o ray tracing.

Os reflexos gerados pelo ray tracing são mais precisos que os criados usando o mapa de reflexão. O ray tracing permite também gerar reflexos em superfícies onde os mapeamentos falham. Também é possível gerar diversos reflexos na superfície de um objeto. Toda essa capacidade tem um preço alto e poderá demorar mais do que você imaginaria.

7.3.7.2.1. Reflexos

Quando você usar o ray tracing para capturar as qualidades reflexivas de uma superfície, geralmente será mais importante considerar os reflexos da superfície do que sua rugosidade. Isso acontecerá ao criar materiais metálicos.

7.3.7.2.2. Refrações

A segunda capacidade real do ray tracing é a habilidade de copiar a aparência de materiais transparentes. Quando a luz passa por uma superfície transparente, ela geralmente é distorcida. Essa distorção é conhecida como refração e a quantidade de refração é proporcional ao índice de refração (IR). O IR resulta variação da velocidade relativa da luz quando ela muda de meio. Em geral, quanto mais denso for o objeto, mais alto será o IR.

7.3.7.2.3. Metais com Ray tracing

Uma das razões básicas para querer usar o ray tracing é produzir melhor reflexos em metal e em outros materiais reflexivos. A maioria dos materiais reflexivos tem algum grau de rugosidade. Pequenas imperfeições ou sujeira na superfície de um objeto mancharão o reflexo. Marcas de dedos oleosos no vidro ou no alumínio polido de uma roda de automóvel são bons exemplos desse efeito. Manchando levemente o reflexo em um material com ray tracing você poderá acrescentar muito realismo à superfície.

7.3.7.2.4. Ray Tracing em Real-Time Rendering

Apesar da grande redução no tempo de processamento proporcionada pelas técnicas de aceleração de ray tracing, processamento paralelo ainda é necessário para atingir taxas próximas às necessárias ao real-time render. Na verdade, não existe nenhum computador, pelo menos no mercado, capaz de trabalhar com os algoritmos em RT² (Real Time Ray Tracing). Essa limitação será uma questão de tempo afinal, o algoritmo de ray tracing é inerentemente paralelo, o que torna essa aplicação especialmente interessante para implementação em máquinas de mais de um processador. Alguns algoritmos interessantes surgiram no início da década de 1990 como o ray tracing incremental proposto por Murakami e Hirota, o algoritmo de Monte Carlo e as estruturas hierárquicas para manipulação interativa de Brière e Polin.

Por enquanto, quase todas as engrenagens 3D utilizam os *light maps* (mapa de luz), uma solução em que, após o desenvolvimento da fase e posicionamento de luzes fixas da cena, um programa é executado uma única vez para calcular o ray trace a partir de um ponto fixo e armazenar a informação no mapa de luz.

7.3.7.2.5. Caustic

Caustic é uma adição aos métodos de iluminação ray trace para rendering 3D, que aumenta o realismo de uma imagem para perto do fotorrealismo ou hiper-realismo, encontrando diversas aplicações para iluminação de animações, games ou imagens estáticas. Outras aplicações de cáustica referem-se à engenharia onde são utilizadas na análise de superfícies e testes de resistência de material.

Basicamente, podemos entender esse método observando a Figura 7.42. Quando um feixe de luz atinge uma superfície plana, ele é refletido com o mesmo ângulo. Quando a superfície não é plana, os feixes são refletidos para diferentes ângulos, podendo convergir para um mesmo ponto. Essa concentração de raios em um determinado ponto cria um efeito luminoso chamado caustic (Figura 7.43). Mas e se a superfície for transparente?

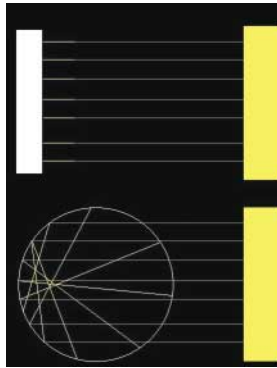


FIGURA 7.42. Quando a superfície não é plana, os feixes são refletidos para diferentes ângulos, podendo convergir para um mesmo ponto.

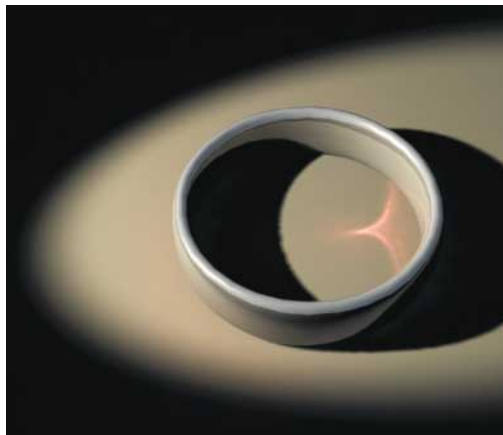


FIGURA 7.43. Concentração de raios em um determinado ponto criando o efeito luminoso chamado *Caustic*.

Podemos então dividir o caustic em duas formas: o catacaustic e o diacaustic. O catacaustic é o efeito provocado pela *reflexão* dos raios de luz na superfície. O diacaustic é o efeito provocado pela *refração* dos raios de luz, ou seja, quando o raio atravessa a superfície. Esses fenômenos são muito comuns na natureza e podem ser observados quando expomos copos de vidro ou águas à luz do sol. Esse mesmo fenômeno pode ser observado em piscinas ou riachos.

Os primeiros estudos da superfície cáustica iniciaram-se por volta de 1678 por Huygens e Tschirnhaus. Eles observaram que, quando os raios da luz do sol entravam em um copo de água, ocorria um fenômeno de iluminação intenso dentro do copo e uma sombra luminosa na superfície da mesa. Essa região de concentração dos raios dentro do copo recebeu o nome de superfície cáustica e a sombra, de “shadow spot”.

Em 1986, Arvo propôs o primeiro algoritmo para a simulação desse fenômeno. Ele propôs uma implementação com ray tracing de duas passadas, onde a informação da primeira se comunica com a segunda. Na primeira passada, os raios de luz atingem os objetos da cena depositando uma quantidade de energia em seus light maps (texture map) que estão associados somente aos objetos com índice de reflexão/refração, alterando assim suas propriedades. Na segunda passada, os raios que atingem os objetos utilizarão o valor depositado para gerar a iluminação indireta.

Diversos sistemas para rendering 3D suportam o caustic, todos com resultados obtidos por análises das propriedades dos materiais com fácil aplicação e entendimento. Existem ainda algumas técnicas de simulação de caustic muito utilizadas por jogos 3D para real-time rendering ou por interessados em agilizar o processo de render em cenas complexas.

O fenômeno natural da superfície cáustica é percebido instintivamente por nossa mente, merecendo uma atenção especial de interessados no desenvolvimento de imagens hiper-realistas.

7.3.9. Radiosidade

A técnica radiosidade tem diversas aplicações para iluminação de animações, games ou imagens estáticas. Radiosidade é uma adição aos métodos de *rendering* 3D que aumenta o realismo de uma imagem. As imagens que resultam dessa renderização são caracterizadas por sombras suaves e graduais. Métodos convencionais de iluminação baseiam-se na reflexão do raio de luz nos objetos até o olho do observador. A radiosidade considera a reflexão de luz se transmitindo objeto para o outro até o olho do observador. A proposta básica da radiosidade não considera reflexões especulares mas existem muitos trabalhos para possibilitar a combinação entre radiosidade e reflexão especular.

Em muitas cenas, principalmente em cenas de interiores, existem zonas que não são diretamente iluminadas pelas fontes de luz. A iluminação de tais zonas é produto da luz refletida, uma ou mais vezes, por superfícies refletoras não-especulares. Para tratar esses casos, os métodos de ray tracing empregam um termo de iluminação ambiente constante cujo cálculo nem sempre é suficientemente preciso. O emprego dessa técnica faz com que as superfícies indiretamente iluminadas aparentem uma iluminação uniforme em vez de uma variação de sombreamento gradual e suave, o que conduz a diferenças abruptas de iluminação entre zonas diretamente iluminadas e zonas que lhes são contíguas e não são diretamente iluminadas. O método da radiosidade tem por objetivo o cálculo da iluminação e do sombreamento em cenas em que predominam superfícies refletoras difusas, é deriva do cálculo das trocas de radiação térmica entre superfícies empregada em Transmissão de Calor adaptado à computação gráfica. Na Figura 7.44, podemos observar uma variação de sombreamento gradual de uma luz oriunda da parte superior descoberta (clara-bóia) e que ilumina o interior da parte coberta.



FIGURA 7.44. Exemplo de Radiosidade e sua variação de sombreamento gradual.

Proposto em 1984 por Cindy Goral, Torrance & Greenberg, da Universidade de Cornell, no artigo *Modelling the Interaction of Light Between Diffuse Surfaces*, o algoritmo de radiosidade tem origem nos métodos fundamentais transferência de calor utilizados na engenharia térmica. A radiosidade define-se como sendo a energia por unidade de tempo e área em cada ponto dos objetos da cena, onde a energia luminosa incidente em uma superfície é refletida com igual intensidade em todas as direções.

O método da radiosidade é baseado em um modelo simples de balanço de energia. Na sua origem, o cálculo da radiosidade empregado em Transmissão de Calor não é mais do que a aplicação da lei da conservação da energia a cada uma das superfícies de um recinto ou cena, e pressupõe a existência de equilíbrio térmico. Em cada superfície de um modelo, a quantidade de energia emitida é a soma entre a energia que a superfície emite internamente mais a quantidade de energia refletida. A quantidade de energia refletida pode ser caracterizada pelo produto entre a quantidade de energia incidente na superfície e a constante de reflexão da superfície.

$$B_j = \rho_j H_j + E_j$$

onde B_j é a radiosidade da superfície j , ρ_j sua reflectividade, H_j a energia incidente nesta superfície e E_j a energia emitida pela superfície j

A radiosidade de uma superfície é a energia dissipada. Isso é usado para determinar a intensidade luminosa da superfície. A quantidade de energia emitida por uma superfície deve ser especificada como um parâmetro do modelo, como nos métodos tradicionais onde a localização e a intensidade das fontes de luz devem ser especificadas. A reflectividade da superfície também deve ser especificada no modelo, como nos métodos de iluminação tradicional. A única incógnita da equação é a quantidade de luz incidente na superfície. Esta pode ser encontrado somando-se todas as outras superfícies à quantidade de energia refletida que contribui com a iluminação dessa superfície:

$$H_j = \sum_{i=1}^n B_j F_{ij}$$

onde H_j é a energia incidente na superfície j , B_j a radiosidade de cada superfície i da cena e F_{ij} uma constante i, j .

A constante dessa equação é definida como a fração de energia que sai da superfície i e chega na superfície j , e é, portanto, um número entre 0 e 1. Essa constante pode ser calculada por métodos analíticos, ou através de semelhança geométrica.

A equação da radiosidade fica assim:

$$B_j = E_j + \rho_j \sum_{i=1}^n B_j F_{ij}$$

A consideração de todas as superfícies da cena forma uma seqüência de N equações lineares com N incógnitas, o que leva a uma solução matricial:

$$\begin{bmatrix} 1 - \rho_1 F_{11} & 1 - \rho_1 F_{12} & \Lambda & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1 - \rho_1 F_{22} & \Lambda & -\rho_2 F_{2n} \\ M & M & O & M \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \Lambda & -\rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ M \\ B_3 \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ M \\ E_3 \end{bmatrix}$$

Essa matriz tem duas propriedades interessantes: é diagonalmente dominante e, portanto, converge quando usada no método iterativo de Gauss Seidel, [GOR, 84]. Métodos alternativos para o cálculo dessa matriz já foram propostos por Cohen et al. [COH, 88]. Alguns permitem uma convergência para a solução correta mais rápida que o algoritmo iterativo de Gauss Seidel.

A renderização com o emprego da radiosidade utiliza diversos procedimentos. Primeiro é gerado um modelo discretizado das superfícies da cena a ser renderizada. Depois essas superfícies tem suas constantes avaliadas. A seguir o sistema de equações é montado e resolvido. Sendo finalmente a cena renderizada.

É importante notar que na renderização de uma cena com radiosidade, após a mudança dos parâmetros não é necessário reiniciar os cálculos desde o primeiro passo. Na verdade, apenas se a geometria do modelo for mudada é que será necessário realizar todos os cálculos novamente. Se os parâmetros da iluminação ou da reflexão forem mudados, o sistema pode reiniciar a partir da montagem e solução do sistema de equações. Se os parâmetros de visualização forem mudados, o sistema deve rapidamente re-renderizar a cena (último passo).

É importante observar que existe um alto custo neste processamento e que nem sempre se terá a melhor solução. Podemos amenizar esse custo aplicando alguns métodos de simulação da radiosidade através da utilização de diferentes tipos de luzes.

Um exemplo de simulação automática ocorre no software RenderMan da Pixar. Ao usarmos uma lâmpada, o sistema ajusta a suavização e projeção das sombras.

Contudo, diversos programas para *rendering* 3D já possuem a opção para *rendering* com radiossidade ajustada como padrão. O Lightflow é um exemplo, possuindo um método eficiente e rápido de cálculo da radiossidade. Na verdade, quase todos os programas tem disponível diferentes métodos de radiossidade, porém, por mais rápido que esses métodos possam ser, não serão rápidos o suficiente para uso em ambiente *real-time rendering*. No entanto aplicar a radiossidade em ambiente *real-time* devemos reduzir um pouco o seu realismo para obter uma melhor performance do algoritmo. Essa técnica foi aplicada em alguns jogos mas foi substituída por uma solução ainda melhor. O método utilizado pelas melhores engrenagens 3D baseia-se em *light maps* (ou *radiosity mapping*), uma solução onde, após o desenvolvimento da fase e posicionamento de luzes fixas do jogo, é executado uma única vez o cálculo da radiossidade total de um objeto a partir de um ponto fixo e depois para todos os outros pontos do objeto. O resultado será uma variação da iluminação depositada nos mapeamentos. Esse método é realmente muito eficiente, mas possui algumas limitações quando trabalhando com luzes móveis, objetos distantes ou grandes mapeamentos.

Outra possibilidade é a implementação do algoritmo no hardware das aceleradoras gráficas. Essa possibilidade ainda está em estudo devido a existência de infinitos métodos disponíveis no mercado.

7.3.9.1. Radiossidade com Luzes Dinâmicas

Como vimos anteriormente, na maioria dos jogos (ou sistemas *real-time*) a radiossidade é pré-calculada para a fase (chão, paredes, céu ou qualquer coisa fixa) e que isso funciona muito bem, desde que não exista uma fonte de luz móvel. Porém, sempre que disparamos um projétil ou míssil estamos criando uma fonte de luz móvel. Recalcular toda a radiossidade de todos os objetos da fase seria impossível. A solução, nesse caso, é segmentação do que é ou não relevante para ser recalculado. Todo objeto que tiver uma grande probabilidade de mudança na radiossidade deverá ser recalculado. Geralmente, só é feito o recálculo dos *light maps* para os objetos que estiverem próximos da luz dinâmica ou dos cantos entre objetos diferentes.

7.3.9.2. Refinamento Progressivo

Em 1988, Cohen, Chen, Wallace & Greenberg publicaram o artigo chamado *A Progressive Refinement Approach to Fast Radiosity Image Generation* que descrevia uma nova solução para geração rápida de imagens com radiossidade.

A idéia básica do refinamento progressivo é encontrar na cena, a superfície que pode contribuir com maior energia. A energia dessa superfície é então misturada com todas as outras superfícies e, em seguida, é feito um *render* da cena. Após finalizar o *render*, o processo se dá novamente, localizando-se a nova superfície que possui maior energia (nem sempre será a mesma), misturando-se a outras superfícies.

Então um novo *render* é feito até que o observador progressivamente possa avaliar o resultado e assim ficar, ou não, satisfeito. Esse processo permite ainda, em tempo de execução, o ajuste de iluminação ou cor da superfície. O processo de *render* com *radiosidade* reduzindo a necessidade de memória e de processador.

7.3.10. Técnicas de Iluminação

A iluminação é uma poderosa ferramenta capaz de atrair a atenção, passar emoções ou iludir o espectador. Grandes estúdios contratam especialistas em iluminação real para trabalhar em conjunto com equipes de computação gráfica. Na verdade, a iluminação é um dos pontos mais difíceis e demorados de um projeto. Portanto, não se sinta frustrado se após horas de modelagem perfeita com texturas maravilhosas, o seu projeto não ficar como imaginado. Como dica, recomendamos observar como a luz se manifesta na natureza e nos ambientes utilizando sempre as diretrizes para iluminação usada por fotógrafos, diretores e desenhistas durante o processo de instalação da iluminação para as cenas.

Geralmente os sistemas possuem um modelo de iluminação padrão. Esse modelo, também chamado de modo de trabalho, consiste de duas lâmpadas omni posicionadas, uma como luz principal e outra como luz de preenchimento. Essa iluminação padrão não atinge um nível de realismo satisfatório forçando o acréscimo de novas lâmpadas. Acrescentar novas lâmpadas significa não só escolher o tipo de lâmpada, mas também saber posicioná-las.

7.3.10.1. Luz Principal (Key Light)

Antes de iniciarmos o processo de iluminação devemos retirar a iluminação padrão do sistema. Esse processo geralmente é automático quando inserimos uma nova luz.

A luz principal cria a iluminação principal da cena representando a fonte dominante, como o sol ou uma lâmpada no centro do quarto. A luz chave é a mais intensa das lâmpadas e a responsável pela sombra.

Geralmente, essa luz é do tipo direcional e posicionada de 15 a 45 graus em relação à câmera (Figura 7.45). Embora essa posição forneça uma boa visão do objeto, o resultado é uma imagem plana e sem vida com sombras marcadas e muito pronunciadas. Além disso, a cena parecerá sempre escura demais, porque não há uma luz ambiente natural para iluminar as áreas sombreadas.

7.3.10.2. Luz de Preenchimento (Fill Light)

A luz de preenchimento suaviza e estende a iluminação provida pela luz principal, torna o objeto alvo mais visível, fornecendo a noção de profundidade e a sensação de realidade à cena. Geralmente usamos uma ou mais lâmpadas posicionadas no ângulo oposto ao ângulo da luz principal, simulando uma iluminação indireta (Figura

7.46). Para isso, a lâmpada, ou a soma delas, deve ter seu brilho reduzido à metade do valor da luz principal.

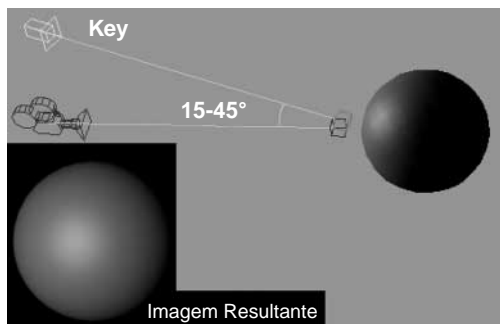


FIGURA 7.45. A luz principal responsável pelas sombras e pela iluminação da cena.

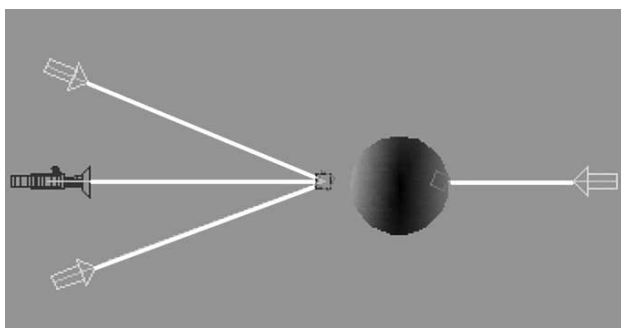


FIGURA 7.46. A luz de preenchimento suaviza e estende a iluminação provida pela luz principal.

7.3.10.3. Luz de Recuo (Back Light)

Também conhecida como luz de fundo ou luz de borda (Figura 7.47). O objetivo da luz de recuo é auxiliar a visualização do objeto alvo separando-o do fundo. Essa luz é posicionada atrás do objeto e oposta a câmera.

7.4. TEXTURAS

Os modelos de iluminação não são apropriados para descrever todas as propriedades da superfície de um objeto, por exemplo, rugosidade e padronagem. Em princípio, é possível modelar esses detalhes com o acréscimo de detalhes na geometria da superfície ou usando materiais de propriedades óticas distintas. Essa forma torna o processamento muito complexo, de modo que, na prática esses efeitos são modelados com o uso de mapas de textura.

A idéia básica é reproduzir sobre a superfície do objeto as propriedades de alguma função ou mapeamento bidimensional.

Para trabalhar com mapas de texturas é fundamental e ter um bom programa de composição e tratamento de imagens. Você poderá também usar bibliotecas de texturas disponíveis na Internet.

7.4.1. Mapas Procedurais

Um bitmap é uma imagem produzida por uma matriz fixa de pixels coloridos. Os mapas procedurais, por exemplo, um tabuleiro de xadrez, pode ser gerado por um algoritmo. Esses mapas dispensam a utilização de imagens e, podem ser inclusive tridimensionais por isso, são muito usados para a síntese de cenas complexas ou aplicações em real-time. Eles também são boas fontes para mapeamento de textura sintéticas.

Dentre os mapas procedurais, o mapa de ruído tem recebido uma atenção especial. Esses mapas têm produzido uma variedade de imagens bastante realistas com texturas geradas por algoritmos fractais (Figura 7.48).

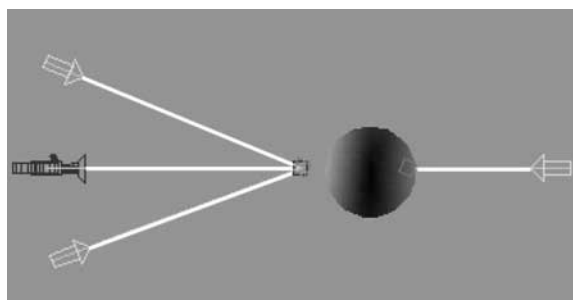


FIGURA 7.47. A luz de recuo auxilia a visualização do objeto alvo separando-o do fundo.

7.4.2. UVW Map

A maior parte dos mapas de materiais são planos (2D) atribuídos à superfície de objetos (3D). O sistema de coordenadas usado para descrever a colocação e transformação de mapas é diferente do sistema de coordenada X, Y, e Z usado para descrever o objeto. Especificamente, os mapas de coordenadas usam as letras U, V, e W; a três letras que precedem as X, Y, e Z no alfabeto empregadas no sistema de coordenadas do objeto.

U é o equivalente de X, e representa a direção horizontal do mapa; V é o equivalente de Y, e representa a direção vertical do mapa; W é o equivalente de Z e representa a direção perpendicular do plano UV. O eixo W é geralmente usado para mapas procedurais tridimensionais ou para girar os mapas na superfície dos objetos, o segundo caso ocorre quando trocamos os sistemas de coordenadas de VW para WU.



FIGURA 7.48. *Exemplo de Mapas Procedurais.*

O mapeamento aplicado deve se deformar com o objeto (Figura 7.49). Quando trabalhamos com objetos simples, como cubos e esferas, estes já possuem o seu mapa de coordenadas predefinido que pode ser utilizado no mapeamento do plano VV para a superfície dos objetos. Porém, objetos criados a partir de scanners ou malhas, não possuem um mapa de coordenadas forçando a criar mapeamentos complexos para que as texturas fiquem devidamente aplicadas ou usar mapeamentos em passados.



FIGURA 7.49. *UVW Map.*

7.4.3. Texture Map

Texture Map foi tradicionalmente usado para adicionar realismo nas imagens geradas por computador. Nos últimos anos, essa técnica saiu do domínio de sistemas de render por software para hardwares de alta performance.

Essa técnica foi inicialmente proposta por [CATMULL, 74]. Em sua forma básica, o mapa de textura aplica uma imagem, ou seja, uma textura sobre um objeto na cena. Por ser muito útil e simples, passou a ser considerado como padrão para as interfaces de softwares e hardwares gráficos. Os mapas de texturas podem ser usados em cenas complexas com um baixo custo de processamento.

Quando mapeamos uma textura em um objeto, a cor do objeto em cada pixel é modificada pela cor correspondente na textura. Para aplicar a imagem nos objetos, devemos antes seguir alguns passos:

- A textura deve ser armazenada como um array;
- A textura deve ser mapeada para adaptar-se a superfícies do objeto e a uma visão da cena em perspectiva;
- A textura deve ser tratada para retirar-se imperfeições devidas ao aliasing.

É importante considerar que as imagens de textura não precisam ser necessariamente bidimensionais. No caso tridimensional, uma fatia bidimensional deve ser selecionada para descrever as propriedades da superfície usadas no cálculo da iluminação e sombreadimento.

Uma generalização da técnica, chamada de projective textures, propõe a projeção da imagem sem coordenadas fixas. O resultado é uma imagem deslizando sobre a superfície. Essa técnica permite simular animações de água ou refletores. Projective texture é também muito útil na simulação de sombras. Nesse caso, uma imagem é construída representando a distância da fonte de luz em relação ao polígono.

Ao técnicas de mapeamento podem também representar objetos transparentes ou semitransparentes. Essa técnica é muito usada na simulação de nuvens e árvores onde mapas 2D são inseridos na cena em retângulos.

7.4.4. Environment Mapping ou Mapa de Reflexão

Os environment maps são usados para fazer o render de objetos reflexivos. Esse modelo foi inicialmente proposto por Blinn & Newell em 1976 e é capaz de simular efeitos de ray tracing a baixo custo. Environment mapping é o tipo de mapeamento que reflete na superfície dos objetos os elementos que compõem a cena.

Essa técnica pode ser alcançada de duas formas. A primeira forma, tomando um cubo como exemplo, requer seis imagens de textura, uma para cada direção contendo as informações dos objetos que compõem o ambiente. Para cada vértice do polígono, um vetor de reflexão é calculado. Esse vetor indica uma das seis imagens. Esse método não é muito exato mas bem convincente. O segundo método é gerar uma única imagem de uma esfera refletindo o ambiente (Figura 7.50). Cada hemisfério da esfera representará um hemisfério do ambiente.

Um dos problemas dessa técnica é que o objeto não reflete a si mesmo, o que pode resultar em erros na imagem de objetos não convexos.

7.4.5. Bump Map

Quando utilizamos uma fotografia de uma superfície áspera como mapa de textura, a superfície renderizada não fica muito correta, porque a direção da fonte de luz utilizada para criar o mapeamento é diferente da direção da iluminação do sólido. Blinn desenvolveu uma técnica para amenizar esse efeito, dando uma perturbação na normal à superfície antes de aplicar o modelo de iluminação. Essa perturbação produz um deslocamento virtual na posição dos pontos da superfície.



FIGURA 7.50. *Environment Mapping.*

Bump Map é uma técnica usada para adicionar realismo sem modificar a geometria ao objeto. Essa técnica adiciona um sombreamento nos pixels, produzindo uma ilusão de relevo no objeto renderizado. A cor de uma superfície está relacionada com ângulo entre o vetor normal da superfície e a direção da luz. Em uma superfície plana, o vetor normal é o mesmo para toda a superfície, logo a cor da superfície será sempre a mesma. No bump map, as propriedades de refração da luz são usadas para indicar quais partes são mais escuras ou mais claras (Figura 7.51). Para isso, a técnica irá perturbar o vetor normal em vários pontos da superfície criando uma ilusão de que algumas partes da superfície estariam elevadas ou rebaixadas.

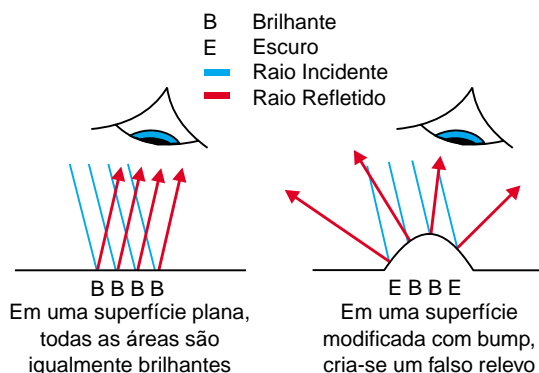


FIGURA 7.51. *As propriedades de refração da luz em Bump Map.*

Com a finalidade de uma perturbação adequada da normal, para produzir determinado efeito usamos um mapa de altura (Height Map), fazendo com que zonas de maior intensidade de cor (branco) pareçam em alto-relevo e as de menor intensidade (preto) pareçam em baixo-relevo, ou vise-versa. Na Figura 7.52, é apresentado o mapa de textura com seu height map usada para o mapeamento maçã. As superfícies que aparentam ter uma rugosidade são boas candidatas para os bump maps. Ótimo exemplo dessa técnica é o muro de tijolos da Figura 7.53, observe como o muro da direita apresenta os relevos entre os tijolos.

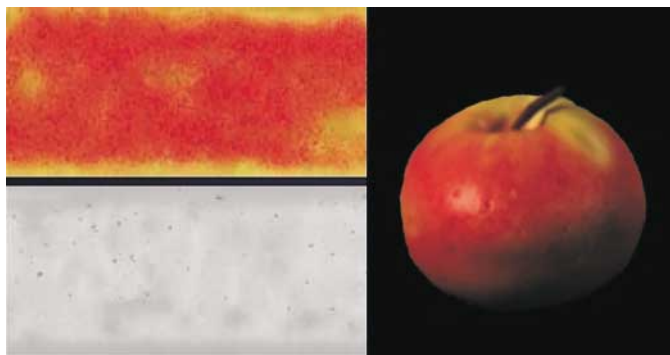


FIGURA 7.52. Mapa de Textura, Bump Map e efeito sobre um objeto.

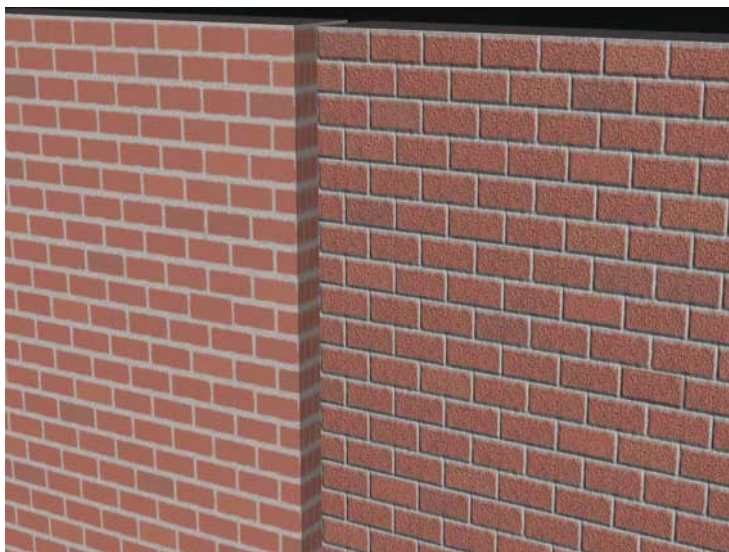


FIGURA 7.53. Paredes da esquerda sem Bump Map e da direita com Bump Map.

7.4.5.1. Emboss Bump Map

O tipo mais comum de bump mapping é o emboss map. Essa técnica usa os mapas de textura para gerar os efeitos de bump map sem precisar de um render específico.

Os passos desse algoritmo são os seguintes (Figura 7.54):

- Renderiza a imagem com uma textura
- Desloca-se as coordenadas da textura
- Re-renderiza-se a imagem com textura, sobrepondo-se a primeira imagem

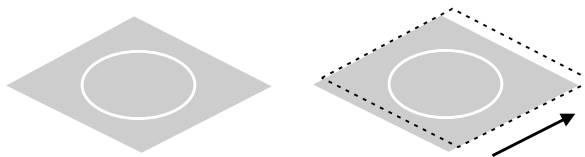


FIGURA 7.54. *Emboss Bump Map.*

Assumindo que os valores de altura podem ser representados como uma função de altura $f(s)$, então os três processos descritos anteriormente podem ser descritos pela função:

$$f(s) - f(s + \text{deslocamento}).$$

Se o deslocamento foi de apenas um pixel em s , teremos

$$f(s) - f(s + \frac{L}{w})$$

onde w é a largura da textura e L o comprimento.

7.4.5.2. Environment Mapped Bump Mapping

Environment Mapped Bump Mapping utiliza três mapas: um mapa de textura, um bump map e um environment map. Essa técnica permite simular bump maps animados, como as ondas, provocando distorções nos objetos.

Essa técnica não utiliza o processo de deslocamento aplicado no emboss bump map mas um environment map para calcular o bump map. O environment map pode conter várias fontes de luz, inclusive especulares, e não necessita de uma passada de render adicional.

7.4.5.3. Per-Pixel Shading

Per-pixel shading é um método de rendering que permite aplicar, a cada pixel, efeitos especiais de materiais ou simulações do mundo real com muito mais intensidade e precisão. Essa técnica vem sendo usada na produção de filmes para criar imagens realistas. No filme *Toy Story*, o personagem Buzz Lightyear (o astronauta) tem no seu capacete transparente uma luz refletia mostrando o ambiente e o fundo da cena.

Per-pixel shading é muito bom para simular fenômenos naturais e atributos de superfície como pele, roupas, metais, vidros, pedras e outras superfícies complexas. Suporta iluminação difusa, especular, spot e point light.

7.4.6. Light Map

Os light maps foram inicialmente usados em 1987 por Arco e Kirk para simular ray tracing em duas passadas. Suas motivações se devem à aplicação do método de armazenamento da iluminação difusa para viabilizar a implementação de um modelo de iluminação global que pudesse ser executado em um tempo razoável. Porém, aplicações recentes utilizam esse recurso para iluminação e sombreamento em aplicações de tempo real.

Os light maps possibilitam um pré-cálculo da luz para armazenamento em mapa de textura bidimensional e são usados para dar velocidade no render de objetos 3D com iluminação complexa. Na teoria, não existe nenhuma limitação para light maps tridimensionais mas, na prática, exigirá uma grande quantidade de memória. Em vez de calcularmos o efeito provocado pela luz no objeto em real-time, armazenamos uma amostra da contribuição da fonte de luz previamente calculada e aplicamos como textura no objeto 3D (Figura 7.55). Uma das grandes vantagens é a possibilidade de se usar, o resultado de métodos, como radiosidade ou outros de iluminação global, que poderiam demorar até 1 hora para serem calculados. É importante perceber que, quando o observador (ou o objeto) muda sua posição, as coordenadas do mapeamento devem ser corrigidas. De qualquer forma, quando lidamos com uma iluminação complexa, modificar as coordenadas será ainda muito mais rápido do que calcular o efeito da iluminação em cada superfície. Outra importante característica do light map é a capacidade de prover um highlight shading mais realístico, sem sacrificar a velocidade de render. Esse benefício é ainda mais acentuado quando lidamos com superfícies reflexivas ou muito brilhantes.

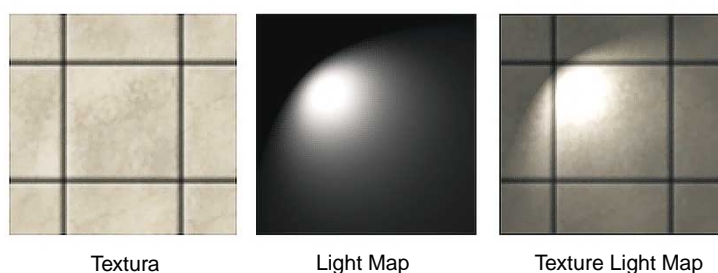


FIGURA 7.55. Amostra da contribuição da fonte de luz aplicada na textura.

Apesar de todos os benefícios obtidos com os light maps, existem algumas limitações quanto ao seu uso. Primeiro, a fonte de luz deve ser estática. Se desejamos utilizar os light maps para uma luz móvel, como por exemplo, um projétil, os light maps podem ser recalculados, as coordenadas do mapeamentos alteradas ou redimensionadas. Recalculá-los completamente para todos os objetos da cena seria impossível. A solução, neste caso, é segmentar o que é ou não relevan-

te para ser recalculado. Todo objeto que tiver uma grande probabilidade de mudança na sua luz deverá ser recalculado, geralmente, os objetos que estiverem próximos da luz móvel ou cantos entre objetos diferentes.

Se sua placa gráfica não suporta multi-texturing (recurso desenvolvido para acelerar o trabalho de misturar diversas texturas em um objeto), o render representará muito custo de tempo. Outra limitação pode ser constatada quando utilizamos os light maps em máquinas lentas, podendo ocorrer um atraso na inicialização da cena. Essa limitação, pode ser contornada com a utilização de light maps customizados.

Quase todas as engrenagens 3D utilizam os light maps. Nelas, os light maps são utilizados para simular os efeitos de luzes locais estacionárias e móveis, produzindo um ótimo resultado final.

Um light map deve ser aplicado a uma superfície como o mapa de textura de uma cena, modulando a intensidade da superfície para simular os efeitos de uma luz local. Porém, quando trabalhamos com real-time, não podemos ficar mudando a textura da memória, o que consumiria um tempo valioso. Nesse caso, podemos criar um light map com características de intensidade e cor de algumas luzes e aplicá-lo em diferentes superfícies alterando somente as coordenadas e dimensões do mapeamento, simulando diferentes tamanhos e distâncias da luzes. A Figura 7.56 (cortesia da Paralelo Computação, www.fly3d.com.br) apresenta um light map de uma fase da engrenagem Fly3D. Nesse light map, todas as contribuições de iluminação da cena são armazenadas em uma única imagem guardando as respectivas coordenadas.



FIGURA 7.56. *Light Map de uma fase da engrenagem Fly3D (Cortesia da Paralelo Computação).*

7.4.7. Mip-Mapping

Mip-Mapping é um tipo de mapeamento que propõe a solução de dois problemas no mapeamento de objetos em cenas de animação ou real-time rendering. O primeiro problema ocorre com o afastamento do observador em relação aos objetos. Quando as texturas destes ficam maiores do que a quantidade de pixels disponíveis, o render terá de aplicar o anti-aliasing realizando uma média final dos pixels da imagem, produzindo como resultado final ruídos de fundo especialmente orientado (moiré) (Figura 7.57). Além desse indesejado ruído, devemos somar o tempo de processamento para o cálculo da média. Para se ter uma idéia, imagine uma textura de 256x256 pixels (65.536 pixels) a ser renderizada em um cubo que ocupa 8x8 pixels (64 pixels) da tela. Isso significará que cada pixel deverá conter a informação de $65.536/64=1024$ pixels.

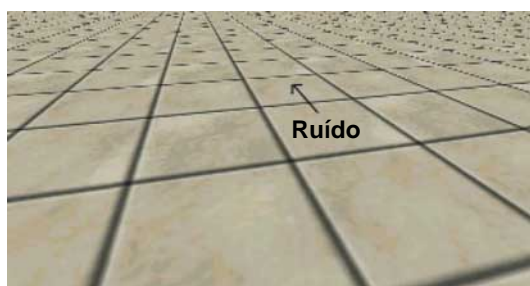


FIGURA 7.57. *Ruído provocado pela redução de pixels disponíveis.*

O segundo problema ocorre com a aproximação do objeto alvo, revelando detalhes indesejados da textura (geralmente por baixa resolução). Esse problema pode ser contornado aumentando a resolução da imagem, mas isso significaria mais memória, tempo de processamento e ruídos provocados pelo primeiro problema.

Para solucionar esses problemas, o mip-mapping propõe a utilização de mais de uma imagem como textura, onde cada imagem possui um nível de resolução decrescente conforme o observador se afasta. Por exemplo, enquanto caminha através de uma galeria virtual, os quadros ou obras de arte próximos ao observador teriam uma resolução alta que permitiria a visualização dos detalhes da obra, enquanto isso, as obras distantes receberiam uma imagem de baixa resolução. Isso significará uma melhor qualidade do render com economia de memória e processador.

Vamos supor uma textura de tamanho 128x128. Se reduzirmos nossa textura por um fator 2, teremos no final uma textura igual (menos detalhada) com um tamanho 64x64. Nesse caso, a perda do detalhe não será percebida porque você terá se afastado do objeto. Segue-se esta idéia até 2x2 onde a textura terá somente uma cor (Figura 7.58).

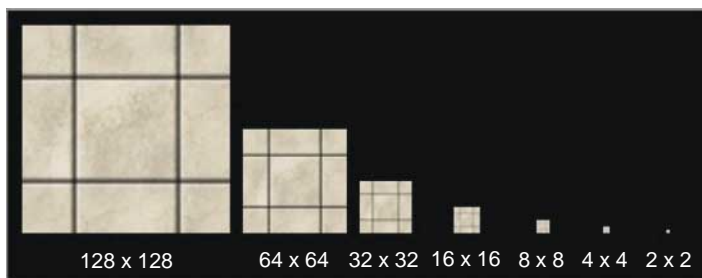


FIGURA 7.58. *Redução da textura pelo fator 2.*

Esse tipo de mip-mapping é o que podemos chamar de mip-mapping convencional e é usado pela maioria das engrenagens e placas 3D. Um outro tipo de mip-mapping é o “mip-mapping bidirecional”. Nesse tipo, a textura não recebe uma redução igual em ambas as direções, mas uma redução diferente no sentido horizontal (U) e outra na vertical (V). O mip-mapping bidirecional apresenta um resultado melhor sempre que a redução for diferente em cada sentido (Figura 7.59).

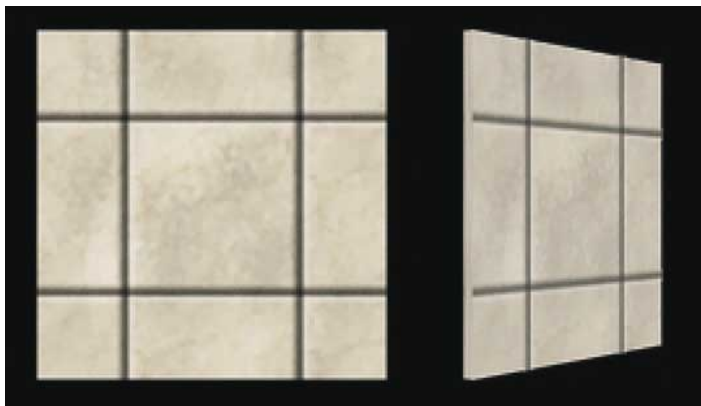


FIGURA 7.59. *Mip-mapping bidirecional.*

7.5. HIPER-REALISMO

As técnicas de hiper-realismos são adições aos métodos de rendering que aumentam o realismo de uma imagem para o fotorrealismo.

Os métodos de iluminação global ray tracing, caustic e radiosidade são exemplos clássicos da síntese de imagens realistas. Grande parte desses métodos ocorre na passada de iluminação ou efeito. Esses recursos geralmente não estão disponíveis no render padrão dos sistemas, forçando os artistas a buscar técnicas de simulação. Porém, você poderá incorporar esses recursos instalando plug-ins como o Final Render ou Render Brazil.

7.5.1. High Dynamic Range Images (HDRI)

Geralmente ao se criar uma cena 3D, os objetos distantes como montanhas, nuvens, lagos, árvores e todo cenário de fundo não são modelados para não sobrecarregar inutilmente o software/hardware, deixando-o lento e travado. Nesses casos, são usadas imagens como fundo (backgrounds), essas imagens são incorporadas a cena, mas não interagem com os objetos, ou seja, não contribuem na iluminação.

Os arquivos de extensão .hdr possuem propriedades de energia passando ao programa quais regiões da imagem são fontes de luz. O que ocorre é que as imagens HDR possuem mais informações, além dos canais de cores (RGB). Essas informações adicionais dizem quais são os pixels da imagem que contêm atributos de energia luminosa. Sendo assim, em uma foto convencional de um céu onde existe o Sol, por exemplo, a área que corresponde a ele seria no máximo branca, “(com informações de RGB em 255,255,255)” mas, na verdade, deveriam existir bem mais intensidades e estas deveriam contribuir para a iluminação da cena. O mapa de imagem com HDR corrige essa deficiência incluindo na imagem informações de intensidade de luz. Uma das coisas mais interessantes deste método é que, em muitos casos, será desnecessária a utilização de qualquer tipo de lâmpadas na cena, ficando a iluminação total a cargo do mapa HDR.

Objetos iluminados com HDR parecem realmente fazer parte de todo o cenário. Isso possibilita um fotorealismo impressionante, e quase sempre as cenas não precisam conter uma única fonte direta de luz. A Figura 7.60 apresenta duas imagens com mapeamento HDR de diferentes intensidades. Observe que nessa cena não existe nenhuma lâmpada e que a luz parte da própria imagem.



FIGURA 7.60. *High Dynamic Range Images (HDRI).*

7.5.2. Atenuação Atmosférica

Neste modelo, os objetos que estão mais distantes do observador vão parecer estar mais escuros. Isso ocorre no mundo físico por causa de materiais presentes na atmos-

fera como poeira, vapor d'água ou fumaça. O que acontece é que a fonte de luz emite raios, que quando se encontram com as camadas atmosféricas, são em parte refletidos e em parte dispersados (Figura 7.61).

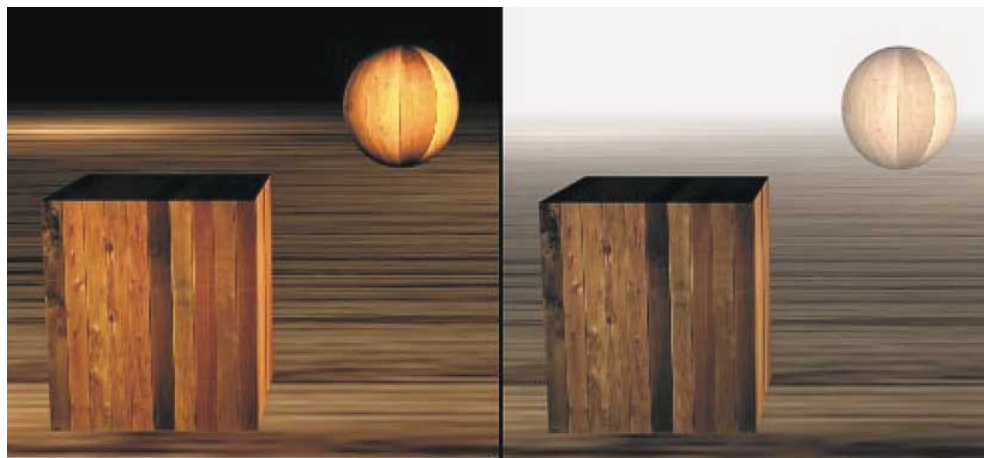


FIGURA 7.61. *Efeito provocado pela atenuação atmosférica.*

7.5.3. Area Light e Soft Shadow

Quando temos uma fonte de luz, os raios emitidos na maioria dos casos não derivam de um único ponto. No caso de uma lâmpada fluorescente tubular, a luz parte de todo o seu extenso cilindro.

Ao projetar sombras, essas luzes, chamadas de Áreas de Luz, formam uma sombra suave que se dispersa à distância, Figura 7.16. Isso permite um sombreamento mais convincente e verdadeiro. Essas sombras são típicas de cenas que usaram radiosidade ou iluminação global, e por isso, o uso simulado desse tipo de iluminação serve também para ajudar na produção de falsa radiosidade.

7.5.4. Sub-Surface Light Scattering

Sub Surface Light Scattering é um aprimoramento das técnicas de radiosidade que leva em conta o material utilizado no objeto. Se colocarmos nossa mão próxima a uma lâmpada, veremos que a luz atravessa a mão produzindo uma coloração avermelhada. A essa propriedade damos o nome de translucidez (cuidado para não confundir com transparência).

A translucidez está presente em diversas obras de arte (como estátuas) e em uma infinidade de elementos na natureza. Na Figura 7.62, uma das estátuas recebeu o tratamento enquanto a outra não.

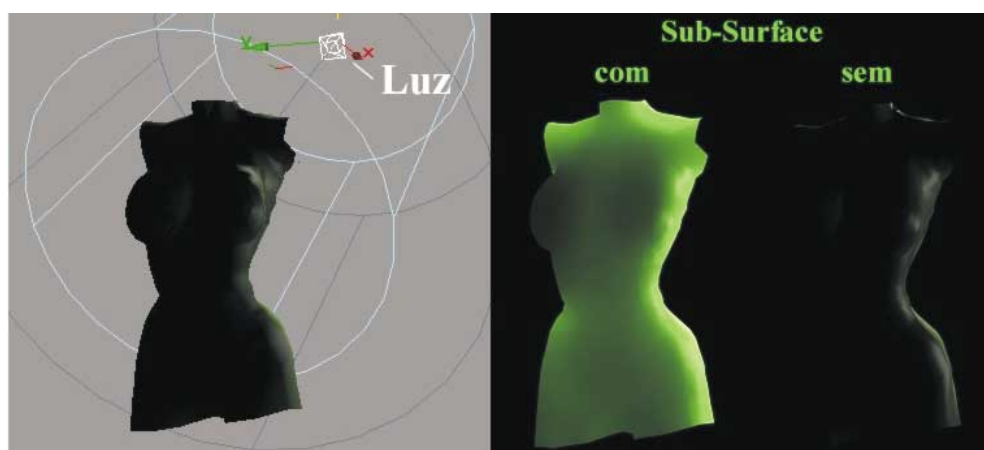


FIGURA 7.62. Sub-Surface Light Scattering aplicado em um objeto transparente.

Neste contexto existem mais duas técnicas: BRDF (*Bidirectional Reflectance Distribution Function*) e BSSRDF (*Bidirectional Surface Scattering Distribution Function*).

A BSSRDF descreve, com maior precisão do que a BRDF, a forma como a luz entra e sai de um material. BSSRDF pode descrever os movimentos da luz no momento em que entra e sai do material, enquanto BRDF assume que o raio de luz entra e sai do material no mesmo ponto.

Sub Surface Light Scattering utilizará BSSRDF para descrever a mudança do ângulo da luz provocada pela particularidade do material. A Figura 7.63 apresenta o detalhamento interno da superfície de uma folha.

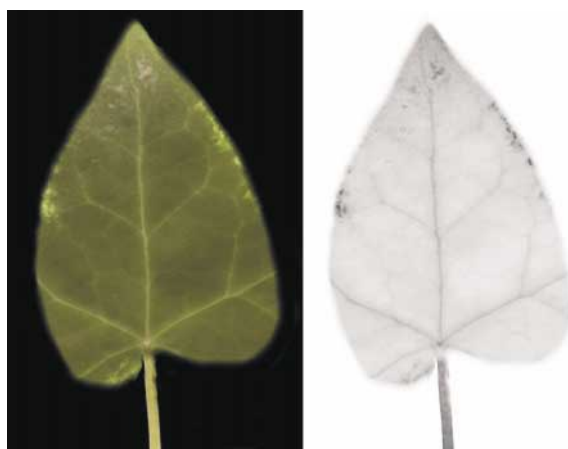


FIGURA 7.63. Detalhamento interno da superfície de uma folha com Scattering.

7.5.5. Depth of Field (DOF) ou Profundidade de Campo

Ao fotografar ou filmar algo com uma câmera, temos de ajustar o foco para o objeto alvo (Figura 7.64). Esse ajuste faz com que todos os objetos que estiverem em outras profundidades (na frente ou atrás) apareçam borrados.

O efeito de DOF simula na iluminação esse efeito em 3D, causado pela lente da câmera, é um recurso muito útil para conseguir mais inteligibilidade em uma cena.

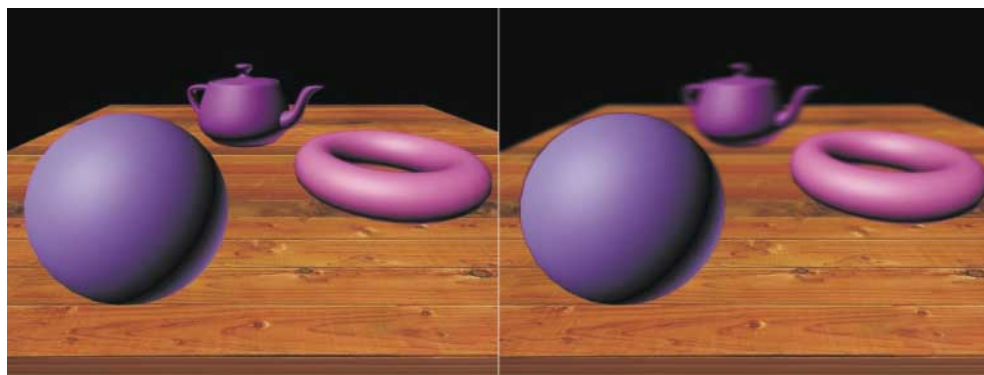


FIGURA 7.64. *Depth of Field (DOF) ou Profundidade de Campo.*

7.5.6. Motion Blur / Desfocagem por Movimento

Quando fotografamos um objeto em movimento, observamos que ele aparece desfocado, embaçado em relação aos outros (exceto quando aumentamos muito a velocidade do diafragma). No processo físico, o desfoque por movimento ocorre em consequência do tempo de exposição do filme à luz. Os sistemas tentam simular esse efeito calculando a posição dos objetos nos quadros anteriores e nos posteriores e misturando todas essas diferentes posições num único quadro. Esse efeito é muito simples e fácil de ser usado, podendo ser aplicado nos objetos ou na cena inteira.

7.5.7. Film Grain

Este efeito recria o efeito granulado das películas filmes. Pouco utilizado pela maioria dos usuários de computação gráfica, é um dos efeitos mais importantes existentes. Um pequeno toque de ruído na cena pode tirar, em grande parte, aquele visual e limpo de imagens geradas por computador.

7.5.8. Lens Flare

São pequenos círculos que você normalmente veria saindo da fonte de luz. Eles são causados pela refração da luz nos diversos elementos de lente na máquina fotográfica ou vídeo. Repare na Figura ColorPlate 4 o pequeno círculo ao lado do sol e o arco-íris formado pela lente.

7.5.9. Glow

Adiciona uma aura ao redor do objeto. Por exemplo, para sistema de partícula ou uma explosão, adicionando o glow as partículas parecerem mais brilhantes e quentes.

O glow é muito usado para simular a exaustão em turbinas de avião, mas se você observar esse efeito na natureza na forma de estrelas, anéis, listras ou outras formas. Na maioria das vezes, essas formas ocorrem em conjunto.

7.6. REALISMO E ILUMINAÇÃO EM OPENGL

7.6.1. Z-buffer

Buffers são áreas reservadas de memória utilizadas para determinados propósitos. Em aplicações de animação, por exemplo, o double-buffer permite que as sucessivas renderizações sejam feitas de modo suave, sem o efeito indesejável de piscar entre cada atualização da janela de visualização.

Para isso, o comando `auxInitDisplayMode` deve ser utilizado com o parâmetro `AUX_DOUBLE` e o comando `auxSwapBuffers` deverá ser acrescentado ao final dos comandos de renderização.

O z-buffer também é bastante comum em aplicações gráficas e é utilizado para calcular a distância do observador e remover superfícies ou partes ocultas de objetos sobrepostos. Para ativar este recurso utilizamos o comando `auxInitDisplayMode` com o parâmetro `AUX_DEPTH`, o comando `glClear` com o parâmetro `GL_DEPTH_BUFFER_BIT` e acrescentamos o comando `glEnable` com o parâmetro `GL_DEPTH_TEST` antes dos comandos de renderização.

```
glDepthRange(znear, zfar);    // default: 0, 1
```

7.6.2. Hidden-Surface Removal

```
While (1) {  
    pegar_o_ponto_de_vista_da_posição_do_mouse( );  
    glClear(GL_COLOR_BUFFER_BIT);  
    draw_3d_object_A( );  
    draw_3d_object_B( );  
}
```

7.6.3. Algoritmo de Eliminação de Faces (Culling)

O Comando `glCullFace(GGLenum mode)` indica se alguma face ou se todo o objeto deve ser descartado antes de serem convertidos para a coordenada de vídeo. As opções deste comando incluem `GL_BACK`, para exclusão da face posterior, `GL_FRONT`, para exclusão da face dianteira e `GL_FRONT_AND_BACK`, para exclusão de todo o objeto.

7.6.4. Iluminação

7.6.4.1. Modelo de Tonalização

O modelo de tonalização pode ser estabelecido com o comando

```
glShadeModel(GGLenum mode);
```

onde **mode** especifica o modelo de tonalização, sendo o padrão `GL_SMOOTH` (modelo de Gouraud) onde a cor de cada ponto da primitiva é interpolada a partir da cor calculada nos vértices. O outro modelo é `GL_FLAT` onde a cor não varia.

7.6.4.2. Propriedades de Iluminação do Material

O comando `glMaterial` e suas variações estabelecem os parâmetros do material que serão usados pelo modelo de iluminação. As variações desse comando são:

```
glMaterialf(GGLenum face, GGLenum pname, GLfloat param);  
glMateriali(GGLenum face, GGLenum pname, GLint param);  
glMaterialfv(GGLenum face, GGLenum pname, const GLfloat *params);  
glMaterialiv(GGLenum face, GGLenum pname, const GLint *params);
```

onde **face** determina se as propriedades do material dos polígonos que estão sendo especificadas são: front (`GL_FRONT`), back (`GL_BACK`) ou ambas (`GL_FRONT_AND_BACK`);

pname para as duas primeiras variações especifica o parâmetro de um único valor que está sendo determinado; para as duas últimas variações, que recebem um vetor como parâmetro, pode determinar as seguintes propriedades do material: `GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR`, `GL_EMISSION`, `GL_SHININESS`, `GL_AMBIENT_AND_DIFFUSE` ou `GL_COLOR_INDEXES`;

param (`GLfloat` ou `GLint`) especifica o valor que será atribuído para o parâmetro determinado por **pname**;

params (`GLfloat*` ou `GLint*`) é um vetor de números reais ou inteiros que contém as componentes da propriedade que está sendo especificada;

Através dessa função, é possível determinar as propriedades de reflexão do material. As propriedades `GL_AMBIENT`, `GL_DIFFUSE` e `GL_SPECULAR` afetam a maneira na qual as componentes de luz incidente são refletidas. `GL_EMISSION` é usado para materiais que possuem “luz própria”. `GL_SHININESS` pode variar de 0 a 128 (quanto maior o valor, maior é a área de highlight especular na superfície). `GL_COLOR_INDEXES` é usado para as propriedades de refletância do material no modo de índice de cores.

7.6.4.3. Modelo de Iluminação

O comando `glLightModel` com suas variações estabelece os parâmetros do modelo de iluminação usado por OpenGL. É possível especificar quantos modelos desejar:

`GL_LIGHT_MODEL_AMBIENT`: é usado para especificar a luz ambiente padrão para uma cena;

`GL_LIGHT_MODEL_TWO_SIDE`: é usado para indicar se ambos os lados de um polígono são iluminados. O padrão ilumina somente o lado frontal;

`GL_LIGHT_MODEL_LOCAL_VIEWER`: modifica o cálculo dos ângulos de reflexão especular;

As variações desse comando são:

```
glLightModelf(GLenum pname, GLfloat param);  
glLightModeli(GLenum pname, GLint param);  
glLightModelfv(GLenum pname, const GLfloat *params);  
glLightModeliv(GLenum pname, const GLint *params);
```

onde **pname** especifica o modelo de iluminação;

param (`GLfloat` ou `GLint`) para `GL_LIGHT_MODEL_LOCAL_VIEWER` um valor 0.0 indica que os ângulos da componente especular tomam a direção de visualização como sendo paralela ao eixo z, e qualquer outro valor indica que a visualização ocorre a partir da origem do sistema de referência da câmera; para `GL_LIGHT_MODEL_TWO_SIDE`, um valor 0.0 indica que somente os polígonos frontais são incluídos nos cálculos de iluminação, e qualquer outro valor indica que todos os polígonos são incluídos nos cálculos de iluminação;

params (`GLfloat*` ou `GLint*`), para `GL_LIGHT_MODEL_AMBIENT` ou `GL_LIGHT_MODEL_LOCAL_VIEWER`, aponta para um vetor de números inteiros ou reais; para `GL_LIGHT_MODEL_AMBIENT` o conteúdo do vetor indica os valores das componentes RGBA da luz ambiente.

7.6.4.4. Fonte de Luz

O Comando `glLight` e suas variações estabelecem os parâmetros da fonte de luz para uma das oito fontes de luz disponíveis. As variações desse comando são:

```
glLightf(GLenum light, GLenum pname, GLfloat param);  
glLighti(GLenum light, GLenum pname, GLint param);  
glLightfv(GLenum light, GLenum pname, const GLfloat *params);  
glLightiv(GLenum light, GLenum pname, const GLint *params);
```

onde **light** especifica qual fonte de luz está sendo alterada (varia de 0 a `GL_MAX_LIGHTS`); valores constantes de luz são enumerados de `GL_LIGHT0` a `GL_LIGHT7`;

pname especifica qual parâmetro de luz está sendo determinado pela chamada dessa função (`GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR`, `GL_POSITION`, `GL_SPOT_DIRECTION`, `GL_SPOT_EXPONENT`, `GL_SPOT_CUTOFF`, `GL_CONSTANT_ATTENUATION`, `GL_LINEAR_ATTENUATION`, `GL_QUADRATIC_ATTENUATION`);

param (`GLfloat` ou `GLint`) para parâmetros que são especificados por um único valor (*param*); esses parâmetros, válidos somente para *spotlights*, são `GL_SPOT_EXPONENT` (determina a concentração da luz), `GL_SPOT_CUTOFF` (especifica a largura do cone), `GL_CONSTANT_ATTENUATION`, `GL_LINEAR_ATTENUATION` e `GL_QUADRATIC_ATTENUATION` (tipos de atenuações);

params (`GLfloat*` ou `GLint*`) um vetor de valores que descrevem os parâmetros que estão sendo especificados.

As duas primeiras variações requerem apenas um único valor para determinar uma das seguintes propriedades: `GL_SPOT_EXPONENT`, `GL_SPOT_CUTOFF`, `GL_CONSTANT_ATTENUATION`, `GL_LINEAR_ATTENUATION` e `GL_QUADRATIC_ATTENUATION`. As duas últimas variações são usadas para parâmetros de luz que requerem um vetor com múltiplos valores (`GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR`, `GL_POSITION` e `GL_SPOT_DIRECTION`). Para eliminar o “brilho” do objeto, como se o material fosse opaco, basta eliminar a componente especular.

Programa Exemplo de Iluminação

```
float pos[4] = {3,3,3,1};  
float color[4] = {1,1,1,1};  
float sp[4] = {1,1,1,1};  
float mat_specular[ ] = {1,1,1,1};  
  
glEnable(GL_DEPTH_TEST);  
glEnable(GL_COLOR_MATERIAL);
```

```

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT3);
glEnable(GL_LIGHT5);
glEnable(GL_LIGHT6);
glLightfv(GL_LIGHT3, GL_SPECULAR, sp);
glLightfv(GL_LIGHT5, GL_SPECULAR, sp);
glLightfv(GL_LIGHT6, GL_SPECULAR, sp);

color[1]=color[2]=0;
glLightfv(GL_LIGHT3, GL_DIFFUSE, color);
color[0]=0;
color[1]=1;
glLightfv(GL_LIGHT5, GL_DIFFUSE, color);
color[1]=0;
color[2]=1;
glLightfv(GL_LIGHT6, GL_DIFFUSE, color);
glLightfv(GL_LIGHT3, GL_POSITION, pos);
pos[0] = -3;
glLightfv(GL_LIGHT5, GL_POSITION, pos);
pos[0]=0;pos[1]=-3;
glLightfv(GL_LIGHT6, GL_POSITION, pos);

glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialf(GL_FRONT, GL_SHININESS, 128.0);

```

7.6.5. Texturas

O padrão OpenGL exige que as dimensões das imagens de texturas sejam em potências de 2. As imagens de textura podem também ter 1 ou dois pixels de borda sobre suas extremidades para definir a cor dos polígonos que estão fora da imagem de textura.

O uso de texturas requer a execução de dois passos distintos: a carga e a aplicação da textura. Em OpenGL cada textura recebe um número de identificação através da função `glGenTextures` que é usado para definir a textura corrente. Sempre que for necessário trabalhar com uma textura (para carga ou aplicação) deve-se chamar a função `glBindTexture` que define a textura corrente através do número de identificação.

7.6.5.1. Carga de uma Textura

Para realizar a tarefa de carga de uma textura é necessário seguir os seguintes passos:

```

// Habilitar o uso de texturas
glEnable (GL_TEXTURE_2D); //GL_TEXTURE_2D define que será usada uma
    textura 2D

```

```
// Definir a forma de armazenamento dos pixels na textura (1= alinhamento por byte)
glPixelStorei (GL_UNPACK_ALIGNMENT, 1);

// Definir quantas texturas serão usadas no programa
GLuint texture_id[MAX_NO_TEXTURES]; // vetor com os números das texturas
glGenTextures (1, texture_id);      // 1 = uma textura;
    // texture_id = vetor que guarda os  números das texturas

// Definir o número da textura do cubo.
texture_id[0] = 1001;

// Definir a textura corrente
glBindTexture (GL_TEXTURE_2D, texture_id[0]);

// carregar uma imagem TGA
image_t temp_image;
tgaLoad ("TGAImage.tga", &temp_image, TGA_FREE | TGA_LOW_QUALITY);
```

7.6.5.2. Aplicando uma Textura

Para a aplicação da textura, é preciso criar uma relação entre os vértices da textura e os vértices dos polígonos sobre os quais se deseja mapear a textura escolhida. O processo de mapeamento de texturas em OpenGL consiste em “aplicar” a imagem 2D sobre o polígono 3D de forma que os pontos A, B, C e D (Figura 7.65) sejam encaixados sobre os pontos A1, B1, C1 e D1. Para permitir a construção dessa correspondência entre a imagem 2D e o polígono 3D, usa-se a função `glTexCoord2f` antes da definição do ponto 3D. Por exemplo

```
glTexCoord2f(0.0f, 0.0f);
glVertex3f(-1.0f, 1.0f, -1.0f);
```

define que o ponto (0.0, 0.0) da textura 2D corresponde ao ponto (-1.0, 1.0, -1.0) do polígono 3D. O Código para aplicação da textura segue como:

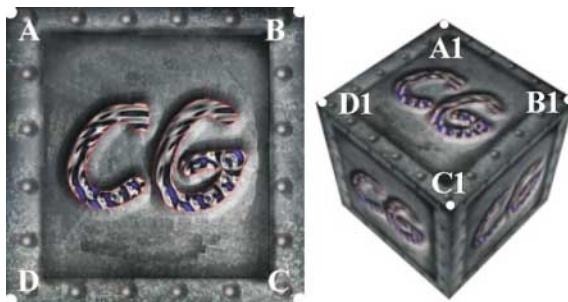


FIGURA 7.65. Relação entre os vértices para aplicar texturas em OpenGL.

```
// Define a textura corrente
glBindTexture (GL_TEXTURE_2D, texture_id[0]);

// associa cada vértice do polígono a um ponto da textura
glTexCoord2f(1.0f, 0.0f); glVertex3f(1.0f, -1.0f, -1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f(1.0f, 1.0f, -1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(1.0f, 1.0f, 1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f(1.0f, -1.0f, 1.0f);
```

7.6.5.3. Mip-Mapping

A OpenGL e grande parte das placas de vídeo suportam texturas com múltiplas imagens. A técnica *mipmapping* seleciona a imagem de textura para um polígono mais próximo da resolução da tela. A carga da texturas *mipmapped* leva um pouco mais de tempo do que uma textura padrão, mas os resultados visuais são melhores. Além disso, as texturas *mipmapped* podem melhorar a performance de exibição reduzindo a necessidade de filtros de imagem GL_LINEAR.

As texturas *mipmapped* são definidas fornecendo um parâmetro específico de nível para cada imagem. Os níveis de imagem são especificados no primeiro parâmetro da chamada `glTexImage1D()`. A imagem nível 0 é a primária, com a maior resolução para a textura. A imagem nível 1 é metade do tamanho da imagem primária e assim por diante. Quando polígonos com uma textura *mipmapped* são desenhados, é necessário o uso de filtros redutores (GL_TEXTURE_MIN_FILTER), veja a seguinte tabela:

| Filtro | Descrição |
|---------------------------|--|
| GL_NEAREST_MIPMAP_NEAREST | Usa a imagem mais próxima da resolução da tela (polígono). Usa o filtro GL_NEAREST quando faz a texturização com essa imagem. |
| GL_NEAREST_MIPMAP_LINEAR | Usa a imagem mais próxima da resolução da tela (polígono). Usa o filtro GL_LINEAR quando faz a texturização com essa imagem. |
| GL_LINEAR_MIPMAP_NEAREST | Interpola linearmente entre duas imagens mais próximas da resolução da tela (polígono). Usa o filtro GL_NEAREST quando faz a texturização com essa imagem. |
| GL_LINEAR_MIPMAP_LINEAR | Interpola linearmente entre duas imagens mais próximas da resolução da tela (polígono). Usa o filtro GL_LINEAR quando faz a texturização com essa imagem. |

7.6.6. Fog (névoa)

O uso de névoa no OpenGL é bem simples, bastando ativar através do comando `glEnable (GL_FOG)`. Os parâmetros de densidade, início e fim são especificados com o comando:

```
glFog(GLenum pname, TYPE param);
```

Exemplo:

```
glEnable(GL_FOG);  
glFogi(GL_FOG_MODE, GL_LINEAR); // especifica o fator da névoa, podendo ser  
                                // também GL_EXP (padrão) ou GL_EXP2  
glFogi(GL_FOG_INDEX, 32);      // especifica a cor  
glFogf(GL_FOG_START, 1.0);     // especifica onde inicia a névoa  
glFogf(GL_FOG_END, 4.0);       // especifica onde termina a névoa
```


Índice

Ábaco
aceleração
AIDS
aliasing
alpha
amortecimento
anatômicas
animatic
anisotrópico
anti-aliasing
antropomórfica
aramada
Arquitetura
Arte
astronomia
auto-similaridade
axiomas
axométricas
bastonetes
Bernstein
Bézier
bicúbica
bilinear
biomecânica
bípede
blur
booleanas
B-rep
Bresenham
BSP
B-Spline
buffer
bump
cabinet
câmeras
captura
cartoon
Catmull

cavaleira
cena
ciclo
cinemática
cinética
cisalhamento
colisão
colorimetria
cones
cônicas
coordenadas
criaturas
CRT
culling
DaVinci
decomposição
deficiência
deformáveis
Descarte
desenho
designers
difusa
dimensões
dimétricas
dinâmicos
Disney
disparidades
DNA
DOFs
educação
efeitos
elástico
eletro-mecânicos
eletrostática
elipsóide
emissores
engenharia
escala

espelhamento
esqueleto
estéreo
estereoscópicas
estímulo
estocástico
Euclides
Euler
extrusão
fantasia
fenômenos
FK
flexíveis
flexores
flipbooks
fluidos
fogo
fonema
fotografia
fotométrico
foto-realismo
fractais
fronteira
fuzzy
genus
Gouraud
gradiente
Hermite
highlight
homogêneas
HSR
IK
inbetweening
incandescentes
indústria
inércia
interação
interface

Interpolações
invisível
isométrica
jogos
keyframe
kinematics
lâmpadas
laser
lentes
malha
Mandelbrot
manifold
Mäntylä
marcadores
mecânica
medicina
memória
meshSmooth
meteorologia
mola
NASA
natureza
Newton
NURBS
NURMS
observador
oclusão
octante
octree
olho
opacidade
óticos
paisagismo
paramétrica
partícula
patches
pêlos
percepção
perspectiva
Phong
Pitágoras
pivô

pixel
poliedros
ponteiros
popularização
Pose-to-Pose
primitivas
procedural
programadores
projeção
projeto
quadrúpedes
quadtree
QWERTY
radiosidade
raio
raster
rasterização
ray-tracing
Realidade Virtual
realismo
realizáveis
real-time
reciclagem
reconstituição
recorte
Reeves
reflexão
refração
resistência
resolução
respiração
reticulado
retina
Riemann
rigidez
robótica
rotação
rotoscopia
ruidos
RV
satélite
scanline

scanners
script
sensores
serrilhamento
shading
shearing
SIGGRAPH
simulações
sincronização
síntese
sintética
slices
solar
sólidos
sombra
sombra
sombreamento.
sons
sotaques
stacks
Stereoscopic
storyboard
suavização
subdivisão
sweeping
tecidos
tempo-real
topologia
track
trajetória
transparência
triangulação
tricromáticas
trilineares
unicidade
varredura
visão
viscosos
voxels
Warps
Winged-Edge
wireframe
Z-Buffer

TEORIA DA COMPUTAÇÃO GRÁFICA



Preencha a **ficha de cadastro** no final deste livro
e receba gratuitamente informações
sobre os lançamentos e as promoções da
Editora Campus.

Consulte também nosso catálogo
completo e últimos lançamentos em
www.campus.com.br

TEORIA DA COMPUTAÇÃO GRÁFICA

A todos que foram meus alunos nos mais diversos cursos nestes anos de magistério principalmente aos mais curiosos e indagadores

Dedico aos meus pais, que sempre me ensinaram a buscar o valor da inteligência, no trabalho e no que é importante além das superficialidades

Aura Conci

A computação gráfica salva vidas, vê a vida de outras formas, acha novas formas de vida, recria a vida e emociona. Mas ela pode iludir, pode manipular e destruir. Este livro é para as pessoas que querem imaginar e criar.

Dedicado aos meus avós, o escritor Dióscoro Campos e o professor Ruy Azevedo.
Aos meus pais Mara Campos e Ruy Azevedo.

Agradecimentos aos meus tios, primos e amigos, ao Prof. Ferraz e equipe, Prof^a Marisa Ortegoza, Prof^a Paula Maciel, Sebastião Lago Jr e equipe, Ronaldo Gueraldi, irmãos Bonnet, Fabio Policarpo/Francisco Meirelles e equipe, Sergio Canabrava, Kimura Lee e Ana Cecília Azevedo.

Eduardo Azevedo

Em memória de tios, avós e amigos (jovens vítimas da violência).

Sumário

CAPÍTULO 1

| | |
|--|----------|
| VISÃO GERAL | 1 |
| 1.1. Computação Gráfica, Arte e Matemática | 3 |
| 1.2. Origens da Computação Gráfica | 4 |
| 1.2.1. Escala Temporal | 5 |
| 1.3. Áreas da Computação Gráfica | 8 |
| 1.4. O Mercado da Computação Gráfica | 8 |
| 1.5. Percepção Tridimensional | 10 |
| 1.5.1. Informações monoculares | 11 |
| 1.5.1.1. Perspectiva | 11 |
| 1.5.1.2. Conhecimento prévio do objeto | 11 |
| 1.5.1.3. Oclusão | 12 |
| 1.5.1.4. Densidade das texturas | 12 |
| 1.5.1.5. Variação da reflexão da luz | 13 |
| 1.5.1.6. Sombras e sombreamentos | 13 |
| 1.5.2. Informações visuais óculo-motoras | 13 |
| 1.5.2.1. Acomodação | 13 |
| 1.5.2.2. Convergência | 13 |
| 1.5.3. Informações visuais estereoscópicas | 13 |
| 1.6. Representação Vetorial e Matricial de Imagens | 14 |
| 1.7. Arquitetura de Hardware | 15 |
| 1.7.1. Dispositivos Gráficos de Entrada | 16 |
| Dispositivos de entrada 3D | 17 |
| 1.7.2. Dispositivos Gráficos de Saída | 18 |
| 1.8. OpenGL | 25 |
| Integração do OpenGL com Windows95/98/2000/XP | 27 |
| O Primeiro Programa | 28 |
| Sintaxe de Comando | 29 |

CAPÍTULO 2

| | |
|--|-----------|
| TRANSFORMAÇÕES GEOMÉTRICAS NO PLANO E NO ESPAÇO | 31 |
| 2.1. Matrizes em Computação Gráfica | 33 |

| | |
|--|----|
| 2.2. Pontos, Vetores e Matrizes | 33 |
| 2.3 Aritmética de Vetores e Matrizes | 34 |
| 2.4 Sistemas de Coordenadas | 36 |
| 2.4.1 Sistema de Referência do Universo (SRU) | 37 |
| 2.4.2 Sistema de Referência do Objeto (SRO) | 37 |
| 2.4.3 Sistema de Referência Normalizado (SRN) | 37 |
| 2.4.4 Sistema de Referência do Dispositivo (SRD) | 38 |
| 2.4.5 Transformações entre Sistemas de Coordenadas | 38 |
| 2.5. Transformações em Pontos e Objetos | 38 |
| 2.5.1. Transformação de Translação | 38 |
| 2.5.2. Transformação de Escala | 40 |
| 2.5.3. Transformação de Rotação | 41 |
| 2.5.4. Transformação de Reflexão, Espelhamento ou Flip | 47 |
| 2.5.5. Transformação de Cisalhamento | 49 |
| 2.6. Coordenadas Homogêneas | 50 |
| 2.7. Projeções Geométricas | 52 |
| 2.7.1. Classificação das Projeções Geométricas | 53 |
| 2.7.2 Projeções Paralelas Ortográficas | 56 |
| 2.7.3 Projeções Paralelas Axométricas | 58 |
| 2.7.4. Projeção Perspectiva ou Cônica | 59 |
| 2.8. Especificação dos pontos de fuga | 63 |
| 2.9. Câmera Virtual | 64 |
| 2.10. Transformações Geométricas com OpenGL | 66 |
| 2.10.1. Transformação de Translação | 66 |
| 2.10.2. Transformação de Escala | 66 |
| 2.10.3. Transformação de Rotação | 66 |
| 2.10.4. Matrizes de transformação | 67 |
| 2.10.5. Armazenando as transformações na matriz | 67 |
| 2.10.6. Alterando a matriz de transformação | 68 |
| 2.10.7. Montando transformações hierárquicas | 68 |
| 2.10.8. Desfazendo o vínculo de hierarquia | 69 |
| 2.10.9. Matriz genérica de projeção | 69 |
| 2.10.10. Projeção paralela ortogonal | 69 |
| 2.10.11. Projeção em perspectiva | 70 |
| 2.10.12. Posição da câmera | 70 |

CAPÍTULO 3

CURVAS E SUPERFÍCIES **73**

| | |
|--|----|
| 3.1. Representação de Curvas | 75 |
| 3.1.1. Conjunto de Pontos | 75 |
| 3.1.2. Representação Analítica | 76 |
| 3.1.3. Formas não-Paramétricas de Representar Curvas | 77 |
| 3.1.4. Formas Paramétricas de Representar Curvas | 79 |
| 3.1.5. Curvas Paramétricas de Terceira Ordem | 81 |

| | |
|--|-----|
| 3.1.6. Hermite | 81 |
| 3.1.7. Bézier | 87 |
| 3.1.8. Splines | 92 |
| 3.1.8.1. Splines Uniformes e Periódicas | 94 |
| 3.1.8.2. Splines Não-Periódicas | 94 |
| 3.1.8.3. Splines Não-Uniformes | 95 |
| 3.1.8.4. Desenvolvimento da Formulação Genérica de B-Splines | 96 |
| 3.1.8.5. Catmull-Rom Splines | 97 |
| 3.1.9. Curvas Racionais | 99 |
| 3.2. Superfícies | 101 |
| 3.2.1. Superfícies de Revolução | 101 |
| 3.2.2. Superfícies Geradas por Deslocamento | 102 |
| 3.2.3. Superfícies Geradas por Interpolação Bi-linear | 103 |
| 3.2.4. Interpolações Trilineares | 105 |
| 3.2.5. Superfícies de Formas Livres | 106 |
| 3.2.6. Superfícies Paramétricas Bicúbicas | 106 |
| 3.2.7. Superfícies de Hermite | 107 |
| 3.2.8. Superfícies de Bézier | 107 |
| 3.2.9. Superfícies de B-Spline | 108 |
| 3.2.10. Normais a Superfícies | 109 |
| 3.2.11. Superfícies Racionais | 110 |
| 3.2.12. NURBS | 110 |
| 3.2.12.1. NURBS em OpenGL | 113 |
| 3.2.13. Subdivisão de Superfícies com Catmull-Clark e NURMS | 114 |

CAPÍTULO 4

REPRESENTAÇÃO E MODELAGEM **119**

| | |
|--|-----|
| 4.1. Pivô | 121 |
| 4.2. Sólidos uni, bi e tridimensionais | 122 |
| 4.3. Sólidos realizáveis | 123 |
| 4.4. Formas de representação de objetos | 124 |
| 4.4.1. Representação Aramada (Wire Frame) | 125 |
| 4.4.2. Representação por Faces (ou Superfícies Limitantes) | 125 |
| 4.4.3. Representação por Faces Poligonais | 127 |
| 4.4.4. Fórmula de Euler | 129 |
| 4.4.5. Operadores de Euler | 130 |
| 4.4.6. Estrutura de dados baseada em vértices | 131 |
| 4.4.7. Estrutura de dados baseada em arestas | 132 |
| 4.4.8. Estrutura de dados Winged-Edge e Half Winged-Edge | 133 |
| 4.4.9. Utilização de Operações Booleanas com B-rep. | 134 |
| 4.4.10. Representação por Enumeração da Ocupação Espacial | 135 |
| 4.4.11. Representação por Decomposição do Espaço em Octrees | 136 |
| 4.4.12. Notação Linear | 137 |
| 4.4.13. Representação por Decomposição do Espaço em Quadrees | 137 |

| | |
|--|-----|
| 4.4.14. Quadtrees e Octrees Lineares | 137 |
| 4.4.15. Quadtrees e Octrees híbridas | 139 |
| 4.4.16. Representação por Partição Binária do Espaço (Binary Space-Partitioning – BSP) | 139 |
| 4.4.17. Representação Implícita | 140 |
| 4.5. Técnicas de Modelagem Geométrica | 140 |
| 4.5.1. Instanciamento de Primitivas | 141 |
| 4.5.2. Objetos de Combinação | 142 |
| 4.5.3. Operações Booleanas e Booleanas Regularizadas | 142 |
| 4.5.4. Geometria Sólida Construtiva (CSG-Constructive Solid Geometry) | 144 |
| 4.5.5. Connect | 144 |
| 4.5.6. Modelagem por Varredura (Sweep) ou Deslizamento | 145 |
| 4.5.6.1. Varredura Translacional (Extrusão) | 145 |
| 4.5.6.2. Varredura Rotacional | 146 |
| 4.5.7. Modelagem por Seções Transversais | 146 |
| 4.5.8. Modelagem por Superfícies | 146 |
| 4.6. Modificadores | 147 |
| 4.6.1. Bend ou Curvatura | 147 |
| 4.6.2. Lattice ou Malha | 148 |
| 4.6.3. MeshSmooth ou Suavização | 148 |
| 4.6.4. Optimize | 149 |
| 4.6.5. Bevel | 149 |
| 4.6.6. Melt | 149 |
| 4.6.7. Skew | 150 |
| 4.6.8. Squeeze e Stretch | 151 |
| 4.6.9. Taper | 151 |
| 4.6.10. Twist | 152 |
| 4.6.11. Displace ou Deslocamento | 152 |
| 4.6.12. Space Warps | 152 |
| 4.7. Interfaces para Modelagem Geométrica | 153 |
| 4.7.1. Tape (Fita de Medida) | 153 |
| 4.7.2. Protractor (Transferidor) | 153 |
| 4.7.3. Compass (Bússola) | 153 |
| 4.7.4. Array (Vetor) | 153 |
| 4.8. Modelagem Geométrica com OpenGL | 154 |
| 4.8.1. Desenhando um Ponto | 154 |
| 4.8.2. Desenhando uma Linha | 154 |
| 4.8.3. Desenhando um Polígono | 155 |
| 4.8.4. Primitivas | 155 |
| 4.8.5. Objetos Sólidos | 155 |
| 4.8.6. Wireframe | 156 |
| 4.9. Modelagem pelo Número de Ouro | 156 |
| 4.9.1. A Seqüência de Fibonacci | 158 |
| 4.10. Modelagem Fractal | 159 |
| 4.10.1. Os Objetos de Mandelbrot | 162 |

| | |
|--|-----|
| 4.11. Shape from X – Reconstrução Tridimensional | 163 |
| 4.11.1. Implementação | 169 |
| 4.12. Sistemas de Partículas | 171 |
| 4.12.1. Distribuição de partículas no sistema | 173 |
| 4.12.2. Noção de centro de massa | 174 |
| 4.12.3. Velocidade do centro de massa | 174 |
| 4.12.4. Quantidade de movimento | 174 |
| 4.12.5. Aceleração do centro de massa | 175 |
| 4.12.6. Movimento de uma partícula em relação ao seu centro de massa | 176 |
| 4.12.7. Choque | 176 |
| 4.12.8. Quantidade de movimento de um sistema de duas partículas | 177 |
| 4.12.9. Energia de um sistema de duas partículas | 177 |
| 4.12.10. Conceito de Sistemas de Partículas | 177 |
| Posição | 179 |
| Velocidade | 179 |
| Cor | 179 |
| Transparência | 179 |
| Tamanho | 179 |
| Tempo de Vida | 180 |
| 4.12.11. Renderizando as Partículas | 180 |
| 4.12.12. Características dos Sistemas de Partículas | 181 |

CAPÍTULO 5

CORES **183**

| | |
|--|-----|
| 5.1. Sistema Visual Humano | 185 |
| 5.2. Descrição da Cor da Luz | 189 |
| 5.3. As Ondas Eletromagnéticas | 191 |
| 5.4. Sistemas de Cores Primárias | 193 |
| 5.5. Sistemas de Cores Aditivas | 195 |
| 5.6. Sistemas de Cores Subtrativas | 196 |
| 5.7. Famílias de Espaços de Cor | 197 |
| 5.7.1. Modelo fisiológico | 197 |
| 5.7.2. Modelo baseado em medidas físicas | 197 |
| 5.7.3. Modelo de sensações oponentes | 198 |
| 5.7.4. Modelo psicofísico | 198 |
| 5.8. O Modelo RGB | 198 |
| 5.9. O Modelo CMYK | 199 |
| 5.10. Espaço de percepção subjetiva | 199 |
| 5.11. O modelo HSV | 201 |
| 5.12. O Modelo HSL | 202 |
| Vantagens dos modelos HSV e HLS [JACK 94] | 203 |
| Desvantagens dos modelos HSV e HLS [JACK 94] | 203 |
| 5.13. O Modelo YIQ | 203 |
| 5.14. Transformação dos Espaços de Cor | 204 |

| | |
|---|-----|
| 5.14.1. De RGB para XYZ | 204 |
| 5.14.2. De RGB para Eixos-Oponentes (OPP) | 206 |
| 5.14.3. De RGB para HSV | 206 |
| 5.15. Uso de Cores nas Imagens | 207 |
| 5.16. Descrição das Cores | 208 |
| Branco | 208 |
| Preto | 208 |
| Cinza | 208 |
| Vermelho | 209 |
| Amarelo | 209 |
| Verde | 209 |
| Azul | 210 |
| 5.17. Histograma de cores | 210 |
| 5.18. Espaço de Cores Oponentes | 211 |
| 5.19. Ilusões Relacionadas às Cores | 213 |
| 5.20. Problemas com Cores na Computação | 213 |
| 5.21. Cores em OpenGL | 214 |
| 5.21.1. Transparência | 214 |

CAPÍTULO 6

ANIMAÇÃO **217**

| | |
|---|-----|
| 6.1. Histórico | 219 |
| 6.2. Aplicações da Animação | 220 |
| 6.3. Animação por Computador | 220 |
| 6.4. Formas de Animação | 221 |
| 6.4.1. Animação por Quadro-Chave (Keyframe) | 221 |
| 6.4.2. Animação por Script | 221 |
| 6.4.3. Animação Procedural | 222 |
| 6.4.4. Animação Representacional | 222 |
| 6.4.5. Animação Estocástica | 223 |
| 6.4.6. Animação Straight Ahead | 223 |
| 6.4.7. Animação Pose-to-Pose | 223 |
| 6.4.8. Animação por Comportamento ou Comportamental | 223 |
| 6.4.9. Animação Track Based | 223 |
| 6.5. Canal Alpha | 223 |
| 6.6. Composição | 224 |
| 6.6.1. Filmagem da Cena Real | 225 |
| 6.6.2. Reconstrução da Cena em 3D | 226 |
| 6.7. Captura de Movimento | 226 |
| 6.7.1. Rotoscopia | 226 |
| 6.7.2. Sistemas de Captura | 227 |
| 6.7.2.1. Ótico | 228 |
| 6.7.2.2. Mecânico | 228 |
| 6.7.2.3. Magnético | 228 |

| | |
|---|-----|
| 6.7.2.4. Acústico | 228 |
| 6.7.3. Cartoon Motion Capture | 228 |
| 6.8. Animação de personagens 3D | 229 |
| 6.8.1. Cinemática | 230 |
| 6.8.1.1. Cinemática Direta | 230 |
| 6.8.1.2. Cinemática Inversa | 232 |
| 6.8.2. Ossos | 232 |
| 6.8.3. Articulações | 233 |
| 6.8.3.1. Junta de Revolução | 233 |
| 6.8.3.2. Junta esférica | 233 |
| 6.8.3.3. Grau de Liberdade | 233 |
| 6.8.4. Esqueleto | 235 |
| 6.8.4.1. Controladores IK | 237 |
| 6.8.4.2. Ciclo de Animação | 238 |
| 6.8.4.3. Sistemas de Animação IK | 239 |
| 6.8.5. Músculo Flexor | 240 |
| 6.8.6. Cabelos e Pêlos | 241 |
| 6.8.7. Animação Facial | 242 |
| 6.8.7.1. Sincronização Labial | 243 |
| 6.8.7.2. Seqüência de Texturas | 243 |
| 6.8.7.3. Morphing | 243 |
| 6.8.7.4. Esqueleto | 243 |
| 6.8.7.5. Free Form Deformation | 243 |
| 6.8.7.6. Weighted Morphing | 243 |
| 6.9. Animação de Superfícies Deformáveis | 244 |
| 6.10. Produção de Animação | 246 |
| 6.11. Princípios da Animação | 247 |
| 6.12. Cena de Animação | 248 |
| 6.13. A Animação no Processo de Aprendizado | 249 |

CAPÍTULO 7

REALISMO VISUAL E ILUMINAÇÃO **251**

| | |
|--|-----|
| 7.1. Rendering | 253 |
| 7.1.1. Fases do Processo de Realismo Visual | 254 |
| 7.1.2. Realismo por Passadas | 255 |
| 7.1.3. Acabamentos não-fotográficos | 257 |
| 7.2. Rasterização | 258 |
| 7.2.1. Algoritmo de Bresenham para traçado de linhas | 260 |
| 7.2.2. Rasterização de Polígonos | 261 |
| 7.2.3. Preenchimento de polígonos por scanline | 262 |
| 7.2.4. Remoção de Linhas e Superfícies Escondidas | 263 |
| 7.2.4.1. Algoritmo de Visibilidade por Prioridade ou Algoritmo do Pintor | 267 |

| | |
|---|-----|
| 7.2.4.2. Algoritmo de Eliminação de Faces Ocultas pelo Cálculo da Normal | 269 |
| 7.2.4.3. Algoritmo Z-Buffer | 272 |
| 7.3. Iluminação | 276 |
| 7.3.1. Tipo de Emissores | 278 |
| 7.3.1.1. Emissor Natural | 278 |
| 7.3.1.2. Luz Ambiente | 278 |
| 7.3.1.3. Emissores de Luz Artificiais | 279 |
| 7.3.2. Reflexões | 280 |
| 7.3.2.1. Reflexão Ambiente | 281 |
| 7.3.2.2. Reflexão Difusa | 283 |
| 7.3.2.3. Reflexão Especular | 285 |
| 7.3.3. Refração | 289 |
| 7.3.4. Transparência | 291 |
| 7.3.5. Sombreamento (Shading) | 292 |
| 7.3.5.1. Modelo de Sombreamento Constante | 292 |
| 7.3.5.2. Sombreamento de Gouraud | 294 |
| 7.3.5.3. O Modelo de Phong | 296 |
| 7.3.6. Sombras | 298 |
| 7.3.6.1. Volume de Sombra | 298 |
| 7.3.6.2. Sombra Projetada | 298 |
| 7.3.6.3. Algoritmo de Atherton-Weiler-Greeberg | 299 |
| 7.3.6.4. Algoritmo de Mapeamento | 299 |
| 7.3.6.5. Algoritmo de Appel | 299 |
| 7.3.6.6. Sombreamento Anisotrópico | 299 |
| 7.3.7. Modelo de Iluminação Global | 300 |
| 7.3.7.1. Ray Tracing | 301 |
| Reflexos | 310 |
| Refrações | 310 |
| Metais com Ray tracing | 311 |
| 7.3.7.2. Caustic | 311 |
| 7.3.7.3. Radiosidade | 313 |
| 7.3.8. Técnicas de Iluminação | 317 |
| 7.3.8.1. Luz Principal (Key Light) | 318 |
| 7.3.8.2. Luz de Preenchimento (Fill Light) | 318 |
| 7.3.8.3. Luz de Recuo (Back Light) | 319 |
| 7.4. Texturas | 319 |
| 7.4.1. Mapas Procedurais | 320 |
| 7.4.2. UVW Map | 320 |
| 7.4.3. Texture Map | 321 |
| 7.4.4. Environment Mapping ou Mapa de Reflexão | 322 |
| 7.4.5. Bump Map | 323 |
| 7.4.5.1. Emboss Bump Map | 324 |
| 7.4.5.2. Environment Mapped Bump Mapping | 325 |
| 7.4.5.3. Per-Pixel Shading | 325 |

| | |
|--|-----|
| 7.4.6. Light Map | 326 |
| 7.4.7. Mip-Mapping | 328 |
| 7.5. Hiper-Realismo | 329 |
| 7.5.1. High Dynamic Range Images (HDRI) | 330 |
| 7.5.2. Atenuação Atmosférica | 330 |
| 7.5.3. Area Light e Soft Shadow | 331 |
| 7.5.4. Sub-Surface Light Scattering | 331 |
| 7.5.5. Depth of Field (DOF) ou Profundidade de Campo | 333 |
| 7.5.6. Motion Blur / Desfoque por Movimento | 333 |
| 7.5.7. Film Grain | 333 |
| 7.5.8. Lens Flare | 334 |
| 7.5.9. Glow | 334 |
| 7.6. Realismo e Iluminação em OpenGL | 334 |
| 7.6.1. Z-buffer | 334 |
| 7.6.2. Hidden-Surface Removal | 334 |
| 7.6.3. Algoritmo de Recorte (Culling) | 335 |
| 7.6.4. Iluminação | 335 |
| 7.6.4.1. Modelo de Tonalização | 335 |
| 7.6.4.2. Propriedades de Iluminação do Material | 335 |
| 7.6.4.3. Modelo de Iluminação | 336 |
| 7.6.4.4. Fonte de Luz | 337 |
| Programa Exemplo de Iluminação | 337 |
| 7.6.5. Texturas | 338 |
| 7.6.5.1. Carga de uma textura | 338 |
| 7.6.5.2. Aplicando uma Textura | 339 |
| 7.6.5.3. Mip-Mapping | 340 |
| 7.6.6. FOG (névoa) | 341 |

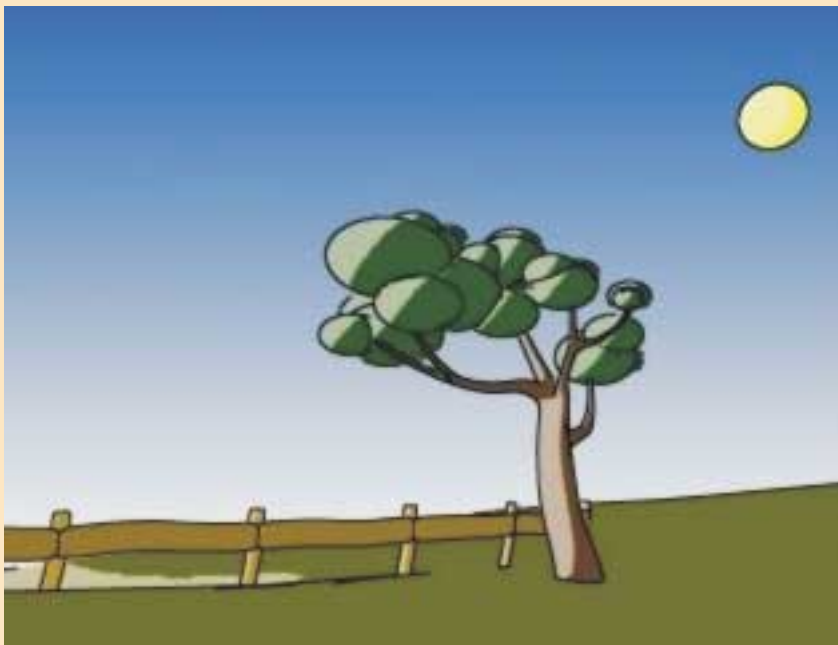


FIGURA 1 *Render NPR, Toon Shading.*



FIGURA 2 *Render NPR, simulação das pinceladas de pinturas.*



FIGURA 3 *Detalhe do sistema de elevação da escavadeira.*



FIGURA 4 *Água e nuvens criadas com fractais.*



FIGURA 5 *Sistema de Partículas simulando água em colisão com as pedras.*

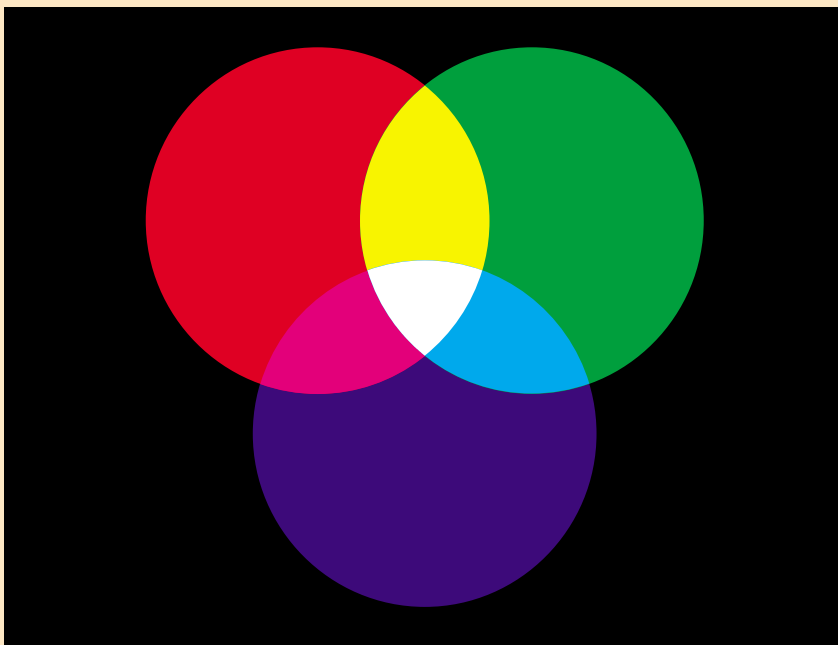


FIGURA 6 *Processo aditivo das cores primárias.*

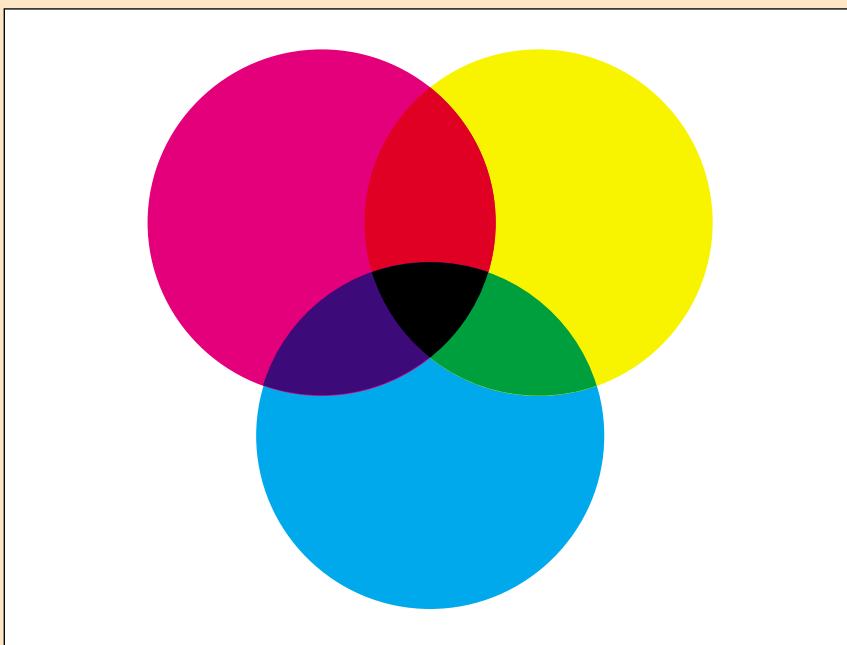


FIGURA 7 *Processo subtrativo das cores secundárias.*

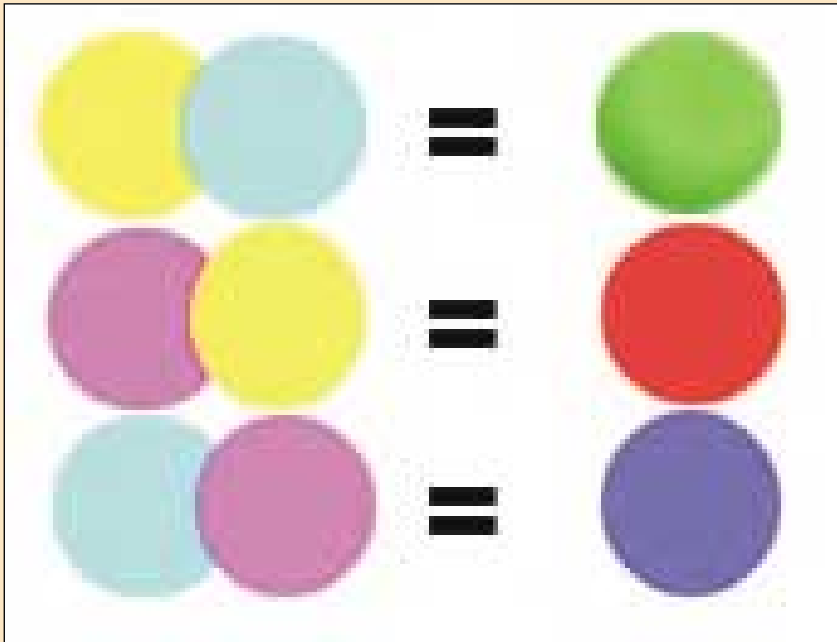


FIGURA 8 *Processo de obtenção das cores secundárias.*

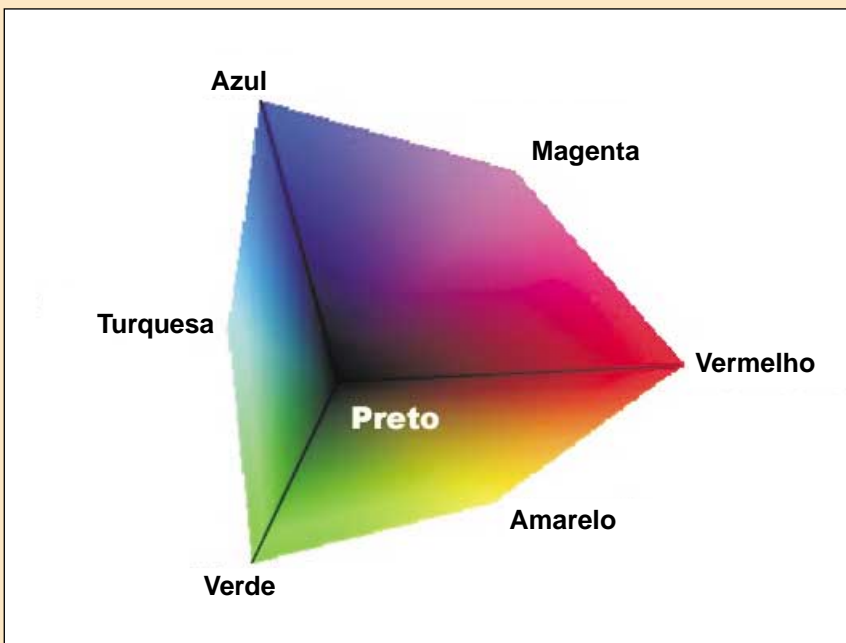


FIGURA 9 *Subespaço do modelo RGB.*



FIGURA 10 O sistema de Munsell.

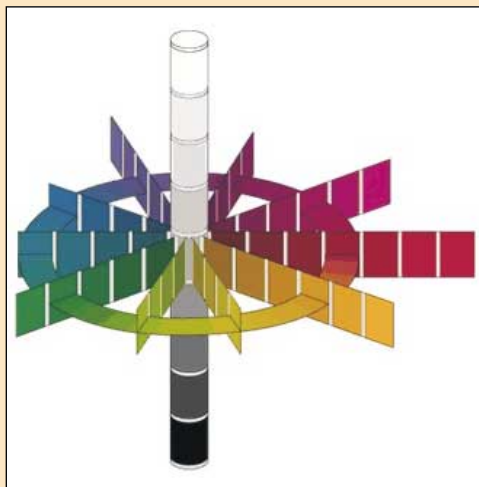


FIGURA 11 O espaço de Munsell

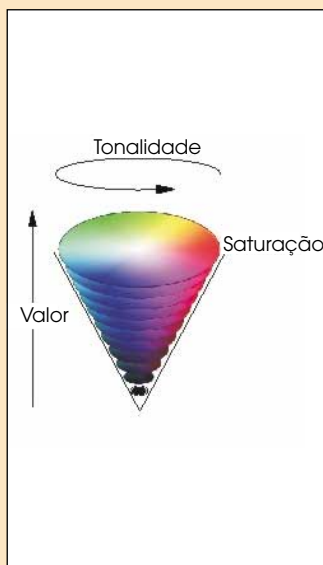


FIGURA 12 O modelo HSV.

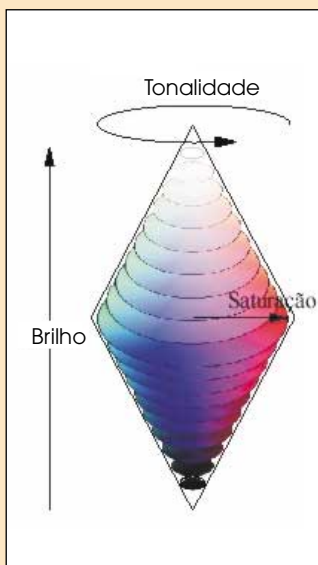


FIGURA 13 O modelo HLS.

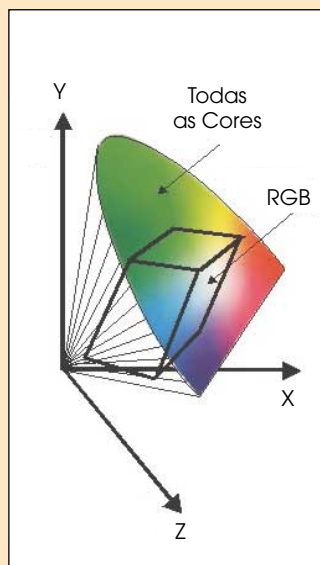


FIGURA 14 Distribuição no espaço XYZ do sistema RGB.



FIGURA 15 *Cena com adição de texturas.*



FIGURA 16 *Cena com adição da iluminação e reflexões.*



FIGURA 17 *Radiosidade e sua variação de sombreamento gradual.*



FIGURA 18 *Light Map de uma fase da engrenagem Fly3D (Cortesia da Paralelo Computação).*
