Introdução à Computação Gráfica

Isabel Harb Manssour [1]
Marcelo Cohen [1]

Resumo: Computação Gráfica é a área da Ciência da Computação que estuda a geração, manipulação e análise de imagens, através do computador. Atualmente, é uma das áreas de maior expansão e importância que propicia o desenvolvimento de trabalhos multidisciplinares. A elaboração de aplicações gráficas se popularizou com o surgimento de placas gráficas para computadores pessoais e de bibliotecas gráficas, como OpenGL, que não exigiam conhecimentos extensivos de programação ou de hardware. Este tutorial procura abranger os principais conceitos e definições da Computação Gráfica, abordando técnicas de realismo, animação e visualização. Exemplos de aplicações gráficas são apresentados como motivação para o estudo e pesquisa nesta área. Todos os passos do processo de visualização são descritos de forma objetiva através da apresentação de imagens. Questões em aberto e tendências para o desenvolvimento de aplicações gráficas também são abordadas.

Abstract: Computer Graphics is the Computer Science area which studies the creation, manipulation and analysis of images through the computer. Currently it is an important and ever growing area, as it allows the development of multidisciplinary work. The development of graphics applications has become widely popular, partly due to the availability of graphics hardware for personal computers and partly to the creation of graphics libraries, such as OpenGL, which does not require extensive programming or hardware knowledge. This tutorial aims to present a comprehensive view of the Computer Graphics field, reviewing the main techniques for visualization, animation and rendering of realistic imagery. Examples of graphical applications are presented as motivation for the study and research in this area. The visualization pipeline steps are described through the presentation of images. Open issues and trends for the the development of graphics applications are also discussed.

1 Introdução

A Computação Gráfica (CG) é uma área da Ciência da Computação que se dedica ao estudo e desenvolvimento de técnicas e algoritmos para a geração (síntese) de imagens através do computador. Atualmente, a CG está presente em quase todas as áreas do conhecimento humano, desde o projeto de um novo modelo de automóvel até o desenvolvimento de ferramentas de entretenimento, entre as quais os jogos eletrônicos.

[1]Faculdade de Informática, PUCRS, Av. Ipiranga 6681, Prédio 30
{manssour,flash@inf.pucrs.br}

Atualmente, com as facilidades disponíveis nas bibliotecas gráficas existentes, a programação das aplicações está mais simples. Por exemplo, OpenGL (*Open Graphics Library*), também definida como uma "interface para hardware gráfico", é uma biblioteca de rotinas gráficas e de modelagem, bidimensional (2D) e tridimensional (3D), portável e rápida [26, 21, 3]. Ela permite desenvolver aplicações interativas e gerar imagens de cenas 3D (ou conjunto de objetos), com um alto grau de realismo. Entretanto, a sua maior vantagem é a velocidade, uma vez que incorpora vários algoritmos otimizados, incluindo o desenho de primitivas gráficas, o mapeamento de textura e outros efeitos especiais.

Este artigo procura abranger os principais conceitos e definições da CG, utilizando imagens para auxiliar no seu entendimento. Um breve histórico da área é apresentado na Seção 1.1. Alguns exemplos de aplicações e de hardware gráfico são apresentados, respectivamente, na Seção 1.2 e Seção 1.3. Já a Seção 1.4 descreve duas áreas relacionadas à CG. Uma introdução à modelagem geométrica é feita na Seção 2 e o processo de visualização 3D é descrito na Seção 3. Finalmente, técnicas de realismo, animação e visualização são abordadas na Seção 4, e alguns comentários finais são feitos na Seção 7.

1.1 Breve Histórico

Antes de apresentar o histórico da CG, deve-se considerar que o desenvolvimento de aplicações gráficas depende do hardware. A criação e evolução dos dispositivos gráficos possibilitaram os avanços na área de CG. Neste sentido, um marco importante foi o desenvolvimento do primeiro computador a possuir recursos gráficos de visualização de dados numéricos, o *Whirlwind*, pelo MIT (*Massachusetts Institute of Technology*) na década de 50. O primeiro sistema, o SAGE (*Semi-Automatic Ground Environment*), surgiu em 1955 para converter informações do radar em imagens para monitoramento e controle de vôos. O conceito de CG interativa da forma que conhecemos atualmente, foi desenvolvido pelo Dr. Ivan Sutherland na sua tese de doutorado no início da década de 60. Ele introduziu conceitos de estruturação de dados e CG interativa, despertando o interesse das indústrias automobilísticas e aeroespaciais, levando a GM a desenvolver o precursor dos sistemas CAD em 1965 [8].

Na década de 70 foram desenvolvidas novas técnicas e algoritmos que são utilizados até hoje, tal como o algoritmo de *z-buffer*. Além disso, o surgimento da tecnologia de circuitos integrados permitiu a popularização dos computadores pessoais, disseminando os aplicativos prontos e integrados, como os editores gráficos. Também foi nesta década o lançamento do primeiro computador com interface visual, predecessor do Macintosh. Posteriormente, houve o surgimento e a popularização dos dispositivos para interação 3D usados em RV. A popularização das placas aceleradoras gráficas contribuiu para o crescimento da capacidade dos PCs, permitindo a geração de imagens com grande realismo em tempo real.

1.2 Aplicações

Atualmente a CG está presente em quase todas as áreas do conhecimento humano, da engenharia que utiliza as tradicionais ferramentas CAD (*Computer-Aided Design*), até a medicina que trabalha com modernas técnicas de visualização para auxiliar o diagnóstico por imagens. Nesta área, também têm sido desenvolvidos sistemas de simulação para auxiliar no treinamento de cirurgias endoscópicas. Outros tipos de simuladores são usados para treinamento de pilotos e para auxiliar na tomada de decisões na área do direito (por exemplo, para reconstituir a cena de um crime).

Uma área de aplicação da CG que tem crescido muito nos últimos anos é a visualização. Inicialmente, surgiu a visualização científica, que visava o desenvolvimento de representações gráficas para grandes volumes de dados gerados, por exemplo, por satélites e equipamentos de sensoriamento remoto. Hoje, existe também a visualização volumétrica e a visualização de informações. A primeira trata apenas de como gerar representações para dados volumétricos, tais como dados meteorológicos, oceanográficos e médicos. Já a segunda, é uma área que visa auxiliar na análise de dados financeiros, mineração de dados, estudos do mercado ou de gerenciamento de redes de computadores.

Animação é outro exemplo de aplicação que tem sido bastante desenvolvida. Através da exibição de imagens em seqüência é possível representar o comportamento de objetos reais ou simulados. Além de desenho animados por computador, a simulação de humanos virtuais e de comportamento de multidões são temas de pesquisas atuais. Técnicas de CG também são usadas em RV, que introduziu um novo paradigma de interface com o usuário. Através de dispositivos especiais que captam movimentos do corpo do usuário é possível interagir em diferentes ambientes projetados por computador. Simuladores para treinamento de pessoal, tratamento de fobias e jogos são alguns exemplos de aplicações de RV. Outros exemplos de aplicações podem ser encontrados na literatura [10, 1].

1.3 Hardware Gráfico

Antigamente os computadores e impressoras só eram capazes de emitir resultados sob a forma de listagens alfanuméricas, de maneira que as imagens eram obtidas pela composição de símbolos. Durante os anos 50 e 60 foram projetadas as primeiras configurações de sistemas gráficos que possuíam um novo conceito em visualização: em vez de caracteres, passou a ser necessário administrar os pontos individuais da tela ou *pixels* (*picture elements*). Assim, os programas passaram a contar com a possibilidade de apresentar saídas na forma gráfica.

Os dispositivos gráficos podem ser classificados quanto à finalidade em dispositivos de entrada, de saída ou de entrada e saída, e quanto ao formato dos dados em dispositivos matriciais ou vetoriais [12]. Dispositivos matriciais são aqueles cujos dados são capturados e/ou exibidos na forma de matrizes, ou seja, um conjunto de *pixels*. O digitalizador de vídeo

e o *scanner* são exemplos de dispositivos matriciais de entrada. Pode-se dizer que a grande maioria dos dispositivos de saída são matriciais, tais como impressoras, *stereo glasses*, *caves* e monitores de vídeo.

Já dispositivos vetoriais incluem os equipamentos através dos quais se pode coletar e/ou exibir dados de forma isolada, associados a uma posição do plano/espaço. Entre os dispositivos vetoriais de entrada destacam-se o *mouse* (2D ou 3D), o digitalizador espacial, a luva eletrônica e os dispositivos para rastreamento. O exemplo mais comum de dispositivo vetorial de saída é o plotador gráfico (*plotter*), mas os primeiros monitores de vídeos também eram vetoriais, pois eram capazes apenas de exibir seqüências de linhas programadas.

Finalmente, dispositivos de entrada e saída são aqueles onde há captura de informações e exibição/resposta ao usuário. Por exemplo, dispositivos de resposta tátil (*force feedback*), ou seja, que permitem a captura de movimentos e geram sensações de tato e força, são considerados de entrada e saída. Outro exemplo é o monitor com tela sensível ao toque.

1.4 Áreas Relacionadas

Há duas áreas que têm uma relação bastante próxima com a CG: **Processamento de Imagens** (PI) e **Visão Computacional**. A área PI abrange o estudo e a pesquisa de técnicas para realizar a manipulação de imagens, tais como ajustes de cor, brilho, contraste ou aplicação de filtros, entre outras. Sistemas de PI são encontrados atualmente, por exemplo, em consultórios de cirurgiões plásticos e salões de beleza. Neste caso, uma pessoa pode ver como será o resultado de uma plástica ou corte de cabelo através de simulações no computador. A Visão Computacional trabalha com a análise de imagens, buscando obter a especificação dos seus componentes para identificação dos modelos geométricos que a compõem. Uma aplicação de técnicas de Visão Computacional é o reconhecimento automático de impressões digitais.

2 Modelagem Geométrica

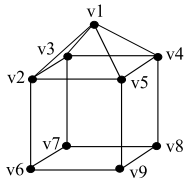
Em CG, modelos são usados para representar entidades e fenômenos do mundo físico real no computador. Existem várias categorias ou métodos de construção de modelos tridimensionais. Cada um tem suas vantagens e desvantagens, adaptando-se melhor para uma ou outra aplicação. Modelagem consiste em todo o processo de descrever um modelo, objeto ou cena, de forma que se possa desenhá-lo. Na verdade, a modelagem engloba dois tópicos de estudo: formas de representação dos objetos (Seção 2.1), que se preocupa com a forma (ou estruturas de dados) como os modelos são armazenados; e técnicas de modelagem dos objetos, que trata das técnicas interativas (e também das interfaces) que podem ser usadas para criar um modelo de objeto.

2.1 Representação

Várias técnicas de representação de objetos 3D estão sendo desenvolvidas em Computação Gráfica, e em algumas delas a estrutura de dados é determinada pela técnica de modelagem. Em geral, a forma de representação determina a estrutura de dados a ser utilizada, o custo do processamento de um objeto através do *pipeline* de visualização 3D, a aparência final de um objeto e a facilidade para alterar a sua forma.

Segundo Watt [24], é possível enumerar quatro formas de representação, de acordo com a importância e freqüência de utilização: (1) malha de polígonos; (2) superfícies paramétricas; (3) Geometria Sólida Construtiva (CSG); (4) enumeração de ocupação espacial. As representações 1 e 4 consistem numa aproximação da forma do objeto que está sendo modelado. A 2 e a 3, por sua vez, são representações exatas. Por outro lado, a 1 e a 2 representam apenas a superfície do objeto, sendo o volume inteiro representado pela 3 e pela 4.

A forma mais comum de representar modelos 3D é através de uma **malha de polígonos**. Ou seja, define-se um conjunto de vértices no espaço (geometria) e como esses vértices devem ser ligados para formarem polígonos fechados, chamados de face (topologia), que podem ser triângulos ou quadrados. O armazenamento desse tipo de estrutura é usualmente realizado através de vetores de estruturas, matrizes ou listas. Por exemplo, a Figura 1 apresenta a lista de vértices e faces necessárias para desenhar uma casa simplificada.



Vértices (geometria)

1	x1 y1 z1
2	x2 y2 z2
3	x3 y3 z3
4	x4 y4 z4
5	x5 y5 z5
6	x6 y6 z6
7	x7 y7 z7
8	x8 y8 z8
9	x9 y9 z9

Faces (topologia)

1	v1 v4 v5
2	v1 v5 v2
3	v1 v2 v3
4	v1 v3 v4
5	v4 v3 v7 v8
6	v5 v4 v8 v9
7	v2 v5 v9 v6
8	v3 v2 v6 v7
9	v6 v9 v8 v7

Figura 1: Exemplo de um objeto representado por uma malha de polígonos.

É aconselhável que todos os modelos sejam sempre definidos de maneira que o seu centro, ou o ponto ao redor do qual ele pode ser rotacionado, seja posicionado na origem $(0,\ 0,\ 0)$. Assim, quando for aplicada uma transformação geométrica (Seção 3.5) sobre o objeto, ele não trocará de quadrante. A este espaço de coordenadas, adequado à criação dos modelos, dá-se o nome de Sistema de Referência do Objeto (SRO). Em outras palavras, o SRO é o sistema de coordenadas no qual se define o modelo geométrico.

2.2 Instanciamento de Primitivas

No instanciamento de primitivas, o sistema de modelagem define um conjunto de objetos primitivos 3D que são relevantes para a área de aplicação. Tais objetos podem ser representados, por exemplo, por malhas de polígonos ou superfícies paramétricas. Depois, estes objetos podem ser parametrizados, tanto em termos de transformações geométricas (Seção 3.5), como em outras propriedades, e agrupados. A Figura 2 apresenta uma mesa formada pelo instanciamento de três cilindros, com diferentes alturas e raios.

Figura 2: Exemplo de uma mesa modelada por instanciamento de primitivas.

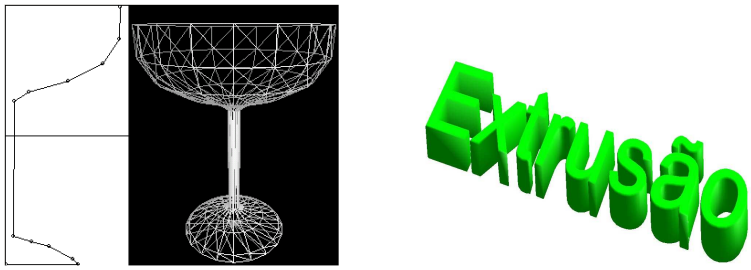
2.3 Varredura Rotacional e Translacional

A modelagem por varredura é útil para a construção de objetos 3D que possuam algum tipo de simetria. De maneira simplificada, através da varredura, um objeto é gerado pelo arrastar de uma curva ou superfície (chamada geradora ou geratriz), que determina a sua forma, ao longo de uma trajetória (diretriz). Os dois tipos mais comuns são: Varredura Rotacional (ou *sweep*), quando a trajetória é um círculo ao redor de um dos eixos; e Varredura Translacional (também chamada de extrusão), quando a trajetória é uma linha. Conforme a geratriz vai sendo arrastada, os pontos que a formam são guardados e conectados, formando uma malha de polígonos. A Figura 3 apresenta dois exemplos de objetos gerados por esta técnica de modelagem.

2.4 Superfícies Paramétricas

Superfícies paramétricas são usadas quando se necessita trabalhar com superfícies suaves na modelagem de objetos de forma livre (*Free Form Objects*). Neste caso, uma representação muito utilizada são os *patches* paramétricos bicúbicos, que permitem calcular as coordenadas de todos os pontos que formam uma superfície curva através da definição de 16 pontos de controle e da utilização de três equações, uma para x, uma para y e uma para z. Cada equação possui duas variáveis (ou parâmetros) e termos para todo domínio dos parâmetros até o seu cubo (daí as expressões bi e cúbico).

Em outras palavras, o *patch* é uma superfície curva na qual cada um dos pontos que a formam deve ser processado. Para isto, inicialmente devem ser definidos 16 pontos 3D, chamados pontos de controle. Quatro destes pontos que determinam a forma do *patch* pertencem aos seus cantos. A partir da especificação dos pontos de controle, são usadas três funções para calcular os valores intermediários que, simplificadamente, são resultantes de uma interpolação. Através de parâmetros passados para as funções, é possível determinar a quantidade de valores intermediários calculados. Além disso, sempre que um ponto de controle é alterado, os pontos que formam a superfície devem ser gerados novamente.



(a) (b)

Figura 3: Exemplo de uma taça gerada por varredura rotacional (a) e uma palavra cujas letras foram geradas por varredura translacional (b).

Diferentes superfícies podem ser geradas alterando as funções utilizadas, tais como Bezier, B-Spline e NURBS. A Figura 4 apresenta dois exemplos de superfícies Bezier, cujos pontos de controle são representados através de pequenos cubos. É possível observar como a quantidade de pontos intermediários influencia na qualidade da superfície. As superfícies Bézier são interessantes nos projetos interativos, pois os pontos de controle podem ser facilmente manipulados para alterar a forma do *patch* da superfície.

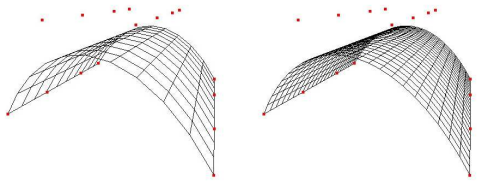


Figura 4: Exemplo de superfície paramétrica Bezier com menos (esquerda) e mais (direita) pontos intermediários.

3 Processo de Visualização 3D

Existe um conjunto de técnicas em CG que permite transformar as informações a respeito de um modelo contidas em uma estrutura de dados, em uma imagem que pode ser exibida em um monitor. Portanto, considera-se que uma imagem consiste em uma matriz de pontos e um modelo é uma representação computacional de um objeto, conforme descrito na Seção 2.1. As seções a seguir apresentam como funcionam e como são abordadas em OpenGL algumas das etapas do processo de visualização 3D para a geração de uma imagem a partir de um modelo ou conjunto de modelos. Para finalizar, todas as etapas do *pipeline* de visualização são apresentadas na Seção 3.8.

3.1 Introdução

Para entender como funciona o processo de visualização é importante conhecer o conceito de **Universo**, que pode ser definido como a região do espaço utilizada em uma aplicação. Como a descrição geométrica de um modelo normalmente envolve coordenadas geométricas, é preciso adotar um sistema de referência que irá definir uma origem em relação à qual todos os posicionamentos do universo são descritos. Em geral, este **Sistema de Referência do Universo** (SRU) consiste em três eixos ortogonais entre si (x, y, z) e uma origem (0, 0, 0). Uma coordenada, então, é formada pelos valores de x, y e z, que correspondem às posições ao longo dos respectivos eixos (denominados cartesianos) e todos os procedimentos são definidos em relação a este sistema de referência.

Para facilmente identificar como o eixo z é posicionado em relação à x e y, normalmente se utiliza a regra da "mão direita" ou a regra da "mão esquerda". Por exemplo, na regra da "mão direita", deve-se posicionar a mão direita de maneira que o indicador aponte para direção positiva de y (para cima), o polegar aponte para a direção positiva de x (para o lado) e o dedo do meio aponte para a direção positiva de z (para frente, como se estivesse "saindo da tela do computador"), como ilustra a Figura 5.

3.2 Câmera Sintética

A primeira etapa do processo de visualização 3D é a definição da cena 3D. Nesta etapa cada um dos objetos que farão parte do mundo 3D é incluído e posicionado no SRU. Este posicionamento é feito através de operações de escala, rotação e translação (Seção 3.5). O próximo passo consiste na especificação do observador virtual, que define de que local se deseja que a cena 3D seja exibida, por exemplo, de cima, do lado direito ou do lado esquerdo. Portanto, a especificação do observador inclui a sua posição e orientação, ou seja, onde ele está e para onde está olhando dentro do universo. A necessidade da existência desse observador deve-se ao fato de que um mesmo conjunto de objetos no universo 3D, visto de

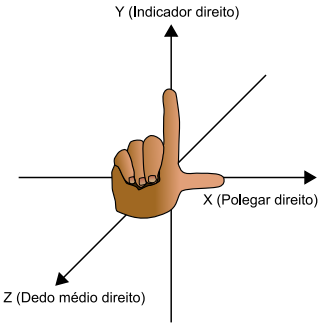
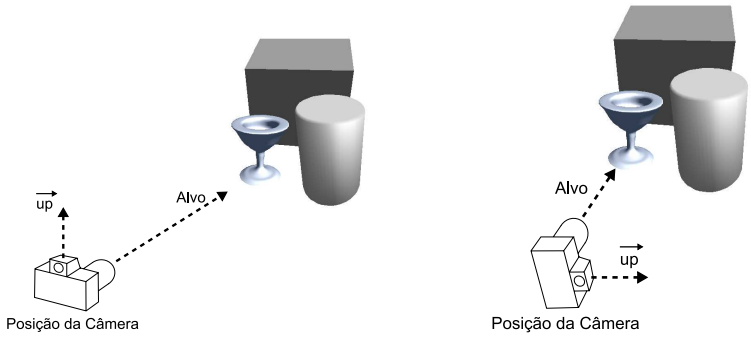


Figura 5: Regra da mão direita para a orientação dos eixos 3D.

diferentes lugares tem, para quem os observa, diferentes coordenadas para cada posição.

Como a imagem gerada a partir da posição e orientação do observador é estática, se faz analogia com uma foto. Pode-se dizer que se obtém uma fotografia quando a câmera está numa determinada posição direcionada para o objeto. A posição da câmera é dada por um ponto (x, y, z) em relação ao mesmo universo no qual os objetos estão posicionados (SRU) e sua orientação é dada por um ponto alvo (x, y, z) e um vetor, aqui chamado de *up*. A Figura 6 ilustra estes conceitos: a câmera é posicionada de duas maneiras diferentes, ocasionando a geração de duas imagens distintas: na Figura 6a, os objetos aparecerão da mesma maneira que estão posicionados; na Figura 6b os objetos aparecerão inclinados 90º para a direita.



(a) Câmera na orientação normal (b) Câmera inclinada

Figura 6: Modelo de câmera sintética.

A partir da posição e orientação da câmera é criado um novo sistema de referência, o chamado Sistema de Referência da Câmara (SRC). O SRC é criado para que seja possível definir a posição de cada objeto em relação ao observador, ou seja, em relação à origem do SRC, pois só assim se pode saber qual é o objeto que está mais próximo ou mais afastado do observador, bem como se um objeto está encobrindo outro.

3.3 Projeções

Quando se está trabalhando com a representação de objetos no espaço tridimensional existe uma etapa obrigatória: mapear suas representações 3D para imagens 2D que serão exibidas em um dispositivo como um monitor. Esta operação de obter representações bidimensionais de objetos tridimensionais é chamada de projeção.

Como a maioria dos objetos é representada por uma coleção de vértices, sua projeção é definida por raios de projeção (segmentos de retas) chamados de projetantes, que passam através de cada vértice do objeto e interseccionam um plano de projeção. Esta classe de projeções é chamada de projeções geométricas planares. As projeções usualmente são divididas em dois tipos principais: **Projeção Paralela Ortográfica**, na qual as projetantes são paralelas entre si, passam pelos vértices dos objetos e interseccionam o plano com um ângulo de 90^o (Figura 7a); **Projeção Perspectiva**, quando as projetantes emanam de um único ponto que está a uma distância finita do plano de projeção e passam pelos vértices (Figura 7b).

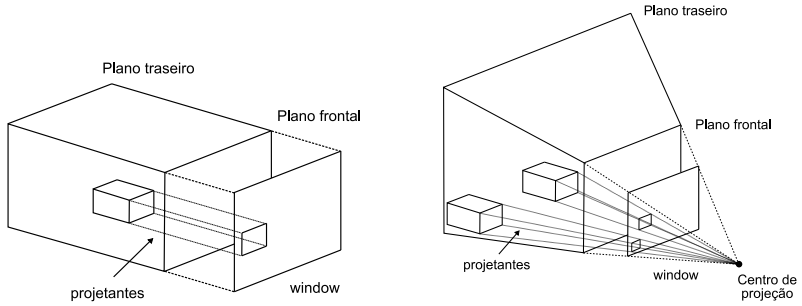


Figura 7: Projeção de paralelepípedos usando a projeção paralela ortográfica (a) e a projeção perspectiva (b).

Na projeção paralela ortográfica (Figura 7a) não há alteração nas medidas do objeto. Sua construção é bastante simples, pois, basicamente, consiste em omitir uma das componentes de cada vértice. Entretanto, a projeção perspectiva é mais utilizada, uma vez que representa melhor o que acontece na realidade. Por exemplo, se dois objetos possuem o

mesmo tamanho e estão posicionados a diferentes distâncias do plano de projeção, o objeto que está mais longe vai parecer menor do que o objeto que está próximo, conforme ilustra a Figura 7b.

Na visualização 3D também é preciso definir um volume de visualização, ou seja, a região exata do universo 3D que se deseja visualizar. O tamanho e a forma do volume de visualização dependem do tipo de projeção. Para a projeção paralela ortográfica, os quatro lados do volume de visualização e os planos frontal e traseiro na direção do eixo z, formam um paralelepípedo. Para a projeção perspectiva o volume de visualização é um tronco de pirâmide, limitado pelos planos frontal e traseiro, cujo topo é o centro de projeção, como demonstra a Figura 7b. Os planos frontal e traseiro do volume de visualização são paralelos ao plano de projeção, formando um volume de visualização limitado por seis planos e permitindo excluir partes da cena de acordo com a profundidade.

Portanto, os objetos que ficam, total ou parcialmente, fora do volume de visualização definido não devem ser exibidos. Ao processo de retirada dos objetos que não estão dentro deste volume dá-se o nome de **recorte**. Por isso, o procedimento de recorte também é incluído no *pipeline* de visualização (Seção 3.8). Os algoritmos para realizar o recorte, tal como o de Cohen-Sutherland, podem ser encontrados na literatura [8, 1, 10].

3.4 Mapeamento para a Tela

Mesmo após a etapa de projeção, as imagens já 2D no plano de projeção ainda não estão em coordenadas do Sistema de Referência da Tela (SRT) adotado no monitor do computador. Por exemplo, no SRT a origem fica no canto superior esquerdo do monitor e a resolução pode variar, enquanto que o Sistema de Referência de Projeção (SRP), que consiste no plano de projeção, é geralmente definido como $[-1, 1]$. Portanto, torna-se necessário realizar o mapeamento entre estes sistemas de referência, que consiste, basicamente, em uma regra de proporção. Além disso, é necessário definir em qual parte da tela se deseja exibir o que foi selecionado e projetado. A esta região dá-se o nome de janela de exibição ou *viewport*. Uma *viewport* é delimitada pelas coordenadas de seus cantos, os quais sempre são dados em valores que dizem respeito ao SRT. A Figura 8 ilustra o funcionamento do mapeamento.

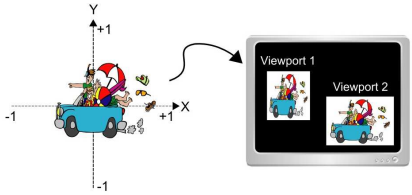


Figura 8: Mapeamento do SRP para diferentes *viewports*.

3.5 Transformações Geométricas

Uma das funcionalidades mais importantes das aplicações gráficas, é a possibilidade de manipular e alterar interativamente as características dos objetos que compõem uma cena. Já foi visto na Seção 2 que a maioria dos objetos em OpenGL consiste de uma combinação de um conjunto de primitivas gráficas definidas através de vértices. As transformações geométricas, simplificadamente, consistem em operações matemáticas realizadas sobre estes vértices, permitindo alterar uniformemente o aspecto de um modelo já armazenado no computador. Tais alterações não afetam a estrutura do desenho, mas sim o aspecto que ele vai assumir. Os três tipos fundamentais de transformações geométricas, ilustradas na Figura 9, são translação, rotação e escala.

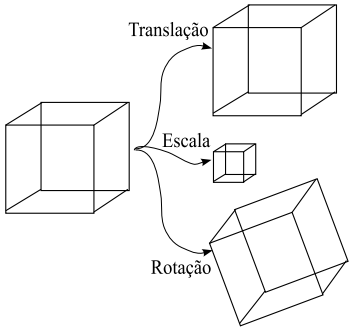


Figura 9: Exemplo da aplicação das transformações geométricas em um cubo.

Em CG este processamento é feito através de matrizes utilizando coordenadas homogêneas. Neste tipo de representação, um ponto (x, y, z) é representado como ($w.x$, $w.y$, $w.z$, w) para qualquer fator de escala $w \neq 0$. Sendo assim, utiliza-se 1 para o valor de w para representação de pontos em coordenadas homogêneas. O objetivo de utilizar coordenadas homogêneas é possibilitar a combinação das transformações geométricas, de maneira a reduzir a quantidade de cálculos.

A transformação geométrica de **translação** é usada para definir a posição de um objeto ou cena. Matematicamente, esta operação consiste em adicionar constantes de deslocamento a todos os vértices, ou seja, trocando o objeto ou cena de lugar. Na forma matricial a translação consiste na operação apresentada na Equação 1. As variáveis *tx*, *ty* e *tz* correspondem, respectivamente, aos valores de translação que devem ser aplicados nos eixos x, y e z.

$$[x'y'z'1] = [xyz1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ tx & ty & tz & 1 \end{bmatrix} \quad (1)$$

A transformação geométrica de escala serve para definir a escala a ser usada para exibir o objeto ou cena. Matematicamente, esta operação consiste em multiplicar um valor de escala por todos os vértices do objeto (ou conjunto de objetos) que terá seu tamanho aumentado ou diminuído. Como se trata de uma multiplicação, para aumentar o tamanho deve ser aplicado um fator de escala maior que 1.0, em um ou nos três eixos. Para diminuir o tamanho basta aplicar um valor de escala entre 0.0 e 1.0. No caso de se aplicar um fator de escala negativo, o objeto terá os sinais de suas coordenadas invertidos, gerando como resultado o seu "espelhamento" no eixo que teve o fator de escala negativo. Na forma matricial a escala consiste na operação apresentada na Equação 2. As variáveis *ex*, *ey* e *ez* indicam, respectivamente, os valores de escala que devem ser aplicados nos eixos x, y e z.

$$[x'y'z'1] = [xyz1] \begin{bmatrix} ex & 0 & 0 & 0 \\ 0 & ey & 0 & 0 \\ 0 & 0 & ez & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

A transformação geométrica de rotação é usada para rotacionar um objeto ou cena em torno de um eixo. Matematicamente, esta operação consiste em aplicar uma composição de cálculos utilizando o seno e cosseno do ângulo de rotação a todas as coordenadas dos vértices que compõem o objeto ou cena. Quando se trabalha em 3D, também se deve definir em torno de qual eixo se procederá a rotação. Portanto, conforme apresentado na Equação 3, três matrizes diferentes são definidas, uma para cada eixo. O símbolo α indica o ângulo de rotação. Seguindo a convenção da regra de mão direita, quando o valor do ângulo de rotação é positivo, a rotação é feita no sentido anti-horário. Isso porque se considera que o eixo z (ou x, ou y) é "envolvido" pela mão direita, alinhando o polegar com o sentido positivo do primeiro, e fechando os demais dedos no sentido positivo da rotação, ou seja, anti-horário.

$$r_z = \begin{bmatrix} cos\alpha & sen\alpha & 0 & 0 \\ -sen\alpha & cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} r_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\alpha & sen\alpha & 0 \\ 0 & -sen\alpha & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} r_y = \begin{bmatrix} cos\alpha & 0 & sen\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -sen\alpha & 0 & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

Uma rotação também pode ser feita ao redor de qualquer eixo, isto é, em torno de um eixo que não coincide com x, y ou z. Neste caso é feita uma combinação de translações e rotações ao redor dos eixos x, y ou z. O primeiro passo consiste em se alinhar o eixo de

rotação especificado com um dos eixos cartesianos, para depois aplicar a rotação de acordo com o ângulo selecionado. No final, o eixo de rotação deve ser colocado na sua posição original. A Figura 10 exemplifica estes passos: a partir de um objeto e o eixo ao redor do qual será feita a rotação (a), é feita uma translação para que o eixo de rotação passe pela origem (b). Depois, é realizada uma rotação para alinhar o eixo de rotação com o eixo z (c). Dessa forma, agora basta aplicar uma rotação com o ângulo especificado em torno do eixo z (d). Finalmente, aplica-se a rotação inversa ao passo (c) para retornar o eixo de rotação para a sua orientação original, e por último, realiza-se a translação inversa ao passo (b) para retornar o eixo de rotação para a sua posição original (e).

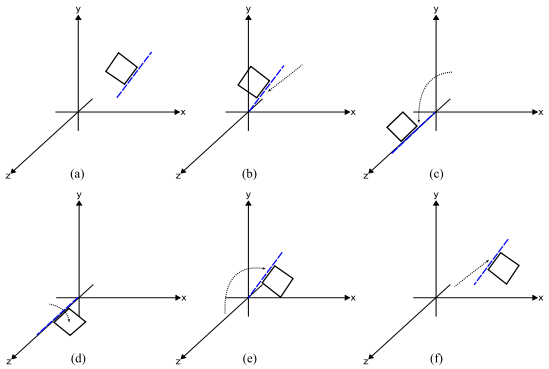


Figura 10: Exemplo de rotação em torno de um eixo arbitrário: (a) posição inicial; (b) translação para a origem; (c) rotação para alinhar o eixo com o eixo z; (d) rotação do objeto ao redor do eixo z; (e) rotação inversa para a orientação original; (f) translação inversa para a posição original (adaptada de [10]).

Através das matrizes com coordenadas homogêneas apresentadas nesta seção é possível combinar as transformações geométricas, de maneira a reduzir a quantidade de operações matemáticas a serem aplicadas em cada vértice do modelo. Assim, para fazer uma combinação das transformações geométricas, todas as matrizes de transformação que serão aplicadas são multiplicadas entre si, e cada vértice é multiplicado somente pela matriz resultante, chamada de matriz de transformação corrente.

É muito importante observar que a ordem na qual as funções são chamadas altera o resultado final. Isto ocorre porque na verdade, a translação é feita através da soma de constantes de deslocamento, enquanto a rotação e a escala consistem em operações de multiplicação. Portanto, dependendo do resultado esperado, deve-se cuidar para que a translação seja feita antes ou depois das demais transformações.

3.6 Rasterização

Após a etapa de mapeamento para a tela, é preciso definir exatamente quais são os *pixels* que irão compor uma linha e como preencher os diferentes polígonos com uma determinada cor. Estes procedimentos são conhecidos como rasterização. Atualmente os algoritmos para desenhos de linhas, tal como o de *Bresenham* [10], e para preenchimento de polígonos, como o *flood-fill* [10], são implementados pelas bibliotecas gráficas, bastando apenas chamar as funções definidas para desenhar linhas e polígonos.

3.7 Remoção de Superfícies Escondidas

A idéia de remoção de superfícies escondidas parte do princípio que os objetos serão exibidos como sólidos opacos. Portanto, é necessário descobrir quais são as faces de cada objeto que realmente devem aparecer ou não na cena. Dependendo da posição do observador, podem ocorrer duas situações: em um mesmo objeto pode haver faces que não são visíveis por estarem posicionadas atrás de outras faces - são as chamadas faces traseiras; pode haver objetos que ficam oclusos por outros objetos. Por exemplo, na Figura 6a, a câmera está na frente da taça e do cilindro, que cobrem uma parte do cubo. Este, por sua vez, tem suas faces frontais e laterais cobrindo a face traseira.

Existem vários algoritmos para remoção de superfícies escondidas, sendo o mais simples e eficiente denominado de *z-buffer*. A idéia deste algoritmo, cuja descrição completa pode ser encontrada em [8] é bastante simples: dois *buffers* do tamanho da janela de exibição são utilizados, um para guardar valores de profundidade (*z-buffer*) e outro para guardar valores de cor (*color buffer*). O *z-buffer* é inicializado com o maior valor de profundidade possível e o *color buffer* é inicializado com a cor de fundo da imagem. Depois, cada face projetada é percorrida, pixel por pixel, e o valor de profundidade de cada pixel é comparado com o valor já armazenado no *z-buffer*: se este pixel estiver mais próximo do observador, coloca-se seu valor de profundidade no *z-buffer* e seu valor de cor no *color buffer*. Assim, no final do processamento o *color buffer* irá conter a imagem final.

3.8 *Pipeline* de Visualização

Em CG, é comum agrupar um conjunto de objetos descritos geometricamente para formar o que chamamos de cena 3D. Como uma imagem na tela do computador consiste em uma matriz de *pixels*, é necessário executar todas etapas descritas nesta seção e sumarizadas na Figura 11, para fazer a Síntese de Imagens. Inicialmente, é feita a modelagem e o instanciamento dos objetos para formar a cena 3D. Depois, a câmera é posicionada e direcionada, e os objetos da cena 3D são mapeados para o SRC. Uma vez definido o volume de visualização, é realizado o recorte 3D e a projeção. Os últimos passos consistem em fazer o mapeamento

para o SRT e a rasterização.

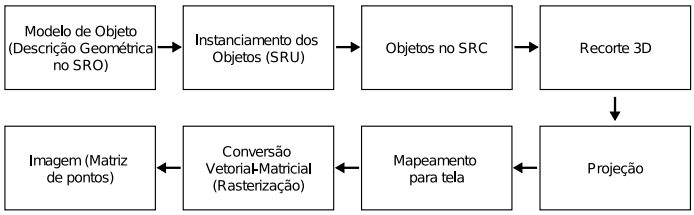


Figura 11: *Pipeline* de visualização 3D.

4 Realismo

As técnicas explicadas na Seção 3 permitem a geração de imagens simples, cuja aparência é artificial. Para que essas imagens tenham mais realismo, é preciso considerar outros aspectos. Dois desses aspectos serão abordados nesta seção, a idéia de iluminação e a técnica de mapeamento de texturas.

4.1 Iluminação

Se considerarmos o funcionamento da visão humana, o requisito essencial para se enxergar objetos é a existência de luz. A luz, presente de várias formas, é necessária para que os objetos possam refletir e absorver seus raios, e então, serem percebidos pelo olho humano. Dessa forma, quando se fala de realismo em imagens geradas por CG, é necessário inicialmente compreender a natureza das fontes de luz e suas interações com os objetos.

Denomina-se fonte de luz **puntual** (Figura 12a) aquela cujos raios de luz emanam uniformemente em todas as direções a partir de um único ponto no espaço. Esse tipo de fonte de luz é comparável a uma lâmpada, e a iluminação na superfície de um objeto evidentemente irá variar de acordo com a distância e direção da fonte de luz.

Já uma fonte de luz **direcional** (Figura 12b) é aquela cujos raios de luz vêm sempre da mesma direção. Dessa forma, a orientação de cada superfície e a direção da fonte de luz determinam o seu efeito. Por exemplo, uma face é plenamente iluminada se estiver perpendicular aos raios de luz incidentes. Quanto mais oblíqua uma face estiver em relação aos raios de luz, menor será a sua iluminação. A luz solar é o exemplo clássico de fonte de luz direcional: apesar do Sol estar em um ponto no espaço (o que a princípio poderia sugerir que é uma fonte de luz puntual), como a sua distância até a Terra é muito grande, considera-se que os raios de luz chegam praticamente paralelos à nossa superfície.

O terceiro tipo mais comum de fonte de luz é a do tipo ***spot*** (Figura 12c). Esse tipo é na verdade uma combinação de uma luz puntual com um componente direcional: os raios de luz são emitidos na forma de um cone, apontado para uma determinada direção. O exemplo mais comum desse tipo de fonte de luz é um abajur ou uma lanterna. Na luz do tipo *spot*, a intensidade diminui conforme o raio de luz é desviado da direção para a qual a fonte de luz está apontada. Freqüentemente, modela-se fontes de luz desse tipo através de parâmetros adicionais, que definem o ângulo de abrangência e a função de atenuação da intensidade de luz.

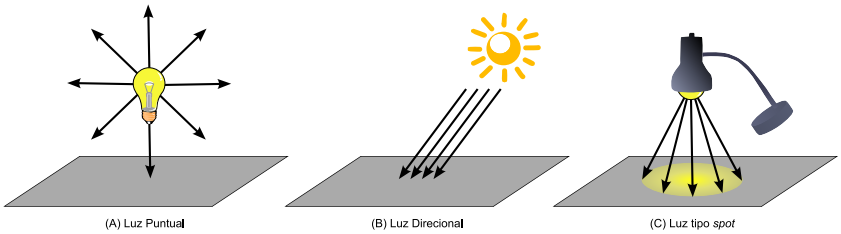


Figura 12: Fontes de luz mais comuns em Computação Gráfica.

As interações entre as fontes de luz e as superfícies são descritas através de **modelos de reflexão**. Em um modelo desse tipo, considera-se as propriedades da superfície e a natureza da(s) fonte(s) de luz. A idéia de **cor** é fundamental para se entender como se dá a interação entre luz e superfície: a percepção de cor nada mais é do que o resultado dessa interação, combinado com a interpretação desse fenômeno pelo sistema visual humano. Em um computador, cores são representadas através de sistemas de cores, como por exemplo, o sistema RGB (vermelho, verde e azul). É importante lembrar que a fonte de luz também pode possuir cor, e não apenas ser branca.

O modelo de reflexão mais simples é denominado **reflexão ambiente** (ou luz ambiente) - Figura 13a. Seu objetivo é modelar a luz que está presente no ambiente, mas cuja origem não pode ser precisamente determinada - por exemplo, a luz que vem por baixo de uma porta, ou por uma fresta em uma janela. O modelo de luz ambiente também representa as diversas inter-reflexões que ocorrem no ambiente, e que não podem ser facilmente medidas. Ou seja, a luz indireta que os objetos recebem. Portanto, esse tipo de reflexão permite a visibilidade de superfícies que não estejam recebendo diretamente raios de luz. Por definição, depende apenas da cor do objeto e gera uma iluminação constante em toda a sua superfície.

A **reflexão difusa** (ou reflexão *Lambertiana*) ocorre na superfície da maioria dos objetos que não emitem luz. Todo objeto absorve a luz do Sol ou a luz emitida de uma fonte artificial, e reflete parte desta luz. Dessa forma, a reflexão difusa deve-se ao fato de haver uma interação molecular entre a luz incidente e o material da superfície. Por exemplo, supondo

um objeto azul que está sendo iluminado por uma fonte de luz branca: o objeto absorve os raios de luz "brancos" e reflete apenas o "componente azul" da cor. Portanto, uma superfície perfeitamente difusa reflete os raios de luz igualmente em todas as direções, como mostra a figura Figura 13b. Esse tipo de reflexão depende da cor do objeto e da posição da fonte de luz: a quantidade de luz refletida percebida pelo observador não depende da sua posição. A reflexão difusa normalmente produz uma variação da cor ao longo das superfícies dos objetos.

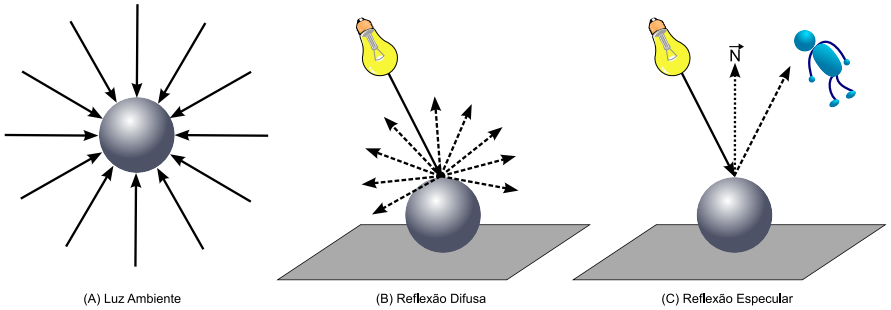
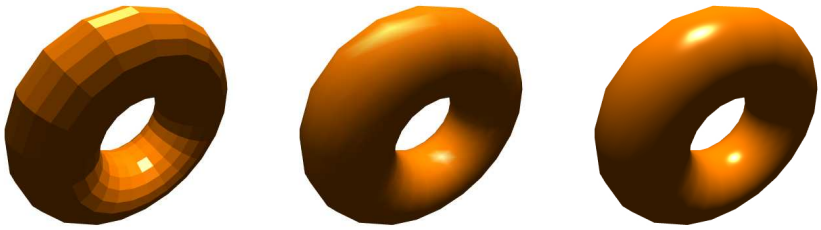


Figura 13: Modelos de reflexão da luz.

Entretanto, na realidade a maioria das superfícies não é um refletor difuso perfeito, ou seja, parte da reflexão se comporta de forma diferente. Por exemplo, superfícies polidas exibem normalmente pontos de brilho, onde pode ser vista uma reflexão da própria fonte de luz. Esta reflexão é denominada **reflexão especular** e a cor do brilho usualmente é a mesma da fonte de luz. A posição do observador é importante, uma vez que a luz refletida de uma superfície polida deixa a superfície com o mesmo ângulo que o raio de luz incidente forma com um vetor perpendicular à superfície (denominado *vetor normal*). Se o observador estiver exatamente na frente desse raio refletido (Figura 13c), ele enxergará o brilho na superfície do objeto com a maior intensidade. Naturalmente, à medida em que a posição do observador muda em relação ao raio refletido, a intensidade do brilho diminui.

Finalmente, para se aplicar um modelo de iluminação e de reflexão costuma-se empregar um modelo de **tonalização**. Por exemplo, denomina-se tonalização *flat* (também chamado constante ou facetado) o modelo no qual o cálculo de iluminação é realizado uma única vez por superfície, geralmente tomando-se um ponto no centro desta. Esse tipo de tonalização produz imagens com rapidez, mas sem muito realismo já que não há variação de cor ao longo das superfícies (Figura 14a). Usualmente, são empregados modelos mais sofisticados, como o de *Gouraud*, onde a iluminação é calculada nos vértices e as cores geradas podem então ser rapidamente interpoladas no interior das superfícies (Figura 14b). O modelo de *Gouraud* também permite a suavização de objetos facetados, pois utiliza a média entre os vetores normais, empregados no cálculo de iluminação. Porém, uma desvantagem desse método é que

os pontos de brilho especular são atenuados, devido à interpolação de cores. Para resolver esse problema, o modelo de *Phong* interpola os vetores normais dos vértices que compõem uma superfície, e realiza o cálculo de iluminação por pixel, durante a rasterização. Isso obviamente produz um resultado visual muito superior (Figura 14c), porém o custo computacional também é consideravelmente maior. Por fim, cabe observar que hoje em dia já é possível implementar tonalização de Phong através de programação direta do *hardware* gráfico, o que é muito mais eficiente.

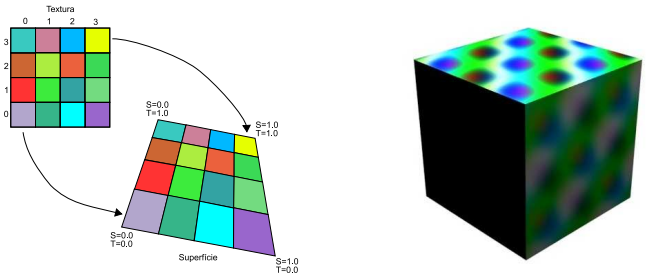


(a) *Flat* (b) *Gouraud* (c) *Phong*

Figura 14: Modelos de tonalização comumente empregados em Computação Gráfica.

4.2 Textura

Mesmo através do recurso de iluminação, freqüentemente os objetos não terão uma aparência realística. Isso acontece porque os materiais na realidade não tem simplesmente cores diferentes, mas sim uma série de características físicas, como rugosidade e textura, que determinam como exatamente refletem os raios de luz. Em CG, costuma-se modelar essas características através de um conjunto de diversas técnicas, que podem ser combinadas. Aqui será abordada a técnica mais comum, denominada genericamente mapeamento de textura, ou textura mapeada. A idéia é utilizar uma imagem que contenha a aparência da superfície desejada. Por exemplo, madeira ou metal. Essa imagem é então, durante o processo de rasterização, mapeada sobre a superfície através de **coordenadas de textura**. Estas coordenadas (usualmente chamadas de s e t) determinam como o mapeamento é realizado, conforme a Figura 15a apresenta. A textura funciona então, como um decalque ou um papel de parede, sendo "colada" à superfície. É importante observar que o mapeamento de textura pode (e geralmente é) combinado com o processo de iluminação, gerando um resultado ainda melhor (Figura 15b).



(a) Coordenadas de textura (b) Efeito com iluminação

Figura 15: Mapeamento de textura.

5 Animação

A idéia de animação está intimamente ligada à de movimento, mas animação também pode ser mudança de cor ou forma. De uma forma geral, animação se dá quando exibe-se uma seqüência de imagens rapidamente, onde cada imagem é um pouco diferente da anterior. Se essas imagens (quadros ou *frames*) forem exibidas com uma velocidade suficiente (denominada taxa de exibição ou *frame rate*), será produzida a ilusão de movimento. Porém, em geral se caracteriza a animação computadorizada de duas formas distintas [23, 16]: animação *assistida* por computador e animação *modelada* por computador. No primeiro tipo (assistida), o computador é utilizado como uma ferramenta de desenho, pintura ou até mesmo para comandar a câmera de filmagem. Já no segundo tipo, popularizado recentemente pelo surgimento de vários filmes animados, o computador é empregado durante todo o processo, desde a modelagem e criação de cenários e personagens, até a geração da imagem de cada quadro individual, aplicando técnicas de realismo como iluminação, texturas e muitas outras.

Uma das técnicas mais comuns de animação é a denominada **animação paramétrica**, onde as propriedades de cada objeto são parametrizadas (como por exemplo, posição, rotação, escala ou cor) e são animadas, ou seja, alteradas ao longo do tempo, produzindo o efeito de animação. Geralmente utiliza-se a idéia de quadro-chave (*key frame*), isto é, determinados instantes de tempo onde algumas ou todas as propriedades são especificadas pelo usuário (Figura 16). O sistema de animação então calcula as propriedades nos demais instantes de tempo através de algum tipo de interpolação. Outro tipo usual de animação é aquela baseada em simulação, onde as propriedades dos objetos são alteradas de acordo com o resultado da simulação. Por exemplo, o lançamento de um projétil pode ser modelado através de simulação da Lei da Gravidade.

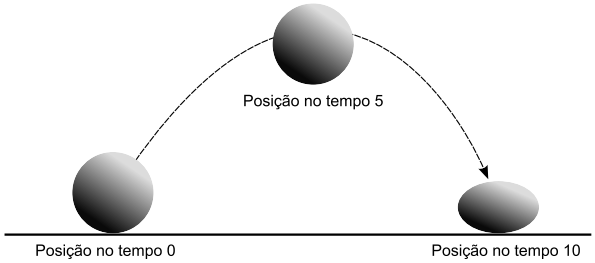


Figura 16: Animação paramétrica: três instantes de tempo onde a posição e escala do objeto são especificadas - no final, a escala é reduzida no eixo y, para simular o achatamento da bola.

6 Visualização

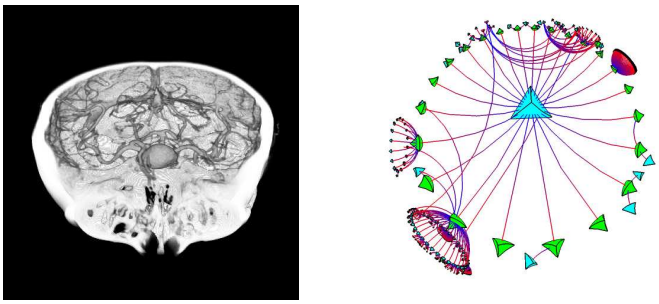
Uma área de grande importância dentro da CG é a Visualização, que consiste, de maneira simplificada, na representação gráfica adequada de um conjunto de dados alfanuméricos. A Visualização Científica [14] (VisC) aplica técnicas sofisticadas de CG e PI para produzir imagens de fenômenos/dados físicos complexos e permitir a sua exploração gráfica com o objetivo de facilitar o seu entendimento. Como, muitas vezes, a quantidade de dados é tão grande, entendê-los sem visualizá-los é quase impossível. Por isso, a VisC surgiu como uma forte ferramenta, permitindo que cientistas e pesquisadores simulem os problemas em estudo, interpretem seus dados e progridam na compreensão da solução. Entre as grandes áreas de aplicação da Visualização Científica encontram-se a Medicina, Meteorologia, Química, Física, Geologia, Sistemas de Informações Geográficas, entre outras.

Visualização Volumétrica (Figura 17a) designa o conjunto de técnicas utilizadas para produzir imagens que representam dados associados a posições (ou regiões) do espaço 3D, que constituem o que costuma-se referenciar por grade [13]. O seu principal objetivo segundo [11] é exibir as estruturas existentes no interior do volume de dados, permitindo a identificação de características significativas e a compreensão dos dados e fenômenos representados.

Já a Visualização Colaborativa é outra categoria de sistemas de visualização que surgiu com o crescimento explosivo da Internet [25]. A idéia é desenvolver ferramentas que permitam a colaboração entre diversos usuários, possibilitando, por exemplo, a inserção de anotações e a manipulação e debate, em tempo real, sobre a visualização de um determinado conjunto de dados.

Visualização de Informações [9, 22] é uma área de aplicação de técnicas de computação gráfica, geralmente interativas que combina aspectos de CG, Interação Humano-

Computador(IHC) e mineração de dados. Com a crescente importância da área de IHC, existe uma preocupação com o projeto, avaliação e implementação de sistemas computacionais interativos para uso humano e aos fenômenos que os cercam [18]. Além disso, atualmente, a interação pode ser feita através de diversos dispositivos, tais como mouse, teclado, *data glove*, entre outros. Ferramentas para a visualização do crescimento de uma rede, visualização de hierarquias e estruturas de árvore (Figura 17b), visualização e manipulação de grafos abstratos, auxiliando a análise e visualização de sistemas complexos, e para fazer uma busca e análise visual em grandes bases de dados para descobrir conhecimentos interessantes, são alguns exemplos de aplicações desta área.



(a) (b)

Figura 17: Tipos de visualização: (a) científica, reconstrução volumétrica de um aneurisma cerebral a partir de imagens de tomografia computadorizada [4]; (b) de informações, exibição de um extenso sistema de arquivos através de uma técnica de projeção hiperbólica [15].

7 Comentários Finais

Como a evolução da Computação Gráfica depende da evolução do hardware, recentemente, as novas gerações de placas gráficas possibilitaram o desenvolvimento de várias aplicações e um grande avanço em muitas áreas da CG. Por exemplo, os jogos eletrônicos hoje apresentam imagens com alto grau de realismo, além de permitir a conexão de jogadores através da Internet [5, 2, 6]. Muitas imagens podem ser geradas em tempo real devido à possibilidade de programar o hardware gráfico [7, 17, 19]. Isso também tem beneficiado a área de visualização, que, em geral, trabalha com uma grande quantidade de dados que pode ser armazenada na memória das placas gráficas. Por isso, a programação de hardware gráfico tem sido uma área em grande evolução.

Outra facilidade hoje em dia é a grande disponibilidade de bibliotecas e *toolkits* que auxiliam a programação. Por exemplo, para o desenvolvimento de aplicações simples em

OpenGL costuma-se empregar a *GLUT (GL Utility Toolkit)*, que oferece recursos básicos para gerenciamento de janelas e interação com o usuário. Já para a implementação de interfaces gráficas mais complexas, pode-se citar o *FLTK (Fast Light Toolkit)*, disponível em `http://www.fltk.org`, e o *wxWidgets* (`http://www.wxwidgets.org`). Um *toolkit* mais completo para programação gráfica interativa é o *OpenInventor* (`http://oss.sgi.com/projects/inventor/`), que fornece para inclusão além de primitivas 3D, tais como cubos e polígonos, câmeras, luzes, visualizadores, entre outros elementos. Para a implementação de aplicações de visualização e processamento de imagens, pode-se utilizar o *VTK (Visualization Toolkit)* e o *ITK* (*Insight Registration and Visualization Toolkit*), este último com ênfase na manipulação de imagens médicas), ambos portáveis, disponibilizados gratuitamente e com código-fonte aberto. Os endereços para obter mais informações sobre estes *toolkits* são `http://www.vtk.org` e `http://www.itk.org`.

Recentemente, também houve um avanço muito grande nas interfaces gráficas. Novas versões de ambientes operacionais possuem interfaces 3D que permitem trabalhar com janelas transparentes que podem rotacionadas ao redor de qualquer eixo. O reconhecimento de gestos, e até mesmo expressões faciais, também pode ser utilizado como mais uma forma de interação. Animação de humanos virtuais, simulação de comportamento de multidões, aplicações de Realidade Virtual [20] e Realidade Aumentada são outros exemplos de áreas que têm sido muito pesquisadas.

Referências

[1] E. Angel *Interactive Computer Graphics: a top-down approach with OpenGL*. Reading, MA: Addison-Wesley, 2000. 611 p.

[2] D. Astle, K. Hawkins *Beginning OpenGL Game Programming*. Muska & Lipman/Premier-Trade, 1^{st} edition, 2004, 336 p.

[3] M. Cohen, I.H. Manssour. *OpenGL - Uma Abordagem Prática e Objetiva*. São Paulo: Novatec, 2006. 486 p.

[4] M. Cohen. *Focus and Context for Volume Visualization*, PhD thesis. School of Computing, University of Leeds, 2006. 206 p.

[5] M. Collins, M. Donlin, S. Baker, B. Campbell. Linux Game Programming (Prima Tech's Game Development). Muska & Lipman/Premier-Trade, 1^{st} edition, 2002, 368 p.

[6] M. Dickheiser. *Game Programming Gems 6 (Game Development Series)*. Charles River Media, 1^{st} edition. 2006. 736 p.

[7] R. Fernando, M.J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley, 2003. 384 p.

[8] J.D. Foley, et al. *Computer Graphics: Principles and Practice*. New York, Addison Wesley, 1990, 2^{th} ed.

[9] C.M.D.S. Freitas, O.M. Chubachi, P.R.G. Luzzardi, R.A. Cava. *Introdução à Visualização de Informações*. Revista de Informatica Teórica e Aplicada, Porto Alegre, RS, v. 8, n. 2, p. 143-158, 2001.

[10] D. Hearn, M. Baker. *Computer Graphics with OpenGL*. 3^{rd} edition. Upper Saddle River, NJ: Pearson Prentice-Hall, 2004. 855p.

[11] A.E. Kaufman, D. Cohen, R. Yagel. *Volume Graphics*. Computer, v. 26, n. 7, 1993. p. 51-64.

[12] O. Lathrop. *The Way Computer Graphics Works*. New York, NY: John Wiley Sons, 1997. 202 p.

[13] B. Lichtenbelt, R. Crane, S. Naqvi. *Introduction to Volume Rendering*. Upper Saddle River, NJ: Prentice Hall, 1998.

[14] B. H. McCormick, T. A. DeFanti, M. D. Brown. *Visualization in Scientific Computing*, Computer Graphics, v. 21, n. 6, p. 15–21, 1987.

[15] T. Munzner, P. Burchard. *Visualizing the structure of the World Wide Web in 3D hyperbolic space*. VRML '95: Proceedings of the first symposium on Virtual reality modeling language, ACM Press, p. 33–38, 1995.

[16] R. Parent. *Computer animation: algorithms and techniques*. San Francisco: Morgan Kaufmann, 2002. p. 51-64. 527 p.

[17] M. Pharr, R. Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, 2005, 880 p.

[18] H.V. da Rocha, M.C.C. Baranauskas. *Design e Avaliação de Interfaces Humano-Computador*. NIED/UNICAMP. 2003. 244 p.

[19] R.J. Rost. *OpenGL(R) Shading Language*. 2^{nd} edition. Addison-Wesley Professional. 2006. 800 p.

[20] W.R. Sherman, A. Craig. *Understanding Virtual Reality: Interface, Application, and Design (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann, 1^{st} edition, 2003, 582 p.

[21] D. Shreiner, M. Woo, J. Neider, T. Davis. *OpenGL Programming Guide: the official guide to learning OpenGL, version 1.4*. 4^{rd} Edition. Reading, Massachusetts: Addison Wesley, 2004. 759 p.

[22] R. Spence. *Information visualization*. Harlow: Addison-Wesley, 2003. 206 p.

[23] J. Vince. *3-D Computer Animation*. Addison-Wesley, New York, 1992. 363 p.

[24] A. Watt. *3D Computer Graphics*. 3^{rd} Edition. Harlow: Addison-Wesley, 2000. 570 p.

[25] J. Wood, H. Wright, K. Brodie, *Collaborative visualization*. IEEE Visualization 1997 (VIS'97), 1997. p. 253

[26] R.S. Wright, B. Lipchak. *OpenGL SuperBible*. Sams, 3^{rd} Edition, 2004, 1200 p.