Visualização Interativa 3D de Dados Volumétricos

Marcelo Medeiros Carneiro
PUC-Rio/Departamento de Informática
mmc@inf.puc-rio.br

Luiz Fernando Martha
TeCGraf – Grupo de Tecnologia em Computação Gráfica/PUC-Rio
lfm@tecgraf.puc-rio.br

PUC-RioInf.MCC15/00 Março/2000

ABSTRACT: Volume Rendering is a method of extracting useful information from a data set in the tridimensional space. However, this method requires the use of adequated tools that allows the exploration of the data in a flexible and interactive fashion. This work presents a set of techniques that helps the development of user interfaces for volumetric data manipulation. This study allows us to identify the most important aspects of the architecture of these interfaces.

KEYWORDS: Volume Rendering, Interactive Volume Rendering, 3D User Interfaces, 3D Widgets, 3D Direct Manipulation.

RESUMO: Visualização Volumétrica é um método utilizado para extrair informações úteis a partir de um conjunto de dados no espaço tridimensional. Entretanto, isto requer a utilização de ferramentas que permitam ao usuário explorar o dado volumétrico de uma maneira flexível e interativa. Este trabalho apresenta algumas técnicas que facilitam o desenvolvimento de interfaces para a manipulação de dados volumétricos. Este estudo permite identificar as características mais importantes da arquitetura de tais interfaces.

PALAVRAS-CHAVE: Visualização Volumétrica, Visualização Volumétrica Interativa, Interfaces 3D, Widgets 3D, Manipulação Direta 3D.

Sumário

Lista de Figuras

1 INTRODUÇÃO

Técnicas de visualização volumétrica podem ser utilizadas para criar uma imagem bidimensional a partir de um conjunto de dados tridimensional. Um dos principais objetivos da visualização volumétrica é permitir que o usuário extraia informações relevantes dos dados. Isto é importante para analisar um fenômeno físico, diagnosticar uma doença etc.

Entretanto, por ser uma área relativamente nova da computação gráfica e ainda em grande expansão, o principal enfoque da visualização volumétrica ainda está relacionado com o problema da visualização em si, isto é, com os algoritmos de *rendering* de volumes. Ainda há necessidade de realizar progressos para permitir um maior controle sobre a exploração dos dados.

A Figura 1.1 mostra uma situação típica que requer mecanismos eficientes de interação 3D. O paciente possuia um grande tumor facial que foi retirado através de cirurgia. A seguir um implante de titânio (mostrado em amarelo na figura) foi projetado para reconstruir a mandíbula inferior. Este implante deve ser corretamente posicionado na face do paciente para que o resultado seja satisfatório. A direita pode-se observar como ficou a face do paciente após o posicionamento do implante.

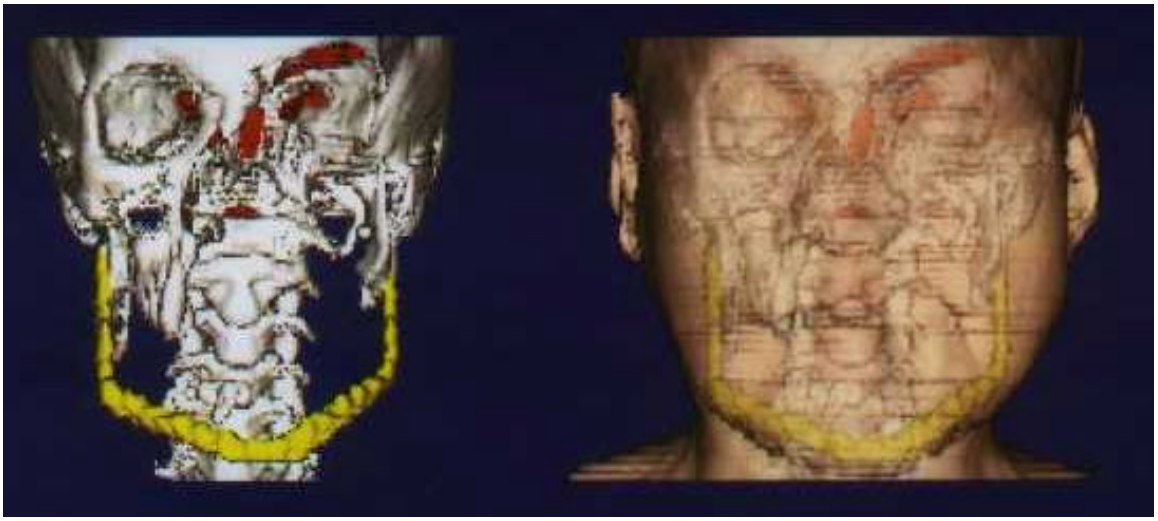


Figura 1.1: Posicionamento de um implante em um paciente [Robb e outros, 99].

A Figura 1.2 ilustra a utilização de planos de corte em um volume 3D para permitir o planejamento de neurocirurguria em um paciente com eplepsia. Essa técnica de interação, apesar de ser bastante simples, é muito utilizada em imagens médicas, pois a visualização de imagens bidimensionais é imediata utilizando dispositivos de saída (tela do computador) tradicionais. O plano de corte pode ser posicionado livremente pelo usuário e imediatamente as fatias correspondentes são visualizadas.

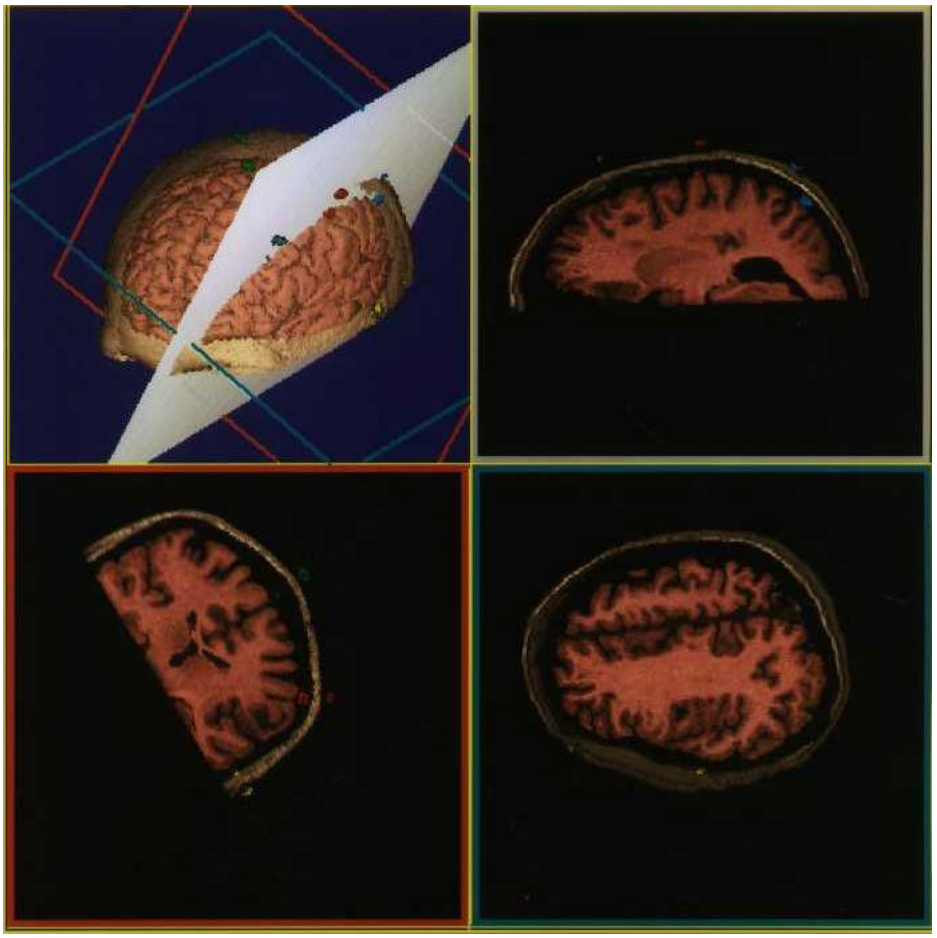


Figura 1.2: Posicionamento de planos de corte no volume [Robb e outros, 99].

Por fim, a Figura 1.3 ilustra um problema típico em imagens médicas: a modelagem de próteses ortopédicas. A interface de tais sistemas é geralmente rica e complexa, pois requer o uso de mecanismos sofisticados de interação 3D. A utilização de ferramentas adequadas para o gerenciamento da interação é um fator determinante para a implementação de tais sistemas.

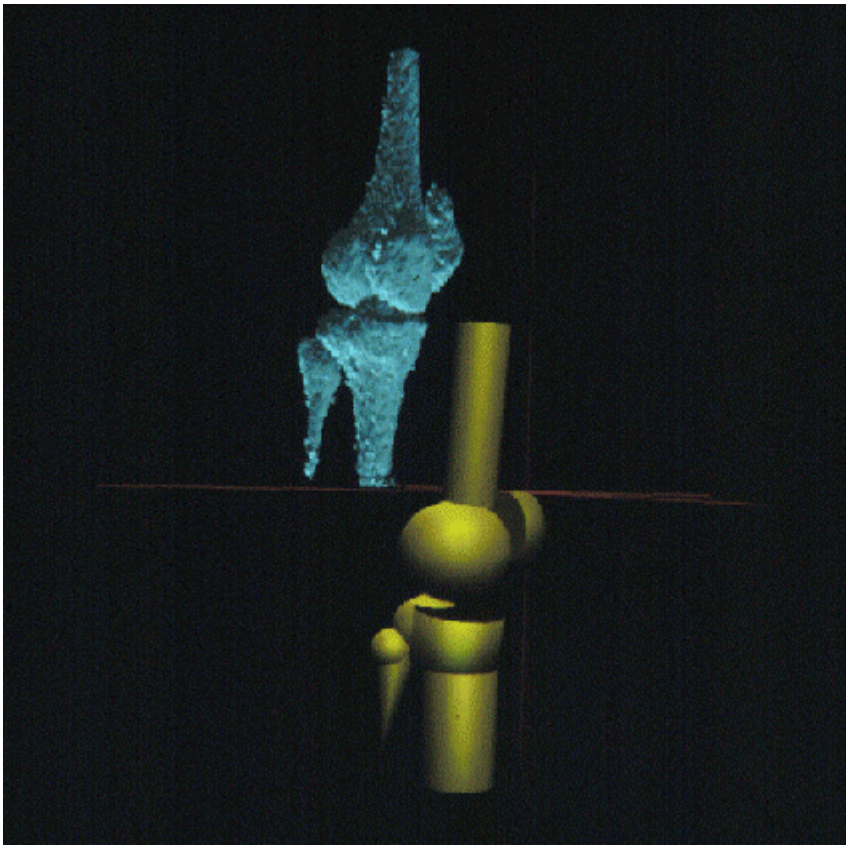


Figura 1.3: Modelagem de próteses [Kaufman e outros, 98].

Este trabalho tem o principal objetivo de investigar técnicas de interação 3D que facilitem a visualização exploratória de dados volumétricos. As interfaces tradicionais do tipo WIMP (*windows, icons, menus* e *pointers*) são eficientes para os sistemas convencionais 2D, mas podem não ser a mais adequadas para os sistemas 3D. Portanto, é necessário investigar os mecanismos de interação 3D e as características mais importantes que uma ferramenta de interação deve possuir para permitir a visualização exploratória de dados volumétricos.

2 VISUALIZAÇÃO VOLUMÉTRICA

Visualização Volumétrica é um termo geral utilizado para descrever técnicas que permitem a projeção de um conjunto de dados no espaço tridimensional (conhecido também por dados volumétricos) em uma superfície de visualização bidimensional [Elvis, 92]. Essas técnicas auxiliam o entendimento de propriedades e estruturas contidas dentro do volume, pois permitem a visualização do dado como um todo.

Em geral, o dado volumétrico é amostrado ao longo de três eixos ortogonais, definindo um reticulado cartesiano. Esse reticulado pode ser regular, onde os intervalos são idênticos em todas as direções, ou retilíneo, onde os intervalos podem ser diferentes em cada direção, porém constantes em uma mesma direção. Entretanto, isso nem sempre acontece. Em algumas aplicações, os dados são amostrados em pontos aleatórios, o reticulado pode ser esférico, curvo etc. Essa generalização torna os algoritmos de visualização volumétrica extremamente ineficientes e, sempre que possível, deve ser evitado.

Um dos maiores problemas encontrados na visualização de volumes é a grande quantidade de dados. Uma matriz de 512x512x512 contendo dados escalares de apenas 1 byte cada necessita de 128 Mbytes para ser armazenada. Esse problema trouxe sérias limitações para a visualização interativa, pois o tempo necessário para o *rendering* do volume ainda era muito grande.

2.1 Aplicações da Visualização Volumétrica

A Visualização Volumétrica ainda é um campo da computação gráfica com grandes possibilidades de crescimento. Muitos algoritmos foram desenvolvidos e alguns aperfeiçoados visando melhorar a qualidade das imagens ou diminuir seu tempo de geração, permitindo o desenvolvimento de sistemas interativos. Pode-se citar muitas aplicações de técnicas de visualização volumétrica, porém uma das mais importantes e relevantes ainda é na medicina (imagens médicas). Nesse campo, o dado volumétrico é usualmente obtido através de algum processo de aquisição, seja por tomografia computadorizada (CT), ressonância magnética (MRI), microscopia confocal, entre outros [Paiva e outros, 99].

Tipicamente, em uma aplicação em imagens médicas, o dado volumétrico é apresentado como uma seqüência de fatias bidimensionais (*slices*, normalmente com resolução de 512x512 pontos), separadas por uma pequena distância (poucos milímetros). Cada fatia representa um seção transversal de um objeto de estudo (um órgão do corpo humano, por exemplo) e o conjunto das fatias permite a construção de uma imagem tridimensional do objeto, auxiliando o médico no diagnóstico de uma doença, tratamento mais adequado ou planejamento de uma cirurgia.

Além de larga aplicação na medicina, técnicas de visualização volumétrica também são muito empregadas em outras áreas da ciência, por exemplo, meteorologia, geologia, engenharia mecânica etc. Entre essas áreas, destaca-se a utililização de visualização volumétrica na indentificação de estruturas geológicas sob a superfície do solo. Isso permite, através da interpretação de dados sísmicos, inferir propriedades do terreno, auxiliando o geólogo na identificação de bacias de petróleo [Gerhardt e outros, 98].

2.2 Rendering de Volumes

As técnicas de *rendering* de volumes permitem gerar uma imagem do volume diretamente a partir do dado volumétrico, sem a necessidade de utilizar representações intermediárias, tais como polígonos ou outras primitivas geométricas. Desse modo, técnicas de *rendering* de volumes são especialmente úteis para a visualização de objetos amorfos, ou seja, de difícil representação geométrica, tais como nuvens e fluidos.

Para os objetivos deste trabalho, o dado volumétrico é representado através de um conjunto de valores escalares distribuídos no espaço tridimensional em um reticulado cartesiano regular. Os valores escalares representam alguma propriedade do objeto volumétrico, tipicamente densidade, temperatura, pressão etc.

É importante observar que o dado volumétrico está inserido em um meio que pode ser participativo ou não participativo. O meio não participativo é caracterizado por não atenuar a luz durante sua propagação. Esse é o caso mais simples, pois a iluminação precisa ser calculada apenas nas fronteiras dos objetos. O caso mais geral, quando o meio é participativo, é muito mais complexo e a visualização exata da cena, segundo os modelos físicos, é computacionalmente difícil.

No caso da visualização volumétrica, não se espera a perfeita simulação da realidade, mas sim a visualização exploratória dos dados. Além disso, o tempo para a geração de uma imagem deve ser razoável o suficiente a ponto de permitir o mínimo de interatividade. Por esse motivo, simplificações devem ser realizadas no modelo físico da propagação da luz. Pode-se dizer que a visualização volumétrica é uma situação intermediária, entre meio não participativo e meio participativo.

Uma das simplificações mais comuns realizadas no modelo físico é admitir que há pouco espalhamento da luz quando ela se propaga pelo volume. Nesse caso, o meio pode ser modelado como um gelatina semi-transparente. A luz, ao percorrer o meio, interage com a gelatina, podendo ser absorvida, emitida ou espalhada (Veja a Figura 2.1). Nesse caso, como há interação com o meio, o cálculo da iluminação tem que realizado continuamente sendo, portanto, necessário integrar o efeito da atenuação da luz durante a sua propagação pelo meio em que se encontra. Uma outra simplificação muito comum é não considerar a atenuação da luz emitida pelas fontes luminosas pelo meio.

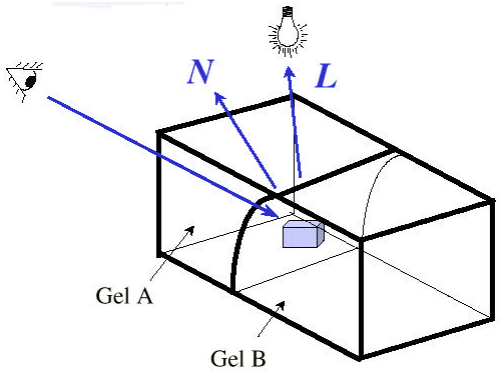


Figura 2.1: Propagação da luz no ambiente volumétrico [Paiva e outros, 99].

Todas essas simplificações reduzem bastante a complexidade dos algoritmos de visualização, diminuindo o tempo de geração das imagens e permitindo a visualização exploratória dos dados. Isso, obviamente, causa um pouco de perda na qualidade das imagens. Porém, para a maioria das aplicações (principalmente em medicina, onde não há tanta necessidade de visualização de imagens reais), o ganho de velocidade justifica a pequena perda de qualidade.

2.3 *Pipeline* de Visualização

Os algoritmos de visualização volumétrica requerem, em sua maioria, os passos apresentados na Figura 2.2. O primeiro passo consiste na aquisição do dado volumétrico, que normalmente é realizado através do uso de um equipamento de sensoriamento apropriado. A seguir, o dado volumétrico bruto é classificado, normalmente com o auxílio do usuário. Essa etapa pode ser vista como uma forma de segmentação, pois sua finalidade principal é identificar estruturas internas do volume, através do uso adequado de cores e opacidades. A seguir, o volume classificado é iluminado, facilitando sua interpretação pelo usuário, pois a forma tridimensional do volume é realçada. O último passo consiste na projeção do volume classificado e iluminado em um plano de visualização, gerando assim uma imagem bidimensional que será visualizada pelo usuário.

Essas etapas são muito comuns na grande maioria dos algoritmos de visualização. Nessa análise, está sendo considerado que o dado volumétrico é apresentado sob a forma de valores escalares em um reticulado retilínio. Esta etapa já foi previamente realizada através de alguma simulação numérica ou através de algum processo de aquisição.

Tipicamente, esses dados podem ser pré-processados, visando reduzir o ruído, realçar algumas características do objeto em estudo. Em algumas situações, os dados precisam ser reconstruídos, visando estimar valores perdidos durante o processo de aquisição, reamostrar o volume, através de interpolação, para adequar sua topologia (convertendo um reticulado irregular para um reticulado regular, por exemplo), novas fatias podem ser criadas e outras podem ser descartadas etc.

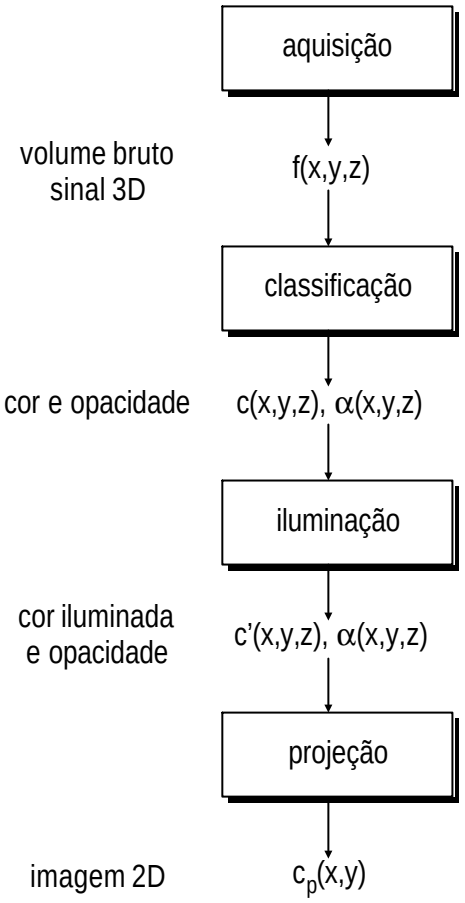


Figura 2.2: Pipeline de visualização volumétrica.

Pressupondo que o dado volumétrico já esteja armazenado em uma estrutura adequada, o processo de visualização volumétrica consiste em, a partir de uma função escalar discreta $f(x_i, y_j, z_k)$ que define o volume, gerar uma imagem bidimensional $p(x_i, y_j)$ que represente a projeção desse volume em um determinado plano, de acordo com os parâmetros de visualização.

O interessante ao estudar o *pipeline* de visualização é perceber que em cada etapa uma novidade é introduzida. Após a aquisição dos dados, tem-se o volume em seu estado bruto. Ao separar as estruturas que serão visualizadas, através da definição das funções de transferências, tem-se o dado classificado. Após o posicionamento das luzes, tem-se o dado iluminado. Finalmente, após o posicionamento da câmera, tem-se o dado projetado em uma imagem bidimensional. Fica claro, então, que qualquer alteração nesse processo é necessário revisitar as etapas seguintes. Por exemplo, se a posição da câmera for alterada, apenas a etapa de projeção precisa ser executada. Entretanto, se o usuário mudar

uma das funções de transferência, as etapas de classificação, iluminação e projeção precisam ser novamente executadas.

2.3.1 Classificação

A classificação do volume é etapa mais complexa para o usuário de um sistema de visualização volumétrica (por exemplo, um médico). Normalmente, esse usuário não tem conhecimentos técnicos a respeito do processo de visualização, entretanto o requisito fundamental para uma boa classificação do volume é o conhecimento prévio do objeto em estudo, i.é., o usuário deverá reconhecer sua anatomia, as estruturas que poderão eventualmente serem encontradas, em quais posições etc. [Elvins, 92].

O processo de classificação tem a finalidade principal de identificar as estruturas internas do volume. Com isso, um médico pode, por exemplo, isolar elementos distintos, tais como gordura, músculo e osso, em um exame de ressonância magnética. No caso de algoritmos de *rendering* direto, cujos dados são representados por grandezas escalares, a classificação do volume envolve tipicamente a definição de funções de transferência de cor e opacidade.

A função de transferência de cor permite mapear valores contidos no volume em uma cor RGB característica. Por exemplo, em um exame de tomografia computadorizada, pode-se definir uma faixa de intensidade como sendo gordura e associar a cor amarela. Para uma outra faixa, pode-se definir como sendo músculo e atribuir a cor vermelha. Osso pode ser representado pela cor branca etc (Veja a Figura 2.3).

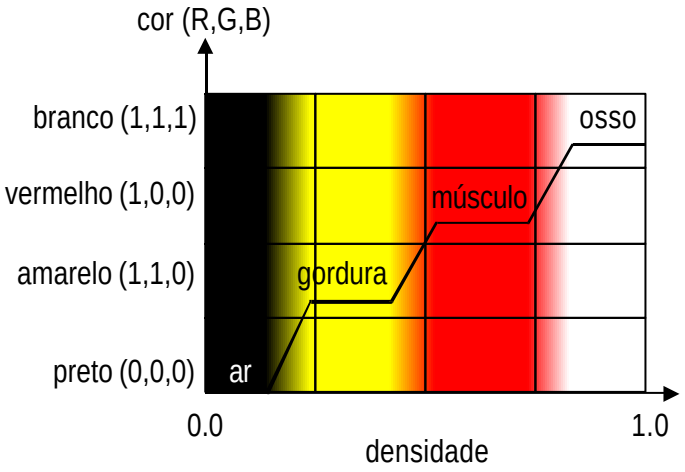


Figura 2.3: Função de transferência de cor

A função de transferência de opacidade permite mapear valores contidos em um volume em uma determinada transparência, normalmente representada por uma número real no intervalo de 0.0, para totalmente transparente, a 1.0, para totalmente opaco (Veja a Figura 2.4).

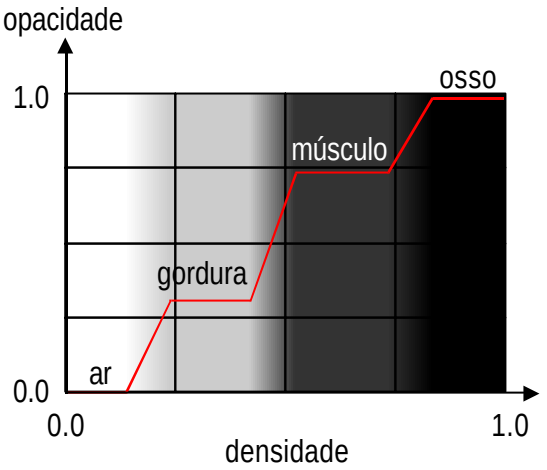
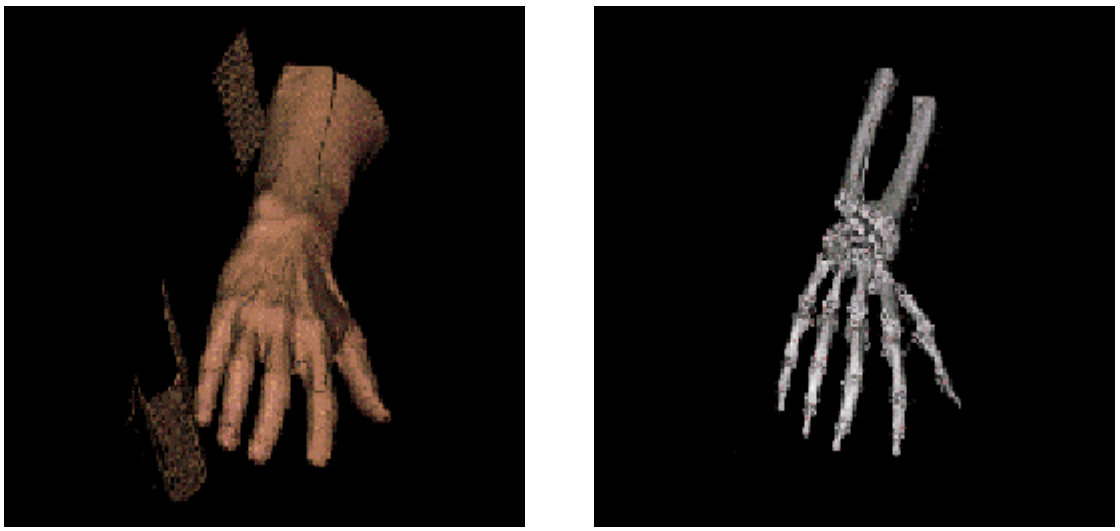


Figura 2.4: Função de transferência de opacidade.

A modificação da função de transferência de opacidade permite ao usuário visualizar determinadas estruturas do volume. Para isso, basta torná-las não transparentes e deixar as demais transparentes, ou quase transparentes. Tipicamente, quando se trata de imagens médicas, as camadas mais externas do volume são colocadas mais transparentes que as mais internas. Com isso, o usuário (por exemplo, um médico) pode ter uma visualização das partes mais internas do volume. (Veja a Figura 2.5).



(a) (b)

Figura 2.5: Mudança nas funções de transferência [Paiva e outros, 99].

Existem algumas técnicas que permitem a classificação automática do volume. Uma delas é muito utilizada em exames de tomografia computadorizada e baseia-se em um modelo de classificação probabilístico [Drebin e outros, 88]. Nesse modelo, é assumido que apenas um ou no máximo dois elementos podem existir em um *voxel*. Além disso, a existência de dois elementos só é possível na interface que os separam, por exemplo, a região entre músculo e osso.

Apesar de existirem outras técnicas automáticas de classificação de volumes, na maioria dos casos ainda é necessária a interferência do usuário para definir adequadamente as funções de transferências. O método utilizado para este fim ainda é o da tentativa e erro. Obviamente, existem ferramentas capazes de auxiliar o usuário nesse processo. Uma das mais importantes é o histograma do volume, onde é apresentado a distribuição dos valores escalares que ocorrem no volume. Com isso, o usuário pode ter uma idéia mais precisa das estruturas ali contidas.

2.3.2 Iluminação

Para criar uma ilusão de profundidade, realçar bordas e características do volume, é muito comum a utilização de um modelo de iluminação para a visualização do volume. Os algoritmos de visualização volumétrica tipicamente utilizam aproximações de modelos de iluminação de baixo espalhamento, pois são computacionalmente mais

simples e apresentam bons resultados, principalmente com imagens médicas. A inclusão de modelos de alto espalhamento pode proporcionar maior realismo para a cena, porém, no caso de dados médicos, não traz grandes benefícios para sua interpretação. Portanto, na maioria dos casos, não justifica a utilização de modelos de alto espalhamento.

A aproximação de um modelo de iluminação de volumes de baixo espalhamento é realizado através de um modelo local de iluminação. Os modelos locais consideram apenas a reflexão da luz na superfície do objeto. Isso significa que, nesses modelos, apenas a luz ambiente e as reflexões difusa e especular das fontes de luz existentes na cena são consideradas no cálculo da iluminação.

Entre os modelos locais de iluminação, os mais conhecidos são Gouraud [Gourad, 71] e Phong [Phong, 75]. No método de Gourad, a iluminação é calculada nos vértices da célula e interpolada em seu interior. Já no método de Phong, a normal nos pontos dentro de uma célula é calculada através da interpolação das normais nos vértices da mesma. A iluminação é, portanto, calculada em cada ponto no interior da célula.

No caso de algoritmos de *rendering* direto, o conceito de superfície pode não fazer muito sentido, mas é de extrema importância para o correto cálculo da iluminação dos *voxels*. Nessa situação, imagina-se que o ponto sendo iluminado pertence a uma isosuperfície existente no volume. Efetivamente, os algoritmos procuram fazer uma estimativa da normal à superfície para efeito de iluminação.

Um dos métodos mais utilizados para a estimativa da normal em um determinado ponto do volume é utilizando uma aproximação do gradiente do volume. Essa aproximação é calculada através de diferenças finitas em uma vizinhança do campo escalar volumétrico. Utilizando diferenças centrais, a normal **`N`** pode ser estimada pela fórmula abaixo: (`D[i,j,k]` representa o valor escalar na posição `(i,j,k)` do volume)

N_x `= D[i+1,j,k] - D[i-1,j,k]`

N_y `= D[i,j+1,k] - D[i,j-1,k]`

N_z `= D[i,j,k+1] - D[i,j,k-1]`

Essa estimativa da normal é bastante simples de ser realizada, porém, por ser baseada em uma vizinhança muito próxima do volume, é muito sensível a ruídos e descontinuidades do volume. Em algumas situações, a estimativa não pode ser utilizada. Para reduzir o efeito da localidade, pode-se estimar o gradiente considerando não apenas os seis vértices vizinhos, mas sim os 26 vizinhos (estimativa de segunda ordem). Apesar

de ser um método bastante simples, a estimativa da normal pode consumir muito tempo de computação, pois deve ser feito em todos os pontos do volume. Por isso, esse cálculo é normalmente realizado como uma etapa de pré-processamento do volume.

2.3.3 Projeção

Essa etapa no *pipeline* de visualização recebe o volume já iluminado e realiza a projeção dos *voxels* sobre o plano da imagem. Tipicamente, a projeção é realizada através do lançamento de um raio a partir de cada *pixel* da tela (plano de projeção), determinando, através da composição da cor iluminada e opacidade dos *voxels* atingidos pelo raio, a cor final do *pixel*.

A projeção pode ser paralela ou em perspectiva. Entretanto, a projeção em perspectiva possui algumas desvantagens. Uma das mais graves é o problema da divergência dos raios. A densidade de raios, i.é., o número de raios lançados por *voxel*, diminui a medida que se caminha sobre o raio. Ou seja, a amostragem realizada nos *voxels* mais próximos do observador é mais detalhada que a realizada nos *voxels* mais distantes. Isso significa que pequenos detalhes são perdidos ao serem projetos os *voxels* mais distantes. Para resolver este problema, pode-se usar a força bruta, aumentando o número de raios lançados (*oversampling*). Obviamente, este método é bastante ineficiente. Uma outra forma é utilizar um algoritmo adaptativo, aumentando-se a densidade de raios a medida que se afasta do observador [Novins e outros, 90].

3 INTERAÇÃO 3D

Existem diversas ferramentas para o desenvolvimento de aplicações gráficas interativas em 2D. A arquitetura dessas ferramentas já está bastante sedimentada devido a intensa pesquisa já realizada na área [Hartson, 89; Szekely e outros, 93; Vlissides e Linton, 90; Carneiro e outros, 97]. Entretando, o mesmo ainda não acontece com as ferramentas interativas em 3D.

Ferramentas, tais como *OpenGL* e *PHIGS+*, são adequadas para trabalhar com *rendering*, porém pouco, ou quase nada, existe para tratar o *input*, ou seja, das tarefas de interação em si. Estes sistemas são, essencialmente, ferramentas de desenho. Tradicionalmente, os objetos são definidos como uma seqüência de desenhos (pontos, linhas etc.). Nenhum tipo de abstração é oferecida, além de *display lists.* As principais dificuldades encontradas estão ligadas com a visualização e manipulação em 3D [Ware e Jessome, 98].

A visualização em 3D é muito mais complexa pois a visão espacial de uma cena tridimensional é bastante prejudicada pelo uso de dispositivos de saída essencialmente bidimensionais. Da mesma forma, a manipulação direta em 3D dos objetos da aplicação utilizando os dispositivos de entrada disponíveis no mercado também é uma tarefa bastante complexa. Isto ocorre porque o ambiente tridimensional tipicamente oferece mais graus de liberdade que o dispositivo de entrada consegue manipular. Por exemplo, especificar a posição e orientação de um objeto em 3D utilizando o *mouse* pode ser complicado pois apenas dois graus de liberdade podem ser utilizados por vez. Quando isso acontece, é necessário compor várias tarefas de interação com menos graus de liberdade, ao invés de utilizar manipulação direta.

Além disso, as técnicas de interação em 3D estão intrinsicamente relacionadas com a aplicação em questão. Uma aplicação de Cirurgia Assistida por Computador [Robb e outros, 99] tem requisitos de interface muito diferentes de um modelador de sólidos, por exemplo. Daí a dificuldade de estabelecer um consenso com relação as técnicas de interação em 3D.

Com relação aos dispositivos de entrada, pode-se, através de recursos de *software*, simular mais graus de liberdade em um dispositivo bidimensional ou utilizar um

dispositivo de entrada essencialmente tridimensional. Vários exemplos podem ser encontrados na literatura, tais como o *spaceball* e *data glove* [Foley e outros, 90], *roller mouse* [Venolia, 93] e *bat* [Ware e Jessome, 88].

Apesar desses dispositivos capturarem mais graus de liberdade, sendo assim compatíveis com o ambiente tridimensional, eles apresentam várias limitações que ainda impedem seu efetivo uso. Uma delas é o custo elevado que impede a popularização. Além disso, a precisão normalmente não é muito boa e ainda pode causar muita fadiga para o usuário. Segundo Kettner [Kettner, 94], o uso de dispositivos bidimensionais tradicionais como o *mouse*, ainda é a solução mais empregada devido a vários motivos:

- São mais simples, baratos e populares;
- Causam menos fadiga ao usuário pois pode-se apoiar o braço sobre a mesa;
- Normalmente o usuário não fica o tempo todo interagindo em 3D, pois tipicamente também utiliza interface 2D baseada em janelas, menus, botões, caixas de diálogos etc. A constante mudança de interação 2D para 3D e vice-versa restringe o uso de dispositivos 3D pois, em uma interface 2D, eles podem não ser adequados;
- É mais difícil trabalhar com mais graus de liberdade, além de ser menos preciso.

3.1 Formas de Interação

O termo *interação com o usuário* é bastante abrangente e envolve diversos aspectos. Em [Foley e outros, 1990] tem-se uma classificação das formas de interação baseada em duas categorias:

- *Basic Interaction Tasks* (BITs). São tarefas indivisíveis de interação, i.é., correspondem à especificação de uma unidade de informação no contexto da aplicação;
- *Composite Interaction Tasks* (CITs). São as tarefas construídas através de uma combinação de *basic interaction tasks.*

Segundo esta classificação, as tarefas básicas de interação (BITs) se dividem em:

- *Position*. Significa especificar uma posição (*x*,*y*) ou (*x*,*y*,*z*) no espaço de visualização da aplicação, geralmente utilizando o *mouse* ou teclado;

- *Select*. Significa escolher um elemento dentro de um conjunto fixo ou variável de opções. Foley e outros fazem uma distinção quanto ao tamanho deste conjunto. Por exemplo, a escolha de uma opção no menu de um programa geralmente utiliza técnicas bem diferentes da escolha de um objeto criado pelo usuário em um típico programa de desenho;
- *Text*. Significa entrar, geralmente através do teclado, um conjunto de caracteres sem nenhum significado especial para a aplicação. Por exemplo, digitar uma legenda de um gráfico;
- *Quantify.* Significa especificar um valor numérico, geralmente dentro de uma determinada faixa. Esta forma de interação comumente utiliza-se de metáforas visuais tais como potenciômetros, barras de rolamento e réguas.

A composição destas tarefas básicas permite construir formas mais elaboradas de interação que são as *Composite Interaction Tasks* (CITs):

- *Dialogs Boxes*. Utiliza bastante metáforas visuais tais como botões, listas, campos para entrada de dados, janelas, etc;
- *Construction*. Utilizada principalmente para criação de objetos na aplicação como, por exemplo, linhas, retângulos e outros objetos gráficos;
- *Manipulation.* Geralmente utilizada para modificação na forma geométrica dos objetos criados.

Este trabalho não aborda todas as formas de interação apresentadas acima. Enfoca pricipalmente as formas compostas de construção e manipulação. Mais especificamente, este trabalho trata de tarefas de interação em um *canvas* 3D (uma região dentro da janela da aplicação utilizada para visualização da cena). Uma tarefa de interação permite ações do usuário sobre representações gráficas dos objetos da aplicação, principalmente com o objetivo de posicionar e transformar os objetos e também investigar o conteúdo da cena, nesse caso, uma cena contendo dados volumétricos. De uma forma geral, as seguintes tarefas de interação são comuns em um ambiente 3D [Herndon e outros, 94]:

- Criação de objetos;
- Seleção de objetos;

- Posicionamento e edição de objetos. Transformações geométricas afins (translação, rotação, mudança de escala, cisalhamento, alinhamento etc.), mudança de forma (edição de nós), mudança de outros parâmetros (cor, material etc.);
- Controle da câmera;
- Percepção. Extração de informação cognitiva da cena;
- Programação. Definição do comportamento dos objetos e de sua relação com os demais.

É importante notar que, segundo a classificação de Foley e outros, uma tarefa básica de interação em 3D é obtida através de composição de tarefas básicas de interação em 2D. Por exemplo, a especificação de um ponto em 3D normalmente é realizada através de uma seqüência de tarefas básicas em 2D. Entretanto, isso não deve ser confundido com uma tarefa composta, pois semanticamente é uma tarefa básica, já que a especificação de um ponto em 3D corresponde a uma unidade de informação no contexto da aplicação, ou seja, essa tarefa não é divisível.

Com o objetivo de facilitar ou aumentar a precisão, principalmente nas tarefas compostas de interação, pode-se utilizar diversos recursos auxiliares, muito comuns nos sistemas atuais [Hearn e Baker, 94]:

- Alinhadores contínuos. São utilizados para restringir o movimento do *cursor* sobre uma geometria pré-definida, como um segmento de reta ou plano. Essa técnica é comumente utilizada para facilitar a interação quando o dispositivo de entrada não oferece controle de todos os graus de liberdade necessários;
- Alinhadores discretos. Semelhante ao anterior, porém restringe o movimento do *cursor* em pontos discretos sobre a geometria. No caso de um plano, os pontos podem estar igualmente espaçados, formando uma grade de alinhamento;
- Campo de Gravidade. Permite que o *cursor* seja atraído para determinadas posições a medida que este se aproxima delas. Muito utilizado para a construção de objetos em 3D, pois permite que o usuário não se preocupe com o posicionamento preciso do *cursor* para, por exemplo, construir um polígono fechado. É bastante eficiente e natural para os usuários, sendo que muitos utilizam esse recurso sem perceber que ele existe [Herndon e outros, 94];

- Elástico (*rubber band*). Muito utilizado durante a construção de um objeto especificado por pontos de controle. É um mecanismo muito importante de *feedback*, permitindo que o usuário acompanhe a construção do objeto;
- Arrastadores (*draggers*). São também objetos importantes para o *feedback*. Possuem representação geométrica própria, que é inserida na cena, permitindo que, durante a manipulação de um arrastador (através do movimento do *cursor*), este é atualizado e, possivelmente, o objeto da aplicação associado ao arrastador também sofre atualização de acordo com o tipo da interação.

3.2 Requisitos básicos de uma ferramenta de Interação 3D

Apesar de ainda não existir um consenso sobre a arquitetura mais adequada de uma ferramenta para auxiliar a construção de interfaces interativas 3D, algumas questões básicas devem ser levadas em consideração no desenvolvimento de tais ferramentas.

Um ponto fundamental é garantir a reusabilidade da ferramenta. Isso pode ser obtido caso ela seja responsável apenas pelas tarefas independentes da aplicação. De uma certa forma, isso pode reduzir o poder de expressão da ferramenta, pois a comunicação aplicação/ferramenta pode ocorrer em um nível de granularidade baixo, pouco contribuindo para aumentar a abstração da interação 3D.

O problema de disponibilizar na ferramenta apenas os recursos independentes da aplicação também está diretamente associado com a questão do modelo de dados utilizado pela ferramenta. Para manipular interativamente os objetos da aplicação, a ferramenta precisa conhecer algumas características desses objetos. Isso é essencial para qualquer mecanismo de *feedback*. Entretanto, definir um limite adequado para o poder de modelagem dos objetos da aplicação pela ferramenta não é uma tarefa muito fácil.

Caso a ferramenta possua pouco conhecimento dos objetos da aplicação, ou seja, seu modelo de dados é simples, facilita bastante a reusabilidade, pois promove a desejável independência entre a ferramenta e a aplicação. Entretanto, aumenta a complexidade da comunicação aplicação/ferramenta e diminui a abstração da interação.

Por outro lado, embutir na ferramenta uma modelagem muito forte dos objetos da aplicação pode aumentar bastante o nível de abstração da interação, deixando a aplicação mais livre para tratar de questões mais importantes. Entretanto, isso também pode acarretar em duplicação de dados, caso a modelagem oferecida pela ferramenta não seja

satisfatória para a aplicação. Note que isso também afeta a reusabilidade. Em sistemas de visualização volumétrica, onde os dados tipicamente ocupam muito espaço em memória, isso pode ser uma limitação muito grande. Nessa situação, a ferramenta só vai ser realmente útil caso os objetos da aplicação possam ser facilmente mapeados no modelo de dados da ferramenta.

Um outro ponto importante no projeto de uma ferramenta de interação 3D é se preocupar em separar a sintaxe da semântica da interação [Malheiros e outros, 98]. Por exemplo, a ferramenta pode implementar um conjunto de elementos gráficos manipuláveis através de arrasto pelo usuário. A sintaxe da interação define como esses elementos podem ser arrastados. Note que isso pode ser implementado totalmente pela ferramenta. Entretando, a semântica da interação somente pode ser dada pela aplicação. Por exemplo, a aplicação pode mapear tal arrasto em uma mudança de escala um objeto da aplicação. Em uma outra situação, pode mapear o arrasto em uma translação. Além disso, a aplicação pode impor restrições ao arrasto de forma a atender às suas necessidades.

3.3 Ferramentas básicas de Interação 3D

A seguir serão apresentadas algumas ferramentas que permitem a construção de interfaces 3D baseadas em manipulação direta. Cada uma delas é importante, pois os conceitos envolvidos serão úteis para identificar os requisitos básicos de interfaces interativas em ambientes volumétricos.

Nota-se claramente essas ferramantas procuram explorar diversos modelos, técnicas de interação e arquiteturas diferentes, em função do sistemas gráficos e *hardware* suportados. Apesar de intensa pesquisa, o estado atual de desses sistemas está muito aquém dos equivalentes em 2D.

Basicamente, as ferramentas de interação 3D requerem a utilização de bibliotecas gráficas de baixo nível e mesmo assim ainda não há um padrão de arquitetura nesses sistemas. No futuro pode haver ferramentas genéricas para a criação de interfaces 3D para todos os domínios de aplicações. Entretanto, essas ferramentas deverão suportar uma ampla variedade de tarefas de interação [Herndon e outros, 94].

3.3.1 Open Inventor

O sistema apresentado pelos autores [Strauss e Carey, 1992; Wernecke, 1994; Sharma, 1995] é basicamente um *toolkit* orientado a objetos (utilizando C++) para dar suporte ao desenvolvimento de aplicações 3D baseadas em manipulação direta. O *toolkit* oferece um conjunto bastante amplo de objetos gráficos, um tratador de eventos e diversos mecanismos de interação.

A idéia central é permitir que o usuário defina toda a cena utilizando unicamente os objetos oferecidos pelo *toolkit*, evitando assim a duplicação de dados, onde um objeto da aplicação precisa ser convertido para um objeto equivalente do *toolkit* e vice-versa. Naturalmente, o *Inventor* oferece meios para que o usuário crie seus próprios objetos. Os objetos do *toolkit* são manipulados de um maneira semelhante, permitindo que todas as técnicas de manipulação direta oferecidas pelo *Inventor* sejam utilizadas sem esforço adicional.

A base de todo o *toolkit* está construída em torno de um *Scene Database*, onde é armazenada a representação 3D de todas as primitivas, chamada de *nodes.* Uma cena 3D é tipicamente armazenada em uma estrutura do tipo grafo dirigido acíclico (Veja Figura 3.1).

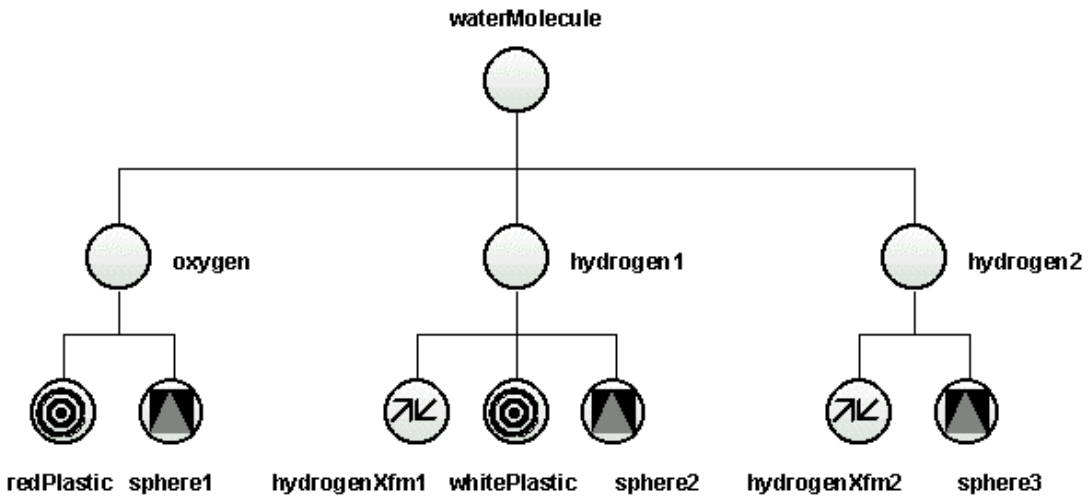


Figura 3.1: Exemplo de cena no *Open Inventor*.

Vários tipos de *nodes* são definidos, desde primitivas gráficas (*shape nodes*, tais como cubos, cones, esferas, *nurbs*, objetos multi-facetados etc.), *nodes* de propriedades descrevendo materiais (por exemplo, plástico, metal etc.) e transformações geométricas (matrizes de translação, rotação etc.) e *nodes* que agrupam outros *nodes*. Toda a parte de *rendering*, exigida pela maioria dos *nodes*, é realizada através do *OpenGL* [Neider e Woo, 1993].

Um ponto interessante nesta organização é que os mais diversos tipos de *nodes* são armazenados em uma mesma estrutura de dados, unificando e facilitando bastante a forma de tratamento destes objetos. Uma crítica, entretanto, pode ser feita nesta organização, pois uma aplicação que utiliza o *Inventor* fica sob a dependência do *toolkit*. Isto, às vezes, pode não ser razoável, visto que nem sempre esta é a melhor estrutura de dados para aplicação.

Uma vez que toda a cena está armazenada no *Inventor*, o *toolkit* oferece um conjunto de operações, chamada de *actions*, sobre a estrutura de dados. Por exemplo, existem *actions* para redesenhar a cena, calcular *bounding box* de um grupo de objetos, fazer *picking* através de raio, escrever a cena em um arquivo etc. Todas estas operações são realizadas sem nenhum custo adicional de programação, visto que todos os objetos são controlados pelo *toolkit*.

Cada *node* possui um conjunto de dados, chamado *fields*, que representa os aspectos da primitiva. Por exemplo, um *node* que representa estilo de linha, possui os *fields* que definem a espessura, padrão (tracejado, pontilhado, contínuo etc.) etc. Os *fields* só podem ser alterados ou consultados através de métodos específicos da classe do *node*.

O *Inventor* permite a conexão de diversos *fields* a outros através de objetos especiais, chamados *engines*. Uma *engine* pode ser considerada como uma "caixa-preta" que realiza algum tipo de computação. Tipicamente, uma *engine* possui várias entradas (que normalmente estão conectadas a *fields* de vários *nodes*) e uma saída, que é o resultado da computação de alguma função nos dados de entrada. A saída pode ser conectada a um *field* de algum outro *node*. Este mecanismo é bastante poderoso, pois permite manter restrições nos *fields* dos *nodes*. A aplicação principal deste recurso é na realização de animações.

Um outro tipo de *node* oferecido pelo *toolkit*, chamado *sensors*, permite monitorar a cena e identificar algum tipo de alteração, por exemplo, mudança de um

determinado *field* em um *node*. O *sensor* avisa a aplicação através de um mecanismo simples de *callback*.

O *toolkit* oferece um mecanismo simples de tratamento de eventos. Todos os eventos (de *mouse*, teclado etc.) são capturados e transformados em formato independente do sistema de janela. Depois, o evento é distribuído pela cena até que se encontre algum *node* capaz de tratar o evento. Neste modelo, somente alguns tipos de *nodes* são "espertos" o bastante para tratar eventos. Estes *nodes* são, usualmente, *nodes* que permitem controle da interação (por exemplo, através de manipulação direta). Os mais comuns são os *nodes* chamados *draggers* e *manipulators*.

Os *draggers* são os objetos mais simples capazes de tratar eventos e controlar a interação. Eles possuem uma interface própria, inserindo sua geometria diretamente na cena. Todas as interações são da forma *<click-drag-release>*. O modo mais comum de utilizar um *dragger* é conectando seus *fields* aos *fields* de outros *nodes* (através de uma conexão direta ou, de uma maneira mais flexível, através de *engines*). Como são objetos simples, a idéia é compor vários *draggers* para realizar uma tarefa mais complexa. Por exemplo, existem *draggers* para realizar translação, escala e rotação. Compondo os três, pode-se realizar as três tarefas simultaneamente.

Manipulators são, por sua vez, *nodes* bem mais completos, constituindo-se a forma mais adequada de realizar manipulação direta no *Inventor*. Tipicamente, um *manipulator* é internamente implementado através da composição de vários *draggers*. A diferença básica é que os *manipulators* são subclasses de outros *nodes*. Isto permite que um determinado *node* da cena seja substituído por uma outra versão, "interativa", no caso, um *manipulator*. Por exemplo, existe um tipo de *node* que realiza transformações geométricas. Este *node*, essencialmente, contém não muito além do que de matrizes de transformação que, quando aplicadas a um outro *node*, modfica sua posição, tamanho e orientação. Existe uma outra versão deste *node*, no caso um tipo de *manipulator*, que permite tratar eventos e controlar a interação. Basta, portanto, que o *node* "não-interativo" seja substituído por esta nova versão "interativa" (Veja Figura 3.2).

Recentemente, a *Silicon Graphics*, responsável pela concepção do *Inventor*, está investindo em uma nova *API*, chamada de *OpenGL Optimizer*. Essa nova ferramenta pretende compatibilizar os requisitos de desempenho, fundamentais para as aplicações de CAD/CAM/CAE.

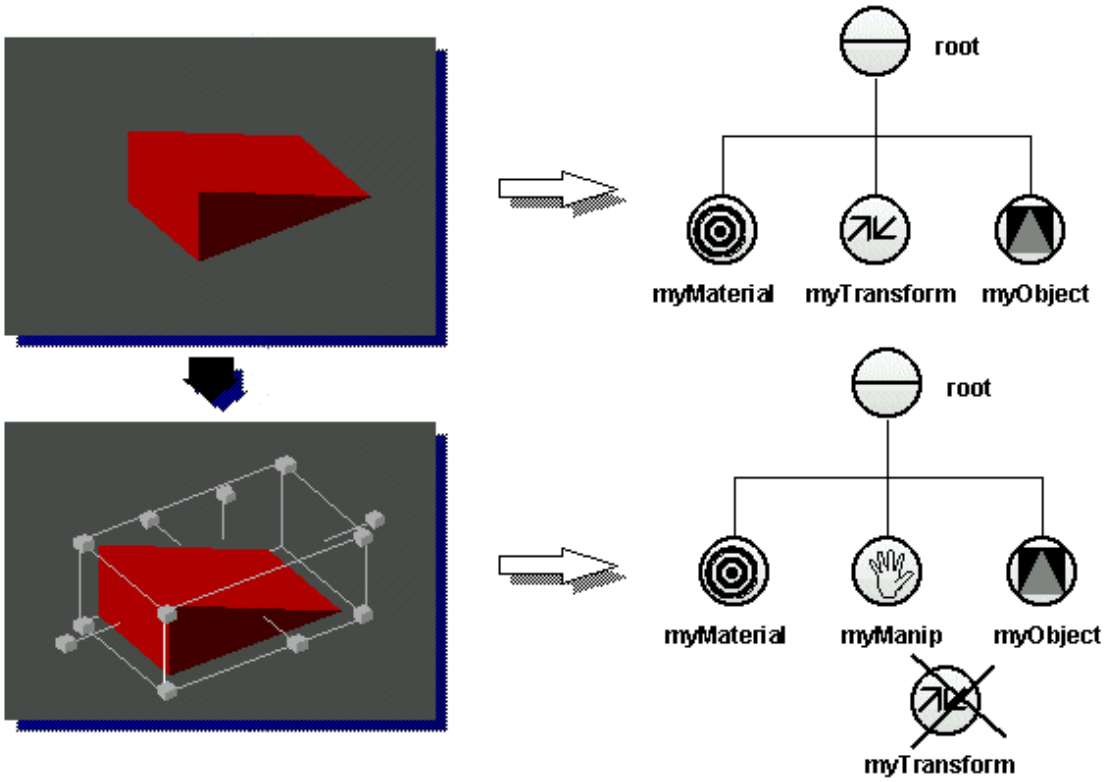


Figura 3.2: Exemplo de *manipulator* no *Open Inventor*.

3.3.2 Virtual Reality Modelling Language (VRML)

VRML é uma linguagem basicamente utilizada para descrever cenas e animações em 3D principalmente para a *World Wide Web*. Um arquivo em VRML contém uma descrição hierárquica de uma cena em 3D baseada em um conjunto de nós que especificam formas, luzes, sons, animações etc. O formato deste arquivo é praticamente o mesmo adotado pelo sistema *OpenInventor* da *Silicon Graphics.*

De uma forma geral, a interação com o usuário em VRML é realizada através do tratamento de eventos. Eventos são gerados por nós especiais na cena, chamado de sensores. Os sensores geram eventos quando o usuário interage com ele, por exemplo, tocando ou arrastando o *cursor* sobre um sensor ou então sem a interferência do usuário, através de eventos gerados em função do tempo. Diversos sensores estão disponíveis na linguagem. Os mais importantes são descritos a seguir:

- `TouchSensor`. Monitora nós de geometria e envia eventos quando o usuário pressiona o botão do *mouse* sobre o objeto, quando move o *cursor* sobre o objeto, quando o ponto de contado do *cursor* sobre o objeto muda, quando a normal muda etc.;

- `PlaneSensor`. Gera eventos de translação 2D sobre o plano XY local do sensor;
- `CylinderSensor`. Gera eventos de rotação sobre um cilindro imaginário cujo eixo de simetria é o eixo Y do sistema de coordenadas local do sensor;
- `SphereSensor`. Gera eventos de rotação sobre uma esfera imaginária centrada na origem do sistema de coordenadas local do sensor e raio igual a distância do ponto apontado pelo *cursor* a origem.

Vários nós em VRML possuem campos especiais que geram eventos ou recebem eventos gerados em outros nós. Para isto, para que os eventos possam fluir pela cena, deve-se construir uma ligação entre nós que geram e recebem eventos. O comando `ROUTE` é utilizado para esta finalidade. Basicamente, quase todos os nós VRML geram eventos e/ou estão aptos a receberem eventos. Através deste mecanismo, podem ser criadas diversos tipos de animações e controles.

Para facilitar o desenvolvimento de mundos interativos em VRML, foi proposta a criação de um nó especial que oferece suporte a programação. Isto é realizado através de nós *Scripts*, que utiliza a linguagem interpretada *VRMLScript*, semelhante a *JavaScript*, porém muito mais simples e eficiente. Um programa em *VRMLScript* pode ser incluído no próprio arquivo em VRML.

Através do uso de sensores e nós *Scripts*, pode-se implementar diversos *widgets* 3D em VRML, por exemplo, *widgets* para realizar translação, rotação e mudança de escala. A Figura 3.3 ilustra um *widget* implementado em VRML. Pode-se realizar rotações (através das esferas vermelhas) e translações (através dos cubos verdes). Os detalhes de implementação deste e de outros *widgets* em VRML pode ser encontrado em [Carneiro, 97].

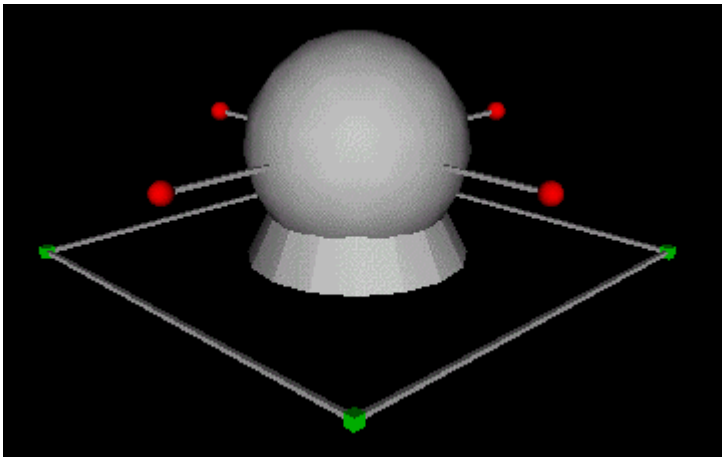


Figura 3.3: *Widget* implementado em VRML [Carneiro, 97].

3.3.3 Widgets 3D / Brown University

Este trabalho [Stevens e outros, 1994] discute a arquitetura de um *toolkit* para a construção de *widgets* 3D, ilustrações interativas e interfaces 3D. Este *toolkit* oferece manipulação direta de primitivas 3D, utilizando mecanismos de restrições, tudo isto através de uma linguagem visual.

Os autores criticam a forma tradicional de construção de interfaces, baseadas em *toolkits*, onde os objetos são criados com o uso de bibliotecas e linguagens tradicionais de programação (C, C++). Deste modo, somente programadores podem utilizar tais ferramentas. Os autores abordaram este problemas através da especificação de uma linguagem visual, permitindo que programadores e não programadores utilizem o sistema.

O *toolkit* proposto foi utilizado para realizar experimentos, tais como criação de ilustrações matemáticas (utilizadas como ferramenta de ensino), criação de *widgets* 3D, ferramentas de visualização científica e ferramentas de modelagem de peças mecânicas.

Essencialmente, o *toolkit* define um conjunto de primitivas básicas, chamadas de *Classes*. Essas primitivas são: *Point*, *Vector*, *Plane* e *Graphic*. Os objetos são instanciados dinamicamente, através da classe que representa a primitiva. Por sua vez, cada classe possui um conjunto de dados, chamados *Slots*, que representam as características da classe. Por exemplo, uma primitiva da classe *Point* tem um único *slot*, chamado *position*, que identifica a posição do ponto. Já uma primitiva da classe *Vector*,

possui os *slots position*, *direction* (direção) e *length* (comprimento). As primitivas da classe *Graphic* representam objetos volumétricos, tais como cubo, esfera etc.

A idéia central deste trabalho é permitir a definição de restrições (*constraints*) entre *slots* de primitivas. Esta operação é chamada de *linking*. Para exemplificar o mecanismo, a posição de uma primitiva da classe *Vector* (*slot position*) pode estar restrita a posição de uma primitiva da classe *Vector* (*slot position*), conforme indicado na Figura 3.4. Assim, o vetor ficará ancorado ao ponto. A Figura 3.5a mostra esta operação antes do *linking*, enquanto a Figura 3.5b mostra o resultado após o *link* ser estabelecido.

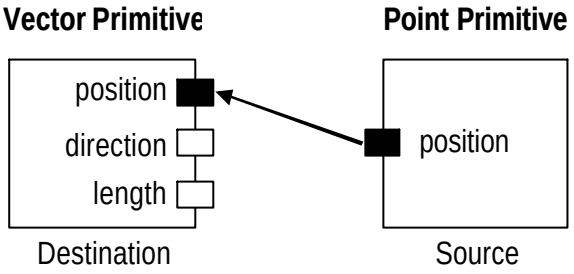


Figura 3.4: Operação de *linking* entre *slots* [Stevens e outros, 94]

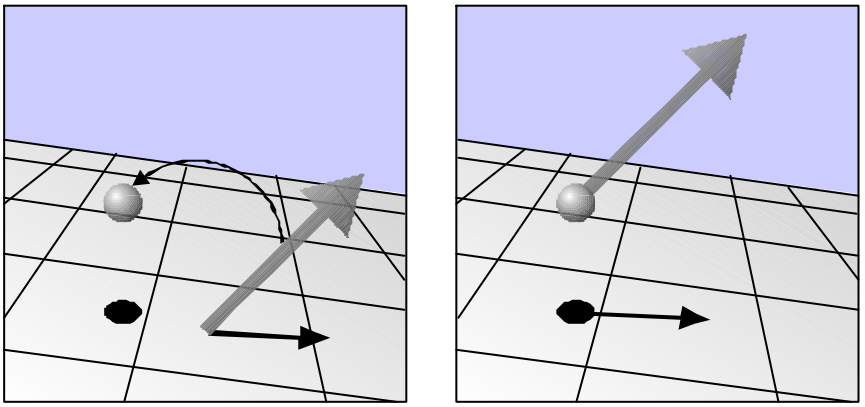


Figura 3.5: Efeito da operação de *linking* [Stevens e outros, 94].

O mecanismo utilizado para manter as restrições é baseado na definição de métodos simples de *inquiry* e *assignment*, para consultar e modificar o valor de um *slot*. Técnicas mais sofisticadas, tais como *numerical solvers*, foram abandonadas por questões

de performance e complexidade. O interessante é que todo o mecanismo de definição das restrições é realizada através da linguagem visual.

O *toolkit* implementa um conjunto de técnicas de interação bem simples, basicamente translações e rotações. Usualmente, cada tipo de primitiva suporta uma forma diferente de interação. Por exemplo, um ponto pode ser deslocado livremente pelo espaço e não pode ser rodado, um vetor, entretanto, pode sofrer rotações. Obviamente, este comportamento vai depender do mecanismo de restrições. O usuário tem a liberdade de escolher a técnica de interação mais adequada para cada tipo de *linking*.

Obviamente, o *toolkit* oferece mecanismos para estender o conjunto básico de primitivas. Para isto, as primitivas podem ser combinadas ou, na terminologia dos autores, encapsuladas. Existem duas formas: *Structurally Encapsulation* e *Class Encapsulation*. Essencialmente, a diferença é que na *Class Encapsulation* uma nova classe é associada às primitivas encapsuladas. Por exemplo, considere a existência de duas primitivas da classe *Point* e uma primitiva da classe *Vector*, conectando os dois pontos. Caso essas três primitivas sejam *class encapsulated*, uma nova primitiva será gerada, isto é, já não se trata de dois pontos e um vetor. Desta forma, novas classes podem ser criadas através de encapsulamento. Inclusive, os *slots* podem ser renomeados e o mecanismo de interação pode ser modificado.

3.3.4 MTK

Este trabalho [Malheiros e outros, 98] apresenta uma ferramenta geral que oferece um conjunto de tarefas básicas de interação 3D que podem ser expandidas de forma a implementar mecanismos sofisticados de manipulação direta em 3D. Esta ferramenta foi inspirada no paradigma MVC (Modelo/Visualização/Controle) para projetos de interfaces gráficas com o usuário.

Muitas idéias dessa ferramenta foram baseadas no *Open Inventor*, porém não há uma integração tão forte da estrutura de dados da aplicação com a estrututa de dados da ferramenta. O MTK oferece uma solução intermediária entre a complexidade do *Scene Database* do *Open Inventor* e a simplicidade de uma *display list* do *OpenGL*. Segundo os autores, o MTK oferece uma *display list* mais inteligente que a do *OpenGL*, permitindo que a ferramenta conheça o suficiente da representação 3D dos objetos da aplicação de forma a possibilitar manipulações diretas dessas representações. Os objetos oferecidos

pela ferramenta incluem um conjunto simples de primitivas gráficas, tais como pontos, linhas e polígonos.

Os autores têm a preocupação de estabelecer uma separação clara entre modelo de dados da aplicação que, obviamente, é dependente da aplicação, e as técnicas de interação e visualização. Com isso, a ferramenta fica responsável apenas pelas funcionalidades independentes da aplicação. Com isso, promove-se o reuso de *software*.

O componente de visualização no MTK é similar aos demais *toolkits* que utilizam *display lists*, ou seja, a aplicação precisa converter seus objetos para a estrutura particular da ferramenta, entretanto esse processo é simples. Além disso, ele oferece alguns guias visuais para auxiliar a percepção 3D do usuário, tais como eixos e grades. Mecanismos mais sofisticados, tais como *groundplane* [Gleicher, 93], sombras interativas [Herndon e outros, 92] e uso de sons [Herndon e outros, 94], não foram implementados.

Com relação ao componente de interação, a idéia adotada é bem semelhante ao *Open Inventor*. A ferramenta oferece um conjunto de interadores e manipuladores. Os interadores são objetos virtuais com uma representação gráfica 3D que dá suporte a tarefa básica de posicionamento 3D. A idéia principal é permitir a interação com os objetos da aplicação de uma forma indireta, através da interação com os interadores, que inserem sua geometria na cena. A Figura 3.6 ilustra o interador *boundingboxdragger*, que pode ser utilizado para realizar rotação, mudança de escala e translação.

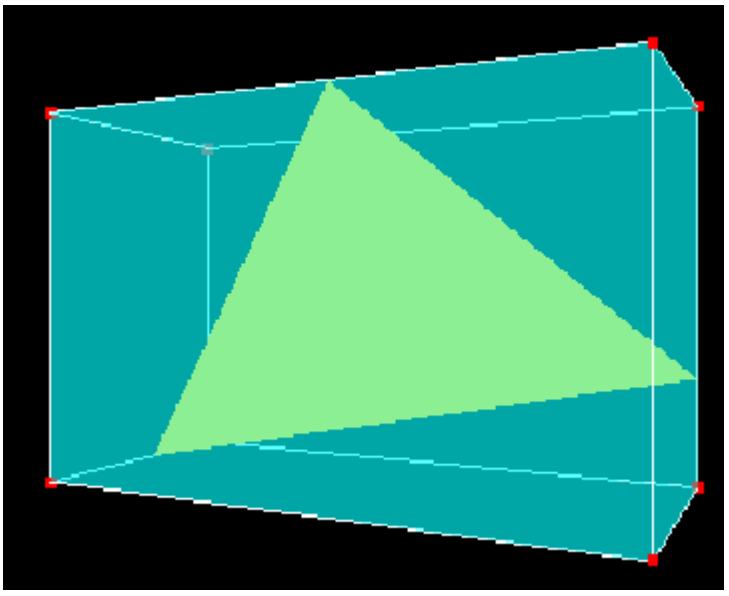


Figura 3.6: Interador *boundingboxdragger* [Malheiros e outros, 98].

Os interadores são objetos passivos formados por diversas partes sensíveis a eventos do dispositivo de entrada (*mouse*). Pode-se aplicar restrições a eles, da mesma forma utilizada em [Bier, 90]. Portanto, a interação pode ser restrita a um determinado segmento de reta, plano ou outra superfície (ex. esfera) e pode ser contínua ou discreta.

Uma das principais funções dos interadores é realizar o *feedback* com o usuário durante a interação. Além disso, eles substituem visualmente os objetos da aplicação enquanto estão sendo manipulados pelo usuário. Entretanto, os interadores cuidam apenas da sintaxe da interação não tendo, portanto, semântica própria. Quando sofre uma ação do usuário, o interador executa uma *callback* para indicar uma mudança. Esse mecanismo é utilizado para implementar formas muito mais complexas de interação.

Os manipuladores, por sua vez, não são objetos passivos. Utilizam-se de um ou mais interadores, associados às restrições. Os manipuladores, por serem ativos, coordenam os interadores para realizar uma tarefa mais complexa de interação, além de provocar a atualização do modelo de dados da aplicação. Ao contrário dos interadores, os manipuladores cuidam da semântica da interação pois, conhecendo o modelo de dados da aplicação, podem manipular seus objetos. Além disso, os manipuladores não possuem representação gráfica. Sua geometria depende apenas dos interadores empregados.

De uma forma geral, o manipulador é associado a um objeto da aplicação e fica responsável por ele durante todo o processo de interação, que é do tipo *<click-drag-release>*. Após feita a associação, o manipulador cria os interadores necessários e o usuário começa a realizar ações sobre eles. Os interadores informam suas posições para o manipulador, que pode corrigí-las, de forma a garantir uma operação semânticamente válida no objeto da aplicação manipulado. Essa operação também é aplicada ao modelo de dados da aplicação e no componente visualização, de forma a manter a consistência necessária.

4 INTERAÇÃO 3D EM DADOS VOLUMÉTRICOS

Um dos principais objetivos de uma ferramenta de visualização científica é, sem dúvida, permitir que o usuário extraia informações relevantes a respeito do dado visualizado. Para isto, é necessário que a interface da ferramenta ofereça os recursos necessários para permitir o entendimento do dado. Springmeyer e outros [Springmeyer e outros, 92] realizaram um estudo empírico sobre como os cientistas analisam o dado e fizeram algumas recomendações básicas para o *design* de ferramentas de análise e visualização. Uma das recomendações mais importantes é, sem dúvida, facilitar a visualização exploratória, tanto qualitativa quanto quantitativa, pois ela permite que o usuário extraia informações do dado com muito mais facilidade.

A visualização volumétrica exploratória ainda é um campo de intensa pesquisa tendo em vista que a natureza e as dimensões do dado volumétrico estão se tornando cada vez mais complexas. Isto requer diversos recursos das ferramentas de análise e visualização, tais como capacidade de processamento para permitir respostas rápidas, controle interativo da visualização e um bom *display* 3D [Encarnação e outros, 94].

Segundo Meyer e Globus [Meyer e Globus, 93], a visualização campos escalares 3D (dado volumétrico de dimensão escalar, por exemplo, densidade, temperatura, pressão etc.) pode ser realizado através do uso de três técnicas comuns:

- Planos de Corte. Permite que uma fatia plana seja extraída do dado volumétrico, sendo facilmente visualizada, tendo em vista sua natureza bidimensional;
- Isosuperfícies. Permite que várias superfícies poligonais com o mesmo valor de propriedade sejam extraídas do volume. Um dos algoritmos mais utilizados para este processo é o *Marching Cubes* [Lorensen e Cline, 87]. Devido a facilidade de realizar *rendering* de polígonos (esta capacidade é comumente implementada em *hardware*), as primeiras técnicas de visualização implementadas foram deste tipo;
- Visualização Volumétrica. Permite, conforme mencionado no capítulo 2, gerar uma imagem 2D do volume sem utilizar representações intermediárias, tais como objetos geométricos.

Tendo em vista o aparecimento de máquinas com crescente poder de processamento e o desenvolvimento de algoritmos cada vez mais rápidos e precisos, as técnicas de visualização volumétrica estão sendo cada vez mais utilizadas para a visualização de dados volumétricos. Este capítulo tem o objetivo de apresentar algumas técnicas que permitem a exploração de dados volumétricos.

4.1 *Widgets* 3D

As técnicas de manipulação direta estudadas no capítulo anterior, principalmente os *widgets* 3D, podem ser bastante úteis para dar suporte a exploração do dado volumétrico. O *design* de tais *widgets* deve contemplar os seguintes principios gerais [Meyer e Globus, 93]:

- A geometria do *widget* deve ser clara o bastante para indicar seu comportamento, i.é., seus *handles* (alças) devem sugerir que tipo de interação será realizada através de sua interface (*affordances*);
- Normalmente os *widgets* controlam explicitamente seus parâmetros, através das ações dos usuários sobre os *handles*. Entretanto, em algumas situações outros parâmetros podem ser controlados implicitamente, com a finalidade de simplificar a interface;
- A eficiência de um *widget* pode ser melhorada removendo-se os graus de liberdade desnecessários para a interação;
- A especificação de um *widget* é dependente do uso pretendido para ele. Por exemplo, um *widget* para uso artístico tem requisitos de interface diferentes de um *widget* para uso em engenharia;
- Quando se está interagindo com *widget* através, por exemplo, do arrasto de uma de suas alças, algum mecanismo de *feedback* deve ser utilizado para reforçar a interação;
- A geometria do *widget* deve ser limpa, isto é, não pode esconder o objeto que está sendo manipulado.

Fonseca [Fonseca, 97] propõe o uso de *widgets* 3D para possibilitar a exploração de dados volumétricos. Este trabalho discute uma arquitetura para a construção de tais *widgets* e implementa um *toolkit* seguindo a filosofia de orientação a objetos contendo

um conjunto de manipuladores para visualização interativa de dados sísmicos. Este *toolkit* se baseia em idéias do *Open Inventor*, sendo que algumas classes do *toolkit* têm correspondência direta com o *Open Inventor*. As principais classes para manipulação 3D utilizadas pelo *toolkit* são:

- Manipuladores. Análogo aos manipuladores do *Open Inventor* e MTK. Utilizam funções de *callbacks* para responder aos eventos de interação. Assim como nos demais *toolkits*, os manipuladores não possuem geometria própria;
- Componentes de Arrasto (*Draggers*). Análogo aos *draggers* do *Open Inventor* e interadores do MTK. Possuem geometria própria e reagem diretamente aos eventos de interação provocados pela manipulação de seus *handles* pelo usuário. Também utiliza funções de *callbacks* para tratá-los;
- Projetores. Define objetos que são utilizados pelos componentes de arrasto para projetar, ou seja, restringir o movimento do *cursor* sobre tais objetos (por exemplo, segmentos de retas, planos e cilindros). É utilizado como mecanismo de restrição, semelhante ao adotado pelos demais *toolkits.*

4.1.1 Planos de Corte

Muitas técnicas para a visualização de uma região de interesse do volume têm o inconveniente de ocultar outras partes também importantes, devido a regiões de grande opacidade ou de superfícies que escondem o dado volumétrico. Isto ocorre devido a perda da dimensão de profundidade quando o dado volumétrico tridimensional é projetado em uma superfície bidimensional. Para resolver este problema, é bastante comum a utilização de planos de corte para visualizar apenas um determinada fatia do volume, esquecendo-se das demais.

[Fonseca, 97] implementa um manipulador de planos de corte chamado *SliceMover* que permite a visualização interativas das fatias de um volume. Este manipulador é composto de quatro componentes de arrasto de translação sobre linhas, posicionados nos quatro cantos da fatia. A geometria dos componentes de arrastos sugere o comportamente de um *slider*, ou seja, indicam a posibilidade de movimentos unidimensionais (Figura 4.1).

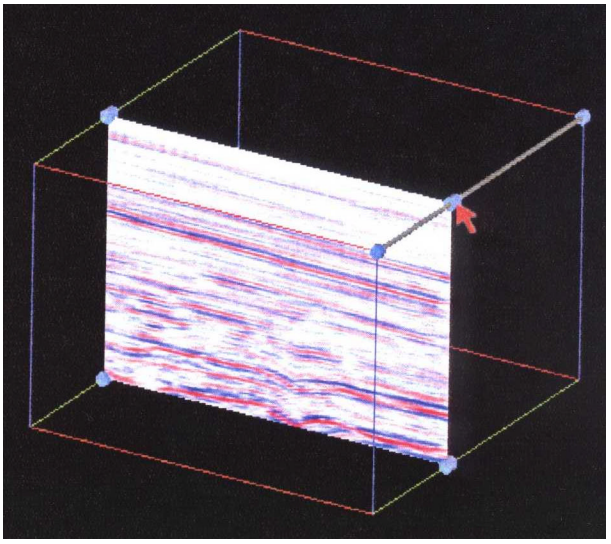


Figura 4.1: Manipulador *SliceMover* [Fonseca, 97]

[Meyer e Globus, 93] apresentam o *widget Cutting Plane* conforme mostrado na Figura 4.2. Este *widget* pode ser manipulado de diversas maneira e possui geometria que incluem *handles* para *resize*, vetor normal ao plano, *up vector*, projeção do *up vector* sobre o plano. O usuário pode mover o plano livremente arrastando o interior do plano. Além disso, pode-se mudar a orientação do plano arrastando-se uma única aresta. O excesso de geometria do *widget* acabou ocultando o dado e o *widget* foi gradualmente simplificado de forma se tornar apenas um retângulo com quatro arestas finas.

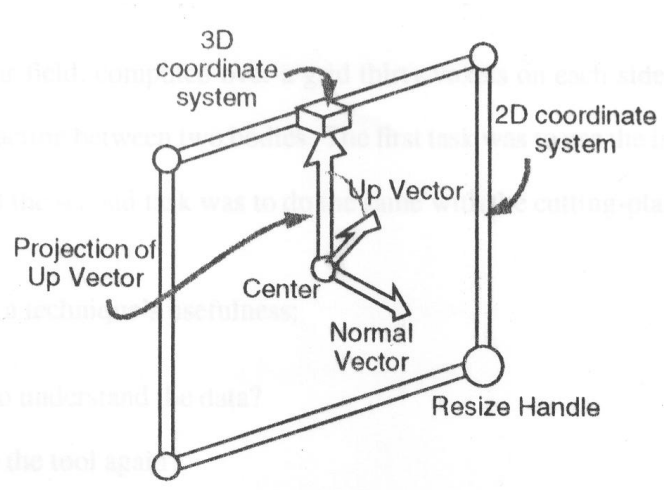


Figura 4.2: *Widget Cutting Plane* [Meyer e Globus, 93].

Outro exemplo de widget para planos de corte é apresentado na Figura 4.3. Este *widget* ilustra o uso simultâneo de três planos de corte utilizando o sistema VTK (*Visualization Toolkit*).

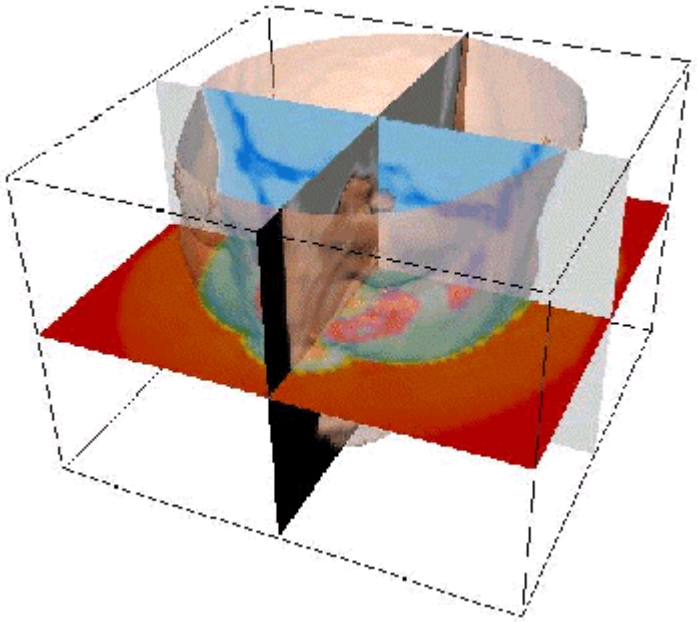


Figura 4.3: Múltiplos planos de corte [Lorensen, 98].

4.1.2 Probes

Este também é um mecanismo bastante usual de visualização interativa. *Probes* nada mais são que objetos geométricos inseridos na cena. Através deles, pode-se investigar quantitativamente o que está acontecendo localmente naquela região. Como são objetos geométricos, os *probes* podem ser facilmente manipulados pelo usuário. Aplicando-se transformações tais como translação, rotação etc., o usuário pode caminhar com o *probe* por todo o volume e investigar seu interior.

Um exemplo bastante simples de *probe* é o *cursor* 3D. Através de translações, o usuário pode posicionar o *probe* em qualquer ponto do volume e obter informações sobre aquele ponto, tais como o valor da propriedade volumétrica, cor, opacidade, vetor gradiente etc. Fonseca [Fonseca, 97] implementa um manipulador deste tipo (Figura 4.4). Apesar de ter sido colocado como uma outra forma de interação, os planos de corte também podem ser considerados *probes*.

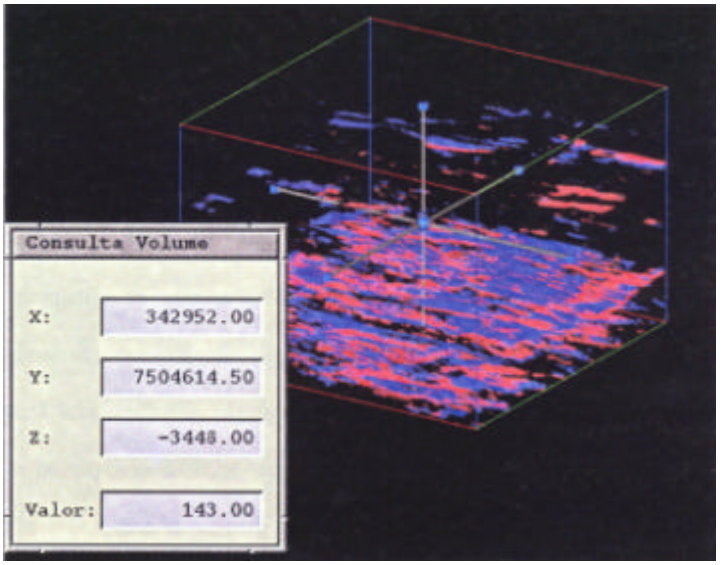


Figura 4.4: *Widget probe* [Fonseca, 97].

A geometria de um *probe* pode ser mais complexa e mais informações podem ser investigadas. Um exemplo de *probe* mais elaborado pode ser encontrado em [Wijk e outros, 94]. Este *probe* é utilizado para inspeção local em fluidos, permitindo extrair informações de velocidade, aceleração, cisalhamento na direção do fluido e torção (Figura 4.5).

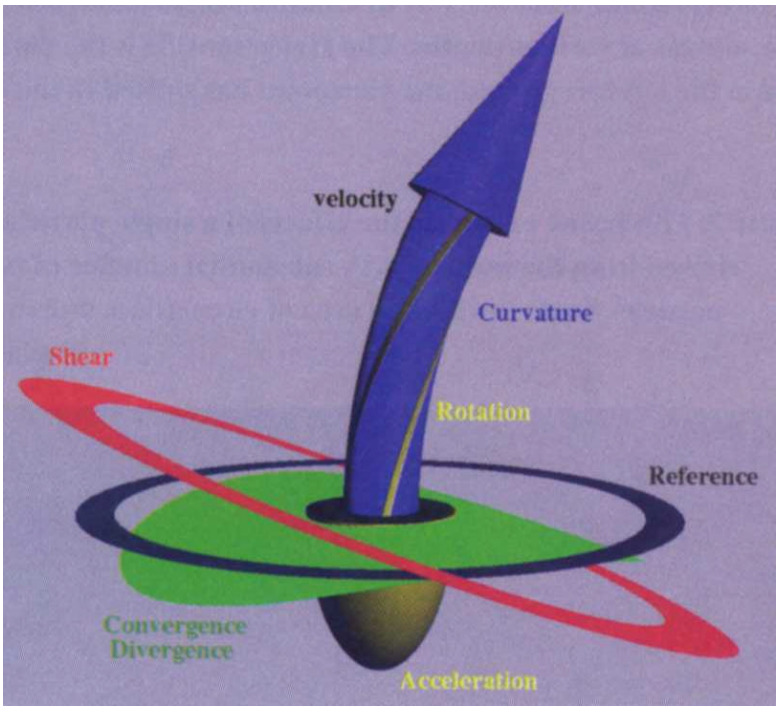


Figura 4.5: *Probe* para investigar fluidos [van Wijk e outros, 94].

Para testar a eficiência do *probe*, os autores desenvolveram uma aplicação interativa (Figura 4.6) que oferece duas janelas de visualização: uma mostrando o volume como um todo e a posição do *probe* e a outra mostrando uma ampliação do *probe*. O usuário pode interagir tanto com o *probe* pelo volume como também com a sua versão ampliada, permitindo que o usuário observe-o de diversos ângulos diferentes.

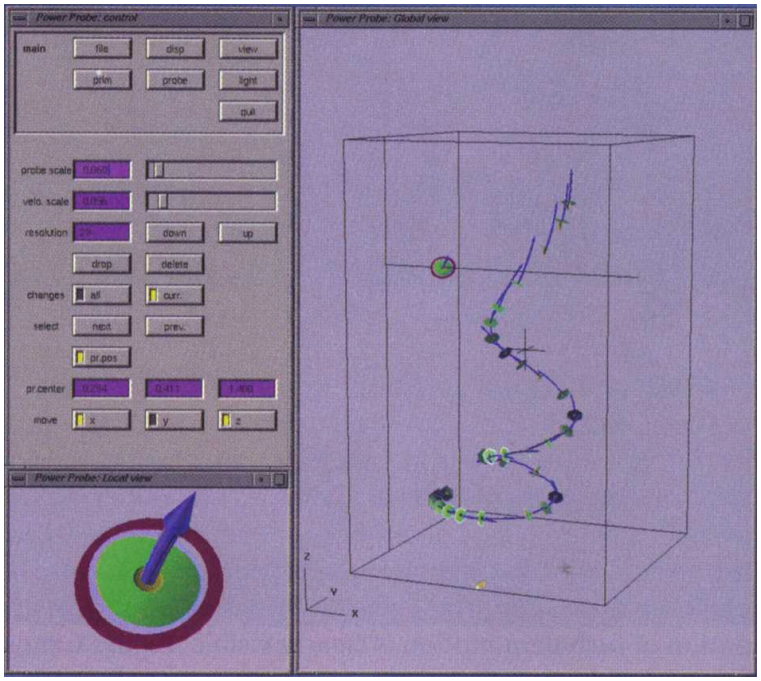


Figura 4.6: Sistema de interface utilizando o *probe* [van Wijk e outros, 94] .

4.1.3 Especificação de subvolumes

Fonseca [Fonseca, 97] implementa um manipulador simples de especificação de subvolumes. Através dele, o usuário pode especificar um subconjunto do volume original (Figura 4.7). Este manipulador é bastante simples e o usuário pode interagir com as alças dos vértices do subvolume (alterando duas dimensões simultaneamente), alças do meio das arestas (alterando uma dimensão do subvolume) ou com a face do subvolume (alterando a dimensão perpendicular à face).

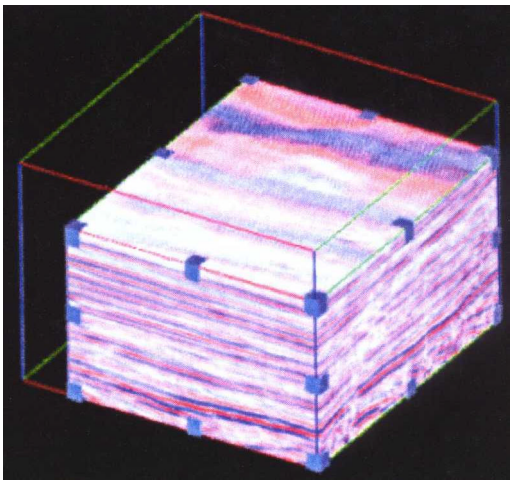


Figura 4.7: Manipulador *Subvolume* [Fonseca, 97].

4.2 Realidade Virtual

Um ambiente de realidade virtual pode ser definido como sendo uma simulação gerada por computador caracterizada por ser imersiva e criar um senso real de presença para o usuário [Meyer e Globus, 93].

Uma das principais vantagens da interação em um ambiente de realidade virtual é permitir ao usuário controlar livremente todos os graus de liberdade necessários, pois as interfaces de realidade virtual podem mostrar a cena 3D sem ambigüidades e oferecem mecanismos de exploração da cena diretamente em 3D, de uma forma mais intuitiva

Entretanto, um dos principais fatores que dificulta a efetiva utilização da realidade virtual é ainda a necessidade de *hardware* rápido o bastante para garantir o *rendering* da cena a uma taxa de pelo menos 10 quadros por segundo. Caso contrário, o usuário perde o efeito da imersão.

Realidade Virtual aplicada a visualização científica ainda é um campo de muita pesquisa e existem poucos sistemas comerciais desenvolvidos até o momento. Por permitir a manipulação direta em 3D, através do uso de dispositivos de entrada apropriados, tais como *data gloves*, *space balls* etc., e visão espacial, através do uso de *head-mounted displays* (HMD's) ou óculos especiais, os sistemas que utilizam realidade virtual abrem portas para métodos de interação muito mais sofisticados que os abordados neste trabalho. Pode-se utilizar métodos muito mais eficientes de *feedback* e navegação através da cena com muito mais realismo para o usuário.

Atualmente, há muito mais campo para estudar novas formas de interação 3D e visualização exploratória utilizando ambientes de realidade virtual do que utilizando os ambientes tradicionais apresentados neste trabalho.

5 CONCLUSÕES

Este trabalho inicialmente apresenta uma visão geral de *rendering* de volumes mostrando os aspectos mais importantes do *pipeline* de visualização volumétrica direta. A seguir é estudado os mecanismos mais usuais de interação 3D, especialmente aquelas que ocorrem no *canvas* da aplicação, onde o usuário realiza ações sobre as representações gráficas dos objetos da aplicação, tais como posicionamento, transformações e investigação do conteúdo da cena. Alguns recursos que facilitam a interação, tais como alinhadores e campo de gravidade, também são apresentados.

A seguir, é feito um estudo sobre os principais requisitos necessários para o desenvolvimento de ferramentas para o gerenciamento da interação 3D, especialmente com relação ao reuso de *software*. Com relação a este ponto, é importante disponibilizar na ferramenta apenas os recursos que são independentes da aplicação. Isto não é simples pois os mecanismos de interação estudados são muito dependentes da aplicação. Outro ponto importante levantado é a questão do modelo de dados utilizado pela ferramenta. Este ponto pode ocasionar a duplicação de dados na aplicação e na ferramenta, além de dificultar o uso da ferramenta quando seu modelo de dados não for adequado para a aplicação. Foi visto também que a separação entre a sintaxe e semântica da interação é um ponto que deve ser respeitado.

A seguir, o trabalho faz um estudo mais detalhado sobre algumas ferramentas de interação 3D, em especial o *Open Inventor*, VRML, *Widgets* 3D e MTK. Foi visto que muitas delas utilizam o conceito de *draggers*, responsáveis pela sintaxe da interação e *manipulators*, responsáveis pela semântica da interação. Além disso, é comum a utilização de mecanismos de restrições para desprezar os graus de liberdade desnecessários para a interação. Em algumas ferramentas com modelo de dados mais sofisticado (por exemplo, *Open Inventor* e VRML), pode-se ainda especificar dependências para formalizar o controle da interação.

Por fim, o trabalho apresenta um estudo da interação 3D em ambientes volumétricos, sempre procurando identificar as técnicas de interação que possam permitir a visualização exploratória dos dados, já que esse é um dos principais objetivos dos sistemas de visualização volumétrica, principalmente os voltados para a área médica e engenharia. As principais questões referentes ao uso de *widgets* 3D são levantadas,

especialmente a necessidade utilizar geometria simples no *design* de *widgets*, uso de *affordances* para simplificar a interação e remoção dos graus de liberdades desnecessários para o controle da interação. A seguir, os principais mecanismos de visualização exploratória foram apresentados, especialmente os planos de corte, *probes* e manipuladores de subvolumes.

Chegou-se a conclusão que não há muitas novidades na área de visualização volumétrica exploratória, ou seja, as ferramentas utilizadas em sistemas tradicionais de interação também são utilizadas em sistemas de visualização volumétrica. Entretanto, ainda há campo para ser explorado em ambiente de realidade virtual, pois permite o o controle de todos os graus de liberdade livremente, sendo, portanto, mais adequado para o estudo de novos mecanismos de interação 3D.

6 REFERÊNCIAS

[Bier, 90] Bier, E., *Snap-dragging in three dimensions*. In Proceedings of the 1990 Symposium on Interactive 3D Graphics, Volume 24, pp. 193-204, March, 1990.

[Carneiro e outros, 97] Carneiro, M. M., Gattass, M., Levy, C. H., Russo, E. M. R., *Interact: um modelo de interação para interfaces 2D por manipulação direta*, X Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens, 1997.

[Drebin e outros, 88] Drebin, R. A., Carpenter, L., Hanrahan, P., *Volume Rendering*, Computer Graphics, Volume 22, Number 4, pp. 65-74, 1988.

[Elvis, 92] Elvis, T. Tood, *A Survey of Algorithms for Volume Visualization*, Computer Graphics, Volume 26, Number 3, August 1992, pp 194-201.

[Encarnação e outros, 94] Encarnação, J., Foley, J., Bryson, S., Feiner, S. K., Gershon, N., *Research Issues in Percepcion and User Interfaces*, IEEE Computer Graphics and Applications, Vol. 14, No. 2, pp. 67-69, March, 1994.

[Foley e outros, 90] Foley, J. D., van Dam, A., Feiner, S. K., Hughes, J. F., *Computer Graphics: Principales and Pratice*. Addison-Wesley, 1990.

[Fonseca, 97] Fonseca, L. T. S., *Uma Arquitetura para Construção de Ferramentas de Manipulação para Visualização Interativa de Dados Volumétricos*, Dissertação de Mestrado, Departamento de Informática da PUC-Rio, 1997.

[Gallagher, 94] Gallagher, R. S., *Computer Visualization*, editado por Richard S. Gallagher, CRC Press, 1994.

[Gerhardt e outros, 98] Gerhardt, A., Paiva, A., Schmidt, A.E., Martha, L.F.; Carvalho, P.C., Gattass, M., *Aspects of 3-D Seismic Data Volume Rendering*, Proceedings of the GOCAD ENSG Conference – 3D Modeling of Natural Objects: A Challenge for the 2000's, Nancy, França, 04 a 05 de junho de 1998.

[Gleicher, 93] Gleicher, M., *A Graphics Toolkit Based on Differential Constraints*, In Proceedings of the 1993 ACM Symposium of User Interface Technology (UITS'93), pp. 109-120, 1993.

[Gouraud, 71] Gouraud, H., *Continuos Shading of Curved Surfaces*, IEEE Transactions of Computers, Vol. 20, No. 6, pp. 623-629, 1971.

[Hartson, 89] Hartson, R., *User-Interface Management Control and Communication*, IEEE Software, 1989, January, 62-70.

[Hearn e Baker, 94] Hearn, D., Baker, M. P., *Computer Graphics*, Prentice Hall Inc., 1994.

[Herndon e outros, 92] Herndon, K. P., Zeleznik, R. C., Conner, D. B., Snibbe, A,m van Dam, A., *Interactive Shadows*, In Proceedings of the 1992 Symposium on User Interface Software and Technology, pp. 1-6, 1992.

[Herndon e outros, 94] Herndon, K. P., van Dam, A., Gleicher, M., *Workshop on the Challenges of 3D Interaction*, SIGCHI Bulletin, Volume 26, Number 4, pp. 1-9, October, 1994.

[Kaufman e outros, 98] Kaufman, A., Lorensen, B., Pfister, H., Silva, C., Sobierajski-Avila, S., *Advances in Volume Visualization*, Course Notes for Siggraph'98, July, 1998.

[Kettner, 94] Kettner, L., *Theoretical foundations of 3D-metaphors*. In Workshop on the Challenges of 3D Interaction at the CHI'94, February, 1994.

[Lorensen e Cline, 87] Lorensen, W. E., Cline, H. E., *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*, Computer Graphics (Proceedings of SIGGRAPH'87), Vol. 21, No. 4, pp 163-169, 1987.

[Lorensen, 98] Lorensen, B., *Creating Surfaces from Cross-Section Data*, ACM Siggraph'98 Course 23 (3D Visualization in Medicine, Chapter 5), 1998.

[Malheiros e outros, 98] Malheiros, M. G., Fernandes, F. N., Wu, Shin-Ting, *MTK: A Direct 3D Manipulation Toolkit*, SCCG'98, 1998.

[Meyer e Globus, 93] [Meyer, T., Globus, A., *Direct Manipulation of Isosurfaces and Cutting Planes in Virtual Environments*, Technical Report No. CS-93-94, Brown University, Computer Science Department, December, 1993.

[Neider e Woo, 1993] Neider, J., Woo, M., *OpenGL Programming Guide*, Addison-Wesley, Reading, Massachussets, USA, 1993.

[Novins e outros, 90] Novins, K. L., Sillion, F., Greenberg, D. P., *An efficient method for volume rendering using perspective projection*. Computer Graphics, Vol. 24, No. 5, pp 95-102.

[Paiva e outros, 99] Paiva, A. C., Seixas, R. B., Gattass, M., *Introdução à Visualização Cientifica*, Monografia em Ciência da Computação, nº 3/99, Departamento de Informática, PUC-Rio, 1999.

[Phong, 75] Phong, B. T., *Illumination for Computer Generated Pictures*, Communications of ACM, Vol. 18, No. 6, pp. 311-317, 1975.

[Robb e outros, 99] Robb, R. A., Camp, J. J., Hanson, D. P., *Computer Aided Surgery and Treatmeny Planning at the Mayo Clinic*, Biomedical Imaging Resource, Mayo Foundation/Clinic, Rochester, MN, 1999, http://www.mayo.edu/bir/reprints/CAS.html.

[Sharma, 1995] Sharma, R., *Open Inventor: a Toolkit for 3D Graphics*, The X Advisor, Vol. 1, No. 7, December, 1995, http://landru.unx.com/DD/advisor

[Springmeyer e outros, 92] Springmeyer, R. R., Blattner, M. M., Max, N. L., *A characterization of the scientific data analysis process*, In Visualization, pp. 235-242, 1982.

[Stevens e outros, 1994] Stevens, M. P., Zeleznik, R. C., Hughes, J. F., *An Architecture for an Extensible 3D Interface Toolkit*, Dep. of Computer Science, Brown University, April, 1994.

[Strauss e Carey, 1992] Strauss, P., Carey, R., *An Object-Oriented 3D Graphics Toolkit*, Proceedings of the 1992 SIGGRAPH, Computer Graphics, Volume 26 (2), pp. 341-349, July, 1992.

[Szekely e outros, 93] Szekely, P., Luo, P., Neches, R., *Beyond Interface Builders: Model Based Interface Tools*, CHI'93, 383-390, April, 1993.

[Venolia, 93] Venolia, D., *Facile 3D direct manipulation*. In Proceedings of ACM CHI'93 Conference on Human Factors in Computing Systems, Addison-Wesley, pp. 348-355, Amsterdam, abril 1993.

[Vlissides e Linton, 90] Vlissides, J. M., Linton, M. A., *Unidraw: A Framework for Building Domain-Specific Graphical Editors*, ACM Transactions on Information Systems, Vol. 8, No. 3, July, 1990, pp. 237-268.

[Ware e Jessome, 88] Ware, C., Jessome, D. R., *Using the bat: A six-dimensional mouse for object placement*. IEEE Computer Graphics & Applications, 8 (6), pp. 65-70, Novembro, 1988.

[Wernecke, 1994] Wernecke, J., *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor Release 2*, Addison-Wesley, 1994.

[Wijk e outros, 94] Wijk, J. J., Hin, A. J. S, Post, F. H., *Three Ways to Show 3D Fluid Flow*, IEEE Computer Graphics & Applications, Vol. 14, No. 5, pp. 33-39, September, 1994.