Universidade Estadual de Campinas

Faculdade de Engenharia Elétrica e de Computação

Departamento de Engenharia de Computação e Automação Industrial

Uma Arquitetura para Construção de Interfaces com Manipulações Diretas 3D

Autor: **Flávio Navarro Fernandes**

Orientadora: **Profa. Dr-Ing. Wu, Shin-Ting**

Dissertação submetida à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, para preenchimento dos pré-requisitos parciais para obtenção do Título de Mestre em Engenharia Elétrica.

11 de Julho de 1999

Tese: Uma Arquitetura para Construção de Interfaces com Manipulações Diretas 3D

Autor: Flávio Navarro Fernandes

Orientador: Profa. Dr-Ing. Wu, Shin-Ting

Aprovado em 25 de agosto de 1998 pela banca examinadora:

Profa. Profa. Dr-Ing. Wu, Shin-Ting (Presidente)

Prof. Dr. Marcelo Gattass (PUC/Rio)

Prof. Dr. Léo Pini (FEEC/Unicamp)

Agradecimentos

- À professora Ting pela orientação e pelos conhecimentos transmitidos a mim.
- Ao amigo Marcelo Malheiros, pelas discussões sobre Computação Gráfica e pela inestimável ajuda na implementação do MTK.
- À amiga Perla, pelos artigos que me auxiliaram em meu trabalho.
- Ao Mauro e ao Nelson pela grande amizade, ao Nelson em especial, por juntamente com Perla, terem me ajudado a fortalecer minha fé em Jesus.
- Aos professores do DCA, com quem tive o prazer de estudar.
- À CAPES pelo apoio financeiro.
- A todos aqueles que de uma maneira ou de outra, colaboraram com a realização deste trabalho.

A Deus, pela vida.
À memória de meu pai, Flávio,
e a minha mãe, Maria Antonia, pelos primeiros 22 anos.
À Mauriceia, pelos próximos 50.

Resumo

Os sucessivos avanços tecnológicos dos computadores têm feito de sistemas interativos 3D ferramentas cada vez mais comuns em várias áreas de aplicação. Para auxiliar o desenvolvimento de aplicações com capacidade de visualização e manipulação de dados tridimensionais, foram propostas algumas arquiteturas para sistemas de desenvolvimento de interfaces 3D. Apesar destes esforços, tais arquiteturas ainda apresentam limitações, como a alta complexidade de programação, baixa reusabilidade ou grau de interatividade e flexibilidade limitados. O objetivo deste trabalho é apresentar uma arquitutetura que permita a construção de interfaces com manipulações diretas 3D. A arquitetura provê elementos básicos de interação, a partir dos quais é possível criar mecanismos de manipulação direta 3D sofisticados. Tais mecanismos de manipulação direta podem ser facilmente reutilizados, devido ao desacoplamento entre a interface e o modelo específico da aplicação.

Abstract

Recent technological advance in computers have made interactive 3D systems valuable tools in a variety of areas. In order to facilitate the development of 3D interactive graphics applications, several architectures for 3D interface development systems were proposed. In spite of these efforts, these architectures still present some limitations, such as high programming complexity, low reusability, low interactivity, and limited flexibility. This work aims at presenting a framework which allows the construction of 3D direct manipulation interfaces. The framework provides basic interaction elements, which can be used to create more sophisticated 3D direct interaction metaphors. Such metaphors can also be easily re-used in several applications, due to the decoupling between the application-independent interface components and the application dependet model.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Com o aumento do poder de processamento dos computadores, trazendo a potência de *workstations* para o *desktop*, são cada vez mais comuns aplicações que utilizam ambientes e conjuntos de dados tridimensionais. Tais aplicações são ferramentas básicas de projetistas industriais, engenheiros, arquitetos, animadores, publicitários, médicos, entre outros. Contudo, a interação com objetos 3D não é trivial. Conforme Ware e Jessome[42], distinguem-se basicamente dois problemas nessas interações:

- a visualização - como tornar possível a percepção da relação espacial entre os objetos em um ambiente 3D; e
- a manipulação - como manipular os objetos de forma intuitiva, confortável, eficiente e precisa.

Embora existam dispositivos para manipulação e visualização em ambientes de realidade virtual [13, 8, 41, 30] (Apêndice A), problemas como seu alto custo, fadiga visual e muscular, e imprecisão, ainda impedem sua plena difusão em aplicações 3D[17]. Por outro lado, dispositivos 2D são bastante utilizados na interação em ambientes 3D, devido à sua popularidade e baixo custo. Entretanto, a *manipulação direta*[1] em 3D através de dispositivos 2D requer abordagens especiais, diferentes daquelas utilizadas em aplicações bidimensionais, como Corel Draw[20], Photoshop[38] e Xfig[44]. Um dos problemas reside na dificuldade de percepção das relações espaciais entre os componentes de uma cena tridimensional. Outro fator restritivo está na interação em um ambiente com mais graus de liberdade do que os oferecidos pelo dispositivo utilizado na sua manipulação.

[1] O termo manipulação direta refere-se ao tipo de interface no qual objetos, relações ou atributos que podem sofrer modificações, são alterados através de ações sobre suas representações gráficas, geralmente através de um *mouse*[13].

Desenvolver aplicações 3D com interfaces gráficas interativas torna-se então uma tarefa complexa.

Esta complexidade advém tanto da necessidade de implementar técnicas que permitam visualizar dados tridimensionais em dispositivos de saída 2D, quanto de escrever código que estabeleça a correspondência entre ações nos dispositivos de entrada 2D e ações no ambiente 3D.

Alguns pacotes gráficos, tais como GKS[16], PHIGS+[18] e OpenGL[27], oferecem recursos de visualização 3D, mas pouco ou nenhum suporte à interação, deixando a cargo do projetista a tarefa de desenvolver mecanismos de manipulação tridimensional[37]. Por outro lado, alguns experimentos mais recentes têm sido feitos com o objetivo de criar soluções para a construção de aplicações com recursos de interação 3D.

Stevens et al.[36] apresentaram uma ferramenta visual que permite a construção e prototipação de objetos e *widgets*[2] tridimensionais a partir de primitivas básicas, como ponto, vetor e plano. A construção dos próprios *widgets* é feita pela manipulação direta destas primitivas. Nesta abordagem, os objetos do modelo geométrico da aplicação e as ferramentas utilizadas para manipulá-los estão integrados em único ambiente.

Outra abordagem consiste no desenvolvimento de bibliotecas de objetos que são utilizados na criação de interfaces 3D, como a biblioteca *Open Inventor*[43]. Em sua arquitetura, cada objeto é "capaz de definir a sua representação gráfica e realizar as operações concernentes a ele". Com isso, exceto as informações dependentes dos sistemas de janelas, todos os objetos graficamente manipuláveis de um aplicativo estão encapsulados na biblioteca. O *Open Inventor* provê também mecanismos para inclusão de novos objetos, desde que sejam criados os respectivos algoritmos de modelagem, visualização e manipulação.

Tanto na abordagem de Stevens quanto a na abordagem do *Open Inventor*, a integração dos recursos de modelagem, visualização e interação facilita o gerenciamento das tarefas de manipulação direta 3D por parte da biblioteca. Além disso, é preciso aprender apenas um paradigma para desenvolver objetos da aplicação e elementos da interface. Por outro lado, essa integração pode comprometer a simplicidade e a modularidade dos sistemas desenvolvidos com estas arquiteturas[31].

Outro fator a ser considerado é a flexibilidade dos mecanismos de interação oferecidos. É desejável que a arquitetura ofereça diferentes mecanismos de interação, de forma que a aplicação possa escolher o mais adequado aos seus problemas particulares.

[2] Objetos que encapsulam geometria e comportamento.

Deve-se considerar ainda que estas propostas se mostram boas soluções na construção de interfaces tridimensionais somente quando os objetos da aplicação podem ser mapeados diretamente para os objetos da biblioteca ou ferramenta. Caso contrário, torna-se necessário adicionar funcionalidades específicas para gerenciar os objetos, criar suas representações gráficas e interagir com eles. Mais uma vez, a utilização de uma ferramenta para a construção de uma interface 3D é uma tarefa complexa, já que é preciso dominar os recursos de Computação Gráfica para implementar as funções necessárias e conhecer a arquitetura interna da ferramenta para estender suas funcionalidades.

Dentro deste enfoque, apresentamos a proposta de uma arquitetura para desenvolvimento de interfaces gráficas 3D com manipulações diretas, através de dispositivos de entrada 2D, destinada a programadores envolvidos na criação de aplicações com interfaces 3D.

Os pontos-chave desta arquitetura são:

- *reusabilidade* das técnicas de interação construídas;
- *flexibilidade* para construção de diferentes técnicas de interação; e
- *extensibilidade* para inserir novos objetos na arquitetura quando necessário.

A reusabilidade está relacionada com a capacidade de reutilizar, em diferentes aplicações, os *widgets* 3D desenvolvidos.

Para assegurar a reusabilidade, propomos o desacoplamento entre o par visualização/manipulação e os objetos específicos da aplicação. Como consequência deste desacoplamento, não é necessário inserir objetos da aplicação dentro da biblioteca, o que requer a criação de código de visualização e manipulação para cada novo tipo de objeto. A implementação destes códigos nem sempre é trivial, além de aumentar o tempo de implementação.

Tal independência é possível graças à definição precisa da fronteira entre a aplicação e a visualização/manipulação e à simplicidade da estrutura de dados gerenciada pela biblioteca, composta de primitivas consideradas básicas. Estas primitivas podem ser agrupadas para formar objetos mais complexos, em qualquer nível de granularidade.

A flexibilidade da arquitetura está baseada na capacidade de construir novas técnicas de interação. Em nossa proposta, a manipulação direta é feita através de elementos chamados *interadores*. Um interador é um símbolo gráfico com a capacidade de responder à interação do usuário. Um interador não possui um comportamento pré-definido, pois a sua semântica é definida pela aplicação. Esta semântica, por sua vez, é especificada através de outro tipo de elemento, as *restrições*, que definem os comportamentos possíveis para os interadores. Estes dois recursos podem ser combinados para criar um grande número de mecanismos de manipulação

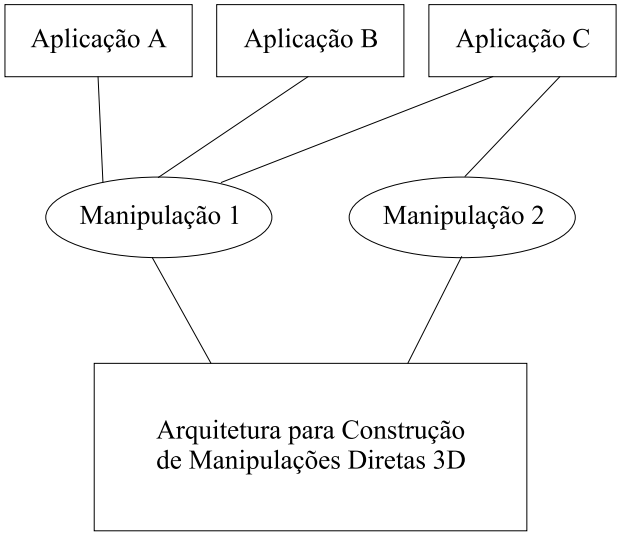


Figura 1.1: *Arquitetura para construção de manipulações diretas 3D.*

direta. Traçando um paralelo com *widgets*, os interadores representam sua geometria, enquanto as restrições representam seu comportamento.

A extensibilidade refere-se à capacidade de incluir novos interadores e restrições. Em virtude da arquitetura orientada a objetos, e do projeto das classes utilizadas, a inclusão de novos objetos pode ser feita através da derivação das classes já existentes.

No exemplo da Figura 1.1, a arquitetura é utilizada para construir a "Manipulação 1", responsável por fazer translações em 3D, e a "Manipulação 2", capaz de fazer rotações em 3D, ambas através da manipulação direta de objetos. Como exemplo de reusabilidade, note que a mesma "Manipulação 1" é usada em três aplicações diferentes. Quanto à flexibilidade da arquitetura, note que ela foi utilizada para construir duas manipulações diferentes, a "Manipulação 1" e a "Manipulação 2".

A arquitetura será implementada sob a forma de bibliotecas de funções. Esta biblioteca será utilizada para criar manipulações voltadas para transformações lineares, mais especificamente, as operações de translação, rotação e mudança de escala. A biblioteca e as manipulações serão então aplicadas para criar uma interface gráfica com manipulações diretas 3D, de um modelador geométrico de sólidos por instanciação.

A documentação desta biblioteca, bem como seu código fonte, podem ser encontrados no endereço da Internet *http://www.dca.fee.unicamp.br/~navarro/mtk*.

A dissertação está organizada em seis capítulos.

O Capítulo 2 descreve o estado da arte das arquiteturas para construção de interfaces gráficas em 3D , além de técnicas de interação em 3D através de dispositivos 2D, utilizadas para desenvolver algumas manipulações a partir da arquitetura proposta.

O Capítulo 3 apresenta a nossa solução de arquitetura para manipulação direta 3D. Serão descritos os pontos-chave da arquitetura: a sua reusabilidade em aplicações com modelos geométricos distintos; a sua flexibilidade para construir novas manipulações, a partir dos elementos básicos oferecidos pela arquitetura; e a capacidade de incluir novos elementos básicos de interação. Em seguida, os componentes da arquitetura serão descritos e detalhados.

O Capítulo 4 mostra a implementação da arquitetura sob a forma de uma biblioteca de funções, chamada MTK. A partir do enfoque de orientação a objetos, serão apresentadas as classes implementadas para a criação do MTK.

O Capítulo 5 aborda as manipulações construídas a partir da biblioteca MTK. Serão descritas as operações envolvidas nas manipulações, além do método empregado para construí-las. Uma discussão dos resultados da aplicação destas manipulações será apresentada. Em seguida, será apresentada a aplicação destas manipulações em um modelador geométrico de sólidos por instanciação.

Finalmente, o Capítulo 6 apresenta as conclusões sobre o trabalho desenvolvido e sugestões para desenvolvimentos futuros.

Capítulo 2

Revisão Bibliográfica

Este capítulo tem como objetivo principal apresentar uma visão geral do atual estágio de desenvolvimento de ferramentas para a construção de interfaces 3D (Seção 2.1). Na Seção 2.2 são discutidas algumas técnicas de interação 3D utilizando dispositivos 2D, e que serviram de inspiração para nossa proposta.

2.1 Bibliotecas de Interação 3D

Nesta Seção, descreveremos algumas bibliotecas e um ambiente visual de desenvolvimento. Suas arquiteturas provêem recursos de interação e visualização em 3D, e que auxiliam o desenvolvimento de aplicativos gráficos interativos 3D.

2.1.1 Open Inventor

O *Open Inventor*[43] é uma biblioteca 3D orientada a objetos, que provê uma estrutura para descrever, visualizar e manipular cenas tridimensionais. As cenas são organizadas sob a forma de um grafo, constituído de estruturas chamadas nós. Estes nós armazenam informações como a geometria dos objetos, seus materiais, fontes de luz e câmeras, entre outros.

O mecanismo de interação do *Open Inventor* está baseado em tipos especiais de nós, os *draggers* e *manipulators*. Um *dragger* é um objeto com geometria própria, que pode ser manipulado diretamente pelo usuário e que é capaz de realizar operações específicas em 3D, como rotação e mudança de escala. Ao interagir com um *dragger*, são retornados valores correspondentes às operações efetuadas. Os *draggers* podem ser combinados em objetos mais complexos, mas seu comportamento é pré-definido, não podendo ser alterado.

Os *draggers* não aplicam os valores gerados por eles diretamente sobre os objetos manipulados. Um *manipulator* associa um ou mais objetos a um dragger, e substitui os nós dos objetos manipulados durante a interação. No fim da interação, o grafo é restaurado com o nó original, que tem seus atributos atualizados de acordo com os atributos correntes do *manipulator*.

A integração dos recursos de modelagem, visualização e interação facilita o gerenciamento das tarefas de manipulação direta 3D por parte da biblioteca. Ao mesmo tempo, essa complexidade pode comprometer a simplicidade e a modularidade do sistemas desenvolvidos com essa biblioteca[31].

Outro problema desta abordagem é que ela se mostra uma boa solução na construção de interfaces tridimensionais somente quando os objetos da aplicação podem ser mapeados diretamente para os objetos da biblioteca. Caso contrário, torna-se necessário adicionar à biblioteca funcionalidades específicas para gerenciar os objetos, criar suas representações gráficas e interagir com os objetos. Mais uma vez, a utilização de uma ferramenta para a construção de uma interface 3D é uma tarefa complexa, já que é preciso dominar os recursos de Computação Gráfica para implementar as funções necessárias e conhecer a arquitetura interna do *Open Inventor* para estender suas funcionalidades.

2.1.2 Act

A biblioteca *Act*[5] é similar ao *Open Inventor*, pois também é uma biblioteca orientada a objetos, e que oferece recursos para a criação, visualização e manipulação de objetos 3D, integrados em um único ambiente. Os objetos podem ser organizados hierarquicamente, como na estrutura de grafos do *Open Inventor*.

A organização começa a partir do objeto *Item*, que representa a classe base de qualquer outro objeto gráfico, representando qualquer objeto que faça parte de uma cena. As classes *Setting*, *Transformation*, *Object*, *Attribute* e *Callback* são derivadas diretamente da classe *Item*, e definem as seguintes funcionalidades:

Setting: configurações globais para geração de imagens, como incluir ou não *fog*;

Transformation: transformações geométricas (translação, rotação e mudança de escala);

Object: objetos com geometria própria (primitivas como esfera, cubo, toro); também atendem ao propósito de armazenar outros objetos, para formar hierarquias, por exemplo;

Attribute: características que podem influir em como os objetos são visualizados, como cor, textura e material ; e

Callback: permite atribuir ações a pontos específicos do grafo de cena, de tal forma que elas sejam executadas quando o grafo for percorrido.

Para interagir com os objetos de uma cena, *Act* provê mecanismos que transformam posições bidimensionais do mouse em um cursor tridimensional. O cursor 3D é definido pela sua posição e orientação, e atua em dois modos. No primeiro, a posição do cursor 3D é mapeada para o ponto na superficie do objeto mais proximo na cena. Uma aplicação deste modo é em operações de translação. Um ou mais objetos são selecionados e levados com o cursor 3D. Em um segundo modo, o cursor 3D pode ter seus movimentos restritos a uma linha, um plano, uma circunferência ou à superficie de uma esfera. Por exemplo, o cursor 3D pode ter seus movimentos restritos sobre uma circunferência. Os objetos selecionados acompanham então o cursor 3D, em um movimento de rotação ao longo da circunferência definida.

A biblioteca *Act*, escrita em C++, pode ser estendida através da derivação das classes existentes. Chamadas diretas a funções do OpenGL também podem ser incorporadas. Entretanto, o recurso mais interessante para ampliar suas funcionalidades é através de uma linguagem interpretada chamada *Lua*.

Lua apresenta recursos para a criação de estruturas de dados baseados em arranjos associativos[1]. Estão disponíveis também todas as estruturas de controle de linguagens convencionais, como expressões, laços, comandos condicionais e chamadas de função, além de recursos de programação orientada a objetos. Devido a sua sintaxe simples, a linguagem permite a criação de aplicações com recursos tridimensionais, mesmo por pessoas com pouca experiência em programação.

Certamente, a simplicidade da linguagem Lua permite inserir estruturas de dados específicas e seus respectivos algoritmos gráficos com maior facilidade. No entanto, a utilização de uma linguagem interpretada leva a problemas de desempenho. Os próprios autores aconselham utilizar a linguagem Lua para prototipação e, após gerar um código estável, convertê-lo para C++ de forma a melhorar o desempenho. Devido à maior complexidade de C++, a tarefa de conversão pode não ser trivial.

2.1.3 IQL

O IQL, *Inquire Quick Line*[24] é o módulo de recursos gráficos tridimensionais de uma biblioteca de componentes de interface denominada PRODIA[9][2]. O IQL foi desenvolvido com base em funções da biblioteca *Xlib*, e possui, entre outros recursos:

[1] Em inglês, associative arrays

[2] PRODIA foi projetada na década 80 pelo grupo de Computação Gráfica no IGD da Sociedade Fraunhofer Gesellschaft (FhG), na Alemanha

- um modelo de entidades gráficas 3D;
- um modelo de visualização 3D;
- uma técnica de identificação; e
- guias visuais: grades e eixos 3D.

Tanto o PRODIA quanto os módulos nele incluídos foram escritos em linguagem *C*. Com o intuito de reaproveitar ao máximo os códigos de programação, foi efetuado o desacoplamento dos recursos do módulo IQL do PRODIA pelo grupo de Computação de Imagens da Faculdade de Engenharia Elétrica e de Computação da Unicamp, e as funções do IQL foram usadas como suporte para o desenvolvimento de técnicas de visualização e interação 3D[40].

A estratégia de modelagem de objetos no IQL considera que um modelo para representação gráfica de uma cena 3D deve permitir a construção de objetos complexos na cena a partir de entidades gráficas simples[3]. As entidades gráficas básicas são segmentos, arestas e vértices. Essas entidades possuem um relacionamento hierárquico. O segmento é formado de arestas que por sua vez são definidas a partir de vértices. Ao mesmo tempo, um segmento pode fazer parte de um outro segmento, construindo-se assim hierarquias de entidades simples até entidades mais complexas.

O mecanismo de seleção do IQL é baseado em uma técnica que permite a identificação de entidades gráficas 3D numa tela de visualização. As entidades correspondentes a todas as entidades da cena são previamente armazenadas tanto em coordenadas do mundo como em coordenadas do dispostivo. O algoritmo de seleção ordena as entidades 3D a serem exibidas na tela de acordo com a sua distância em relação à posição do observador/câmera, utilizando o procedimento de *ray-casting*.

Três técnicas[39] estão disponíveis para a implementação de manipulações diretas: a técnica de particionamento do espaço do dispositivo, baseada no trabalho de Nielson e Olsen (Seção 2.2.4), a técnica de movimentos circulares, inspirada na proposta por Evans[12] e a técnica do Plano de Trabalho, equivalente ao movimento com restrição sobre um plano apresentado por Bier (Seção 2.2.5). Esta última técnica recupera a informação de profundidade a partir de um plano do espaço, com restrição de alinhamento contínuo ou alinhamento discreto. A realimentação visual deste plano no espaço é feita através de um plano em forma de grade.

Diferentemente do *Open Inventor* e *Act*, IQL é uma biblioteca de funções independente das aplicações, e o mapeamento entre os dados geométricos de uma aplicação específica e os dados geométricos do IQL são bastante simples, pois:

[3]Entidades gráficas simples são elementos básicos para a composição de objetos gráficos mais complexos a partir delas. No padrão gráfico PHIGS+ as entidades simples são, por exemplo, vértices (*polymark*), arestas (*polyline*), polígono (*fill area*) e polígonos (*fill area set*).

- possui um conjunto de primitivas simples; e
- provê um mecanismo para construir objetos mais complexos a partir das primitivas simples, de forma hierárquica.

Entretanto, um obstáculo à utilização da biblioteca IQL é o seu suporte à visualização e interação 3D ainda com poucos recursos. Além disso, o fato de ter sido construída sobre o Xlib não permite tirar proveito de *hardware* acelerado com recursos 3D, bastante comum atualmente.

2.1.4 Manipulação de Dados Volumétricos

O trabalho de Fonseca et al.[14], apresenta uma arquitetura para construção de *widgets* 3D, chamados manipuladores, utilizando a biblioteca gráfica OpenGL. Tais *widgets* são indicados para aplicações de visualização de dados volumétricos. A idéia principal é usar objetos com comportamento simples para compor a geometria e comportamentos mais complexos dos manipuladores.

A arquitetura é constituída pelos seguintes componentes:

Objetos gráficos: pontos esparsos, linhas poligonais, grades 2D e grades 3D.

Visualizações: formas de *rendering* para os objetos gráficos.

Visualizador: *widget* convencional de um sistema de janelas. Contém uma área de desenho e uma interface para funções específicas.

Manipuladores: *widget* 3D associado a uma visualização, e que pode ser editado, resultando em alterações na visualização.

Componentes de arrasto: usados para compor a geometria e o comportamento dos manipuladores.

Projetores: fazem o mapeamento de movimentos do *mouse* em movimentos 3D.

Geometria: provêem diferentes realimentações visuais para os componentes de arrasto.

O enfoque de orientação a objetos permite que novos manipuladores, componentes de arrasto, projetores e objetos gráficos sejam implementados.

A comunicação entre uma visualização e um manipulador é feita através de funções *callback* definidas no manipulador: uma *callback* de início quando o botão do *mouse* é pressionado, uma *callback* de movimento para arrasto do *mouse* e uma *callback* de fim, quando o botão é liberado.

Os manipuladores são compostos por componentes de arrasto, os quais possuem geometria própria e reagem diretamente aos eventos do usuário, realizando edições simples, como escala, rotação e translação. Os componentes de arrasto tratam os eventos de interação através de *callbacks* de início, movimento e fim, como os manipuladores, e possuem um quarto tipo de *callback*, que é a de valor alterado. Esta função é associada ao componente para que ele se comunique com seu manipulador, passando as informações sobre seu movimento.

Os componentes de arrasto podem executar os seguinte tipos de movimento: translação sobre uma linha, sobre um plano e em um volume, e rotação sobre a superfície de uma esfera e de um cilindro. Os projetores fazem o mapeamento dos movimentos do *mouse* para uma das restrições de movimento mencionadas.

Ao receber um evento de botão do *mouse* pressionado, é utilizado o mecanismo de *picking* do OpenGL para verificar se ele ocorreu sobre um componente de arrasto. Em caso afirmativo, o identificador deste componente e o evento são passados para o manipulador, que efetua as operações necessárias para iniciar a interação, como estabelecer comunicação com a visualização.

Nos eventos de botão arrastado, o componente de arrasto utiliza seu projetor para interpretar o movimento de *mouse* e informa o manipulador sobre seu movimento. O manipulador atualiza seus componentes e, se necessário, interage com a visualização com a qual está associado.

Ao receber um evento de botão liberado, os componentes de arrasto executam as operações necessárias para o fim da interação, como restaurar seu estado inativo, e o manipulador encerra a comunicação com a visualização.

Baseado na arquitetura do *Open Inventor*, esta proposta também encapsula o modelo geométrico da aplicação juntamente com os recursos de visualização e manipulação em 3D. Conforme a discussão da Seção 2.1.1, esta abordagem leva ao aumento da complexidade de programação, devido à necessidade de implementar as funcionalidades necessárias para visualizar e manipular novos objetos na biblioteca.

2.1.5 Construção Interativa de Widgets

As propostas de ferramentas para construção de aplicações 3D apresentadas até agora, são baseadas no uso de bibliotecas de funções, geralmente acessadas através de uma linguagem de programação. Stevens et al.[36] sugeriram como abordagem alternativa, uma ferramenta visual que permite a construção e prototipação de objetos e *widgets* tridimensionais a partir de primitivas básicas, como ponto, vetor e plano. A construção dos próprios *widgets* é feita pela manipulação direta destas primitivas. Os objetos do modelo geométrico (sólidos como cubos e

esferas) da aplicação e as ferramentas utilizadas para manipulá-los estão integrados em único ambiente.

O comportamento dos *widgets*, por sua vez, é definido através de *restrições*. Mais especificamente, as restrições definem o tipo de movimento permitido. Por exemplo, uma primitiva pode ter seu movimento restrito ao longo de uma linha ou sobre um plano.

Restrições podem ser associadas a primitivas através do mecanismo de *slots*. *Slots* são variáveis relacionadas à primitiva, e que podem ter seus valores alterados através de restrições. Há quatro tipos de primitivas básicas, cada uma com *slots* específicos:

Ponto: é representado graficamente por uma pequena esfera. Possui um *slot*, que corresponde a uma posição no espaço tridimensional. Sua técnica de interação padrão é um movimento de translação livre no espaço.

Vetor: é representado por uma seta, e possui três *slots*, para posição, direção e comprimento do vetor. A técnica de interação padrão associada ao vetor é um movimento de rotação.

Plano: representado por um quadrado preenchido, possui cinco *slots*, um vetor normal ao plano, o centro do plano, dois *slots* para altura e largura do plano e um quinto para representar a orientação do plano.

Volume: é representado por sólidos como cubos e esferas. Possui os mesmos *slots* que a primitiva do tipo plano, além de atributos não geométricos, como atributo de cor, necessários para modelar objetos a partir destas primitivas.

Deve-se notar que, apesar das primitivas apresentarem um comportamento padrão, o mesmo pode ser alterado. Segundo os autores, "atribuir uma técnica de interação fixa para cada primitiva não permite que seja possível selecionar a técnica mais adequada para cada problema particular".

Quando primitivas e restrições são usados para criar novos objetos, elas podem ser encapsuladas para facilitar sua reutilização. Existem dois tipos de encapsulamento: estrutural e por classe. No modo estrutural, ao reutilizar um novo objeto, é criada uma nova cópia, com as mesmas primitivas e restrições. Essa cópia continua a referenciar as primitivas a partir das quais foi criada.

No modo de encapsulamento por classe, também é criada uma nova cópia do novo objeto, também com as mesmas primitivas e restrições. Entretanto, estas restrições podem ser alteradas para criar outros comportamentos para o novo objeto, característica não disponível no encapsulamento estrutural.

2.1.6 Interact

Nas seções anteriores foram discutidas várias ferramentas para a construção de interfaces 3D. Carneiro et al.[31] apresentam em seu trabalho uma proposta de ferramenta para a construção de interfaces 2D, mas suas idéias podem ser aplicadas em três dimensões. Seu modelo de interação visa atender principalmente aplicações baseadas na criação de objetos gráficos 2D, a partir de um conjunto de pontos, utilizados para formar linhas (*polyline*).

Nesta abordagem, o Interact não define as estruturas de dados da aplicação, como no Open Inventor (Seção 2.1.1), pois propõe que toda a semântica deve estar embutida na própria aplicação. A idéia central é que um programa de aplicação possa especificar uma tarefa a ser executada e só se preocupar com o resultado final. Segundo os autores, a comunicação deve ser realizada em um nível mais abstrato, através da definição e manipulação de vértices de controle e *bounding boxes*.

Utilizando os conceitos de programação orientada a objetos, foram definidas duas classes principais:

Canvas: representa a área de desenho, onde a aplicação exibe seus objetos, e é responsável pela captura e tratamento de eventos, como movimento do cursor e pressionamento de teclas;

Task: representa tarefas de interação, como a captura de uma linha, posicionamento, seleção, transformação de objetos, etc.

A partir da classe Canvas, existem três tipos básicos de tarefas que podem ser realizadas utilizando a classe Task:

Construção: criação de objetos definidos a partir de vértices de controle (linhas, quadrados, círculos, etc.);

Seleção e Transformação: escolha e alteração dos atributos dos objetos criados (cor, tamanho, posição, forma, etc.);

Visualização: controle da área visível de desenho (zoom, pan, etc.).

Por exemplo, o desenho de dois pontos (que podem ser usados para definir uma linha, um retângulo, etc.), pode ser feito em três passos:

- pressionando o botão esquerdo do mouse, o primeiro ponto é definido na posição atual do cursor;

- o usuário arrasta o cursor até o segundo ponto;
- soltando o botão do *mouse*, a construção é terminada, e o segundo ponto é definido na posição do cursor.

Deve-se notar que o Interact desconhece quaisquer estruturas de dados da aplicação. Portanto, as tarefas de construção não desenham a forma definitiva do objeto. Por exemplo, durante a construção de um retângulo, é feito seu desenho temporário, que é removido após a aplicação receber as coordenadas finais de seus pontos. A partir daí, torna-se responsabilidade da aplicação realizar o desenho final.

Para aplicar transformações sobre objetos já construídos, é preciso primeiro selecionar estes objetos. A seleção pode ser realizada através de duas técnicas, *pick* e *fence*. *Pick* consiste em posicionar o cursor sobre o objeto e clicar o *mouse* sobre o objeto. A técnica de *fence* permite definir uma região, normalmente retangular, no *canvas* da aplicação, e todos os objetos que estiverem nesta região são selecionados.

Após selecionados, os objetos podem sofrer transformações lineares afins (translação, escala, rotação e cisalhamento), aplicadas através de *bounding boxes*, e transformações locais, através da manipulação de pontos de controle em *polylines*.

As translações são realizadas através do arrasto dos objetos selecionados. As outras transformações afins são feitas através do arrasto de alças, pequenos quadrados colocados ao redor dos objetos selecionados. A mudança de escala é obtida arrastando um destes quadrados na mesma direção ou em direção contrária à alça oposta. Ao clicar em um conjunto de objetos previamente selecionados, são criados dois tipos diferentes de alças, pequenos círculos e setas. Os círculos são utilizados nas transformações de rotação, e as setas nas transformações de cisalhamento.

A manipulação de pontos de controle permite alterar a forma de um objeto. Após selecionar um objeto, todos os seus nós aparecem. Um ou mais nós devem ser selecionados, através das técnicas de *pick* e *fence*, quando então torna-se possível mover nós, inserir novos nós ou remover nós já existentes.

2.2 Tecnologia em Software

Uma abordagem comum em aplicações com recursos de interação 3D, é implementar estas interações de forma que sejam executadas através de dispositivos 2D. Esta é a solução mais utilizada, devido ao uso bastante difundido dos dispositivos 2D e ao seu baixo custo em relação a dispositivos 3D. Por outro lado, por apresentar um número menor de graus de liberdade,

são necessários alguns artifícios para apresentar e captar coerentemente dados 3D através de dispositivos 2D. Na Seção 2.2.1, são descritas as técnicas mais comuns para aumentar a percepção da relação espacial entre os objetos de uma cena tridimensional, enquanto na Seção 2.2.2 são apresentadas várias propostas de manipulação encontradas na literatura.

2.2.1 Aumento da Percepção Espacial

A noção de profundidade pode ser explicada pela fusão das duas imagens bidimensionais captadas pelos olhos. Mesmo que utilizemos apenas um de nossos olhos, ainda assim podemos manter um pouco de nossa percepção espacial. Isto é possivel graças a um conjunto especial de condições que indicam a profundidade de diferentes partes de uma cena, conforme Foley et al.[13]:

- a **projeção perspectiva** tem um efeito similar ao de sistemas fotográficos e da visão humana. O tamanho da projeção perspectiva de um objeto varia inversamente com a distância do objeto do centro de projeção;
- a **eliminação de superfícies escondidas** indica quais faces de um objeto estão à frente;
- **cores** proporcionam um realismo maior aos objetos. Partes de um objeto que estão mais longe são exibidas com uma intensidade de cores menor;
- a **interposição** de objetos fornece uma noção de quais objetos estão mais à frente e quais estão mais atrás na cena;
- a **transparência** dos objetos também provê pistas sobre a oclusão de outros objetos;
- **fontes de luz** fazem com que objetos mais distantes pareçam menos iluminados, e objetos mais próximos pareçam mais iluminados;
- **sombras** ajudam na indicação da direção de uma fonte de luz; sombras de um objeto projetadas sobre outros objetos também oferecem uma boa indicação da relação espacial entre eles;
- **fog**[4] que faz com que os objetos fiquem menos nítidos à medida que aumenta a profundidade;
- a projeção bidimensional de uma imagem 3D parece fortemente tridimensional quando a cena está sendo **rotacionada**;
- objetos que estão sendo **focados** são exibidos de forma mais nítida do que objetos que estão fora do foco da câmera; e

[4]Uma espécie de névoa

- **paralaxe de movimento**, o fenômeno visual decorrente da diferença aparente de velocidade de deslocamento de dois objetos ou partes de um mesmo objeto, movendo-se à mesma velocidade, mas em profundidades diferentes.

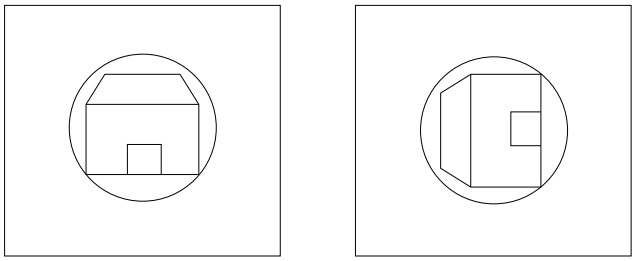
2.2.2 Manipulação 3D Através de Dispositivos 2D

Segundo Kettner[22], o uso dos dispositivos 2D, como *mouse* e mesa digitalizadora, para interações diretas 3D possui várias vantagens em relação aos mecanismos tridimensionais:

- Dispositivos bidimensionais são largamente usados em interfaces tradicionais. E por utilizarem menos sensores que mecanismos 3D, são mais viáveis economicamente.
- O antebraço pode respousar sobre o mesa onde se encontra o dispositivo, enquanto alguns dispositivos 3D podem fatigar o usuário.
- Normalmente, a interação em 3D está embutida em um contexto bidimensional de janelas, caixas de diálogo, menus e entrada de dados via teclado. Frequentes mudanças entre um dispositivo e outro pode diminuir a validade de dispositivos 3D. A *data glove*, por exemplo, não é apropriada para grande quantidade de dados digitados.
- Dependendo da aplicação, a interação em 3D pode tirar proveito da restrição de possuir dois graus de liberdade. A dificuldade dos usuários de utilizarem um dispositivo com muitos graus de liberdade pode induzir a erros frequentes e, consequentemente, em um tempo extra para a correção desses erros.
- O espaço físico é um fator limitante para aplicações de um dispositivo, como em *notebooks*. Os *trackballs* utilizados em *notebooks* são pequenos e adequados para o pouco espaço disponível. Além disso, seria pouco confortável carregar, junto com um *notebook*, um outro dispositivo como um *spaceball* ou uma *data glove*.

Devido a essas vantagens, várias estratégias de interação têm sido desenvolvidas com o intuito de implementar as tarefas de interação 3D a partir das técnicas de interação 2D. Algumas são tarefas de posicionamento e seleção e outras são tarefas para construção e transformações afins.

Serão apresentadas a seguir, técnicas de interação 3D utilizadas principalmente em transformações lineares, mais especificamente translação, rotação e mudança de escala, com enfoque na geometria dos objetos utilizados para a manipulação, seu comportamento e suas técnicas de interação. No trabalho de Velasquez[39], pode ser encontrada uma explicação mais detalhada de algumas das técnicas de interação apresentas a seguir.



a) Esfera Virtual a) Objeto Rotacionado

Figura 2.1: *Controlador virtual de rotação.*

2.2.3 Proposta de Chen

Em seu estudo, Chen[6] avaliou controladores virtuais para *rotacionar* objetos em 3D, mais especificamente em torno dos eixos x, y e z do sistema de coordenadas da câmera.

Segundo o autor, o melhor destes controladores é a esfera virtual. O símbolo gráfico deste controlador é um círculo (Figura 2.1). O objeto pode ser imaginado como se estivesse dentro de uma esfera de vidro. O comportamento do controlador é tal que, para obter uma rotação, basta rolar a esfera e, consequentemente, o objeto que está dentro dela. Movimentos para cima-para baixo e esquerda-direita no centro do círculo são equivalentes a rotacionar o objeto em relação aos eixos x e y, respectivamente. Um movimento ao longo da borda do círculo ou fora dela produz rotação ao redor do eixo z. Uma varredura completa do *mouse* em linha reta produz 180 graus de rotação no plano xy, e um círculo completo produz 360 graus ao redor do eixo z.

Na Figura 2.2, o uso do *mouse* nesta técnica de interação é representado através de um *diagrama de transição estados*. No estado inicial, o botão do mouse não está pressionado. Ao pressionar o botão, ocorre a transição para o estado de rotação, que é mantido enquanto o botão continuar pressionado. Ao liberar o botão do *mouse*, a execução da rotação é encerrada e a interação volta a seu estado inicial.

2.2.4 Proposta de Nielson e Olsen

Nielson e Olsen[28] contribuiram com uma técnica bastante interessante para *posicionamento em 3D* com o uso de um *mouse* convencional. O posicionamento é feito através de um *cursor* denominado *tríade*, representado graficamente por um cubo ou três eixos alinhados com o sistema de coordenadas do mundo ou com o sistema de coordenadas local de um objeto. A tríade pode ser manipulada dentro de um cubo maior, envolvendo a cena, e que serve como estrutura de referência em 3D.

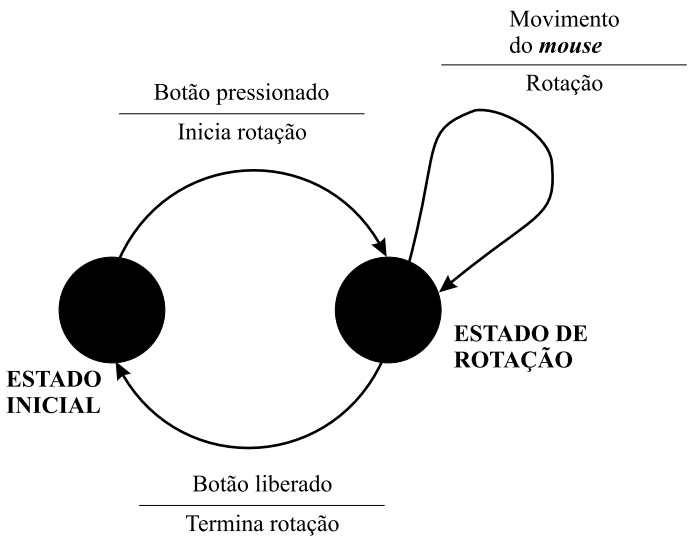


Figura 2.2: *Diagrama de transição de estados para rotação.*

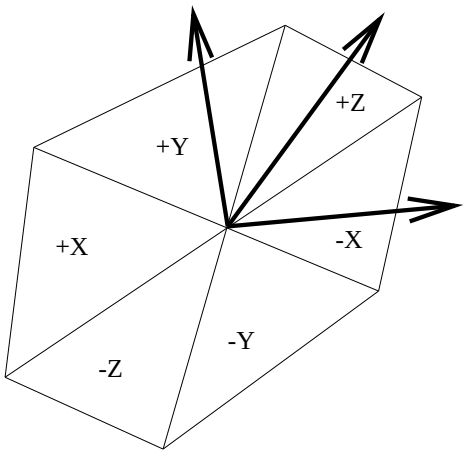


Figura 2.3: *Particionamento do espaço em 6 regiões.*

A tríade é manipulada em três dimensões através da amostragem sucessiva das posições captadas pelo *mouse* 2D. O vetor de deslocamento do *mouse*, (Dx_i, Dy_i), na amostra i, sendo $Dx = x_{i-1} - x_i$ e $Dy = y_{i-1} - y_i$, é comparado com cada uma das projeções dos eixos da tríade. O comportamento da tríade é um deslocamento 3D ao longo do eixo da tríade que estiver mais próximo do vetor de movimento $\mathbf{D}$. O movimento fica restrito apenas ao eixo cujo cosseno tem o maior valor absoluto. Isto tem o efeito de particionar o espaço 2D em seis zonas (Figura 2.3). Os limites dessas zonas estão nos bissetores dos ângulos entre os vetores da tríade.

A tríade pode ser aplicada em operações de translação. Ao selecionar um objeto, o sistema de coordenadas local deste objeto define os 3 eixos da tríade. A tríade pode então ser movida ao longo de um de seus três eixos, levando consigo o objeto. Na Figura 2.4, é apresentado

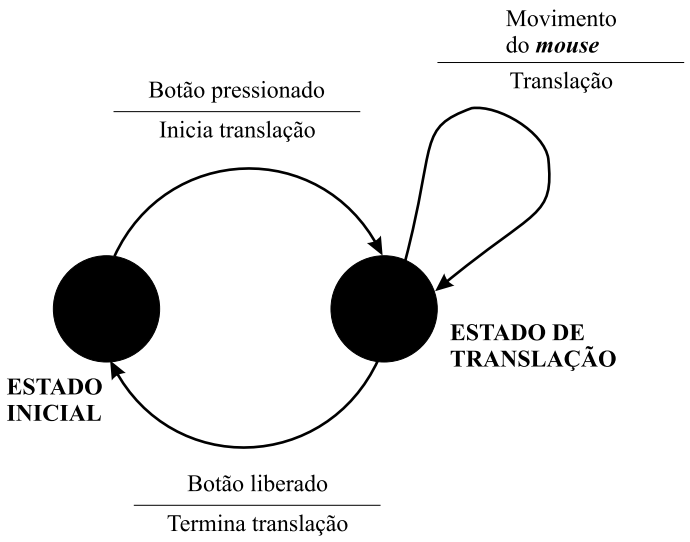


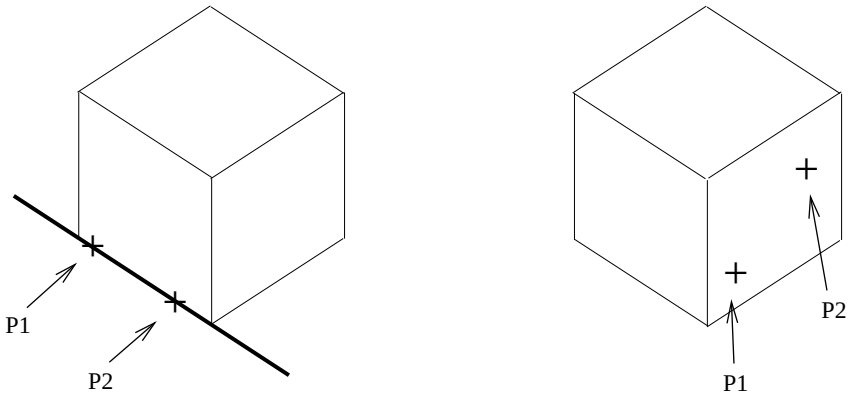
Figura 2.4: *Diagrama de transição de estados para translação.*

um diagrama de estados para a aplicação da tríade em uma operação de translação.

Adicionalmente, foram propostas algumas técnicas de interação para *translação*, *rotação* e *mudança de escala*, baseadas na geometria do objeto, mais especificamente, através das arestas e faces de objetos poliédricos.

A operação de translação pode ser feita através da aresta de um objeto ou através de uma face. No primeiro modo de translação (Figura 2.5.(a)), selecionamos um ponto p_1 em uma aresta de um objeto. Um ponto p_2 selecionado sobre a mesma aresta define a quantidade de translação como sendo a distância entre p_1 e p_2. O segundo modo de translação, sobre uma face (Figura 2.5.(b)), opera de modo similar. São selecionados dois pontos p_1 e p_2 sobre uma face de um objeto, e a distância entre p_1 e p_2 é a magnitude do vetor de deslocamento.

Para executar rotações são usadas duas técnicas. A primeira utiliza uma aresta de um objeto como eixo de rotação, e a segunda utiliza um eixo normal a uma face qualquer de um objeto como eixo de rotação. Na primeira técnica, é selecionada uma aresta que servirá como eixo de rotação A, escolhida através de um ponto p_1. Além do eixo de rotação, determina-se o ângulo de rotação através de dois outros pontos, p_2 e p_3. O ponto p_2 deve ser escolhido em uma face qualquer de um objeto. O ponto p_3 também será escolhido pelo usuário, mas será projetado em um plano que contenha p_1 e p_2, e que tenha A como vetor normal. O ângulo entre os dois vetores de p_1 a p_2 e de p_1 a p_3 passa a ser o ângulo de rotação. Na segunda técnica, um ponto p_1 seleciona uma face F de um objeto. O eixo de rotação A é o vetor normal à face F no ponto p_1. A face F define o plano de rotação. Outros dois pontos, p_2 e p_3, são escolhidos na face F. O ângulo entre os vetores de p_1 a p_2 e de p_1 a p_3 é o ângulo de rotação.



a) Translação sobre uma aresta b) Translação sobre uma face

Figura 2.5: *Translação utilizando a geometria dos objetos.*

A mudança de escala pode ser feita ao longo da aresta de um objeto ou uniforme à uma face. Na mudança de escala ao longo de uma aresta, são escolhidos 3 pontos p_1, p_2 e p_3 da aresta de um objeto. O fator de escala usado é $(p_3 - p_1)/(p_2 - p_1)$, e é aplicado sobre o objeto ao longo da aresta selecionada. Na mudança de escala uniforme à uma face, uma face F é selecionada com um ponto p_1. Um ponto p_2, escolhido também sobre a face F, define um eixo de p_1 a p_2. Um transformação leva p_1 à origem, o eixo de p_1 a p_2 a ao eixo x e o vetor normal à face selecionada, no ponto $p1$, ao eixo z. Deve ser escolhido um ponto p_3 sobre o novo eixo x. Finalmente, o fator de escala p_3/p_2 é aplicado tanto em x como em y, resultando em um aumento de escala uniforme sobre a face F.

2.2.5 Proposta de Bier

Bier[2] propôs uma tarefa de interação básica para posicionamento com o uso de *skitters* e *jacks* (Figura 2.6.a). *Skitters* tem um papel análogo a cursores 2D, e seu símbolo gráfico é representado por três semi-eixos perpendiculares. Em vez de gerarem posições (x, y), eles geram coordenadas (x, y, z). *Jacks*, por sua vez, possibilitam redefinir dinamicamente os referenciais dos movimentos, como eixo de rotação ou centro de escala. Sua geometria é especificada por três eixos perpendiculares, que se interceptam ao meio.

Prevêem-se três modos de posicionamento através de *skitters*:

Posicionar sobre as faces dos objetos 3D: A posição em 3D é obtida por um *skitter* através da técnica de *ray casting*, isto é, um raio é disparado da posição da câmera até o ponto da tela onde está o *skitter*. O ponto obtido corresponde ao ponto de interseção deste raio com o objeto cuja face estiver mais próxima da câmera. Não há,

porém, nenhuma restrição em escolher um ponto na face mais distante. Para aprimorar a percepção do deslocamento de um *skitter* sobre a face, a saída gráfica do *skitter* é de tal forma que a "origem" do *skitter* sempre coincide com o ponto selecionado e o eixo z do *skitter* coincide com o vetor normal neste ponto. O eixo x do *skitter* é calculado como o eixo ortogonal ao plano determinado pelo eixo z do *skitter* e o eixo y do sistema de coordenadas local, C, do objeto. Se estes dois eixos forem paralelos, o plano não será único, e o eixo x do *skitter* passa a ser definido como paralelo ao eixo x de C. O eixo y é o resultado do produto vetorial entre z e x.

Posicionar sobre o referencial local dos objetos: a "origem" do *skitter* é atraída para o sistema de coordenadas local mais próximo. Caso mais de um sistema de referência esteja à mesma distância do *skitter*, é escolhido aquele que estiver mais próximo do observador. Se a igualdade persistir, a escolha é aleatória. A orientação do *skitter* no espaço é a mesma do referencial local de cada objeto que o atraiu.

Posicionar em relação a um *jack*: o *skitter* se move sobre um dos três eixos ou sobre um dos três planos de um *jack*, previamente selecionado pelo mecanismo de "referencial local" mais próxima, descrito acima. Através de menus, é possível selecionar o tipo de restrição de movimento, eixo ou plano, e a orientação das restrições, x, y ou z. Após a escolha da restrição, é criado um *plano de seleção*, um pequeno quadrado cortado ao meio por um segmento de linha perpendicular ao plano. O plano de seleção, centralizado no *jack* e alinhado com um dos seus três eixos, serve como indicador visual da direção da restrição de movimento.

Utilizando *skitter* e *jacks*, foram propostas algumas técnicas de interação com *posicionamento relativo* de objetos. Este posicionamento requer que tanto o objeto manipulado quanto o objeto que servirá como referência possuam um *jack* associado a cada um deles. Note que foram criados 8 diferentes comportamentos utilizando os mesmos controladores virtuais, *skitters* e *jacks*:

- *MoveToTangeny*: move um objeto até que o *jack* em sua superfície seja coincidente com o *jack* na superfície de outro objeto. Na Figura 2.6, um *jack* é posicionado no corpo de um objeto, e outro é colocado na alça do objeto. A alça é movida até que seu *jack* coincida com o *jack* do corpo do objeto.
- *AbutX*, *AbutY*, *AbutZ*, *Abut*: os três primeiros comandos movem um objeto até que a origem de seu *jack* esteja nos planos x=0 (*AbutX*), y=0 (*AbutY*), e z=0 (*AbutZ*) de outro *jack*. O último comando, Abut, move um objeto até que a origem de seu *jack* coincida com a origem de outro *jack*. Em todos os quatro casos, a orientação do objeto que está sendo movido não muda, mas somente sua posição.

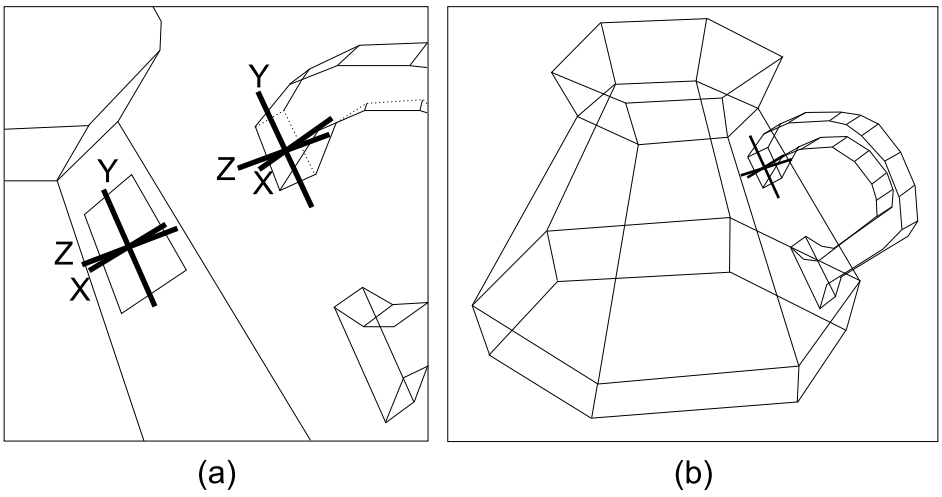


Figura 2.6: *(a) Posicionamento dos jacks. (b) Objeto depois da transformação.*

- *NormalizeRotation*: altera a orientação de um objeto de forma que todos os eixos de seu *jack* fiquem paralelos aos eixos de outro *jack*. Ao contrário do comando anterior, este muda apenas a orientação de um objeto, mas não sua posição.
- *Align*: altera a orientação de um objeto de forma que os eixos de seu *jack* fiquem paralelos aos eixos mais próximos de outro *jack*. É similar ao comando *NormalizeRotation*, mas permite que eixos diferentes alinhem-se entre si, como o eixo x de um *jack* alinhar-se ao eixo y de outro *jack*, por exemplo.
- *Normalize*: move um objeto até que coincida com outro objeto em posição e orientação.

Estas operações envolvem movimentos discretos, ou seja, os objetos "pulam" para sua nova posição ou orientação. Além destas transformações discretas, há dois modos de transformações contínuas, que acompanham o movimento do *skitter*:

- translação paralela ao plano de projeção; e
- translação paralela ao plano de seleção de um jack. O plano de seleção é especificado selecionando um *jack* através do *skitter* e escolhendo um dos três planos deste *jack*, (x, y), (x, z) e (y, z).

O diagrama de transição da Figura 2.7 ilustra este tipo de transformação. Após escolher se a translação será paralela ao plano de projeção ou ao plano de seleção, o *skitter* é utilizado para selecionar um objeto, utilizando um de seus três modos de posicionamento descritos anteriormente. Na figura, este estado corresponde ao "ESTADO DE SNAP". Após selecionar o objeto, é iniciada a translação, que na figura corresponde ao "ESTADO DE TRANSLAÇÃO".

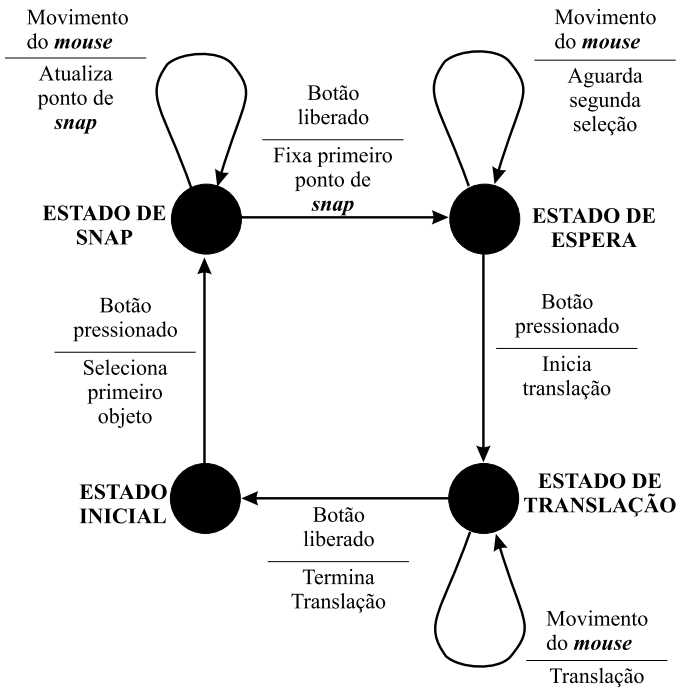


Figura 2.7: *Diagrama de transição estados para translações contínuas com o skitter.*

Em outro trabalho[3], Bier propôs ainda o uso de *alinhadores* e *gravidade*, para aumentar a eficiência e a precisão nas tarefas de interação.

Os alinhadores podem ser retas, círculos, planos e esferas. Os usuários definem estes alinhadores através de parâmetros como inclinação, ângulo e distância, e podem posicioná-los em todos os vértices e arestas, considerados de "grande interesse". Uma vez definidos os alinhadores, o *skitter* é sempre *atraído* a um alinhador ou a uma interseção de alinhadores, antes de realizar algum movimento. Garante-se, portanto, que o seu movimento (de *arrasto*) se realizará precisamente dentro de uma região de interesse. Esta combinação *fixar* e *arrastar* é denominada *técnica de fixação-arrasto*[5] pelo Bier.

O algoritmo de gravidade determina, no sistema de coordenadas do dispositivo, qual ponto de qual objeto é projetado mais próximo ao *cursor*, e encontra o correspondente desse ponto no sistema de coordenadas do mundo, posicionando a "origem" do *skitter* nesse ponto. Para determinar a orientação do *skitter*, seu eixo z é calculado como sendo perpendicular à aresta ou face que o atraiu, e o eixo x deve ser tangente à aresta (caso seja uma aresta) que atraiu o *skitter*. Se o *skitter* for atraído por um vértice, a orientação de seus eixos é determinada pelas faces e arestas adjacentes àquele vértice.

Para determinar o comportamento do *skitter*, quando atraído por outros objetos, o

[5]Em inglês, snap-dragging

sistema adota um critério hierárquico de preferência, na seguinte ordem:

Pontos: dá-se preferência aos pontos. O *skitter* é atraído por vértices ou pontos de intersecção de objetos, se houver algum perto do *cursor*.

Arestas: se não existirem pontos próximos ao cursor, o *skitter* é atraído pela aresta que estiver mais próxima.

Faces: se não houver uma aresta próxima ao cursor, o *skitter* é atraído pela face mais próxima.

Plano padrão: se não houver uma face próxima o cursor, o *skitter* é atraído por um *plano padrão*, que é um plano paralelo ao plano de projeção cuja profundidade pode ser alterada pelo usuário.

Note que, ao adotar o critério de preferência, vértices, arestas e faces próximos ao *cursor* sempre atraem o *skitter*, mesmo que estes estejam ocultos por algum outro objeto. Isso permite fácil acesso a partes ocultas de um objeto em foco.

Um objeto importante utilizado durante as transformações é a âncora, que pode ser usada como eixo de rotação, centro de escala ou posição com função de gravidade. A âncora pode ser posicionada precisamente, tomando sua posição e orientação a partir do *skitter*, e é similar aos *jacks* de [2], com a diferença de que pode haver somente uma âncora por cena.

A Figura 2.8 mostra uma sequência de interações com o *skitter* e a âncora. A Figura 2.8.(a) apresenta apenas o *skitter*. A Figura 2.8.(b) mostra a âncora, que pode ser posicionada em qualquer lugar através do *skitter*. Finalmente, a Figura 2.8.(c) mostra a âncora livre do *skitter* depois de posicionada.

A função de gravidade, e os alinhadores são utilizados para executar transformações afins interativamente (translação, rotação e mudança de escala). Durante uma transformação, os objetos seguem o movimento do *skitter*, que continua a ser atraído pela geometria dos objetos da cena e pelos objetos alinhadores

O comportamento do *skitter* durante uma operação de translação é definido pelo deslocamento dos objetos selecionados da posição inicial do *skitter* a sua posição final. Quando não há objetos suficientemente próximos ao *cursor* para atrair o *skitter*, o plano padrão assegura que os objetos se moverão suavemente nas duas direções da tela, com descontinuidade na profundidade cada vez que o *skitter* "pular" para algum objeto que o atrair. Essa descontinuidade auxilia a percepção de quando o *skitter* foi atraído por um objeto.

Em uma operação de rotação (Figura 2.9), os objetos selecionados giram em relação à "origem" da âncora definida pelo *skitter*. O eixo de rotação passa pelo ponto de âncora e é

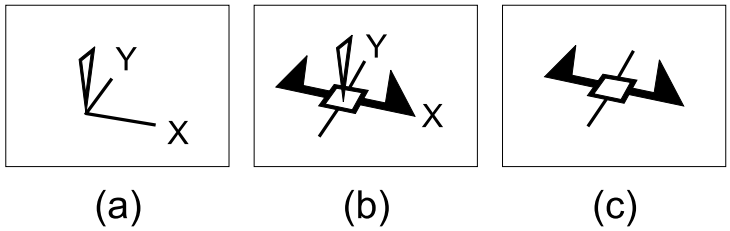


Figura 2.8: *(a) Skitter. (b) Skitter e âncora. (c) Âncora.*

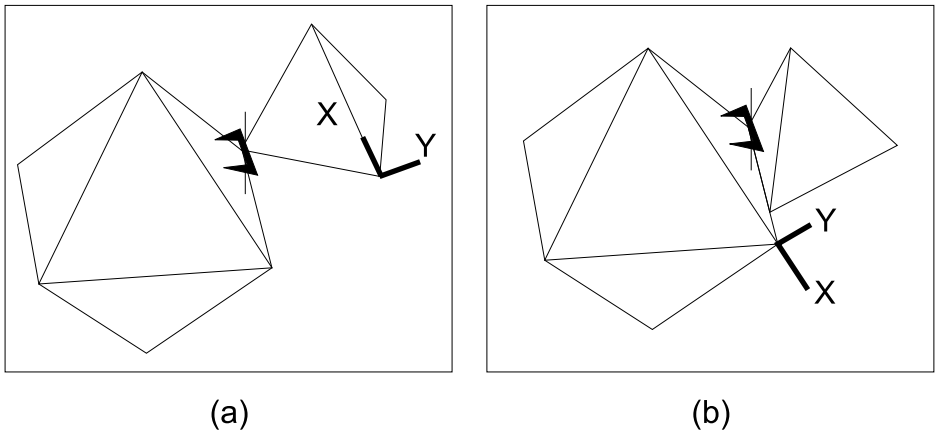


Figura 2.9: *Rotação com o auxílio da âncora.*

perpendicular ao plano determinado por três pontos: a posição original do *skitter*, da âncora e a posição final do *skitter*. O ângulo de rotação é igual ao ângulo entre a linha determinada pela âncora e a posição inicial do *skitter* e a linha determinada pela âncora e a posição final do *skitter*. A operação de rotação executada dessa forma é ideal para rotacionar duas arestas de forma que fiquem coincidentes.

Na Figura 2.10, um diagrama de transição de estados mostra a rotação da Figura 2.9, após a âncora ter sido posicionada. O estado "ESTADO DE SNAP" na Figura 2.10, corresponde à atração do *skitter* pelo vértice do tetraedro na Figura 2.9. Após posicionar o *skitter* no primeiro objeto, o *skitter* é atraído por outros objetos da cena, levando consigo o objeto selecionado. Esta ação corresponde ao estado "ESTADO DE SNAP E ROTAÇÃO". Depois de escolher o segundo ponto de *snap*, é encerrada a interação, e o objeto movido permanece em sua nova posição.

Finalmente, na mudança de escala, o fator de aumento ou redução dos objetos é determinado pela razão entre a distância da posição final do *skitter* à âncora e a distância entre a posição inicial do *skitter* à âncora.

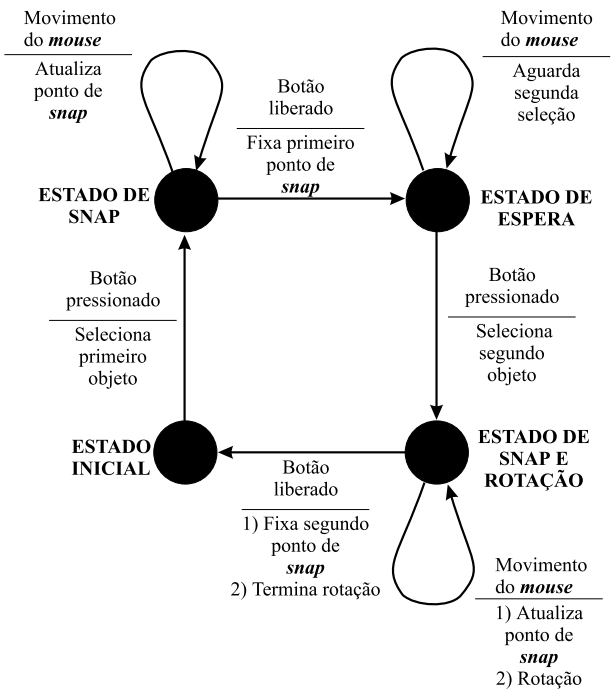


Figura 2.10: *Diagrama de transição estados para rotações com snap.*

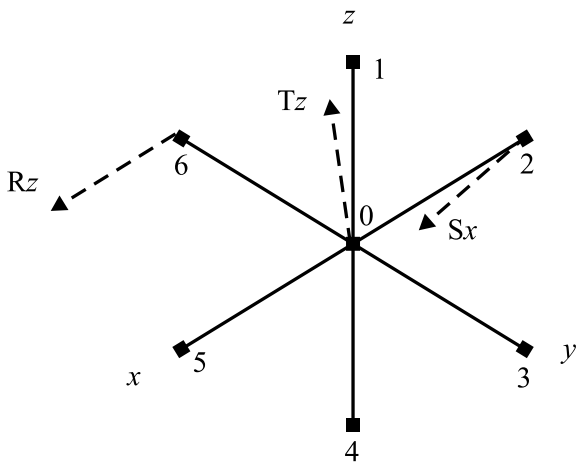


Figura 2.11: *Rz: rotação ao redor de z; Tz: translação ao longo de z; e Sx: mudança de escala ao longo de x.*

2.2.6 Proposta de Emmerik

Emmerik[10] apresentou um conjunto de tarefas de interação para *translação, rotação e mudança de escala* de objetos através das interações com as suas *projeções perspectiva ou paralela*. As técnicas de interação 2D utilizadas se restringem só a duas operações: selecionar e arrastar os componentes de um *interador* com o *mouse*.

O *interador* é representado graficamente através de três eixos que se cortam ao meio ortogonalmente. Sua geometria é semelhante à tríade, aos *skitters* e *jacks*, com a diferença de possuir 7 pontos de controle (Figura 2.11). Estes três eixos coincidem com os eixos, e o ponto de interseção, com a origem do sistema de coordenadas local do objeto respectivamente. O ponto central define uma operação de *translação*, e os outros seis pontos nas extremidades dos eixos do sistema de coordenadas para especificam as operações de *rotação* e *mudança de escala*. Para realizar essas transformações geométricas, o usuário deve selecionar um dos 7 pontos de controle e arrastá-lo. O comportamento do controlador virtual depende do ponto de controle selecionado, da direção do movimento do *mouse* e da orientação dos eixos projetados.

Por exemplo, se o usuário selecionar o ponto de controle central e arrastá-lo paralelamente ao eixo z, o sistema de coordenadas será movido na direção z. Se selecionarmos um dos dois pontos de controle do eixo x, e arrastarmos esse ponto em direção ao centro ou para fora dele, o sistema de coordenadas local sofre uma mudança de escala na direção x. Uma rotação ao redor do eixo z pode ser obtida arrastando um dos pontos de controle do eixo x paralelamente ao eixo y, ou vice-versa. Deve-se observar que as transformações estão restritas aos eixos do sistema de coordenadas, não sendo possível executá-las em relação a pontos ou vetores arbitrários. A Figura 2.12 apresenta o diagrama de transição de estados para estas 3 transformações.

Para *manipular* um objeto existente, é preciso selecioná-lo. Isto é feito selecionando um ponto na tela. É traçado, então, um raio partindo do observador e passando por este ponto. Calcula-se em seguida a interseção do raio com as superfícies dos objetos. Se existir um objeto interceptado pelo raio, esse objeto é selecionado. Se mais de um objeto for atingido pelo raio, cada objeto é selecionado sucessivamente, na ordem de mais próximo para o mais distante, ao selecionar repetidamente o mesmo ponto.

2.2.7 Proposta de Castier

Castier et al.[4] apresentaram uma proposta de interação baseada em uma "caixa envolvente". Os objetos manipulados são envolvidos por um objeto cujo símbolo gráfico é uma caixa, e a manipulação direta é feita sobre esta caixa. Os efeitos da manipulação são então aplicados sobre os objetos envolvidos pela caixa. O modelo propõe rotações através do arrasto de arestas

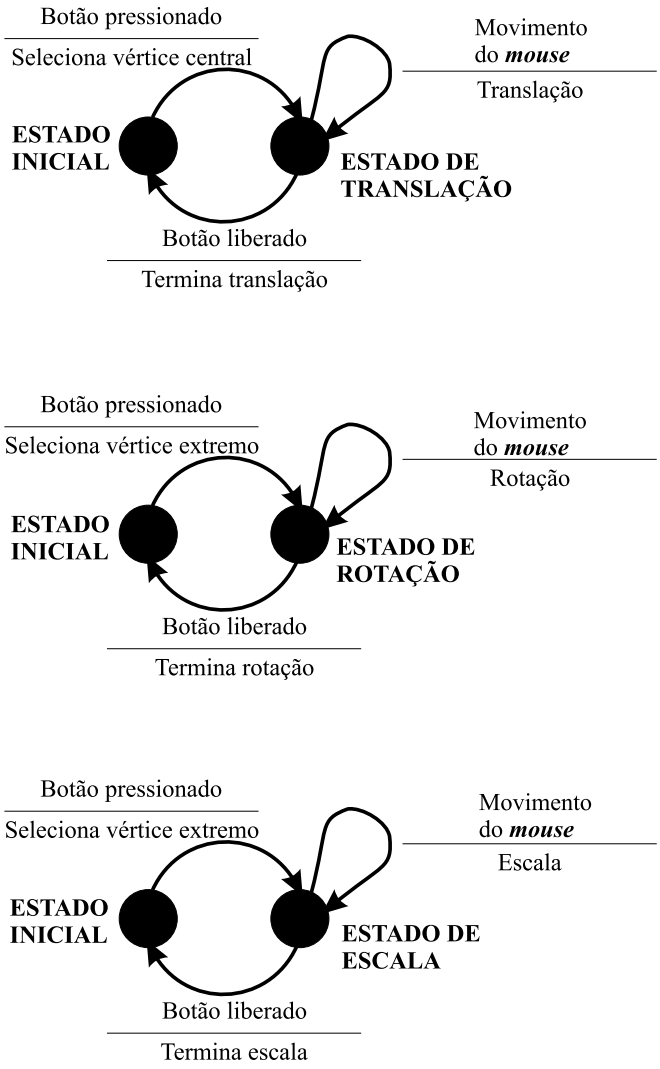


Figura 2.12: *Diagrama de transição de estados para translação, rotação e mudança de escala.*

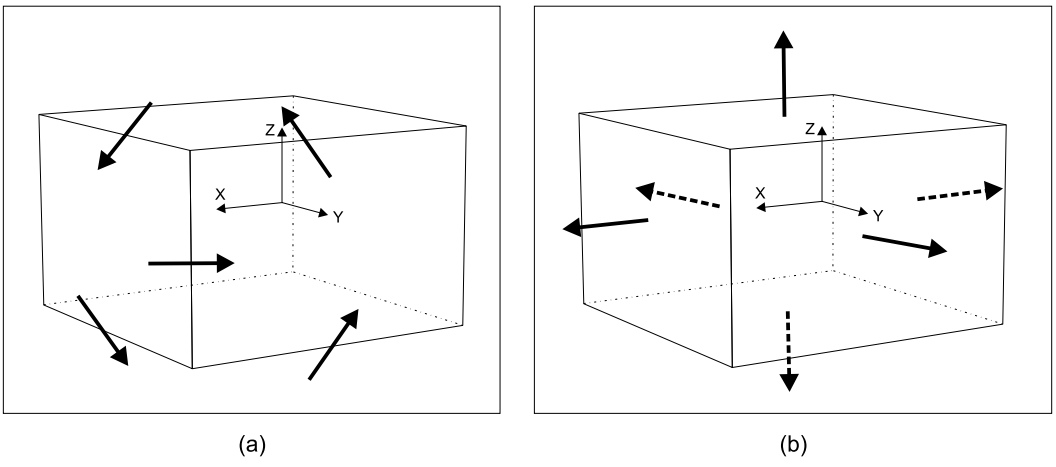


Figura 2.13: *Transformações utilizando a caixa envolvente. (a) Rotação. (b) Translação.*

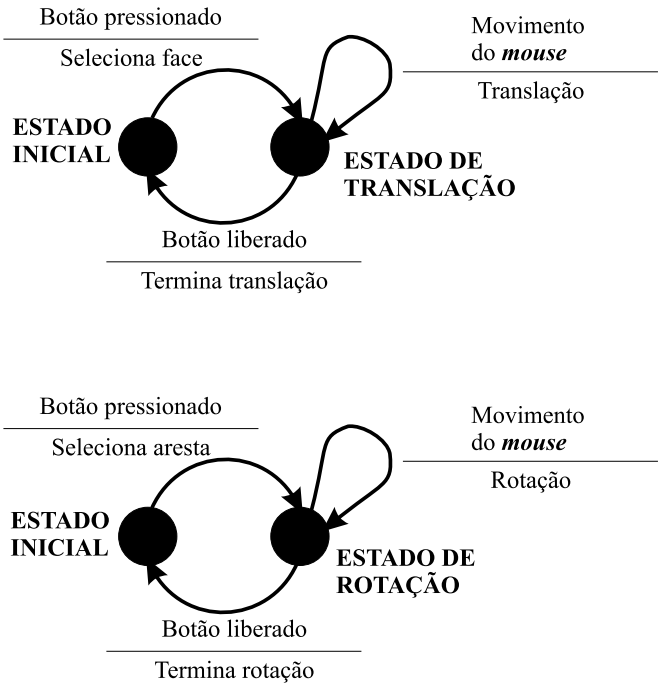


Figura 2.14: *Diagrama de transição de estados para translação e rotação.*

e translações empurrando ou puxando uma face.

Na rotação, uma aresta da caixa envolvente é selecionada e arrastada através de um *mouse*. O comportamento é uma rotação feita em relação ao eixo que passa pelo centro da caixa e é paralelo ao eixo selecionado (Figura2.13.(a)).

O ângulo de rotação e o sentido da rotação são derivados do movimento do mouse. Cada aresta contém uma direção tangente de giro que está contida em um plano perpendicular à aresta. Estas direções tangentes fazem ângulos de 45 graus com os planos adjacentes das arestas. Arrastos nestas direções resultam em um máximo de rotação possível, cujo sentido obedece à regra da mão direita. Em contrapartida, movimentos perpendiculares às direções tangentes não produzem rotação alguma.

Na operação de translação, uma face da caixa envolvente é selecionada pelo *mouse*, e a caixa comporta-se como se estivesse sendo puxada ou empurrada no sentido perpendicular à ela (Figura 2.13.(b)). A direção do movimento durante a translação é indicada pela normal da face selecionada.

O diagrama de transição de estados da Figura 2.14 mostra as ações envolvidas nas operações de rotação e translação.

Proposta	*Geometria*	*Comportamento*
Esfera virtual	círculo representando uma esfera	rotação
Tríade	3 eixos perpendiculares	translação
Skitters e jacks	3 eixos perpendiculares	Translação, rotação e mudança de escala
Eixos 3D	3 eixos perpendiculares com pontos de controle	Translação, rotação e mudança de escala
Caixa envolvente	caixa	Translação e rotação

Tabela 2.1: *Comparação de diferentes técnicas de interação.*

2.2.8 Geometria e comportamento

Através da análise das técnicas de interação apresentadas anteriormente, podemos perceber que um mesmo símbolo gráfico pode denotar diferentes comportamentos, diferentes semânticas. A tabela 2.1 resume esta comparação.

A tríade, os *skitters* e *jacks* e os eixos 3D da proposta de Emmerik, são baseadas em controladores virtuais semelhantes, cujo símbolo gráfico é representado por 3 eixos perpendiculares. Enquanto a tríade oferecia translações com direção restrita a um de seus 3 eixos, a proposta de Emmerik acrescentava a operação de mudança de escala, também na direção dos eixos, e uma operação de rotação paralela ao eixo remanescente (Seção 2.2.6). Já os *skitters* tinham seus movimentos restritos pela proximidade com os vértices, arestas e faces de objetos da cena. Estes movimentos podiam então ser usados em operações de translação, rotação e mudança de escala.

Por outro lado, podemos observar também que diferentes símbolos gráficos podem ser usadas para executar a mesma operação, como a caixa envolvente, usada em operações de rotação, e a esfera virtual, também aplicada em operações de rotação.

Esta independência entre geometria e comportamento é uma característica fundamental da nossa arquitetura, pois é um dos fatores que vai permitir a criação de diferentes técnicas de manipulação a partir dos elementos básicos de interação apresentados no Capítulo 3.

Capítulo 3

Arquitetura

Neste capítulo, apresentamos uma arquitetura para desenvolvimento de interfaces gráficas 3D com manipulações diretas, através de dispositivos de entrada 2D.

Conforme pôde ser observado na seção 2.1, existem algumas propostas de bibliotecas para a programação de aplicações com recursos de visualização e interação 3D. Vimos que, com exceção da biblioteca IQL (Seção 2.1.3), as arquiteturas apresentam o modelo geométrico integrado à interface.

Como alternativa para arquiteturas baseadas na integração de um modelo geométrico próprio, com a funções de visualização e interação, apresentamos uma nova proposta de arquitetura baseada do desacoplamento entre estes dois tipos de componentes.

Esta nova arquitetura está voltada para a construção de aplicações 3D, com enfoque no desenvolvimento, e não em seu uso por parte do usuário final. Os pontos-chave desta de nossa proposta são:

Reuso **das técnicas de interação construídas:** segundo Conner et al.[11], "as razões para separar a aplicação da interface são válidas, mas os benefícios de um único ambiente de desenvolvimento podem ser maiores do que os benefícios desta separação, especialmente em aplicações 3D". Primeiro, um ambiente integrado aumentaria a produtividade, graças à utilização da mesma ferramenta para construir tanto a aplicação quanto a interface. Segundo, seria necessário aprender apenas um paradigma, em vez de um paradigma para a aplicação e outro para a interface. E finalmente, paradigmas distintos poderiam ser difíceis de integrar, tanto na fase de projeto quanto na fase de implementação.

Todavia, a abordagem de desacoplamento apresenta vantagens. Não é necessário criar código de visualização e manipulação para inserir objetos da aplicação dentro da biblioteca, uma restrição da maioria das ferramentas analisadas no Capítulo 2. Além disso,

este desacoplamento promove a reusabilidade, pois as mesmas manipulações podem ser reutilizadas em diferentes aplicações.

Flexibilidade **para construção de diferentes técnicas de interação:** em um de seus trabalhos, Stevens et al.[36] defendem a necessidade da arquitetura prover um conjunto de primitivas de interação sem uma semântica pré-definida. Desta maneira, a aplicação não é obrigada a utilizadar as técnicas de interação impostas pela arquitetura, como é o caso das bibliotecas Open Inventor e IQL, mas cada usuário pode definir o tipo de comportamento mais adequado para as suas manipulações, como nas propostas das Seções 2.1.4 e 2.1.5.

Extensibilidade **para inserir novos objetos na arquitetura quando necessário:** mesmo que a biblioteca ofereça um conjunto extenso de primitivas de manipulação, não é possível cobrir todas as diferentes possibilidades de manipulações. Por isso, a arquitetura permite incluir novas primitivas de manipulação, visando atender a novas necessidades.

3.1 Requisitos da Arquitetura

Visando satisfazer os requisitos de reusabilidade, flexibilidade e extensibilidade, foram identificadas as seguintes necessidades:

3.1.1 Tarefas Básicas de Interação

Segundo Foley et al.[13], as *técnicas de interação* são as maneiras como os dispositivos de entrada são utilizados para inserir um dado no computador, enquanto uma *tarefa de interação* corresponde ao tipo de informação que o usuário deseja fornecer ao computador, com o uso de uma ou mais técnicas de interação. Em outras palavras, as tarefas tem a ver com *o que* será adquirido e as técnicas com *como* será adquirido. Por exemplo, o uso de *mouse* para selecionar um objeto gráfico é uma técnica de interação e o objeto selecionado é o resultado de uma tarefa de interação.

As tarefas de interação dividem-se em 2 tipos: básicas e compostas.

Uma tarefa básica de interação corresponde a uma unidade indivisível de informação, que possua um significado no contexto de uma aplicação. Quatro tarefas básicas mais usuais são:

Posicionamento: envolve a especificação das coordenadas x, y ou x, y, z de uma posição;

Seleção: envolve a escolha de um elemento em um conjunto;

Entrada de texto: entrada de uma sequência de caracteres; e

Entrada de uma quantidade: envolve a especificação de um valor numérico.

As tarefas básicas constituem um conjunto a partir do qual é possível construir tarefas de interação compostas, ou manipulações mais complexas, como:

Caixas de diálogo: para entrar um conjunto de informações não mutuamente exclusivas;

Construção: de um objeto a partir de uma sequência de posicionamentos; e

Manipulação: de um objeto gráfico existente através de uma sequência de combinações entre seleção e posicionamento ou entrada de um valor

As tarefas básicas de entrada de texto e entrada de uma quantidade funcionam da mesma maneira em um ambiente 2D ou 3D. O texto pode ser inserido através de dispositivos comuns, como o teclado, enquanto quantidades também podem ser inseridas pelo teclado ou ainda através de potenciômetros.

Entretanto, as outras duas tarefas básicas de interação tornam-se mais difíceis em 3D: o posicionamento, devido à dificuldade de mapear movimentos em 2D para movimentos em 3D, e a seleção, devido à dificuldade de perceber a relação de profundidade entre os objetos da cena.

Em nossa proposta, as tarefas básicas de seleção e posicionamento 3D constituem um conjunto conveniente de tarefas de interação, a partir dos quais é possível implementar manipulações 3D mais complexas, estas sim dependentes do contexto.

A tarefa básica de posicionamento 3D é implementada através de uma sequência de tarefas de posicionamentos 2D. Vale então ressaltar aqui que uma tarefa básica 3D implementada através da composição de um conjunto de tarefas básicas 2D, não deve ser confundida com uma tarefa composta. A tarefa básica de seleção 3D, geralmente é implementada através da técnica de ray-casting, conforme explicado na seção 2.2.5.

3.1.2 Mapeamento entre Movimentos 2D e 3D

O posicionamento de um ponto no espaço 3D através de um dispositivo 2D não é trivial. Segundo Velásquez[39], a dificuldade está no fato de que um ponto no plano de projeção pode corresponder a vários pontos no espaço 3D. Após a projeção da cena tridimensional sobre o plano da imagem, uma dimensão de cada objeto é perdida. Por exemplo, na Figura 3.1, os três pontos são projetados em um mesmo ponto na tela, sendo impossível neste caso distinguir a profundidade relativa de cada um.

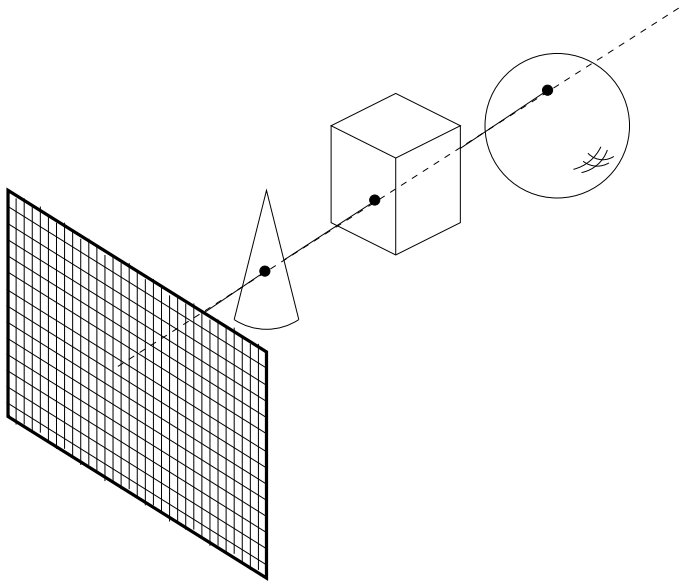


Figura 3.1: *Ambiguidade durante a projeção.*

O problema passa a ser então estabelecer uma correspondência biunívoca entre as coordenadas no espaço 3D e o espaço bidimensional da tela, de forma a recuperar a coordenada de profundidade na cena.

Na Seção 2.2, vimos que existem várias técnicas com o objetivo de estabelecer esta correspondênca entre movimentos 2D e movimentos 3D.

Algumas são baseadas em um plano de projeção, que normalmente corresponde à tela. Na técnica do *trackball* de Evans[12], por exemplo, movimentos circulares do *mouse* correspondem a rotações apenas ao redor do eixo z. A correspondência entre pontos 3D e pontos 2D é obtida eliminando dois graus de liberdade do movimento. Desta forma, movimentos bidimensionais do *mouse* geram movimentos apenas ao redor de um eixo, eliminando qualquer ambiguidade.

Outras são baseadas na geometria do objeto que deve ser manipulado, como as técnicas de Nielson e Olsen (Seção 2.2.4), que sugeriram a aplicação de transformações de translação baseada nas faces de objetos da cena. Movimentos bidimensionais do *mouse* correspondem a movimentos bidimensionais sobre a face selecionada de um objeto, o que também elimina a ambiguidade.

Outras ainda são baseadas em objetos que substituem ou representam os objetos que precisam ser manipulados, como Castier et al.[4] e Emmerik[10]. Na técnica de Emmerik (Seção 2.2.6), por exemplo, os movimentos no espaço 3D são feitos apenas nas direções dos eixos do sistema de coordenadas usado como referência. Portanto, movimentos 2D do *mouse* são

mapeados em apenas uma direção no espaço 3D de cada vez, permitindo determinar sempre um único ponto tridimensional para cada movimento 2D do *mouse*.

Segundo alguns autores[34, 19], a utilização de um objeto *virtual*, que represente o objeto que realmente deve ser manipulado, apresenta algumas vantagens sobre a técnica baseada no plano de projeção e a que utiliza a geometria do próprio objeto:

- Controlar objetos utilizando, por exemplo, uma caixa envolvente [1], provê um modo consistente para manipular desde objetos simples até objetos de formas geométricas complexas;
- É possível criar um modo composto de manipulação, que permita por exemplo integrar operações de translação e rotação no mesmo objeto virtual. Desta forma, é possível executar ajustes repetitivos de posição de uma maneira suave; e
- Pode-se criar objetos virtuais cuja forma reflita a ação a ser executada, tornando seu aprendizado mais simples. Por exemplo, um objeto virtual usado para torcer outros objetos pode ser ele mesmo torcido.

A técnica de objetos virtuais foi escolhida para executar manipulações em nossa biblioteca, por oferecer um mecanismo de interação independente da geometria do objeto. Outro fator importante é a possibilidade de incorporar mais de uma operação no mesmo objeto, diminuindo a mudança de contextos de interação. Na técnica de Castier et al. (Seção 2.2.7), por exemplo, não é necessário recorrer a menus ou botões para alternar entre operações de rotação e translação. A mudança de operação é feita diretamente sobre a caixa envolvente, alternando entre o arrasto de arestas e faces, que correspondem a rotações e translações respectivamente.

Conforme a tabela da Seção 2.2.8, o mesmo objeto virtual pode ser utilizado com diferentes semânticas. Em contrapartida, a mesma semântica de operação pode ser executada utilizando diferentes objetos virtuais.

A partir desta independência entre a geometria do objeto utilizado na manipulação, e as operações realizadas, é possível criar vários pares geometria/semântica diferentes. Em outras palavras, pode-se combinar estes dois elementos para gerar novas manipulações, de acordo com as necessidades da aplicação.

As arquiteturas da bibliotecas discutida nas Seções 2.1.4 e 2.1.5 apresentam um mecanismo semelhante, no qual o comportamento e a geometria dos objetos de manipulação podem ser combinados. Esta abordagem difere, por exemplo, daquela apresentada no *Open Inventor*, na qual vários *draggers* podem ser combinados mas, seu comportamento não pode ser alterado.

[1] Em inglês, bounding box

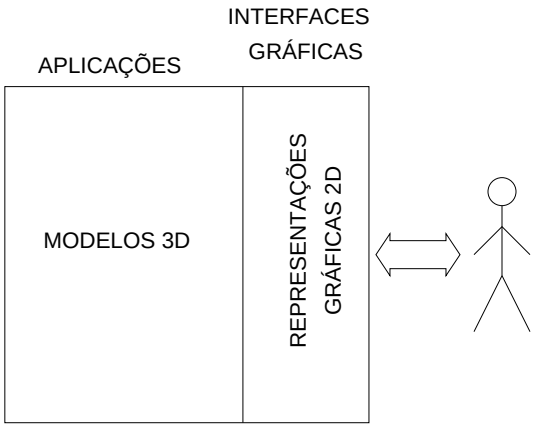


Figura 3.2: *Arquitetura orientada a objetos 3D.*

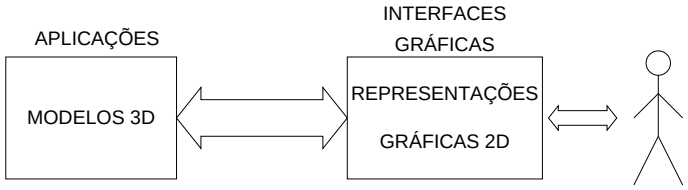


Figura 3.3: *Arquitetura orientada à visualização interativa.*

3.1.3 Desacoplamento

A biblioteca atua em resposta a requisições feitas pela aplicação, para gerar uma saída gráfica dos objetos ou para retornar informações 3D relativas a uma tarefa de interação. Para executar essas tarefas, é imprescindível que a arquitetura tenha conhecimentos dos dados da aplicação. Por outro lado, é desejável que os métodos de visualização e interação desenvolvidos possam ser utilizados em outras aplicações, mesmo que estas possuam estruturas de dados distintas.

Na Seção 2.1, vimos que a maioria das propostas resolviam este problema encapsulando o modelo geométrico, e os recursos de visualização e manipulação, dentro da própria arquitetura, um paradigma orientado a objetos 3D (Figura 3.2).

Seguindo outra abordagem, a nossa solução para satisfazer estes dois requisitos opostos foi utilizar o conceito de visualização interativa, aplicado na biblioteca IQL (Seção 2.1.3). Desta maneira, a conversão dos dados da aplicação para o formato dos dados da biblioteca é suficiente para que seja possível visualizar e manipular objetos 3D da aplicação (Figura 3.3).

A interação é baseada na atribuição de identificadores aos objetos da cena. Se uma primitiva possuir um identificador próprio, pode ser manipulada individualmente. Por outro lado, se desejamos manipular um objeto formado por várias primitivas, deve ser atribuído um

único identificador a todo este conjunto de primitivas. Os objetos devidamente identificados podem ser então selecionados e manipulados.

O processo de modularização da arquitetura poderia ser levado um passo além, através do desacoplamento dos recursos de visualização e manipulação. Entretanto, note-se que, embora a visualização e a manipulação sejam conceitualmente distintos, na prática não são independentes, principalmente quando se trata de manipulações diretas, pois as operações são efetuadas sobre a representação gráfica dos objetos visualizados.

3.2 Concepção

Na Seção 3.1, estabelecemos os requisitos de nossa arquitetura e os mecanismos para satisfazer estes requisitos.

A reusabilidade está baseada no desacoplamento dos recursos de visualização/manipulação do modelo geométrico da aplicação. A conversão dos objetos do modelo para o formato da arquitetura é suficiente para visualizá-los, enquanto a atribuição de identificadores permite selecionar e manipular os objetos.

A flexibilidade para construção de diferentes técnicas de interação está baseada no uso das tarefas básicas de interação de seleção 3D e posicionamento 3D. A partir da tarefa de posicionamento é possível definir vários comportamentos para os objetos virtuais.

Finalmente, a separação entre geometria e comportamento dos mecanismos de interação permite inserir novas geometrias de manipulação e novos comportamentos, tornando mais fácil estender os recursos providos pela arquitetura.

Para chegar a uma arquitetura que contemplasse estes requisitos, identificamos a necessidade dos seguintes componentes:

Modelo geométrico simples: para armazenar os dados gráficos visualizáveis e manipuláveis;

Identificadores: para suportar a tarefa básica de seleção 3D;

Interadores: para suportar a tarefa básica de posicionamento 3D; e

Restrições: para fazer o mapeamento entre movimentos 2D e movimentos 3D.

A arquitetura proposta tem como ponto central a construção de manipulações diretas 3D. No entanto, como as manipulações diretas são realizadas sobre a representação gráfica dos

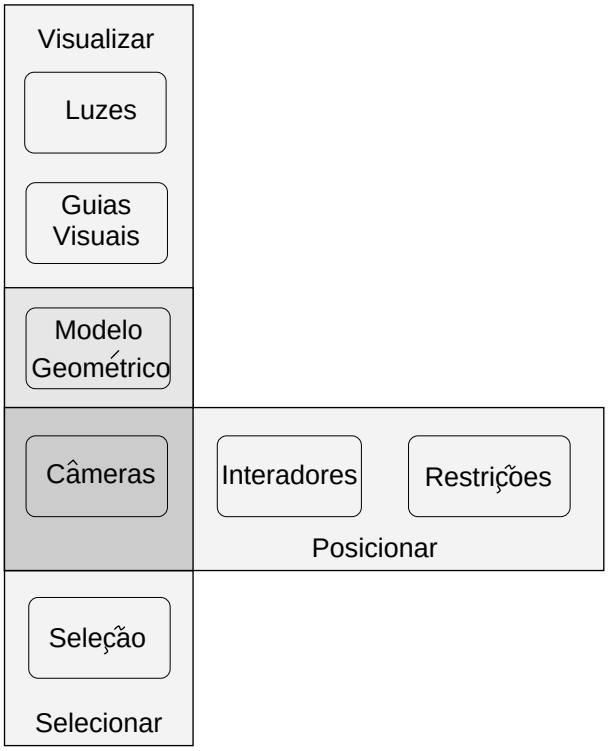


Figura 3.4: *Componentes da arquitetura.*

objetos, são necessários recursos de visualização destes objetos. Deste modo, definimos ainda outros 3 módulos referentes à visualização:

Guias visuais: para melhorar a percepção 3D dos objetos visualizados, e como realimentação visual em operações de interação;

Câmeras: para criar uma ou mais vistas da mesma cena; e

Fontes de luz: para aumentar o realismo de uma cena através de sua iluminação.

A Figura 3.4 mostra os componentes da biblioteca. O Modelo Geométrico armazena as primitivas que representam os objetos 3D da aplicação. Estas primitivas e os Interadores são visualizados através das Câmeras disponíveis, com realce visual proporcionado pelas Luzes e Guias Visuais. As primitivas e os interadores podem ser selecionados pelo componente Seleção. As primitivas podem ser manipuladas através da geometria dos Interadores, que por sua vez têm seu comportamento definido pelas Restrições.

3.2.1 Modelo Geométrico

O projeto de uma estrutura de dados depende dos algoritmos de visualização e de interação que ela pretende suportar. Como a nossa proposta é aproveitar, sempre que possível, os códigos existentes, decidimos adotar uma estrutura semelhante àquela utilizada pelos pacotes gráficos mais conhecidos. Embora esta solução requeira que todas as aplicações convertam o formato nativo de seus dados para o formato da biblioteca, podendo até perder a precisão dos dados, os nossos experimentos têm mostrado que o ganho na reusabilidade pode compensar esta deficiência. Conforme veremos no Capítulo 6, tem sido possível reutilizar as mesmas técnicas de manipulação direta na construção de aplicações baseadas em modelos geométricos distintos.

As primitivas gráficas 3D do modelo geométrico da biblioteca são pontos, arestas e polígonos. Superfícies de Bèzier foram incluídas como uma primitiva adicional, com o objetivo de tirar proveito de placas gráficas 3D, quando presentes. As primitivas são agrupadas juntamente com seus atributos gráficos, em conjuntos definidos como *elementos*. Cada conjunto possui seu próprio identificador. Quando a aplicação cria um elemento, deve ser associado um identificador exclusivo a este elemento. Este identificador será usado pela biblioteca sempre que for preciso referir-se ao elemento especificado.

Em certas situações, pode ser desejável que não seja possível selecionar um elemento. Por exemplo, uma aplicação pode desejar inserir, em cenas muito grandes, objetos que funcionem como referência para localização dentro da cena. Objetos deste tipo são inativos e não precisam ser selecionados. Neste caso, o elemento é enviado à biblioteca com uma indicação de que não deve ser selecionado. Portanto, o elemento apresenta dois estados distintos: selecionável e não selecionável.

Os pontos, as arestas e os polígonos são suficientemente básicos para visualizar a superfície de qualquer objeto da aplicação, após um processo de poligonalização apropriado. Temos, porém, o problema de associar cada primitiva do resultado da poligonalização com o objeto original. Por exemplo, uma esfera na aplicação é representada na biblioteca como um conjunto de polígonos. Com o uso de algoritmo de *ray-casting* é possível selecionar qualquer um destes polígonos (Seção 3.2.2), que pode não fazer sentido para a aplicação. Usando um "elemento" como a menor unidade de informação, é possível controlar como o objeto é percebido pela biblioteca. Embora a esfera possa ser formada por polígonos, se todos os polígonos forem agrupados em um só elemento, serão tratados como um único objeto. Por outro lado, se for associado um identificador a cada polígono, cada poligono poderá ser identificado e manipulado individualmente.

As primitivas possuem propriedades de cor e material, baseadas no modelo da biblioteca gráfica 3D *OpenGL*[27]. A cor é definida através de quatro componentes: vermelho, verde, azul

e transparência. A propriedade de material é especificada por cinco componentes:

Fator ambiente e fator difuso: fator refletido pelo objeto, independente da posição do observador;

Fator especular: fator refletido pelo objeto, dependente da posição do observador;

Fator de emissão: luz emitida pelo objeto; e

Fator de decaimento: intensidade de brilho do componente especular;

As propriedades de material só têm efeito quando utilizadas em conjunto com fontes de luz. Neste caso, é necessário ainda fornecer os vetores normais de cada vértice para gerar efeitos corretos de iluminação.

3.2.2 Seleção

Para retornar o identificador semanticamente válido de um objeto 3D visualizado em um dispositivo 2D, é necessário:

1. identificar a primitiva geométrica sobre a qual o *cursor* está localizado; e
2. determinar o identificador do conjunto ao qual ela pertence.

A técnica mais comum para executar a primeira etapa é *ray-casting*. Esta técnica consiste em determinar a interseção entre o raio de projeção que passa pela posição corrente do *cursor* com as primitivas gráficas visíveis. A ambigüidade de mais de uma interseção ao longo do raio pode ser resolvida com a especificação do *modo de identificação*. No modo *mais próximo* seleciona-se a primitiva mais próxima do observador; no modo *mais longe*, o mais afastado do observador; e no modo *ordenado*, a escolha é sequencial, do mais próximo para o mais distante.

Uma vez selecionada uma primitiva, é preciso relacioná-la ao objeto da aplicação que a contém, de forma que ações executadas sobre a primitiva sejam refletidas no objeto. Para isso, o modelo geométrico é percorrido a partir da primitiva, até localizar o seu conjunto pai. Se a primitiva possuir um identificador próprio, as manipulações são executadas somente sobre a primitiva. Por outro lado, se a primitiva fizer parte de um objeto composto, as mesmas manipulações são aplicadas a todas as primitivas que compartilham o mesmo identificador.

Após determinar os identificadores dos elementos selecionados, estes identificadores são armazenados em uma lista de elementos selecionados. Podem ser executadas algumas operações sobre esta lista, como incluir ou excluir um elemento, por exemplo. Quando é necessário realizar algum tipo de modificação nas informações geométricas dos objetos 3D, tais como fazer a

realimentação visual ou manipular elementos selecionados, esta lista de elementos selecionados é consultada para verificar quais elementos devem ser atualizados.

Na Seção 3.2.1, vimos que os elementos (conjunto de primitivas identificadas) possuem dois estados distintos: selecionável e não selecionável. Quando um elemento é selecionável, ele pode apresentar ainda 2 outros estados: um elemento pode estar selecionado ou não selecionado.

Ao selecionar um elemento, seu identificador é retornado e enviado para a aplicação. Deste modo, a aplicação pode definir as ações a serem executadas sobre este elemento, desde uma simples realimentação visual do objeto selecionado até a sua manipulação.

Este componente é responsável ainda por oferecer funções de seleção de mais alto nível:

Seleção exclusiva: apenas um elemento pode ser selecionado.

Seleção inclusiva: podem ser selecionados vários elementos.

Alternar estado: se um elemento estiver seleciona, este modo anula a sua seleção, e vice-versa.

Como exemplo, pode-se utilizar o modo de seleção exclusiva para selecionar apenas um elemento ao pressionar o botão do *mouse*, enquanto pressionar o botão do *mouse* em conjunto com a tecla *SHIFT* ativa o modo de seleção inclusiva.

3.2.3 Interadores

Um interador é um objeto virtual com uma representação gráfica 3D, constituída por um conjunto de primitivas gráficas. A cada primitiva pode ser associada uma estratégia de mapeamento de coordenadas 2D para 3D. Uma ação sobre qualquer uma destas primitivas corresponde sempre a uma tarefa de interação básica 3D, resultando na geração das coordenadas de um ponto (x, y, z).

A idéia básica é prover ao usuário ferramentas de interação com posições e orientações bem definidas para gerar operações semanticamente válidas, como translação, rotação e mudança de escala, sobre os objetos da aplicação. O usuário, ao invés de interagir diretamente com os objetos, muitas vezes com uma geometria "sem uma orientação precisa", age sobre os interadores.

Existem diferentes propostas para determinar as três coordenadas (x, y, z) a partir das duas coordenadas (x, y). Algumas são mais livres, porém imprecisas, como a proposta do Nielson e Olsen (Seção 2.2.4), e outras mais precisas, porém restritivas, como a proposta do Bier (Seção 2.2.5). Optamos pelo segundo paradigma em que os movimentos 3D são reduzidos em

movimentos sobre uma figura geométrica de dimensão menor, imersa em 3D (esta abordagem é discutida em detalhes na Seção 3.2.4). O ponto-chave é a configurabilidade das restrições sob as quais os movimentos de cada parte do interador estão sujeitos. Por exemplo, na técnica de Emmerik (Seção 2.2.6), são permitidos movimentos apenas ao longo de um dos três eixos em um dado instante. No entanto, é possível chegar em qualquer ponto tridimensional através da composição de uma sequência de movimentos em cada eixo.

Os interadores não possuem uma semântica pré-definida, mas oferecem um mecanismo através do qual seu comportamento pode ser definido a partir da aplicação. A aplicação pode criar uma interpretação a partir de uma sequência de pontos 3D gerados pelos interadores. Um mesmo interador pode ser utilizado para realizar diferentes operações, dependendo das necessidades da tarefa que deve ser executada, como translações em uma determinada aplicação ou rotações em uma segunda aplicação. Além disso, diferentes partes de um interador podem suportar operações distintas. Como discutido na Seção 3.1.2, integrar mais de um tipo de operação no mesmo interador provê um meio suave e contínuo para aplicar tais operações.

As coordenadas de um ponto (x, y, z) retornadas por um interador não são aplicadas automaticamente nas primitivas da estrutura de dados da biblioteca. Cabe à aplicação decidir quando utilizar estes valores para atualizar seus próprios dados e os dados da biblioteca, inclusive a geometria dos próprios interadores.

A vantagem de deixar esta atualização a cargo da aplicação é um maior controle sobre a exibição de informação. Por exemplo, o interador pode substituir graficamente os elementos durante as interações, no sentido de que somente o interador é representado graficamente no dispositivo de saída durante as interações. Essa substituição é conveniente, principalmente quando se trata de objetos com grande volume de dados, propiciando um aumento na eficiência do processo de interação.

Para prover uma realimentação visual sobre os conjuntos que estão sendo manipulados, a dimensão dos interadores procura acompanhar visualmente a dimensão destes conjuntos. Outro fator importante na realimentação visual é o destaque do interador em relação aos outros objetos que ele "manipula", usando atributos gráficos como cor. Este e outros atributos não devem ser impostos pela biblioteca, mas devem ser configuráveis, de acordo com as necessidades de cada aplicação.

Segundo Snibe et al.[34], a forma do interador é extremamente importante para intuir o tipo de operação que está sendo executada. Entretanto, nenhum interador conhecido apresenta uma forma que seja apropriada para todas as manipulações desejadas. Por isso, é interessante que novos interadores possam ser facilmente incluídos na arquitetura.

O interador *trackball* (Figura 3.5), por exemplo, gera a idéia de que pode ser usado para

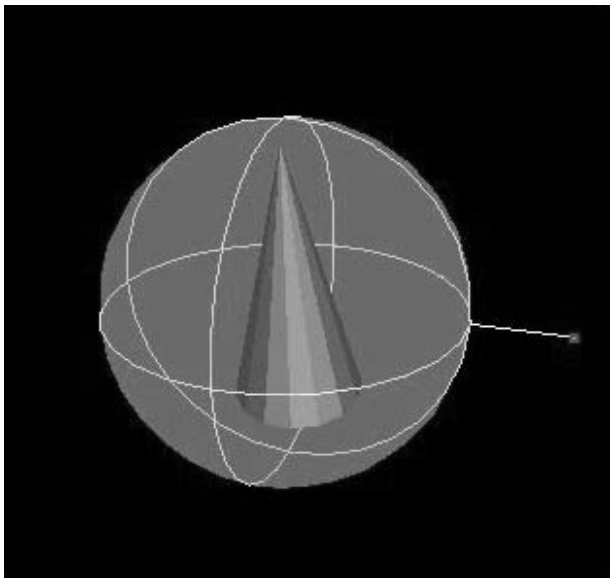


Figura 3.5: *Interador trackball.*

fazer rotações, já que seu formato esférico lembra um bola que pode ser rolada. Outro interador, que denominamos *jack* (Figura 3.6), pode ser útil para posicionar um ponto em 3D, já que seu ponto central pode ser posicionado sobre o vértice de um objeto sem obscurecê-lo.

Uma característica importante de um *widget* 3D é a sua capacidade de evitar mudanças frequentes de contexto[19]. Por exemplo, um *widget* pode ser usado para fazer duas operações distintas, ativadas através de um menu, provocando mudanças frequentes no contexto de interação. Um *widget* pode resolver este problema integrando operações diferentes em partes diferentes de sua geometria. O interador *bounding box* (Figura 3.7), por exemplo, apresenta três classes de partes distintas: vértices, arestas e faces, e a cada uma destas classes podem ser atribuídas diferentes operações. Na Seção 2.2.7 é apresentada uma técnica na qual uma caixa envolvente é utilizada para fazer translações através de suas faces e rotações através de suas arestas.

3.2.4 Restrições

As restrições são usadas para estimar as coordenadas de profundidade. Um ponto na tela pode corresponder a um ou mais pontos no espaço (Figura 3.8). A restrição corresponde a uma condição para resolver esta ambiguidade.

A obtenção da informação de profundidade, durante um movimento bidimensional do *mouse*, começa com o cálculo da intersecção de um raio disparado a partir da posição do *cursor* com a geometria que impõe a restrição. Com este cálculo, é obtido o ponto 3D correspondente

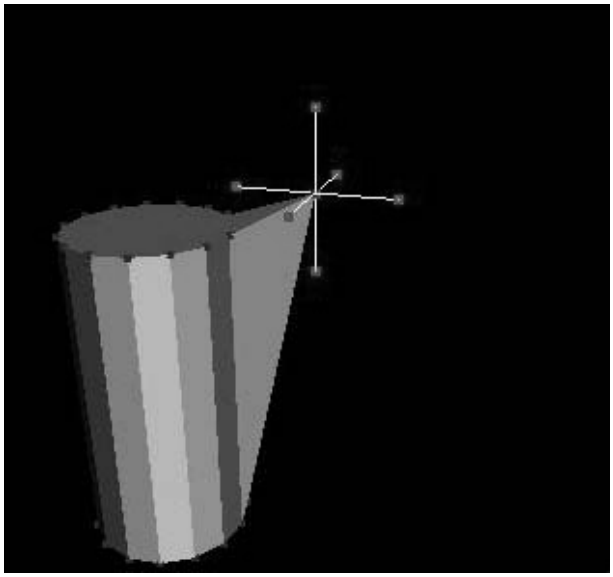


Figura 3.6: *Interador jack*.

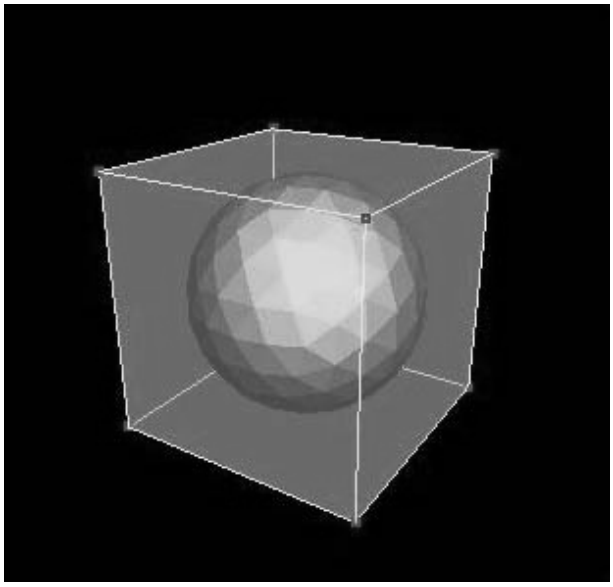


Figura 3.7: *Interador bounding box*.

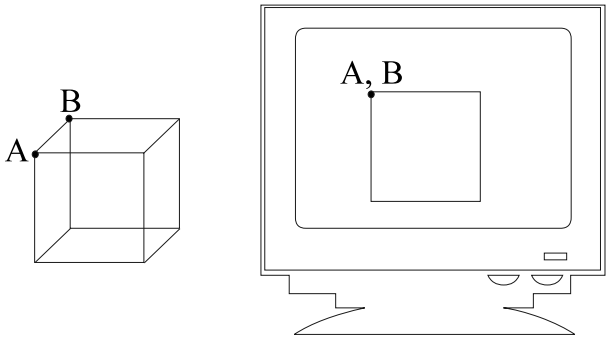


Figura 3.8: *Correspondência entre um ponto na tela e dois pontos no sistema de coordenadas do mundo.*

ao ponto 2D no plano de projeção. Os pontos 3D seguintes são deduzidos a partir deste ponto inicial e do tipo de restrição imposta.

Há vários tipos de restrições, como as discutidas no trabalho de Bier 2.2.5, onde o *skitter* é atraído pela geometria dos objetos da cena, tendo seu movimento restrito ao mover-se próximo a vértices, arestas e faces de objetos.

Nas bibliotecas apresentadas nas Seções 2.1.2 e 2.1.4, estão disponíveis restrições de movimentos sobre uma linha, sobre um plano, sobre um círculo e sobre a superfície de uma esfera.

Em nossa arquitetura estão previstas restrições sobre uma linha, sobre um plano e sobre a superfície de uma esfera. Existem também restrições paralelas a uma linha ou a um plano. De maneira similar aos interadores, também é possível incluir novos tipos de restrições em nossa arquitetura.

Adicionalmente, distinguimos dois tipos de restrições, quanto ao movimento do *cursor*: contínuas e discretas. No modo contínuo, o *cursor* "desliza" sobre a figura geométrica definida pela restrição e, no modo discreto, o *cursor* "salta" sobre esta figura.

Para flexibilizar a configuração do comportamento de uma parte de um interador, propomos definir a classe de restrições, separada da classe de interadores. No entanto, como uma manipulação é um comportamento (restrição), que ocorre sempre que há interação com a geometria (interador), é preciso relacionar estes dois componentes.

Quando uma restrição é associada a uma parte de um interador, na verdade ela está sendo registrada para esta parte do interador. Quando o interador é invocado, ele executa os métodos da restrição associada a ele, de forma que ela devolva para o interador pontos 3D corretos de acordo com a restrição escolhida.

3.2.5 Guias Visuais

Um recurso simples, porém eficiente, no auxílio à percepção visual são os *guias visuais*. Um guia visual é um objeto virtual que faz o papel de referência ou realimentação visual de uma restrição (Seção 3.2.4) usada para manipulações no espaço. Os guias podem servir como referências para as câmeras, ou podem ser usados para ampliar a noção de profundidade em uma cena. Os guias podem ser usados também para prover uma realimentação visual para restrições aplicadas aos interadores. Por exemplo, um guia sob a forma de plano pode ser utilizado para representar um "chão" sobre o qual o interador é restrito a mover-se.

São exemplos de guias visuais o eixo e a grade (Figura 4.6).

O eixo serve como referência, e é composto de 3 semi-eixos perpendiculares entre si, unidos em um extremo, à maneira de um sistema de coordenadas. Geralmente apresenta uma posição fixa na cena e orientação correspondente ao sistema de referência global.

A grade, por sua vez, é composta de linhas que se interceptam em um plano, em duas direções, formando uma malha retangular ou trapezoidal no espaço. Esta grade pode ser utilizada como uma realimentação visual para os dois modos de restrição de movimentos, contínua e discreta (Seção 3.2.4). No modo contínuo, os objetos se movem paralelamente ao plano de maneira uniforme e suave. No modo discreto, o movimento é feito aos saltos. O tamanho destes saltos pode ser definido como sendo igual ao espaçamento entre as linhas da grade, ou seja, as linhas da grade podem ser imaginadas como atratores dos objetos que estão sendo manipulados.

3.2.6 Câmeras

As câmeras são responsáveis pela criação de uma ou mais vistas particulares da mesma cena ou objeto presentes no Modelo Geométrico, e pela manutenção das transformações de visualização, que fazem a projeção de uma cena 3D em uma tela 2D. Cada câmera funciona de maneira independente, possuindo seus próprios atributos, como posição, direção, abertura e tipo de projeção.

Como mostra a Figura 3.4, uma câmera provê dados para os três blocos básicos funcionais da arquitetura: visualização, seleção e posicionamento. Os objetos tridimensionais, formados por primitivas geométricas enviadas pela aplicação, e os interadores, têm sua realimentação visual criada pelas câmeras. Os objetos tridimensionais são selecionados pelo componente de Seleção e manipulados através dos Interadores e Restrições. Tanto a seleção quanto a manipulação são feitas sobre a representação gráfica dos objetos 3D, fornecida pelas câmeras.

3.2.7 Fontes de Luz

As fontes de luz iluminam os objetos presentes em uma cena, proporcionando maior realismo e melhorando a percepção da profundidade em três dimensões.

São oferecidos três tipos de fontes de luz:

Pontual: a fonte de luz possui uma posição definida.

Direcional: a fonte de luz é considerada como se estivesse localizada a uma distância infinita.

Spot: a luz emitida é restrita a um cone de luz.

Fontes de luz também poderiam ser manipuladas através de interadores especiais e restrições. Uma luz do tipo *spot* por exemplo, poderia ser representada graficamente por um interador com a forma de cone. A abertura do cone de luz poderia ser ampliada ou reduzida através de uma restrição de movimento sobre o plano que contém a base do cone.

3.3 Relação entre os Componentes da Arquitetura

Nesta seção, será apresentada a relação entre os componentes da arquitetura, e como cada um destes componentes é utilizado para criar manipulações, visualizar e interagir com os objetos gráficos da aplicação.

Na Figura 3.9 vemos esta relação. O usuário é identificado pela letra (a), e este visualiza a representação gráfica dos objetos da aplicação através do objeto Câmera. Além disso, as ações executadas pelo usuário, como selecionar objetos e manipulá-los, também ocorrem sobre a representação gráfica exibida pela câmera.

Em uma primeira etapa, são criadas as configurações iniciais de visualização e manipulação, como o tipo de realimentação visual utilizada e o tipo de interador empregado:

- (1) A aplicação determina como deve ser a realimentação visual dos elementos selecionados.
- (2) A aplicação determina qual deve ser o interador utilizado.
- (3) As restrições são associadas a uma ou mais partes de um interador.
- (4) A aplicação converte seus dados geométricos 3D e os envia para a arquitetura.
- (5) Os dados geométricos, guias visuais e luzes são utilizados para criar uma representação gráfica dos dados.

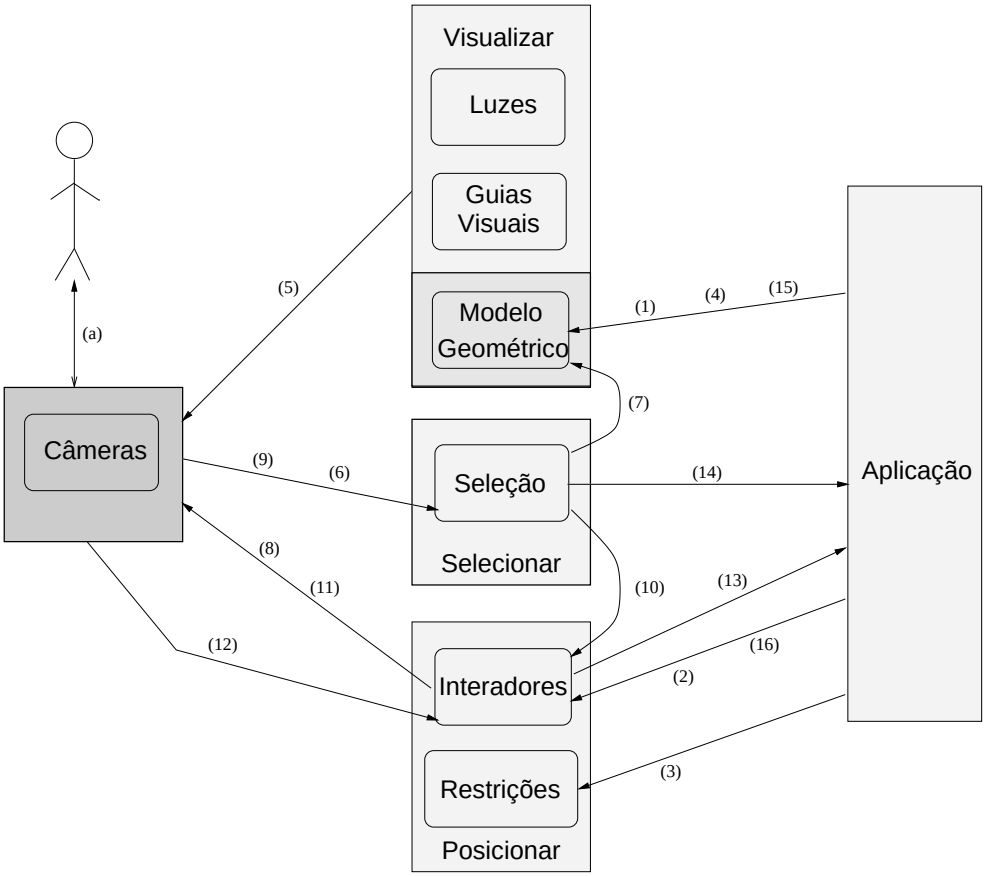


Figura 3.9: *Relação entre o componentes da arquitetura.*

Após receber a realimentação visual da cena 3D, o usuário pode selecionar objetos da aplicação:

- (6) Os elementos geométricos são selecionados.
- (7) É ativada a realimentação visual definida no passo (1).
- (8) É feita a realimentação visual da seleção, através da criação de um interador envolvendo os elementos selecionados.

O processo de manipulação começa com a seleção de uma parte de um interador, que deve estar associado a um ou mais objetos da cena 3D:

- (9) É selecionada uma parte de um interador.
- (10) É ativada a manipulação definida nos passos (2) e (3).
- (11) A interação é iniciada.
- (12) O interador devolve valores de deslocamento 3D.
- (13) Os valores de deslocamento 3D devolvidos pelo interador são passados para a aplicação.
- (14) Os identificadores dos elementos selecionados, que sofreram modificações, são passados para a aplicação.

Finalmente, a partir das informações recebidas nos passos (13) e (14), a aplicação decide se deve atualizar seu próprio modelo geométrico, o modelo geométrico da arquitetura (15), e a geometria do interador (16).

Através da combinação dos componentes da arquitetura, é possível criar os mais variados tipos de manipulação. Por exemplo, combinando Restrições de movimento com a geometria de Interadores, podem ser "fabricadas" desde manipulações precisas até interações de movimentos mais livres.

A partir das discussões deste capítulo, vimos que o conceito de reusabilidade pode ser alcançado a partir da definição de uma fronteira precisa entre modelos dependentes da aplicação e técnicas de visualização e interação. A flexibilidade foi alcançada através da definição de um conjunto básico de tarefas básicas de interação a partir do qual seja possível implementar manipulações mais complexas. A extensibilidade pode ser obtida graças à separação entre os componentes, permitindo derivar novas classes especializadas de classes genéricas.

É importante observar, ainda, que a separação entre os componentes resultou em uma definição clara dos limites funcionais de cada um deles. Esta característica foi bastante explorada na implementação desta arquitetura, apresentada no próximo capítulo.

Capítulo 4

Uma Implementação: MTK

Para validar a arquitetura proposta, foi desenvolvida uma biblioteca de funções denominada MTK, para construção de interfaces gráficas 3D.

O objetivo desta biblioteca é encapsular recursos de visualização e interação previstos na arquitetura, de forma que seja possível construir uma interface gráfica com manipulações diretas 3D.

Exemplos de aplicações serão discutidos no Capítulo 5.

4.1 Organização da Biblioteca

Toda a funcionalidade provida pela biblioteca é disponibilizada através da API de uma classe denominada MTK. Essa classe é composta de uma série de módulos que tratam de aspectos específicos do processo de visualização e interação. A estruturação de tais módulos é exibida na Figura 4.1.

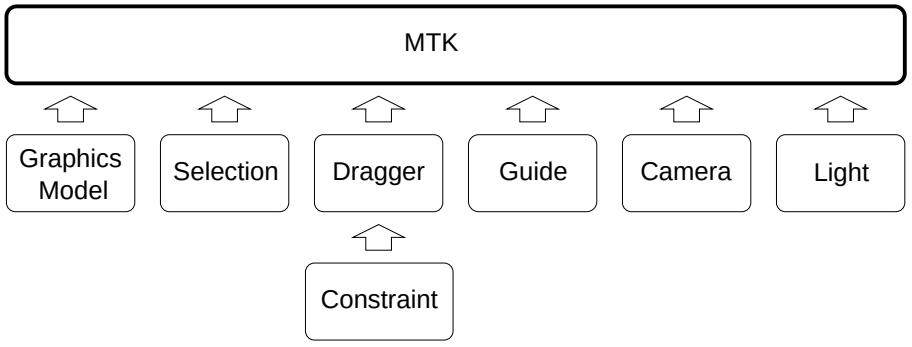


Figura 4.1: *Módulos que compõem a biblioteca.*

Cada módulo é implementado por uma ou mais classes, que fazem parte do MTK através de simples agregação. Para que a funcionalidade destes módulos possa ser acessada através da API geral, é utilizado um esquema de delegação de métodos. Ou seja, o MTK provê métodos semelhantes aos providos pelos módulos, sempre que é importante externar tais funcionalidades. Na maioria das vezes os métodos do MTK simplesmente chamam os métodos correspondentes, ao passo que em certos casos a função do módulo é estendida.

Os módulos são tipicamente compostos de uma classe base, abstrata e genérica, e de uma classe derivada, especializada para utilizar uma determinada biblioteca gráfica. No nosso protótipo foi utilizado o pacote gráfico OpenGL. Seguindo esta estrutura de classes, por exemplo, temos que a classe base para o mecanismo de seleção é `mtkSelection`, enquanto que a versão para OpenGL chama-se `mtkSelectionGL`.

4.2 Módulo Graphics Model

Este módulo é responsável pelo armazenamento dos dados geométricos exportados pela aplicação.

Toda a informação geométrica é armazenada em conjuntos de primitivas gráficas denominados *elementos*. Um elemento é identificado univocamente por um valor inteiro, não nulo, atribuído pela aplicação. Cada elemento pode conter uma ou mais primitivas gráficas.

A definição de elementos é feita por chamadas de métodos de criação de primitivas, delimitadas pelos métodos `createElement` e `finishElement` do MTK.

O método `createElement` declara um elemento com o identificador especificado. Se já existe este elemento, as primitivas declaradas substituirão o conteúdo do mesmo. Alternativamente, pode-se iniciar a declaração com `appendElement`, que adiciona as primitivas caso o elemento já exista.

A seguir, é feita a criação de uma ou mais primitivas. Cada primitiva é descrita em termos de seu tipo, especificado pelo método `begin`, e do conjunto de vértices que a define, dados através do método `setVertex`. Podem ser atribuídos aos vértices propriedades de cor ou material. Cada vértice pode tomar sua cor da cor corrente, que pode ser alterada via `setColor`. Caso contrário, cada vértice recebe a propriedade de material do material corrente, que pode ser alterado pela função `setMaterial`. No caso dos materiais, é importante atribuir um vetor normal a cada vértice, para gerar resultados corretos ao utilizar fontes de luz. As normais são especificadas através do método `setNormal`. A descrição de uma primitiva é terminada com a chamada do método `end`.

Os tipos de primitivas atualmente suportados são:

- `MTK_POINTS`: Cada vértice representa um ponto individual.
- `MTK_LINES`: Cada par de vértices define um segmento de reta.
- `MTK_LINE_STRIP`: Define uma seqüência de segmentos de reta, unidos dois a dois por um vértice em comum.
- `MTK_POLYGON`: Define os vértices de um polígono plano.
- `MTK_TRIANGLE_STRIP`[1]: Define uma sequência de triângulos, unidos dois a dois por uma aresta em comum.
- `MTK_BEZIER`: Define os pontos de controle de uma superfície de Bèzier.

A definição de um elemento é terminada com o método `finishElement`. Somente após a chamada desde método é que ocorre a efetiva criação do elemento.

O módulo Graphics Model também provê uma série de outras funcionalidades:

- métodos para apagar um elemento e testar sua existência;
- métodos para remover todos os elementos e para obter os identificadores de todos os elementos existentes;
- métodos para calcular o *bounding box* e o *bounding sphere* de um conjunto de elementos;
- métodos para definir e consultar o estado de um dado elemento;
- métodos para definir e consultar o atributo booleano de *highlight* de um elemento, juntamente com a cor que corresponde a tal estado;
- métodos para estabelecer uma função de *callback* para cada elemento; e
- métodos para aplicar matrizes arbitrárias de transformação sobre um conjunto de elementos.

O estado de um elemento pode ser: *ativo*, que torna o elemento visível e passível de seleção (Seção 4.3); *inativo*, quando o elemento ainda é visível mas não selecionável; e *invisível*, quando não é exibido.

O atributo de *highlight* permite que um elemento possa ser exibido com uma única cor temporariamente, podendo voltar às suas cores originais quando desejado. Esta funcionalidade

[1] Este tipo de primitiva ainda nao foi implementado

é particularmente útil para implementar uma realimentação visual do processo de seleção de elementos.

Cada elemento pode ter uma função de *callback* associada. Esta função é utilizada pelo MTK para sinalizar para a aplicação que o elemento sofreu uma mudança de estado de seleção, de selecionado para não selecionado ou vice-versa.

Observe que tanto a API para a definição dos elementos como os tipos de primitivas gráficas foram inspirados nas funcionalidades providas pelo OpenGL. Em muitos aspectos, um elemento tal como discutido aqui se assemelha a um *display list* do OpenGL. A diferença consiste em que um elemento pode ser alterado após sua criação, o que não é possível com um *display list*.

4.3 Módulo Selection

O módulo Selection é responsável por duas funcionalidades básicas: a seleção e o gerenciamento do estado de seleção.

A seleção é iniciada pelo usuário, tipicamente utilizando o *mouse* para posicionar o *cursor* sobre a representação gráfica de objetos da cena. Em seguida, o usuário pressiona um botão, o que é detectado pela aplicação e sinalizado pelo MTK para o módulo Selection.

A partir do ponto selecionado na tela, este módulo utiliza o mecanismo de seleção provido pela biblioteca gráfica em uso. No caso do OpenGL, é empregado o modo `GL_SELECT`, que retorna uma lista de primitivas gráficas sob o *cursor*.

Esta lista é "filtrada", obtendo-se somente elementos do módulo Graphics Model e partes de interadores que tenham sido selecionados. Este processo é controlado por diferentes modos de seleção[2]:

- `MTK_PICK_NEAREST`: retorna apenas o identificador do objeto mais próximo do observador (*default*);
- `MTK_PICK_ALL`: retorna identificadores para todos os objetos sob o *cursor*;
- `MTK_PICK_STEP`: se mais de um objeto estiver sob o *cursor*, ao primeiro aperto no botão do *mouse* é retornado o identificador do objeto mais próximo ao observador, no segundo aperto, do segundo objeto mais próximo do observador, e assim por diante.

[2] Atualmente apenas o modo `MTK_PICK_NEAREST` foi implementado.

A lista obtida é então combinada com o conjunto de elementos previamente selecionados, gerando um novo estado de elementos selecionados. Este processo de combinação é controlado por três possíveis modos de seleção:

- `MTK_SEL_EXCLUSIVE`: é possível selecionar apenas um objeto de cada vez (*default*). Se um objeto A está selecionado, selecionar um objeto B anula a seleção do objeto A;
- `MTK_SEL_INCLUSIVE`: a seleção é aditiva. Ao selecionar um novo objeto, ele é incluído na lista de objetos selecionados, e se o objeto já estiver selecionado, seu estado não se altera;
- `MTK_SEL_TOGGLE`: inverte o estado de seleção de um objeto. Se o objeto está selecionado, passa a ser não selecionado, e se não está selecionado, passa a ser selecionado.

Além das operações para configurar e consultar os modos de seleção e estado de seleção, o módulo Selection provê operações de baixo nível, que possibilitam alterar o estado de seleção de elementos sem a necessidade de ocorrer um evento de seleção. São eles:

- `setSelection`: permite configurar o estado de um objeto para selecionado ou não selecionado;
- `isSelected`: verifica se um objeto está selecionado ou não;
- `getAllSelected`: retorna uma lista contendo todos os objetos selecionados em um dado momento.

Adicionalmente, quando o estado de seleção de um elemento do Graphics Model é alterado, o módulo MTK invoca automaticamente a rotina de *callback* associada a tal elemento. Esta *callback* pode ser utilizada, por exemplo, para criar uma realimentação visual dos elementos selecionados, criando um *bounding box* envolvendo os elementos selecionados.

Por outro lado, se uma parte de um interador for selecionada, é invocada automaticamente uma função *callback* de interação que deve ser provida pela aplicação. Exemplos de atribuições para esta função *callback* são associar uma restrição a uma ou mais partes de um interador, e prover uma realimentação visual condizente com a ação do usuário, tanto para o interador quanto para os objetos geométricos da aplicação e do Graphics Model.

4.4 Módulo Constraint

O módulo Constraint implementa as restrições discutidas na Seção 3.2.4, responsáveis pela determinação dos movimentos espaciais através de ações 2D. Como vimos a restrição é um

recurso amplamente utilizado para "amarrar" a variável livre z em relação às coordenadas x e y obtidas pelo dispositivo de entrada.

Em conjunto com um interador, uma restrição é responsável por prover movimentos espaciais mais precisos em uma interação completa. Por interação completa entende-se a definição de um ponto inicial, evento denominado *pick*, zero ou mais movimentos bidimensionais, que são eventos do tipo *move*, e um evento final, chamado *release*. A sinalização destes eventos é feita pelo módulo Dragger, do qual o módulo Constraint faz parte.

Existe apenas uma classe de restrições, usada para todos os tipos de mapeamentos possíveis. Após instanciado um objeto desta classe, é preciso inicializá-lo, definindo o tipo de mapeamento a ser empregado. Esta inicialização é feita pela chamada de métodos específicos, que fornecem os parâmetros para o mapeamento desejado e colocam o objeto no estado correspondente. A partir deste ponto, o objeto passa a atuar como uma restrição do tipo especificado, até que seja feita uma nova inicialização.

Ao ocorrer um evento *pick*, é passado ao módulo um ponto 2D, em coordenadas do dispositivo, e o seu correspondente 3D, em coordenadas do mundo. Utiliza-se para tanto o método `start`.

Nos eventos *move* seguintes, são passadas somente as coordenadas de um novo ponto 2D, através do método `next`. O objeto de restrição utiliza o seu tipo de mapeamento corrente para extrapolar um novo ponto 3D, associado ao ponto 2D recebido, que é então devolvido como resultado do método. Desta forma é sempre mantida uma correspondência entre um ponto 2D, obtido por dispositivos bidimensionais, e um ponto 3D, que sempre satisfaz a restrição imposta.

Ao final de uma interação, sinalizada por um evento *release*, é chamado o método `end`, que fornece um novo ponto 3D, correspondente à nova posição do *mouse*. Na Figura 4.2, um diagrama de transição de estados ilustra a relação entre os eventos.

Os tipos de mapeamento possíveis podem ser divididos em duas categorias: restrições *contínuas* e *restrições* discretas. Os mapeamentos contínuos fazem a associação de um ponto 2D ao ponto 3D que melhor lhe corresponde. Já os mapeamentos discretos fazem a associação a um ponto 3D dentre um conjunto pré-definido de pontos. Um exemplo é a restrição a um eixo no espaço. Se tal restrição for contínua, qualquer ponto sobre este eixo é um candidato válido a ser escolhido como correspondente a um ponto 2D. Se for discreta, apenas serão válidos um conjunto finito de pontos sobre este eixo.

Considerando se um mapeamento leva ou não em conta o ponto inicial da interação, outra distinção possível pode ser feita entre mapeamentos *absolutos* e *relativos*. Por exemplo, uma restrição absoluta a um dado plano faria com que os pontos 3D estivessem sempre sobre

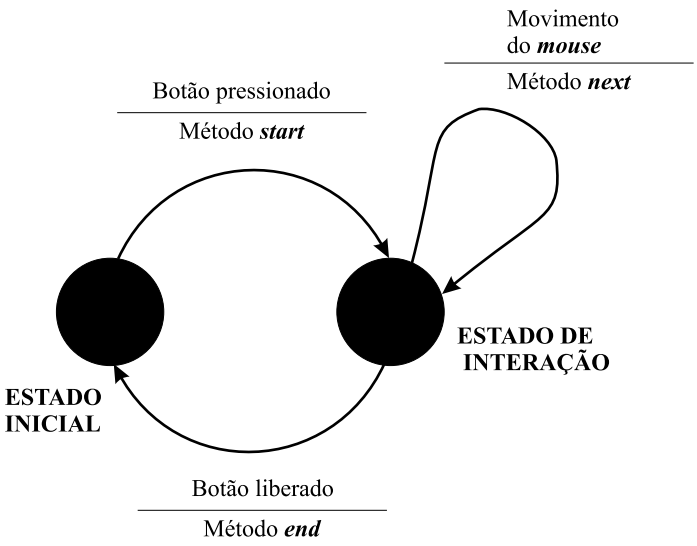


Figura 4.2: *Diagrama de transição de estados para as restrições.*

este plano. Caso a restrição fosse relativa, tais pontos estariam restritos a um plano paralelo ao plano de restrição e que contém o ponto inicial.

O módulo Constraint atualmente provê os seguintes métodos para a definição de restrições:

- `setSameDepth`: Restringe o mapeamento a um plano paralelo ao plano de visualização. É uma restrição contínua e relativa.
- `setSameDepthHorizontal`: Limita o mapeamento a um plano paralelo ao plano de visualização, mas apenas na direção horizontal da janela de projeção. É uma restrição contínua e relativa.
- `setSameDepthVertical`: Restringe o mapeamento a um plano paralelo ao plano de visualização, mas apenas na direção vertical da janela de projeção. É uma restrição contínua e relativa.
- `setSnapAxis`: Corresponde a uma restrição a um eixo especificado. É contínua porém absoluta.
- `setSnapNearestAxis`: Restringe o mapeamento a um eixo dentre três especificados. A escolha é feita pelos dois primeiros pontos bidimensionais, que definem um vetor 2D. Esse vetor é então comparado com a projeção de cada um dos três eixos, sendo escolhido o eixo que apresentar o menor ângulo com o vetor 2D. A partir deste momento a restrição se aplica somente ao eixo escolhido. É contínua e absoluta. Pode ser aplicado, por exemplo, no movimento de translação da tríade, descrito na Seção 2.2.4. Nesta técnica, o

movimento no espaço é definido na direção do eixo mais próximo ao vetor de movimento do *mouse*.

- `setSnapParallelAxis`: Restringe o mapeamento a um eixo paralelo ao eixo especificado, passando pelo ponto inicial. É contínua e relativa. Ao utilizar a restrição `setSnapAxis`, deslocamentos do *mouse* fora do eixo de restrição não produzem movimentos, requerendo uma grande precisão do usuário ao mover o dispositivos. Por outro lado, ao definir a restrição como paralela a um eixo, através do método `setSnapParallelAxis`, permite que movimentos menos precisos do *mouse* ainda resultem em deslocamentos no espaço.
- `setSnapNearestParallelAxis`: Semelhante a `setSnapNearestAxis`, com a diferença de que o mapeamento ocorre em um eixo paralelo ao eixo escolhido, passando pelo ponto inicial. Pode ser aplicado em substituição ao método `setSnapNearestAxis`, pois também permite um movimento menos preciso por parte do usuário.
- `setSnapPlane`: Corresponde a uma restrição a um plano especificado. É contínua e absoluta.
- `setSnapParallelPlane`: Restringe o mapeamento a um plano paralelo a um plano especificado e que contém o ponto inicial. É contínua e relativa. Pode ser usado, por exemplo, para definir a restrição do movimento paralelo ao plano de projeção, ou seja, os movimentos seriam paralelos à tela.
- `setSnapSphere`: Restringe o mapeamento a uma esfera especificada. É contínua e absoluta.

Com exceção das restrições `setSameDepth`, `setSameDepthHorizontal` e `setSameDepthVertical`, as outras restrições possuem uma versão de seu comportamento para o caso discreto. O método recebe o mesmo nome, com o acréscimo da letra "D": `setSnapAxisD`, `setSnapParallelAxisD`, `setSnapNearestAxisD`, `setSnapNearestParallelAxisD`, `setSnapPlaneD`, `setSnapParallelPlaneD` e `setSnapSphereD`.

Para efetuar os diversos tipos de mapeamentos descritos, o módulo Constraint precisa conhecer a transformação de visualização utilizada pela câmera corrente, sobre a qual é executada a interação. Essa informação é obtida consultando o estado do OpenGL, a partir do qual são calculadas as matrizes de transformação direta e inversa, provendo um mapeamento 1:1 entre os sistemas de coordenadas do mundo (objetos) e da tela de projeção (representação gráfica dos mesmos).

Adicionalmente, este módulo provê métodos para o cálculo do ponto 3D correspondente a um ponto 2D dado sobre a projeção de uma aresta ou face. Esta funcionalidade é utilizada pelos interadores.

4.5 Módulo Dragger

O módulo Dragger implementa os objetos virtuais denominados interadores, descritos na Seção 3.2.3.

Um interador é um objeto sensível, na medida em que pode sofrer a ação do usuário. Cada tipo de interador é composto de partes individualmente indentificáveis, conforme veremos mais detalhadamente nas Seções de 4.5.1 a 4.5.3.

Cada grupo de partes do mesmo tipo de um interador tem um objeto de restrição associado. Este objeto tem por função mapear a interação bidimensional feita pelo usuário em um deslocamento no espaço 3D. A restrição utilizada por cada grupo de partes é configurável externamente, sendo válidas todas as técnicas descritas na Seção 4.4.

Observe que os interadores são totalmente autônomos, não tendo nenhuma relação com os elementos armazenados no Graphics Model. Qualquer associação é mantida externamente ao MTK, pela aplicação. Em particular, os interadores são usualmente criados sob demanda pela aplicacão, que passam a ficar responsáveis pela existência dos primeiros e por sua inicialização.

Como não possui uma semântica própria, ao sofrer uma ação do usuário um interador chama uma *callback* para sinalizar este evento. Essa *callback* é tipicamente estabelecida pela aplicação. Os dados passados através da *callback* informam a parte do interador que gerou o evento, o ponto tridimensional associado à interação e o deslocamento ocorrido.

Uma vez de posse dos dados de uma interação, a aplicação pode fazer a realimentação visual adequada ao interador associado, conforme a semântica por ela definida. Isto é feito através de um método que permite a aplicação de uma matriz de transformação às coordenadas do interador. Por exemplo, se uma ação sobre um vértice de um interador do tipo *bounding box* significa uma translação, a aplicação usará uma matriz de translação para fazer com que o interador mude de posição.

De maneira semelhante aos elementos gráficos do módulo Graphics Model, os interadores também podem ter três tipos de estado: *ativo*, que torna o interador visível e passível de interação; *inativo*, quando o interador é visível mas não pode sofrer ações; e *invisível*, quando não é exibido.

O módulo Dragger implementa uma classe genérica de interador, do qual são derivados, no momento, os três tipos mais populares: *bounding box*, *trackball* e *jack*. Na Seção 5.1, veremos como se pode combiná-los com as restrições apresentadas na Seção 4.4 para construir distintas manipulações.

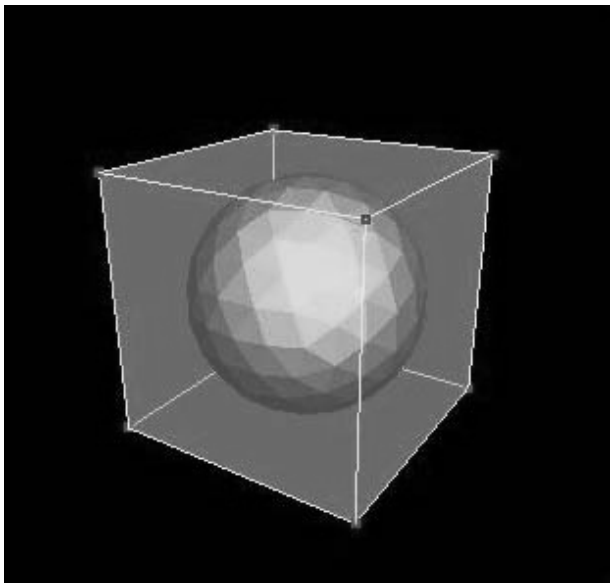


Figura 4.3: *Interador bounding box.*

4.5.1 Bounding Box

O interador do tipo *bounding box* consiste de uma caixa retangular, composta de 8 vértices, 12 arestas e 6 faces (Figura 4.3).

A forma deste interador é definida por seus vértices, que podem ser alterados pelo método `setBoxVertices`, definido pelo MTK. Atributos como tamanho e cor dos vértices, espessura e cor das arestas, e cor e opacidade das faces podem também ser modificados.

Tipicamente este interador é criado envolvendo um conjunto de elementos gráficos que estão selecionados, para serem manipulados como um todo. Apesar de ser possível modificar a forma deste interador para algo mais arbitrário, tal como um paralelepípedo, é comum usar-se a forma com ângulos retos e arestas paralelas aos eixos coordenados.

4.5.2 Trackball

Um interador do tipo *trackball* consiste de uma casca esférica, cortada por três círculos máximos mutuamente perpendiculares, e uma *flecha*, representada por um segmento de reta com um ponto na extremidade (Figura 4.4). Cada círculo é tratado como uma parte individual.

A forma deste interador é definida pelo centro da esfera, pelo seu raio e pelo tamanho relativo da flecha. Seus atributos gráficos consistem do tamanho e cor do ponto, da espessura e cor dos segmentos (tanto da flecha como dos círculos), da cor e opacidade da superfície, e da resolução da esfera. Por resolução entende-se o nível de discretização de uma esfera ideal em

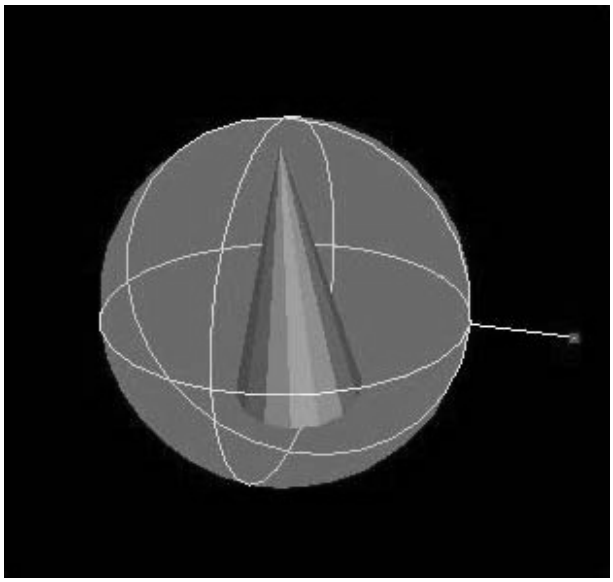


Figura 4.4: *Interador trackball*.

termos de polígonos; quanto maior a resolução, mais fiel será a sua representação gráfica.

O uso mais comum deste interador é envolver apenas um objeto, possibilitando que ele seja rotacionado em torno do centro da esfera e que tenha sua escala modificada através da flecha.

4.5.3 Jack

O interador do tipo *jack* é composto de um ponto central e de seis segmentos de reta que partem deste ponto, limitados por mais 6 pontos (Figura 4.5). Cada ponto e segmento é tratado individualmente.

Este interador é definido pela posição de seus 7 pontos. Podem ser alterados os atributos gráficos relativos à cor e tamanho dos pontos e da espessura e cor dos segmentos.

Um *jack* é normalmente utilizado como referência para uma operação. Neste caso seus segmentos são colocados de maneira a formar 3 eixos ortogonais, alinhados com os eixos coordenados, e com segmentos de tamanho uniforme.

4.6 Módulo Guide

O módulo Guide fornece ferramentas de auxílio à visualização (vide Seção 3.2.5). Em particular este módulo implementa as ferramentas `axis 3D` e `grid 3D` (Figura 4.6).

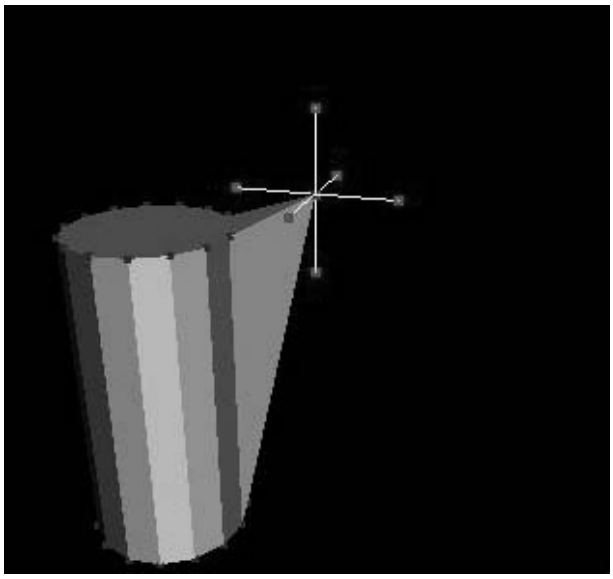


Figura 4.5: *Interador jack*.

O `axis 3D` é um sistema de coordenadas orientado de acordo com o sistema de coordenadas do mundo. É construído visualmente por 3 eixos unidos em um extremo comum. Esta ferramenta tem as seguintes propriedades:

- atributos geométricos, tais como posição no espaço, comprimento e direção de cada um dos três eixos; e
- atributos gráficos, como cor e espessura dos eixos.

O `grid` é formado por um paralelogramo construído sobre um plano qualquer no espaço. Ele é construído através de linhas paralelas às direções de largura e altura do paralelogramo. O espaçamento entre linhas paralelas entre si deve ser igual, mas o espaçamento entre linhas que se interceptam pode ser diferente. Suas propriedades são as seguintes:

- atributos geométricos, representados por um ponto e dois vetores quaisquer no espaço. A direção de cada vetor fornece as duas direções das linhas do `grid`, enquanto o comprimento de cada vetor fornece o espaçamento entre linhas paralelas. O ponto serve como referencial e define um dos vértices do paralelogramo;
- número de linhas na direção dada pelo primeiro vetor e número de linhas na direção dada pelo segundo vetor;
- atributos gráficos, tais como cor, espessura das linhas e indicação se o `grid` é vazado ou preenchido; e

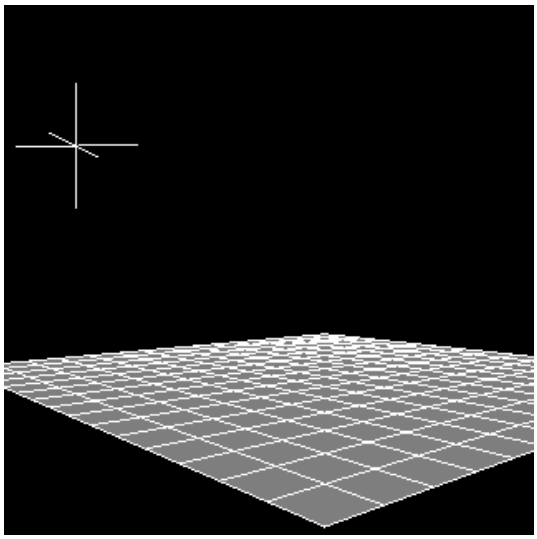


Figura 4.6: *Guias visuais*.

- visibilidade, indicando se o `grid` é exibido ou não.

Observe que estas ferramentas são passivas e incapazes de sofrer ação do usuário. Elas podem ser utilizadas como realimentação visual das restrições impostas pela aplicação a um interador.

4.7 Módulo Camera

O módulo Camera implementa um modelo de câmera alternativo, tendo como base o modelo empregado pela biblioteca OpenGL.

Neste modelo, uma visão do espaço tridimensional é especificada pelos seguintes parâmetros:

- `aim`: Indica o centro de interesse da câmera, ou seja, para onde ela está apontada.
- `position`: Indica aonde se localiza a câmera.
- `near`: Representa a distância em que é colocado o plano de corte frontal.
- `far`: Indica a distância em que fica o plano de corte de fundo.
- `window`: Representa o tamanho vertical da janela de visualização.
- `FOV`: Indica o ângulo de abertura vertical da câmera.
- `roll`: Representa o ângulo de rotação da câmera em torno de seu eixo ótico, isto é, do eixo que parte da câmera e vai até o centro de interesse.

- `projection`: Indica o tipo de projeção a ser utilizado.
- `x, y, width` e `height`: Representam a posição e as dimensões do *viewport* associado à janela de visualização.

A razão de aspecto (*aspect*) é obtida internamente a partir das dimensões do *viewport*. Esta mesma razão é utilizada para calcular o tamanho horizontal da janela de visualização, garantindo uma projeção 1:1 para a câmera.

A câmera pode operar tanto com projeções perspectivas como com projeções paralelas. Para projeções perspectivas, o parâmetro `window` é ignorado, ao passo que para projeções paralelas não se utiliza o parâmetro `FOV`.

Adicionalmente aos métodos de consulta e modificação dos parâmetros citados, é provido um conjunto de operações de mais alto nível, para a conveniência da aplicação:

- `dolly`: Corresponde ao movimento de se aproximar ou se afastar do centro de interesse, seguindo ao longo do eixo ótico.
- `tracking`: Efetua a movimentação da câmera em um plano perpendicular ao eixo ótico. Equivale a ir para cima, para baixo ou para os lados, em relação ao plano de projeção.
- `orbit`: Efetua rotações em torno do centro de interesse, sempre voltado para o mesmo e mantendo a mesma distância.
- `pan`: Corresponde ao movimento de girar a câmera em torno de seu eixo vertical, mantendo a mesma posição. Equivale a um giro para os lados.
- `tilt`: Semelhante ao `pan`, porém em relação ao eixo horizontal. Equivale a um giro para cima ou para baixo.
- `roll`: Efetua rotações em torno do eixo ótico.
- `zoom`: Corresponde a um efeito de *zoom*. Faz a alteração de `FOV` para projeções perspectivas e de `window` para projeções paralelas.
- `look`: Aponta a câmera para uma esfera de referência, fazendo com que sua projeção se ajuste à tela. Esta operação é estendida pelo MTK para observar quaisquer objetos visíveis.
- `level`: Faz o nivelamento da câmera, alinhando o eixo ótico ao plano XZ.

As classes que fazem parte deste módulo implementam apenas uma câmera. Porém, o MTK é capaz de instanciar diversas destas classes simultaneamente, fazendo com que a biblioteca suporte múltiplas câmeras.

4.8 Módulo Light

O módulo `Light` é baseado no modelo de fontes de luz do OpenGL, mas com acesso às funcionalidades através de algumas abstrações de mais alto nível.

Há três tipos de fontes de luz disponíveis:

- `pontual`: a fonte de luz possui uma posição na cena, e irradia em todas as direções.
- `direcional`: a fonte de luz é considerada como se estivesse localizada a uma distância infinita, irradiando em uma única direção.
- `spot`: a fonte de luz possui uma posição na cena, e é irradiada no formato de um cone de luz.

Uma fonte de luz do tipo *spot* possui dois parâmetros adicionais em relação aos outros dois tipos de fontes de luz:

- `exponent`: concentração da luz na área do cone.
- `cuttoff`: abertura do cone de luz.

A cor de uma fonte de luz e seu efeito sobre os objetos da cena são resultados da combinação dos seguintes componentes:

- `ambiente`: é um tipo de luz espalhado pelo ambiente, a ponto de não ser possível determinar sua direção.
- `difuso`: a luz difusa vem de uma única direção, e se espalha em todas a direções ao atingir um objeto.
- `especular`: a luz especular vem de uma única direção, e é refletida em uma direção particular.
- `emissão`: é o tipo de luz emitido a partir de um objeto.

No mundo real, a intensidade da luz diminui à medida que a distância aumenta. Como uma luz direcional está localizada a uma distância infinita, sua intensidade não é atenuada. Já no caso de fontes de luz do tipo pontual ou *spot* podem sofrer três tipos diferentes de atenuação: constante, linear e quadrática.

Existem dois tipos de estados para habilitar ou desabilitar as fontes de luz. Primeiro, a iluminação da cena deve ser "ligada", através do método `setLighting`. Em seguida, cada fonte de luz desejada deve ser acionada individualmente, pelo método `setLight`, indicando como parâmetro o identificador da fonte de luz, em um total de 8 fontes de luz possíveis em uma cena.

4.9 Detalhes de implementação

A biblioteca MTK foi implementada em C++. As estruturas de dados foram baseadas no mecanismo de templates da linguagem, mais especificamente no padrão STL (Standard Template Library), um conjunto de classes que oferecem vários tipos de estruturas de dados e métodos para a manipulação destas estruturas.

O MTK foi construído sobre a biblioteca gráfica 3D *OpenGL*, utilizando suas funções de visualização tridimensional. Também foi utilizada uma versão do *OpenGL*, implementada em *software* e disponível gratuitamente.

Os códigos foram compilados em *workstations Sun*, rodando o sistema operacional *SunOS*, máquinas RISK/6000 da IBM, rodando o sistema AIX, e em *PC*'s rodando o sistema Linux, sem a necessidade de alteração de código.

A biblioteca *OpenGL* foi utilizada em *workstations Ultra*, da *Sun*, com a placa gráfica aceleradora *Creator 3D*. Nas demais máquinas, sem *hardware* gráfico especial, foi utilizada a implementação por *software* do *OpenGL*, *Mesa*, apresentando um desempenho satisfatório.

Capítulo 5

Resultados e Discussão

Este capítulo mostra os resultados obtidos com a biblioteca MTK, implementada segundo a proposta apresentada no Capítulo 3. Na Seção 5.1 serão mostrados os detalhes de implementação das manipulações criadas. Na Seção 5.2, serão discutidos alguns testes realizados durante a construção das manipulações. Na Seção 5.3, será apresentado um modelador geométrico de sólidos, no qual foram aplicadas algumas manipulações construídas a partir do MTK.

5.1 Algumas Manipulações

Nesta Seção, serão apresentados os detalhes da criação das manipulações utilizadas no modelador geométrico da Seção 5.3, as manipulação através do *bounding box* e do *jack*. Será apresentada ainda um terceiro exemplo de manipulação através do interador *trackball*.

5.1.1 Bounding Box

Foram implementadas três tipos de manipulações, a partir do interador *bounding box*. Na Figura 5.1, vemos as restrições associadas a cada parte do interador para obter as operações de translação, rotação e mudança de escala.

A translação foi implementada utilizando a restrição `setSnapPlane` (Seção 4.4), associada às faces da caixa, produzindo movimentos de translação somente sobre o plano que contém a face selecionada. Duas posições consecutivas p_1 e p_2 do cursor são suficientes para obtermos o vetor de deslocamento.

Na rotação, a restrição `setSnapParallelPlane` foi associada às arestas. Movimentos paralelos ao plano perpendicular à aresta selecionada, produzem uma rotação cujo ângulo é igual

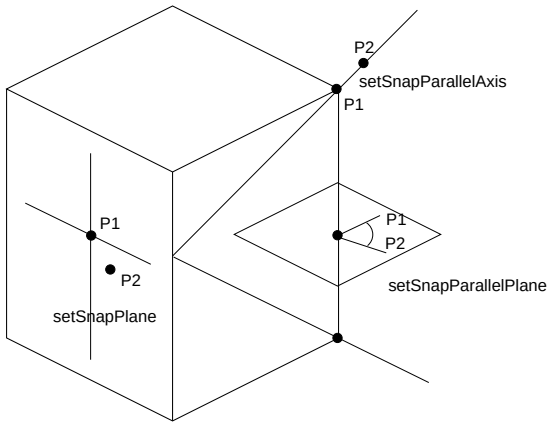


Figura 5.1: *Restrições utilizadas juntamente com o interador bounding box.*

ao ângulo entre a linha formada pelo novo ponto 3D e o centro da caixa, e a linha formada pelo ponto 3D anterior e o centro da caixa.

Em operações de mundaça de escala, foi usada a restrição `setSnapParallelAxis`, associada aos vértices do *bounding box*. Essa restricão limita o movimento apenas ao eixo determinado pelo vértice v_i selecionado e o centro da caixa.

A Figura 5.2 apresenta o diagrama de transição de estados para as três manipulações implementadas com o interador *bounding box*.

Para elucidar os detalhes de implementação, vejamos alguns trechos de código concernentes à operação de mudança de escala baseada nos vértices. Todas as operações descritas nesta Seção seguem o mesmo esquema.

É interessante criar uma função *callback* de seleção, que deve ser registrada para objeto da aplicação visualizável pelo MTK. Cada vez que um objeto da aplicação for selecionado, esta *callback* é chamada. Dentro desta função, podem ser colocadas, por exemplo, tarefas responsáveis pela realimentação visual dos elementos selecionados. Em nosso caso, esta realimentação consiste em calcular o tamanho do *bounding box*, de forma que ele envolva todos os objetos selecionados, e passar essa informação à biblioteca, para que a caixa seja criada com a posição e tamanho corretos.

No trecho de código a seguir, a função `callback_select`, provida pela aplicação, é responsável por estas tarefas. O método `calcBoundingBox` do MTK recebe como parâmetro uma lista com os identificadores dos elementos selecionados. Internamente, as informações geométricas dos elementos selecionados são consultadas, gerando as coordenadas limitantes da caixa. Esta informação é passada para o método `setBoxGeometry`, responsável por desenhar a caixa.

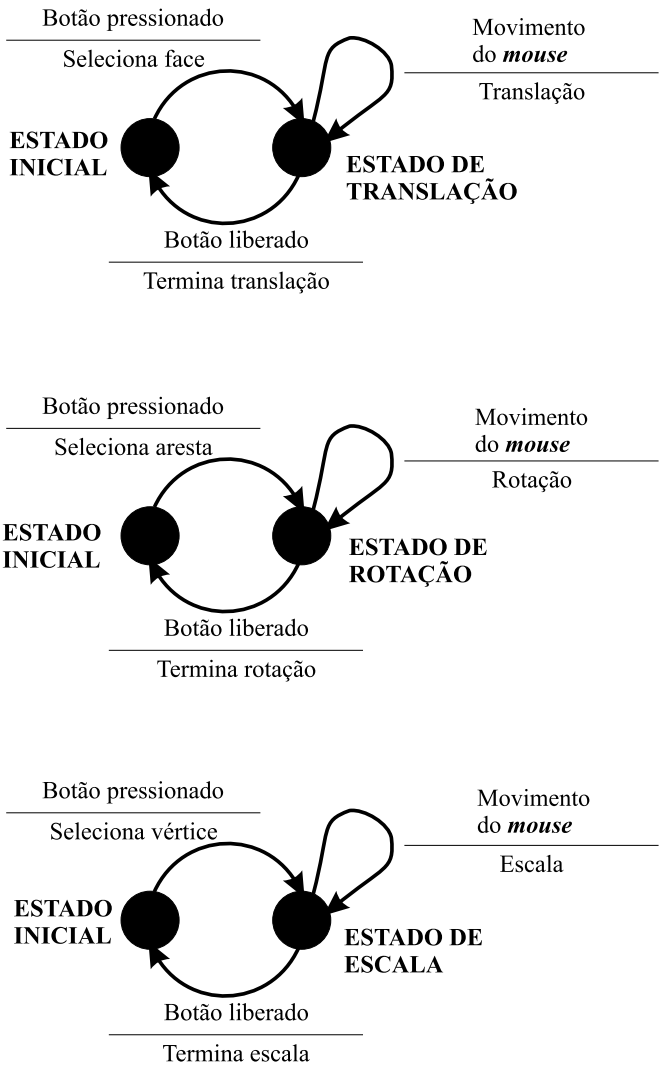


Figura 5.2: *Diagrama de transição de estados para as operações de translação, rotação e mudança de escala, utilizando o interador bounding box.*

```
callback_select (selected)
{
  mtk->calcBoundingBox(selected, &min, &max);
  mtk->setBoxGeometry(min,
                      gmVector3(0.0, 0.0, max[2]-min[2]),
                      gmVector3(0.0, max[1]-min[1], 0.0),
                      gmVector3(max[0]-min[0], 0.0, 0.0));
}
```

É importante que a aplicação forneça uma função *callback* para o processo de interação, que será registrada no interador desejado. Em nosso exemplo, esta função deve ser registrada no interador *bounding box* através do método `setBoxCallback`. A *callback* registrada recebe as seguinte informações:

- o identificador (`mtkDraggerId`) do interador que estiver sendo manipulado;
- a parte do dragger (`mtkDraggerPart`) que estiver sendo manipulada;
- o último ponto 3D calculado pelas restrições;
- um vetor 3D indicando o deslocamento do último ponto para o novo ponto;
- o tipo de evento (`mtkEvent`), que pode ser um evento de botão do *mouse* pressionado, em arrasto, ou liberado.

Em nosso próximo exemplo, a função `callback_interact` corresponde à *callback* de interação:

```
callback_interact (mtkDraggerId id,
          mtkDraggerPart part,
          gmPoint3 p,
          gmVector3 v,
          mtkEvent type)
{

  if (part == MTK_BOX_VERTEX) {
    if (type == MTK_PICK) {
      mtk->getBoxCenter(&c);
```

```
      mtk->setSnapParallelAxis(part, p-c);
    }
    else if (type == MTK_MOVE) {
      mtk->getBoxCenter(&c);
      mtk->getBoxVertex(part, &vtx);
      factor=1.0+v[0]/(vtx[0]-c[0]);

      m=m.translation(-c[0], -c[1], -c[2]);
      m=m.scaling(factor, factor, factor)*m;
      m=m.translation(c[0], c[1], c[2])*m;

      mtk->applyBoxMatrix(m);

      updateModel(selected, c, gmVector3(0.0, 0.0, 0.0), factor);
    }
  }

  if (part == MTK_BOX_EDGE) {
    .
    .
    .
  }

  if (part == MTK_BOX_FACE) {
    .
    .
    .
  }
}
```

Basicamente, devem ser tomadas duas providências. Primeiro, é preciso identificar qual parte do interador foi selecionada, em nosso caso um vértice, uma aresta ou uma face da caixa. Para cada parte do interador, devem ser seguidos os próximo passos.

Se o evento for do tipo botão pressionado (`MTK_PICK`), é criada uma restrição para que o movimento da mudança de escala seja feito apenas no eixo que passa pelo centro da caixa e pelo vértice selecionado. da caixa. A restrição `setSnapParallelAxis` recebe a informação sobre o

eixo de movimento, e a restrição torna-se ativa.

Em seguida, é preciso dar uma interpretação para as informações recebidas, no caso interpretar o ponto 3D e o vetor de deslocamento 3D como uma mudança de escala. De acordo com a equação 5.3, este fator de escala é aplicado sobre a geometria do *bounding box* pelo método `applyBoxMatrix`, que recebe uma matriz homogênea 4x4 contendo a transformação desejada.

Finalmente, o modelo geométrico da aplicação também pode ser atualizado com esta transformação. No trecho de código anterior, a função `updateModel` é responsável por esta tarefa. Esta função, faz parte da aplicação, e não da biblioteca. Note que esta função foi inserida dentro da função *callback* registrada no interador. Desta maneira, sempre que o interador for invocado, poderá refletir as mudanças no modelo da aplicação.

5.1.2 Jack

O interador *jack* foi utilizado para implementar as operações de translação e rotação e mudança de escala. No modelador geométrico, a mudança de escala não foi aplicada, já que não há sentido em aumentar ou diminuir um vértice, que na verdade representa as coordenadas x, y, z de um ponto (Veja o diagrama de transição de eventos na Figura 5.3).

A translação foi implementada utilizando a restrição `setSnapNearestAxis` (Figura 5.4), associada ao vértice central do *jack*. Esta restrição aguarda por um movimento do *mouse* para definir a sua direção, e só então limita o movimento na direção do eixo projetado mais próximo ao vetor de movimento do cursor. Os sucessivos deslocamentos na direção do eixo escolhido são interpretados como translações, e aplicados sobre o modelo da aplicação e sobre o próprio *jack*.

Durante a rotação, são usadas duas restrições. Primeiro, a restrição `setSnapNearest-ParallelAxis` determina em direção de qual eixo o movimento será limitado, como no exemplo acima. No entanto, se essa restrição fosse mantida, a rotação seria produzida não por movimentos circulares, mas por movimentos retos da direção de um eixo, o que não corresponde a "rodar" um objeto.

Portanto, após o cálculo da direção do eixo sobre o qual foi feito o movimento, pelo algoritmo do *jack* (Seção 5.3), determina-se a rotação como sendo feita ao redor do terceiro eixo remanescente. Determinando o plano perpendicular a este eixo, a restrição é alterada para movimentos paralelos a esse plano com `setSnapParallelPlane`. Movimento paralelos a este plano produzem uma rotação cujo ângulo é igual ao ângulo entre a linha formada pelo novo ponto 3D e o centro do *jack*, e a linha formada pelo ponto 3D anterior e o centro do *jack*.

A mudança de escala foi implementada com a restrição `setSnapParallelAxis`, associada aos eixos do *jack*. Deslocamentos na direção do eixo selecionado são interpretados como uma

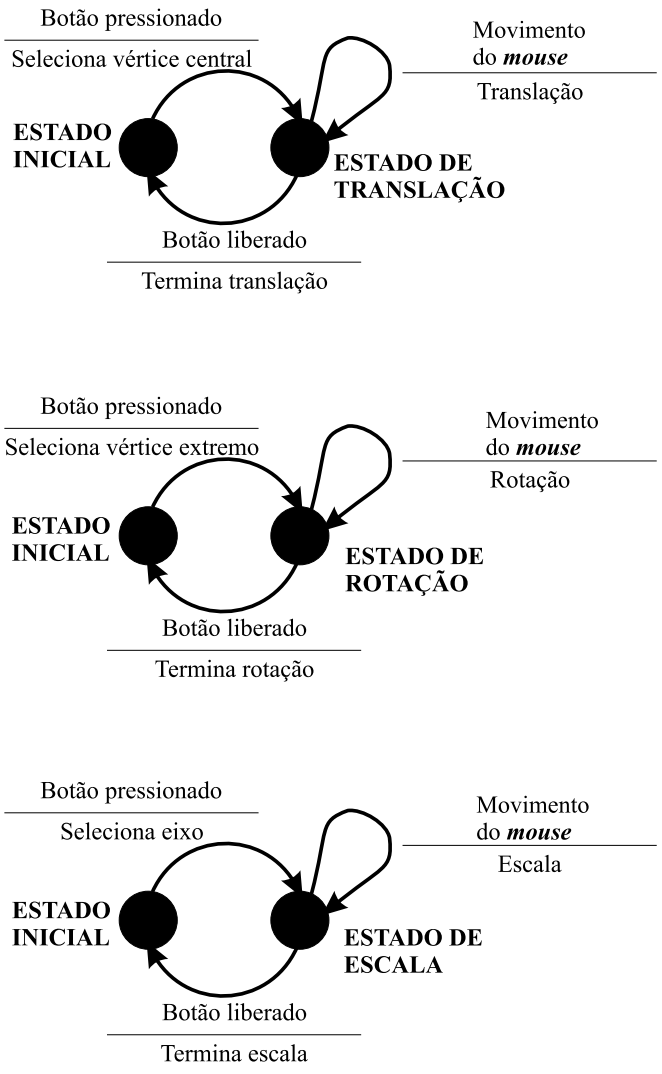


Figura 5.3: *Diagrama de transição de estados para as operações de translação, rotação e mudança de escala, utilizando o interador jack.*

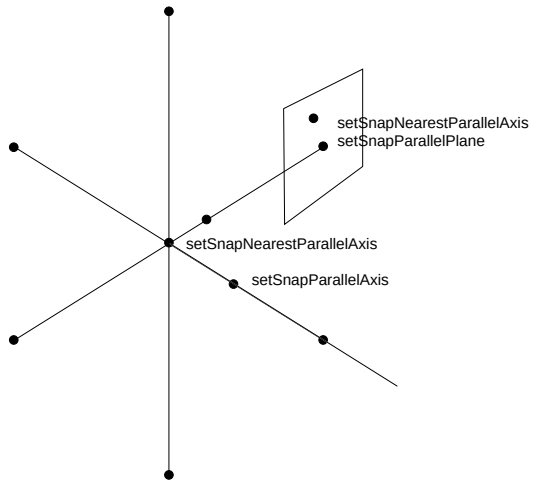


Figura 5.4: *Restrições utilizadas juntamente com o interador jack.*

mudança de escala. O fator de escala é igual a razão entre a distância do ponto 3D corrente ao centro do *jack* e a distância do novo ponto 3D ao centro do *jack*.

5.1.3 Trackball

A partir do interador *trackball*, foram implementadas três tipos de manipulações. Na Figura 5.5, vemos as restrições associadas a cada parte do interador para obter as operações de translação, rotação e mudança de escala.

A translação foi implementada utilizando a restrição `setSnapParallelAxis`, associada à linha que parte do *trackball*. Ao arrastar a linha, são permitidos apenas movimentos paralelos a ela. Estes deslocamentos então interpretados como translações pela aplicação.

A mudança de escala é feita de forma semelhante. A restrição `setSnapParallelAxis` é associada ao ponto de controle na extremidade da linha que parte esfera. Esta restrição limita os movimentos do interador à direção definida por aquela linha. Os deslocamentos são interpretados como uma mudança de escala, cujo fator é determinado pela razão entre o distância do ponto 3D corrente ao centro da esfera e a distância entre o novo ponto 3D e o centro da esfera.

Foram criados dois tipos de rotação. O primeiro, mais livre, associa a restrição `setSnapSphere` à esfera do *trackball*. Esta restrição retorna dois pontos 3D sobre a superfície da esfera: o ponto corrente e o novo ponto. Para interpretar esta informação como uma rotação, é preciso definir o ângulo e o eixo de rotação. O ângulo entre a linha do ponto 3D corrente ao centro da esfera e a linha entre o novo ponto 3D e o centro da esfera, é o ângulo de rotação. O eixo de rotação é determinado como o eixo perpendicular ao plano formado por estas duas linhas.

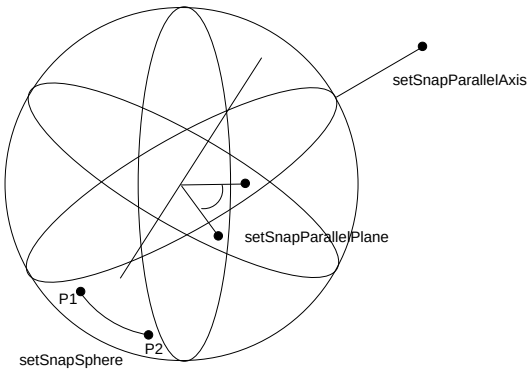


Figura 5.5: *Restrições utilizadas juntamente com o interador trackball.*

O segundo tipo de rotação, mais preciso, é restrito a um dos três círculos do *trackball*. Quando um dos círculos é selecionado, a restrição `setSnapParallelPlane` é associada a ele. O movimento é restrito ao plano que contém o círculo selecionado. O ângulo e o eixo de rotação podem ser determinados de modo similar à rotação livre.

A Figura 5.6 apresenta o diagrama de transição de estados para as operações de translação e mudança de escala, enquanto a Figura 5.7 apresenta as operações de rotação livre e restrita.

5.2 Testes

Durante a implementação das manipulações, foram feitos vários testes para se chegar aos resultados apresentado na seção anterior.

Durante os testes, o interador *bounding box* mostrou-se como o mais adequado para agregar várias operações, em função de sua geometria, que proporciona fácil acesso a suas diferentes partes. Pelos nossos experimentos, concluímos que é relativamente simples selecionar e arrastar seus vértices, arestas e faces. Desta maneira foi possível atribuir as operações de translação, rotação e mundaça de escala sem tornar complexo o uso do interador.

O interador *trackball* é ideal para rotações, pois a partir de sua forma esférica é possível intuir a tarefa de "rolar" a esfera. Além disso, as rotações na esfera e nos círculos, permitem alternar facilmente entre rotações livres e ajustes mais finos. A linha que parte da superfície da esfera e o ponto de controle em sua extremidade, apresentaram um problema de oclusão. Frequentemente, ao rolar a esfera, estas duas partes do interador ficam escondidas, inpossibilitanto o acesso a elas. O problema pode ser resolvido utilizando uma esfera totalmente transparente. Neste caso, entretanto, torna-se difícil identificar os limites da esfera, de forma a selecioná-la com o *mouse*.

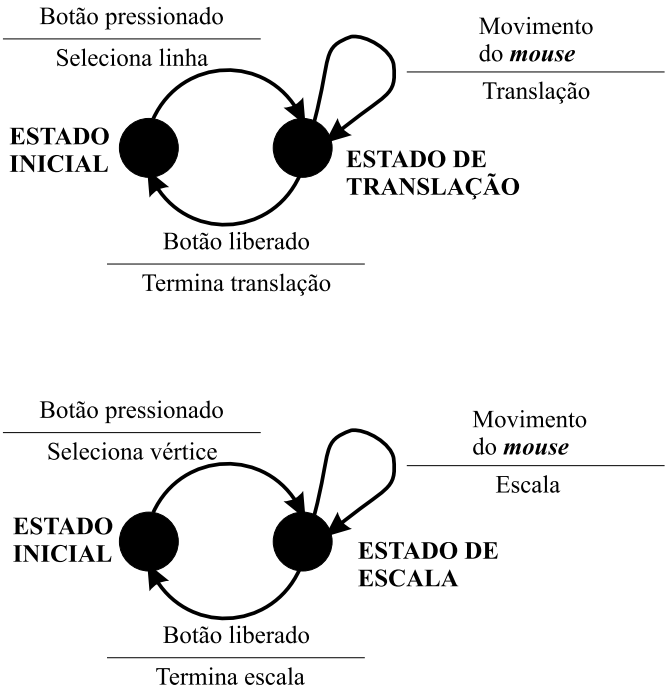


Figura 5.6: *Diagrama de transição de estados para as operações de translação e mudança de escala, utilizando o interador trackball.*

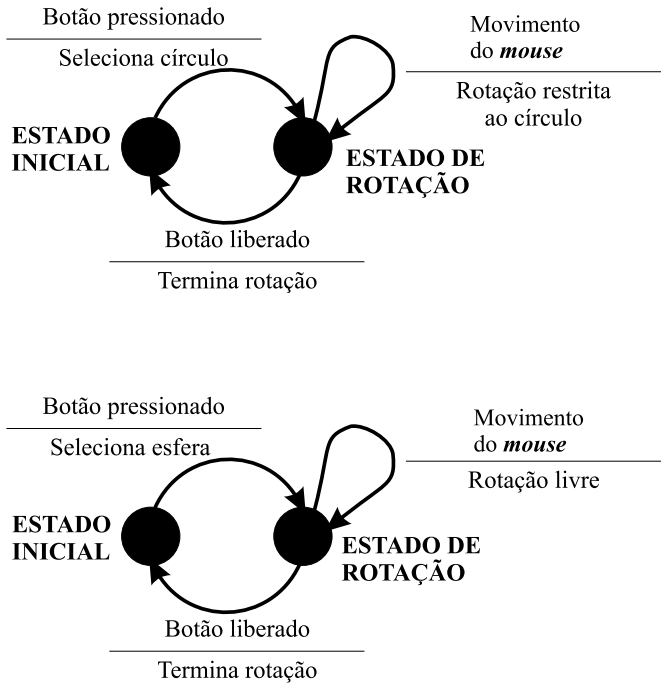


Figura 5.7: *Diagrama de transição de estados para as operações de rotação restrita e rotação livre, utilizando o interador trackball.*

O *jack* foi testado de duas maneiras. Na primeira, a operação de translação foi associada aos eixos do *jack*, e as operações de rotação e mudança de escala aos vértices. A rotação era feita conforme detalhado na Seção 5.1.2. A mudança de escala era obtida com movimentos paralelos ao eixo que continha o ponto de controle selecionado.

Esta abordagem é mais adequada para manipular objetos que obscurecem o vértice central, já que o mesmo não é utilizado. Por outro lado, a integração da rotação e mudança de escala nos vértices extremos tornou sua utilização difícil. Como as duas operações utilizavam a mesma parte do interador *jack*, muitas vezes o limiar entre os movimentos que produziam rotação e os movimentos que produziam escala era muito pequeno. Assim, uma tentativa de rotação podia resultar em mudança de escala, e vice-versa.

A segunda abordagem não é recomendada para ser usada com objetos que obscurecem o vértice central, uma vez que ele possui a função de translação. Sua vantagem é que as operações ficam mais distribuídas pelo interador. Nessa configuração, o *jack* é ideal para arrastar vértices de objetos ou pontos de controle, pois o ponto de controle central pode ser colocado sobre eles sem ficar obscurecido.

Nas Figuras 5.18 e 5.19, o interador *jack* é usado para mover um vértice e "torcer" um objeto, respectivamente. Embora o enfoque do nosso trabalho tenha sido prover mecanismos para construção de manipulações diretas voltadas para transformações lineares, os testes com o *jack* mostraram a versatilidade deste interador, que também pode ser utilizado para a deformação de objetos.

5.3 Interface para um Modelador Geométrico

Para validar a implementação de nossa arquitetura, foi desenvolvido um modelador geométrico de sólidos por instanciação. A parte da interface responsável pela visualização 3D e pela manipulação direta foi implementada com o MTK. Para os elementos de interface bidimensionais (menus e gerenciamento de janela), foi utilizada a biblioteca Motif[15]. O modelo geométrico da aplicação é gerenciado pela biblioteca TDM[32].

O modelador compreende os seguintes sólidos: cubo, esfera, cone, cilindro, toro e superfície de Bèzier (Figura 5.8). Os sólidos são criados através de um menu, e inseridos na cena na posição da origem (Figura 5.9).

Uma cena pode ser exportada ou importada no formato do sistema de *rendering* POV-Ray[29], juntamente com seus atributos de cor, material e informações sobre câmera e fontes de luz. Nas Figuras 5.10 e 5.11 são apresentados os diálogos utilizados para exportar e importar

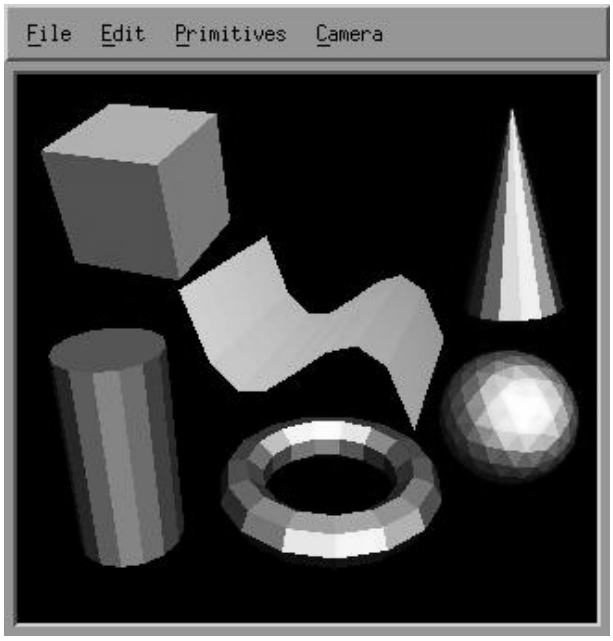


Figura 5.8: *Sólidos do modelador geométrico.*

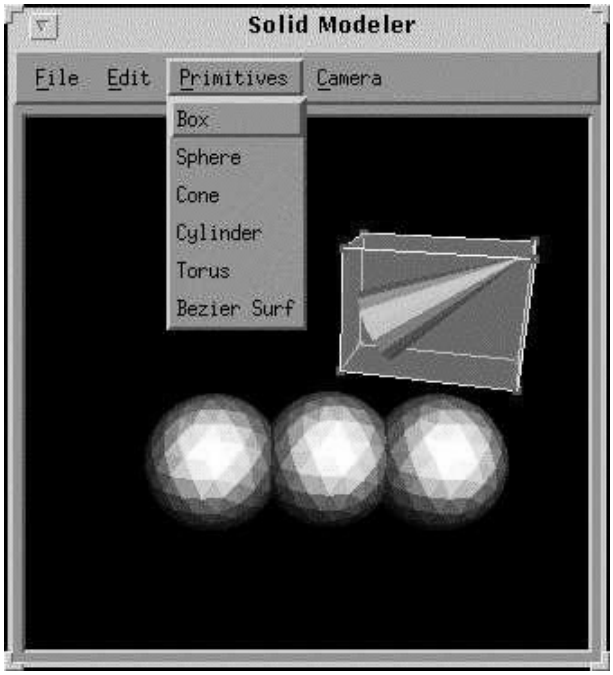


Figura 5.9: *Criação de novas instâncias de sólidos.*

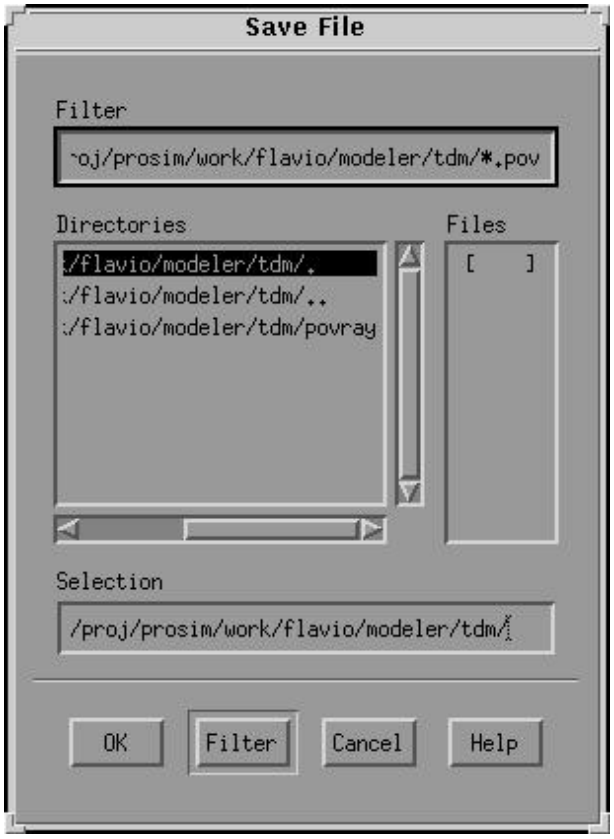


Figura 5.10: *Diálogo para salvar arquivos POVRay.*

arquivos no formato POV-Ray.

Os atributos de cor e material dos objetos da cena podem ser alterados através de um diálogo (Figura 5.12), bem como os atributos de câmeras (Figura 5.13) e fontes de luz (Figura 5.14). Outra funcionalidade disponível é a possibilidade de criar várias vistas da cena, através do suporte a múltiplas câmeras.

Após sua criação, os objetos podem ser selecionados para a execução de operações sobre eles. Há três operações de seleção disponíveis:

- `select all`: seleciona todos os objetos da cena.
- `select none`: anula a seleção de qualquer objeto selecionado.
- `invert`: inverte o estado de seleção de todos os objetos da cena. Se o objeto não estiver selecionado, passa a ser selecionado, e vice-versa.

Quando um ou mais objetos estão selecionados, é possível realizar três operações com eles: remover os objeto da cena, aplicar transformações diretas sobre os objetos (translação,

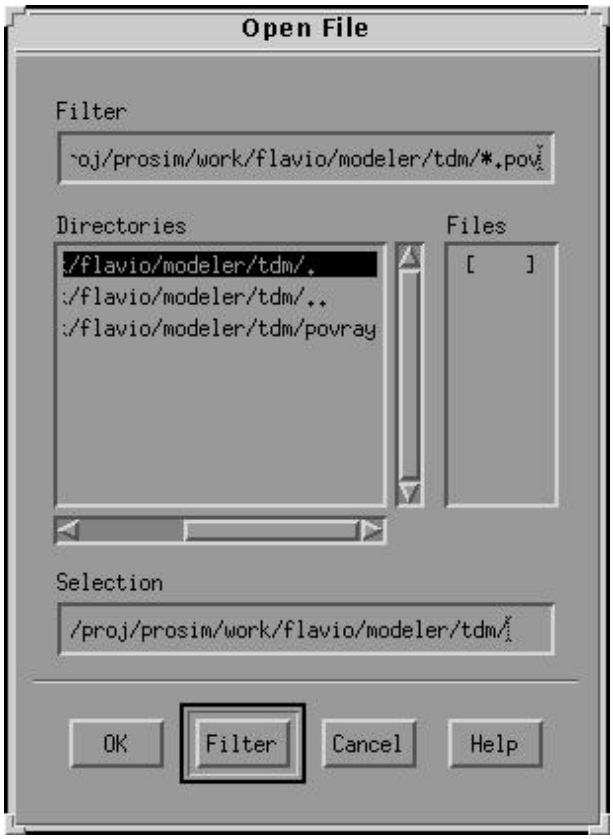


Figura 5.11: *Diálogo para abertura de arquivos POVRay.*

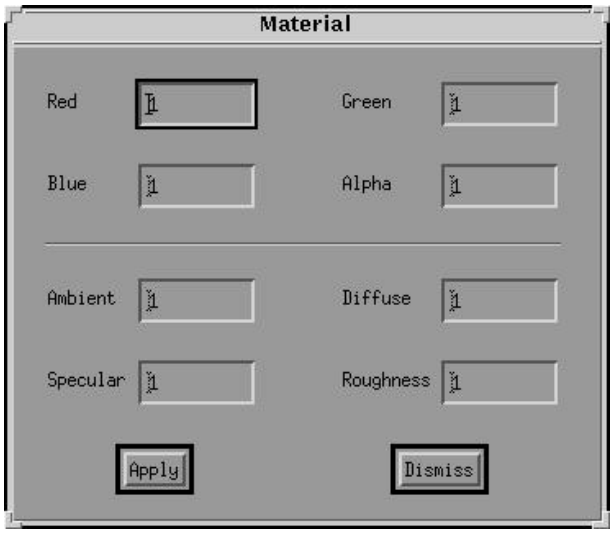


Figura 5.12: *Propriedades de material das primitivas.*

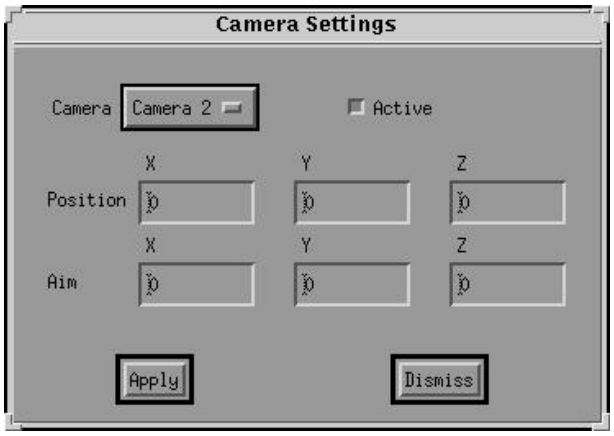


Figura 5.13: *Configurações de câmeras.*

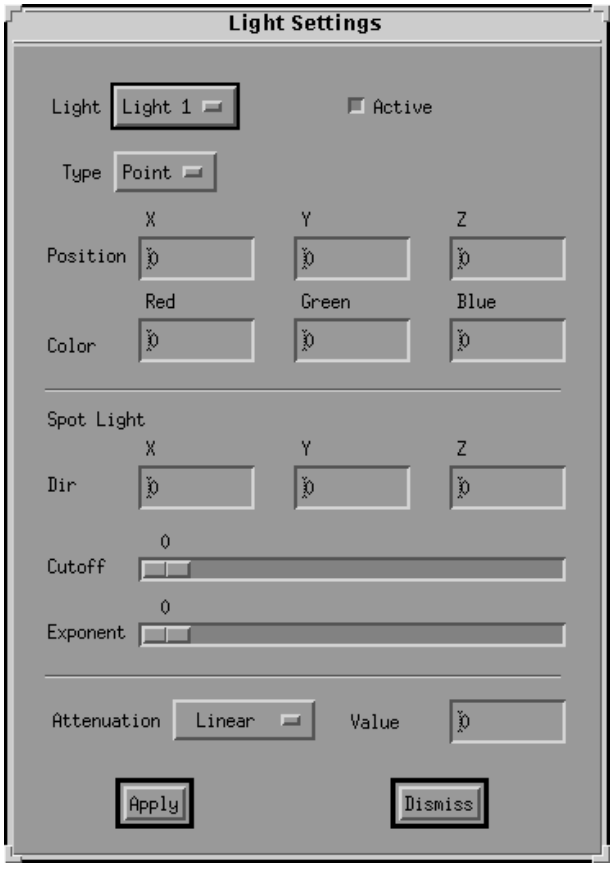


Figura 5.14: *Configurações de fontes de luz.*

rotação e mudança de escala), e editar um objeto localmente, movendo seus vértices individualmente ou em conjunto.

A operação de remoção apaga os objetos do modelo geométrico da aplicação e seus correspondentes enviados ao MTK.

No projeto do modelador geométrico, os interadores *bounding box* e *trackball* foram testados para translações, rotações e mundaças de escala (Seção 5.2). O *bounding box* foi escolhido, devido à sua facilidade em agregar várias operações. As transformações lineares são feitas através da manipulação direta do interador *bounding box*, da seguinte maneira:

Translação: a translação é feita clicando sobre qualquer face do *bounding box* e arrastando o *mouse* com o botão pressionado. O movimento de translação será feito sobre o plano que contém a face selecionada. Na Figura 5.16, a caixa é arrastada para a frente, através do arrasto da face superior, levando o cone a passar por dentro de um toro.

Rotação: a rotação é feita através do arrasto de qualquer aresta da caixa. Arrastos paralelos ao plano perpendicular à aresta produzem uma rotação do objeto em relação ao eixo que passa pelo centro da caixa e que é paralelo à aresta selecionada. O arrasto de uma aresta na Figura 5.15, produz uma rotação do cilindro e do toro selecionados.

Escala: a mundaça de escala é obtida através do arrasto de qualquer vértice da caixa. A escala é proporcional nos três eixos, e ocorre na direção do eixo que passa pelo vértice selecionado e pelo centro da caixa (Figura 5.17).

A edição local dos vértices dos objetos é feita com o interador *jack*. As operações podem ser feitas sobre um vértice individualmente ou sobre ou conjunto de vértices. Os vértices selecionados podem ser movidos ou rotacionados.

A translação de um ou mais vértices é feita em dois passos: primeiro, o ponto de controle no centro do *jack* é selecionado. Com o botão do *mouse* pressionado, é feito um movimento na direção de um dos eixos do *jack*. A translação passa a ser restrita à direção deste eixo (Figura 5.18).

A rotação de um ou mais vértices dos objetos é feita selecionando um dos seis pontos de controle nos extremos dos eixos do *jack*. Com o botão do *mouse* pressionado, movimentos paralelos à direção de um dos dois eixos que sobraram, produzem uma rotação ao redor do eixo remanescente. Por exemplo, selecionar um dos dois pontos de controle do eixo z, e arrastá-lo paralelamente ao eixo y, resulta em uma rotação ao redor de x. Na Figura 5.19, note que a rotação tem o efeito de "torcer" o objeto.

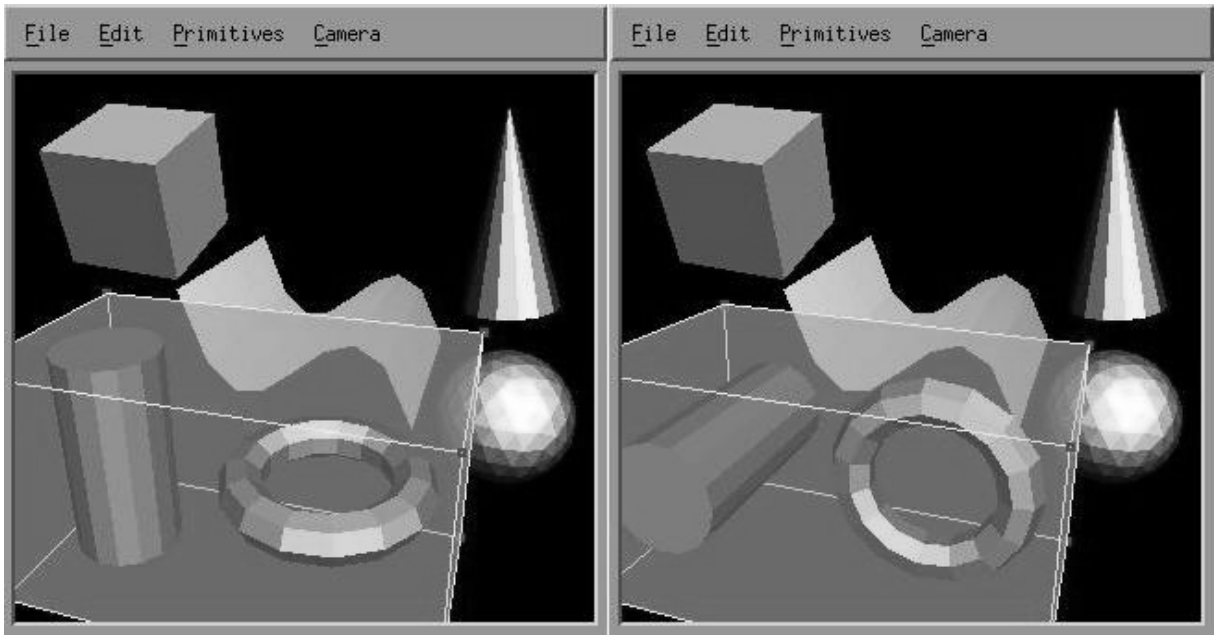


Figura 5.15: *Rotação de um cilindro e um toro.*

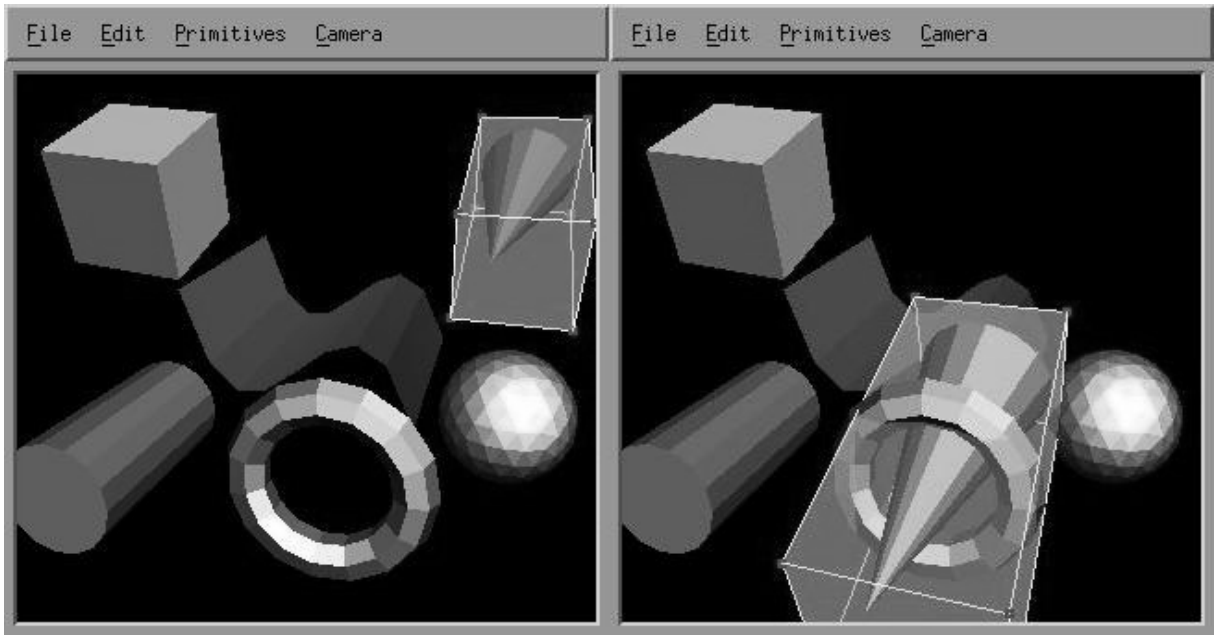


Figura 5.16: *Translação de um cone.*

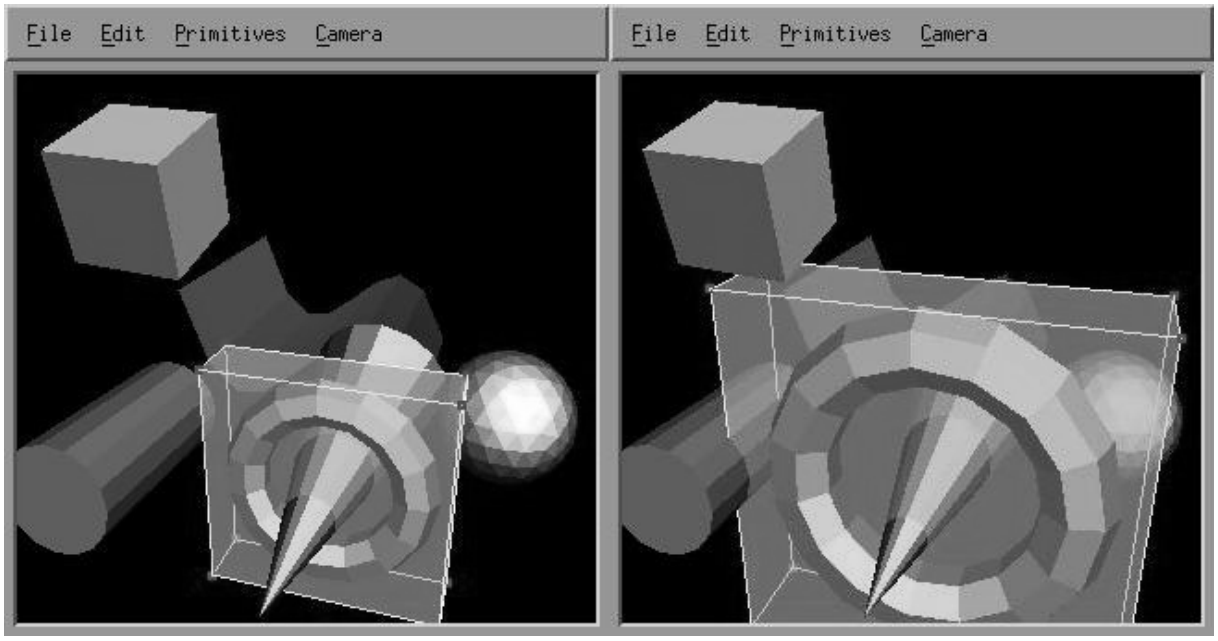


Figura 5.17: *Aumento do fator de escala de um toro.*

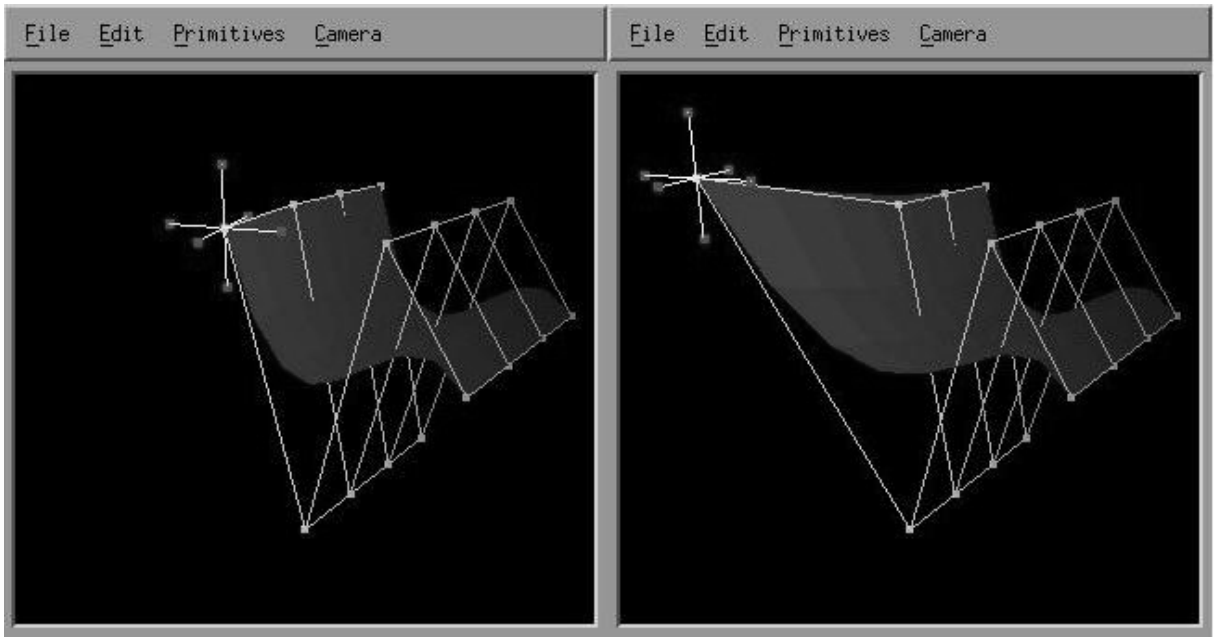


Figura 5.18: *Edição local do ponto de controle de uma superfície de Bèzier.*

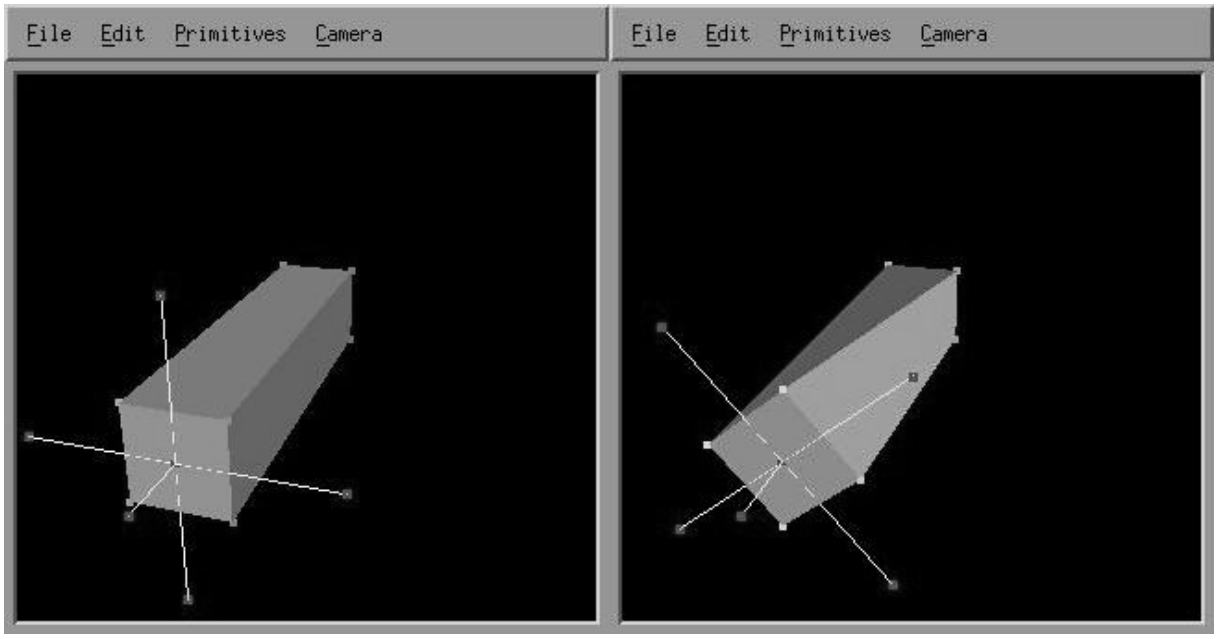


Figura 5.19: *Rotação de um jack, utilizado para "torcer" um objeto.*

Também foram utilizados os *Guias Visuais*, descritos na Seção 4.6, o Eixo 3D e a Grade 3D. Através da Grade 3D (Figura 5.20), por exemplo, é possível ter uma percepção mais realista da profundidade na cena. Os parâmetros do Eixo 3D e da Grade 3D podem ser configurados pelos diálogos apresentados nas Figuras 5.21 e 5.22 respectivamente.

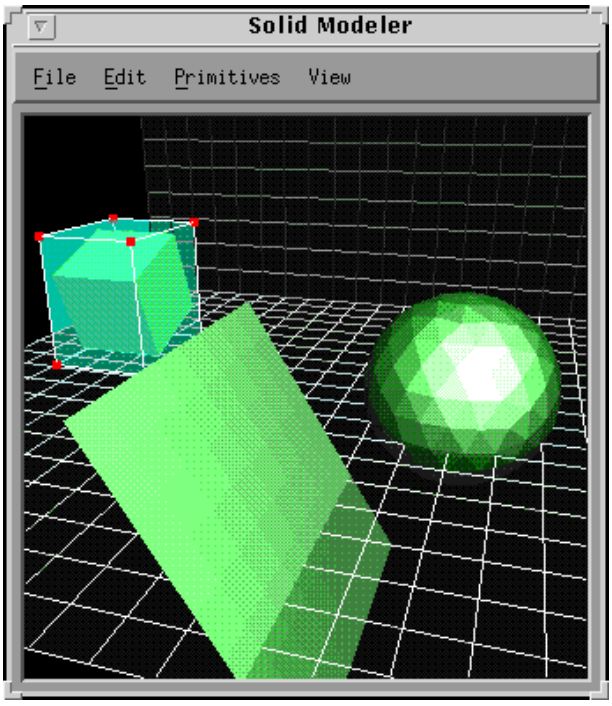


Figura 5.20: *Objetos e a grade 3D.*

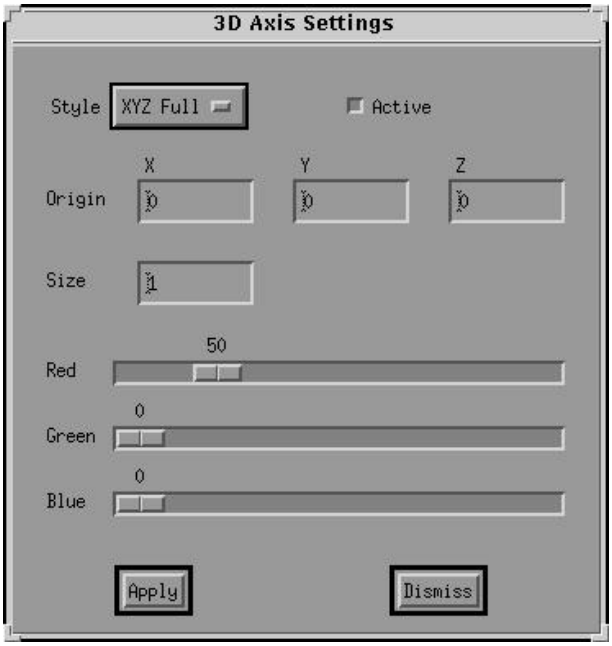


Figura 5.21: *Configurações do eixo 3D.*

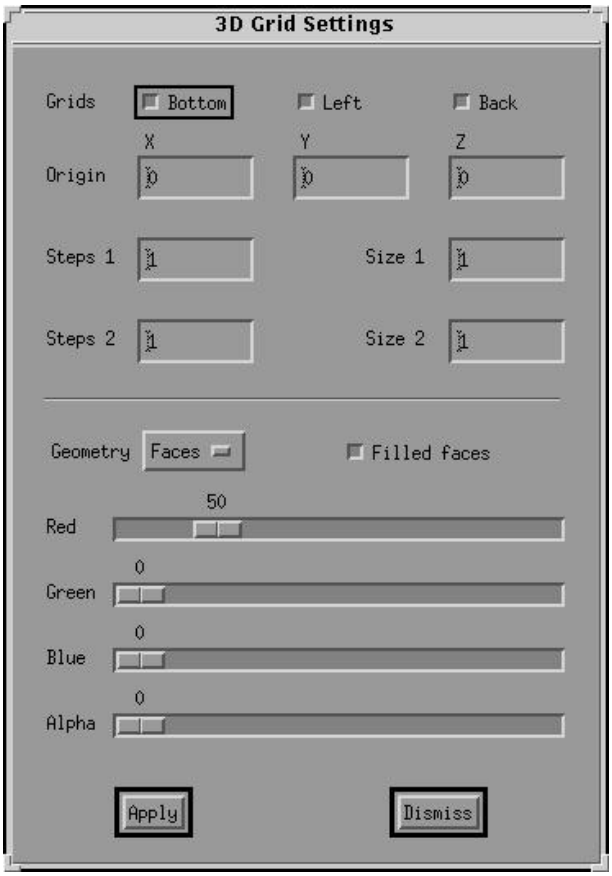


Figura 5.22: *Configurações da grade 3D.*

Capítulo 6

Conclusões e Trabalhos Futuros

Este trabalho teve como objetivo criar uma arquitetura para o desenvolvimento de interfaces gráficas 3D com manipulações diretas, utilizando dispositivos 2D.

Para tanto, a arquitetura foi proposta com as seguintes características:

- *Reusabilidade* das tarefas de interação construídas.
- *Flexibiblidade* para a construção de diferentes tarefas de interação.
- *Extensibilidade* para inserir novos objetos na arquitetura quando necessário.

A reusabilidade foi obtida através do desacoplamento das funções de visualização/manipulação das funções do modelo geométrico da aplicação. Outro fator importante foi a aplicação dos interadores para a manipulação, permitindo a manipulação de objetos 3D da aplicação, independentes de sua geometria.

A flexibilidade baseou-se em dois pontos:

- Definição de um conjunto de tarefas básicas de interação, livres de contexto, que pudessem ser aplicadas na construção de interações mais complexas. As tarefas básicas de seleção e posicionamento mostraram-se adequadas para tal objetivo.
- Utilização das restrições, que permitem associar diversos comportamentos diferentes às partes dos interadores.

A extensibilidade está baseada na separação entre os componentes da arquitetura, mais especificamente a separação entre a geometria da manipulação (interadores) e seu comportamento (restrições), permitindo criar novos interadores e novas restrições de forma independente.

A arquitetura foi implementada sob a forma de uma biblioteca de funcões chamada MTK. A partir do MTK, foram implementadas as operações de translação, rotação e mudança de escala. Estas operações foram criadas nos três interadores disponíveis na biblioteca, o *bounding box*, o *trackball* e o *jack*, através da correta associação de cada parte de um interador com uma restrição que levasse ao comportamento esperado. A biblioteca foi utilizada ainda para implementar a interface 3D de um modelador geométrico simples, com manipulação direta.

Como parte de outro trabalho, as manipulações apresentadas no Capítulo 5 foram encapsuladas em objetos chamados *manipuladores*[33]. O objetivo dos manipuladores é prover técnicas de manipulação direta, prontas para o uso, que possam ser facilmente reutilizadas em outras aplicações, com modelos geométricos distintos.

Enquanto a biblioteca MTK contribui para a reusabilidade das manipulações criadas, através do desacoplamento das funções de visualização/manipulação das funções do modelo geométrico (Seção 3.1.3), os manipuladores estendem esta reusabilidade um passo além, fornecendo técnicas de manipulação direta reutilizáveis.

A biblioteca MTK e os manipuladores estão sendo utilizados em outros dois projetos do Grupo PROSIM.

O primeiro projeto envolve o desenvolvimento de um modelador geométrico de objetos definidos por superfícies implícitas, mais conhecidos como *metaballs*. Além dos manipuladores atuais, foi desenvolvido outro manipulador para este projeto em particular. Este novo manipulador é baseado no interador *bounding box*, e é capaz de fazer rotações a partir de seus vértices, mudança de escala com as arestas e cisalhamento através das faces. O segundo projeto visa estender a arquitetura para um framework que suporte sistemas colaborativos com interações 3D.

Além deste modelador, o MTK e os manipuladores estão sendo aplicados em um modelador geométrico distribuído e um modelador geométrico para um sistema cooperativo.

Como pontos forte da biblioteca, podemos destacar:

- A reusabilidade alcançada, consequência do desacoplamento entre a interface e o modelo da aplicação. Em nossos experimentos, a biblioteca e as manipulações construídas foram reutilizadas com sucessos em dois modeladores geométricos. O primeiro, apresentado nesse trabalho, é baseado em sólidos simples. O segundo, parte do trabalho de outro membro do grupo PROSIM, é baseado em um modelador de objetos do tipo *metaball*.
- A flexibilidade apresentada pelo MTK durante a construção de diferentes manipulações. Foi possível testar, em pouco tempo, diferentes comportamentos em cada interador.

Para encerrar, sugerimos como trabalhos futuros:

- Implementar interadores mais simples, como pontos, linhas e planos (Seção 2.1.5), visando aumentar a flexibilidade na construção de novas manipulações.
- Implementar também interadores mais específicos, como interadores voltados para deformações[34].
- Implementar interadores para fontes de luz.
- Criar novos manipuladores, prontos para o reuso.
- Implementar novos tipos de restrições.

Apêndice A

Tecnologia em Hardware

A.1 Tecnologia em Hardware

Graças à popularização dos sistemas de realidade virtual em áreas como a manufatura de carros e aviões, desenvolvimento de novas substâncias na indústria farmacêutica, jogos e medicina, são cada vez mais populares os dispositivos de visualização e manipulação em 3D, devido ao seu alto grau de realismo durante a imersão em mundos virtuais. O custo relativamente alto destes dispositivos em conjunto com problemas ergonômicos e de precisão ainda impedem sua maior disseminação.

A.1.1 Dispositivos de Saída

Quando observamos objetos do mundo real, cada um de nossos olhos percebe a mesma imagem vista de uma perspectiva ligeiramente diferente. Através da fusão destas duas imagens bidimensionais, o cérebro consegue obter a informação de profundidade sobre objetos do ambiente, tornando possível a visão estéreo ou tridimensional. Vários produtos existentes no mercado geram a "ilusão" de enxergar em três dimensões, simulando esta característica de nossa visão.

O método aplicado por StereoGraphics[21] (Figura A.1) consiste em exibir uma imagem completa para o olho esquerdo, seguida de uma para o olho direto, em um monitor comum. Óculos especiais com lentes de cristal líquido que escurecem, operam em sincronia com as imagens exibidas no monitor. Estes óculos bloqueiam a visão do olho esquerdo para a imagem do olho direito, e vice-versa, alternando a uma velocidade de vários quadros por segundo, fazendo com que essa troca não seja notada.

Outro dispositivo comum, conhecido como *head-mounted*[13], é composto de uma espécie de capacete com dois pequenos monitores que geram imagens separadas para cada olho. Sen-

Figura A.1: *Óculos para visão estereoscópica.*

sores adaptados ao capacete captam o deslocamento da cabeça, permitindo alterar o ponto de observação da cena de acordo com os movimentos do usuário.

O uso de holografia elimina a necessidade de óculos, capacetes ou outros dispositivos especiais para a visualização de imagens em três dimensões. Uma holografia é uma imagem tridimensional gravada e lida a partir de filme fotográfico de alta resolução com o uso de raios laser. A imagem resultante tem a aparência de um sólido ou de uma cena 3D flutuando sobre o filme, podendo ser vista de vários ângulos diferentes com a sensação real de profundidade[1].

Um sistema holográfico desenvolvido pela empresa Voxel Inc.[7], permite criar imagens 3D do crânio de crianças com fechamento prematuro da caixa craniana. As imagens são usadas no planejamento da cirurgia, que envolve processos delicados. Composto de três partes, um processo que permite a gravação de várias fatias de uma imagem sólida, uma câmera para a gravação de imagens de tomografia computadorizada no filme, e um emissor de luz comum para a reconstrução do holograma, o sistema estava sendo testado em 1996 na Universidade da Califórnia.

Todavia, mesmo com a vantagem de gerar imagens verdadeiramente tridimensionais, a necessidade de gravação das imagens em filme através de laser torna o processo mais lento, além de adicionar um custo extra a dispositivos holográficos.

Como uma das tecnologias mais recentes, pesquisadores da Universidade de Stanford[8] estão desenvolvendo um dispositivo de saída que consiste em um cubo de vidro especial. Canhões de laser são direcionados em regiões específicas do cubo para criar voxels individuais de uma imagem 3D. Como a projeção é feita em cubo, a imagem pode ser observado de qualquer ângulo, e sem a necessidade de aparatos especiais para os olhos.

A principal vantagem dos dispositivos de visualização em 3D é a percepção de profun-

didade em um ambiente 3D de forma próxima à realidade, o que não é possível utilizando-se simplesmente telas planas de monitores comuns. Por outro lado, duas grandes desvantagens são seu alto custo, a fadiga muscular e visual provocada pelo uso prolongado de óculos e capacetes de realidade virtual, além de desconfortos como tontura e dores de cabeça.

A.1.2 Dispositivos de Entrada

Além do problema da percepção da profundidade durante a visualização de um ambiente tridimensional, existe ainda a questão de como realizar manipulações neste ambiente. Encontra-se em literatura uma série de dispositivos para manipulação direta em 3D, capazes de executar desde operações de posicionamento, como translação e rotação até deformações de objetos.

Ware e Jessome[42] descreveram um tipo de *mouse* chamado *bat*, utilizado no posicionamento em um ambiente 3D. O *bat* é um dispositivo com 6 graus de liberdade, 3 para translação e 3 para orientação, medidos em relação a um ponto de referência. Steed e Slater[35] avaliaram o *bat* para algumas tarefas de interação 3D, como navegação, translação e rotação. Liang e Green[23] apresentaram um sistema interativo de modelagem em 3D, no qual o *bat* é utilizado em conjunto com um segundo sensor de seis graus de liberdade, posicionado na cabeça do usuário. Os movimentos do *bat* são mapeados em relação ao segundo sensor, que atua como um ponto de referência da posição do usuário.

Venolia[41] apresentou uma interface que utiliza outro dispositivo para manipulação direta, o *roller mouse*. O *roller mouse* baseia-se no modelo convencional de *mouse*, o qual possui uma pequena bola localizada em sua parte inferior, para captar os movimentos bidimensionais no plano onde o mesmo repousa. A diferença entre o *mouse* comum e o *roller mouse* consiste em duas rodas, uma em cada lado do *roller mouse*, na parte frontal superior. Essas duas rodas podem ser giradas ao mesmo tempo em que o mouse é arrastado, acrescentando a terceira dimensão de movimento.

Outro *mouse* com três graus de liberdade, denominado Guaiá, foi proposto por Merkle e Scheer[25]. O dispositivo é similar ao *mouse* convencional, com uma esfera para captar movimentos sobre um plano. Nesta proposta, são utilizadas duas esferas, através das quais é medida, além do deslocamento, a rotação do corpo do dispositivo.

Sachs et al.[30] criaram um dispositivo com seis graus de liberdade, constituído de dois sensores. Os dois sensores permitem que o usuário posicione e oriente os objetos uns em relação aos outros. Um dos sensores é embutido em uma paleta um pouco maior que o tamanho de uma mão. Esta paleta, normalmente segurada pelo usuário por uma das mãos, corresponde a um plano que pode ser movido no mundo virtual. Objetos criados no mundo virtual são automati-

Figura A.2: *Luva virtual Cybergrasp*.

camente ligados a esse plano, e consequentemente, movem-se de acordo com os movimentos da paleta. O outro sensor localiza-se em uma espécie de caneta com funções configuráveis, como por exemplo desenhar ou selecionar objetos. Esta caneta é manipulada pela outra mão. O uso simultâneo dos sensores tira vantagem de uma habilidade inata de todos nós: a consciência de onde as duas mãos estão localizadas uma em relação à outra.

Em [13] é apresentado o *spaceball*, uma esfera rígida que contém sensores para captar a pressão e a direção dos movimentos da mão. Esta esfera é montada sobre uma base de plástico, que serve também como repouso para a mão ou o braço do usuário. O dispositivo consegue fornecer posição e a orientação em 3D quando a esfera é pressionada ou puxada em diferentes direções.

A *data glove*[13] (Figura A.2) reconhece a posição e a orientação das mãos e dos dedos. É composta de uma luva coberta com pequenos sensores que medem a flexão dos dedos e a posição das mãos. Através de uma representação gráfica da luva, inserida no ambiente 3D, é possível "pegar" objetos diretamente com as mãos, movê-los, rotacioná-los e então soltá-los.

Existem ainda esforços de construir dispositivos para realizar deformações, como o "cubo deformador" proposto por Murakami e Nakajima[26]. O dispositivo consiste em um cubo feito de material elástico, condutor de corrente, cuja representação gráfica na tela é uma caixa em forma de malha deformadora. Para deformar um objeto 3D, a representação gráfica deste objeto deve ser envolta pela malha deformadora. Todas as deformações aplicadas ao cubo são refletidas nesta malha e, consequentemente, no objeto envolvido por ela. Assim, o cubo pode ser pressionado, dobrado e torcido, deformado de várias maneiras, por duas mãos, resultando em deformações nos objetos de interesse.

Os dispositivos apresentados proporcionam uma correspondência direta entre os movimentos do usuário e manipulações aplicadas no ambiente, resultando em manipulações bastante naturais e intuitivas. No entanto, seu custo ainda restringe a sua ampla difusão. Mecanismos como a *data glove* e o *bat* não oferecem apoio para os braços, tornando-os inadequados para trabalhos mais precisos ou de longa duração.

Referências Bibliográficas

[1] R. M. R. Bastos and A. Laschuk. Hologramas gerados por computador aplicados à visualização tridimensional em computação gráfica. In *Sibgrapi92*, pages 135–144, 1992.

[2] Eric A. Bier. Skitters and jacks: Interactive 3d positioning tools. In *Proceedings 1986 ACM Workshop on Interactive 3D Graphics*, pages 183–196, Chapel Hill, North Carolina, New York, outubro 1986.

[3] Eric A. Bier. Snap-dragging in three dimensions. In Rich Riesenfeld and Carlo Sequin, editors, *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, volume 24, pages 193–204, Snowbird, Utah, 25-28, março 1990.

[4] B. Castier, L. F. Martha, and M. Gattass. A taxonomy for interactive manipulation and visualization of 3d objects. In *Sibgrapi94*, pages 149–156, 1994.

[5] W. Celes and J. Corson-Rikert. Act: an easy-to-use and dinamically extensible 3d graphics library. In *Sibgrapi97*, pages 26–33, Campos de Jordão, 13–16, outubro 1997. SBC, IEEE Computer Society.

[6] Michael Chen and Joy Mountford. A study in interactive 3d rotation using 2d control devices. In *Proceedings of ACM Siggraph'88*, pages 121–129. Addison-Wesley, 1988.

[7] PennWell Publishing Company. Holograms are helping surgeons. *CGW - Computer Graphics World*, 19(11):20, novembro 1996.

[8] PennWell Publishing Company. Full-color display in a cube. *CGW - Computer Graphics World*, 20(3):19, março 1997.

[9] D. Ehmke, W. Hinderer, M. Kreiter, D. Krömker, T. Batz, P. Baumann, K.G Höft, D. Köhler, and H.P. Subel. *PRODIA und PRODAT, Dialog und Datenbankschnittstellen für Systemenentwurfswerkzeuge*. Springer Verlag, Heidelberg, 1990.

[10] M. van Emmerik. A direct manipulation technique for specifying 3D object transformations with a 2D input device. *Computer Graphics Forum*, (9):355–361, 1990.

[11] D. Brookshire Conner et al. Three-dimensional widgets. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, volume 25, pages 183–188, março 1992. http://www.cs.brown.edu/research/graphics/publications.html.

[12] Kenneth Evans B., Peter P. Tanner, and Marceli Wein. Tablet-based valuators that provide one, two, or three degrees of freedom. In *Proceedings of ACM Siggraph'81*, volume 15, pages 91–97. Addison-Wesley, agosto 1981.

[13] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, 2nd. edition, 1990.

[14] Lúcia Teresa Schalcher da Fonseca, Luciano Pereira dos Reis, and Luiz Fernando Martha. Uma arquitetura para construção de ferramentas de manipulação para visualização interativa de dados volumétricos. In *Sibgrapi97*, Campos de Jordão, 13–16, outubro 1997. SBC, IEEE Computer Society.

[15] Open Software Foundation. *OSF/Motif Style Guide.* Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[16] International standard information, processing systems - computer graphics - graphical kernel system for three dimensions (gks-3d) functional description. American National Standards Institute, New York, 1988. ISO Document Number 8805:1988(E).

[17] K. P. Herndon, A. van Dam, and M. Gleicher. Workshop on the challenges of 3d interaction. In *SIGCHI Bulletin*, volume 26, pages 1–9, outubro 1994.

[18] W. T. Hewitt. Phigs: Programmer's hierarchical interactive graphics system. *Computer Graphics Forum*, 3(4):299–300, dezembro 1984.

[19] Stephanie Houde. Interactive design of an interface for easy 3D direct manipulation. In *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems*, pages 135–142, maio 1992.

[20] Corel Inc. Coreldraw! http://www.core.com.

[21] StereoGraphics Inc. Stereographics. http://www.stereographics.com.

[22] Lutz Kettner. Theoretical foundations of 3d-metaphors. fevereiro 1994. ftp://ftp.ira.uka.de/pub/
uni-karlsruhe/papers/.

[23] J. Liang and M. Green. Geometric modeling using six degrees of freedom input devices. *3rd International Conference on CAD and Computer Graphics Proceedings*, pages 217–222, agosto 1993.

[24] Fritz Loseries. Functional interface of the IQL frame type of PRODIA/11. Technical report, IGD, FhG, 1990.

[25] Luis Ernesto Merkle and Sérgio Scheer. Guaiá: um mouse com três graus de liberdade. In *Sibgrapi92*, pages 187–192. SBC, 1992.

[26] T. Murakami and Naomasa N. Direct and intuitive device for 3-d shape deformation. In *CHI'94*, pages 465–470, abril 1994.

[27] Jackie Neider, Tom Davis, and Mason Woo. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, release 1.* Addison-Wesely, 1993.

[28] G. M. Nielson and D.R. Olsen Jr. Direct manipulation techniques of 3D objects using 2D locator devices. In *Proceedings 1986 Workshop on Interactive 3D Graphics*, pages 175–182, Chapel Hill, North Carolina, New York, outubro 1986.

[29] Persistence of Vision Development Team. Pov-ray. http://www.povray.org.

[30] E. Sachs, Andrew Roberts, and David Stoops. 3-draw: A tool for designing 3d shapes. *IEEE Computer Graphics & Applications*, pages 18–26, novembro 1991.

[31] William J. Schroeder, Kenneth M. Martin, and William E. Lorensen. The design and implementation of an object-oriented toolkit for 3d graphics and visualization. In *Visualization'96*, pages 93–100, novembro 1996.

[32] Wu Shin-Ting. Non-manifold data models: Implementational issues. In Hermès, editor, *Proceedings of the 11th International Conference on the CAD/CAM, Computer Graphics and Computer Aided Technologies, Paris*, volume 1, pages 37–56, 1992.

[33] Wu Shin-Ting, Malheiros, Marcelo de Gomensoro, and Flávio Navarro Fernandes. Componentes para contrução de interfaces gráficas 3d. Technical report, DCA, FEE, UNICAMP, 1998.

[34] S.S. Snibbe and et al. Using deformations to explore 3D widget design. *Computer Graphics*, 26(2):351–352, julho 1992. Proceedings of the ACM SIGGRAPH'92. *http://www.cs.brown.edu/research/graphics/publications.html.*

[35] A. Steed and Mel Slater. 3d interaction with the desktop bat. *Computer Graphics Forum*, 14(2):97–104, 1988.

[36] M.P. Stevens, R.C. Zeleznik, and J.F. Hughes. An architecture for an extensible 3d interface toolkit. In *Proceedings of the ACM UIST'94*, novembro 1994.

[37] Paul S. Strauss and Rikk Carey. An object-oriented 3d graphics toolkit. *Computer Graphics*, 26(2):341–349, julho 1992.

[38] Adobe Systems. Adobe photoshop. http://www.adobe.com.

[39] Délia Perla Patrícia Velásquez Alegre. Técnicas básicas para interações 3d através do mouse. Master's thesis, FEEC - Unicamp, Campinas, outubro 1995.

[40] Délia Perla Patrícia Velásquez Alegre and Shin Ting Wu. Canvas 3D: Um novo componente de interface no XView. *VI Sibgrapi - Comunicação*, novembro 1993.

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

[42] Colin Ware and Danny R. Jessome. Using the bat: A six-dimensional mouse for object placement. *IEEE Computer Graphics & Applications*, 8(6):65–70, novembro 1988.

[43] Josie Wernecke. *The Inventor Mentor: Programming Object-Oriented 3D Graphics with OpenInventor, release 2.* Addison-Wesley Publishing Company, 1994.

[44] Xfig - facility for interactive generation of figures under x11. ftp://ftp.x.org/contrib/applications /drawing_tools. Responsável pela manuntenção do software: Brian V. Smith.