UNIVERSIDADE FEDERAL DA PARAÍBA - UFPB

CENTRO DE CIÊNCIAS E TECNOLOGIA - CCT

COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA - COPIN

JOÃO-DE-BARRO : UMA PROPOSTA DE SISTEMA HÍBRIDO PARA MODELAGEM DE SÓLIDOS

Luiz Antônio Mendes Garcia

Campina Grande - PB

Dezembro - 1992

JOÃO-DE-BARRO : UMA PROPOSTA DE SISTEMA HÍBRIDO PARA MODELAGEM DE SÓLIDOS

Luiz Antônio Mendes Garcia

Dissertação apresentada ao curso de MESTRADO EM INFORMÁTICA da UNIVERSIDADE FEDERAL DA PARAÍBA, em cumprimento às exigências para obtenção do grau de mestre.

Área de Concentração : Ciência da Computação

PAULO ROBERTO CAMPOS DE ARAÚJO
Orientador

PEDRO SÉRGIO NICOLLETTI
Co-Orientador

Campina Grande - PB
Dezembro - 1992

G216j Garcia, Luiz Antonio Mendes.
Joao-de-barro : uma proposta de sistema hibrido para modelagem de solidos / Luiz Antonio Mendes Garcia. - Campina Grande, 1992.
140 f.

Dissertacao (Mestrado em Informatica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Modelagem e Simulacao de Sistema. 2. Sistema Joao-de-Barro. 3. Computacao Grafica. 4. Dissertacao - Informatica. I. Araujo, Paulo Roberto Campos de. II. Nicolletti, Pedro Sergio. III. Universidade Federal da Paraiba - Campina Grande (PB). IV. Título

CDU 004.414.23(043)

"JOAO-DE-BARRO: UMA PROPOSTA DE SISTEMA HIBRIDO PARA MODELAGEM DE SOLIDOS".

LUIZ ANTONIO MENDES GARCIA

DISSERTAÇAO APROVADA EM 18.12.1992

PROF. PAULO ROBERTO CAMPOS DE ARAÜJO, M.Sc.
- Presidente -

PROF. PEDRO SERGIO NICOLLETTI, M.Sc.
- Examinador -

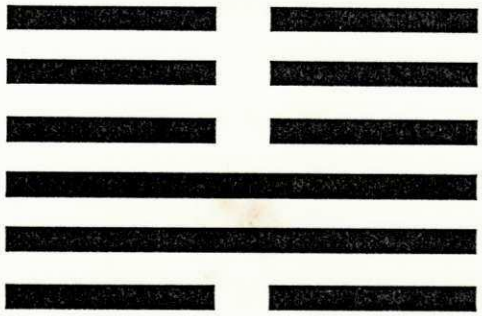
PROFA MARIA DE FATIMA Q. V. TURNELL, Ph.D.
- Componente da Banca -

PROF. ARNALDO DE ALBUQUERQUE ARAÜJO, Dr.
- Componente da Banca -

Campina Grande, 18 de dezembro de 1992

Este trabalho é dedicado aos meus pais, por todo o afeto, dedicação, e pela enorme ajuda sempre prestada. A eles, o meu maior carinho e admiração.

Luiz Antônio



A ASCENSÃO

"A imagem da árvore crescer dentro da terra sugere a idéia da ASCENSÃO.

Ao contrário do PROGRESSO FÁCIL, onde o sol, ao ascender sobre a terra, indica uma expansão fácil, a ASCENSÃO está associada ao esforço, do mesmo modo como a planta necessita de energia para irromper da terra.

O que aqui está indicado é uma ascensão vertical realizada pela força de vontade, mas que está em harmonia com o momento, partindo do anonimato e de uma condição inferior rumo ao poder e à influência. A ascensão começa por baixo. O impulso ascendente não retrocede.

A suavidade é interior; a devoção é exterior. O que possibilita a ascensão não é a violência, mas a modéstia, a adaptabilidade.

Graças às condições propícias do momento, o homem avança. Ele não deve temer o encontro com pessoas autoritárias, pois o êxito está assegurado pelo tempo.

A causa do sucesso não é social, mas trascendente. O aspecto favorável das circunstâncias vem de esferas invisíveis. É preciso aproveitar este momento ao máximo, enquanto pudermos, através do trabalho".

Extraído da obra "I CHING - O Livro das Mutações"

Sumário

RESUMO

Este texto é parte de um trabalho de pesquisa destinado à proposta de um sistema de modelagem de sólidos com representação híbrida. O sistema desenvolvido, denominado "Sistema João-de-Barro", combina as técnicas conhecidas como **Boundary Representation** e **Constructive Solid Geometry,** para a representação dos modelos construídos. Dentre os recursos de modelagem implementados, destacam-se as operações de transformação geométrica, opções de posicionamento do observador e operações booleanas de união, interseção e subtração. Para uma melhor apresentação de resultados, o sistema inclui recursos de visualização, tais como eliminação de faces ocultas, exibição em perspectiva e iluminação.

ABSTRACT

This text is part of a research work which proposes a solid modeling system with hybrid architecture. The system, so called "Sistema João-de-Barro", combines the techniques known as **Boundary Representation** and **Constructive Solid Geometry** for representing the models. Among the modeling tools implemented we can mention geometrical transformations, viewer positioning and the boolean set operations of union, intersection and difference. The system also implements visualization techniques like hidden-surface elimination, perspective projection and ilumination.

Capítulo 1

Introdução

1. INTRODUÇÃO

A área de Computação Gráfica tem assumido um papel cada vez mais presente no cotidiano das pessoas. Hoje, é bastante comum que indivíduos das mais diversas classes sociais e de vários ramos profissionais tenham algum tipo de contato, direto ou indireto, com esta área da Ciência da Informática. Seja pelo uso científico através de programas de simulação gráfica, pelo uso profissional, como programas de engenharia e arquitetura, ou mesmo através de vinhetas e propagandas exibidas em televisão e outros meios de comunicação, cresce a cada dia o número de pessoas que consomem Computação Gráfica.

A Computação Gráfica é uma área em plena evolução. Desde o aparecimento dos primeiros dispositivos gráficos em meados dos anos 50, a cada dia surgem novas aplicações e novos recursos. Existem, atualmente, métodos, técnicas, algoritmos, equipamentos e sistemas bastante sofisticados, com crescente potencialidade e variedade de aplicações.

O estado atual da tecnologia fornece métodos, técnicas e ferramentas para diversas tarefas e efeitos visuais através de Computação Gráfica. Alguns desses métodos e técnicas podem ser classificados de acordo com os itens e subitens abaixo :

A. Modelagem
  A.1. Modelagem Geométrica
    A.1.1. Modelagem de Sólidos
    A.1.2. Modelagem de Curvas e Superfícies
  A.2. Modelagem Procedural

B. Síntese de Imagens ("RENDERING")
  B.1. Exibição com Modelos de Linhas
  B.2. Exibição com Imagens Realísticas

Vale ressaltar que nem todas as técnicas da Computação Gráfica estão incluídas nos tópicos acima, além do fato de reconhecer-se que essa classificação é um tanto quanto geral e abrangente, passível de imprecisões e ambiguidades. Existem, na literatura, algumas divergências nesse sentido. No entanto, entende-se que esta classificação possa ser utilizada para se ter um panorama geral dos tópicos relevantes a este trabalho.

A área de **Modelagem** engloba técnicas para a criação e manipulação de modelos matemáticos que representem objetos, substâncias, seres vivos, fenômenos naturais, dentre outros elementos, sejam eles fictícios ou existentes no mundo real. Dentre os vários tipos de modelagem, destaca-se a área de **Modelagem Geométrica**, através da qual criam-se modelos poliédricos, objetos sólidos, superfícies curvas etc. Dentro dessa área, existem ainda outras subdivisões, como

a Modelagem de Sólidos, por exemplo, dedicada à elaboração de modelos que representem objetos sólidos presentes no mundo físico.

A Modelagem de Sólidos constitui a área-alvo do desenvolvimento deste trabalho. Os procedimentos inseridos nesta área envolvem a criação de modelos que representem objetos sólidos tridimensionais do mundo físico. Este texto inclui abordagens e definições formais sobre modelos sólidos. Um dos aspectos mais fundamentais de um sistema de modelagem de sólidos consiste na sua forma de representar os objetos. Atualmente, os chamados Sistemas Híbridos de Modelagem, assim denominados por incluírem mais de uma técnica de representação de modelos, são apontados pela literatura como o estado da arte em sistemas de modelagem de sólidos.

A área de Síntese de Imagens inclui técnicas para empregar efeitos visuais que tornem os modelos criados mais reais. A fidelidade visual é a palavra-chave dos estudos de síntese de imagens. Dentre os efeitos mais utilizados, encontram-se a simulação de iluminação, reflexividade, transparência, textura, entre outros.

As várias técnicas de Computação Gráfica podem ser utilizadas de diversas formas, em diversos tipos de aplicação. Segundo [Gome90], as aplicações da Computação Gráfica podem ser classificadas em três grandes áreas :

A. Projeto e Produção Auxiliados por Computador (CAD/CAM)
B. Visualização de Dados
C. Interação Homem-Máquina

Devido à sua abrangência e variedade de aplicações, pode-se, ainda, subdividir a área de Visualização de Dados em duas subáreas : Visualização Científica e Visualização Artística. Com essa pequena modificação, chega-se à seguinte classificação :

A. Projeto e Produção Auxiliados por Computador (CAD/CAM)
B. Visualização de Dados
    B.1. Visualização Científica
    B.2. Visualização Artística
C. Interação Homem-Máquina

Dentre os tipos de aplicação acima citados, podemos classificar o presente trabalho como pertencente à área de **Visualização Artística**.

Os trabalhos desenvolvidos em Visualização Artística visam a aplicações tais como entretenimento, publicidade, artes plásticas, entre outras. A indústria de entretenimento tem utilizado a Computação Gráfica em vídeo-games, cinema, televisão, parques de diversões, etc. As áreas de marketing e propaganda, fazem extenso uso de sistemas de síntese de imagens

tridimensionais e programas de animação. Esses sistemas representam um recurso valioso para a criação de cenas estáticas reais ou fictícias, para a criação de cenas animadas como as utilizadas em vinhetas de televisão, entre outras aplicações. Nas ártes plásticas, alguns artistas têm criado suas obras utilizando computadores no lugar de telas, pincéis e tintas. Esses artistas chamam essa nova técnica de "infoestética".

Os estudos envolvidos com a área de Visualização Artística têm dado origem a vários produtos e ferramentas gráficas. Os chamados **Sistemas de Visualização** consistem numa das aplicações mais atraentes. Esses sistemas oferecem recursos para a criação, manipulação e exibição de cenas sintéticas tridimensionais, sejam elas representações de cenas reais ou criações totalmente fictícias. Através dos Sistemas de Visualização, uma determinada cena idealizada pode ser construída e visualizada em computador. Devido ao caráter da aplicação, cenas como essa seriam criadas com objetivos estéticos, voltados para aplicações de cunho fotográfico. Esta mesma tarefa, executada por métodos tradicionais, demandaria maior tempo de trabalho e consumiria mais recursos humanos e materiais. Quando dotados de recursos de animação, os Sistemas de Visualização ficam ainda mais completos.

Um Sistema de Visualização aplica vários dos métodos e técnicas desenvolvidos para a Computação Gráfica. É necessária, por exemplo, a presença de procedimentos de modelagem, para que objetos diversos possam ser definidos e construídos de forma ágil e flexível. É desejável, também, que um Sistema de Visualização ofereça recursos de síntese de imagens para conferir maior realismo visual às cenas construídas.

Em termos bastante básicos, pode-se dizer que o processo de construção de uma cena em computador, utilizando-se um Sistema de Visualização, divide-se em duas etapas :

A) Construção da cena idealizada : esta fase é também chamada de modelagem da cena. Nesta etapa, os objetos (modelos) que farão parte da cena são construídos e posicionados no espaço abstrato da mesma. Para isso, um Sistema de Visualização deve oferecer funções e ferramentas que permitam ao usuário a construção eficaz e flexível dos modelos desejados.

B) Visualização da cena construída : nesta fase, os modelos sofrem ajustes, para que possam ser exibidos na tela do computador. Além disso, aplicam-se técnicas de síntese de imagens que simulam efeitos diversos, tais como iluminação e textura, aumentando o realismo visual da cena.

A figura 1.1 ilustra as etapas do processo de construção de um determinado modelo num Sistema de Visualização.

Motivado pelo desenvolvimento de projetos em torno de Sistemas de Visualização, este trabalho de dissertação tem como foco principal o estudo de técnicas de Modelagem de Sólidos aplicadas a sistemas de síntese de imagens. Dentre os estudos apresentados, propõe-se

uma arquitetura de representação híbrida, como um mecanismo de se obterem modelos complexos a partir de um sistema amigável e flexível. Como resultado paralelo, desenvolveu-se uma interface homem-máquina básica que prevê o acréscimo de novas funções de modelagem e visualização.

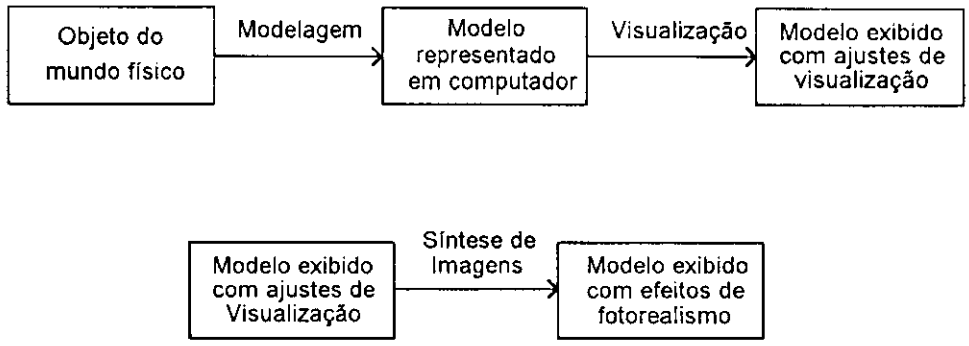


Figura 1.1 : Etapas da construção de um modelo para visualização.

Os estudos realizados envolvem o projeto e desenvolvimento do sistema intitulado "**João-de-Barro : Sistema Híbrido de Modelagem de Sólidos**". O nome do sistema faz uma homenagem à ave conhecida no Brasil, que, analogamente ao processo de modelagem de sólidos, constrói gradativamente, modelos em argila.

Este sistema consiste no que chamamos de um "sistema de modelagem mínimo para síntese de imagens", que oferece funções básicas de modelagem para um Sistema de Visualização. Nele são incluídas ferramentas diversas de modelagem de sólidos, tais como transformações geométricas, posicionamento do observador, operações booleanas de união, interseção e subtração, fusão de objetos, entre outras. Algumas das operações implementadas, tais como translações, rotações, escalas e posicionamentos do observador, podem ser executadas interativamente, visando ao maior conforto para o usuário. O sistema inclui, também, alguns recursos de síntese de imagens para uma melhor apresentação de resultados.

Ao longo deste texto, abordamos vários aspectos pertinentes ao desenvolvimento do sistema João-de-Barro. O capítulo 2 é dedicado ao estudo teórico de técnicas utilizadas em modelagem de sólidos. Este estudo inclui dados como aplicações, métodos de representação, técnicas de transformações de modelos, bem como outras questões críticas referentes a sistemas de modelagem. O capítulo 3 dedica-se à descrição funcional do sistema desenvolvido. Nele, o Sistema João-de-Barro é descrito através de seus módulos e suas funções, apresentado na forma de um simplificado manual do usuário. No capítulo 4, são apresentados aspectos da arquitetura

interna do sistema, tais como estruturas de dados, técnicas empregadas, métodos de representação, principais algoritmos utilizados, entre outros tópicos. O capítulo 5 é totalmente dedicado ao desenvolvimento das operações booleanas, por consistirem num aspecto fundamental para o sistema, além de serem de complexa implementação. No capítulo 6, dedicado à conclusão do trabalho, são feitas análises dos resultados obtidos e são apresentadas algumas sugestões para a continuidade do trabalho. O capítulo 7 inclui, finalmente, a bibliografia utilizada nos estudos desenvolvidos. Visando a uma maior abrangência bibliográfica, o capítulo 7 é organizado em três seções distintas : o item 7.1 relaciona trabalhos diretamente citados neste texto e fontes principais utilizadas para embasamento. Já no item 7.2, são listados aqueles trabalhos não citados no texto mas utilizados como importantes fontes de consulta. Finalmente, no item 7.3, é apresentado um levantamento bibliográfico contendo uma série de referências, visando complementar a bibliografia.

Capítulo 2

Fundamentos em Modelagem de Sólidos

2. FUNDAMENTOS EM MODELAGEM DE SÓLIDOS

Este capítulo é destinado à exposição de alguns aspectos ligados à fundamentação teórica estudada ao longo do desenvolvimento do trabalho, sendo a sua leitura necessária para a perfeita compreensão deste texto. São abordadas definições, conceitos e formalismos, bem como a apresentação básica de técnicas e estudos envolvidos em modelagem de sólidos.

2.1. O que é Modelagem de Sólidos ?

Como passo inicial deste estudo, deve-se fazer uma boa caracterização do que vem a ser **modelagem de sólidos**, apresentando-se definições, delineando-se os objetivos da área e ressaltando-se os objetos de estudo utilizados.

No capítulo 1, verificamos que a área de modelagem de sólidos consiste numa subárea inserida nos estudos sobre **modelagem geométrica**. Segundo [Fole82], a área de modelagem geométrica lida com modelos dotados de informações e propriedades geométricas, o que os torna adequados para representação gráfica. Deve ficar claro, portanto, que os estudos de modelagem de sólidos lidam com uma classe especial de modelos geométricos denominados **modelos sólidos**.

Para uma boa definição dos estudos e objetivos da área, torna-se bastante pertinente a análise semântica do termo "modelagem de sólidos". De antemão, a palavra "modelagem" pode ser interpretada como a construção de modelos. Por sua vez, o termo "sólido" sugere a imagem de corpos físicos. Os parágrafos seguintes apresentam uma abordagem mais detalhada.

Existem, na literatura, diversas definições que podem ser utilizadas para a compreensão do que vem a ser um **modelo**. Segundo [Ferr86], o termo "modelar" pode ser interpretado como "(...) representar por meio de modelo". A palavra "representar" traduz muito bem o papel de um modelo qualquer, que nada mais é do que um mapeamento visando à representação de uma determinada entidade. Um modelo é uma ferramenta bastante genérica, não sendo necessariamente construído em computador. Mesmo os modelos computadorizados podem assumir características bastante distintas entre si. Deve-se, portanto, se restringir ao escopo deste trabalho e empregar o termo "modelo" para referir-se a modelos geométricos computadorizados.

Numa abordagem já voltada para a área de Computação, [Fole82] afirma que modelos são usados para representar entidades físicas ou abstratas, além de fenômenos envolvendo essas entidades. O mesmo autor completa, ainda, que um modelo consiste numa rica

descrição dos componentes e dos processos que, em conjunto, especificam tanto a estrutura quanto o comportamento da entidade modelada. Outra definição importante nesse sentido é apresentada por [Gome90], afirmando que "(...) o objetivo da Computação Gráfica é criar, visualizar e manipular modelos no computador".

Às etapas e processos envolvendo a criação, representação e manipulação de modelos no computador, dá-se o nome de **Modelagem**.

Um **corpo sólido** é um tipo de entidade bastante presente no cotidiano das pessoas, fato que facilita a compreensão do termo "sólido". A primeira idéia que vem à mente é que um sólido consiste num objeto qualquer do mundo físico real. Segundo [Ferr86], um sólido pode ser definido como "corpo que tem três dimensões e é limitado por superfícies fechadas". Os corpos sólidos (ou sólidos, simplificadamente) do mundo físico real podem ser representados através de modelos computadorizados. Esses modelos são chamados de **modelos sólidos**.

Ao longo deste trabalho, os termos "sólido" e "modelo sólido" serão empregados com o mesmo significado.

A construção de modelos sólidos em computador apresenta algumas questões críticas. Posteriormente, apresentaremos algumas dessas questões e outros aspectos formais sobre tais modelos.

A partir das considerações apresentadas anteriormente, é possível definir **modelagem de sólidos** como o processo de criação, representação, e manipulação de modelos matemáticos que representem objetos sólidos do mundo físico. Definições semelhantes são propostas por [Gome90] e [Mill89].

Os estudos de modelagem de sólidos deram origem a vários produtos e ferramentas. Sistemas computacionais que lidam com processos e funções de modelagem de sólidos são conhecidos como **Sistemas de Modelagem de Sólidos (SMS)**.

[Gome90] divide o processo de modelagem de sólidos em três níveis lógicos (ver figura 2.1) :

- **Universo Físico** : É o mundo físico em que vivemos e que contém os objetos que desejamos modelar, ou seja, representar através de modelos.

- **Universo Abstrato** : Contém as idealizações dos modelos do mundo físico, ou seja, as abstrações de como serão construídos os modelos dos objetos pertencentes ao Universo Físico. Essas abstrações podem ser descritas através de modelos matemáticos.

- **Representação** : É a consumação da criação do modelo. Consiste na construção de estruturas simbólicas que representam o modelo matemático idealizado no universo abstrato.

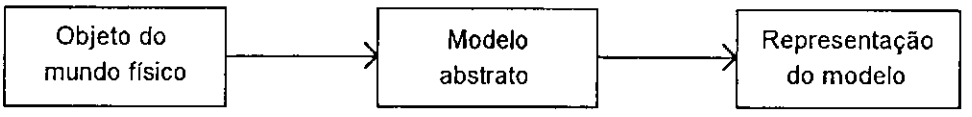


Figura 2.1 : Etapas envolvidas no processo de modelagem.

A área de modelagem de sólidos aplica-se a diversos propósitos, sendo útil a diversos tipos de aplicação. É comum, por exemplo, a existência de sistemas de modelagem de sólidos atuando como subsistemas de ferramentas mais complexas. O tópico seguinte apresenta em detalhes alguns aspectos sobre as aplicações de sistemas de modelagem de sólidos.

2.2. Áreas de Atuação e Aplicações Típicas

A construção de modelos sólidos em computador pode ter vários propósitos. [Gome90] afirma que "o objetivo da criação de modelos no computador é o de se efetuar simulações com o mesmo". Tipos diversos de simulação envolvem modelos com características diferentes utilizados em programas diferentes ,originando, portanto, aplicações diferentes. Atualmente existem diversas utilizações para modelos sólidos. Destacamos aqui algumas das mais importantes.

Uma das aplicações mais presentes de modelos sólidos são os sistemas de CAD/CAM. Como já citado no capítulo 1, esses sistemas auxiliam o projeto, estudo e fabricação de objetos sólidos tridimensionais em geral. Os modelos construídos em sistemas CAD/CAM permitem a simulação de diversos fenômenos. Em primeiro lugar, a simples construção de um modelo já simula a existência de um objeto sólido no computador. Além disso, é desejável que um sistema CAD/CAM considere as propriedades físicas dos objetos modelados. O sistema deve ser capaz de fornecer informações, tais como volume do objeto, propriedades de massa, peso etc. Podem, ainda, ser simulados fenômenos de colisão, mecânica estrutural, teste de funcionamento mecânico, entre outros.

Segundo [Thal87], em sistemas dessa natureza os modelos são construídos para serem estudados, visando enfaticamente ao estudo de suas propriedades funcionais. Os modelos

construídos em sistemas de CAD/CAM devem ser representados com precisão, ressaltando-se as suas propriedades físicas.

Outra aplicação de modelagem de sólidos, também já citada na introdução deste trabalho, são os Sistemas de Visualização. É interessante observar as diferentes características entre os modelos construídos em sistemas desse tipo e aqueles utilizados em CAD/CAM. Em Sistemas de Visualização, os modelos são construídos com uma finalidade estética ([Thal87]), atendendo a aplicações de caráter fotográfico. Um Sistema de Visualização deve permitir a construção de modelos que representem objetos diversos, que possam, por sua vez, ser utilizados para compor toda uma cena sintética.

Num Sistema de Visualização, além de simular-se a existência de objetos tridimensionais, é feita também a simulação do ambiente no qual os modelos se encontram, ou seja, simula-se o comportamento de luzes incidindo sobre os modelos. Para isso, podem ser simulados efeitos de cor, reflexibilidade, transparência, sombreamento, textura, entre outros. A complexidade dos modelos construídos varia de acordo com o realismo fotográfico que se deseja obter da cena. O grau de precisão necessário a esses modelos é determinado pela percepção da visão humana e pela qualidade estética desejada.

2.3. Aspectos Formais de um Modelo Sólido

Como todo modelo, os modelos sólidos devem preservar as características e o comportamento dos objetos originais por eles representados. Esse fato impõe diversas questões e pontos críticos a serem considerados ao se construir um modelo sólido. Existem alguns formalismos desenvolvidos com o intuito de compor uma base matemática para o estudo desses modelos. Um trabalho pioneiro nessa direção é apresentado por [Requ80], cujas abordagens incluem as seguintes definições :

- A **fronteira de um sólido** consiste na superfície que distingue as porções do espaço que se encontram no interior e no exterior do sólido.

- Um **modelo sólido** consiste em um subconjunto do espaço euclidiano tridimensional (E^3), e pode ser definido através de sua fronteira.

Segundo o ponto de vista matemático, um modelo sólido deve obedecer às seguintes propriedades (ver [Requ80], [Thal87] e [Gome90]) :

- **Rigidez :** um modelo sólido abstrato deve ter sua forma invariante, independentemente de sua localização e orientação.

- **Homogeneidade** : Um modelo sólido deve ter um interior definido, e sua fronteira não deve conter faces ou arestas abertas ("dangling faces/edges").

- **Finitude** : Um modelo sólido deve ser descrito por um número finito de elementos (faces, arestas, vértices), ocupando uma porção finita do espaço.

- **Fecho em movimentos rígidos e operações booleanas** : Operações de movimentação (translação/rotação), bem como operações booleanas (união/interseção/subtração), efetuadas em modelos sólidos devem resultar também em sólidos.

- **Determinismo de fronteira** : A fronteira de um modelo sólido deve determinar, sem ambigüidade, o interior e o exterior do sólido.

2.4. Esquemas de Representação e Domínio de um Sistema de Modelagem

2.4.1. Definições

No item 2.1, foi mencionado que diferentes aplicações de modelagem de sólidos podem envolver modelos com características diferentes. Assim, Sistemas de Modelagem de Sólidos (SMS) construídos com finalidades diferentes podem consistir em programas com muito poucos aspectos em comum. Esta diversidade pode ser encontrada em vários elementos dos sistemas, tais como no conjunto de ferramentas e operações oferecidas pelo sistema e na natureza dos modelos abordados. Dessa forma, durante as etapas de projeto e concepção de um SMS é importante que se definam os tipos de modelos que serão tratados pelo sistema bem como as operações a serem realizadas.

Ao se especificar o escopo de modelos que serão tratados por um Sistema de Modelagem de Sólidos define-se o chamado **Domínio** do sistema. O domínio de um SMS consiste no conjunto de todos os modelos válidos possíveis de serem construídos e tratados pelo sistema. Um modelo é considerado válido para um determinado sistema se ele obedece a certas propriedades previamente estabelecidas, assim como aquelas mencionadas no item 2.3.

Uma vez definido o escopo de modelos de um Sistema de Modelagem de Sólidos, é necessário definir-se a maneira como esses modelos serão representados e armazenados internamente no sistema. A forma de representação deve ser estipulada de acordo com a finalidade dos modelos a serem construídos e das ferramentas e facilidades que serão oferecidas para

manipular esses modelos. Evidentemente, os aspectos de eficiência computacional, tais como quantidade de memória ocupada, tempo de acesso, facilidade de implementação e manutenção, entre outros, também devem ser considerados. A maneira como modelos são representados em um Sistema de Modelagem de Sólidos é chamada de **Esquema de Representação**.

[Requ80] faz um estudo matemático enfocando esquemas de representação e domínios em Sistemas de Modelagem de Sólidos. Esse estudo é também citado por [Gome90], que apresenta explicações de caráter mais prático para os formalismos apresentados por [Requ80]. Dentre as diversas colocações apresentadas, [Requ80] e [Gome90] citam propriedades desejáveis de serem encontradas em um esquema de representação. Algumas dessas propriedades são :

- **Poder de descrição (domínio do sistema) :** O esquema de representação deve ser eficiente no sentido de englobar todos os modelos que se pretende modelar.

- **Validade** : Uma situação desejável é o fato de todas as representações admissíveis serem válidas.

- **Não-ambigüidade** : Uma dada representação deve representar um único modelo.

- **Unicidade** : Um dado modelo deve ter uma única representação.

- **Concisão** : Por questões de custo e otimização, um modelo deve ocupar o menor espaço possível em memória de máquina.

- **Operações** : Devem-se identificar as transformações que podem ser realizadas sobre os modelos, sem que esses percam a validade da representação.

- **Fidedignidade** : A geometria de um modelo deve ser representada com o maior maior grau de precisão possível.

Os formalismos pioneiramente introduzidos por [Requ80] dão um suporte matemático bastante útil à área de representação de sólidos. Entretanto, a maior parte da literatura analisa esquemas de representação segundo aspectos práticos e intuitivos. A partir de levantamentos feitos na literatura, especialmente em [Baer79], [Requ80], [Requ82], [Requ83], [East84], [Thal87], [Mill89] e [Gome90], podem-se destacar seis tipos básicos de esquemas de representação para modelos sólidos :

- Representação por Instanciamento de Primitivas

- Representação por Enumeração Espacial

- Representação por Decomposição Celular

- Representação por Arrasto (Sweeping)

- Representação por Bordas (Boundary Representation)

- Representação CSG (Constructive Solid Geometry)

As referências bibliográficas citadas fazem uma abordagem bastante ampla sobre esses vários tipos de esquemas de representação. Será feita, aqui, apenas uma exposição superficial sobre cada um desses esquemas. Ao leitor interessado, recomenda-se que complete a leitura do assunto com a lista de referências bibliográficas inserida ao final deste texto.

2.4.2. Representação através de Instanciamento de Primitivas

Neste tipo de representação, os modelos são representados por tuplas contendo parâmetros. Estes parâmetros especificam a **família** do modelo (ex: cilindro, cone, prisma etc.) e suas características (altura, raio, número de faces etc.). Cada **família** é também chamada de **primitiva genérica**, e cada objeto de uma **família** corresponde a uma **instância de primitiva**. Por exemplo, uma esfera poderia ser representada por uma tupla do tipo <'ESFERA', C, R>, onde 'ESFERA' indica a família do modelo, **C** indica as coordenadas do centro da esfera, e **R** indica o raio. Podemos, também, definir o modelo <'PRISMA', N, R, H>, onde **N** indica o número de faces do prisma, **R** indica o raio, e **H** especifica a altura.

As principais vantagens desse tipo de representação residem na a facilidade de uso para determinadas operações, na não-ambigüidade e na concisão dos modelos construídos. Apesar dessas vantagens, esquemas desse tipo apresentam diversos pontos problemáticos e desvantajosos. Entre os principais, estão o domínio muito restrito, a impossibilidade de se combinarem várias instâncias para se criarem modelos mais complexos, além da dificuldade de se escreverem algoritmos para determinar propriedades dos modelos representados. [Requ80] afirma que esquemas de representação por instanciamento de primitivas tendem ao desuso.

2.4.3. Representação por Enumeração Espacial

Vamos supor a divisão do espaço em células cúbicas de igual tamanho e colocadas em posições fixas. A representação de um sólido por enumeração espacial é feita através da indicação daquelas células que são ocupadas pelo sólido. Cada célula é também

chamada de **voxel** (abreviação de **volume element**). O tamanho de cada célula determina a precisão e a resolução da representação. [Thal87] aponta as vantagens e desvantagens deste tipo de representação. As vantagens mais evidentes são a facilidade de validação dos modelos, a facilidade de acesso a um determinado ponto e a garantia da unicidade dos modelos (ver item 2.4.1). [Thal87] aponta como principais desvantagens, a falta de estruturas que descrevam as conexões entre as partes dos modelos, a grande quantidade de memória necessária e a baixa precisão de representação.

2.4.4. Representação por Decomposição Celular

Uma maneira de se representar um sólido é subdividí-lo em células arbitrárias e representar essas células. Por exemplo, um cubo pode ser dividido em dois tetraedros. Assim é, basicamente, o método de representação por decomposição celular. A representação por enumeração espacial é um caso particular de decomposição celular onde as células são cúbicas e são situadas em posições fixas do espaço. São as seguintes as principais vantagens oferecidas por este tipo de representação :

- Os modelos construídos garantem a não ambiguidade.

- Podem-se calcular algumas propriedades físicas dos modelos.

Como principais desvantagens identificadas neste tipo de representação enumeram-se :

- A unicidade não é garantida, uma vez que existem várias formas de se subdividir um modelo.

- [Requ80] alerta que, geralmente, modelos de decomposição celular não são fáceis de criar.

- [Requ80] também chama atenção para o fato de que, geralmente, modelos de decomposição celular não são concisos.

2.4.5. Representação por Arrasto ("sweeping")

Este método de representação é bastante interessante, e tem uma popularidade maior do que os apresentados anteriormente. As representações por arrasto **("sweeping")** se baseiam no movimento de uma dada superfície bidimensional ao longo de uma determinada

trajetória. Armazenando-se os dados da superfície bidimensional e do caminho percorrido por ela, tem-se a representação do sólido desejado. Um exemplo deste tipo de criação de modelos é apresentado na porção mais à esquerda da figura 2.2. A figura apresenta uma superfície plana, identificada pela letra **S**, pertencente ao plano X-Z do sistema de coordenadas. A translação da superfície **S** ao longo do eixo Y produz um sólido. Os sólidos gerados pela translação de superfícies empregam o chamado **sweeping translacional**.

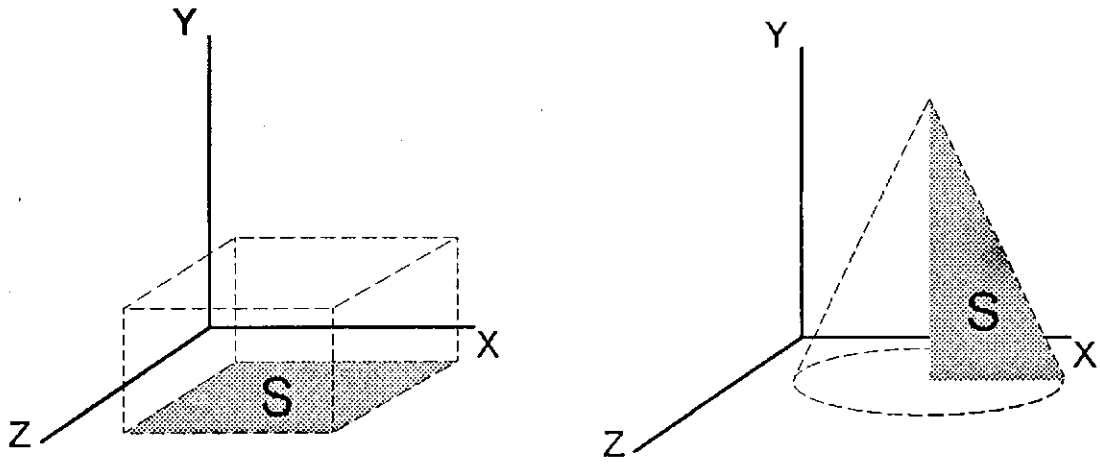


Figura 2.2 : Sólidos produzidos por "sweeping" translacional e rotacional.

Outro tipo de "sweeping" é o chamado **sweeping rotacional**. O processo é análogo ao sweeping translacional, diferindo no fato de a trajetória percorrida pela superfície ser uma rotação, em lugar de uma translação. Na porção direita da figura 2.2, a rotação da superfície **S** em torno do eixo Y produz o sólido indicado.

As técnicas e princípios de "sweeping" podem ser extendidas a outras aplicações. Por exemplo, o deslocamento de uma curva ao longo de uma trajetória pode ser usado para gerar uma superfície. Existe um caso ainda mais complexo, conhecido como "sweeping tridimensional", no qual se faz o arrasto de sólidos para se produzirem novos sólidos.

A unicidade não é garantida nas representações por "sweeping". Além disso, este método é aplicado somente na modelagem de sólidos que tenham simetria rotacional ou translacional, ou seja, que apresentem formato uniforme de acordo com a translação ou rotação realizada. As vantagens oferecidas por este tipo de representação residem na concisão dos modelos construídos e na não-ambigüidade.

2.4.6. Representação por Bordas (Boundary Representation)

Este é um dos tipos de representação mais difundidos e aplicados atualmente, merecendo destaque e atenção especiais. A representação por bordas também é conhecida como representação B-Rep (abreviação de Boundary Representation). O princípio da representação B-Rep é representar um dado sólido através de sua fronteira. Como visto no item 2.3, a fronteira de um sólido consiste na superfície que o envolve e distingue os pontos do espaço que estão contidos no interior do sólido daqueles pontos que se encontram fora do mesmo.

Geralmente, utiliza-se um esquema hierárquico de três entidades para se representar a fronteira de um sólido. Essas entidades são **faces**, **arestas** e **vértices**. Dessa forma, um sólido é composto por um número finito de faces. As faces, por sua vez, são formadas por arestas, e estas são definidas por vértices. A figura 2.3 ilustra uma estrutura típica para a representação de um sólido através da sua fronteira.

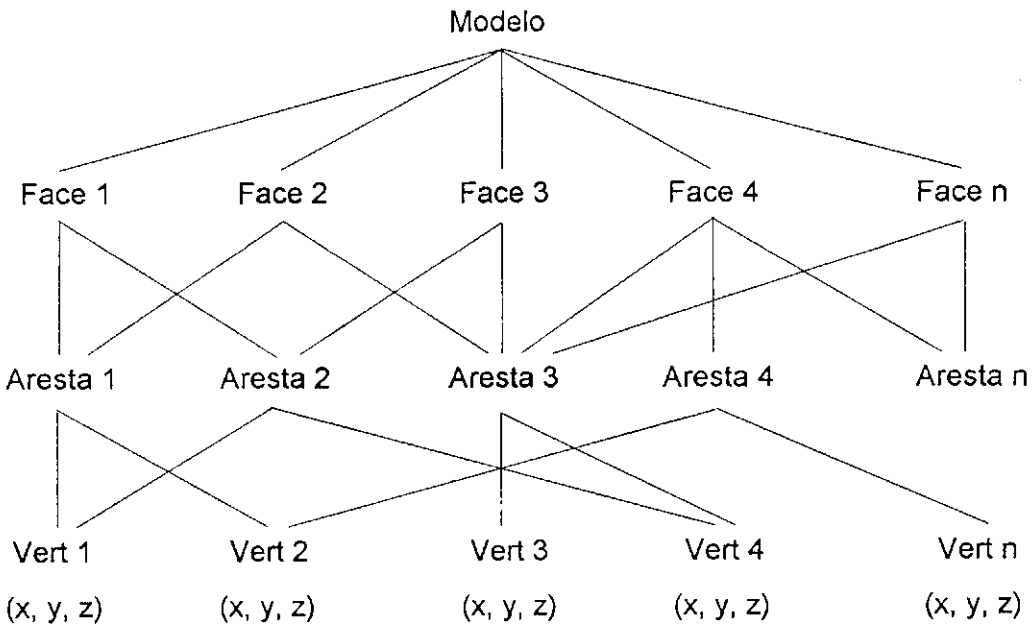


Figura 2.3 : Estrutura de representação de um sólido através de faces, arestas e vértices (Boundary Representation).

Esse tipo de estrutura armazena separadamente as informações geométricas, que incluem a dimensão e a posição do modelo no espaço, das informações topológicas, que definem as conexões entre os vários elementos do modelo (faces, arestas e vértices). [Requ80] afirma que

essa separação pode se tornar conveniente, uma vez que, em estruturas como essa, para se transladar um modelo, por exemplo, basta se transladar o conjunto de seus vértices, sem se alterar as estruturas topológicas do modelo.

Em alguns casos, as representações B-Rep podem permitir a definição de faces curvas. Contudo, essa característica implica em sérias dificuldades na construção de algoritmos. O mais comum é representar faces na forma de polígonos planos. Em representações desse tipo, as porções curvas de um objeto qualquer são representadas por aproximações poligonais. A figura 2.4 ilustra um caso como esse.

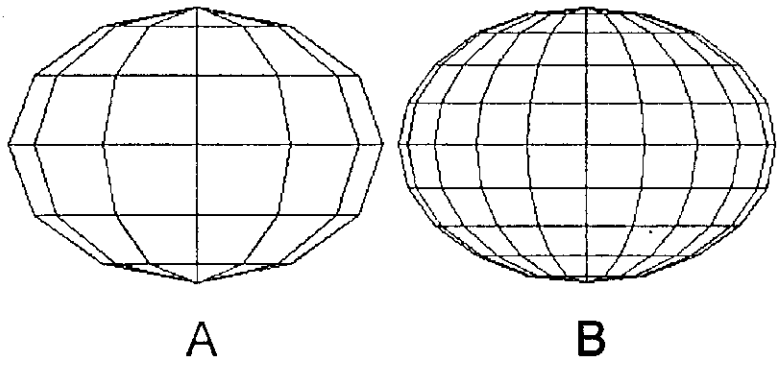


Figura 2.4 : Representação de faces curvas por aproximação poligonal.

[Requ80] faz um estudo abrangente sobre as características da representação B-Rep, incluindo uma análise das propriedades obtidas por modelos desta representação. Em suas considerações, o autor aponta as seguintes vantagens para a representação B-Rep :

- As estruturas B-Rep permitem facilmente a inclusão de informações diversas associadas ao modelo, tais como os parâmetros do vetor normal de uma face, a equação da reta de uma dada aresta, estruturas de dados auxiliares ligadas a qualquer porção de um dado objeto, propriedades de um objeto, como cor, coordenadas do baricentro etc.

- O domínio abrangido pelas representações B-Rep é bastante vasto, sendo comparável ao domínio da representação CSG (ver item 2.4.7).

- Modelos representados em B-Rep não são ambíguos se suas faces são representadas sem ambigüidade.

- Estruturas B-Rep são bastante eficientes em tarefas como traçado de linhas, exibição gráfica, e interatividade.

Segundo [Requ80], alguns dos pontos críticos e desvantagens identificadas são :

- Representações B-Rep são muito extensas, ocupando grandes porções de memória.

- Há dificuldade de criação de modelos em B-Rep. [Requ80] afirma que usuários humanos dificilmente o fazem sem o auxílio de alguma ferramenta computacional.

As representações B-Rep são bastante populares, existindo na literatura diversas propostas para a construção de estruturas de dados e algoritmos envolvendo essa técnica.

2.4.7. Representação CSG (Constructive Solid Geometry)

Esta é outra técnica de representação bastante utilizada, e que também se destaca neste trabalho. Constructive Solid Geometry (Geometria de Sólidos Construtiva) se baseia na representação de sólidos através de operações sucessivas de união, interseção e subtração de primitivas sólidas. Para isso é construída uma árvore binária, onde os nós-folha contêm primitivas sólidas ou parâmetros para operações de transformação (translação, rotação ou escala). Os nós-não-folha representam operações booleanas (união, interseção ou subtração) ou operações de transformação. O nó-raiz dá acesso ao modelo final construído. Estes procedimentos podem ser visualizados através da figura 2.5, que mostra, hipoteticamente, a representação CSG de um modelo bidimensional.

A porção esquerda da figura contém a representação CSG do modelo bidimensional apresentado na parte direita da mesma, no caso, uma casa de pombos. O nó da árvore indicado com o número 1 (um) contém uma operação conjunta de transformação, composta por uma operação de translação e outra de escala. Os argumentos para essas operações estão nos nós-filhos, onde um dos argumentos é uma primitiva bidimensional (no caso, um quadrilátero branco) e o outro são valores explícitos de TX, TY, SX, e SY, indicando respectivamente os valores de translação e escala ao longo dos eixos X e Y. As operações de translação e escala posicionam o modelo (quadrilátero branco) e alteram suas dimensões, de modo a se obter o resultado desejado. Caso fosse necessário, uma operação de rotação também poderia ser realizada, utilizando parâmetros RX e RY.

A árvore CSG apresentada também indica, no nó indicado pelo número 2 (dois), uma operação de transformação sobre um quadrilátero negro.

Os resultados das duas operações anteriores são utilizados em uma operação de união, indicada pelo nó de número 3 (três). Nesse ponto, o modelo obtido consiste na haste de sustentação da casa de pombos (letra A) e no corpo da mesma (letra B).

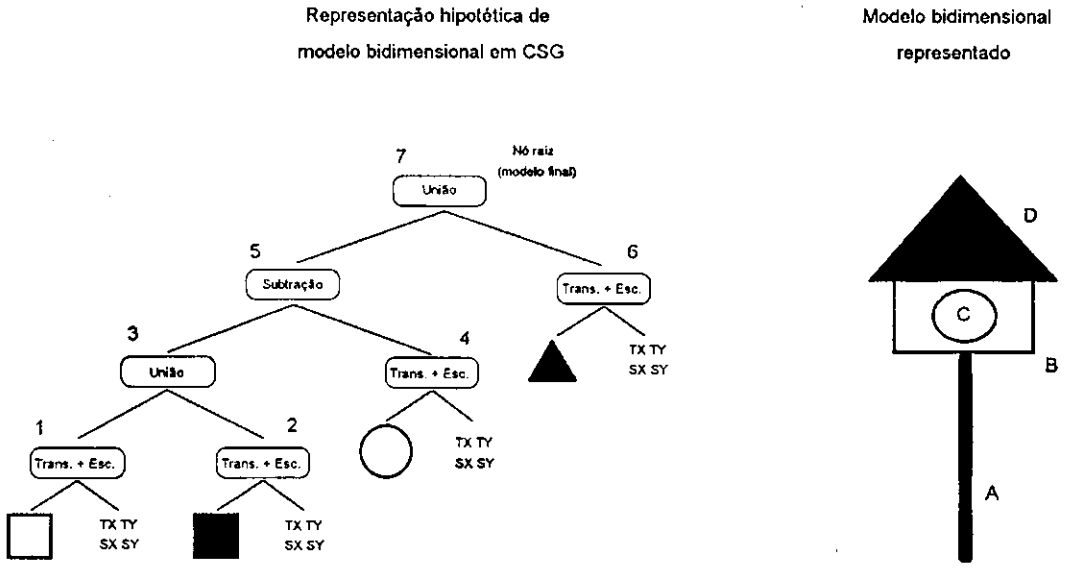


Figura 2.5 : Representação de um modelo bidimensional através de CSG.

Observando-se a árvore construída, pode-se notar que, paralelamente à operação de união anterior, é feita uma operação de transformação sobre uma circunferência branca (nó número 4). Essa operação posiciona e dimensiona a circunferência que, posteriormente, é subtraída (nó número 5) do modelo existente até então. O efeito dessa subtração é criar um "furo" no corpo da casa de pombos (letra C).

O nó de número 6 (seis) indica uma operação de transformação sobre uma primitiva triangular, criada com o objetivo de constituir o telhado da casa de pombos. Depois de posicionada e dimensionada, essa primitiva é adicionada ao modelo anterior, constituindo o modelo final. O nó-raiz da árvore dá acesso ao modelo final construído.

Em representações CSG, as primitivas podem ser definidas tanto através de semi-espaços, quanto por instanciamento de primitivas (ver [Requ80] e [Mill89]).

A representação CSG apresenta os seguintes pontos favoráveis :

- Quando são utilizadas primitivas sólidas válidas, os esquemas CSG garantem que os sólidos finais também serão válidos.

- Esquemas CSG são não ambíguos.

- Um modelo CSG é relativamente fácil de se criar, uma vez que a sua formação é intuitivamente clara.

As representações CSG apresentam os seguintes pontos críticos :

- Representações CSG não são fontes eficientes de dados geométricos para se efetuarem tarefas como traçado de linhas e determinados tipos de interatividade.

- A unicidade dos modelos não é garantida nas representações CSG.

2.5. Esquemas Híbridos de Representação

Como pôde ser visto anteriormente, cada método de representação oferece pontos favoráveis e desfavoráveis. Dependendo dos modelos a serem construídos, além dos tipos de operação e dos algoritmos a serem aplicados, varia a adequação de cada esquema de representação. [Requ80], [Requ82], [Requ83], [East84], [Thal87], [Mill89] e [Gome90] são unânimes em afirmar que nenhum esquema de representação é uniformemente melhor do que outro. Todos eles oferecem vantagens sob alguns aspectos e desvantagens sob outros. Visando a um melhor proveito das vantagens oferecidas por cada método de representação, foram idealizados os chamados **esquemas híbridos de representação.**

A idéia central de um esquema híbrido é representar um modelo combinando-se vários métodos de representação.

Inicialmente, os esquemas híbridos eram implementados de modo que os vários métodos de representação empregados conviviam simultâneamente na mesma estrutura de dados. Segundo [Requ80], os esquemas híbridos mais comuns aliavam CSG com B-Rep e CSG com "Sweeping", construindo-se árvores CSG, cujas primitivas eram representadas em B-Rep ou "Sweeping". Apesar das vantagens oferecidas, os esquemas desse tipo apresentavam sérias complicações algoritmicas (ver [Requ80], [Mill89]). Devido a essas complicações, os esquemas desse tipo não foram muito desenvolvidos, como diz [Mill89].

Uma outra forma a que se chegou para aliar diferentes métodos de representação consiste nas chamadas **representações múltiplas** ou **multirrepresentações**. Em esquemas desse tipo, os modelos são representados redundantemente por vários métodos de representação. Na figura 2.7 é apresentada uma arquitetura ideal (e ainda impossível devido à ausência de todos

os algoritmos de conversão entre representações necessários) proposta por [Mill89] para essa forma de representação.

O comportamento dos esquemas de representação múltipla pode ser descrito da seguinte forma : os vários métodos de representação coexistem paralelamente representando, redundantemente, o mesmo modelo de várias formas diferentes. Um dos esquemas é estabelecido como sendo o **esquema primário** e tido como o esquema principal. Todas as operações de criação e transformação efetuam-se sobre esta representação primária. Desejando-se efetuar uma operação não muito adequada para a representação primária, aplica-se um algoritmo de conversão que, a partir da representação primária, gera uma nova representação para o modelo. A operação desejada é, então, realizada sobre esta nova representação.

Vamos imaginar, como exemplo, um esquema de representação como o apresentado na figura 2.6, que mantém dois métodos : CSG e B-Rep, onde o esquema CSG consiste no esquema primário. O esquema CSG atende bem a operações tais como criação dos modelos e verificação de pontos pertencentes ou não a determinados modelos. Entretanto, existem operações onde a representação B-Rep se torna mais eficiente. Um exemplo disso é o traçado de modelos no monitor de vídeo do computador. Num caso como esse, deve-se aplicar um algoritmo de conversão, que gera a representação B-Rep do modelo a partir de sua representação CSG. Uma vez de posse da representação B-Rep, efetua-se a operação desejada.

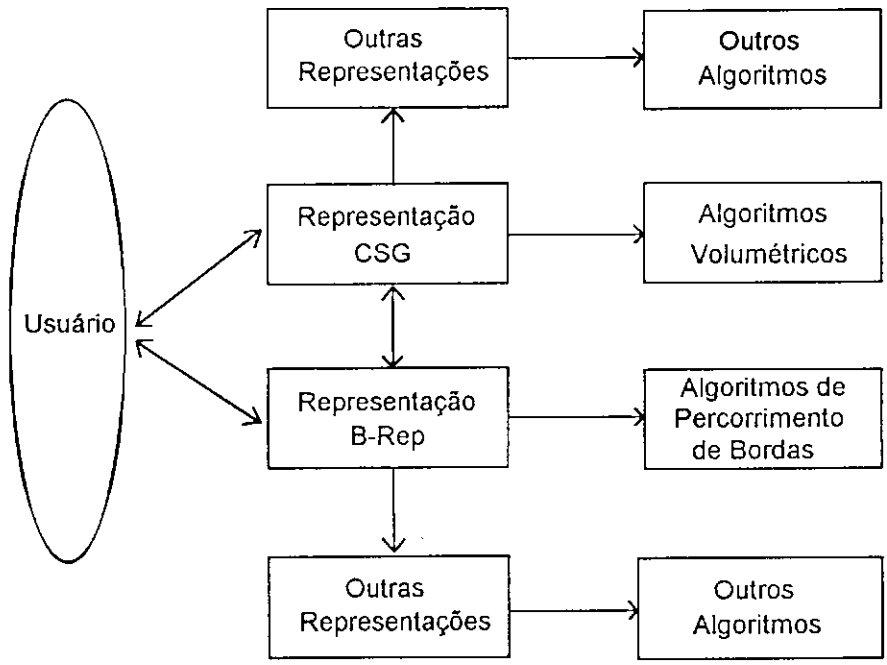


Figura 2.6 : Exemplo de arquitetura multirrepresentacional apontado por [Mill89].

Existem diversos pontos críticos nas representações múltiplas. Entre os principais podemos enumerar :

- **Conversão** : Ainda não existem todos os algoritmos de conversão necessários para a perfeita exploração dos esquemas de multirrepresentação. [Requ80], [Thal87] e [Mill89] fazem considerações sobre o atual estado da tecnologia referente a esses algoritmos. Além disso, os algoritmos de conversão existentes atualmente apresentam custo computacional elevado. É justamente esse ponto crítico que impossibilita, atualmente, a construção de sistemas híbridos segundo a arquitetura ideal proposta por [Mill89].

- **Consistência** : O fato de se manterem representações redundantes leva a problemas de consistência. É necessário que as várias representações sejam consistentes entre si, representando sempre modelos iguais de formas diferentes.

- **Domínio** : Os vários esquemas de representação devem ser projetados de modo que seus domínios sejam iguais.

Os estudos apresentados por [Mill89] incluem ainda outras colocações importantes. O primeiro ponto ressaltado pelo autor é a forte tendência de consolidação dos esquemas de representação múltipla. Apesar da tecnologia ainda não oferecer subsídios suficientes para o uso irrestrito dessas representações, existem fortes pesquisas nessa direção, chegando a por em desuso os esquemas híbridos inicialmente concebidos. Devido a este fato, [Mill89] chama atenção para o fato de ter havido, na literatura, alguma confusão sobre o emprego do termo **esquema híbrido de representação**. [Mill89] diz que o termo pode ser empregado tanto para se referir aos esquemas híbridos, apresentados anteriormente nesta seção, como aos esquemas de representação múltipla. Devido à atual tendência de se desenvolverem esquemas multirrepresentacionais, [Mill89] emprega o termo "híbrido" somente para se referir aos esquemas de representação múltipla, desconsiderando os esquemas híbridos iniciais.

2.6. Transformações Geométricas

Além da criação e representação de modelos, um Sistema de Modelagem de Sólidos deve oferecer operações para a transformação e manipulação de modelos. As chamadas **transformações geométricas** são operações que permitem que os modelos criados sejam modificados e manipulados, alterando-se suas dimensões e sua posição no espaço. Transformações Geométricas são um tema essencial em modelagem, estando presente em inúmeras referências bibliográficas. Recomenda-se, como fonte de leitura sobre o assunto, as

referências [Newm81], [Fole82], [Muft83], [Park85], [Berg86], [Hear86], [Thal87], [Picc88] e [Gome90]. Essas referências apresentam desde aspectos teóricos, até nuances e técnicas de implementação. Neste texto, será feita uma abordagem introdutória daquelas operações consideradas mais importantes e fundamentais.

Uma das transformações geométricas mais elementares é a operação de **translação**. Essa operação permite que os modelos sejam deslocados ao longo dos eixos X, Y e Z, assumindo novas posições do espaço. A figura 2.7 ilustra um modelo antes e depois de sofrer uma operação de translação.

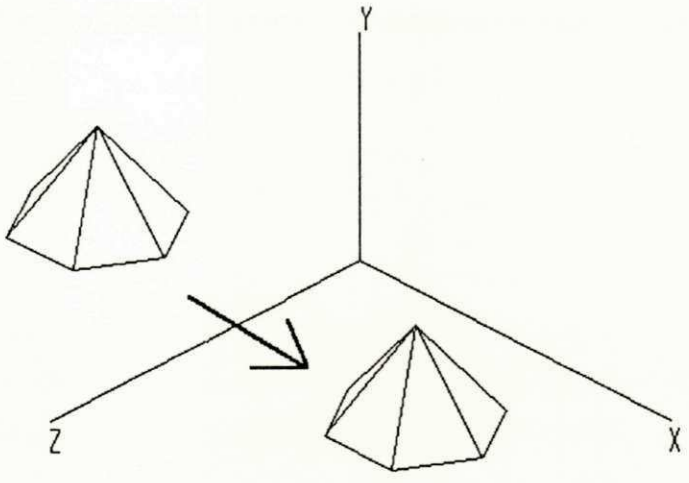


Figura 2.7 : Translação de um modelo sólido.

Outra transformação geométrica essencial em um Sistema de Modelagem de Sólidos é a operação de **rotação**. Essa operação também afeta o posicionamento dos modelos. Com ela os modelos podem ser rotacionados em torno dos eixos X, Y e Z, com ângulos de rotação previamente informados. Uma rotação em torno de um eixo arbitrário pode ser obtida através da combinação de rotações em torno dos eixos X, Y e Z. Comumente, as operações de rotação são feitas rotacionando-se o modelo em torno de seu baricentro, alterando somente a orientação do modelo. Na figura 2.8, é apresentado um modelo, inicialmente em sua posição original, sofrendo translações e rotações sucessivas em torno dos eixos X, Y e Z.

Outra transformação geométrica que se deve apresentar consiste na operação de **escala**. Essa operação se distingue das anteriores pelo fato de alterar a dimensão dos modelos, ao invés de seu posicionamento e orientação. Através de uma operação de escala, um dado modelo pode ter suas dimensões aumentadas ou diminuídas, ao longo dos eixos X, Y e Z, de acordo com o resultado final desejado. A figura 2.9 ilustra os resultados de diferentes operações de escala efetuadas sobre um paralelepípedo.

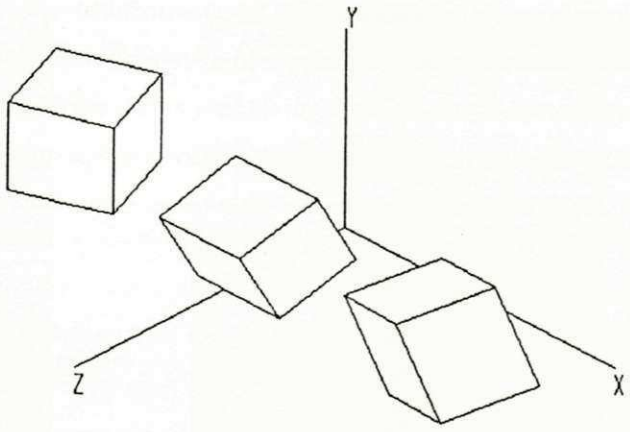


Figura 2.8 : Rotações e translações sucessivas em um modelo sólido.

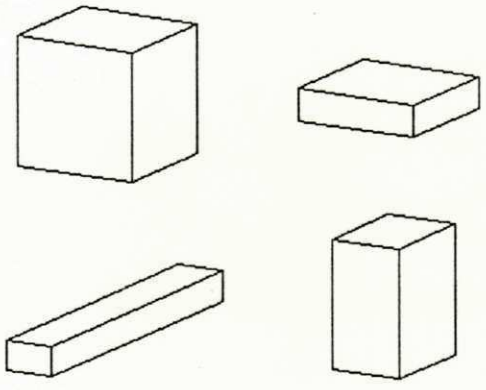


Figura 2.9 : Operações de escala efetuadas sobre um paralelepípedo.

2.7. Transformações de Visualização

As chamadas **transformações de visualização** são operações ligadas ao processo de exibição dos modelos construídos. Estas operações agem sobre os modelos, empregando efeitos de apresentação aos mesmos. As transformações de visualização exigem o estudo de questões fundamentais e nem sempre triviais, tais como a posição de observação dos modelos, o tamanho da janela de observação, a supressão das porções não visíveis, a exibição em perspectiva, entre outras. Mencionaremos aqui o papel de algumas transformações de visualização. Para um estudo mais detalhado recomenda-se a consulta às referências [Newm81], [Fole82], [Muft83], [Park85], [Berg86], [Hear86], [Thal87], [Picc88] e [Gome90].

Uma das transformações de visualização mais destacadas é a operação de **posicionamento do observador**. Essa operação permite que o modelo construído seja observado a partir de diversas posições do espaço. A figura 2.10 ilustra um mesmo modelo observado de diferentes posições. Internamente, o funcionamento dessa operação consiste em se efetuar uma rotação em todos os modelos presentes na cena, simulando uma mudança na posição do observador.

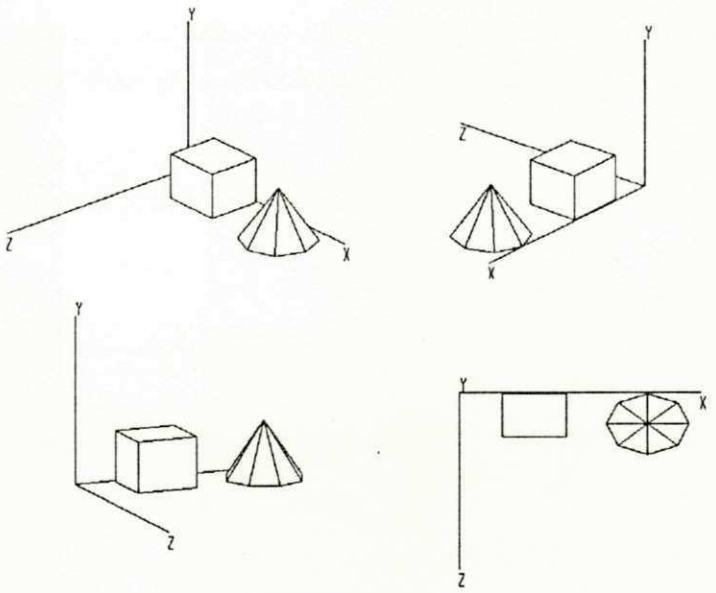


Figura 2.10 : Mudança na posição do observador.

Um efeito visual bastante empregado ao se exibirem modelos sólidos é o **apagamento de linhas ocultas**. Essa operação consiste em eliminar da cena aquelas linhas que se encontram "atrás" de outros modelos ou porções de modelos. Esta operação tem a utilidade de eliminar a ambigüidade presente nas exibições em "**wire-frame**" (arame). A figura 2.11 exemplifica esta ambigüidade, mostrando um modelo exibido inicialmente em "wire-frame" (ambíguo) e as duas possíveis exibições do mesmo modelo utilizando-se técnicas de apagamento de linhas ocultas.

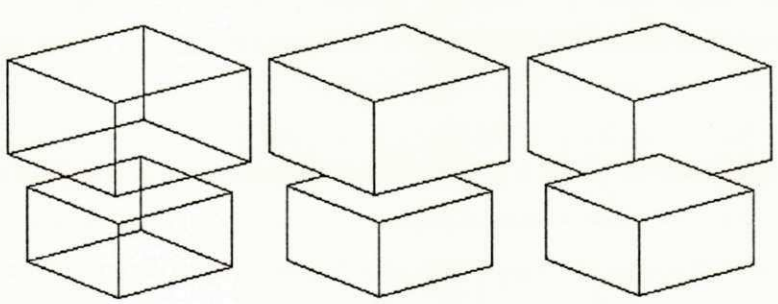


Figura 2.11 : Exemplo da ambigüidade da exibição em wire-frame, eliminada com o apagamento de linhas ocultas.

Outro efeito visual bastante popular e que também visa dar mais realismo aos modelos exibidos é a chamada **exibição em perspectiva**. Essa operação visa alterar a apresentação dos modelos de modo a simular efeitos de profundidade. A figura 2.12 exemplifica um caso como esse.

Existem, ainda, muitas outras questões e técnicas empregadas na visualização de modelos. Entretanto, entende-se que esses aspectos são bastante conhecidos na área de computação gráfica e bastante abordados na literatura, não cabendo aqui uma abordagem extensa sobre o assunto. O leitor interessado pode consultar as referências já indicadas.

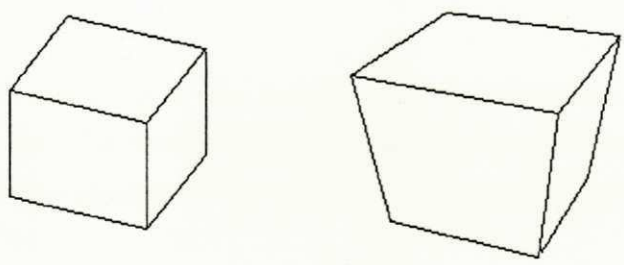


Figura 2.12 : Exibição de um modelo em perspectiva.

2.8. Que características devem estar presentes num Sistema de Modelagem de Sólidos ?

Através de levantamentos feitos na literatura especializada, podem-se constatar algumas características básicas de um Sistema de Modelagem de Sólidos. Segundo os estudos apresentados em [Requ80], [Requ82], [Requ83], [Picc88] e [Mill89], entre outras referências, podem-se delinear alguns elementos essenciais componentes de um SMS de propósito geral. São eles :

A) Subsistema de Entrada de Dados : O subsistema de entrada agrupa as funções de um SMS responsáveis pela aquisição de dados e realização de operações. Este subsistema faz parte da interface homem-máquina e define a forma de alimentação de dados e utilização das funções do SMS. Existem várias formas de implementação do subsistema de entrada. Em alguns SMS's, o usuário dispõe de linguagens de definição, utilizadas para a especificação dos modelos construídos e para a realização de operações sobre esses modelos. Outros SMS's

adotam interfaces baseadas em métodos interativos. Existem, ainda, sistemas que utilizam paralelamente ambas as técnicas.

B) Esquema de Representação : Segundo [Requ80], um Sistema de Modelagem de Sólidos tem de implementar, obrigatoriamente, um ou mais esquemas de representação (ver itens 2.4 e 2.5). O esquema de representação contém os dados a serem utilizados pelas diversas funções do sistema.

C) Operações de Transformação : É imperativo que Sistemas de Modelagem de Sólidos ofereçam algum tipo de operação para a transformação de modelos. Caso contrário, seria descaracterizado o papel de modelagem. As operações tipicamente presentes são as funções de transformação geométrica (translação, rotação, escala), além das operações booleanas de união, interseção e subtração.

D) Visualização : As operações de visualização estão ligadas aos procedimentos de exibição e apresentação dos modelos construídos. Apesar de estarem presentes na maioria dos SMS's, estas funções não são obrigatórias nos casos em que o SMS's é usado como subsistema de uma ferramenta mais abrangente.

E) Subsistema de Saída : O subsistema de saída tem como papel a apresentação dos resultados finais obtidos. Dependendo da forma como o sistema é desenvolvido, a apresentação dos resultados pode ser feita em telas de vídeo, papel impresso, máquinas fotográficas, "plotters", entre outros tipos de mídia. Os resultados também podem ser apresentados na forma de arquivos de dados a serem utilizados como alimentação para outras ferramentas.

A figura 2.13 esquematiza os componentes um Sistema de Modelagem de Sólidos, propostos por [Mill89].

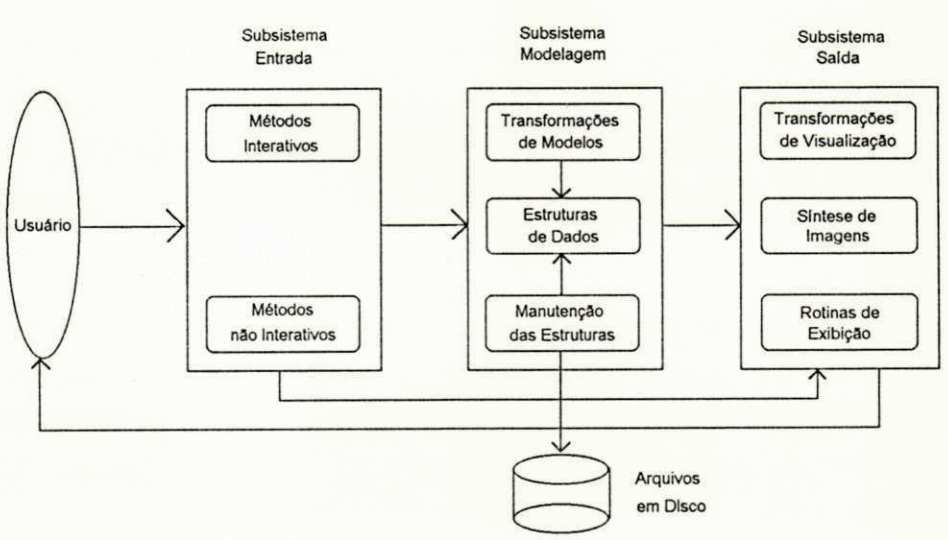


Figura 2.13 : Elementos de um Sistema de Modelagem de Sólidos segundo [Mill89].

Os tópicos abordados no capítulo 2 completam a fundamentação teórica considerada indispensável à perfeita compreensão deste trabalho. A seguir, o capítulo 3 aborda a descrição funcional do sistema João-de-Barro.

Capítulo 3

Descrição Funcional do Sistema João-de-Barro

3. DESCRIÇÃO FUNCIONAL DO SISTEMA JOÃO-DE-BARRO

Este capítulo tem como finalidade descrever o funcionamento do sistema João-de-Barro, apresentando seus módulos, as funções oferecidas, a interface com o usuário, entre outras informações. São apresentadas, também, instruções básicas de como operar o sistema e utilizar seus recursos.

3.1. Apresentação

O sistema João-de-Barro consiste numa ferramenta para atividades de modelagem de sólidos, concebida com o intuito de implementar e avaliar alguns dos fundamentos teóricos apresentados no capítulo 2. O desenvolvimento deste sistema representa um passo inicial de um trabalho de pesquisa sobre sistemas de visualização e síntese de imagens. Assim, o João-de-Barro consiste num modelador construído especificamente para atender a aplicações de caráter visual e estético.

O sistema foi desenvolvido na forma de um modelador mínimo, ou seja, contendo somente as ferramentas básicas para as tarefas de modelagem a que se propõe. Entretanto, o sistema foi projetado de forma a ser facilmente expandível, prevendo e propiciando o acréscimo de novas funções. Espera-se que, em trabalhos futuros, novos recursos de modelagem e visualização sejam adicionados ao sistema, tornando-o mais completo e flexível.

O projeto do sistema seguiu algumas diretrizes básicas. Primeiramente, devido ao escopo limitado do trabalho, restringiu-se o domínio do sistema a modelos sólidos tridimensionais. A definição do esquema de representação adotado tomou como base as considerações expostas nos itens 2.4.6, 2.4.7 e 2.8, tendo-se optado pela proposta de um esquema de representação híbrido, aliando as técnicas de Boundary Representation e CSG. Também baseado no item 2.8, foi escolhido o conjunto de operações para transformações geométricas e transformações de visualização a serem incorporadas ao sistema. Todos esses aspectos de arquitetura interna e técnicas empregadas são detalhadamente abordados e argumentados no capítulo 4, referente à implementação do sistema.

Apesar do sistema ter sido desenvolvido diretamente para a área de síntese de imagens, acredita-se que, com algumas adaptações, os recursos desenvolvidos possam também ser utilizados em outras aplicações, tais como CAD/CAM, robótica, sistemas de controle numérico, entre outras. Dentre essas adaptações, poderiam-se citar a inclusão de novas informações nas estruturas de dados, a adoção de outros esquemas de representação paralelos ao já existente, e

melhorias na interface homem-máquina do sistema que permitissem maior precisão na construção dos modelos.

3.2. Descrição Geral

3.2.1. Filosofia de Trabalho

Como dito anteriormente, a finalidade do sistema é a construção de cenas sintéticas tridimensionais para efeitos de visualização. Uma cena é composta por vários objetos, dispostos em posições diversas do espaço tridimensional. Identificamos, de antemão, os dois elementos básicos de trabalho do sistema : **Objetos** e **Cenas**.

No sistema João-de-Barro um **Objeto** consiste em um modelo sólido que represente um objeto físico do mundo real. Um objeto é um elemento unitário, ou seja, é a unidade mínima de trabalho do sistema e não pode ser tratado parcialmente.

Uma **Cena** é composta por vários objetos dispostos em posições específicas do espaço tridimensional. Assim, a construção de uma cena implica em construir e posicionar todos os objetos participantes da mesma. Outros elementos componentes de uma cena são as características de visualização da cena construída, tais como a posição do observador, o padrão de exibição (pespectiva, linhas ocultas apagadas, iluminação etc.), a posição da(s) fonte(s) de iluminação etc.

O processo de construção de uma cena através do sistema João-de-Barro segue as seguintes etapas :

a) Idealização e projeto da cena a ser construída.

b) Idealização e projeto de cada objeto componente da cena.

c) Construção de cada objeto participante da cena.

d) Posicionamento de cada objeto da cena.

e) Indicação das características de visualização da cena.

f) Exibição final da cena construída.

A construção de um objeto tem início a partir de uma **primitiva sólida**, que nada mais é do que um modelo sólido, tal como um cubo, uma esfera, um cilindro, ou um outro modelo qualquer que pode ser transformado e combinado com outros modelos para gerar modelos mais complexos e rebuscados. Como ilustração deste processo, a figura 3.1 apresenta três cilindros utilizados como primitivas que, após transformados, posicionados e anexados, formam um eixo com duas rodas. O modelo final criado poderia ainda ser utilizado em processos subseqüentes, sendo transformado e combinado para compor outros modelos ainda mais complexos.

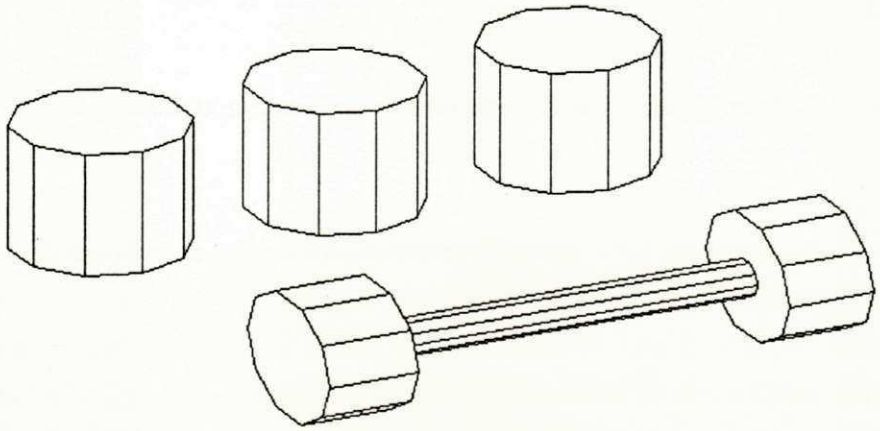


Figura 3.1 : Construção de um modelo a partir da transformação e combinação de primitivas.

O sistema dispõe de várias funções para a criação, transformação e combinação de primitivas, bem como as demais ferramentas necessárias. As seções subsequentes apresentam e descrevem estas ferramentas.

3.2.2. Organização Geral do Sistema

O sistema é dividido em quatro módulos principais e dois módulos de apoio. Cada módulo tem um papel definido e reúne funções diversas de acordo com esse papel. O diagrama apresentado na figura 3.2 esquematiza a organização geral do sistema. Os módulos principais são : "Principal", "Segmentação", "Edição" e "Visualização". Os módulos de apoio oferecidos pelo sistema são "Configuração" e "Miscelâneas".

O **módulo principal** tem por função unicamente dar acesso aos demais módulos do sistema. O módulo principal é o módulo inicialmente ativo quando o sistema é colocado em

execução. Através deste módulo, tem-se acesso direto aos módulos de segmentação, edição, visualização e configuração

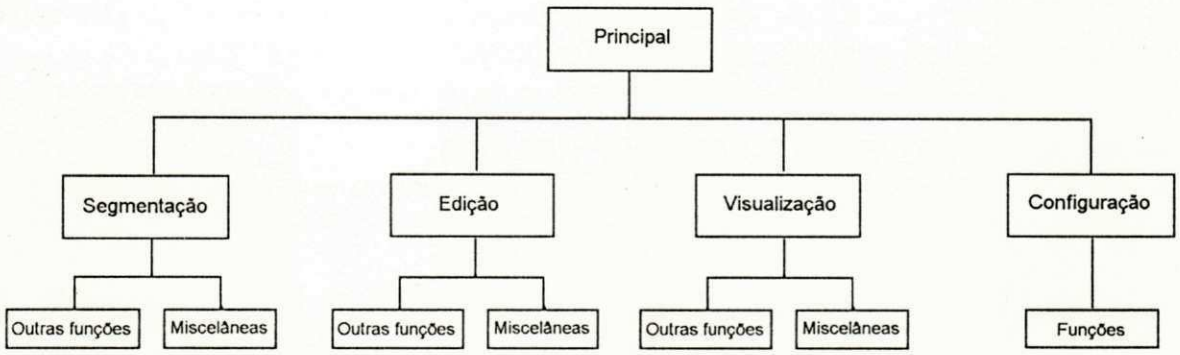


Figura 3.2 : Organização geral dos módulos do sistema.

O **módulo de segmentação** oferece funções para a manutenção de **segmentos**. Um segmento consiste na porção das estruturas de dados do sistema destinada ao armazenamento de um determinado objeto. As operações de manutenção de segmentos mais evidentes incluem : carga em memória e salvamento em disco de objetos e cenas, remoção de um determinado objeto das estruturas de dados, troca dos nomes internos de identificação de objetos e cenas, inicialização das estruturas de dados, criação de primitivas, entre outras.

O **módulo de edição** contém as funções de transformação e manipulação de objetos e cenas. Pode-se dizer que este é o módulo que contém as funções de modelagem em si. O módulo oferece operações de transformação geométrica, tais como translação, rotação e escala. Outro recurso presente são as operações booleanas de união, interseção e subtração de objetos, que compõem as ferramentas CSG do sistema. O módulo oferece, ainda, algumas funções de apoio que auxiliam no processo de modelagem, tais como a operação de anexação de objetos, o posicionamento do observador e a exibição dos eixos do sistema de coordenadas universais.

O **módulo de visualização** reúne funções diversas para a visualização da cena construída. As operações presentes neste módulo especificam as características de exibição a serem empregadas na cena corrente, tais como controle de visibilidade/invisibilidade de objetos, perspectiva, exibição em arame, escondimento de faces ocultas, modelos de iluminação, entre outras.

O **módulo de configuração** constitui um dos módulos de apoio oferecidos pelo sistema. Através das funções deste módulo, o usuário pode modificar certas características de funcionamento visando a um maior conforto e adequação na utilização da interface homem-máquina do sistema. Inicialmente, este módulo foi desenvolvido contendo funções para configuração da tela de interface.

Outro módulo de apoio do sistema João-de-Barro é o **módulo de miscelâneas**. Nesse módulo, estão presentes alguns recursos da interface homem-máquina do sistema, visando a um maior conforto e flexibilidade para o usuário humano.

O sistema oferece, também, algumas funções chamadas de **funções especiais**, tais como abertura para a apresentação de instruções de uso do sistema, traçado de cenas e exibição em "zoom". Todos os módulos do sistema, bem como as funções especiais, são descritos em detalhe nas seções subseqüentes.

3.2.3. A Interface Homem-Máquina do Sistema

A tela de interface do sistema é dividida em quatro janelas : **janela de menus**, **janela do observador**, **janela de mensagens** e **janela de exibição**. Essa divisão proporciona boa distribuição das funções na tela, facilitando a leitura e a manipulação da interface do sistema.

A **janela de menus** é situada na porção superior direita da tela, onde são exibidos os menus do sistema. Os menus dão acesso às funções disponíveis em cada módulo. Nessa janela, é feita a manipulação do cursor sobre os menus, para a seleção de opções. Além das funções disponíveis, a janela de menus exibe um campo de identificação do menu corrente.

A **janela do observador**, situada na porção inferior direita da tela, exibe permanentemente dados sobre a posição do observador em relação à cena exibida. Os dados exibidos referem-se aos ângulos Azimute e Elevação (descritos no item 4.11) que definem o posicionamento do observador. Essa janela também é utilizada para a atualização da posição do observador.

A **janela de mensagens** situa-se na parte inferior esquerda da tela, funcionando como ponto de exibição de mensagens e outras informações ao usuário. A janela dispõe de três linhas distintas, consideradas suficientes para uma boa comunicação. Eventualmente, a janela de mensagens também é utilizada para solicitar dados ao usuário. A terceira linha da janela de mensagens é utilizada para indicar as teclas referentes às funções especiais (ver item 3.8).

A **janela de exibição** ocupa a maior parte da tela, situando-se na sua porção superior esquerda. É nessa janela que são exibidos os objetos e as cenas construídas. Eventualmente, essa janela é utilizada para apresentar outras informações e para solicitar dados ao usuário. A figura 3.3 apresenta a tela de interface do sistema João-de-Barro.

A interface do sistema opera de modo interativo, solicitando dados ao usuário e apresentando os resultados prontamente. A interface é relativamente auto-explicativa, facilitando a

utilização, principalmente para usuários não familiarizados com o sistema. A manipulação da interface é feita via teclado alfanumérico. Em expansões futuras, outros dispositivos, tais como mouse e mesa digitalizadora, poderão ser aceitos pelo sistema.

As diversas funções oferecidas pelo sistema são acessadas através de menus de opções. Para selecionar uma dada função de um menu, o usuário pode proceder de duas formas distintas :

a) Posicionar o cursor, através das teclas de direção <Up> e <Down> até que o mesmo se encontre sobre a opção desejada. Pressionar, então, a tecla <Enter>. O procedimento de movimentação do cursor exibe, na janela de mensagens, frases explicativas sobre cada opção percorrida.

b) Pressionar diretamente a tecla correspondente à letra maiúscula da opção desejada. Esse procedimento evita a manipulação do cursor e agiliza muito a utilização do sistema, sendo bastante útil principalmente para usuários mais experientes.

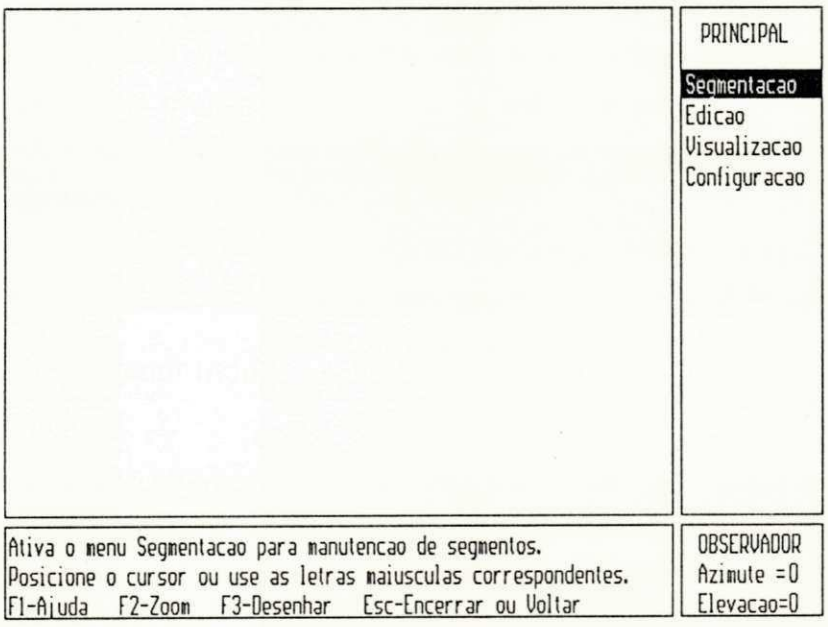


Figura 3.3 : Tela de interface do sistema.

As chamadas funções especiais do sistema são ativadas via teclas de função programável (F1 a F3). A tela de interface indica que teclas utilizar para cada função especial.

3.2.4. O Espaço de Referência

Por espaço de referência entende-se o conjunto de todos os pontos pertencentes ao espaço tridimensional abstrato oferecido pelo sistema para a construção da cena desejada. Por exemplo, um dado sistema pode oferecer, por convenção, um espaço de referência contido entre os pontos P1(X=-100, Y=-200, Z=-50) e P2(X=100, Y=200, Z=50). Isso significa que o sistema só admite a representação de pontos cujas coordenadas X variam de -100 a 100, cujas coordenadas Y variam de -200 a 200, e cujas coordenadas Z variam de -50 a 50. As coordenadas do ponto utilizado como origem do sistema de coordenadas também são convencionadas. Geralmente, utiliza-se o ponto (0, 0, 0) como origem do sistema de coordenadas.

No sistema João-de-Barro, o espaço de referência é bem amplo, variando entre os pontos 1.7E-308 a 1.7E+308 para cada eixo de coordenadas, de acordo com o escopo das variáveis do tipo DOUBLE da linguagem C. A origem do sistema de coordenadas é situada no ponto (0, 0, 0) e se encontra sempre no centro da janela de exibição. Apesar da amplitude do espaço de referência, a área de visualização das cenas é fixa, restringindo-se aos pontos ao redor da origem do sistema de coordenadas presentes na janela de exibição. A razão de se haver definido um amplo espaço de referência reside na previsão de expansões futuras do sistema que possam vir a exigir tal capacidade.

3.3. O Menu de Segmentação

Conforme mencionado anteriormente, o menu de segmentação é responsável por agrupar operações destinadas à criação e manutenção de segmentos. Dentre essas operações, incluem-se a inicialização das estruturas de dados do sistema, a criação de primitivas geométricas, a carga de objetos em memória, o salvamento de objetos na forma de arquivos em disco, a carga e o salvamento de toda uma cena previamente construída, a troca do nome de identificação de objetos e cenas, entre outras afins. A figura 3.4 apresenta o menu de segmentação. As funções presentes nesse menu são detalhadas nos itens subseqüentes.

3.3.1. A Função "Inicializar"

A função "Inicializar" é identificada pelo item **Inicializar** do menu de segmentação. O objetivo desta função é inicializar todas as estruturas de dados e variáveis internas do sistema. São eliminados todos os segmentos presentes na memória, além de se estabelecerem valores-padrão para variáveis, tais como ângulos de posicionamento do observador, exibição dos eixos de coordenadas, modo de exibição da cena, distância do plano de projeção etc.

Ao ser ativada, a função exibe, na janela de mensagens, frases alertando o usuário quanto à perda do atual conteúdo da memória, e solicita confirmação para prosseguimento. Ao término da função, o menu de segmentação reassume o controle do sistema.

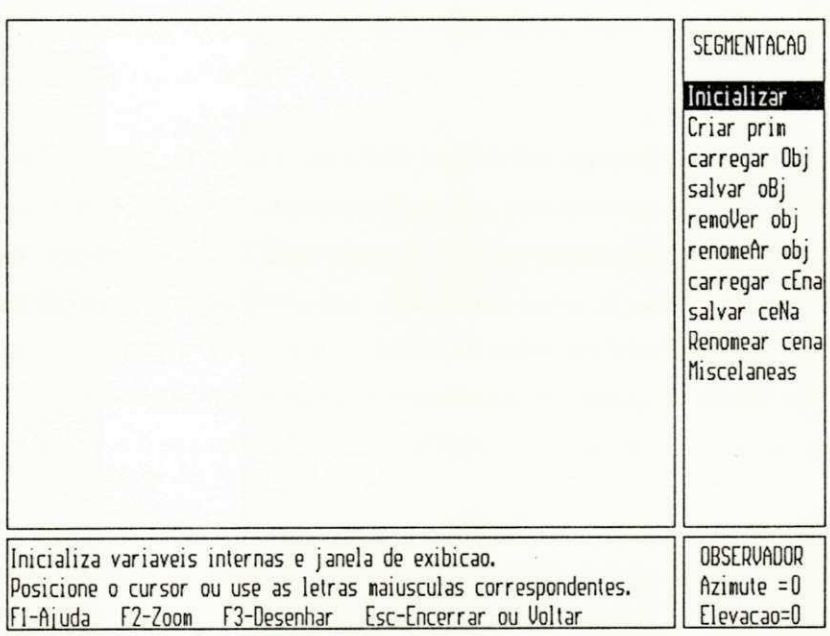


Figura 3.4 : O menu de segmentação.

3.3.2. A Função "Criar Primitiva"

No menu de segmentação, a função "Criar Primitiva" é identificada pelo item **Criar prim**. O objetivo desta função é oferecer métodos para a criação de primitivas geométricas. Como dito no item 3.2.1, uma primitiva é um modelo sólido que passa por transformações e/ou

combinações com outras primitivas para gerar modelos mais complexos. A função "Criar Primitiva" dispõe de métodos para a criação de primitivas básicas para serem usadas como ponto de partida na construção de modelos sólidos.

Ao ser ativada, a função exibe um menu contendo todos os métodos oferecidos pelo sistema para a criação de primitivas. Atualmente, o sistema dispõe de dois métodos para este fim : o método **numérico** e o método **poliédrico**.

O **método numérico** constitui um método trabalhoso para o usuário porém flexível. O seu funcionamento ocorre da seguinte forma : internamente, os modelos sólidos são armazenados no sistema de acordo com a representação Boundary Representation, ou seja, através de suas faces, arestas e vértices. A interligação de dois vértices forma um aresta, e várias arestas interligadas compõem uma face. Utilizando o método numérico, o usuário deve informar, explicitamente, o nome da primitiva a ser criada, o número de faces da primitiva, as arestas componentes de cada face, os vértices componentes de cada aresta, e as coordenadas X, Y e Z de cada vértice. O método funciona interativamente, conduzindo o usuário para que informe gradativamente os dados necessários.

A utilização do método numérico implica em algumas dificuldades para o usuário. Em primeiro lugar, é necessário que o usuário disponha, previamente, de um projeto que descreva a primitiva a ser construída e que especifique seus elementos (faces, arestas e vértices). Em modelos com elevado número de elementos, essa tarefa torna-se extremamente trabalhosa. Em segundo lugar, a própria tarefa de entrada de dados no sistema é relativamente longa e monótona, dependendo da complexidade da primitiva a ser criada. A vantagem desse método reside na flexibilidade, uma vez que o usuário se encontra livre para especificar a geometria e a topologia das primitivas.

O **método poliédrico** representa um meio mais automático de se criarem primitivas, demandando menos trabalho do usuário. O método atende à criação de três classes de primitivas : primitivas **cilíndricas**, **cônicas** e **esféricas**. O usuário informa o nome da primitiva, a classe da primitiva desejada (cilíndrica, cônica ou esférica), o número de vértices da base (no caso de primitivas cilíndricas e cônicas) ou o número de faces por hemisfério (no caso de primitivas esféricas). A função cria automaticamente a primitiva solicitada.

O método poliédrico é bastante confortável para o usuário e não muito oneroso computacionalmente. Sua restrição está no fato de se poder criar somente uma das três classes de primitivas mencionadas anteriormente.

3.3.3. A Função "Carregar Objeto"

A função "Carregar Objeto" é indicada no menu de segmentação através do item **carregar Obj**. Seu objetivo é bastante simples : carregar em memória um determinado objeto salvo em disco. Ao ser ativada, a função exibe, na janela de exibição, uma relação com todos os objetos disponíveis. O usuário é solicitado a informar o nome do objeto desejado. Feito isso, a operação de carga é executada.

3.3.4. A Função "Salvar Objeto"

A função "Salvar Objeto" é identificada no menu de segmentação pelo item **salvar oBj**. O objetivo desta função é salvar, em disco, os dados de um determinado objeto carregado em memória. Quando ativada, a função exibe, na janela de exibição, a relação de todos os objetos correntemente carregados em memória. O usuário é solicitado a indicar qual o objeto a ser salvo e o nome do arquivo a ser gravado no disco. Caso já exista um arquivo com nome idêntico ao informado, o sistema alerta o usuário quanto à possível superposição dos arquivos, antes de prosseguir com o salvamento.

3.3.5. A Função "Remover Objeto"

No menu de segmentação, a função "Remover Objeto" é indicada pelo item **remoVer obj**. A função objetiva retirar da memória um determinado objeto, eliminando o mesmo da cena corrente. Quando ativada, a função exibe a relação de todos os objetos carregados em memória e solicita ao usuário que informe o nome do objeto a ser removido. Feito isso, é efetuada a operação de remoção.

3.3.6. A Função "Renomear Objeto"

A função "Renomear Objeto" é identificada através do item **renomeAr obj**. O efeito desta função é substituir o nome de identificação de um determinado objeto carregado em

memória. Quando solicitada, a função exibe, na janela de mensagens, uma relação contendo todos os objetos correntemente carregados em memória. O usuário é solicitado, então, a informar qual o objeto a ser renomeado e o novo nome a ser atribuído a ele. É feita, então, a troca do nome do referido objeto.

3.3.7. A Função "Carregar Cena"

O conjunto de objetos correntemente carregados em memória, juntamente com outras variáveis do sistema, compõem uma cena. A função "Carregar Cena", indicada pelo item **carregar cEna**, carrega em memória uma determinada cena salva em disco. O sistema permite a carga de somente uma cena de cada vez, conseqüentemente a carga de uma cena sobrepõe uma determinada cena pré-existente na memória do computador. Quando solicitada, a função exibe, na janela de exibição, uma relação com todos os arquivos de cenas disponíveis e pede ao usuário que informe o nome do arquivo a ser lido. Antes de prrosseguir com a operação de carga, o sistema alerta o usuário quanto à perda da cena corrente. A figura 3.5 ilustra a tela de interface do sistema, no momento em que é solicitada uma operação de carga de uma cena.

3.3.8. A Função "Salvar Cena"

A função "Salvar Cena", identificada pelo item **salvar ceNa** do menu de segmentação, salva em disco a cena correntemente presente em memória. Quando ativada, a função pede ao usuário que informe o nome do arquivo a ser gravado, alertando-o no caso de uma possível destruição de um arquivo existente devido à repetição do nome de arquivo informado. Feito isso, é efetuado o salvamento da cena.

3.3.9. A Função "Renomear Cena"

A função "Renomear Cena" é identificada no menu de segmentação através do item **Renomear cena**. O objetivo desta função é trocar o nome de identificação da cena corrente. A função solicita ao usuário que informe o novo nome desejado. A cena corrente passa, então, a ser identificada pelo novo nome informado.

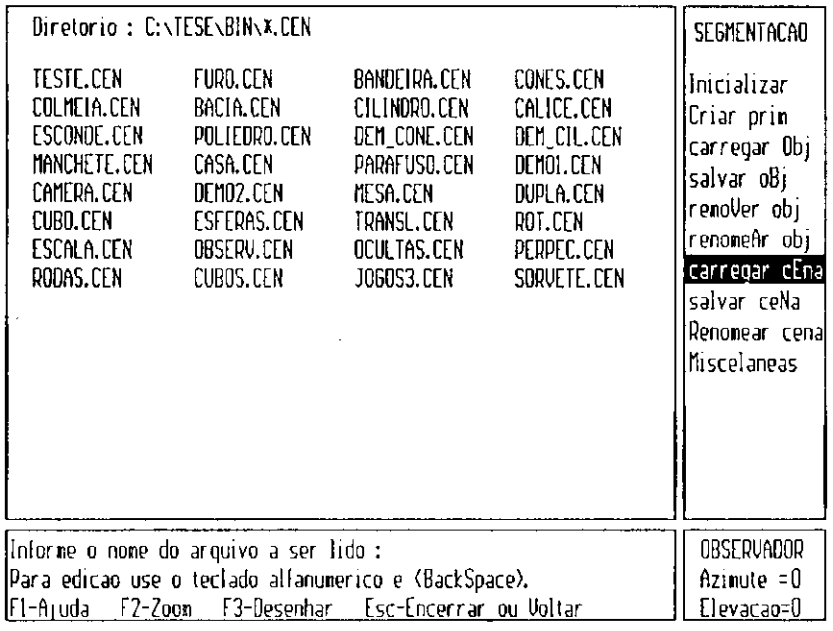


Figura 3.5 : Operação de carga de uma cena em memória.

3.3.10. A Função "Miscelâneas"

Esta função se encontra no menu de segmentação e em outros menus do sistema. Ela é sempre identificada pelo item **Miscelâneas**. Seu objetivo é dar acesso às funções do menu de miscelâneas, descrito no item 3.7.

3.4. O Menu de Edição

De acordo com o item 3.2.2, o menu de edição agrupa funções para manipulação e transformação dos modelos, sendo o menu responsável pelas tarefas de modelagem em si. Dentre as funções incluídas neste menu, destacam-se as operações de transformação geométrica como translação, rotação e escala, as operação booleanas de união, interseção e subtração, funções para o posicionamento do observador, a operação de anexação de objetos, além de algumas funções de apoio. A figura 3.6 apresenta o menu de edição.

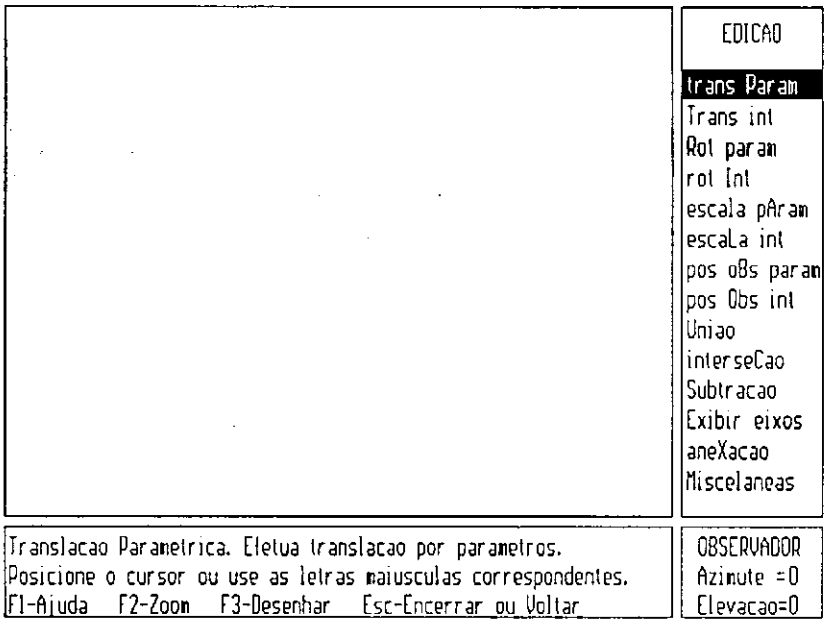


Figura 3.6 : O menu de edição.

3.4.1. A Função "Translação Paramétrica"

A função "Translação Paramétrica" é identificada pelo item **trans Param** do menu de edição. Esta função corresponte a uma das transformações geométricas oferecidas pelo sistema (ver item 2.6), realizando operações de translação de objetos. A função permite que um dado objeto presente na cena em construção seja deslocado ao longo dos eixos X, Y e Z.

Quando ativada, a função exibe, na janela de exibição, a relação de todos os objetos correntemente carregados em memória. O usuário é solicitado, então, a informar o nome do objeto a ser transladado. Feito isso, o sistema pede ao usuário que informe os parâmetros de translação TX, TY e TZ, que correspondem, respectivamente, aos deslocamentos ao longo dos eixos X, Y e Z. Os deslocamentos são especificados em termos de número de pontos do espaço de referência (ver item 3.2.4).

3.4.2. A Função "Translação Interativa"

No menu de edição, a função "Translação Interativa" é ativada através do item **Trans int**. Esta função consiste numa forma alternativa de se realizarem operações de translação.

Ao invés de efetuar a translação por parâmetros explícitos, como na função "Translação Paramétrica", a função "Translação Interativa" permite que o usuário desloque um dado objeto interativamente, até que este atinja a posição desejada.

Uma vez ativada, a função apresenta uma relação contendo todos os objetos presentes na memória, e pede ao usuário que informe o nome do objeto a ser transladado. Feito isso, o sistema reexibe a cena em construção e apresenta, na janela de mensagens, uma linha com informações sobre a posição do objeto selecionado e o eixo correntemente ativo ao longo do qual será feito o deslocamento. A posição do objeto é expressa pelas coordenadas do seu baricentro. O usuário passa, então, a proceder da seguinte forma : as teclas de direção <Right> e <Left> significam, repectivamente, incremento e decremento. Acionando-se a tecla <Right>, para incremento, o objeto é transladado ao longo do eixo ativo, percorrendo a distância equivalente a um ponto em sentido positivo. Numa operação de decremento, o objeto é deslocado de um ponto em sentido negativo. A cada incremento/decremento, a cena é atualizada, exibindo o objeto em sua nova posição. Para se modificar o eixo ativo, basta pressionar a tecla <X>, <Y> ou <Z> de acordo com o novo eixo desejado. A partir daí, os deslocamentos serão efetuados ao longo deste novo eixo. A tecla <Esc> suspende este processo, e retorna o controle ao menu de edição.

3.4.3. A Função "Rotação Paramétrica"

Esta função é identificada pelo item **Rot param**, presente no menu de edição. O objetivo desta função é permitir a realização de outro tipo de transformação geométrica : a operação de rotação de objetos. A operação de rotação faz com que objetos sejam rotacionados em torno dos eixos X, Y e Z. A posição dos objetos é mantida, altera-se apenas a sua orientação.

Ao ser ativada, a função exibe uma relação listando os objetos carregados em memória no momento. O usuário é, então, solicitado a informar o nome do objeto a ser rotacionado. Feito isso, a função pede ao usuário que informe os parâmetros de rotação RX, RY e RZ. Esses parâmetros correspondem aos ângulos de rotação em torno de cada um dos eixos X, Y e Z. Os ângulos são expressos em graus, e podem assumir valores positivos e negativos. Os sentidos de rotação obedecem a regra da mão direita (ver [Fole82] pág. 256, [Park85] pág. 133, [Berg86] pág. 252 e [Hearn86] pág. 182). A partir daí, é efetuada a operação de rotação.

3.4.4. A Função "Rotação Interativa"

Esta função é indicada através do item **rot Int** presente no menu de edição. Analogamente à função "Translação Interativa", a função "Rotação Interativa" consiste numa forma alternativa de se realizar uma operação de rotação. O efeito causado é o mesmo da operação de rotação paramétrica, variando somente a forma de interação com o usuário. O objetivo desta função é permitir que o usuário rotacione um dado objeto interativamente, acompanhando a rotação passo a passo.

Quando ativada, a função exibe uma relação informando quais os objetos correntemente carregados em memória. O sistema pede ao usuário que indique o nome do objeto a ser rotacionado. Feito isso, a cena em construção é projetada na janela de exibição, enquanto que a janela de mensagens exibe uma linha de informações para o acompanhamento da operação. Essa linha informa o eixo ativo, que corresponde ao eixo (X, Y ou Z), em torno do qual se está rotacionando o objeto, e as coordenadas do baricentro do objeto, exibidas para fins de orientação. O eixo ativo pode ser atualizado pressionando-se a tecla correspondente ao novo eixo desejado, X, Y ou Z. A rotação prossegue da seguinte forma : as teclas de direção <Right> e <Left> significam, respectivamente, incremento e decremento. Um incremento rotaciona o objeto de um grau positivo em torno do eixo ativo. Um decremento rotaciona o objeto de um grau negativo. A cada incremento/decremento, a cena é atualizada com o objeto na sua nova posição. Os sentidos de rotação obedecem à regra da mão direita (ver [Fole82] pág. 256, [Park85] pág. 133, [Berg86] pág. 252 e [Hearn86] pág. 182). A figura 3.7 ilustra a tela de interface do sistema no momento em que se realiza uma operação de rotação interativa.

3.4.5. A Função "Escala Paramétrica"

Esta função é identificada pelo item **escala pAram** presente no menu de edição. O objetivo desta função é permitir a realização de operações de escala de objetos (ver item 2.6). As operações de escala alteram as dimensões e as proporções do objetos.

Quando ativada, a função "Escala Paramétrica" exibe uma relação contendo todos os objetos correntemente carregados em memória. Após isso, o sistema pede ao usuário que indique o objeto a ser escalonado e que informe os parâmetros SX, SY e SZ a serem utilizados na operação. Esses parâmetros significam, respectivamente, os valores de escala ao longo dos eixos X, Y e Z. Um valor para SX igual a 2 (dois) significa que o objeto terá suas dimensões duplicadas

ao longo do eixo X. Um valor SX igual a 0,5 (meio) indica que as dimensões do objeto ao longo do eixo X serão reduzidas pela metade. Valores de escala iguais a 1 (um) mantêm as mesmas dimensões correntes.

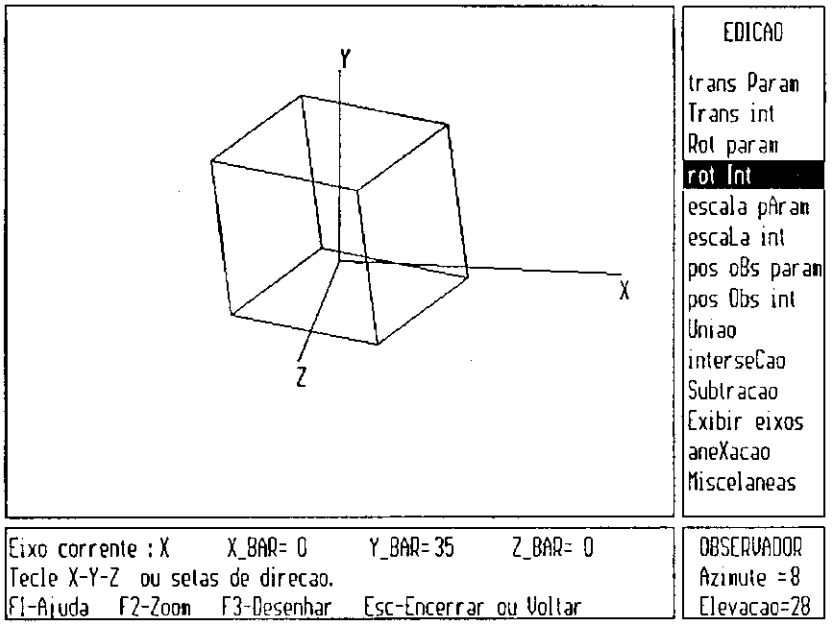


Figura 3.7 : Tela de interface ilustrando operação de rotação interativa.

3.4.6. A Função "Escala Interativa"

Esta função é representada pelo item **escaLa int**. Seu objetivo é oferecer uma forma alternativa de se realizarem operações de escala, tornando a operação mais confortável, de acordo com a conveniência do usuário. Com esta função, as operações de escala podem ser realizadas interativamente.

O funcionamento da operação é análogo às demais operações interativas descritas anteriormente. Inicialmente, a função exibe a relação de objetos e solicita ao usuário que informe aquele a ser escalonado. Feito isso o usuário utiliza as teclas <Right>, <Left>, <X>, <Y> e <Z> para alterar as dimensões do objeto ao longo de cada um dos eixos X, Y e Z.

3.4.7. A Função "Posicionar Observador Paramétrica"

O item **pos oBs param** identifica a função "Posicionar Observador Paramétrica". Esta função oferece um recurso bastante importante, permitindo que a cena em construção seja observada de diferentes posições do espaço (ver item 2.7).

No sistema João-de-Barro, o observador "olha" sempre para a origem do sistema de coordenadas universais, que, por sua vez é exibida sempre no centro da janela de exibição. O observador pode ser deslocar "girando" em torno do eixo Y e do eixo X. O deslocamento em torno do eixo Y é definido pelo chamado "ângulo azimute", enquanto que o deslocamento em torno do eixo X é definido pelo "ângulo de elevação". Os ângulos azimute e elevação são expressos em graus, podendo assumir valores positivos e negativos.

A janela do observador, presente no canto inferior direito da tela de interface, exibe permanentemente os ângulos azimute e elevação do observador da cena. Ao ser ativada, a função "Posicionar Observador Paramétrica" solicita ao usuário que informe os novos valores dos referidos ângulos, utilizando, para isso, a própria janela do observador. As exibições subseqüentes da cena considerarão os novos valores dos ângulos azimute e elevação.

3.4.8. A Função "Posicionar Observador Interativa"

Esta é mais uma função presente no menu de edição, identificada pelo item **pos Obs int**. O objetivo desta função é oferecer um meio interativo para se efetuarem operações de mudança na posição do observador.

O funcionamento desta função segue o padrão estipulado para as operações interativas do sistema João-de-Barro. Ao ser solicitada, a função exibe, na janela de mensagens, uma linha de informações indicando qual o ângulo, azimute ou elevação, correntemente em atualização. O usuário dispõe das teclas de direção <Right> e <Left> para incrementar ou decrementar em 1 (um) grau o ângulo ativo. As teclas <A> e <E> ativam, respectivamente, os ângulos azimute e elevação, para atualização. A cada incremento/decremento, a cena é atualizada de acordo com a nova posição do observador.

3.4.9. A Função "União"

A função "União" é identificada pelo item **Uniao** do menu de edição. Esta é uma das funções oferecidas pelo sistema João-de-Barro que realizam operações booleanas e compõem o módulo CSG do sistema. Como o próprio nome indica, a função aqui descrita realiza operações de união de dois objetos.

Uma vez ativada, a função "União" exibe, na janela de exibição, a relação dos objetos carregados em memória e solicita ao usuário que informe os dois objetos que participarão da operação. O usuário é solicitado, também, a informar o nome a ser atribuído ao novo objeto resultante da operação. Por demandar tempo de execução, ao longo da operação são apresentadas mensagens informando o usuário quanto às etapas percorridas para a efetuação da união. Após a operação, os dois objetos originais passam a não existir mais, restando somente o objeto final da operação.

3.4.10. A Função "Interseção"

A função "Interseção", identificada pelo item **interseCao**, consiste em outra operação booleana oferecida pelo sistema. A operação envolve sempre dois objetos distintos, e resulta em um novo objeto que consiste somente nas porções comuns aos dois objetos originais.

O funcionamento desta função é análogo à função "União". Inicialmente, o usuário é solicitado a informar os dois objetos que participarão da operação, bem como o nome a ser atribuído ao novo objeto resultante. Mensagens apresentadas na janela de mensagens informam o usuário quanto ao prosseguimento da operação. Os objetos originais são perdidos, restando somente o objeto resultante da operação.

3.4.11. A Função "Subtração"

Esta função é indicada pelo item **Subtracao**, presente no menu de edição, e constitui mais uma operação booleana oferecida pelo sistema. O efeito de uma operação de subtração é extrair de um objeto, todas as porções que esse tiver em comum com o outro objeto

envolvido na operação. Por exemplo, podemos "furar" um objeto, subtraindo-se um cilindro do mesmo.

Esta função tem o mesmo funcionamento das demais operações booleanas presentes no sistema. Inicialmente, a função solicita ao usuário que indique os objetos participantes da operação e que informe o nome a ser atribuído ao objeto resultante. Vale ressaltar que a ordem na qual os objetos são informados interfere no resultado da operação. Numa subtração, o objeto final consiste no primeiro objeto menos as porções comuns ao segundo objeto. Ao longo da execução, o usuário recebe mensagens informativas quanto à ao prosseguimento da operação. Após a operação, os objetos originais são destruídos, restando somente o objeto final.

3.4.12. A Função "Exibir Eixos"

A função "Exibir Eixos" é indicada pelo item **Exibir eixos** do menu de edição. Esta função consiste num recurso da interface do sistema, desenvolvido por questões de conforto ao usuário. A função controla a exibição dos eixos X, Y e Z do sistema de coordenadas universais, para fins de orientação do usuário em relação à cena em construção.

O funcionamento desta função é bastante simples. Quando os eixos do sistema de coordenadas não estão sendo exibidos, o acionamento da função causa a exibição desses eixos. A partir daí, os eixos serão sempre exibidos. O novo acionamento da função suspende a exibição dos eixos. A requisição desta função provoca a atualização imediata da cena em exibição. A figura 3.8 ilustra uma cena com os eixos do sistema universal de coordenadas.

3.4.13. A Função "Anexação"

A função "Anexação" é indicada pelo item **aneXacao** do menu de edição, e tem como papel fundir dois objetos inicialmente distintos resultando em um único objeto final. O funcionamento dessa operação consiste simplesmente em fundir as estruturas de dados dos dois objetos, atribuindo as faces, arestas e vértices dos objetos envolvidos a um único objeto final.

É importante frisar que o papel e o objetivo dessa operação diferem daqueles propostos para a operação booleana de união. Nesta última, as porções desnecessárias não são mantidas no objeto final (ex: porções internas ao objeto), além de se preservarem as propriedades formais essenciais à consistência do objeto.

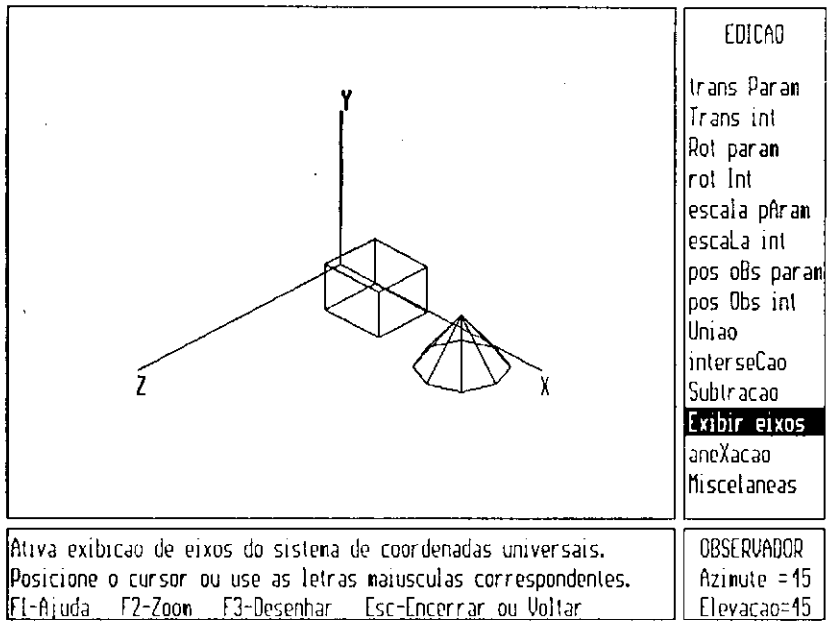


Figura 3.8 : Exibição de cena com eixos do sistema de coordenadas universais.

Por sua vez, a operação de anexação pode ser usada indiscriminadamente para fundir quaisquer objetos. Entretanto, o usuário deve estar atento para a possibilidade de se criarem objetos inadequados para certas operações do sistema, tais como as operações booleanas e as operações de iluminação.

Ao ser ativada, a função "anexação" solicita ao usuário que informe o nome de cada um dos objetos a serem fundidos. Feito isso, o usuário é ainda solicitado a informar qual o nome de identificação a ser atribuído ao objeto final. Informados todos os dados, a função prossegue com a fusão das estruturas de dados dos objetos, retornando o controle do sistema ao menu de edição. A partir desse ponto, o usuário não conta mais com os objetos iniciais, mas somente com o objeto resultante da operação.

3.4.14. A Função "Miscelâneas"

Esta função se encontra no menu de edição e em outros menus do sistema. Ela é sempre identificada pelo item **Miscelâneas**. Seu objetivo é dar acesso às funções do menu miscelâneas, descrito no item 3.7.

3.5. O Menu de Visualização

O menu de visualização reúne funções destinadas ao emprego de efeitos visuais que dêem maior realismo às cenas construídas. Os efeitos implementados no sistema João-de-Barro incluem controle de visibilidade de objetos, mudança de cores de objetos, exibição em perspectiva, exibição em arame, exibição com escondimento de faces ocultas e exibição com iluminação segundo o modelo "flat-shading". O modelo de iluminação conhecido como "flat-shading" é descrito em detalhe no item 4.18.

Os efeitos disponíveis neste menu somente são exibidos quando o menu de visualização estiver como o menu corrente do sistema, ou quando se solicitar uma exibição com "zoom" (ver item 3.8.2). Nos demais casos, as cenas são exibidas em arame. A figura 3.9 apresenta o menu de visualização.

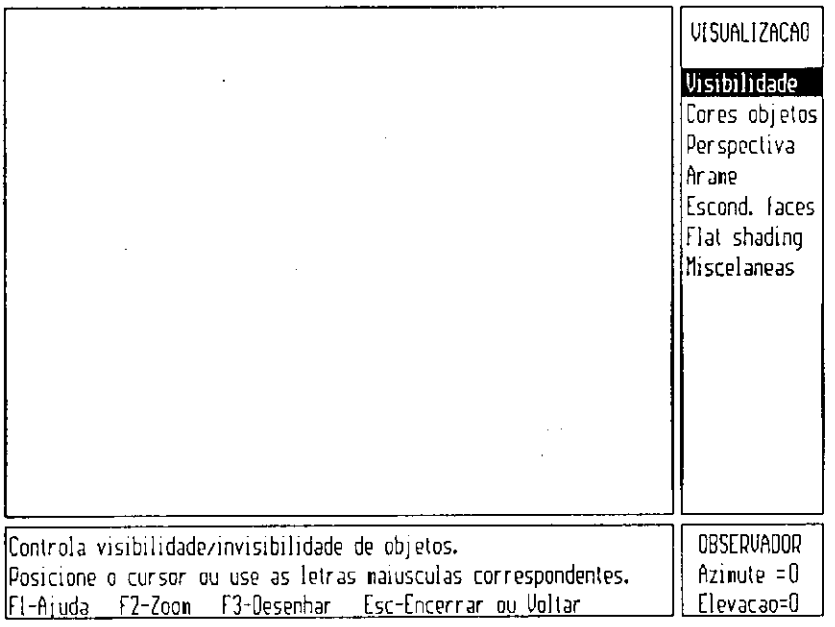


Figura 3.9 : O menu de visualização.

3.5.1. A Função Visibilidade

A função "Visibilidade" é indicada pelo item **Visibilidade** do menu de visualização. Esta função permite que determinados objetos, apesar de integrantes da cena, não sejam exibidos.

Quando requisitada, a função exibe uma relação contendo todos os objetos correntemente carregados em memória, e solicita ao usuário que informe qual o objeto a ter sua visibilidade alterada. Caso o objeto informado esteja visível no momento da operação, o mesmo se torna invisível. Caso o objeto selecionado esteja invisível, o mesmo torna-se visível.

3.5.2. A Função "Cores de Objetos"

A função "cores de objetos" é identificada através do item **Cores objetos**, e tem como objetivo alterar a cor matiz dos objetos correntemente carregados em memória.

Quando ativada, a função exibe a lista de todos os objetos disponíveis na memória, e solicita ao usuário que informe aquele desejado. Feito isso, a função exibe a palheta com as cores disponíveis, apresentando cada cor com o seu respectivo código numérico. O usuário é solicitado, então, a informar o código da nova cor escolhida para o objeto selecionado. Após a operação, o objeto é automáticamente exibido com a nova cor matiz.

3.5.3. A Função "Perspectiva"

A função "Perspectiva", disponível no menu de visualização, é identificada pelo item **Perspectiva**. Seu objetivo é permitir a exibição dos objetos pertencentes à cena, usando efeitos de perspectiva e profundidade (ver item 2.7).

Caso a opção de exibição em perspectiva esteja destivada, a função solicita ao usuário que informe a distância desejada para o plano de projeção da perspectiva. A partir daí, está ativada a opção de perspectiva. O novo acionamento da função desativa o modo de exibição em perspectiva.

Vale ressaltar que esta opção só é exibida com o menu de visualização ativo.

3.5.4. A Função "Arame"

A função "Arame", identificada pelo item **Arame**, é responsável pelo controle da exibição da cena corrente, utilizando o modo "arame", também conhecido como **wire-frame**. Quando ativada, esta função estabelece que a cena carregada em memória será exibida em

arame. A função exibe uma mensagem informando o usuário quanto à concretização da nova opção. O estabelecimento dessa opção desativa as opções "escondimento de faces ocultas" e "flat-shading", caso uma delas tenha sido previamente selecionada.

3.5.5. A Função "Escondimento de Faces Ocultas"

Identificada pelo item **Escond. faces**, a função "Escondimento de Faces Ocultas" é responsável por controlar a opção de exibição de cenas eliminando-se as porções de faces que porventura estejam cobertas por outras porções de faces. Quando ativada, a função exibe uma mensagem informando o usuário sobre a nova característica e estabelece que as exibições subseqüentes utilizarão o efeito de escondimento de faces, substituindo qualquer seleção prévia das opções "arame" ou "flat-shading".

3.5.6. A Função "Flat Shading"

A função "Flat Shading" é identificada pelo item **Flat Shading** do menu de visualização. Esta função oferece a possibilidade de se aplicar um modelo de iluminação ao se exibir a cena construída. Como o próprio nome indica, é oferecido o modelo de iluminação conhecido como "flat shading" ("constant shading" em algumas referências da literatura), descrito em detalhes no item 4.18.

Uma vez acionada, a função ativa o modo de iluminação, fazendo com que as exibições subseqüentes da cena sejam iluminadas de acordo com o modelo de iluminação oferecido. O acionamento desta função suprime qualquer seleção prévia das opções "arame" e "flat-shading".

3.5.7. A Função "Miscelâneas"

Esta função se encontra no menu de visualização e em outros menus do sistema. Ela é sempre identificada pelo item **Miscelâneas**. Seu objetivo é dar acesso às funções do menu miscelâneas, descrito no item 3.7.

3.6. O Menu de "Configuração"

Conforme mencionado no item 3.2.2, o menu de configuração agrupa funções destinadas ao estabelecimento de determinadas características do funcionamento do sistema. Atualmente, estão disponíveis duas funções destinadas à configuração da interface. A figura 3.10 apresenta o menu de configuração.

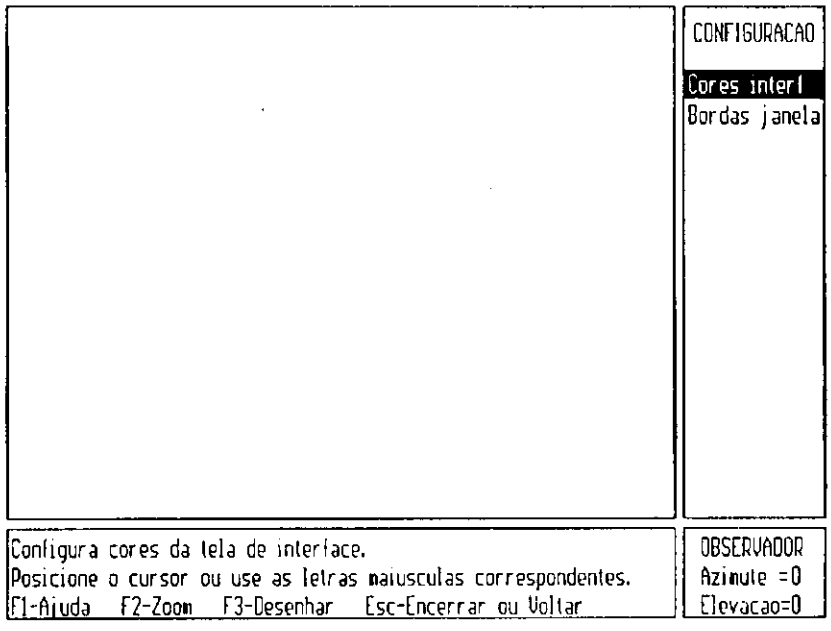


Figura 3.10 : O menu de configuração

3.6.1. A Função "Alterar Cores da Interface"

A função "Alterar Cores da Interface", presente no menu de configuração, é identificada pelo item **Cores interf**. Seu objetivo é permitir que o usuário altere as cores presentes na tela de interface do sistema, visando ao maior conforto e à melhor adaptação a diferentes monitores de vídeo.

Quando requisitada, a função solicita ao usuário que informe as novas cores de fundo e de escrita a serem atribuídas às janelas de menus, do observador e de mensagens. As cores disponíveis são apresentadas juntamente com seus respectivos códigos numéricos, devendo

o usuário informar o código desejado. Após a entrada dos dados solicitados, a interface do sistema é exibida em suas novas cores.

3.6.2. A Função "Exibir Bordas da Janela"

Esta função é identificada pelo item **Bordas janela** do menu de configuração. Esta é mais uma opção que permite a adaptação da interface do sistema à conveniência do usuário. A função permite que o usuário opte entre a exibição ou não das bordas que envolvem a janela de exibição na tela de interface do sistema.

Caso as bordas estejam sendo exibidas, a ativação desta função elimina as bordas da tela de interface. Caso contrário, a inerface é prontamente atualizada com a apresentação das bordas.

3.7. O Menu de Miscelâneas

O acesso ao menu de miscelâneas encontra-se em diversos pontos do sistema, sempre a partir de alguns dos menus apresentados anteriormente. A função deste menu é agrupar ferramentas de propósito geral, úteis ao usuário em pontos distintos do sistema. Dentre as funções disponíveis neste menu, destacam-se a apresentação das opções correntemente ativas no sistema, a listagem dos objetos presentes na memória, a identificação dos objetos exibidos, a impressão das estruturas de dados principais, a exibição da palheta de cores em utilização e a identificação dos vértices dos objetos. A figura 3.11 apresenta o menu de miscelâneas.

3.7.1. A Função "Opções Ativas"

A função "Opções Ativas" é associada ao item **Opcoes ativas** do menu de miscelâneas. Seu papel é apresentar o estado corrente de algumas opções de funcionamento do sistema configuráveis pelo usuário. Algumas dessas opções são cores da tela da interface, presença das bordas da janela de exibição, presença de eixos de coordenadas, a distância do plano de projeção utilizada para cálculo de perspectiva, o modo atual de operação do vídeo, entre outras.

Quando ativada, a função exibe as opções correntes na janela de exibição. Após consultá-las, o usuário deve acionar qualquer tecla para suspender a operação e restaurar a tela original. A figura 3.12 apresenta a tela de interface exibindo as opções correntemente ativas.

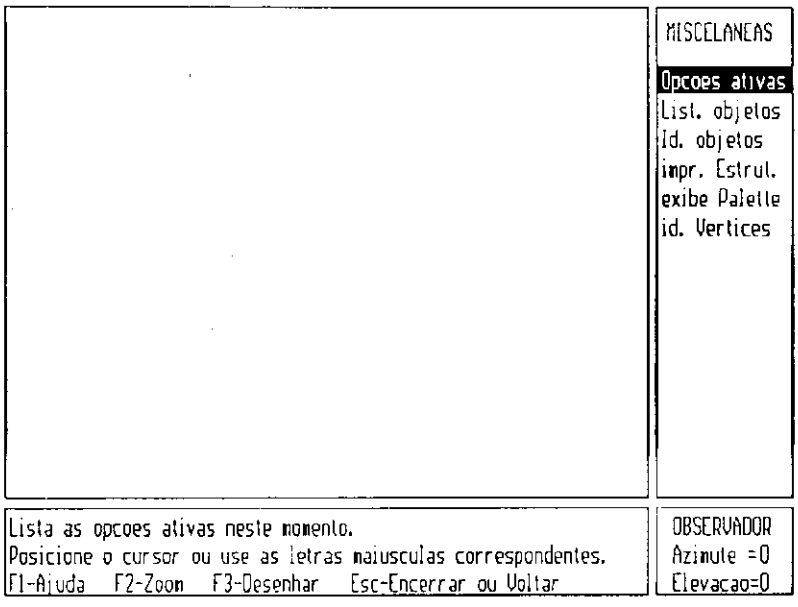


Figura 3.11 : O menu de miscelâneas.

3.7.2. A Função "Listar Objetos"

A função "Listar Objetos" é identificada pelo item **List. Objetos** do menu de miscelâneas. O objetivo desta função é apresentar ao usuário uma relação contendo todos os objetos correntemente carregados em memória. Para cada objeto, essa relação fornece dados como o número de identificação, o nome correntemente atribuído ao objeto, a visibilidade e as coordenadas do baricentro.

Quando ativada, a função utiliza a janela de exibição para apresentar a relação dos objetos. Mediante a apresentação da lista de objetos, o usuário dispõe de mecanismos para rolamento da tela. Após o término da operação, a cena original é restaurada na janela de exibição. A figura 3.13 ilustra a relação de objetos carregados apresentada pela função "Listar Objetos".

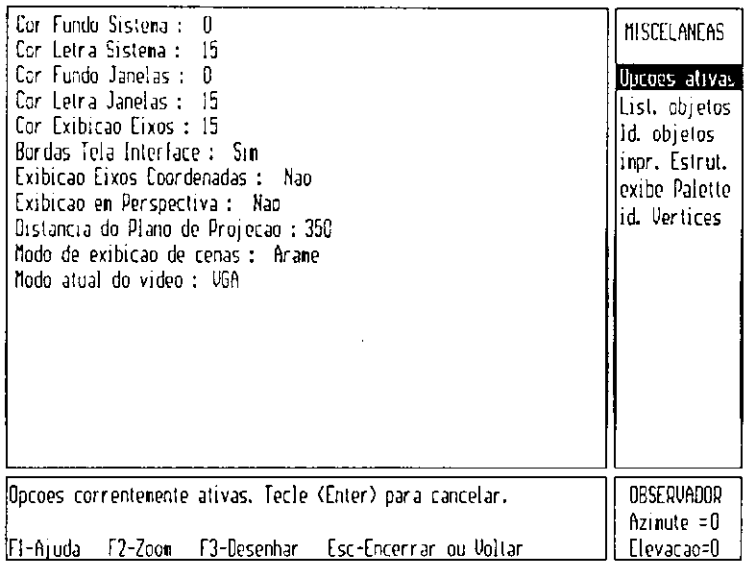


Figura 3.12 : Apresentação das opções ativas para as variáveis do sistema.

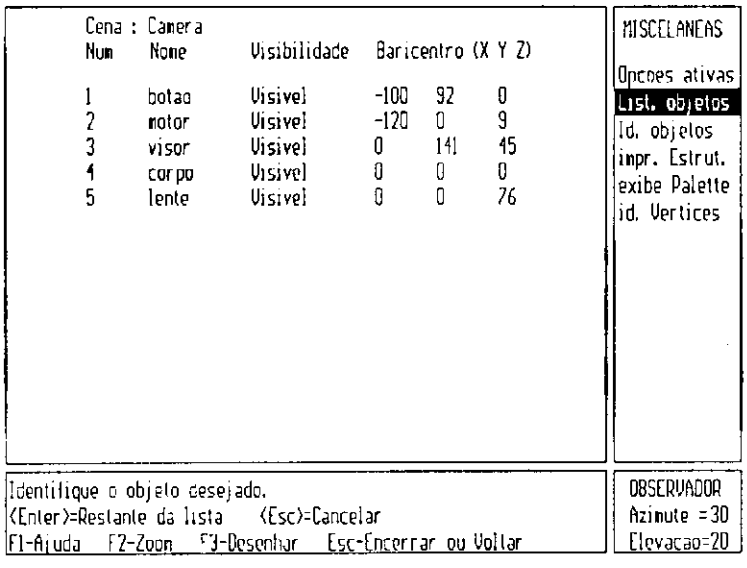


Figura 3.13 : Relação dos objetos correntemente carregados em memória.

3.7.3. A Função "Identificar Objetos"

No menu de miscelâneas, a função "Identificar Objetos" é associada ao item **Id. objetos**. O objetivo desta função é informar ao usuário o número de identificação de cada objeto presente na cena corrente.

Ao ser ativada, a função exibe, sobre cada objeto, o número de identificação de cada um. Esse número faz parte das relações de objetos apresentadas eventualmente pelo sistema. Uma vez identificados os objetos, o acionamento de qualquer tecla suspende a operação, voltando o sistema ao seu funcionamento normal. A figura 3.14 ilustra o efeito da função "Identificar Objetos".

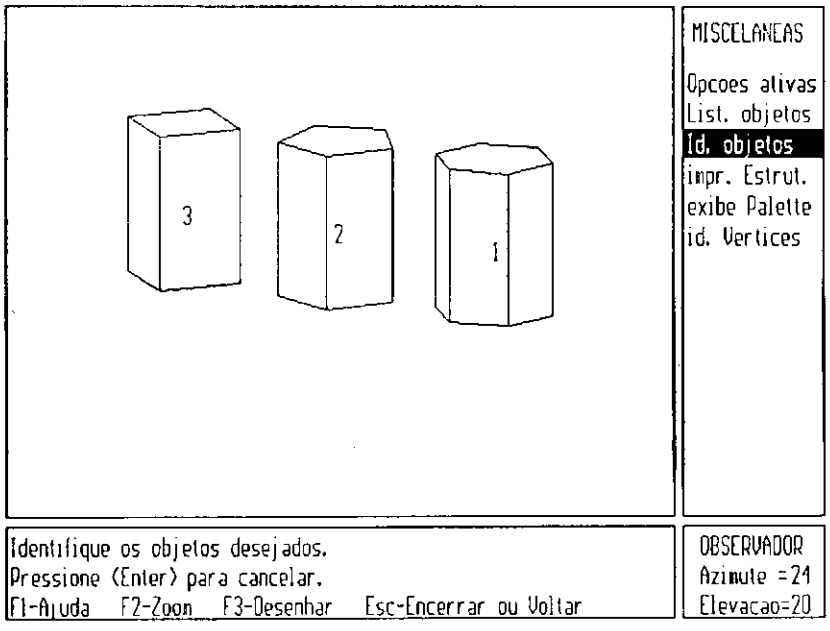


Figura 3.14 : Apresentação do número de identificação de objetos.

3.7.4. A Função "Imprimir Estruturas de Dados"

Esta função é associada ao item **impr. Estrut.** do menu de miscelâneas. O papel desta função é emitir um relatório listando o conteúdo corrente das principais estruturas de dados do sistema. Esta função é bastante útil, principalmente para os implementadores do sistema que, eventualmente necessitem depurar variáveis internas. A perfeita utilização da listagem emitida requer conhecimento prévio de como são montadas as estruturas de dados do sistema. Esta informação é detalhada no capítulo 4.

3.7.5. A Função "Exibir Palheta de Cores"

A função "Exibir Palheta de Cores" é identificada pelo item **exibe Palette** do menu de miscelâneas. Seu papel é exibir a palheta de cores completa em utilização no sistema. Caso o sistema esteja operando no modo VGA, a função exibe a palheta das 16 (dezesseis) cores presentes. Caso o vídeo esteja no modo SVGA, a palheta apresentada conterá 256 (duzentas e cinqüenta e seis) cores distintas.

Quando ativada, a função exibe a palheta devida. O acionamento de qualquer tecla suspende a operação.

3.7.6. A Função "Identificar Vértices"

A função "Identificar Vértices" é acessada através do item **id. Vertices,** presente no menu de miscelâneas. Esta função é útil principalmente para tarefas de depuração do sistema, devido ao seu papel de exibir o número de identificação dos vértices de cada objeto exibido na cena.

Quando ativada, a função prontamente exibe a identificação junto a cada vértice presente na cena. O acionamento de qualquer tecla suspende a operação.

Em alguns casos, a proximidade de vértices pode tornar a sua identificação ilegível, devido à superposição de caracteres na tela. Esse problema pode ser contornado, mudando-se a posição do observador ou a orientação do objeto em questão. A figura 3.15 exibe o resultado de uma operação de identificação de vértices.

3.8. As Funções Especiais

As chamadas funções especiais estão disponíveis ao usuário praticamente em qualquer posição do sistema, não estando ligadas a nenhum dos menus descritos anteriormente. Estas funções são relacionadas na janela de mensagens e representadas pelas teclas <F1>, <F2>, <F3> e <Esc>. As funções são : **ajuda**, **zoom**, **desenhar** e **encerrar**.

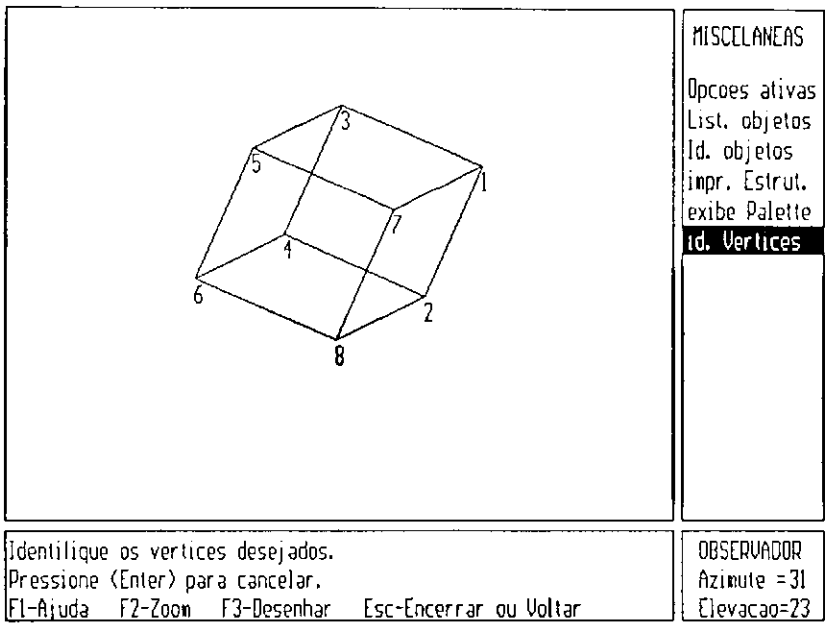


Figura 3.15 : Apresentação do número de identificação dos vértices de um objeto.

3.8.1. A Função "Ajuda"

A função "Ajuda" é associada à tecla <F1>. O objetivo desta função é apresentar, interativamente, instruções ao usuário de como utilizar o sistema. Esta função ainda não está disponível, devendo ser implementada em trabalhos futuros (ver capítulo 6).

3.8.2. A Função "Zoom"

A função "Zoom", identificada pela tecla <F2>, representa mais um recurso de visualização da cena construída. A função amplia a janela de exibição, fazendo com que a mesma ocupe toda a extensão da tela de interface. Esse recurso permite uma melhor visualização da cena corrente, e oferece, por exemplo, a possibilidade de se fotografar a cena sem a presença das demais partes da tela de interface. Ao se exibir uma cena em "zoom", alguns efeitos visuais, previamente definidos através do menu de visualização, são empregados na cena. O acionamento de qualquer tecla suspende a operação, restaurando a tela de interface original.

3.8.3. A Função "Desenhar"

A função "Desenhar" está associada à tecla <F3>, e é responsável pela exibição da cena corrente na tela do computador. A cena corrente é sempre exibida de acordo com a posição corrente do observador e a opção pela exibição dos eixos do sistema de coordenadas universais. Caso o menu corrente seja o menu de visualização, a exibição da cena também emprega os efeitos de realismo visual eventualmente solicitados pelo usuário. Nos demais menus, a exibição sempre é efetuada em "arame".

3.8.4. A Função "Encerrar ou Voltar"

Esta função é associada à tecla <Esc>, sendo utilizada para suspender a execução de alguma função pendente, ou para retornar ao menu hierarquicamente anterior ao menu corrente. Caso o menu corrente seja o menu principal, o acionamento da tecla <Esc> encerra a execução do sistema.

Capítulo 4

Implementação do Sistema João-de-Barro

4. IMPLEMENTAÇÃO DO SISTEMA JOÃO-DE-BARRO

Este capítulo é destinado à descrição das técnicas e métodos implementados no sistema João-de-Barro, incluindo aspectos como a arquitetura e concepção do sistema, descrição dos algoritmos utilizados, métodos empregados, entre outros. Ao longo do capítulo, são incluídas argumentações e citações a referências bibliográficas, fundamentando as técnicas adotadas.

4.1. A Arquitetura do Sistema

O sistema João-de-Barro é constituído implicitamente de três subsistemas : o **subsistema de entrada**, o **subsistema de modelagem** e o **subsistema de saída**. Esta arquitetura é semelhante à arquitetura proposta no item 2.8, diferindo em alguns pequenos detalhes. A figura 4.1 esquematiza a arquitetura do sistema.

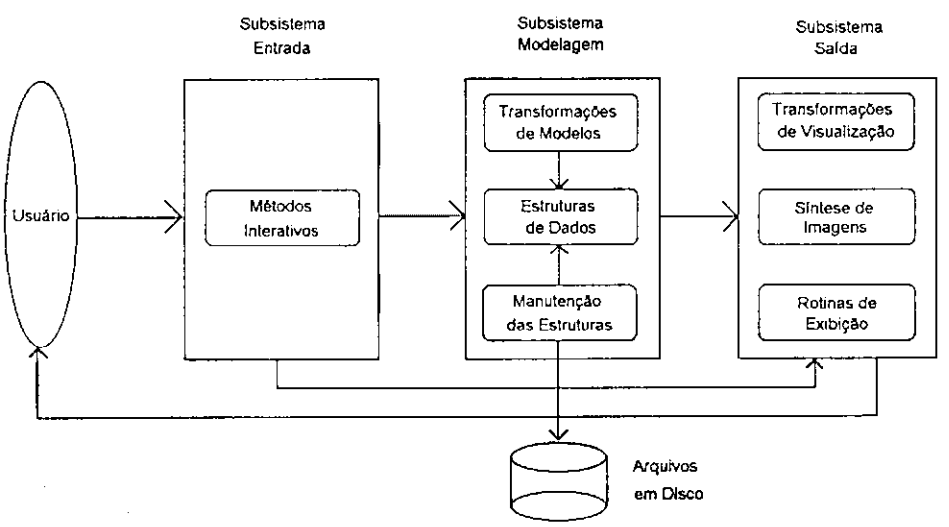


Figura 4.1 : Arquitetura geral do sistema João-de-Barro.

O **subsistema de entrada** consiste basicamente na interface do sistema, oferecendo meios de acesso às ferramentas disponíveis no sistema e estabelecendo comunicação com o usuário. Esse subsistema dá acesso às funções de transformação e manipulação de modelos, a operações diversas de manutenção das estruturas de dados e às funções do subsistema de saída.

O **subsistema de modelagem** contém as funções de transformação e manipulação de modelos, bem como outras funções de manutenção das estruturas de dados. As operações de transformação geométrica e as operações booleanas são exemplos de funções deste subsistema. O subsistema de modelagem também tem acesso aos arquivos em disco utilizados pelo sistema.

O **subsistema de saída** é responsável pela exibição dos modelos construídos. Nele estão funções de transformação de visualização, rotinas de exibição, emprego de efeitos visuais, entre outras.

4.2. O Domínio do Sistema

Conforme visto no capítulo 2, o domínio de um dado esquema de representação caracteriza o poder descritivo do sistema implementado, uma vez que o domínio estabelece o conjunto de entidades manipuláveis pelo sistema. O sistema João-de-Barro foi projetado visando, como domínio de representação, a modelos sólidos que satisfaçam as propriedades listadas no item 2.3, que definem as características a serem seguidas por modelos sólidos matematicamente válidos. A observância dessas propriedades se faz importante, principalmente para garantir a consistência e o funcionamento das operações booleanas, descritas ao longo do capítulo 5.

Formalmente, considera-se o domínio do sistema João-de-Barro como um conjunto de modelos sólidos matematicamente válidos, por serem esses os modelos capazes de serem tratados consistentemente em todos os pontos do sistema. Entretanto, existe uma particularidade em relação ao domínio adotado. Primeiramente, vale lembrar que o sistema foi concebido visando a aplicações de caráter visual e artístico. Dentro dessa concepção, podem-se imaginar casos em que o usuário deseje construir modelos não necessariamente sólidos, que não obedeçam às propriedades formais mencionadas anteriormente. Na sua proposta de sistema híbrido, descrita no item 4.3, o sistema João-de-Barro oferece ferramentas de representação em "Boundary Representation", permitindo a construção de primitivas através da informação explícita de suas faces, arestas e vértices (ver item 3.3.2). Por motivos de flexibilidade, essas ferramentas foram desenvolvidas de modo a não limitar o usuário à construção unicamente de modelos sólidos formalmente válidos. Essa facilidade foi incluída no sistema prevendo casos em que o usuário deseje maior liberdade na construção de seus modelos. Entretanto, o usuário deve estar atento para o fato de que modelos não-válidos podem ocasionar resultados desastrosos quando utilizados com as operação booleanas e com alguns algoritmos de visualização. Dentre algumas conseqüencias dessas operações inválidas, podem-se citar a "queda" do sistema, a violação de integridade das estruturas de dados, e a criação de modelos dissonantes do resultado esperado.

Vale ressaltar, também, que tais modelos não são considerados como integrantes do domínio do sistema.

4.3. O Esquema de Representação

Nos itens 2.4 e 2.5 deste texto, foram apresentados alguns fundamentos a respeito de esquemas de representação para sistemas de modelagem de sólidos, ressaltando a extrema importância e o papel fundamental que o esquema de representação exerce em um sistema de modelagem de sólidos.

O esquema de representação adotado pelo sistema João-de-Barro consiste em uma proposta de representação híbrida, combinando as técnicas "Boundary Representation (B-Rep)" e "Constructive Solid Geometry (CSG)". Essas técnicas foram escolhidas por apresentarem diversas conveniências de acordo com o propósito do sistema, além de diversos indicativos presentes na literatura que as apontam como técnicas de maior destaque. No esquema adotado, a representação B-Rep é utilizada como esquema primário e explícito, ou seja, as estruturas de dados utilizadas para armazenar os modelos são construídas em B-Rep. Já a representação CSG é utilizada de forma implícita e transitória, na qual as árvores CSG não são construídas de forma explícita, existindo, porém, as operações CSG para serem executadas sobre as estruturas B-Rep.

A decisão de incluir no sistema estruturas de dados construídas em B-Rep teve por base diversos fatores e referências. As principais vantagens e desvantagens oferecidas pelas representações B-Rep foram apresentadas no item 2.4.6. Porém, cabe aqui relembrar alguns aspectos visando fundamentar o projeto do esquema de representação adotado. Em primeiro lugar, [Requ80], [Requ82], [Requ83], [Mill89], dentre outras referências, apontam a grande disseminação, o poder de representação e a forte tendência à adoção de representações B-Rep. Um dos fatores mais atraentes dessas representações é a sua alta adequação para procedimentos de traçado de linhas e desenho dos objetos. Em sistemas interativos, onde se necessita de um bom tempo de resposta, essa característica é bastante desejável, sendo uma das razões para a larga utilização das representações B-Rep ([Requ80]). Outra característica interessante, também colocada por [Requ80], é o fato das representações B-Rep serem potencialmente capazes de abranger domínios tão extensos quanto as representações CSG. Além disso, modelos construídos em B-Rep facilitam a inclusão, nas estruturas de dados, de informações adicionais, tais como nome de identificação, cor matiz, vetores normais das faces, apontadores para estruturas auxiliares, entre outras. Em face a esses aspectos e outras questões consideradas, decidiu-se utilizar a representação B-Rep como uma das componentes do esquema híbrido proposto.

A segunda forma de representação escolhida para compor o esquema de representação do sistema João-de-Barro, já apresentada neste texto como "Constructive Solid Geometry", é conhecida por vantagens como grande abrangência de domínio, concisão, facilidade de construção de modelos, entre outras. Juntamente com as já mencionadas técnicas em B-Rep, as representações CSG são salientadas por [Requ80], [Requ82], [Requ83], [Mill89] como uma das mais importantes e promissoras técnicas de representação. No sistema João-de-Barro, os aspectos que mais favoreceram a inclusão de CSG foram a abrangência de domínio e a facilidade intuitiva com que usuários humanos manipulam seus modelos.

O projeto do esquema de representação para o sistema João-de-Barro consiste numa proposta de representação híbrida, aliando as técnicas B-Rep e CSG. O esquema é colocado na forma de uma proposta, devido à existência de dúvidas e divergências na literatura a respeito de quais representações podem ser consideradas como híbridas ou não. Assim, é apresentada aqui uma alternativa de se atribuir um caráter híbrido a um sistema de modelagem de sólidos, sem a pretensão de se rotular o sistema, categoricamente, como um sistema híbrido.

No esquema adotado, a representação B-Rep é tida como a representação primária do sistema, uma vez que os modelos são armazenados em estruturas B-Rep explicitamente construídas. Essas estruturas são construídas em caráter definitivo, permanecendo inalteradas até que novas operações exijam a sua atualização. Quando gravadas em disco, as estruturas B-Rep também são utilizadas para o arquivamento dos modelos construídos.

A forma de representação CSG é oferecida como mecanismo de entrada de dados presente na interface do sistema. Ao operar o sistema, o usuário tem a visão de um sistema CSG, onde as ferramentas básicas de edição consistem nas operações de transformação e nas operações booleanas. Essa forma de representação é colocada aqui como implícita e transiente, uma vez que as árvores CSG não são armazenadas como estruturas de dados, apesar de estarem implícitas nos modelos finais construídos.

Classificar um esquema de representação como híbrido/não-híbrido é tema de polêmica e divergências na literatura. Uma breve análise sobre essa questão torna-se pertinente, uma vez que a abordagem de alguns autores permite questionamentos sobre o caráter híbrido do sistema João-de-Barro. Primeiramente, vale salientar que a própria aplicação do termo híbrido é imprecisa na literatura, conforme alerta [Mill89], colocando duas possíveis interpretações para o termo.

De acordo com algumas correntes, um sistema pode ser considerado puramente como B-Rep, oferecendo operações booleanas como ferramentas de edição. Essas correntes defendem que a inclusão das operações booleanas não tornam o sistema como CSG, uma vez que as facilidades de edição oferecidas pela presença das árvores de construção não estão

disponíveis. Um ponto de apoio para essa interpretação consiste nos aspectos abordados por [Requ80], que menciona a existência de sistemas de modelagem com essas características.

Analisando por outro ponto de vista, mesmo em [Requ80] são mencionados aspectos arquiteturais que propiciam a construção de sistemas híbridos dotados de um esquema primário (principal) da representação, adicionado de métodos baseados em CSG para entrada de dados. Em um estudo mais recente, [Mill89] propõe as arquiteturas multirrepresentacionais como arquiteturas ideais para sistemas de modelagem de sólidos, mas ressalta que o estado da arte da tecnologia ainda não permite a construção de sistemas com tais arquiteturas. [Mill89] conclui que, dentre as possibilidades de construção de sistemas híbridos, encontram-se sistemas com arquitetura primária B-Rep, capazes de construir árvores CSG (sejam elas árvores completas ou simplesmente operações booleanas isoladas aplicadas a modelos já existentes), sem que essas árvores possam ser modificadas. Ainda segundo [Mill89], essas árvores consistem em estruturas transientes que não são armazenadas pelo sistema.

Em face aos pontos apresentados, considera-se viável e fundamentável que o sistema João-de-Barro seja apresentado como uma proposta de sistema híbrido. Obviamente, essa classificação pode ser revista, de acordo com o surgimento de novas abordagens e eventuais evoluções da tecnologia. A figura 4.2 esboça uma das arquiteturas de esquemas híbridos propostas por [Mill89], na qual se enquadra o sistema João-de-Barro.

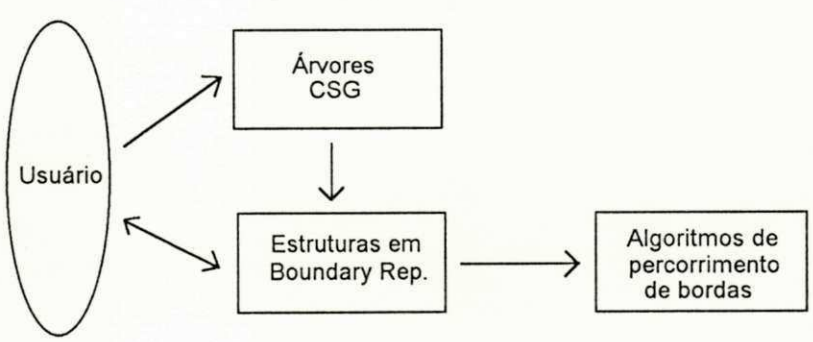


Figura 4.2 : O esquema híbrido de representação empregado no sistema João-de-Barro.

Para uma análise mais profunda das questões colocadas, é indicada a consulta a [Requ80], [Requ82], [Requ83], [East84] e [Mill89]. Outros aspectos sobre a utilização desse esquema de representação podem ser consultados nos itens 2.4.6, 2.4.7 e 2.5.

4.4. As Estruturas de Dados

O sistema utiliza várias estruturas de dados globais para representar e controlar seus modelos e componentes. As cenas e objetos são armazenados na forma de um conjunto de listas duplamente encadeadas. As demais estruturas consistem em variáveis globais simples e não necessitam de maiores esclarecimentos. Eventualmente, estruturas de dados auxiliares são criadas transitoriamente para a execução de determinados algoritmos. Conforme o esboço geral presente na figura 4.3, é apresentada aqui a forma básica de representação dos objetos e cenas no sistema João-de-Barro.

As estruturas de dados utilizadas para representar um objeto seguem o método Boundary Representation, representando o objeto através de suas faces, arestas e vértices. As estruturas construídas formam um conjunto de listas encadeadas interligadas. Para cada objeto, existe um nó (ver figura 4.3) contendo variáveis que determinam características daquele objeto, tais como nome de identificação, coordenadas do baricentro, variáveis de controle, entre outras. Cada nó de objeto contém também apontadores para a lista de vértices e para a lista de faces do objeto, ambas construídas na forma de listas duplamente encadeadas. São presentes, ainda, os apontadores para outros nós de objeto, visando à construção de listas duplamente encadeadas. A estrutura dos nós de objeto são apresentadas pela definição abaixo, extraída do programa fonte em linguagem C.

```
typedef struct tipo_stru_modelo {
    char nome[9];
    char visibilidade;
    double x_bar;
    double y_bar;
    double z_bar;
    double x_bar_visual;
    double y_bar_visual;
    double z_bar_visual;
    stru_vertice *ap_vertice;
    stru_face *ap_face;
    struct tipo_stru_modelo *ap_anterior;
    struct tipo_stru_modelo *ap_proximo;
} stru_modelo;
```

Conforme mencionado anteriormente, cada objeto contém uma lista de vértices. Os nós dessa lista concentram variáveis tais como as coordenadas X, Y e Z reais e visuais do vértice, o seu tipo, um apontador para a criação eventual de uma lista de adjacências, além dos apontadores para outros nós de vértices. A definição abaixo, apresentada em linguagem C, ilustra a estrutura dos nós de vértices.

```
typedef struct tipo_stru_vertice {
    int numero;
    double x_real;
    double y_real;
    double z_real;
    double x_visual;
    double y_visual;
    double z_visual;
    char   tipo;
    struct tipo_stru_adjac   *ap_adjac;
    struct tipo_stru_vertice *ap_anterior;
    struct tipo_stru_vertice *ap_proximo;
} stru_vertice;
```

Cada nó de objeto aponta, também, para uma lista que contém as faces que compõem o objeto, cujos nós contêm dados como número da face, cor_matiz, intensidade de cor calculada para iluminação, tipo da face, um apontador para a lista das arestas componentes da face, além dos apontadores para outros nós de faces. Um nó de face é construído de acordo com a seguinte estrutura, extraída do programa fonte :

```
typedef struct tipo_stru_face {
    int numero;
    int cor_matiz;
    int intensidade_cor;
    char tipo;
    stru_aresta *ap_aresta;
    struct tipo_stru_face *ap_anterior;
    struct tipo_stru_face *ap_proximo;
} stru_face;
```

Os nós pertencentes às listas de arestas armazenam o número de identificação da aresta em questão, apontadores para os vértices que compõem a aresta, bem como apontadores para a construção da lista duplamente encadeada de arestas. A definição abaixo, escrita em linguagem C e extraída diretamente do programa fonte do sistema, esboça o conteúdo dos nós de arestas :

```
typedef struct tipo_stru_aresta {
    int numero;
    stru_vertice *ap_vert1;
    stru_vertice *ap_vert2;
    struct tipo_stru_aresta *ap_anterior;
    struct tipo_stru_aresta *ap_proximo;
} stru_aresta;
```

A figura 4.2 apresenta uma visão geral de como as estruturas descritas acima são organizadas :

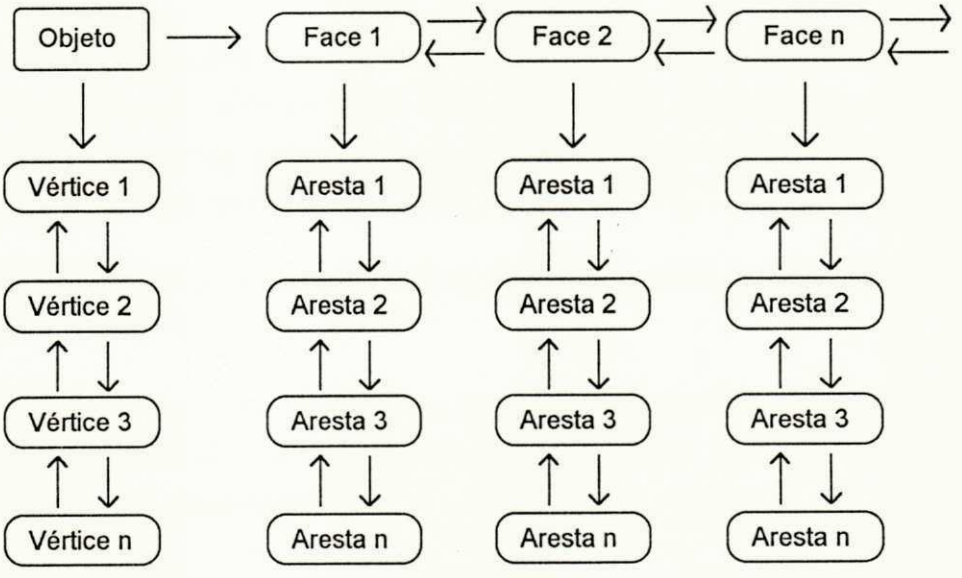


Figura 4.3 : Organização básica das estruturas de dados utilizadas para a representação de um objeto.

Outros elementos manipuláveis pelo sistema são as cenas, que consistem simplesmente em listas duplamente encadeadas contendo todos os objetos componentes da cena. Além dessa lista, uma cena contém outras variáveis globais, tais como as coordenadas do observador, as opções de visualização selecionadas, a distância do plano de projeção usada para perspectiva, dentre outras variáveis de controle. O arquivo de uma dada cena armazena todas essas variáveis globais, seguidas pelas estruturas B-Rep de cada objeto componente da cena.

4.5. A Geração Automática de Primitivas Cilíndricas

Algoritmos responsáveis pela criação automática de determinadas classes de primitivas geométricas são um recurso importante em um sistema de modelagem de sólidos. Neste tópico, descreve-se o algoritmo implementado para a criação de primitivas cilíndricas, desenvolvido a partir da idéia básica exposta em [Picc88].

O algoritmo aplica técnicas de revolução de superfícies, ou seja, parte-se do princípio de que a rotação de uma superfície quadrangular em torno de uma de suas arestas pode gerar um modelo sólido cilíndrico. As linhas gerais do algoritmo são apresentadas pela seqüência de passos abaixo que devem ser acompanhados com o auxílio da figura 4.4 e da rotina em pseudo-código, fornecida para uma melhor compreensão.

1) É informado pelo usuário, o número de vértices (N) desejado para a base da primitiva. Nessa concepção, um cubo pode ser criado como uma primitiva cilíndrica com quatro vértices na base. Da mesma forma, pode-se criar uma primitiva cilíndrica hexagonal, solicitando-se seis vértices para a "base" da primitiva.

2) Cria-se, nas estruturas de dados do sistema, um novo modelo já com duas faces, referentes à "tampa" e à "base" da primitiva.

3) Define-se uma superfície quadrangular "S", cujos vértices têm valores pré-definidos por convenção do sistema. Essa superfície deverá conter uma das arestas coincidente com o eixo "Y" do sistema de coordenadas.

4) Inserem-se os vértices externos da superfície "S" nas faces referentes à tampa e à base da primitiva. Por vértices externos, entendam-se os vértices que não estão sobre o eixo "Y". Esses vértices serão referenciados como V1 e V2.

5) Calcula-se o ângulo de rotação (alfa) a ser usado em cada passo do algoritmo, para rotacionar a superfície "S" em torno do eixo "Y". Esse ângulo é obtido por 360 / N.

6) Executam-se os passos de 7 a 10, N vezes.

7) Rotaciona-se a superfície "S" em torno do eixo "Y", empregando-se um ângulo de "beta" graus, obtendo-se um vértice V3 correspondente a V1 rotacionado, bem como V4 correspondente a V2 rotacionado.

8) Cria-se uma nova face lateral na primitiva, composta pelos vértices V1, V2, V3 e V4, definindo arestas devidamente orientadas no sentido anti-horário.

9) Insere-se V3 na face referente à tampa, e V4 como integrante da face referente à base da primitiva.

10) Faz-se V1 = V3 e V2 = V4

11) Finalizadas as N rotações, inverte-se o sentido das arestas componentes da base da primitiva, para que a mesma tenha a orientação devida do vetor normal.

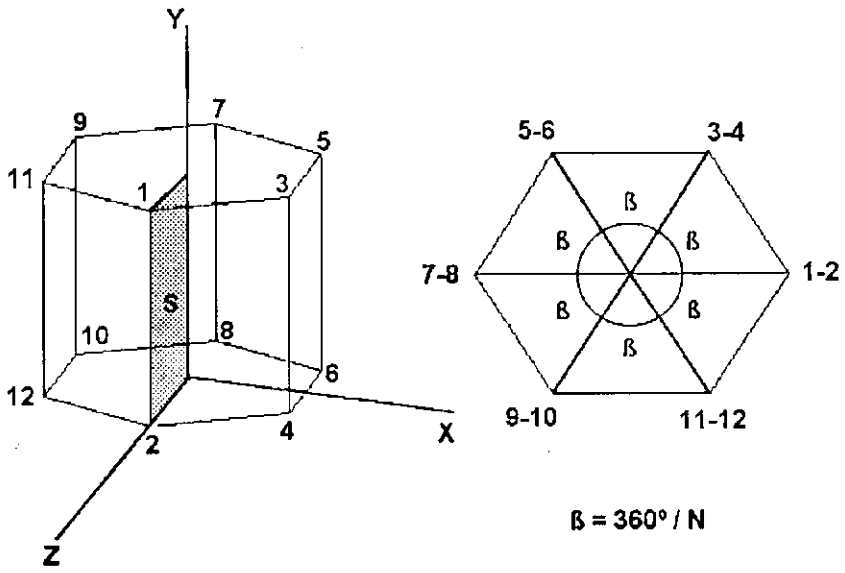


Figura 4.4 : Mecanismo utilizado para a criação de primitivas cilíndricas.

```
CRIA_PRIMITIVA_CILINDRICA ( numero_vertices_base )
Início
  N = numero_vertices_base
  Cria novo nó de modelo
  Cria novo nó de face para a "tampa" da primitiva
  Cria novo nó de face para a "base" da primitiva
  Insere vértices 1 e 2 com coordenadas previamente estipuladas
```

```
Alfa = 0
Para I = 1 a N faça
Início
  Alfa = Alfa + 360 / N
  Rotaciona vértice 1 de alfa graus em torno de Y
  Obtem coordenadas X, Y e Z do vértice I * 2 + 1
  Rotaciona vértice 2 de alfa graus em torno de Y
  Obtem coordenadas X, Y e Z do vértice I * 2 + 2
  Insere novos vértices na lista de vértices
  Cria nova face lateral com novos vértices
  Cria novas arestas na "tampa" e na "base"
Fim
Inverte a orientação das arestas componentes da base da primitiva
Fim
```

A figura 4.5 dá uma demonstração de várias primitivas cilíndricas geradas no sistema João-de-Barro utilizando-se esse algoritmo.

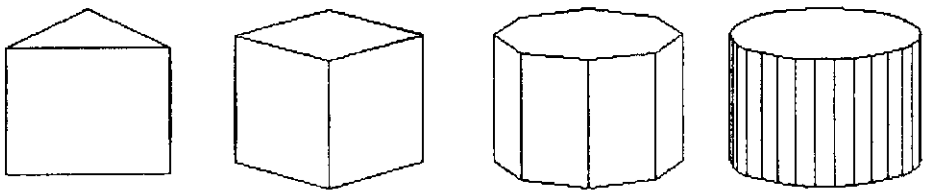


Figura 4.5 : Primitivas cilíndricas geradas automaticamente pelo sistema João-de-Barro.

Apesar da pequena confusão semântica que o termo **"cilíndrico"** possa ocasionar, deve-se esclarecer que este o termo é empregado somente para uma melhor assimilação do tipo de primitiva criada por este algoritmo, uma vez que muitas das primitivas geradas não constituem cilindros propriamente ditos.

4.6. A Geração Automática de Primitivas Cônicas

Analogamente à geração de primitivas cilíndricas, o algoritmo implementado para a criação automática de primitivas cônicas foi obtido a partir de modificações sobre as idéias apresentadas em [Picc88].

O algoritmo também utiliza técnicas de revolução de superfícies, a partir do princípio de que a revolução de um triângulo retângulo em torno de um de seus catetos pode originar primitivas sólidas cônicas. Os procedimentos básicos adotados são descritos pelos passos abaixo e complementados pelo pseudo-código incluído. Para uma melhor compreensão, a figura 4.6 ilustra os procedimentos adotados.

1) É informado pelo usuário o número de vértices (N) desejado para a base da primitiva. Nessa concepção, um tetraedro pode ser criado como uma primitiva cônica com três vértices na base. Da mesma forma, pode-se criar uma primitiva com formato de pirâmide solicitando-se quatro vértices para a base da primitiva.

2) Cria-se, nas estruturas de dados do sistema, um novo modelo já dotado de uma face correspondente à base da primitiva.

3) Define-se uma superfície "S" na forma de um triângulo retângulo, cujos vértices têm valores pré-definidos por convenção do sistema. Essa superfície deverá conter um dos catetos posicionado sobre o eixo "Y" do sistema de coordenadas.

4) Insere-se o vértice externo da superfície "S" na face referente à base da primitiva. Por vértice externo deve-se entender o vértice que não está sobre o eixo "Y". Esse vértice será referenciado como V2.

5) Calcula-se o ângulo de rotação (beta) a ser usado em cada passo do algoritmo para rotacionar a superfície "S" em torno do eixo "Y". Esse ângulo é obtido por 360 / N.

6) Executam-se os passos de 7 a 10, "N" vezes.

7) Rotaciona-se a superfície "S" em torno do eixo "Y" empregando-se um ângulo de "beta" graus, obtendo-se um vértice V3 correspondente a V2 rotacionado.

8) Cria-se uma nova face lateral na primitiva, composta pelos vértices V1, V2, e V3, definindo arestas devidamente orientadas no sentido anti-horário.

9) Insere-se V3 na face referente à base da primitiva.

10) Faz-se V2 = V3.

11) Finalizadas as N rotações, inverte-se o sentido das arestas componentes da base da primitiva, para que a mesma tenha a devida orientação do vetor normal.

```
CRIA_PRIMITIVA_CÔNICA ( número_vértices_base )
Início
  N = número_vértices_base
  Cria novo nó de modelo
  Cria novo nó de face para a base da primitiva
  Insere vértices 1 e 2 com coordenadas previamente estipuladas
  Beta = 0
  Para  I = 1 a N  faça
  Início
    Beta = Beta + 360 / N
    Rotaciona vértice 1 de beta graus em torno de Y
    Obtem coordenadas X, Y e Z do vértice I + 1
    Insere novo vértice na lista de vértices
    Cria nova face lateral com novo vértice
    Cria nova aresta na "base" da primitiva
  Fim
  Inverte sentido das arestas pertencentes à base da primitiva
Fim
```

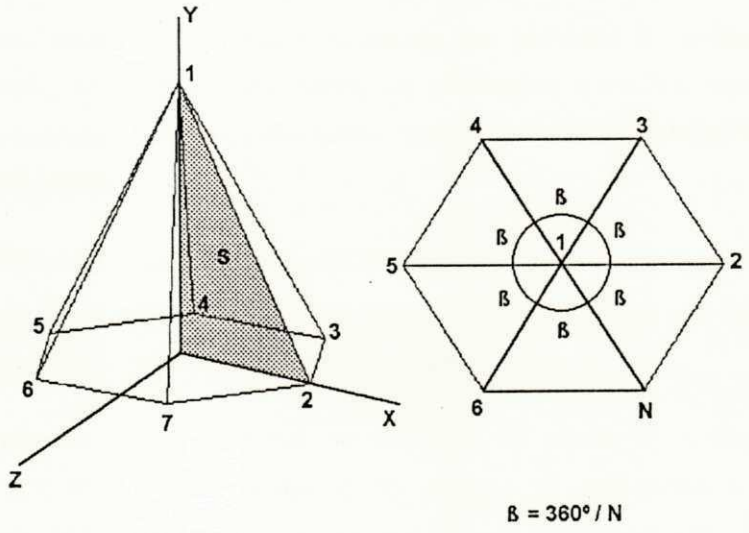


Figura 4.6 : Ilustração do mecanismo utilizado para a geração de primitivas cônicas.

A figura 4.7 apresenta algumas primitivas cônicas geradas automaticamente pelo algoritmo apresentado.

Apesar da pequena confusão semântica que o termo **"cônico"** possa ocasionar, deve-se esclarecer que esse termo é empregado somente para uma melhor assimilação do tipo de primitiva criada por este algoritmo, uma vez que muitas das primitivas geradas não constituem cones propriamente ditos.

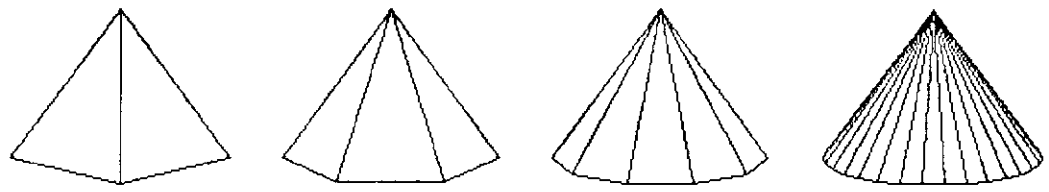


Figura 4.7 : Primitivas cônicas geradas automaticamente pelo sistema João-de-Barro.

4.7. A Geração Automática de Primitivas Esféricas

No sistema João-de-Barro desenvolveu-se um algoritmo, baseado na proposta de [Picc88], para a criação de primitivas esféricas. Vale ressaltar que as primitivas geradas não consistem em esferas perfeitas, mas sim em aproximações poliédricas.

Uma esfera aproximada é composta por paralelos e meridianos. O algoritmo consiste, basicamente, em gerar um dos meridianos e rotacioná-lo em 360 (trezentos e sessenta) graus, gerando os paralelos e os novos meridianos da esfera. Para um melhor entendimento, valem as seguintes considerações :

1) O usuário informa ao algoritmo o número de faces em cada hemisfério sul/norte da esfera. Esse número de faces indica quantas faces planas existirão em cada um dos hemisférios sul e norte da esfera, entre o "equador" e o respectivo "pólo".

2) O primeiro passo do algoritmo é gerar um meridiano. As coordenadas dos pólos e o raio da esfera são estabelecidos por convenção do sistema. O pólo norte é posicionado com coordenadas (0, RAIO, 0). O pólo sul é posicionado nas coordenadas (0, -RAIO, 0). A esfera gerada tem o seu baricentro exatamente sobre a origem do sistema de coordenadas. Sendo

"N" o número de faces por hemisfério informado pelo usuário, para gerar o meridiano inicial o algoritmo executa, sobre o pólo norte, 2N rotações de 180 / 2N graus. Os novos pontos obtidos com cada rotação vão compondo o meridiano. A figura 4.8a ilustra essa operação.

3) Uma vez gerado o meridiano inicial, o algoritmo efetua sobre ele 4N rotações de 360 / 4N graus (ver figura 4.8b). [Picc88] afirma que esse procedimento garante uma boa proporção entre o número de meridianos e o número de paralelos da esfera.

O algoritmo pode ser esboçado através do seguinte pseudocódigo :

```
CRIA_PRIMITIVA_ESFÉRICA ( número_vértices_base )
Início
  N = número_vértices_hemisfério
  Cria nodo de modelo para armazenar a primitiva
  Insere vértices referentes aos pólos sul e norte
  Theta = 0
  Para I = 1 a 2N faça
  Início
    Theta = Theta + 180 / 2N
    Rotaciona pólo norte de Theta graus em torno do eixo Z
    Insere novo ponto na lista de vértices
  Fim
  Phi = 0
  Para I = 1 a 4N faça
  Início
    Phi = Phi + 360 / 4N
    Para cada paralelo faça
    Início
      Rotaciona vértice anterior no mesmo paralelo em Phi graus em torno de Y
      Insere novo ponto na lista de vértices
      Cria novo nodo de face com os devidos vértices
    Fim
  Fim
Fim
```

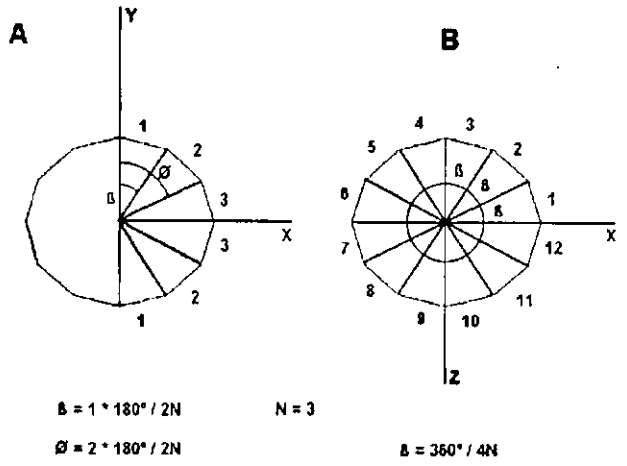


Figura 4.8 : Mecanismos para a criação de primitivas esféricas.

A figura 4.9 apresenta algumas esferas geradas automaticamente utilizando-se este algoritmo.

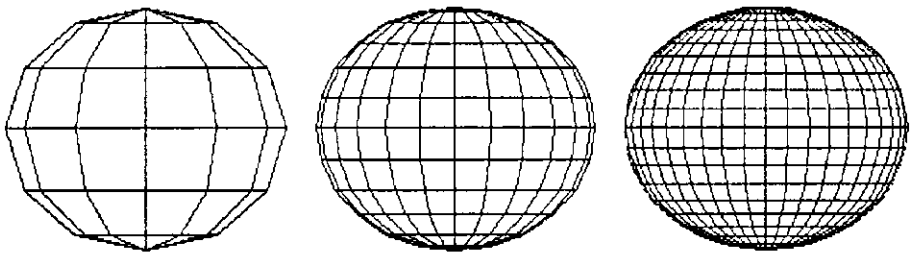


Figura 4.9 : Esferas aproximadas geradas automaticamente no sistema João-de-Barro.

4.8. As Operações de Translação

Por consistir numa operação básica da área de modelagem de sólidos, existem inúmeras referências na literatura que mencionam o desenvolvimento de operações de translação. Sobre o assunto, recomenda-se a consulta a [Faux79], [Newm81], [Fole82], [Muft83], [Park85], [Berg86], [Hear86], [Thal87] e [Picc88].

Uma operação de translação causa o deslocamento de um determinado objeto através do espaço abstrato estabelecido pelo sistema. Esse deslocamento é expresso pelos

parâmetros TX, TY e TZ, que indicam os deslocamentos ao longos dos eixos X, Y e Z respectivamente. Assim, as novas coordenadas de um dado ponto podem ser obtidas por :

$$X' = X + TX$$
$$Y' = Y + TY$$
$$Z' = Z + TZ$$

Para se efetuar a translação de um objeto, basta que se percorra a sua lista de vértices, aplicando-se as equações para as coordenadas reais de cada um dos vértices. A rotina abaixo, escrita em pseudo-código, esboça um procedimento com esse propósito.

```
EFETUA_TRANSLAÇÃO ( objeto , TX , TY , TZ )
Início
  Para cada vértice do objeto faça
    X_REAL = X_REAL + TX
    Y_REAL = Y_REAL + TY
    Z_REAL = Z_REAL + TZ
  Fim-para
Fim
```

Conforme mencionado no capítulo 3, o sistema João-de-Barro permite que os objetos sejam transladados tanto pela informação explícita dos parâmetros TX, TY e TZ quanto por mecanismos interativos. No caso da translação paramétrica, a implementação da operação consiste simplesmente na execução da rotina EFETUA_TRANSLAÇÃO, passando-se os parâmetros informados pelo usuário. Por sua vez, a execução de translações interativas é feita através de translações sucessivas a cada comando do usuário, utilizando-se TX=TY=TZ=1 como parâmetros da operação e atualizando-se a cena a cada translação.

4.9. As Operações de Rotação

As operações de rotação também são largamente abordadas na literatura, sendo recomendada a consulta a [Faux79], [Newm81], [Fole82], [Muft83], [Park85], [Berg86], [Hear86], [Thal87] e [Picc88] para pesquisas sobre esta operação.

Uma operação de rotação é determinada pelos parâmetros RX, RY e RZ, que indicam o ângulo de rotação em torno de cada um dos eixos X, Y e Z. Para se rotacionar um dado ponto com coordenadas X, Y e Z, devem-se aplicar as seguintes equações :

A) Rotação em torno do eixo X

X' = X

Y' = Y cos (RX) - Z sen (RX)

Z' = Y sen (RX) + Z cos (RX)

B) Rotação em torno do eixo Y

X' = X cos (RY) + Z sen (RY)

Y' = Y

Z' = - X sen (RY) + Z cos (RY)

C) Rotação em torno do eixo Z

X' = X cos (RZ) - Y sen (RZ)

Y' = X sen (RZ) + Y cos (RZ)

Z' = Z

Assim, para se efetuar a rotação de um determinado objeto, basta que as equações acima sejam adequadamente aplicadas às coordenadas de cada um dos vértices do objetos. Um procedimento com esse fim é apresentado pela rotina abaixo escrita em pseudo-código :

```
EFETUA_ROTAÇÃO ( objeto , RX , RY , RZ )
Início
  calcula_baricentro ( objeto ) {obtem x_bar, y_bar e z_bar}
  efetua_translação ( objeto , -x_bar, -y_bar, -z_bar )
  Para cada vértice do objeto faça
    Se RX <> 0
      Y' = Y_REAL cos (RX) - Z_REAL sen (RX)
      Z' = Y_REAL sen (RX) + Z_REAL cos (RX)
      Y = Y'
      Z = Z'
    Fim-se
    Se RY <> 0
      X' = X_REAL cos (RY) + Z_REAL sen (RY)
      Z' = - X_REAL sen (RY) + Z_REAL cos (RY)
      X = X'
      Z = Z'
    Fim-se
```

```
    Se RZ <> 0
      X' = X_REAL cos (RZ) - Y_REAL sen (RZ)
      Y' = X_REAL sen (RZ) + Y_REAL cos (RZ)
      X = X'
      Y = Y'
    Fim-se
  Fim-para
  efetua_translação ( objeto , x_bar , y_bar , z_bar )
Fim
```

Aqui, vale uma observação importante : para que o objeto seja rotacionado em torno do seu baricentro, ou seja, mantendo sua posição e tendo somente sua orientação alterada, é necessário, ao se efetuar a rotação, que o baricentro do objeto coincida com a origem do sistema de coordenadas universais. Se assim desejado, antes de efetuar-se a rotação deve-se transladar o objeto para a origem do sistema de coordenadas, onde os parâmetros da translação consistem nas coordenadas do baricentro do objeto com sinais invertidos. Após a rotação, o objeto deve ser transladado de volta ao seu ponto de origem.

No sistema João-de-Barro, as operações de rotação paramétrica são implementadas de forma a executar a rotina EFETUA_ROTAÇÃO passando-se os parâmetros RX, RY e RZ informados pelo usuário diretamente. Já nas operações de rotação interativa, cada comando do usuário rotaciona o objeto de um grau em torno do eixo corrente, atualizando-se a cena em exibição a cada rotação efetuada.

4.10. As Operações de Escala

Da mesma forma que as operações de translação e rotação, as operações de escala têm ampla abordagem na literatura. Recomenda-se a consulta às referências [Faux79], [Newm81], [Fole82], [Muft83], [Park85], [Berg86], [Hear86], [Thal87] e [Picc88] para pesquisas sobre esta operação.

Uma operação de escala é expressa em termos dos parâmetros SX, SY e SZ, que especificam os fatores de multiplicação das coordenadas X, Y e Z de cada vértice do objeto. Essa multiplicação acarretará mudança das dimensões e proporções do objeto envolvido na operação. Para se efetuar uma operação como esta basta se aplicarem as seguintes equações sobre as coordenadas de cada vértice do objeto :

X' = X * SX
Y' = Y * SY
Z' = Z * SZ

A rotina abaixo, escrita em pseudo-código, descreve uma função destinada à aplicação de fatores de escala sobre objetos. Um procedimento equivalente a esse foi desenvolvido no sistema João-de-Barro.

```
EFETUA_ESCALA ( objeto , SX , SY , SZ )
Início
  Para cada vértice do objeto faça
    X_REAL = X_REAL * SX
    Y_REAL = Y_REAL * SY
    Z_REAL = Z_REAL * SZ
  Fim-para
Fim
```

Conforme mencionado no capítulo 3, o sistema João-de-Barro oferece operações de escala paramétricas e interativas. As operações paramétricas foram implementadas de modo a executar um procedimento semelhante ao EFETUA_ESCALA utilizando os parâmetros SX, SY e SZ informados explicitamente pelo usuário. Nas operações interativas, a cada comando do usuário é aplicado um fator de escala igual a 1.1 para incremento, e 0.9 para decremento, de acordo com o eixo corrente, atualizando-se a cena a cada passo efetuado.

4.11. As Operações de Posicionamento do Observador

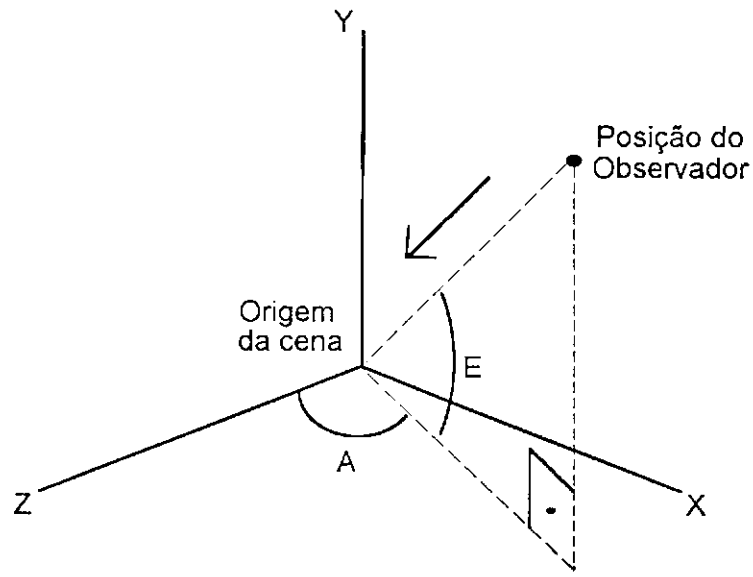
Os procedimentos que lidam com o posicionamento do observador são bastante clássicos, presentes em diversas referências da literatura. Indicamos a consulta a [Faux79], [Newm81], [Fole82], [Muft83], [Park85], [Berg86], [Hear86], [Thal87] e [Picc88].

A abordagem mais freqüente na literatura utiliza mecanismos de transformação de sistemas de coordenadas para o posicionamento do observador. Entretanto, [Picc88] propõe um modelo simplificado e bastante claro para este tipo de operação, que foi também adotado no sistema João-de-Barro.

Segundo o modelo proposto por [Picc88], a posição do observador é definida por dois ângulos : Azimute e Elevação. O ângulo azimute determina o ângulo de rotação do observador em torno do eixo Y do sistema de coordenadas universais. O ângulo Elevação determina a rotação do observador em torno do eixo X. A figura 4.10 ilustra os ângulos determinantes da posição do observador. Por convenção do sistema, o observador tem a visão direcionada sempre para a origem do sistema de coordenadas universais. Para se efetuar a operação basta se rotacionar cada objeto da cena, utilizando-se um ângulo de "-Azimute" em torno do eixo Y, bem como rotacionar os objetos com um ângulo de "+Elevação" em torno do eixo X.

A rotina seguinte esboça um procedimento para posicionar o observador, implementado segundo o modelo aqui descrito :

```
POSICIONA_OBSERVADOR ( azimute , elevação )
Início
  azimute = azimute * -1
  Para cada vértice do objeto faça
    Se elevação <> 0
      Y' = Y_REAL cos (elevação) - Z_REAL sen (elevação)
      Z' = Y_REAL sen (elevação) + Z_REAL cos (elevação)
      Y = Y'
      Z = Z'
    Fim-se
    Se azimute <> 0
      X' = X_REAL cos (azimute) + Z_REAL sen (azimute)
      Z' = - X_REAL sen (azimute) + Z_REAL cos (azimute)
      X = X'
      Z = Z'
    Fim-se
  Fim-para
Fim
```



A = AZIMUTE

E = ELEVAÇÃO

Figura 4.10 : Ângulos Azimute e Elevação assumidos pelo observador da cena.

As operações de posicionamento do observador podem ser realizadas tanto pela informação de parâmetros quanto por métodos interativos. A implementação da versão paramétrica consiste em executar-se um um procedimento semelhante ao POSICIONA_OBSERVADOR, passando-se os parâmetros "azimute" e "elevação" informados pelo usuário. Na abordagem interativa, cada comando do usuário causa o incremento do ângulo ativo em 01 (um) grau, atualizando-se a cena a cada mudança efetuada.

4.12. A Exibição dos Eixos de Coordenadas

Conforme descrito no capítulo 3, o sitema João-de-Barro oferece recursos para que os eixos do sistema de coordenadas universais sejam exibidos junto com a cena cena corrente. Esses recursos são interessantes para uma melhor orientação e visão espacial dos objetos contruídos e são amplamente abordados na literatura.

A implementação dos eixos de coordenadas é bastante simples, podendo ser descrita pelas seguintes colocações :

1) Existe um "flag" de controle que indica a opção do usuário por exibir ou não os eixos de coordenadas. Em caso afirmativo, a rotina de exibição da cena corrente executa procedimentos responsáveis pela exibição dos eixos. Caso contrário, tais procedimentos são ignorados.

2) Os eixos são construídos internamente como um objeto especial dotado de 04 (quatro) vértices : os pontos (0,0,0), (coord,0,0), (0,coord,0) e (0,0,coord), respectivamente correspondentes à origem do sistema de coordenadas e às extremidades dos eixos X, Y, e Z. A variável "coord" consiste num valor previamente convencionado pelo sistema. Para a exibição dos eixos basta efetuar-se o traçado entre a origem do sistema de coordenadas e cada uma das extremidades.

3) Os eixos são afetados pelas operações de posicionamento do observador, descritas no item 4.11, e pelas operações de visualização em "zoom" a serem apresentadas mais adiante no item 4.18. Nessas operações, as mesmas transformações aplicadas aos objetos da cena são também efetuadas sobre as coordenadas que definem o sistema de eixos.

4.13. A Operação de Anexação

Conforme já apresentado no item 3.4.13, a operação de anexação tem como papel fundir dois objetos distintos em um único objeto, fundindo as suas estruturas de dados. É importante frisar que esta operação tem um papel diferente da operação booleana de união, uma vez que nessa última as porções desnecessárias não são mantidas no objeto final (ex: porções internas ao objeto) além de se preservarem as propriedades formais essenciais para a consistência do objeto.

A implementação da operação de anexação consiste simplesmente na manipulação de estruturas de dados e seus apontadores. Consequentemente, de acordo com as estruturas de dados descritas no item 4.4, para fundir um dado objetoA a um objetoB, basta que a lista de vértices do objetoB seja acrescentada ao final da lista de vértices do objetoA. Da mesma forma, a lista de faces do objetoB deve ser adicionada à lista de faces do objetoA. O único cuidado a ser tomado refere-se ao número de identificação dos vértices e das faces anexados, devendo-se efetuar um procedimento de renumeração para tais identificadores.

4.14. A Construção da Palheta de Cores

Visando a uma melhor apresentação de resultados, o sistema João-de-Barro inclui métodos de iluminação dos modelos criados. Para tal, o sistema foi concebido de forma a utilizar os recursos de placas de vídeo Super-VGA, capazes de exibir 256 (duzentas e cinqüenta e seis) cores simultâneas. Conseqüentemente, mostrou-se necessária a implementação de um procedimento para a construção da palheta de cores a ser utilizada.

Geralmente, a construção da palheta de cores em um sistema de modelagem envolve algumas questões críticas que, quando não solucionadas adequadamente, podem comprometer a qualidade na apresentação de resultados. Dentre as questões mais relevantes, destacam-se os seguintes pontos :

1) Deve-se estabelecer um modelo de cores a ser utilizado. Em [Fole82], [Thal87] e [Gome90] são apresentados os clássicos modelos RGB e HSV. Segundo o modelo RGB (red/green/blue), cada cor é definida por componentes primárias em vermelho verde e azul. Já no modelo HSV (hue/saturation/value) as cores são definidas a partir da matiz, saturação e valor. Esses modelos são descritos em detalhe nas referências mencionadas anteriormente, que também incluem estudos e análises sobre distribuição espectral, combinação de cores, sistemas de cores, dentre outros aspectos.

2) Alguns sistemas implementam mecanismos dinâmicos de construção de palhetas de cores, que atualizam a palheta em tempo real, de acordo com características da cena e dos objetos a serem exibidos, ao invés de utilizarem uma palheta fixa. Em sistemas com essa característica, supondo uma palheta de 256 cores para uma cena que contém um único objeto de uma única cor matiz, pode-se construir a palheta dinamicamente utilizando-se 256 gradações da cor matiz do objeto. Na maioria dos casos esse procedimento permite maior precisão e realismo visual dos modelos exibidos.

3) É necessário estabelecer-se um mecanismo, para que a palheta construída seja capaz de abranger toda a faixa de gradações necessárias para cada cor matiz que se deseje incluir na palheta. Este procedimento costuma não ser trivial, principalmente em palhetas com número total de cores reduzido. Por exemplo, supondo um palheta de 256 (duzentas e cinqüenta e seis) cores simultâneas onde sejam desejadas 16 cores matizes cujas faixas iniciem na cor preta, passando pela cor matiz pura e terminando na cor branca, conclui-se que serão disponíveis 16 (dezesseis) tons para abranger cada faixa. Uma faixa de cores deve abranger todo o espectro desejado (no caso, do preto ao branco) com suave gradação de tons, evitando mudanças abruptas de tonalidades, e com tons distinguíveis entre si pela visão humana, uma

vez que a diferença entre algumas tonalidades muito próximas não podem ser visualmente diferenciadas. Essas questões são abordadas em [Fole82], [Thal87], [Gome90], dentre outras referências. Em [Abra92], é apresentado um estudo específico a respeito da construção de palhetas de 256 (duzentas e cinqüenta e seis) cores em vídeos padrão VGA.

A palheta de cores construída no sistema João-de-Barro segue o modelo RGB, no qual cada cor é expressa através das tonalidades primárias vermelho, verde e azul. Esse modelo foi adotado devido à sua simplicidade e suporte direto pelo "driver" de vídeo utilizado, desenvolvido e fornecido pela JORDAN HARGRAPHIX SOFTWARE para a programação de placas SVGA em linguagens C e PASCAL.

A palheta contém 256 (duzentas e cinqüenta e seis) cores no total, distribuídas em 16 (dezesseis) faixas. Por motivos de compatibilidade com o padrão VGA, a primeira faixa contém as 16 (dezesseis) cores da palheta VGA padrão. A partir daí, são construídas 15 (quinze) faixas com gradações para cada uma das cores matizes presentes na palheta padrão VGA.

Cada faixa de tonalidades foi montada iniciando na cor preta, atingindo a cor matiz pura como ponto de saturação, e finalizando na cor branca. A determinação das tonalidades seguiu o mesmo modelo proposto por [Picc88], que utiliza os primeiros 4/5 (quatro quintos) de cada faixa para graduar as tonalidades entre o preto e a cor de saturação, calculando-se as tonalidades com distribuição linear. O último 1/5 (um quinto) da faixa é utilizado para as gradações entre o ponto de saturação e a cor branca.

4.15. A Exibição em Perspectiva

O sistema João-de-Barro oferece a possibilidade de exibir cenas com projeções em perspectiva, como um recurso adicional de visualização. A implementação dessa função é bem abordada pela literatura, principalmente em [Faux79], [Newm81], [Fole82], [Muft83], [Park85], [Berg86], [Hear86], [Thal87], [Picc88], entre outras referências.

As operações de projeção em perspectiva consistem, basicamente, em operações de escalonamento, que "distorcem" os objetos exibidos simulando os efeitos visuais de perspectiva. A operação baseia-se na distância "d" entre a origem do sistema de coordenadas visuais e o plano estabelecido para receber a projeção. Para tal, basta aplicarem-se as equações abaixo às coordenadas visuais dos vértices de cada um dos objetos a serem exibidos :

$$\frac{Xv'}{d} = \frac{Xv}{Zv} \quad ==> \quad Xv' = \frac{Xv}{Zv / d}$$

$$\frac{Yv'}{d} = \frac{Yv}{Zv} \quad ==> \quad Yv' = \frac{Yv}{Zv / d}$$

Alternativamente, esta operação pode ser efetuada através da notação matricial das equações acima, representada pela matriz a seguir :

$$Mp = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

No sistema João-de-Barro as transformações de visualização, inclusive a projeção em perspectiva, foram implementadas utilizando-se notações matriciais. Entretanto, mesmo não sendo fiel ao procedimento implementado, a rotina abaixo utiliza a aplicação direta das equações apresentadas acima e exemplifica um procedimento para projeções em perspectiva.

```
EFETUA_PERSPECTIVA ( objeto, distância )
Início
  Para cada vértice do objeto faça
    X_VISUAL = X_VISUAL / ( Z_VISUAL / distância )
    Y_VISUAL = Y_VISUAL / ( Z_VISUAL / distância )
    Z_VISUAL = Z_VISUAL / ( Z_VISUAL / distância )
  Fim-para
Fim
```

4.16. O Algoritmo do Pintor

Neste item, descreve-se o algoritmo do "pintor" como um algoritmo de classificação e exibição de faces largamente utilizado para determinados casos de visualização de modelos geométricos. O algoritmo foi implementado no sistema João-de-Barro como ferramenta necessária

às funções de escondimento de faces ocultas e iluminação em "flat-shading", tendo sido desenvolvido com base nas idéias expostas em [Newm81], [Fole82], [Hear86] e [Picc88].

De acordo com a sistemática proposta pelo algoritmo, as faces dos objetos são classificadas de acordo com a sua distância em relação ao observador. Uma vez classificadas, a faces são exibidas em ordem decrescente de distância, ou seja, primeiro as faces mais distantes, depois as mais próximas. Dessa forma, as faces vão-se superpondo umas às outras, de modo que as porções dos objetos obscurecidas por determinadas faces não estarão presentes na cena final. Esse procedimento é análogo ao de um artista que pinta sobre uma tela, pintando primeiramente as porções mais afastadas da cena e, consecutivamente, sobrepondo-as com as porções mais próximas.

A idéia apresentada pelo algoritmo do "pintor" pode ser implementada de diversas maneiras e abordagens. No sistema João-de-Barro, a implementação adotada resume-se em criar uma lista auxiliar que contenha todas as faces da cena. Inicialmente, são eliminadas da lista aquelas faces que possuem o vetor normal apontando para o sentido oposto ao observador ("back-faces"), pois essas faces podem ser consideradas, de antemão, como invisíveis. A partir daí, um procedimento de classificação encarrega-se de ordenar as faces da lista por ordem decrescente, de acordo com a distância de cada face ao observador. Nesse ponto, as faces já estão classificadas e prontas para serem exibidas. Os subítens a seguir detalham cada um dos passos envolvidos no algoritmo implementado.

4.16.1. A Construção da Lista de Profundidade

O primeiro passo para a execução do algoritmo do "pintor" consiste na construção de uma lista auxiliar, denominada "lista de profundidade", contendo as faces de todos os objetos presentes na cena corrente. Cada nó armazena um apontador para a face em questão, os valores mínimos e máximos presentes nas coordenadas x_visual, y_visual e z_visual da face, as componentes A, B, C e D determinantes da equação do plano da face, um "flag" de controle de superposição cíclica, além de apontadores para outros nós da lista. O conteúdo dos nós da lista é apresentado pelas definições a seguir extraídas diretamente do programa fonte :

```
typedef struct tipo_plano {
  double a;
  double b;
  double c;
  double d;
```

```
} plano;

typedef struct tipo_stru_profundidade {
  struct tipo_stru_face   *ap_face;
  double                  x_visual_min;
  double                  x_visual_max;
  double                  y_visual_min;
  double                  y_visual_max;
  double                  z_visual_min;
  double                  z_visual_max;
  plano                   eq_plano;
  char                    super_cicl;
  struct tipo_stru_profundidade *ap_anterior;
  struct tipo_stru_profundidade *ap_proximo;
} stru_profundidade;
```

A rotina em pseudo-código a seguir ilustra um procedimento semelhante ao utilizado no sistema para a construção da lista de profundidade :

```
MONTA_ESTRUTURA_PROFUNDIDADE ()
Início
 Para cada objeto presente na cena
  Para cada face do objeto faça
    cria_nó_profundidade ( ap_prof )
    ap_prof->x_visual_min = menor valor de x_visual da face
    ap_prof->y_visual_min = menor valor de y_visual da face
    ap_prof->z_visual_min = menor valor de z_visual da face
    ap_prof->x_visual_max = maior valor de x_visual da face
    ap_prof->y_visual_max = maior valor de y_visual da face
    ap_prof->z_visual_max = maior valor de z_visual da face
    ap_prof->eq_plano = calcula_equação_plano ( face )
    ap_prof->super_cicl = NAO
  Fim-para
 Fim-para
Fim
```

4.16.2. A Classificação da Lista de Profundidade

Uma vez construída a lista de profundidade, dá-se início à ordenação dessa lista de acordo com as distâncias entre cada face e o observador da cena, verificando-se a existência de superposição entre faces. Essa ordenação é feita testando-se as faces entre si, determinando quais as faces que se sobrepõem e quais delas devem ser exibidas primeiramente em função das distâncias calculadas. Os passos listados a seguir esboçam as idéias básicas utilizadas para ordenar a lista de profundidade :

Obs : Vale lembrar que o sistema de coordenadas definido para o sistema João-de-Barro é construído segundo a regra de mão direita, onde as coordenadas do eixo "Z" crescem em direção ao observador da cena.

1) Por motivo de economia de processamento, antes de se proceder à ordenação, são eliminadas todas as "back-faces", ou seja, aquelas faces cujos vetores normais apontam em sentido oposto ao observador. Todas as faces nessa situação podem ser consideradas com invísiveis e não necessitam ser incluídas nos procedimentos de classificação. A detecção das "back-faces" é feita testando-se o sinal da componente "C" da equação do plano visual da face em questão. Caso a componente "C" seja menor que zero, a face consiste numa "back-face".

2) Num segundo passo, as faces presentes na lista de profundidade são classificadas por ordem crescente da variável Z_VISUAL_MIN. Essa ordenação procura estabelecer uma ordem prévia das faces, segundo a distância de cada uma em relação ao observador.

3) Com a lista previamente ordenada segundo os valores mínimos das coodenadas Z_VISUAL, as faces da lista são comparadas duas a duas, efetuando-se testes mais elaborados para detectar os casos onde ocorre superposição de faces. Nesses casos, determina-se, também, quais são as faces mais próximas e as mais afastadas do observador. A seqüência de testes é apresentada nos ítens 4 a 8. Para tal, considerem-se Face1 e Face2 como as duas faces envolvidas nos testes.

4) TESTE 1 : Se Face1.z_max <= Face2.z_min, então Face1 está garantidamente mais afastada que a Face2, devendo ser exibida primeiramente.

5) TESTE 2 : Se Face1.x_max <= Face2.x_min ou Face1.x_min >= Face2.x_max, as faces não se sobrepõem, tornando irrelevante a ordem na qual elas são exibidas.

6) TESTE 3 : Se Face1.y_max <= Face2.y_min ou Face1.y_min >= Face2.y_max, as faces também não se sobrepõem, podendo ser exibidas em qualquer seqüência.

7) TESTE 4 : Se todos os testes anteriores tiverem sido inconclusivos, verifica-se em qual dos semi-espaços da Face2 (mais próximo ou mais afastado do observador) encontra-se a Face1. Para isso, basta substituirem-se as coordenadas X, Y e Z de cada um dos vértices da Face1 na equação do plano da Face2. Se alguma das substituições satisfizer a equação Ax + By + Cz + D > 0, conclui-se que a Face1 superpõe a Face2, devendo ser exibida primeiramente.

8) TESTE 5 : Como último teste, verifica-se em qual dos semi-espaços da Face1 (mais próximo ou mais afastado do observador) encontra-se a Face2. Para tal, basta substituirem-se as coordenadas X, Y e Z de cada vértice da Face2 na equação do plano da Face1. Se todas as substituições satisfizerem a equação Ax + By + Cz + D >= 0, conclui-se que a Face2 é superposta pela Face1.

9) O "flag" de superposição cíclica protege o algoritmo de entrar em "loop", garantindo que duas faces trocadas de posição na lista, devido à superposição, não sejam mais trocadas entre si.

Para o controle da etapa de ordenação, foi implementado um procedimento semelhante ao apresentado abaixo em pseudocódigo. Outras rotinas associadas a este algoritmo também são apresentadas visando a um melhor entendimento.

```
ORDENA_ESTRUTURA_PROFUNDIDADE ()
Início
 elimina_back_faces()
 executa_sort_z()
 Face1 = primeira face da lista de profundidade
 Enquanto Face1->proxima <> FIM DA LISTA faça
  Face2 = Face1->proxima
  Se testa_minmax_z ( Face1 , Face2 ) = SIM
   Se testa_minmax_x ( Face1 , Face2 ) = SIM
    Se testa_minmax_y (Face1 , Face2 ) = SIM
     Se face1_superpõe_face2 ( Face1 , Face2 ) = NAO
      Se face2_superposta_face1 ( Face1 , Face2 ) = NAO
       Se Face2->supercíclica = NAO
         inverte posição dos nós de Face1 e Face2 na
         lista de profundidade
       Fim-se
      Fim-se
     Fim-se
    Fim-se
```

```
  Fim-se
  Face1 = Face1->proxima
 Fim-enquanto
Fim

ELIMINA_BACK_FACES ()
Início
 Para cada face da lista de profundidade
  Se face->eq_plano.c < 0
   elimina_nó_profundidade ( face )
  Fim-se
Fim

EXECUTA_SORT_Z ()
Início
 face1 = primeira face da lista de profundidade
 Enquanto face1->próxima <> FIM DA LISTA faça
  face2 = face1->próxima
  Enquanto face2 <> FIM DA LISTA faça
   Se face1->z_min > face2->z_min
     inverta posição da face1 com face2
     na lista de profundidade
   Fim-se
   face2 = face2->próxima
  Fim-enquanto
  face1 = face1->próxima
 Fim-enquanto
Fim

TESTA_MINMAX_Z ( face1 , face2 )
Início
 Se face1->z_max <= face2->z_min
   retorne NÃO
 senão
  retorne SIM
Fim

TESTA_MINMAX_X ( face1 , face2 )
Início
```

```
  Se face1->x_max <= face2->x_min  ou
    face1->x_min >= face2->x_max
      retorne NÃO
  senão
      retorne SIM
Fim

TESTA_MINMAX_Y ( face1 , face2 )
Início
  Se face1->y_max <= face2->y_min  ou
    face1->y_min >= face2->y_max
      retorne NÃO
  senão
      retorne SIM
Fim

FACE1_SUPERPÕE_FACE2 ( face1 , face2)
Início
  A = face2->eq_plano.a
  B = face2->eq_plano.b
  C = face2->eq_plano.c
  D = face2->eq_plano.d
  Para cada vértice da face1 faça
    x = vértice->x_visual
    y = vértice->y_visual
    z = vértice->z_visual
    valor = Ax + By + Cz + d
  Fim-para
  Se todos "valor" forem <= 0
    retorne SIM
  senão
    retorne NÃO

Fim

FACE2_SUPERPOSTA_FACE1 ( face1 , face2 )
Início
  A = face1->eq_plano.a
  B = face1->eq_plano.b
```

```
  C = face1->eq_plano.c
  D = face1->eq_plano.d
  Para cada vértice da face2 faça
    x = vértice->x_visual
    y = vértice->y_visual
    z = vértice->z_visual
    valor = Ax + By + Cz + d
  Fim-para
  Se todos "valor" forem >= 0
    retorne SIM
  senão
    retorne NÃO
Fim
```

4.17. A Exibição com Escondimento de Faces Ocultas

O escondimento de faces ocultas foi implementado no sistema João-de-Barro como um dos recursos de visualização dos modelos construídos. Conforme abordado no item 2.7, o escondimento de porções ocultas dos objetos permite a visualização dos mesmos com uma dose maior de realismo, eliminando certas ambiguidades apresentadas pelas exibições em "wire-frame" (arame).

Existem diversos algoritmos e técnicas para a eliminação de faces ocultas. No sistema João-de-Barro, adotou-se uma estratégia baseada no algoritmo do pintor, descrito no item 4.16, onde as faces são classificadas por ordem decrescente de distância ao observador, exibidas com contorno na sua cor_matiz e preenchidas com a cor de fundo da janela de exibição.

Uma vez disponível o algoritmo do pintor, a implementação do escondimento de faces ocultas resume-se em montar um procedimento de controle que executa os passos do algoritmo e solicita a exibição de cada face com as devidas cores de contorno e preenchimento. Tal procedimento pode ser ilustrado pela seguinte rotina em pseudocódigo :

```
EFETUA_ESCONDIMENTO_FACES ()
Início
  monta_estrutura_profundidade()
  ordena_estrutura_profundidade()
  Para cada face da lista de profundidade faça
```

```
    cor_contorno = face->cor_matiz
    cor_fundo    = cor_fundo_janela_exibição
    exibe_face ( cor_contorno , cor_fundo )
  Fim-para
  elimina_lista_profundidade()
Fim
```

4.18. A Exibição em Flat Shading

A capacidade de exibição em "flat shading" constitui mais um recurso de visualização implementado no sistema João-de-Barro visando a um maior realismo e qualidade na exibição dos modelos construídos.

"Flat Shading", também conhecido como "Constant Shading", constitui um método de iluminação de modelos inserido em estudos de síntese de imagens. De acordo com esse método, um dado modelo é iluminado calculando-se uma intensidade de cor constante para cada uma de suas faces. Os efeitos de iluminação são empregados através das diferentes tonalidades entre as faces do modelo. [Newm81], [Fole82], [Hear86], [Thal87] e [Gome90] são referências indicadas para consulta sobre iluminação em "flat shading".

De uma forma geral, a implementação de procedimentos de iluminação envolvem dois elementos distintos : o método e o modelo de iluminação. Dentre os métodos mais clássicos de iluminação estão "Flat Shading", "Gouraud Shading", "Phong Shading", métodos de radiosidade, entre outros. Por sua vez, o modelo de iluminação define as equações a serem utilizadas para o cálculo das intensidades de cores para os diversos pontos do espaço. Essas equações podem envolver variáveis diversas, tais como coeficientes de reflexão, ângulos de incidência da luz sobre os objetos, intensidade das fontes de luz, entre outros.

O modelo de iluminação utilizado no sistema João-de-Barro é expresso pela função de intensidade de luz $I_p = (I_a K_a + I_p K_d) . \cos (\text{theta})$, onde :

I_a = intensidade da luz ambiente
K_a = coeficiente de reflexão da luz ambiente
I_p = intensidade da fonte de luz
K_d = coeficiente de reflexão difusa do objeto
theta = ângulo de incidência entre os raios de luz e os vetores normais das faces do objeto

A partir dessa equação, calcula-se a intensidade de luz I_p para cada face dos objetos presentes na cena. Algumas funções de normalização determinam o código de cada cor na palheta de cores do sistema, em função da cor-matiz da face e da intensidade I_p calculada. A fonte de luz implementada consiste numa fonte pontual, com posição e intensidade previamente convencionados pelo sistema.

Os passos descritos a seguir esquematizam os procedimentos adotados e implementados para os efeitos de iluminação :

1) O sistema define, previamente, uma fonte de luz pontual, com coordenadas X,Y,Z e intensidade estabelecidas por convenção. O estado atual da implementação do sistema não permite que o usuário altere essas definições.

2) Inicialmente, é executado o algoritmo do pintor, descrito no item 4.16, para a ordenação das faces e serem exibidas.

3) Para cada face a ser exibida, são executados os passos de 4 a 8.

4) Define-se um raio de luz na forma de um vetor, com origem na fonte de luz e término no baricentro da face em questão.

5) Calcula-se o ângulo (theta) formado entre o raio de luz e o vetor normal da face.

6) Calcula-se a intensidade de luz da face através da equação $I_p = (I_a K_a + I_p K_d) . \cos (theta)$.

7) Aplica-se uma função de normalização que utiliza a cor matiz da face e a intensidade I_p calculada, a fim de fornecer o devido código da cor presente na palheta de cores do sistema.

8) Efetua-se a exibição da face com o contorno e o preenchimento na cor final obtida.

Uma visão mais sistemática do algoritmo implementado pode ser obtida pelo procedimento apresentado a seguir em pseudocódigo :

```
EFETUA_FLAT_SHADING ()
Início
  monta_estrutura_profundidade ()
  ordena_estrutura_profundidade ()
  Para cada face da lista de profundidade faça
    calcula_baricentro_face ( face )
    raio_luz = vetor entre a fonte de luz e o baricentro
    normal_face = obtem_normal_face ( face )
```

```
    theta = calcula_ângulo_vetores ( raio_luz , normal_face )
    Ip = ( Ia * Ka ) + ( Kd * Ip ) * cos (theta)
    cor_final = normaliza_intensidade_cor ( Ip , face )
    exibe_face ( face , cor_final )
    preenche_face ( face , cor_final )
  Fim-para
  elimina_estrutura_profundidade ()
Fim
```

4.19. A Exibição em Zoom

Um outro recurso de visualização presente no sistema consiste na exibição em zoom. Conforme apresentado no item 3.8.2, a função de zoom exibe a cena corrente de modo a ocupar toda a área da tela do computador. Esse recurso se mostra útil em ocasiões onde se deseja uma visão ampliada da cena, ou quando se pretende fotografar a mesma sem a presença das demais porções da interface do sistema.

O desenvolvimento da função de zoom consiste na elaboração de funções de normalização, que projetam o conteúdo da janela de exibição numa nova janela que abrange toda a extensão da tela do monitor, mantendo todas as proporções da cena. Por consistir numa função de visualização, a função de zoom age somente sobre as coordenadas visuais dos modelos.

A função de zoom é uma das funções classificadas como funções especiais do sistema, e está disponível a qualquer momento, independentemente do menu corrente, de acordo com as características expostas no capítulo 3. Também conforme já apresentado, a função de zoom exibe as cenas utilizando alguns dos padrões de visualização que tenham sido eventualmente definidos pelo usuário, tais como exibição dos eixos de coordenadas, escondimento de faces ocultas e iluminação. Para isso, a função verifica o conteúdo de certas variáveis de controle do sistema, invocando as rotinas de visualização necessárias de acordo, com as novas coordenadas visuais obtidas. Ao ser suspensa pelo acionamento de alguma tecla, a função aplica a normalização reversa sobre as coordenadas visuais, restaurando o estado original dos modelos.

Capítulo 5

Implementação das Operações Booleanas

5. IMPLEMENTAÇÃO DAS OPERAÇÕES BOOLEANAS

5.1. Apresentação

A capacidade de realização de operações booleanas é uma das principais características do sistema João-de-Barro. Essas operações de união, interseção e subtração entre modelos somam ao sistema as vantagens das representações CSG, tais como abrangência de domínio e facilidade na criação de modelos complexos. Devido à sua importância, a implementação dessas operações merece um capítulo à parte neste texto.

As operações booleanas de união, interseção e subtração envolvem sempre dois objetos. Dados dois objetos A e B (ver figura 5.1a), a **união** entre eles resulta em um objeto final composto de todos elementos componentes do objetoA adicionados de todos os elementos do objetoB. Uma operação de união é exemplificada na figura 5.1b. Por sua vez, o objeto resultante de uma operação de **interseção** entre o objetoA e o objetoB consiste nas porções comuns aos dois objetos, ou seja, às porções que pertencem tanto ao objetoA quanto ao objetoB. O resultado de uma operação de interseção pode ser visualizado na figura 5.1c. Finalmente, uma operação de subtração entre o objetoA e o objetoB é efetuada, excluindo-se, do objetoA, todas as porções pertencentes ao objetoB. O objeto resultante consiste nas porções restantes ao objetoA. É importante observar que a operação booleana de subtração não é comutativa.

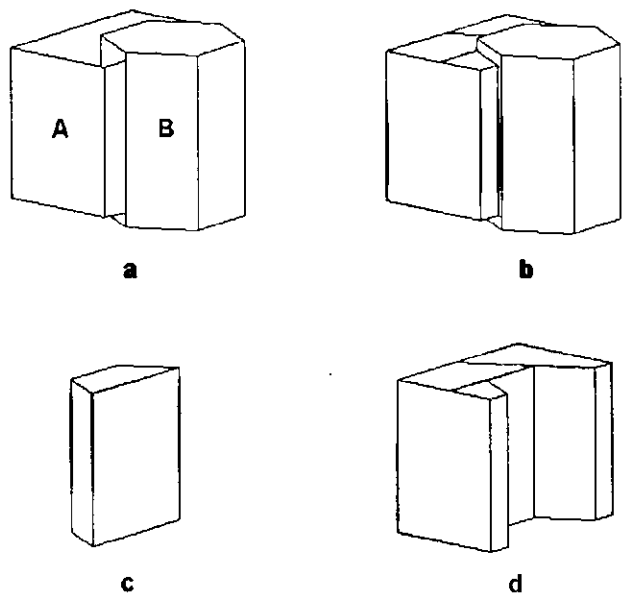


Figura 5.1 : Operações booleanas regularizadas de união, interseção e subtração.

Obs : Na figura 5.1, as faces e arestas aparentemente excedentes são criadas por mecanismos pertinentes ao algoritmo implementado. Tais mecanismos são descritos em detalhe ao longo deste capítulo.

Nas representações como "Boundary Representation", onde um sólido pode ser representado pela sua fronteira, operações booleanas podem ser efetuadas sobre modelos agindo-se somente sobre as fronteiras dos mesmos. Ao se realizarem operações booleanas sobre objetos sólidos, deve-se atentar para o detalhe de que as operações booleanas convencionais não são perfeitamente adequadas. Como pode ser visto na figura 5.2, a interseção entre objetos A e B resulta num objeto com uma face solta ("dangling face"), comprometendo a consistência matemática dos objeto criado. A mesma figura apresenta o resultado regularizado da mesma operação, Para contornar problemas dessa natureza, utilizam-se as chamadas **operações booleanas regularizadas**, que garantem a consistência dos sólidos resultantes. Essas operações são representadas por um sinal de asterisco ("*"), adicionado ao símbolo referente à operação (união, interseção ou subtração). Essas questões são abordadas mais detalhadamente em [Requ80] e [Laid86].

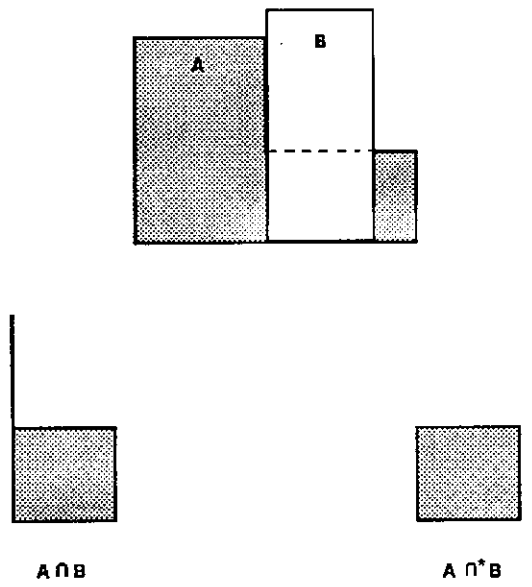


Figura 5.2 : Resultados de uma operação booleana de interseção, com abordagem convencional e regularizada.

Na literatura especializada, existem algumas publicações que enfocam a implementação de operações booleanas em modelos construídos com Boundary Representation. Dentre elas podemos citar [Tilo80a], [Tilo80b], [Requ85], [Laid86], [Dech88], [Flaq88] e [Xiny88].

O algoritmo implementado no sistema João-de-Barro é baseado na idéia publicada por [Laid86]. Nesse artigo, é apresentado em detalhe um algoritmo que envolve estruturas de dados bastante semelhantes às utilizadas no sistema João-de-Barro. Outro ponto favorável à escolha desse algoritmo como base, foi o fato da compatibilidade de resultados, ou seja, os modelos resultantes do algoritmo de [Laid86] são perfeitamente compatíveis com as técnicas utilizadas e os propósitos do sistema João-de-Barro.

Na implementação do algoritmo, foram introduzidas algumas modificações na idéia original de [Laid86]. Tais modificações não alteraram a concepção básica do algoritmo; porém, simplificaram a sua implementação e, acredita-se, diminuíram o custo computacional de sua execução.

5.2. Visão Geral do Algoritmo

A execução de uma operação booleana envolve, sempre, dois objetos distintos. Algumas condições essenciais para o funcionamento do algoritmo referem-se à construção dos modelos envolvidos. As operações booleanas podem ser realizadas sobre modelos sólidos que sigam as propriedades formais descritas no item 2.3. É necessário, também, que todas as faces de ambos os modelos sejam construídas com suas arestas definindo vetores orientados no sentido anti-horário, possibilitando o cálculo de vetores normais a essas faces. Além disso, todas as faces dos modelos envolvidos deverão ser convexas, apesar dos modelos em si poderem ser côncavos ou convexos.

Alguns sistemas de modelagem implementam rotinas de verificação das chamadas **regras de integridade**, que verificam as características dos modelos envolvidos antes de efetuar a operação booleana. Rotinas dessa natureza são úteis para a detecção prévia da inadequabilidade dos modelos e prevenção contra resultados desastrosos.

Obs : Durante a exposição do algoritmo, os objetos envolvidos serão identificados como **objetoA** e **objetoB**, bem como determinadas faces desses objetos como **faceA** (face do objetoA) e **faceB** (face do objetoB).

A execução do algoritmo se dá em três grandes etapas : **subdivisão** dos objetos, **classificação** de faces e vértices e **eliminação** de faces e vértices. Cada uma dessas etapas apresenta, ainda, uma série de subetapas.

A primeira etapa do algoritmo, destinada à **subdivisão** dos objetos envolvidos na operação, tem como finalidade identificar as faces de cada objeto que se interceptam com uma ou mais faces do outro objeto, e subdividí-las, de modo a eliminar a interseção entre essas faces. Para cada interseção detectada entre duas faces (faceA e faceB), ambas são subdivididas ao longo da reta de interseção, dando origem a quatro faces filhas (duas faces filhas por cada face original) que não se interceptam (salvo a interseção ao longo de uma aresta, que não será relevante para efeitos deste algoritmo). Uma vez efetuado este procedimento para todas as faces do objetoA e do objetoB, ter-se-à como resultado ambos os objetos subdivididos, compostos por um novo subconjunto de faces que não se interceptam.

Com os objetos subdivididos, dá-se início ao procedimento de **classificação**, que verifica e rotula as faces e vértices de cada objeto de acordo com a sua posição em relação ao outro objeto. As classificações possíveis são :

UNKNOWN : para faces e vértices de ambos os objetos ainda não classificados.
INSIDE : para faces e vértices do objetoA que se encontrem no interior do objetoB e vice-versa.
OUTSIDE : para faces e vértices do objetoA que se encontrem no exterior do objetoB e vice-versa.
BOUNDARY : para faces e vértices do objetoA que se encontrem na fronteira (ver item 2.3) do objetoB e vice-versa.

Obs : A classificação BOUNDARY, quando atribuída a faces, é desmembrada em dois casos especiais (SAME e OPPOSITE), de acordo com a direção do vetor normal da mesma. Esse procedimento é explicado em detalhe no item 5.5.

Durante os procedimentos de classificação, são criadas listas contendo informações sobre adjacência de vértices, que indicam, para cada vértice, quais são os vértices a ele ligados por arestas. Essas listas são bastante úteis para a otimização da etapa.

Uma vez classificados as faces e os vértices dos objetos, é iniciada a etapa de **eliminação**. Durante essa etapa são eliminados aquelas faces e vértices que não deverão pertencer ao objeto final resultante da operação. Essa eliminação é feita de acordo com a classificação da face/vértice, da operação em andamento (união, interseção ou subtração) e do objeto (A ou B) ao qual a face/vértice pertence. Ao final dessa etapa, restará um conjunto de faces e vértices que comporão o objeto final da operação.

O algoritmo é finalizado executando-se alguns procedimentos auxiliares, tais como inversão dos vetores normais das faces restantes no objetoB, no caso de se estar efetuando uma operação de subtração; além da fusão das porções restantes em um único objeto, obtendo-se o objeto final resultante da operação.

A rotina em pseudocódigo apresentada a seguir ilustra o procedimento que controla a execução das operações booleanas :

```
OPERAÇÃO_BOOLEANA ( objetoA , objetoB , operação )
Início
  inicA = início da lista de faces do objetoA
  fimA = final da lista de faces do objetoA
  inicB = início da lista de faces do objetoB
  fimB = final da lista de faces do objetoB
  subdivide_objetos ( inicA , fimA , inicB , fimB , "S" )
  cria_lista_adjacencias ( objetoA )
  cria_lista_adjacencias ( objetoB )
  classifica_regiões ( objetoA , objetoB )
  classifica_regiões ( objetoB , objetoA )
  elimina_lista_adjacências ( objetoA )
  elimina_lista_adjacências ( objetoB )
  elimina_faces_vértices ( objetoA , operação )
  elimina_faces_vértices ( objetoB , operação )
  Se operação = SUBTRAÇÃO
    inverte_vetores_normais ( objetoB )
  Fim-Se
  efetua_anexação_objetos ( objetoA , objetoB )
Fim
```

Os procedimentos de cada uma das etapas do algoritmo são descritos em detalhe nos tópicos seguintes.

5.3. Identificando Interseções entre Objetos

Os procedimentos de identificação de interseções entre objetos são invocados como subtarefa da etapa de subdivisão. Os procedimentos comparam duas faces (faceA e faceB), verificam a existência de interseção entre elas, calculam a reta de interseção e determinam, para cada face, os pontos de interseção entre a reta calculada e as arestas de cada uma das duas faces. Uma função de controle retorna os valores "A", "B", AMBAS ou NENHUMA, identificando quais das faces A e B são interceptadas. Os pontos de interseção são usados posteriormente pelos procedimentos de subdivisão.

O passo inicial é verificar se as duas faces em questão se interceptam. Para efeitos deste algoritmo, duas faces se interceptam se existir alguma aresta de uma das faces que cruze o interior da outra face, estejam ou não a aresta e a face interceptada no mesmo plano. Não são consideradas como interseções aquelas situações onde as faces se cruzam somente ao longo de arestas ou em vértices comuns às duas faces. Também não são consideradas as situações onde uma face é tocada somente por um vértice de da outra. Além disso, por convenção, faces coplanares nunca se interceptam.

O primeiro teste para verificar a existência de interseção, aqui apelidado de **Teste dos Semiespaços**, procede da seguinte forma :

1) Dadas faceA e faceB, calcula-se a equação do plano ao qual pertence a faceB, na forma Ax + By + Cz + D = 0. O plano obtido será aqui referenciado por planoB.

2) Calcula-se a distância entre cada vértice da faceA e o planoB.

3) Caso todas as distâncias tenham o mesmo sinal (positivo ou negativo), conclui-se que todos os vértices da faceA estão no mesmo semiespaço definido pelo planoB. Conseqüentemente não existe interseção entre as faces e pode-se encerrar o procedimento.

4) Caso existam vértices da faceA com distâncias ao planoB de sinais contrários, caracteriza-se uma situação em que a faceA cruza o plano da faceB, levantando-se a possibilidade de existência de interseção entre as faces.

5) Sob a suspeita de interseção entre as faces, repetem-se os procedimentos 1,2,3 e 4, trocando-se a faceA pela faceB e vice-versa.

Os procedimentos envolvidos nos testes dos semiespaços podem ser esboçados pelas seguintes instruções em pseudocódigo :

```
TESTA_SEMI_ESPAÇOS ( faceA , faceB )
Início
 planoB = obtem_equacao_plano ( faceB )
 semiespaco = calcula_distancia ( vertA [1] , planoB )
 trocou = NAO
 Para todos os vértices da faceA faça
 Início
   distancia = calcula_distancia ( vertA , planoB )
   Se ( distancia > 0 e semiespaco < 0 ) ou
     ( distancia < 0 e semiespaco > 0 )
```

```
    trocou = SIM
  Fim
  retorna trocou
Fim
```

Caso os testes dos semiespaços ainda tenham sido inconclusivos, indicando o cruzamento entre os planos das faces, dá-se prosseguimento a testes mais rebuscados :

1) Calcula-se as equações dos planos das faces A e B, obtendo-se planoA e planoB, respectivamente. Calcula-se também a reta R, consistindo na reta de interseção entre planoA e planoB.

2) Calcula-se, tanto para a faceA quanto para a faceB, os pontos de interseção entre a face e a reta R calculada. Cada face terá dois pontos de interseção com a reta R, dando origem a dois segmentos de reta, respectivamente segmentoA e segmentoB.

3) É utilizado, então, um procedimento para testar se os segmentos A e B se sobrepõem. Caso haja superposição, ambas as faces são interceptadas; caso contrário, não há interseção.

De forma global, os procedimentos de verificação de interseção entre faces descritos acima podem ser visualizados pelo seguinte pseudocódigo :

```
INTERCEPTA_FACES ( faceA , faceB )
Início
  interA = cruza_semi_espaço ( faceA , faceB )
  interB = cruza_semi_espaço ( faceB , faceA )
  Se ( interA = SIM  e  interB = SIM )
  Início
    planoA = obtem_equacao_plano ( faceA )
    planoB = obtem_equacao_plano ( faceB )
    retaR  = calcula_intersecao_planos ( planoA , planoB )
    segmentoA = calcula_pontos_intersecao ( faceA , retaR )
    segmentoB = calcula_pontos_intersecao ( faceB , retaR )
    teste = verifica_superposicao ( segmentoA , segmentoB )
    Se teste = SIM
      retorna "AMBAS"
    senão
      retorna "N"
  Fim-Se
```

```
  senão
  Início
    Se interA = SIM
      retorna "A"
    senão
      Se interB = SIM
        retorna "B"
      senão
        retorna "N"
  Fim-senão
Fim


CALCULA_PONTOS_INTERSECAO ( faceF , retaR )
Início
  primeiro = SIM
  Para cada aresta da faceF faça
  Início
    retaAux = obtem_reta ( aresta )
    ponto = calcula_intersecao_retas ( retaR , retaAux )
    Se ponto <> NULO
        Se primeiro = SIM
            ponto_inter_face1 = ponto
            primeiro = NAO
        senão
            ponto_inter_face2 = ponto
  Fim-Para
Fim
```

5.4. Subdividindo Objetos

A etapa de subdivisão é a primeira grande etapa na execução do algoritmo. O papel desta etapa consiste em identificar as faces de cada objeto que se interceptam com alguma face do outro objeto, calcular os pontos onde ocorre esta interseção e subdividir ambas as faces em duas faces-filhas cada. O objetivo desse procedimento é fazer com que não haja mais interseção entre as faces do dois objetos.

O procedimento de subdivisão implementado no sistema João-de-Barro difere daquele apresentado na idéia original. No algoritmo proposto por [Laid86], existe uma função de subdivisão responsável pela subdivisão de somente um dos objetos por vez. Nesse caso, essa função é invocada para a subdivisão do objetoA, reinvocada para a subdivisão do objetoB e executada uma terceira vez para subdividir novamente o objetoA, em função de novas interseções porventura geradas durante a subdivisão do objetoB.

Já no sistema João-de-Barro, os procedimentos de subdivisão são efetuados de maneira recursiva, permitindo que somente uma única execução desses procedimentos se encarregue de subdividir ambos os objetos, da seguinte forma :

1) O procedimento é iniciado recebendo, como parâmetros, quatro apontadores : inicA, inicB, fimA e fimB, que apontam para o início e o fim das listas de faces dos objetos A e B respectivamente. Também é passada uma variável RECURS (SIM/NÃO), que indica a continuação ou não da recursão. Na primeira chamada do procedimento, RECURS = SIM.

2) A partir do início das listas de faces de cada objeto, para cada faceA comparada com cada faceB, são executados os passos de 3 a 8.

3) Utilizar os procedimentos descritos no item 5.3, para verificar a existência de interseção entre a faceA e a faceB. Vale lembrar que este mesmo procedimento calcula os pontos onde porventura ocorra a interseção.

4) Se a faceA for interceptada, criar uma variável proxA = "próxima faceA na lista de faces" e proceder a subdivisão da faceA em duas faces-filhas, através do procedimento DIVIDE_FACE descrito em seguida.

5) Se a faceB for interceptada, criar uma variável proxB = "próxima faceB na lista de faces" e proceder a subdivisão da faceB, utilizando-se, também, o procedimento DIVIDE_FACE.

6) A subdivisão da faceA e da faceB geram, no total, quatro faces-filhas. Surge, então, a necessidade de se garantir que todas as faces-filhas geradas ao longo do processo de subdivisão serão também confrontadas com as devidas faces do outro objeto e, possivelmente, subdivididas em faces-netas, e assim por diante. Essa prevenção se faz necessária para evitar a permanência de novas interseções oriundas do surgimento de faces descendentes. Para sanar esse problema, executa-se o procedimento de subdivisão recursivamente, de acordo com o especificado no passo 7. A recursão é efetuada em somente um nível, ou seja, a primeira chamada recursiva se encarrega de quebrar a recursão. Essa execução recursiva do procedimento é responsável pelo tratamento de todas as faces-filhas da faceA, bem como de

suas descendentes, oriundas da primeira subdivisão da face mãe. As faces-filhas da faceB são tratadas após a quebra da recursão, ao longo da continuação do procedimento.

7) A chamada recursiva é feita da seguinte forma : caso RECURS seja igual a SIM, o procedimento é chamado passando-se os parâmetros inicA = proxB , fimA = fimB , inicB = inicA , fimB = proxA e RECURS = NAO.

8) Finalizada a execução recursiva, efetuar as atribuições inicA = proxA , inicB = "início da lista de faces do objetoB" , RECURS = SIM.

Os procedimentos envolvidos na etapa de subdivisão podem ser visualizados pela rotina em pseudocódigo a seguir :

```
SUBDIVIDE_OBJETOS ( inicA , fimA , inicB , fimB , recurs )
Início
 Para cada faceA do objetoA
  Para cada faceB do objetoB
   teste = intercepta_faces ( faceA , faceB )
   Se teste = "A" ou teste = "AMBAS"
     proxA = faceA->próxima   { face seguinte à faceA }
     divide_face ( faceA )
   Se teste = "B" ou teste = "AMBAS"
     proxB = faceB->próxima   { face seguinte à faceB }
     divide_face ( faceB )
   Se recurs = SIM
     subdivide_objetos ( proxB, fimB, inicA, proxA, NAO )
     inicA = proxA
     inicB = primeira face do objetoB
     recurs = SIM
  Fim-Para
 Fim-Para
Fim
```

Acredita-se que a abordagem aqui descrita, considerando-se especificamente o desenvolvimento deste sistema, tenha oferecido ganhos de eficiência e diminuição do custo computacional do algoritmo. Isso devido ao fato de se acreditar que, na implementação adotada, são realizados menos testes de interseção entre faces, bem como menos execuções de subdivisão de faces. Conforme mencionado anteriormente, [Laid86] afirma serem necessárias três etapas de

subdivisão distintas na sua proposta de algoritmo. Essa crença é ainda mais fortalecida pelo fato dos procedimentos de subdivisão serem extremamente custosos computacionalmente. Entretanto, o desenvolvimento deste trabalho não incluiu a implementação e avaliação da idéia original de [Laid86], tomada apenas como referência básica. Isso dificulta, portanto, qualquer afirmação categórica, em termos de se colocar uma ou outra abordagem como mais adequada ou eficiente.

5.5. Dividindo Faces em Faces-Filhas

Os procedimentos de divisão de faces são responsáveis por dividir uma determinada face de um dado objeto em duas faces-filhas, tomando-se como parâmetro dois pontos, pertencentes a arestas distintas da face, que determinam uma reta de "corte" da face.

Esses procedimentos diferem daqueles implementados na idéia básica apresentada em [Laid86]. No algoritmo implementado no sistema João-de-Barro, uma face sempre será dividida em duas faces-filhas, tomando-se, como eixo de corte, a reta determinada pela interseção entre duas faces. [Laid86] estabelece 17 (dezessete) possibilidades diferentes de divisão de faces, de acordo com a maneira como ocorre a interseção entre as faces.

Os procedimentos de divisão de faces adotados no sistema João-de-Barro se resumem da seguinte forma :

1) O procedimento é chamado recebendo, como parâmetros, um apontador "face_mãe", indicando a face a ser dividida, além dos pontos "ponto1" e "ponto2", definindo o eixo de corte da face.

2) Inicialmente, são criadas duas faces filhas (face_filha1 e face_filha2) nas estruturas de dados do sistema. Nesse ponto, são criadas somente as entradas das faces, não contendo ainda nenhuma aresta.

3) Para cada aresta da face_mãe, testar se o ponto1 está sobre a aresta. Caso negativo, inserir a aresta em questão na face_filha1, passando-se para o teste da próxima aresta.

4) Ao se identificar a aresta na qual se encontra o ponto1, verificar se esse ponto coincide com algum vértice já existente no objeto. Caso negativo, o ponto1 é inserido como um novo vértice do objeto em questão. Em caso afirmativo, o vértice coincidente é utilizado, evitando a duplicação de vértices. Em ambos os casos, é atribuída ao vértice uma classificação BOUNDARY, a ser utilizada nas etapas posteriores do algoritmo.

5) Em seguida, deve-se criar uma aresta na face_filha1, com vértice inicial igual ao vértice inicial da aresta corrente na face_mãe, e vértice final igual ao ponto1. É criada, também, uma aresta na face_filha2, com vértice inicial igual ao ponto1 e vértice final igual ao vértice final da aresta corrente na face_mãe.

6) A partir daí, para cada aresta da face_mãe, testar se o ponto2 pertence à aresta em questão. Caso negativo, inserir essa aresta na face_filha2, passando-se para o teste da aresta seguinte.

7) Ao se identificar a aresta que contém o ponto2, verificar se existe algum vértice no objeto coincidente com esse ponto. Em caso negativo, o ponto2 é inserido como um novo vértice do objeto em questão.

8) Criar uma aresta na face_filha2 com vértice inicial igual ao vértice inicial da aresta corrente na face mãe e vértice final igual ao ponto2. Analogamente, criar uma aresta na face_filha1 com vértice inicial igual ao ponto2 e vértice final igual ao vértice final da aresta corrente na face_mãe.

9) Inserir todas as arestas restantes da face_mãe na face_filha1.

10) Criar, então, uma aresta final na face_filha1, unindo o ponto1 ao ponto2, fechando essa face. Analogamente, a face_filha2 recebe uma aresta final, unindo o ponto2 ao ponto1, sendo também fechada.

11) Ao final do procedimento eliminar a face_mãe da lista de faces, permanecendo na estrutura de dados somente as faces_filhas.

É importante frisar que a divisão de uma dada face deve manter, nas faces filhas, a ordenação das arestas, definindo vetores ordenados no sentido anti-horário. Essa característica permite o cálculo do vetor normal a uma face, sendo fundamental para outras funções do sistema.

Os procedimentos de divisão de faces são apresentados a seguir, através de rotina em pseudocódigo :

```
DIVIDE_FACE ( objeto , face_mãe , ponto1 , ponto2 )
Início
  aresta = primeira aresta da face mãe
  teste = verifica_ponto_aresta ( ponto1 , aresta )
  Enquanto teste = NAO
    insere_nova_aresta ( face_filha1 , aresta )
    aresta = aresta->próxima   { aresta seguinte à corrente }
```

```
    teste = verifica_ponto_aresta ( ponto1 , aresta )
  Fim-Equanto
  Se não existe vértice coincidente a ponto1
    insere_novo_vértice ( objeto , ponto1 )
  Fim-Se
  aresta_aux = [ vértice inicial de aresta , ponto1 ]
  insere_nova_aresta ( face_filha1 , aresta_aux )
  aresta_aux = [ ponto1 , vértice final de aresta ]
  insere_nova_aresta ( face_filha2 , aresta_aux )

  teste = verifica_ponto_aresta ( ponto2 , aresta )
  Enquanto teste = NAO
    insere_nova_aresta ( face_filha2 , aresta )
    aresta = aresta->próxima   { aresta seguinte à corrente }
    teste = verifica_ponto_aresta ( ponto2 , aresta )
  Fim-Equanto
  Se não existe vértice coincidente a ponto2
    insere_novo_vértice ( objeto , ponto2 )
  Fim-Se
  aresta_aux = [ vértice inicial de aresta , ponto2 ]
  insere_nova_aresta ( face_filha2 , aresta_aux )
  aresta_aux = [ ponto2 , vértice final de aresta ]
  insere_nova_aresta ( face_filha1 , aresta_aux )

  Para cada aresta restante na face_mãe
    insere_nova_aresta ( face_filha1 , aresta restante )
  Fim-Para
  aresta_aux = [ ponto1 , ponto2 ]
  insere_nova_aresta ( face_filha1 , aresta_aux )
  aresta_aux = [ ponto2 , ponto1]
  insere_nova_aresta ( face_filha2 , aresta_aux )
  elimina_face ( objeto , face_mãe )
Fim
```

Conforme dito anteriormente, o processo aqui descrito efetua somente uma forma de divisão de faces (duas faces filhas com divisão ao longo da reta de interseção), diferentemente daquele adotado por [Laid86], que inclui 17 (dezessete) casos distintos de divisão. A maior simplicidade de implementação foi o fator principal que determinou a abordagem seguida pelo

sistema João-de-Barro, uma vez que os resultados obtidos se mostraram satisfatórios de acordo com os objetivos do sistema.

Existe, entretanto, uma questão bastante relevante que merece ser colocada. A forma como são divididas as faces no sistema João-de-Barro cria modelos com mais de dois vértices colineares, ou seja, mais de dois vértices pertencentes a uma mesma aresta. Essa característica não ocorre com os modelos gerados pelo algoritmo de [Laid86], podendo vir a ser indesejável em determinadas aplicações. Para os objetivos estabelecidos até o momento para o sistema João-de-Barro, a colinearidade de múltiplos vértices não apresenta qualquer inconveniente.

5.6. Classificando Regiões

Uma vez subdivididos ambos os objetos, através dos procedimentos descritos anteriormente, chega-se a uma situação onde não existem mais interseções entre as faces dos objetos. Também é interessante notar que, nesse ponto, cada face e vértice de um dado objeto pode ser classificado como estando no interior, exterior ou na fronteira do outro objeto. Chamamos de uma **região**, o conjunto de faces e vértices que possuem a mesma posição com relação ao outro objeto. Dessa forma, cada objeto possui uma região interna, uma região externa e uma região de fronteira. A etapa de classificação tem como papel atribuir, a cada elemento envolvido (faces e vértices), um tipo que identifique a sua posição em relação ao outro objeto. Os tipos utilizados são UNKNOWN, INSIDE, OUTSIDE, BOUNDARY, SAME e OPPOSITE, explicados em maior detalhe posteriormente.

Os procedimentos de classificação implementados no sistema João-de-Barro seguem a mesma abordagem sugerida por [Laid86]. Existe um procedimento denominado CLASSIFICA_REGIÕES, responsável pelo controle de toda a etapa de classificação. Esse procedimento conta com alguns acessórios, tais como uma rotina de classificação de faces, denominada CLASSIFICA_FACE, as listas de adjacências de vértices, além de uma outra rotina denominada CLASSIFICA_VÉRTICE.

A rotina CLASSIFICA_FACE, descrita em detalhe no item 5.8, é responsável por classificar uma determinada faceA em relação ao objetoB, e vice-versa. Essa rotina recebe como parâmetros a face a ser classificada e o objeto alvo da classificação. A rotina retorna um dos tipos possíveis para faces : INSIDE, OUTSIDE, SAME e OPPOSITE.

Outro acessório utilizado são as listas de adjacências de vértices. Cada vértice contém uma lista que indica quais são os seus vértices adjacentes, ou seja, aqueles vértices a ele

ligados através de arestas. A criação das listas de adjacências é invocada pelo procedimento geral denominado OPERAÇÃO_BOOLEANA descrito no item 5.2.

É implementado, também, um procedimento denominado CLASSIFICA_VÉRTICE, cujo papel é propagar recursivamente o tipo de um determinado vértice conhecido para todos os vértices adjacentes e pertencentes à mesma região. Esse processo de propagação de tipos foi adotado por razões de melhoria de desempenho. Isso porque uma possível abordagem para a etapa de classificação seria executar-se a rotina de classificação de faces para cada face dos objetos envolvidos, atribuindo-se os tipos das faces para os seus vértices componentes. Contudo, esse procedimento seria muito dispendioso, uma vez que a rotina CLASSIFICA_FACE é bastante onerosa computacionalmente. Valendo-se da idéia de que todos os vértices e faces de uma dada região possuem o mesmo tipo, executa-se a rotina de classificação de faces poucas vezes e propaga-se os tipos obtidos por toda a região em questão.

O procedimento CLASSIFICA_REGIÕES trata somente um objeto por vez, devendo ser executado separadamente para cada um dos objetos. A sistemática utilizada para a execução do procedimento pode ser apresentada da seguinte forma :

1) Todas as faces e vértices do objeto são inicializadas com o tipo UNKNOWN, exceto aqueles vértices marcados como BOUNDARY durante o processo de subdivisão (ver item 5.5). Os vértices de tipo BOUNDARY são utilizados como delimitadores entre as regiões interna e externa dos objetos.

2) Para cada face do objeto, cada uma sendo aqui referenciada por "face corrente" quando estiver sendo tratada, e para cada vértice da face corrente, efetuar os passos de 3 a 7.

3) Se tipo do vértice for UNKNOWN e o tipo da face também for UNKNOWN, não há informação suficiente para atribuir qualquer classificação. Nesse caso, deve-se invocar a rotina de classificação de faces para identificar o tipo da face corrente.

4) Se o tipo do vértice for UNKNOWN e o tipo da face for SAME ou OPPOSITE, chamar a rotina CLASSIFICA_VÉRTICE, passando como parâmetros o vértice corrente e o tipo BOUNDARY.

5) Se o tipo do vértice for UNKNOWN e o tipo da face for INSIDE ou OUTSIDE, chamar a rotina CLASSIFICA_VÉRTICE, passando o vértice corrente e o tipo da face como parâmetros.

6) Caso o tipo do vértice corrente seja diferente de UNKNOWN e diferente de BOUNDARY, se o tipo da face for UNKNOWN deve-se atribuir o tipo do vértice ao tipo da face.

7) Tratados todos os vértices da face corrente, se o tipo dessa face permanecer como UNKNOWN, executar a rotina de classificação de faces.

Os procedimentos descritos acima, pertencentes às funções CLASSIFICA_REGIÕES e CLASSIFICA__VÉRTICE, são apresentados a seguir na forma de pseudocódigo :

```
CLASSIFICA_REGIÕES ( objetoA , objetoB )
Início
 Para cada face do objeto
  Para cada vértice pertencente à face
   Se tipo do vértice = UNKNOWN
    Se tipo da face = UNKNOWN
     classifica_face ( face , objetoB )
    senão
     Se tipo da face = SAME ou OPPOSITE
      classifica_vértice ( vértice , BOUNDARY )
     senão
      classifica_vértice ( vértice , tipo da face )
    Fim-senão
   senão
    Se tipo face = UNKNOWN e tipo vértice <> BOUNDARY
     tipo da face = tipo do vértice
    Fim-Se
   Fim-senão
  Fim-Para
  Se tipo da face = UNKNOWN
   classifica_face ( face , objetoB )
 Fim-Para
Fim

CLASSIFICA_VÉRTICE ( vértice , tipo )
Início
  tipo do vértice = tipo
  Para cada vértice adjacente ao vértice corrente
   Se tipo do vértice_adjacente = UNKNOWN
    classifica_vértice ( vértice adjacente , tipo )
   Fim-Se
  Fim-Para
Fim
```

5.7. Classificando Faces

Conforme mencionado anteriormente, os procedimentos de classificação incluem uma rotina de classificação de faces, responsável por identificar a posição de uma determinada face em relação ao outro objeto envolvido na operação booleana. A rotina recebe, como parâmetros, a face a ser classificada e o objeto alvo da classificação, retornando o tipo atribuído à face.

Uma face pode receber 05 (cinco) tipos de classificação : UNKNOWN, INSIDE, OUTSIDE, SAME e OPPOSITE. O tipo UNKNOWN ("desconhecido") consiste na classificação inicial de todas as faces, antes de se executar a rotina de classificação. Os tipos INSIDE e OUTSIDE são atribuídos a faces do objetoA posicionadas no interior e no exterior do objetoB, respectivamente. Os tipos SAME e OPPOSITE são utilizados quando a face se encontra sobre a fronteira do objeto.

Quando uma dada faceA se encontra na fronteira do objetoB, existe ao menos uma faceB coincidindo no mesmo plano da faceA. Nesses casos, quando a faceA e a faceB têm seus vetores normais apontando na mesma direção, atribui-se a classificação SAME para a face. Caso os vetores normais estejam orientados em sentidos opostos, atribui-se o tipo OPPOSITE.

O funcionamento básico da rotina consiste em "disparar-se" um raio (uma reta, se preferir) com origem no baricentro da faceA, orientada do sentido de seu vetor normal, em direção a cada face do objetoB. Utilizando-se o vetor normal da faceB em questão, calcula-se o produto escalar entre os dois vetores e a distância entre o baricentro da faceA e o plano da faceB. A rotina visa encontrar a faceB mais próxima da faceA e que seja interceptada pelo raio disparado. A partir desses cálculos, além de outros procedimentos, é determinado o tipo da faceA. Essa sistemática pode ser detalhada de acordo com os seguintes passos :

1) A rotina é iniciada recebendo como parâmetros a face a ser classificada, aqui referenciada por faceA, e o objeto relativo à classificação, aqui denominado objetoB.

2) Calcula-se o baricentro e o vetor normal da faceA, aqui referenciado por vetorA.

3) Para cada faceB, executam-se os passos de número 4 a 10.

4) Calcula-se o baricentro, a equação do plano e o vetor normal da faceB, denominado vetorB.

5) Calcula-se o produto escalar entre o vetorA e o vetorB, bem como a distância (com sinal) entre o baricentro da faceA e o plano ao qual pertence a faceB.

6) Se a distância < 0 e o produto escalar <> 0, tem-se uma situação onde o raio disparado a partir da faceA tem origem após a faceB, não a interceptando. Deve-se, então, ignorar essa faceB e testar a seguinte.

7) Se a distância = 0 e o produto escalar = 0, tem-se uma situação onde o raio está sobre o mesmo plano que a faceB. Nesse caso, o raio deve ser desviado randomicamente e testado novamente para todas as facesB.

8) Se a distância > 0 e o produto escalar = 0, tem-se o caso em que o raio é paralelo ao plano da faceB. Essa face deverá ser, então, desconsiderada, continuando-se os testes com as faces seguintes.

9) Se a distância = 0 e o produto escalar <> 0, a origem do raio encontra-se sobre o plano da faceB. Se o raio disparado atravessar o interior da faceB, essa é a face mais próxima possível da faceA.

10) Se a distância > 0 e o produto escalar <> 0, o raio "perfura" o plano da faceB. Caso esta "perfuração" se dê no interior da faceB, e caso não exista nenhuma interseção previamente salva com distância menor que a atual, deve-se salvar essa interseção como uma possível interseção mais próxima.

11) Se não houve nenhuma interseção, retornar o tipo OUTSIDE.

12) Para a faceB mais próxima da faceA e interceptada pelo raio criado, efetua-se os testes abaixo.

13) Se a distância = 0 e o produto escalar > 0, retornar o tipo SAME.

14) Se a distância = 0 e o produto escalar < 0, retornar o tipo OPPOSITE.

15) Se a distância <> 0 e o produto escalar > 0, retornar o tipo INSIDE.

16) Se a distância <> 0 e o produto escalar < 0, retornar o tipo OUTSIDE.

Essa rotina pode ser apresentada, alternativamente, pelo pseudocódigo a seguir :

```
CLASSIFICA_FACE ( faceA , objetoB )
Início
 baricA = calcula_baricentro ( faceA )
 vetorA = calcula_vetor_normal ( faceA )
 Para cada faceB faça
  baricB = calcula_baricentro ( faceB )
  vetorB = calcula_vetor_normal ( faceB )
  planoB = calcula_equação_plano ( faceB )
  prod_esc = calcula_produto_escalar ( vetorA , vetorB )
  distância = calcula_dist_ponto_plano ( baricA , planoB )
  Se distância = 0 e prod_esc = 0
    desviar o raio por algum valor randômico e reiniciar os  testes com o novo raio.
  Se distância = 0 e prod_esc <> 0
    caso o raio atravesse o interior da faceB, esta é a face mais próxima, deve-se pular para o
    label" ATRIBUIÇÃO. Em caso contrário, desconsiderar esta face e continuar com os testes.
  Se distância > 0 e prod_esc <> 0
    caso o raio atravesse o interior da faceB e caso não exista nenhuma interseção previamente
    salva com distância menor que a atual, salvar a faceB corrente como uma possível interseção
    mais próxima.
 Fim-Para

Atribuição
  Se não houver interseções
   retornar o tipo OUTSIDE
 Para a face obtida como interseção mais próxima faça
  Se distância = 0
    Se prod_esc > 0
      retornar o tipo SAME
    senão
      retornar o tipo OPPOSITE
  senão
    Se prod_esc > 0
      retornar o tipo INSIDE
    senão
      Se prod_esc < 0
        retornar o tipo OUTSIDE
  Fim-senão
 Fim-Para
Fim
```

5.8. Eliminando Faces e Vértices

Uma vez encerrada a etapa de classificação, estando classificados as faces e vértices de ambos os objetos, dá-se início à ultima etapa do algoritmo : a eliminação de faces e vértices.

A rotina de eliminação é chamada recebendo como parâmetros o objeto a ser tratado e a operação booleana (união, interseção ou subtração) em andamento. São utilizados os tipos identificados para as faces e vértices do objeto em questão, além de duas tabelas de consulta. A etapa consiste simplesmente em percorrer as listas de faces e vértices do objeto, verificando-se, junto às tabelas de consulta, quais elementos deverão permanecer no objeto resultante da operação. As tabelas de consulta utilizadas são as seguintes :

Obs. 1 : A palavra "Sim" indica que o elemento consultado permanece no objeto final, equanto que "Não" indica a eliminação do elemento.

Obs. 2 : Todos os vértices com tipo BOUNDARY devem permanecer no objeto final. Para o tratamento de vértices, são válidas somente as colunas referentes aos tipos INSIDE e OUTSIDE.

Para o objeto A:

	INSIDE	OUTSIDE	SAME	OPPOSITE
UNIÃO	Não	Sim	Sim	Não
INTERSEÇÃO	Sim	Não	Sim	Não
SUBTRAÇÃO	Não	Sim	Não	Sim

Para o objeto B:

	INSIDE	OUTSIDE	SAME	OPPOSITE
UNIÃO	Não	Sim	Não	Não
INTERSEÇÃO	Sim	Não	Não	Não
SUBTRAÇÃO	Sim	Não	Não	Não

```
ELIMINA_FACES_VÉRTICES ( objeto , operação )
Início
  Para cada face do objeto faça
   teste = verifica_tabela ( objeto , tipo face , operação )
   Se teste = SIM
     elimina_face ( objeto , face )
  Fim-Para

  Para cada vértice do objeto faça
   teste = verifica_tabela ( objeto , tipo vert , operação )
   Se teste = SIM
     elimina_vértice ( objeto , vértice )
  Fim-Para
Fim


VERIFICA_TABELA ( objeto , tipo , operação )
Início
 Caso ( tipo )
   INSIDE : Se operação = UNIÃO ou
            ( operação =  SUBTRAÇÃO  e objeto = "A" )
            retorne SIM
          senão
            retorne NÃO

   OUTSIDE : Se operação = INTERSEÇÃO  ou
            ( operação = SUBTRAÇÃO e objeto = "B" )
            retorne SIM
          senão
            retorne NÃO

   SAME  : Se  operação = SUBTRAÇÃO e objeto = "B"
            retorne SIM
          senão
            retorne NÃO

   OPPOSITE : Se  operação = SUBTRAÇÃO e objeto = "A"
              retorne NÃO
```

senão
retorne SIM

BOUNDARY : retorne NÃO
Fim

5.9. Finalizando a Operação

A finalização do algoritmo requer, ainda, dois procedimentos : a inversão de alguns vetores normais e a fusão das porções restantes do objetoA e do objetoB em um único objeto final.

A inversão de vetores normais se faz necessária somente sobre as faces restantes do objetoB, no caso de se haver realizado uma operação de subtração. Esse procedimento é necessário para que os vetores normais passem a apontar para "fora" do objeto, e não para "dentro". Essa operação é realizada invertendo-se, simplesmente, o sentido de orientação das arestas de cada face, passando-as do sentido anti-horário para o sentido horário.

Como último passo, uma vez que neste momento restam duas estruturas de dados distintas com as porções restantes de cada objeto, deve-se fundir essas partes restantes em um único objeto, dando-se origem ao objeto final da operação. Existe um procedimento denominado EFETUA_ANEXAÇÃO_OBJETOS, elaborado para esse fim.

Capítulo 6

Conclusão

6. CONCLUSÃO

Por constituir um trabalho de pesquisa, alguns dos aspectos mais importantes do desenvolvimento do sistema João-de-Barro referem-se à análise dos resultados obtidos e à continuidade do trabalho em novos projetos. Neste capítulo de conclusão, são apresentados alguns dos aspectos mais relevantes em relação à elaboração deste trabalho, juntamente com algumas sugestões para a sua continuidade. O capítulo é apresentado em tópicos, onde cada um deles refere-se a algum ponto ou questão específica com que se deparou ao longo do desenvolvimento do projeto.

a) O sistema João-de-Barro foi desenvolvido tendo como base um equipamento PC 386DX, 33 Mhz, com co-processador aritmético 80387 sob ambiente MS-DOS. Foi utilizado o padrão de vídeo SVGA, operando a uma resolução de 640 X 480 pixels com 256 cores simultâneas. Apesar de se ter obtido os resultados desejados, esse ambiente foi considerado restrito. O desempenho da máquina, mesmo reforçado pela presença do co-processador matemático, deixa aparente a sobrecarga imposta pelo sistema em algumas operações e limita a implementação de algoritmos mais exigentes. Além disso, os 640 Kbytes de memória oferecidos pelo ambiente MS-DOS mostraram-se insuficientes, uma vez que foram comuns os "estouros" de memória quando da construção de cenas complexas. Para a continuidade deste trabalho, é recomendável a migração do sistema para outros ambientes operacionais, tais como estações de trabalho baseadas em arquitetura RISC com sistema operacional UNIX. Sistemas semelhantes ao João-de-Barro têm sido implementados positivamente em máquinas como Sun SparcStation, HP Apollo série 9000 e IBM RS6000. Além disso, a ampliação do sistema deverá prever o tratamento de outros dispositivos gráficos, tais como mesas digitalizadoras, "plotters", gravadores de vídeo, entre outros.

b) Totalmente escrita em linguagem C, a primeira versão do programa fonte contém aproximadamente 9.500 (nove mil e quinhentas) linhas de código distribuídas em 14 arquivos. O sistema foi implementado de modo a se mostrar transportável e modularizado, facilitando tarefas de expansão, manutenção e instalação em diferentes ambientes operacionais. Ao longo do trabalho, tarefas como essas foram realizadas eficientemente.

c) A interface implementada atendeu às expectativas iniciais, apresentando-se ágil e auto-explicativa. Essas características foram observadas ao longo da construção de diversos modelos com diferentes características. Além disso, a interface se mostrou facilmente transportável para diferentes padrões de vídeo, graças à sua implementação parametrizada. Entretanto, com o surgimento de novos padrões de interfaces gráficas, detectou-se a necessidade de se desenvolver uma nova interface para o sistema. Existem, atualmente, algumas ferramentas para esse fim, tal como a biblioteca ZINC 3.0 para ambiente DOS, além de produtos voltados para os padrões OSF/MOTIF, X WINDOWS, Sun OpenLook, Microsoft

Windows, IBM OS/2, entre outros. Em ambientes como esses, o sistema pode ser acrescido de alguns benefícios, principalmente devido aos recursos de multitarefa neles presentes. Assim, o usuário humano poderia, por exemplo, dispor de diversas janelas com vistas diferentes dos modelos, além de mais recursos para a manipulação dos mesmos. O aprimoramento da interface do sistema exige a implementação das funções de "ajuda (help)", ainda não disponíveis mas com inclusão já prevista no projeto. As funções de ajuda poderiam ser implementadas com um caráter "sensível ao contexto", uma vez que o sistema dispõe de estruturas de dados que controlam elementos como o menu corrente, a opção corrente dentro desse menu, situações de erro porventura atingidas, entre outras.

d) Os métodos de criação de primitivas oferecidos pelo sistema mostraram-se suficientes para a maioria das necessidades enfrentadas. Entretanto, a inclusão de novos métodos seria bastante interessante. Podem ser incluídos, por exemplo, métodos de "sweeping" e outros modelos matemáticos que criem modelos sólidos, Alguns sistemas chegam a implementar editores bidimensionais de curvas e perfis que, uma vez rotacionados ou transladados, produzem diversos tipos de primitivas sólidas. Pode-se notar que, em muitos casos, um conjunto mais elaborado de métodos automáticos para a criação de primitivas facilita o trabalho final do usuário, uma vez que diversos modelos passam a ser criados com um número menor de operações de transformação. Isso devido ao fato de se ter maior flexibilidade quando da criação das primitivas.

e) A escolha do esquema de representação consistiu numa das questões mais críticas no desenvolvimento do projeto. Apesar de uma certa clareza quanto à adoção de um esquema híbrido combinando B-Rep e CSG, restavam, ainda, algumas dúvidas em relação à construção das estruturas de dados que armazenariam os modelos. Chegou-se a cogitar, por exemplo, a possibilidade de se construir as árvores CSG explicitamente, compostas por diversos níveis de estruturas em B-Rep. Entretanto, devido a fatores como redundância e quantidade de memória ocupada (concisão), optou-se por manter as árvores CSG de forma implícita e transitória, conforme descrito no item 4.4. O esquema de representação adotado foi considerado uma boa escolha, facilitando a implementação das operações, propiciando flexibilidade e extensão de domínio, e permitindo a obtenção de resultados que atendem aos propósitos do sistema. Para continuidade, sugere-se o estudo do acoplamento de novos esquemas de representação, desenvolvendo-se algoritmos de conversão entre representações e aproximando o sistema da arquitetura ideal proposta por [Mill89].

f) Conforme observado no item 4.11, o modelo do observador implementado no sistema João-de-Barro apresenta algumas restrições. Dentre elas destacam-se a posição fixa no infinito e a direção de observação fixa na origem do sistema de coordenadas universais. Assim, é importante a adoção de modelos mais sofisticados que permitam, por exemplo, alterar-se a direção de observação. Uma das vantagens oferecidas por esse procedimento seria a melhor

utilização do espaço de referência abrangido pelo sistema. Outra sofisticação iteressante de ser implementada, é a possibilidade de haver rotação em torno do eixo Z do sistema de coordenadas visuais, permitindo que o observador "enxergue" cenas inclinadas.

g) Os resultados obtidos com o algoritmo desenvolvido para as operações booleanas foram considerados positivos, uma vez que o algoritmo se mostrou eficaz no cumprimento de seus propósitos. As modificações efetuadas na idéia original de [Laid86] facilitaram a implementação e, possivelmente, diminuíram o custo computacional. Até a conclusão deste trabalho, não foram encontradas restrições que invalidem as modificações propostas. Possivelmente, a característica, já comentada no capítulo 5, de se originarem modelos com mais de dois vértices colineares, se torne indesejável para algumas aplicações. Entretanto, esse detalhe não causou nenhum impecilho segundo as propostas do sistema João-de-Barro. Um ponto observado foi a falta de procedimentos para se desfazer ("undo") uma dada operação booleana realizada indevidamente. A implementação de tais procedimentos aumentaria a flexibilidade d sistema.

h) A palheta de cores é um dos pontos do sistema que merecem maiores estudos e melhorias. A palheta atualmente construída apresenta imperfeições, tais como número reduzido de cores matizes e repetição de tonalidades em diferentes posições da palheta. Além da revisão da palheta atual, recomenda-se a utilização de outros padrões de vídeo, tais como os padrões "TrueColor", capazes de exibir em torno de 16.000.000 (dezesseis milhões) de cores simultâneas.

i) A adoção do método "flat-shading" de iluminação permitiu uma melhor qualidade na apresentação de resultados, mesmo porque os modelos exibidos em arame não são, sempre, visualmente precisos. Uma melhoria que pode ser adotada de antemão é a possibilidade de se alterar a posição da fonte de luz, uma vez que a mesma foi implementada com posicionamento fixo no sistema. Para isso, basta se implementarem as funções de interface homem-máquina que solicitem e estabeleçam a nova posição da fonte de luz, uma vez que toda a parte de cálculo matemático já se encontra pronta para essa modificação. Além disso, por consistir numa ferramenta voltada para aplicações de síntese de imagens, a evolução do sistema deve abranger melhorias significativas nos recursos de visualização dos modelos. Torna-se quase que imperativa a inclusão de novos métodos e modelos de iluminação, tais como os métodos de Gouraud e Phong, funções para cálculo de sombras, criação de padrões de textura, entre outros recursos.

7. Bibliografia

7. BIBLIOGRAFIA

Conforme mencionado no capítulo 1, as referências bibliográficas foram organizadas em três porções distintas. No item 7.1, são relacionadas aquelas referências diretamente utilizadas como base para o trabalho e que foram explicitamente citadas ao longo deste texto. Já no item 7.2, são relacionados os trabalhos que não foram citados no texto mas que foram utilizados como subsídios e fontes de consulta. Por último, no item 7.3, é apresentado um levantamento bibliográfico destinado a complementar a bibliografia sobre os assuntos abordados.

7.1. Bibliografia Referenciada no Texto

[Abra92] ABRASH, M. *Color Modeling in 256-color Mode.* Dr. Dobb's Journal. nº 191, Agosto, 1992.

[Baer79] BAER, A. et al. *Geometric Modelling : a survey.* Computer-Aided Design. 11(5):253-272, Setembro, 1979.

[Berg86] BERGER, M.. Computer Graphics with Pascal. Ed. The Benjamin/Cummings Publishing Company Inc., 1986.

[Dech88] DECHANG, M. e RONGXI, T.. *Realizing the Booleans Operations in Solid Modeling Technique via Directed Loops.* Computer & Graphics. 12(3/4):319-322, 1988

[East84] EASTMAN, C. M. e PREISS, K.. *A Review of Solid Shape Modelling Based on Integrity Verification.* Computer-Aided Design. 16(2):66-80, Março, 1984.

[Faux79] FAUX, I.D. e PRATT, M.J.. Computational Geometry for Design and Manufacture. Ed. Ellis Horwood Limited, 1979.

[Ferr86] FERREIRA, A.B.H.. Novo Dicionário da Língua Portuguesa. 2ª edição, Editora Nova Fronteira, Rio de Janeiro, 1986.

[Flaq88] FLAQUER, J. e RODIL, J.L.. *Boolean Operations Based on the Planar Polyhedral Representation.* Computer & Graphics. 12(1):39-64, 1988.

[Fole82] FOLEY, J.D. e VAN DAM, A.. Fundamentals of Interactive Computer Graphics. Ed. Addison-Wesley Publishing Company, 1982.

[Gome90] GOMES, J.M. e VELHO, L.C.. *Conceitos Básicos de Computação Gráfica*. Publicado pela VII Escola de Computação, IME-USP, Julho, 1990.

[Hear86] HEARN, D. e BAKER, M.P.. *Computer Graphics*. Ed. Prentice-Hall, 1986.

[Mill89] MILLER, J.R.. *Architectural Issues in Solid Modelers*. *IEEE Computer Graphics and Applications*. pp 72-87, Setembro, 1989.

[Muft83] MUFTI, A.A.. *Elementary Computer Graphics*. Ed. Reston Publishing Company, 1983.

[Newm81] NEWMAN, W.M. e SPROULL, R.F.. *Principles of Interactive Computer Graphics*. 2ª edição, Ed. McGraw-Hill, 1981.

[Park85] PARK, C.S.. *Interactive Microcomputer Graphics*. Ed. Addison-Wesley Publishing Company, 1985.

[Picc88] PÍCCOLO, H.L.. *Editor de Figuras Tridimensionais Dotado de Operações Poliédricas Interativas*. Tese de Mestrado apresentada ao Depto. Engenharia Elétrica da Universidade de Brasília, 1988.

[Requ80] REQUICHA, A.A.G.. *Representations for Rigid Solids : Theory, Methods and Systems*. *ACM Computing Surveys*, 12(4):436-465, 1980.

[Requ82] REQUICHA, A.A.G. e VOELCKER, H.B.. *Solid Modeling : A Historical Summary and Contemporary Assessment*. *IEEE Computer Graphics and Applications*. 2(2):9-24, Março, 1982.

[Requ83] REQUICHA, A.A.G. e VOELCKER, H.B.. *Solid Modeling : Current Status and Research Directions*. *IEEE Computer Graphics and Applications*. 3(7):25-37, Outubro, 1983.

[Requ85] REQUICHA, A.A.G. e VOELCKER, H.B.. *Boolean Operations in Solid Modeling : Boundary Evaluation and Merging Algorithms*. *Proceedings of the IEEE*. 73(1):30-44, Janeiro, 1985.

[Thal87] THALMANN, N. e THALMANN, D.. *Image Syntesis : Theory and Practice*. Ed. Springer-Verlag, 1987.

[Tilo80a] TILOVE, R.B. e REQUICHA, A.A.G.. *Closure of Boolean Operations on Geometric Entities*. *Computer-Aided Design*. 12(5):219-220, Setembro, 1980.

[Tilo80b] TILOVE, R.B.. *Set Membership Classification : A Unified Approach to Geometric Intersection Problems.* IEEE Transsactions on Computers. C-29(10)874:883, Outubro, 1980.

[Xiny88] XINYOU, L., ZESHENG, T., e JIAGUANG, S.. *The Implementation of Set Operations for Regularized Geometric Objets.* Computer & Graphics. 12(3/4):309-318, 1988.

7.2. Bibliografia não Referenciada no Texto

[Athe78] ATHERTON, P. et al. *Polygon Shadow Generation,* Computer Graphics. 12(3):275-281, 1978.

[Avil82a] ÁVILA, G.. *Cálculo 1 : Funções de Uma Variável.* Ed. Livros Técnicos e Científicos, Rio de Janeiro, 1982.

[Avil82b] ÁVILA, G.. *Cálculo 3 : Funções de Várias Variáveis.* Ed. Livros Técnicos e Científicos, Rio de Janeiro, 1982.

[Barr86] BARRASS, R.. *Os Cientistas Precisam Escrever : Guia de Redação para Cientistas, Engenheiros e Estudantes.* 2ª edição, Ed. T.A. Queiroz, São Paulo, 1986.

[Bars81] BARSKY, B.A. *Computer-Aided Geometric Design : A Bibliography with Keywords and Classified Index.* IEEE Computer Graphics and Applications, 1(3):67-109, Julho, 1981.

[Borl88a] BORLAND INTERNATIONAL. *Turbo C User's Guide Version 2.0.* 1988.

[Borl88b] BORLAND INTERNATIONAL. *Turbo C Reference Guide Version 2.0.* 1988.

[Boys79] BOYSE, J.W.. *Interference Detection Among Solids and Surfaces.* Communications of the ACM. 22(1):3-9, Janeiro, 1979.

[Boys82] BOYSE, J.W. e GILCHRIST, J.E.. *GMSolid : Interactive Modeling for Design and Analysis of Solids.* IEEE Computer Graphics and Applications. 2(2):27-40, Março, 1982.

[C&CA] Computer & Control Abstracts, Science Abstracts Series C, vols. 25-26-27, números mensais de janeiro de 1990 a outubro de 1992, IEE - Institution of

Electrical Engineers em associação com IEEE - Institute of Electrical and Electronics Engeineers Inc.

[Cega89] CEGALLA, D.P.. *Novíssima Gramática da Língua Portuguesa*. 31ª edição, Companhia Editora Nacional, São Paulo, 1989.

[Chiy85] CHIYOKURA, H. e KIMURA, F.. *A Method of Representing the Solid Design Process*. *IEEE Computer Graphics and Applications*. 5(4):32-41, Abril, 1985.

[Fren89] FRENKEL, K.A.. *The Next Generation of Interactive Technologies*. *Communications of the ACM*. 32(7):872-889, Julho, 1989.

[Hall86] HALL, R.. *A Characterization of Illumination Models and Shading Techniques*. *The Visual Computer*. 2(5):268-277, 1986.

[IBM92] IBM - INTERNATIONAL TECHNICAL SUPORT CENTERS, *3D Computer Graphics Concepts - An Overview*, Abril, 1992.

[Kala82] KALAY, Y.. *Determining the Spatial Containment of a Point in General Polyhedra*. *Computer Graphics and Image Processing*. pp. 303-334, 1982.

[Laid86] LAIDLAW, D.H. et al. *Constructive Solid Geometry for Polyhedral Objects*. *Computer Graphics - SIGGRAPH' 86*. 20(4):161-170, 1986.

[Leit82] LEITHOLD, L. *O Cálculo com Geometria Analítica*. 2ª edição, vols 1 e 2, Editora Harper & Row do Brasil, São Paulo, 1982.

[Nort85] NORTON, P. *The Peter Norton Programmer's Guide* to the IBM PC. Ed. Microsoft Press, 1985.

[Schi87] SCHILDT, H. *C : The Complete Reference*, Ed. Osborne McGraw-Hill, 1987.

[Suth74] SUTHERLAND, I.E. e HODGMAN, G.W. *Reentrant Polygon Clipping*. *Communications of the ACM*. 17(1):32-42, Janeiro, 1974.

[Wyvi86] WYVILL, G. et al. *Space Subdivision for Ray Tracing in CSG*. *IEEE Computer Graphics and Applications*. 6(4):28-34, Abril, 1986.

7.3. Outras Fontes Recomendadas para Consulta

[Agar92] AGARWAL, S.C. e WAGGENSPACK W.N. Jr., *Decomposition Method for Extracting face Topologies from Wireframes Models*, Computer-Aided Design, 4(3):123-40, Março, 1992.

[Anwe89] ANWEI, L. et al, *HYBRID - A Solid and Surface Modeling System based on Data Base*, Proceedings of International Conference on Computer-Aided Design and Computer Graphics, Beijing, China, 10-12 Aug, p.222-7, Acad Publishers,1989.

[Arba90] ARBAB, F., *Set models and Boolean operations for solids and assemblies*, IEEE Computer Graphics and Applications, vol. 10, n. 6, p. 76-86, 1990.

[Arqu89] ARQUES, D. e KOCH, P., *Solid modelling by pavements*, Proceedings of the 2nd International Conference, Pixim, 89, Computer Graphics in Paris, p.47-61, 1989.

[Arqu91] ARQUES, D. E JACQUES, I., *Unorganized solids : definition, rooting and embeeding*, Inform. Theor. Appl., vol. 25, n. 3, p.219-46, França, 1991.

[Bing92] BINGFENG, Z., *A localizing algorithm for Boolean Operations of solid models represented by polyhedron*, Chin. J. Comput., vol. 15, n. 1, p. 33-40, 1992.

[Blim92] BLIMS, J.F., *A Trip Down the Graphics Pipeline : Grandpa, what does 'viewport' mean ?*, IEEE Computer Graphics and Applications, 12(1):83-7, Janeiro, 1992.

[Bowy89] BOWYER, A. et al, *Applications of computer algebra in solid modeling*, EUROCAL '87. European Conference on Computer Algebra Proceedings, Leipzig, East Germany, 2-5 Junho 1987, p. 244-5, Springer-Verlag, 1989.

[Brun89] BRUN, J.-M. Solid modeling schemes and solid reconstruction, Theory and Practice of Geometric Modeling, Tubingen, West-Germany, 3-7 outubro de 1988, p. 383-401, Springer-Verlag, 1989.

[Brun91] BRUNET, P., e VINACUA, A., *Surfaces in Solid Modeling*, Geometric Modeling - Methods and Applications, Sringer-Verlag, p. 17-34, 1991.

[Casu90] CASU, L. e FALCIDIENO, B., *Boolean operators in a geometric modeller of boundary type*, Riv. Inform., vol. 20, n. 1, p. 5-28, 1990.

[Chan89] CHAN, T.K. e GARGANTINI, I., *Conversion and integration of boundary representations with octrees*, Proceedings. Graphics Interface '89, London, Ont. Canadá, 19-23 de junho de 1989, p. 203-10, 1989.

[Chao92] CHAO, M., *Solid Modelling based on Polyhedron Approach*, Computer Graphics, 16(1):401-5, 1992.

[Corn91] CORNWAY, D., *Constructive solid geometry using the isoluminance contour model*, Computer Graphics (UK), 15(3), p.341-7, 1991.

[Cosm91] COSMAS, J.P. e HIBBERG, R., *Geometrical testing of three-dimensional objects with the aid of pattern recognition*, IEE Proc. E Comput. Digit. Tech. (UK), vol. 138, n. 4, p.250-4, Julho, 1991.

[Crok91] CROKER, G.A. e REINKE, N.F., *An editable nonmanifold boundary representation*, IEEE Computer Graphics and Applications, vol. 11, n. 2, p.39-51, Março, 1991.

[Desa92] DESAUHNIRS, H. *An Extension of Manifold Boundary Representations to the R-Sets*, ACM Trasactions on Graphics, 11(1):40-60, Janeiro, 1992.

[Dew90] DEW, P.M. et al, *Progress towards an interactive solid modelling system on parallel computers*, Transputer/Occam Japan 3. Proceedings of the 3rd Transputer/Occam International Conference, Tóquio, Japão, 17-18 Maio de 1990, p. 209-21, 1990.

[Feij91] FEIJO, B. et al, *Better criteria for the development of solid modelling software*, Reliability and Robustness of Engineering Software II. Proceedings of the Second International Conference, Milan, Italy, 22-24 april 1991, p.353-62, 1991.

[Ferr91] FERRUCCI, V. e PAOLUZZI, A., *Extrusion and bondary evaluation for multidimensional polyhedra*, Computer Aided Design (UK), vol. 23, n. 1, p. 40-50, 1991.

[Feng91] FENG, Z. et al, *A new algorithm of Boolean operation in three dimensional space*, Journal of Shangai Jiatong University, China, vol. 25, n. 1, p.56-62, 1991.

[Galy91] GALYEAN, T.A., *Sculpting : an interactive volumetric technique*, Computer Graphics (USA), 25(4):267-74, Julho, 1991.

[Ge91] GE, Q., et al, *2.5D geometry modelling system CPDGM-2.5D*, Computer Applications in Production and Enginnering. Proceedings of the Third International IFIP Conference, p.314-24, 1991.

[Gerv89] GERVAUTZ, M., *SOL - An Object Description Language for Solid Modeling*, Proceedings of International Conference on Computer-Aided Design and Computer Graphics, Beijing, China, 10-12 Aug, p.288-91, Acad Publishers,1989.

[Glas92] GLASSNER, A.S., *Geometric Substitution : a Tutorial*, IEEE Computer Graphics and Applications, 12(1):22-36, Janeiro, 1992.

[Gome91] GOMES, A.J.P. e TEIXEIRA, J.C.G., *Form feature modelling in a hybrid CSG/B-Rep scheme*, Computer Graphics (UK), vol 15, n.2, p.217-29, 1991.

[Gudu90] GUDUKBAY, U. e OZGUE, B., *Free-form solid modeling using deformations*, Computer Graphics (UK), vol. 14, n. 3-4, p. 491-500, 1990.

[Gurs91] GURSOZ, E.L. et al, *Boolean set operations on non-manifold boundary representation objects*, Computer Aided Design, (UK), vol. 23, n. 1, p. 33-9, 1991.

[Hard89] HARDER, T. et al, Information structures and database support for solid modeling, Theory and Practice of Geometric Modeling, Tubingen, West-Germany, 3-7 outubro de 1988, p. 433-47, Springer-Verlag, 1989.

[Harr90] HARRAND, V.J. et al, *Scientific data visualization : a formal introduction to the rendering and geometric aspects*, Proceedings of Supercomputing '90, p.775-83, Computer Society Press, USA, 1990.

[Hong89] HONGSHENG, L., *Microcomputer Solid Modeling using Cell Model Scheme*, Proceedings of International Conference on Computer-Aided Design and Computer Graphics, Beijing, China, 10-12 Aug, p.245-54, Acad Publishers,1989.

[Hui91] HUI, K.C. e TAN, S.T., *Display Techniques and Boundary Evaluation of a Sweep-CSG modeler*, Visual Computer, 8(1):18-34, 1991.

[Hui92] HUI, K.C. e TAN, S.T., *Construction of a Hybrid Sweep-CSG Modeler - The Sweep-CSG Representation*, Eng. Comput., 8(2):101-19, 1992.

[Ross91] ROSSIGNAC, J.R. et al, *Constructive non-regularized geometry,* Computer Aided Design (UK), vol. 23, n. 1, p. 21-32, 1991.

[Shel91] SHELLEY, T., *3D and solids demand workstation power,* Eng. Comput. (UK), vol 10, nº 4, p.20-1, Julho, 1991.

[Sinc91] SINCLAIR, D.C., *Solid Modelling in HASKELL,* Proceedings of the 1990 Glasgow Workshop, Ullapool, UK, 13-15 Agosto 1990, Springer-Verlag, Berlim, 1991, p.246-63, 1991.

[Sitt91] SITTAS, E., *3D design reference framework,* Computer Aided Design, vol. 23, n. 5, p.380-4, 1991.

[Shap91] SHAPIRO, V., *Construction and optimization of CSG representations,* Computer Aided Design (UK), vol. 23, n. 1, p. 4-20, 1991.

[Sugi90] SUGIHARA, K., e IRI, M., *A solid modelling system free from topological inconsistency,* J. Inf. Process., vol. 12, n. 4, p. 380-93, 1990.

[Tori91] TORIYA, H. et al, *Boolean operations for solids with free-form surfaces through polyhedral approximation,* Visual Computer, vol. 7, n.2-3, p.97-103, 1991.

[Zali91] ZALIK, B., *Boolean operations on the solids represented by boundary representation,* Graphics, Hangzhou, China, 23-26 de setembro de 1991, p. 269-74, Acad. Publishers, 1991.