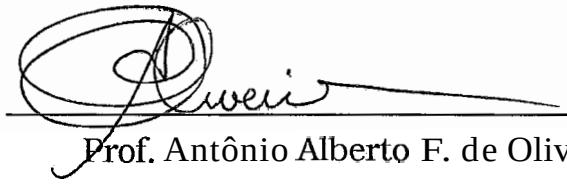
UMA ESTRATÉGIA PARA CONSTRUÇÃO DE REPRESENTAÇÕES BREP DE SÓLIDOS CSG

Vitor Pêgo Hottum

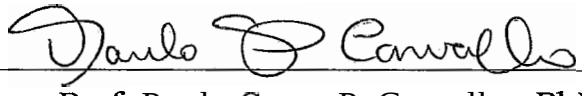
TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Ronaldo Cesar Marinho Persiano, D. Sc.
(Presidente)



Prof. Antônio Alberto F. de Oliveira, D. Sc.



Prof. Paulo Cezar P. Carvalho, PhD

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 1992

HOTTUM, VITOR PEGO

Uma Estratégia para Construção de Representações BRep de Sólidos CSG [Rio de Janeiro] 1992 7, xiii + 105 p. 29,7 cm (COPPE/ UFRJ, M. Sc., Engenharia de Sistemas e Computação, 1992)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Modelagem de Sólidos

I. COPPE/UFRJ

II. Título (série).

Resumo da Tese apresentada a COPPE/UFRJ como parte dos requisitos necessários para obtenção do grau de Mestre em Ciências (M.Sc.).

UMA ESTRATÉGIA PARA CONSTRUÇÃO DE REPRESENTAÇÕES BREP DE SÓLIDOS CSG

Vitor Pêgo Hottum

Julho de 1992

Orientador: Prof. Ronaldo Cesar Marinho Persiano

Programa: Engenharia de Sistemas e Computação

O objetivo deste trabalho é o desenvolvimento de uma estratégia que permita a conversão de representações CSG de sólidos em suas correspondentes Representações por Fronteira (BRep). O método proposto faz uso do paradigma de subdivisão espacial objetivando a redução da complexidade do problema. É também projetado um modelador BRep cujo processo de construção de representações se adequa a estratégia de conversão adotada.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M. Sc.).

A STRATEGY FOR THE CONSTRUCTION OF BREP REPRESENTATIONS OF CSG SOLIDS

Vitor Pêgo Hottum

July, 1992

Thesis Supervisor: Prof. Ronaldo Cesar Marinho Persiano

Department: Engenharia de Sistemas e Computação

The goal of this work is the development of a strategy that permits the conversion of CSG solid representations in its corresponding Boundary Representations (BRep). The proposed method makes use of the space subdivision paradigm with the objective of reducing the problem complexity. It is also pmjected a BRep modeller whose constructive process adequates with the adopted conversion strategy.

Aos meus pais pelo eterno incentivo;
a Marcia pelas grandes emoções;
a mim por superar os obstáculos.

AGRADECIMENTOS

Muitos foram aqueles que, direta ou indiretamente colaboraram das mais diversas formas para que este trabalho fosse concluído. Nas poucas palavras que se seguem procuro expressar minha gratidão a todos vocês que, com seu carinho, estímulo ou auxílio financeiro, participaram desta etapa da minha vida.

Aos meus pais e a meu irmão agradeço pelo ambiente de paz, amor e confiança raramente encontrado nos lares. Obrigado, também, por todo o esforço, muitas vezes não reconhecido, empreendido na função de me educar e preparar para o mundo.

Recorrer aos conhecimentos do orientador Ronaldo Cesar Marinho Persiano foi uma experiência ímpar. Se mostrando sempre atencioso e com idéias novas, que invariavelmente significavam mais trabalho, tomou possível o desenvolvimento e conclusão destes estudos.

Com seu jeito todo aloprado de ser, Marcelo Salim foi o principal responsável por criar um clima de descontração que, por muitas vezes, me fez esquecer os pmblemas e superar os momentos difíceis. Pela semelhança entre nossos estudos, foi de muita valia a troca de informações e o aprendizado (ou vôo) conjunto de muitos tópicos.

Imperativa foi a ajuda prestada por alguns companheiros do LCG que, com seus programas e experiência computacional, simplificaram a tarefa de implementaçáo. Agradeço a João Luiz Dihl Comba pelo programa *GeraOCsg* e por sua prestatividade em modificá–lo, objetivando sua compatibilidade as minhas necessidades. Igualmente valiosas foram nossas inúmeras conversas a respeito do processo de subdivisão. A Cláudio Esperança devo o desenvolvimento do programa *CompilaCsg* e de alguns outros, mas, sobretudo, lhe agradeço pelos ensinamentos profissionais ou não, transmitidos a mim a partir de sua capacidade e simplicidade. O programa de visualização *lcgrender*, elaborado por André Fonseca, foi de grande valia na verificação visual dos resultados obtidos.

Aos palpiteiros André Guedes e Leonardo Monteiro, meu muito obrigado pelas dicas que sempre levaram a resultados positivos e me limaram de grandes dores de cabeça.

Eliminar a tranquilidade da equipe de suporte do LCG, com perguntas idiotas e problemas insolúveis, foi uma necessidade constante no decorrer deste trabalho. Sou grato pelos serviços prestados pela equipe formada por Ricardo Dias Campos, Abel Amorim e Alberto Escarlate.

Um reconhecimento especial deve ser dado ao apoio emocional e a amizade daqueles que souberam me valorizar e suportar meus desabafos. Obrigado Sérgio, Suely e Teresa pelo ombro amigo sempre presente.

Tenho também muito a agradecer a todos os birutas do Laboratório de Estatística, a saber, Andrew, Hedibert e Otávio, que participaram comigo dos entusiasmados momentos de diversão durantes as pesquisas de opinião lá processadas. Ao Andrew especialmente, por todas as aventuras vividas ao longo deste tempo.

Em parte pelos momentos felizes, que me deram forças para continuar lutando e, em parte pelos momentos de crise, que me fizeram ver que eu estava rodeado de amigos, agradeço a ti, minha pequena Marcia.

Agradeço a CAPES e a Sociedade Cultural e Beneficente Guilherme Guinle pelo auxílio financeiro durante a elaboração desta tese.

Meus sinceros agradecimentos a LIGHT, pelas faltas constantes de luz, ao serviço de dedetização, as greves e paralizações que, embora sempre fora de hora, me cederam alguns momentos de descanso ao provocarem o fechamento do LCG.

Obrigado Jericoacoara, Morro Branco, Arraial D'Ajuda, Trancoso, Arraial do Cabo, Salvador, Aracajú, Genipabú, Recife, Fortaleza, João Pessoa, Gramado, Canoa Quebrada, Cumuruxatiba, Porto de Galinhas, Itapuã, Suape, Tororão, Morro de São Paulo, Angra dos Reis, Praia Seca, Itatiaia, Visconde de Mauá, Ibicuí, Teresópolis, Petrópolis, Açu, Pedra da Gávea, Pico das Agulhas Negras e Búzios, por simplesmente existirem.

INDICE

FIGURAS E TABELAS

CAPÍTULO 1 _ Introdução

CAPÍTULO 2 _ Introdução à Modelagem de Sólidos

CAPÍTULO 3 – Subdivisão Espacial Aplicada sobre Árvores CSG

CAPÍTULO 4 – O Modelo de Representação de Superfícies

CAPÍTULO 5 – A Proposta de Conversão CSG → BRep

CAPÍTULO 6 – Características da Implementação Desenvolvida

CAPÍTULO 7 – Considerações Finais

REFERÊNCIAS BIBLIOGRÁFICAS

APÊNDICES

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo I

Introdução

1.1. INTRODUÇÃO AO TRABALHO APRESENTADO

Este trabalho se situa dentro da área da Computação Gráfica denominada *Modelagem de Sólidos*, na qual discutem-se metodologias sobre a construção e manipulação de estruturas de dados que permitam representar sólidos em computador. Uma definição formal de tal ciência pode ser encontrada em [REQU83]: *"O termo Modelagem de Sólidos engloba um corpo de teorias, técnicas e sistemas focalizados em representações informacionalmente completas de sólidos. Representações que permitam, ao menos em princípio, o cálculo de qualquer propriedade geométrica bem definida de qualquer sólido representáuel."*.

A visualização e produção de peças mecânicas foram fatores decisivos no desenvolvimento da Modelagem de Sólidos, que teve como precursora os Sistemas de Projeto Assistido por Computador ou CAD (*Computer Aided Design*). Estes sistemas se utilizavam de elementos de uma ou duas dimensões do E^3 (linhas e polígonos) na tentativa de modelar um objeto tridimensional. Tal procedimento não garantia uma forma unificada de representação, impedindo em geral a computação de propriedades inerentes ao sólido, como o volume, área, peso e centro de massa. Os estudos acerca da modelagem de sólidos se acentuaram a medida que esta passou a ser utilizada em diversas outras atividades. Entre elas estão a arquitetura, a robótica, a cartografia e a medicina. Estes estudos resultaram no surgimento de diversas técnicas e algoritmos que acabaram por constituir uma especialização dentro da computação gráfica.

No estado atual da Modelagem de Sólidos dispomos de várias formas de representar computacionalmente um sólido. Cada qual possui vantagens e desvantagens provenientes do paradigma sobre o qual se encontram baseadas. Dentre os modelos existentes temos a Geometria Sólida Construtiva (*Constructive Solid*

Geometry - CSG), as árvores octais ou *Octrees* e os modelos de representação por fronteiras (*boundary representation - BRep)*.

Um sistema de modelagem ideal deve, entre outras características, admitir mais de uma forma de representação de sólidos, permitindo que as operações desejadas sejam realizadas no modelo que melhor convir, seja para obter resultados mais eficientes ou velozes. O termo *híbrido* é associado a um sistema com esta particularidade.

Para que se tenha um sistema híbrido é necessário dispor de artifícios que transportem consistentemente objetos entre as diversas possibilidades de representação oferecidas. O trabalho aqui apresentado visa o desenvolvimento de uma estratégia de conversão que leva uma representação CSG de sólido em sua correspondente representação por fronteiras (BRep). Nossa proposta, a ser exposta nos capítulos vindouros, faz uso da subdivisão espacial para que a tarefa de computar a fronteira de um sólido tenha sua complexidade reduzida.

Os capítulos deste texto estão dispostos de modo que o capítulo 2 exibe um painel do estado atual da Modelagem de Sólidos. Nele vamos apresentar os modelos de representação de sólidos comumente utilizados e suas propriedades básicas. Serão também abordadas superficialmente duas estratégias de conversão de representações CSG para representações BRep, sendo uma delas a por nós adotada.

A finalidade do capítulo 3 será a de expor como o paradigma de subdivisão espacial pode ser aplicado sobre a representação CSG de um dado sólido, simplificando a árdua tarefa de computar sua fronteira.

Para que o processo de conversão pudesse ser posto em prática foi preciso desenvolver um modelador BRep capaz de lidar com as representações da fronteira do sólido. Este modelador será descrito com detalhes ao longo do capítulo 4 deste texto.

A estratégia de conversão proposta se encontra desenvolvida passo a passo no decorrer do capítulo 5.

O capítulo 6 fornecerá ao leitor uma noção a respeito dos programas e seus respectivos módulos que constituem a implementação por nós elaborada para o procedimento estudado.

Finalizando, o capítulo 7 sintetiza toda a informação transmitida, apontando alguns problemas e sugerindo modificações. Uma análise dos tempos de processamento e dos tempos proporcionais gastos por cada etapa da conversão é ali exposto para alguns sólidos de testes.

Capítulo 2

Introdução a Modelagem de Sólidos

2.1. INTRODUÇÃO

Ao nos depararmos com uma esfera, um cubo, etc, somos capazes de identificar cada um destes objetos. Isto é possível por termos em nossa memória todos os dados necessários para que, a partir da simples visualização de um determinado objeto, o mesmo possa ser reconhecido. O ponto central em torno da *Modelagem de Sólidos* vem a ser a determinação de um conjunto mínimo de informações necessárias a identificação e construção de sólidos. O processo natural de descrição de um sólido, passa principalmente pela definição de sua forma, ou seja, sua geometria. Pode-se então definir um *modelo de representação de sólidos* como sendo um conjunto de informações que descreva implícita ou explicitamente a geometria de objetos sólidos. Este conjunto de informações pode ser selecionado de modos distintos, gerando assim, vários modelos de representação de sólidos.

Cada um dos modelos atualmente existentes tem suas vantagens e desvantagens. A escolha adequada de um deles depende, fundamentalmente, das aplicações às quais o mesmo se destina. Os modelos de representação comumente utilizados podem ser divididos em três categorias:

- **Modelos Construtivos**

Visam representar sólidos por um conjunto de pontos, definido a partir de operações sobre conjuntos de pontos mais simples denominados *primitivas*. Os modelos portadores desta característica são conhecidos pelo nome de *Geometria Sólida Construtiva* (*Constructive Solid Geometry* – CSG).

- **Esquemas de Subdivisão do Espaço**

Têm como particularidade o retalhamento do espaço em um conjunto de unidades discretas de volume denominadas células. Assim sendo, um objeto é descrito por uma

enumeração destes elementos. Esta é, portanto, uma representação aproximativa, cuja precisão é proporcional ao refinamento da subdivisão espacial. Uma consequência negativa desta forma de representação vem a ser o gasto volumoso de memória. A *Enumeração da Ocupação Espacial*, onde o sólido é representado por uma lista de células localizadas em uma malha regular do espaço e as *Árvores Octais* (Octrees), nas quais as células podem diferir em tamanho, são exemplos clássicos de modelos baseados neste tipo de representação.

• Modelos de Representação por Fronteiras

Conhecidos como *BRep* (*Boundary Representation*), os modelos de representação por fronteiras baseiam-se em uma estrutura hierárquica que visa descrever o sólido a partir de sua fronteira. No topo desta estrutura encontram-se as *faces* que compõem a *casca* externa do objeto. Estas faces são limitadas por curvas unidimensionais; as *arestas*, cujas extremidades são identificadas por dois pontos; os *vértices*.

2.2. UMA DEFINIÇÃO DE SÓLIDO

Ao se construir um modelador, o objetivo é representar de alguma forma objetos do mundo real. Entretanto, o conjunto dos objetos representáveis forma apenas um subconjunto dos objetos fisicamente realizáveis. Portanto, deve-se sempre definir o subconjunto com o qual se deseja trabalhar. Em nosso caso, iremos nos restringir inicialmente ao subconjunto de pontos do espaço euclideano de dimensão 3 (E^3) que sejam *limitados* e *fechados*. Estes conceitos foram extraídos do ramo da matemática conhecido como *Topologia de Conjuntos de Pontos*. Um entendimento teórico a respeito destes termos é encontrado em [LIMA87]. Em linguagem informal, um conjunto de pontos limitado é aquele para o qual existe uma esfera com raio suficientemente grande para englobar todos os seus pontos. Conjunto fechado é aquele cujos pontos de fronteira pertencem a si próprio. O conjunto de pontos definido pela equação $X \geq 1$ não é limitado e o conjunto formado pelos pontos que satisfazem a equação $X^2+Y^2+Z^2<1$ não é fechado.

Pelo critério adotado até então, um conjunto composto por faces, linhas e pontos isolados, seria considerado um sólido. Vamos então adicionar a esta definição de sólido a noção de *conjunto regular*. Um conjunto C é dito regular se $C=r(C)$, onde $r(C)=fecho(interior(C))$. Com isto, são excluídos os elementos isolados de dimensão inferior a três (pontos, linhas e faces).

A última restrição a ser considerada diz respeito ao tipo de superfícies através das quais a fronteira do objeto é expressa. Para que sua representação seja viável, o sólido deve, necessariamente, ser descrito por um número finito de superfícies relativamente simples, em geral superfícies algébricas. Requicha [REQU80] desenvolve esta idéia assumindo que as superfícies que compoem a fronteira do sólido devem ser *analíticas por partes*.

Agregando as informações acima, obtem-se a definição a seguir; que é conhecida na literatura pelo nome de *r-set* [REQU80]:

Sólido é um subconjunto fechado e regular do E^3, limitado por superficies semi-analíticas

2.3. FORMULAÇÃO MATEMÁTICA DOS ESQUEMAS DE REPRESENTAÇÃO

Ao selecionarmos um subconjunto do universo de objetos que atendam a determinadas características, estamos definindo o *espaço de modelagem (M)* do esquema de representação. Em nosso caso, o espaço de modelagem equivale ao conjunto de todos os sólidos.

Uma *representação sintaticamente correta* é aquela concebida por regras sintáticas (gramática) aplicadas sobre símbolos de um certo alfabeto. O uso de termos como gramática ou alfabeto no contexto acima, pode ser esclarecido na referência [HOPC69].

Uma vez definida a gramática e o alfabeto de um modelador, o mesmo pode vir a gerar, a partir de suas regras sintáticas, representações que podem ou não estar relacionadas a algum objeto do espaço de modelagem. Ao conjunto de todas estas representaçóes denominamos *espaço das representações sintaticamente corretas (R)*.

Os elementos do espaço de modelagem, para os quais existe uma representação sintaticamente correta, originam um subconjunto do espaço de modelagem denominado *domínio do esquema de representação (D)*.

O subconjunto de representações sintaticamente corretas, que identificam algum elemento de D, é dito *imagem de D* sobre *e (I)*. Um esquema de representação é então, uma função $e: D \rightarrow I$. A figura 2.1 sintetiza as informações acima.

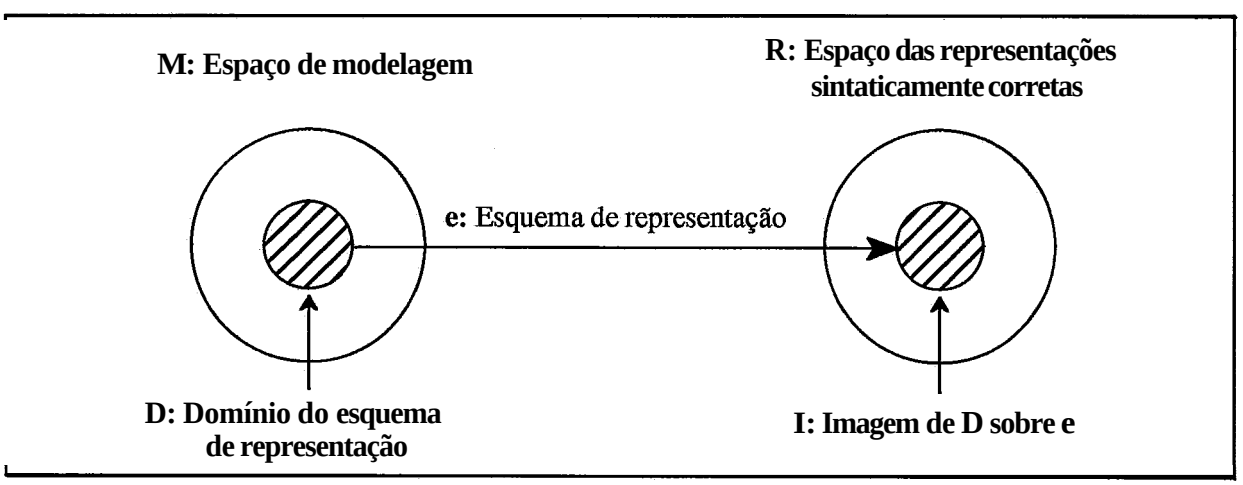


Figura 2.1 – Formulação matemática dos esquemas de representação

2.4. PROPRIEDADES DOS ESQUEMAS DE REPRESENTAÇÃO

Na seleção do esquema de representação que melhor se adeque as necessidades da aplicação a qual o mesmo se destina, é de grande importância a análise das

propriedades deste modelo. Requicha [REQU80] expõe uma série de propriedades dos esquemas de representação, das quais destacamos:

• Poder de Expressão

Em geral, os modeladores não são capazes de criar representações sintaticamente corretas para qualquer sólido existente. A abrangência do subconjunto dos sólidos passíveis de representação nos é dada pelo poder de expressão do esquema de representação (conjunto D da figura **2.1).**

• Validade

Um modelador é dito válido se todas as representações sintaticamente corretas geradas por ele correspondem a algum sólido de seu domínio. Hoje em dia, grande parte dos sistemas embutem restrições em seus procedimentos no intuito de garantir a validade das representações obtidas. Nos referindo uma vez mais a figura **2.1;** em um esquema válido, tem-se $I=R$.

• Unicidade

Classifica-se um esquema como sendo único se, para cada objeto do conjunto D da figura **2.1** houver apenas uma representação sintaticamente correta. Em outras palavras, se existe uma única sequência de regras sintáticas pertinentes ao modelador capaz de gerar a representação de um objeto do seu domínio.

• Ambiguidade

A ambiguidade se mostra presente em um modelador se existir alguma representação sintaticamente correta relacionada a mais de um sólido de seu domínio. Na figura **2.1** equivale a dizer que existem S_1, $S_2 \in D$, tal que $e(S_1) = e(S_2) \in I$.

• Concisão

Quanto menor for a quantidade de informação necessária a descrição de uma representação, mais conciso será o modelador. Em termos computacionais, a quantidade de memória ou espaço de armazenamento gasto são os parâmetros a se observar ao se classificar um modelo como sendo ou não conciso.

2.5. GEOMETRIA SÓLIDA CONSTRUTIVA (CSG)

Uma forma natural de descrever um sólido, é considerar inicialmente o conjunto vazio e, passo a passo, acrescentar ou remover sólidos mais simples. Pmcedendo-se desta forma, ao fim de uma série de alterações, o conjunto vazio inicial transforma-se na representação do sólido desejado. Uma arruela, por exemplo, pode ser obtida pela subtração de dois cilindros chatos com eixos centrais coincidentes, sendo o raio do segundo menor que o do primeiro. Os esquemas de representação CSG seguem este

princípio, permitindo, em geral, a aplicação de duas classes de operações sobre os sólidos, a saber, a *instanciação* de primitivas, que permite a manipulação de sólidos simples de modo que os mesmos gerem toda uma família de objetos e as operações de *composição* ou *booleanas*, responsáveis pelas operações entre os conjuntos de pontos de 2 sólidos distintos.

2.5.1. Primitivas CSG

Os sólidos simples, abordados anteriormente, são conhecidos pelo nome de *primitivas* e correspondem a conjuntos de pontos suficientemente simples para serem representados por um número pequeno de desigualdades algébricas. Nomalmente, as primitivas CSG são expressas por um conjunto de semiespaços, estando entre as mais utilizadas, a esfera, o bloco, o cone, o cilindro e o toro.

É comum definir apenas primitivas que correspondam a conjuntos limitados de pontos, sendo o cone, por exemplo, composto pela interseção de dois semiespacos; o primeiro, um semiespaco quádrico, conico e ilimitado e o segundo, um semiespaco plano correspondendo a sua base. Esta preocupação objetiva a idealização de um modelo válido. Pois, a partir de primitivas ilimitadas, é possível obter representações sintaticamente corretas que não satisfaçam as características de sólido. No modelador CSG disponível a este trabalho [ESPE90], dispomos das seguintes primitivas: Bloco, Esfera, Cone, Toro, Plano e Polinomial.

A primitiva piano e ocasionalmente fornecida pelos modeladores devido a versatilidade provida por ela a geração de poliedms. Cabendo ao usuário do sistema, o cuidado para que o uso inadequado da mesma não venha a gerar representações inválidas (que não correspondam a sólidos). A mesma atenção deve ser dada quando do uso da primitiva Polinomial.

2.5.2. Instanciação de Primitivas CSG

As instanciações de primitivas são amplamente utilizadas para que se possa dispor de toda uma família de objetos a partir de um único sólido primitivo. Uma instanciação e composta por uma primitiva e uma série de transfomações a ela aplicadas. Um paralelepípedo, por exemplo, pode ser descrito pela aplicação de uma operação de escala sobre as dimensões de um bloco primitivo; podemos ainda, gerar a família dos parabolóides a partir da aplicação de transformações projetivas sobre uma esfera padrão.

Os operadores responsáveis por movimentos rígidos, os quais lidam com o posicionamento espacial dos objetos, permitindo a locomoção destes para uma determinada região do espaço, são igualmente utilizados ao ser definida uma instanciação do sólido.

De posse destes conceitos, ao nos referirmos a primitivas CSG de agora em diante estaremos na verdade considerando as possíveis instanciações da mesma. O modelador no qual nos basearemos no decorrer deste trabalho se utiliza dos três operadores de instanciação descritos a seguir:

Escala(a,b,c): escala o objeto a unidades no eixo X, b em Y e c em Z.

Translação(a,b,c): move o objeto a unidades no eixo X, b em Y e c em Z.

Rotação(a,b,c): rotaciona o objeto a graus em tomo de X, b em Y e c em Z.

2.5.3. Combinação de Sólidos CSG

Tendo definido o que são primitivas CSG, resta agora desenvolver um artifício para que estas primitivas possam se combinar, dando origem a um sólido complexo. As chamadas *operações booleanas* vêm preencher esta lacuna e equivalem as operações entre conjunto de pontos. Dentre elas, as mais comuns são as operações de *união* (∪), interseção (∩) e diferença (\). Sendo S_1 e S_2 dois sólidos CSG, as operações acima são definidas da seguinte forma:

$$S_1 \cup S_2 = \{ p \in E^3 / p \in S_1 \text{ ou } p \in S_2 \}$$

$$S_1 \cap S_2 = \{ p \in E^3 / p \in S_1 \text{ e } p \in S_2 \}$$

$$S_1 \setminus S_2 = \{ p \in E^3 / p \in S_1 \text{ e } p \notin S_2 \}$$

O resultado de uma operação booleana entre dois sólidos não é necessariamente um sólido. Por exemplo, o operador de interseção, ao ser aplicado sobre duas esferas tangentes em um único ponto, resulta em um conjunto composto apenas pelo ponto de interseção. Este conjunto, obviamente, não possui as características enumeradas em 2.2 do que vem a ser um sólido. É preciso, então, acrescentar algo mais a estes operadores. Do mesmo modo que em 2.2 foi introduzida a noção de conjuntos regulares, aqui será incorporada a idéia de *operadores* booleanos regularizados, no intuito de preservar a natureza. sólida do resultado das operações. As operações booleanas regularizadas são dadas por:

$$S_1 \cup^* S_2 = \mathit{fecho}(\mathit{interior}(S_1 \cup S_2))$$

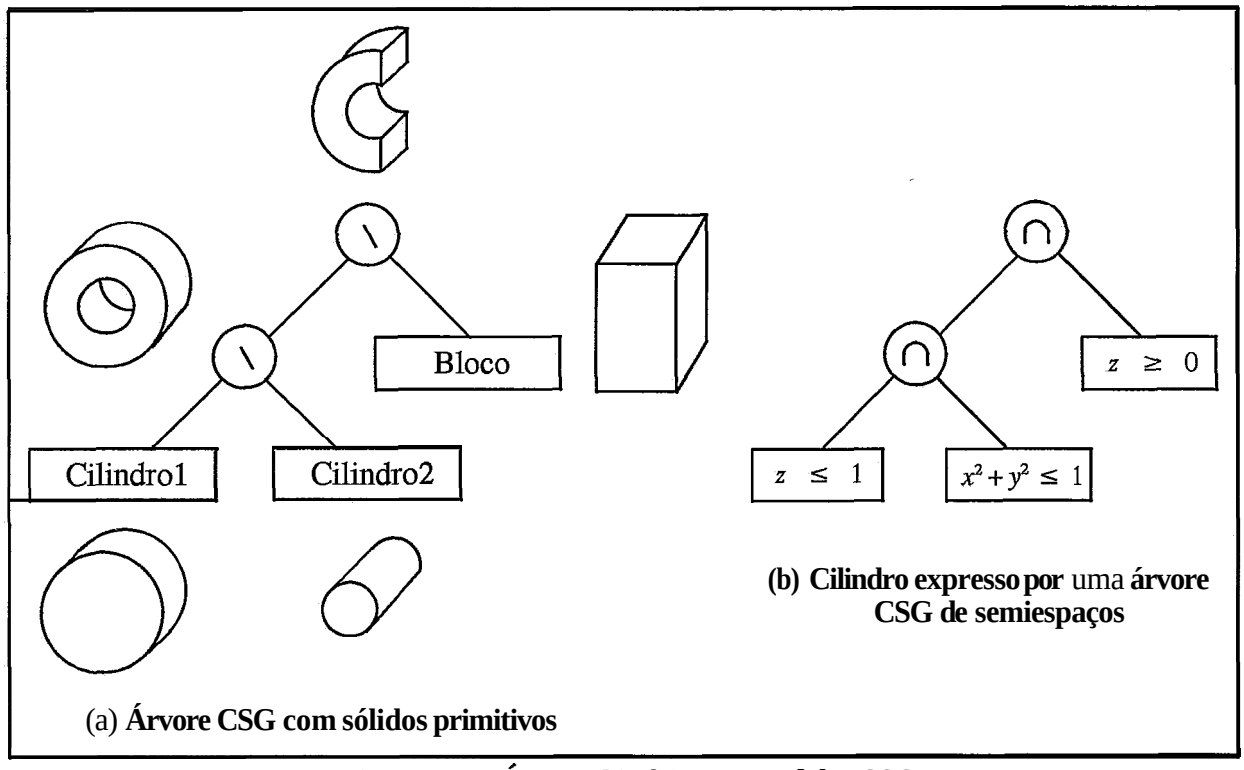
$$S_1 \cap^* S_2 = \mathit{fecho}(\mathit{interior}(S_1 \cap S_2))$$

$$S_1 \setminus^* S_2 = \mathit{fecho}(\mathit{interior}(S_1 \setminus S_2))$$

Um estudo a respeito da necessidade do uso de operadores regularizados em modelos CSG e encontrado em [TILO80].

2.5.4. Estrutura de Dados dos Esquemas de Representação CSG

Busca-se aqui, a elaboração de uma estrutura de dados capaz de armazenar expressões advindas de operações booleanas regularizadas. o fato destes operadores estarem relacionados, impreterivelmente, a dois operandos, sugere o uso de árvores *binárias* [SZWA84], em cujos nós terminais alojam-se primitivas CSG devidamente instanciadas e, em seus nós intermediários encontram-se os operadores booleanos regularizados. A figura 2.2 ilustra um caso de árvore CSG cujas primitivas são sólidos e outro tendo semiespaços como primitivas.



(a) **Árvore CSG com sólidos primitivos**

(b) Cilindro expresso por uma **árvore CSG de semiespaços**

Figura 2.2 – Árvores binárias em modelos CSG

É comum, durante o processo de modelagem, possibilitar a aplicação de operadores como os de movimento rígido sobre sólidos não primitivos (sub–árvores CSG). Nestes casos, a operação é propagada em efeito cascata, por entre os nós internos da sub–árvore CSG, até finalmente atingir um nó terminal, onde ela será efetivamente aplicada sobre a primitiva instanciada lá alojada. Além da instanciação, outras informações à respeito da primitiva, tais como cor, material, textura, podem ser adicionadas aos nós terminais.

Uma variação desta representação, sugerida em [MANT88], permite que uma dada sub–árvore CSG seja referenciada por vários nós da estrutura de dados, cada um deles contendo um conjunto de transformações a ser aplicado sobre a sub–árvore mencionada. Deste modo, várias cópias transformadas de um objeto podem ser obtidas sem que o mesmo seja repetido na representação do sólido final. Para o armazenamento de tal modelo, as árvores binárias são substituídas por *grafos acíclicos direcionados* [SZWA84].

Deste ponto em diante, nos referiremos as árvores binárias contendo uma representação CSG como *árvores* CSG.

2.5.5. Propriedades dos Esquemas de Representação CSG

O poder de expressão de modelos CSG é dado por sua biblioteca de primitivas em conjunto com os operadores disponíveis, estando o domínio limitado ao conjunto dos *r–sets*.

Modelos CSG baseados em operações booleanas regularizadas e primitivas sólidas são *válidos*. Cabe comentar, que em esquemas onde hajam primitivas não sólidas (por

exemplo semiespaços), a validade da representação obtida passa a ser responsabilidade do usuário do sistema.

As várias representações sintaticamente corretas para o sólido vazio exemplificam claramente o fato da *unicidade* não estar presente nos modelos construtivos.

Uma representação sintaticamente correta gerada por um modelo CSG identifica um único sólido. Sendo assim, os esquemas de representação CSG são *não-ambíguos*.

Por se basearem em uma pequena gama de informações, os esquemas construtivos necessitam de bem pouco espaço para o armazenamento de suas representações. Por conseguinte, são considerados modelos *concisos*.

2.5.6. Conclusões e Notas Bibliográficas

O manuseio de sólidos primitivos, objetivando a geração de objetos complexos, é um processo natural e de fácil compreensão. Aliando-se este fato ao pouco gasto de espaço físico de armazenamento, conclui-se serem os esquemas construtivos poderosas ferramentas para a representação de sólidos.

A estrutura de dados em forma de árvore binária sugere o uso de algoritmos baseados no paradigma de dividir para conquistar. Tais algoritmos, seguem o seguinte procedimento: o problema é subdividido em vários outros de soluções menos complexas (no caso de modelos CSG estas partes correspondem a sub-árvores esquerda e direita). Cada um deles é então resolvido e suas soluções agregadas de forma a se obter a solução final.

A abordagem da modelagem CSG sob o ponto de vista de rigorosa teoria matemática se deu na década de 70 e teve como seus idealizadores os pesquisadores Requicha e Voelcker, do projeto PADL da Universidade de Rochester [REQU77A, REQU77B, VOEL77]. Deste projeto originaram-se dois modeladores CSG: o PADL-1 e o PADL-2, que se tornaram muito conhecidos, tanto em meios industriais como acadêmicos. Em um trabalho pioneiro, Rossignac e Requicha [ROSS91] formulam a teoria para uma extensão dos modelos CSG, propondo o que denominaram de *Constructive Non-Regularized Geometry* (CNRG), onde objetivam a representação CSG de sólidos constituídos por conjuntos de pontos não regularizados.

2.6. ÁRVORES OCTAIS (OCTREES)

A *octree* é um caso particular de subdivisão espacial. Consiste na subdivisão recursiva de um volume cúbico do espaço em oito porções cúbicas isovolumétricas denominadas octantes. Dá-se o nome de *espaço octree* a região delimitada do espaço, sobre a qual inicia-se o processo de subdivisão. O comprimento do *lado* do espaço octree é conhecido como *largura da octree*.

Os métodos fundamentados no paradigma da subdivisão espacial, como a octree, são de natureza aproximativa, isto é, apenas em casos específicos reproduzem

perfeitamente o sólido idealizado. A cada novo refinamento da subdivisão a exatidão da representação obtida é melhorada. Entretanto, quanto maior for este refinamento, mais espaço será necessário para o seu armazenamento. De acordo com a disponibilidade de espaço de armazenamento ou a exatidão desejada da representação, define-se o nível máximo de subdivisão com o qual se pretende trabalhar. Este nível é conhecido por *profundidade* da *octree*.

O processo de subdivisão descrito acima é interrompido ao se atingir um dos critérios de parada a seguir: o octante e vazio (nenhuma porção do sólido intercepta a região delimitada pelo octante), o octante é cheio (o sólido engloba completamente a região delimitada pelo octante), o nível máximo de subdivisão estipulado foi alcançado (se neste nível ainda não for possível classificar o octante como cheio ou vazio, o seu *status* é então escolhido arbitrariamente). Com os critérios acima, os octantes são classificados em vazios, cheios ou mistos. Dependendo da aplicação, outras regras de subdivisão podem ser definidas, dando origem a novos tipos de octantes. Em [BRUN85], por exemplo, propõe-se a adição de três categorias, a saber: octante face (uma única face do objeto intercepta o octante), octante aresta (uma única aresta do objeto intercepta o octante) e octante vértice (um único vértice do sólido intercepta o octante).

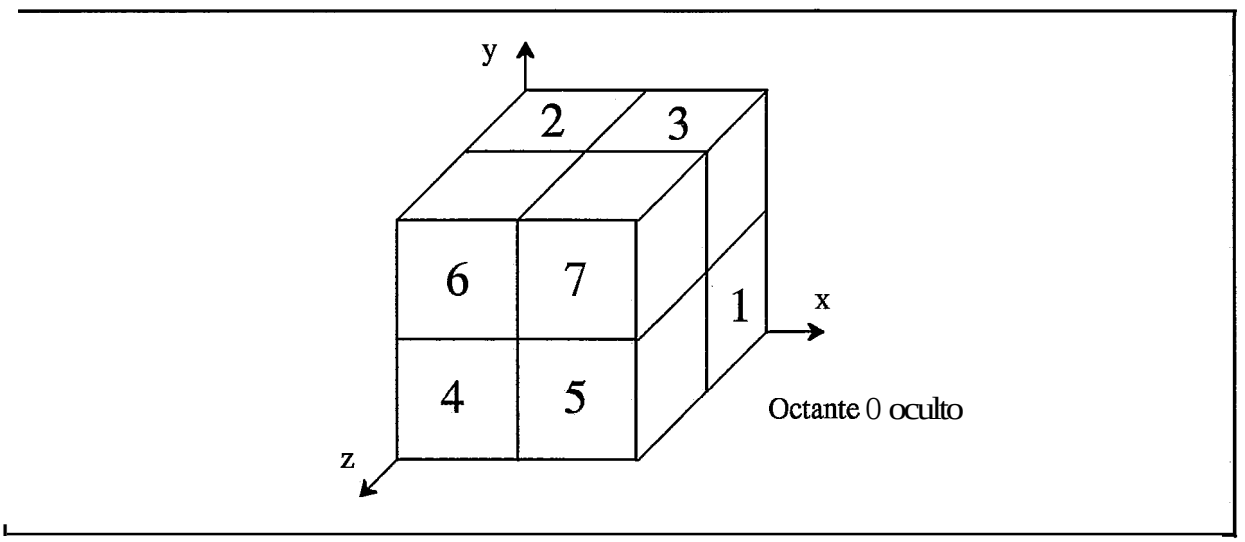


Figura 2.3 – Ordenação dos octantes proposta por Yamaguchi

Quanto a enumeração dos octantes, são diversas as opções encontradas na literatura [BRUN85, JACK80, MANT88]. A proposta adotada aqui foi desenvolvida por Yamaguchi [YAMA84]. Nela, os octantes filhos recebem uma numeração baseada em seu posicionamento no interior do octante pai. Para tal, considera-se o número binário de três dígitos $d_z d_y d_x$. Caso haja um deslocamento no eixo coordenado i para se atingir o octante, $d_i=1$, caso contrário, $d_i=0$. O octante 5, por exemplo, resulta no binário 101, o que significa dizer que é necessário um deslocamento em X e outro em Z, para que o mesmo seja localizado. Esta ordenação se encontra exposta na figura 2.3.

2.6.1. Operações entre Sólidos Octree

Operações booleanas (união, interseção, diferença) entre sólidos definidos em um mesmo espaço octree são de simples execução. Entretanto, transformam-se em tarefas complexas caso os sólidos operados estejam definidos em espaços octree distintos.

As operações de instanciação (escala, translação, rotação) são processadas morosa e exaustivamente. Algoritmos que solucionam casos específicos, tais como rotação de 90 graus ou translações onde a unidade de movimento é a própria largura do octante mínimo [JACK80], são frequentemente desenvolvidos objetivando o contorno desta dificuldade.

2.6.2. Estrutura de Dados dos Modelos Octree

A estutura de dados usual para a representação de uma octree vem a ser as árvores octais ou *árvores* octree [SAME90], onde cada nó intermediário contém 8 filhos. A cada nó da árvore é associado um octante da subdivisão. Em seus nós terminais, além da informação sobre o *status* do octante (vazio, cheio ou misto), podem ser inseridas informações inerentes ao sólido ali contido, como por exemplo, sua cor, material, peso, etc. A figura 2.4 ilustra o uso de árvores octais na representação octree de um sólido.

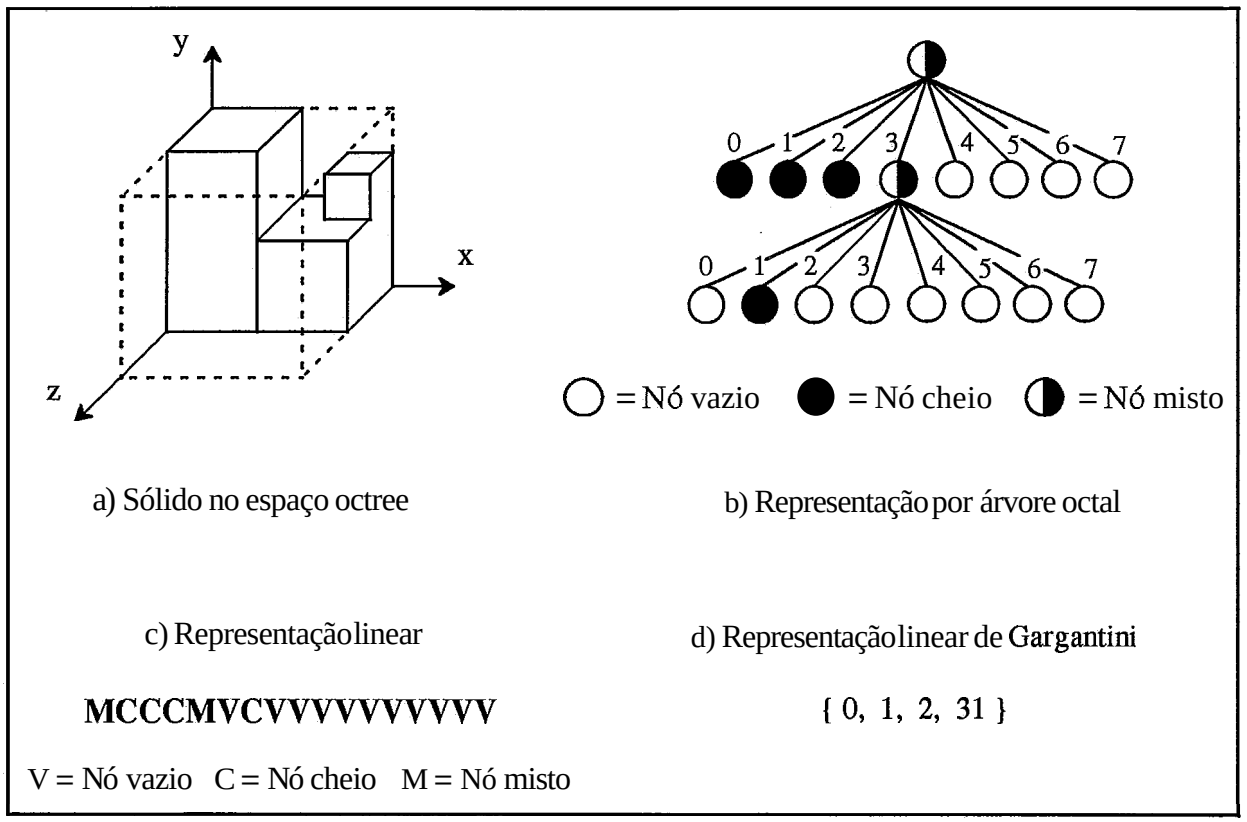


Figura 2.4 - Formus de representação e armazenamento de octrees

Para efeito de armazenamento, alguns algoritmos foram criados visando a representação linear de uma árvore octree. Kawaguchi [KAWA80] utiliza uma *string* (conjunto de caracteres), onde cada elemento representa um nó da octree. Para gerar tal *string*, a árvore octree é percorrida em pré-ordem, isto é, dado um nó misto, todos os seus filhos são percorridos antes que seu vizinho seja visitado. Gargantini [GARG82] aperfeiçoa esta idéia, propondo um método onde apenas os nós cheios são armazenados. Novamente a octree é percorrida em pré-ordem e os caminhos que levam aos nós cheios vão sendo inseridos em uma lista que, ao fim do processo, constituirá uma representação linear da árvore octree. O caminho de um nó, referido anteriormente, nada mais é, senão a sequência numérica dos nós percorridos desde a

raiz da árvore (nó inicial) até si próprio. Os esquemas de linearização citados encontram-se exemplificados na figura 2.4.

2.6.3. Propriedades dos Esquemas de Representação Octree

Com relação ao poder de expressão, o conjunto de sólidos perfeitamente modeláveis por octrees (domínio do modelo octree) é bem restrito, sendo constituído pelos poliedros cujas faces sejam todas paralelas as do espaço octree. Entretanto, uma representação aproximada de qualquer sólido regular pode ser obtida.

Toda representação octree sintaticamente correta corresponde a algum sólido de seu domínio, sendo portanto, um modelo válido.

Dado um sólido do domínio do modelo octree, existe uma única representação octree *compacta* para o mesmo. Uma octree compacta é aquela na qual um nó misto corresponde necessariamente a uma região não homogênea do espaço, ou seja, um volume que não poderia ser representado por um único nó cheio ou vazio.

Para cada representação octree sintaticamente correta, encontra-se associado um único sólido do domínio do modelo octree, o que torna a representação octree um modelo não-ambíguo.

Meagher [MEAG82] comprova em seu trabalho que, normalmente, o número de nós de uma representação octree é proporcional a área do objeto. Consequentemente, os modelos octree necessitam de grandes áreas de armazenamento. Segundo Mantyla [MANT88], a representação octree de uma peça de engenharia, atinge facilmente a cifra de 1 milhão de bytes de memória.

2.6.4. Conclusões e Notas Bibliográficas

Aparentemente, as octrees foram idealizadas independentemente por vários pesquisadores. Dentre eles citamos Hunter [HUNT78], Jackings e Tanimoto [JACK80] e Meagher [MEAG80]. As octrees se comportam como representações capazes de exprimir aproximadamente sólidos com quaisquer tipos de peculiaridades. A partir destas representações, cálculos de propriedades geométricas ou físicas, tais como área, volume, peso, centro de massa, são facilmente efetuados. Contudo, caso os objetos sendo representados extrapolem o domínio dos modelos octrees, os resultados obtidos serão apenas aproximações.

Devido a simplicidade com que são tratadas as operações booleanas em estruturas octree, vê-se em tal modelo uma boa solução para os problemas de colisão entre objetos. Em contrapartida, a quantidade de espaço necessário para o armazenamento e a dificuldade com que são processados os operadores de instanciação, representam até então, obstáculos intransponíveis ao uso de modelos octrees em aplicações onde a obtenção de resultados precisos seja imprescindível.

2.7. MODELOS DE REPRESENTAÇÃO POR FRONTEIRAS (BREP)

Os modelos construtivos e de decomposição do espaço vistos até então, interpretavam sólido como sendo um conjunto de pontos e buscavam uma

representação para o mesmo a partir de sua discretização ou da composição de conjuntos de pontos mais simples. Os modelos de representação por fronteira (*Boundary Representation Models*) ou BReps agem de modo distinto, compondo uma representação indireta do sólido através da descrição de sua fronteira.

Historicamente, os modelos de representação por fronteiras são os sucessores dos modelos poliedrais usados em computação gráfica no processo de eliminação de superfícies e linhas ocultas. A evolução nasceu da necessidade de encontrar soluções inerentes a estes modelos, tal como a representação do objeto como entidade única ao invés de uma coleção de polígonos.

2.7.1. Entidades Básicas dos Modelos BRep

A representação BRep de um objeto se dá de forma hierárquica a partir da subdivisão de sua fronteira (conjunto de superfícies de dimensão 2 do E^3) em faces, de modo que, cada face tenha uma representação matemática compacta. Por exemplo, faces situadas sobre superfícies planas, quádricas ou tomidais do E^3. No segundo nível desta hierarquia temos as arestas, isto e, curvas unidimensionais do E^3 que limitam as faces. Por último, surgem os vértices, elementos de dimensão zero situados nos extremos destas arestas.

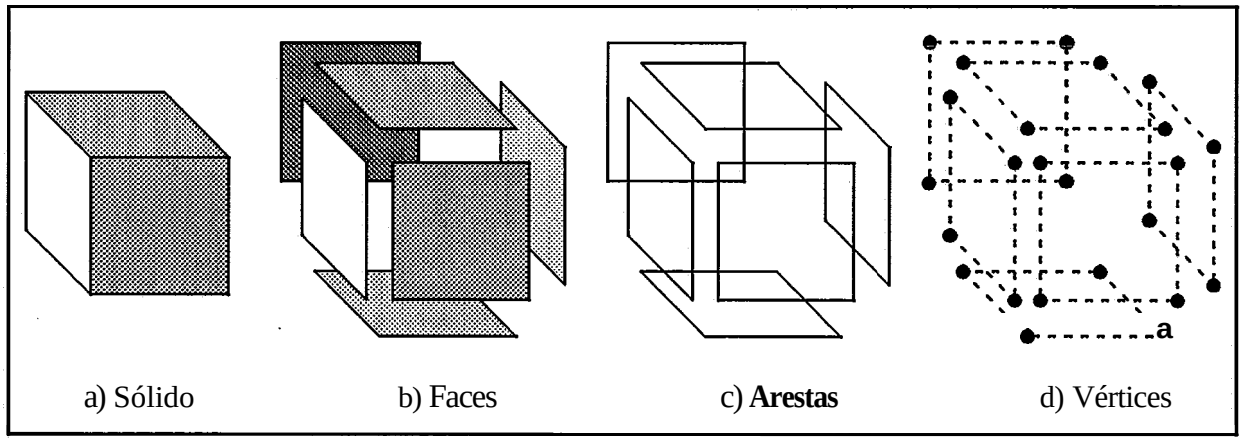


Figura 2.5 – Elementos básicos de um modelo BRep

Nos modelos BRep, os vértices, arestas e faces, apresentam as mesmas funções das primitivas CSG. Isto é, correspondem as entidades básicas a partir das quais as representações dos sólidos são construídas (figura 2.5). Dois tipos de informações coexistem, associadas aos elementos básicos do modelo BRep: as informações *geométricas* e as *topológicas*. As informações geométricas descrevem a geometria das superfícies, curvas e pontos que compõem a subdivisão de faces, arestas e vértices. As informações topológicas indicam a forma como vértices, arestas e faces se encontram relacionados, fornecendo respostas as seguintes perguntas:

- A quais arestas está associado cada vértice ?
- Que arestas compõem cada face e em que ordem devem ser percomdas ?
- Quais são as faces adjacentes e suas arestas comuns ?

O embasamento teórico da topologia de superfícies aqui utilizada é na verdade um assunto bem complexo. Entretanto, para fins de entendimento do texto aqui apresentado, é suficiente a introdução apresentada por Zeeman [ZEEM].

Quanto a definição de face, há uma certa discordância na literatura. Alguns artigos aceitam faces compostas por mais de um ciclo de arestas, ou seja, são permitidas faces com buracos [MANT82]. Em outros, no caso da existência de buracos, os mesmos são interligados por uma aresta auxiliar ao ciclo de arestas externo, de modo que uma face seja sempre composta por um único ciclo de arestas [MANT84, GUIB85]. A teoria por trás da definição do modelo BRep a ser aqui utilizado baseia-se na álgebra de arestas desenvolvida por Guibas e Stolfi [GUIB85], na qual as faces são descritas por um único ciclo de arestas. Análogo ao desenvolvimento do conceito de sólido, visto na seção 2.2, uma face será definida como a seguir:

Face é um subconjunto fechado, regular e conexo, de pontos situados sobre uma superficie semi-analítica por partes do E^3 e limitado por um conjunto de curvas semi-analíticas

Esta definição, permite a existência de faces com múltiplos ciclos de arestas. Para efeitos deste trabalho, as faces com buracos serão representadas interligando-se os ciclos internos ao externo através de uma aresta auxiliar. Topologicamente falando, nossas faces serão homeomorfas a um disco. A figura 2.6 exemplifica o conceito de face aqui adotado.

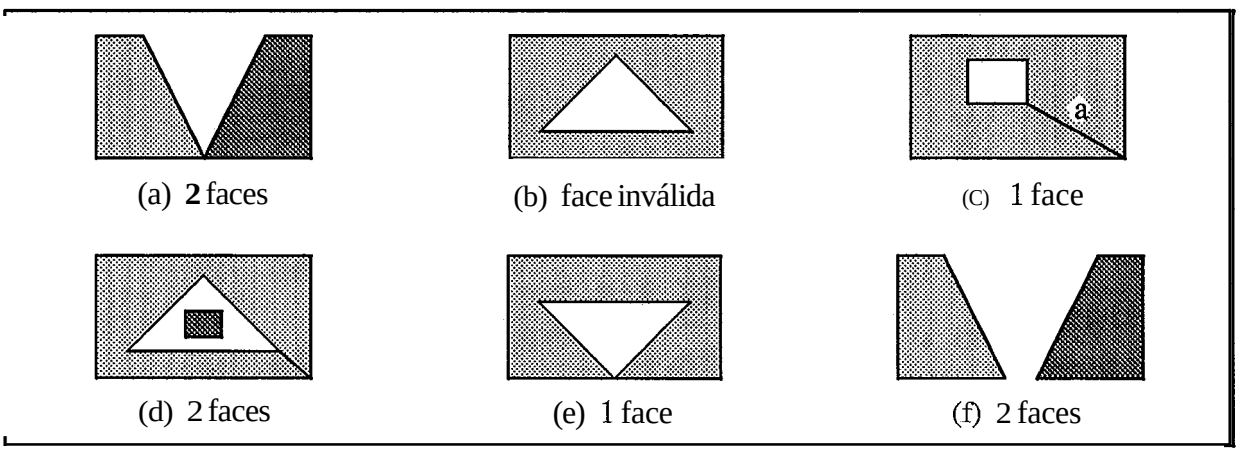


Figura 2.6 – Faces BRep válidas e inválidas

2.7.2. Definindo uma 2-variedade

Para que um sólido possa ser representado por um modelo BRep sua fronteira deve obedecer a certas restrições. O motivo destas restrições será discutido mais tarde ao serem analisados os operadores básicos de criação e alteração de uma representação BRep. Por ora, este estudo se limitará a enumerar algumas regras que, segundo [MANT84], são suficientes para garantir que uma coleção de faces, definidas como em 2.7.1, componha uma 2-variedade (ou *2-manifold*) do E^3. Cabe mencionar que, em geral, as 2-variedades compõem o domínio dos esquemas de representação por fronteira:

(1) Duas faces quaisquer só se interceptam em arestas e vértices da subdivisão

(2) Cada aresta é compartilhada por, exatamente, duas faces

(3) As faces ao redor dos vértices, formam um único circuito, isto é, elas podem ser organizadas em um ciclo, onde cada par consecutivo de faces, se intercepta em uma aresta adjacente ao vértice

Cabe aqui observar, que o ítem 2 acima se refere as faces topológicas, ou seja, uma aresta deve ser sempre compartilhada por duas faces topológicas, que podem, contudo, ter a mesma representação geométrica. Um exemplo disto é visto no exemplo [c) da figura 2.6 (aresta *a)*. Uma definição matemática para uma 2–variedade nos é também fornecida por Mantyla [MANT84]:

Uma 2–variedade V é um espaço, tal que cada ponto de V tem uma vizinhança aberta, topologicamente equivalente a um disco do E^2

Dois objetos cujas fronteiras não são 2–variedades e, portanto, não podem ser representadas em modelos por fronteira se encontram ilustrados na figura 2.7. Nela, pode–se observar que o vértice v, ou qualquer ponto situado sobre a aresta a não possui vizinhança equivalente a um disco do E^2.

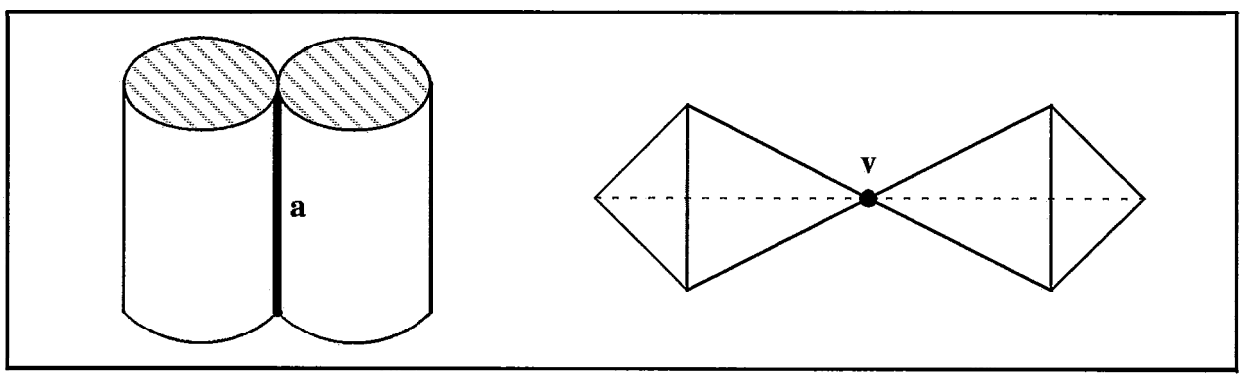


Figura 2.7 – Objetos sem representação BRep consistente

2.7.3. Operações em um Modelo BRep

As operações disponíveis aos modeladores BRep são classificadas em geométricas ou topológicas, conforme o tipo de dado por elas acessado. Os operadores topológicos de um modelo BRep têm por finalidade a criação e alteração de sua estrutura topológica, bem como, a manutenção da consistência desta estrutura. Eles serão vistos, detalhadamente, em etapas vindouras deste trabalho. Os operadores geométricos se encontram desvinculados da topologia e sua finalidade é preencher e atualizar os dados geométricos da representação, mantendo assim, sua consistência geometrica.

A consistência geometrica em um modelo BRep é uma idealização de implementação árdua, pois, como será explicado posteriormente, a aplicação do mesmo operador topológico pode acarretar resultados distintos, como a fusão de duas faces, ou a subdivisão de uma delas em duas outras. Cabe observar que, devido a natureza dos operadores topológicos a consistência topológica do modelo é sempre preservada [MANT82, GUIB85].

Quanto as operações geométricas de mais alto nível, a estrutura BRep se mostra favorável à aplicação de transformações afins (escala, rotação, translação), sendo de grande utilidade na visualização de um objeto sob diferentes pontos de refêrencia do espaço.

As operações booleanas, classificadas como mistas por envolverem tanto aspectos geométricos como topológicos, encontram grandes obstáculos a sua implementação. Isto se deve a dificuldade em se atualizar, simultaneamente, a geometria e a topologia da representação e também, ao grande número de cálculos de interseção entre superfícies, que resulta em processo moroso e carregado de erros numéricos.

2.7.4. Estrutura de Dados de Modelos BRep

Para que uma estrutura de dados possa suportar uma representação BRep ela deve ser capaz de armazenar a topologia do modelo, isto é, a forma com que se relacionam as faces, arestas e vértices de uma 2–variedade. Estas relações de vizinhança, vistas na figura 2.8 (onde se encontram ilustradas as arestas vizinhas a aresta a), foram estudadas por Baumgart [BAUM74], que propôs em sua representação *winged-edge* o armazenamento indiscriminado de todos os elementos vizinhos a cada aresta. Além disto, Baumgart sugeriu o uso dos ditos Operadores de *Euler* como estrutura de dados para manipular representações BRep.

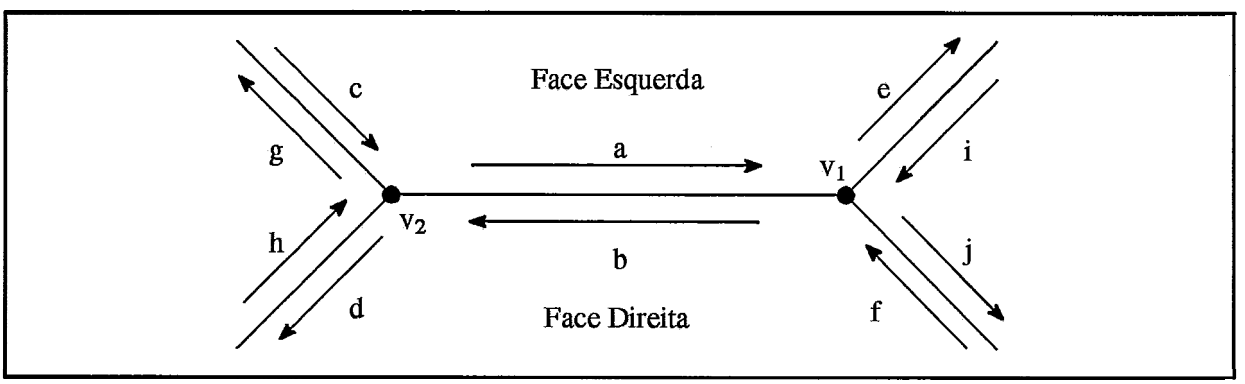


Figura 2.8 – Aresta BRep e sua vizinhança

Na década de 80, Guibas e Stolfi [GUIB85] surgiram com um trabalho elaborado, no qual demonstram que com apenas 2 operadores (2 informações de vizinhança) pode–se descrever por completo a topologia de uma 2–variedade. A esta estrutura de dados denominaram *Álgebra de Arestas*, na qual a aresta topológica é o único elemento explicitamente necessário (sendo cada aresta geométrica composta de 2 semi–arestas topológicas orientadas). Isto significa que, a partir da topologia das arestas somos capazes de extrair os vértices e faces ao seu redor, os quais correspondem a ciclos de arestas. Além disto, devido as características dos operadores sugeridos, os acessos as informações de vizinhança relativas a uma dada aresta, são feitos em tempo constante, como veremos no capítulo 4.

2.7.5. Propriedades dos Modelos BRep

O espaço de modelagem dos esquemas de representação por fronteiras, depende dos tipos de superfícies aceitas pelo modelo. Além disto, o objeto representado é, em geral, uma 2–variedade, muito embora já hajam pmpostas para eliminar esta restrição.

A validade geométrica de modelos BRep é extremamente difícil de ser assegurada sem restrições rígidas a aplicação dos operadores topológicos e geométricos. Em contrapartida, a consistência topológica é uma característica do modelo.

São várias as formas com que a fronteira de uma 2-variedade pode ser subdividida em faces. Havendo, portanto, diversas representações sintaticamente corretas de um mesmo objeto. Esta característica faz dos modelos BRep esquemas de representação não-únicos.

Cada representação BRep sintaticamente correta está associada a uma única 2-variedade orientável sem borda, não existindo ambiguidade na representação.

Quanto a concisão do modelo, representações BRep de objetos usuais podem se tornar muito extensas. Este problema é acentuado se for adotada a aproximação de superfícies curvas por faces planas.

2.7.6. Conclusões e Notas Bibliográficas

Os modelos de representação por fronteira têm evoluído desde a concepção dos Operadores de Euler propostos por Baumgart [BAUM74]. Tais modelos estão sedimentados sob vasta e rigorosa teoria matemática: a *Topologia de Superfícies* [ZEEM].

Modelos BRep se mostram de grande utilidade na visualização de objetos, visto ser inerente a sua estrutura os dados necessários a tal operação.

2.8. MODELOS HÍBRIDOS

Nenhum dos três esquemas de representação abordados nos ítens anteriores se mostra superior aos outros em todos os aspectos. Os modelos construtivos são os mais concisos e os que apresentam maiores facilidades na geração de sólidos complexos. Os modelos de decomposição, por sua vez, têm grande valia na computação de propriedades integrais, mesmo sendo os resultados obtidos apenas aproximações. Já os modelos de representação por fronteiras são portadores de uma estrutura de dados altamente favorável a aplicações gráficas.

Em um sistema de modelagem ideal devem coexistir vários modelos de representação de sólidos, permitindo que as operações desejadas sejam processadas através do modelo que melhor convir, objetivando resultados mais eficazes ou velozes. O termo *híbrido* é associado aos sistemas portadores desta característica.

Para que modelos distintos possam iteragir em um sistema híbrido é necessário a elaboração de algoritmos de conversão entre os esquemas de representação ali presentes. Estes algoritmos devem levar em consideração o fato de serem distintos os domínios dos modelos envolvidos na conversão. O domínio dos modelos construtivos, por exemplo, limita-se aos r-sets, os quais não possuem necessariamente representação BRep (figura 2.7).

Outro problema, decorrente da multiplicidade de modelos, vem a ser a consistência global. Isto é, ao se aplicar uma operação a uma representação, seus resultados devem ser consistentemente propagados às outras representações existentes.

O leitor interessado encontrará em [REQUSO, REQU82, MANT88] informações a respeito de sistemas híbridos em funcionamento, bem como suas arquiteturas e os modelos de representação coexistentes em cada um. Com relação aos algoritmos de conversão, em Kunii *et all* [KUNI85] é visto um método de conversão de Octree para Brep. A situação inversa é tratada por Tang *et all* [TANG88]. Requicha e Voelcker [REQU85] propõem uma forma de se passar de uma representação CSG do sólido para seu equivalente BRep, ao passo que a volta é sugerida por Juan [JUAN88]. Por último, cita-se Navazo *et all* [NAVA87], onde é apresentado um algoritmo de conversão de CSG para Octree.

2.9. AVALIAÇÃO DE FRONTEIRAS

Seja para a existência de um modelo híbrido CSG/BRep, ou para a aplicação de algorítmos de visualização em representações CSG, é fundamental a presença de um método de conversão capaz de obter a representação BRep correspondente a um sólido expresso por uma árvore CSG. Este procedimento é conhecido como *Avaliação* de *Fronteiras* (*Boundary Evaluation*).

2.9.1. A Proposta de Requicha e Voelcker

Uma abordagem natural para o procedimento de avaliação de fronteiras é composta de duas etapas. Na primeira, cada primitiva da árvore CSG tem sua representação BRep processada. A seguir, as operações booleanas indicadas na árvore CSG são aplicadas sobre estas representações. Esta estratégia se encontra presente nos algoritmos propostos por Requicha e Voelcker [REQU85], os quais passamos a descrever.

O primeiro passo consiste na geração de todo o conjunto de possíveis faces e arestas do sólido. A seguir, cada elemento é testado contra o modelo CSG a fim de identificar os que efetivamente fazem parte da fronteira do sólido em questão. O conjunto das faces e arestas candidatas mencionado é gerado a partir das primitivas que compõem o modelo CSG. Ou seja, cada face ou aresta que pertença a representação BRep da primitiva é adicionada ao referido conjunto. O algoritmo segue então para o cálculo das faces e arestas resultantes da interseção destes elementos. Esta vem a ser a etapa mais dispendiosa computacionalrnente, devendo cada face de uma primitiva ser testada contra todas as faces das demais. Erros numéricos podem decorrer deste processo, principalmente no caso da existência de faces quase coplanares.

Para finalizar, as faces e arestas candidatas obtidas são classificadas com relação ao sólido composto através de um procedimento conhecido como *Classificação* de *Pertinência a Conjunto (Set Membership Classification)*. Apenas as arestas e faces situadas sobre a fronteira do sólido serão aproveitadas, sendo descartadas aquelas que

foram avaliadas como no interior ou no exterior do objeto. Por exemplo, ao considerarmos o sólido composto de duas primitivas P_1 e P_2, associadas ao operador de diferença, cada elemento candidato é classificado inicialmente com relação a P_1 e P_2 isoladamente; apenas aqueles pertencentes a fronteira de P_1 e no exterior de P_2 OU então, os presentes na fronteira de P_2 e interiores a P_1 serão considerados na representação por fronteiras do sólido $P_1 \backslash P_2$. Regras semelhantes são aplicadas para os operadores de união e interseção e o procedimento é repetido, recursivamente, no restante dos nós da árvore CSG.

A classificação de pertinência pode vir a apresentar ambiguidade no caso de haver fronteiras que se tangenciem [REQU85]. A solução adotada consiste na adição de informações de vizinhança as classificações, de modo a identificar o lado sobre o qual o material do objeto se encontra.

2.9.2. A Nossa Proposta

Na técnica apresentada anteriormente, parte-se das representações por fronteira das primitivas CSG e a partir da computação de operações booleanas entre elas obtém-se o modelo por fronteira do sólido composto. Uma segunda estratégia, seria a construção do modelo por fronteiras a partir, diretamente, da forma do objeto. Isto é, a forma do sólido composto seria assimilada, de modo a permitir a computação dos elementos pertencentes a sua fronteira sem a necessidade da aplicação de operações booleanas complexas sobre representações BRep.

Avaliar a fronteira de um objeto descrito por uma árvore CSG pode parecer, a princípio, uma tarefa árdua. Entretanto, se dividirmos o problema pela metade, separando o sólido em duas partes, a avaliação da fronteira de cada uma das metades seria, possivelmente, menos trabalhosa. Por outro lado, ao ser efetuado um corte sobre o objeto, o mesmo é separado de tal forma que, para ser reconstituído basta agregar novamente suas duas porções. Isto é possível a partir de uma simples união entre conjuntos não penetrantes, ou seja, conjuntos de pontos que só se interceptam em suas fronteiras. Com esta técnica, os operadores genéricos de união, interseção e diferença são substituídos por um único operador, bem menos complexo, que chamaremos de *união disjunta.*

A avaliação da fronteira de cada urna das metades citadas pode se apresentar ainda, como uma tarefa que exija o dispendio de grande esforço. Neste caso, deve-se continuar o processo de retalhamento do objeto até que a porção de sua fronteira a ser avaliada seja simples o suficiente para ser computada diretamente. Tendo sido calculados todos os retalhos que compõem a casca do sólido, resta apenas realizar a união disjunta entre as partes vizinhas a fim de gerar o modelo BRep do sólido.

A proposta utilizada neste trabalho, baseada no método descrito acima, pode ser decomposta nas três etapas listadas a seguir:

- **Subdivisão do espaço**

O paradigma da subdivisão espacial é usado na simplificação do problema de se avaliar a fronteira do sólido como um todo. O espaço é subdividido até obtermos porções do mesmo, nas quais a computação da fmnteira do objeto seja trivial.

- **Geração de retalhos**

Para cada célula da subdivisão espacial é construída uma representação BRep correspondente ao pedaço da casca do sólido (retalho) que passa por seu interior.

- **União dos retalhos**

Uma a uma, as representações parciais da fmnteira do objeto (retalhos) são conectadas através de colagem (uniões disjuntas) de modo a constituir o modelo BRep do sólido completo.

2.10. CONCLUSÕES

Foram abordadas neste capítulo as diversas técnicas de representação de sólidos mais comumente encontradas na literatura de modelagem de sólidos. Tivemos também a oportunidade de exibir superficialmente duas metodologias de conversão de representações CSG para BRep. Vimos que a estratégia por nós adotada se utiliza, além de algum método de subdivisão espacial, praticamente de todos os modelos de representação citados, a saber, do modelo CSG, de uma variação do modelo Octree e, finalmente, do modelo BRep.

Capitulo 3

Subdivisão Espacial Aplicada sobre Árvores CSG

3.1. INTRODUÇÃO

A idéia de utilizar o paradigma de subdivisão espacial no âmbito de representações CSG foi abordada na literatura com própositos distintos: Tilove [TILO84] aplicou tal procedimento na detecção de objetos CSG nulos; Navazo at all. [NAVA87] o utilizou ao elaborar uma estrutura intermediária (a qual chamou de octree extendida) entre as representações CSG e BRep; Wyvill e Kunii [WYVI86], bem como Comba [COMB91], trabalharam com uma estrutura híbrida entre octree e CSG objetivando a otimização do procedimento de visualização de objetos CSG.

Aqui, como em [NAVA87], o objetivo é reduzir a complexidade do processo de conversão de representações CSG para BRep.

3.2. A ESTRUTURA HÍBRIDA ENTRE OCTREE E *CSG*

O processo de subdivisão do espaço CSG através de uma octree, consiste em subdividir o problema inicial, a saber, a aplicação de algum algoritmo sobre a árvore CSG, em oito problemas de menor complexidade e assim o fazer, sucessivamente, até que os mesmos se tomem simples o suficiente para serem efetivamente solucionados. Neste ponto, os resultados parciais obtidos são reagrupados na ordem inversa a da subdivisão, obtendo assim, a solução final pmcurada.

É preciso, portanto, uma estrutura capaz de gerenciar a subdivisão do espaço em octantes e, simultaneamente, armazenar representações CSG simplificadas em cada um deles. Tal estrutura híbrida entre octree e CSG foi proposta inicialmente por Wyvill e Kunii [WYVI86]. Ela se assemelhava a um modelo octree, porém, além dos nós cheios ou

vazios, suas folhas poderiam conter um novo tipo de informação: uma simplificação da árvore CSG original. Denominaremos de *árvore* ***OCSG*** a uma estrutura portadora desta característica. Assim sendo, uma árvore OCSG possui os seguintes tipos de octantes terminais:

- **Octante vazio:** região do espaço não interceptada pelo sólido.
- **Octante cheio:** região do espaço completamente contida no sólido.
- **Octante** CSG: contém uma simplificação da árvore CSG original que descreve a geometria do sólido no interior do octante.

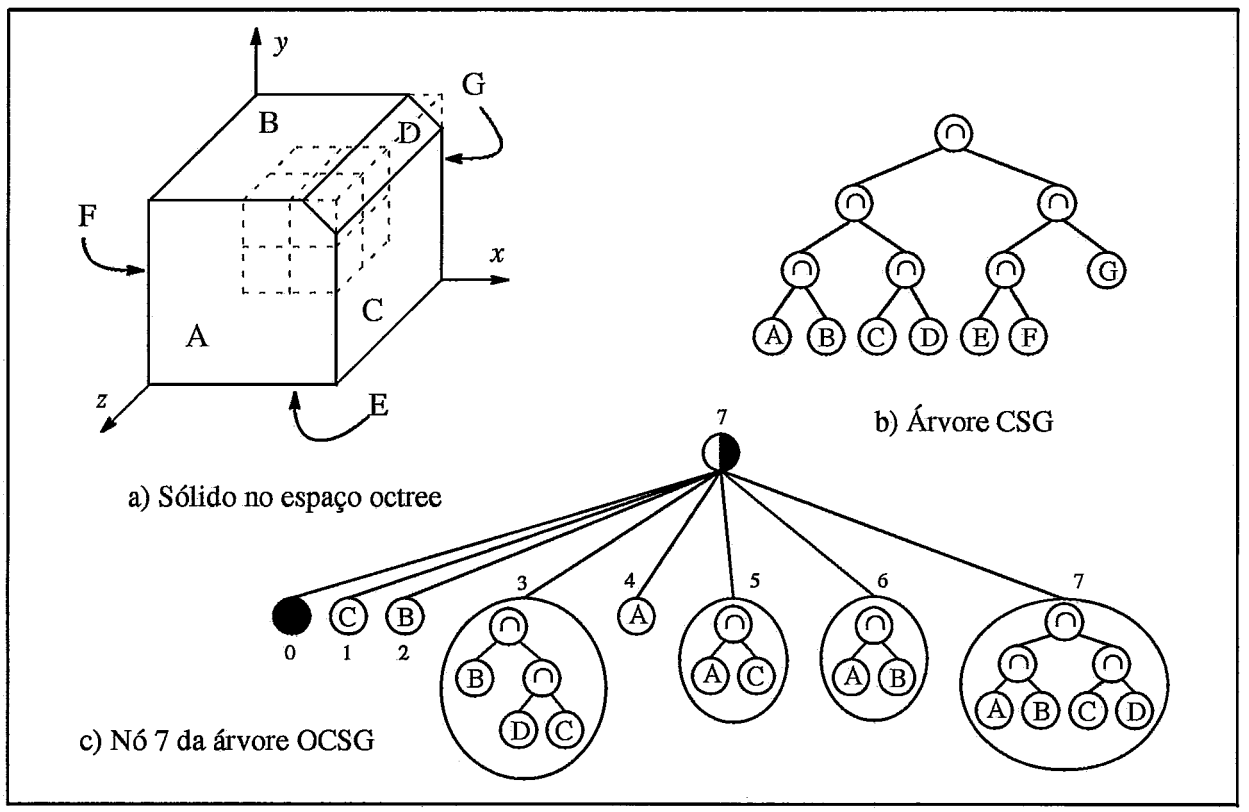


Figura 3.1 – Estrutura híbrida Octree/CSG

A figura **3.1** exemplifica o processo de subdivisão espacial descrito acima, exibindo um dos nós mistos da árvore OCSG obtida. O restante deste capítulo será dedicado a compreensão do processo de localização de uma árvore CSG. Para tanto, vamos nos basear nos resultados obtidos por Tilove [TILO84], objetivando responder a questão formulada a seguir:

Qual a menor sub-árvore CSG, **extraída da representação** CSG **de um sólido, que o representa plenamente em uma dada região do espaço** ?

3.3. CLASSIFICAÇÃO DE PRIMITIVAS CSG

Uma primitiva CSG é dita *redundante*, caso não seja responsável por nenhuma contribuição na forma do sólido representado. Ou seja, sua presença na árvore CSG não remove nem acrescenta material ao conjunto de pontos que compõem o sólido.

A constatação da redundância de uma primitiva nos é dada, se a mesma puder ser substituída na árvore CSG, ou pelo conjunto vazio ($\emptyset$), ou pelo universo ($\mathcal{W}$), sem que o sólido representado seja alterado. Nestes casos, a primitiva é dita @-redundante ou W-redundante, respectivamente. Deste modo, na união de duas primitivas esféricas P_1 e P_2, ambas centradas na origem e tendo P_2 um raio menor que P_1, P_2 corresponde a uma primitiva @-redundante. Já na diferença $P_2 \setminus P_1$, P_1 faz o papel de uma primitiva $\mathcal{W}$-redundante.

A redundância de primitivas CSG pode ser analisada sob os pontos de vista global (como acabamos de ver) e local. Em termos globais, chega-se a conclusão da redundância de uma primitiva testando-a contra o universo (o espaço onde o sólido se encontra modelado). Entretanto, existem casos, como no uso da subdivisão espacial, onde surge a necessidade de se conhecer a geometria do sólido em uma determinada região limitada do espaço. Para estes fins, uma primitiva será considerada iedundante em uma região R do espaço se a sua troca pelo conjunto vazio ou universo na expressão do sólido não modificar sua geometria no interior de R.

Comba [COMB91] propõe critérios de classificação, onde cada primitiva é substituída por um conjunto de semiespaços planos, os quais podem ser facilmente testados contra uma dada região do espaço. Salim et all [SALI91] sugere um método, aplicável a subdivisões simpliciais do espaço, válido para primitivas polinomiais expressas na base de Bemstein.

O primeiro passo da localização de uma árvore CSG, denominado Classificação de *Primitivas* ***CSG,*** consiste em substituir suas primitivas classificadas como $\emptyset$ ou $\mathcal{W}$-redundantes pelos conjuntos vazio ou universo, respectivamente.

3.4. SIMPLIFICAÇÃO DE ÁRVORES CSG

A partir da árvore cujas primitivas redundantes foram devidamente classificadas, inicia-se o processo de simplificação. Este consiste na remoção dos nos vazios e universo, aplicando-se recursivamente as propriedades das operações entre conjuntos de pontos enumeradas na tabela 3.1, onde A e B são árvores CSG.

Expressão	$A \cup \emptyset$ $\emptyset \cup A$ $A \cap \mathcal{W}$ $\mathcal{W} \cap A$ $A \setminus \emptyset$	$A \cap \emptyset$ $\emptyset \cap A$ $\emptyset \setminus A$ $A \setminus \mathcal{W}$	$A \cup \mathcal{W}$ $\mathcal{W} \cup A$ $\mathcal{W} \setminus \emptyset$	$\mathcal{W} \setminus A$
Simplificação	A	$\emptyset$	$\mathcal{W}$	$\overline{A}$

Tabela 3.1 – Simplificação de expressões booleanas

A única dificuldade decorrente da aplicação destas regras surge na operação de diferença entre o conjunto universo We um conjunto de pontos definido por uma árvore A, pois, nesta situação, é preciso computar o complemento de A ($\tilde{A}$). Tal problema é

solucionado ao se interpretar as operações booleanas de uma árvore CSG como operações entre conjunto de pontos e aplicar o operador de complemento sobre os conjuntos envolvidos, donde chegamos a:

$$\overline{(A \cup B)} = \overline{A} \cap \overline{B} \qquad \overline{(A \cap B)} = \overline{A} \cup \overline{B} \qquad \overline{(A \setminus B)} = \overline{A} \cup B$$

As propriedades acima são aplicadas recursivamente até que um nó terminal da árvore CSG seja atingido. Só então, o complemento da primitiva é efetivamente computado.

3.5. LOCALIZAÇÃO DE ÁRVORES *CSG*

A combinação do procedimento de classificação com o de simplificação de uma árvore CSG consiste naquilo que conhecemos como localização de árvores CSG. Uma definição formal deste conceito é vista em Esperança [ESPE90] nos seguintes termos:

Seja S um sólido representado por uma árvore CSG A, formada pelas primitivas P_1, $P_2, P_3, \ldots, P_n$. A localização de A com relação a uma região R do espaço é uma árvore CSG A^L composta pelas primitivas $P_{i1}, P_{i2}, \ldots, P_{im}$, tal que:

$$A \cap R = A^L \cap R \qquad \& \qquad \{P_1, P_2, P_3, \ldots, P_n\} \supset \{P_{i1}, P_{i2}, \ldots, P_{im}\}$$

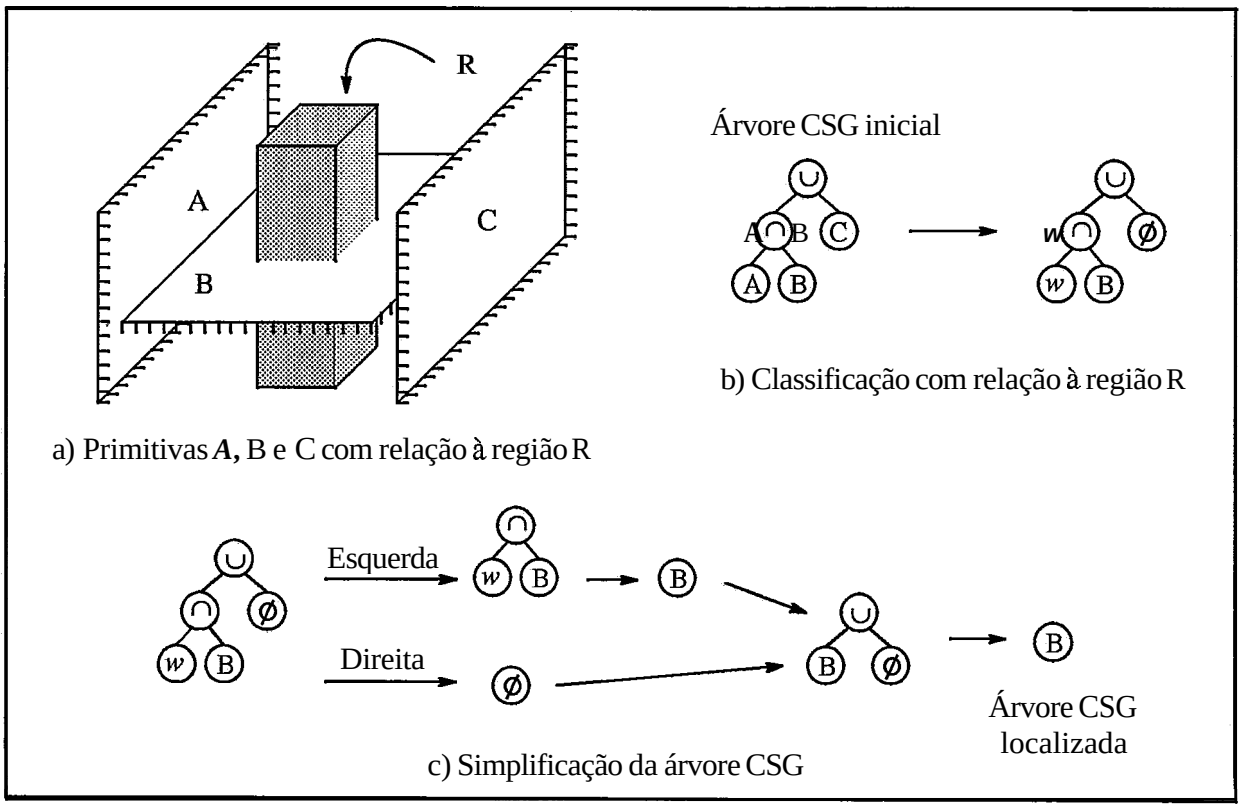


Figura 3.2 – O processo de localização de uma árvore CSG

E importante notar, que as árvores CSG localizadas podem ser consideravelmente mais simples que Aquelas das quais foram extraídas. Fato este, decorrente da remoção

das primitivas redundantes. A figura 3.2 ilustra a técnica de localização de uma árvore CSG.

3.6. O ALGORITMO DE LOCALIZAÇÃO DE ÁRVORES CSG

Com base nas informações deste capítulo, podemos propor um algoritmo que compute a árvore CSG localizada com relação a uma região R do espaço. Este é constituído por dois procedimentos básicos, a saber:

- ***ClassificaPrimitiva***

Recebe como parâmetro um nó primitiva de uma árvore CSG e o classifica quanto a sua redundância com relação a uma dada região do espaço, substituindo-o por um nó vazio ou universo, caso o mesmo seja classificado como $\emptyset$ ou $\mathcal{U}$-redundante, respectivamente.

- ***SimplificaCsg***

Recebe como parâmetros as sub-árvores esquerda (E) e direita *(D)* de um nó CSG, bem como a operação que as combina *[op).* Aplica as propriedades expostas na tabela **3.1** a expressão booleana E *op* D e retoma o resultado obtido.

```
LocalizaCSG (ÁrvoreCsg, R) {
    Se NÓ(ÁrvoreCsg) for Primitiva Retorne(ClassificaPrimitiva(ÁrvoreCsg, R))
    senão {
        (" A localização é aplicada as sob-Arvores esquerda e direita *)
        ÁrvoreEsq = LocalizaCSG(Esq(ÁrvoreCsg), R)
        ÁrvoreDir = LocalizaCSG(Dir(ÁrvoreCsg), R)
        Retorne(SimplificaCSG(ÁrvoreEsq, ÁrvoreDir, OP(ÁrvoreCsg)))
    }
}
```

3.7. CONCLUSÕES

Vimos neste capítulo como o paradigma de subdivisão espacial pode ser aplicado no âmbito da representação CSG de sólidos. Embora este processo tenha sido aqui descrito com o uso de subdivisão octree do espaço, observa-se que o mesmo não oferece restrições com relação ao método de subdivisão adotado, desde que seja possível determinar o posicionamento espacial e as dimensões de cada célula da subdivisão.

Capítulo 4

O Modelo de Representação de *Superfícies*

4.1. INTRODUÇÃO

Este capítulo tem por objetivo a descrição do modelo de representação de superfícies por nós desenvolvido e utilizado na construção de representações BRep de sólidos. *Alguns* conceitos extraídos da topologia de superfícies e que serão aqui utilizados podem ser encontrados em [LIMA85, LIMA87, ZEEM].

O modelo BRep de representação de sólidos visa a descrição destes a **partir** de sua fronteira. A fronteira de um sólido pode ser definida claramente como sendo uma superfície fechada e sem bordos, imersa no espaço E^3, cujas vizinhanças de cada ponto pertençam parcialmente ao interior e ao exterior do sólido. Donde se segue que um modelador BRep e na verdade um veículo que nos fornece meios consistentes de representação de superfícies fechadas e sem bordos.

Dentre as possíveis estratégias de construção de representações BRep, iremos aqui comentar duas delas. A primeira, parte do conceito intuitivo de gerar um sólido complexo através de operações booleanas entre sólidos primitivos. Em outras palavras, a representação da fronteira de um sólido complexo seria gerada a partir de operações booleanas entre representações das fronteiras de sólidos primitivos. Para que esta idéia pudesse ser posta em prática, deveríamos dispor de operadores booleanos genéricos que seriam aplicados sobre representações BRep. Estudos sobre este tema [REQU85, MANT88, PAOL89] têm apresentado grandes dificuldades e altos custos na implementação de tal estratégia.

Uma segunda proposta, a qual foi por nós adotada, pode ser informalmente descrita como um processo análogo ao da confecção de uma colcha de retalhos. Em um primeiro passo, porções mínimas da superfície desejada, as quais denominaremos

retalhos, são criadas isoladamente. Os retalhos correspondem a superfícies fechadas com bordo e uma única face. Estes retalhos serão posteriormente *costurados* um a um através de seus bordos, dando origem a superfície final. A princípio, toda superfície assim gerada possuirá tantas faces quanto tenha sido o número de retalhos utilizados em sua construção. Com isto, ficará definida sobre esta superfície uma subdivisão decorrente de seu processo gerador, na qual cada face estará associada a um retalho. A figura 4.1 ilustra a construção de um cubo a partir deste procedimento. Note que a implementação de tal estratégia requer apenas 2 operadores de alto nível, um deles responsável pela criação dos retalhos e o outro pela costura dos mesmos. Observe por último que, como cada retalho corresponde a uma superfície fechada com bordo, no decorrer do processo descrito iremos constantemente nos deparar com a necessidade de representarmos e manipularmos tais superfícies. Nossa representação equivalerá ao BRep de algum sólido apenas ao término da costura dos retalhos, quando então teremos efetivamente representada uma superfície fechada e sem bordo.

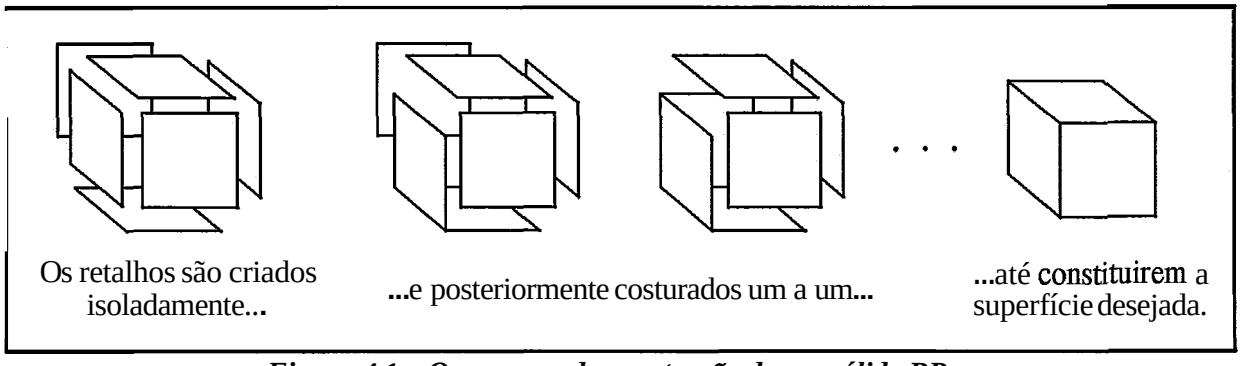


Figura 4.1 – O processo de construção de um sólido BRep

Passaremos a seguir a descrição pormenorizada do processo adotado para a geração de uma representação BRep. Trataremos inicialmente de definir o conceito de subdivisões de superfícies, pois nestas subdivisões encontraremos os dados necessários a criação dos retalhos citados anteriormente. Veremos na sequência, a metodologia desenvolvida para que tais entidades, bem como as superfícies decorrentes de sua *costura*, sejam consistentemente representadas. Por fim esclareceremos os detalhes do processo de costura dos retalhos.

4.2. SUBDIVISÕES DE SUPERFÍCIES COM BORDO

Aqui descreveremos formas de representação da topologia de uma superfície através de um grafo não direcionado. Trabalhos girando em tomo deste assunto são fartamente encontrados na literatura voltada a modelagem de sólidos [JAME55, MANT84, GUIB85].

Nesta seção serão apresentados vários resultados advindos do estudo da topologia de superfícies, não sendo do escopo deste trabalho a demonstração de tais afirmações. O leitor interessado poderá encontrar um embasamento teórico nas referências [ZEEM, LIMA85, LIMA87]. Antes de nos familiarizarmos com subdivisões de superfícies vamos enunciar alguns conceitos topológicos básicos necessários ao seu entendimento.

Chama-se de uma *2-variedade* todo conjunto de pontos *V*, onde cada ponto *v* de V possui uma vizinhança homeomorfa ao disco unitário aberto expresso pelo conjunto $B_1 = \{p \in R^2 / |p| < 1\}$. Uma *2-variedade com bordo* é um conjunto de pontos *V*, onde cada ponto *v* de *V* possui uma vizinhança homeomorfa ao disco ou ao semi-disco unitário aberto. Este último é constituído pelos pontos pertencentes ao conjunto $B_2 = \{p=(x,y) \in R^2 / |p| < 1 \ \& \ x \geq 0\}$.

Vamos denominar de *contorno* o conjunto de pontos de uma superfície, cujas vizinhanças sejam homeomorfas ao ao semi-disco aberto unitário. A cada componente conexa do contorno de uma superfície denominaremos de *bordo*. Sob esta concepção, o contorno de um cano oco é composto por dois bordos, cada um situado em uma de suas extremidades.

Dois subconjuntos A e B do espaço topológico V são ditos *incidentes* se toda vizinhança de A possuir pontos em comum com toda vizinhança de B.

Uma *curva* de *V* é um subespaço de V homeomorfo ao intervalo aberto $I=(0,1)$ dos reais.

De posse das definições expostas acima, podemos finalmente expor o conceito de *subdivisão de uma 2-variedade* V, como sendo uma partição S de V em três coleções finitas de partes disjuntas, a saber, seus vértices, arestas e faces. Estes elementos, por sua vez, devem possuir as seguintes propriedades:

(S1) Todo vértice é um ponto de V

(S2) Toda aresta é uma curva de V

(S3) Toda face é um disco aberto de V

(S4) A fronteira de toda face é composta por um único ciclo fechado de arestas e vértices

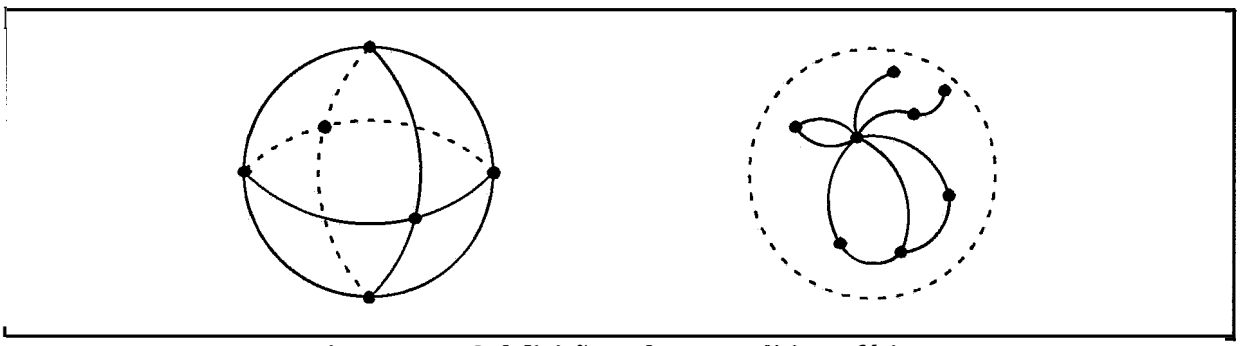


Figura 4.2 - Subdivisões sobre superfícies esféricas

Esta definição, extraída de [GUIB85], foi lá trabalhada no âmbito de 2-variedades. A figura 4.2 exibe exemplos de subdivisões sobre uma casca esférica atendendo as propriedades *S1-S4*. Concluiremos aqui, que uma subdivisão assim caracterizada é perfeitamente aplicável a 2-variedades com bordos. Para tanto, basta importarmos da topologia de superfícies o resultado de que um bordo forma uma 1-variedade, ou seja, é

um conjunto de pontos cujas vizinhanças são homeomorfas ao intervalo aberto $I = (0,1)$ dos reais. Desta informação pode-se deduzir que um bordo é combinatoriamente descrito por um ciclo fechado de vértices e arestas da subdivisão. Combinatoriamente falando, um bordo e uma face da subdivisão são igualmente definidos. Por formarem uma 1-variedade, não é possível que os bordos de uma dada superfície compartilhem uma mesma aresta ou vértice da subdivisão, isto é, não devem haver bordos incidentes. A figura 4.3 ilustra uma subdivisão válida, realizada em uma superfície com bordo.

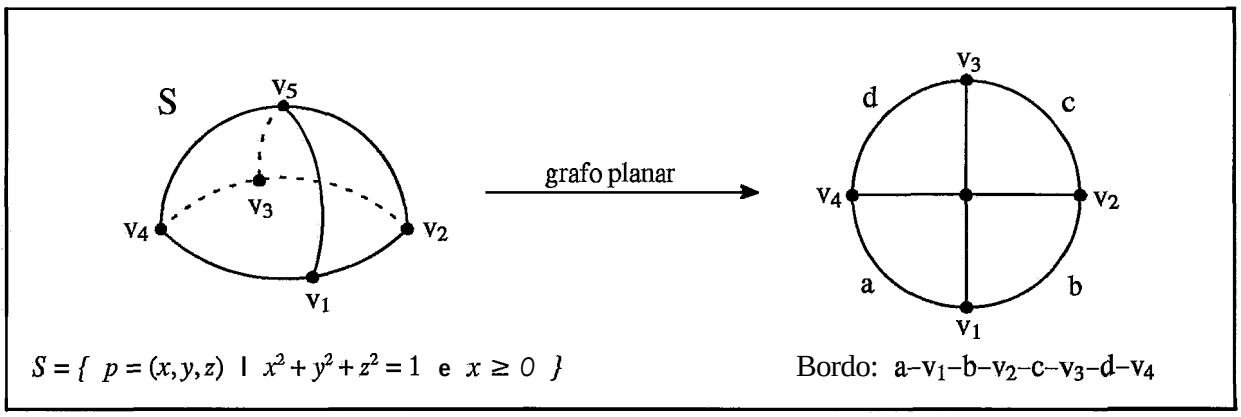


Figura 4.3 - Subdivisão de unta superfície com bordo

Toda aresta da subdivisão possui, obviamente, dois lados. Um resultado importante é o fato de, a cada um destes lados estar associado ou uma face ou um bordo da subdivisão. Vale também observar que a subdivisão, conforme definida acima, satisfaz a equação de Euler: $v - a + f = 2(c - g)$, onde v é o número de vértices, a o de arestas, f o de faces mais o de bordos, c o de componentes conexas e g o genus da superfície.

Com relação a definição de face, vemos que neste modelo não são aceitas faces com buracos, isto é, faces constituídas por mais de um ciclo fechado de vértices e arestas. A representação de tais faces se dará a partir do acréscimo de novas arestas interligando os diversos ciclos existentes em um único ciclo fechado de vértices e arestas.

4.3. O MODELO DE REPRESENTAÇÃO DE SUBDIVISÕES

Vimos na seção anterior como se comporta urna subdivisão sobre uma superfície com bordo. Agora será discutido o esquema de composição do modelo por nós desenvolvido com o intuito de representar tais subdivisões. O esquema aqui proposto e detalhado a seguir se encontra ilustrado na figura 4.4.

Uma superfície sendo representada pode ser constituída por várias partes desconexas. Estas partes serão identificadas em nosso modelo por uma lista de componentes conexas. Cada elemento desta lista possui informações a respeito da componente da subdivisão a ele relacionado. Podemos ter, por exemplo, a totalização de seus números de vértices, arestas e faces (a partir dos quais é possível deduzir o genus da componente).

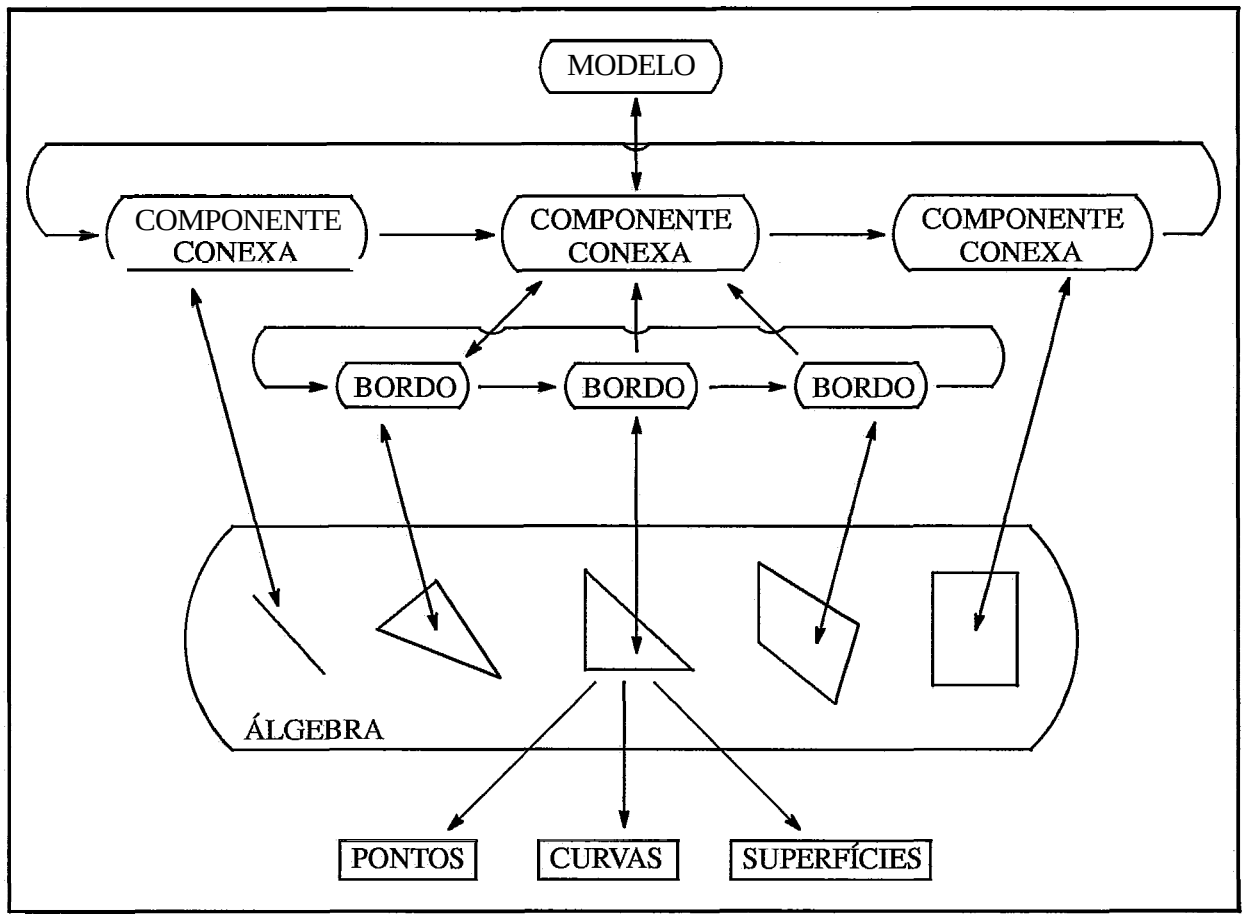


Figura 4.4 – Modelo de representação de superfícies com bordo

Vimos que, em uma subdivisão, bordos e faces podem ser confundidos por serem igualmente representados através de ciclos de vértices e arestas. No modelo adotado, cada componente conexa terá a si associada uma lista cujos elementos referenciam os seus bordos e, onde cada bordo é representado pelo ciclo que o compõe.

Na base do modelo, encontramos a representação da topologia da subdivisão da superficie. A estrutura de dados utilizada para este fim será denominada álgebra *de arestas.* Nesta representação, os ciclos de arestas e vértices que compõem os bordos se encontram diretamente associados aos elementos da lista de bordos que os identificam. Observe que, no caso de representações BRep de sólidos, a lista de bordos de cada uma das componentes conexas existentes será vazia.

Por fim, isolada da álgebra, porém a ela associada, temos as informações não topológicas da subdivisão sendo representada. Em outras palavras, os dados que descrevem os pontos, curvas e superfícies da 2–variedade com bordo, relacionados respectivamente aos vértices, arestas e faces da subdivisão. Um maior detalhamento deste modelo e dos interrelacionamentos de seus elementos será descrito no decorrer deste capítulo, através de suas particularidades e implementação.

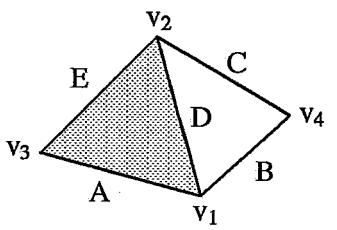
4.4. A ESTRUTURA DE DADOS PARA REPRESENTAÇÃO DE SUBDIVISÕES

A base do modelo de representação de subdivisões de superfícies recai em uma estrutura de dados responsável por representar estas subdivisões. A estrutura de dados

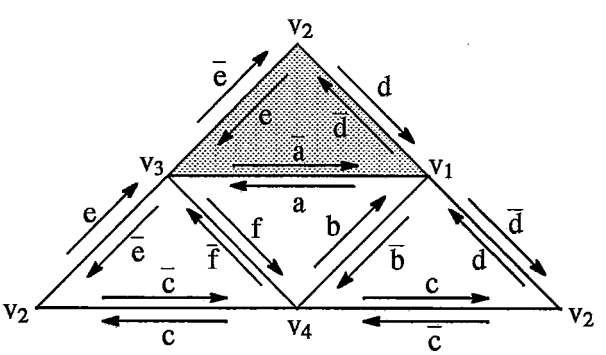
comumente citada na literatura, que tem por finalidade a execução desta tarefa, está fundamentada em uma família de operadores topológicos conhecidos como Operadores de *Euler* [BAUM74, MANT82, MANT84]. Guibas e Stolfi [GUIB85], na busca de novos caminhos para solucionar o problema de computar Diagramas de Voronói, conceberam sua *Álgebra* de Arestas, uma estrutura de dados sedimentada na topologia de superfícies e que pode, como veremos, ser facilmente adaptada ao âmbito da modelagem BRep.

Dois resultados fundamentais podem ser atribuídos a álgebra de arestas. Um deles é o de que qualquer subdivisão construída, que atenda as propriedades *S1-S4* vistas na seção 4.2, tem sua topologia capturada via alguma álgebra. O segundo resultado garante que, dada uma álgebra de arestas genérica, esta corresponderá a uma subdivisão de alguma superfície, isto é, qualquer que seja a álgebra de arestas construída, a ela estará associada uma superfície realizável.

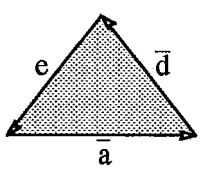
Mecanismos são desenvolvidos objetivando a associação dos elementos da álgebra as informações geométricas da subdivisão por ela representada. Se isto for feito de modo a garantir a consistência geométrica do modelo, enquanto o mesmo esteja sendo construído, pode-se concluir que esta estrutura de dados representa fidedignamente a subdivisão de uma superfície.



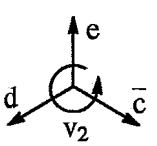
a) Tetraedro no espaço



b) Representação através de um grafo planar



c) Identificação de uma face a partir de um ciclo de arestas



d) Identificação de um vértice a partir de um ciclo de arestas

Figura 4.5 – A representação BRep de um sólido através de grafos planares

Na estrutura de Guibas e Stolfi o grafo contendo a subdivisão é orientado, de modo que cada uma de suas arestas passa a ser composta por duas arestas orientadas. Além disto, o único elemento que aparece explicitamente representado é a aresta, uma vez que vértices e faces (ou bordos) podem ser combinatóriamente descritos por ciclos de arestas orientadas. A figura 4.5 exibe este conceito a partir do grafo orientado correspondente a subdivisão da superfície de um tetraedro. Neste trabalho, as arestas serão orientadas de modo que as faces por elas cicladas se situem a sua esquerda. Deste

momento em diante, ao nos referirmos a uma aresta da álgebra, estaremos identificando uma dada aresta orientada da subdivisão.

4.4.1. Funções de Arestas

Vimos que a estrutura de dados adotada para a representação de subdivisões tem a aresta orientada como único elemento explicitamente representado. Assim, qualquer acesso a estrutura de dados nos retomará uma aresta da álgebra. As *funções de arestas* surgem como meios de acesso aos elementos da álgebra vizinhos a uma dada aresta orientada. Elas podem ser interpretadas como relações de adjacências pois, dada uma aresta do modelo, nos retomam uma segunda aresta associada a primeira por alguma relação de adjacência. Estamos aqui considerando que duas arestas a e b são adjacentes se estiverem dispostas sucessivamente em um ciclo de vértice ou de face. Em notação matemática, as funções de arestas se apresentam na forma abaixo, onde **A** é o conjunto de todas as arestas da álgebra:

$$f: A \rightarrow A$$
$$a \in A \mapsto b \in A$$

Na figura 4.6 temos ilustradas as 9 funções de arestas suficientes para que todas as relações de adjacência de uma dada aresta da álgebra sejam computadas (no caso, da aresta a). A notação e o significado de cada uma destas funções é visto a seguir:

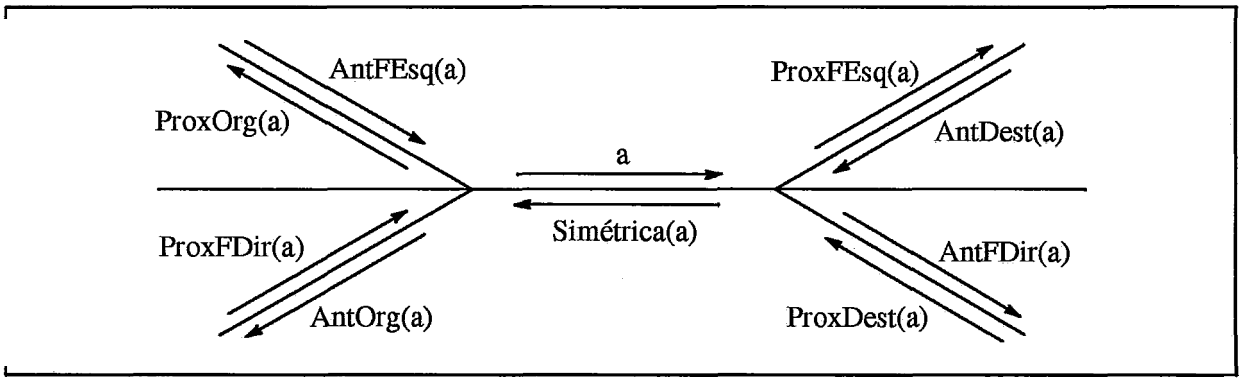


Figura 4.6 – As funções de arestas

Simétrica(a): Devolve a aresta com mesma direção de a e sentido contrário.

AntFEsq(a): Devolve a aresta cuja face esquerda é a mesma de a e cujo vértice destino equivale ao vértice origem de ***a.***

ProxFEsq(a): Devolve a aresta de mesma face esquerda de a e cujo vértice origem equivale ao vértice destino de a.

AntFDir(a): Devolve a aresta com mesma face direita de a e cujo vértice origem equivale ao vértice destino de a.

ProxFDir(a) Devolve a aresta com mesma face direita de a e cujo vértice destino equivale ao vértice origem de ***a.***

AntOrg(a): Devolve a aresta cujo vértice origem é o mesmo de *a* e cuja face esquerda é a face direita de *a*.

ProxOrg(a): Devolve a aresta com mesma origem de a e cuja face direita coincide com a face esquerda de *a*.

AntDest(a): Devolve a aresta cujo vértice destino coincide com o de a e cuja face direita equivale a face esquerda de *a*.

ProxDest(a): Devolve a aresta cujo vértice destino coincide com o de a e cuja face esquerda equivale a face direita de *a*.

As funções acima definidas podem ser expressas a partir de uma base composta por apenas duas delas. Em outras palavras, qualquer aresta adjacente pode ser acessada através de uma sequência finita de aplicações destas duas funções. Listamos a seguir as relações entre ProxFEsq e *ProxOrg* e cada uma das 7 funções restantes:

Simetrica(a) = ProxOrg(ProxFEsq(a))
AntFEsq(a) = ProxOrg(ProxFEsq(ProxOrg(a)))
AntFDir(a) = ProxOrg(ProxOrg(ProxFEsq(a)))
ProxFDir(a) = ProxOrg(ProxFEsq(ProxFEsq(ProxOrg(ProxFEsq(a)))))
AntOrg(a) = ProxFEsq(ProxOrg(ProxFEsq(a)))
AntDest(a) = ProxOrg(ProxFEsq(ProxFEsq(a)))
ProxDest(a) = ProxOrg(ProxFEsq(ProxOrg(ProxOrg(ProxFEsq(a)))))

Observe ainda, que a representação de cada função através da base sugerida independe da forma com que as arestas do modelo se encontram relacionadas. Isto significa que cada função de aresta pode ser expressa por uma sequência, de comprimento fixo, de aplicações sucessivas dos elementos da base. Uma vez que estas funções são utilizadas para consulta a estrutura de dados, objetivando o acesso a arestas adjacentes, pode-se imaginar que sua aplicação sucessiva, regida por estratégias bem definidas, resultaria em procedimentos de percurso nos quais toda a estrutura seria visitada. Esta suposição será por nós comprovada no momento adequado.

4.4.2. A Álgebra de Arestas

Uma álgebra de arestas será por nós definida como sendo uma álgebra abstrata (A, *ProxOrg*, ProxFEsq), onde A é um conjunto finito de arestas e ProxOrg e *ProxFEsq* são funções de A satisfazendo as propriedades abaixo, onde *Simétrica(x)* = *ProxFEsq(ProxOrg(x))*:

(P1) Para todo $a \in A$, Simétrica(*a*) ≠ a

(P2) Para todo $a \in A$, Simétrica(Simétrica(*a*)) = *a*

Quando satisfeitas estas condições, Persiano [PERS91] mostra que tanto ProxFEsq quanto ProxOrg são inversíveis, definindo com isto, ciclos em A. A função ProxFEsq, por convenção, cicla faces e a ProxOrg gira em tomo de vértices.

A álgebra abstrata assim obtida equivale a álgebra de arestas especincada por Guibas e Stolfi [GUIB85]. Portanto, todos os resultados por eles demonstrados podem ser aqui aplicados. Sabe-se, por exemplo, que qualquer subdivisão que atenda as propriedades *S1-S4*, enumeradas na seção 4.2, é representável por meio de alguma álgebra de arestas com as características acima. Uma segunda propriedade extraída vem a ser a realizabilidade da álgebra. Ou seja, qualquer álgebra de arestas arbitrária obtida segundo as normas acima enumeradas, corresponderá a uma certa subdivisão de alguma superfície sem bordo.

Com o objetivo de trabalharmos com 2-variedades com bordo, adicionaremos uma nova função denominada Bordo que, dada uma aresta a qualquer da álgebra, nos diz se apertence ou não a composição de algum bordo. Vimos anteriormente que os bordos se comportam como 1-variedades. Assim sendo, a função Bordo deve incorporar certas propriedades para que a representação obtida seja consistente. Tais propriedades são abaixo apresentadas, onde a e b são arestas e v é um vértice de uma álgebra **A:**

(B1) Bordo(*a*) E {Falso, Verdadeiro}

(B2) Bordo(*a*) = Bordo(ProxFEsq(*a*))

(B3) Se Bordo(*a*) = Verdadeiro, então Bordo(Simétrica(*a*)) = Falso

(B4) Se *3* a e b incidentes a v, tais que Bordo(*a*) = Bordo(*b*) = Verdadeiro, então, ou ProxFEsq(*a*) = b, ou ProxFEsq(*b*) = a

A propriedade (B2) faz com que as arestas pertencentes a um dado bordo formem um ciclo fechado. Em *(B3)* evitamos bordos que compartilhem de uma mesma aresta, ao passo que *(B4)* impede a existência de bordos distintos incidentes a um mesmo vértice e ainda, que um vértice ocorra mais de uma vez no ciclo de um bordo.

Vamos agora, concluir que qualquer álgebra de arestas A, sujeita a estas condições, atende as propriedades de representabilidade e realizabilidade. Analisando inicialmente a realizabilidade, verifica-se que a partir de A, pode-se chegar a uma álgebra B, equivalente a álgebra de Guibas. Para tanto, basta que todas as arestas de A, cuja função Bordo retorne Verdadeiro, tenham o resultado desta função alterado para Falso. Guibas a f i a que a álgebra B é realizável. Isto é, existe uma subdivisão S de alguma 2-variedade sem bordo associada a álgebra B. Os ciclos de arestas que na álgebra A tiveram o resultado da função Bordo alterado de Verdadeiro para Falso, estão em *S* associados as faces da subdivisão. Estas faces, por sua vez, correspondem a superfícies de alguma 2-variedade. O que nos resta fazer é recortar estas faces de modo que os ciclos de arestas que as representavam combinatoriamente, passem então a representar um bordo. Desta forma, como queríamos demonstrar, obtém-se uma nova subdivisão S* de alguma 2-variedade com bordo.

A representabilidade da álgebra sugerida corresponde a certeza de que toda subdivisão de uma 2-variedade com bordo pode ser representada via uma álgebra que atenda as propriedades por nós especificadas. A demonstração fornecida por Guibas e Stofi não apresenta qualquer restrição a existência de bordos. Portanto, a prova que desejamos pode ser desenvolvida de modo análogo.

4.4.3. Os Operadores Topológicos Básicos

Uma das principais vantagens oferecidas pela álgebra de arestas descrita é o fato de qualquer álgebra poder ser criada ou alterada com o uso de apenas 3 operadores topológicos básicos ou operadores de construção. Ao longo desta seção, iremos apresentá-los e mostrar que eles são responsáveis pela criação, modificação e eliminação de uma álgebra de arestas que, a todo momento, preserva as propriedades *P1* e *P2* enunciadas na seção anterior. Isto implica na afirmação de serem os operadores de construção capazes de gerar representações válidas para qualquer subdivisão da superfície de uma 2-variedade.

O primeiro destes operadores, o *CriaAresta*, adiciona a álgebra duas novas arestas orientadas a e b possuindo as seguintes características:

(C1) ProxFEsq(*a*)= b & ProxFEsq(*b*)= *a*

(C2) ProxOrg(*a*) = a & ProxOrg(*b*) = b

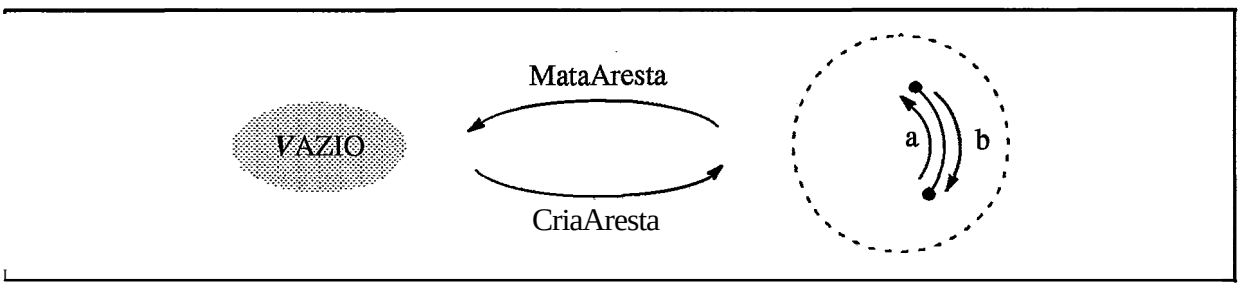


Figura 4.7 - O operador CriaAresta e seu inverso

Topologicamente falando, parte-se da álgebra existente e se obtém como resultado uma particular subdivisão de uma superfície esférica. Este processo se encontra ilustrado na figura 4.7, onde o operador CriaAresta é aplicado sobre uma álgebra vazia. É trivial a constatação de serem válidas as propriedades *P1* e *P2* sobre estes novos elementos acrescentados a álgebra.

O segundo operador, o *MataAresta*, nada mais é senão o inverso do primeiro. Sua finalidade vem a ser a eliminação de um par de arestas da álgebra que atenda as características *C1*-C2 mencionadas.

Enquanto os dois primeiros operadores criam e eliminam elementos da álgebra de arestas, o terceiro e último deles, o *AlteraÁlgebra*, tem por objetivo efetuar modificações topológicas sobre uma dada álgebra. Para tanto, o *AlteraÁlgebra* recebe duas arestas a e b da álgebra e modifica suas relações de adjacências segundo as normas listadas a seguir:

- Permuta os resultados de ProxFEsq(*c*) e ProxFEsq(*d*), onde c = AntFEsq(*a*) e *d* = AntFEsq(*b*)

- Permuta os resultados de ProxOrg(*a*) e ProxOrg(*b*)

A figura 4.8 exibe o esquema da aplicação do operador AlteraÁlgebra. Nele temos parcialmente representada uma álgebra de arestas arbitrária. Os pontos **a,** b, ... , f correspondem a arestas desta álgebra e as setas indicam as relações de adjacência ProxFEsq e ProxOrg. Observe que a aplicação consecutiva do operador AlteraÁlgebra sobre o mesmo par de arestas nos leva de volta a configuração inicial. Conclui-se, portanto, ser o operador AlteraÁlgebra seu próprio inverso.

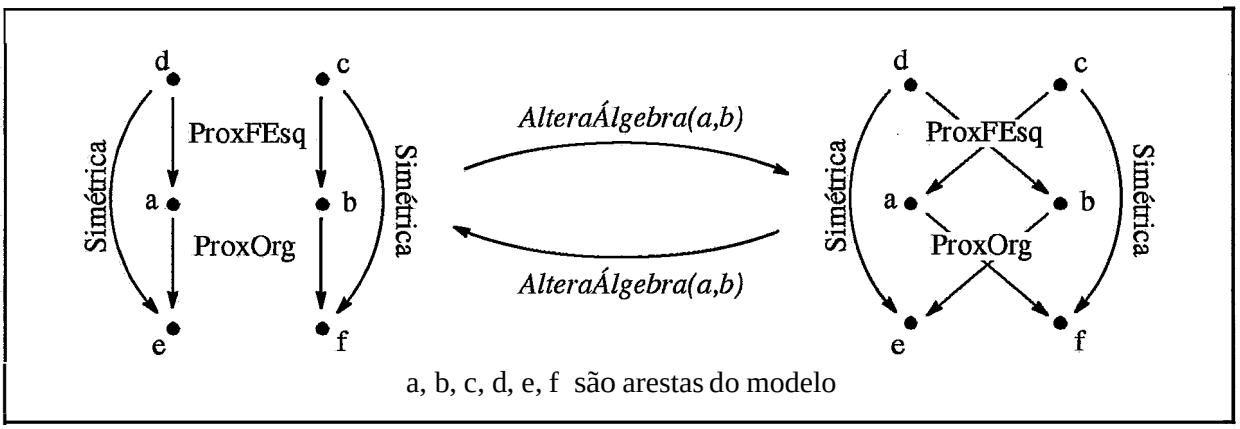


Figura 4.8 – A inversibilidade do operador AlteraÁlgebra

Queremos agora mostrar que o AlteraÁlgebra, quando aplicado sobre uma álgebra de arestas arbitrária A, que satisfaça as propriedades *P1* e P2 citadas na seção 4.4.2, mantém válidas estas propriedades na estrutura resultante. Como veremos adiante, basta verificar que ele não altera o resultado da relação de adjacência Simétrica(*x*), para todo x pertencente a A. Sendo válida esta propriedade e sabendo que na álgebra inicial era constatado o fato de Simétrica(*x*) diferir de x, conclui-se que na álgebra modificada o mesmo resultado será válido. Assim, a propriedade *P1* mantém-se intacta. Aplicando em cadeia a preservação da relação de adjacência Simétrica a expressão Simétrica(Simétrica(*x*)), verifica-se a validade da propriedade P2 sobre a álgebra de arestas resultante. Deste modo, basta provar que a expressão Simétrica(*x*) = ProxOrg(ProxFEsq(*x*)) retorna a mesma aresta antes e depois da aplicação do AlteraÁlgebra.

Nossa demostração se referirá a ilustração encontrada na figura 4.8. Para cada x pertencente a A temos:

- Se x = *c* ⇒ Simétrica(*c*) = e, seja antes ou após a aplicação do operador
- Se x = *d* ⇒ Simétrica(*x*) = *f*, tanto antes quanto após a operação
- Se x ⊄ {*c*,*d*} ⇒ ProxFEsq(*x*) = y não se modifica, pois apenas c e d têm seus ProxFEsq alterados. Pode-se ainda observar, pela inversibilidade da função ProxFEsq, que y ⊄ {*a,* b}, donde se conclui que ProxOrg(*y*) não será modificado, pois somente *a e b* têm estas informações trocadas. Logo, Simétrica(*x*) = ProxOrg(*y*) manterá seu resultado após a aplicação do AlteraÁlgebra.

Um fato importante a ser considerado é que as alterações provocadas pelo operador AlteraÁlgebra variam conforme a conectividade entre os ciclos de arestas que definem os vértices origens e as faces esquerdas das arestas envolvidas, a saber:

- Se os ciclos forem distintos, estes serão conectados em um único ciclo

Se os ciclos forem idênticos, estes serão divididos em dois outros ciclos

Por exemplo, se temos ciclos distintos tanto para vértices origens como para faces esquerdas, estes serão conectados resultando na eliminação de um vértice e de uma face. Como a equação de Euler deve ser satisfeita, temos para este caso particular:

$$(v-1) - a -(f-1) = 2\ (\ (c+n) - (g+m)\ )$$
$$v - a + f = 2\ (c-g) + 2 + 2n + 2m$$
$$1 + n + m = 0$$

Ora, se as arestas envolvidas pertencerem a componentes conexas distintas, ao conectarmos seus ciclos origens e faces esquerdas, juntamos igualmente suas componentes. Assim sendo, $n=-1$, o que implica em termos $m=0$. Entretanto, no caso das componentes serem idênticas, elas assim permanecerão, o que nos dá n=O e, consequentemente, $m=1$. Em outras palavras, quando aplicamos o AlteraÁlgebra sobre arestas de origens e faces esquerdas distintas, o resultado pode ser a criação de um genus ou a eliminação de uma componente conexa, dependendo delas pertencerem ou não a uma mesma componente. Isto serve como exemplo de que o operador AlteraÁlgebra pode causar alterações distintas sobre a álgebra, dependendo de certas relações de conectividade e adjacência entre as arestas parâmetros.

Ciclos Origens / Ciclos Face Esq.	Idênticos	Distintos
Idênticos	+1 vértice +1 face –1 genus ou +1 comp. conexa	–1 vértice +1 face
Distintos	+1 vértice –1 face	–1 vértice –1 face +1 genus ou –1 comp. conexa

Tabela 4.1 – Alterações topológicas causadas pelo operador AlteraÁlgebra

A Tabela 4.1 sintetiza as alterações topológicas decorrentes da aplicação do AlteraÁlgebra segundo a conectividade dos ciclos de vértices origens e faces esquerdas das arestas parâmetros. Os casos listados nesta tabela encontram-se devidamente exemplificados na figura 4.9, onde vemos novamente constatada a inversibilidade do operador AlteraÁlgebra.

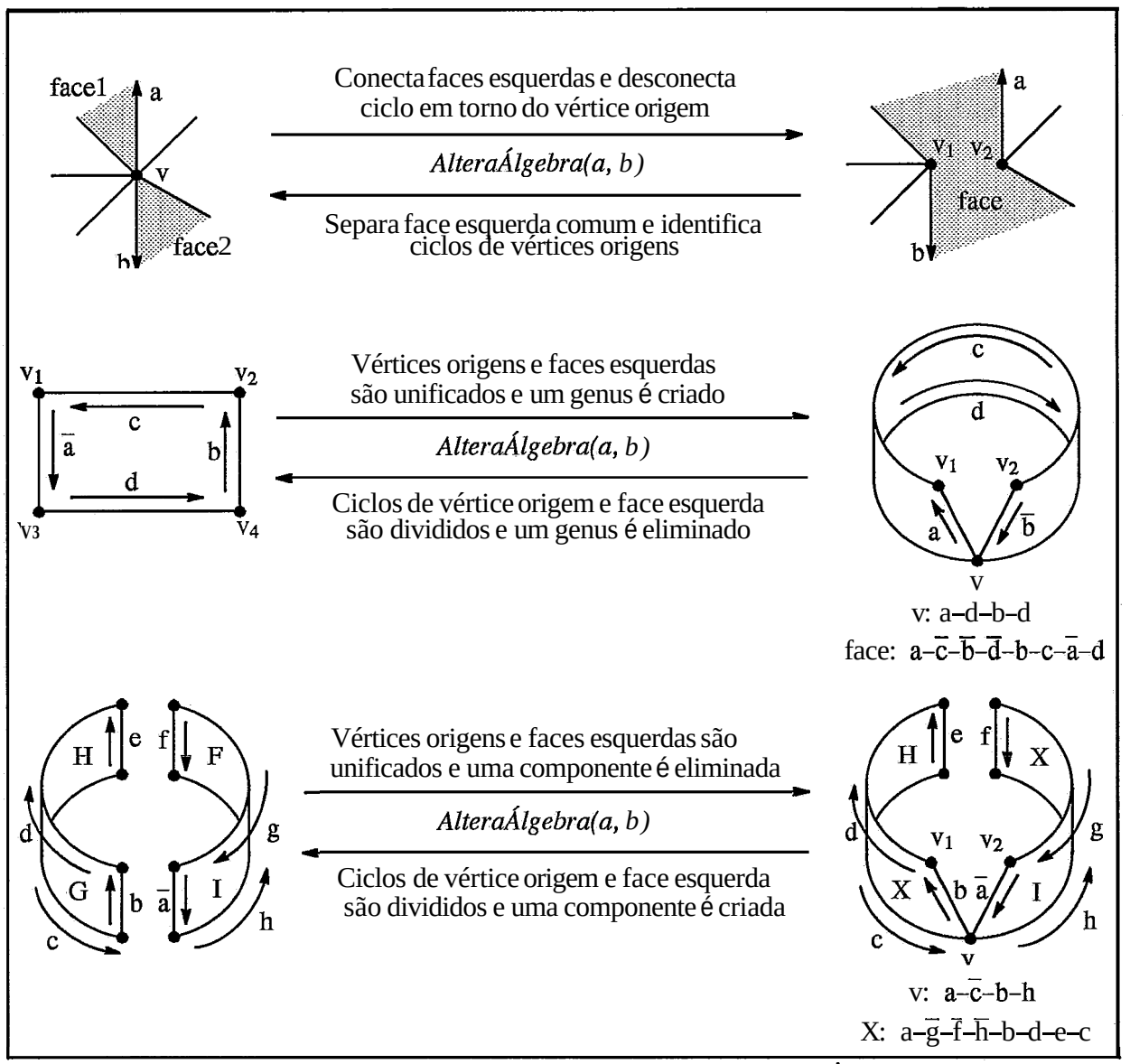


Figura 4.9 – Alterações topológicas do operador AlteraÁlgebra

4.4.4. A Implementação da Álgebra

Passemos afinal a descrição da forma com que foi implementada a estrutura de dados descrita anteriormente. O cerne desta questão vem a ser a representação da entidade básica denominada aresta. Como visto anteriormente, e suficiente que cada aresta referencie o resultado das funções de aresta *ProxFEsq* e *ProxOrg* a ela aplicada. Mais adiante serão acrescentados outros campos no intuito de capturar as informações não topológicas de uma dada subdivisão. A figura 4.10 exemplifica a implementação sugerida.

Com esta forma de representação, as funções de arestas são implementadas como procedimentos que recebem como parâmetro uma aresta e retornam uma segunda como resultado, consistindo em uma sequência de acessos aos campos *ProxFEsq* e *ProxOrg* das arestas. O número de campos acessados é fixo para cada uma das funções existentes, atingindo seu limite máximo de 5 acessos nas funções *ProxFDir* e *ProxDest*.

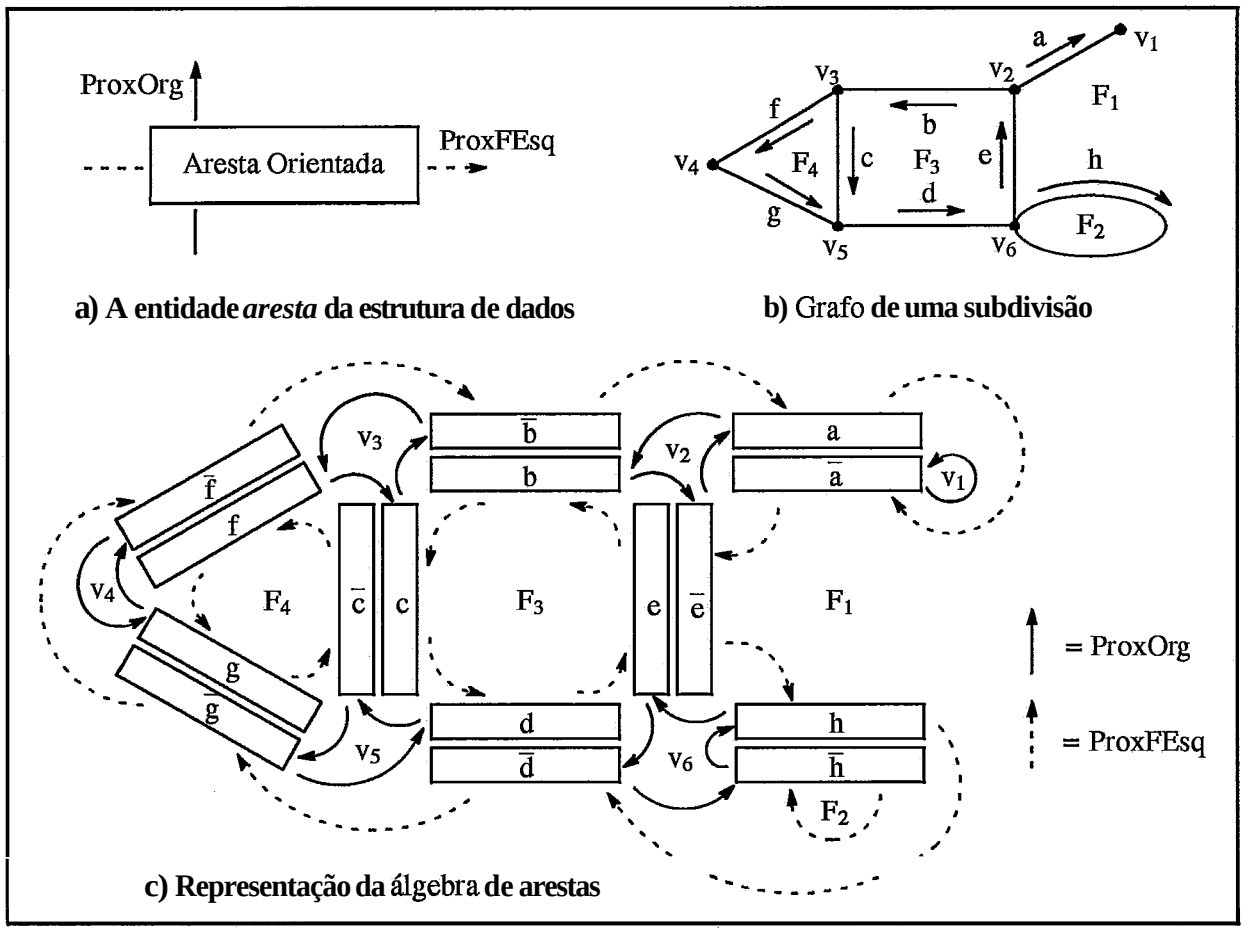


Figura 4.10 – A representação da álgebra de arestas

Na implementação do primeiro dos operadores de construção, o *CriaAresta*, duas arestas são criadas e seus campos preenchidos de forma a satisfazer as características *C1-C2* expostas na seção 4.4.3. Como resultado ele devolve uma das arestas criadas. Seu inverso, o *MataAresta*, recebe uma aresta que deve atender às mesmas propriedades *C1-C2* citadas acima e a elimina junto com sua simétrica. Caso as propriedades mencionadas não forem verificadas, nenhuma alteração é efetuada sobre o modelo. O AlteraÁlgebra, por sua vez, corresponde a um procedimento que recebe duas arestas a e b e troca entre si o conteúdo de dois pares de campos da estrutura, expostos no pseudo-código a seguir:

```
AlteraÁlgebra(a, b) {
    Troca(ProxFEsq(AntFEsq(a)), ProxFEsq(AntFEsq(b)))
    Troca(ProxOrg(a), ProxOrg(b))
}
```

Pela simplicidade constatada na implementação dos operadores, concluímos termos em mãos uma estrutura capaz de lidar eficiente e objetivamente com representações de subdivisões. O apêndice A exibe integralmente a implementação da álgebra de arestas proposta.

4.4.5. Informações não Topológicas

Para que a estrutura de dados apresentada possa ser útil na representação da geometria de uma dada subdivisão de uma superfície e, consequentemente, possa ser aplicada na modelagem de sólidos BRep, alguns campos são adicionados a entidade aresta. Estes campos serão acrescentados de tal modo que as informações a eles relacionadas se encontrem a parte da estrutura de dados. Assim sendo, uma aresta não conterá explicitamente as informações não topológicas por ela representadas, mas sim uma referência para tais dados. Com esta filosofia evitamos a repetição destas informações ao longo do modelo. Por exemplo, para um vértice definido por uma sequência de arestas, cada uma destas deveria conter uma cópia dos dados geométricos que o posicionam no espaço. Ao invés disto, elas referenciam uma única área de trabalho na qual são encontradas estas informações.

Outra vantagem advinda desta particular implementação é a flexibilidade fornecida pelo modelo no que conceme ao teor das informações não topológicas a si relacionadas, podendo estas apresentar variações de forma a atender as necessidades da aplicação em vigor. Em um determinado momento, por exemplo, uma face pode referenciar a sua normal, enquanto que, em uma dada aplicação de visualização, o mesmo campo da estrutura pode indicar uma área onde tenhamos dados como o ângulo que esta face faz com um determinado *spot* de luz, bem como seus coeficientes de reflexão, refração e absorção. Para cumprir esta finalidade, uma aresta recebe três campos de dados que correspondem às informações associadas aos pontos, curvas e discos abertos da superfície sendo representada pela álgebra de arestas. Estes são acessados respectivamente, pelas funções *InfoVertOrg, InfoAresta e InfoFaceEsq.*

Além dos campos já citados, em geral associados a geometria do modelo, temos três outros campos presentes em uma aresta. Dois deles equivalem a índices dados a arestas e vértices do modelo. Estes índices têm papel fundamental no processo de gravação e leitura de representações BRep e serão discutidos em momento oportuno, bastando por hora tomarmos conhecimento de sua existência. O último deles, um marcador de arestas é utilizado pelos procedimentos de percurso a serem expostos na seção vindoura.

4.4.6. Percursos

Os percursos são meios de acesso a estrutura de dados, utilizados por aplicações nas quais toda a superfície do modelo deva ser visitada. Como na representação de superfícies as entidades básicas são faces, arestas e vértices, é util dispormos de procedimentos de passeio pelo modelo que se baseiem nestes elementos. Assim sendo, adotamos três tipos de percurso neste trabalho. Estes se utilizam de um campo pertencente a entidade aresta da estrutura de dados, responsável por indicar o status da aresta, isto é, se ela se encontra ou não marcada. O status inicial de toda aresta é desmarcada e, após a execução de qualquer um dos procedimentos de percurso é assim que as encontraremos, de modo que, somente nos deparamos com arestas marcadas durante os procedimentos de percurso.

Os percursos por faces ou vértices são, por dualidade, idênticos, a despeito da função de acesso a próxima aresta em tomo do elemento sendo visitado, a saber,

ProxFEsq no caso de percurso por faces e ProxOrg caso estejam sendo visitados os vértices do modelo. O procedimento, de natureza recursiva, recebe por parâmetros, além da função de acesso ao próximo elemento, uma aresta inicial e um procedimento de tratamento, sendo este último executado uma única vez para cada elemento visitado.

O primeiro passo do algoritmo é marcar as arestas pertencentes ao ciclo de face (ou vértice) do qual faz parte a aresta parâmetro. A seguir, a função de tratamento é aplicada sobre esta face (ou vértice) e então, cada aresta deste ciclo é testada. Se ela e sua simétrica se encontram manadas, ambas são desmarcadas, caso apenas a simétrica esteja desmarcada, o procedimento é recursivamente aplicado sobre a simétrica, sendo assim visitadas as faces (ou vértices) adjacentes a face visitada. Neste ponto, o fato da aresta testada se encontrar desmarcada, significa que ela e sua simétrica já foram percorridas e devidamente desmarcadas, não devendo portanto, ser novamente processadas.

Exibimos abaixo o procedimento de percurso sobre faces e vértices do modelo. Atente para o fato de que apenas uma componente conexa é percorrida cada vez que um percurso é executado, o que torna necessário o reconhecimento de todas as componentes de um dado sólido para que este possa ter sua fronteira inteiramente visitada.

```
Percorre(ArestaInicial, Trata, Próxima) {
    ("Marca as arestas do ciclo de faces ou vértices *)
    Aresta = ArestaInicial
    faça {
        Marca(Aresta)
        Aresta = Próxima(Aresta)
    } enquanto Aresta ≠ ArestaInicial
    (* Trafa a enfidade visitada *)
    Trata(ArestaInicial)
    (* Percorre os ciclos adjacentes não marcados *)
    Aresta = ArestaInicial
    faça {
        se EstáMarcada(Aresta) então faça {
            se EstáMarcada(Simétrica(Aresta)) então faça {
                Desmarca(Aresta)
                Desmarca(Simétrica(Aresta))
            }
            senão então faça {
                Percorre(Simétrica(Aresta), Trata, Próxima)
            }
        }
        Aresta = Próxima(Aresta)
    } enquanto Aresta ≠ ArestaInicial
}
```

O percurso por arestas é análogo ao apresentado, com a diferença de que o procedimento de tratamento passa a ser executado para cada aresta ao invés de para cada face (ou vértice). Uma aresta visitada é tratada somente na condição de sua

simétrica não estar marcada. O pseudo-código abaixo mostra a implementação do percurso por arestas para uma dada componente conexa do modelo:

```
PercorreArestas(ArestaInicial, Trata) {
    (* Marca o ciclo de face e trata suas arestas *)
    Aresta = ArestaInicial
    faça {
        Marca(Aresta)
        se não EstáMarcada(Simétrica(Aresta)) então Trata(Aresta)
        Aresta = Próxima(Aresta)
    } enquanto Aresta ≠ ArestaInicial
    (* Percorre as faces adjacentes ainda não visitadas *)
    Aresta = ArestaInicial
    faça {
        se EstáMarcada(Aresta) então faça {
            se EstáMarcada(Simétrica(Aresta)) então faça {
                Desmarca(Aresta)
                Desmarca(Simétrica(Aresta))
            }
            senão então faça {
                PercorreArestas(Simétrica(Aresta), Trata)
            }
        }
        Aresta = Próxima(Aresta)
    } enquanto Aresta for diferente de ArestaInicial
}
```

O restante desta seção será dedicado a demonstração de que o pmcedimento acima visita todas a s arestas de uma dada componente conexa C e, que cada aresta de C é tratada uma única vez. Seja a_1 a aresta inicial visitada. Pelas propriedades do modelo, sabemos que através de uma sequência de *ProxFEsq* e *ProxOrg* pode-se caminhar de a_1 para qualquer outra aresta de C. Observe pelo pseudo-código exposto, que o ciclo de face de uma aresta visitada é todo percorrido e tratado. Daí concluímos que a aplicação da função *ProxFEsq* sobre uma aresta tratada irá sempre retornar uma segunda aresta também tratada pelo procedimento de percurso. Sabemos ainda, que a função *ProxOrg(x)* pode ser computada a partir da expressão *Simétrica(AntFEsq(x))*. Como a função *AntFEsq* pode ser substituída por uma série de aplicações da função *ProxFEsq*, deduzimos que ela retorna, igualmente, uma aresta tratada. Ora, toda aresta tratada teve o algoritmo reiniciado para o ciclo de face de sua simétrica, de modo que todo este ciclo foi tratado do mesmo modo. Com isto, conclui-se que qualquer que seja a sequência de *ProxFEsq* e *ProxOrg* aplicada sobre a aresta inicial a_1, teremos acessado uma aresta que tenha sido tratada pelo procedimento de percurso. Portanto, como queríamos demonstrar, todas a s arestas de C são corretamente tratadas.

Resta então, provarmos que cada aresta de C é tratada uma única vez. Suponha por absurdo, que uma dada aresta a_2 de C esteja sendo visitada uma segunda vez e que esta seja a primeira aresta de C a ser duplamente acessada. Isto só ocorreria se a simétrica de a_2, a qual denominaremos s_2, se encontrasse marcada e a_2 desmarcada, caso em que o procedimento seria recursivamente aplicado sobre a_2. Além disto, s_2

estaria sendo visitada pela primeira vez, pois partimos da premissa que a_2 é a primeira aresta a ser novamente visitada. Entretanto, se a_2 já foi visitada (e consequentemente marcada) e se encontra desmamada, ao contrário de s_2, em algum momento do procedimento, ocorreu de a_2 e s_2 estarem marcadas, o que causou a desmamação de ambas. Ora, mas s_2 está agora marcada, o que significa ter sido ela novamente visitada, contrariando assim a nossa premissa. Com isto, concluimos que nenhuma aresta de C é marcada mais de uma vez, completando assim a nossa demonstração.

Observe ainda, que após a execução do percurso, toda aresta da componente visitada possui status desmarcada. Para verificarmos esta afirmação vamos partir da suposição de que uma dada aresta a_1 tenha, ao final do percurso, o status *marcada*. Ora, pela fase inicial do algoritmo de percurso por arestas vemos que se a_1 está marcada e sua simétrica s_1 também, ela não será tratada (pelo fato de s_1 já o ter sido) e o algoritmo seguirá para sua segunda etapa, onde certamente ambas serão desmarcadas. Na hipótese entretanto, de a_1 estar marcada e s_1 desmarcada, esta será tratada e o algoritmo chamado recursivamente, sendo passada s_1 como parâmetro e então, nesta segunda iteração, s_1 será marcada e sua simétrica $a_.$ já estará mamada, o que implica que a segunda etapa do algoritmo será responsável pela desmarcação de a_1 e s_1. Donde se conclui que a_1, uma vez tendo sido marcada, certamente será demarcada até o fim do procedimento de percurso.

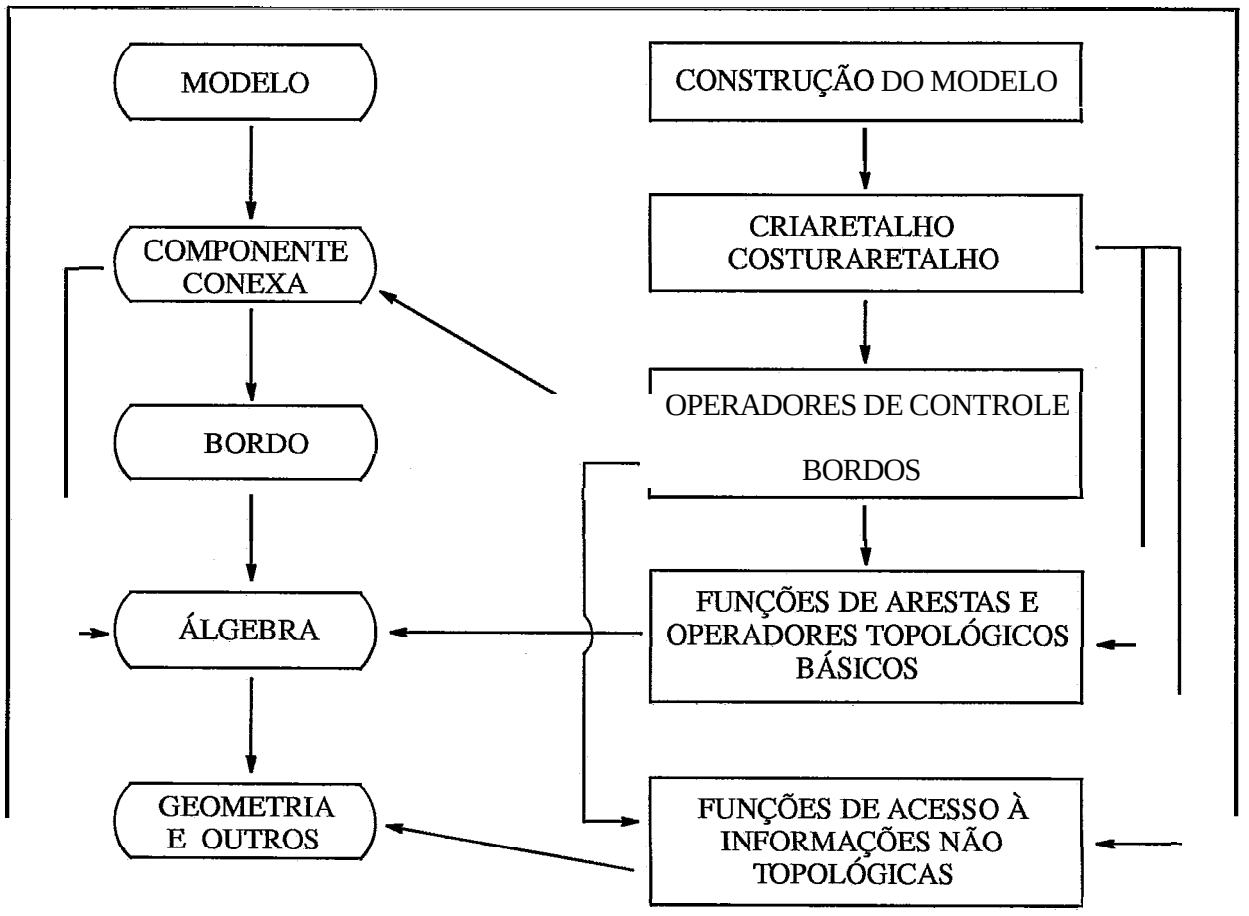


Figura 4.11 – O modelo funcional de construção de um sólido BRep

Estes mesmos resultados acima podem ser obtidos para os percursos por faces e vértices de uma componente conexa, mostrando com isto a correção dos procedimentos de percurso propostos.

4.5. O MODELO FUNCIONAL DE CONSTRUÇÃO DE SÓLIDOS BREP

O modelo funcional de construção de um sólido BRep pode ser subdividido em dois níveis, como mostra o esquema da figura 4.11. Em seu primeiro nível, encontramos os operadores *CriaRetalho* e *CosturaRetalho*. Todo o acesso à estrutura de dados necessário a execução destes operadores são efetuados indiretamente por um conjunto de operadores de mais baixo nível separados em 3 classes, uma para cada tipo de informação acessada. Assim sendo, um destes grupos é responsável pelos acessos as informações geométricas, outro efetua alterações sobre as entidades que administram as componentes conexas e seus respectivos bordos e o último conjunto, constituído das funções de arestas e dos operadores topológicos básicos, é o único a ter acesso direto a álgebra de arestas.

Esta seção tem por finalidade esclarecer as relações existentes no modelo citado a partir da descrição pormenorizada dos operadores de alto nível CriaRetalho e CosturaRetalho e de seus efeitos sobre uma álgebra arbitrária.

4.5.1. O Operador CriaRetalho

Sabemos que um retalho é uma 2–variedade com bordo composta de uma única face e um bordo, sendo ambos combinatoriamente representados através de um ciclo fechado de arestas orientadas. A diferença está no fato do ciclo que identifica a face possuir a si associado a geometria desta. Diante disto, a rotina CriaRetalho é implementada de modo a receber como parâmetros seu número de arestas (que é igual ao de vértices), uma lista com referência aos dados que descrevem os pontos da variedade associados aos vértices da subdivisão, outra com os dados das curvas associadas as arestas e, por fim, a informação que descreve a superfície homeomorfa ao disco aberto associado a face do retalho.

A construção do retalho, como vista no pseudo–código adiante, parte da criação da aresta inicial ligando seus dois primeiros vértices, para posteriormente gerar e conectar as demais arestas até que o ciclo da face seja completo. Neste momento as informações não topológicas são adicionadas e uma nova componente conexa criada, a qual contém um único bordo associado.

A criação de uma componente é feita pela rotina *CriaComponenteConexa*, que recebe por parâmetros seu número de vértices, arestas e faces (para que o modelo satisfaça a equação de Euler, bordos são computados como faces). Todas as arestas que compõem o ciclo do bordo gerado passam a ter suas informações de faces indicando o bordo ao qual pertencem. Este bordo, por sua vez, irá referenciar uma das arestas de seu ciclo. Estas associações entre arestas e bordos é efetivada pelo próprio procedimento resonsável pela criação do bordo, a saber, o *CriaBordo*, que recebe por parâmetro uma aresta pertencente ao ciclo que o identifica e retorna a entidade bordo gerada.

```
CriaRetalho(NumArestas, ListaInfoVert, ListaInfoAresta, InfoFace) {
    (* Cria-se aresta que une o primeiro ao último vértice *)
    ArestaInicial = CriaAresta()
    A1 = ProxFEsq(ArestaInicial)
    (* Geração e conexão das demais aresfas *)
    faça NumArestas-1 vezes {
        A2 = CriaAresta()
        AlteraÁlgebra(A1, A2)
        A1= AntFEsq(A2)
    }
    ("A face é fechada com a junção da primeira com a última aresta")
    AlteraÁlgebra(ArestaInicial, A1)
    (* Preenchimento dos campos de informações não topológicas *)
    A1 = ArestaInicial
    para Ind = 0 até Ind = NumArestas-1 faça {
        InfoVertOrg(AI ) = ListaInfoVert[Ind]
        InfoAresta(A1) = ListaInfoAresta[Ind]
        InfoFaceEsq(A1) = InfoFace
    }
    (* Cria componente conexa com um único bordo *)
    Componente = CriaComponenteConexa(NumArestas, NumArestas, 2)
    Bordo = CriaBordo(Simétrica(ArestaInicial))
    ListaDeBordos(Componente) = Bordo
    (* Retorna aresta ligando o primeiro ao segundo vértice *)
    retorne(ArestaInicial)
}
```

Com as interrelações existentes entre os elementos do modelo podemos, a partir de uma aresta pertencente a um bordo, visitar a entidade bordo que o representa ou a componente conexa que o contém. O caminho inverso é igualmente realizável, o que torna possível a identificação de uma aresta pertencente a uma dada componente conexa ou a um bordo.

4.5.2. O Operador CosturaRetalho

O operador CosturaRetalho foi idealizado de forma a permitir que os retalhos criados isoladamente pudessem ser costurados um a um, objetivando a construção da representação conexa da superfície de um sólido. Esta costura, como vimos anteriormente, será sempre relizada sobre arestas comuns aos retalhos, ou seja, pares de arestas que correspondam a uma única curva da variedade representada.

O processo intuitivo que levou a implementação do operador CosturaRetalho pode ser descrito como a seguir. Partindo dos dois retalhos isolados ilustrados na figura 4.12 e identificados pelas arestas a e b, o primeiro passo é unificar o ciclo de arestas em tomo dos vértices equivalentes, a saber, os pares de vértices dados respectivamente pela origem de b (v_4) e destino de a (v_2) e pelo destino de b (v_3) e origem de a (v_1). Note que, ao conectarmos o primeiro par de ciclos, aplicamos o operador AlteraÁlgebra sobre duas arestas cujos vértices origens pertenciam a ciclos distintos. O mesmo acontece com suas faces (uma face e um bordo). Com isto, tanto os ciclos dos vértices como os das

faces foram unificados. Ao aplicarmos novamente o AlteraÁlgebra, agora sobre um segundo par de arestas, seus vértices origens se encontram em ciclos distintos, ao passo que suas faces são idênticas (e corresponde exatamente a face unificada anteriormente). Portanto, os vértices são conectados e a face duplicada, retomando sua configuração anterior (uma face e um bordo).

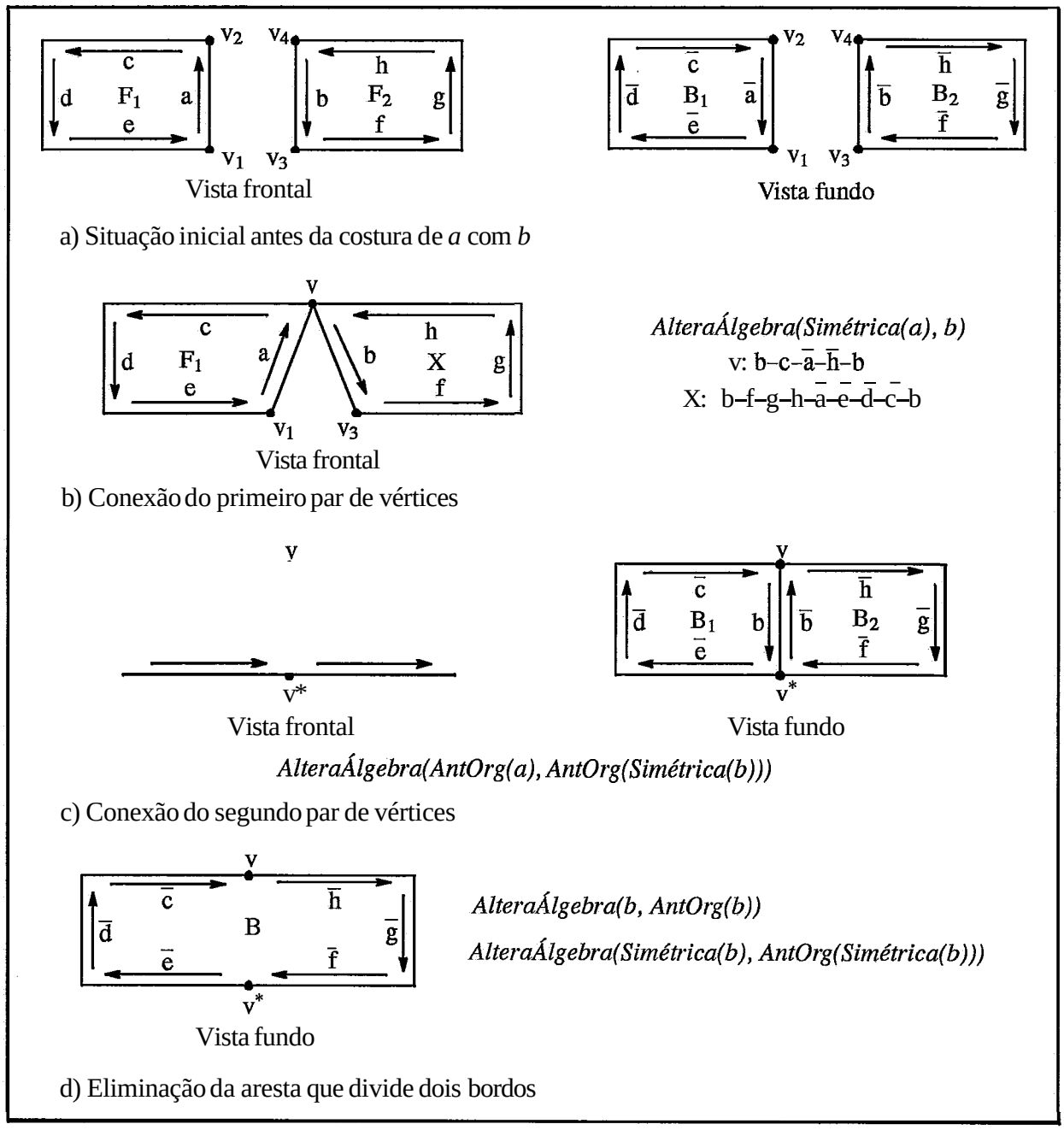


Figura 4.12 – O algoritmo de costura de retalhos

Observe que, no decorrer do procedimento descrito, as arestas Simétrica(*a*) e b trocaram de posições em seus ciclos de face. Embora isto não afete a topologia do modelo, um cuidado adicional deve ser tomado se pretendemos garantir a consistência geométrica deste. Para tal, as informações não topológicas associadas a estas arestas devem também ser permutadas. Além disto, ao fim da costura, a álgebra resultante possui dois bordos adjacentes por uma aresta comum, composta pelas arestas b e

Simétrica(*b*) e, como discutido anteriormente, nossa definição de bordo não permite a existência de bordos adjacentes. Logo, a última etapa do processo de costura consiste na eliminação desta aresta problemática. Isto é feito através de uma dupla aplicação do operador AlteraÁlgebra desconectando-a do restante do modelo e, uma aplicação final do MataAresta.

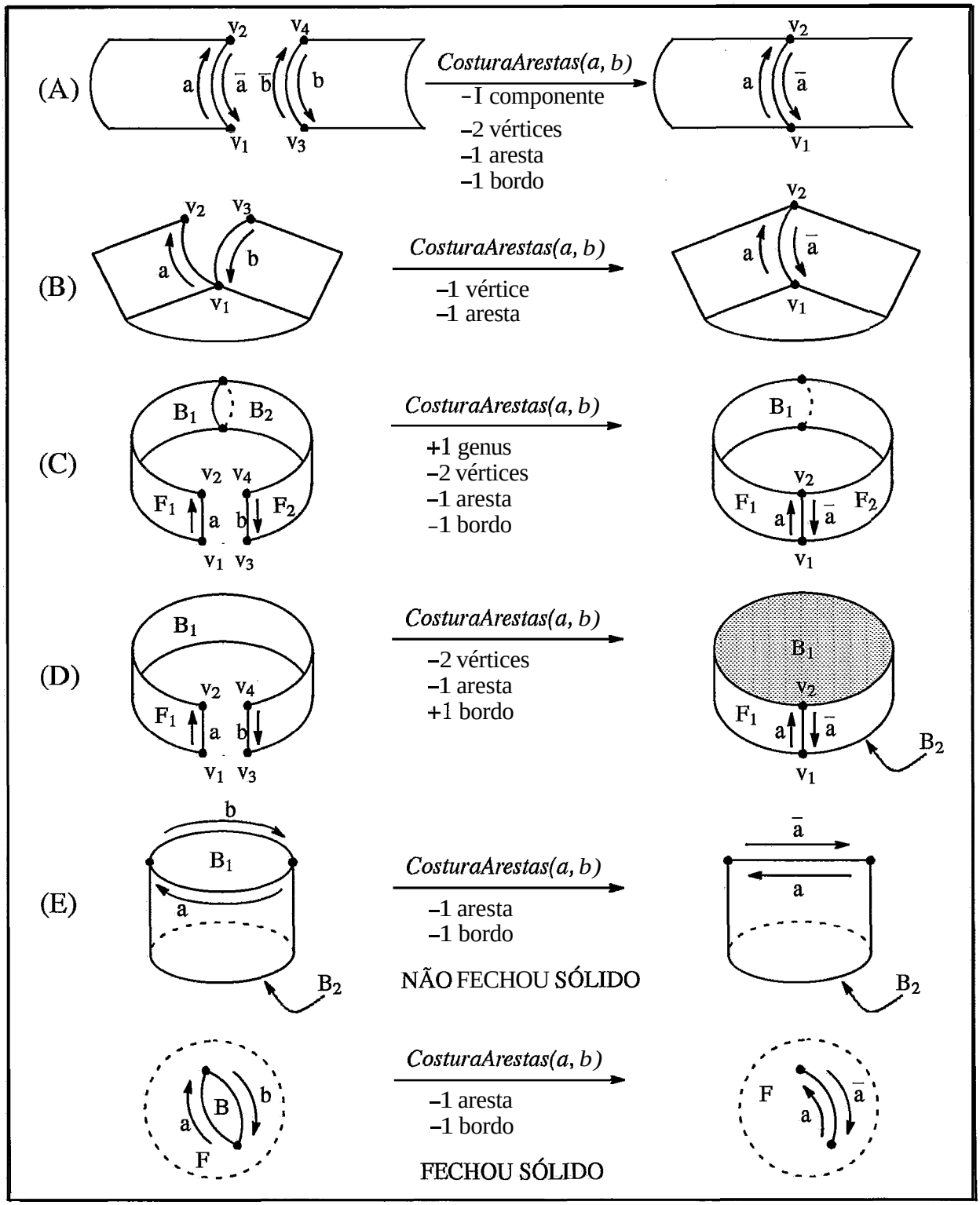


Figura 4.13 – Configurações possíveis na costura de retalhos

O pseudo-código a seguir implementa o procedimento acima descrito. Nele, a função *PoeCicloOrigem* atualiza o ciclo em tomo do vértice origem da aresta parâmetro, fazendo com que as arestas deste ciclo referenciem a informação também passada por parâmetro.

```
CosturaRetalhos(a, b) {
    ("Ajustes das informações de faces e arestas *)
    Troca(InfoAresta(Simétrica(a)), InfoAresta(b))
    Troca(InfoFaceEsq(Simétrica(a)), InfoFaceEsq(b))
    (* Ajustes nas informações dos vértices *)
    PoeCicloOrigem(b, InfoVertOrg(Simétrica(a)))
    PoeCicloOrigem(Simétrica(b), InfoVertOrg(a))
    (* Costura topológica *)
    AlteraÁlgebra(AntOrg(a), AntOrg(Simétrica(b)))
    AlteraÁlgebra(Simétrica(a), b)
    ("Eliminação da aresta b *)
    AlteraÁlgebra(b, AntOrg(b))
    AlteraÁlgebra(Simetrica(b), AntOrg(Simetrica(b)))
    MataAresta(b)
}
```

Ao longo do processo de construção da representação BRep de um sólido, vamos nos deparar não só com a situação exposta acima, como também com diversas outras configurações, expostas na figura 4.13, onde as arestas sendo costuradas, além de poder pertencer a uma mesma componente conexa, podem ter seus pares de vértices já pertencentes a um mesmo ciclo de arestas. São estas informações, em conjunto com a relação de vizinhança entre os bordos envolvidos na costura, que irão determinar as alterações impostas à álgebra de arestas pelo operador CosturaRetalho. A seguir listamos cada caso ilustrado na figura 4.13 e descrevemos as modificações a serem realizadas sobre a estrutura de controle das componentes conexas e seus bordos:

(A): Componentes conexas distintas &
Bordos distintos &
Ambos os pares de vértices distintos

Os dois bordos são unificados, o mesmo acontecendo com as componentes conexas envolvidas.

(B): Componentes conexas idênticas &
Bordos idênticos &
Apenas 1 dos dois pares de vértices é **distinto**

Nenhuma alteração no número de componentes conexas ou de bordos.

(C): Componentes conexas idênticas &
Bordos distintos &
Ambos os pares de vértices distintos

Os bordos são unificados e o número de componentes é mantido.

(D): Componentes conexas idênticas &
Bordos idênticos &
Ambos os pares de vértices distintos

O bordo é duplicado sem alterar o número de componentes conexas.

(E): Componentes conexas idênticas &
Bordos idênticos &
Ambos os pares de vértices idênticos

O bordo é eliminado, nada ocorrendo com relação ao número de componentes conexas. Caso este tenha sido o último bordo da lista desta componente, fica caracterizada a geração de um sólido BRep.

As modificações enumeradas acima podem ser melhor detalhadas no pseudo-código apresentado abaixo:

```
de acordo com o caso faça {
    CASO A {
        some NumVértices(Componente(b)) a Componente(a)
        some NumArestas(Componente(b)) a Componente(a)
        some NumFaces(Componente(b)) a Componente(a)
        decremente NumVértices(Componente(a)) de 2 unidades
        decremenfe NumArestas(Componente(a)) de I unidade
        decremenfe NumFaces(Componente(a)) de I unidade
        UnificaBordos(Bordo(a), Bordo(b))
        UnificaComponentesConexas(Componente(a), Componente(b))
    }
    CASO B {
        decremente NumVértices(Componente(a)) de 1 unidade
        decremente NumArestas(Componente(a)) de 1 unidade
    I
    CASO C {
        decremente NumVértices(Componente(a)) de 2 unidades
        decremente NumArestas(Componente(a)) de 1 unidade
        decremente NumFaces(Componente(a)) de 1 unidade
        UnificaBordos(Bordo(a), Bordo(b))
    I
    CASO D {
        decremente NumVértices(Componente(a)) de 2 unidades
        decremenfe NumArestas(Componente(a)) de I unidade
        incremente NumFaces(Componente(a)) de 1 unidade
        DuplicaBordo(Bordo(a))
    I
    CASO E {
        decremente NumArestas(Componente(a)) de 1 unidade
        decremente NumFaces(Componente(a)) de 1 unidade
        EliminaBordo(Bordo(a))
    I
I
```

4.6. ARMAZENAMENTO E RECUPERAÇÃO DE REPRESENTAÇÕES BREP

A estrutura de dados até então analisada, foi desenvolvida voltada para a representação do modelo em dispositivos de armazenamento de acesso não sequenciais (a memória de um computador), onde se permite a referência aos dados através de seus *endereços*. Entretanto, uma vez gerada a representação BRep, devemos dispor de meios que viabilizem seu armazenamento em dispositivos de saída, nos quais não é comum referenciarmos as informações desta forma. Para superar tal dificuldade, proporemos a seguir uma segunda estrutura de dados capaz de agrupar todas as informações pertinentes a uma representação BRep e que possa ser implementada em um dispositivo sequencial.

O artifício normalmente utilizado para substituir as referências por endereço é a indexação, onde cada elemento que anteriormente era acessado via seu posicionamento físico, recebe um índice pelo qual passa a ser identificado. Uma outra alternativa consiste em garantir que a ordem com que os dados são armazenados será a mesma ordem com que estes serão posteriormente recuperados. Em nossa proposta, a indexação é adotada em dois casos, a saber, nas arestas e vértices do modelo. Em contrapartida, as faces são gravadas e lidas em uma sequência lógica, que evita a necessidade de índices.

NVértices
NArestas
NFaces
$InfoVert_1$ $InfoVert_2$ ... $InfoVert_{NVértices}$
$InfoFace_1$ IndAresta IndVert IndAresta IndVert ... IndAresta IndVert CódigoFimFace
$InfoFace_2$ IndAresta IndVert IndAresta IndVert ... IndAresta IndVert CódigoFimFace

$InfoFace_{NFaces}$ IndAresta IndVert IndAresta IndVert ... IndAresta IndVert CódigoFimFace

Figura 4.14 – Representação sequencial de uma componente conexa

Passemos pois, a descrição da estrutura de armazenamento de uma dada componente conexa de nosso modelo BRep, a qual se encontra ilustrada na figura 4.14. Os primeiros dados armazenados indicam o número de vértices, arestas (não orientadas) e faces. Logo a seguir, a geometria (coordenadas) de cada vértice do modelo é armazenada na mesma ordem dos índices a eles atribuídos. A partir daí, cada face é armazenada, sendo primeiramente gravada sua informação não topológica associada (em nosso caso o material que a compõe), a seguir o ciclo de arestas que a constitui é percorrido ao passo que os índices de suas arestas e vértices origens são escritos. Após ter sido visitada a última aresta deste ciclo, uma marca é inserida indicando ter sido a face completamente descrita.

Já tivemos a oportunidade de constatar que, em nossa representação BRep, as únicas informações topológicas necessárias são aquelas fomecidas pelos operadores topológicos ProxFEsq e ProxOrg. Ora, o primeiro destes é facilmente obtido na estrutura acima, pois os ciclos de arestas que definem as faces se encontram armazenados

sequencialmente. Vamos nos ater portanto, a localização da aresta acessada pela função ProxOrg. Observe que ProxOrg(*a*) = Simétrica(AntFEsq(*a*)) e que a função AntFEsq pode ser computada através de aplicações sucessivas de ProxFEsq. Com isto, o problema se resume em localizamos a simétrica de uma dada aresta. A solução adotada consiste simplesmente em indexarmos as arestas e suas simétricas simultaneamente, de modo que, se a aresta a possui índice *i*, então a simétrica de a terá índice *i+1*. Assim sendo, localizar a simétrica de uma aresta equivale a busca daquela de índice uma unidade superior ao seu.

Concluímos então, que a estrutura apresentada é capaz de representar qualquer componente conexa do nosso modelo BRep. A extensão desta estrutura para a representação de um sólido BRep contendo várias componentes, consiste no acréscimo do número de componentes conexas ao início da estrutura e na repetição do processo visto acima para cada uma delas.

4.6.1. A Gravação de uma Componente Conexa

O algoritmo de armazenamento de uma componente conexa, constitui—e basicamente da escrita de seu número de vértices, arestas e faces e de 2 aplicações sucessivas de procedimentos de percurso. Inicialmente, os vértices do objeto são visitados, indexados e têm sua geometria armazenada. Desta forma, a partir do índice de um dado vértice, podemos localizar sua geometria no arquivo de saída. A seguir é efetuado um percurso por faces, onde são gravadas as informações que as caracterizam, como sua cor, material e outras. Além disto, as arestas do ciclo de cada face são também indexadas (junto com suas simétricas) e têm estes índices armazenados em conjunto com os índices de seus vértices origens, havendo um código a ser gravado para indicar o fim do ciclo de arestas de cada face. O pseudo-código a seguir ilustra a implementação deste algoritmo, que recebe por parâmetro a componente conexa a ser armazenada.

```
GravaComponenteConexa(Componente) {
    Grava(NumVértices(Componente), NumArestas(Componente), NumFaces(Componente))
    (* É atribuido um valor inicial ao índice de vértices *)
    IndVert = 1
    (* Vértices são indexados e gravados *)
    Percorre(ArestaDaComponente(Componente), GravaVértice, ProxOrg)
    (* É atribuido um valor inicial ao índice de arestas *)
    ÍndiceDaAresta = 1
    (* As faces são gravadas e as arestas indexadas *)
    Percorre(ArestaDaComponente(Componente), GravaFace, ProxFEsq)
}

GravaVértice(Aresta) {
    Grava(InfoVertOrg(Aresta))
    para toda aresfa ACiclo do ciclo origem de Aresta faça ÍndiceOrigem(ACiclo) = IndVert
    incrementa IndVert
}
```

```
GravaFace(Aresta) {
    Grava(InfoFaceEsq(Aresta))
    para foda aresta ACiclo do ciclo de face de Aresta faça {
        se ACiclo não foi indexada então faça {
            ÍndiceAresta(ACiclo) = IndAresta
            ÍndiceAresta(Simétrica(ACiclo)) = IndAresta +I
            incrementa IndAresta de 2 unidades
        }
        Grava(ÍndiceAresta(ACiclo), ÍndiceOrigem(ACiclo))
    }
    Grava(CódigoFimDeFace)
}
```

4.6.2. A Leitura de uma Componente Conexa

A recuperação de uma componente conexa ocorre de modo análogo a forma com que a mesma foi armazenada. O primeiro passo é portanto, a leitura de seu número de vértices, arestas e faces. A seguir, a geometria de seus vértices é recuperada e posta em um vetor de modo que estes dados possam ser acessados através do índice do vértice. Outro vetor é criado, cujo objetivo é guardar as arestas do modelo a partir de seus índices, ou seja, se desejamos aresta de índice 7, esta será achada na posição 7 do referido vetor. Lembre-se que o número de arestas armazenado é o de arestas não orientadas (geométricas), as quais equivalem a duas arestas orientadas (topológicas).

As arestas são geradas aos pares a partir do operador de construção CriaAresta, responsável pelo surgimento de uma áigebra composta de uma aresta e sua simétrica. Ora, como as arestas foram indexadas junto com suas simétricas (se o índice da aresta padrão é i, o da sua simétrica é $i+1$), estas podem agora ser diretamente armazenadas em posições consecutivas do vetor de arestas. Assim sendo, o primeiro par de arestas gerado é armazenado nas posições 1 e 2 do vetor e assim por diante. A partir daí, cada uma das faces da componente é lida, o que inclui a recuperação de sua informação não topológica associada e da lista intercalada de índices de arestas e dos seus respectivos vértices origens.

As informações de vértices origens, localizadas a partir do vetor de vértices, são desta forma atribuídas as respectivas arestas. Estas, por sua vez, podem ser acessadas a partir do vetor de arestas e conectadas através do operador AlteraÁlgebra até que a face seja fechada. Quando todas as faces tiverem sido processadas, a representação da componente terá sido reconstituída. Apresentamos a seguir o pseudo-código do procedimento exposto acima.

```
LeComponenteConexa() {
    Le(NumVértices, NumArestas, NumFaces)
    (* O vetor de vérfices é preenchido *)
    para IndVértice = I até NumVértices faça VetorDeVértices[IndVértice] = Le(InfoVertOrg)
    (* O vetor de aresfas é preenchido *)
    IndAresta = 1
```

```
    repita {
        Aresta = CriaAresta()
        VetorDeArestas[IndAresta] = Aresta
        VetorDeArestas[IndAresta+1] = Simétrica(Aresta)
        incrementa IndAresta de 2 unidades
    } até que IndAresta = 2*NumArestas
    ("As faces são lidas *)
    para cada face faça {
        InfoDeFace = Le(InfoFaceEsq)
        ("Leitura da primeira aresta da face *)
        Le(IndAresta, IndVértice)
        ArestaInicial = Aresta1 = VetorDeArestas[IndAresta]
        InfoVertOrg(Aresta1) = VetorDeVértices[IndVértice]
        ("As outras arestas da face são lidas e conectadas *)
        enquanto não achar código de fim de face faça {
            Le(IndAresta, IndVértice)
            Aresta2 = VetorDeArestas[IndAresta]
            InfoVertOrg(Aresta2) = VetorDeVértices[IndVértice]
            AlteraÁlgebra(ProxFEsq(Aresta1), Aresta2)
            Aresta1 = Aresta2
        }
        (* Conexão da última aresta com a primeira *)
        AlteraÁlgebra(ProxFEsq(Aresta1), ArestaInicial)
        ("A face recebe sua informação não topológica *)
        PoeCicloFace(ArestaInicial, InfoDeFace)
    }
}
```

4.7. CONCLUSÕES

Foi aqui desenvolvido um modelo de representação de superfícies com bordo. Esta ferramenta visa a construção de representações BRep de sólidos a partir da conexão de diversos recortes de sua fronteira, sendo estes recortes superfícies com bordo.

A estrutura de dados adotada para representar subdivisões sobre as superfícies foi a álgebra de arestas definida por Guibas e Stolfi [GUIB85]. A esta estrutura acrescentamos mecanismos de controle que nos permitem manipular superfícies com bordo. Um resultado importante obtido foi a completude da álgebra de arestas sob o domínio das 2-variedades com bordo. Com isto queremos dizer que a álgebra atende as propriedades de realizabilidade e representabilidade. Vimos também, que bastam apenas 3 operadores topológicos básicos (CriaAresta, MataAresta e AlteraÁlgebra) de simples implementação para que qualquer álgebra de arestas possa ser criada e modificada.

Outros 2 operadores de mais alto nível (CriaRetalho e CosturaRetalho) são suficientes para que a estratégia sugerida de construção da representação BRep de um sólido seja efetivada.

Capítulo 5

A Proposta de Conversão CSG → BRep

5.1. INTRODUÇÃO

O intuito deste capítulo é esclarecer o processo por nós desenvolvido de conversão de representações CSG para representações por fronteira. Para tal, serão utilizados os conceitos abordados em capítulos anteriores. Inicialmente apresentaremos o método como um todo para, posteriormente, detalharmos cada uma de suas etapas.

O procedimento de construção de uma representação BRep a partir de um sólido expresso através de uma árvore CSG está fundamentado sob o *paradigma de dividir para conquistar*. Deste modo, o sólido CSG inicial é dividido em duas metades, para as quais são computados os respectivos BRep's e, finalmente, estes são unidos obtendo-se a representação BRep do sólido original. Note com isto, que o problema em si não foi resolvido, mas sim dividido em outros provavelmente menos complexos e que podem vir a ser processados de modo idêntico, constituindo assim um procedimento recursivo de subdivisão espacial que em algum momento deve ser interrompido. Neste ponto somos obrigados a dispor de um método que efetivamente compute a representação BRep da porção do sólido contida no interior da região delimitada por esta subdivisão. A figura 5.1 ilustra a estratégia acima descrita. Nela o termo *integração* refere-se ao procedimento através do qual, após serem computadas, as representações BRep são unidas umas as outras na ordem inversa da recursividade ditada pela própria subdivisão espacial.

A vantagem oferecida por tal estratégia é, portanto, o fato de limitarmos o procedimento de conversão a uma região do espaço que provavelmente contém um número de superfícies menor que aquele encontrado no sólido completo, simplificando com isto a construção da representação BRep. Além disto, erros numéricos ou de natureza aproximativa tomam-se aceitáveis se levarmos em consideração o fato destes ocorrerem em regiões do espaço de dimensões tão pequenas quanto se queira.

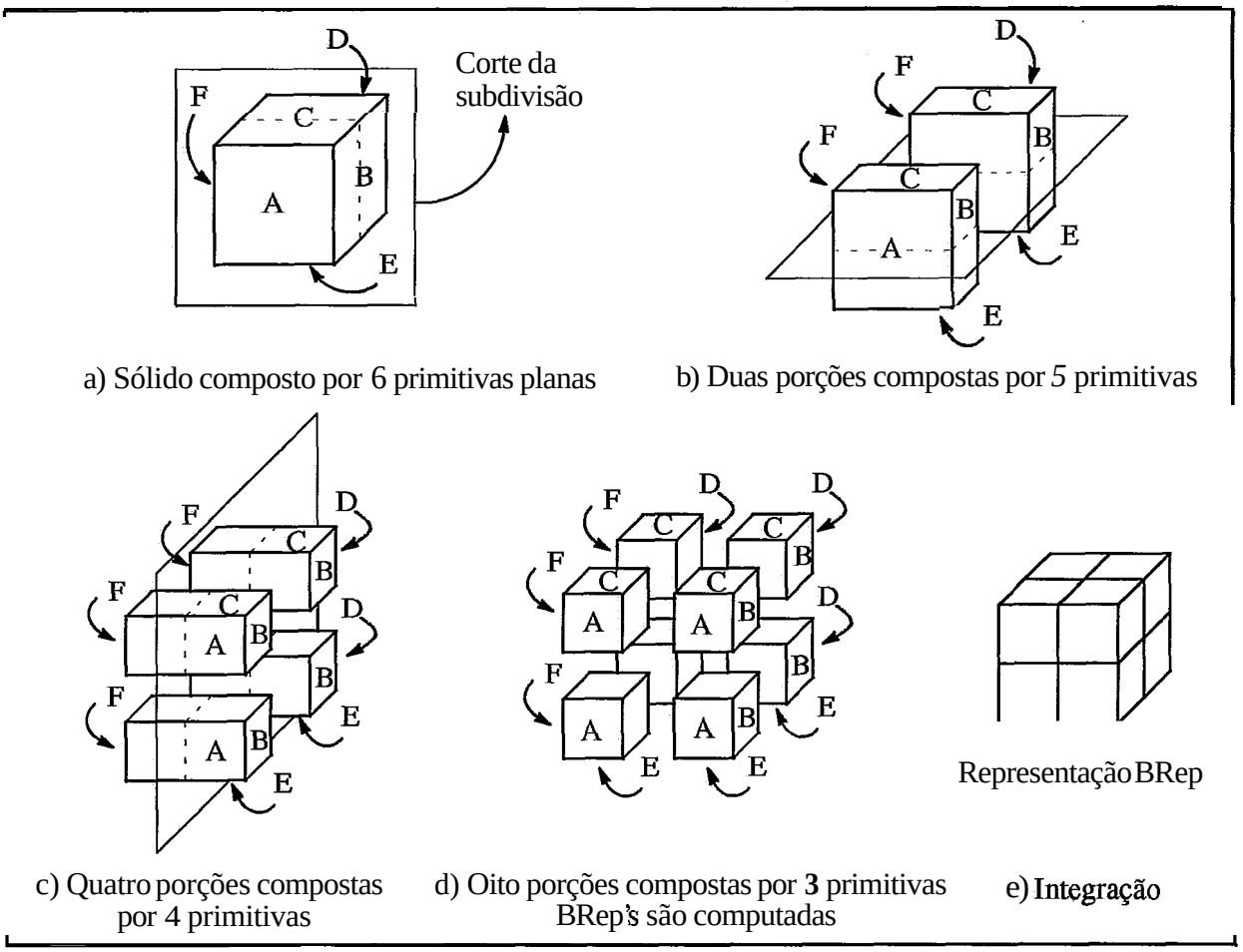


Figura 5.1 – A estratégia de conversão CSG → BRep

O pseudo-código exposto a seguir, sintetiza o processo de conversão proposto. Este procedimento recursivo recebe por parâmetros o sólido CSG a ser trabalhado e o nível atual em que se encontra a subdivisão espacial. O primeiro passo vem a ser a tomada de decisão quanto a subdividir ou não o sólido, o que é feito através da rotina *CompensaComputarBRep*, responsável pela verificação do critério de interrupção adotado. Este critério, geralmente leva em conta, entre outros fatores, a complexidade do objeto e o nível atual da subdivisão. Dependendo da conclusão chegada, ou o BRep é construído ou então o sólido é dividido em dois outros e o procedimento de conversão é recursivamente aplicado a cada uma das partes, computando seus BRep's e reagregando-os.

```
Converte(Sólido, Nível){
    se CompensaComputarBRep(Sólido, Nível) então reforne BRep(Sólido)
    senão faça {
        Subdivide(Sólido, Sólido0, Sólido1)
        BRep0 = Converte(Sólido0, Nível+1)
        BRep1 = Converte(Sólido1, Nívek1)
        retorne Integração(BRep0, BRep1)
    }
}
```

Uma vez tendo sido apresentada, a estratégia deve ter seus pontos chaves agora esclarecidos. Este será então o objetivo do restante deste capítulo, onde detalharemos as metodologias empregadas pelos procedimentos *Subdivide*, *CompensaComputarBRep*, *Integração* e *BRep*, presentes no algoritmo acima. Na seção seguinte descreveremos o processo de subdivisão adequado para por em prática a estratégia citada. A seção 5.3 apresentará um painel dos critérios de interrupção do processo recursivo de subdivisão espacial mais comumente utilizados. O procedimento de integração será discutido minuciosamente na seção 5.4. A seção 5.5 descreverá a técnica adotada na computação do BRep em uma célula da subdivisão. Finalmente, apresentaremos a implementação da estratégia de conversão com base nos conhecimentos adquiridos até então.

5.2. A SUBDIVISÃO BINTREE

Dentre os vários modelos de subdivisão espacial conhecidos, o que aparentemente melhor se adapta ao processo descrito é o da subdivisão Bintree. No caso aqui abordado, esta visa a criação de uma subdivisão espacial sobre uma região inicial do espaço a partir de um processo recursivo, onde cada iteração é composta de três etapas.

Inicialmente, a região (ou célula) é dividida em duas outras através de um corte plano perpendicular a um dos eixos coordenados, a seguir, cada uma das partes obtidas é novamente atravessada por um segundo plano perpendicular a um eixo distinto do primeiro. Um terceiro e último plano, perpendicular ao eixo restante é utilizado para repartir as células assim obtidas. Ao término da iteração, as células resultantes podem ser igualmente submetidas ao processo descrito, dando origem a uma subdivisão recursiva do espaço.

Observe que uma iteração completa da subdivisão Bintree acima relatada corresponde perfeitamente ao processo de subdivisão octree visto no capítulo 2. Entretanto, a iteração não necessita ser completa, ou seja, se em determinado momento desta for obtida uma célula que possa ser diretamente processada, esta não será subdividida, independentemente do que seja decidido para as demais células deste nível da subdivisão.

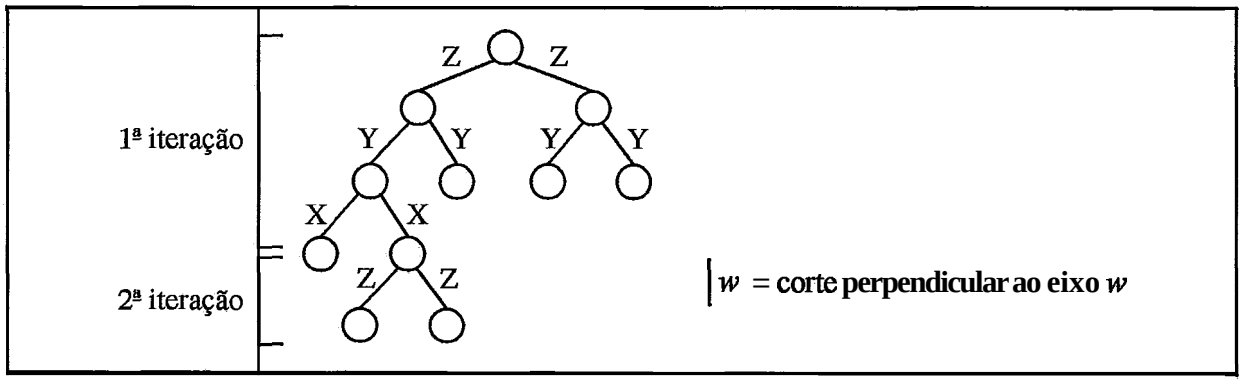


Figura 5.2 – A representação de uma Bintree

A estrutura de dados frequentemente utilizada para representar tal procedimento (figura 5.2) é conhecida como *árvore binária* e corresponde a um grafo acíclico onde cada

nó não terminal possui exatamente dois filhos. Como aqui a subdivisão é aplicada sobre um sólido CSG, cada nó da árvore binária contém uma árvore CSG localizada (capítulo 3) que representa a porção do sólido contida no interior da região do espaço delimitada pela célula correspondente.

5.3. CRITÉRIOS DE INTERRUPÇÃO DO PROCESSO DE SUBDIVISÃO

Concluimos anteriormente, que a subdivisão recursiva do espaço tem de ser obrigatoriamente interrompida em algum nível, para que o problema localizado em cada uma de suas células seja finalmente solucionado. Caso contrário, teríamos um processo que se estenderia infinitamente.

Em geral, os critérios de parada adotados em algoritmos baseados no paradigma de subdivisão espacial são de dois tipos. O primeiro deles trata-se de uma espécie de medição da complexidade do problema. Em termos do nosso caso específico, se a porção do sólido residente internamente a região delimitada pela célula for suficientemente simples para que se possa computar sua representação BRep sem maiores dificuldades, então o processo de subdivisão é interrompido. Com este critério garantimos que as células nas quais a complexidade do problema inicial foi substancialmente reduzida, não serão desnecessariamente submetidas ao processo recursivo de subdivisão.

Mostra-se ainda imprescindível, a existência de um segundo artifício que impeça a subdivisão indefinida de células complexas. Por exemplo, uma célula que englobe um vértice que seja a interseção de vários semiespaços. Assim sendo, define-se um nível máximo aceito para a subdivisão, estando este diretamente relacionado com a precisão do modelo. Em outras palavras, isto equivale a definição das dimensões da região do espaço, no interior da qual os erros numéricos ou aproximativos cometidos são considerados desprezíveis quando comparados com as dimensões do modelo, ou ainda, com a precisão desejada.

No caso do nível máximo estabelecido ser atingido, o problema localizado em seu interior deve então ser processado, mesmo que a solução obtida não reflita corretamente a realidade. No contexto aqui abordado, isto corresponde a definição de uma célula mínima, cuja representação BRep associada deva ser construída não importando a complexidade da porção do sólido ali representada.

Resta então, esclarecer o que se considera um sólido simples ou complexo. Esta classificação vai normalmente depender da implementação da rotina de cálculo da representação BRep pois, de acordo com a metodologia utilizada na abordagem do problema, o mesmo sólido pode ser visto como simples ou não. A despeito disto, um critério aparentemente independente do procedimento que computa a representação BRep, consiste em considerar como simples uma célula contendo apenas uma única primitiva CSG. Pois é de se esperar que, qualquer que tenha sido a estratégia de conversão implementada, esta se comporte de maneira exemplar nesta particular situação.

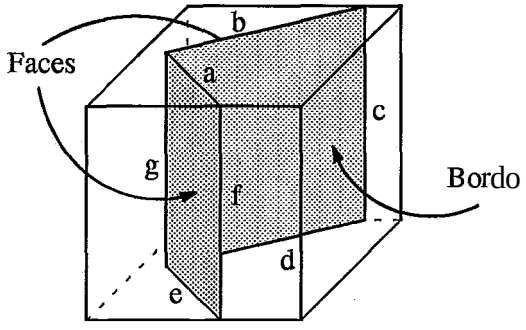
Vale mencionar que celulas localizadas ou no interior ou no exterior do sólido, ou seja, aquelas que são completamente cheias ou vazias e, por conseguinte, não contêm

em si nenhuma porção da fronteira do objeto, não necessitam ser submetidas ao procedimento responsável pela construção da representação BRep. Assim sendo, o algoritmo de conversão pode ser otimizado de modo a evitar o processamento de células que apresentem tais características.

5.4. O PROCEDIMENTO DE INTEGRAÇÃO

Uma vez que subdividimos o sólido CSG em inúmeras porções menores e provavelmente mais simples e que computamos a representação BRep de cada uma delas, resta então efetuar sua conexão, o que resultará na representação BRep do sólido original. Este processo será por nós denominado *integração*.

A integração pode ser entendida como sendo o procedimento inverso ao da subdivisão, no qual as células, ao invés de serem divididas, são unificadas. Consideraremos inicialmente, para facilitar o raciocínio, que a subdivisão espacial é completa. Em outras palavras, todos os nós terminais da árvore Bintree onde devem ser computadas representações BRep se encontram no mesmo nível da subdivisão.



a, b, c, d, e, f são arestas de contorno

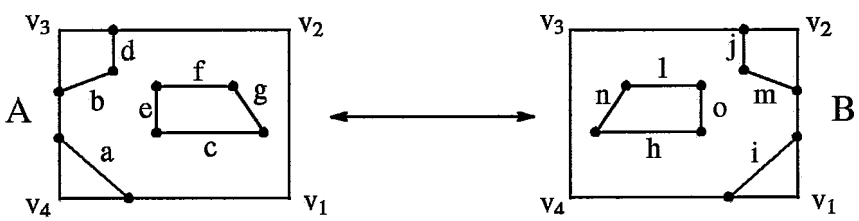
Figura 5.3 –Arestas de contorno

As arestas das representações BRep localizadas sobre as faces da célula da subdivisão serão por nós denominadas *arestas* de contorno (figura 5.3). Em termos do que foi dito no capítulo 4, este nome é a elas atribuido por fazerem parte do contorno da superfície ali representada. Estas arestas são incidentes a uma face e um bordo da superfície.

Os contornos das representações BRep devem ser costurados entre si, a partir da conexão das representações BRep contidas em células vizinhas, para que a representação BRep do sólido inicial seja aos poucos obtida. Note que as arestas de contorno são geradas a partir da interseção das faces da célula da subdivisão com o sólido em questão. A idéia da integração é então desfazer a divisão imposta ao sólido no decorrer do processo de subdivisão. Isto é feito a partir da costura das arestas de contorno situadas sobre as faces das células desta subdivisão. Portanto, as arestas de contorno encontradas em *faces comuns* a duas células devem ser ligadas entre si. Duas faces de células da subdivisão são ditas comuns caso sejam de mesma área (pertençam a células de um mesmo nível da subdivisão) e possuam a mesma localização espacial.

Ao ser separada por um plano de corte, uma célula dá origem a duas outras que compartilham de uma única face comum. É de se esperar que as arestas de contorno

situadas sobre estas faces sejam equivalentes entre si, ou seja, ao ser cortado, o sólido tem sua fronteira dividida de modo que as duas porções obtidas se *encaixem* perfeitamente. Assim sendo, basta que os pares de arestas de cada um dos lados sejam corretamente identificados e conectados. Esta identificação pode ser feita simplesmente através da geometria de cada aresta. Entretanto, esta seria uma solução de execução lenta e sujeita a erros numéricos pois, para cada aresta de um dos lados, teríamos que percorrer todas as do lado oposto até que fosse localizada a aresta de mesma geometria e, no caso de haver arestas muito próximas, estas poderiam ser tomadas por engano. Assim, pares de arestas poderiam ser indevidamente relacionados causando alterações na topologia do modelo.



Listas de arestas de A: {a, b, c, d, e, f, g}

Listas de arestas de B: {i, m, h, j, o, 1, n}

Figura 5.4 – Emparelhamento de listas de arestas de contorno

Nossa proposta é fazer com que a rotina responsável por computar a representação BRep no interior de uma célula devolva uma lista de arestas de contorno para cada face desta e que estas arestas estejam ordenadas na mesma sequência com que seus pares aparecem na lista da face comum da célula vizinha. A figura 5.4 ilustra esta pmpriedade. Deste modo, o algoritmo de integração dos nós da subdivisão consiste em percorrer sequencialmente as listas de arestas de contorno do par de faces compartilhadas por eles, conectando o n-ésimo elemento da primeira com o n-ésimo da segunda, não havendo necessidade de testes geométricos ou de procedimentos de busca. Assim sendo, o algoritmo apresenta uma estratégia otimizada e livre de erros numéricos. Com isto, fica ainda garantida a continuidade da fronteira do sólido, visto termos para cada aresta de contorno uma segunda aresta que se identifica topológica e geometricamente com aquela.

Um problema que pode surgir, decorrente da existência de um plano de corte coincidente com alguma superfície do sólido (primitiva CSG), se encontra ilustrado na figura 5.5. Ali, para cada face comum das células adjacentes foram computados dois ciclos desconexos de arestas de contorno, que ao serem processados originam duas componentes conexas ao invés de uma única como visto no sólido inicial. Para evitar que nos deparemos com tal situação, imporemos uma segunda propriedade ao procedimento que constrói o BRep interno a uma dada célula da subdivisão. Sempre que tivermos uma primitiva coplanar a algum plano de corte, este deve ser *perturbado* o suficiente para que aquela perca a condição de coplanatidade com o referido plano. A figura 5.5 ilustra tal procedimento.

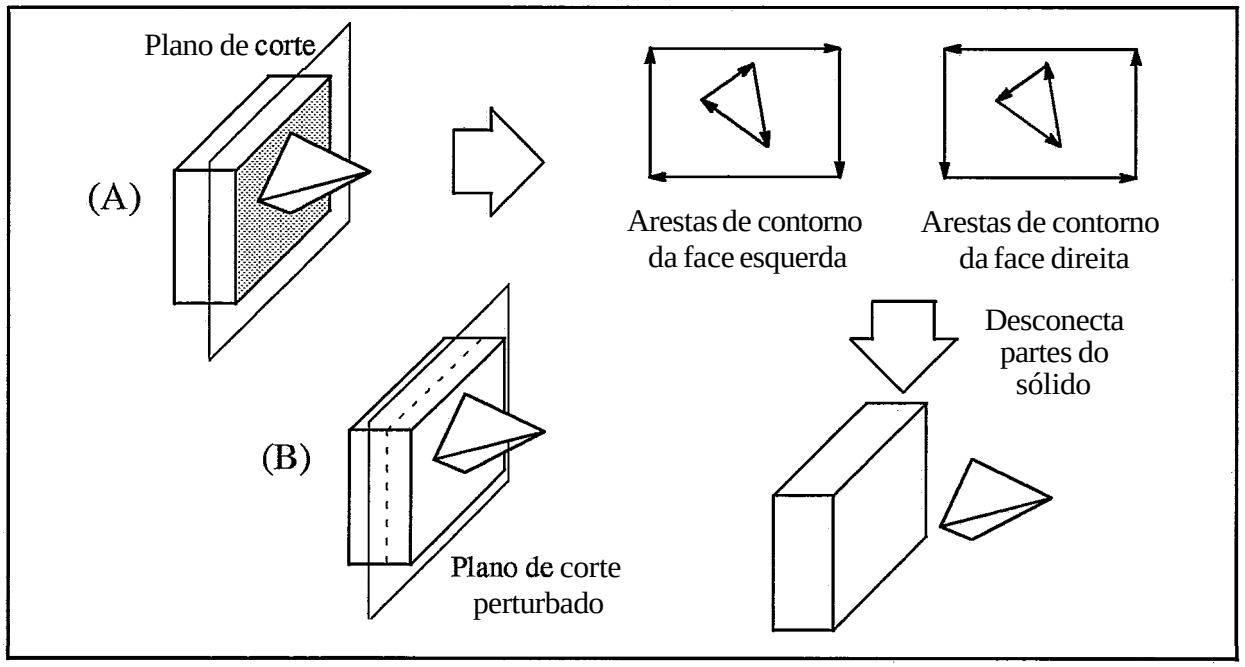


Figura 5.5 – Plano de corte coincidente com parte da superfície do sólido

Quando dois nós são integrados, os contornos de suas representações BRep (arestas de contorno) ao longo das faces comuns são costurados. Feito isto, estes nós são reagrupados passando a constituir uma **única** célula da subdivisão, situada um nível inferior na árvore Bintree. Ora, cada nó tinha a si associado uma lista de arestas de contorno para cada uma das suas seis faces, portanto, tais listas tem de ser transfendas para o nó resultante. A figura 5.6 exemplifica esta operação, a qual denominanos *concatenação* das listas de arestas de *contorno*. Deste modo, após a colagem das arestas de contorno das faces comuns, que faz com que suas respectivas listas sejam esvaziadas, parte-se para a concatenação das listas restantes. Por exemplo, a face de cima do nó resultante terá associada uma lista de arestas de contorno constituída das arestas pertinentes a face de cima do primeiro nó mais aquelas pertencentes a face de cima do segundo. O mesmo será válido para as demais faces do nó resultante.

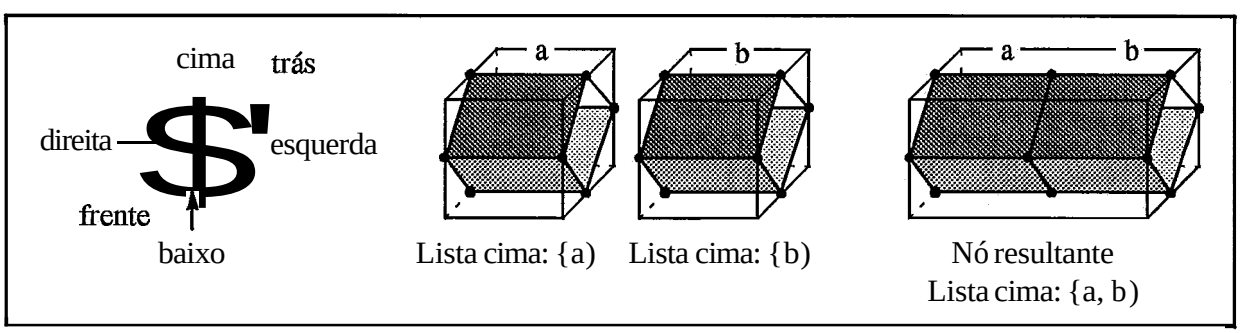
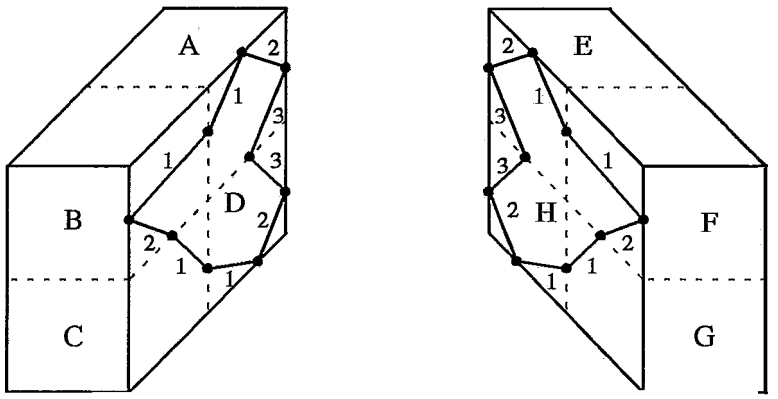


Figura 5.6 – Concatenação das listas de arestas de contorno

O processo de concatenação mencionado deve levar em conta a ordem com que as listas de arestas são concatenadas. Esta ordem não pode diferir da adotada na face vizinha da subdivisão, pois só assim podemos garantir o correto emparelhamento das listas de arestas de contorno, nas iteraçõesseguintes do processo de integração. A figura 5.7 nos fornece um exemplo deste procedimento.

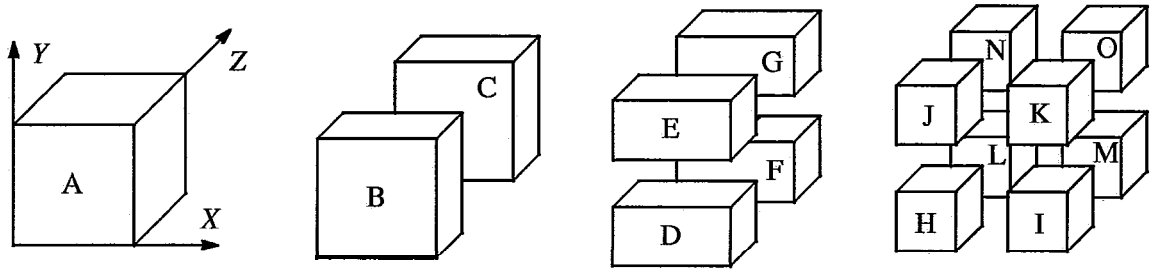


Ordem de concatenação: A, C, B, D
{1A, 2A, 3A, 1C, 1B, 2B, 1D, 2D, 3D)

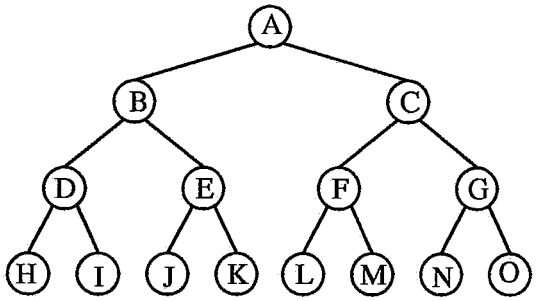
Ordem de concatenação: E, G, F, H
{1E, 2E, 3E, 1G, 1F, 2F, 1H, 2H, 3H}

Figura 5.7 – Equivalência das listas de arestas de contorno após a concatenação

Observe que, devido a integração ser o inverso da subdivisão, estaremos sempre lidando com faces comuns de igual área. Isto significa que ambas estão associadas ao mesmo número de arestas de contorno. Este fato, em conjunto com o correto critério de ordenação durante o procedimento de concatenação, nos leva a concluir que as listas de arestas associadas as faces sendo integradas sempre serão *compatíveis*. Por compatível, entendemos aqui, a propriedade de seus elementos serem identificados aos pares na ordem em que se encontram nestas listas.



a) Subdivisão Bintree do espaço (cortes em *Z*, Y e X, respectivamente)



b) Representação da Bintree através de uma árvore binária

Figura 5.8 – A subdivisão Bintree

O pseudo-código apresentado a seguir define uma ordenação para o procedimento de integração que nos garante o *emparelhamento* entre as listas de arestas das faces

comuns. Ele foi desenvolvido voltado para subdivisões Bintree que estejam de acordo com o padrão especificado pela figura 5.8, onde uma iteração da subdivisão constitui-se de três cortes planos do espaço, sendo o primeiro perpendicular ao eixo *Z* e, na sequência, dois outros cortes perpendiculares aos eixos Y e X, respectivamente. Seus parâmetros são um nó da subdivisão (inicialmente a raiz da árvore Bintree) e o eixo, perpendicular ao qual será efetuado o corte da célula. A rotina *Integração* unifica dois nós quaisquer da subdivisão a partir da identificação das listas de arestas de contorno de suas faces adjacentes passadas por parâmetro e concatena as demais, seguindo a mesma ordem com que seus respectivos nós são fornecidos. Por exemplo, a chamada *Integração(NóA, NóB,* Direita, *Esquerda)* faz com que a face direita do NÓA seja justaposta a face esquerda de *NóB*, ou seja, suas arestas de contorno são devidamente costuradas. A seguir, as listas de arestas de contorno associadas as demais faces destas células são concatenadas de modo que os elementos pertencentes ao *NóA* se situem no início das novas listas geradas.

```
Ordena(Nó, Eixo) {
    se Nó for terminal então retorne Nó
    senão caso Eixo seja {
        Z então faça {
            NóA = Ordena(Esquerda(Nó), Y)
            NóB = Ordena(Direita(Nó), Y)
            retorne Integração(NóA, NóB, Trás, Frente)
        }
        Y então faça {
            NóA = Ordena(Esquerda(Nó), X)
            NóB = Ordena(Direita(Nó), X)
            retorne Integração(NóA, NóB, Cima, Baixo)
        [
        X então faça {
            NóA = Ordena(Esquerda(Nó), Z)
            Nó5 = Ordena(Direita(Nó), Z)
            retorne Integração(NóA, NóB, Direita, Esquerda)
        [
    [
[

Integração(NóA, NóB, Face1, Face2) {
    CosturaListaArestas(Lista(Face1, NÓA), Lista(Face2, NóB))
    (*Face I e Face2 têm agora suas listas de arestas vazias *)
    para Face = Direita, Esquerda, Frente, Trás, Cima, Baixo faça {
        Lista(Face, Nó) = Concatena(Lista(Face, NóA), Lista(Face, NóB))
    [
    retorne (Nó)
}
```

Até então, consideramos a subdivisão espacial como sendo completa, isto é, todos os nós terminais da árvore Bintree pertencem ao mesmo nível da subdivisão. A figura 5.9 exibe um caso em que esta propriedade não é válida. Ali, é exposto um par de faces comuns, uma delas pertinente a uma célula não submetida ao processo de subdivisão e

uma segunda composta da união de quatro faces vindas de nós situados um nível superior na subdivisão. Neste caso, o procedimento de subdivisão irá nos garantir que ambas as faces são portadoras de representações geometricamente equivalentes. Entretanto, o número de arestas da face que se encontra repartida pode vir a ser superior ao da face não subdividida, como de fato ocorre na citada figura. Conclui-se então, não ser possível, neste caso, efetuar o emparelhamento das listas de arestas de contorno.

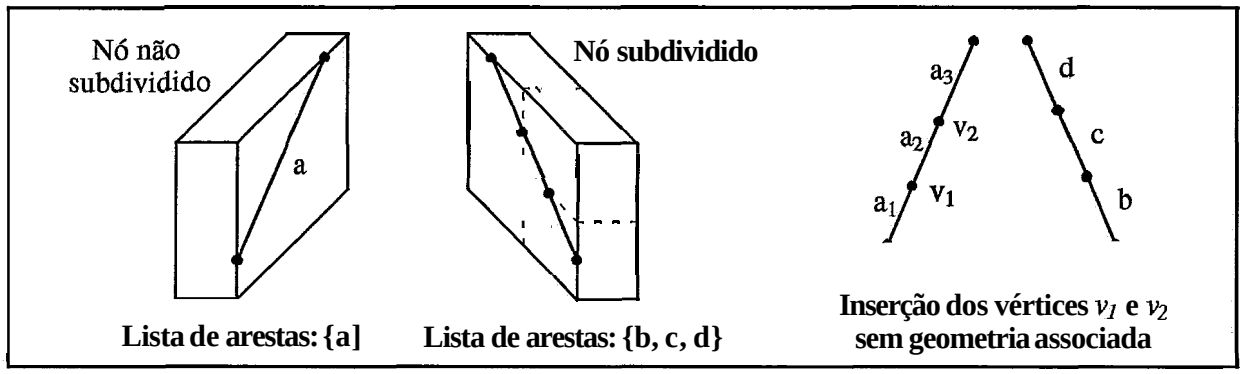


Figura 5.9 – Integração de nós pertencentes a níveis distintos da subdivisão

O obstáculo acima é superado mediante a imposição de uma propriedade sobre o procedimento responsável por computar o BRep interno a uma célula da subdivisão. Este procedimento deve ser capaz de identificar se a célula vizinha se encontra ou não subdividida. Em caso positivo, como vemos na figura 5.9, vértices devem ser consistentemente inseridos ao longo das arestas de contorno obtidas, visando viabilizar o emparelhamento destas com as da face comum da célula vizinha. Note que estes vértices não necessitam ter geometria associada, pois na face adjacente seus pares já terão estes dados computados. Generalizando este raciocínio, concluimos que para cada par de faces vizinhas da subdivisão, apenas uma delas precisa ter associada a geometria de suas arestas de contorno.

5.4.1. A Colagem das Arestas de Contorno

O procedimento de colagem de duas arestas de contorno foi exposto no capítulo 4 deste trabalho. Seus efeitos sobre a representação BRep em construção são basicamente dois, a saber, manter o controle dos bordos do modelo representando-os de modo o mais compacto possível e unir as faces externas mantendo sempre uma aresta entre as mesmas. Com isto, obtemos uma representação da fronteira do sólido composta por um número excessivo de faces pois, mesmo as arestas cujas faces adjacentes pertencem a uma mesma primitiva permanecerão no modelo. Isto faz com que uma mesma primitiva seja descrita por inúmeras faces ao invés de uma única. O resultado é então um modelo repleto de informações redundantes que necessita, consequentemente, de grandes áreas de armazenamento e que toma onerosa qualquer operação a ser realizada, como por exemplo, sua visualização.

Visando a obtenção de um modelo conciso, podemos adicionar ao pmcedimento de colagem, um critério para que as arestas redundantes sejam eliminadas, de modo que nossa representação seja composta apenas por arestas que, ou separem faces pertencentes a primitivas distintas, ou sejam responsáveis pela não formação de faces

com buracos, ou seja, aquelas cuja remoção implicaria na obtenção de uma face composta por mais de um ciclo desconexo de arestas, entidade esta não representável no modelo BRep por nós adotado. Portanto, uma aresta (o conjunto constituido da aresta orientada e sua simétrica) será dita redundante se possuir uma das propriedades enumeradas a seguir e ilustradas na figura 5.10:

(i) os ciclos de arestas que definem suas duas faces vizinhas são distintos, embora representem uma mesma primitiva CSG

(ii) ela aparece **2** vezes em seu ciclo de face e sua iemoção não gera ciclos desconexos de arestas

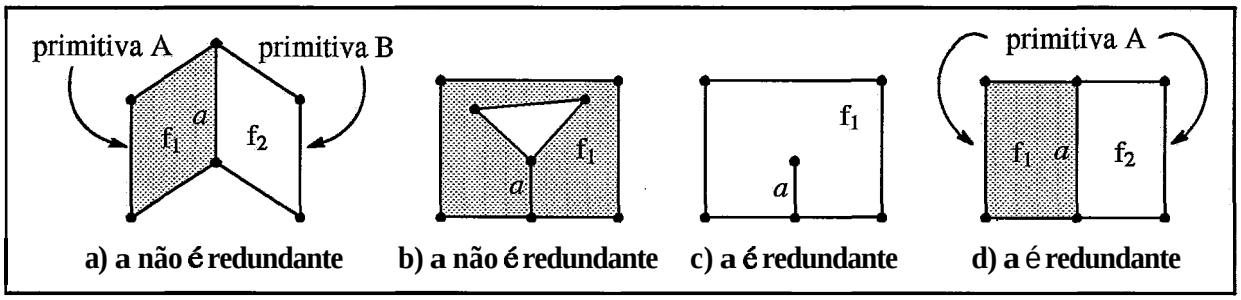


Figura 5.10 – Arestas redundantes

Observe que, no caso particular de trabalharmos com um modelo BRep baseado em faces planas (como de fato o fatemos), a propriedade *(i)* será válida apenas no caso de primitivas planas (semiespaços planos). Fato decorrente de ser este um modelo aproximativo e, deste modo, as diversas faces obtidas para se representar uma mesma primitiva equivalem a sua aproximação poligonal, a qual coincidirá com a própria primitiva somente no caso específico de semiespaços planos.

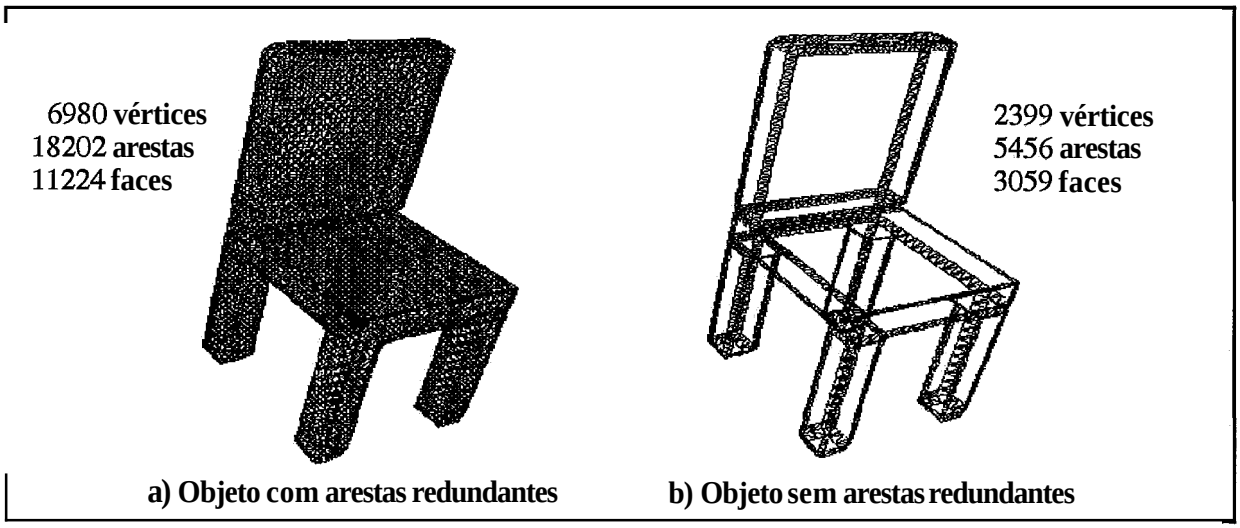


Figura 5.11 –Exemplo de remoção de arestas redundantes

Note que, se a propriedade *(ii)* for válida, devemos ainda verificar se a aresta sendo analisada não é a aresta derradeira que representa uma dada primitiva. Isto ocorrerá sempre que a aresta analisada for inteiramente responsável pela representação de uma componente conexa do modelo. Neste caso, esta não deve ser removida para que não

sejam perdidas as informações correspondentes a componente citada. Veja por exemplo, que no ítem (E) da figura **4.13** do capítulo anterior, temos a representação de uma superfície esférica por uma única aresta que satisfaz a propriedade *(ii)*. A remoção desta aresta implicaria na eliminação da referida componente.

A figura 5.11 exemplifica a redução da complexidade da representação BRep ao aplicarmos o artifício de remoção de arestas redundantes aqui sugerido.

5.5. A CONSTRUÇÃO DA REPRESENTAÇÃO BREP

O objetivo principal deste estudo é o traçado de uma estratégia de conversão de sólidos CSG em representações BRep a partir do uso de subdivisão espacial. A proposta apresentada independe da implementação do procedimento que efetivamente computa o BRep da superfície do sólido interna a uma dada célula da subdivisão. Esta, deve simplesmente satisfazer as duas propriedades descritas até então e listadas a seguir:

(i) as arestas da representação BRep situadas sobre faces comuns da subdivisão espacial devem ser geometricamente equivalentes e em igual número e, além disto, devem estar dispostas em listas de modo a permitir a identificação imediata dos pares.

(ii) caso haja um plano de corte coplanar a uma primitiva CSG, este deve ser perturbado até que a coplanaridade seja desfeita.

Embora a metodologia proposta esteja voltada para representações BRep genéricas, analisaremos aqui um algoritmo baseado em representações poligonais, lembrando que este pode ser substituido a qualquer momento por outro procedimento mais aprimorado que atenda as características *(i)* e *(ii)* acima.

5.5.1. A Aproximação Poligonal Interna a uma Célula

Buscamos aqui representar a porção do sólido restrita a uma célula da subdivisão a partir de polígonos aproximadores. Bloomenthal [BLO088] aborda o problema da aproximação poligonal de superfícies implícitas com base nos sinais que estas assumem nos vértices da célula que as contém. No caso de sólidos CSG, um sinal positivo indica, por exemplo, um ponto exterior ao objeto, ao passo que um sinal negativo corresponderia a um ponto interior ao mesmo. Portanto, considera-se que uma aresta da célula é interceptada pelo objeto caso esta possua vértices de sinais contrários, o que significa dizer que parte da aresta se encontra fora do sólido e o restante em seu interior, ou seja, a aresta é cortada pela fronteira do objeto. Assim sendo, arestas com esta característica possuirão um vértice do polígono aproximador. Uma vez localizados os demais vértices deste polígono, resta orientá-los de modo que o ciclo de arestas que os una forme uma face consistente, onde sejam corretamente identificados os lados externo e interno da fronteira do sólido por ela representada. Para isto, Bloomenthal propõe um algoritmo, ilustrado na figura 5.12, ao qual passamos a nos referir.

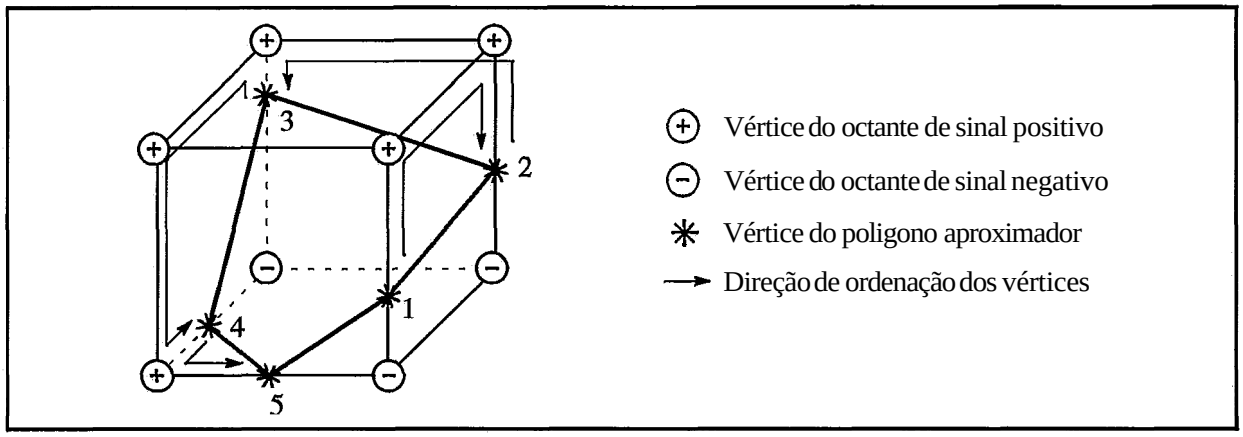


Figura 5.12 – Algoritmo de ordenação dos vértices da aproximação poligonal

O primeiro passo do algoritmo de ordenação dos vértices é a localização de uma aresta interceptante qualquer. O vértice do polígono aproximador nela computado será o primeiro da nossa face. A seguir, a aresta escolhida é orientada tal que sua origem seja o vértice de sinal positivo. Sua face esquerda é então circulada no sentido horário até ser atingida outra aresta interceptante, a qual será responsável pelo próximo vértice da face BRep. Daí por diante, o procedimento é repetido até que a aresta inicial seja novamente visitada.

A aplicação do método exposto sobre células de 8 vértices, como é o nosso caso, pode resultar em situações ambíguas ou em multiplicidade de faces. No intuito de contornar tais dificuldades, Bloomenthal sugere a subdivisão das células em tetraedros, nos quais, devido ao reduzido número de vértices são eliminadas as ambiguidades e a possibilidade de ser computado mais de um polígono aproximador. Dentre as diversas opções disponíveis, optamos pela *subdivisão* CFK [SALI91, MIRA89], advinda da teoria simplicial e reproduzida na figura 5.13.

Ao aplicarmos a estratégia acima proposta sobre um tetraedro, para computar uma aproximação linear da porção do sólido nele contido, podemos garantir que, para cada face do tetraedro haverá no máximo uma única aresta de contorno associada. Chega-se a esta conclusão, notando que uma face possui exatamente 3 vértices, o que torna possível a sseguintes distribuíções de sinais: três sinais positivos (ou negativos) ou dois sinais positivos e um negativo (ou vice-versa). No primeiro caso, nenhuma aresta interceptante é gerada, ao passo que, no segundo, teremos duas delas, cada qual contendo um vértice do polígono aproximador. Estes vértices, ao serem conectados, originam uma única aresta situada sobre a face do tetraedm.

5.5.2. A Montagem das Listas de Arestas de Contorno de uma Célula

A geração de uma representação BRep para uma dada célula da subdivisão consiste em unir os diversos retalhos obtidos em cada um dos seus tetraedms. Vimos que, sobre cada face de um tetraedro existe no máximo uma única aresta do retalho BRep criado em seu interior. Deste modo, a conexão destes retalhos equivale a colagem das arestas BRep associadas às faces adjacentes de tetraedms vizinhos. Observando a figura 5.13, nota-se que cada tetraedro de uma célula possui exatamente dois vizinhos com faces comuns. Isto nos dá, portanto, um total de 6 colagens a serem efetuadas.

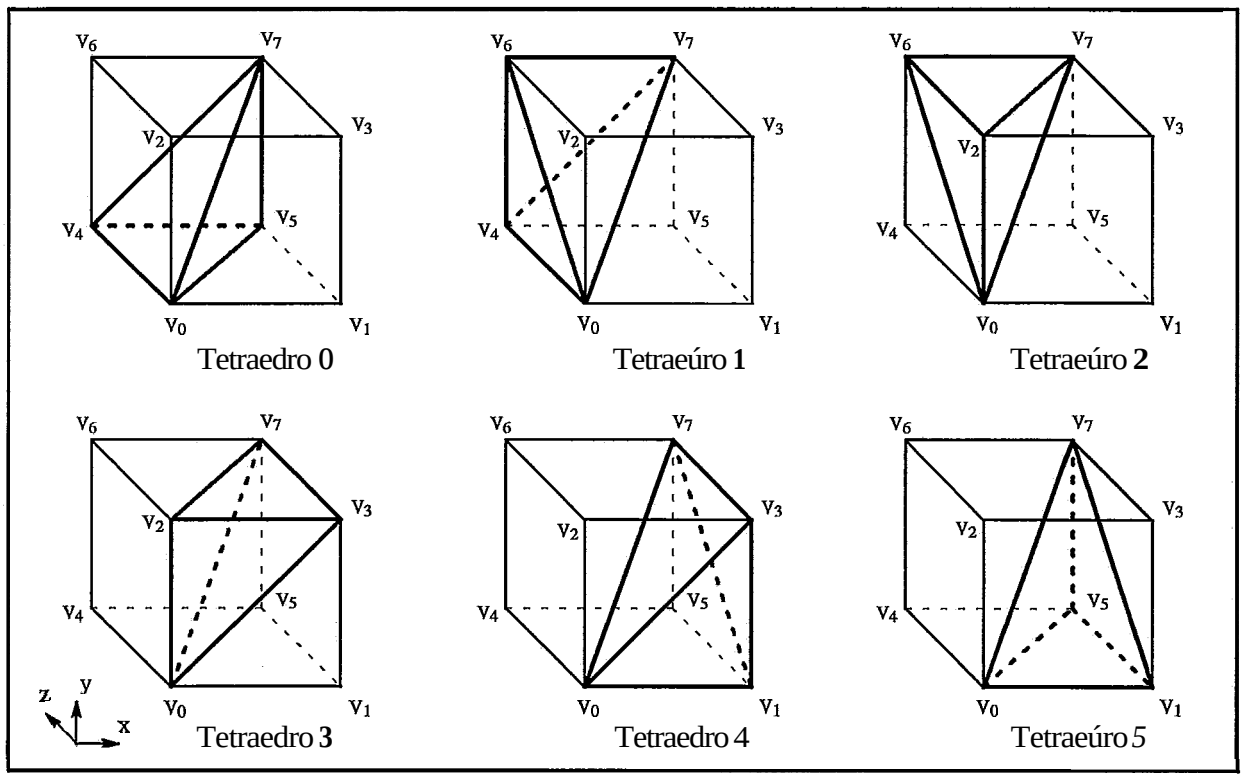


Figura 5.13 – Divisão do octante em tetraedros

Considerando os tetraedros como sólidos BRep e, supondo que estes sejam identificados pela aresta que tem por vértices origem e destino, respectivamente os vértices de índice 0 e 7 da célula, apresentamos o pseudo-código a seguir, responsável pelo processo de conexão dos retalhos ali obtidos. Vamos assumir que a informação associada às faces corresponda a aresta do retalho que as intercepta. Cabe lembrar, que o procedimento de colagem de arestas (*ColaArestas*) recebe por parâmetro duas arestas de contorno associadas a uma face pelo seu lado esquerdo e a um bordo pelo lado direito.

```
ColaRetalhosDosTetraedros() {
    para Ind=0 até 5 faça {
        (* A face do retalho está à esquerda da aresta *)
        se existe InfoFaceEsq(Tetraedro[Ind]) então faça {
            A1 = InfoFaceEsq(Tetraedro[Ind])
            A2 = InfoFaceEsq(Simétrica(Tetraedro[(Ind+1) mod 6]))
            ColaArestas(A1, A2)
        }
    }
}
```

Feito isto, temos para o octante em questão uma representação BRep onde faces com arestas comuns se encontram interligadas. Um último detalhe a ser cuidado é o preenchimento das listas de arestas de contorno relativas a cada face desta célula.

A forma com que foram construídos os tetraedros faz com que cada face do octante seja compartilhada por exatamente duas faces de tetraedros distintos. Isto implica em termos para cada face da célula o máximo de 2 arestas BRep associadas. Estas são inseridas na lista de cada face do octante de modo que a lista associada ao seu vizinho

por face seja também preenchida nesta mesma ordem. Por exemplo, se as arestas a_1 e a_2 foram inseridas nesta ordem na lista da face direita de um octante, a face esquerda do octante vizinho deve ter sua lista preenchida com arestas que sejam equivalentes a a_1 e a_2, nesta ordem.

O pseudo-código apresentado a seguir trata do correto preenchimento das listas de arestas BRep associadas as faces de uma dada célula de forma a validar a propriedade (i) exposta na seção 5.5:

```
IniciaListaArestasFacesCélulas(Célula) {
    (* Face direita *)
    Insere(Célula, Direita, InfoFaceEsq(AntFDir(Tetraedro[4])))
    Insere(Célula, Direita, InfoFaceEsq(AntFDir(Tetraedro[5])))
    (* Face superior *)
    Insere(Célula, Cima, InfoFaceEsq(AntFDir(Tetraedro[2])))
    Insere(Célula, Cima, InfoFaceEsq(AntFDir(Tetraedro[3])))
    (* Face de trás *)
    Insere(Célula, Trás, InfoFaceEsq(AntFDir(Tetraedro[0])))
    Insere(Célula, Trás, InfoFaceEsq(AntFDir(Tetraedro[1])))
    (* Face esquerda *)
    Insere(Célula, Esquerda, InfoFaceEsq(ProxOrg(Tetraedro[2])))
    Insere(Célula, Esquerda, InfoFaceEsq(ProxOrg(Tetraedro[1])))
    (* Face inferior *)
    Insere(Célula, Baixo, InfoFaceEsq(ProxOrg(Tetraedro[0])))
    Insere(Célula, Baixo, InfoFaceEsq(ProxOrg(Tetraedro[5])))
    (* Face frontal *)
    Insere(Célula, Frente, InfoFaceEsq(ProxOrg(Tetraedro[4])))
    Insere(Célula, Frente, InfoFaceEsq(ProxOrg(Tetraedro[s3])))
]
```

5.5.3. A Função Característica

O cálculo do vértice de corte situado sobre as arestas interceptantes dos tetraedros é feito a partir dos valores assumidos em seus vértices pela porção do sólido contido no interior dos mesmos. Este procedimento pode ser visto em detalhes em [SALI92]. A função que retoma estes valores denominamos *função característica*.

Em nosso caso, a função característica equivale a uma função que, dada uma árvore CSG e um ponto do espaço retorna um real. Uma interpretação geométrica para este número seria uma espécie de distância entre o ponto fornecido e a fronteira do sólido descrita pela árvore CSG.

Nossa função característica é computada recursivamente sobre a árvore CSG seguindo as regras abaixo listadas, onde P_1 e P_2 são primitivas:

(1) $P_1 \cup P_2 = \{ x \,/\, P_1(x) \leq 0 \;\; ou \;\; P_2(x) \leq 0 \}$
$P_1 \cup P_2 = \{ x \,/\, mínimo(P_1(x), P_2(x)) \leq 0 \}$

(2) $P_1 \cap P_2 = \{ x \,/\, P_1(x) \leq 0 \;\; e \;\; P_2(x) \leq 0 \}$
$P_1 \cap P_2 = \{ x \,/\, máximo(P_1(x), P_2(x)) \leq 0 \}$

Portanto, a função característica adotada simplesmente substitui a operação booleana de união pelo *mínimo* dos valores obtidos nas sub-árvores esquerda e direita. De modo análogo, o operador de interseção equivale ao *máximo* destes valores.

Ao ser computada individualmente para cada vértice, a função característica apresenta problemas de continuidade de uma célula para a sua vizinha, em [SALI91] vemos que isto é contornado se, ao ser computado o valor em um dado vértice de uma aresta, a árvore for previamente podada de modo a conter apenas as primitivas que efetivamente interceptem a aresta em questão.

Por último, para que o BRep computado atenda as exigências da estratégia proposta, temos que validar a restrição *(ii)* da seção 5.5, que diz respeito a planos de corte coplanares a primitivas CSG. Ora, uma primitiva que seja coplanar a um dado plano de corte apresentará valor nulo nos vértices de algumas células da subdivisão. Se evitarmos que isto ocorra estaremos, consequentemente, nos livrando da coplanaridade e satisfazendo a *(ii)*. Basta então que, ao nos depararmos com um valor nulo, resultante da aplicação da função característica em um dado vértice, este tenha suas coordenadas ligeiramente perturbadas de modo a obtermos um valor não nulo ao computarmos novamente a função característica.

5.6. A IMPLEMENTAÇÃO DA PROPOSTA DE CONVERSÃO *CSG* → BREP

Tendo descrito a metodologia proposta no sentido de realizar a conversão CSG → BRep de um sólido, passamos a exibir a forma com que tal procedimento foi por nós implementado.

Muito embora a subdivisão Bintree se adeque perfeitamente as necessidades do algoritmo desenvolvido, tínhamos em mãos um programa, elaborado por Comba [COMB91] e denominado *GeraOCSG*, cuja finalidade era gerar uma subdivisão octree do espaço, no qual encontrava-se um sólido CSG. Ou seja, ele gera uma árvore octree em cujas folhas encontram-se árvores CSG localizadas que representam a porção do objeto nelas contido. Tal estrutura de dados foi vista com detalhes no capítulo 3 e é conhecida pelo nome de *OctreeCSG*.

Para evitar o caso da união de nós vizinhos localizados em diferentes níveis da subdivisão (figura 5.9) vamos trabalhar sempre com árvores octrees onde todos os nós que não sejam vazios ou cheios estejam no mesmo nível da subdivisão. Dispondo de tal árvore, resta então adaptar a estratégia de integração descrita para árvores Bintree ao âmbito da subdivisão octree. Basta para tanto, como vimos anteriormente, considerar a subdivisão Bintree completa e a cada uma de suas células associar o octante correspondente. Assim sendo, como ilustra a figura 5.14, cada nó misto da OctreeCSG tem, primeiramente, seus filhos 0 e 1 lidos e processados para, posteriormente, serem unidos formando o paralelepípedo **A**. Na sequência, o mesmo se dá com os octantes 2 e 3, os quais geram o paralelepípedo B. A e B são então conectados dando origem a placa C. Os filhos 4 e 5 carregados a seguir formam o paralelepípedo D. Neste ponto, os octantes 6 e 7 são lidos totalizando 4 nós sendo processados simultâneamente, número este que não é ultrapassado durante o tratamento de um nó misto. O paralelepípedo E, criado a

partir da colagem dos dois últimos octantes é agregado ao paralelepípedo D, constituindo a placa F que, ao ser justaposta a placa C, finaliza o algoritmo de montagem do nó misto *G*.

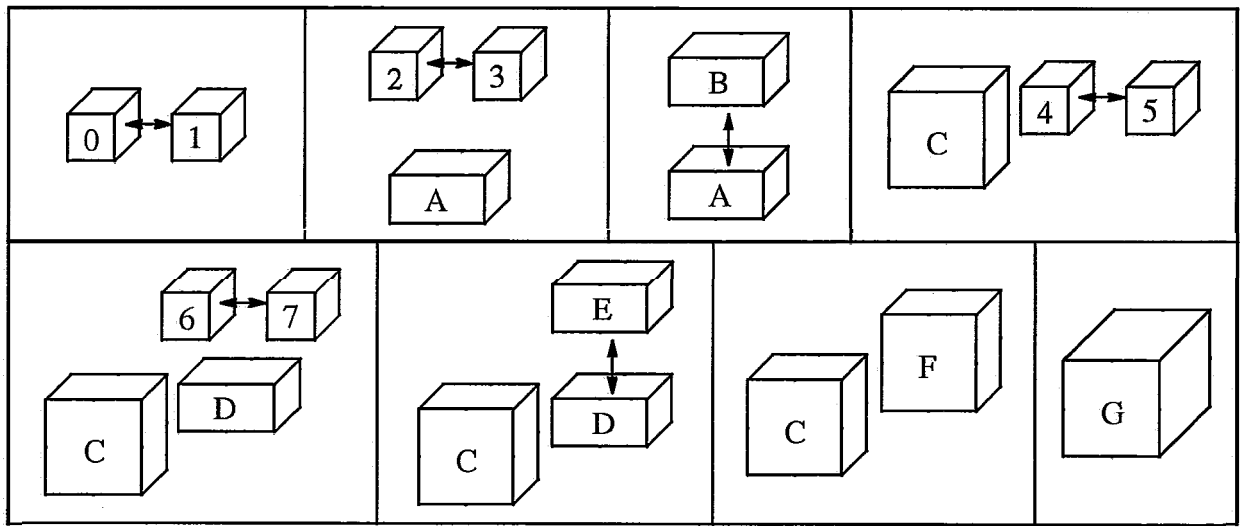


Figura 5.14 – O Processo de integração dos octantes

De posse desta estratégia, garantimos que o número total de nós carregados simultâneamente é no máximo igual a 4 para cada nível da subdivisão. Isto é, se a profundidade da árvore OctreeCSG sendo processada é P, teremos que arcar com o custo de manter o máximo de *4P+1* nós na área de trabalho, onde o número 1 somado corresponde ao primeiro octante misto (a raiz da árvore). Podemos ter uma boa idéia da quantidade de área de trabalho poupada ao observar que uma árvore octree completa de profundidade 8 possui um total de 19.173.961 nós. O número destes que devem ser simultaneamente processados não ultrapassa 33.

O pseudo-código que exibimos adiante corresponde a implementação das idéias apresentadas sob o ponto de vista da subdivisão octree do espaço. A rotina *Integração* foi discutida na seção 5.4.2, ao passo que o procedimento BRep se encontra detalhado ao longo da seção 5.5. Existem 4 tipos de nós OctreeCSG: Universo, Vazio, Primitiva ou Misto, estando estas classificações definidas no capítulo 3.

```
Converte() {
    Nó = LeNóOctreeCSG()
    caso Nó seja do tipo {
        UNIVERSO ou VAZIO então nada faça
        PRIMITIVA enfão retorne BRep(Nó)
        MISTO então faça {
            (* Geração do paralelepipedo A *)
            Nó0 = Converte(Filho(Nó,0))
            Nó1 = Converte(Filho(Nó,1))
            ParalelepípedoA = Integração(Nó0, Nó1, Direita, Esquerda)
            (* Geração do paralelepipedo B *)
            Nó2 = Converte(Filho(Nó,2))
            Nó3 = Converte(Filho(Nó,3))
            ParalelepípedoB = Integração(Nó2, Nó3, Direita, Esquerda)
```

```
            (* Geração da placa C *)
            PlacaC = Integração(ParalepípedoA, ParalepípedoB, Cima, Baixo)
            (* Geração do paralelepipedo D *)
            Nó4 = Converte(Filho(No,4))
            Nó5 = Converte(Filho(No,5))
            ParalelepípedoD = Integração(Nó4, Nó5, Direita, Esquerda)
            (* Geração do paralelepípedo E *)
            Nó6 = Converte(Filho(Nó,6))
            Nó7 = Converte(Filho(Nó,7))
            ParalelepípedoE = Integração(Nó6, Nó7, Direita, Esquerda)
            (* Geração da placa F *)
            PlacaF = Integração(ParalelepípedoE, ParalelepípedoF, Cima, Baixo)
            (* O octante G é montado *)
            OctanteG = Integração(PlacaC, PlacaF)
            reforne (OctanteG)
        ]
    ]
]
```

5.7. CONCLUSÕES

Este capitulo abordou a estratégia proposta para obtenção de representações BRep aproximadas de sólidos expressos por árvores CSG. Artifícios foram adotados ou sugeridos no intuito de acelerar o processo, minimizar a área de trabalho necessária e aprimorar a representação obtida.

No texto aqui exposto, visamos prioritariamente o entendimento da estratégia adotada, não nos atendo aos detalhes de natureza técnica, como algumas otimizações dos algoritmos ou a descrição minuciosa dos mesmos.

Cabe ainda mencionar que, na irnplementação disponível, somos capazes de identificar as componentes conexas da cena BRep obtida, bem como algumas propriedades topológicas como seu genus ou o número de vértices, arestas e faces.

Ressaltamos ainda, a flexibilidade oferecida pelo sistema, ao permitir que a rotina responsável pelo cálculo do BRep interno a uma célula possa ser substituida por qualquer outra mais elaborada que, por conseguinte, venha a produzir melhores resultados.

Vale por fim chamarmos atenção para a robustez do processo de integração, a qual decorre do fato deste ser independente de qualquer dado numérico do modelo, ou seja, de sua geometria.

Capítulo 6

Características da Implementação Desenvolvida

6.1. INTRODUÇÃO

Até este momento discutimos a metodologia proposta para solucionar o problema de conversão CSG → BRep. Ao longo deste capítulo daremos ao leitor uma visão geral do trabalho de implementação desenvolvido com base nestes estudos. Os recursos utilizados foram estações gráficas *SUN* 260, 3/60, *SPARC1+* e *SPARC2+*, rodando sistema operacional *SUNOS* 4.1 *(UNIX/BSD)*. A linguagem de programação adotada foi C (padrão *K&R*), muito embora alguns programas utilizados tenham sido desenvolvidos em *MODULA II*.

A filosofia por trás do sistema projetado consiste em tratar funções afins, ou que lidam com um certo tipo de estrutura de dados, em módulos isolados e independentes. Desta forma, o sistema e provido de uma certa flexibilidade, permitindo que alterações sejam efetuadas em pontos específicos do mesmo sem haver a necessidade de modificar o restante do sistema.

Cada um dos módulos criados, bem como os programas que os utilizam, serão apresentados a seguir. A sequência escolhida para a descrição dos mesmos busca refletir sua aparição no processo de conversão exposto na figura 6.1. Assim, na próxima seção falaremos a respeito do modelador CSG disponível. Logo depois, na seção *6.3* serão apresentados os programas que constroem árvores OctreeCsg a partir de representações CSG de sólidos. O módulo que lida com árvores OctreeCsg assim geradas será exposto na seção 6.4. Logo a seguir, na seção 6.5, detalharemos o modelador BRep construído. Por último, veremos a forma com que o programa conversor gerencia todos estes módulos e, na seção 6.7 apresentaremos as interfaces gráficas utilizadas na visualização dos resultados obtidos.

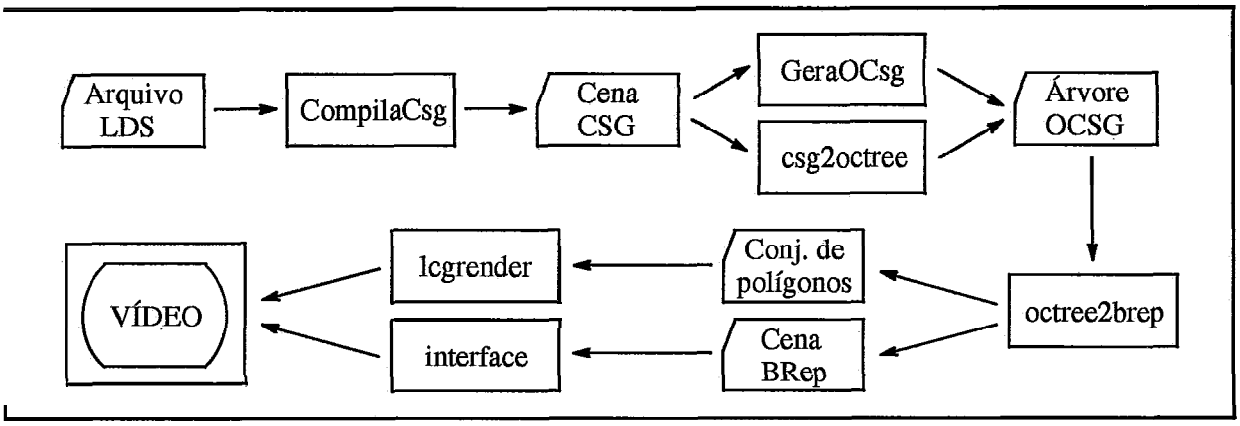


Figura 6.1 - O processo de conversão CSG → BRep

6.2. O MODELADOR CSG

O ponto de partida do processo de conversão trata-se da representação inicial do sólido CSG. Para tal, parte-se de um arquivo contendo a descrição do sólido em uma linguagem de alto nível denominada *LDS* (Linguagem de Definição de Sólidos). Esta linguagem, idealizada por Esperança [ESPE90], fornece meios para a construção de representações CSG de sólidos a partir de um conjunto de primitivas (Bloco, Esfera, Cilindro, Plano, Cone, Toro, Polinomial) e de alguns operadores (União, Interseção, Diferença, Rotação, Translação e Escala).

O arquivo com a descrição LDS do sólido deve ser submetido a um *filtro* (programa que recebe dados como entrada, os processa, e gera uma saída como resultado) cuja finalidade é gerar uma versão compactada e completa das informações fornecidas pela linguagem de definição de sólidos. O filtro responsável pelo procedimento descrito é denominado *CompilaCsg* e foi igualmente desenvolvido por Esperança [ESPE90].

6.3. A GERAÇÃO DA ÁRVORE OCTREECSG

Uma vez de posse do arquivo contendo a cena CSG, este é submetido a um segundo filtro denominado *GeraOCsg*. Sua finalidade é a construção da estrutura híbrida entre Octree e CSG descrita no capítulo 3. Este programa, escrito em linguagem MODULA *II* por Comba [COMB91], aceita que em sua linha de comandos sejam definidas algumas propriedades do processo de subdivisão do espaço através do qual a árvore OctreeCsg é obtida. Dois destes parâmetros devem ser aqui comentados, pois a eles devem ser atribuídos valores específicos para que a árvore gerada seja corretamente interpretada pelo restante do procedimento de conversão.

O primeiro parâmetro a ser discutido define um dos critérios de interrupção do processo recursivo de subdivisão, a saber, o número máximo X de primitivas aceitas em um nó não mínimo da árvore OctreeCsg. Em outras palavras, o processo de subdivisão é suspenso caso o nível máximo da subdivisão tenha sido atingido, ou se restam no máximo X primitivas incidentes ao octante sendo analisado. Como visto no capítulo 5,

em nossa implementação é necessário que todos os nós *Primitivas* se encontrem no mesmo nível da subdivisão. Assim sendo, o valor de Xdeve ser igual a *zero*. Desta forma, todos os nós primitivas se encontrarão no nível máximo da subdivisão.

O segundo parâmetro que merece nossa atenção é o *flag* que ativa a expansão de primitivas compostas. Isto é, primitivas constituídas por vários semiespaços são expandidas, passando a ser representadas por árvores CSG em cujos nós terminais encontramos cada um destes semiespaços. Um Bloco, por exemplo, passa a ser tratado como uma árvore CSG contendo seis semiespaços planos, ao passo que um Cone é interpretado como a interseção de um semiespaço cônico com um semiespaço plano. Deste modo passamos a trabalhar apenas com primitivas CSG que sejam semiespaços.

O fato de termos primitivas compostas na árvore sendo processada não é, de modo algum, um impecilho a estratégia proposta. Basta que seja fornecido para cada primitiva composta um procedimento que compute seu valor em um dado ponto do espaço. Optamos pela expansão de primitivas compostas por constatarmos que estas necessitam ter todos os seus semiespaços processados sempre que forem encontradas na árvore de subdivisão. Inclusive aqueles que não interceptam a célula correspondente e teriam sido eliminados no decorrer da subdivisão espacial.

Um detalhe técnico a ser mencionado é o fato do programa GeraOCsg não efetuar os cálculos de pertinência de uma primitiva com relação a uma determinada célula da subdivisão octree do mesmo modo que nosso programa de conversão o faz. Portanto, algumas primitivas por ele class cadas como pertencentes ao interior de algum nó, podem ser por nós classificadas como pertencentes a um nó vizinho ou a ambos (quando possuir pontos de tangência com faces do octante).

Diversas podem ser as situações de inconsistência, como a acima citada, provenientes de erros numéricos. Estes erros decorrem da precisão utilizada por cada programa, da forma com que são feitos os cálculos e até mesmo do modo com que compiladores distintos lidam com operações numéricas. Lembre-se que o programa GeraOCsg foi desenvolvido em *MODULA II*, ao passo que nosso trabalho se encontra implementado em linguagem C. Este impecilho, aliado ao fato do GeraOCsg ainda não ser capaz de tratar algumas das primitivas por nós utilizadas, nos levou a confecção de um programa para testar os casos problemáticos. O *filtro* desenvolvido é denominado *csg2octree*.

O csg2octree gera uma árvore OctreeCsg completa (todos os nós terminais se encontram no nível máximo da subdivisão)sem se preocupar com a poda da árvore CSG ao longo da subdivisão espacial. Ou seja, a árvore CSG inicial é transmitida integralmente de nível para nível da subdivisão e, somente ao atingirmos o nível máximo ela é podada. Para tal, as primitivas são classificadas contra os octantes da mesma forma e com a mesma precisão com que o fazemos no decorrer do processo de conversão, evitando deste modo os resultados inconsistentes. Obviamente, obtemos um tempo de processamento superior ao conseguido pelo programa GeraOCsg, uma vez que este efetua a poda da árvore CSG ao longo da subdivisão, deixando pelo caminho vários nós classificados como cheios ou vazios não situados no nível máximo. Entretanto, o programa csg2octree se apresentou como uma ótima ferramenta de testes e ampliou nosso universo de primitivas para qualquer forma expressa implicitamente por

desigualdades. A sintaxe do csg2octree, bem como um exemplo de sua aplicação, é exibida no apêndice B,

6.4. O MÓDULO OCTREE

Reunidas neste módulo, estão as funções ligadas ao processo de conversão que lidam diretamente com a estrutura de dados responsável pelo armazenamento da árvore OctreeCsg. Vamos encontrar nele, por exemplo, as rotinas de leitura. Estas carregam uma árvore OctreeCsg gerada pelos programas GeraOCsg ou csg2octree discutidos na seção anterior. Vale lembrar que elas se utilizam da estratégia descrita na seção 5.6, que exige a presença simultânea de apenas *4*NivelMax+1* nós da árvore na área de trabalho.

O módulo OCTREE é também responsável por coordenar o procedimento de integração discutido na seção 5.4. É ele ainda, quem envia os nós da árvore OctreeCsg para serem tratados pelo programa de conversão.

Observamos durante a implementação, que os procedimentos pertinentes a linguagem C, responsáveis pela gerência de áreas livres de memória, se comportavam de modo a consumir grandes parcelas do tempo de execução. Devido a propriedade mencionada de podermos computar previamente o número máximo de nós simultaneamente carregados e, além disto, deste ser um número relativamente pequeno, optamos por controlar nós mesmos os processos de alocação e desalocação de nós da árvore OctreeCsg. Assim sendo, uma área inicial, suficiente para armazenar o número máximo de nós carregados simultaneamente, é alocada uma única vez. Esta área é então gerenciada de modo a ter seus elementos já processados substituidos pelos novos elementos lidos. Em experimentos realizados chegamos a obter um ganho de até 20% do tempo total de execução.

6.5. O MODELADOR BREP

O modelador BRep por nós desenvolvido suporta a representação de subdivisões de 2-variedades com bordo. Ele é composto por seis módulos independentes. Cada um destes trata exclusivamente de um dado tipo de informação, ou é responsável por um certo grupo de funções que atendam a propositos semelhantes. O interrelacionamento entre os diversos módulos que constituem nosso modelador BRep pode ser visto na figura 6.2. Esta seção se dedica a descrição do conteúdo e finalidade de cada um destes módulos.

6.5.1. *O* Módulo BASE

O módulo BASE do modelador BRep é o responsável pela definição e manipulação da estrutura de dados formulada para representar subdivisões de 2-variedades. Nele foi implementada a álgebra de arestas de Guibas e Stolfi [GUIB85] com o acréscimo do controle de componentes conexas e bordos do modelo (capítulo 4), permitindo a representação consistente de subdivisões de superfícies com bordo.

A álgebra de arestas é diretamente acessada apenas por este módulo, cabendo aos demais módulos uma comunicação de mais alto nível, através das funções de arestas, operadores topológicos, e alguns operadores não topológicos que acessam determinados campos da entidade aresta. Assim sendo, não há a necessidade dos módulos externos terem conhecimento do modo específico como foi implementada a álgebra.

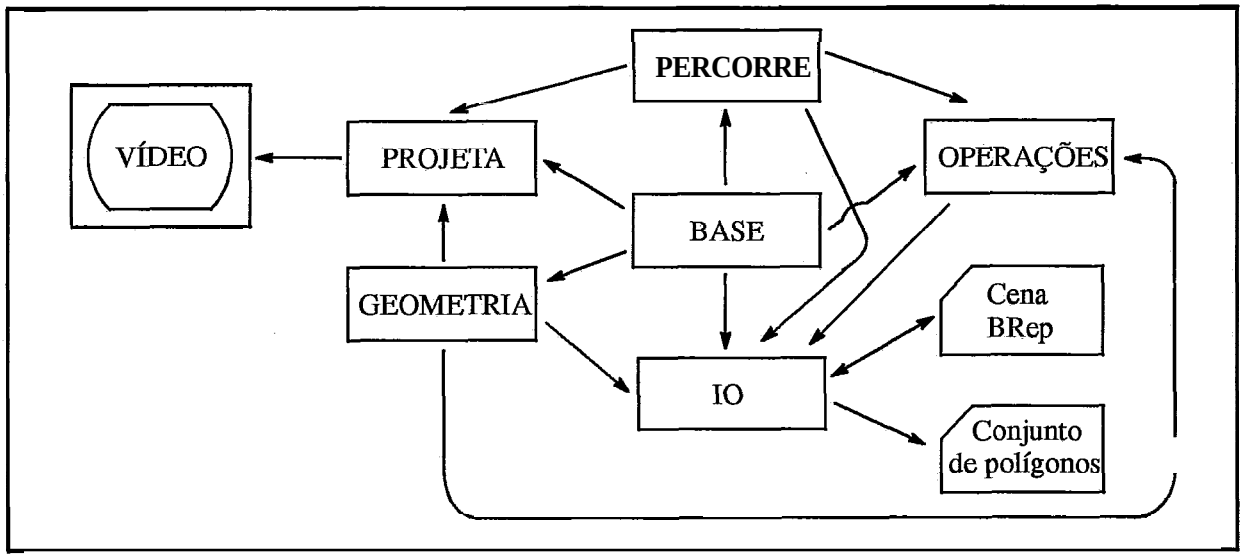


Figura 6.2 – A estrutura funcional do modelador BRep

A implementação da álgebra aqui encontrada não especifica a natureza das informações não topológicas associadas aos seus elementos básicos (vértices, arestas e faces), pois ela apenas referencia tais informações através de ponteiros. A consistência geométrica do modelo deve ser cuidada pelos operadores de mais alto nível desenvolvidos em módulos extemos.

A função Bordo, descrita na seção 4.4.2 e responsável pela limitação do domínio de representação da álgebra ao âmbito das 2–variedades com bordo, não foi diretamente implementada na álgebra de arestas. Entretanto, o controle de componentes conexas e bordos adicionado ao modelo fornece as informações necessárias para que os operadores de mais alto nível cuidem para que não seja criada nenhuma representação sintáticamente correta que não corresponda a alguma subdivisão de alguma 2–variedade com bordo.

Como vimos no capítulo 4, o modelo é representado inicialmente através de sua lista de componentes conexas. Em cada uma destas entidades armazenamos seus números de vértices, arestas e faces, a partir dos quais podemos computar (como uso da equação de Euler) o seu genus. A atualização destes campos é determinada, mais uma vez, pelos operadores de alto nível projetados.

6.5.2. O Módulo GEOMETRIA

A geometria do modelo, assim como os dados não topológicos do mesmo, são definidos neste módulo. Em nossa implementação lidamos apenas com representações de faces planas. Devido a isto, apenas os vértices se encontram relacionados a

informações geométricas. Estas informações são simplesmente as coordenadas dos pontos do espaço a eles associados.

O campo de informação não topológica das faces é utilizado para indicar uma estrutura contendo seu material e a primitiva que a gerou. Estas informaçõessão úteis para algoritmos de visualização e de eliminação de arestas redundantes. Observe que, para lidar com faces genéricas, basta ampliar a estrutura que contém as informaçõesde faces de modo a registrar, também, a geometria da superfície a ela associada.

Eventualmente, seria útil que as arestas também tivessem sua geometria descrita (atualmente nenhuma informação é a elas relacionada). Observe contudo, que estas modificações em nada afetariam os demais módulos, com a única exceção de que novos operadores de alto nível deveriam ser desenvolvidos para trabalhar com esta nova representação de modo a manter a consistência geométrica do modelo.

6.5.3. *O* Módulo PERCORRE

Neste modulo foram implementadas as rotinas de percurso de uma componente conexa do modelo abordadas na seção 4.4.6. Temos o total de três procedimentos implementados, a saber, o percurso por vértices, que visita todos os vértices da componente, o percurso por arestas e o por faces. Em todos eles, um procedimento de tratamento é passado por parâmetro e executado uma única vez para cada elemento visitado.

6.5.4. O Módulo OPERAÇÕES

Este modulo reúne a implementação dos operadores de alto nível responsáveis pela construção da representação BRep de sólidos a partir da estratégia descrita no capítulo 5. Isto é, nele se encontram implementados os operadores *CriaRetalho* e *CosturaRetalho*.

Além dos operadores já citados, algumas outras funçõesforam incluídas, como por exemplo, a eliminação de uma dada componente conexa da representação e a remoção de arestas redundantes. Esta última faz com que, durante o processo de costura dos retalhos, as arestas que separem duas faces originárias de uma mesma primitiva plana seja suprimida. Esta remoção é efetivada somente nos casos em que ela não venha a gerar faces com buracos. Tal procedimento, exposto na seção 5.4.3, resulta na obtenção de uma representação BRep mais concisa. A título de exemplo, o arquivo contendo a representação BRep da cadeira ilustrada no ítem (b) da figura 5.11 (sem as arestas redundantes) é aproximadamente 30% menor que o arquivo correspondente a cadeira do ítem (a)(com arestas redundantes).

6.5.5. *O* Módulo PROJETA

Aqui vamos encontrar as rotinas de saída gráfica, através das quais uma componente conexa da representação é exibida. Em outras palavras, o modulo PROJETA contém a implementação do sistema projetivo utilizado.

Não foi nossa preocupação produzir uma saída que mereça maiores comentários. Nosso objetivo foi, simplesmente, ter uma noção dos resultados obtidos a partir do

traçado dos segmentos ao redor das faces da subdivisão representada. Este traçado é feito levando em consideração o material de cada face, atribuindo a cada um, uma cor distinta. As arestas que foram geradas a partir de operações entre primitivas de materiais distintos recebem a cor branca, identificando uma legião de transição.

6.5.6. O Módulo IO

No módulo IO foram implementadas as rotinas de armazenamento e leitura de representações BRep de sólidos. Dentre elas, temos os procedimentos descritos na seção 4.6 do capítulo 4.

Foi também implementado um procedimento cuja finalidade é gerar um arquivo de saída contendo apenas um conjunto restrito de informações a respeito do sólido representado. Isto foi feito para que pudessemos utilizar um visualizador de polígonos disponível no LCG, o *lcgrender* [FONS92]. Neste arquivo gravamos inicialmente o número de vértices e faces do modelo. A seguir uma lista com os vértices é armazenada. Finalmente, cada face é descrita a partir dos índices dos vértices que a compõem.

O leitor irá encontrar no apêndice B uma descrição pormenorizada dos formatos dos arquivos de saída adotados.

6.6. A OBTENÇÃO DA REPRESENTAÇÃO BREP

O programa *octree2brep* é o responsável por gerenciar o restante do processo de conversão que resulta na obtenção da representação BRep do sólido CSG inicial. Para isto, o filtro octree2brep se utiliza do modelador BRep (aqui tratado como um módulo) e do módulo OCTREE especificados nas seções anteriores. Sua estrutura funcional pode ser vista na figura **6.3.**

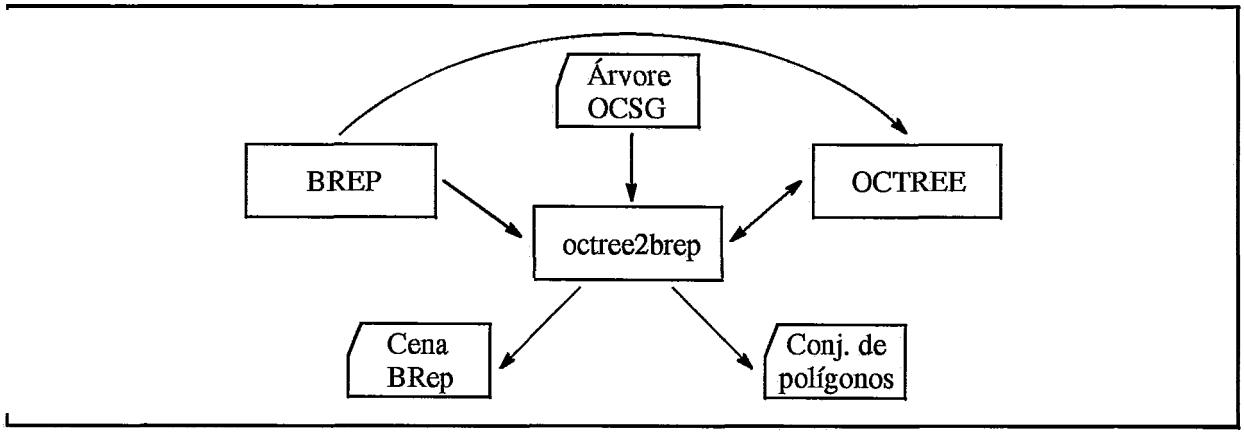


Figura 6.3 – A estrutura funcional do programa octree2brep

O octree2brep recebe como entrada um arquivo contendo uma árvore OctreeCsg. Através da estratégia exposta ao longo do capítulo 5, ele itera com os módulos BREP e OCTREE no intuito final de gerar a representação BRep do sólido desejado. Esta representação pode ser exteriorizada através de dois tipos distintos de arquivos de saída já discutidos na seção 6.5.6.

Aíém de podermos optar pelo formato do arquivo de saída, é também permitido ativar, através de um *flag*, a eliminação de arestas redundantes do modelo. Um segundo *flag* é responsável por ativar o aproveitamento de componentes que ainda possuam bordos ao fim do processo de conversão. Se ativado, todas as componentes conexas que se encontrarem nesta situação terão seus bordos substituídos por faces, podendo ser tratadas normalmente como sólidos. Isto é útil quando não sabemos exatamente o posicionamento espacial do objeto sendo processado. Nestes casos, somos obrigados a repetir o procedimento até dimensionarmos corretamente a *octree* de modo a não deixar de fora nenhuma porção deste sólido. Se este *flag* não for ativado, todas as componentes que possuirem bordos serão automaticamente eliminadas. O apêndice B exibe a sintaxe do programa octree2brep e exemplifica sua aplicação.

6.7. A INTERFACE DE VISUALIZAÇÃO

Para que o resultado do processo de conversão pudesse ser examinado visualmente elaboramos uma interface gráfica. Nesta interface, uma cena BRep pode ser carregada e exibida através da iteração com o módulo PROJETA descrito na seção 6.5.5. Nela temos acesso isolado a cada uma das componentes conexas da cena, das quais podemos obter o genus, o número de vértices, o de arestas e o de faces. A cena é vista através de projeção em perspectiva, onde a posição do observador é fornecida pelo usuário.

Uma segunda opção para visualizar o resultado da conversão é o visualizador *lcgrender* [FONS92]. Este programa dispõe de meios para exibir um conjunto de polígonos planos utilizando-se de algoritmos de iluminação que permitem o posicionamento de fontes de luz. Desta forma, obtemos imagens bem mais atraentes que as obtidas a partir da interface mencionada acima.

6.8. CONCLUSÕES

Apresentamos ao longo deste capítulo as características da implementação dos diversos procedimentos que se sucedem ao longo do processo de conversão proposto. É interessante notar que contamos com o trabalho prévio de vários outros companheiros que nos antecederam, trabalho este, que nos poupou muitos esforços e economizou-nos algum tempo ao nos fornecer ferramentas importantes como os programas CompilaCsg, GeraOCsg e lcgrender.

Vimos também, alguns obstáculos surgidos durante a implementação bem como a solução adotada para cada um deles. A arquitetura do sistema foi exposta em vários níveis no intuito de transmitir a noção modular do mesmo. Característica esta, que agrega em módulos, funções que possuam finalidades afins ou que lidem com um certo tipo de estrutura de dados. Isto toma o sistema bastante flexível, pois ao efetuarmos alterações sobre qualquer estrutura de dados basta nos concentrarmos em modificar consistentemente um único módulo.

Capítulo 7

Considerações Finais

7.1. AVALIAÇÃO DO TRABALHO

Foram obtidos dois importantes resultados decorrentes do trabalho aqui apresentado. O primeiro deles vem a ser o desenvolvimento de um modelador BRep e o segundo, a definição de uma estratégia de conversão de modelos CSG para modelos BRep.

Em geral, as técnicas encontradas na literatura [MANT82, HIRO85, REQU85, PAOL89] que visam a construção de representações BRep de sólidos estão baseadas em operadores de alto nível que, a todo instante, transformam uma representação BRep consistente em outra. Ou seja, no decorrer de todo o processo de construção da representação BRep final estaremos sempre operando com representações BRep de sólidos (fronteiras que englobam volumes).

A metodologia por trás do modelador aqui proposto é inovadora, posto que, a construção do modelo BRep é feita a partir da *costura* de recortes da superfície do sólido a ser representado. Estes recortes equivalem a superfícies com bordo. Com isto, a obtenção de uma representação BRep consistente é feita através da união de diversas componentes que não correspondem a representações de sólidos, mas sim de superfícies com bordo. Para que esta metodologia pudesse ser posta em prática o modelador foi equipado com uma estrutura capaz de gerenciar suas diversas componentes conexas e seus respectivos bordos. Daí, concluimos dispor de um modelador cujo *poder de expressão* (capítulo *2)* é ligeiramente mais abrangente do que o de um modelador BRep, sendo capaz de representar consistentemente subdivisões de 2–variedades com bordo. Algumas outras propriedades podem ser extraídas do modelador de superfícies desenvolvido, dentre elas destacamos:

- O modelador está fundamentado em uma estrutura de dados (álgebra de arestas) sedimentada sob uma rigorosa teoria matemática, com a qual, reduzimos a níveis mínimos a redundância encontrada nos modeladores de superfícies conhecidos.

- Com apenas 3 operadores básicos pode-se construir ou eliminar qualquer representação sintaticamente correta.

- Dois operadores de alto nível são suficientes para tornar operacional a estratégia de construção de representações BRep a partir da costura de retalhos.

Quanto a estratégia proposta de conversão de modelos CSG para modelos BRep, pode-se dizer que sua inovação é o fato desta se utilizar do paradigma de subdivisão espacial. Existe na literatura especializada uma série de trabalhos seguindo este princípio [NAVA87, ROSS89]. Entretanto, até onde nos foi possível pesquisar, eles apenas sugerem o uso da subdivisão espacial, sem contudo estabelecer critérios que permitam a conexão das porções da superfície do sólido computadas no interior de cada célula da subdivisão.

Uma segunda técnica abordada na literatura, busca a construção da representação BRep a partir da descoberta e conexão das arestas de interseção entre as diversas primitivas da árvore CSG. Uma versão deste último método pode ser vista em [REQU85]. Entretanto, sem as vantagens oferecidas pela subdivisão espacial, somos ali obrigados a testar cada primitiva contra todas as outras a cada vez que novas interseções são procuradas. Este processo é extremamente moroso e sujeito a erros numéricos, principalmente quando as primitivas não são lineares.

Vale comentar que nossa proposta é, primariamente, a descrição de uma metodologia que permita computar representações BRep correspondentes a modelos CSG utilizando-se do paradigma da subdivisão espacial. No decorrer deste processo tornou-se necessário optar por várias ferramentas que nos permitissem por em prática a nossa idéia. Nestas decisões não foi levado em consideração o fato da nossa escolha ser a melhor possível em termos de velocidade, precisão ou concisão das representações obtidas. Buscamos apenas tornar operacional a metodologia idealizada, implementando-a de forma a aproveitar os recursos existentes em nosso local de trabalho. Por este motivo, não faz sentido computar a ordem do algoritmo apresentado, visto que esta dependerá das ordens dos procedimentos adotados para a subdivisão espacial (árvore Octree CSG), para a computação da representação da superfície interna a uma célula desta subdivisão (aproximação linear por faces planas) e, ainda, para a conexão destas representações (procedimento de integração).

Observe que o uso de subdivisão espacial sobre o sólido CSG inicial faz com que a fronteira do mesmo seja *recortada* em diversas porções que, posteriormente, devem ser reagregadas com a aplicação do procedimento de integração descrito no capítulo 5. Com isto, o processo de construção da representação BRep se adapta perfeitamente ao critério adotado em nosso modelador.

Na seção seguinte veremos alguns testes efetuados a partir da nossa particular implementação. Aproveitaremos estes testes para expormos alguns problemas do método e sugerir soluções e otimizações.

7.2. AVALIAÇÃO DOS TESTES

Para os testes que se seguem serão utilizados os objetos ilustrados na figura 7.1. Estes objetos foram obtidos a partir de árvores CSG submetidas ao processo de conversão descrito, utilizando-se o nível 5 de subdivisão. Destes, apenas o cálice foi submetido a remoção de arestas redundantes. Na figura, exibimos sem o tratamento de linhas ocultas, o *wireframe* da representação BRep de cada sólido. O icosaedm é um poliedro constituído pela interseção de 20 semiespaços planos, a *cadeira* é formada pela união de 6 blocos, o *pluft* corresponde a união de 11 esferas e o cálice é composto pela diferença entre duas elipsóides unida com dois cilindros.

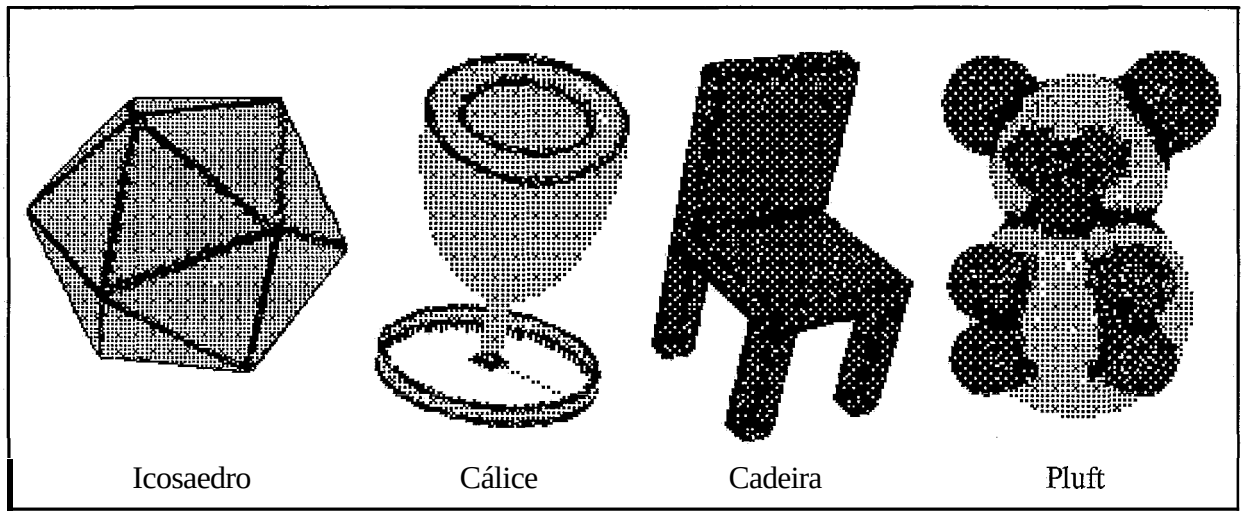


Figura 7.1 – Sólidos de teste

Note que, apesar de não operarmos com faces com buracos, o processo de conversão cuida, automaticamente, de manter a consistência topológica do modelo a partir da conservação de arestas interligando o ciclo de arestas de uma face com os ciclos de arestas que gerariam buracos nesta face. Observe, por exemplo, o cálice visto na figura 7.1, onde em sua base, formada por uma face plana, podemos verificar claramente a existência de uma ligação entre esta face e o restante do objeto. Caso não houvesse esta ligação o cálice seria representado por duas componentes conexas.

Uma noção do tempo gasto no processo de conversão é fornecida pela tabela **7.1.A** conversão de todos os sólidos ali exposta foi executada com a opção de remoção de arestas redundantes discutida no capítulo 5. Nesta tabela vamos encontrar o procedimento subdividido em duas etapas, a saber, a subdivisão espacial e a conversão. O tempo gasto na primeira fase pode ser consideravelmente reduzido se buscarmos técnicas mais velozes de subdivisão espacial. Métodos que se utilizam da subdivisão simplicial do espaço têm se mostrado muito eficientes no tratamento deste tópico [SALI92].

A tabela 7.2 divide o processo de conversão, apresentado na tabela 7.1, nos tempos gastos com o tratamento da geometria do modelo, do gerenciamento da leitura e escrita de informações, da construção do modelo topológico e de administração da memória de trabalho. Várias conclusões podem ser extraídas das informações ali expostas.

O grande gargalo do processo de conversão se encontra no carregamento e armazenamento das informações, chegando no pior caso apresentado a 50% do tempo total. Observe que os sólidos compostos por primitivas não planas apresentam um tempo de IO superior ao obtido para sólidos que contenham apenas primitivas planas. Isto ocorre porque as primitivas planas são, em geral, aproximadas por um número significativamente inferior de faces (devido a remoção das arestas redundantes), fato este que agiliza o armazenamento do modelo BRep. Para superarmos este obstáculo sugerimos a substituição da aproximação linear por um modelo mais elaborado capaz de manipular faces genéricas, permitindo desta forma, que a remoção de arestas redundantes possa abranger todas as primitivas do modelo.

Sólido	Nível da subdivisão	Tempo de subdivisão (min)	Tempo de conversão (min)	Tempo total (min)
Cadeira	3	00:01:10	00:00:80	00:01:90
	4	00:01:70	00:06:08	00:07:78
	5	00:03:90	00:23:52	00:27:42
	6	00:10:90	01:30:00	01:40:90
Cálice	3	00:02:20	00:02:77	00:04:97
	4	00:09:80	00:18:87	00:28:67
	5	00:54:80	01:25:86	02:20:66
Icosaedro	3	00:01:50	00:04:15	00:05:65
	4	00:03:10	00:14:65	00:17:75
	5	00:07:50	00:51:88	00:59:38
	6	00:21:20	02:49:16	03:10:36
Pluft	3	00:03:00	00:02:88	00:05:88
	4	00:10:50	00:13:50	00:24:00
	5	00:55:20	01:02:18	01:57:38
	6	05:18:50	05:03:95	10:22:45

Tabela 7.1 – Tempos de execução do procedimento de conversão

Embora o tratamento geométrico do modelo não represente altos custos, acreditamos que muito ainda há a ser aperfeiçoado neste sentido. O processo de propagação da informação geométrica discutido na seção 5.4 e por nós implementado deixa a desejar, no sentido de que os sinais dos vértices das células da subdivisão devem sempre ser computados quando, na verdade, esta informação poderia ser recuperada das células vizinhas. Entretanto, vale lembrar que, com a nossa estratégia somos capazes de garantir a ausência de erros numéricos que venham a causar danos ao procedimento de conversão (seção 5.4).

Como era de se esperar, o gerenciamento da topologia do modelo ocupa uma pequena parcela do tempo de conversão. Isto se dá pelo fato do modelador adotado estar construído sobre uma estrutura de dados de fácil manutenção e que gera representações com um mínimo de redundância.

Sólido	Nível	IO	Geometria	Topologia	Memória	Outros
Cadeira	5	31%	11%	14%	24%	20%
Cálice	5	45%	7%	13%	19%	16%
Icosaedro	5	34%	10%	16%	22%	18%
Pluft	5	50%	14%	12%	12%	12%

Tabela 7.2 – Percentuais de tempo gastos em cada etapa da conversão

Quanto ao tempo utilizado no gerenciamento de memória, tivemos a oportunidade de constatar na seção 6.4 que um tratamento mais cuidadoso pode resultar em grandes economias. A necessidade destes cuidados surge devido a fragmentação das áreas de memória disponíveis quando rotinas de alocação e desalocação de memória são utilizadas em volumosa quantidade para estruturas de diferentes tamanhos. Esta fragmentação, como é sabido, toma moroso o processo de alocação de novas áreas de trabalho.

Sólido	Nível	Num Faces Ideal	Num Faces Geradas	N Faces s/ aresta red.
Cadeira	4	31	2700	2232
	5	31	11224	4783
Cálice	4	7	9644	7247
	5	7	40304	29219
Icosaedro	4	20	5472	4496
	5	20	20808	10112
Pluft	4	11	4568	4568
	5	11	18928	18928

Tabela 7.3 – AFalta de Concisão do modelo

Os dados contidos na tabela 7.3 mostram que nossa estratégia de conversão gera um modelo extremamente deficiente com relação a concisão. Isto se dá, principalmente, pela forma com que computamos o BRep no interior de um nó da árvore OctreeCsg. Este cálculo considera complexo qualquer nó que contenha mais de uma primitiva CSG. Assim sendo, nós contendo arestas do sólido (interseção de duas de suas primitivas) ou vértices do mesmo, são classificados como complexos e as faces BRep ali geradas (que podem atingir o limite de 6 por cada nó) não serão simplificadas através do procedimento

de remoção de arestas redundantes. Por este motivo, mesmo quando operamos com primitivas planas, encontraremos um elevado número de faces nas redondezas das arestas e vértices do sólido original. Contudo, rotinas mais elaboradas tratando estas particularidades podem ser desenvolvidas e incluídas no procedimento proposto. Para tanto, basta que estas atendam as propriedades *(i)* e *(ii)* listadas no início da seção 5.5. Obviamente, a adoção do uso de faces planas, ao invés de faces genéricas, contribui expressivamente para a falta de concisão do modelo quando as primitivas existentes não forem lineares.

Note ainda, que a falta de concisão do modelo é um fator que influencia diretamente no tempo gasto para o seu armazenamento. Um modelo pouco conciso acarreta, inclusive, em dificuldades de armazenamento em memória. Em testes com sólidos contendo primitivas não lineares as dificuldades surgiram a partir do nível 6 de subdivisão, onde é comum se obter representações ocupando mais de 2 milhões de bytes.

No apêndice *C* o leitor irá encontrar exemplos de sólidos e primitivas CSG submetidos ao processo de conversão. Buscamos com isto ilustrar o desempenho do algoritmo proposto ao lidar com as mais diversas primitivas.

7.3. FINALIZAÇÃO

Lançamos aqui a semente de uma idéia que se mostra promissora no âmbito da modelagem de sólidos. Como vimos, muito ainda há a ser desenvolvido para que se chegue a resultados que tomem esta idéia prática e eficaz. Esperamos entretanto, que com nosso esforço inicial tenhamos reunido informações e experiências necessárias a execução desta tarefa.

Referências Bibliográficas

[BAUM74] BAUMGART, B., "A Polyhedron Representation for Computer Vision", tese de doutorado, Stanford University, 1974

[BLOO88] BLOOMENTHAL, J., "Polygonization of Implicit Surfaces", Computer Aided Geometric Design, Maio de 1988

[BRUN85] BRUNET, P., NAVAZO, I., "Geometric Modelling Using Exact Octree Representation of Polyhedral Objects", Anais do EUROGRAPHICS 1985

[COMB91] COMBA, J. L. D., "Acompanhamento de Raios Otimizado para Sólidos CSG, COPPE/UFRJ, Dissertação de Mestrado, Engenharia de Sistemas e Computação, Abril de 1991

[ESPE90] ESPERANÇA, C., "Técnicas para Visualização Direta de Modelos CSG, COPPE/UFRJ, Dissertação de Mestrado, Engenharia de Sistemas e Computação, Junho de 1990

[FONS92] FONSECA, A., "LCGRENDERER: Um Visualizador de Modelos Baseado em PHIGS", IM/UFRJ, Projeto Final do Curso de Graduação em Informática, 1992

[GARG82] GARGANTINI, I., "Linear Octrees for Fast Processing of Three-Dimensional Objects", Computer Graphics and Image Processing, Vol. 20, pp. 365–374, 1982

[GUIB85] GUIBAS, L., STOLFI, J., "Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams", ACM Transactions on Graphics, Vol. 4, no. 2, pp. 74–123, Abril de 1985

[HIRO85] HIROAKI, C., FUMIHIKO, K., "A Method of Representing the Solid Design Process", IEEE CG&A, Vol. 5, no. 4, pp. 32–41, Abril de 1985

[HOPC69] HOPCROFT, J. E., ULLMAN, J. D., "Formal Languages and Their Relation To Automata", Addison–Wesley Publishing Company, 1969

[HUNT78] HUNTER, G. M., "Efficient Computation and Data Structures for Graphics", PhD thesis, department of Electrical Engineering and Computer Science, Princeton University, Princeton, N. J., 1978

[JACK80] JACKINS, C. L., TANIMOTO, S. L., "Oct–Trees and their use in Representing Three–Dimensional Objects", Computer Graphics and Image Proccessing, Vol. 14, pp. 249–270, 1980

[JAME55] JAMES, R. C., "Combinatorial Topology of Surfaces", Mathematics Magazine, vol. 29, pp. 1–39, 1955

[JUAN88] JUAN, R., "Boundary to Constructive Solid Geometry: A Step Towards 3D Conversion", Anais do EUROGRAPHICS 1988, pp. 129–139

[KAWA80] KAWAGUCHI, E., ENDO, T., "On a Method of Binary Picture Representation and its Application to Picture Compression", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 2, no. 1, pp. 27–35, 1980

[KUNI85] KUNII, T. L., SATOH, T., YAMAGUSHI, K., "Generation of Topological Boundary Representations from Octree Encoding", IEEE CG&A, Vol. 5, no. 3, pp. 29–38, Março de 1985

[LIMA85] LIMA, E. L., "Curso de Análise", Vol. 2, Livros Técnicos e Científicos Editora S. A., 1985

[LIMA87] LIMA, E. L., "Curso de Análise", Vol. 1, Livros Técnicos e Científicos Editora S. A., 1987

[MANT82] MANTYLA, M., SULONEN, R., "GWB: A Solid Modeler with Euler Operators", IEEE CG&A, Vol. 2, no. 7, pp. 17–31, Setembro de 1982

[MANT84] MANTYLA, M., "A Note on the Modeling Space of Euler Operators", Computer Vision, Graphics and Image Processing, Vol. 26, no. 1, pp. 45–60, Janeiro de 1983

[MANT88] MANTYLA, MARTTI, "An Introduction to Solid Modeling", Computer Science Press, 1988

[MEAG80] MEAGHER, D., "Octree Encoding: A New Technique for the Representation, Manipulation, and Display of Arbitrary Three–Dimensional Objects by Computer", Technical Report IPL–TR–80, 111, Image Processing Laboratory, Rensselaer Polytechnic Institute, Troy, New York, Outubro de 1980

[MEAG82] MEAGHER, D., "Geometric Modeling Using Octree Encoding", Computer Graphics and Image Processing, Vol. 19, pp. 129–147, 1982

[MIRA89] MIRANDA, J., TAVARES, G., "Métodos Simpliciais em Computação Gráfica", 17º Colóquio Brasileiro de Matemática, Instituto de Matemática Pura e Aplicada do CNPq, 1989

[NAVA87] NAVAZO, I., FONTDECABA, J., BRUNET, P., "Extended Octtrees, Between CSG Trees and Boundary Representations", Anais do EUROGRAPHICS 1987, pp. 239–247

[PAOL89] PAOLUZZI, A., RAMELLA, M., SANTARELLI, A., "Boolean Algebra Over Linear Polyhedra", Computer Aided Design, Vol. 21, no. 8, pp. 474–484, Outubro de 1989

[PERS91] PERSIANO, R. C. M., "Manifold Data Structure and B–Rep Solid Models", Palestra ministrada no evento 'Workshop on Geometric Modeling" realizado no IMPA/Rio de Janeiro, Janeiro de 1991

[REQU77A] REQUICHA, A. A. G., VOELCKER, H. B., "Constructive Solid Geometry", Tech. Memo, No. 25, Production Automation Project University of Rochester, 1977

[REQU77B] REQUICHA, A. A. G., VOELCKER, H. B., "Mathematical Models of Rigid Solids", Tech. Memo, No. 28, Production Automation Project University of Rochester, 1977

[REQU80] REQUICHA, A. A. G., "Representations for Rigid Solids: Theory, Methods, and Systems", Computing Surveys, Vol. 12, no. 4, pp. 437–464, Dezembro de 1980

[REQU82] REQUICHA, A. A. G., VOELCKER, H. B., "Solid Modeling: A Histoncal Summary and Contemporary Assessment", IEEE CG&A, Vol. 2, no. 2, pp. 9–24, Março de 1982

[REQU85] REQUICHA, A. A. G., VOELCKER, H. B., "Boolean Operations in Solid Modeling: Boundary Evaluation and Merging Algorithms", Proceedings of the IEEE, Vol. 73, no. 1, pp. 30–44, Janeiro de 1985

[ROSS89] ROSSIGNAC, J. R., VOELCKER, H. B., "Active Zones in CSG for Accelerating Boundary Evaluation, Redundancy Elimination, Interference Detection, and Shading Algorithms", ACM Transactions on Graphics, Vol. 8, no. 1, pp. 51–87, Janeiro de 1989

[ROSS91] ROSSIGNAC, J. R., REQUICHA, A. A. G., "Constructive Non–Regularized Geometry", Computer Aided Design, Vol. 23, no. 1, pp. 21–32, Janeiro de 1991

[SALI91] SALIM M., BUENO L. P., PERSIANO R C. M., "Aproximação Linear por Partes de Variedades Implicitamente Definidas Usando Poda do Espaço", Anais do SIBGRAPI 91, Julho de 1991

[SALI92] SALIM M., "Aproximação Linear por Partes de Sólidos CSG através de Subdivisão Simplicial Adaptativa", COPPE/UFRJ, Dissertação de Mestrado, Engenharia de Sistemas e Computação, 1992

[SAME90] SAMET, H., "The Design and Analysis of Spatial Data Structures", AddisonWesley Publisshing Company, Inc., 1990

[SZWA84] SZWARCFITER, J. L., "Grafos e Algoritmos Computacionais", Editora Campus Ltda., 1984

[TANG88] TANG, Z., LU, S., "A New Algorithm for Converting Boundary Representation to Octree", Anais do EUROGRAPHICS 1988, pp. 105–116

[TILO80] TILOVE, R. B., REQUICHA, A. A. G., "Closure of Boolean Operations on Geometric Entities", Computer Aided Design, Vol. 12, no. 5, pp. 219–220, Setembro de 1980

[TILO84] TILOVE, R. B., "A Null–Object Detection Algorithm for Constructive Solid Geometry", Communications of the ACM, Vol. 27, no. 7, pp. 684–694, Julho de 1984

[VOEL77] VOELCKER, H. B., REQUICHA, A. A. G., "Geometric Modeling of Physical Parts and Processes", IEEE Computer, Vol. 10, no. 2, pp. 48–57, 1977

[WYVI86] WYVILL, G., Kunii, T. L., "Space Division for Ray Tracing in CSG, IEEE CG&A, Abril de 1986

[YAMA84] YAMAGUCHI, K., KUNII, T. L., FUJIMURA, K., "Octree–Related Data Structures and Algorithms", IEEE CG&A, Janeiro de 1984

[ZEEM] ZEEMAN, C. E., "Uma Introdução Informal a Topologia das Superfícies", IMPA/CNPq, Monografias de Matemática

Apêndice A

A Implementação da Álgebra de Arestas

A.1. Introdução

Um resultado fundamental obtido com o uso do modelador BRep proposto no capítulo 4 deste trabalho é a eficiência com que o mesmo lida com suas representações. Esta propriedade decorre diretamente da simplicidade presente na implementação de sua estrutura de dados, a saber, a *álgebra* de arestas.

Neste apêndice vamos exibir o código em linguagem C da implementação por nós desenvolvida da álgebra de arestas descrita no capítulo 4. Assim, veremos na próxima seção a composição da entidade aresta da estrutura de dados. A seção A.3 exibe o código C correspondente a implementação das funções de arestas. A última seção mostra como são implementados os operadores topológicos *CriaAresta*, *MataAresta* e *AlteraÁlgebra*.

A.2. A Entidade Aresta

A álgebra de arestas tem como único elemento a entidade denominada aresta. Cada aresta da álgebra corresponde a uma aresta do grafo orientado que representa a subdivisão do sólido sendo processado.

Na implementação vista a seguir observa-se que a aresta corresponde a uma estrutura que referencia as próximas arestas de seu ciclo de face esquerda (*ProxFEsq*) e de seu ciclo de vértice origem (*ProxOrg*). Estas duas referências são suficientes para se capturar a topologia do modelo. Além disto, cada aresta possui ponteiros para as informações não topológicas de seu vértice origem (*InfoVertOrg*), da própria aresta (*InfoAresta*) e de sua face esquerda (*InfoFaceEsq*). Estas três informações descrevem a geometria do modelo. Por último, temos os índices da aresta (*ÍndiceAresta*) e do vértice origem (*ÍndiceOrigem*), utilizados pelos procedimentos de leitura e armazenamento de representações. Uma marcação de percurso (*CódigoPercurso*), utilizada pelos procedimentos de percurso em componentes conexas, também faz parte da representação adotada para a entidade aresta.

```
typedef struct TypeAresta *PtrAresta;

typedef  struct  TypeAresta {
    PtrAresta     ProxOrg;
    PtrAresta     ProxFEsq;
    void          *InfoVertOrg;
    void          *InfoAresta;
    void          *InfoFaceEsq;
    int           ÍndiceAresta;
    int           ÍndiceOrigem;
    char          CódigoPercurso;
} TypeAresta;
```

A.3. As Funções de Arestas

Apresentamos aqui a implementação das funções de passeio topológico sobre a álgebra de arestas analisadas no capítulo 4. Elas nada mais são, senão uma sequência finita de acessos consecutivos aos campos *ProxOrg* e *ProxFEsq* da entidade aresta. Ambas recebem uma aresta como parâmetro e retornam uma segunda aresta vizinha a ela.

```
void *AntFDir(Aresta)
PtrAresta Aresta;
{
    refurn ((void *) Aresta→ProxFEsq→ProxOrg→ProxOrg);
}

void *AntFEsq(Aresta)
PtrAresta Aresta;
{
    return ((void *) Aresta+  ProxOrg→ProxFEsq→ProxOrg);
}

void *AntOrg(Aresta)
PtrAresta Aresta;
{
    return ((void *) Aresta→ProxFEsq→ProxOrg→ProxFEsq);
}

void *ProxFDir(Seta)
PtrAresta Aresta;
{
    return ((void *")resta+  ProxFEsq→ProxOrg→ProxFEsq→ProxFEsq→ProxOrg);
}

void *ProxFEsq(Aresta)
PtrAresta Aresfa;
{
    return ((void *) Aresta→ProxFEsq);
}

void *ProxOrg(Aresta)
PtrAresta Aresfa;
{
    return ((void *) Aresta→ProxOrg);
}

void *Simétrica(Aresta)
PtrAresta Aresta;
{
    return ((void *")resta+  ProxFEsq→ProxOrg);
}
```

```
void *AntDest(Aresta)
PtrAresta Aresta;
{
    return ((void *) Aresta→ProxFEsq→ProxFEsq→ProxOrg};
}

void *ProxDest(Aresta)
PtrAresta Aresta;
1
    return ((void *) Aresta→ProxFEsq→ProxOrg→ProxOrg–, ProxFEsq→ProxOrg};
}
```

A.4. Os Operadores Topológicos

Ao longo do capítulo 4 foram discutidos três operadores topológicos, através dos quais é possível criar e modificar qualquer álgebra de arestas que atenda as propriedades lá descritas. Segue-se a implementação de cada um deles:

```
void *CriaAresta() {
    PtrAresta Aresta, SimAresta;
    (* Alocação de área para a aresta e sua simétrica *)
    Aresta = (PtrAresta) calloc(1, sizeof(TypeAresta));
    SimAresta = (PtrAresta) calloc(1, sizeof(TypeAresta));
    (* Preenchimento dos campos ProxFEsq e ProxOrg *)
    ProxFEsq(Aresta) = SimAresta;
    ProxFEsq(SimAresta) = Aresta;
    ProxOrg(Aresta) = Aresta;
    ProxOrg(SimAresta) = SimAresta;
    return ((void *) Aresta);
}

void MataAresta(Aresta)
PtrAresta Aresta;
{
    (* Só fem efeito se a aresta parâmetro possuir as características de uma aresfa gerada pelo CriaAresta *)
    if ( (ProxFEsq(Aresta) == Simétrica(Aresta)) && (ProxFEsq(Simétrica(Aresta)) == Aresta) ) (
        free(Simétrica(Aresta));
        free(Aresta);
    }
}
```

```
void AlteraÁlgebra(ArestaA, ArestaB)
PtrAresta ArestaA, ArestaB;
{
    PtrAresta ArestaAux1, ArestaAux2, ArestaAux3;
    ("Alguns valores são armazenados em variáveis auxiliares *)
    ArestaAux1 = AntFEsq(ArestaA);
    ArestaAux2 = AntFEsq(ArestaB);
    ArestaAux3 = ProxFEsq(ArestaAux1);
    ("Alterações topológicas dos ciclos de faces esquerdas *)
    ProxFEsq(ArestaAux1) = ProxFEsq(ArestaAux2);
    ProxFEsq(ArestaAux2) = ArestaAux3;
    ("Alterações topológicas dos ciclos de vértices origens *)
    ArestaAux3 = ProxOrg(ArestaA);
    ProxOrg(ArestaA) = ProxOrg(ArestaB);
    ProxOrg(ArestaB) = ArestaAux3;
}
```

Apêndice B

Instruções para a Execução dos Programas

B.1. Introdução

Diversos programas foram desenvolvidos e outros, já existentes, foram aproveitados para que o procedimento proposto de conversão de representações CSG em BRep fosse posto em prática. Estes programas se encontram escritos em linguagem C ou *MODULA II* e foram desenvolvidos em ambiente *UNIX (SUNOS* 4.1) nas estações gráficas *SUN 260, 3/60,* SPARC 1+ e SPARC 2+. Vamos a seguir enumerar cada um deles, passando posteriormente a uma descrição minuciosa de suas sintaxes e dos formatos de arquivos por eles utilizados.

- **CompilaCsg**

É um compilador da *Linguagem* de *Definição* de Sólidos *(LDS),* que constrói a representação CSG do objeto por ela descrito.

- **GeraOCsg e csg2octree**

Programas cuja finalidade é aplicar a subdivisão espacial na construção da estrutura híbrida *OctreeCSG.*

- **octree2brep**

Gera a representação BRep a partir dos dados fornecidos pela OctreeCSG ou csg2octree.

- **interface e lcgrender**

Programas de visualização da representação BRep obtida.

Os programas citados acima podem ser executados independentemente ou em *pipeline* (excetuando-se neste último caso, os programas de visualização). A figura B.1 ilustra a ordem com que eles devem ser executados. Passamos agora a descrição pormenorizada de cada um dos programas mencionados.

B.2. O Compilador CompilaCsg

O usuário descreve o sólido desejado em um arquivo cuja terminação padrão é ".LDS". Para tal e utilizada a linguagem LDS. Este arquivo é compilado pelo programa CompilaCsg, o que resulta na criação de um segundo arquivo, contendo a representação CSG do sólido idealizado. A terminação padrão deste arquivo de saída é ".CSG".

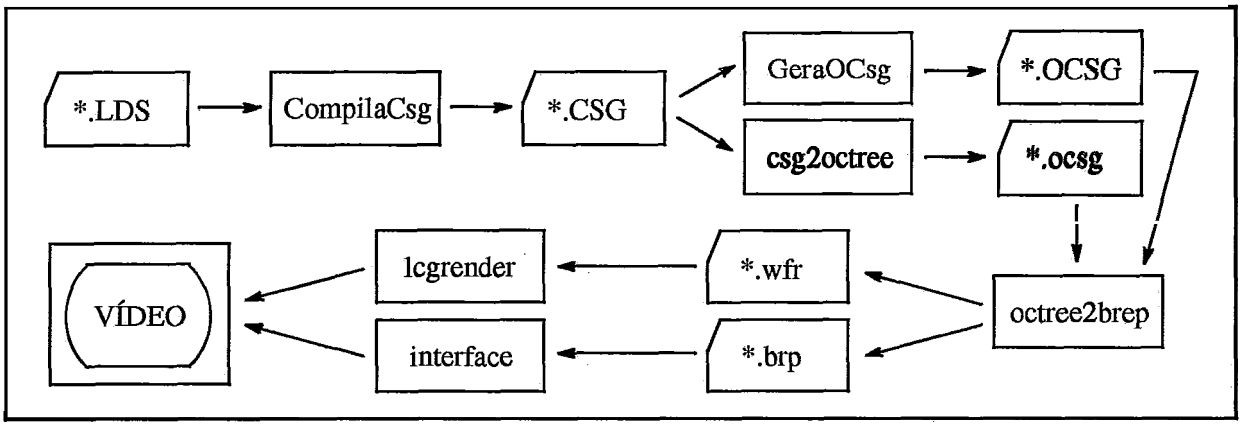


Figura B.1 – O processo de conversão CSG → BRep

A sintaxe de execução do CompilaCsg, a descrição da linguagem LDS e do arquivo de saída gerado são encontradas na referência [ESPE90]. A simples execução do comando *CompilaCsg* isoladamente também fornecerá a sua sintaxe.

A título de exemplo, listamos abaixo o conteúdo do arquivo *Cadeira.LDS* com a descrição do sólido *Cadeira* e, a seguir, apresentamos o comando dado para compilá-lo:

```
Cena {
    Assento {
        ESCALA [1.1, 0.2, 1] BLOCO
    }
    Encosto {
        ESCALA [0.2, 1.1, 1] TRANSL[0, 0.19, 0] BLOCO
    }
    Pe {
        ESCALA [0.2, 0.8, 0.2] BLOCO
    }
    Pes {
        TRANSL[0,-0.8,0] Pe + TRANSL[0.9,-0.8, 0] Pe + TRANSL[0,-0.8,0.8] Pe + TRANSL[0.9,-0.8,0.8] Pe
    }
    Cadeira {
        Assento + Encosto + Pes
    }
    MATERIAL [5] Cadeira
}
```

Comando de compilação: *CompilaCsg Cadeira*

B.3. O Programa GeraOCsg

Uma vez de posse do arquivo com a representação CSG do sólido (*.CSG), este deve ser processado pelo programa GeraOCsg. Este processamento resulta na criação de um segundo arquivo contendo a estrutura híbrida OctreeCSG, obtida a partir de um processo de subdivisão recursiva aplicado sobre a árvore CSG que representa o sólido.

A sintaxe de execução do programa pode ser obtida a partir de sua execução, isto é, do comando *GeraOCsg*. Ela também pode ser encontrada, juntamente com os formatos dos arquivos de entrada e saída, na referência [COMB91]. Por ter sido substancialmente modificado desde a concepção inicial do programa, o formato do arquivo .OCSG será especificado com maiores detalhes mais adiante.

Um parâmetro e um *flag* merecem nossa especial atenção. O parâmetro vem a ser o que define o número máximo aceitável de primitivas CSG internas a um octante para que este possa ser classificado como simples, interrompendo o processo de subdivisão. O valor atribuido a este parâmetro é necessariamente *zero* pois, com isto, todos os nós terminais da árvore OctreeCSG estarão situados no mesmo nível da subdivisão (no nível máximo definido). Esta propriedade é exigida, não pela estratégia de conversão proposta, mas sim pela implementação desenvolvida. O parâmetro mencionado equivale a *string* "*-P0*" a ser inserida na linha de comando.

Também por restrição imposta pela implementação desenvolvida, toma-se obrigatório a ativação do *flag* "*-E*". Isto se da pelo fato de nossos programas lidarem com expansões das primitivas compostas. Em outras palavras, toda primitiva constituída por mais de um semiespaço deve ser extendida de modo a ser representada por uma árvore CSG formada pelos seus diversos semiespaços. A figura **B.2** ilustra o processo de expansão da primitiva CILINDRO obtida com a ativação do *flag* citado.

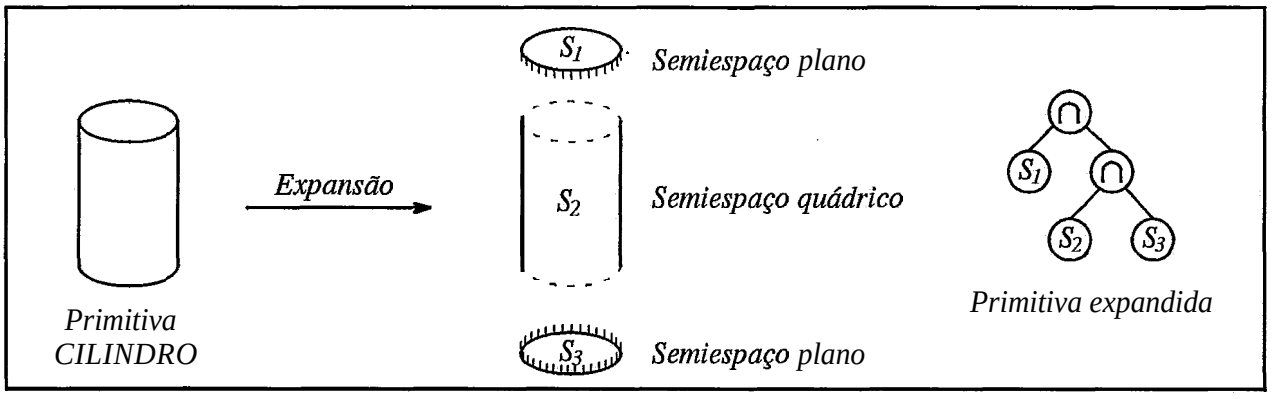


Figura B.2 – Expansão de primitivas compostas

Para que a saída do programa GeraOCsg possa ser utilizada pelo restante do processo de conversão, devemos ter a seguinte sintaxe mínima, onde *nomearq* corresponde ao nome do arquivo de saída do compilador CompilaCsg:

GeraOCsg -P0 -E <nomearq>

Em seu estado atual, o programa GeraOCsg é capaz de lidar com as primitivas PLANO, BLOCO, ESFERA, CILINDRO, CONE, TORO e HELICÓIDE. Dando prosseguimento ao exemplo do sólido *Cadeira* iniciado na seção anterior, o arquivo *Cadeira.CSG*, obtido através da execução do programa CompilaCsg, é agora submetido ao programa GeraOCsg através do seguinte comando (usando nível 4 de subdivisão):

GeraOCsg -O-0.1,-0.9,-0.1 -L2.3 -P0 -D4 -E Cadeira

B.2.1. O Formato do Arquivo de Saída (*.OCSG)

O arquivo de saída do programa GeraOCsg, contendo as informações a respeito da estrutura híbrida OctreeCSG, é gerado pela gramática abaixo:

<arq .OCSG> ::= <nprimitivas> <lista de primitivas> <origem octreecsg> <largura octreecsg> <octreecsg>

<nprimitivas> ::= *inteiro* ≥ 0

<lista de primitivas> ::= <primitiva>$_1$ <primitiva>$_2$ ... <primitiva>$_{nprimitivas}$

<primitiva> ::= *<material> <tipo> <matriz localcena> |*
<material> <tipo> <raio menor do toro> <matriz localcena>

<material> ::= *inteiro* > 0

<tipo> ::= *BLOCO = 0 (byte) | ESFERA = 1 (byte) | CILINDRO = 2 (byte) | PLANO = 3 (byte) |*
CONE = 4 (byte) | TORO = 5 (byte) | HELICÓIDE = 7 (byte)

<matriz localcena> ::= *matriz 4x4 de reais (floats)*

<raio menor do toro> ::= *real* > 0

<origem octreecsg> ::= *3 reais*

<largura octreecsg> ::= *real*

<octreecsg> ::= <octante>$_1$ <octante>$_2$ ... <octante>$_8$

<octante> ::= *<octante universo> | <octante vazio> | <octante misto> <octreecsg> |*
<octante primitiva> <índice árvore Csg> <num primitivas Csg> <árvore Csg>

<octante universo> ::= *0 (byte)*

<octante vazio> ::= *1 (byte)*

<octante misto> ::= *2 (byte)*

<octante primitiva> ::= *3 (byfe)*

<índice árvore Csg> ::= *inteiro* ≥ 0

<num primitivas Csg> ::= inteiro > 0

<árvore Csg> ::= <nó operação> <operação> <árvore esq> <árvore dir> |
<nó primitiva> <índice da primitiva>

<nó operação> ::= *0 (byte)*

<operação> ::= *UNIÃO = 0 (byte) | INTERSEÇÃO = 1 (byte) | DIFERENÇA = 2 (byfe)*

<árvore esq> ::= *<árvore dir>* ::= *<árvore Csg>*

<nó primitiva> ::= *1 (byfe)*

<índice da primitiva> ::= *inteiro*

Se *<índice da primitiva>* for negativo, isto significa que deve ser tomado o complemento da primitiva indicada pelo módulo do valor lido. A sintaxe de *<primitiva>* vai depender do tipo da mesma (*<tipo>*), no caso de termos um TORO, seu raio menor deve ser indicado.

B.3. O Programa csg2octree

Devido a alguns problemas decorrentes de erros numéricos, e também para que pudéssemos testar nosso processo de conversão para um conjunto mais abrangente de primitivas, desenvolvemos o programa *csg2octree.* Este tem a mesma finalidade do GeraOCsg discutido anteriormente. Ou seja, a partir do arquivo de saída do programa CompilaCsg, ele gera um segundo arquivo contendo uma árvore OctreeCSG, obtida através de um processo rudimentar. Este processo consiste na subdivisão exaustiva do espaço até que todos os octantes se encontrem no nível máximo da subdivisão. Só neste ponto a árvore CSG inicial é podada com relação a cada octante.

Com o uso do csg2octree, temos disponível, além das primitivas usadas pelo GeraOCsg, primitivas polinomiais ou primitivas definidas por quaisquer conjunto de desigualdades. A sintaxe do programa csg2octree é dada a seguir:

csg2octree {-D} {-N nível} {-O origem] {-L largura] < arq_entrada > arq_saída

D: flag que executa o programa no modo default
nível: nível máximo de subdivisão (default = 3)
origem: três reais separados por espaços indicando a origem da octree (default = -1.1 -1.1 -1.1)
largura: três reais separados por espaços indicando a largura da octree (default = 2.2 2.2 2.2)
arq_entrada: nome do arquivo (incluindo sua terminação) com a definição da cena CSG (default = sfdin)
arq_saída: nome do arquivo (incluindo sua terminação) de saída contendo a OctreeCsg (default = sfdout)

Para o exemplo do sólido *Cadeira* teremos a seguinte linha de comando:

csg2octree -O -0.1 -0.9 -0.1 -L 2.3 2.3 2.3 -N4 < Cadeira.CSG > Cadeira.ocsg

B.3.1. O Formato do Arquivo de Saída (*.ocsg)

Optamos por um formato de saída ligeiramente distinto do formato seguido pelo programa GeraOCsg. Isto se deu pelo fato de termos em mãos dados que, se não aproveitados, teriam de ser novamente computados na etapa seguinte do procedimento de conversão. Para diferenciar os dois tipos de arquivos contendo árvores OctreeCsg sugerimos o uso da terminação ".ocsg" para os arquivos de saída do programa csg2octree. As modificações impostas na gramática de formação do arquivo de saída são as seguintes:

Ao invés de <*matriz localcena*>, vamos trabalhar com <*matriz cenalocal*>, que é uma matriz 4x4 de reais, que armazena os dados da transformação a ser efetuada para se levar um ponto da cena ao espaço das primitivas.

A largura da octree <*largura octreecsg*> é agora dada por 3 reais, o que permite que seus eixos possam ter dimensões distintas.

O acréscimo da primitiva POLINOMIAL (byte = 6), faz com que seja necessário a definição de seu grau e a enumeração de seus coeficientes. Com isto, a sintaxe de

<primitiva> é alterada para suportar esta modificação. Na nova sintaxe, NCoefs corresponde ao número de coeficientes da polinomial de grau dado por <grau da polinomial>:

<primitiva>:= <material> <tipo> <matriz localcena> |
<material> <tipo> <raio menor do toro> <matriz localcena> |
<material> <tipo> <grau da polinomial> <coefs da polinomial> <matriz localcena>

<grau da polinomial>:= inteiro r 0

<coefs da polinomial>:= NCoefs reais

Os campos <índice árvore Csg> e <num primitivas Csg> deixam de ser definidos por não serem necessários ao restante do procedimento de conversão. Deste modo, <octante> passa a ser definido como:

<octante> ::= <octante universo> | <octante vazio> | <octante misto> <octreecsg> |
<octante primitiva> <árvore Csg>

A árvore Csg interna a cada octante da OctreeCSG tem, durante a execução do csg2octree, os nós contendo operações de INTERSEÇÃO transformados, de modo que a árvore passe a conter apenas os operadores UNIÃO e DIFERENÇA. Portanto teremos:

<operação> := UNIÃO = 0 (byfe) | INTERSEÇÃO = 1 (byte)

B.4. O Programa octree2brep

A etapa final do processo de conversão implementado e a construção da representação BRep. Isto é feito partindo-se do arquivo contendo a árvore OctreeCSG, que pode ter sido gerada pelos programas GeraOCsg ou csg2octree. O octree2brep constrói uma representação por fronteiras utilizando-se de técnicas de aproximação linear. Com isto, a fronteira do sólido idealizado é aproximada por faces planas.

Existem dois possíveis formatos de saída gerados pelo octree2brep. O primeiro deles, visa uma representação integral de todas as propriedades geométricas e topológicas do modelo. Padronizamos a terminação de tais arquivos com o sufixo ".brp". O segundo formato de arquivo de saída busca apenas representar a geometria do modelo e é utilizado, por exemplo, pelo visualizador lcgrender. Sua terminação padrão é dada por ".wfr". Estes dois formatos serão detalhados nas seções vindouras.

A sintaxe de execução do octree2brep é exibida a seguir:

octree2brep {-E tipoe} {-S tipos} {-I} {-A} < arq_entrada > arq_saida

tipoe: se = C, o arquivo de enfrada foi gerado pelo csg2octree, se = G, pelo GeraOCsg (default = C)
tipos: se = B, o arquivo de saída terá formato ".brp", se = L, formato ".wfr" (default = B)
I: "flag" que ativa o aproveitamento de componentes incompletas
A: "flag" que ativa a remoção de arestas redundantes
arq_entrada: nome do arquivo (incluindo sua terminação) com a definição da OctreeCSG (default = stdin)
arq_saída: nome do arquivo de saída (incluindo sua terminação) contendo a cena BRep (default = stdout)

Para gerarmos a representação BRep final do sólido Cadeira sendo exemplificado até então, executamos o comando a seguir, assumindo que o arquivo com a OctreeCSG foi gerado pelo programa csg2octree e ativando a eliminação de arestas redundantes. A saída desejada é no formato ".brp":

octree2brep -EC -SB -A < Cadeira.ocsg > Cadeira.brp

B.4.1. O Formato do Arquivo de Saída (*.brp)

Vamos aqui exibir a gramática do arquivo de saída contendo as informações geométricas e topológicas da aproximação por faces planas da fronteira do sólido idealizado. Este formato é o adotado pelo visualizador *interface* a ser visto na seção B.5.

Cada sólido da gramática corresponde a uma componente conexa do modelo. O *status* do sólido serve para indicar se a componente conexa, em sua configuração final, não possuia bordos (*COMPLETO*) ou se houve a necessidade de substituir seus bordos por faces sem geometria associada, obtendo assim, uma superfície sem bordos.

Na descrição de cada face, o índice da primitiva tem por finalidade indicar se a face foi gerada por apenas uma primitiva CSG (caso em que o índice possui valor > 0) ou se dela fazem parte várias primitivas (índice = 0).

<arquivo .brp> ::= <nsólidos> <lista de sólidos>

<nsólidos> ::= inteiro ≥ 0

<lista de sólidos> ::= <sólido>$_1$ <sólido>$_2$ ... <sólido>$_{nsólidos}$

<sólido> ::= <status> <nvértices> <narestas> <nfaces> <lista de vértices> <lista de faces>

<status> ::= INCOMPLETO = 0 (byte) | COMPLETO = 1 (byte)

<nvértices> ::= <narestas> ::= <nfaces> ::= inteiro $\geq$ *0*

<lista de vértices> ::= <vértice>$_1$ <vértice>$_2$ ... <vértice>$_{nvértices}$

<vértice> ::= 3 reais (floats) indicando as coordenadas tridimensionais (x,y,z) do vértice

<lista de faces> ::= <face>$_1$ <face>$_2$ ... <face>$_{nfaces}$

<face> ::= <material> <índice primitiva> <lista de arestas>

<material> ::= <índice primitiva> ::= inteiro $\geq$ *0*

<lista de arestas> ::= <aresta> <lista de arestas> | <marca fim de face>

<aresta> ::= <índice aresta> <índice origem,

<índice aresta> ::= inteiro > 0

<índice origem> ::= inteiro ≥ 0

<marca fim de face, ::= 0 (inteiro)

B.4.2. O Formato do Arquivo de Saída (*.wfr)

O segundo formato de arquivo de saída gerado pelo programa octree2brep se preocupa basicamente com as informações geométricas do modelo. Muito embora, mediante custos elevados, sua topologia possa ser reconstituida. Este formato é adotado pelo visualizador *lcgrender*.

A gramática dos arquivos ".wfr" é apresentada a seguir. Nela, *<num info>* indica o número de informações associadas aos vértices, como por exemplo suas normais. Em nosso caso seu valor é zero.

<arquivo .wfr> ::= <nvértices> <num info> <lista de vértices> <nfaces> <lista de faces>

<nvértices> ::= inteiro r *0*

<num info> = 0 (inteiro)

<lista de vértices> ::= <vértice>$_1$ *<vértice>*$_2$ *... <vértice>*$_{nvértices}$

<vértice> = 3 reais (floafs) contendo as coordenadas (x,y,z) do vértice

<nfaces> = inteiro ≥ 0

<lista de faces> = <nvertface> <índice vértice>$_1$ *<índice vértice>*$_2$ *... <índice vértice>*$_{nvertface}$

<nvertface> = inteiro > 0

<índice vértice> = inteiro ≥ 0

B.5. O Visualizador interface

O último dos programas a ser comentado é o visualizador responsável por exibir as representações BRep obtidas através do programa octree2brep. O programa *interface* carrega e exibe uma representação BRep armazenada no formato descrito em **B.4.1.**

A janela iterativa do programa interface, ilustrada na figura B.3, se encontra dividida em 3 regiões. A região inferior direita é usada para mensagens ao usuário e também para a entrada de dados. A região superior direita vem a ser a janela gráfica onde são exibidas as representações BRep. Na região esquerda vamos encontrar os comandos disponíveis ao usuário, acionáveis via *mouse*. Cada um destes comandos será agora analisado:

Ao ser pressionado o comando "*Sólido*", o usuário verá na janela de mensagens o número de componentes conexas e indicará qual o índice da componente desejada.

O comando "*Topologia*" fornece o número de vértices, arestas e faces da componente conexa selecionada, bem como o seu genus.

Para carregar uma representação BRep, aciona-se o botão "*Carrega*" e digita-se o nome do arquivo (e sua terminação ".brp") na janela de mensagens.

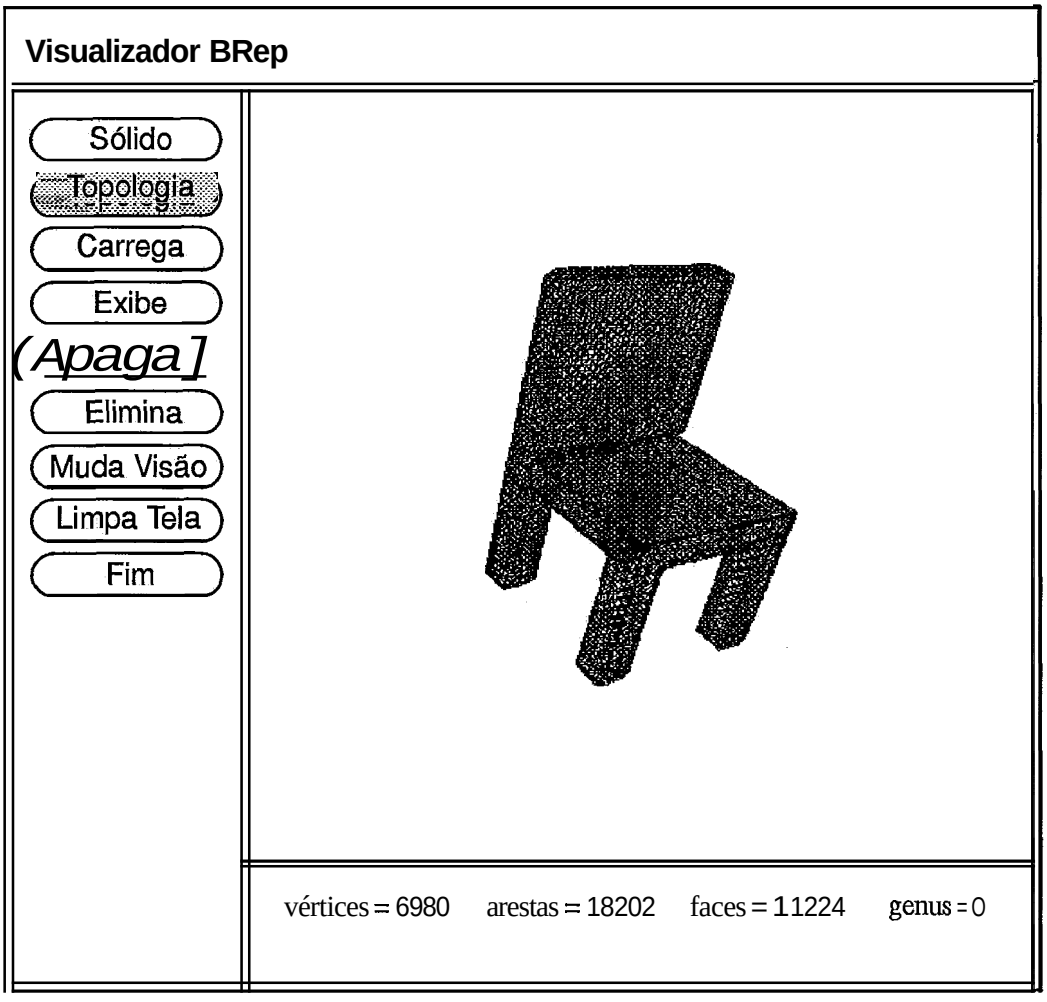


Figura B.3 – A janela iterativa do visualizador "interface"

A exibição da componente selecionada é feita através do comando "Exibe", que faz com que o *wireframe* da representação BRep seja projetada na janela gráfica.

O botão "*Apaga*", apaga da janela gráfica a componente conexa corrente. Esta componente continua a existir e ainda estará selecionada.

A eliminação da componente conexa selecionada é feita com o comando "*Elimina*", que causa a liberação da área de memória por ela ocupada.

O acionamento de "*Muda* Visão" provoca a abertura de um menu com duas alternativas, a saber, Ângulos e *PObs*. A seleção é feita com o uso do *mouse*. Ângulos deve ser escolhido quando se desejar uma visão do objeto rotacionado. Neste caso, devemos fornecer os valores dos novos ângulos de rotação em tomo dos eixos X, Y *e* Z. A opção PObs serve para aproximarmos ou afastarmos o observador do objeto. Um decremento no valor de *PObs* corresponde a um afastamento do observador.

O acionamento do botão "*Limpa* Tela" faz com que a janela gráfica seja completamente varrida, sendo apagadas todas as componentes conexas exibidas.

Ao ser pressionado, o botão "*Fim*" causa o término de execução do visualizador.

Apêndice C

Exemplos de Conversões CSG → BRep

C.1. Introdução

Este apêndice tem por objetivo exibir o efeito do processo de conversão proposto sobre alguns sólidos ou primitivas CSG. As figuras aqui presentes foram obtidas com o uso do vizualizador *lcgrender* [FONS92]. Ao lado de cada figura encontraremos a descrição de seu modelo. O modelo do sólido exposto na figura C.5 foi extraído de [SALI92].

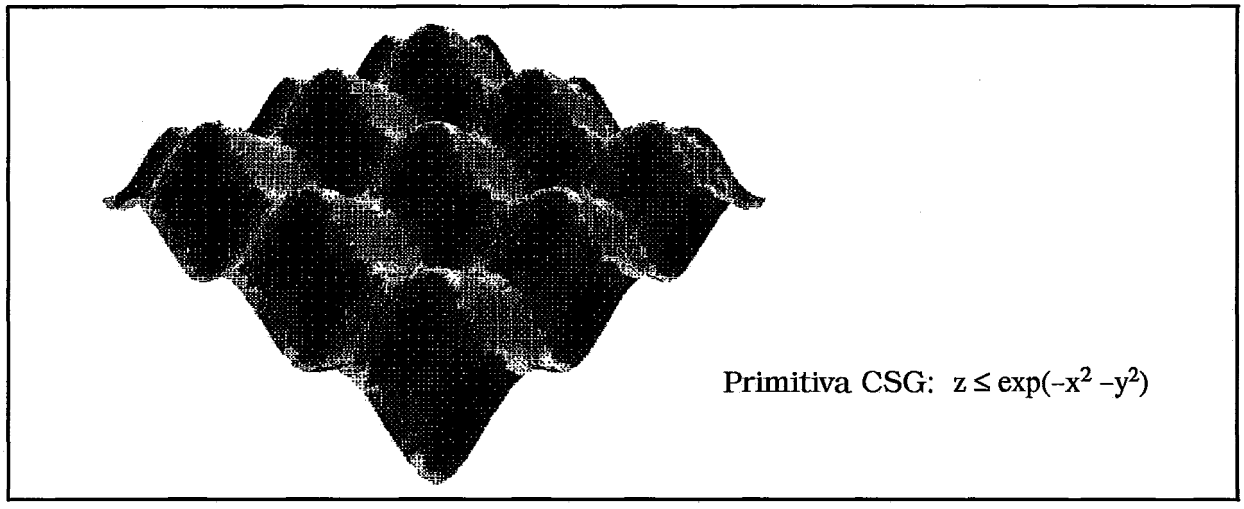


Figura C.1 – A conversão de unta primitiva definida por unta exponencial

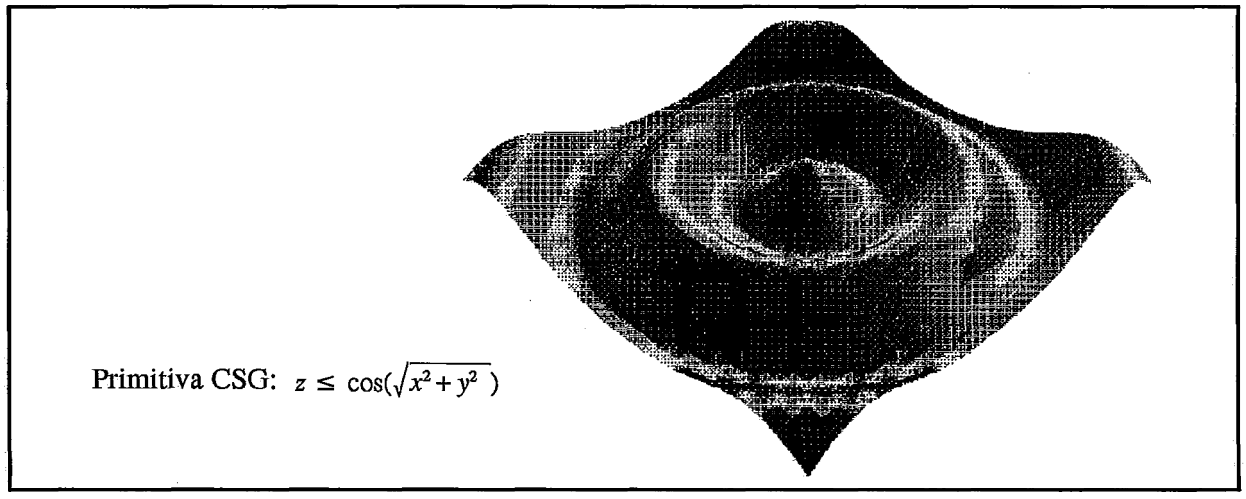
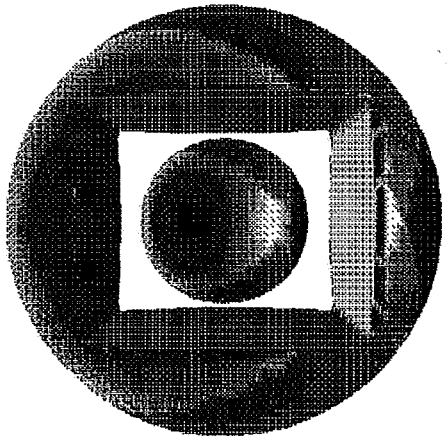


Figura C.2 – A conversão de uma primitiva expressa por função trigonométrica



(Esfera maior \ Bloco) U Esfera menor

Figura C.3 – A conversão do sólido Globo

União de 5 esferas e 4 cilindros

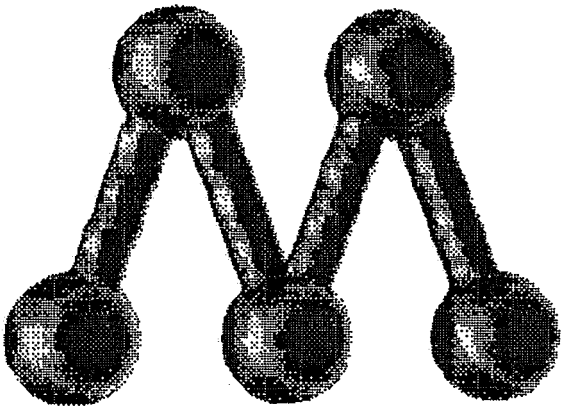
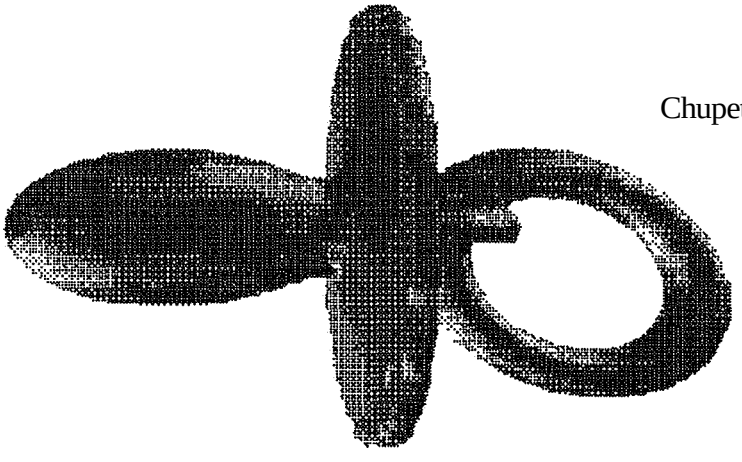


Figura C.4 – A conversão do sólido Manchete



Chupeta = Bico ∪ Base U Presilha U Argola

Bico = Polinomial de grau 3 ∩ Plano
Base = Cilindro
Presilha = Toro ∩ Plano
Argola = Toro

Figura C.5 – A conversão do sólido Chupeta