Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Geometria construtiva de sólidos combinada à representação b-rep

Danilo Balby Silva Castanheira

Dezembro, 2003

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Disciplina: Trabalho de Graduação
Turma: A
Orientador: Prof. Dr. Aluizio Arcela
Período: 2/03
Data: 15/12/2003
Aluno: Danilo Balby Silva Castanheira

GEOMETRIA CONSTRUTIVA DE SÓLIDOS COMBINADA À REPRESENTAÇÃO B-REP

Folha de Aprovação

Autor:
Danilo Balby Silva Castanheira

Título:
Geometria construtiva de sólidos combinada à representação b-rep

Aprovado em:
16/12/2003

Orientador:
Prof. Dr. Aluizio Arcela

Banca examinadora:
Prof. Dr. Aluizio Arcela

Prof. Dr. Márcio da Costa Pereira Brandão

Prof. Dr. Marco Aurélio de Carvalho

Sumário

Lista de Ilustrações

Lista de Algoritmos

Lista de Tabelas

Resumo

As operações booleanas de sólidos são umas das mais eficientes e intuitivas técnicas aplicadas à modelagem 3D, sendo largamente utilizadas por softwares de modelagem atualmente. Várias são as representações de sólidos que suportam as operações booleanas, sendo duas delas a geometria construtiva de sólidos e a representação de fronteira (b-rep). Ambas podem ser combinadas, resultando em uma representação flexível e robusta que faz uso das vantagens que ambas proporcionam. Este trabalho se propõe a descrever uma solução para o problema de se implementar as operações booleanas de sólidos baseada em tal combinação.

Abstract

The boolean set operations are ones of the most efficient and intuitive techniques applied to 3D modelling, being quite used by modelling softwares nowadays. There are many solid representations that support boolean set operations, two of them are the constructive solid geometry and the boundary representation (b-rep). Both can be combined, resulting on a flexible and robust representation that makes use of the advantages provided by them. This work intends to describe a solution for the problem of implementing the boolean set operations based on such combination.

1 INTRODUÇÃO

1.1 Apresentação

Este é um projeto proposto como Trabalho de Graduação no Bacharelado em Ciência da Computação da Universidade de Brasília, durante o segundo período do ano letivo de 2003, sob a orientação do Prof. Dr. Aluizio Arcela, pelo aluno Danilo Balby Silva Castanheira. O projeto foi desenvolvido também ao longo da disciplina Estudos em Computação Multimídia durante o primeiro período de 2003.

1.2 Objetivo

O principal objetivo deste trabalho é descrever uma solução eficiente para o problema de se implementar as operações booleanas de sólidos. Tal descrição deve ainda servir de base para uma implementação que reflita os conceitos expostos.

1.3 As operações booleanas de sólidos

As **operações booleanas de sólidos** (*boolean set operations*) são formas intuitivas e populares de combinar sólidos baseadas nas operações aplicadas a conjuntos. Podem ser submetidas a vários sólidos simultaneamente, mas usualmente o são a apenas dois deles. As três principais são:

- **União** (U): O sólido resultante ocupa todo o volume que os sólidos operandos ocupavam. É comutativa e não gera sólidos vazios, exceto quando os seus operandos também o forem. A figura 1.1(a) apresenta uma possível união de uma esfera e um cilindro;

- **Interseção** (∩): O sólido resultante ocupa o volume dos sólidos operandos que é coincidente a todos eles É comutativa e pode gerar sólidos vazios, o que ocorre quando não há volume coincidente. A figura 1.1(b) apresenta uma possível interseção de uma esfera e um cilindro;

- **Diferença** (-): O sólido resultante ocupa o volume de um dos sólidos operandos que os outros não ocupam. Assim, os operandos têm papéis definidos: ou subtraem ou são

subtraídos. Não é comutativa e não gera sólidos vazios, exceto quando seu "operando subtraído" também o for. As figuras 1.1(c) e (d) apresentam possíveis diferenças de um cilindro por uma esfera e de uma esfera por um cilindro, respectivamente.

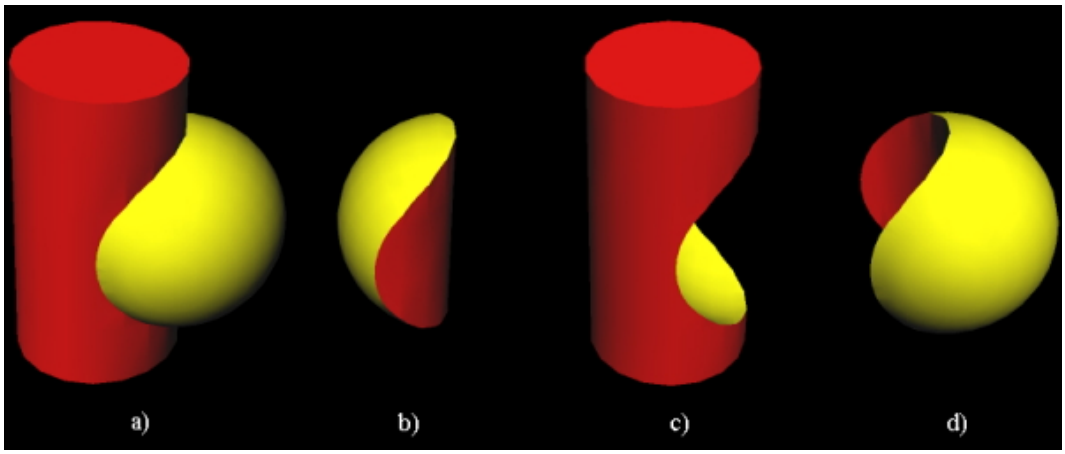

Figura 1.1 Operações booleanas de sólidos aplicadas a um cilindro e a uma esfera.
(a) Cilindro U esfera. (b) Cilindro ∩ esfera. (c) Cilindro – esfera. (d) Esfera – cilindro

O conceito das operações booleanas de sólidos é muito semelhantes ao das operações aplicadas a conjuntos – toma-se o sólido como um conjunto e seu volume como seu conteúdo, sendo o sólido então tratado pelas operações booleanas como o conjunto pelas operações de conjunto. Tal conceito, portanto, não corresponde àquele atribuído originalmente às operações booleanas, relativo a operações binárias.

Devido a sua simplicidade e eficiência, as operações booleanas de sólidos foram implementadas na maioria das ferramentas de modelagem 3D atuais, servindo de recurso para a composição de modelos tridimensionais.

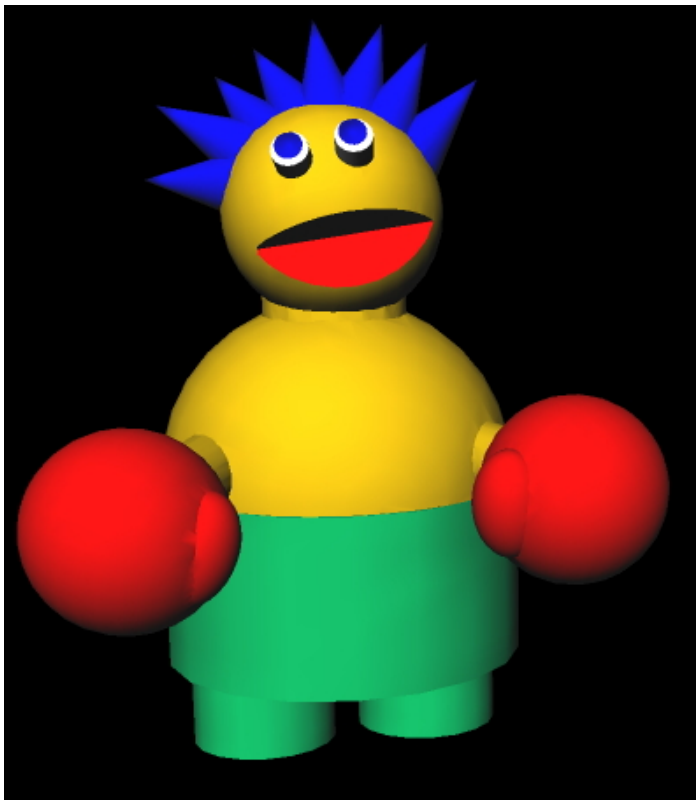

Figura 1.2 Modelo tridimensional gerado a partir da aplicação de operações booleanas em cubos, esferas, cilindros e cones.

1.4 Abordagem do problema

Representação diz respeito às estruturas de dados que definem um sólido computacional. Duas delas são comumente utilizadas para se lidar com operações booleanas de sólidos. A **representação de fronteira** (*boundary representation*), ou **b-rep,** dispõe de algoritmos conhecidos para a aplicação das operações booleanas. A **geometria construtiva de sólidos** (*constructive solid geometry*), ou **CSG**, é uma representação especialmente concebida para a utilização de tais operações. A combinação das duas é freqüentemente utilizada por ferramentas de modelagem, obtendo-se bons resultados. Há ainda outras representações que dispõem de facilidades para o uso das operações booleanas de sólidos. Dentre elas, estão a *Octree e a Binary Space Partitioning Tree* [FOL92].

Sabendo dos resultados satisfatórios obtidos pelo uso da representação que combina b-rep e CSG no trato das operações booleanas de sólidos, adotamos essa como base para

a solução do problema proposto. Ademais, ambientes de desenvolvimento de aplicações em 3D geralmente representam sólidos por b-rep, o que faz com que uma solução que envolva tal representação seja interessante. No caso de se fazer uso de tais ambientes, os recursos da geometria construtiva de sólidos podem ser adaptados para os mesmos. A solução adotada, assim como os conceitos relacionados, serão detalhados nas próximas seções.

2 AS REPRESENTAÇÕES PRIMÁRIAS

A solução adotada para o problema de se implementar as operações booleanas de sólidos baseia-se em duas representações primárias: a representação de fronteira e a geometria construtiva de sólidos. Para que o processo possa ser entendido, elas serão detalhadas a seguir.

2.1 Representação de fronteira

A representação de fronteira, mais conhecida como b-rep, representa um sólido por uma malha de planos poligonais que define sua superfície. Tal malha é definida em função de vértices ou arestas, e é geralmente compostas por triângulos ou quadriláteros. A figura 2.1(a) apresenta um sólido b-rep cujas faces foram preenchidas, e (b) o mesmo exibido em arame, para que suas faces pudessem ser melhor identificadas. B-rep é tida como uma representação clássica de sólidos, e é extensamente utilizada por aplicações 3D.

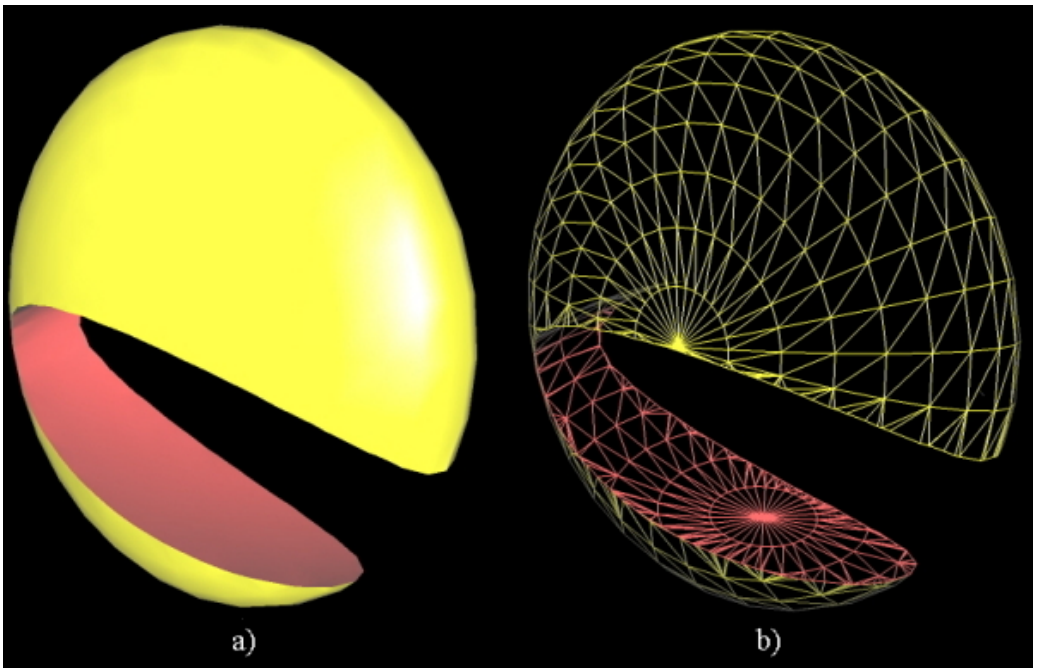

Figura 2.1 Sólido b-rep renderizado de diferentes formas
(a) Sólido b-rep com as faces preenchidas. (b) Sólido b-rep exibido em arame

B-rep é bastante eficiente para representar sólidos cujas faces são planas. Já superfícies curvas podem ser apenas aproximadas, devido à representação ser por meio de polígonos planos. Quanto mais polígonos são utilizados, maior a aproximação em relação a uma superfície curva (figura 2.2). Assim, a precisão do modelo nessas condições é diretamente proporcional ao custo computacional implicado na sua representação.

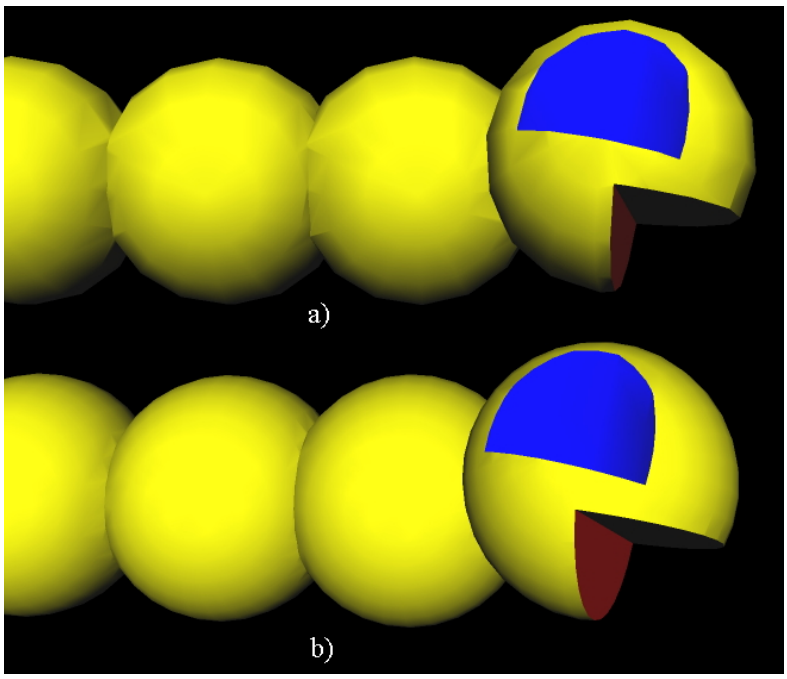

Figura 2.2 Modelos compostos por esferas definidas por diferentes quantidades de polígonos.
(a) Modelo composto por esferas de 224 polígonos. (b) Modelo composto por esferas de 960 polígonos

O custo computacional de se renderizar sólidos b-rep é reconhecidamente baixo. Por isso, b-rep é utilizada freqüentemente como forma intermediária por outras representações para fins de renderização. Ou seja, outras representações são convertidas para b-rep para serem renderizadas a um custo menor do que seriam se o fossem no formato original.

2.2 Geometria construtiva de sólidos

A geometria construtiva de sólidos (CSG) é um modelo de representação concebido em função das operações booleanas de sólidos. Sólidos assim representados são resultantes da aplicação das operações booleanas em sólidos elementares, chamados **primitivas**. Esferas, cones, cilindros e sólidos retangulares são formas comumente tidas como primitivas. A

representação dos sólidos CSG se dá em função das primitivas e operações utilizadas na sua composição.

A estrutura que representa um sólido CSG é uma árvore em que as operações booleanas utilizadas para construí-lo são hierarquizadas. Tal estrutura é chamada de **árvore CSG** (*CSG tree*). Cada uma das operações utilizadas é representada por um nó interno (não folha) da árvore, e cada primitiva por um nó folha. Os nós de uma árvore são organizados de forma que a operação relativa a um nó interno tenha sido aplicada aos sólidos obtidos a partir de seus nós filhos (figura 2.3). Assim, uma árvore CSG estará organizada conforme a ordem em que as operações booleanas foram aplicadas, de forma que quanto mais rasos são os nós de uma árvore (mais próximos da raiz) mais recentes serão as operações correspondentes.

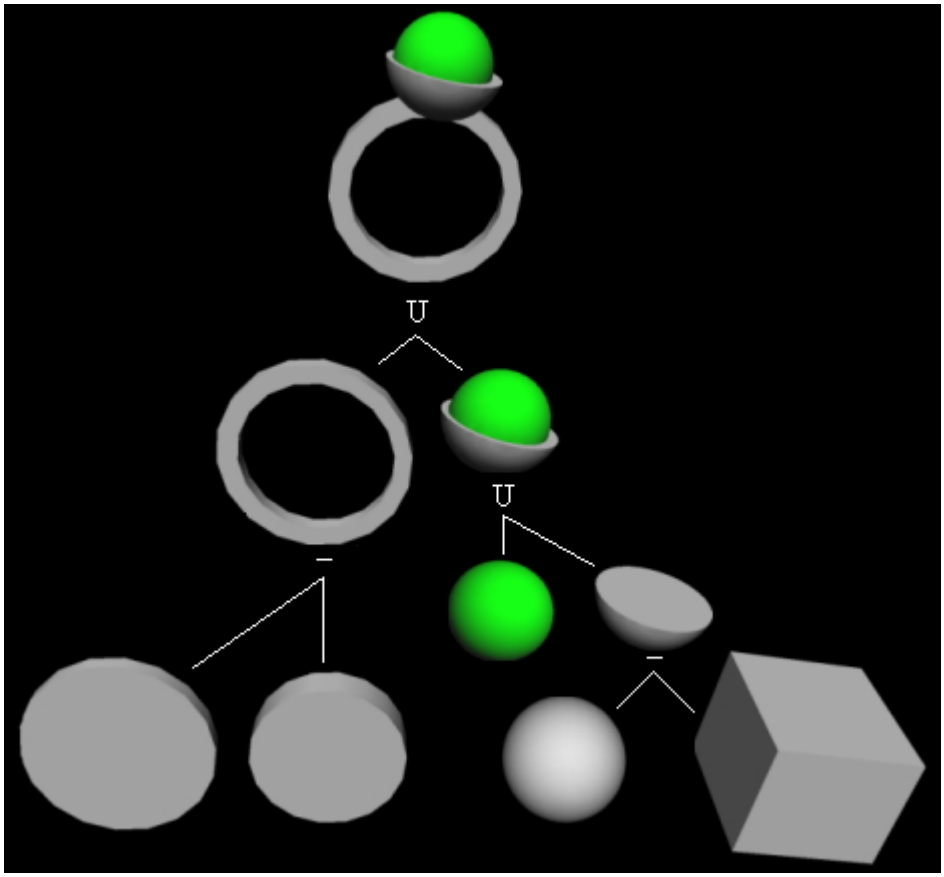

Figura 2.3 Árvore CSG cujos nós são representados pelos sólidos criados a cada etapa da composição do sólido

A construção de uma árvore CSG se dá em função do processo de composição do respectivo sólido: quando uma operação booleana é realizada, ela resulta em um novo sólido cuja árvore CSG terá um nó referente à operação feita como raiz e as árvores dos sólidos operandos como ramos partindo da mesma, sendo a árvore de uma primitiva composta por uma folha apenas.

A ordem em que os ramos vindos de um nó (que correspondem aos operandos de uma operação booleana) estão dispostos deve ser considerada na interpretação de uma árvore CSG, pois há operações não comutativas, e portanto, a inversão de seus operandos fará com que os resultados se alterem. Árvores CSG costumam ser binárias, ou seja, suportam operações booleanas com dois operandos apenas. Existem, porém, implementações cujas árvores não o são. O que faz pouca diferença, já que árvores não-binárias podem ser facilmente convertidas em árvores binárias, quebrando-se nós com *n* filhos em nós de dois apenas dispondo da mesma operação e em uma seqüência hierárquica, como é feito na figura 2.4.

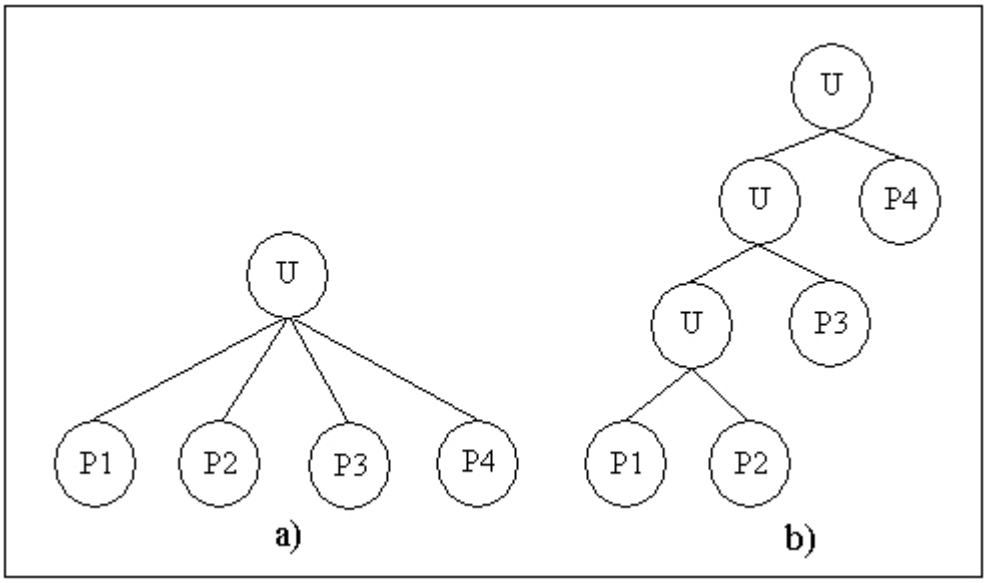

Figura 2.4 Transformação de uma árvore não binária em binária.
(a) Árvore CSG não binária. (b) Árvore de (a) transformada em binária

Pode-se ter nós representando transformações lineares (rotação, translação e mudanças de escala) utilizadas para mover primitivas a partir de sua posição inicial ou sólidos relativos a subárvores a partir da posição obtida por meio das operações booleanas utilizadas para

compô-lo. Entretanto, tal artifício se faz desnecessário, já que tais transformações podem ser associadas implicitamente aos nós modificados. E como tais operações são bastante freqüentes, uma árvore pode tornar-se bastante confusa pelo uso de tais nós, perdendo seu poder expressivo. Portanto, seu uso é desaconselhável.

A geometria construtiva de sólidos precisa fazer uso de uma outra representação, tida como sua representação interna, para que sólidos assim representados possam ser renderizados. Dados a serem utilizados de acordo com alguma metodologia de renderização aplicável à representação interna serão mantidos na árvore CSG, sendo esta processada quando a renderização do sólido correspondente é requerida. Um nó interno deve manter uma referência para a operação booleana a qual se refere, e um nó folha deve manter dados relativos a uma primitiva. O processamento é feito pelo uso da busca em profundidade, já que, assim, as operações são percorridas na ordem de aplicação das mesmas. Dessa forma, dados relativos às primitivas serão obtidos nos nós folhas, e combinações serão realizadas nos nós internos.

Freqüentemente, a representação interna utilizada pela geometria construtiva de sólido é definida pelos **semi-espaços** (*half-spaces*). Semi-espaços são estruturas definidas por funções f(x,y,z)=0 que dividem o espaço em duas partes. Primitivas são compostas por eles, correspondendo a espaços finitos delimitado pelos mesmos. Para criarmos um cilindro, por exemplo, bastaria a composição de um cilindro circular reto e dois planos (figura 2.5).

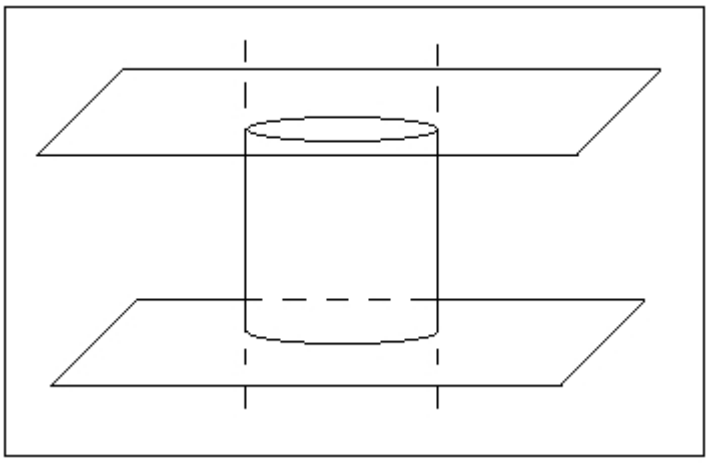

Figura 2.5 Cilindro composto por semi-espaços

Sólidos representados por CSG tendo semi-espaços como representação interna têm sua visualização processada sem que suas primitivas sejam explicitamente combinadas. Tais sólidos são renderizados pelo uso de ***ray tracing*** de uma forma especialmente concebida para tal: a árvore CSG é processada para cada raio, obtendo-se pontos de interseção para cada primitiva nos nós folhas e selecionando-se os mesmos nos nós internos conforme a operação correspondente [FOL92].

Em contrapartida, quando utilizadas outras representações internas, é comum que se utilize na renderização o sólido em representação interna correspondente ao sólido CSG, que é definido pela combinação direta de suas primitivas. Para tal, quando uma árvore CSG é processada, as primitivas em representação interna são obtidas quando nós folhas são visitados e sólidos em representação interna obtidos a partir dos nós descendentes são combinados quando nós internos o são (de acordo com uma estratégia compatível com a representação interna). Os nós internos podem manter a estrutura da primitiva correspondente em representação interna ou apenas informações a serem utilizadas na construção da mesma, dependendo de qual seja ela. Um elipsóide, por exemplo, poderia ser definido em função de seus raios nos eixos ‘x’, ‘y’ e ‘z’ e de sua posição no espaço. Esses valores seriam então mantidos nos nós folhas cujas primitivas fossem do tipo “elipsóide”. Tal forma de representar primitivas, caso utilizada, fará da árvore CSG uma forma bastante compacta de representar sólidos.

Uma árvore CSG define não somente a visualização de um sólido, mas também seu histórico de modelagem. Ela é um registro de todas as operações realizadas para a criação de um sólido. Assim, pode-se retornar a uma etapa anterior da modelagem facilmente, alterando primitivas ou operações. Basta que se faça acesso a um nó e se modifique o parâmetro desejado. Tal recurso pode ser utilizado de forma bastante produtiva por softwares de modelagem 3D, de forma que o usuário possa acessar nós de uma árvore relativa a um sólido, modificando seus parâmetros e visualizando o sólido modificado em seguida. A visualização do sólido deve ser reprocessada depois de efetuada a alteração.

A fim de ilustrar a modificação de parâmetros de árvores CSG., tomemos como exemplo a esfera com um buraco ilustrada na figura 2.6(a), resultante da diferença de uma esfera

por um cilindro. Pode-se aumentar o diâmetro do buraco acessando o nó folha relativo ao cilindro e modificando o raio do mesmo. Pode-se modificar também a cor interna do buraco modificando a cor do cilindro. Um possível resultado para tais alterações pode ser visto na figura 2.6(b). A operação aplicada no cilindro e na esfera também pode ser modificada. Basta que se acesse o nó interno relativo à diferença aplicada às primitivas e se modifique a respectiva operação. A figura 2.6(c) ilustra a alteração da operação utilizada para a união.

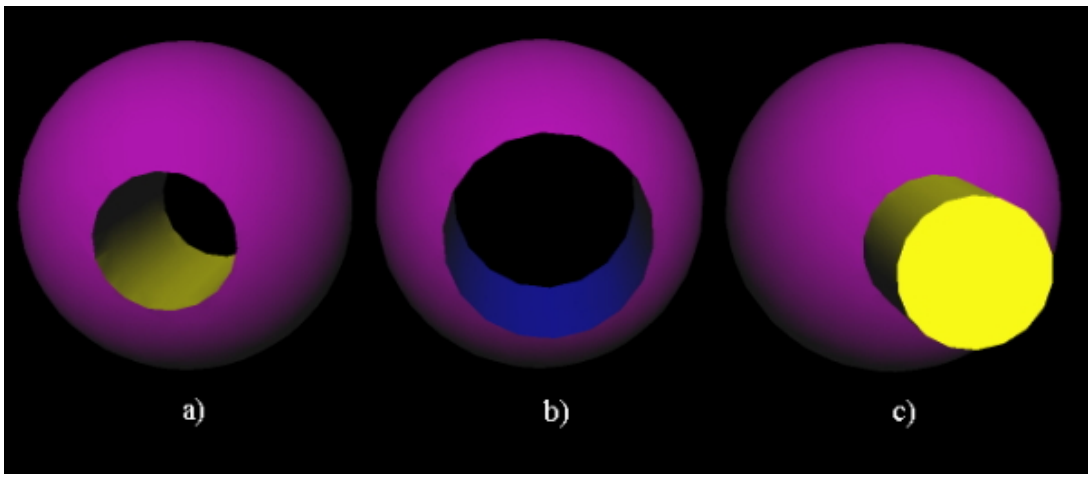

Figura 2.6 Modificação dos parâmetros de construção de um sólido.
(a) Sólido resultante da diferença de uma esfera por um cilindro. (b) Sólido de (a) tendo raio e cor do cilindro modificados. (c) Sólido de (a) tendo a operação modificada para união.

A geometria construtiva de sólidos, devido a sua flexibilidade e à forma compacta com que representa sólidos, tornou-se um modelo de representação bastante popular para se lidar com operações booleanas. Porém, em geral não é utilizada da forma clássica, e sim adaptada para uma estrutura que suporte outros tipos de operações.

3 UMA REPRESENTAÇÃO HÍBRIDA B-REP/CSG

Esta seção se propõe a descrever uma implementação para a representação de sólidos adotada para o problema em questão: uma representação que combina a representação de fronteira e a geometria construtiva de sólidos de forma que a primeira seja utilizada como representação interna da última. Em um primeiro momento, discutiremos as vantagens em adotá-la ao invés das representações em separado. Em seguida, sua implementação será descrita em linhas gerais.

3.1 Motivação

A motivação de se utilizar uma representação híbrida está em se aproveitar das vantagens que as diversas representações proporcionam, remediando eventuais fraquezas características a algumas delas. Por isso o uso de uma representação híbrida que faz uso de b-rep e CSG.

CSG é uma forma eficiente e robusta de se representar sólidos combinados por operações booleanas. Porém, representar um sólido dessa forma pode impossibilitar seu uso interativo. Isso porque sólidos CSG são renderizados geralmente por *ray tracing* (uma técnica de grande complexidade computacional), devido às suas primitivas serem freqüentemente representadas por semi-espaços. Uma alternativa para sanar o problema seria que a representação interna adotada fosse b-rep. Assim, o sólido b-rep obtido a partir da combinação direta das primitivas de um sólido CSG seria utilizado na renderização. Isso é possível porque operações booleanas são aplicáveis a sólidos b-rep. A renderização do sólido equivalente em b-rep em vez da renderização em *ray tracing* traz ganhos em eficiência, mas também perdas em qualidade, devido a sua falta de precisão para representar superfícies curvas.

A utilização unicamente de b-rep para se representar sólidos implicaria em se utilizar uma representação bem menos flexível, já que o histórico das operações realizadas não seria acessível, e portanto, primitivas e operações utilizadas não poderiam ser alteradas depois que as operações já tivessem sido realizadas. E menos compacta, já que o sólido representado da forma híbrida poderia ser representado apenas em função das operações realizadas e dos

parâmetros de composição das primitivas quando a renderização não fosse requerida, em vez de pelos dados relativos a todas as suas faces.

Apesar das vantagens implicadas na adoção de tal representação, ela só se justifica se duas condições forem satisfeitas. A primeira é que a visualização dos sólidos manipulados se dê pela renderização em b-rep, já que a motivação para o uso de tal representação é justamente seu baixo custo de renderização. A segunda é que tais sólidos devem poder ser combinados através das operações booleanas de sólidos, caso contrário, não se faz necessário o uso da geometria construtiva de sólidos.

3.2 Funcionamento

A seguir, será descrita uma possível implementação para a representação híbrida b-rep/CSG em linhas gerais, de forma a não focar detalhes relativos à estrutura de dados ou ambiente de desenvolvimento, por exemplo.

Sólidos a serem renderizados serão representados por uma árvore CSG adaptada ao suporte à representação b-rep, que é ilustrada na figura 3.1. Tal árvore deve guardar além dos parâmetros de composição do respectivo sólido, como faria uma árvore CSG convencional, estruturas em b-rep que definem sólidos intermediários para cada nó, para fins de economia de processamento, como veremos a seguir.

O processo de composição de tais sólidos, e de suas respectivas árvores em paralelo, tem início com a criação das primitivas. Elas são representadas por árvores de uma folha apenas, que mantém seus parâmetros de composição. No ato da criação, seus correspondentes em b-rep são criados a partir de tais parâmetros, sendo então associados às respectivas folhas e utilizados na renderização das mesmas. A figura 3.2(a) apresenta duas árvores referentes a primitivas. A criação de sólidos b-rep correspondentes a formas usualmente utilizadas como primitivas a partir de parâmetros de criação é relativamente simples: pode-se construí-los algoritmicamente ou a partir de coordenadas predeterminadas a serem modificadas. Um bom exemplo de uso deste último método de construção está na manipulação de sólidos retangulares. Define-se um cubo cujas faces tenham comprimento unitário e centro na origem. Daí, quando se

quer um sólido retangular com determinada largura, altura e profundidade, utiliza-se tais valores na aplicação de escala no sólido predeterminado nos respectivos eixos coordenados.

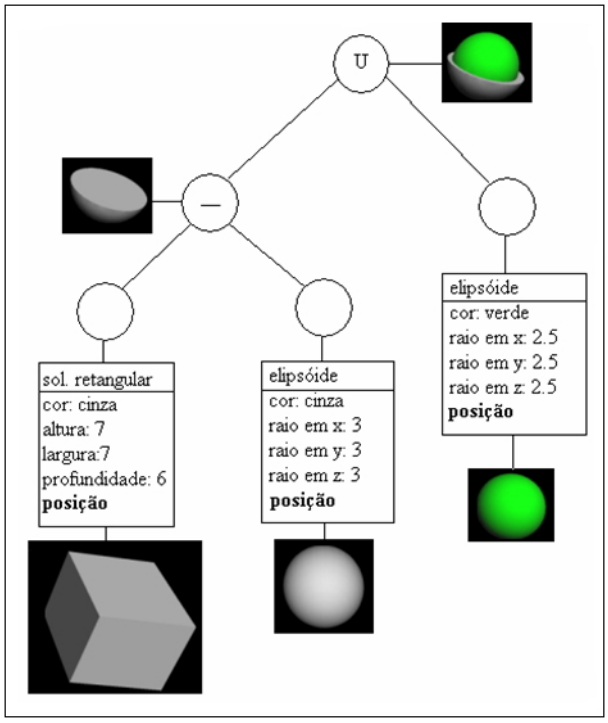

Figura 3.1 Árvore CSG adaptada ao suporte à representação b-rep

A construção de sólidos segue pela composição de primitivas, e posteriormente pela composição de sólidos compostos e primitivas. Sendo somente aceitas operações booleanas de sólidos binárias, tomam-se dois sólidos a serem combinados por uma dessas operações. Obtemos as estruturas em b-rep associadas aos nós raízes de suas árvores (que correspondem às versões em b-rep dos sólidos) e nelas aplicamos a operação. Há algoritmos específicos para a aplicação de operações booleanas em sólidos b-rep, um deles será detalhado na próxima seção. Será criado então um novo sólido cuja árvore terá como raiz um nó referenciando a operação utilizada e o sólido b-rep resultante da operação aplicada. A partir da raiz, partirão dois ramos que correspondem às árvores dos sólidos operandos. A figura 3.2(b) apresenta a árvore CSG resultante da aplicação da diferença nas duas primitivas da figura 3.2(a). O sólido b-rep referenciado pela raiz será utilizado na renderização.

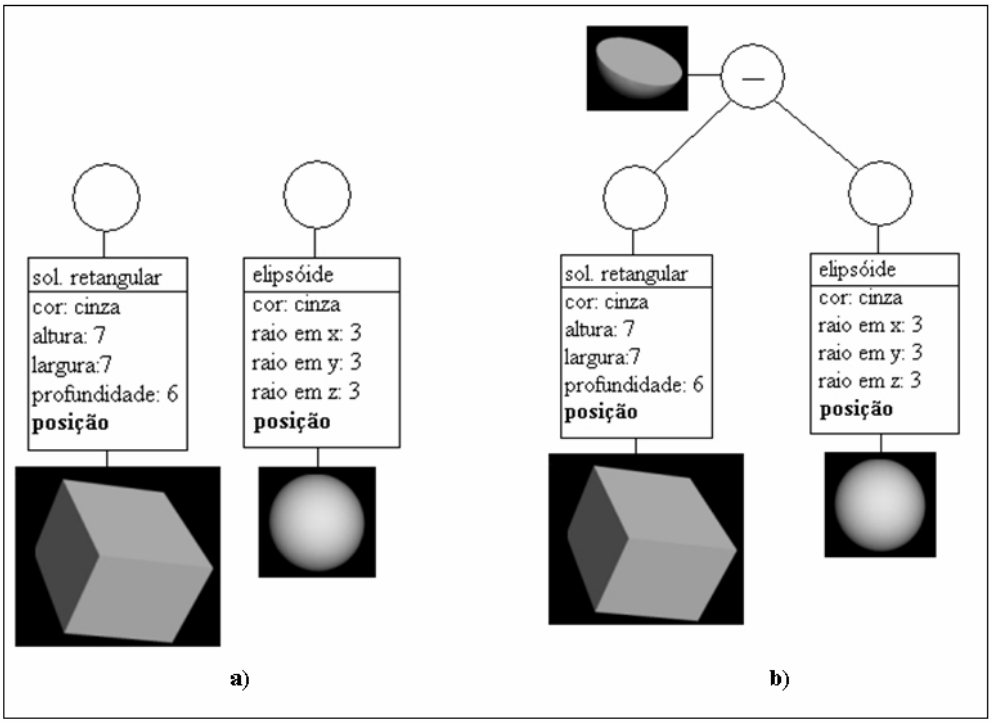

Figura 3.2 Processo de composição de sólido.
(a) Árvores CSG referentes a duas primitivas. (b) Árvore CSG referente à composição das duas primitivas de (a).

Deve-se atentar para a economia no processamento conseguida a partir da manutenção dos sólidos b-rep intermediários na árvore, apesar do maior consumo de memória envolvido. Caso isso não fosse feito, cada nova operação booleana faria com que seus sólidos operandos fossem reprocessados, ou seja, suas primitivas em b-rep seriam recriadas e as operações booleanas utilizadas para compor seus equivalentes em b-rep refeitas. O que não ocorre, somente a nova operação é aplicada.

Uma das funcionalidades esperadas de uma representação que faz uso da geometria construtiva de sólidos é que se possa mudar os parâmetros de composição de um sólido armazenados em uma árvore CSG. Isso pode ser feito facilmente, mas tal árvore terá que ser reprocessada (b-reps das primitivas recriados e operações relativas aos nós refeitas) de modo que os sólidos b-rep intermediários e o final reflitam as modificações feitas. Porém, nem toda a árvore terá que ser reprocessada, somente o nó modificado e os cujos b-reps intermediários sofrerão alguma influência por conta disso, ou seja, aqueles que se encontram no caminho dele até a raiz. Isso será feito nessa ordem, de modo que um nó seja atualizado somente depois que seus nós

filhos o forem. A figura 3.3 apresenta uma árvore correspondente à da figura 3.1 tendo os parâmetros de cor modificados em duas de suas primitivas. Os nós das primitivas até a raiz foram reprocessados, fazendo com que os b-reps associados a cada um deles se mantivessem atualizados em relação aos parâmetros das primitivas.

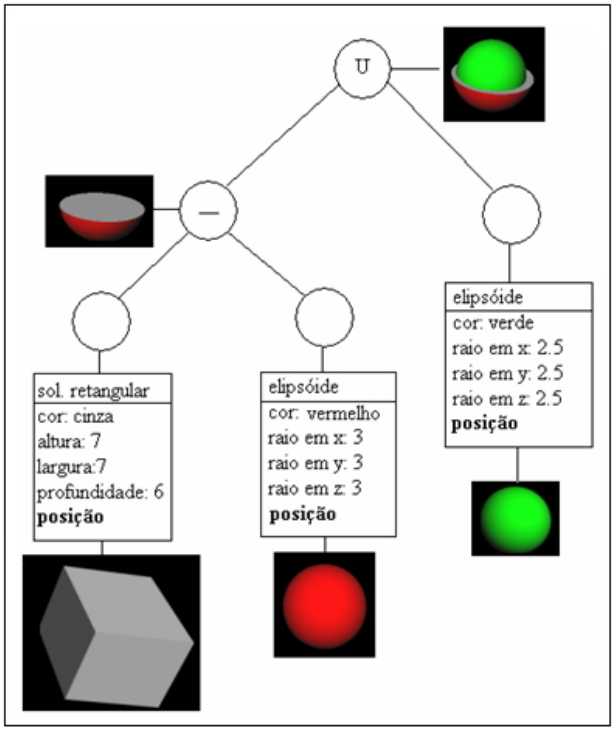

Figura 3.3 Árvore CSG correspondente a apresentada na figura 3.1 com as cores de primitivas modificadas

Pode-se utilizar uma árvore CSG que não armazena os sólidos b-rep intermediários para fins de armazenamento, já que tais dados podem ser obtidos posteriormente a partir dos parâmetros de composição. A figura 3.4 apresenta uma árvore desse tipo. Tal procedimento resulta em grande economia de memória. Quando o sólido for novamente renderizado, é feita uma busca em profundidade em que todos os nós serão reprocessados, obtendo-se assim os b-reps intermediários relativos a cada nó. A árvore CSG utilizada na renderização é então recriada a partir dos parâmetros de criação e dos sólidos b-rep obtidos.

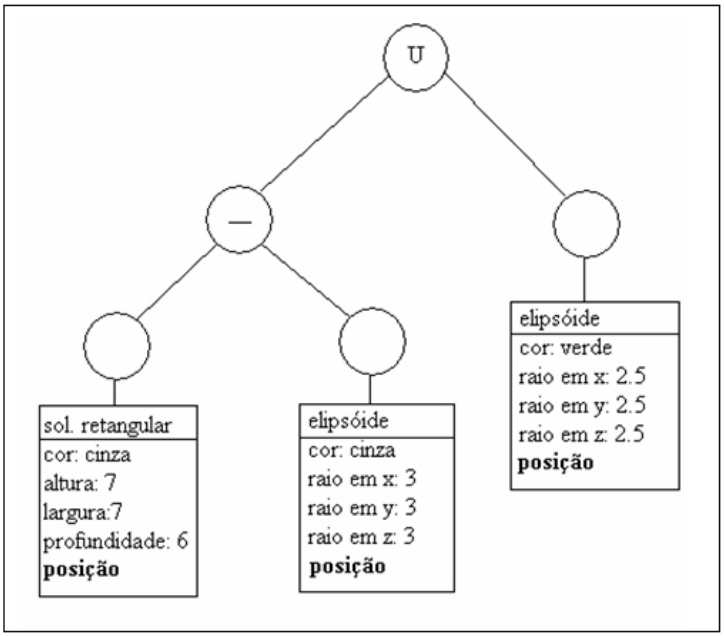

Figura 3.4 Árvore CSG utilizada para armazenamento

É esperado ainda que se possa movimentar um sólido de forma interativa. Fazê-lo requer que as posições das primitivas sejam atualizadas com relação ao movimento feito. A posição no espaço será, então, um dos parâmetros de composição das primitivas. Ela pode ser dada por uma matriz de transformação que informe a seqüência de todos os movimentos, rotações, translações e escalas, promovidos em um sólido a partir de uma posição inicial [WAT93]. Assim, quando uma primitiva for recriada a partir dos parâmetros de um nó folha, ela será composta a partir dos outros parâmetros como estando em uma posição inicial, e em seguida sua matriz de transformação é aplicada à mesma.

Para que a matriz de transformação de uma primitiva corresponda à sua real posição, ela deve ser multiplicada pelas matrizes de transformação relativas a todas as movimentações aplicadas à primitiva. Portanto, quando o usuário realiza um movimento em um sólido, para que a sua árvore CSG não fique defasada em relação à sua nova posição, a matriz de transformação de cada primitiva utilizada na sua composição deve ser multiplicada pela matriz relativa ao movimento. Todos os b-reps intermediários mantidos pela árvore também devem ser atualizados em relação à nova posição. Como todas as primitivas foram alteradas, toda a árvore terá que ser recomputada. E todo esse processamento pode prejudicar a fluidez da movimentação

quando ela for feita de forma interativa. Uma alternativa para amenizar o problema seria que uma matriz de transformação fosse mantida para uma seqüência de movimentos realizados em um sólido, não devendo tais movimentos provocar alterações na sua árvore em um primeiro momento. Finalizada a seqüência, ela enfim seria atualizada a partir da matriz obtida. Quando a interação é feita a partir do mouse, por exemplo, uma forma intuitiva de delimitar seqüências seria que cada “carregamento” do sólido correspondesse a uma delas. Assim, quando o usuário soltasse o botão do mouse, a árvore correspondente ao sólido movimentado seria atualizada.

4 OPERAÇÕES BOOLEANAS EM SÓLIDOS B-REP

O uso de b-rep como representação interna da geometria construtiva de sólidos requer que operações booleanas sejam aplicadas a sólidos b-rep. A implementação de algoritmos que realizem tais operações costuma ser complexa. Existem várias abordagens para o problema, e uma delas será descrita a seguir. Ela é uma adaptação de [LAI86] para o suporte apenas de faces triangulares e de cores nos vértices.

4.1 Introdução

O algoritmo aqui tratado opera em dois sólidos simultaneamente. Em um primeiro momento, ele cria as estruturas de dados a serem utilizadas ao longo de sua execução, que serão baseadas nos dois sólidos b-rep operandos. Em seguida, as faces desses sólidos são subdivididas de modo que não haja interseção de faces entre eles. Finalmente, as faces de um sólido são classificadas com base na superfície do outro como estando dentro, fora ou na fronteira do mesmo. Dependendo de como as faces foram classificadas, elas serão ou não selecionadas para compor o novo sólido, dependendo de qual foi a operação booleana utilizada. O sólido b-rep resultante será então composto a partir das faces selecionadas dos dois sólidos. O algoritmo 4.1 lista os passos acima, que consistirão nos módulos do algoritmo. São informadas também as seções em que cada módulo será detalhado.

```
//seção 4.2
criar estrutura de dados relativa ao sólidoA
criar estrutura de dados relativa ao sólidoB

//seção 4.3
subdividir faces de sólidoA com base no sólidoB
subdividir faces de sólidoB com base no sólidoA

//seção 4.4
classificar faces de sólidoA com base no sólidoB
classificar faces de sólidoB com base no sólidoA

//seção 4.5
selecionar faces do novo sólido com base na operação utilizada

gerar b-rep relativo ao novo sólido
```

Algoritmo 4.1 Estrutura do algoritmo para aplicação de operações booleanas em sólidos b-rep

Como as faces de um sólido são classificadas de acordo com a superfície do outro, as faces de um sólido devem de fato compor uma superfície: devem ter as normais orientadas “para fora” do sólido por elas definido e devem cobrir sua superfície inteiramente, sem que haja “buracos”. Além disso, todas as faces de um sólido são tidas pelo algoritmo como sendo parte de sua superfície, e portanto, lá deverão estar todas elas. Tais restrições permitem que se defina facilmente o interior e o exterior de um sólido. Submeter o algoritmo a sólidos que não respeitam os pré-requisitos acima resultará em um novo sólido contendo falhas em sua superfície.

Dois fatores devem ser levados em conta no ato no ato da implementação. O primeiro é que deve-se usar de precisão dupla para representar os pontos no espaço, pois como as faces geradas na etapa de subdivisão de faces podem ser bastante pequenas, a precisão simples pode não ser suficiente para representá-las. O segundo é que para todas as comparações feitas deve haver algum nível de tolerância envolvido, de forma que números que difiram menos que um determinado valor predefinido sejam considerados iguais. Isso devido às imprecisões inerentes à manipulação de pontos flutuantes.

Os próximos tópicos detalharão cada módulo que compõe o algoritmo.

4.2 Estrutura de dados

Há três estruturas de dados principais a serem utilizadas pelo algoritmo, que mantém informações relativas a vértices, faces e ao próprio sólido.

A estrutura de dados do vértice deve manter sua posição no espaço, sua cor, uma lista de vértices adjacentes e seu status. A cor deve ser a mesma para todos os vértices de uma face. A lista de vértices adjacentes é utilizada na classificação das faces para se descobrir regiões cujas faces tem o mesmo status. O status define a posição do vértice em relação ao outro sólido, que pode estar “dentro”, “fora” ou na “fronteira” . Tal valor estará inicialmente definido como “desconhecido”, sendo redefinido na etapa de classificação das faces como uma das posições possíveis.

A estrutura de dados da face deve manter referências para três vértices (somente faces triangulares são permitidas), sua normal, seu status e seus extremos.O status, como o status

do vértice, estará definido como “desconhecido” inicialmente, sendo redefinido na etapa de classificação das faces em função de sua posição relativa à superfície do outro sólido. As posições possíveis são: “dentro”, “fora”, “coincidente” e “oposta”. As duas últimas ocorrem quando a face coincide com uma face do outro sólido, sendo que na primeira situação as normais também coincidem, e na segunda são opostas. Os extremos são os valores máximo e mínimo que a face ocupa em cada eixo coordenado, sendo utilizados para determinar rapidamente se duas faces não se interceptam (testando se seus extremos não se interceptam)

A estrutura de dados do sólido deve manter duas listas, uma com as referências para suas faces e outra com as referências para seus vértices, e ainda seus extremos, utilizados para determinar rapidamente se uma face não intercepta um sólido. Serão criadas duas dessas estruturas, uma para cada sólido operando, no início da execução do algoritmo. Tais estruturas serão utilizadas nas próximas etapas, representando os respectivos sólidos.

4.3 Subdivisão de faces

Depois de criadas as estruturas de dados a partir dos sólidos operandos, deve-se quebrar suas faces de modo que faces de sólidos diferentes não se interceptem. Faces se interceptam caso uma das arestas ou o interior de uma face sejam interceptados por uma aresta ou contenham um vértice de outra face. Faces que compartilham vértices, arestas ou são coplanares não se interceptam. É importante que isso seja feito porque faces deverão ser classificadas como estando dentro, fora ou na fronteira do outros sólido, e faces com parte dentro e parte fora não podem ser devidamente definidas.

O algoritmo 4.2 lista os passos a serem executados para subdividir as faces de um sólido de forma que elas não interceptem as faces do outro sólido. Sua proposta é: para cada par de faces de sólidos distintos cujos extremos se interceptam, testar se tais faces se interceptam e, em caso afirmativo, quebrar a face do primeiro sólido em várias delas de modo que elas não interceptem a face do segundo sólido. Tal algoritmo deve ser aplicado aos dois sólidos operandos.

```
se extremos do sólidoA interceptam extremos do sólidoB
{
        para cada faceA do sólidoA
        {
                se extremos da faceA interceptam extremos do sólidoB
                {
                        para cada faceB do sólidoB
                        {
                                se extremos da faceA interceptam extremos da faceB
                                {
                                        //seção 4.2.1
                                        rotina para testar interseção das faces A e B
                                        se faces se interceptam
                                        {
                                                //seção 4.2.2
                                                rotina para subdivisão da face A
                                        }
                                }
                        }
                }
        }
}
```

Algoritmo 4.2 Rotina para subdivisão das faces de um sólido

O uso dos extremos faz com que muitos dos pares que não se interceptam, que constituem a grande maioria deles, sejam detectados rapidamente. Assim, não se faz necessário testar se há interseção entre tais pares, o que resulta em uma grande economia de processamento.

Para cada par de faces cujos extremos se interceptam, é executada a "rotina para testar interseção de faces", para verificar se suas faces de fato se interceptam. Os resultados possíveis são: "se interceptam", "não se interceptam" e "são coplanares". Se o resultado para um dos pares é "se interceptam", é executada a "rotina para subdivisão de face". A face quebrada será removida da lista de faces de seu sólido e as novas faces obtidas pela quebra serão adicionadas ao final da mesma, sendo também testadas futuramente.A seguir, serão detalhadas as rotinas acima citadas .

4.3.1 Rotina para testar interseção de faces

Descreveremos aqui a rotina que determina se duas faces (que chamaremos de A e B) se interceptam, se são coplanares ou se não se interceptam. O primeiro passo é realizar testes preliminares para detectar rapidamente alguns casos de não interseção. Para tal, obtém-se as distâncias dos vértices da face A até o plano da face B, sendo positivas as distâncias dos vértices que estão do lado do plano apontado pela normal da face B, e negativas as distâncias dos

que estão do lado oposto. Se todas as distâncias forem zero, as faces são coplanares. Se todas forem negativas ou positivas, a face A está inteiramente de um dos lados do plano da face B, e portanto, tais faces não se interceptam. Em qualquer uma das situações descritas, já foi constatado que as faces não se interceptam, e portanto, a execução da rotina termina para que outras faces possam ser testadas quando detectadas. Caso nenhuma delas seja verdade, ou seja, os sinais das distâncias são diferentes, elas ainda podem se interceptar, e a execução da rotina prossegue. Agora os papéis se invertem: os vértices da face B são testados com o plano da face A. Novamente, se as distâncias dos vértices ao plano forem todas positivas ou negativas (as faces coplanares já teriam sido detectadas no último teste), a execução termina. Caso contrário, os testes preliminares foram inconclusivos, e a execução prossegue.

Em seguida, calculamos a linha de interseção L entre os dois planos. Tal linha será definida por um ponto P e uma direção D. P pode ser qualquer ponto pertencente aos planos das duas faces, e D pode ser calculada por uma multiplicação vetorial entre as normais das faces. A partir de L, obtém-se os segmentos de reta S1 e S2 correspondentes às interseções da linha com as faces A e B, respectivamente. Caso os segmentos S1 e S2 se interceptem, as faces se interceptam, caso contrário, não. A figura 4.1 apresenta duas faces e os elementos a partir delas obtidos: a linha de interseção entre os planos e os segmentos correspondentes às interseções da linha com cada face.

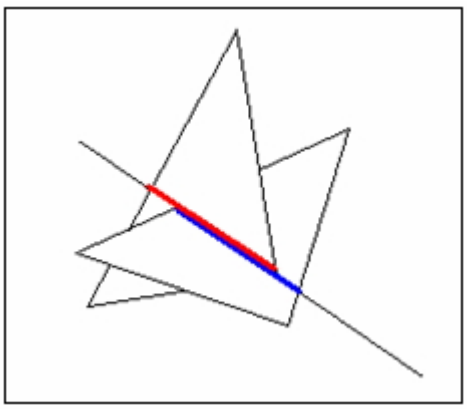

Figura 4.1 Representação de duas faces, da linha de interseção entre os planos das faces (em preto) e dos segmentos de reta correspondentes às interseções da linha com as faces (em vermelho e azul)

Os segmentos S1 e S2 serão criados de forma a manterem informações importantes a serem utilizadas na rotina de subdivisão de faces individualmente. Serão mantidos os dados relativos aos seus extremos inicial e final: ponto de interseção (com sua face), distância de P até seu ponto de interseção, descritor relativo a qual parte da face corresponde e um vértice próximo. Será ainda mantido um descritor relativo ao segmento intermediário que está entre os dois extremos.

Sobre os descritores dos pontos extremos e do segmento intermediário, os pontos sempre estarão em um dos extremos da face, seja em um dos vértices ou em uma das arestas. E o segmento intermediário pode estar tanto nos extremos quanto no interior da face. A figura 4.1 ilustra as situações possíveis.

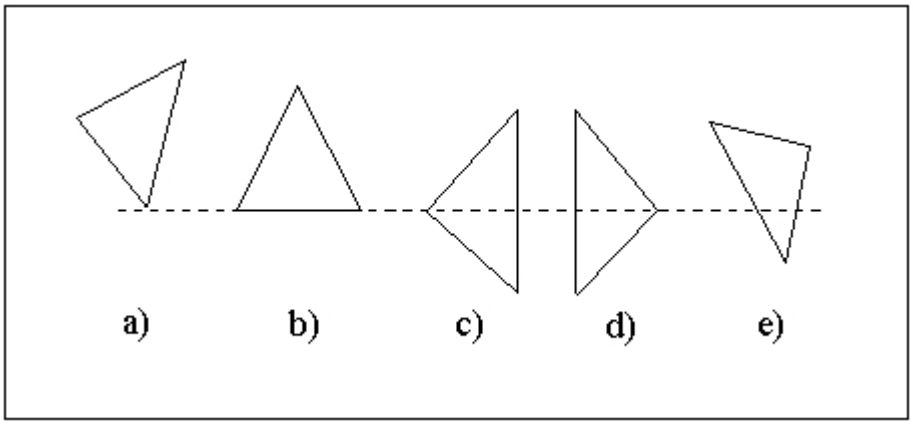

Figura 4.2 Interseções possíveis entre segmentos e faces.
(a) vértice-vértice-vértice. (b) vértice-aresta-vértice. (c) vértice-face-aresta.
(d) aresta-face-vértice. (e) aresta-face-aresta

As distâncias dos vértices da face A até o plano da face B calculadas anteriormente serão utilizadas na obtenção dos dados relativos ao segmento da face A. O primeiro passo é obter os dois extremos de A interceptados por L, que podem estar em vértices ou arestas. Primeiramente, os vértices são testados: se alguma das distâncias dos vértices ao plano for zero, o respectivo vértice conterá um dos extremos. Caso isso ocorra e os outros vértices estiverem do mesmo lado do plano (distância de mesmo sinal), o mesmo vértice conterá o outro extremo também, caracterizando a situação “vértice-vértice-vértice”. Quando um vértice contém um extremo, os dados deste serão assim definidos: seu descritor estará definido como “vértice”, o

seu ponto de interseção estará na mesma posição do vértice e seu vértice próximo será o próprio vértice que o contém. Caso não tenham sido encontrados dois extremos ainda, as arestas serão testadas. Caso as distâncias dos vértices componentes de uma aresta até o plano da face B tenham sinais diferentes, tal aresta contém um extremo. Quando uma aresta contém um extremo, os dados deste serão assim definidos: seu descritor estará definido como "aresta", o seu ponto de interseção será obtido pela interseção de L com sua aresta e seu vértice próximo será o primeiro vértice da aresta (levando em conta que os vértices de uma face são ordenados). Depois de obtidos os dois extremos, o descritor do segmento intermediário pode ser descoberto: será "vértice" se os dois extremos estiverem no mesmo vértice, será "aresta" se os dois extremos estiverem em vértices distintos e será "face" se as duas situações anteriores não forem verdadeiras. A definição de qual dos extremos será inicial e final dependerá das distâncias de seus pontos até P: aquele que tiver a menor distância será o extremo inicial, o outro será o final. Os dados relativos ao segmento da face B serão obtidos da mesma forma em relação a face A, porém fazendo-se uso das distâncias dos vértices da face B até o plano da face A

Para se verificar se duas faces se interceptam, basta verificar se a distância de P até o ponto final de alguma delas é menor que a distância de P até o ponto inicial do outro. Em caso afirmativo, é retornado que as faces se interceptam e serão subdivididas na próxima rotina. Caso contrário, é retornado que as faces não se interceptam e o próximo par de faces é testado.

4.3.2 Rotina para subdivisão de face

Tendo sido detectadas duas faces que se interceptam, uma delas será subdividida de forma que a outra não seja interceptada pelas faces criadas. Isso será feito através da rotina descrita a seguir.

Para se quebrar a face A de modo que nenhuma das faces criadas intercepte a face B, devemos achar o segmento relativo a interseção dos segmentos S1 e S2. Na figura 4.3, tal segmento é apresentado em amarelo. Em seguida, devemos determinar como ele se relaciona com a face A. Seus extremos podem estar em vértices, arestas ou internamente. A face A deverá ser quebrada em função desse relacionamento.

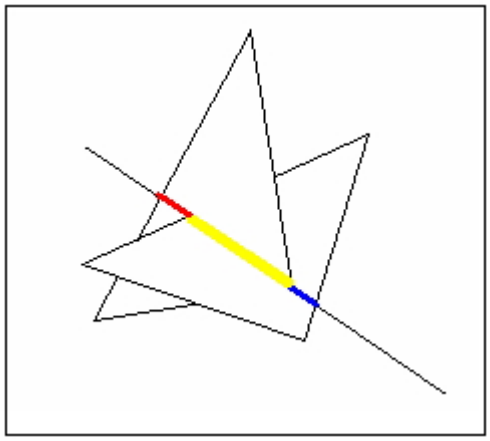

Figura 4.3 Representação do segmento de reta relativo à interseção dos segmentos da figura 4.1 (em amarelo)

O primeiro passo é obter o segmento correspondente à interseção dos segmento S1 e S2. Para tal, deve-se descobrir os extremos inicial e final do mesmo, que serão os extremos inicial e final de alguns dos segmentos S1 e S2. O extremo inicial será o mais distante do ponto P, e o final será o mais próximo. Os descritores dos extremos obtidos a partir de S2 serão modificados. O descritor intermediário será igual ao descritor intermediário do segmento S2. O processo é apresentado de forma mais detalhada no algoritmo 4.3.

```
Se a distância do início do segmento S2 até P > distância do início do segmento S1 até P
{
       extremo inicial da interseção = extremo inicial do segmento S2
       descritor inicial da interseção = descritor intermediário do segmento S1
}
senão
{
       extremo inicial da interseção = extremo inicial do segmento S1
}
se a distância do fim do segmento S2 até P < distância do fim do segmento S1 até P
{
       extremo final da interseção = extremo final do segmento S2
       descritor final da interseção = descritor intermediário do segmento S2
}
senão
{
       extremo final da interseção = extremo final do segmento S2
}
descritor intermediário da interseção = descritor intermediário do segmento S1
```

Algoritmo 4.3 Rotina para construção do segmento relativo à interseção dos segmentos S1 e S2

A próxima etapa será descobrir a disposição do segmento obtido em relação à face a ser quebrada. Dependendo de qual seja ela, a quebra se dará de uma forma diferente. As disposições possíveis serão enumeradas a seguir:

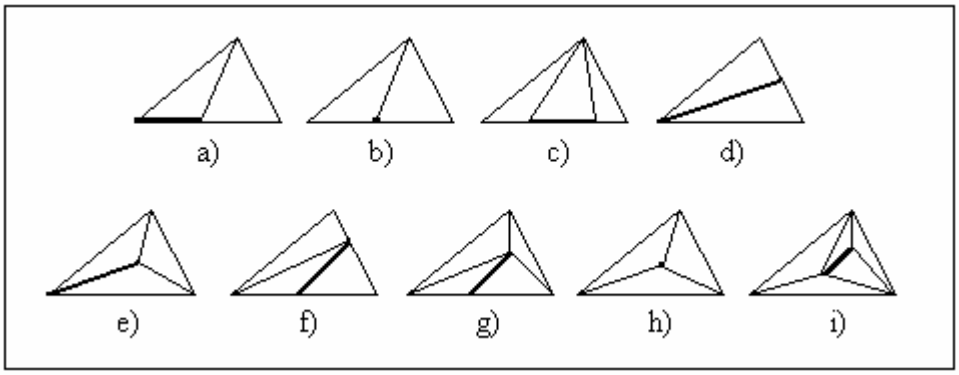

Figura 4.4 Relacionamentos segmento/face possíveis e as quebras geradas

- Vértice-vértice-vértice: a face é interceptada no seu vértice, não precisa ser quebrada.
- Vértice-aresta-vértice: a face é interceptada em uma de suas arestas de ponta a ponta, não precisa ser quebrada.
- Vértice-aresta-aresta: O segmento parte de um vértice, segue por uma aresta e finaliza na mesma. Resulta na criação de um novo vértice e na quebra da face em duas (figura 4.2(a))
- Aresta-aresta-aresta: o segmento inicia e finaliza em uma mesma aresta. Caso o segmento seja de um ponto apenas, um vértice será criado e a face será quebrada em duas (figura 4.2(b)). Caso seja maior que um ponto, dois vértices serão criados e a face será criada em três (figura 4.2(c)).
- Vértice-face-aresta: o segmento inicia em um vértice, segue por dentro da face e finaliza em uma aresta. Resulta na criação de um vértice e na quebra da face em duas (figura 4.2(d)).
- Vértice-face-face: o segmento inicia em um vértice e finaliza dentro da face. Resulta na criação de um vértice e na quebra da face em três (figura 4.2(e)).
- Aresta-face-aresta: o segmento inicia em uma aresta e termina em outra. Resulta na criação de dois vértices e na quebra da face em três (figura 4.2(f)).

- Aresta-face-face: o segmento inicia em uma aresta e termina no interior da face. Resulta na criação de dois vértices e na quebra da face em três (figura 4.2(g)).
- Face-face-face: o segmento inicia e finaliza dentro da face. Caso o segmento seja de um ponto apenas, um vértice será criado e a face será quebrada em três (figura 4.2(h)). Caso seja maior que um ponto, dois vértices serão criados e a face será criada em cinco (figura 4.2(i)).

Como pôde ser visto, é criado um vértice para cada extremo do segmento que não está contido em outro vértice (estando os dois extremos em um mesmo ponto, um só é criado). A cor do vértice será a mesma dos outros vértices da face. Quando um vértice é criado, é procurado se há um outro na lista de vértices do sólido que ocupe o mesmo ponto no espaço (ou um ponto muito próximo) e a mesma cor. Caso não haja, ele é inserido na lista de vértices e sua referência é utilizada na criação das faces. Caso contrário, será utilizada uma referência para o vértice equivalente encontrado. Vértices criados e interceptados pelo segmento terão seus status definidos como "fronteira". Isso será bastante útil na etapa de classificação de faces.

As faces criadas serão inseridas na lista de faces do sólido, e a face quebrada removida. A normal das faces criadas deve apontar para a mesma direção que a normal da face removida. Se essa é obtida levando em conta a ordem dos vértices, deve-se atentar para a ordem dos mesmos nas faces criadas. Deve-se tomar também alguns cuidados no ato da inserção das faces na lista. Deve-se verificar se há dois vértices semelhantes referenciados pela face (na mesma posição ou muito próximos e com a mesma cor), em caso afirmativo, a inserção não é efetuada. É verificado também se a área da face é nula ou muito próxima disso, não sendo inserida na lista nesse caso também.

4.4 Classificação de faces

Não existindo mais interseções de faces entre os dois sólidos, essas devem ser classificadas. Para classificá-las individualmente, é utilizada a rotina listada no algoritmo 4.4, que é baseada na técnica r*ay tracing* de computar interseções. Ela recebe uma face e um sólido, e retorna a posição da face em relação ao sólido, que pode ser "dentro", "fora", "coincidente" e "oposta", sendo as duas últimas relativas a situações em que a face coincide com uma face do outro sólido, sendo que na primeira situação suas normais coincidem, e na segunda são opostas.

```
criar RAIO partindo do baricentro da faceA na direção da normal da faceA
enquanto o tracejo do raio não for bem sucedido
{
	para cada faceB do sólidoB
	{
		achar PRODUTO ESCALAR a partir da direção do RAIO e da normal da faceB
		achar PONTO DE INTERSEÇÃO entre RAIO e o plano da faceB
		se PONTO DE INTERSEÇÃO existe
		{
			achar DISTÂNCIA do início do raio até o PONTO DE INTERSEÇÃO
			se DISTÂNCIA = 0 e PRODUTO ESCALAR = 0
			{
				perturbar RAIO
				reiniciar (tracejo não foi bem sucedido)
			}
			senão se DISTÂNCIA = 0 e PRODUTO ESCALAR ≠ 0
			{
				se faceB contém o PONTO DE INTERSEÇÃO
				{
					FACE MAIS PRÓXIMA = faceB
					DISTANCIA DA FACE MAIS PROXIMA = DISTANCIA
					finalizar loop
				}
			}
			senão se DISTANCIA > 0
			{
				se DISTANCIA < DISTÂNCIA DA FACE MAIS PRÓXIMA encontrada
				{
					se faceB contém o PONTO DE INTERSEÇÃO
					{
						FACE MAIS PRÓXIMA = faceB
						DISTANCIA DA FACE MAIS PROXIMA = DISTANCIA
					}
				}
			}
		}
	}
}
se nenhuma face foi interceptada
{
	retornar "fora"
}
achar PRODUTO ESCALAR a partir da direção do RAIO e da normal da FACE MAIS PRÓXIMA
	se DISTANCIA DA FACE MAIS PROXIMA =0
	{
		se PRODUTO ESCALAR >0
		{
			retorna "coincidente"
		}
		senão
		{
			retorna "oposta"
		}
	}
se PRODUTO ESCALAR >0
{
	retorna "dentro"
}
se PRODUTO ESCALAR <0
{
	retorna "fora"
}
```

Algoritmo 4.4 Rotina para classificação de face

Sendo o baricentro da face o ponto correspondente à média das posições dos vértices da mesma, um raio é tracejado a partir do baricentro da face A na direção de sua normal, sendo testado em todas as faces B do sólido B. Procura-se, assim, obter a face mais próxima interceptada pelo raio, para que se descubra a partir dela em que posição a face está em relação à superfície do sólido. Os testes são feitos de forma a enquadrar o par de faces em uma das situações possíveis, operando dependendo de qual seja ela. Para cada face B a ser testada, é obtido o ponto de interseção do seu plano com o raio. Se o ponto não pôde ser obtido, o plano é paralelo ao raio e não o contém, e portanto não há interseção e a face não será mais testada. Se o ponto de interseção estiver junto ao baricentro e o raio for paralelo ao plano da face B, tal raio deve ser levemente perturbado e todas as faces devem ser novamente testadas, pois a face encontrada será a mais próxima e nada se pode concluir de faces paralelas ao raio. Se o ponto de interseção estiver junto ao baricentro e o raio não for paralelo ao plano da face B, caso o ponto esteja de fato na face B, esta será coplanar a face A, sendo portanto a face mais próxima possível, não sendo necessário que as outras faces sejam testadas. Se a distância do baricentro da face A até o ponto de interseção for positiva (se fosse negativa, o ponto de interseção estaria atrás do raio), caso a distância seja menor que a distância até a face mais próxima obtida até então e o ponto de interseção de fato esteja na face B, esta passará a ser a face mais próxima da face A.

Depois de testadas as faces do sólido B, se não foi detectada interseção do raio com nenhuma face do sólido B, ela estará fora do mesmo. Se a face mais próxima encontrada estiver a distância zero da face A, ambas são coplanares. Nesse caso, se suas normais apontam para a mesma direção, elas são tidas como coincidente, caso contrário, opostas. Se a normal da face encontrada apontar para o plano da face A, esta será tida como estando fora, e se apontar para o lado oposto, dentro.

O procedimento descrito acima equivaleria a traçagem de um raio a partir de uma face na direção de sua normal, sendo então classificada conforme a orientação da primeira face interceptada do sólido em relação ao mesmo raio (ou como estando “fora” caso não interceptasse nenhuma). A figura 4.5 ilustra tal conceito. Nela, um raio é traçado a partir de uma face (em vermelho), interceptando uma face do sólido testado (em preto). Levando em consideração a normal da face interceptada pelo raio, a face testada seria classificada como estando “dentro” do sólido.

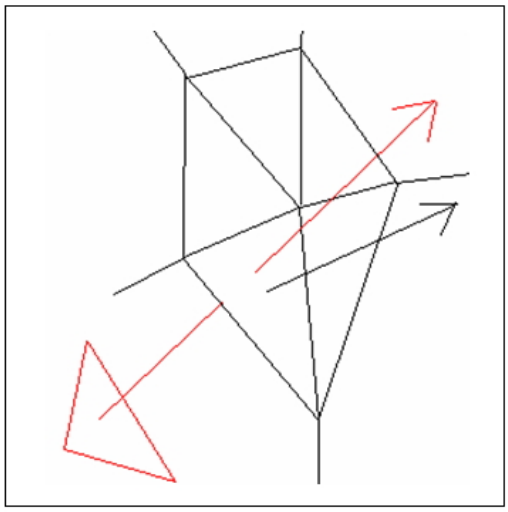

Figura 4.5 Traçagem de raio a partir de uma face para se descobrir sua posição relativa a um sólido

Tal rotina para classificação de faces poderia ser utilizada para classificar todas as faces dos dois sólidos operandos. Tal operação seria, porém, de grande complexidade computacional. Ela seria então utilizada de outra forma: as faces conexas a serem classificadas da mesma forma comporão regiões em que basta que uma delas seja classificada pela rotina de classificação de faces para que todas o sejam da mesma forma. Tais regiões são delimitadas pelos vértices de fronteira, marcados assim na etapa de subdivisão de faces.

A abordagem acima é realizada pela rotina definida no algoritmo 4.5. Em um primeiro momento, as listas de vértices adjacentes de cada vértice são montadas: para cada face do sólido, cada dois vértices são acrescentados à lista de vértices do terceiro deles. A partir daí, cada uma das faces será classificada. O processo se dá da seguinte forma: todos os vértices em princípio estarão definidos como “desconhecido” ou “fronteira”. As primeiras faces serão classificadas pela rotina de classificação de faces (algoritmo 4.4). Quando isso acontece, os vértices “desconhecidos” das faces classificadas serão também classificados conforme o status da face. Todos os vértices “desconhecidos” adjacentes aos vértices classificados também serão classificados da mesma forma, e assim por diante, conforme a recursão do algoritmo 4.6. Assim, todos os vértices dentro de uma região delimitada por vértices de “fronteira” serão classificados da mesma forma, como “dentro” ou “fora”. Quando as faces que contém tais vértices forem classificadas, o serão conforme a classificação de seus vértices, sem que seja necessário executar

a rotina para classificação de faces. Esta só será necessária quando estiver sendo classificada a primeira face de uma região (que eventualmente pode ser composta por uma face apenas cujos vértices são de "fronteira"). Tal procedimento deve ser realizado para os dois sólidos operandos.

```
calcular vértices adjacentes para todo vértice do sólidoA
para cada faceA do sólidoA
{
        se algum vérticeA da faceA tem status definido como "dentro" ou "fora"
        {
                status da face = status do vérticeA
        }
        senão
        {
                rotina para classificação da faceA
                para cada vértice da faceA
                {
                        rotina para classificação de vértice para status da faceA
                }
        }
}
```

Algoritmo 4.5 Rotina para classificação das faces de um sólido

```
definir status de vértice para status recebido
para cada vértice adjacente
{
        se status de vértice adjacente é "desconhecido"
        {
                rotina para definição de status de vértice para status recebido
        }
}
```

Algoritmo 4.6 Rotina para classificação de vértice

Outra tática possível para classificar faces individualmente seria contar o número de faces do outro sólido interceptadas pelo raio. Caso o número fosse ímpar, estaria dentro, caso contrário, estaria fora. Nesse caso, teria que se considerar devidamente as situações em que o raio intercepta arestas e vértices. Uma implementação equivocada poderia considerar duas interseções para arestas e três para vértices (uma para cada sólido que o compartilha) enquanto o certo seria contabilizar apenas uma interseção.

4.5 Seleção de faces

Estando as faces dos dois sólidos operandos de uma operação booleana classificadas, as faces que compõem o sólido resultante devem ser selecionadas. A tabela 4.1 mostra o esquema de seleção de faces utilizado.

			Sólido A			**Sólido B**		
	Dentro	**Fora**	**Coincidentes**	**Opostas**	**Dentro**	**Fora**	**Coincidentes**	**Opostas**
AUB	-	X	X	-	-	X	-	-
A∩B	X	-	X	-	X	-	-	-
A-B	-	X	-	X	X	-	-	-

Tabela 4.1 Esquema de seleção de faces para compor um novo sólido

Como pode ser visto na tabela, a união engloba as faces externas dos dois sólidos, a interseção as faces internas e a diferença as faces internas de um e as externas de outro. Faces classificadas como “coincidentes” ou “opostas” tem uma outra face que coincide com ela no outro sólido. Somente uma deve ser mantida. Por convenção, as faces do sólido A o serão. Quando aplicada a diferença, as normais das faces internas do sólido B devem ser invertidas.

5 CONCLUSÃO

A adoção de uma representação que combina a geometria construtiva de sólidos e a representação b-rep para a implementação das operações booleanas de sólidos provou-se acertada afinal. Isso pôde ser constatado pelos resultados obtidos a partir da aplicação criada em paralelo a este documento. Uma solução bem concebida baseada em tal representação deve resultar em uma forma flexível e robusta de se modelar sólidos, dispondo de vários facilitadores para o desenvolvimento de modelos tridimensionais baseados nas operações booleanas.

A solução aqui adotada, em especial, pôde aproveitar bastante bem a representação híbrida b-rep/CSG, apresentando as qualidades esperadas de tal uso. A aplicação criada mostrou-se flexível, por permitir a alteração de quaisquer parâmetros de modelagem utilizados na composição de um sólido, e compacta, por permitir o armazenamento de um sólido consumindo uma quantidade mínima de memória, apresentando ainda um baixo custo associado à renderização, devido ao uso de b-rep para tal. E a implementação do algoritmo que realiza operações booleanas em sólidos b-rep mostrou-se relativamente eficiente, não sobrecarregando o sistema na maioria de suas aplicações. Crê-se, então, que o objetivo inicial foi cumprido, que era descrever uma solução eficiente para o problema de se implementar as operações booleanas de sólidos.

O maior entrave na concepção da solução adotada foi o algoritmo para aplicar operações booleanas em sólidos b-rep. Este é um algoritmo complexo, elaborá-lo no tempo ao qual nos propomos a resolver o problema seria inviável. E toda a bibliografia que tratava do assunto encontrada até um determinado momento não o fazia em detalhes suficientes para sua implementação, sempre ignorando fatores importantes e de difícil concepção, em especial os que dizem respeito à subdivisão de faces. Até mesmo textos clássicos como o [FOL92] tratavam o assunto de forma incompleta. A partir do momento que se dispôs de [LAI86], que descreve a solução em detalhes suficiente para ser implementada, esse passou a ser adotado como base para a solução do problema. A implementação de tal algoritmo foi bastante trabalhosa, sendo talvez mais ainda a busca por erros que se deu posteriormente. Problemas esses bastante difíceis de serem isolados, decorrentes principalmente de imprecisões e situações inesperadas. Atualmente,

porém, parece estar funcionando corretamente. O restante pôde ser implementado sem que houvesse grandes sobressaltos.

A API Java 3D, utilizada no desenvolvimento da aplicação, confirmou ser uma grande ferramenta para o desenvolvimento de aplicações em 3D. Isso devido ao aparato que disponibiliza para representar e visualizar sólidos, bastante poderoso e de uso relativamente simples, e às facilidades que a linguagem de programação Java oferece.

Pôde-se notar o quão úteis são as operações booleanas no domínio da modelagem de sólidos, devido à eficiência e simplicidade envolvidas na concepção de modelos em determinadas situações. Elas não são, porém, auto-suficientes, ou seja, determinados modelos não serão facilmente concebidos pelo uso unicamente das operações booleanas, mesmo que o número de primitivas disponíveis seja expressivo. Ainda assim, não devem faltar em um software de modelagem.

Este trabalho também pode ser encontrado no *site* do Laboratório de Computação Multimídia (LCMM) do Departamento de Ciência da Computação da Universidade de Brasília. Lá, estão disponíveis este documento e as implementações feitas. O endereço é: http://primordial.cic.unb.br/lcmm/projetos/boolean/

6 REFERÊNCIAS BIBLIOGRÁFICAS

[CAR02] Carrara V. **Manual de computação gráfica**. 2002. Disponível em: http://www.directnet.com.br/users/val/tutor/tutor.html

[FOL92] Foley J. D. et. al. Van Dam A.; Feiner S. K.; Hughes J. F. **Computer graphics – principles and practice.** 2. ed. Addison-Wesley, 1992.

[LAI86] Laidlaw D. H.; Trumbore W. B.; Hughes J. F. **Constructive solid geometry for polyhedral objects.** SIGGRAPH Proceedings, 1986. p.161.

[MAR98] Martin, J. **Voxel based CSG modeler.** Graz, Austria: Institute for Computer Graphics and Vision, University of Technology, 1998. Disponível em: http://www.icg.tu-graz.ac.at/~Education/Seminar_Projekt/fertig/martin/project.html

[MEL98] Melax S. **A simple, fast, and effective polygon reduction algorithm**, Game Developer Magazine, 1998. Disponível em www.melax.com/polychop/gdmag.pdf

[PIO] Pio J. L. S. **Modelagem de sólidos**. Disponível em http://www.dcc.ufmg.br/~josepio/discp/modgeo/modgeo.html

[SUN03] Sunday, D. **Geometry Algorithms**, 2001-2003. Disponível em http://geometryalgorithms.com/index.htm

[SWO95] Swokowski, E. W. **Cálculo com Geometria Analítica**. 2. ed. São Paulo: Makron Books do Brasil, 1995. v.2.

[WAT93] Watt A. **3D Computer graphics.** 2 ed. Addson-Wesley, 1993.

APÊNDICE A IMPLEMENTAÇÃO

Como já foi dito,os conceitos aqui expostos seriam implementados a fim de pô-los em prática. Para tal, foi utilizada a API (*Application Programming Interface*) Java 3D, um poderoso ambiente de desenvolvimento de aplicações em 3D desenvolvido para a linguagem Java.

Foi implementada uma API para a aplicação de operações booleanas em sólidos b-rep, concebida com base na descrição contida na seção 4. Ela é bastante eficiente e fácil de usar, podendo ser utilizada por qualquer aplicação que faça uso da API Java 3D. Basicamente, funciona da seguinte maneira: dois sólidos do tipo Solid (uma especialização do Shape3D) são passados como parâmetros para a biblioteca. Em seguida, solicita-se dela o sólido relativo à aplicação nos dois sólidos de uma das operações booleanas disponíveis: união, interseção e diferença. O sólido obtido pode ser utilizado de duas formas: ele pode ser inserido diretamente em um grafo de cena (por ser um tipo de Shape3D) ou pode-se obter suas coordenadas para se fazer outro uso delas.

Foi implementada também uma aplicação para se aplicar operações booleanas em primitivas baseada na metodologia apresentada na seção 3, que combina a geometria construtiva de sólidos com a representação b-rep. Através dela, pode-se criar sólidos compostos por combinações de primitivas de forma eficiente e intuitiva. Os sólidos apresentados nas figuras ao longo do texto foram gerados a partir dessa ferramenta. Sua implementação foi baseada nas estruturas de representação e visualização de sólidos do Java 3D, que lidam com sólidos em b-rep. Os recursos da geometria construtiva de sólidos foram adaptados a essa estrutura, fazendo uso da coleção de componentes gráficos Swing para a exibição da árvore CSG e dos parâmetros armazenados nos seus nós. A API que realiza operações booleanas em sólidos b-rep é utilizada pela aplicação para o uso de tais operações quando necessário.

Quatro primitivas podem ser criadas: cilindro, cone, elipsóide e sólido retangular. A figura A.1 mostra as primitivas disponíveis e o painel para criação de um sólido retangular, que contém os parâmetros para sua criação. Há um desses painéis para cada uma das primitivas.

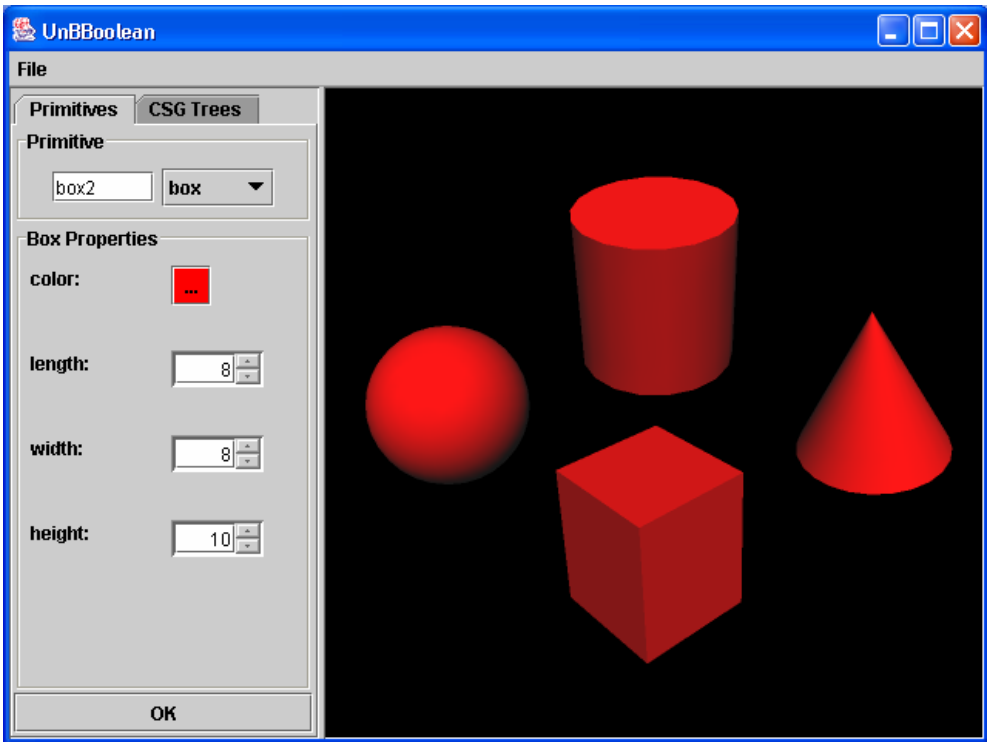

Figura A.1 Primitivas disponíveis na aplicação

Quando um sólido é selecionado, sua árvore CSG é apresentada ao usuário. Através dela, é possível obter todos os parâmetros utilizados para compor um sólido. Os nós internos apresentam diretamente a operação booleana que a eles corresponde, e os nós folhas os nomes das primitivas utilizadas. Quando selecionadas, apresentam seus parâmetros de criação. Estes podem ser modificados, basta que os valores apresentados sejam modificados e a operação confirmada pelo uso do botão OK. O mesmo pode ser feito para as operações: um nó pode ser selecionado e sua operação modificada. A figura A.2 apresenta uma xícara modelada pelo uso de cilindros e um sólido retangular. Sua árvore é apresentada à esquerda. Um nó folha foi selecionado, fazendo com que os parâmetros da primitiva correspondente fossem apresentados. A figura A.3 apresenta o mesmo modelo tendo os parâmetros da primitiva selecionada modificados.

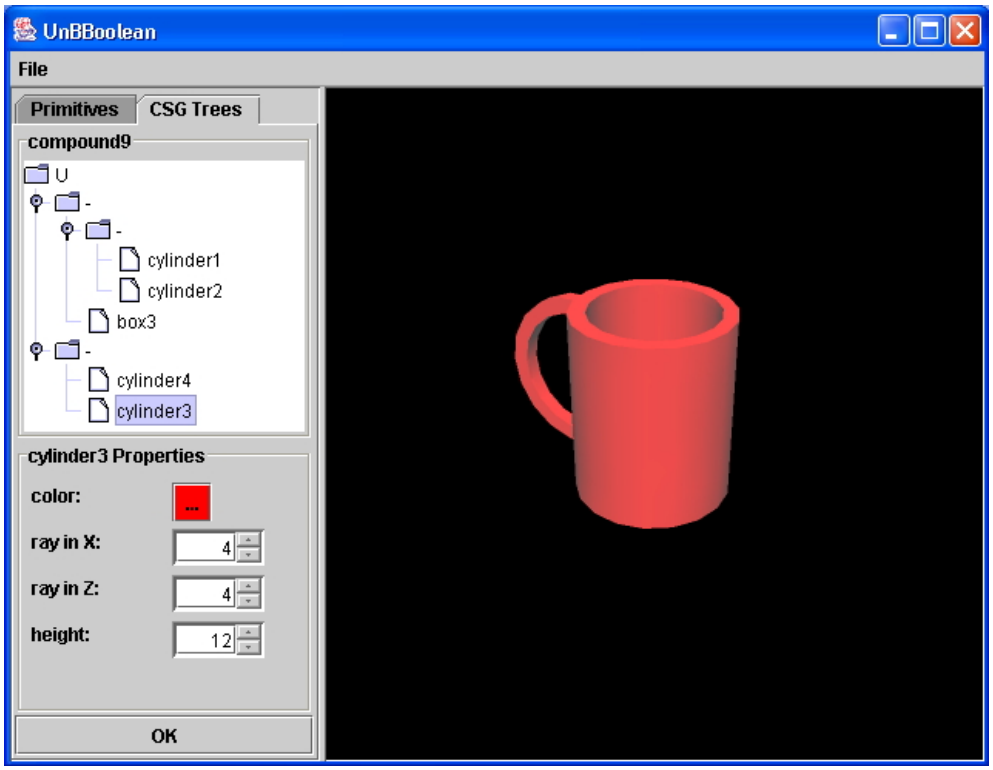

Figura A.2 Tela apresentando xícara modelada e dados relativos à mesma

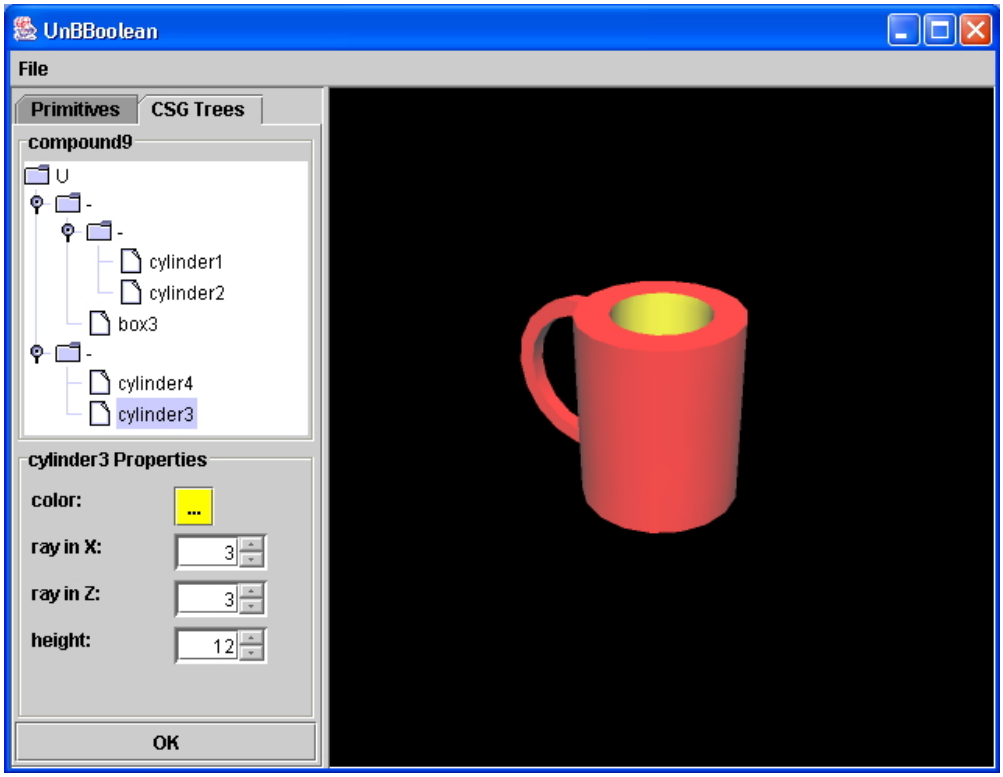

Figura A.3 Tela apresentando xícara modelada da figura A.2 modificada e dados relativos à mesma

Por intermédio da árvore CSG pode-se ainda mover primitivas utilizadas para compor um sólido e copiar sólidos relativos a qualquer nó.

Sólidos podem ser compostos de forma bastante intuitiva. Para tal, os dois sólidos a serem combinados devem ser selecionados. Em seguida, deve-se escolher a operação a ser realizada e confirmar a operação pelo uso do botão OK. A figura A.4 apresenta dois sólidos que foram selecionados. À esquerda, são apresentadas as árvores CSG relativas a cada um deles e um painel onde é selecionada a operação booleana a ser aplicada nos sólidos selecionados. A figura A.5 apresenta o sólido gerado a partir da operação booleana aplicada. Sua árvore CSG é apresentada à esquerda. Os sólidos operandos foram excluídos.

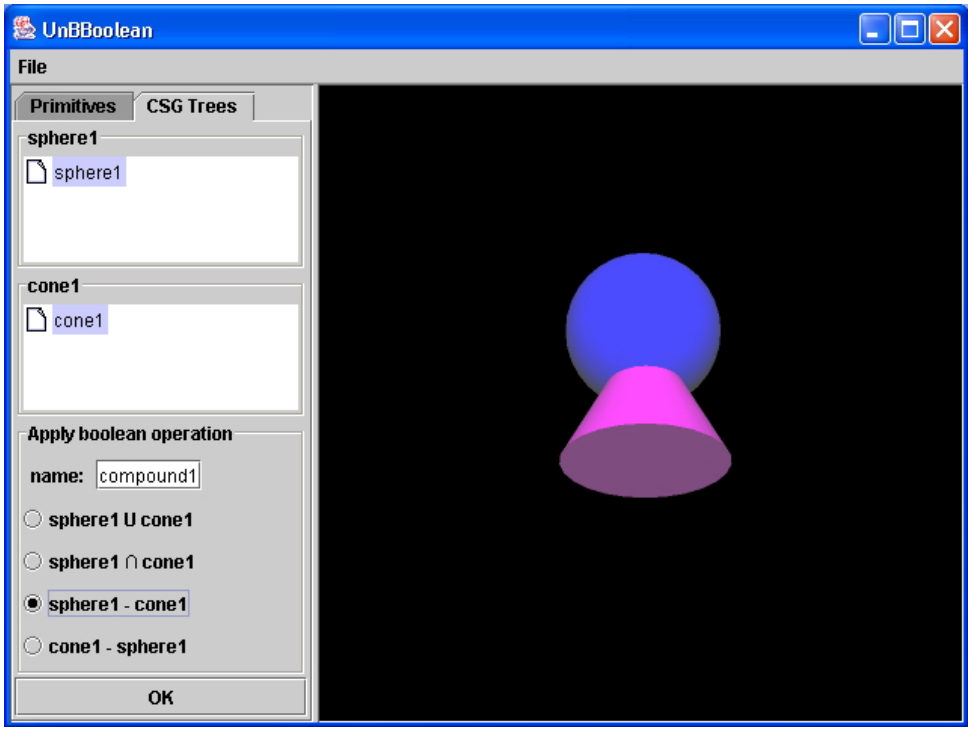

Figura A.4 Tela apresentando dois sólidos selecionados

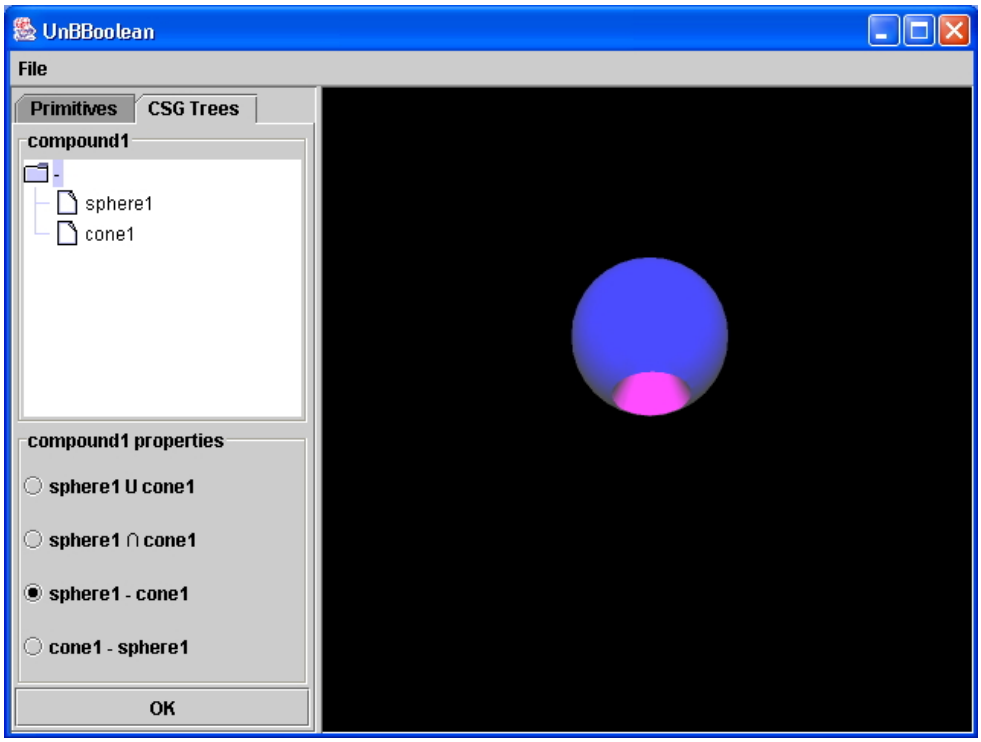

Figura A.5 Tela apresentando sólido resultante da diferença aplicada entre os sólidos da figura A.4

Sólidos são salvos pelo uso de uma estrutura baseada na Árvore CSG, composta hierarquicamente por parâmetros de composição. Assim, o consumo de memória decorrente do armazenamento de um sólido é bastante baixo. O modelo apresentado na figura 1.2, por exemplo, apesar da infinidade de primitivas envolvidas, consome apenas 5,95 kbytes quando salvo. Quando a estrutura salva for carregada, o sólido correspondente é reconstruído a partir dela.

Um sólido pode ser rotacionado e transladado interativamente pelo uso do mouse. Quando isso é feito, sua árvore, que mantém dados relativos às posições das primitivas, é automaticamente atualizada

A.1 Contribuições Futuras

Várias melhorias podem ser empregadas na implementação feita. A primeira e mais importante delas é relativa à aplicação de operações booleanas em sólidos b-rep. Eventualmente, os sólidos assim gerados são compostos por uma quantidade excessiva de polígonos, muito além do que seria necessário para representá-los. A figura A.5 mostra um modelo gerado pelo uso de operações booleanas de sólidos. Nele, há uma quantidade

desnecessária de polígonos representando um plano. A solução para isso seria aplicar um algoritmo para redução de polígonos nas faces coplanares dos sólidos gerados por operações booleanas. Existem várias abordagens para tal, sendo [MEL98] um bom ponto de partida.

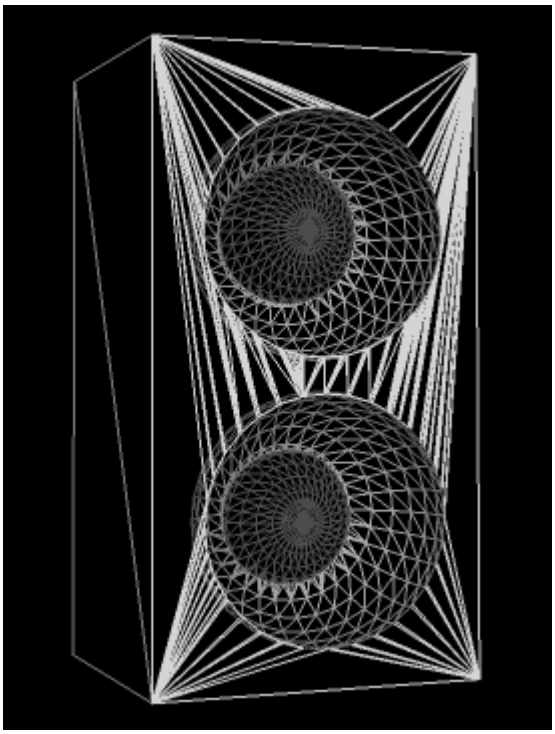

Figura A.6 Sólido gerado por operação booleana representado por um número excessivo de polígonos (exibição em arame)

Caso houvesse o interesse em gerar imagens complexas a partir dos sólidos gerados, seria interessante que se adicionasse à aplicação como recurso a aplicação de *ray-tracing*. A estratégia a ser utilizada seria aquela citada na seção 2.2 para a renderização de sólidos CSG que fazem uso de semi-espaços na representação interna. Para tal, os nós folhas deverão armazenar parâmetros adicionais a serem utilizados na composição das primitivas por semi-espaços. A fim de enriquecer as imagens geradas, deveriam ser considerados fatores como sombreamento, reflexão e transparência no ato de sua implementação.

Outros recursos seriam bastante úteis para fazer da aplicação criada uma robusta ferramenta de modelagem. Dentre elas, enquadram-se mecanismos mais precisos para movimentar um sólido, a visualização a partir de várias câmeras e a possibilidade de movimentá-

las. Seria também interessante que houvesse uma maior disponibilidade de primitivas, a fim de aumentar as possibilidades de criação.

APÊNDICE B LISTAGEM DAS CLASSES

Abaixo, segue a estrutura de classes do aplicativo final. O pacote *bool*, responsável pela aplicação de operações booleanas em sólidos b-rep, será distribuído também de forma avulsa, a fim de poder ser utilizado por outras aplicações

unbboolean
- *bool*
  - BooleanModeller
  - Bound
  - Face
  - Line
  - Object3D
  - Segment
  - Solid
  - Vertex
- *gui*
  - *save*
    - CSGFilter
    - ObjFilter
    - SaveBoxSolid
    - SaveCompoundSolid
    - SaveCylinderSolid
    - SavePrimitiveSolid
    - SaveSolid
    - SaveSphereSolid
  - *scenegraph*
    - GeneralPickBehavior
    - SceneGraphManager
    - SolidSelectionListener
  - *solidpanels*
    - BoxPanel
    - ColorChooserDialog
    - CompoundSolidPanel
    - ConePanel
    - CylinderPanel
    - InvalidBooleanOperationException
    - NewCompoundSolidPanel
    - PrimitivePanelsManager
    - SolidPanel
    - SpherePanel
  - CSGPanel
  - CSGTreeModel
  - PrimitivePanel

UnBBooleanFrame
solids
BoxSolid
CompoundSolid
ConeSolid
CSGSolid
CylinderSolid
PrimitiveSolid
SphereSolid
UnBBooleanMain