Siovani Cintra Felipussi

Modelamento Geométrico com Operadores Booleanos: Uma Descrição Teórica do Esquema de Representação BSP

Modelamento Geométrico com Operadores Booleanos: Uma Descrição Téorica do Esquema de Representação *BSP*

Siovani Cintra Felipussi

Dissertação de Mestrado
Pós-Graduação em Matemática Aplicada
MAP-034

Modelamento Geométrico com Operadores Booleanos: Uma descrição teórica do esquema de Representação BSP

Siovani Cintra Felipussi

Dissertação apresentada ao Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto - São Paulo, para a obtenção do título de Mestre em Matemática Aplicada.

Orientador: Prof. Dr. José Márcio Machado

São José do Rio Preto
Junho de 1998

Agradecimentos

Ao Prof. Dr. Manoel Ferreira Borges Neto,
co-orientador do presente trabalho,
ao meu orientador, aos meus pais,
a minha esposa, a Lazinha e a
todos, que de alguma forma
contribuíram para esta
realização.

A Deus.

"Mesmo que a fonte seja
desconhecida ainda assim
o regato flui"

H. Poincaré

Resumo

A representação de objetos geométricos é um assunto de considerável interesse e importância para não poucas áreas que lidam com computação geométrica tais como arquitetura, engenharias mecânica e civil, bem como para a Física, quando ao tratar-se numericamente as equações diferenciais parciais da Física-Matemática, tais como as equações clássicas do eletromagnetismo e as equações de Einstein do campo gravitacional, impõe-se como etapa prévia a construção dos domínios geométricos de interesse. Como em todo processamento, a representação/estrutura de dados determina o algorítmo que será necessário para prover as operações associadas de um certo domínio semântico. Esta dissertação objetiva descrever e apresentar alguns algorítmos do esquema de representação de sólidos geométricos baseados em "BSP tree (Binary Space Partitioning Trees)" usando operadores Booleanos.

Abstract

Methods for representing geometric objects is an issue of considerable interest and importance to disciplines dealing with geometric computation such as architecture, mechanical and civil engineering as well as all the theoretical physics on topics of great interest such as geometric computation of eletromagnetic fields and numerical treatment of Einstein field equations (in both cases is required a priori the construction of geometrical domains). As in all computation, the data representation/structure determines the algorithms that are needed to provide the operations associated with a semantic domain. In this paper we describe the state of art, goals and some algorithms of the representation scheme for rigid solids, based on the dimensionless approach of Binary Space Partitioning Trees by Using the Boolean Operators.

ÍNDICE

INTRODUÇÃO

A modelagem geomética computacional abstrata desempenha um importante e crescente papel na representação de objetos físicos. O aparecimento do esquema de representação "Binary Space Partitioning" (BSP) tree na literatura deveu-se sobretudo a Shumacker et al [9]. Este método (BSP), como originalmente proposto, envolvia o desenho da simulação de um cenário geométrico particionando manualmente "agrupamentos" tais como prédios, árvores, montanhas, de tal forma que planos poderiam ser colocados entre os agrupamentos para variadas extensões. Cada agrupamento correspondia a um nó área. Após revisão do trabalho de Schumacker, Sutherland et al [8] elaboraram para cada área uma pré-computação e armazenaram-se os agrupamentos sequencialmente. Desde que o método de Sutherland requeria espaço de armazenamento da ordem de m^2 (onde m é número de agrupamentos), não ficou claro que esta aplicação apresentava vantagens. A aplicação de BSP trees introduzida por Fuchs, Kedem e Naylor [7], em muitos casos complementa o que foi apresentado por Sutherland, descreve um novo algorítmo para gerar mais rapidamente imagens realísticas em cenas 3-D compostas de polígonos, e certamente, apresenta quase todos os fundamentos teóricos da área, presentes em desenvolvimentos recentes. Uma solução foi apresentada então para um problema de superfície, e que aparentemente era mais eficiente do que as soluções anteriores, qual seja a de que muitas imagens eram geradas para algum ambiente estático. Pela aplicação de Fuchs et al, um algorítmo é facilmente implementado desde que ambas as fases, a de pré-processamento e a de geração de imagens, possam cada uma, serem declaradas em uma pequena rotina recursiva. Um provável problema desse novo método, a saber, um grande aumento do número a partir de polígonos iniciais na base de dados para o número de BSP trees, o qual apareceu na aplicação de Sutherland, não ocorre em quaisquer dos ambientes encontrados.

Recentemente uma nova aplicação foi proposta por Langstrof e Weiland [14] para descrever o esquema de representação para sólidos que está baseado em dimensões

menores de BSP trees. Este método para modelagem de estruturas complexas é a combinação de objetos primitivos como esferas ou cubos por meio de operações booleanas. Em uma Geometria Sólido Construtiva (CSG), um objeto é armazenado como uma árvore, com operadores para os nós internos, e primitivos simples para as folhas. Alguns operadores podem ser operadores booleanos. Nesta tese nos concentraremos no cálculo de problemas da forma, $\mu \otimes \phi$ nos quais μ and ϕ são "polytopes" e $\otimes$ representa o operador união, intersecção, ou diferença.

Fazemos uma comparação entre a simplicidade dos algorítmos propostos com os outros propostos anteriormente. O novo esquema de representação é não ambíguo, eficiente, e provê um domínio de representação maior, e mais adequado, para descrever um conjunto de objetos físicos.

INTRODUCTION

Abstract geometric computational modeling plays an important and increasing role in representing physical objects. The appearance of a "Binary Space Partitioning" (BSP) tree representation scheme in the literature was due to Schumacker et al [9]. This method (BSP), as originally proposed, involved the designer of a simulation scene manually positioning "clusters" such as buildings, trees, mountains, so that vertical dividing planes could be placed between the clusters to varying extents. Each cluster corresponded to an area node. After reviewing the work of Schumacker, Sutherland et al [8] made for each area a pre-computing and storing ordering on the clusters (as the dividing planes were not part of the scene they need through this approach not be included in the ordering). Since, Sutherland's method requires m^2 storage space (where m is the number of clusters), it was not clear whether this approach was advantageous. The application of BSP trees introduced by Fuchs, Kedem and Naylor [7], in many ways a complement to that presented by Sutherland, describes a new algorithm to more rapidly generate realistic images of 3-D scenes composed of polygons and certainly all presents the development of almost all theoretical foundations of the area. A solution has been presented to the visible surface problem which appears to be more efficient than those previous solutions whenever many images are to be generated of the same static environment. Through the approach of Fuchs et all the algorithm is easy to implement since both phases, the pre-processing and the image generation, can each be stated in a short recursive procedure. The major potential weakness, a large increase from the number of original polygons in the data base to the number in the BSP tree, which appeared in Sutherland's approach, has not occurred in any environment encountered.

Recently a new approach has been proposed by Langstrof and Weiland [14] in order to describe a representation scheme for rigid solids that is based on the dimensionless of BSP trees. This method for modeling complex structures is the combination of primitive objects like spheres or boxes by means of booleans operations. In Constructive Solid Geometry (CSG), an object is stored as a tree with operators at the

internal nodes and simple primitives at the leaves. Some of the operators may be boolean operators. In this thesis we concentrate on the evaluation of problems of the form $\mu \otimes \phi$, where μ and ϕ are polytopes and $\otimes$ represents one of the boolean operators union, intersection or difference.

We make a comparison between the simplicity of algorithms proposed through this approach, with the others proposed earlier. This new representation scheme is unambiguous, efficient and provide a domain of representation which is large enough to describe an adequate set of physical objects.

CAPÍTULO 1:

ÁLGEBRA BOOLEANA E OPERADORES

1.1 O conceito de conjunto

A noção de conjunto ou classe ou simplesmente uma coleção de objetos é geralmente considerada como o mais fundamental conceito da matemática. É também um conceito familiar, assim como a expressão "um conjunto de livros", "um rebanho de ovelhas", "uma coleção de selos" e assim por diante. Os objetos de um conjunto são chamados de membros ou elementos.

Em muitos casos, a forma mais simples de definir um conjunto seria listar todos os seus elementos, por exemplo, "o conjunto consistindo dos inteiros positivos 2, 3, 5, 7". Para decidir se um objeto pertence ou não ao conjunto em questão, basta examinar a lista.

Um conjunto pode também ser definido por algumas propriedades ou uma combinação de propriedades pelas quais nós podemos testar se um objeto pertence ou não ao conjunto, ou seja, os membros de todo conjunto exibem todas as propriedades em questão, mas não são verdadeiros para alguns objetos do conjunto. Assim, o conjunto cujos elementos são 2, 3, 5, 7 pode ser definido como "o conjunto de todos os números primos menores que 10".

Quando os elementos de um conjunto são muitos, o que é verdade no caso do conjunto dos números primos é impossível listar todos os elementos do conjunto. Entretanto, a frase "o conjunto de todos os números primos" define o conjunto completamente. Por outro lado, o conjunto consistindo da "Estrela Norte", do nome Cleópatra, e do número 13 não pode prontamente ser definido a não ser na forma de listar seus três elementos, pois é difícil definir alguma propriedade ou uma combinação de propriedades, que possam caracterizar uma correspondência entre seus elementos.

Quando um conjunto é definido por uma propriedade, a propriedade pode ser formulada com a ajuda do que chamamos de "sentença aberta", isto é, o resultado de excluir cada exemplo de um certo número de substantivos ou nomes de uma sentença ou inserir símbolos para variáveis em seus lugares. Por exemplo, consideremos a sentença aberta "x é um assíncrono, computador digital", onde a variável x pode ser substituída pelo nome de algum objeto correspondente. Se, por algum nome específico, este tornar a sentença verdadeira, o objeto correspondente pertence ao conjunto de computadores

digitais assíncronos. Se a sentença é falsa, o objeto não pertence a classe. Um outro exemplo, a sentença aberta "x é branco e x é uma rosa", define certamente um conjunto de flores. Para decidir se tais declarações são ou não verdadeiras para determinadas substituições de variáveis, nós devemos poder determinar se as propriedades estão ou não definindo o objeto em questão (de ser assíncrono um computador digital, uma rosa ser branca e assim por diante). Às vezes, este problema é não trivial, por exemplo, quando classificações biológicas estão sendo feitas.

É interessante observar que os elementos de um conjunto podem ser conjuntos. Podemos listar a medida dos três ângulos de um triângulo da forma (a,b,c) onde $a+b+c=180$. Denotemos o conjunto de todas as triplas de A. Então cada membro de A é uma tripla de números reais positivos e uma determinada tripla de números reais positivos pertence a A se e somente se a soma de todos os três números são ***180***. Assim a tripla ***(10, 15, 155)*** é membro de A, mas também é um conjunto cujos elementos são ***10, 15 e 155.***

O símbolo habitual para a propriedade pertencer a um conjunto é $\in$. Assim, nós expressamos o fato que um elemento a pertence ao conjunto A da forma $a \in A$. Se o elemento b não pertence a A, nós escrevemos da forma $b \notin A$.

A expressão "o conjunto consiste de" é habitualmente denotada como reforço. Assim ***{2, 3, 5, 7}*** denota o subconjunto consistindo dos inteiros ***2, 3, 5, 7***. A propriedade que define um conjunto pode também ser colocada para reforçar a escrita, por exemplo:

- ***{ (a, b, c) | a, b, c real: a + b + c = 180 }*** significa "o conjunto de todas as triplas, ***(a, b, c)*** de números reais cuja soma seja ***180"*** (a barra vertical significa "tais que" e os parênteses significam para "o conjunto todo")
- ***{ (x, y) | $y=x^2$; x, y real }***, significa "o conjunto de todos os pares ordenados ***(x, y)*** de números reais tais que ***$y=x^2$***"
- ***{ x | x primo, x < 10 }***, significa que: "o conjunto de todos os números x tais que x é um número primo menor que ***10***", ou seja, o conjunto ***{2, 3, 5, 7}***.

É possível para propriedades diferentes definir um mesmo conjunto. Assim, "o conjunto de todos os múltiplos inteiros de um primo" e "o conjunto de todas as diferenças de dois inteiros distintos" são de fato um mesmo conjunto, isto é, o conjunto de todos os inteiros. Nós consideraremos a propriedade equivalente para este caso.

Consideremos dois conjuntos iguais (escreveremos como =) se e somente se eles contêm exatamente os mesmos elementos.

Quando uma propriedade é tal que possua nenhum elemento, é chamada de propriedade nula e nós definimos como conjunto nulo ou conjunto zero ou conjunto vazio. Desde que dois conjuntos vazios contêm exatamente os mesmos elementos (isto é, nada) ambos são conjuntos vazios, entretanto considerados iguais. Na álgebra dos conjuntos, o conceito de conjunto vazio é análogo ao aplicado à função nula na álgebra tradicional.

Se nós indagamos se ou não os elementos de um determinado conjunto possuem alguma propriedade adicional, nós somos conduzidos ao conceito de um subconjunto de um determinado conjunto. Quando todos os elementos do conjunto A também são elementos de conjunto B, então nós dizemos que A é um subconjunto de B e que B é um supraconjunto de A. Nós representamos esta relação escrevendo $A \subset B$ que pode ser lida por "A é um subconjunto de B", ou "A está contido em B". É conveniente considerar um conjunto como um subconjunto dele mesmo, o que estaria de acordo com a definição. O conjunto vazio é um subconjunto de todo conjunto, novamente pela definição. Um caso especialmente importante é o caso no qual A é um subconjunto não vazio e distinto de B, e ainda, B contêm um elemento que não pertence a A. Então A é chamado de um subconjunto próprio de B. Se A é um subconjunto próprio de B, nós escrevemos $A \subset B$.

Alguns outros conceitos básicos podem ser encontrados em A. Ambrose e M. Lazerowitz [1].

1.2 Operações sobre Conjuntos

Vamos supor que nós desejamos considerar exclusivamente os subconjuntos de algum "superset" fixo S. Nós então chamaremos S de conjunto universo e o denotaremos pelo símbolo U. O conjunto vazio será denotado por 0. Subconjuntos próprios de U serão denotados pelas letras maiúsculas $A, B,.....$

É conveniente representar o conjunto universal por meio de ilustrações, por um retângulo plano, e representar seus subconjuntos por um oval ou por regiões

amolduradas dentro do retângulo. Tais diagramas são chamados de diagrama de Venn. Diagrama de Venn auxiliam-nos na visualização de noções fundamentais relativas a conjuntos, mas desde que muitos conjuntos não sejam regiões planas, tais diagramas não contêm provas de teoremas sobre conjuntos em geral. Eles podem, entretanto, provar contra-exemplos para provar declarações falsas.

O complemento $\overline{A}$ de um conjunto A contido em U é um conjunto de todos os elementos de U que não pertencem a A. A união $A \cup B$ de dois subconjuntos A e B de U é um subconjunto de U que consiste precisamente dos elementos que pertencem a A ou a B ou de ambos. A intersecção $A \cap B$ de A e B é um subconjunto de U que consiste de todos os elementos pertencentes simultaneamente a A e B. Este conceito é ilustrado pela região hachurada. Por conveniência os símbolos $\cup$ e $\cap$ são freqüentemente chamados de "cup" e "cap" respectivamente.

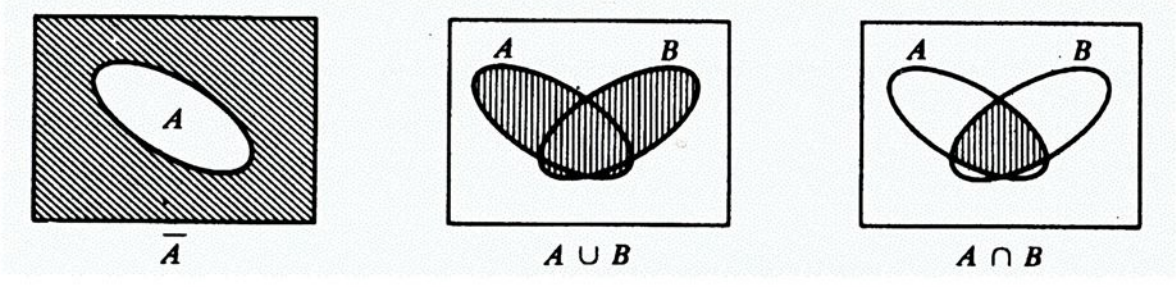


Figura 1 – Complemento, União e Intersecção de Conjuntos

Seja U o conjunto de todos os humanos vivos, A denota o subconjunto de todas as pessoas que são marinheiros, e B denota o subconjunto de todas as pessoas que já foram náufragas. Então $A \cup B$ representa o conjunto de todas as pessoas vivas que são marinheiras ou que tenham sido náufragas ou ambas, e $A \cap B$ representa o conjunto de todos os marinheiros que tenham sidos náufragos. Este exemplo pode ser interpretado pelo diagrama de Venn (Figura 2).

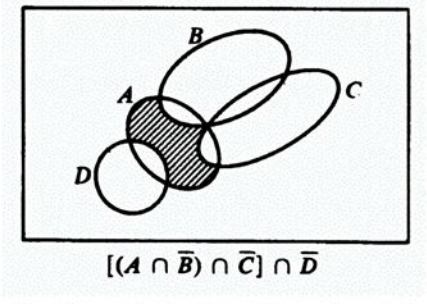


Figura 2 – Exemplo de Complemento e Intersecção

Combinações mais complicadas podem também ser representadas por um outro Diagrama de Venn. Neste caso, o conjunto D é indicado como não tendo elementos em comum com B ou C. Consequentemente, neste caso, nós temos que $B \cap D = B \cap D = \varnothing$. Quando a intersecção de dois conjuntos é um conjunto vazio, então dizemos que são disjuntos.

Como a definição e os exemplos anteriores demonstram, a união é análoga para o conectivo ou seja, se um elemento pertence a $A \cup B$ significa que o elemento está em A ou em B ou em ambos. Similarmente, a intersecção é análoga ao conectivo ou seja, se um elemento pertence a $A \cap B$, significa que ele está em ambos os conjuntos A e B. Assim, a expressão $((A \cap \overline{B}) \cap \overline{C}) \cap \overline{D}$ significa que os elementos estão em A e fora de B, C e D.

Como ilustração adicional a estes conceitos, observe que no caso de funções de três variáveis, as oito possíveis combinações de valores destas variáveis podem ser representadas como vértices de um cubo (Figura 3). Vamos tratar este conjunto de oito vértices como o conjunto universal U.

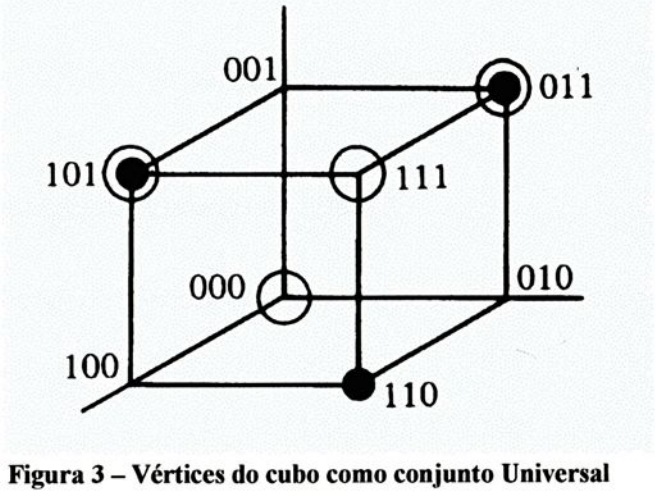


Figura 3 – Vértices do cubo como conjunto Universal

Então uma determinada função pode ser representada como um subconjunto de U, ou seja, como um conjunto de vértices correspondendo às combinações de quando a função assume valor 1. Por exemplo, a função:

$$f = x_1 \overline{x_2} x_3 \vee \overline{x_1} x_2 x_3 \vee x_1 x_2 \overline{x_3}$$

corresponde ao conjunto de vértices V_f onde:

$$V_f = \{101, 011, 110\}$$

Estes vértices estão negritados na figura 3. O complemento $\overline{f}$ é representado pelos vértices restantes do cubo, que seria $\overline{V_f}$ vejamos então:

$$V_{\overline{f}} = \overline{V}_f = \{000, 001, 010, 100, 111\}$$

Vamos considerar, como um segundo exemplo, a função:

$$g = x_1\overline{x_2}x_3 \vee \overline{x_1}x_2x_3 \vee \overline{x_1x_2x_3} \vee x_1x_2x_3$$

Então:

$$V_g = \{101, 011, 000, 111\}$$

Estes vértices estão circundados na figura 3.

Agora o conjunto de vértices correspondente a $f \vee g$ é facilmente descrito para ser $V_f \cup V_g$:

$$V_{f \vee g} = V_f \cup V_g = \{101, 011, 110, 000, 111\}$$

E o conjunto de vértices correspondentes a fg é encontrado para ser:

$$V_{fg} = V_f \cap V_g = \{101, 011\}$$

As três regras ilustradas neste exemplo, $V_{\overline{f}} = \overline{V}_f$, $V_{f \vee g} = V_f \cup V_g$ e $V_{fg} = V_f \cap V_g$, são tomadas para funções arbitrárias f e g de três variáveis.

A interpretação geométrica e as três regras estendem-se a dimensões mais altas. Uma função de n variáveis correspondente para um subconjunto de vértices de um cubo n-dimensional. Esta é uma interpretação útil desde que haja um certo conhecimento sobre geometria (especialmente sobre propriedades simétricas) de um ***n-cube***, e todos estes conhecimentos podem ser trazidos para influenciar o estudo de funções. De fato, é exaustivamente usado em pesquisas teóricas.

Outras operações básicas sobre conjuntos podem ser encontrados em A. A. Bennett e C. A. Bayliss [2].

1.3 Teoremas básicos relativos à Igualdade e Inclusão

Desde que a equação $A=B$ significa que os conjuntos A e B são de fato um mesmo conjunto, nós temos imediatamente que:

Se $A = B$, então para todos os conjuntos C, $A \cup C = B \cup C$ e $A \cap C = B \cap C$,

Se $A = B$ e $C = D$ então $A \cup C = B \cup D$ e $A \cap C = B \cap D$

Se $A = B$ então, $\overline{A} = \overline{B}$ e reciprocamente,

para cada equação, ambos os elementos representam na verdade o mesmo conjunto. Isto é, um pode unir iguais com iguais, interseccionar iguais com iguais e complementar iguais para obter iguais.

Por outro lado, da equação $A \cup C = B \cup C$ ou $A \cap C = B \cap C$ não necessariamente segue que $A = B$, por exemplo, seja:

$A = \{1, 2, 3\}$, $B = \{1, 2, 4\}$, $C = \{3, 4, 5\}$ $D = \{1, 2, 5\}$

Então:

$A \cup C = B \cup C = \{1, 2, 3, 4, 5\}$, mas $A \neq B$, também para $A \cap D = B \cap D$, mas novamente $A \neq B$. A implicação destes exemplos é que não há uma lei geral de cancelamento para união e intersecção.

Vamos examinar algumas propriedades da relação de inclusão. *Maiores detalhes e outras propriedades são encontradas em P. R. Halmos [3] e R. Sikorski [4].*

Desde que o vazio não contêm elementos, é correto dizer que todo elemento de um conjunto vazio é um elemento do conjunto A, e desde que todo elemento de um conjunto A de U é um elemento de U, nós temos a "propriedade de fronteira universal".

Para todo subconjunto A de um conjunto universal U,

$$\varnothing \subseteq A \subseteq U$$

Desde que todo conjunto A é um subconjunto dele mesmo, temos então a propriedade *reflexiva*:

$$A \subseteq A$$

Se todo elemento de A pertence a B e todo elemento de B pertence a A, então A e B contêm precisamente os mesmos elementos. Está formalizada a propriedade *anti-simétrica*:

$$\text{Se } A \subseteq B \text{ e } B \subseteq A \text{ então } A = B$$

A propriedade *anti-simétrica* é freqüentemente usada para provar que dois conjuntos são iguais: Nós, somente temos que mostrar que cada conjunto é um subconjunto de outro, assim provaremos que eles são de fato um mesmo conjunto.

Se todo elemento de A pertence a B e todo elemento de B pertence a C, então todo elemento de A pertence a C. Esta é conhecida como propriedade *transitiva*:

$$\text{Se } A \subseteq B \text{ e } B \subseteq C \text{ então } A \subseteq C$$

As próximas duas leis dão-nos o princípio da consistência:

$$A \subseteq B \text{ se e somente se } A \cup B = B$$

$$A \subseteq B \text{ se e somente se } A \cap B = A$$

Para provar a primeira destas, note que se todo elemento de A é também um elemento de B, então o conjunto $A \cup B$ contêm todos os elementos de B mas não contêm elementos que não estejam em B. Consequentemente $A \cup B = B$. Reciprocamente, Se $A \cup B = B$, então a união de A e B não contêm elementos que não estejam em B, e que todo elemento de A pertence a B. Consequentemente $A \subseteq B$.

No caso 2, se todo elemento de A pertence a B, então o conjunto de elementos comuns a A e B é justamente A. Isto é, $A \cap B = A$. Reciprocamente, se $A \cap B = A$, então o conjunto de elementos comuns a A e B é justamente A, tal que $A \subseteq B$.

Estes testes para inclusão são claramente visualizados pelo diagrama de Venn (Figura 4).

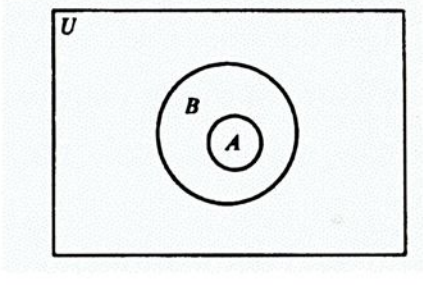


Figura 4 – Princípio da Consistência

1.4 Teoremas básicos relativos à União e Intersecção

As *Leis Comutativas* de união e intersecção são válidas para todos os subconjuntos de A e B de U, isto é:

$$A \cup B = B \cup A$$

$$A \cap B = B \cap A$$

Desde que, pela definição de $\cup$ e $\cap$, em cada equação ambos os elementos contêm exatamente os mesmos elementos.

Seja $A \cup B \cup C$ e $A \cap B \cap C$ respectivamente, denotamos o conjunto de elementos pertencentes a pelo menos um dos conjuntos ***A, B*** e ***C*** (união de ***A, B*** e ***C***) e o conjunto de elementos pertencentes simultaneamente a todos os três conjuntos ***A, B,*** e ***C*** (a intersecção de ***A, B*** e ***C***). Então para todo subconjunto ***A, B*** e ***C*** de ***U***, nós temos as *Leis Associativas* para união e intersecção:

$$A \cup B \cup C = (A \cup B) \cup C = A \cup (B \cup C)$$

$$A \cap B \cap C = (A \cap B) \cap C = A \cap (B \cap C)$$

De fato, todo elemento de $A \cup B \cup C$ pertence a $(A \cup B) \cup C$ desde que cada elemento pertença a pelo menos um subconjunto ***A, B*** ou ***C***. Analogamente, todo elemento de $(A \cup B) \cup C$ pertence a alguém de $A \cup B$ ou ***C***, consequentemente a pelo menos um subconjunto ***A, B*** ou ***C*** o que implica que pertença a $A \cup B \cup C$. Então, por

$$A \cup B \cup C \subseteq (A \cup B) \cup C$$

e

$$(A \cup B) \cup C \subseteq A \cup B \cup C$$

então:

$$A \cup B \cup C = (A \cup B) \cup C$$

As equações restantes são provadas de forma análoga.

Consideremos os elementos de $A \cap (B \cup C)$. Eles pertencem a A e a alguém de B e C, ou seja, pertence a $A \cap B$ ou a $A \cap C$ ou a ambos. Então, $A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C)$, reciprocamente, todo elemento de $(A \cap B) \cup (A \cap C)$ pertence a alguém de $A \cap B$ ou $A \cap C$, consequentemente para ***A, B*** ou ***A, C***. Isto é, pertence a A e a $B \cup C$, ou seja, a $A \cap (B \cup C)$. Assim, $(A \cap B) \cup (A \cap C) \subseteq A \cap (B \cup C)$, que nada mais é do que a primeira *Lei Distributiva*:

Para todo subconjunto A, B e C de U,

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

A Figura 5 exemplifica tal contexto:

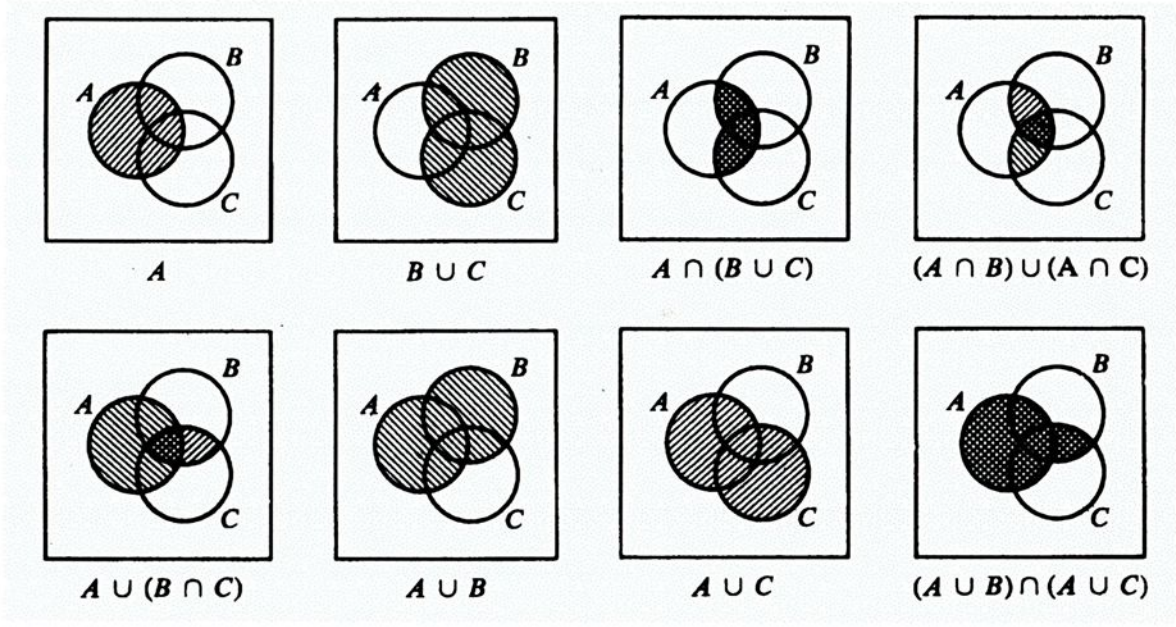


Figura 5 – Ilustrações referenciadas aos Teoremas de União e Intersecção

As *Leis Idempotentes,* são válidas para todo subconjunto A de U, tal que:

$$A \cup A = A$$

$$A \cap A = A$$

segue imediatamente da definição de união e intersecção.

Analogamente, as definições de $\cup, \cap, \varnothing$ e U implicam nas leis de operações com $\varnothing$ e U:

$$A \cup \varnothing = A,$$

$$A \cap U = A,$$

$$A \cup U = A,$$

$$A \cap \varnothing = \varnothing,$$

$$\overline{\varnothing} = U,$$

$$\overline{U} = \varnothing.$$

De fato, é imediato, que em cada igualdade, o conjunto equiparado tem precisamente os mesmos elementos.

Vale ressaltar, que $\varnothing$ atua como 0 na adição: $a + 0 = a$, enquanto o U atua como 1 na multiplicação $a \cdot 1 = a$. Ou seja, $\varnothing$ é o elemento identidade com respeito à união enquanto U é o elemento identidade com respeito à intersecção

No segundo par destas leis, $\varnothing$ atua como 0 na multiplicação: $a \cdot 0 = 0$. Dizemos que U e $\varnothing$ são elementos dominantes com respeito às operações de união e intersecção, respectivamente.

Desde que todo elemento de U pertence a A ou para $\overline{A}$, onde A é algum subconjunto de U, e desde que A e $\overline{A}$ tenham por definição nenhum elemento em comum, temos então a *Lei da Complementação*:

Para um subconjunto A de U, temos que:

$$A \cup \overline{A} = U,$$

$$A \cap \overline{A} = \varnothing.$$

Leis de DeMorgan:

Para todo subconjunto A e B de U,

$$\overline{A \cup B} = \overline{A} \cap \overline{B},$$

$$\overline{A \cap B} = \overline{A} \cup \overline{B},$$

Consideremos a primeira equação. Ela simplesmente diz que se um elemento de U não pertence a um conjunto A ou B, então ele não pertence a A e também não pertence a B, e reciprocamente. A segunda lei diz que se um elemento não pertence a ambos A e B, então ele não pode pertencer a qualquer um dos dois e reciprocamente. É útil aprendermos as formas verbais para estas leis: O complemento da união é a intersecção dos complementos separados e o complemento da intersecção é a união dos complementos separados.

Lei da Involução:

Para todo $A \subseteq U$, $\overline{\overline{(A)}} = A$.

Ou seja, todo elemento de U, que não pertence ao complemento de A, pertence ao próprio A.

1.5 A Álgebra de subconjuntos de U como uma Álgebra Booleana

Nós definimos a Álgebra Booleana como um conjunto B para o qual são definidas as relações $=, \leq$ (aqui $\subseteq$), e as três operações $\vee, \bullet, \bar{}$, (aqui $\cup, \cap, \bar{}$) que contêm dois elementos distintos 0 e 1, (aqui $\varnothing$ e U), e que satisfazem certas leis básicas. Com as trocas de notação indicadas aqui, estas mesmas leis são válidas para o conjunto de subconjuntos de um conjunto universal. Ou seja, a álgebra de subconjuntos de um conjunto universal é uma Álgebra Booleana, da mesma maneira o conjunto $\{0,1\}$ e o conjunto de funções de n variáveis.

1.6 O Princípio da Dualidade; algumas identidades úteis

Tem-se uma lista de leis bastante sólidas de forma que é possível manipular expressões até certo ponto semelhantes aos da álgebra ordinária e assim provar úteis e interessantes identidades em seu mais puro e formal estado. Um exemplo é a *Lei de Absorção.*

Para todo subconjunto A e B de U:

$$A \cup (A \cap B) = A$$

e

$$A \cap (A \cup B) = A$$

A primeira equação pode ser provada como:

$A \cup (A \cap B) = (A \cap U) \cup (A \cap B)$,
$= A \cap (U \cap B)$,
$= A \cap U$,
$= A$.

A segunda pode ser provada analogamente:

$A \cap (A \cup B) = (A \cup \varnothing) \cap (A \cup B)$,
$= A \cup (\varnothing \cap B)$,
$= A \cup \varnothing$,
$= A$.

Note que essas leis, exceto a "*Lei da Involução*", ocorrem em pares e que cada lei de um par é obtida em recorrência da outra,

1. Permuta da união e intersecção, e
2. Permuta de $\varnothing$ e U

O símbolo de complemento permanece intacto. Esta situação é descrita afirmando que as leis aparecem em pares duais. Porque não envolve união nem intersecção, nem $\varnothing$ nem U. Quando as permutas descritas são feitas, então permanece-se inalterado. Isto é descrito dizendo que é um *auto-dual*.

Supomos que podemos provar um teorema pela aplicação de certas leis (*Lei Comutativa*), pela *Lei de Involução*. Então se em cada passo nós usamos as leis duais em seu lugar, a conclusão tem que ser um teorema dual para aquele que nós provamos no princípio. Ou seja, para todo teorema que nós provemos, nós saberemos imediatamente se o teorema dual está correto, sem que nenhuma outra prova adicional seja necessária. Esta dupla e efetiva prova é conhecida como Princípio da Dualidade. Desde que $A \cap (A \cup B) = A$ é um teorema dual de $A \cup (A \cap B) = A$, sua prova implica em $A \cap (A \cup B) = A$ sem maiores considerações. Notemos que sobre estas provas empregamos dois passos duais.

De $A \cup (A \cap B) = A$ podemos esperar que $A \cup (A \cap B) = A = A \cup \varnothing$ o que deve implicar em $A \cap B = \varnothing$. Entretanto, em muitos casos $A \cap B \neq \varnothing$. Consequentemente, nós não podemos efetuar cancelamentos arbitrários no caso da operação união. De maneira análoga, $A \cap (A \cup B) = A = A \cap U$, que deve implicar em $A \cup B = U$, mas novamente, alguns exemplos simples nos mostram que $A \cup B \neq U$. Logo, não podemos efetuar cancelamentos arbitrários no caso da operação intersecção.

Outra identidade útil e seu dual é:

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

e

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

Prova:

$$A \cup (\overline{A} \cap B) = (A \cup \overline{A}) \cap (A \cup B),$$
$$= U \cap (A \cup B),$$
$$= (A \cup B) \cap U,$$
$$= A \cup B.$$

Um par final de identidade dual é:

$$(A \cup B) \cap (\overline{A} \cup C) \cap (B \cup C) = (A \cup B) \cap (\overline{A} \cup C),$$

Prova:

$$(A \cup B) \cap (\overline{A} \cup C) \cap (B \cup C) = (A \cup B) \cap (\overline{A} \cup C), \cap ((A \cap \overline{A}) \cup B \cup C)$$

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

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

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

Sendo o seu dual:

$$(A \cap B) \cup (\overline{A} \cap C) \cup (B \cap C) = (A \cap B) \cup (\overline{A} \cap C),$$

com prova análoga

1.7 Mais algumas importantes propriedades da Relação de Inclusão

Da definição de intersecção, nós temos:

$A \cap B \subseteq A$, e $A \cap B \subseteq B$.

temos então que: $A \cup (A \cap B) = A$,

Da definição de união, temos:

$A \subseteq A \cup B$, e $B \subseteq A \cup B$.

temos então que: $A \cap (A \cup B) = A$

Os quais podemos visualizar pelo diagrama de Venn (Figura 6):

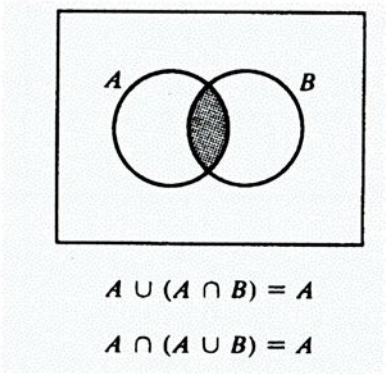


Figura 6 – Relação de Inclusão

Provemos então que:

Se $A \subseteq B$, então, para todo C, $A \cup C \subseteq B \cup C$, e $A \cap C \subseteq B \cap C$.

Como, $A \subseteq B$, nós temos que $A \cup B = B$ e $(A \cup B) \cup C = B \cup C$, usando as *Leis Comutativas e Associativas* temos:

$(A \cup C) \cup (B \cup C) = (B \cup C)$, consequentemente $A \cup C \subseteq B \cup C$.

De $A \subseteq B$, nós temos que $A \cap B = A$ e $(A \cap B) \cap C = A \cap C$, utilizando-se das *Leis Associativas e Comutativas*, nós obtemos $(A \cap C) \cap (B \cap C) = A \cap C$, Consequentemente, $A \cap C \subseteq B \cap C$.

Outro importante resultado é facilmente provado para:

$A \subseteq B$, se e somente se $A \cap \overline{B} = \varnothing$

Prova:

Seja $A \subseteq B$, então $A = A \cap B$, interseccionando ambos os lados com $\overline{B}$ temos:

$A \cap \overline{B} = (A \cap B) \cap \overline{B} = A \cap (B \cap \overline{B}) = A \cap \varnothing = \varnothing$, ou seja, $A \cap \overline{B} = \varnothing$

(volta)

Digamos que $A \cap \overline{B} = \varnothing$, então temos as equações:

$A = A \cap U = A \cap (B \cup \overline{B}) = (A \cap B) \cup (A \cap \overline{B}) = (A \cap B) \cup \varnothing = A \cap B$,
consequentemente $A \subseteq B$ (Figura 7)

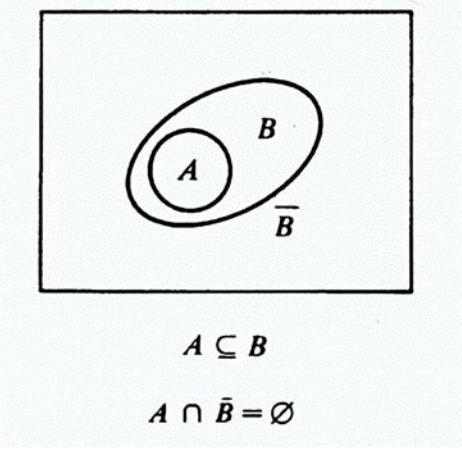


Figura 7 – Inclusão, Intersecção, Complemento e Vazio

Como exemplo final, provemos

Se $A \subseteq B \subseteq C$, então $(\overline{A} \cap B) \cup (\overline{B} \cap C) = \overline{A} \cap C$,

Prova:

De $A \subseteq B$, temos que $A \cap B = A$, consequentemente $\overline{A \cap B} = \overline{A}$, ou seja, $\overline{A} \cup \overline{B} = \overline{A}$, analogamente, se $B \subseteq C$, temos que $B \cap C = B$, vejamos então:

$(\overline{A} \cap B) \cup (\overline{B} \cap C) = [\overline{A} \cap (B \cap C)] \cup (\overline{B} \cap C)$

$= [(\overline{A} \cap B) \cap C)] \cup (\overline{B} \cap C)$

$= [(\overline{A} \cap B) \cup \overline{B})] \cap C = (\overline{A} \cup \overline{B}) \cap C = \overline{A} \cap C$ (Figura 8)

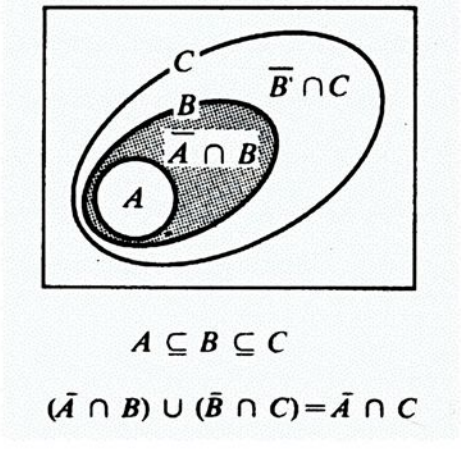


Figura 8 – Complemento, Intersecção e União

Os teoremas e exemplos deste capítulo ilustram alguns tipos de operações e argumentos que são úteis e naturais para a Álgebra Booleana.

1.8 Operações Básicas da Álgebra Booleana

Resumidamente, apresentaremos agora e nas seções que se seguem (até finalizar o capítulo 1) um tratamento abstrato da Álgebra Booleana, enfatizando-se que a Álgebra é de fato uma estrutura matemática completamente independente de suas aplicações.

Seja B um conjunto de elementos para os quais, a igualdade (=) significa identidade, ou seja, se a e b são elementos de B, então $a = b$ se e somente se a e b são de fato os mesmos elementos de B. Isto não implica, é claro, de se excluir a possibilidade de que um elemento de B pode ter mais que uma representação, uma que pode ser mais útil para um dado outro propósito. Para todo a, b, c pertencente a B, a igualdade é *reflexiva*, ou seja, $a = a$; a igualdade é *simétrica*, se $a = b$ então $b = a$, e a igualdade é *transitiva*, se $a = b$ e $b = c$ então $a = c$.

Assumimos que existem duas operações binárias, denotadas por $\vee$ e $\wedge$, e chamadas de *união* e *encontro*, respectivamente, e que cada qual possa ser aplicada para algum par ordenado de elementos de B e produzir novamente um elemento de B. O fato de que a união e o encontro de dois elementos de B produzirem novamente elementos de B é descrito dizendo que B é fechado com respeito a estas duas operações. Nós diremos $a \vee b$ como "a une a b" ou como "a união de a e b" e "$a \wedge b$" como "a encontra b" ou como "o encontro de a e b". A estas operações exige-se que sejam satisfeitas, para todos os ***elementos a, b, c*** de B, as propriedades seguintes:

Lei Comutativa:

$a \vee b = b \vee a,$

$a \wedge b = b \wedge a;$

Lei Associativa:

$a \vee (b \vee c) = (a \vee b) \vee c,$

$a \wedge (b \wedge c) = (a \wedge b) \wedge c;$

Lei Distributiva:

$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c),$

$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$;

Lei Idempotente:

$a \vee a = a$,

$a \wedge a = a$;

Assumimos que B contêm dois elementos especiais distintos, "0" (zero) e o "1" (um) que para todo elemento a de B, satisfaz:

$a \vee 0 = a$,

$a \wedge 1 = a$,

$a \wedge 0 = 0$,

$a \vee 0 = a$.

Observe, que no primeiro par acima, o elemento a sempre sobrevive de forma que 0 e 1 são os elementos identidade com respeito a união e encontro, respectivamente.

Definimos em B uma única operação chamada complementação, denotada por "$\bar{}$", que define para cada elemento a de B um complemento; $\bar{a}$, também um membro de B, que satisfaz:

Leis de Complementação:

$a \wedge \bar{a} = 0$,

$a \vee \bar{a} = 1$,

Leis de DeMorgan:

$\overline{a \vee b} = \bar{a} \wedge \bar{b}$,

$\overline{a \wedge b} = \overline{a} \vee \overline{b}$,

e finalmente, a *Lei da Involução:*

$\overline{(\bar{a})} = a$

Antes de demonstrarmos alguns teoremas básicos, devemos definir que:

Para todo a, b, c pertencentes a B, se $a = b$, então $\bar{a} = \bar{b}$,

$a \vee c = b \vee c$,

e

$a \wedge c = b \wedge c$.

Primeiro notemos que definição implica em:

$\bar{0} = 1$ e $\bar{1} = 0$.

De fato, temos:

$1 = 0 \vee \bar{0} = \bar{0} \vee 0 = \bar{0}$

$0 = 1 \wedge \bar{1} = \bar{1} \wedge 1 = \bar{1}$

Outro fato relevante é que alguns postulados são listados em pares. Cada membro de um par pode ser obtido de outro, permutando-se as operações $\vee$ e $\wedge$ e os símbolos 0 e 1. O símbolo de complemento, entretanto resta inalterado. Assim, ao longo da lista de postulados, as operações $\vee$ e $\wedge$ e os símbolos 0 e 1, desfrutam de uma certa igualdade de tratamento. Esta simetria pode ser chamada de *princípio de dualidade,* e os postulados são apresentados para aparecerem em pares duais. A Lei de Involução, é chamada de *auto-dual*. A importância do *princípio da dualidade* é que nos permite concluir imediatamente, da prova de uma determinada identidade, a verdadeira identidade dual, sem que seja necessária uma prova em separado.

Mostraremos que cada elemento a de B tem um único complemento. De fato, a tem, por hipótese, pelo menos um complemento, $\bar{a}$. Então $a \wedge \bar{a} = 0$, $a \vee \bar{a} = 1$. Supomos que x é um complemento de a de forma que nós também temos $a \wedge x = 0$, $a \vee x = 1$. Então $x = x \vee 0 = x \vee (a \wedge \bar{a}) = (x \vee a) \wedge (x \vee \bar{a}) = (a \vee x) \wedge (x \vee \bar{a}) = 1 \wedge (x \vee \bar{a}) = (a \vee \bar{a}) \wedge (x \vee \bar{a}) = (a \vee \bar{a}) \wedge (\bar{a} \vee x) = \bar{a} \vee (a \wedge x) = \bar{a} \vee 0 = \bar{a}$. Assim, $x = \bar{a}$, de forma que $\bar{a}$ é somente um complemento de a então o teorema é provado.

Aplicaremos agora o fato que se $a \wedge x = 0$ e $a \vee x = 1$, x é necessariamente $\bar{a}$, temos então:

$$(a \wedge b) \wedge (\bar{a} \vee \bar{b}) = [(a \wedge b) \wedge \bar{a}] \vee [(a \wedge b) \wedge \bar{b}] =$$

$$= [(a \wedge \bar{a}) \wedge b] \vee [a \wedge (b \wedge \bar{b})] = 0$$

e

$$(a \wedge b) \vee (\bar{a} \vee \bar{b}) = [(a \wedge b) \vee \bar{a}] \vee \bar{b} = [\bar{a} \vee (a \wedge b)] \vee \bar{b}$$

$$= [(\bar{a} \vee a) \wedge (\bar{a} \vee b)] \vee \bar{b} = [1 \wedge (\bar{a} \vee b)] \vee \bar{b} = (\bar{a} \vee b) \vee \bar{b}$$

$$= \bar{a} \vee (b \vee \bar{b}) = \bar{a} \vee 1 = 1$$

O que mostra que o único complemento de $(a \wedge b)$ é $(\bar{a} \vee \bar{b})$

1.9 Ainda sobre a Relação de Inclusão

Definimos, para elementos de B, a relação binária $\subseteq$ que é chamada de inclusão. A expressão $a \subseteq b$ é verbalizada da forma, "a está incluso em b", e pode ser definida assim:

$$a \subseteq b \Leftrightarrow a \wedge \bar{b} = 0$$

Definimos então os seguintes teoremas que atuam para todos os elementos a, b, c pertencentes a B:

Propriedade de Encapsulamento:

$0 \subseteq a \subseteq 1$,

Prova:

$0 \wedge \bar{a} = \bar{a} \wedge 0 = 0$, como $0 \subseteq a$, desde que $\bar{1} = 0$, temos que $a \wedge \bar{1} = a \wedge 0 = 0$, então $a \subseteq 1$.

Lei Reflexiva:

$a \subseteq a$

Lei Anti-Simétrica:

Se $a \subseteq b$ e $b \subseteq a$, então $a = b$

Lei Transitiva:

Se $a \subseteq b$ e $b \subseteq c$, então $a \subseteq c$

Princípio da Consistência:

$a \subseteq b$ se e somente se $a \wedge b = a$

$a \subseteq b$ se e somente se $a \vee b = b$

CAPÍTULO 2:

MODELOS MATEMÁTICOS PARA SÓLIDOS ABSTRATOS

2.1 Sólidos Abstratos

Poucos subconjuntos do espaço Euclidiano representam modelos convenientes para objetos físicos. Requicha [5] introduz a noção de sólidos abstratos que devem possuir as seguintes propriedades, e prover então o espaço de um modelador adequado:

Rigidez: Um sólido abstrato tem que ter uma forma invariante, que seja independente de sua localização ou orientação.

Dimensionalidade Homogênea: Um sólido abstrato tem que ter um interior bem definido e bordas que não tenham nenhum ponto isolado ou porções oscilantes.

"Finiteness": Um sólido abstrato tem que ocupar um volume finito.

Fechamento: A aplicação de um movimento rígido[1] e um conjunto de operações booleanas para um sólido abstrato têm que ter como resultante um sólido abstrato.

Descrição Finita: Um sólido abstrato tem que ter uma descrição finita, por exemplo, um número finito de polígonos de fronteira.

Determinação de Fronteira: A fronteira de um sólido abstrato tem que ser semi-analítica, o que permite que o interior e o exterior do sólido sejam determinados sem ambigüidades.

Em Requicha [5] é argumentado que modelos convenientes para sólidos são limitados, fechados, e são subconjuntos semi-analíticos e regulares do Espaço Euclidiano. Um conjunto é fechado se seu contorno ***bd*** é parte do próprio conjunto. Um conjunto regular ***S,*** é um conjunto que é idêntico ao fechamento (***cl***) do seu interior

[1] Movimento Rígido: Translação ou Rotação

(*int*): $S = cl(\ int\ (S)\ )$. Do ponto de vista geométrico isto significa que as bordas do sólido não contêm pontos isolados ou porções oscilantes.

2.2 Estruturas de Representação

Estruturas de representação para um sólido podem ser vistas como um sistema que produz estruturas simbólicas – os objetos físicos representados – pela aplicação sintética de regras definidas por algumas regras gramaticais ou símbolos do alfabeto. Estes símbolos são sólidos abstratos e o conjunto de todos os objetos representáveis – o alfabeto – é um *espaço modelador* ***M*** da estrutura de representação. A coleção de todas as estruturas simbólicas produzidas pela estrutura de representação é chamada de *espaço de representação* ***R***.

Matematicamente falando, uma estrutura de representação é uma relação $f: M \rightarrow R$, que estabelece uma correspondência entre o espaço modelador ***M*** no qual os elementos são sólidos abstratos e a representação do espaço ***R*** que contêm todas as representações sintáticas corretas. Note que ***M*** pode conter objetos que não são representáveis por ***f*** e que cada elemento de ***R*** não é necessariamente a imagem de um elemento de ***M*** (figura 9). O conjunto de elementos de ***R*** que é representável por ***f*** é chamado domínio ***D*** da estrutura de representação e sua imagem em ***R*** é a área ***W*** daquela representação.

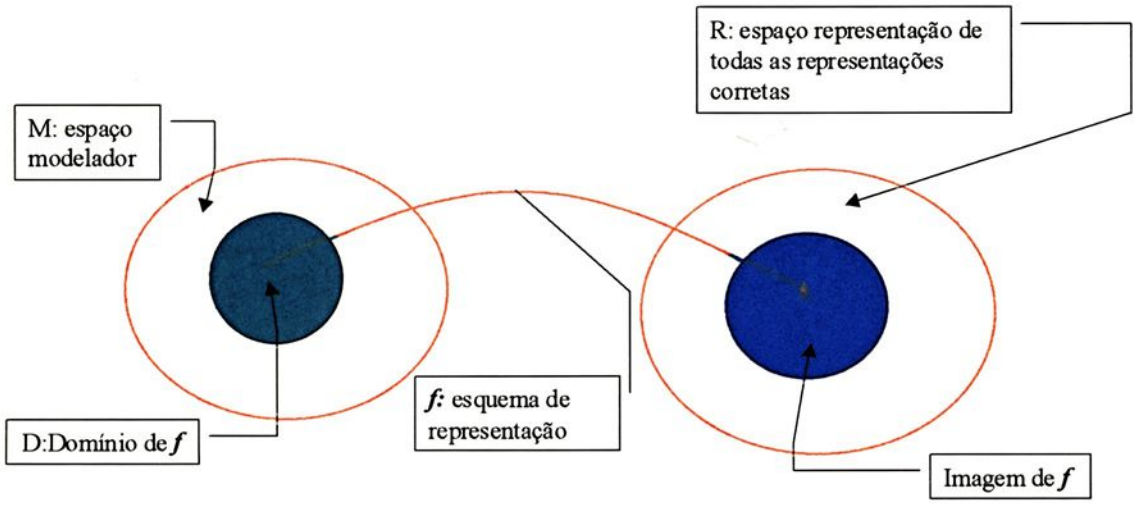


Figura 9 - Estrutura de Representação são relações entre o espaço Modelador M e o espaço R de toda representação sintática

2.3 Conjunto de Operações Regularizadas

O conjunto de operações Booleanas é um dos mais populares e intuitivos métodos para a combinação de objetos [Capítulo 1 desta dissertação]. Com as três operações básicas União ($\cup$), Intersecção ($\cap$) e Diferença ($\bar{}$), objetos de complexibilidade arbitrárias podem ser construídos a partir de primitivos simples como esferas, cubos ou cilindros. Como visto anteriormente, a aplicação do conjunto de operações booleanas para um sólido abstrato resulta em um sólido abstrato *(fechamento)*, o que assegura a validade do objetos geométricos quando estes são manipulados.

A figura 10, mostra o uso do conjunto de operações booleanas padrão para combinar dois objetos. Dependendo de sua posição, o resultado pode ser um conjunto regular válido (figura 10a), um polígono bidimensional (figura 10b), uma linha (figura 10c) ou simplesmente um ponto (figura 10d). Para obter fechamento algébrico de um conjunto regular, operações booleanas regularizadas, são usadas. Elas são denotadas como $\cup^*$, $\cap^*$ e $\bar{}^*$. Em contraste aos operadores padrões, o conjunto de operadores regularizados são definidos para retornar ao fechamento o interior do objeto resultante:

$$A \otimes^* B = cl(\text{int}(A \otimes B)),$$

na qual $\otimes$ é um operador padrão $\cup$, $\cap$ ou $\bar{}$ e $\otimes^*$ é sua versão regularizada. O conjunto de operações booleanas regularizadas produz somente conjuntos regulares quando aplicado a conjuntos regulares.

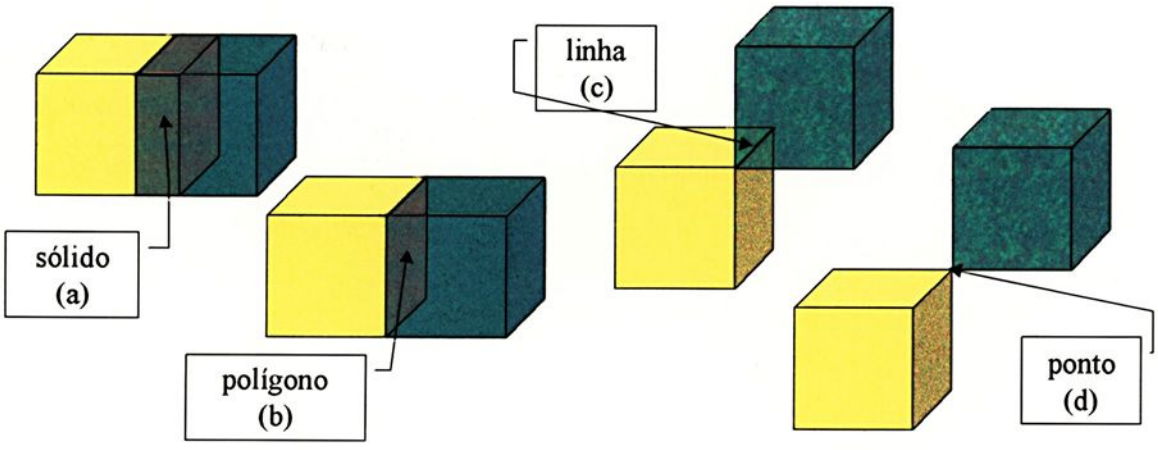


Figura 10 - Conjunto de Operações Booleanas padrão. O operador intersecção é aplicado a dois cubos. Dependendo de sua posição relativa, esta operação produz um sólido (a), um polígono (b), uma linha (c) ou um ponto (d)

2.4 Geometria Sólido Construtiva (CSG)

Modelos de decomposição descrevem sólidos como uma combinação de pequenos blocos colocados juntos, *como bem enfatizado por M. Mantyla [6].*

Os tipos de objetos fundamentais usados e o modo como a combinação destes é realizada leva a variações no esquema de representação do sólido. Todos CSM (Constructive Solid Models) consideram sólidos como pontos ou melhor, conjuntos de pontos em E^3. A idéia básica é começar a partir de um conjunto de pontos suficientemente simples que possam ser diretamente representados, e modelar a partir daí outros conjuntos de pontos em termos de uma combinação geral daqueles primeiros. Os então chamados "half-space" aplicam-se a este procedimento.

"Half-Space":

Cada conjunto de pontos A pode ser pensado como tendo uma função característica $g_a(x): x \rightarrow \{0,1\}$, para a qual afirma-se se o ponto x é um membro de A ou não:

$$g_a(x) = 1 \Rightarrow x \in A$$

$$g_a(x) = 0 \Rightarrow x \notin A$$

Para funções analíticas todos os pontos x, tais que $f(x) \geq 0$, pertenceriam ao conjunto A, enquanto $f(x) < 0$ define o seu complemento. Portanto $f(x) = 0$ divide todo o espaço em 2 subconjuntos de pontos definidos por $f(x) \geq 0$ e $f(x) < 0$, conhecidos como "Half-spaces".

Representação do Modelo CSG

CSG adota a "construção de blocos" aplicado ao sólido modelador em sua mais pura forma. O usuário de um modelador CSG opera somente sobre exemplos parametrizados de sólidos primitivos e operações booleanas. Cada primitivo, em troca, é definido como uma combinação de "half-spaces"; o usuário, porém, não tem acesso direto aos "half-spaces" individuais. Por exemplo, o usuário pode criar "half-spaces" planares e cilíndricos por meio de um primitivo cilíndrico, mas ele não pode manipulá-lo diretamente. Consequentemente o espaço modelador matemático do modelo CSG é exatamente o mesmo que o modelo "half-space" original, exceto que somente conjuntos de pontos finitos estão incluídos.

A forma mais natural de representar um modelo CSG é também chamada de "CSG-tree", que pode ser definida como segue:

< CSG tree > ::= < primitive >|
< CSG tree > < conjunto de operações > < CSG tree >|
<CSG tree > < movimento rígido >

< primitive > é um exemplo de um sólido primitivo, representado através de um tipo de identificador primitivo e uma seqüência de parâmetros de dimensão.

< movimento rígido > é uma translação ou uma rotação.

< conjunto de operações > é $\cup$ ou $\cap$ ou $^-$.

Consequentemente, primitivos são representados como "folhas" de uma "CSG tree", enquanto os nós interiores são marcados com operações Booleanas ou com um movimento rígido (figura 11). Deste modo, o conjunto de operações e movimentos são interpretados como operadores sobre "CSG trees". Isto nos leva naturalmente a criação de "CSG trees" que modela subconjuntos de um desenho, o qual depois da transformação apropriada é usado diversas vezes em uma "CSG tree" representando o desenho construído. Neste caso a árvore binária de fato se torna um gráfico acíclico direto.

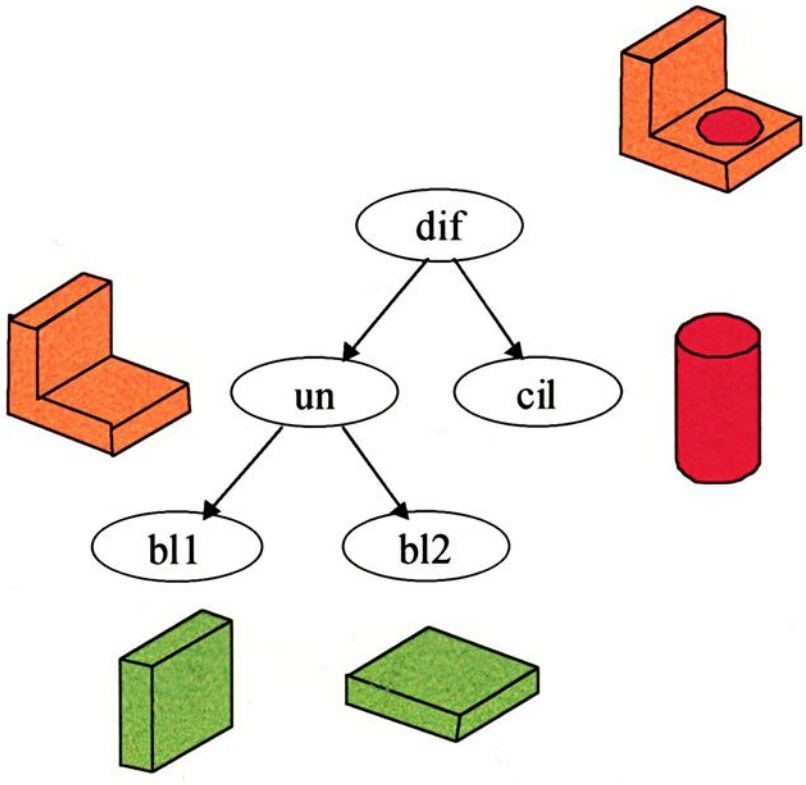
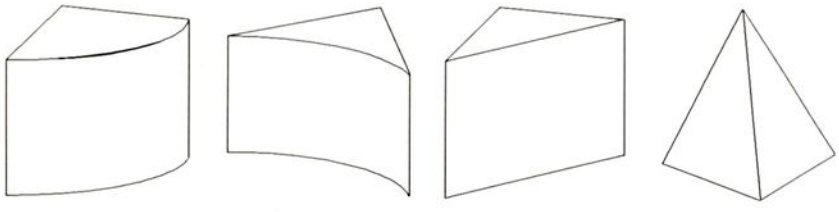
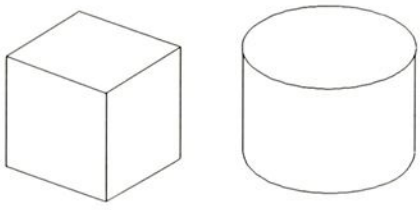


Figura 11 - Uma "CSG tree"

Cada primitivo é escolhido para definir um ponto limitado do conjunto de E^3. Como o conjunto de operações disponíveis não pode destruir as fronteiras, modelos CSG devem definir conjuntos limitados. A facilidade de uso de um modelador CSG depende muito da coleção de primitivos disponíveis. Por exemplo, a figura 12 descreve duas coleções de primitivos CSG. Note como a coleção (a) inclui várias cunhas para auxiliar no arredondamento de suas arestas, freqüentemente necessárias em partes mecânicas. Observe que todo primitivo da figura 12 pode ser expressado como uma combinação Booleana do simples "half-space" (no caso, planos e cilindros).



(a)



(b)

Figura 12 - Duas coleções de primitivos CSG

O domínio atual de um modelador CSG depende da variedade de "half-spaces" disponíveis em seus primitivos, na disponibilidade de movimentos rígidos, e na disponibilidade do conjunto de operações. Note que as duas coleções da figura 12 têm o mesmo domínio, apesar de primitivos diferentes, porque a coleção de "half-spaces" disponível é a mesma em ambos os casos (um modelador CSG com somente um primitivo cilíndrico tem exatamente o mesmo domínio).

Operações em um conjunto regular

Algumas combinações de primitivos CSG (ou "half-spaces") não satisfazem totalmente nossa noção de "Solidez". Consideremos, por exemplo, o caso descrito na figura 13. De acordo com a definição de operações de um conjunto de pontos, a intersecção de dois objetos consiste de um objeto retangular mais um segmento de linha pendente (b). Este efeito é indesejável, por exemplo, se um conjunto de operações é usado para testar a validade de um objeto gerado pela intersecção de todos os seus componentes e checando se o resultado é um objeto vazio. Neste caso, nós preferiríamos que a intersecção de meramente tocar componentes fosse vazia.

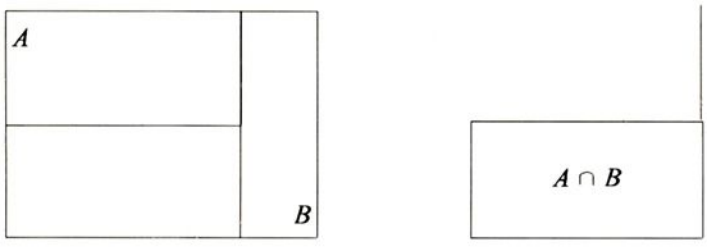


Figura 13 - Um conjunto de operações não regulares

Levando em conta a caracterização da solidez do conjunto-ponto, o problema do objeto resultante é que ele não seja regular: "linhas que sobram" violam a definição de regularidade.

O conceito de regularidade é introduzida em CSG considerando-se seu conjunto de operações ter a regularidade preservada variando das usuais operações conjunto-ponto definidas abaixo.

O conjunto de operações regularizadas união*, intersecção* e conjunto diferença*, denotados por $\cup^*$, $\cap^*$ e $\setminus^*$ são definidos como:

$$A \cup^* B = cl(\text{int}(A \cup B))$$

$$A \cap^* B = cl(\text{int}(A \cap B))$$

$$A \setminus^* B = cl(\text{int}(A \setminus B))$$

onde, $\cup$, $\cap$ e $\setminus$ denotam o conjunto de operações usuais.

Se primitivos CSG são escolhidos para serem conjuntos regulares limitados, conjuntos de operações regulares têm a propriedade desejável de ser algebricamente fechada em sua classe de conjuntos regulares fechados. Ou seja, todo "CSG tree" é garantido para definir um conjunto regular limitado.

Algoritmos para modelos CSG

O "CSG tree" pode ser visto como uma descrição implícita da geometria de sólidos modelados que deve ser "calculado" da mesma forma e ordem para criar gráficos ou executar cálculos.

As várias estruturas de uma "CSG tree" sugerem o uso de poderosos e gerais "divida-e-conquiste" aplicado ao desenho de algoritmos para "CSG trees". A idéia básica de dividir e conquistar é para dividir o problema de certa forma em duas partes, recursivamente soluciona-se cada parte, e junta-se as soluções parciais para se obter a solução total. A recursão termina quando o problema tiver sido subdivido em suas partes primitivas que permitem uma solução direta.

Quando aplicada a "CSG trees", a forma natural para subdividir o problema é processar duas subtrees para cada nó interior, denotando um conjunto de operações Booleanas de uma árvore separadamente. A recursão termina nas folhas da árvore, na qual o problema é resolvido para um primitivo. Soluções de subproblemas são combinados enquanto obtendo-se o conjunto de operações ao nó em respeito.

Divide e conquista molda para um algoritmo eficiente, se a combinação de subsoluções é simples e o problema pode sempre ser dividido em partes de aproximadamente o mesmo tamanho. Infelizmente, "CSG trees" estão normalmente longe de serem estabilizadas. O Programa 1 nos dá um algoritmo genérico divida e conquiste para processamento de "CSG trees". As seções seguintes nos dão exemplos específicos deste esquema.

```
/* Cálculo da propriedade P de uma CSG tree */
P =Tree_P(S, args)            /* onde S é o sólido primitivo */
CSG_Tree *S;                  /* onde *S é a CSG tree */
{
        if(S->op==<primitive>)
                return(Primitive_P(S, args)); /* não há mais recursão
*/
        else    return(Combine_P(Tree_P(S->Left, args),
                                 Tree_P(S->Right, args),
                                 S->Op));
            /* aplica o operador booleano para cada "ramo" da árvore */
}
/* Cálculo de P para um primitivo */
P =Primitive_P(S, args)
{

}
/* Combinação de dois cálculos de P (esquerdo e direito) com seu
respectivo operador */
P =Combine_P(Left_P, Right_P, Op)
{

}
```

Programa 1 - Divide e conquiste para uma CSG Tree

Classificação Conjunto Sociedade

O algoritmo chamado *classificação conjunto sociedade* é uma classe útil de algoritmos baseados na aplicação divida e conquista. Em geral, um algoritmo *classificação conjunto sociedade* trabalha sobre dois conjuntos de pontos, isto é, o conjunto candidato ***C*** e o conjunto referência ***R***. O algoritmo é esperado para classificar ***C*** junto a ***R*** pela formação de três conjuntos ***CinR***, ***ConR*** e ***CoutR*** representando as partes de ***C***, interna, de fronteira e externa de ***R***.

A especificação do "propósito" e a representação dos conjuntos envolvidos (***C, R, CinR, ConR, CoutR***) definem um algoritmo particular de *classificação conjunto sociedade*. Por exemplo, o algoritmo para classificar um segmento de linha finito ("aresta") junto a uma "CSG tree" é moldado dentro de uma estrutura geral pela escolha dos conjuntos que segue:

- ***C***: Uma aresta ***E*** dada em termos de uma par ordenado de coordenadas triplas;
- ***R***: Uma "CSG tree" ***S***;
- ***CinR, ConR, CoutR***: Conjuntos ***EinS, EonS, EoutS,*** cada um sendo um conjunto de subsegmentos não vazios de ***E*** tal que ***EinS*** $\cup$ ***EonS*** $\cup$ ***EoutS*** = ***E,*** e seus elementos são internos, de fronteira ou externos a ***S***, respectivamente.

O algoritmo resultante do Programa 2 bem se ajusta na aplicação geral do programa 1. A "propriedade" calculada é agora a tripla ***M = (EinS, EonS, EoutS)***. Precisamos prover um algoritmo para classificar uma aresta junto ao primitivo (Prim_M no Programa 2), e um algoritmo para combinar duas classificações (Combine_M)

```
/* Conjunto classificação sociedade de uma aresta vs. uma CSG tree */
M =Tree_M(S, E) /* onde S é o sólido primitivo e E sua aresta */
CSG_Tree *S;
Edge *E;
{
      if (S->Op==<primitive>)
            return(Prim_M(S, E));
      else  return(Combine_M(Tree_M(S->Left, E),
                        Tree_M(S->Right, E),
                        S->Op));
}
/* classificação da aresta (E) junto ao primitivo */
M =Prim_M(S, E)
```

```
{

}

/* Combinação de duas classificações das arestas E (direita e
esquerda)*/
M =Combine_M(Left_M, Right_M, Op)
{

}
```

Programa 2 - Classificação Aresta-Sólido

Classificação junto ao Primitivo

Para executar esta tarefa, a rotina Prim_M tem que calcular a intersecção (se existir) entre ***E*** e o primitivo, e subdividir ***E*** adequadamente. Por exemplo, no caso da Figura 14 o resultado desejado do primitivo de classificação é a tripla:

M = <{(p2,p3)), (***EinS***)
{}, (***EonS***)
{(p1,p2),(p3,p4)}> (***EoutS***)

Como todo primitivo CSG é uma combinação de "half-spaces", a tarefa básica necessária a ser empreendida é a classificação de ***E*** junto ao "half-space". Esta é relativamente uma operação direta.

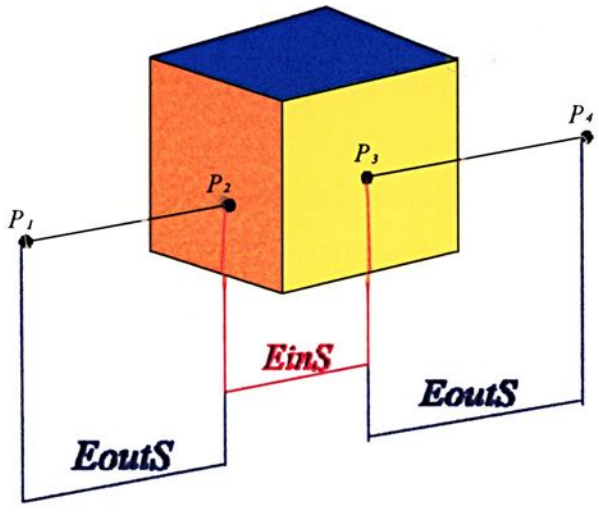


Figura 14 - Aresta versus Primitivo de bloco

Por exemplo, supomos que necessitamos classificar uma aresta E junto a um cilindro "half-space". Pela transformação E adequada, nós podemos organizar as "coisas" de forma que o cilindro seja da forma:

$$x^2 + y^2 - r^2 = 0,$$

e que E seja representado pela equação paramétrica:

$$E(t) = p_1 + t(p_2 - p_1),\ t=[0,1]$$

onde $p_i = (x_i, y_i, z_i)$ é o ponto final de E, e t é uma linha parametrizada.

Resultam as três equações:

$$\begin{aligned} x^2 + y^2 - r^2 &= 0, \\ x &= x_1 + t(x_2 - x_1) \\ y &= y_1 + t(y_2 - y_1) \end{aligned}$$

Pela substituição da segunda e terceira equação na primeira, nós obtemos uma equação de segunda ordem em t. Se a equação não tem solução real, E não intersecciona o cilindro como todo, se uma raiz dupla ocorre, E é tangente ao cilindro, caso contrário, o valor dos dois pontos da intersecção é fornecido. O que resta é checar se a intersecção ocorre dentro da aresta, isto é, se o valor está dentro de 0 e 1. No caso mais complicado, o toro, uma rotina similar fornece um polinômio de quarta ordem em t, isto é, ainda que algumas coisas possam ser resolvidas com técnicas padrão.

Combinação de Classificação: Cada resultado da classificação da aresta-primitivo (***EinS, EonS, EoutS***) consiste de uma seqüência de segmentos de arestas. Organizando estas seqüências de forma que os segmentos apareçam classificados em ordem ao longo de E, a combinação de classificações pode ser reduzida fundindo-se os conjuntos de segmentos de linha. A tarefa anterior desta operação é a combinação de dois segmentos de arestas por um conjunto de operações Booleanas Op. Para ***EinS*** e ***EoutS*** isto é moldado para regras de combinação ilustradas na figura 15.

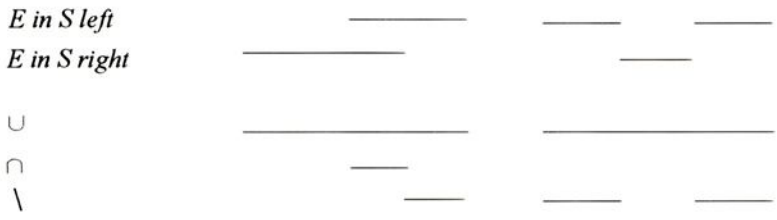


Figura 15 - Regras de Combinação

Oportunamente, as regras para combinação de segmentos de "esquerdos" ***EonS*** com os "direitos" ***EonS*** não são simplesmente isso. Como a figura 16 ilustra, as regras de combinação para estes "on-on" levam em conta a orientação de suas respectivas superfícies.

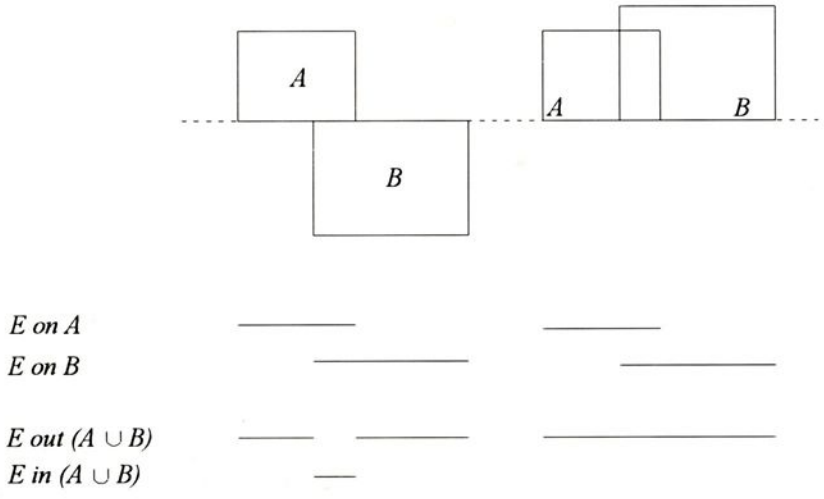


Figura 16 - Combinação de regras para casos "on-on"

Geração de Armação de arame

Como um exemplo sobre o uso de classificação de aresta-sólido, vamos considerar o seguinte problema:

Dado uma "CSG tree" ***S***, determine um conjunto de "Wire Frame" de segmentos de linhas que formam a figura "wire frame" de ***S***.

Baseado na classificação aresta-sólido, a geração do algoritmo da armação de arame pode ser combinada dos seguintes passos:

1. Geração: Gerar o conjunto de todas as tentativas de arestas pelo cálculo do conjunto das intersecções dos pares de curvas de todo "half-space" apresentados em ***S***.
2. Classificação: Classificar cada tentativa de aresta junto a ***S***, e juntá-lo ao componente ***EonS*** resultante.

```
WireFrameGen(S)
CSGTree S;
{
        /* Etapas de Geração: */
        Tentative_list= empty;
        for each half-space G of S
        {
                for each half-space H of S
                {
                   Tentative_list= union(Tentative_list, Intersec(G,
H));
                }
        }

        /* Etapas de classificação */
        Wireframe = empty;
        for each edge E of Tentative_list
        {
                <EinS, EonS, EoutS> = Tree_M(S, E);
                WireFrame = union(WireFrame, EonS);
        }
}
```

Programa 3 - Algoritmo de geração da armação de arame

Cálculo de Borda

A geração de imagens com linhas ocultas removidas, requer um algoritmo mais complicado que calcule as faces do sólido modelado via "CSG tree". Este processo é chamado de cálculo de borda. Ordinariamente, cálculo de borda é baseado sobre *classificação conjunto sociedade* entre faces primitivos (faces derivadas de um primitivo CSG) e "CSG trees". É claro, que a capacidade de cálculo de intersecções de superfícies quadráticas é necessária.

Modeladores CSG podem construir um modelo de borda a partir de um modelo CSG, o que permite, obviamente, utilizar algoritmos para modelos de borda sempre que pareçam mais apropriados que os métodos autocontidos para modelos CSG.

"Ray Casting"

"Ray Casting" é uma aproximação natural; ao invés de exatamente calcular uma "CSG tree", a árvore é testada ao longo de um número finito de linhas, consequentemente reduzindo o cálculo do conjunto de operações tridimensionais para uma unidimensional (Figura 17).

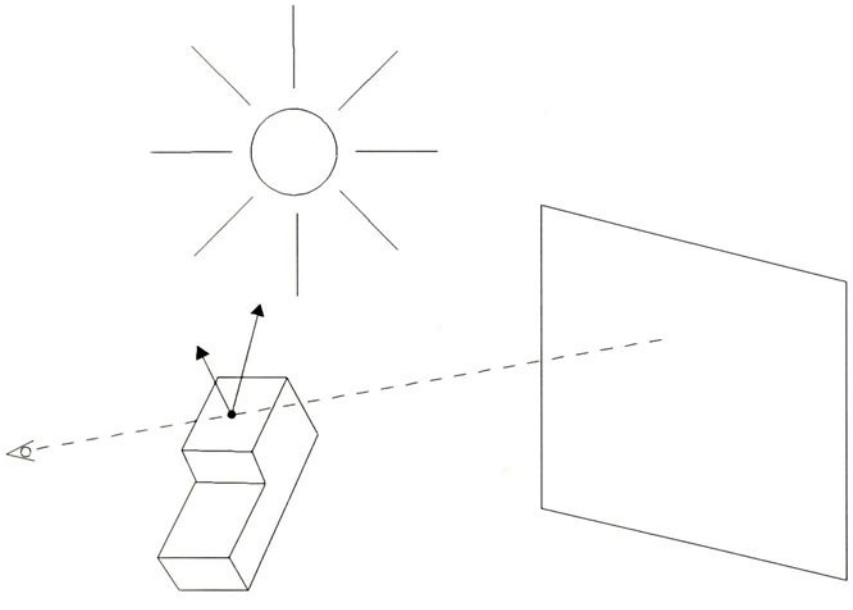


Figura 17 - O paradigma ray casting

O termo origina do problema de geração de imagem de um modelo CSG para um terminal colorido. Um "view ray" (raio de visualização) é enviado da localização do examinador através de cada pixel da imagem. Baseada na localização onde o raio primeiramente "acerta" o objeto visualizado, a sombra e intensidade do pixel é calculada. Enviando raios secundários da localização "acertada", efeitos visuais especiais tais como sombreamentos, transparências, espelhamentos podem ser criados. Este processo é chamado de "Ray Tracing."

A aplicação "ray casting" é atualmente uma variação de uma *classificação aresta-sólido*, onde ao invés de uma aresta finita, um raio semi-infinito é classificado por uma rotina de *classificação ray*. Como uma das duas formas de classificação "in-out" é suficiente, e como pode ser assumido que a localização do "olho" é externo ao sólido, a

rotina de *classificação ray* pode ter o resultado representado por meio de uma seqüência de pontos de intersecção entre o "ray" e o sólido.

Uma representação adequada é ter duas ordens:

t(1)	***t(2)***	***t(3)***	***t(4)***	....
s(1)	***s(2)***	***s(3)***	***s(4)***	....

onde os ***t(i)*** são dados como valores da intersecção dos pontos em classificação crescente, e os ***s(i)*** identidades dos "half-spaces" interseccionados com estes pontos. Os segmentos "ray" ***[0, t(1)], [t(2), t(3)]***, são considerados para que ocorram fora do sólido, e os segmentos ***[t(1), t(2)], [t(3), t(4)]*** interiores ao sólido.

A implementação de uma rotina de classificação "ray" segue linhas similares como ao da *classificação aresta-sólido*, sendo ainda mais simples. O esboço do algoritmo dado no Programa 4 indica os testes de intersecção necessários entre o raio e os primitivos "half-spaces".

```
Classification
RayCast(S, R)
CSGTree *S;
Ray *R;
{
        if(S->Op == <set operation>)
        {
                LeftClassification = RayCast(S->Left, R);
                RightClassification = RayCast(S->Right, R);
                return(Combine(LeftClassification,
                               RightClassification,
                               S->Op);
        }
        else
        {
                switch(S->Op)
                {
                        case "block"
                           do 6 ray-plane intersection tests
                        case "sphere"
                           do 1 ray-quadratic intersection test
                        case "cylinder"
                           do 2 ray-plane and 1 ray-quadratic
tests
                        case "cone"
                           do 1 ray-plane and 1 ray-quadratic test
                        case "torus"
                           do 1 ray-quartic intersection test
                }
```

```
                Classification = results of tests
                return(Classification);
        }
}
```

Programa 4 - Algoritmo classificação ray

O algoritmo de geração de imagens pode agora simplesmente selecionar a primeira intersecção da seqüência, e calcular a cor baseada nas informações de seu vetor normal, e a cor da superfície interseccionada e a localização de fontes de luzes.

"Ray casting" é útil não somente para a geração de imagens coloridas, mas também para cálculo de desenho de linhas e propriedades integradas de sólidos. O problema é que os requisitos computacionais são enormes: para calcular uma imagem de 1000^2 de resolução, um milhão de classificação "ray" são necessárias. Como cada classificação é um algoritmo recursivo, provavelmente, soluções numéricas de polinômios de quarta ordem estão envolvidas, a velocidade de um algoritmo de "ray casting" é muito distante da interação.

Propriedades Integrais

O cálculo aproximado de propriedades de integrais (tais como volume) de um modelo CSG é baseado sobre a conversão de algoritmos que (implicitamente) constroem um modelo de decomposição, aproximando-se de um sólido. A avaliação da propriedade, então reduz o cálculo da contribuição de cada célula da decomposição para o resultado total.

O algoritmo de conversão é geralmente baseado sobre o *conjunto classificação sociedade* e a aplicação *"ray casting"*. Várias alternativas de arranjos de células são possíveis, cada, correspondendo a um algoritmo de *conjunto classificação sociedade* em particular (Figura 18).

Por exemplo, o caso da "coluna de decomposição" corresponde com o *"ray casting"*. Uma coluna é determinada pelo envio de um raio ao longo de seu centro, onde o raio incide sobre o sólido, uma coluna sólido é criada. Este algoritmo é usado em modeladores de sólido empregando a aplicação "ray casting".

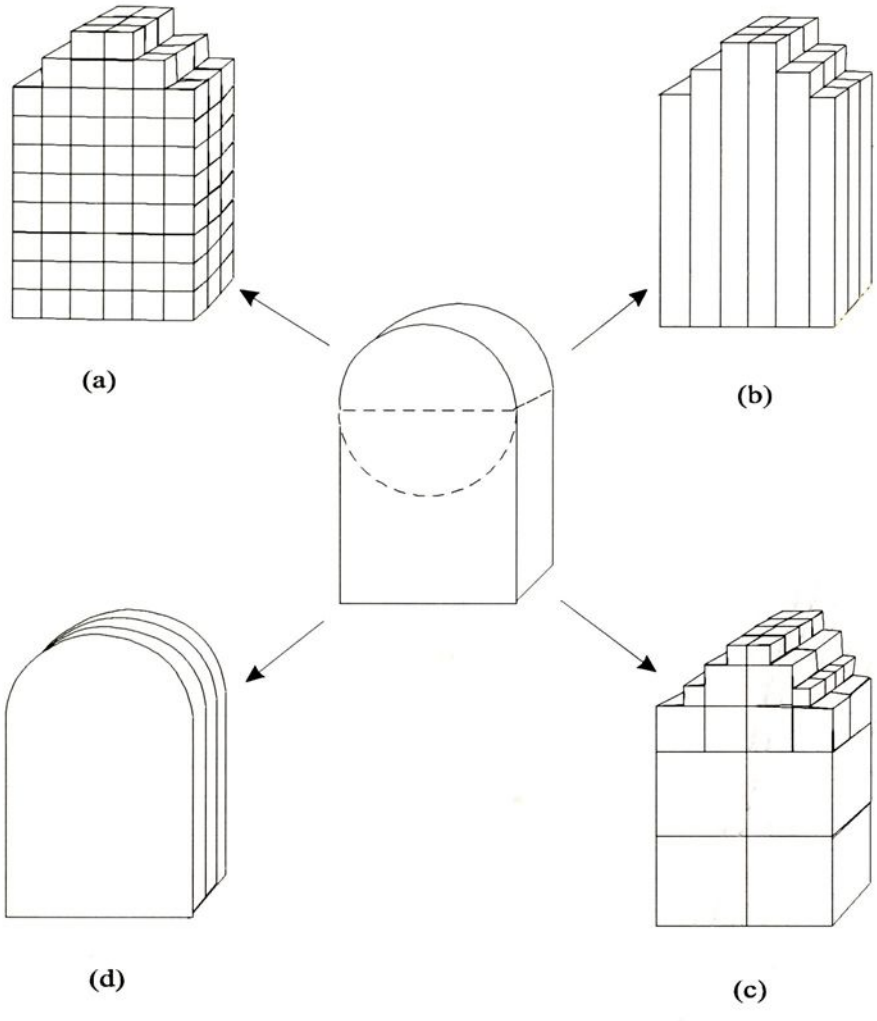


Figura 18 - Conversão para células CSG

O caso mais simples, a enumeração exaustiva, é baseada sobre a *classificação ponto-sólido*: dado um ponto ***P*** (o ponto central da célula), determine se ***P*** é interno, na fronteira ou externo ao sólido. CSG é diretamente útil para responder este tipo de questão, como classificação, com respeito a quantia de primitivos para substituir ***P,*** para definir a função de cada "half-space". Para duas formas de classificação, as regras de combinação são simples e direta. Este método é usado em TIPS *(**T**echnical **I**nformation **P**rocessing **S**ystem for computer-aided design)*.

Primitivos CSG generalizados

Para aplicar o *conjunto classificação sociedade* e a aplicação *"ray casting"* para algoritmos CSG para um modelo particular de CSG, a tarefa essencial seria reduzir a escrita das rotinas que executam as operações de base destes algoritmos. Qualquer primitivo para as quais tais rotinas podem ser escritas, podem potencialmente ser incluídas em um modelador CSG. Em particular, não há necessidade de restringir primitivos CSG para ser simplesmente coleções de "half-spaces".

Alguns sistemas baseados em CSG seguem esta aplicação, e permitem a inclusão de primitivos CSG generalizados de vários tipos em seus modelos CSG. Por exemplo, primitivos poliédricos podem ser adicionados em um modelador *"ray casting"* oferecendo condições suficientes para sua criação, e um algoritmo *"ray casting"* para um poliedro arbitrário. Esta aplicação lida como um híbrido de CSG e aplicações de modelamento de fronteira.

Outra potencialmente útil adição, é a inclusão de primitivos "sweeping" (varredura). Um primitivo de translação "sweeping" é definido por um conjunto base, um subconjunto limitado de um plano, e uma direção "sweeping": informalmente, o primitivo consiste de um conjunto de pontos "swept" pelo conjunto base como se movesse ao longo da direção "sweeping". Um primitivo rotacional "sweeping" é definido pelo conjunto base e pelo eixo de rotação, e consiste de um conjunto de pontos varridos pelo conjunto base resultante de sua rotação ao redor do eixo. Obviamente, restrições admissíveis sobre o conjunto base devem ser apresentadas como garantia para que o objeto resultante seja um sólido válido (figura 19)

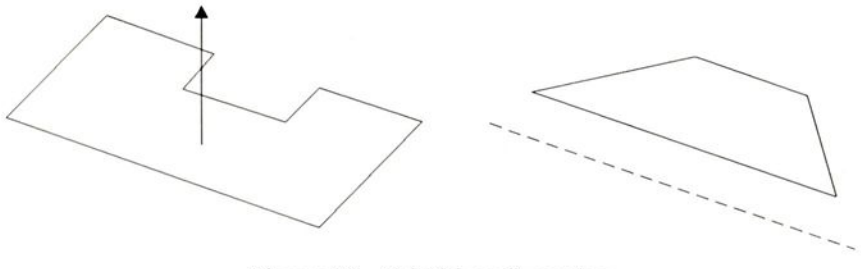


Figura 19 - Primitivos Sweeping

No contexto, o fato essencial é que estes são algoritmos, por exemplo, para o *"ray casting"* de translação e rotação primitivos "sweeping" é direto (sem a conversão para alguma outra representação). Consequentemente, eles podem ser incluídos no domínio de um modelador *"ray casting"*. Além disso, modelos de "duas-e-meia-dimensões" gerados por "sweeping" simples consistindo de linhas e arcos ao longo de um eixo normal, podem ser convertidos diretamente para modelos CSG.

Propriedades dos modelos CSG

Poder expressivo: Depende da classe de "half-spaces" disponível, não pode facilmente ser estendida para superfícies de cobertura.

Validade: Toda "CSG tree" é garantida para modelar um objeto sólido válido, desde que seus primitivos sejam válidos (isto é, conjunto regular limitado)

Unicidade e singularidade: Toda "CSG tree" modela um sólido não ambíguo. Eles não são únicos.

Linguagem de codificação: Normalmente codificação textual. É possível a inclusão de interface gráfica em um modelador CSG.

Concisão: "CSG trees" são em princípio, relativamente concisas.

Operações fechadas: Conjunto de operações são algébricamente fechadas para uma "CSG tree".

Aplicação computacional: O poder computacional de alguns importantes algoritmos CSG (tais como avaliação de borda) é pobre. Entretanto, como seus passos básicos são muito simples, eles são os primeiros candidatos para uma implementação VLSI (Very Large Scale Integration). No caso, tais como o ponto de classificação para conversão para um modelo de decomposição, a aplicação divida e conquiste geralmente lida com

algoritmos eficientes, embora sua eficiência deteriore, se a "CSG tree" está má balanceada.

Modelos CSG estão baseados sobre rigorosos conhecimentos matemáticos, bem como famílias de algoritmos disponíveis a eles, entretanto têm aplicações para muitos problemas. Dentre essas e outras razões, alguns modeladores CSG comerciais estão disponíveis e são amplamente usados.

CAPÍTULO 3:

ESQUEMA DE REPRESENTAÇÃO BSP (Binary Space Partitioning Tree)

3.1 Definição

Uma "Binary Space Partitioning tree (BSP tree)" é uma árvore binária, representando recursiva e hierarquicamente partições de um espaço *d-dimensional* por meio de hiperplanos *(d - 1)*-dimensionais. A representação computacional de objetos físicos por "BSP trees" provê uma estrutura de busca para classificação de pontos e uma representação da geometria.

Cada nó interno v de uma "BSP tree" $\dot{B}$ é associado a um hiperplano $h(v)$ e uma região $R(v)$ do espaço. Para um hiperplano:

$$H = \{(x_1,...,x_n) \mid a_1x_1 + ... + a_dx_d + a_{d+1} = 0\},$$

seja

$$H^+ = \{(x_1,...,x_n) \mid a_1x_1 + ... + a_dx_d + a_{d+1} > 0\},$$

o half-space (direito) positivo frontal do hiperplano particionado H e

$$H^- = \{(x_1,...,x_n) \mid a_1x_1 + ... + a_dx_d + a_{d+1} < 0\},$$

o "half-space" (esquerdo) negativo posterior do hiperplano particionado H.
A cada nó v, $H(v)$ divide $R(v)$ em dois "half-spaces":

$$HS(v, v.right) = R(v) \cap H^+(v),$$

que é associado a aresta de v com seu "filho" direito $v.right$, e

$$HS(v, v.left) = R(v) \cap H^-(v),$$

que é associado a aresta de v com seu "filho" esquerdo $v.left$.

$R(v) \cap H(v)$, define um sub-hiperplano $sH(v)$ que é parte da fronteira de $R(v.right)$ e $R(v.left)$.

Seja ***E(v)*** o conjunto de todas as arestas sobre o "path" das raízes das árvores do nó ***v***. ***R(v)*** é então definido como a intersecção de todos os "half-spaces" abertos nos "paths" das raízes das árvores de ***v***:

$$R(v) = \bigcap_{e \in E(v)} HS(e),$$

Para o nó raiz, ***R(raiz)*** é definido para representar todo *d-space*.

As folhas de uma "BSP tree" estão associadas com a região não particionável do *d-space* (células), no qual qualquer uma ou pertence completamente ao interior (in-cells) ou ao exterior (out-cells) do objeto representado.

Construção de uma "BSP tree"

A construção de uma "BSP tree" para um dado poliedro foi apresentada por Fuchs, Kedem e Naylor [7]. Dada uma representação de borda (B-rep) de um poliedro *d-dimensional*, convertê-lo para uma representação válida tipo "BSP tree" seria particionar seu conjunto de faces *(d-1)-dimensional* recursivamente por hiperplanos. Seja $F = \{f_1, f_2, ..., f_n\}$ o conjunto de todas as faces de fronteira. Iniciando-se com um nó BSP vazio, alguns polígonos $f_i \in F$ são escolhidos para determinar o hiperplano particionado. F é então dividido em três conjuntos: $F_{back}, F_{front}, F_{in}$, correspondentes respectivamente aos três subespaços H^-, H^+ e H. Qualquer face que cruza o hiperplano particionado é dividida e suas partes são colocadas em seu próprio conjunto.

O conjunto F_{in} é retido ao nó BSP enquanto F_{back} e F_{front} são posteriormente particionados pelo processo de repetições recursivas com os "filhos" esquerdo e direito do nó BSP respectivo. O fragmento a seguir do pseudo-code ilustra a construção de uma "BSP tree".

function Build_BSP ***(f: set of faces) -> BSP_node***
v::=new BSP node;
$F_{back}, F_{front}, F_{in}$ ***:: = conjunto de faces vazias;***

Escolha alguns hiperplanos H que tenham uma face de F;

(F_{back} , F_{front} , F_{in}) = *partição F em relação a H;*

Adiciona F_{in} a lista de faces de ***v***;

se F_{back} é vazio então

v's left son = ***IN_CELL***;

senão

v's left son = Build_BSP(F_{back});

se F_{front} é vazio então

v's right son = OUT_CELL;

senão

v's right son = Build_BSP (F_{front});

return ***v***;

end function

A figura 20 (a) mostra um objeto bidimensional e uma das diversas partições possíveis. A figura 20 (b) mostra a resultante de uma "BSP tree".

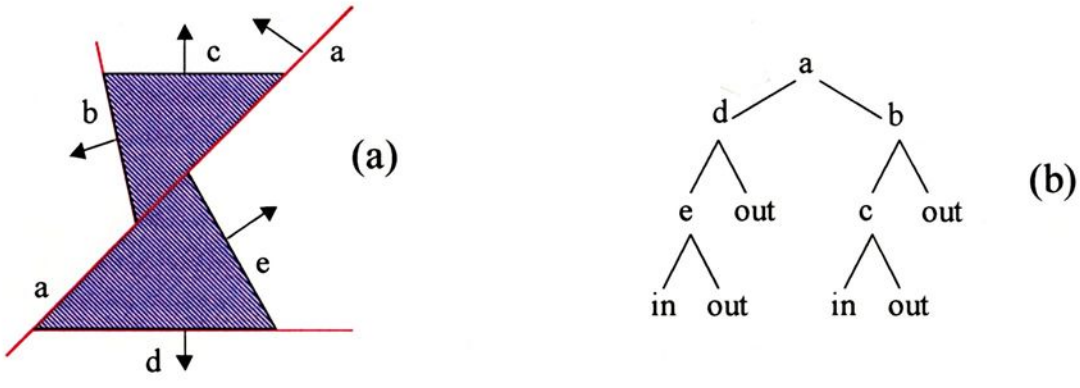


Figura 20 - Uma representação BSP tree de um objeto em 2-D. (a) A fronteira do polígono é impressa em linhas pretas, o hiperplano particionador (linhas) é vermelho e o interior do objeto está hachurado (b) O resultado de uma BSP tree

Note que a seleção de um hiperplano particionado tem uma forte influência na forma da "BSP tree". Enquanto qualquer permutação da ordem de seleção resultará em uma "BSP tree" que representará o mesmo conjunto regular, a ordem da seleção pode produzir árvores mais convenientes. Dependendo do uso adicional, qualquer ordem que resulte árvores balanceadas ou a ordem que produza árvores com poucas divisões de faces são mais desejáveis. A heurística pode ser usada para encontrar o "melhor" hiperplano: a partir de um subconjunto de ***F*** (randomicamente escolhido) encontre a face que maximiza o que é heurísticamente argumentado. O tipo concreto de argumento irá variar, dependendo do uso adicional da "BSP tree". Poder-se-ia levar em conta o número de polígonos divididos, o grau de balanceamento ou uma combinação de ambos.

3.2 Desenvolvimento Formal

Vamos agora examinar a natureza de uma "BSP tree" mais profundamente. A construção pode ser feita, de maneira idêntica, para qualquer dimensão; todavia, a tridimensional é a mais interessante para o nosso trabalho. Entretanto, é fácil explicar sua natureza ao nível bidimensional, da mesma maneira que várias estruturas geométricas emergentes podem ser claramente esboçadas; assim a discussão das três propriedades será apresentada assumindo o universo bidimensional. Precisamos definir algumas terminologias:

Segmento: um subconjunto orientado convexo e fechado ou uma linha, ou seja, um segmento finito, um raio ou uma linha, com uma direção associada a ele.

Região: um conjunto de pontos convexos e fechados de um plano (Uma região é normalmente definida como um conjunto conectado aberto)

Extensão de um segmento em uma região: dado um segmento ***s*** e uma região ***R***, definimos a extensão de ***s*** em ***R*** como sendo a intersecção da linha sobre a qual ***s*** está

situado sobre a região ***R***, obtendo o segmento $\dot{s}_R$. Assumindo $\dot{s}_R$ a direção induzida por ***s*** (indicamos pelo ponteiro da seta a direita) .

Notemos que uma região pode ser ilimitada (um plano), parcialmente limitada (um semi-plano), ou completamente limitada (um polígono finito). A motivação para definir regiões e segmentos está na maneira de como eles são, não temos interesse em fazer distinção entre conjuntos limitados, parcialmente limitados e ilimitados. Em analogia, segmentos e regiões tridimensionais são polígonos (ou alternadamente, regiões) e seções (ou volumes) respectivamente. A orientação dos polígonos correspondentes para uma notação usual de face frontal ou posterior. Vamos examinar um algoritmo geral para a construção da chamada "BSP tree":

Algoritmo I: Construção de uma "BSP tree" bidimensional:

Input: uma região ***R*** e um conjunto de segmentos Σ sobre ***R***

Output: Uma "BSP tree"

Method - chamamos a função BSPT, com ***R*** e Σ como parâmetros e $\hat{\Sigma} \leftarrow \phi$

Procedure: BSPT (R:região; conjunto de segmentos nó)

Seja Σ = ***(a, b, c)*** e ***R*** um quadrado, como na figura 21 abaixo.

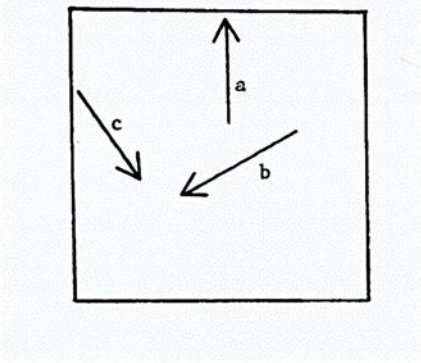


Figura 21 – A região *R* e seus segmentos Σ = (a,b,c)

Se (a) é escolhido primeiro, chegaremos então a figura 22 (a seguir), que gera a figura 23.

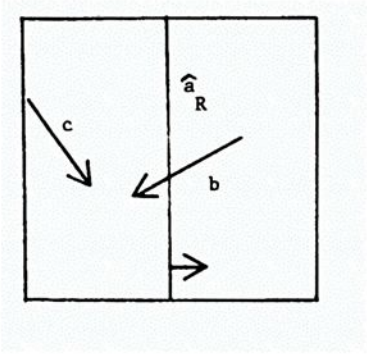


Figura 22 – O segmento (a) é escolhido primeiro

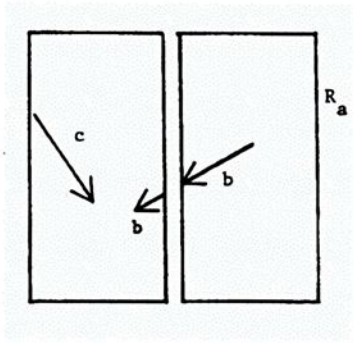


Figura 23 – Criação da região R_a

Se, a seguir, b_{R_a} é escolhido antes de c, o resultado final aparecerá como na figura 24:

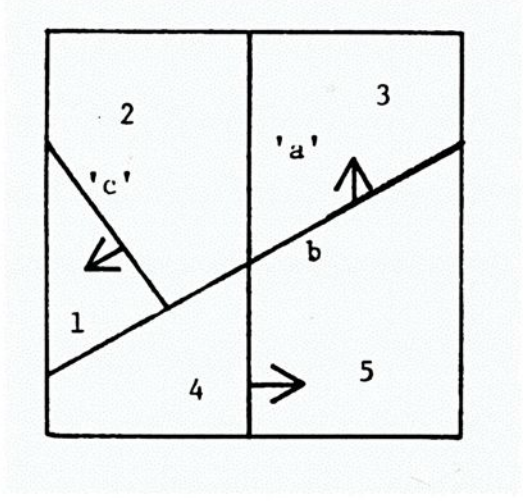


Figura 24 – A "tessellation" criada pelos segmentos $\sum$=(a, b, c)

Consideremos agora o conjunto $\hat{\sum}$ de segmentos, que está completamente dentro de ***R***. É fácil ver que a partição ***R*** é uma região convexa (polígono). Cada região, juntamente com sua borda, estará referenciando-se a uma área (volume para um espaço tridimensional). O conjunto de todas as áreas criadas pelo algoritmo estará

referenciando-se como uma "tessellation"[2]. As áreas podem ser pensadas como intersecção de semi-planos ("half-spaces" no caso tridimensional) criados pelas linhas sobre as quais permanecem os elementos de Σ ou $\hat{\Sigma}$. O propósito da orientação dos segmentos, é para efetuar a distinção entre dois semi-planos. O subscrito de cada região, gerado pelo algoritmo indica o semi-planos cuja intersecção forma a região. Como exemplo, vejamos a figura 25 que é uma "BSP tree" para a "tessellation" da figura 24 onde parênteses são usados para indicar regiões subscritas.

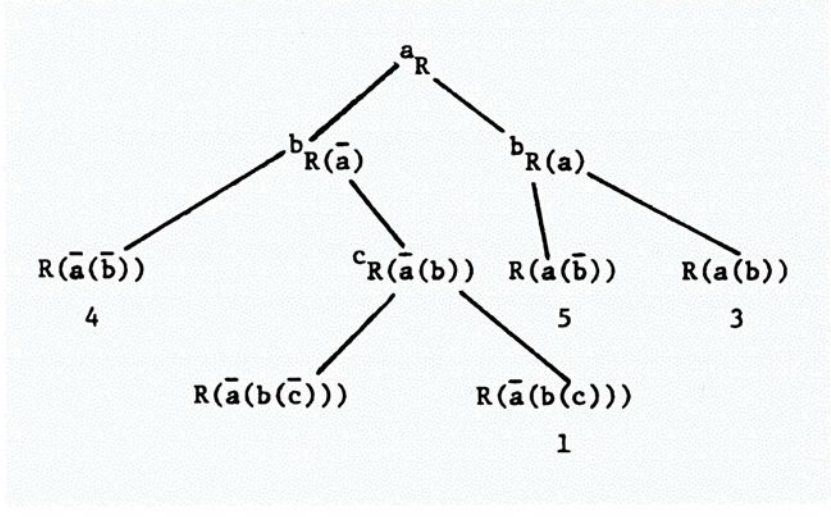


Figura 25 – A "BSP tree" para a "tessellation" da figura 24

3.3 Características de uma "BSP tree"

Deveria estar claro até agora que o algoritmo executa um particionamento recursivo do plano em segmentos. Entretanto, observemos que dado um conjunto de segmentos, mais que uma "tessellation" pode ser gerada pelo algoritmo, dependendo em que seqüência os segmentos são selecionados. Observemos nas figuras 21,22 e 23, que tivesse a ordem de seleção sido ***c, b, a,*** a figura 26 produzida, teria quatro áreas, em oposição às 5 áreas da figura 24. Desde que uma "tessellation" é formada por segmentos estendidos, e o comprimento do segmento estendido é dependente do tamanho da região contida no momento em que ele é estendido, selecionando segmentos em diferentes

[2] "tessellation": arranjo de polígonos numa dada área sem espaçamentos entre si e sem sobreposições.

ordens, regiões diferentes são produzidas, e assim há dependência da "tessellation" sobre a ordem da seleção.

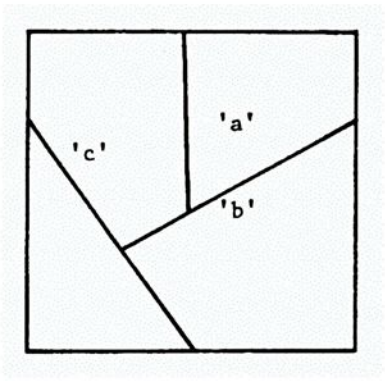


Figura 26 - "Tessellation" criada a partir da escolha dos segmentos c, b, a

É também possível ter, para um dado conjunto de segmentos, mais que uma árvore que descreve a mesma "tessellation". Assumimos o mesmo estágio de construção de uma árvore, estamos examinando a região R_k e o conjunto de segmentos associados:

$$\Sigma_{R_k} = (s_1, s_2, \ldots, s_m)$$

Se $\bigcup_{i=1}^{m} s_i = \bigcup_{i=1}^{m} \hat{s}_i$, com respeito a R_k, então cada permutação π sobre $i = 1,2,\ldots,m$, resultará em uma "subtree" diferente, na qual a "subtree" é gerada pela seleção de segmentos na ordem $s_{\pi(1)}, s_{\pi(2)}, \ldots, s_{\pi(m)}$. Todavia, cada "subtree" descreverá a mesma "tessellation" de R_k. Consequentemente, há árvores distintas que descrevem a mesma "tessellation" de uma região R. Por exemplo, na figura 27, qualquer árvore especifica a mesma "tessellation".

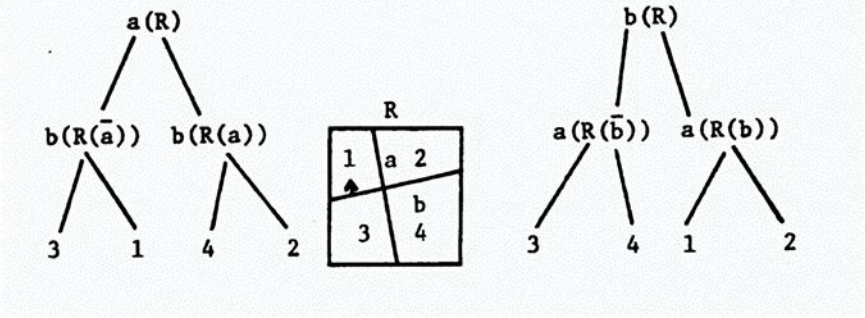


Figura 27 – Duas árvores que especificam a mesma "tessellation"

Um caso importante ocorre quando o conjunto inicial de segmentos é equivalente ao conjunto estendido, isto é, $\cup\{s \mid s \in \Sigma\} = \cup\{\hat{s} \mid \hat{s} \in \hat{\Sigma}\}$. Se em extensão a região inicial é um plano, todos os elementos de Σ seriam linhas. Desde que a extensão não tenha efeito, a "tessellation" é fixada antes que o algoritmo é iniciado. Nós chamamos tal "tessellation" como "tessellation" máxima, porque qualquer conjunto de segmentos permanecendo sobre o mesmo conjunto de linhas pode produzir somente "tessellation", cujas áreas são a união de áreas de "tessellation" máximas, como pode ser visto pela comparação das figuras 24 e 26 com a figura 28. Segue que qualquer conjunto de segmentos tem um correspondente de "tesselation" máxima cuja cardinalidade é o número máximo de áreas produzidas por qualquer "tessellation" resultante do conjunto. Em geral, o número de "tesselattions" diferentes que podem ser derivadas do conjunto Σ, é de certa forma, o complemento do número de árvores distintas que descrevem as mesmas "tessellations".

Uma "BSP tree" construída pelo algoritmo I contêm nós relacionados a segmentos e nós relacionados com áreas. Os nós segmentos são exatamente os nós interiores da árvore e os nós "áreas" são as folhas. O algoritmo pode ser pensado como um primeiro gerador de uma árvore binária composta somente de nós segmentos. Haverá então, nós segmentos que têm um ou dois filhos vazios. (Cada nó de uma árvore binária tem potencialmente dois filhos, esquerdo e direito. Se um nó não tem um ou ambos os filhos, nós os referenciamos como filhos vazios). A cada filho vazio, um nó área é adicionado. A árvore resultante, é tal que, todo nó segmento tem ambos, um filho esquerdo e um filho direito, cada um dos quais poderia ser outro nó segmento ou um nó área. Desde que, árvores binárias de n nós têm $n + 1$ filhos vazios, segue que o número de nós área é um número maior que o número de nós segmentos, assim uma árvore de $2n - 1$ nós é necessária para representar uma "tessellation" contendo n áreas.

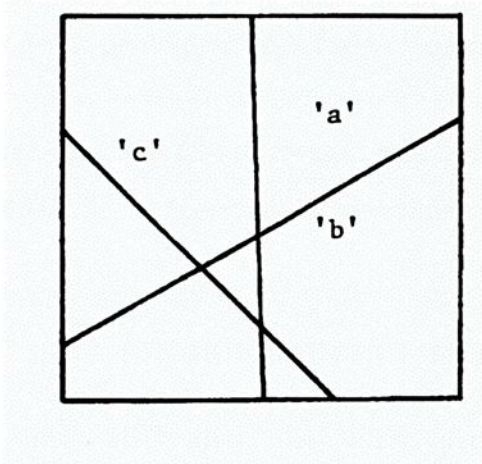


Figura 28 – A "tessellation" máxima para a figura 24

Cada "subtree" de uma "BSP tree" representa a mesma região R_i de forma que a união de todas as áreas representadas pelas folhas de R_i são iguais a R_i (os segmentos representados pelos nós segmentos de R_i são assim, também incluídos nesta união). Para propósito de notação, designaremos a região representada pela árvore inteira como R_o. É óbvio, que é a região original de onde a "tessellation" é formada. A extensão do segmento s representado pela raiz q de uma "subtree" particionada da região R_i, e a região representada pelas duas "subtrees" de q são dois "half-spaces" formados de R_i por $\hat{s}$. Se, em atravessando uma árvore alcançamos q, então tomando as ramificações direita e esquerda de q, teríamos a geometria correspondente a seleção de um destes dois "half-spaces". Um atalho na árvore, então, reflete uma sucessiva seleção de porções menores de R_o. De fato, a região representada pela "subtree" é a intersecção dos "half-spaces" com o respectivo R_o, formado pela extensão dos segmentos que estão sobre o atalho para a raiz da "subtree" q (mas não incluindo q). Segue de imediato que a área que é "adicionada" a cada filho vazio é exatamente a intersecção de "half-spaces" com respeito a R_o, formado pela extensão de segmentos cujos nós estão sobre o caminho para o filho.

É fácil ver como uma "BSP tree" pode ser usada para localizar em que área da "tessellation" um ponto repousa. Começando pela raiz, determinamos sobre qual lado do

segmento o ponto repousa e indo em direção ao filho representando o "half-space" correspondente ao lado escolhido (pontos sobre a linha são nomeados arbitrariamente para um dos dois "half-spaces"). A repetição do processo irá gerar um caminho para um nó folha, que representa a área em que o ponto repousa, assim definindo o que podemos chamar de "problema de localização" com respeito a "tessellation".

3.4 "BSP tree" usada para sequenciamento prioritário

A importância de uma "BSP tree" ser usada para a geração de um sequenciamento prioritário está baseada no princípio que, dado em qual "half-space" repousa o ponto para o qual o sequenciamento é relativo (usualmente pensado como o "olho", ou posição observatória), todos os pontos deste mesmo "half-space" terão prioridade sobre todos os pontos de outro "half-space". Embora este fato, seja razoavelmente evidente para "half-spaces", é também verdadeiro para quaisquer duas regiões convexas.

Para obter um sequenciamento prioritário de uma árvove, um cruzamento irregular é executado. A escolha de tomar a ramificação direita ou esquerda de um nó q representando o segmento s é sempre feito em favor da "subtree" que representa a região em que está contida o mesmo "half-space" que a posição observatória está, este "half-space" tendo sido formado por $\hat{s}$ com relação a R_o. É fácil ver que tal política resultará no primeiro nó área a ser alcançado, sendo um, o qual a posição observatória repousa, isto é, a solução para o problema de localização é antecipada.

Assim, para cada nó q, todos os nós escolhidos de uma "subtree" recebem a mais alta prioridade do que q, e analogamente, todos os nós de uma "subtree" continuam a obter uma prioridade menor que q. O cruzamento total das árvores produzirá então, um sequenciamento total de nós, e isto é precisamente a prioridade visível dos elementos representados pelos nós. Note que ele é requisitado quando R_o seja convexo para garantir esta propriedade. Desde que a partição de um objeto convexo produz dois objetos convexos, a convexidade de R_o implica na mesma propriedade para toda subseqüência de refinamentos de R_o durante a construção da árvore. Assim todas as

áreas são convexas, o que é suficiente para garantir a existência de um sequenciamento prioritário de áreas.

3.5 Comparação do uso de "BSP tree"

A primeira aparição implícita de uma "BSP tree" na literatura ocorreu num artigo de Sutherland et al (1974) [8], revisando o trabalho de Schumacker et al (1969) [9], nos quais a árvore não foi mencionada nem tampouco suas propriedades gerais desenvolvidas. A aplicação apresentada era a de que a invisível "divisão de planos" fosse introduzida para uma base de dados. O método envolveu o desenho de uma simulação de cena manualmente particionada "em agrupamentos", tais como prédios, árvores, montanhas, etc..., de forma que dividindo-os com planos verticais poderiam ser ocupados nestes intervalos por agrupamentos para as mais variadas extensões. Este resultado, em termos de uma "BSP tree", seria a geração de uma "tessellation" de uma superfície por uma divisão representada por nós segmentos, e cada agrupamento era contido completamente dentro de uma área. Assim, cada agrupamento correspondia a um nó área. Um sequenciamento prioritário poderia então ser obtido sobre os agrupamentos.

Em se considerando a "tessellation" como uma "tesselattion" máxima, é possível computar off-line o sequenciamento prioritário para cada caso da posição observatória estando em áreas diferentes. Segue, do fato que desde que, as áreas são formadas por uma "tessellation" máxima, não é possível para dois pontos diferentes numa mesma área estarem em lados diferentes da extensão de um segmento com respeito a R_o (desde que numa "tessellation" máxima todos os segmentos são iguais as suas extensões com relação a R_o) Assim para cada área o cruzamento de árvore é fixado. Sutherland, apresentou sugestão usando a vantagem da pré-computação e armazenando para cada área seu sequenciamento prioritário inerente sobre os agrupamentos (desde que a divisão dos planos não sejam parte da cena, eles necessariamente não seriam incluídos no sequenciamento). Foi então o suficiente para solucionar o problema de localização na ordem para obter o sequenciamento prioritário. Desde que este método requer um espaço n^2 de armazenamento (onde n é o número de agrupamentos) e o cruzamento de árvores é

O(n), não é claro se esta aplicação seja vantajosa. Também desde que uma "tessellation" máxima é requerida, a árvore será a maior possível para um dado conjunto de agrupamentos.

A aplicação de uma "BSP tree", introduzida nesta dissertação é algo complementar ao apresentado por Sutherland. Nesta, esses objetos representados por um nó segmento (ou polígono) constituem os dados visíveis enquanto as áreas de "tessellattion" não têm importância. De fato, a função "make_tree" apresentada no programa 5 produz somente o nó segmento. O nó área somente está implicado pelos filhos vazios. Também, "make_tree" forma uma "BSP tree" para três dimensões, enquanto o método anterior, embora trabalhando-se em 3-D, forma uma "BSP tree" para duas dimensões. Obviamente a "BSP tree" pode ser usada em divisões de planos para dividir volumes tridimensionais, e um híbrido de polígonos e divisões de planos poderiam também ser desenvolvidos. Por exemplo, cada nó área de uma árvore construída com divisões de planos poderia ser substituída por uma "BSP tree" construída de polígonos para agrupamentos contidos na área.

```
PROC Make_tree (pl:polygon_list): tree;

BEGIN
K=Select_polygon (pl);
pos_list:=null;
neg_list:=null;

/* pos refere-se a parte positiva
   neg refere-se a parte negativa */

FOR i:=1 to Size_of(pl) DO
        BEGIN
        IF i <> k THEN
                BEGIN
                Split_polygon(pl[i], pl[k],pos_parts,neg_parts);
                Add (pos_parts, pos_list);
                Add (neg-parts, neg_list)
                END
        END;
RETURN Combine_tree(Make_tree(pos_list),pl(k),Make_tree(neg_list))
END;
```

Programa 5 - Função Make-tree

3.6 Combinações de "BSP trees"

Examinaremos agora o tamanho de uma "BSP tree". Derivaremos algumas fórmulas, ambas para 2-D e 3-D "BSP trees". Embora estamos mais interessados no caso 3-D, 2-D é importante em especial para o caso 3-D, no qual todo objeto que está acomodado sobre a base pode ser separado por planos verticais. Isto é equivalente a 2-D "BSP tree" correspondente a um cenário em 2-D obtido, projetando-se os objetos e planos, e os separando sobre uma base plana.

Como previamente observado, uma "BSP tree" pode ser criada por ambos os objetos finitos e infinitos. Os objetos infinitos são planos para o caso 3-D e são linhas para o caso 2-D. Os objetos finitos são segmentos e polígonos convexos não interseccionáveis.

Uma *d-dimensional* "BSP tree" particiona o espaço *d-dimensional* em objetos ***(d - 1) dimensões***. Assim, examinaremos primeiro, qual é o número máximo $f_d(n)$ de volumes de um espaço *d-dimensional* que pode ser criado por planos de ***n (d - 1) - dimensões.***

No caso 2-D, que tínhamos considerado, isto corresponde à "tessellation" máxima do plano usando linhas.

A fórmula geral é:

$$f_d(n) = \sum_{i=0}^{d} \binom{n}{i}$$

Como a correspondência é um-a-um entre os volumes e as folhas de uma "BSP tree", o número de nós de uma "BSP tree" é $2f_d(n) - 1$

Quantas regiões de ***(d - 1) - dimensões*** foram criadas a partir de planos de mesma dimensão sob a suposição de que 3 planos não se interseccionam ao longo de uma linha de ***(d - 2) - dimensões***?

Pode ser mostrado que o número é:

$$\sum_{i=0}^{d} i\binom{n}{i}$$

No caso extremo, onde o objeto dado é um polígono convexo de ***n (d - 1) - dimensões***, examinemos o caso pior, isto é, computemos o número máximo de regiões ***(d - 1) - dimensões*** que são obtidas a partir dos polígonos pela intersecção de planos ***n (d - 1) - dimensionais*** sobre os quais eles repousam. Pode ser mostrado que o número é:

$$\binom{n}{i} + n^{d-1}$$

Resumimos então os resultados na Tabela 1 para dois casos interessantes $d = 2$ e $d = 3$

	Volume	Objetos Ilimitados	Objetos Limitados
2-D	$\frac{n^2+n}{2}+1$	n^2	$\frac{n^2+n}{2}$
3-D	$\frac{n^3+5n}{6}+1$	$\frac{n^3-n^2+2n}{2}$	$\frac{n^3+3n^2+2n}{6}$

Tabela 1 - Quantidade máxima possível de nós em uma "BSP tree"

3.7 "BSP trees" e Conjuntos Regulares

Com a classificação das folhas de uma "BSP tree" como células interiores ou exteriores, "BSP trees" são capazes de representar conjuntos regulares. Por construção, cada célula é aberta e não vazia e consequentemente tem interior. Um conjunto regular ***S*** representado pela árvore ***B*** é então definido como fechamento da união de todas as células interiores (in-cells):

$$S = cl(\cup_i C_i), \forall C_i \in \{in - cells\}$$

A fronteira de S consiste de todos os pontos que repousam entre o interior e o exterior de S e todos os pontos que repousam sobre sub-hiperplanos dos nós internos de B. Como sub-hiperplanos separam células internas (in-cells) de células externas (out-cells), a fronteira de S pode ser descrita como a união das intersecções de todas as células internas e externas (in-and-out-cells):

$$bd(S) = \bigcup_{i,j}(cl(C_i) \cap cl(C_j)), \forall C_i \in \{in - cells\}, \forall C_j \in \{out - cells\}$$

Classificação de Pontos

Um dos problemas mais comuns que temos que resolver em geometria computacional é o problema de *ponto de classificação*: dado um sólido S e um ponto p, determine se p repousa no interior de S, fora de S ou na fronteira de S. "BSP trees" provêem uma eficiente forma de resolver este problema. Assumindo que B_S seja uma representação "BSP tree" de um sólido S, nós resolvemos o problema de *ponto classificação* observando a vizinhança de p: Se ela é homogênea, então p repousa no interior ou exterior de S. Se a vizinhança de p é não homogênea, é conhecido que p repousa sobre a borda de S. A figura 29 mostra um pseudo-código que determina a posição de p com relação a S. Naylor [10] relatou que o problema de *ponto de classificação* foi solucionado para o caso tridimensional em $O(n)$ para uma representação de fronteira com n faces. Se a "BSP tree" está balanceada (o que pode ou não pode ser possível) o problema pode ser resolvido em $O(\log n)$.

O mesmo algoritmo pode ser usado para produzir a visibilidade do sequenciamento prioritário sobre as faces relativas ao ponto de visão como segue em *Schumacher et al [9], Fuchs et al [7], Naylor [10] e Naylor et al [11].*

```
function Classify_Point ( p: point, B: BSP_Tree )
  ⟶ {in, out or on}
  if B is a leaf then return B's value;
  dp = Dot_Product (p, B's hyperplane);
  if dp < 0 then
    return Classify_Point (p, B's left son);
  else if dp > 0 then
    return Classify_Point (p, B's right son);
  else
    l = Classify_Point (p, B's left son);
    r = Classify_Point (p, B's right son);
    if l equals r return l;
    else return on;
  fi
end function
```

Figura 29 - Implementação do algoritmo do Pseudo-Código para solução do problema de classificação de pontos

Cada nó interno de uma árvore testa o ponto de visão contra o plano particionado associado com o nó. Primeiro, é efetuada a recursão sobre o subespaço distante e então a recursão sobre o subespaço mais próximo. Isto nos dá uma seqüência que pode ser usada para guiarmos o algoritmo de ponteiros.

Complexidade do espaço de uma "BSP tree"

Peterson e Yao [12], desenvolveram fórmulas mostrando a complexidade do espaço de uma "BSP tree" bidimensional, construída a partir de ***n*** polígonos iniciais, ou seja ***O(n log n)***, e a complexidade do espaço "BSP tree" tridimensional $O(n^2)$ para ***n*** faces planas.

3.8 Usando "BSP trees" para Conjuntos de Operações sobre Poliedros

Thibault e Naylor [13] apresentaram um algoritmo para calcular expressões booleanas na forma ***BSP tree <op> B-rep -> BSP tree***. Eles o chamaram de *conjunto de operação incrementado,* devido a forma resultante em que a BSP é construída.

Uma aplicação diferente é apresentada em [11]: o algoritmo de fusão de árvores como que apreende duas "BSP trees" e as fundem de acordo com o operador Booleano específico. Naylor, Amanatides e Thibault [11] descrevem o uso desta técnica em um sistema modelador interativo.

Esta e as próximas duas seções descrevem a teoria e implementação do algoritmo ***conjunto de operação incrementado***, *desenolvido por Thibault e Naylor [13] e de uma extensão obtida por Langstrof e Weiland [14]*. Dado o fato, de que sólidos primitivos são usados para construir objetos complexos, que por sua vez são gerados como listas de faces (B-reps), esta técnica parece-nos superior, uma vez que nós não precisamos convertê-los para uma "BSP tree" primeiro. Para aumentar a eficiência do algoritmo, nós temos que modificá-lo para considerar informações de caixas fechadas de dois operadores.

Conjunto de operações Incrementado

Para calcular operadores booleanos, o algoritmo particiona o espaço em regiões, de forma que ao menos um operante é homogêneo em cada região. Se tal particionamento é encontrado, a operação booleana para cada região pode ser calculada aplicando-se as regras de simplificação mostradas na tabela 2, na qual S é um conjunto arbitrariamente regular, e $\cap^*$, $\cup^*$ e $-^*$ são os operadores booleanos regularizados e $\sim^*$ é o operador complemento. O complemento de uma determinada "BSP tree" pode ser formado revertendo-se a orientação de todas as faces limites e hiperplanos e valores de células de complementação.

Consideremos o seguinte exemplo: dada a expressão $S_1 \cup^* S_2$, um tem que encontrar uma partição do espaço na região R_i tal que $R_i \subseteq \mathbf{int}(S_j)$ ou $R_i \subseteq ext(S_j)$ para $j \in \{1,2\}$. Se temos determinado que, para alguma região R_i na qual $R_i \subseteq S_2$, podemos simplificar a expressão de R_i para S_2 .

operator	left operand	right operand	result
$\cap^*$	S	*interior*	S
	S	*exterior*	*exterior*
	interior	S	S
	exterior	S	*exterior*
$\cup^*$	S	*interior*	*interior*
	S	*exterior*	S
	interior	S	*interior*
	exterior	S	S
$-^*$	S	*interior*	*exterior*
	S	*exterior*	S
	interior	S	$\sim^* S$
	exterior	S	*exterior*

Tabela 2 - Regras de Simplificação

O algoritmo para calcular operações booleanas em "BSP tree" é similar ao usado para construir "BSP trees": dada uma representação "BSP tree" B de operandos esquerdos e uma representação B-rep F de operandos direitos, o algoritmo começa pela inserção coletiva de faces de F em B. Quando em algum nó v de B, nenhuma parte de B é encontrada repousando sobre um lado do hiperplano H_v (exemplo, *v.left*), então este lado tem que ser homogêneo em relação a B. Neste caso, a "subtree" raiz do lado oposto (*v.right*) de H_v é ou uma esquerda intacta ou é substituída pela célula, dependendo da regra particular de simplificação utilizada. Para determinar se a região repousa sobre *int(B)* ou *ext(B)*, um raio poligonal testa contra as faces se o lado não homogêneo pode ser usado. Uma folha ser alcançada por algum subconjunto F_i de F significa que F_i foi classificado em relação a B. Desde que as folhas representem regiões homogêneas, as operações booleanas podem então ser calculadas. Dependendo em particular, da regra de simplificação utilizada, ou as folhas são esquerdas intactas ou substituídas por uma "subtree", construída das faces de F_i.

```
if op = −* then
   B := Negate_B-rep( B )
   op := ∩*

procedure Incremental_Set_op
                    ( op : set_operation ; v : BSPTreeNode ;
                      B : set of Face ) returns BSPTreeNode
   if v is a leaf then
      case op of
         ∪* : case v.value of
            in  : return v
            out : return Build_BSPT( B )
         ∩* : case v.value of
            in  : return Build_BSPT( B )
            out : return v
   else
     <B_left, B_right, B_coincident> := partition B with H_v
     if B_left has no faces then
        status := Test_in/out(H_v, B_coincident, B_right)
        case op of
           ∪* : case status of
              in  : discard_BSPT( v.left )
                    v.left := new "in" leaf
              out : do nothing
           ∩* : case status of
              in  : do nothing
              out : discard_BSPT( v.left )
                    v.left := new "out" leaf
     else
        v.left := Incremental_Set_op( op, v.left, B_left )

     if B_right has no faces then
        (* similar to above *)
     else
        v.right := Incremental_Set_op(op, v.right, B_right)
     return v

end Incremental_Set_op ;
```

Figura 30 - Pseudo code para o algoritmo de cálculo do Conjunto de operações incrementado

O processo de inserção resulta em F sendo distribuído entre os sub-hiperplanos ou células de B. A fig. 31 mostra a aplicação do algoritmo *conjunto de operações incrementado* sobre a expressão $S = O_1 -^* O_2$. A fig. 31a mostra a geometria inicial

dos dois operandos, fig. 31b mostra a "BSP tree" depois da classificação de (***uv, vw, wu***) e a fig. 31c mostra a árvore resultante depois da aplicação das regras de simplificação.

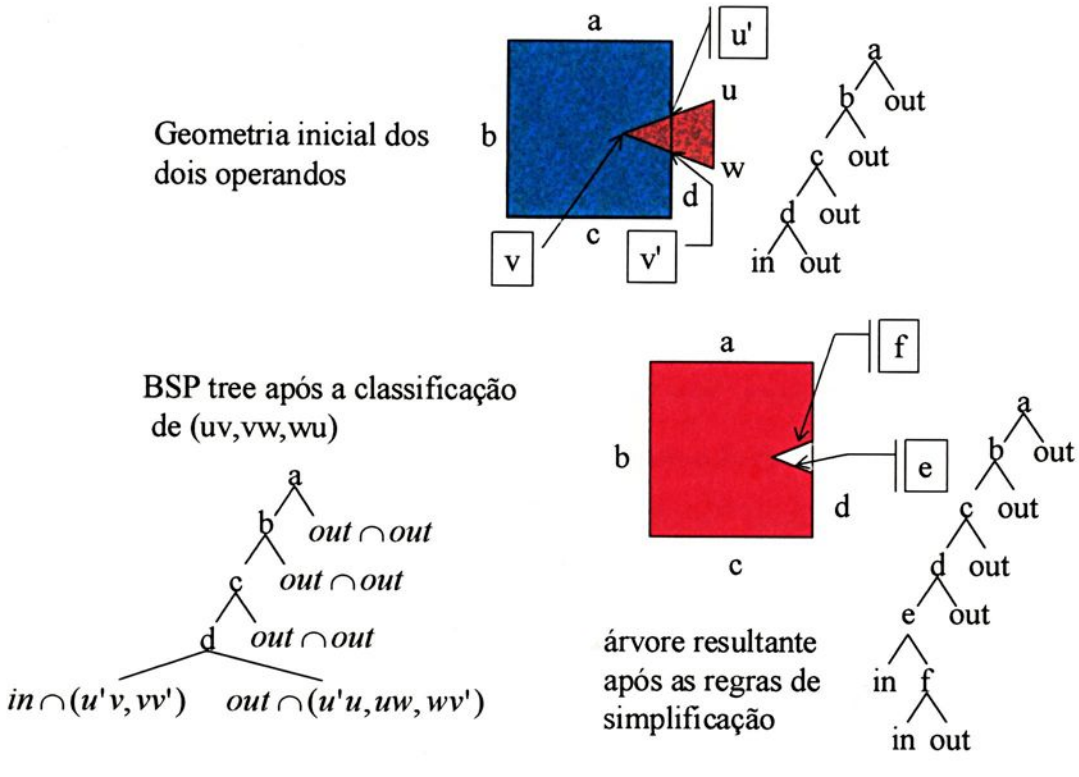


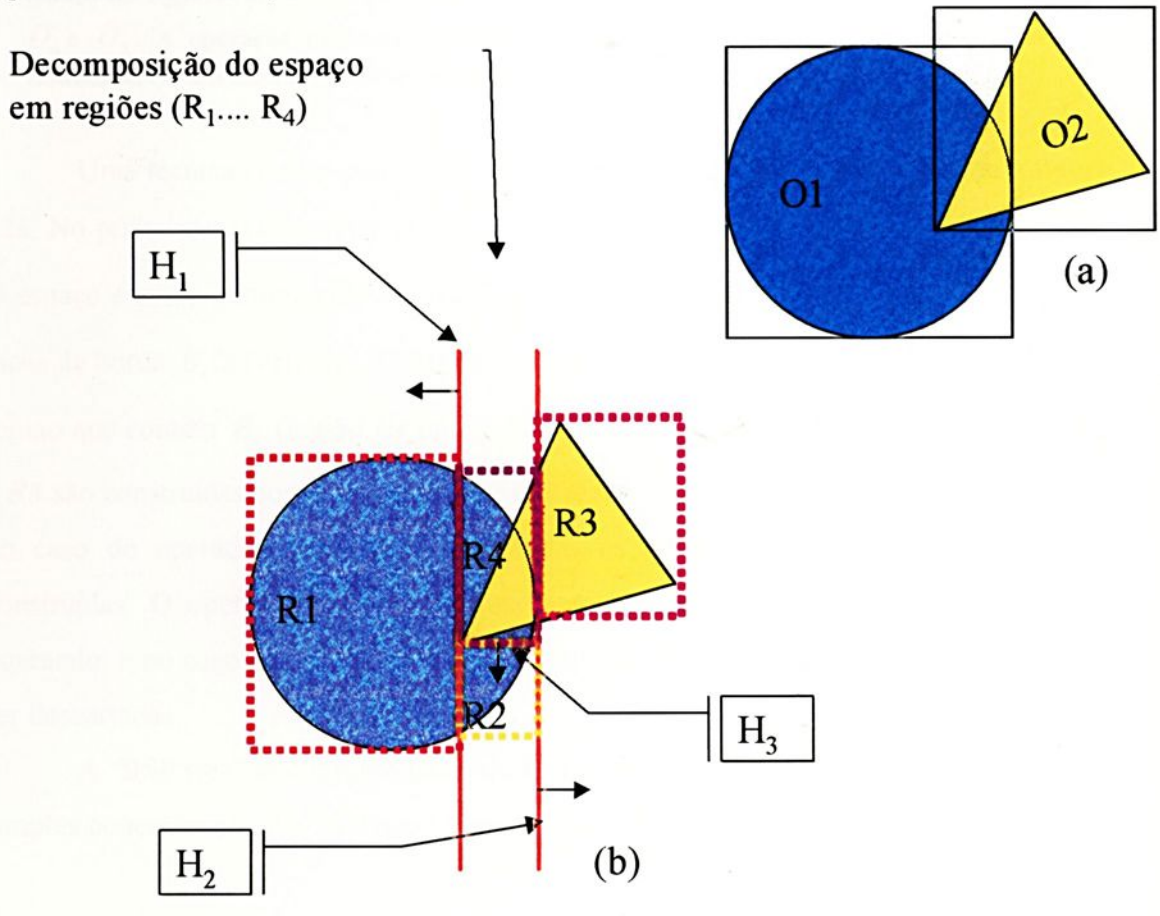
Figura 31 - Calculando a diferença de dois conjuntos regulares

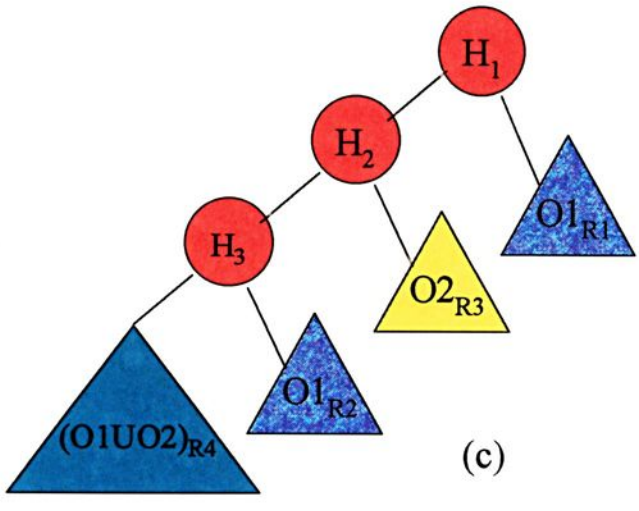
O algoritmo acima descrito produz uma representação "BSP tree" genérica $\hat{S}$ de um conjunto regular ***S*** que resulta da operação booleana sobre dois conjuntos regulares. Entretanto, os nós internos de $\hat{S}$ agora podem conter faces que não pertencem à borda de ***S*** (face ***d*** na fig. 31c). Regerar as bordas de ***S***, requer que para cada nó ***v*** de $\hat{S}$, a intersecção das bordas de ***S*** com o sub-hiperplano associado com ***v*** $(bd(S) \cup sH(v))$ necessitaria ser gerada. Isto pode ser feito inserindo as faces armazenadas a ***v*** dentro das "subtrees" ***v***. Faces, ou parte das faces que pertencem a ***bd(S)*** têm que repousar entre o interior e o exterior das células (in-and-out-cells). Elas alcançarão as células internas esquerdas ***v's*** e as células direitas ***v's*** ou vice-versa, e deveriam armazenar ***v*** como uma nova borda.

Usando Informações de Caixas Limitadas

As "BSP trees" discutidas anteriormente derivam do particionamento de hiperplanos a partir de faces de sólidos limitados. Isto implica, que cada nó interno está associado à borda da face. Se permitirmos o uso de hiperplanos que não são coplanares a qualquer face da borda, e então permitirmos pontos internos sem faces associadas, podemos aumentar a eficiência do algoritmo de cálculo booleano.

Consideremos o caso da operação união sobre dois sólidos disjuntos O_1 e O_2. Ao invés de inserir todas as faces de O_2 dentro da "BSP tree" representada por O_1, encontraremos um hiperplano h que particiona o espaço em duas regiões R_1 e R_2 de tal forma que cada região contenha O_1 ou O_2. A "BSP tree" resultante é então construída pela simples criação do nó raiz v_R, que está associado a h. A representação "BSP tree" de O_1 e O_2 está então inserida como uma "subtree" esquerda e direita de v_R, dependendo da orientação de h.





A árvore resultante

Figura 32 - Operações booleanas otimizadas. A Fig. mostra uma operação união otimizada sobrepondo objetos O_1 e O_2. Fig (a) mostra a geometria inicial. Fig (b) mostra a decomposição do espaço em regiões ($R_1 ... R_4$) que estão aninhadas com a interseção das caixas limitadas dos objetos O_1 e O_2. A operação booleana é calculada somente na região R_4. Fig. (c) mostra a árvore resultante. Os nós internos contêm os hiperplanos usados para a decomposição.

Uma técnica similar pode ser usada sobrepondo operandos como mostra a figura 32a. No primeiro passo, a intersecção B_i da caixa limitada de operandos é computada. O espaço é então particionado em regiões R_i por hiperplanos h_i que são coplanares às faces de borda B_i*'s* (veja fig. 32(b)). A operação booleana é então calculada somente na região que contêm B_i (região ***R4*** na fig. 32(b)). As regiões "BSP tree" restantes ***R1, R2*** e ***R3*** são construídas com o algoritmo BSP descrito na seção anterior. Note que somente no caso do operador booleano união, todas as árvores restantes necessárias serão construídas. O operador diferença requer somente as árvores da região do primeiro operando, e no caso do operador booleano intersecção todas as regiões restantes podem ser descartadas.

A "BSP tree" que representa o resultado final da operação é então construída pela simples concatenação de "subtrees" como mostra a fig. 32(c).

CAPÍTULO 4:

CONCLUSÃO

A descrição de objetos físicos através de modelos computacionais geométricos é um dos temas de maior relevância em computação gráfica. Ao considerarmos o tratamento numérico das equações diferenciais parciais da Física-Matemática, qualquer que seja o método utilizado, impõe-se como etapa prévia a construção dos domínios geométricos de interesse. Notadamente para o caso de problemas tridimensionais com domínios geométricos mais complexos, a modelagem de volumes e sua posterior discretização constitui tópico de interesse para os que se dedicam à simulação computacional de equações diferenciais parciais. Uma das maneiras possíveis de construção de domínios geométricos não triviais é a aplicação de operações Booleanas a sólidos mais simples, normalmente regulares e simplesmente conexos. Nesta dissertação mostra-se como "BSP trees" (Binary Space Partitioning Trees) podem ser usadas para representar poliedros. Uma "BSP tree" é uma árvore binária representando um particionamento recursivo de um espaço *d-dimensional* por *hiperplanos*.

Modelar objetos físicos requer uma adequada base estrutural de dados e de algoritmos, que proverão um esquema de representação não ambíguo e eficiente. Esta dissertação espera prover o leitor de uma descrição do esquema representativo BSP, a partir de versões de Naylor [10] e de Thibault et al [11], e de uma extensão formulada por Langstrof e Weiland, [14] que basearem-se no conceito de dimensão implícita independente, ou de como é melhor chamada, de adimensionalidade da árvore binária. A técnica de Langstrof, aqui dissertada, permite converter representações de contorno de poliedros regulares para "BSP trees" e para avaliar o conjunto de expressões Booleanas na representação de poliedros.

Espero que os algoritmos aqui produzidos sejam, em futuro próximo implementados, em continuação a presente tratativa, e apresentados em um futuro trabalho através do software Mathematica, mais elegante, e de melhor definição gráfica e concisão de dados.

REFERÊNCIAS

[1] A. Ambrose and M. Lazerowitz, Fundamentals of Symbolic Logic, Revised. New York: Holt, Rinehart and Winston, 1962.

[2] A. A. Bennett and C. A. Bayliss, An Introduction to Formal Logic. Englewood Cliffs, N.J.: Prentice-Hall, 1939.

[3] P. R. Halmos, Lectures on Boolean Algebra. Princeton: Van Nostrand, 1963.

[4] R. Sikorski, Boolean Algebras (Second Edition). Berlin: Springer-Verlag, 1964.

[5] Requicha, A.A.G.; "Mathematical Models of Rigid Solids", tech. Memo 28, Production Automation Project, University of Rochester, 1977.

[6] Marti Mantyla, "An Introduction to Solid Modeling", 1988, Computer Science Press.

[7] Fuchs, H.; Kedem, Z. M. and Naylor B. F.; "On Visible Surface Generation by a Priori tree Structures", Computer Graphics, Conf. Proc. of SIGGRAPH' 80, pp 124-133, July 1980.

[8] Sutherland, I.E.; Spranll, R.F. and Schumaker, R.A.; "A characterization of tem Hidden-Surface Algorithms" ACM Computing Surveys, 6(1); 1-55;(1974).

[9] Schumacker, R.A.; Brand, R.; Gilliand, M. and Sharp, W.; "Study for Applying Computer-Generated Images to Visual Simulation", AFHRL-TR-69-14, U.S. Air Force Human Resources Laboratory (1969).

[10] Naylor, B.; "Interactive Solid Geometry via Partitioning Trees", Graphics Interface '92, pp. 11-18, 1992.

[11] Thibault, W.; Amanatides J. and Naylor B.; "Merging BSP Trees Yields Polyhedral Set Operations", Computer Graphics Vol. 24(4), pp. 115-124, 1990.

[12] Peterson, M.; Yao, F.; "Efficiente Binary Space Partitions for Hidden-Surface Removal and Solid Modeling", Discrete and Computacional Geometry, pp. 485-503, 1990.

[13] Thibault, W. and Naylor, B.; "Set Operations on Polyhedra Using Binary Space Partitioning Trees", Computer Graphics, Vol21(4), pp. 153-162, 1987.

[14] Langstrof, A.; Weiland, T.;"Geometric Modeling with Boolean Operators", 7th Internation IGTE Symposium, pp. 385-390,1996.