ACOMPANHAMENTO DE RAIOS OTIMIZADO PARA SÓLIDOS CSG

João Luiz Dihl Comba

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

Aprovada por:

Prof. Ronaldo César Marinho Persiano, D. Sc.

(Presidente)

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

Prof. Paulo Cesar P. Carvalho, Phd

Prof. Luiz Fernando Ramos Martha, Phd

RIO DE JANEIRO, RJ – BRASIL

ABRIL DE 1991

COMBA, JOÃO LUIZ DIHL

Acompanhamento de Raios Otimizado para Sólidos CSG [Rio de Janeiro] 1991
XII + 127p. 29, 7 cm (COPPE/UFRJ, M. Sc., Engenharia de Sistemas e Computação, 1991)
Tese – Universidade Federal do Rio de Janeiro, COPPE
1. Realismo I. COPPE/UFRJ
II Título (série).

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

ACOMPANHAMENTO DE RAIOS OTIMIZADO PARA SÓLIDOS CSG

João Luiz Dihl Comba

Abril de 1991

Orientador: Prof. Ronaldo César Marinho Persiano

Programa: Engenharia de Sistemas e Computação

Este trabalho apresenta uma nova proposta para a otimização do algoritmo de Acompanhamento de Raios ("Ray Tracing"), muito conhecido na área da Computação Gráfica pela geração de imagens com alto grau de realismo. É apresentada uma proposta de otimização para sólidos CSG usando subdivisão espacial (Octree). Uma estrutura híbrida entre Octree e CSG é utilizada e o processo de visualização é tornado mais eficiente utilizando-se um novo algoritmo de Progressão Celular.

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

ACCELERATED "RAY TRACING" FOR CSG SOLIDS

João Luiz Dihl Comba

April, 1991

Thesis Supervisor: Prof. Ronaldo César Marinho Persiano

Department: Engenharia de Sistemas e Computação

This work presents a new approach to accelerate the Ray Tracing algorithm, well–know in Computer Graphics by high realism images. A new proposal is presented for CSG solids using spatial subdivision (OCTREES). An hybrid structure between CSG and Octree is used, and the visualization process is enhanced with a new algorithm called Cell–Progression).

Dedico este trabalho a minha esposa Ângela (que sempre soube me incentivar), ao meu filho Alexandre (por tudo aquilo que o possa inspirar no futuro) e a meus pais e irmãos (que mesmo distantes estiveram sempre perto de mim).

AGRADECIMENTOS:

Muitos contribuiram para que este trabalho fosse concluído. Primeiramente devo agradecer a todo carinho e compreensão da minha esposa Ângela, que foram fundamentais para que eu conseguisse chegar até aqui.

Agradeço a meus pais pela formação que me deram.

Agradeço ao pessoal do Laboratório de Computação Gráfica (LCG), nas pessoas do Ricardo, Abel e Alberto, que sempre me ajudaram muito, não somente como profissionais, mas também como amigos.

Agradeço a todos meus colegas de mestrado, por me permitirem travar discussões proveitosas que ajudaram no desenvolvimento da tese, sem falar no companheirismo sempre existente. Em particular agradeço ao Cláudio Esperança, pelos inúmeros bons palpites e incentivos a sempre fazer um trabalho de qualidade.

Agradeço também as secretárias do Programa de Engenharia de Sistemas e Computação, em particular a Denise e a Cláudia, pelo trato sempre cordial e eficiente das burocracias do dia a dia.

Agradeço a CAPES e FIPEQ pelo incentivo financeiro, indispensável à realização deste trabalho.

Agradeço a COPPE pelo privilégio de ter realizado seu curso de mestrado em Engenharia de Sistemas e Computação, bem como ter permitido acesso a seus recursos, em especial às bibliotecas e ao Laboratório de Computação Gráfica.

Finalmente, agradeço a meu orientador, Prof. Marinho, pela forma como orientou meu trabalho de mestrado, por me fazer cursar um mestrado de alto nível, que complementou ainda mais minha formação em Computação Gráfica.

SUMÁRIO

Capítulo IV – Acompanhamento de Raios Otimizado

Capítulo V – Apresentação da Nova Proposta

Capítulo VI – Geração da Estrutura Híbrida entre Octree e CSG

Capítulo VII – Algoritmos de Progressão Celular

Capítulo VIII – Avaliações e Conclusões

LISTA DE FIGURAS

Lista de Tabelas

Capítulo I

Introdução

I.1 – O USO DE IMAGENS REALISTAS

Um dos grandes interesses de estudo da Computação Gráfica refere–se à produção e exibição de imagens sintéticas no computador. Dentre os diversos tipos de imagens existentes, um caso em especial tem se mostrado de grande importância: a Imagem Realista.

O interesse pela produção de imagens realistas tem crescido à medida que diversas aplicações começaram a fazer uso destas imagens. As mais conhecidas aplicações são encontradas nos campos da Publicidade, Cinema, Televisão, Arquitetura, etc. Estudar as diversas alternativas para produção de imagens realistas torna–se, portanto, importante objeto de estudo.

I.2 – O ALGORITMO DE ACOMPANHAMENTO DE RAIOS

O algoritmo de Acompanhamento de Raios ("Ray Tracing") é um algoritmo popular para a produção de cenas realistas. Inicialmente introduzido por APPEL (1968) como um

método para remoção de superfícies ocultas, foi revisado por WHITTED (1980) para trabalhar com modelos globais de iluminação, e assim permitir a geração de imagens realistas. A idéia principal deste método consiste na simulação da interação da luz com uma determinada cena, usando os conceitos da Ótica Geométrica.

Os resultados produzidos por este algoritmo são excelentes, entretanto o alto custo computacional exigido torna–se sua maior desvantagem. O fator principal desta ineficiência do algoritmo refere–se aos inúmeros e custosos cálculos de interseção entre raios e objetos. A otimização deste processo pode ser feita seguindo duas abordagens:

- Otimização do processo de interseção (cálculo matemático)
- Redução do número de interseções (entre raios e primitivas)

Na primeira abordagem encontram–se estudos para otimizar o cálculo de interseção propriamente dito. O uso de volumes limitantes ("Bounding Box") é um exemplo clássico deste tipo de abordagem. A segunda abordagem tem por objetivo reduzir o número de testes, realizando testes de interseção sobre um número reduzido de objetos. O uso de técnicas de coerência, bem como o uso de técnicas de subdivisão são dois exemplos conhecidos deste tipo de abordagem.

I.3 – OTIMIZAÇÃO DO ALGORITMO USANDO OCTREES

Muitas destas técnicas de otimização são dependentes da natureza dos objetos que compõem a cena. Algumas delas são extremamente eficientes, mas restringem–se a um limitado tipo de objetos. Temos a preocupação de trabalhar com sólidos descritos segundo um modelo de representação de sólidos conhecido, e de razoável poder de expressão. O modelo escolhido baseia–se na Geometria de Sólidos Construtiva (doravante referenciada por CSG). Portanto, todo o processo de Acompanhamento de Raios será concebido para a exibição de cenas descritas segundo o modelo CSG.

Neste trabalho iremos descrever uma proposta de otimização baseada na redução do número de interseções. A técnica de subdivisão utilizada é a Octree, cuja idéia básica consiste na subdivisão recursiva de um espaço cúbico em oito octantes. A aplicação desta técnica sobre um sólido descrito em CSG gera uma estrutura híbrida entre Octree e CSG.

O uso de Octrees, entretanto, cria um novo tipo de problema a se resolver: enumerar as células da Octree por onde o raio sucessivamente passa. Diversas abordagens já foram descritas para o problema, desde o simples cálculo matemático até abordagens incrementais. Nesta tese são discutidas as diversas abordagens existentes, e apresentada uma nova abordagem incremental : o algoritmo de Progressão Celular.

I.4 – ESTRUTURA DO TEXTO

A estrutura do texto é a seguinte. No capítulo II será feita a apresentação formal do algoritmo de Acompanhamento de Raios. No capítulo III teremos uma discussão sobre as diversas propostas de otimização do algoritmo, focalizando as técnicas baseadas na subdivisão espacial. No capítulo IV iremos apresentar a nova proposta de otimização, que será largamente discutida nos dois capítulos seguintes. No capítulo V será descrita a geração da estrutura híbrida entre Octree e CSG. No capítulo VI teremos a descrição do algoritmo de Progressão Celular. A seguir, no capítulo VII, serão discutidos os resultados obtidos. Ao final, teremos um balanço final dos temas discutidos e encaminhamentos de trabalhos futuros.

Capítulo II

Acompanhamento de Raios

II.1 - INTRODUÇÃO

A Computação Gráfica (CG) tem assumido um papel de muita importância no atual contexto profissional. Diversas pessoas dos mais variados campos têm mostrado grande interesse em usar recursos de CG, com o objetivo de obter uma apresentação gráfica de seus respectivos trabalhos. Este intenso interesse agiu e age como um agente catalizador para o desenvolvimento da área, pois a medida que novas necessidades surgem, novas propostas devem ser desenvolvidas para solucioná-las. Como um resultado deste processo, a CG cresceu muito nos últimos anos, especialmente na última década.

Uma consequência direta da popularização da CG foi sua disseminação por campos que trabalham com informações de natureza geométrica ou visual. Áreas mais técnicas, como Física, Matemática e Química começaram a utilizar a CG como forma de complementar os fundamentos teóricos costumeiramente envolvidos, e, em certos casos, até mesmo usá-la como parte na prova de teoremas.

Outras áreas, relacionadas a atividades artísticas, também demonstraram grande interesse no uso de CG. Exibir por computador imagens que retratem com fidelidade uma

típica paisagem do cotidiano é um problema típico deste tipo de atividade. Esta necessidade externa fez com que diversas alternativas fossem estudadas e diversas propostas aparecessem.

Neste trabalho serão abordados os detalhes de uma dessas propostas: o algoritmo de Acompanhamento de Raios ("Ray–Tracing"). Os detalhes e fundamentos teóricos serão discutidos, iniciando com um histórico sobre as origens da idéia do algoritmo e suas implementações iniciais, até as avançadas propostas de otimização.

Este capítulo apresenta os fundamentos básicos do algoritmo. Inicialmente discute–se a importância do uso de imagens realistas produzidas por computador. Feito isto, são abordadas questões referentes a geração de tais imagens no computador: projeções, sistemas de cores, modelos de iluminação. O algoritmo de acompanhamento de raios é apresentado durante esta discussão, desde suas origens históricas, passando pelos trabalhos iniciais até os trabalhos mais modernos que consolidaram a idéia do algoritmo.

II.2 – O ALGORITMO DE ACOMPANHAMENTO DE RAIOS

II.2.1 – A Imagem Realista

O processo de produção de imagens por computador é um dos temas mais populares da CG, sendo a área responsável por este estudo conhecida como Síntese de Imagens. Seu escopo de estudo é muito grande, não sendo de nosso interesse discutir os diversos tipos de propostas para geração de imagens, mas estudar os detalhes e diversas formas de implementação do algoritmo de Acompanhamento de Raios.

O algoritmo de Acompanhamento de Raios gera imagens que buscam aproximar paisagens da vida real, usualmente chamadas de imagens realistas. Este tipo de imagem tem sido extremamente utilizada atualmente, principalmente em campos como simulação, propaganda, pesquisa, educação, etc.

Podemos encontrar uma típica aplicação de imagem realista em simuladores, usados para treinamento de pilotos (de aeronaves, barcos, carros, etc). Para que a simulação seja eficiente, é necessário reproduzir as condições que poderiam ocorrer em uma situação real, e neste contexto a imagem tem crucial importância, pois deve ser a mais próxima possível da realidade.

Outra aplicação refere–se a projetos de objetos tri–dimensionais (tais como automóveis, aeronaves, projetos arquitetônicos, etc). É de grande interesse do projetista visualizar

resultados preliminares de forma mais rápida e menos custosa. Para tanto, criar uma imagem realista de seu projeto torna–se uma solução eficiente e mais efetiva.

A aplicacão mais popular de imagens realistas está na propaganda. Diversos comerciais hoje em dia fazem uso de imagens realistas, podendo se constatar que houve uma grande guinada no rumo da propaganda com o advento de tais imagens. O cinema e desenhos animados também utilizam recursos gráficos especiais, e até mesmo filmes totalmente produzidos através de Computação Gráfica já foram feitos.

Neste contexto de larga utilização é que se encaixa a imagem realista. Estudar as diversas alternativas para a produção deste tipo de imagem é de vital importância para uma sociedade que mais e mais torna–se automatizada.

II.2.2 – Geração de Imagens

As imagens geradas por computador são na sua grande maioria matriciais, e cada elemento desta matriz representa um componente de imagem (usualmente chamado de pixel). Este pixel caracteriza o elemento de cor da imagem.

No caso específico deste trabalho lidamos com um vídeo de 8 bits por pixel, que permite representar uma imagem com 256 tonalidades de cor (Figura II.1). A caracterizacao de cor é dada no sistema de cores RGB (que veremos em detalhe mais adiante).

Cada pixel contém a informação de um índice de uma tabela de cores (que contém as descrições completas das cores utilizadas).

A princípio pode–se pensar que 256 é um número reduzido de cores para exibir imagens que demandem um alto grau de detalhe. Entretanto, se tivermos cuidado de escolher as 256 cores que melhor representem a imagem, podemos ter resultados satisfatórios. Detalhes sobre este processo de escolha podem ser encontrados em COMBA (1990a).

Voltando a analisar o processo de exibição, constatamos que este consiste em descobrir a cor da imagem associada a cada pixel. Para responder a esta simples pergunta muito já se fez em termos de pesquisa em CG, resultando na proposição de diversos algoritmos. Antes de estudarmos em detalhes uma destas propostas (o algoritmo de Acompanhamento de Raios), vamos estudar os fundamentos teóricos envolvidos neste processo.

II.2.3 – A Matemática da Projeção

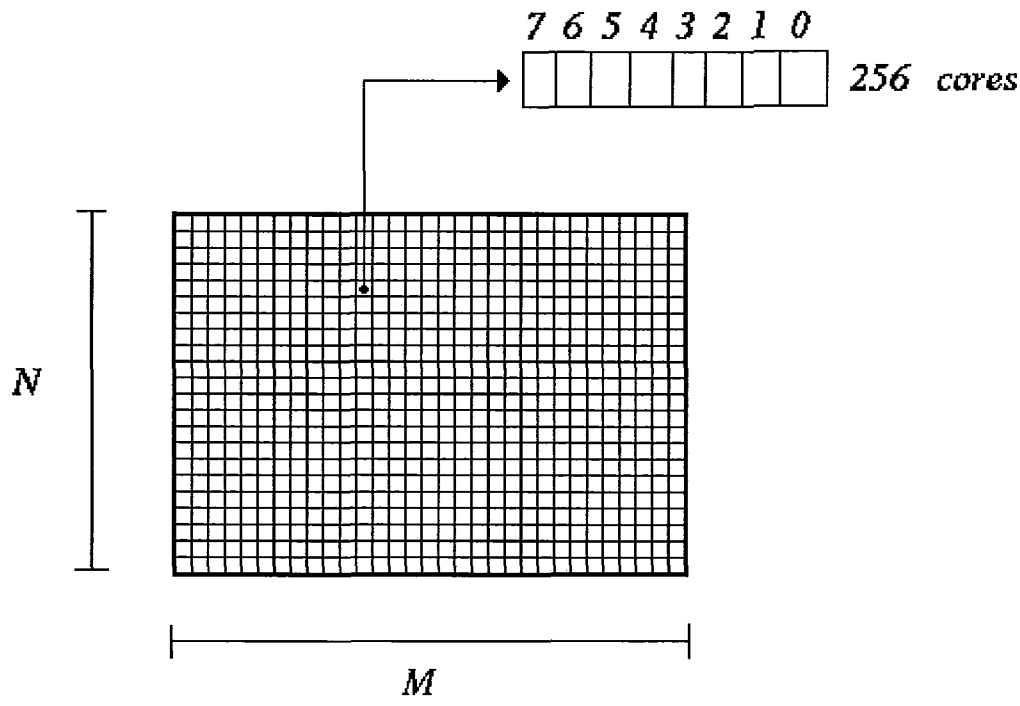


FIGURA II.1 – Vídeo com 8 planos de bits

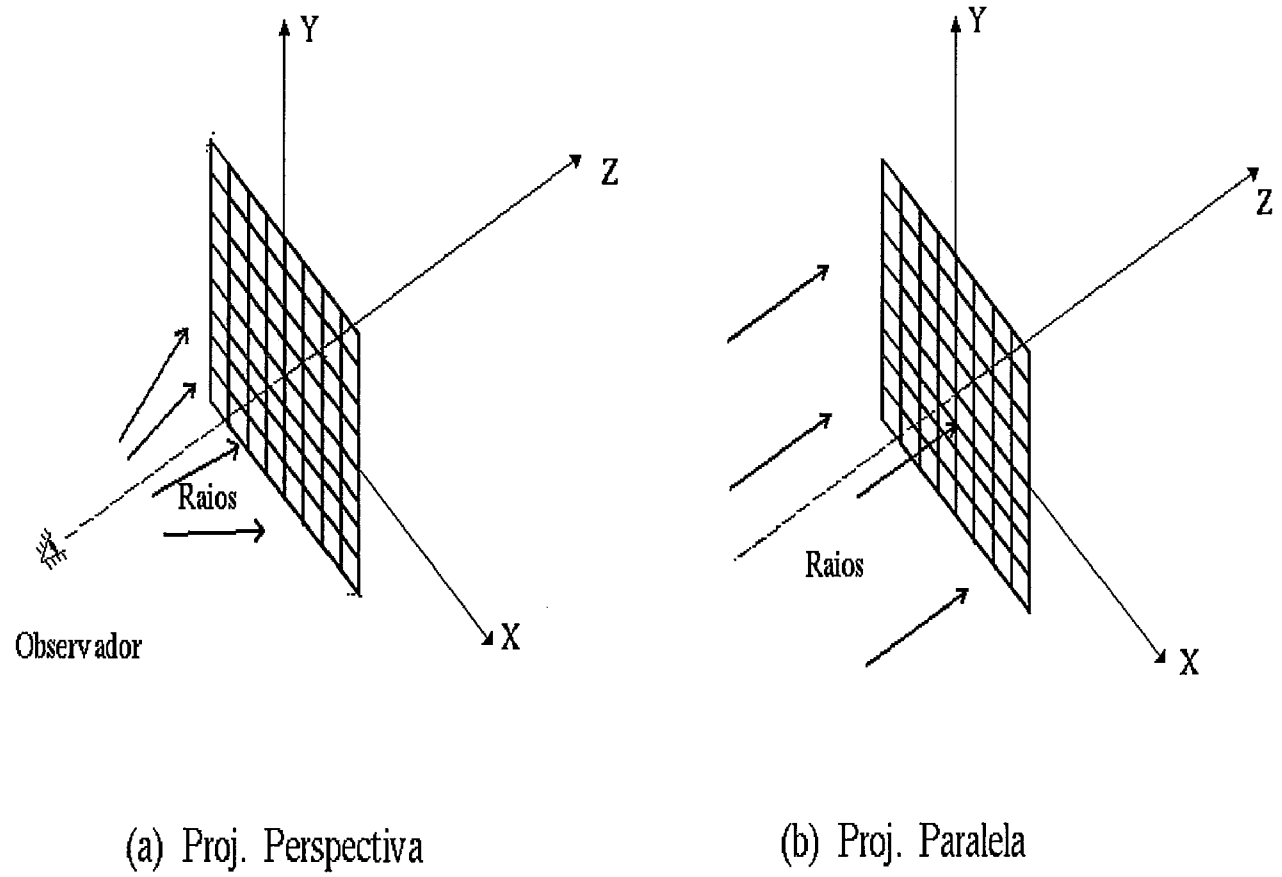
A geração da imagem realista de uma cena tri–dimensional envolve um processo de projeção. Existem diversos tipos de projeções, dentre as quais podemos citar as projeções paralelas e as projeções perspectivas. Para explicarmos os detalhes envolvidos devemos primeiramente definir dois conceitos importantes no processo de projeção: o plano de projeção e as linhas de projeções.

O plano de projeção é o lugar onde a imagem bi–dimensional será obtida após a aplicação do processo de projeção. Para gerar a imagem no plano de projeção devemos percorrer o espaço com linhas de projeção, que devem ser especificadas segundo um determinado critério.

Em uma projeção paralela as linhas de projeção são paralelas entre si. A imagem gerada de um ponto será dada pela interseção de uma linha de projeção que passa pelo respectivo ponto no plano de projeção. O observador está no infinito, observando a cena segundo retas paralelas que dirigem–se ao plano de projeção. O volume de visão (a porção do espaço a ser analisada para projeção) será um prisma retangular, podendo este ser ilimitado, ou limitado (se definirmos dois planos de corte que o limitem, um dianteiro e outro traseiro).

Por outro lado, as projeções perspectivas criam imagens mais próximas do que chamamos de imagens realistas. Os objetos distantes do observador tendem a se mostrar reduzidos, exatamente como em um processo fotográfico. Isto faz com que este tipo de projeção seja extremamente interessante para a geração de imagens realistas.

Os componentes principais da projeção perspectiva são o plano de projeção e o centro de projeção. As linhas de projeção convergem para o centro de projeção, não sendo portanto paralelas entre si. O volume de visão é um tronco de pirâmide, podendo ser definido um plano traseiro que delimite a extensão desta pirâmide. A figura II.2 mostra o exemplo de projeções paralela e perspectiva.



(a) Proj. Perspectiva (b) Proj. Paralela

FIGURA II.2 – Tipos de Projeções

Para especificar um sistema projetivo que comporte projeções paralelas e perspectivas devemos especificar os seguintes parâmetros:

- **Centro da Janela**: Coordenadas de um ponto no plano de projeção

• **Normal ao plano de projeção**: Vetor normal ao plano de projeção

• **Direção da Janela**: Vetor que, quando projetado no plano de projeção, indica a direção vertical da imagem

• **Centro de projeção**: No caso de projeção perspectiva funciona como centro de projeção propriamente dito. No caso de projeção paralela a direção de projeção será paralela ao vetor obtido pela diferença do centro da janela pelo centro de projeção.

• **Tipo de projeção**: Paralela ou Perspectiva

A figura II.3 mostra um exemplo da especificação de um sistema projetivo segundo o sistema proposto.

II.2.4 – A idéia do algoritmo de Acompanhamento de Raios

Como vimos anteriormente, os algoritmos de síntese de imagens matriciais se preocupam em descobrir a cor referente a cada pixel. A idéia do algoritmo de acompanhamento de raios é baseada em uma amostragem da cena tridimensional, levando em conta os fatores de iluminação envolvidos.

Veremos a seguir a descrição do algoritmo. Inicialmente, um histórico sobre as origens do algoritmo será apresentado. A seguir, a primeira implementação será discutida (fazendo–se necessária uma discussão sobre modelos de iluminação e sistemas de cores). Por fim, será discutida a versão recursiva do algoritmo, que caracteriza o que atualmente chamamos de acompanhamento de raios.

II.2.4.1 – Uma abordagem histórica: as idéias de DÜRER

HOFFMANN (1990) investigou as origens históricas do algoritmo. Ele forneceu provas conclusivas que as idéias fundamentais do algoritmo foram desenvolvidas por um pintor, matemático e artista chamado ALBRECHT DÜRER, durante o renascentismo (meados do século XV). DÜRER escreveu dois livros em que os principais conceitos foram introduzidos. Inicialmente DÜRER discute o conceito de sombra ("são as partes de uma cena que estão escondidas de uma fonte de luz, isto é, não recebem contribuições luminosas dela"). Para descobrir qual seriam as regiões de sombra em uma cena com um cubo e um plano, os raios provenientes de uma fonte de luz são acompanhados. A sombra gerada refere–se a silhoueta do cubo projetada no plano (figura II.4).

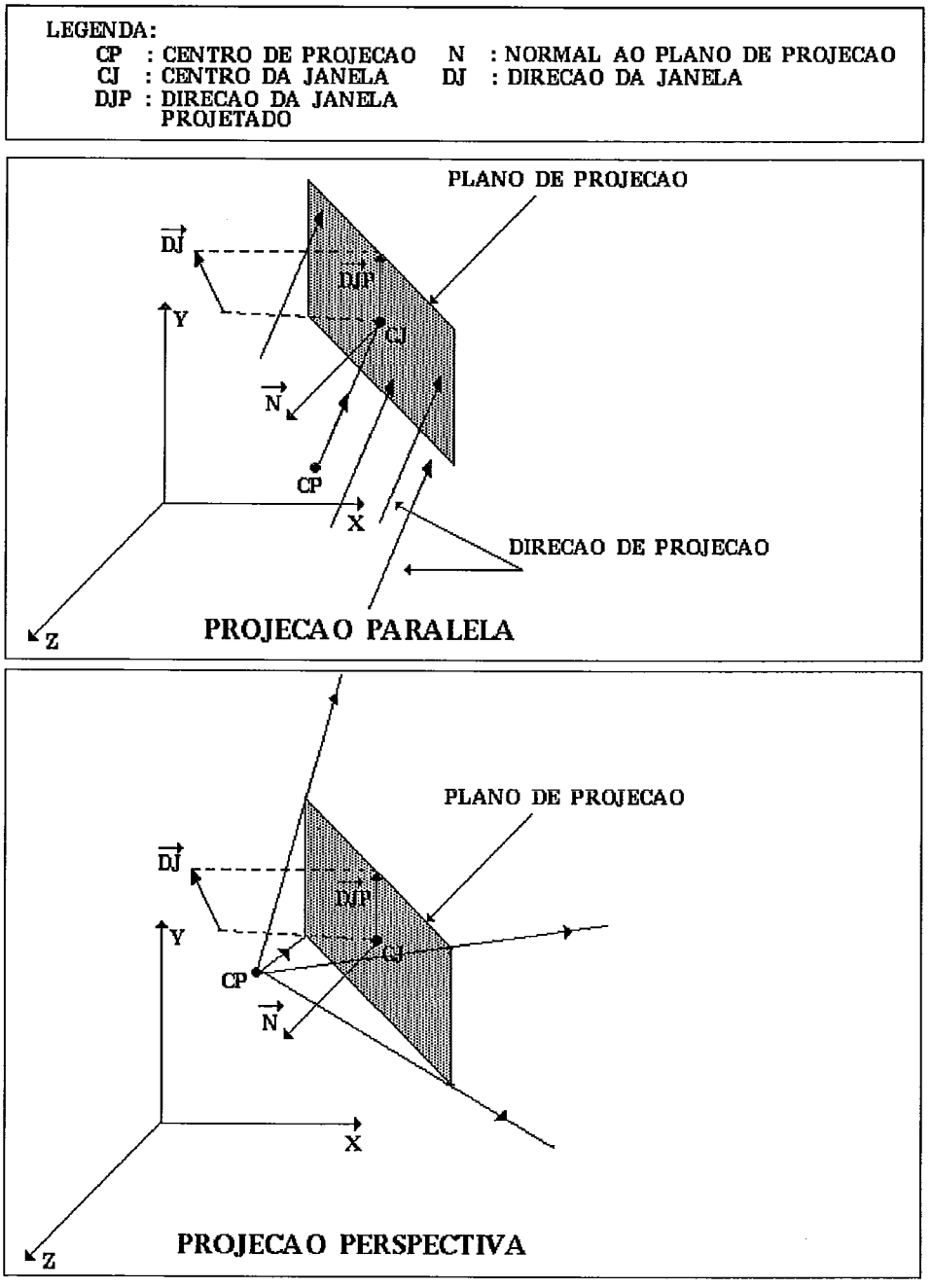


FIGURA II.3 – Usando o sistema projetivo

A seguir DÜRER apresenta os fundamentos da idéia de acompanhamento de raios. Em uma primeira instância, DÜRER mostra um artista pintando um quadro usando uma tela transparente. Olhando através de um bambu, o artista pinta tudo o que está observando através da tela. O incoveniente deste método é que a pintura só aceita uma camada de tinta (pois esta impedia a visão sobre a tela transparente).

Para lidar com esta desvantagem, DÜRER propõe outro método. Novamente olhando através de um bambu, ele não coloca a tela coincidente ao plano de projeção (como fizera

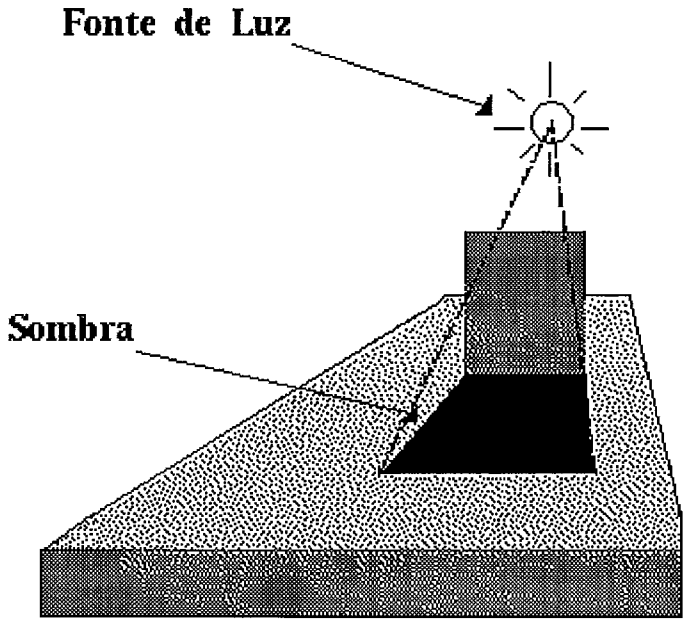


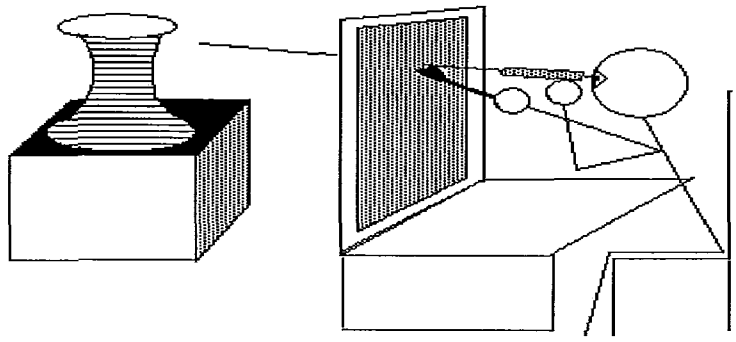
FIGURA II.4 – O conceito de Sombra segundo DÜRER

anteriormente). Desta vez ele usa o plano de projeção como uma malha de reticulados (desta forma a visão que o artista tem da cena é quadriculada). Para repassar a informação observada através dos reticulados, o artista sobrepõe a sua tela um reticulado idêntico ao colocado sobre o plano de projeção. Com isto, o artista desenha na tela o que está observando de acordo com a malha de reticulados, acompanhando raios em direção à cena (fazendo uma amostragem do que está sendo visto).Ver figura II.5

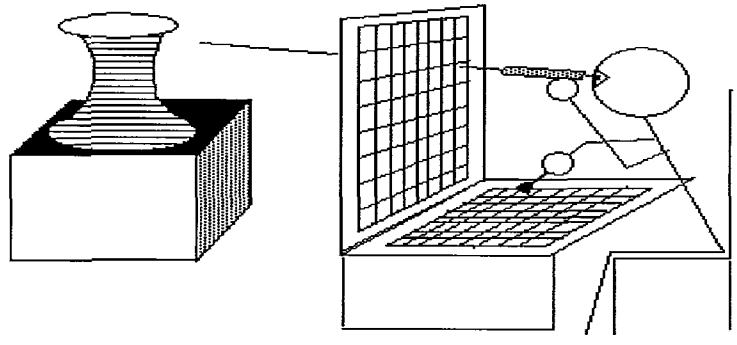
Este método proposto por DÜRER é chamado de acompanhamento de raios, pois os raios são observados em sua trajetória através do plano de projeção. As idéias de DÜRER foram primeiro utilizadas na concepção de um algoritmo de visualização no trabalho de APPEL (1968), que veremos a seguir.

II.2.4.2 – A primeira implementação: o trabalho de APPEL

Até a época da divulgação do trabalho de APPEL, estudava–se o acompanhamento de raios desde a fonte de luz até o observador (de forma a simular a trajetória descrita pelos raios de luz até atingir a retina de nosso olho). Este percurso, embora natural e correto, era computacionalmente ineficiente, visto que milhares de raios não atingem o observador, sendo acompanhados desnecessariamente (ver figura II.6).



Primeira ideia de DURER



Segunda ideia de DURER

FIGURA II.5 – As idéias de DÜRER

APPEL propôs que se fizesse exatamente o contrário (seguindo as idéias de DÜRER). A proposta era acompanhar os raios desde o observador (passando pelo plano de projeção) em direção à cena. Este tipo de percurso é conhecido na literatura como Acompanhamento de Raios Regressivo ("Backward Ray Tracing"), pois os raios luminosos são acompanhados no sentido contrário à sua propagação (figura II.7).

A abordagem de APPEL faz com que os raios sejam acompanhados até que estes colidam com algum objeto pertencente à cena. Quando esta interseção é encontrada (neste caso corresponde a um ponto no espaço), deve–se estimar qual a intensidade luminosa que incide sobre este ponto. Existem diversas formas de estimar a iluminação de pontos, baseadas na simulação do que acontece na vida real (usando conceitos de Física, especialmente os conceitos da Ótica Geométrica). Tais propostas caracterizam o que chamamos de Modelos de Iluminação, que serão estudados a seguir.

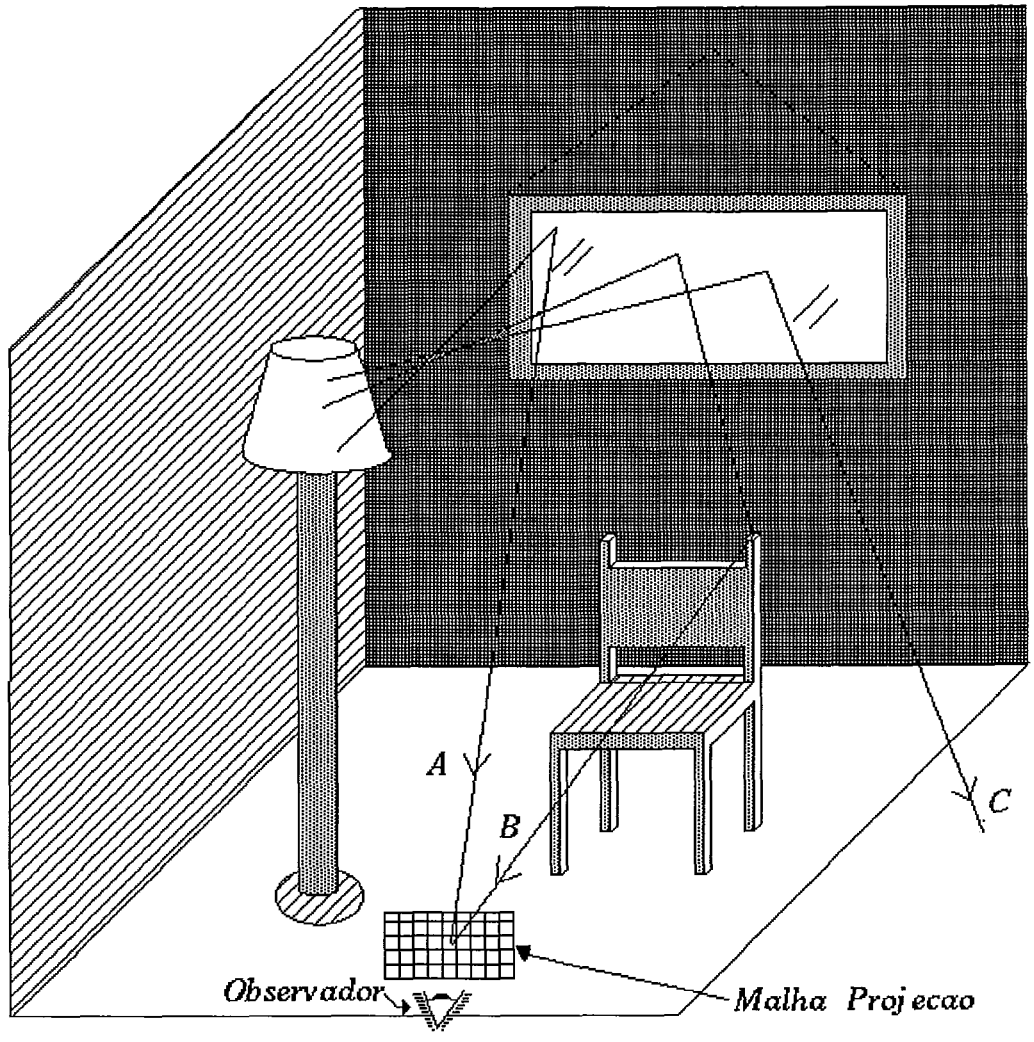


FIGURA II.6 – Acompanhando raios desde a fonte de luz até o observador

II.2.4.3 – Modelos de Iluminação

II.2.4.3.1 – Como a luz interage com a superfície

O modelo de iluminação procura avaliar a luz recebida pelo observador em cada ponto da imagem. Para realizar este cálculo, devem ser levados em conta diversos fatores inerentes à cena, tais como: orientação, posição e características da superfície, bem como posição e intensidade das diversas fontes de luz. Modelos de iluminação de diferentes níveis de complexidade já foram propostos, como os modelos de HAAL e GREENBERG, ou de COOK e TORRANCE (ver ROGERS (1985)).

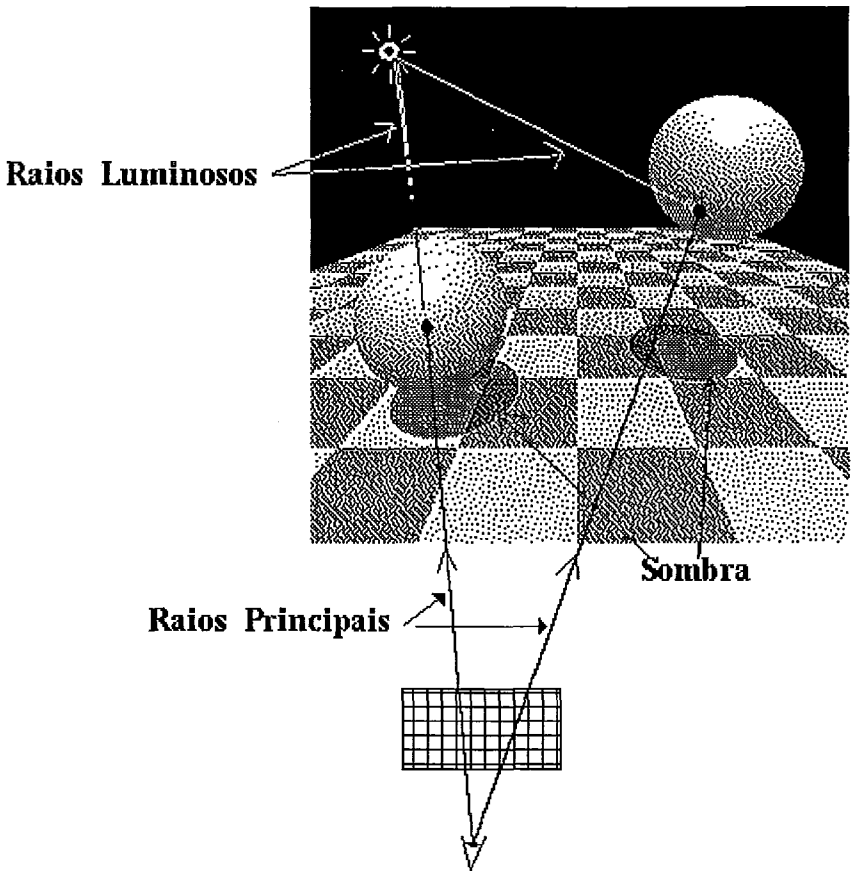


FIGURA II.7 – Acompanhando raios desde o observador até a cena

Estes modelos procuram estimar o comportamento da luz em uma dada cena. Seguindo um destes modelos (referente à Ótica Geométrica), quando a luz atinge uma superfície, três situações podem ocorrer:

- A luz é absorvida (e convertida em calor)
- A luz é refletida (e um ou mais novos raios de luz são gerados)
- A luz é refratada (e um novo raio de luz é gerado, atravessando a superfície)

Uma superfície que absorve totalmente a luz incidente tem a aparência negra, visto que nenhum retorno luminoso é gerado após o contato da luz com a superfície.

A reflexão e a refração determinam a cor da superfície, característica principal que define como a superfície é vista pelo observador. Este conceito de cor está relacionado com a

definição física de luz, e sua realização segue um sistema de cor. A caracterização de tais sistemas é dada a seguir.

II.2.4.3.2 – Sistema de Cores

A luz que incide sobre uma superfície pode ser caracterizada como uma forma de energia, constituída por diversos componentes (cada qual com um comprimento de onda específico). Para determinar as características de reflexão, transmissão e absorção da cada superfície é preciso avaliar como esta reage aos diferentes comprimentos de onda que compõem o espectro de cores da luz incidente. A cada reação deste tipo um estímulo é produzido que ativa nosso sistema sensorial, caracterizando o que conhecemos como cor (Ver ROGERS (1985, p. 383–408).

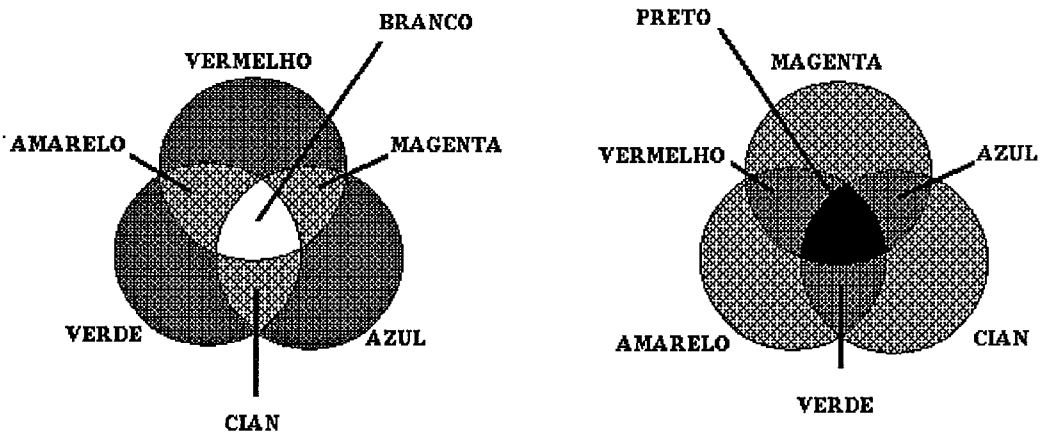
A especificação deste estímulo pelo espectro de comprimento de onda não é (no caso de implementação por computador) barato. Por isto, buscou–se formas mais eficientes e que dessem mais sensibilidade ao processo. O uso de um modelo baseado na definção de cor a partir de três cores primárias foi adotado, fazendo–se uma analogia com o modelo sensorial utilizado em nosso olho, onde a cor é representada pelas variações de intensidades em três canais, chamados de cones.

Os mais conhecidos sistemas de cores primários são:

- **RGB** (Vermelho, Verde e Azul): Trata–se de um sistema de cores aditivo, isto é, a sobreposição das três componentes é que gera a cor final.

- **CMY** (Cian, Magenta e Amarelo): Trata–se de um sistema de cores subtrativo, isto é, as cores são definidas através da subtração de suas componentes. O cian refere–se a subtração do branco pelo vermelho, o amarelo é o branco menos o azul, enquanto que o magenta é o branco menos o verde (figura II.8).

Os vídeos de uma forma geral (tanto de monitores como de televisores) trabalham com o sistema RGB. Portanto, este será o sistema adotado para a especificação de cor doravante. Existem outros sistemas de cores (como o HLS, HSV) que podem ser bastante úteis para determinados tipos de aplicacão, mas não serão discutidos no escopo do presente trabalho. Cada vez que nos referirmos futuramente a uma componente de cor estará subentendido que esta terá associado três valores, referentes ao comportamento desta componente no sistema RGB.



SISTEMA RGB **SISTEMA CMY**

FIGURA II.8 – Sistemas de Cores

II.2.4.3.3 – A Iluminação Ambiente

Além das componentes de reflexão e refração existe outra componente muito importante em um modelo de iluminação: a componente ambiente. Esta componente tenta simular a iluminação indireta que um objeto recebe de sua vizinhança, de tal forma que objetos que não recebam luz direta de nenhuma fonte de luz não sejam tratados como objetos negros. Este tipo de iluminação pode ser entendida como proveniente de uma fonte de luz não direcional (isto é, que não possui um percurso determinado de iluminação). É como se o objeto estivesse sendo iluminado em toda sua superfície de uma forma uniforme, proveniente de todos os lados (Ver ROGERS (1985, p.313)).

A contribuição imposta pela iluminação ambiente é de vital importância. Entretanto, por ser função apenas da característica de material de cada superfície, se for usada de forma isolada não será suficiente para definir com clareza a tri–dimensionalidade da superfície (pois todos os pontos da mesma terão a mesma cor). Isto devese pelo fato de nenhum fator referente a geometria da superfície estar sendo levado em conta (como a normal à superfície).

A componente ambiente é dada pela seguinte equação:

$$\text{Componente Ambiente} = K_a \cdot I_a \qquad \text{(Equação II.1)}$$

A variável K_a define a porcentagem de iluminação ambiente que a superfície recebe. A variável I_a expressa a iluminação ambiente presente na cena, que é uma característica que não

depende do material da superfície. Seu objetivo é o de simular a iluminação indireta presente na cena devida a interreflexões entre objetos.

II.2.4.3.4 – A Reflexão

O cálculo da exata interação da luz com a superfície é muito complexo. A intensidade e comprimento de onda da luz refletida depende da análise de vários fatores. A simples variação de um destes fatores faz com que a determinação exata da componente refletida seja muito complexo.

De acordo com a primeira e segunda lei da reflexão o raio refletido num refletor perfeito parte numa única direção, sobre o plano definido pela direção do raio incidente e a normal ao ponto de incidência. A direção que o raio refletido assume nesse plano é a mesma que faz com a normal um ângulo φ, onde φ é o ângulo da direção de incidência com a normal. A figura II.9 ilustra o processo.

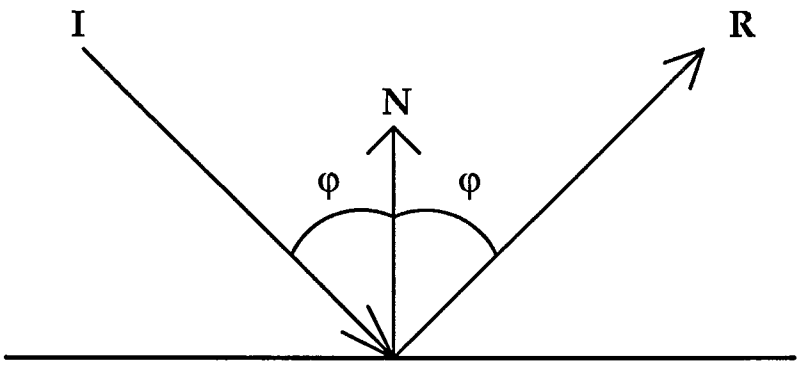


FIGURA II.9 – Reflexão em um Refletor perfeito

Os refletores perfeitos são superfícies ideais, mas incomuns de existir. Nas superfícies não refletores perfeitos há um efeito de difusão da reflexão. Esta difusão faz com que a intensidade não se concentre em uma única direção, mas espalhe-se em várias direções de forma (em geral) não uniforme.

De uma forma geral este espalhamento pode ser modelado por uma função de difusão, que exprime a intensidade a partir de um cone de espalhamento. A natureza da superfície condiciona a função adotada. Nos refletores perfeitos esta função é constante. Em geral, porém, a difusão é não uniforme e concentra-se em torno da direção do refletor perfeito.

De forma a tentar simplificar o cálculo, a componente refletida é separada em duas subcomponentes:

• Componente Difusa: Referente ao espalhamento ocorrido em todas as direções.

• Componente Especular: Referente ao espalhamento ocorrido na direção do refletor perfeito.

O cálculo da componente difusa segue a lei de LAMBERT, segundo a qual a quantidade de luz refletida é proporcional ao cosseno do ângulo entre a direção da fonte luminosa e a normal à superfície. Desta forma, a quantidade de luz vista pelo observador é independente da sua posição (Figura II.10).

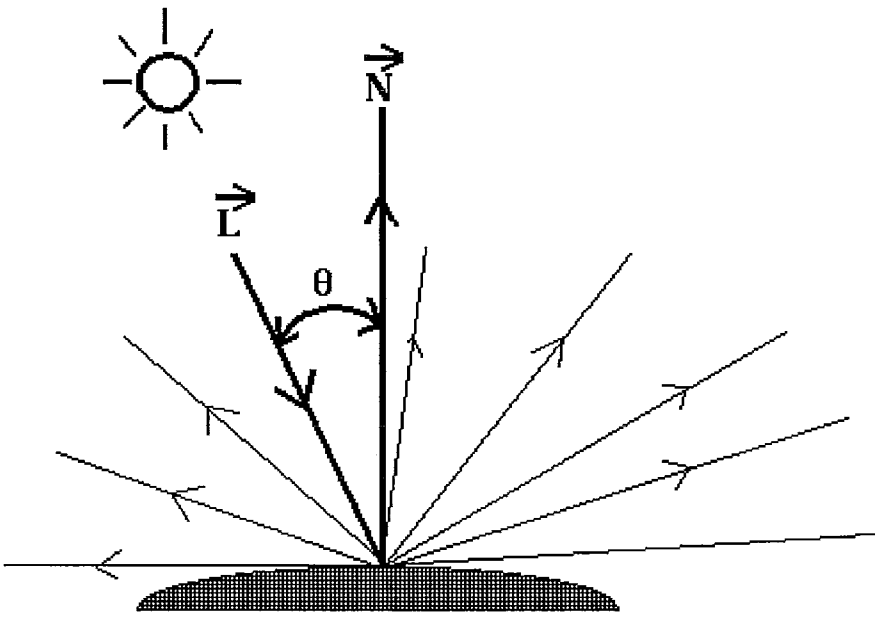


FIGURA II.10 – Reflexão Difusa

A componente difusa é dada pela seguinte fórmula:

$$\textbf{Componente Difusa} = K_d \cdot I_l \cdot \cos\theta \quad (0 \leq \theta \leq \frac{\pi}{2}) \qquad \textbf{(Equação II.2a)}$$

A variável K_d define a porcentagem de reflexão difusa da superfície. A variável I_l expressa a iluminação proveniente da fonte de luz. Se os vetores N e L forem unitários, então a fórmula será dada por:

$$\textbf{Componente Difusa} = K_d * I_l * (\vec{N} \cdot \vec{L}) \qquad \textbf{(Equação II.2b)}$$

A componente especular tenta estimar um espalhamento entre o ângulo formado entre o refletor perfeito e o vetor que une o ponto de incidência ao observador (chamado de vetor de

visão). Ver figura II.11. O vetor R é obtido pela reflexão do raio de luz proveniente da fonte de luz após o choque com a superfície. O vetor S (vetor de visão) é definido com origem no ponto de interseção com a superfície e em direção ao observador.

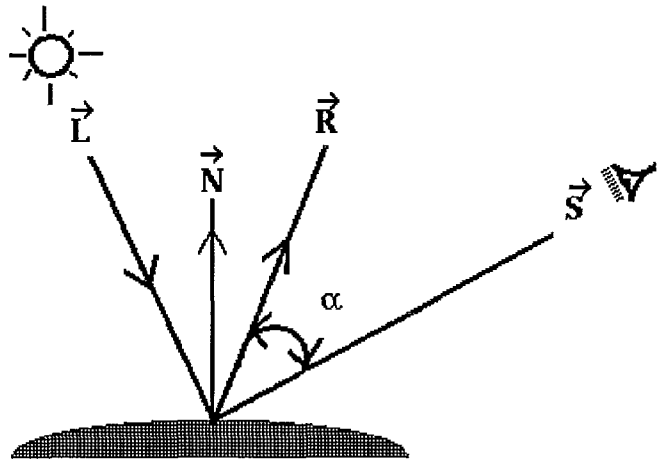


FIGURA II.11 – Reflexão Especular

A quantidade de luz recebida pelo observador é proporcional ao cosseno do ângulo formado por S e R (no caso o ângulo a).

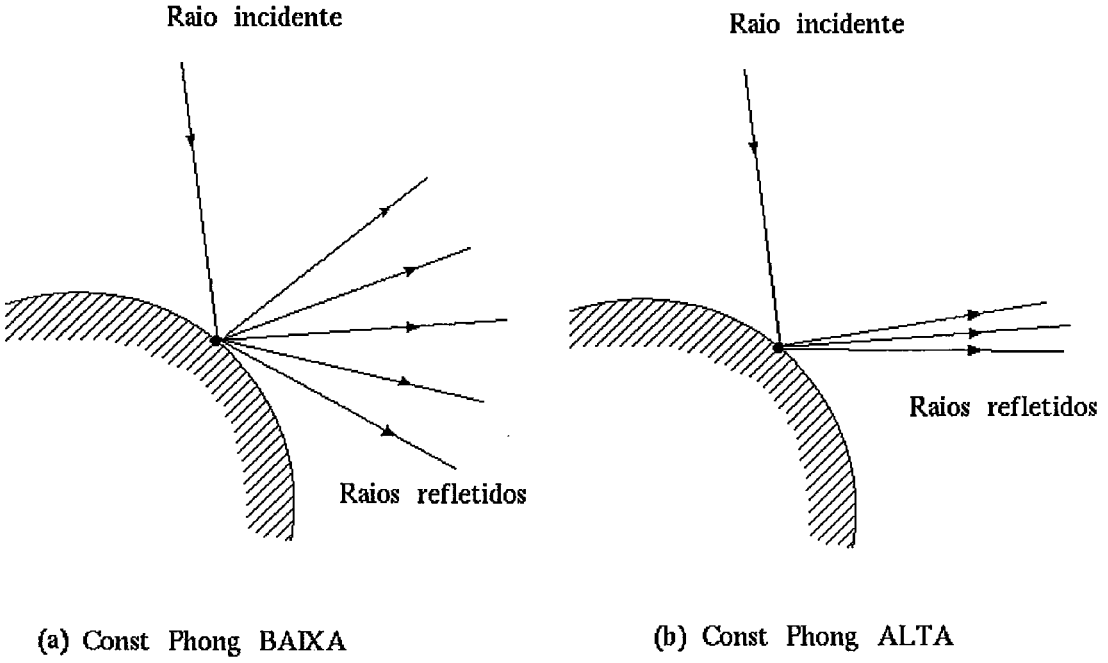
PHONG desenvolveu um modelo para estimar a concentração da reflexão especular na direção da reflexão perfeita.Ele usou o cosseno de α elevado a uma potência **n**, conhecida como constante de PHONG (Ver ROGERS (1985, p. 314)). Altos valores de **n** fazem com que a distribuição especial seja mais focalizada (como em superfícies metálicas), enquanto que valores baixos distribuem mais a luz refletida (figura II.12).

A componente especular é dada pela seguinte fórmula:

$$\text{Componente Especular} = K_s \cdot I_l \cdot \cos^n \alpha \qquad \text{(Equação II.3a)}$$

A variável K_s define a porcentagem de reflexão especular da superfície. I_l expressa a iluminação proveniente da fonte de luz. Se os vetores S e R forem unitários, então a fórmula será dada por:

$$\text{Componente Especular} = K_s \cdot I_l \cdot (\vec{S} \cdot \vec{R})^n \qquad \text{(Equação II.3b)}$$



(a) Const Phong BAIXA (b) Const Phong ALTA

FIGURA II.12 – Aplicação da constante de PHONG

II.2.4.3.5 – Cálculo de Sombras

Em uma cena real é comum o aparecimento de sombras. Sombra (como definido anteriormente por DÜRER) é uma região do espaço que não recebe iluminação direta de um fonte de luz por haver objetos com certo grau de opacidade entre a fonte de luz e esta região.

APPEL propôs a utilização de acompanhamento de raios para também determinar regiões de sombra (estes raios são conhecidos como raios de sombra). Para cada ponto de interseção detectado, joga–se um raio em direção a fonte de luz, para descobrir se existe um objeto entre a fonte de luz e este ponto (no caso de sua implementação todos os objetos eram opacos). Caso alguma interseção fosse encontrada, este ponto estaria em sombra em relação a esta fonte de luz. Para efeitos da aplicação do modelo de iluminação, este ponto receberia somente a componente ambiente, pois a intensidade da fonte de luz (I_l) seria nulo.

II.2.4.3.6 – Cálculo da Intensidade Luminosa

Após termos discutidos os componentes necessários ao cálculo da informação luminosa recebida pelo observador, podemos definir o seu cálculo:

$$\text{Intensidade} = \text{Componente Ambiente} + \text{Componente Difusa} + \text{Componente Especular} \qquad \text{(Equação II.4)}$$

II.2.4.4 – Pseudocódigo do Algoritmo de Acompanhamento de Raios

O algoritmo de acompanhamento de raios proposto por APPEL pode ser descrito pelo seguinte pseudocódigo:

```
PROCEDURE Ilumina
    (
        Raio     : Reta ;              (* Equacao Parametrica do Raio *)
        Cena     : TipoCena ;          (* Descricao da Cena *)
        Colisao : DadosColisao ;       (* Dados sobre a colisao *)
    ) :
        Cor ;                          (* Cor obtida apos aplicacao do modelo *)

    BEGIN
        VetorPontoLuz := Diferenca (PosicaoFonte, Colisao.PtoColisao)
        (* Verifica se este ponto esta em sombra *)
        Sombra := CalculaIntersecao (VetorPontoLuz, Cena) ;
        IF Sombra THEN
            PesoDifusa := 0.0 ;
            PesoEspecular := 0.0 ;
        ELSE
            VetorLuzPonto := InverteDirecao (VetorPontoLuz) ;
            VetorObservador := InverteDirecao (Raio) ;
            VetorRefletido := Reflexao (VetorLuzPonto, Colisao.Normal) ;
            PesoDifusa := ProdutoEscalar (Colisao.Normal, VetorPontoLuz) ;
            PesoEspecular :=
                Potencia(
                    ProdutoEscalar (
                        VetorObservador, VetorRefletido
                    ),
                    Colisao.Material.CtePhong ) ;
        END ;
```

```
        (* Calcula a intensidade para cada uma das cores primarias *)
        FOR ICor := MIN(CorPrimaria) TO MAX(CorPrimaria) DO
            ComponenteAmbiente := Colisao.Material.Ka [ICor] * Ia [ICor] ;
            ComponenteDifusa :=
                Colisao.Material.Kd [ICor] * Fonte.Intensidade [ICor] * PesoDifusa ;
            ComponenteEspecular :=
                Colisao.Material..Ks [ICor] * Fonte.Intensidade [ICor] *PesoEspecular ;
            IntensidadeTotal [ICor] :=
                ComponenteAmbiente +
                ComponenteDifusa +
                ComponenteEspecular ;
        END ;
        RETURN IntensidadeTotal ;
    END Ilumina ;

PROCEDURE AcompanhaRaio
    (
        Raio : Reta ;           (* Equacao Parametrica do Raio *)
        Cena : TipoCena ; (* Descricao da Cena *)
    ) :
        Cor                     (* Cor obtida, descrita no RGB*)

    VAR
        Colisao : DadosColisao ;

    BEGIN
        Colisao := CalculaIntersecao (Raio, Cena) ;
        IF (Colisao.Material  NIL) THEN
            (* Houve intersecao do raio com a cena *)
            RETURN Ilumina (Raio, Cena, Colisao)
        ELSE
            (* Nao houve intersecao, logo retorna a cor de fundo *)
            RETURN CorFundo
        END
    END AcompanhaRaio ;
```

II.2.5 – Acompanhamento de Raios Recursivo: o trabalho de Whitted

O trabalho de APPEL foi muito importante por apresentar as idéias do algoritmo de acompanhamento de raios. Entretanto, o trabalho fundamental para a disseminação do mesmo foi apresentado por WHITTED (1980).

WHITTED trabalhou com o algoritmo para permitir representar cenas com objetos transparentes e refletivos. Isto, como vimos, era impossível de obter com o algoritmo anterior, pois este acompanhava raios somente até a primeira interseção (aplicando a seguir o modelo de iluminação).

Portanto, somente objetos opacos poderiam ser descritos anteriormente. Esta restrição tornou-se sério impedimento para a obtenção de cenas com realismo. WHITTED resolveu

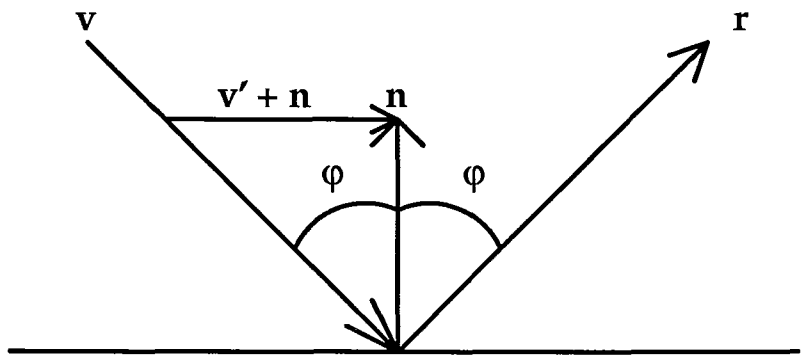
justamente este problema, alterando o algoritmo e fundamentalmente propondo um novo modelo de iluminação.

Vamos estudar a seguir a nova idéia do algoritmo, e o modelo de iluminação proposto, conhecido como modelo global de iluminação.

II.2.5.1 – A nova idéia do algoritmo

WHITTED notou que para conseguir obter superfícies refletidas (como o espelho) ou transparentes (como o vidro) seria necessário que o processo de acompanhamento de raios não parasse na primeira interseção.

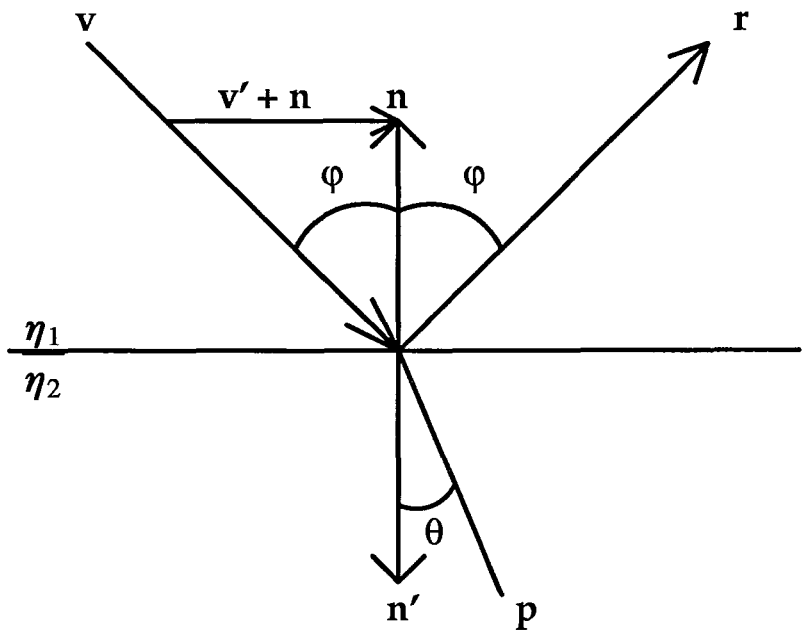
Para superfícies refletidas, como seria o raio a ser acompanhado em sequência ? WHITTED usou os conceitos da ótica, calculando este raio como sendo a reflexão do raio principal (aquele que parte do observador) quando este colide com a superfície. Assim, para toda superfície refletida um novo raio deveria ser gerado e acompanhado (e assim sucessivamente). O cálculo do raio refletido é mostrado na figura II.13.



$$v' = \frac{v}{|\, v \,.\, n \,|} \qquad r = v' + 2n$$

FIGURA II.13 – Cálculo do raio refletido

Assim como na reflexão, raios também podem ser acompanhados através de superfícies transparentes (num processo de refração). Para o cálculo do raio refratado é necessário saber novos dados sobre a superfície, como seu coeficiente de transparência e o índice de refração relativo entre ela e o meio onde está inserida. O cálculo do raio refratado é mostrado na figura (II.14).



$$K_\eta = \frac{\eta_1}{\eta_2} \quad \text{(índice de refração relativo)}$$

$$K_f = (K_\eta^2 \,.\, |\, v' \,|^2 - |\, v' + n \,|^2)^{\frac{-1}{2}}$$

$$p = K_f \,.\, (\, n + v' \,) - n$$

FIGURA II.14 – Cálculo do raio refratado

O acompanhamento de mais raios onera sensivelmente o algoritmo, já que para cada raio principal dois novos raios podem ser gerados (relativos à reflexão e à refração), e assim sucessivamente. A figura II.15 mostra a aplicação do algoritmo sobre uma cena simples.

O modelo de iluminação também dever ser alterado, pois os novos raios acompanhados devem de alguma forma contribuir para a intensidade luminosa recebida pelo observador. O modelo de iluminação proposto por WHITTED é apresentado a seguir.

II.2.5.2 – O modelo de iluminação global

II.2.5.2.1 – O que muda em relação ao modelo anterior

O modelo de iluminação anterior foi projetado para lidar com 3 tipos de componentes : componente ambiente, componente especular e componente difusa. Agora, duas novas

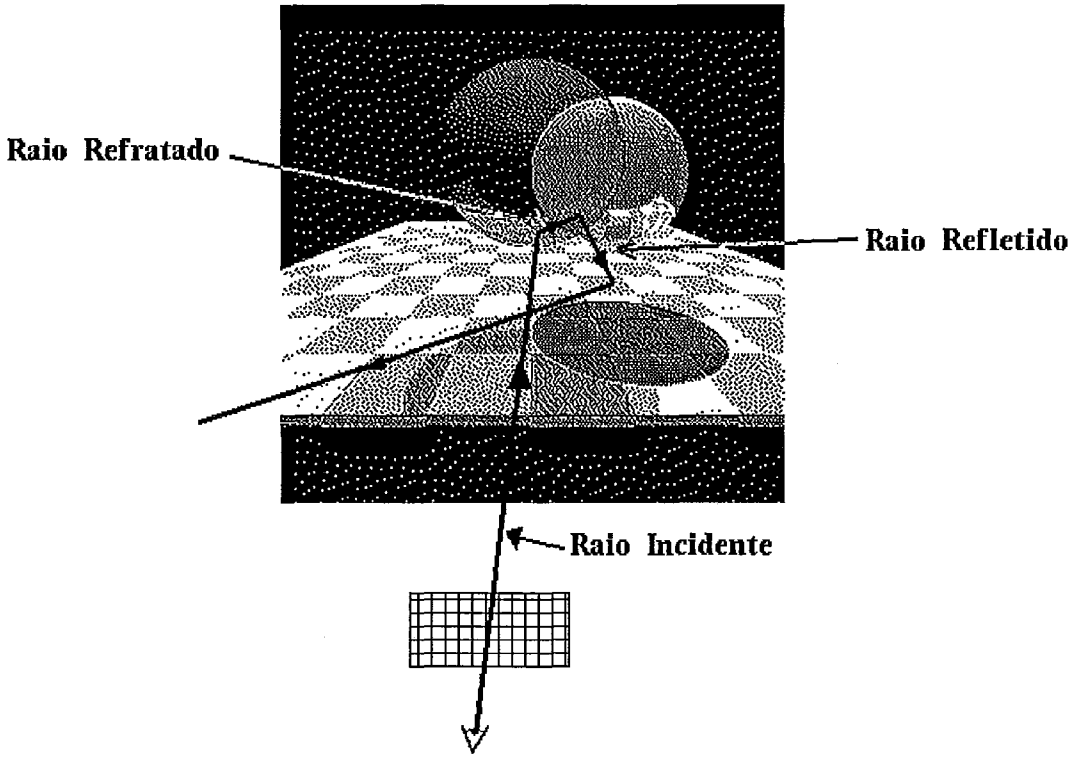


FIGURA II.15 – Acompanhamento de raios refletidos e refratados

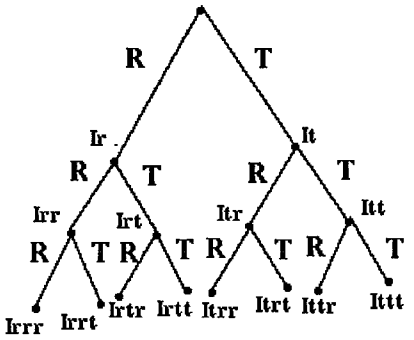
componentes devem ser introduzidas, referente à contribuição imposta pelo acompanhamento de raios refletidos e refratados. Estas serão chamadas de componente refratada e componente refletida.

Como vimos anteriormente, a cada raio principal gerado, podemos obter dois novos raios. Avaliando o pior caso, se acompanharmos raios após n interseções teremos 2 n+1 raios. Com uma simples análise, podemos ver que a estrutura de dados adequada para armazenar este tipo de relacionamento é uma árvore binária. Esta árvore será referenciada como árvore de raios refletidos e refratados (ARR) – figura II.16.

Cada nó não terminal desta árvore representa uma interseção de um raio com a cena durante um determinado acompanhamento. Logo, pode–se aplicar um modelo de iluminação de forma a calcular a quantidade de iluminação recebida por este ponto.

A visita aos nós não terminais desta árvore (para cálculo de iluminação) deve ser feita em pós–ordem (onde os filhos são visitados antes do pai). Este tipo de percurso deve–se ao fato da contribuição imposta pelos raios refletidos e refratados ser feita no sentido contrário ao sentido dos raios durante a aplicação do algoritmo. Usando este percurso, a informação de reflexão e refração é propagada pela árvore no sentido folhas–raiz.

Arvore de Raios Refletidos e Refratados (ARR)



$$Irr = Irrr + Irrt$$
$$Irt = Irtr + Irtt$$
$$Itr = Itrr + Itrt$$
$$Itt = Ittr + Ittt$$
$$Ir = Irr + Irt$$
$$It = Itr + Itt$$

Componente Refletida = Ir

Componente Refratada = It

FIGURA II.16 – Árvore de raios refletidos e refratados (ARR)

WHITTED introduziu no seu modelo de iluminação a possibilidade de existir mais de uma fonte de luz. Com isto, o processamento feito somente uma vez (para uma determinada fonte de luz), deverá agora ser feito tantas vezes quantas forem o número de fontes de luz.

A componente ambiente mantém–se a mesma do modelo anterior. Entretanto, as compontes difusa e especular são estendidas, para suportar um número variável de fontes de luz. A nova equação destas componentes é a seguinte:

$$\text{Componente Difusa} = K_d \cdot \sum_{j=1}^{m} I_{lj} \cdot (\vec{N} \cdot \vec{L}_j) \qquad \text{(Equação II.5)}$$

$$\text{Componente Especular} = K_s \cdot \sum_{j=1}^{m} I_{lj} \cdot (\vec{S} \cdot \vec{R})^n \qquad \text{(Equação II.6)}$$

A variável m refere–se ao número de fontes de luz.

II.2.5.2.2 – As novas componentes criadas: componente refletida e refratada

A nova componente refletida será dada pela seguinte fórmula:

$$\textbf{Componente Refletida} = K_s \cdot I_s \qquad \textbf{(Equação II.7)}$$

A variável I_s refere–se a intensidade luminosa especular da subárvore de reflexão. Isto é, representa o valor propagado desde a folha em direção a raiz (até atingir o ponto usado no cálculo da iluminação).

A nova componente refratada será dada pela seguinte fórmula:

$$\textbf{Componente Refratada} = K_t \cdot I_t \qquad \textbf{(Equação II.8)}$$

A variável K_t representa à porcentagem de trasparência do material. Se K_t for igual a zero o material é totalmente opaco, enquanto que se for igual a 1, será totalmente transparente. I_t refere–se a intensidade luminosa referente à subárvore de refração.

II.2.5.2.3 – O novo cálculo de sombras

O cálculo de sombras fica mais complicado no modelo proposto por WHITTED por dois problemas básicos:

- A possibilidade de existir mais de uma fonte de luz
- O fato de nem todos objetos serem opacos

O primeiro problema é de fácil solução: o procedimento de sombra (que antes era calculado somente para uma fonte de luz) deve agora ser repetido para cada uma das fontes de luz existentes. Com isso, um ponto pode estar em sombra em relação a uma fonte de luz, mas pode ser iluminado por outra. Isto causa um novo acréscimo no número de raios a serem acompanhados, visto que a cada ponto de interseção achado, teremos que acompanhar raios (tantas quantas forem as fontes de luz), para verificar quais fontes de luz incidem sobre este ponto.

Pelo fato dos objetos não serem opacos, é possível que determinados pontos estejam recebendo uma fração de iluminação de uma dada fonte de luz. WHITTED propôs que o acompanhamento do raio de sombra não parasse caso fosse encontrado objetos com certos graus de transparência, atenuando a intensidade da fonte de luz proporcionalmente ao coeficiente de transparência do objeto. O processo só seria encerrado caso um objeto opaco fosse encontrado. Com isto, o algoritmo pode sofrer um acréscimo de tempo, pois o cálculo da sombra não será tão simples.

Um cuidado que se deve tomar quando acompanhar raios de sombra é considerar objetos que estiverem entre a fonte de luz e o ponto de interseção. É comum a rotina de interseção entre raios e objetos usar a equação paramétrica da reta, e as interseções encontradas armazenadas em função do parâmetro da reta. Portanto, deve-se tomar o cuidado de aproveitar valores de parâmetro dentro do intervalo definido entre o ponto de interseção e a fonte de luz. Uma implementação descuidada pode determinar que um objeto que está além ou aquém da fonte de luz (mas que faz interseção com o raio de sombra) está causando sombra neste ponto.

Outro ponto crítico é que a idéia de WHITTED para o cálculo de sombras é errada. O acompanhamento de raios de sombra é feito sobre uma trajetória retilínea entre o ponto de interseção e a fonte de luz. Ao continuar o acompanhamento sobre objetos transparentes, seria correto calcular se o raio não teria mudado de direção. Caso isso acontecesse, este dificilmente atingiria a fonte de luz. Sabendo disso, WHITTED não aplicou a refração sobre raios de sombra, o levou a uma incorreção. Embora este cálculo seja fisicamente errado, ele é bastante utilizado nas implentações conhecidas do algoritmo (inclusive na presente).

II.2.5.2.4 – O novo cálculo da intensidade total

A intensidade total no modelo global é dada pela seguinte fórmula:

$$\begin{aligned} \textbf{IntensidadeTotalModeloGlobal} = {} & \textbf{Componente Ambiente} + \\ & \textbf{Componente Difusa} + \\ & \textbf{Componente Especular} + \\ & \textbf{Componente Refletida} + \\ & \textbf{Componente Refratada} \end{aligned} \qquad \textbf{(Equação II.9)}$$

II.2.5.3 – O pseudocódigo do acompanhamento de raios recursivo

O algoritmo de acompanhamento de raios proposto por WHITTED pode ser descrito pelo seguinte pseudocódigo:

```
TYPE
    TipoRaio = (Incidente, Refletido, Refratado) ;

PROCEDURE IluminaGlobal
    (
        Raio                    : Reta ;          (* Equacao Parametrica do Raio *)
        Cena                    : TipoCena ;      (* Descricao da Cena *)
        Colisao                 : DadosColisao ;  (* Dados sobre a colisao *)
        IntensidadeRefletida,                     (* Proveniente da reflexao acompanhada *)
        IntensidadeRefratada    : Cor             (* Proveniente da refracao acompanhada *)
    ) :
        Cor ;                                     (* Cor obtida *)

    BEGIN
        (* Verifica se este ponto esta em sombra para cada uma das fontes de luz *)
        FOR IFonte := 1 TO NumeroFontes DO
            VetorPontoLuz := Diferenca (Fonte.Posicao, Colisao.PtoColisao) ;
            (* Calcula a atenuacao da fonte de luz, referente a porcentagem
              de R, G e B da fonte luminosa que ilumina o ponto *)
            Atenuacao [IFonte] := AtenuaFonteLuz (VetorPontoLuz, Cena) ;
            VetorLuzPonto := InverteDirecao (VetorPontoLuz) ;
            VetorObservador := InverteDirecao (Raio) ;
            VetorRefletido := Reflexao (VetorLuzPonto, Colisao.Normal) ;
            PesoDifusa [IFonte] := ProdutoEscalar (Colisao.Normal, VetorPontoLuz) ;
            PesoEspecular [IFonte] :=
                Potencia (
                    ProdutoEscalar (VetorObservador, VetorRefletido),
                    Colisao.Material.CtePhong
                ) ;
        END ;
        (* Calcula a intensidade para cada uma das cores primarias *)
        FOR ICor := MIN(CorPrimaria) TO MAX(CorPrimaria) DO
            ComponenteAmbiente := Colisao.Material.Ka [ICor] * Ia [ICor] ;
            ComponenteRefletida := Colisao.Material.Ks [ICor] * IntensidadeRefletida [ICor] ;
            ComponenteRefratada := Colisao.Material.Kt [ICor] * IntensidadeRefratada [ICor] ;
            (* Calcula o somatorio das componentes difusa e especular *)
```

```
            (* Calcula o somatorio das componentes difusa e especular *)
            FOR IFonte := 1 TO NumeroFontes DO
                ComponenteDifusa :=
                    ComponenteDifusa +
                    (PesoDifusa [IFonte] * Fonte.Intensidade [ICor] *
                    Atenuacao [IFonte, ICor]) ;
                ComponenteEspecular :=
                    ComponenteEspecular +
                    (PesoEspecular [IFonte] * Fonte.Intensidade [ICor] *
                    Atenuacao [IFonte, ICor]) ;
            END ;
            ComponenteDifusa := (ComponenteDifusa * Colisao.Material.Kd) ;
            ComponenteEspecular := (ComponenteEspecular * Colisao.Material.Ks) ;
            IntensidadeTotal [ICor] :=
                ComponenteAmbiente + ComponenteDifusa + ComponenteEspecular +
                ComponenteRefletida + ComponenteRefratada ;
        END
        RETURN IntensidadeTotal ;
    END IluminaGlobal ;

PROCEDURE AcompanhaRaioRecursivo
    (
        Raio        : Reta ;        (* Equacao Parametrica do Raio *)
        Cena        : TipoCena ;    (* Descricao da Cena *)
        Prof        : CARDINAL ;    (* Limite da recursividade do algoritmo *)
        Tipo        : TipoRaio ;    (* Tipo do Raio: Incidente, Reflexao ou Refracao *)
    ):
        Cor                         (* Cor obtida, descrita no RGB*)

    BEGIN
        IntensidadeRefletida := CorFundo ;
        IntensidadeRefratada := CorFundo ;
        Colisao := CalculaIntersecao (Raio, Cena) ;
        IF (Colisao.Material = NIL) THEN (* Nao houve intersecao do raio com a cena *)
            RETURN CorFundo
        ELSE
            IF Prof >= ProfundidadeMaxima THEN
                RETURN
                    IluminaGlobal
                        ( Raio, Cena, Colisao,
                          IntensidadeRefletida, IntensidadeRefratada ) ;
            END ;
            IF MaterialReflete (Colisao.Material.Ks) THEN
                RaioRefletido :=
                    CriaReta (
                        Colisao.PtoColisao,
                        VetorRefletido (Raio.Direcao, Colisao.Normal)
                    ) ;
                IntensidadeRefletida :=
                    AcompanhaRaio ( RaioRefletido, Cena, Prof + 1, Refletido ) ;
            END ;
```

```
        IF MaterialRefrata (Colisao.Material.Kt) THEN
            IF (Tipo = Refratado) THEN
                (* O Raio esta saindo do objeto *)
                RaioRefratado :=
                    CriaReta (
                        Colisao.PtoColisao,
                        VetorRefratado (
                            Raio.Direcao,
                            InverteDirecao (Colisao.Normal),
                            1.0 / Colisao.Material.IndiceRefracao
                        )
                    )
            ELSE
                (* O Raio esta entrando no objeto *)
                RaioRefratado :=
                    CriaReta (
                        Colisao.PtoColisao,
                            VetorRefratado (
                                Raio.Direcao,
                                Colisao.Normal,
                                Colisao.Material.IndiceRefracao
                        )
                    );
            END ;
            IntensidadeRefratada :=
                AcompanhaRaio (RaioRefratado, Cena, Prof + 1, Refratado ) ;
        END ;
        RETURN
            IluminaGlobal (
                Raio, Cena, Colisao,
                IntensidadeRefletida, IntensidadeRefratada ) ;
    END
END AcompanhaRaioRecursivo ;
```

Capítulo III

Definição de Cenas

III.1 – INTRODUÇÃO

A apresentação do algoritmo de acompanhamento de raios feita no capítulo anterior não entrou em detalhes sobre a cena a qual o algoritmo estava sendo aplicado. Para efeitos didáticos, a idéia do algortimo foi apresentada de forma a manter uma certa independência em relação à descrição da cena.

Neste capítulo iremos descrever a especificação de cenas. Para tanto, será feito inicialmente uma discussão sobre a área responsável pelo estudo de sólidos, conhecida como modelagem de sólidos.

III.2 – MODELAGEM DE SÓLIDOS

III.2.1 – O uso de modelos

A criação de um modelo de algum objeto deve ser feita baseada em um determinado critério. Este deve definir quais aspectos do objeto a ser modelado serão considerados, de forma a definir quais características serão modeladas. Além disto, este critério deve ser criado de forma a manter a maior fidelidade possível entre o objeto original e seu substituto (obtido através do modelo). A fidelidade é importante no sentido que o substituto possa responder determinadas questões da mesma forma que o original o faria.

Na modelagem de sólidos buscamos representar os objetos do mundo real sobre o ponto de vista de sua geometria, a fim de poder responder sobre uma série de questões geométricas. Entretanto, modelar os objetos do mundo real não é uma das tarefas mais fáceis. Muitos componentes não possuem uma forma geométrica facilmente explicada pela geometria euclideana – principalmente os componentes da natureza (nuvens, árvores, etc). Recentemente, com o advento da teoria de fractais (MANDELBROT (1982)), muitos avanços foram feitos para entender o formato irregular da natureza.

Este exemplo apenas ilustra algumas das dificuldades na modelagem de objetos reais. Entretanto, existem diversos outros componentes de modelagem absolutamente simples, principalmemte aqueles derivados de formas geométricas simples, como por exemplo esferas, cilindros, cones, paralelepípedos, etc. Visto que é muito difícil criar um modelo para representar qualquer tipo de objeto, deve–se projetar um modelo de acordo com os tipos de objetos que se quiser representar

A construção de modelos sobre objetos sólidos tridimensionais é estudada pela área da Modelagem de Sólidos. Uma boa definição para o escopo de estudo desta área nos foi dada por MORTENSON (1985):

"A modelagem de sólidos é a área da modelagem geométrica responsável pela criação e manipulação de formas de objetos físicos, que devem ser descritos segundo um modelo não ambíguo, informal e matematicamente completo".

A seguir veremos alguns dos principais modelos já apresentados para representação de sólidos.

III.2.2 – Modelos de representação de sólidos

REQUICHA (1980) publicou um trabalho muito importante para a área de modelagem de sólidos. O mérito maior do trabalho dele foi organizar a área, criando definições e taxonomias que permitiram um estudo mais sistemático da área.

O trabalho de REQUICHA contribui substancialmente por:

- Definir o que são esquemas de representação de sólidos rígidos
- Estabelecer as propriedades matemáticas dos modelos
- Definir propriedades formais e informais de modelos
- Enumerar esquemas de representação de sólidos

O tratamento matemático, de notável rigor, imposto por REQUICHA na definição de esquemas de representação, bem como a caraterização de propriedades destes esquemas fez com que este trabalho fosse uma referência sempre presente em trabalhos de modelagem de sólidos. Os modelos enumerados por REQUICHA são os seguintes:

- **Esquemas Ambíguos**: Ex. "Wireframe"
- **Instâncias Primitivas**: O sólido é uma instância de uma família de objetos especificados parametricamente
- **Enumeração da ocupação espacial**: O sólidos é representado por uma lista de voxels (cubos) localizados numa malha espacial
- **Decomposição por Células**: Diferem do esquema anterior pelo fato das células não serem cúbicas, mas de forma arbitrária
- **Geometria de Sólidos Construtiva (CSG)**: Os sólidos são definidos como combinações booleanas de outros sólidos
- **"Sweeping"**: Os sólidos são descritos seguindo o movimento de uma região seguindo uma direção (sólidos cilíndricos e de revolução)
- **Representação por Fronteitas (BREP)**: O sólido é descrito por sua superfície de fronteira.

A caracterização destes modelos é apresentada de forma ampla em REQUICHA (1980). A seguir detalharemos o modelo CSG e o modelo OCTREE (que foi apresentado após o trabalho de REQUICHA), que é um caso particular de enumeração da ocupação espacial.

III.2.3 – Geometria de Sólidos Construtiva (CSG)

Uma maneira simples de construir um sólido de formato complexo é obtê–lo através de composições de partes mais simples. Por exemplo, uma mesa poderia ser obtida pela composição de 4 cilindros com um paralelepípedo. Por vezes também podemos criar un sólido informando que um determinado componente não contribui positivamente em relação ao mesmo, isto é, retirando do sólido a porção que este componente ocupa. Este caso é bastante comum, como por exemplo: uma porca de parafuso pode ser criada através de dois

cilindros, um mais estreito que define a forma da porca, e outro mais interno, contribuindo negativamente e assim criando o buraco existente dentro.

Esta metododologia para criação de sólidos é bastante natural para nós. Por isso mesmo seria interessante que quando quisessemos criar um sólido no computador pudéssemos fazê–lo usando este tipo de abordagem.

O modelo CSG usa justamente este tipo de abordagem, em que os sólidos são construídos a partir de formas mais simples. Para explicá–lo mais formalmente devemos esclarecer dois pontos:

- O que são sólidos simples em CSG.
- Como é descrita a composição de sólidos simples.

Os sólidos simples em CSG são usualmente referenciados como primitivas CSG. A composição de sólidos CSG é feita através de operações booleanas regularizadas. A seguir iremos descrever com mais detalhes cada um destes tópicos.

III.2.3.1 – Primitivas CSG

Os sólidos simples em CSG são usualmente chamados de Primitivas. Estas primitivas são instâncias do que se convencionou ser um sólido, isto é, satisfazem as condições enumeradas para o que seja sólido. Não existe, entretanto, necessidade que o conjunto de primitivas disponível em um modelador CSG seja abrangente o suficiente para modelar todos os sólidos que a definição de sólido permite. O sólido CSG é definido da seguinte forma:

Um sólido é um conjunto de pontos P do E3 (espaço euclideano de dimensão 3) que atende as seguintes condições:

- Limitado
- Fecho(Interior(P)) = P
- Descrever sua fronteira por uma superfície semi–analítica por partes

Qual será o critério usado para delimitar o corpo de primitivas de um modelador CSG? O principal critério utilizado para se definir quais primitivas estarão disponíveis está intimamente ligado com a representação de conjuntos de pontos. Como tais conjuntos podem (entre outras maneiras) serem expressos por desigualdades algébricas, o uso de semi–espaços mostrou–se bastante interessante. Estes podem ser entendidos como conjuntos de pontos expressos por desigualdades, que são úteis para a verificação da pertinência de pontos.

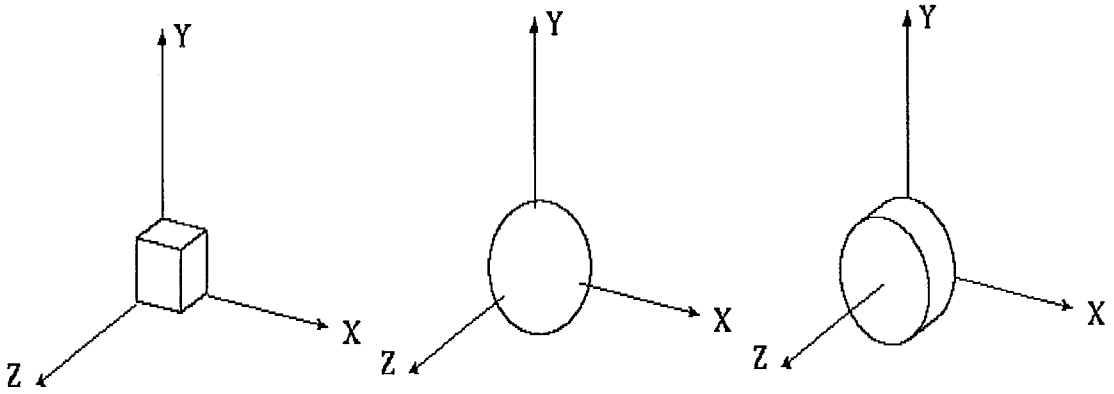
É comum usarmos funções lineares, quádricas, cúbicas, quárticas, etc, para definirmos um semi–espaço. Dentre estas as mais comuns são as equações de esferas, cones, toros, cilindros, etc. Um cuidado deve ser tomado, pois estaremos trabalhando em alguns casos com

semi–espaços limitados (como no caso da esfera), e com semi–espaços ilimitados (como no caso do cilindro).

As primitivas CSG serão representadas por semi–espaços, ou até mesmo um conjunto destes, de tal forma que a condição de ser sólido seja satisfeita. E a condição de que este sólido seja limitado pode em primeira análise nos restringir a semi–espaços limitados. Como poderíamos criar uma primitiva cilindro?

Uma primitiva definida por um semi–espaço cilíndrico nos define dois semi–espaços ilimitados. Para criar um semi–espaço limitado seria necessário que aplicássemos mais dois semi–espaços ao semiespaço cilíndrico, sendo estes planares, criando assim um cilindro limitado.

As primitivas presentes no modelador de sólidos disponível (ver ESPERANÇA (1990b)) são: cilindros, esferas e blocos. O cilindro é constituído de 3 semiespaços, a esfera de 1 e o bloco de 6 semi–espaços. A figura III.1 mostra estas primitivas e sua localização em um sistema de coordenadas – chamado de Sistema de Coordenadas da Primitiva (SCP).



(a) Bloco (b) Esfera (c) Cilindro

FIGURA III.1 – Primitivas definidas no SCPrimitiva

III.2.3.2 – Como combinar primitivas CSG

Como mencionado anteriormente as primitivas CSG são combinadas através de operações booleanas regularizadas. Além destas existem também as operações de instanciação, responsáveis pelo aumento no corpo de primitivas. Tais operações são discutidas a seguir.

III.2.3.3 – Usando operações booleanas regularizadas

Uma vez estabelecidas as primitivas que podem compor um sólido CSG podemos então discutir como esta composição é feita. Basicamente, uma primitiva pode ser composta através de operações booleanas, que são largamente utilizadas na teoria de conjunto de pontos para combinar dois conjuntos de pontos. As operações mais conhecidas e utilizadas são a União, Interseção e Diferença. Uma definição mais formal destas operações é a seguinte:

$$A \cup B = \{ p \in E^3 \mid p \in A \text{ ou } p \in B \} \qquad \text{(Equação III.1)}$$

$$A \cap B = \{ p \in E^3 \mid p \in A \text{ e } p \in B \} \qquad \text{(Equação III.2)}$$

$$A \setminus B = \{ p \in E^3 \mid p \in A \text{ e } p \notin B \} \qquad \text{(Equação III.3)}$$

O uso destas operações deve ser feito tomando um certo cuidado, devido ao fato que o resultado da combinação de duas primitivas segundo uma operação booleana pode resultar um conjunto de pontos que não satisfaça a condição de sólido. Para tanto, usa-se uma variante destas operações boolenas, chamadas de operações booleanas regularizadas, que diferem das tradicionais pelo fato de aplicarem uma operação de regularização sobre o conjunto obtido (ver REQUICHA (1980)). As operações regularizadas são definidas da seguinte forma:

$$A \cup^* B = \text{fecho}\ (\ \text{interior}\ (\ A \cup B\ )\ ) \qquad \text{(Equação III.4)}$$

$$A \cap^* B = \text{fecho}\ (\ \text{interior}\ (\ A \cap B\ )\ ) \qquad \text{(Equação III.5)}$$

$$A \setminus^* B = \text{fecho}\ (\ \text{interior}\ (\ A \setminus B\ )\ ) \qquad \text{(Equação III.6)}$$

III.2.3.4 – Usando operações de instanciação

As primitivas como descritas anteriormente são definidas em um determinado sistema de coordenadas (SCPrimitiva). Para que tenhamos uma maior variedade de sólidos seria interessante que pudéssemos instanciar estas primitivas de diversas formas no espaço, seja através de tranformações rígidas como translação e rotação, seja como transformações de escala. Como estamos trabalhando com primitivas expressas por semi–espaços situadas em um adequado sistema de coordenadas), a aplicação de tais transformações geométricas sobre uma primitiva é feita de forma também simples (envolvendo simples multiplicação de matrizes).

As transformações geométricas que podem usualmente ser aplicadas são as seguintes:

- **Translação** (x, y, z) : Mover a primitiva no espaço x unidades no eixo X, y unidades no eixo Y e z unidades no eixo Z

- **Rotação** (a, b, c) : Rotacionar a primitiva a graus em torno do eixo X, b em torno do eixo Y e c em torno do eixo Z

- **Escala** (x, y, z) : Escalar a primitiva x unidades no eixo X, y unidades no eixo Y e z unidades no eixo Z

Este corpo de transformações geométricas é definido através de uma matriz de transformações geométricas (MTG). Cada primitiva da árvore CSG terá associada uma MTG, responsável por instanciar a primitiva na cena. O sistema de coordenadas para onde a primitiva será transportada será comumente chamado de Sistema de Coordenadas da Cena (SCCena). Por motivos que explicaremos posteriormente, será armazenado junto à primitiva na árvore CSG além da MTG sua correspondente matriz inversa.

III.2.3.5 – Estrutura de dados para representar um sólido CSG

O sólido CSG será nada mais que uma expressão boolena, onde os operandos são as primitivas CSG, e os operadores são as operações booleanas regularizadas. A estrutura de dados natural para se representar dados desta natureza é uma árvore binária.

Em uma árvore binária temos dois componentes fundamentais: os nós terminais e os nós não–terminais. Nos nós terminais são armazenados os dados, enquanto que nos nós não–terminais são armazenadas as operações.

No caso de um sólido CSG os nós terminais serão as primitivas CSG, enquanto que nos nós não terminais temos as operações boolenas regularizadas. A figura III.2 mostra a árvore correspondente a um sólido CSG.

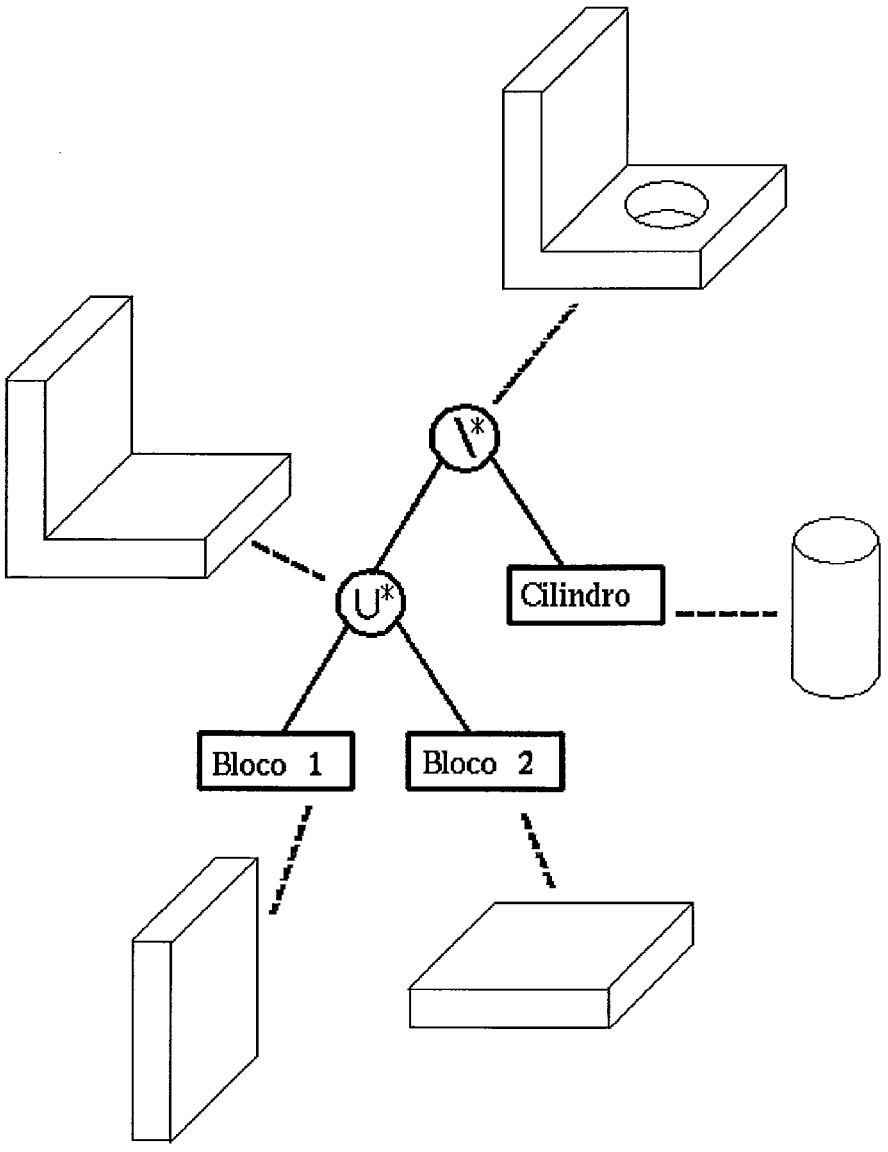


FIGURA III.2 – Árvore correspondente a um sólido CSG

III.2.3.6 – Acompanhamento de Raios em sólidos CSG

Dois problemas surgem ao se jogar um raio em um árvore CSG. O primeiro refere–se ao cálculo de interseção entre o raio e a primitiva. Como as primitivas são passíveis de transformações geométricas, podemos ter testes de interseção de raios contra sólidos mais complexos (elipsóides, romboedros). A idéia de ROTH (1982) simplificou bastante o problema. Sua proposta consiste em se fazer o teste de interseção no SCPrimitiva ao invés de se fazer no SCCena. Para tanto, basta aplicarmos a MTG Inversa sobre o raio, levando–o para o SCPrimitiva. Neste sistema o teste torna–se bastante simples.

O segundo problema consiste em descobrir qual ponto da cena CSG foi primeiramente interceptado pelo raio. É necessário classificar cada primitiva contra o raio em questão, gerando assim uma lista de interseções. Esta lista de interseções define os intervalos em que o raio está dentro da primitiva.

O algoritmo básico consiste em combinar estas listas de forma que tenhamos os intervalos em que o raio esta dentro do sólido CSG. Para tanto, usa-se uma aritmética intervalar para combinar as listas. Este é um algoritmo essencialmente recursivo, cujo núcleo básico deve ser capaz de combinar duas subárvores segundo um operador booleano (Figura III.4).

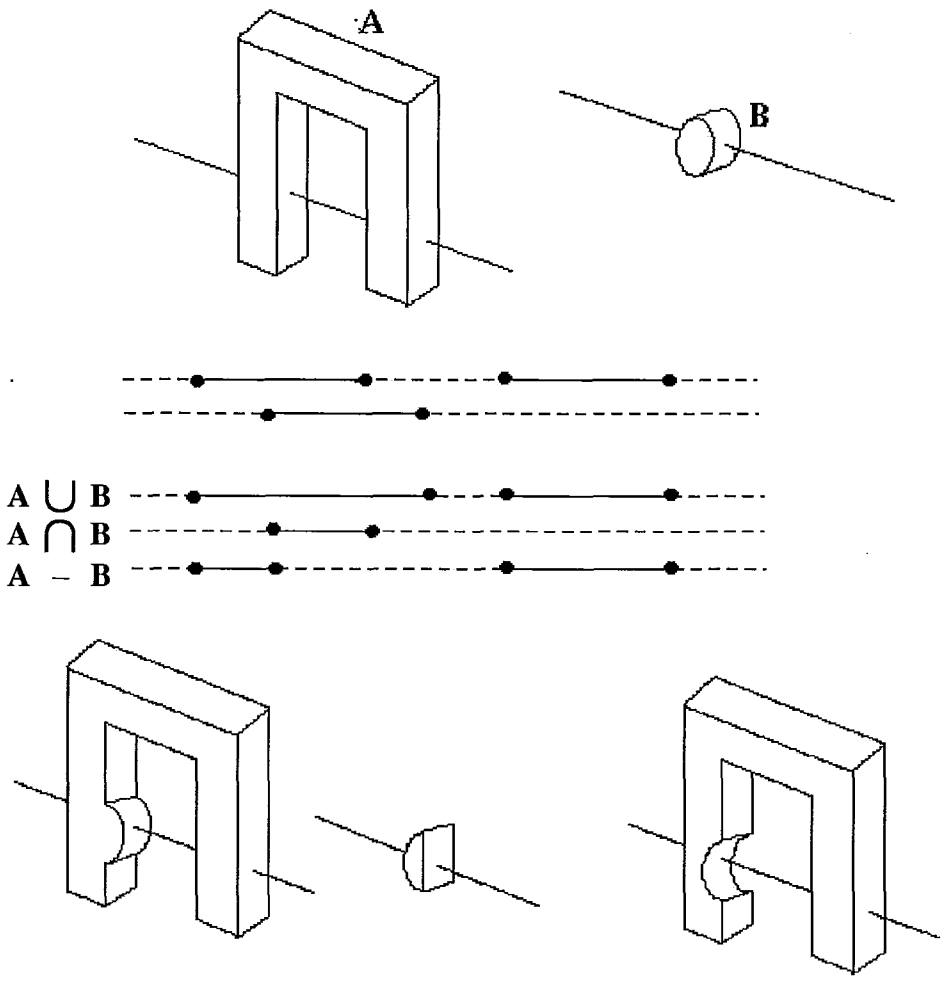


FIGURA III.3 – Acompanhamento de Raios em sólidos CSG

III.2.4 – O modelo OCTREE

III.2.4.1 – Conceituação do modelo

Octree é um caso particular de enumeração espacial de sólidos, onde a representação dos sólidos é baseado na subdivisão recursiva do espaço em oito octantes (células de formato cúbico). Esta subdivisão é feita sobre uma porção cúbica do espaço (chamada de Espaço Octree), onde os sólidos poderão ser definidos.

O modelo Octree pode ser definido a partir dos seguintes parâmetros:

- **Prof**: Parâmetro que define a profundidade da Octree
- **Espaço Octree**: Uma região cúbica de $2^n \times 2^n \times 2^n$ cubos unitários
- **Tipos dos Cubos**: Misto, Vazio ou Cheio
- **Ordenação dos Octantes no cubo**

A representação Octree de objetos é obtida pela subdivisão recursiva deste Espaço Octree até que um determinado critério de parada seja satisfeito. Um octante de tamanho $2^d \times 2^d \times 2^d$ é dividido em oito octantes de tamanho $2^{d-1} \times 2^{d-1} \times 2^{d-1}$ se o cubo unitário contido no octante original não está totalmente preenchido pelo sólido, nem totalmente fora do sólido. O processo continua recursivamente até que cada octante contenha um cubo unitário, ou o processo seja interrompido segundo um dos crtitérios de parada acima.

Cada octante tem associado um número, que é definido pela ordenação proposta para numeração de octantes. Existem diversas formas de criar uma ordenação, mas YAMAGUCHI (1984) propôs uma ordenação (que chamou de ordenação sistemática), que consegue atribuir uma determinada semântica a esta ordenação.

Segundo YAMAGUCHI a ordenação dos octantes (numerados de 0 a 7) pode ser obtida aplicando–se a seguinte função sobre a representação binária do número:

$$\textbf{Seja } p(\text{expressao}) = \begin{cases} 0 & \text{caso expressao falsa} \\ 1 & \text{c.c} \end{cases}$$

$$\textbf{Ordenacao}(x,y,z) = \textbf{ConverteBinarioParaOctal } (p(z \geq 1/2),\ p(y \geq 1/2),\ p(x \geq 1/2))$$

A ordenação proposta por YAMAGUCHI é apresentada na figura III.3.

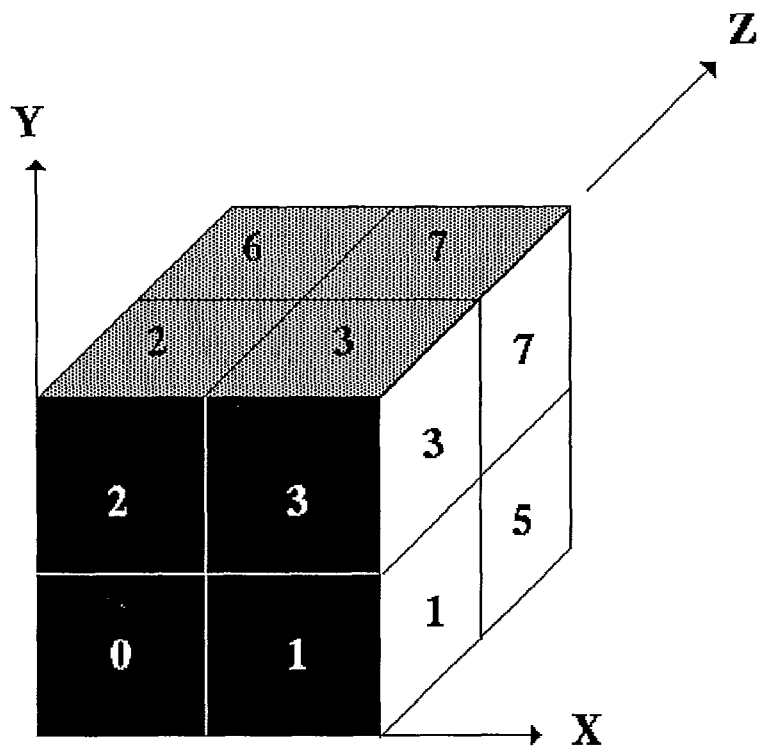


FIGURA III.4 – Ordenação Sistemática proposta por YAMAGUCHI

Na sua proposta mais simples, existem três tipos de nós na Octree:

- nó Misto: Indica que a porção do espaço faz interseção parcial com o objeto, sendo subdivida em oito octantes
- nó Cheio: Indica que a porção do espaço é totalmente preenchida pelo objeto
- nó Vazio: Indica que a porção do espaço não faz interseção com o objeto

Um exemplo de Octree pode ser visto na figura III.5

III.2.4.2 – Vantagens e desvantagens do modelo OCTREE

O modelo Octree possui as seguintes vantagens:

- Poder representar qualquer forma de objeto, sendo eles côncavos ou convexos, com buracos ou não.
- Facilidade no cálculo de propriedades geométricas, tais como: área, centro de massa, volume, interferência.

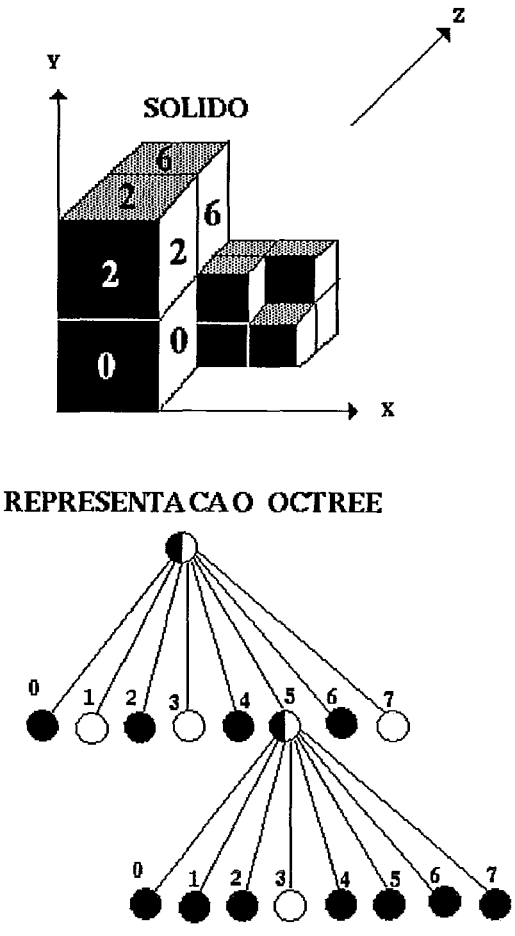


FIGURA III.5 – Sólido descrito no modelo OCTREE

• Facilidade na execução de operações booleanas (como união, interseção, diferença) devido a classificação espacial e a uniformidade na representação.

Por outro lado, possui as seguintes desvantagens:

• A casca de um sólido é representada por um conjunto de facetas quadradas paralelas aos eixos do Espaço Octree, logo detalhes preciosos (como a curvatura da superfície) são perdidos e só podem ser obtidos aproximadamente.

•Sólidos complexos necessitam de um número grande de células para representá–los precisamente

•O sólido descrito em Octree é apenas uma aproximação do sólido, logo somente propriedades aproximadas do sólido podem ser calculadas

• Pelo mesmo motivo acima, sólidos descritos em Octree não podem ser reconstituídos exatamente

• É problemático o cálculo de transformações geométricas (rotação, escala e translação)

III.2.4.3 – Representações de OCTREE

III.2.4.3.1 – Usando ponteiros

A representação mais tradicional de Octrees é através de árvores. Cada nó da árvore possui um campo informando o tipo do nó (misto, cheio, ou vazio) e no caso de nó misto oito ponteiros para os filhos.

Uma implementação por ponteiros poderia ser a seguinte:

```
(* Estruturas de dados para representacao da Octree *)
TYPE
    Tiponó = (VAZIO, MISTO, CHEIO) ;
    PonteiroOctree = POINTER TO nóOctree ;
    nóOctree =
        RECORD
            tipo : Tiponó ;
            CASE tipo OF
                VAZIO,
                CHEIO :
                    materia : CaracteristicasMaterial ;
                MISTO :
                    filhos : ARRAY [0 .. 7] OF PonteiroOctree ;
            END ;
        END ;
```

III.2.4.3.2 – Usando uma representação linear

A idéia da representação linear é a mesma da linearização de árvores. Adota–se um percurso sobre a árvore, gerando a representação linear com os nós que vão sendo visitados sucessivamente pelo percurso. É comum adotar o percurso em preordem durante este processo de linearização, pelo fato de simplificar a conversão da representação linear para a representação por ponteiros.

O algoritmo de linearização adota uma convenção para cada tipo de nó. Seja, por exemplo, a seguinte convenção, onde C refere–se a um nó cheio, V refere–se a um nó Vazio e M refere–se a um nó Misto. A representação linear da Octree da figura III.5 é a seguinte:

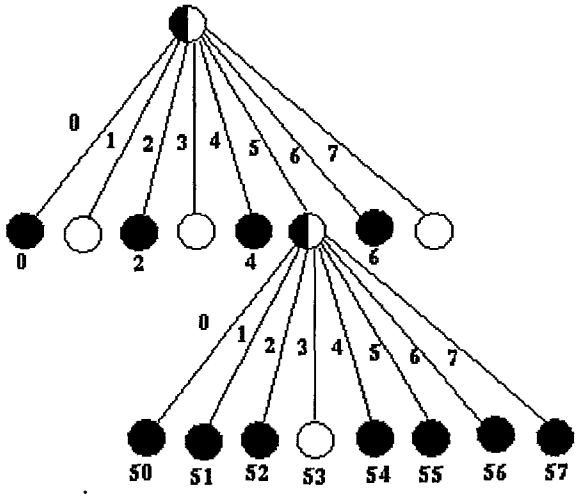
Octree = CVCVCMCCCCCVCCCV

III.2.4.3.3 – Usando códigos de localização

O uso de códigos de localização (chamados de codificação decimal DEDEY por KNUTH) foram estudados por GARGANTINI (1986) para armazenamento de Octrees. O fundamento básico do uso de códigos refere–se ao fato que a árvore pode ser armazenada somente com informações sobre os nós cheios, e como eles estão localizados na árvore.

Para isso, é criado um código que identifica univocamente cada posição da árvore. Este código nada mais é que o caminho mínimo existente entre a raiz da árvore e o nó que se deseja codificar, que consiste em um arranjo de (no máximo) p dígitos octais, onde p é a profundidade máxima da árvore. Este tipo de armazenamento de Octrees é conhecido na literatura como Octree Linearizada (ou Linear), pois, da mesma forma como a proposição anterior, a Octree é armazenada através de uma lista linearizada de valores.

Um exemplo do uso de códigos de localização (aplicados sobre a Octree da figura III.5) pode ser visto na figura III.6 .



OCTREE LINEARIZADA =

{0, 2, 4, 50, 51, 52, 54, 55, 56, 57, 6}

FIGURA III.6 – Exemplo de Octree Linearizada

III.2.4.4 – Acompanhamento de Raios em sólidos OCTREE

A grande dificuldade imposta a aplicação direta do algoritmo de acompanhamento de raios sobre um sólido descrito no modelo Octree refere–se a aplicação do modelo de iluminação. O sólido Octree nada mais é que uma coleção de cubos distribuídos pelo espaço, e a fronteira deste sólido é descrita através das faces destes cubos. Com isso, o número de normais diferentes é bastante reduzido (somente 6), o que faz com que a aplicação de técnicas de sombreamento não produza bons resultados.

A alternativa para a utilização de Octrees está ligada a criação de novos tipos de nós, de forma a poder representar de alguma forma a superfície verdadeira do sólido. Estas estruturas, mescladas entre Octrees e outros tipos de informações, são conhecidas como estruturas híbridas, serão de grande utilidade nas diversas propostas de otimização do algoritmo, que serão vistas no próximo capítulo.

Capítulo IV

Acompanhamento de Raios Otimizado

IV.1 – INTRODUÇÃO

O algoritmo de Acompanhamento de Raios (como proposto por WHITTED) produz imagens com alto grau de realismo, incluindo efeitos como reflexões, refrações e sombras. Entretanto, as primeiras implementações realizadas mostraram que o tempo computacional para a produção de uma imagem por este método era extremamente elevado.

Neste capítulo iremos discutir as diversas propostas de otimização do algoritmo, focalizando em técnicas de subdivisão espacial. Inicialmente será apresentada a motivação para a otimização do método. A seguir, as diversas propostas de otimização serão apresentadas.

IV.2 – MOTIVAÇÃO

O tempo de execução do algoritmo de acompanhamento de raios pode se tornar extremamente grande. Este fato decorre diretamente da natureza do algoritmo, que produz imagens através de uma amostragem exaustiva da cena (método de força bruta).

Para ilustrar o tempo que uma cena seria gerada, vamos analisar a geração de uma determinada imagem. Sejam os seguintes parâmetros:

- **Tempo médio de interseção** (referente ao cálculo de interseção do raio com as primitivas da tela) = 0.33 ms (Obs: Este tempo foi obtido experimentalmente em SUN SparcStation 1+)
- **Número de objetos pertencentes à cena** : 500
- **Profundidade máxima da árvore de raios refletidos e refratados (ARR)**: 3

A tabela IV.I mostra o tempo estimado (somente para cálculo de interseções) para imagens com diversos tamanhos. O melhor caso acontece quando somente um raio é acompanhado por pixel, enquanto que o pior caso acontece quando a ARR é uma árvore completa (neste caso 8 interseções por pixel são computadas).

	Número Pontos	Melhor Caso	Pior Caso
100 x 100	10000	27min 30s	3h 40 min
200 x 200	40000	1h 40 min	14h 40min
300 x 300	90000	4h 7min 30s	1 dia 9 horas
400 x 400	160000	7h 20min	2 dias 10h 40min

TABELA IV.1 – Estimativas de Tempo de Execução

WHITTED notou que a parte crucial do algoritmo eram realmente os testes de interseção. Para cenas simples (isto é, com poucas primitivas) seus testes indicaram que 75% do tempo do algoritmo era destinado a cálculos de interseção. Em cenas complexas, os testes indicaram que até 95% do tempo era gasto com cálculos de interseção.

A constatação que o cálculo de interseção era o ponto chave no projeto de algoritmos de acompanhamento de raios fez com que diversas propostas surgissem. KAPLAN (1987) enumerou uma série de critérios que estas propostas deveriam atingir:

• O tempo de computação do algoritmo deve ser independente da complexidade da cena (isto é, o número de objetos ou a complexidade de cada objeto)

• O tempo de execução deve ser constante por raio (isto é, não depender da origem e direção do raio). Esta propriedade assegura que o tempo total de computação do algoritmo depende essencialmente da resolução da imagem e do modelo de iluminação aplicado

• O tempo de execução do algoritmo deve ser razoável (isto é, não passar de um limite máximo de tempo)

• O algoritmo deve aceitar cenas especificadas de forma simples (isto é, sem nenhuma agregação ou relacionamento hierárquico entre os componentes da mesma)

• O algoritmo deve lidar com uma grande gama de primitivas geométricas, e ser facilmente estensível para comportar novas primitivas

• A utilização de coerência não deve reduzir a possibilidade de se aplicar uma implementação com algoritmos paralelos.

IV.3 – TÉCNICAS DE OTIMIZAÇÃO

Existem duas diferentes abordagens para a otimização do algoritmo de acompanhamento de raios:

• Otimização do custo do cálculo de cada interseção

• Redução do número de cálculos de interseções

A primeira abordagem focaliza o estudo de abordagens matemáticas, que envolvem cálculos de raízes de equações polinomiais, sistemas de equações, etc.

A segunda abordagem possui uma natureza mais algorítmica, e seu intuito é a criação de critérios que reduzam o número de objetos a serem testados contra os raios incidentes (evitando o fato de testar todos os objetos contra cada raio incidente gerado). A ênfase deste trabalho está no estudo das técnicas pertencentes à segunda abordagem.

Existem diversas propostas para a redução do número de interseções. Nesta seção iremos discutir três propostas: o uso de volumes limitantes ("Bounding Box"), o uso de técnicas de coerência e o uso de subdivisão espacial.

IV.3.1. – O uso de volumes limitantes

A idéia de usar volumes limitantes é bastante simples. Ao invés de realizar o cálculo de interseção imediatamente contra cada objeto, é feito um teste preliminar contra um volume envolvente ao objeto. Caso seja detectado a existência de interseção, outro cálculo seria feito, desta vez o cálculo tradicional entre o objeto e o raio (figura IV.1).

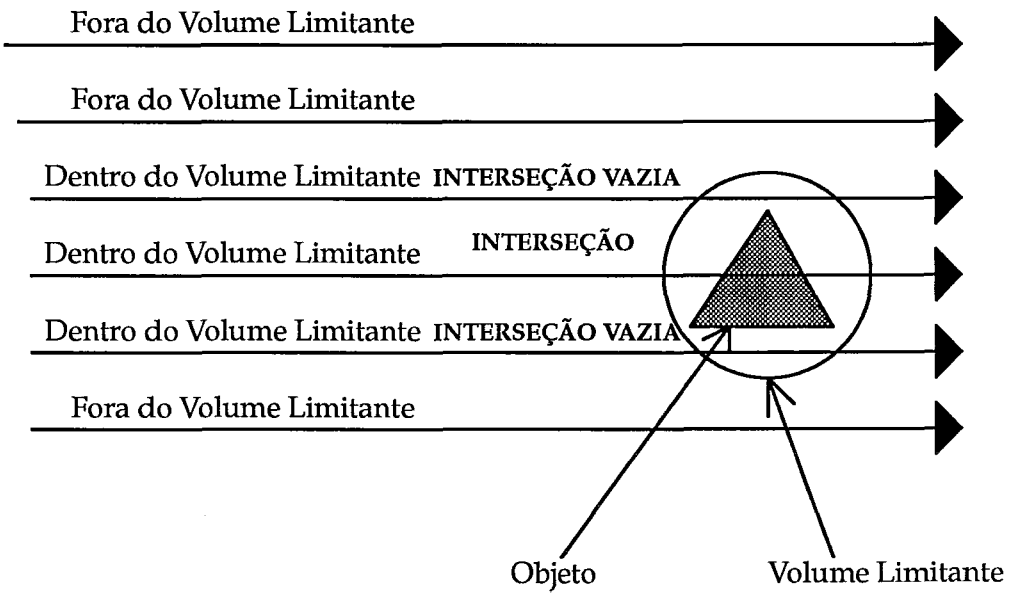


FIGURA IV.1 – A idéia de Volumes Limitantes

A vantagem de usar tais volumes limitantes estaria na redução dos cálculos tradicionais, pois em casos onde não existe interseção estes cálculos não seriam necessários. Para que esta redução no número de cálculos primários seja relevante no tempo total de execução do algoritmo é necessário que o volume limitante escolhido possua as seguintes características:

- O cálculo de interseção entre o raio e o volume limitante deve ser substancialmente mais rápido que o cálculo de interseção entre o raio e o objeto

- A área (no caso de objetos bidimensionais) ou o volume (no caso de objetos tridimensionais) obtida(o) pela diferença entre o volume limitante e o objeto deve ser reduzida(o).

WEGHORST (1984) preocupado com a escolha correta de volumes limitantes para as diferentes formas geométricas propôs a seguinte função de avaliação de volumes limitantes:

$$T = b \cdot B + i \cdot L \qquad \textbf{(Equação IV.1)}$$

onde:

T : o custo total da função

b : o número de vezes que o volume limitante é testado para interseção

B : o custo do cálculo de interseção do volume limitante contra o raio

i : o número de vezes que o objeto é testado para interseção

L: o custo do cálculo de interseção do objeto contra o raio

Um volume limitante adequado para um determinado objeto seria aquele que minimizasse essa função de avaliação. WEGHORST mostrou que os produtos desta função de avaliação são (na maioria das vezes) interdependentes, isto é, a redução no valor de **B** (usar um volume limitante mais simples) leva a um aumento em **i**.

A esfera é, sem dúvida, o volume limitante mais escolhido. Este fato deve-se a simplicidade no cálculo de interseção de raios contra esferas (que pode ser simplesmente implementado com um teste simples), além de sua geometria (que permite um razoável envolvimento de muitas figuras geométricas).

Outro volume limitante muito usado consiste da interseção limitada de semi-espaços planos. Desta forma, o volume limitante seria representado por uma coleção de semi-espaços planos, e o teste de interseção seria proporcional à resolução de um sistema de equações. Esta alternativa, entretanto, deve ser evitada, caso o número de semi-espaços planos utilizados seja muito grande (pois o somatório dos diversos cálculos de equações do primeiro grau pode ser maior que o cálculo contra a primitiva propriamente dita).

IV.3.2 – O uso de Coerência

IV.3.2.1 – Coerência da Imagem

A idéia de usar a coerência da imagem reside no seguinte fato: a intensidade de dois pixels vizinhos tem maior probabilidade de ser semelhante que diferente. Logo, talvez não seja necessário jogar raios para todos pixels da imagem.

Desta forma, somente alguns raios são acompanhados. Os raios não acompanhados têm a cor estimada pelos raios que foram acompanhados na vizinhança (figura IV.2). O

número de raios acompanhados é reduzido usando este tipo de abordagem, entretanto erros podem ser introduzidos durante a geração da imagem, principalmente em regiões de grande variedade cromática da imagem original, onde detalhes muito pequenos podem não ser representados.

O número de raios acompanhados usando este tipo de abordagem pode ser definido a priori (sem levar em consideração propriedades da imagens). Neste caso, a malha que define quais raios devem ser acompanhados é uma malha regular (embora não tão fina quanto o número de pixels que esta malha pode representar).

Outra abordagem não estabelece a priori quantos raios devem ser acompanhados, mas tenta estimar a necessidade ou não de uma subdivisão fazendo uma análise das discrepâncias cromáticas encontradas na vizinhança. A malha é subdividida como se fosse uma Quadtree, e subdividida recursivamente caso as cores dos pontos que definem o quadrante estiverem constrastantes (este critério de discrepância deve ser definido a priori – o que se faz é criar uma métrica sobre o espaço de cores, definir uma função de distância e uma distância mínima). Esta subdivisão torna–se adaptativa, isto é, leva em consideração as características da imagem como critério de subdivisão, o que também não impede o acontecimento de casos patológicos (detalhes muito pequenos inseridos em uma região de cor constante) Ver COMBA (1990a).

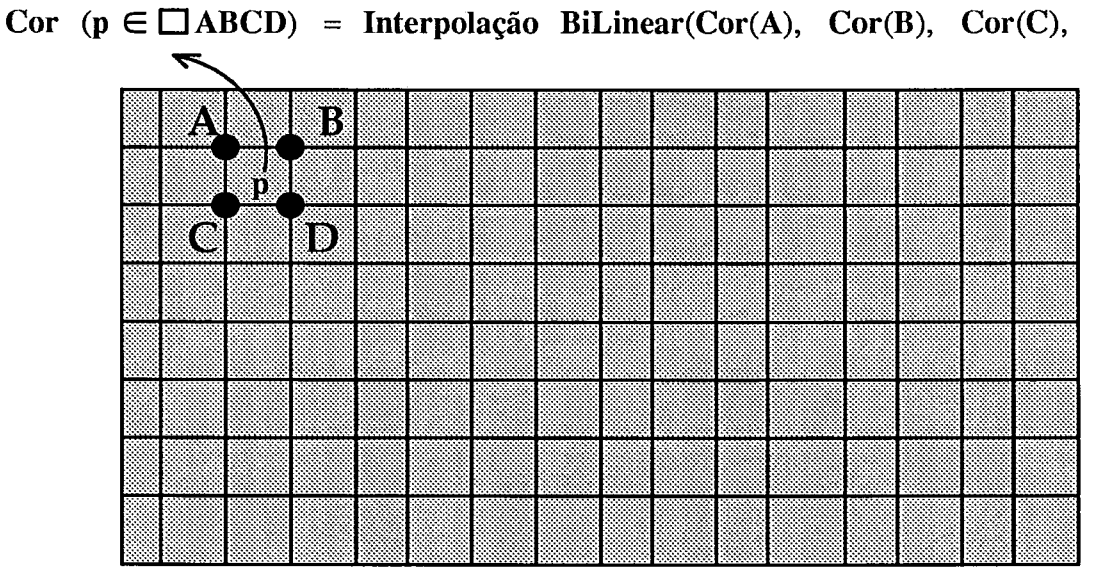


FIGURA IV.2 – O uso de coerência da imagem

IV.3.2.2 – Coerência de Objetos

A idéia de coerência entre objetos é usar o relacionamento entre estes de forma a reduzir o número de interseções necessários. O trabalho de COQUILLARD (1985) é um típico exemplo deste tipo de abordagem. Ele fez um pré–processamento da cena analisando as projeções dos volumes limitantes sobre a malha de projeção, de forma a montar dois tipos de grafos: o grafo de adjacência e o grafo de sobreposição.

O grafo de adjacência exprime as relações de adjacência existentes entre os objetos. O grafo de sobreposição relaciona quais objetos possuem volumes limitantes fazendo interseção. (Figura IV.3).

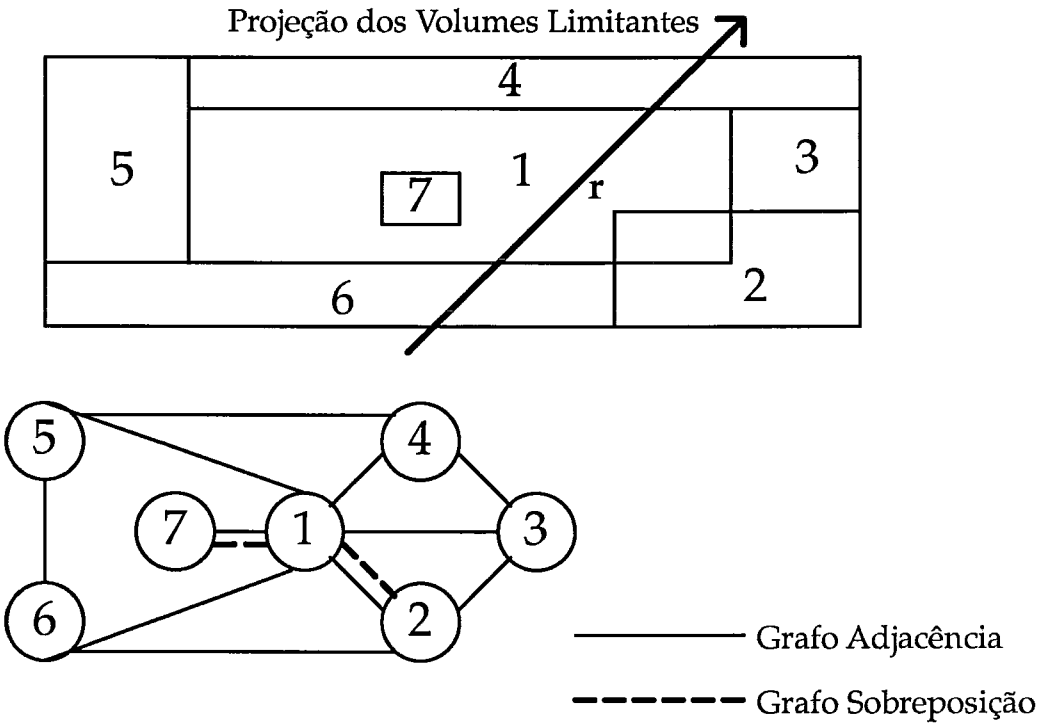


FIGURA IV.3 – O uso de coerência de objetos

O uso destes grafos é feito de maneira bem simples. Primeiramente, é descoberto o volume limitante onde o raio primeiramente entra na cena (no caso acima refere–se a primitiva número 6). O teste de interseção é feito contra a primitiva 6 e todas aquelas cujo volume limitante fizer sobreposição ao volume limitante dela (no caso nenhuma). Caso o teste de interseção for vazio, o próximo volume limitante no caminho do raio será um dos volumes adjacentes ao volume 6. Analisando o grafo de adjacência vemos que o volume 6 é adjacente aos volumes 1, 2, 5. Feito o teste, o próximo volume é o número 1. Neste caso as primitivas 1 e 7 (pois está em sobreposição em relação a 1) serão testadas para interseção. Este processo

continua até que alguma primitiva faça interseção com o raio, ou nenhum volume limitante esteja mais no caminho do raio.

IV.3.2.3 – Coerência de Raios

Uma outra abordagem recorre ao uso de coerência de raios. Em muitas cenas, grupos de raios seguem virtualmente o mesmo caminho. Desta forma, raios vizinhos (ou muito próximos têm essencialmente a mesma lista de objetos de interseção).

Existem três trabalhos muito conhecidos neste tipo de abordagem: o trabalho de AMANATIDES (1984), o trabalho de HECKBERT e HANRAHAN (1984) e o trabalho de SPEER (1985).

AMANATIDES propôs que a amostragem realizada pelo algoritmo de acompanhamento de raios fosse feita com um cone, ao invés de um raio infinito ("CONE TRACING"). Assim sendo, são calculadas as interseções entre o cone e as primitivas componentes da cena (que devem ser restritas à primitivas cujo teste de interseção contra um cone seja de rápido cômputo – como a esfera e polígonos, por exemplo). Caso não haja interseção, os pixels referentes à seção do cone com o plano de projeção são gerados com a cor de fundo. Caso contrário, a cor dos pontos é gerada a partir da seção obtida entre o cone e a primitiva encontrada.

HECKBERT e HANRAHAM possuem uma abordagem muito próxima a de AMANATIDES. Ao invés de acompanhar raios, eles propuseram o acompanhamento de cones piramidais ("BEAM TRACING") sobre cenas constituídas de objetos poligonais.

O algoritmo é proposto de forma recursiva. Inicialmente os "BEAMS" são acompanhados em direção à cena. As interseções entre os "BEAMS" e os polígonos são feitas através do uso de operações de conjuntos sobre polígonos bidimensionais, e o polígono mais próximo é obtido atravéz de uma busca em uma lista de polígonos classificada por profundidade (algorimo do pintor para remoção de linhas ocultas). Para cada interseção obtida, novos "BEAMS" são gerados para o acompanhamento de raios refletidos e refratados.

SPEER (1985) propôs usar o caminho gerado pelo último raio para prever o caminho do raio corrente. A lista de interseção entre o raio anterior e a cena é mantida. Esta lista de interseção conteria os últimos objetos testados para interseção, e esta informação seria usada para reduzir o número de objetos testados para interseção do raio atual.

IV.3.3 – Subdivisão Espacial

A subdivisão espacial é uma das estratégias mais utilizadas no projeto de algoritmos de acompanhamento de raios. A natureza do algoritmo (que consiste no acompanhamento de raios de direções arbitrárias pela cena) não permite que sejam determinados com facilidade quais objetos encontram–se no caminho dos raios. Os algoritmos canônicos fazem um teste exaustivo sobre todos os objetos pertencentes à cena, de forma a descobrir aqueles que estão no caminho dos raios (sendo este teste amenizado pelo uso de volumes limitantes).

A idéia de subdivisão espacial é dividir o espaço em regiões, e associar a estas uma lista de objetos que pertençam a mesma. Desta forma, para enumerar os objetos que podem fazer interseção com o raio é necessário descobrir inicialmente quais as regiões que fazem interseção com o raio, e, posteriormente, calcular interseções somente dos objetos que pertençam a tais regiões.

Assim, o número de objetos testados para interseção tende a diminuir. Esta dimuição irá depender de como a subdivisão aplicada conseguir particionar os objetos no espaço (de forma que cada região do espaço tenha o menor número possível de objetos).

A aplicação de subdivisão espacial pode ser separado em duas etapas:

- **Pré–Processamento:** Construção de uma estrutura de dados que permita que os testes de interseção de raios arbitrários com a cena sejam computados de forma eficiente. Esta estrutura de dados divide o espaço em uma estrutura hierárquica de cubos (caixas) com faces alinhadas aos eixos coordenados da cena.

- **Acompanhamento de Raios:** Os raios agora são acompanhados através de um espaço subdivido em regiões (e não livremente como anteriormente). As seguintes tarefas deverão ser resolvidas durantes esta etapa:

  1) Calcular a primeira região onde o raio passa: Inicialmente é determinado o ponto onde o raio entra na subdivisão espacial. Obtido este ponto, é necessário determinar a qual caixa este ponto pertence (usando um algoritmo de localização de pontos). Este procedimento irá depender do tipo de subdivisão adotado.

  2) Obter a lista dos objetos pertencentes à caixa: Dado uma determinada caixa, obter os objetos da cena que devem ser testados para interseção.

  3) Calcular a próxima caixa no caminho do raio: Caso ainda não seja verificada a interseção do raio com nenhum objeto pertencente à caixa, deve–se analisar a próxima caixa no caminho do raio.

Existem vários tipos de subdivisão utilizados, mas a subdivisão do espaço da mesma forma que o modelo Octree (explicado no capítulo II) tem sido a mais utilizada. GLASSNER

(1984), KUNII e WYVILL (1985), FUJIMOTO (1986) e SAMET [1989] usam Octrees com tipos de nós especiais como tipo de subdivisão espacial. Todas estas abordagens geram na fase de pré–processamento uma estrutura Octree que contém informações sobre os objetos que ocupam as respectivas regiões do espaço.

Os raios são acompanhados em seu trajeto pela Octree, determinando sucessivamente as caixas (no caso voxels da Octree) por onde o raio passa. Este processo é terminado quando um objeto interferindo a um voxel faz interseção com o raio, ou quando este deixa o cubo universo.

A seguir iremos ver com detalhes estas 4 propostas citadas acima, referente ao uso de Octrees para acelerar o algoritmo de acompanhamento de raios.

IV.3.4 – Trabalhos usando Subdivisão Espacial

Nesta seção iremos estudar os trabalhos desenvolvidos por GLASSNER, WYVILL e KUNII, FUJIMOTO e SAMET. Para uma descrição completa de seus trabalhos, estes serão descritos nos seguintes ítens:

- Estrutura de dados para armazenar a Octree
- Critério para descobrir a direção do próximo vizinho
- Algoritmo de busca de vizinho

IV.3.4.1 – O trabalho de GLASSNER

IV.3.4.1.1 – Estrutura de dados para armazenar a Octree

GLASSNER usou o modelo Octree para subdividir o espaço, mantendo nas folhas uma lista de objetos pertencentes a porção do espaço que elas representam. Na geração da octree, o espaço é subdividido caso uma determinada célula possua um número de primitivas maior que o estabelecido anteriormente. Caso contrário, é associado à celula da Octree uma lista de objetos que fazem interseção com a mesma (que pode ser vazia caso nenhum objeto fizer interseção). A figura IV.4 mostra um exemplo de subdivisão até que cada célula contenha somente um único objeto.

GLASSNER usou uma pequena variante da representação de Octrees proposta por GARGANTINI. Como vimos anteriormente, GARGANTINI usava uma codificação linear

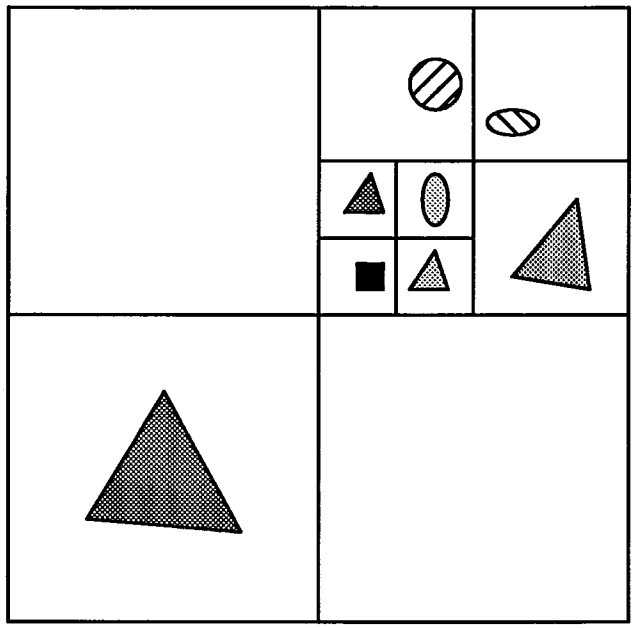


FIGURA IV.4 – Subdividindo até um objeto somente por célula

sobre os nós, sendo necessário somente o armazenamento dos nós cheios. O código gerado representava o caminho percorrido pela árvore desde a raiz até o nó, sendo este representado por um coleção de caracteres (constituindo um "string"). Desta forma, o nó representado pelo código {000} era diferenciado do código {00}. Se este código fosse implementado como um inteiro, esta codificação seria ambígua, pois o número 000 computacionalmente é o mesmo que o número 00.

GLASSNER argumentou que o armazenamento de "strings" não é tão eficiente e econômico quanto por inteiros. Para solucionar esta desvantagem, ele propôs que os octantes fossem numerados de 1 até 8, pois desta forma poderíamos armazenar os octantes por inteiros, além do que não poderíamos confundir o número 111 com o número 1.

O uso de códigos inteiros auxilia também na geração da estrutura de dados para armazenamento da Octree. Pelo fato de armazenar somente os nós cheios, GLASSNER optou pelo uso de uma tabela de hash, de forma a permitir economia no armazenamento dos dados e eficiência no acesso a estes. A figura IV.5 mostra a estrutura de dados proposta por GLASSNER.

IV.3.4.1.2 – Direção do próximo vizinho

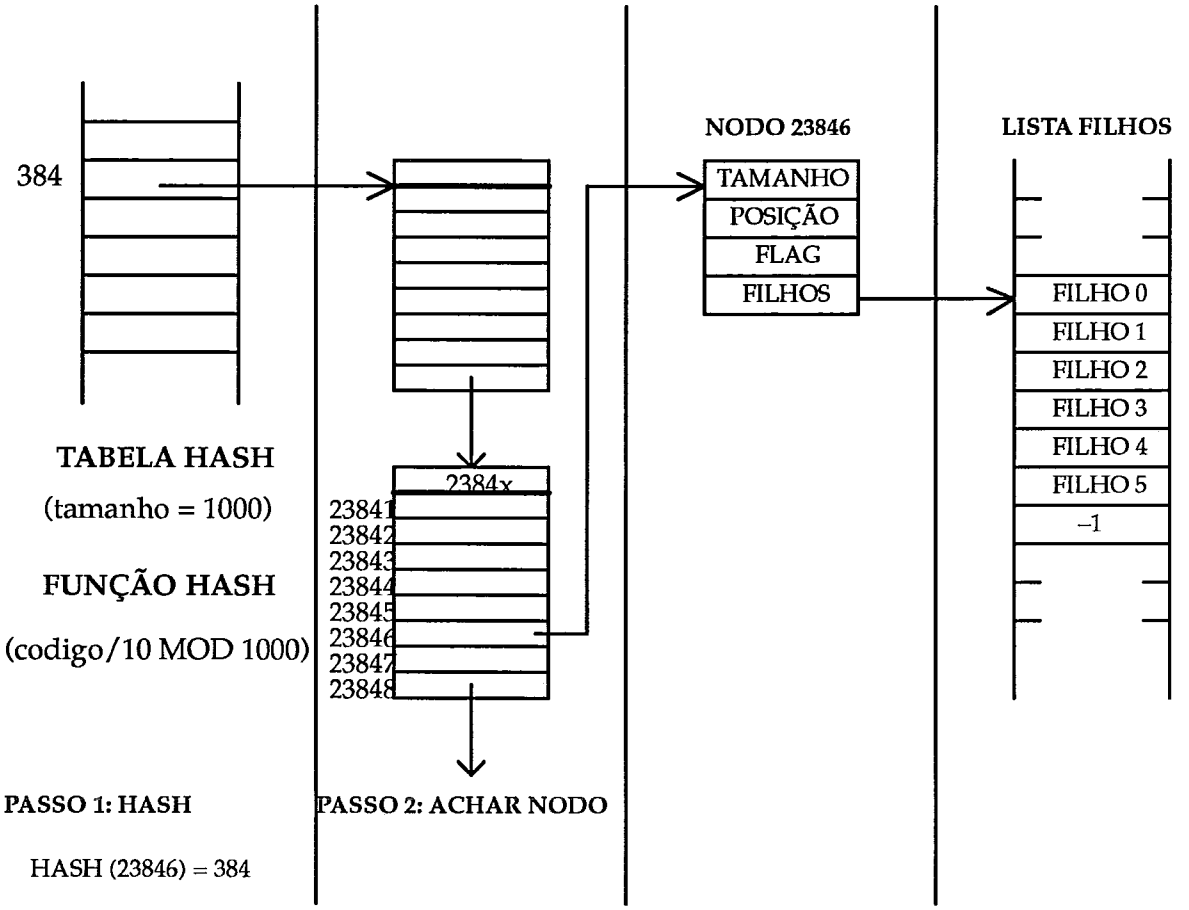


FIGURA IV.5 – Estrutura de dados de nós proposta por GLASSNER

É possível que o primeiro voxel da Octree por onde o raio passa tenha objetos que não fazem interseção com o raio. Logo, o acompanhamento do raio deve continuar nos voxels vizinhos. Nesse ponto, é preciso descobrir o ponto por onde o raio sai do voxel, para determinar a direção de saída do voxel (que permitirá determinar qual é o voxel vizinho).

GLASSNER propôs que se fizessem seis testes de interseção (um contra cada parede do voxel), e assim se determinasse qual o ponto de saída do voxel (figura IV.6).

IV.3.4.1.3 – Achando o voxel vizinho

Para procurar o próximo voxel no caminho do raio, GLASSNER procura inicialmente um ponto que seguramente esteja dentro do vizinho. Após encontrado este ponto, ele aplica um algoritmo de localização de pontos para descobrir a qual voxel este ponto pertence.

Existem três maneiras de um raio sair de um voxel: por um vértice, por uma aresta ou por uma face. Quando o raio sai somente por uma face do voxel, então o voxel vizinho é

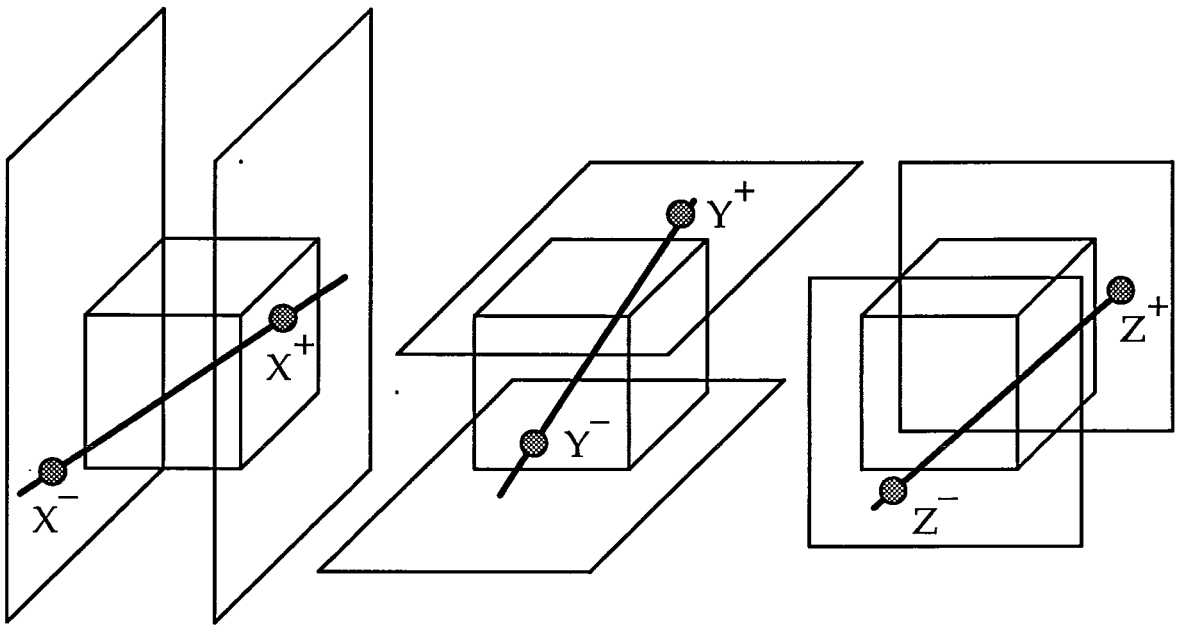


FIGURA IV.6 – Achando o ponto de saída do voxel

considerado um vizinho de face. Quando este raio sai por duas faces, então o voxel vizinho é chamado de vizinho de aresta. Quando este raio sai por três faces, então este vizinho é chamado vizinho de vértice.

A seguinte estratégia é usada para calcular o ponto no vizinho: faz–se uma translação perpendicular às faces de saída de um ponto de metade do menor lado da Octree. Desta forma, pode–se assegurar que o ponto estará dentro do vizinho. Esta estratégia é mostrada na figura IV.7.

IV.3.4.2– O trabalho de WYVILL e KUNII

IV.3.4.2.1 – Estrutura de dados para armazenar a Octree

WYVILL e KUNII restringiram–se a cenas formadas por sólidos descritos segundo o modelo CSG. A idéia básica era gerar uma estrutura híbrida entre Octree e CSG, chamada de RCSG ("Reduced CSG"), que subdividiria o espaço em uma árvore conforme uma Octree, e os nós folhas conteriam subárvores da árvore CSG da cena.

Desta forma, a complexidade da árvore CSG seria quebrada em uma série de subárvores (relativas a cada voxel da subdivisão). Os critérios de decisão do processo de subdivisão foram os seguintes:

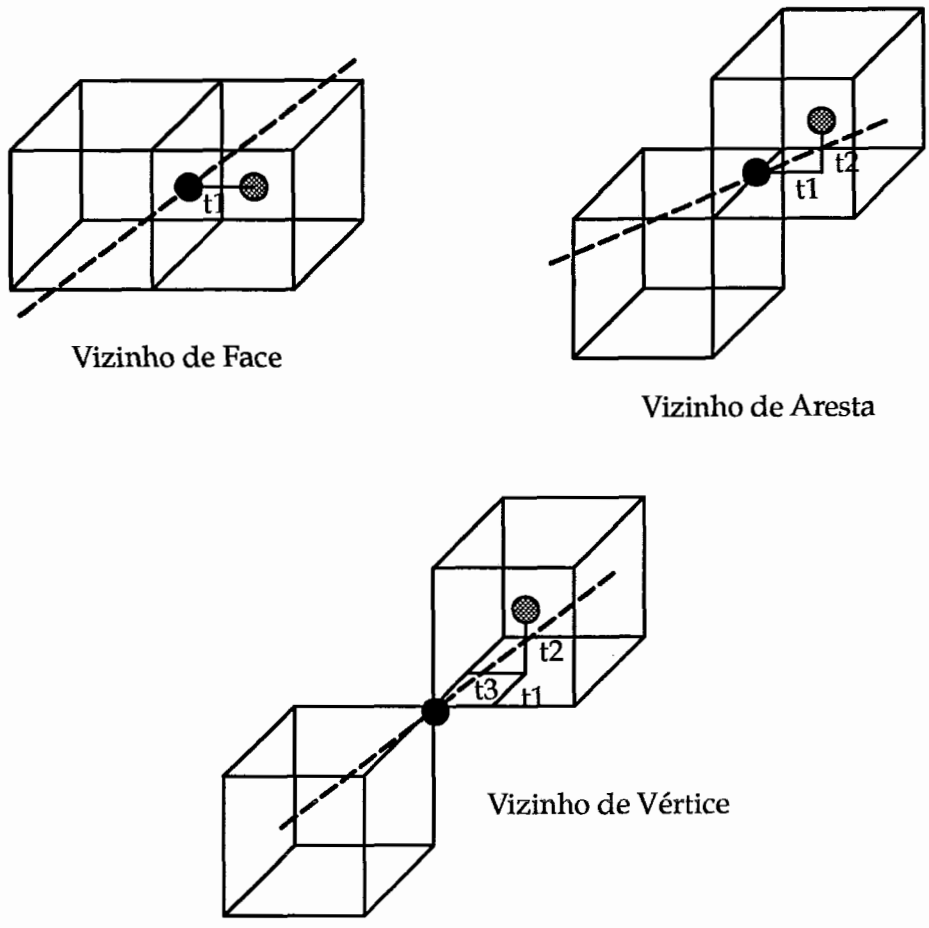


FIGURA IV.7 – Cálculo de um ponto dentro do vizinho

- Nenhuma primitiva da árvore CSG faz interseção com o voxel: **nó Vazio**
- Uma primitiva da árvore CSG engloba o voxel: **nó Cheio**
- Menos de um número determinado de primitivas fazem interseção: **nó folha RCSG**, que nada mais é do que uma simplificação da árvore CSG
- Mais de um número determinado de primitivas fazem interseção: **nó Misto**, que resulta no processamento de oito filhos.

A estrutura híbrida gerada é mostrada na figura IV.8.

O processo chave na geração desta estrutura consiste em descobrir qual a árvore simplificada CSG que reside em um determinado voxel. Este processo, chamado de **Localização**, consiste em duas etapas:

- **Classificação**: Etapa onde todas as primitivas da árvore CSG são testadas contra o voxel para verificação da interferência. Uma primitiva que não faz interferência com o

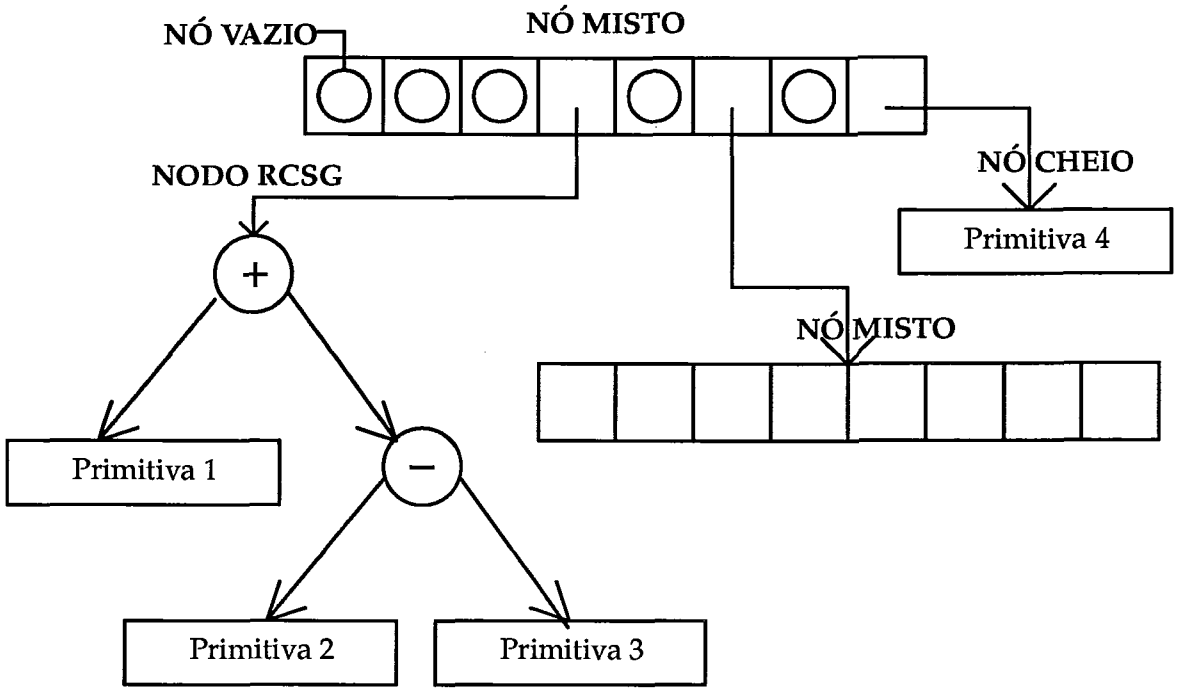


FIGURA IV.8 – Estrutura híbrida entre Octree e CSG

voxel é substituída na árvore por uma primitiva nula. Por outro lado, uma primitiva que engloba o voxel é substituída na árvore como uma primitiva universo. Caso contrário, a primitiva é mantida inalterada na árvore.

• **Simplificação**: Etapa onde a árvore já localizada é simplificada usando os conceitos de operações booleanas entre os conjuntos (tabela IV.2). Esta tabela será utilizada posteriormente quando da geração da OCSG.

IV.3.4.2.2 – Direção do próximo vizinho

Assim como GLASSNER, a determinação da direção do próximo vizinho é descoberta computando interseções com os planos que definem as paredes do voxel. Entretanto, o número de testes computados é reduzido de 6 para três, usando as características do coeficiente angular da reta.

O teste que GLASSNER executava anteriormente procurava obter os valores do parâmetro em relação aos eixos coordenados (x, y, e z), realizando seis testes de interseção. Para determinar o valor do parâmetro de saída somente três testes serão necessários. No caso da reta possuir coeficiente angular positivo nas três coordenadas, o valor do parâmetro será calculado como mostrado no algoritmo a seguir:

União	p2	W	∅
p1	p1 união p2	w	p1
W	w	w	w
∅	p2	w	∅

Interseção	p2	W	∅
p1	p1 inter p2	p1	∅
W	p2	w	∅
∅	∅	∅	∅

Diferença	p2	W	∅
p1	p1 dif p2	∅	p1
W	$p2^{-1}$	∅	w
∅	∅	∅	∅

W refere–se ao conjunto universo
∅ refere–se ao conjunto vazio
p1 e p2 são primitivas da árvore CSG

obs: $p2^{-1}$ é o complemento de p2 em relação ao universo

TABELA IV.2 – Operações Booleanas

```
PROCEDURE CalculaTdeSaída
    (
        Raio    : Reta ;        (* Equacao Parametrica do Raio *)
        Corner  : PontoR3 ;     (* Corner a sudoeste do voxel *)
        Lado    : REAL ;        (* Tamanho do lado *)
    ) :
        REAL ;                  (* Valor do parametro quando a reta sai do voxel *)

    BEGIN
        Menor := MAX (REAL) ;
        FOR i := 1 TO 3 DO
            IF Raio.DIrecao [i] > 0.0 THEN
                t [i] := ((Corner [i] + Lado) – Raio.Origem [i]) / Raio.DIrecao [i] ;
            ELSE
                t [i] := (Corner [i] – – Raio.Origem [i]) / Raio.DIrecao [i] ;
            END ;
            IF t [i] < Menor THEN
                Menor := t [i]
            END ;
        END ;
        RETURN Menor;
    END CalculaSaida ;
```

IV.3.4.2.3 – Achando o voxel vizinho

O procedimento para encontrar o voxel vizinho usado por WYVILL e KUNII é o mesmo proposto por GLASSNER: encontrar um ponto que esteja dentro do vizinho. Feito isso, um algoritmo de localização de pontos na Octree é utilizado.

IV.3.4.3 – O trabalho de FUJIMOTO et allii

IV.3.4.3.1 – Estrutura de dados para armazenar a Octree

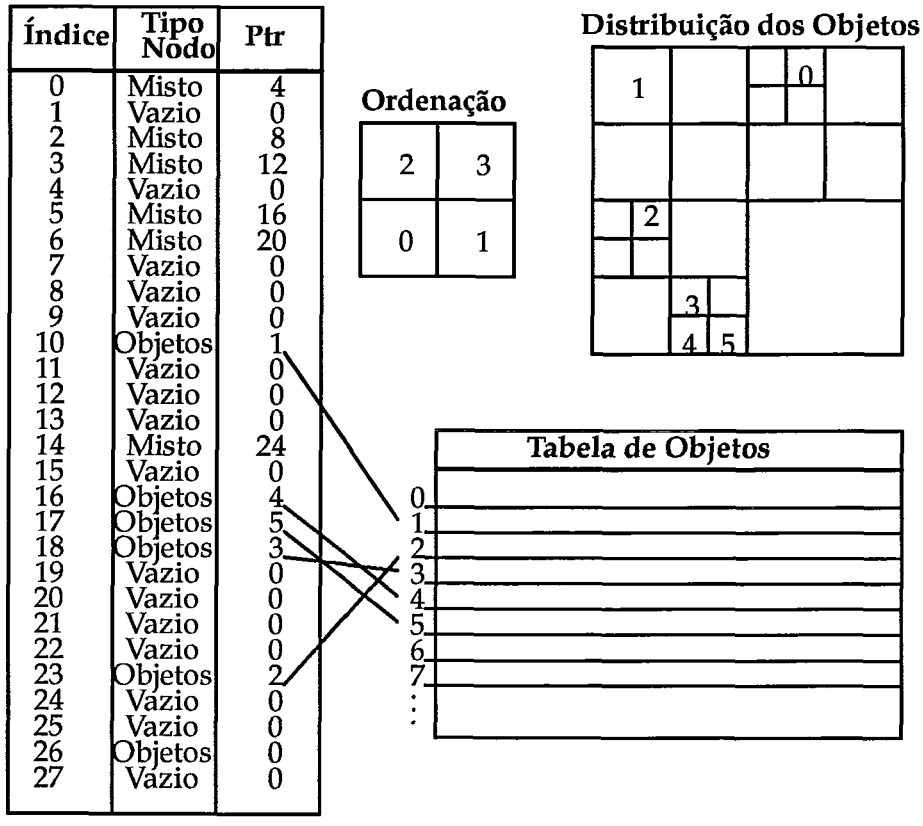
O trabalho de FUJIMOTO possui uma abordagem um tanto quanto diferente dos trabalhos anteriores. Para começar, ele trabalha com um caso particular de Octree, que consiste em uma árvore completa de uma determinada profundidade. A estrutura de dados que foi proposta para armazenar este tipo de Octree foi chamada de SEADS.

Esta estrutura foi projetada com objetivo de permitir o acesso mais rápido possível a um determinado nó, sem importar a quantidade de memória necessária para atingir este objetivo. Para tanto, ele usou um vetor de $8^p - 1$ elementos (onde **p** representa a profundidade da árvore). FUJIMOTO argumentou que a profundidade 5 produzia empiricamente bons resultados (sendo necessário um vetor de $8^5 - 1 = 32768$ elementos). A estrutura SEADS é mostrada na figura IV.9.

Para acessar este vetor é usada uma numeração similar a proposta por GARGANTINI. O código agora não é mais o caminho percorrido desde a raiz da árvore até o nó, mas um cardinal que identifica a colocação do nó em uma busca em pré–ordem. Este código identifica os nós pela ordem em que estes são visitados (variando desde 0 até 8^P–1).

IV.3.4.3.2 – Direção do próximo vizinho

FUJIMOTO criou este tipo de estrutura (que se constitui em um caso particular de Octree) de forma a criar um algoritmo de determinação de vizinho extremamente eficiente. A inovação de seu trabalho foi constatar que os voxels que eram percorridos sucessivamente pelo raio poderiam ser facilmente identificados por um algoritmo de traçado matricial de linhas (estendido para a terceira dimensão).



Obs: O campo Ptr é usado para apontar o objeto (no caso de um nodo objeto), apontar onde estão os filhos (no caso de nodo misto) e indefinido c.c.

FIGURA IV.9 – SEADS (Estrutura proposta por FUJIMOTO)

Os algoritmos de traçados de linha matricial mais conhecidos são o algoritmo DDA ("digital differencial analyser") e o algortimo de BRESENHAM (ver FOLEY (1990)). FUJIMOTO estudou o algoritmo DDA, de forma a modificá–lo a fim de enumerar os voxels no caminho do raio. A escolha por este tipo de algoritmo deve–se ao fato dele ser essencialmente incremental, além do que passível de implementação com aritmética inteira (é comum evitar operações que envolvam ponto flutuante, pois, devido a sua complexidade, são muito mais demoradas que as operações envolvento aritmética inteira).

O primeiro problema enfrentado por FUJIMOTO foi que nem todas as células no caminho do raio seriam geradas pelo DDA, pois o objetivo deste é obter uma boa representação matricial para uma linha (sem necessariamente passar por todas as células). Este problema é de fácil solução alterando o DDA, desmembrando o movimento em diagonal por dois movimentos (um horizontal e outro vertical, não necessariamente nesta ordem). Este problema é apresentado na figura IV.10.

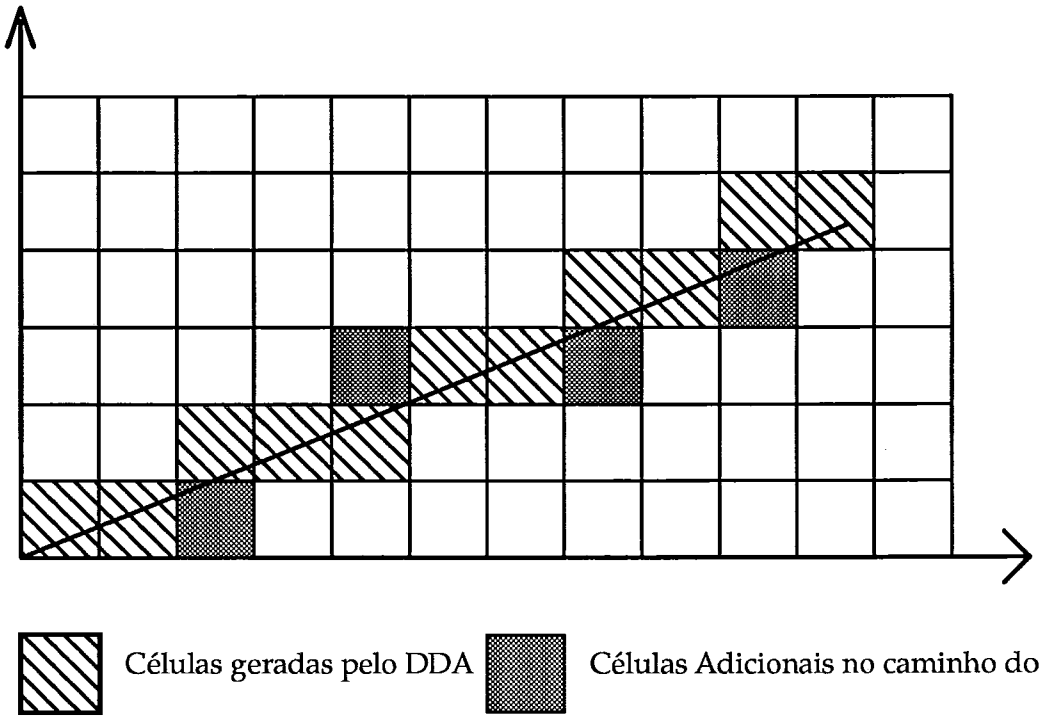


FIGURA IV.10 – Modificação no DDA

Um outro problema enfrentado por FUJIMOTO foi calcular as variáveis de controle do DDA. Como este algoritmo foi projetado para traçado de linhas, o início de cada linha sempre se dava no canto de uma célula. Assim, a maioria dos cálculos era simplificado. No caso de um raio em uma Octree este pode ter qualquer origem (não necessariamente sobre um dos cantos do voxel). Logo, este cálculo inicial foi alterado de forma a permitir retas com qualquer origem.

Uma vez estendido o algoritmo DDA para passar por todas as células no caminho do raio, FUJIMOTO trabalhou na extensão do algoritmo para a terceira dimensão. Ele notou que poderia fazê–lo trabalhando simultaneamente com dois DDA bi–dimensionais (referentes a projeção da linha sobre dois planos perpendiculares do espaço. A linha tri–dimensional seria representada pela projeção das linhas bi–dimensionais nestes planos (Figura IV.11).

IV.3.4.3.3 – Achando o voxel vizinho

Usando o DDA3D, FUJIMOTO pode obter com facilidade a direção do voxel vizinho (seria dado pelo movimento a ser executado pelo DDA). O seu problema agora seria descobrir em que posição da estrutura de dados este voxel poderia ser encontrado.

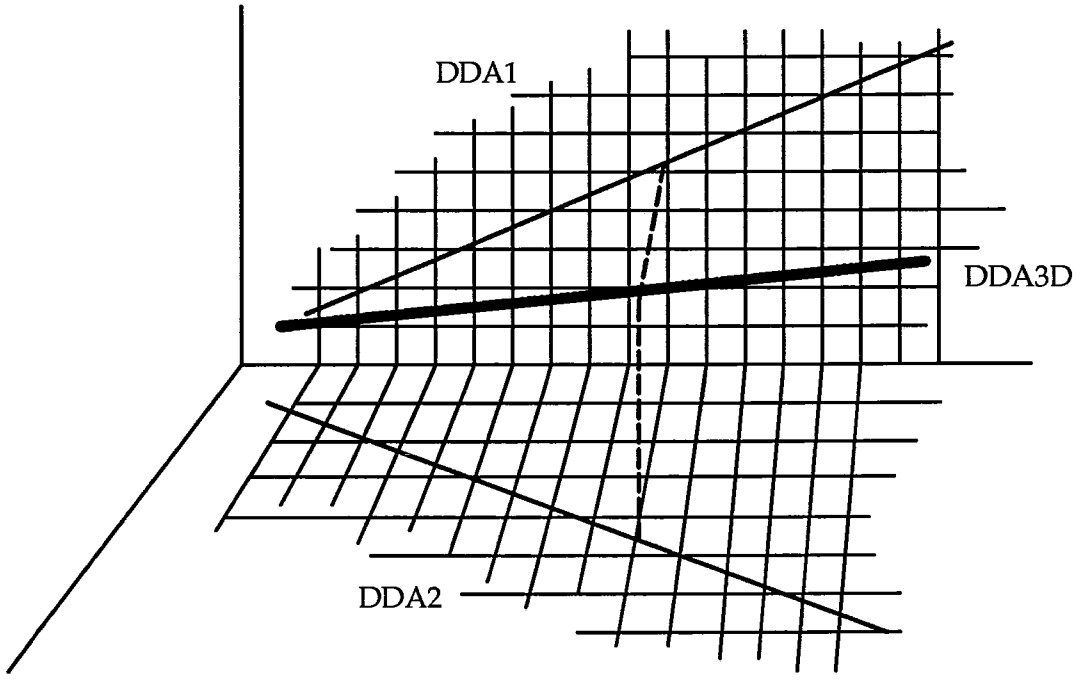


FIGURA IV.11 – DDA3D obtido a partir de dois DDA2D

Para que este acesso seja feito de forma otimizada, FUJIMOTO argumentou que a ordenação dos octantes seria de fundamental importância, pois iria permitir que o algoritmo de busca de vizinhos envolvesse operações simples. Ele adotou a ordenação sistemática proposta por YAMAGUCHI (ver capítulo II), que embute nesta ordenação informações de como o octante está localizado no espaço.

A ordenação de YAMAGUCHI pode ser representada também por três dígitos binários (e cada um destes dígitos representa um deslocamento no espaço em cada um dos eixos coordenados). Cada nó teria associado **p** números octais (onde **p** representaria a profundidade do nó). O cálculo de vizinhos pode ser feito de forma simples, manipulando estes números octais. A figura IV.12 mostra um exemplo de cálculo de vizinhos em uma Quadtree (para a Octree bastaria processar deslocamentos também em z).

Neste exemplo procuramos descobrir o vizinho a direita do nó 03. A representação binária do nó é montada, resultando nos códigos 00 e 11. Para realizar o movimento, deve–se somar um na coluna referente ao eixo X de baixo para cima (isto é, das folhas para a raiz da árvore). Caso a coluna esteja com o valor 1, este é substituído por 0 e ocorre um "vai 1" para a casa acima. Este processo acaba quando não houver mais propagações de 1's para cima.

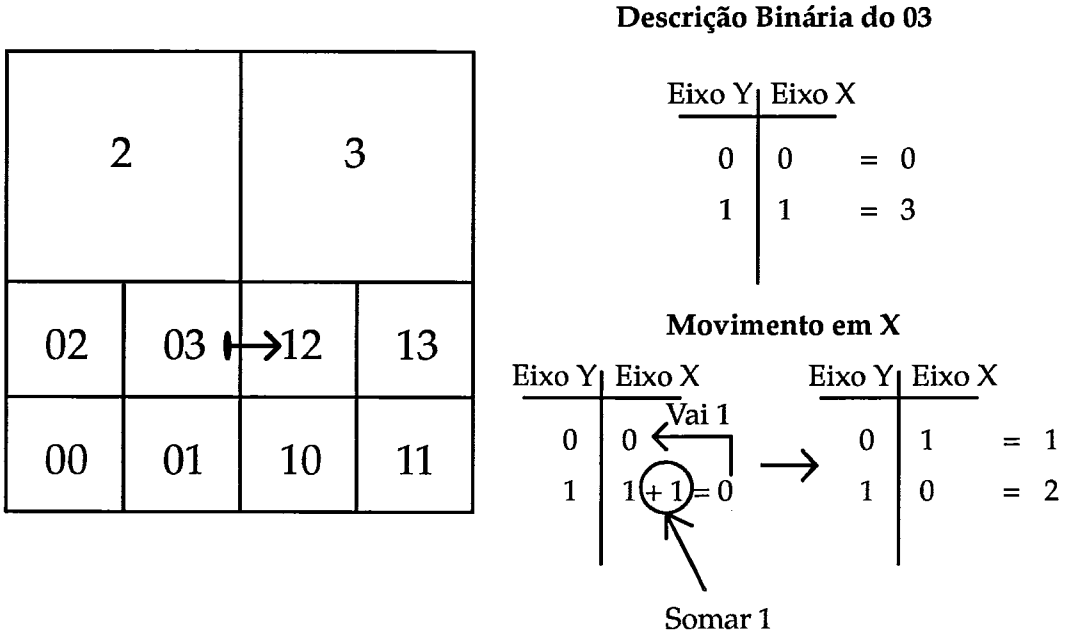


FIGURA IV.12 – Achar vizinhos usando códigos de localização

IV.3.4.4 – O trabalho de SAMET

IV.3.4.4.1 – Estrutura de dados para armazenar a Octree

O trabalho de SAMET lida com a estrutura de dados tradicional usando ponteiros para armazenar uma Octree. Para a implementação do algoritmo de busca de vizinhos desenvolvido pelo próprio SAMET, ele precisou incluir na Octree para cada nó filho um ponteiro para seu pai . Este fato deve–se a necessidade de percorrer a Octree no sentido das folhas para a raiz.

SAMET não entra em detalhes sobre os objetos que irão definir a cena. Seu trabalho é mais acadêmico, no intuito de sugerir como usar o modelo Octree para acelerar o método de Acompanhamento de Raios.

IV.3.4.4.2 – Direção do próximo vizinho

A direção do próximo vizinho é obtida fazendo três testes de interseção (contra as prováveis paredes por onde o raio pode sair do voxel). Este cálclulo é realizado de forma similar ao proposto por WYVILL e KUNII.

IV.3.4.4.3 – Achando o voxel vizinho

O método de busca de vizinho proposto por SAMET é diferente dos outros três propostos anteriormente. Os métodos usados por GLASSNER e WYVILL consistiam em achar um ponto que estivesse seguramente no vizinho, aplicando a seguir um algoritmo de localização de pontos. Nesta abordagem, o pior caso seria quando o nó vizinho estivesse no maior nível da árvore, sendo necessário percorrer **p** nós até chegar nele. No melhor caso, somente 1 acesso seria necessário.

Na abordagem de FUJIMOTO, o cálculo do vizinho é feito manipulando os códigos dos octantes desde a folha até a raiz (parando quando um nó possuísse uma componente zero). Nesa abordagem, o pior caso e o melhor caso são idênticos aos da abordagem anterior.

SAMET propôs que se buscasse o vizinho através de um percurso na árvore. Ele apenas estendeu o algoritmo que ele tinha proposto em 1982 para Quadtrees. A idéia básica deste método consiste em encontrar um ancestral comum entre o nó corrente e o nó vizinho, fazendo um percurso na árvore no sentido das folhas para a raiz. Uma vez encontrado, o caminho de subida é refletido sobre um determinado eixo, gerando um caminho de descida que leva até o nó vizinho. (Figura IV.13) .

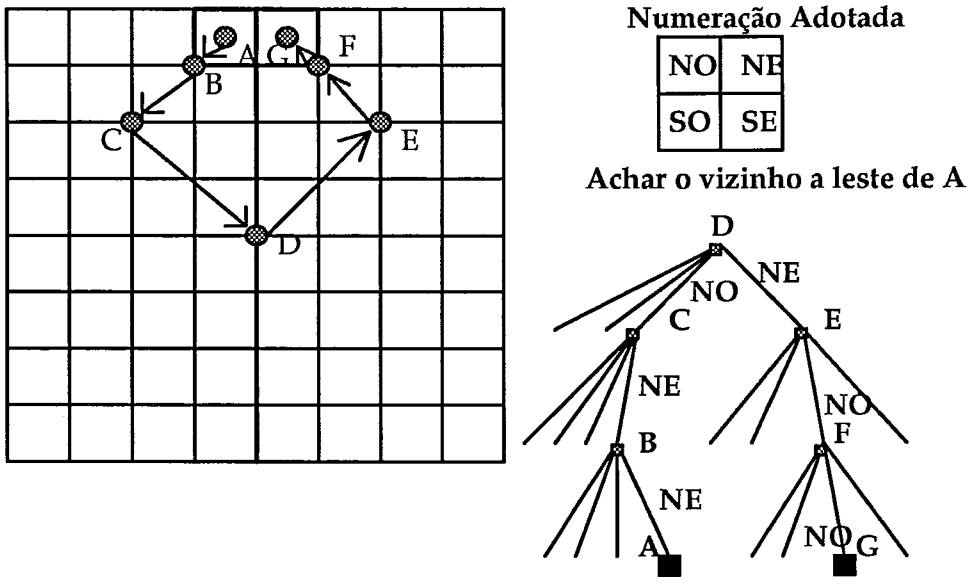


FIGURA IV.13 – Busca de vizinhos proposta por SAMET

A idéia básica deste algoritmo é ir subindo na árvore até encontrar um ancestral que possua um vizinho imediato na direção desejada. Neste caso, o vizinho imediato será aquele

cujo vizinho na direção desejada é um irmão seu (isto é, são filhos do mesmo pai). Uma vez encontrado (este nó é o ancentral mais próximo com pai comum), o caminho de subida é refletido na descida (nesse caso a reflexão do nó nordeste NE será o nó noroeste NO). O caminho de subida descrito acima foi NE, NE, NO. Logo o caminho de descida será a reflexão deste caminho, isto é NE, NO, NO.

Na figura IV.13 deseja–se descobrir o vizinho a leste do nó A. Como o nó A não possui vizinho imediato a direita, então seu ancestral é acessado (no caso B). Como B também não possui vizinho imediato a direita seu ancestral é achado (no caso C). O nó C finalmente possui vizinho imediato a direita, logo este deve ser acessado. O caminho de subida descrito acima foi NE, NE, NO. Logo o caminho de descida será a reflexão deste caminho, isto é NE, NO, NO. Assim passa–se pelos nós D (ancestral de C), E (Vizinho imediato de C), F (reflexão referente ao nó B), alcançando finalmente o nó G, o vizinho.

Em 1988, SAMET apresentou a extensão deste algoritmo para Octrees. A busca de vizinhos na Octree é um tanto quanto mais complexa do que em Quadtrees, mas SAMET simplificou o processo (bastando criar novas tabelas para representar os diversos tipos de adjacência entre os nós.

Capítulo V

Apresentação da Nova Proposta

V.1 – INTRODUÇÃO

Neste capítulo iremos apresentar uma nova proposta para otimização do algoritmo de acompanhamento de raios, baseada nas técnicas de subdivisão apresentadas no capítulo anterior. Inicialmente a estrutura de dados para armazenamento da Octree será apresentada, apontando os principais problemas inerentes à sua geração. Logo após, será apresentado um novo algoritmo para determinação dos voxels no caminho do raio (chamado de algoritmo de Progressão Celular).

V.2 – MOTIVAÇÃO

Após estudar as diversas propostas para otimização do algoritmo de acompanhamento de raios, procurou–se definir uma nova proposta que congregasse as vantagens de cada uma delas. O mais interessante das propostas estudas era o seguinte:

• **FUJIMOTO**: O algoritmo de determinação da direção do próximo vizinho era o ponto alto de sua proposta. Pelo fato de usar somente aritmética inteira e incremental, o novo vizinho é determinado através de somas e deslocamentos de bits.

• **WYVILL e KUNII**: Este trabalho, dentre os quatro apresentados, é o único que usa um modelo de sólidos conhecido para a modelagem de cenas (CSG). Esta preocupação revela–se importante, pois em um ambiente de síntese de imagens a utilização de um modelo adequado para a criação e representação de sólidos é de vital importância.

• **GLASSNER**: O trabalho de GLASSNER foi de grande importância, por ser o primeiro a usar a idéia de Octrees para acelerar o algoritmo de acompanhamento de raios. Entretanto, muitas de suas propostas foram melhoradas nos trabalhos seguintes.

• **SAMET**: O trabalho de SAMET discorre com mais detalhes sobre as minúcias da implementação. Para uma primeiro protótipo, o algoritmo de vizinhos proposto por ele é bastante indicado, por ser de fácil implementação.

O trabalho de FUJIMOTO é o que apresentou resultados mais expressivos. Mas, um preço foi pago para a dimuição no tempo de execução do algoritmo: um acréscimo no uso de memória. Como vimos no capítulo anterior, FUJIMOTO usava uma Octree completa, na qual todos os caminhos mínimos da raiz até uma folha possuem tamanho p (onde **p** representava a profundidade da Octree), sendo necessário um vetor de 8^p posições para armazená–la. Este fato está diretamente ligado ao uso do DDA3D para acompanhar sucessivamente os voxels no caminho do raio.

A partir da constatação deste forte interligamento entre a estrutura de dados e o algoritmo para determinação de vizinhos, estudou–se a modificação dos algoritmo de traçado de linhas para usá–los como forma de determinar os vizinhos em uma Octree não necessariamente completa. Verificada esta viabilidade, poderíamos lidar com Octrees não completas (reduzindo assim a quantidade de memória necessária), e usando um algoritmo incremental e inteiro para determinação do vizinho.

Para manter a coerência em relação aos trabalhos descritos no capítulo anterior, vamos apresentar a nova proposta da mesma forma como fizemos anteriormente, subdividindo a apresentação nos ítens estrutura de dados, determinação da direção do vizinho e busca do vizinho. Neste capítulo iremos apresentar as idéias principais da proposta, mostrando os problemas que devem ser resolvidos durante a implementação. No capítulo seguinte, iremos aprofundar a discussão sobre a geração da estrututa de dados. No capítulo VII iremos discutir com muito mais detalhes o algoritmo de determinação dos voxels no caminho do raio.

V.3 – ESTRUTURA DE DADOS PARA ARMAZENAR A OCTREE

Como realçado anteriomente, existe um grande interesse de criar uma nova proposta de aceleração do algoritmo de acompanhamento de raios que possa ser incluída num ambiente de síntese de imagens. Para tanto, é necessário que a cena seja modelada segundo um modelo de representação de sólidos conhecido.

No nosso ambiente de síntese de imagens, possuímos dois modeladores (para sólidos CSG(ESPERANÇA (1990b)) e para sólidos Octree (COMBA (1989))). Como o sólido descrito no modelo Octree não é adequado para a aplicação do algoritmo de acompanhamento de raios, a escolha mais óbvia recaiu sobre o modelo CSG.

A geração de uma estrutura híbrida entre Octree e CSG foi a alternativa escolhida para a representação da cena (usando, portanto, uma estrutura similar à proposta por WYVILL e KUNII). O processo envolvido na geração desta estrutura (como vimos anteriormente) chama–se **Localização**, que consiste nas seguintes etapas:

- **Classificação**: Processo que consiste em determinar se uma determinada primitiva da árvore CSG faz interferência parcial, total, ou nenhuma com um determinado voxel (Figura V.1). No caso do nosso modelador CSG, temos somente 4 tipos de primitivas : esferas, cilindros, paralelepípedos e semi–espaços), estando sujeitas cada uma destas a transformações geométricas como translação, rotação e escala. Devem existir rotinas de interseção que possam responder a estas consultas acima.

- **Simplificação**: Processo de geração das árvores simplificadas referentes a cada voxel. Para tanto, deve obter podas da árvore inicial e associar a cada voxel. Este processo, se feito de uma maneira desatenta, pode levar a cópias desnecessárias da árvore inicial. Deve–se, portanto, formular um algoritmo que crie as diversas árvores podadas com pouco desperdício de memória.

V.4 – DIREÇÃO DO PRÓXIMO VIZINHO

A grande inovação desta proposta reside no algoritmo de determinação das células por onde o raio passa em uma Octree. O problema pode ser colocado da seguinte forma:

PROBLEMA: Dado uma Octree e um raio qualquer no espaço que atravessa essa Octree, determinar quais os voxels da Octree que sucessivamente são atravessados pelo raio (ver figura V.2).

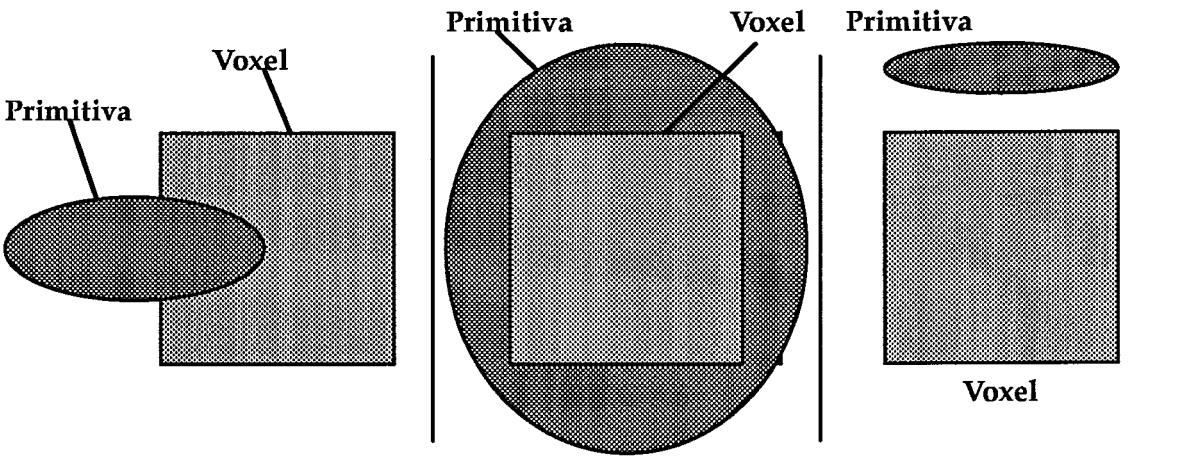


Figura V.1 – Tipos de interseção de uma primitiva e um voxel

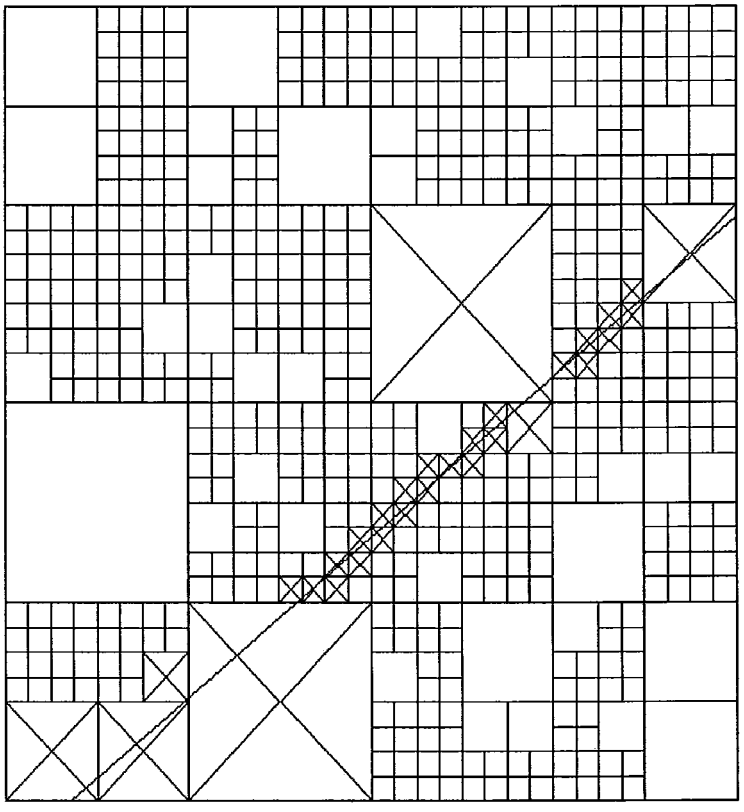


Figura V.2 – Descobrir os voxels no caminho do raio

A idéia de usar um algoritmo de traçado de matricial de linhas é uma boa alternativa se todos os nós da Octree tiverem o mesmo tamanho. Isso foi o que FUJIMOTO fez. Entretanto, caso a Octree seja genérica, o uso deste algoritmo deve ser alterado de alguma forma.

Imagine que temos um determinada resolução para um traçado comum de linhas. A representação matricial da reta será dada da melhor forma usando esta resolução. Caso essa resolução seja dobrada, a representação matricial da reta será mais precisa. A representação matricial da reta depende, portanto, essencialmente da resolução da matriz onde a reta será traçada (Figura V.3).

Imagine agora um dispositivo matricial hipotético que não possui um resolução

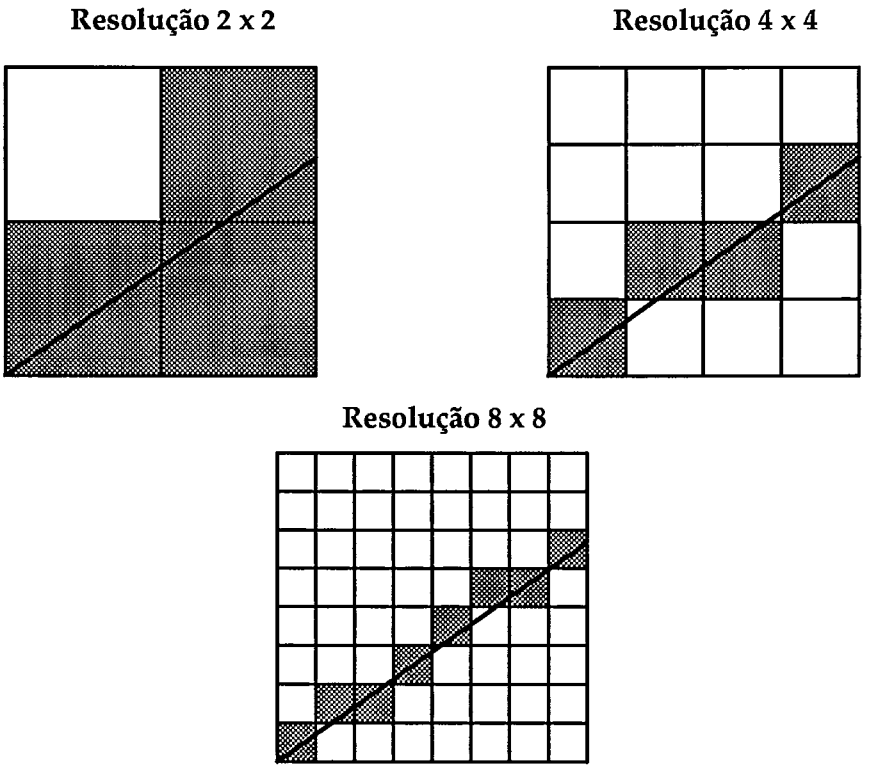


Figura V.3 – Diferentes resoluções, Diferentes Linhas

constante, mas diferentes resoluções em diferentes áreas do dispositivo. Para traçar uma reta neste dispositivo hipotético, deveríamos implementar um algoritmo que pudesse trocar de resolução a medida que as características de resolução do vídeo mudassem (Figura V.4).

Esta situação, embora incomum, retrata exatamente o problema que enfrentamos: dado uma Octree (ou uma Quadtree no plano), descobrir quais as células pelas quais o raio sucessivamente passa. Embora o algoritmo tradicional não necessariamente passe por todas as células, pode–se modificá–lo facilmente para atingir este objetivo.

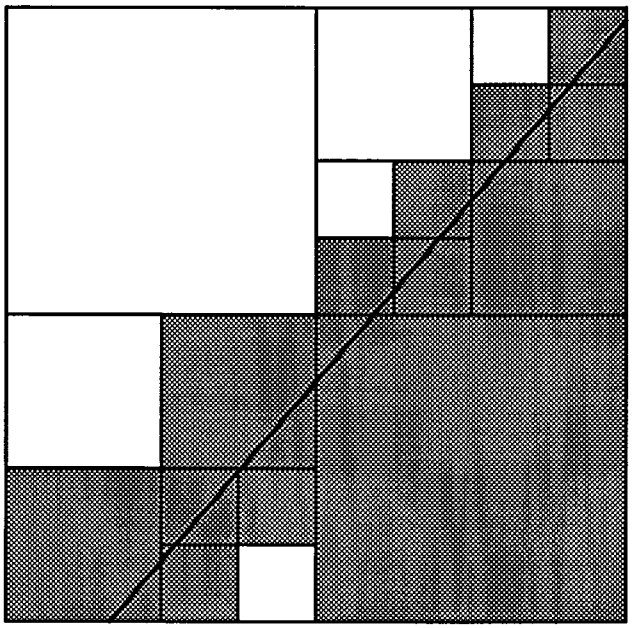


Figura V.4 – Traçado de linha num dispositivo de multi–resolução

A tarefa consiste em conseguir projetar um algoritmo de traçado de linha que consiga conviver com diferentes resoluções. Esta idéia é a base do algoritmo que intitulamos de algoritmo de Progressão Celular.

A apresentação formal deste algoritmo se dará no capítulo VII. Primeiramente será discutida a implementação deste algoritmo para o caso bi–dimensional, que consiste em acompanhar um raio por uma Quadtree. Após esta etapa, será discutida a extensão dele para a terceira dimensão, que consiste em acompanhar uma reta por uma Octree.

Capítulo VI

Geração da Estrutura Híbrida entre Octree e CSG

VI.1 – INTRODUÇÃO

Neste capítulo iremos discutir os detalhes referentes à implementação da estrutura híbrida entre Octree e CSG apresentada no capítulo anterior. Os processos de classificação e simplificação serão discutidos mais aprofundadamente.

VI.2 – A ESTRUTURA HÍBRIDA ENTRE OCTREE E CSG

VI.2.1 – A idéia da estrutura

A idéia de utilizar uma estrutura híbrida entre Octree e CSG foi proposta inicialmente por WYVILL e KUNII (como vimos no capítulo IV). A estrutura de dados proposta por eles era

similar à estrutura de ponteiros para o armazenamento de Octrees, com a diferença que existia um novo tipo de nodo, correspondente a uma simplificação da árvore CSG (ver figura IV.8).

Este tipo de nodo será referenciado como nodo CSGReduzido. A implementação deste tipo de nodo deve ser feita com muito cuidado, pois existe uma grande probabilidade de uma primitiva pertencer a diferentes simplificações (referentes a voxels diferentes), no caso dela interceptar mais de um voxel. Existe, portanto, uma grande possibilidade de haver informações redundantes nesta árvore.

Uma forma de amenizar o uso de memória causado por este problema seria associar a cada primitiva da árvore CSG um rótulo, que o identificasse. Assim, ao gerar um nodo CSGReduzido não seria necessário montar uma simplificação de uma árvore CSG (com informações detalhadas sobre as folhas), mas guardar o rótulo referente a cada primitiva contida no voxel. Para acessar as informações referentes a uma determinada primitiva, este rótulo seria utilizado como indexador de uma tabela de primitivas, que guardaria informações sobre todas as primitivas da árvore CSG.

Deve-se definir um critério para a escolha deste rótulo. O rótulo mais natural seria um número referente a ordem com que uma visita em pré-ordem passaria pela primitiva. Desta forma, uma árvore com 1000 primitivas teria rótulos variando entre 1 e 1000.

A figura VI.1 mostra um exemplo da estrutura de dados (chamada de Árvore OCSG).

VI.3 – LOCALIZAÇÃO DAS PRIMITIVAS CSG

VI.3.1 – Classificação

VI.3.1.1 – Fundamentos teóricos

O processo de classificação envolve a verificação do tipo de interferência existente entre a primitiva CSG e um voxel da Octree. Como vimos no capítulo anterior, três resultados podem ser obtidos do processo de classificação: interferência, exclusão e inclusão (ver figura V.1). A implementação da rotina de classificação irá depender do tipo de primitiva permitida na árvore CSG, de forma a poder apresentar um dos três resultados acima.

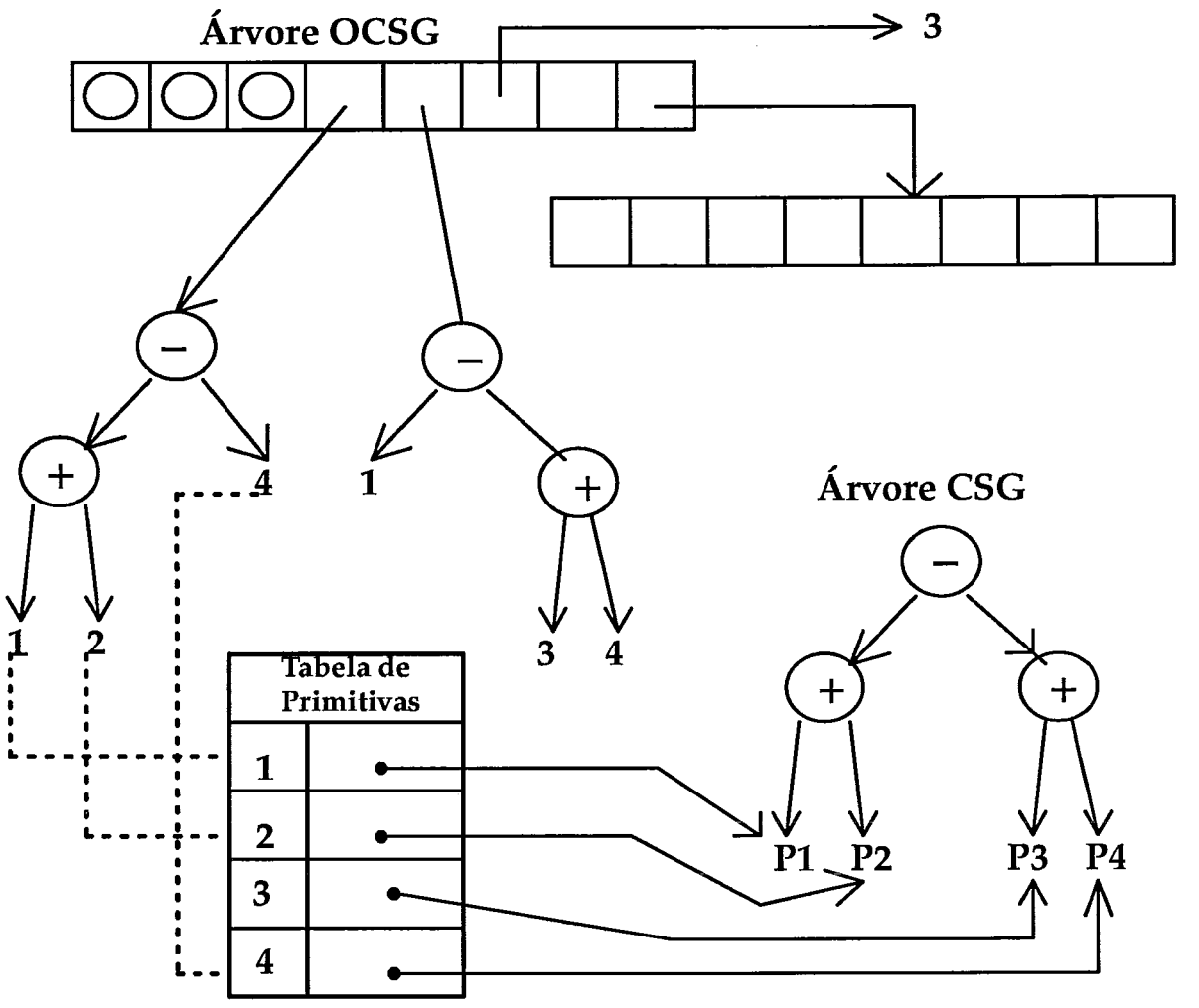


FIGURA VI.1 – Estrutura de dados para armazenar a OCSG

As primitivas disponíveis segundo o modelador CSG usado são: Bloco, Cilindro, Esfera e Semi-Espaço Plano. Inicialmente estas primitivas estão definidas no SCPrimitiva (como visto no capítulo III) de forma bastante conveniente, isto é, posicionadas de forma a facilitar determinados cálculos geométricos. Entretanto, o sistema não se limita a estas instâncias, mas permite que muitas outras possam ser especificadas.

Tais instâncias são obtidas através da aplicação de transformações afins inversíveis sobre as primitivas no SCPrimitiva, sendo levadas a um novo sistema de coordenadas chamado de SCCena (Sistema de Coordenadas da Cena). As transformações são estabelecidas quando da especificação do sólido em CSG (ver o capítulo III para maiores detalhes sobre as transformações possíveis).

O elenco de primitivas passíveis de representação aumenta com a aplicação destas transformações, pois além das primitivas tradicionais podem ser agora representados elipsóides, romboedros e cilindróides.

A rotina de classificação deve saber responder sobre o comportamento de todo este elenco de primitivas contra voxels da Octree. Serão necessários, portanto, rotinas que avaliem a interferência de elipsóides, romboedros e cilindróides contra cubos.

Antes de apresentar como fazê-lo, devemos primeiramente discutir um teste de interseção que deve ser feito ao implementar o algoritmo de acompanhamento de raios: verificar a interseção de um raio contra uma primitiva da árvore CSG. Este teste de interseção pode ser feito de maneira simples, segundo um artifício proposto por ROTH [82].

ROTH argumentou que este teste de interseção não precisaria ser feito no SCCena, onde rotinas de interseção de reta contra todo tipo de primitiva deveria ser implementado. ROTH mostrou que este teste poderia ser feito no SCPrimitiva, sendo necessário aplicar a transformação inversa ao raio, obtendo-se por linearidade um raio no SCPrimitiva. Este poderia ser testado contra as primitivas esfera, cilindro e bloco de maneira simples e bem conhecida. Esta solução simplificou sobremaneira a implementação do algoritmo sobre sólidos CSG (Ver figura VI.2).

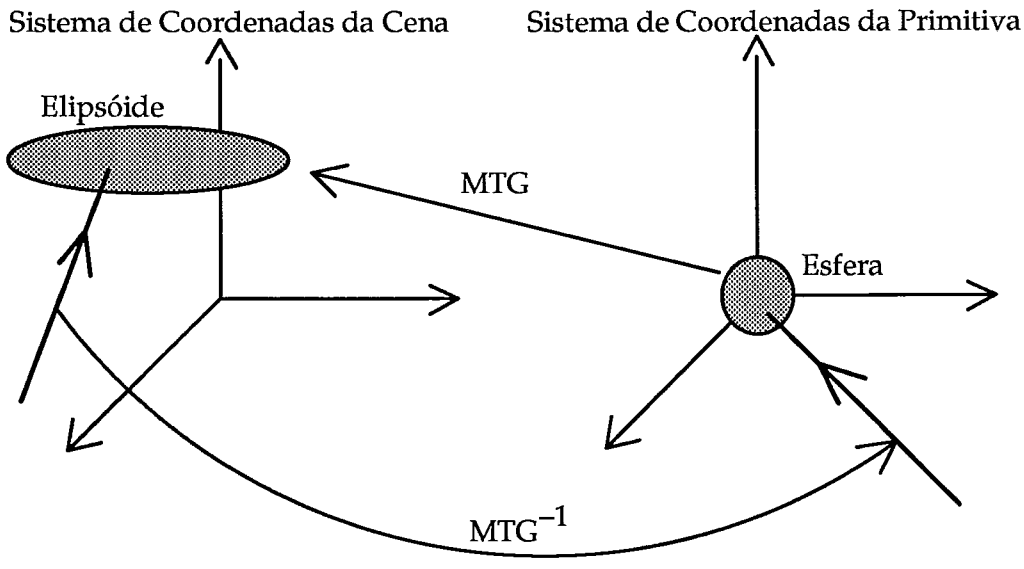


FIGURA VI.2 – Cálculo de interseções contra a primitiva

A idéia de ROTH utiliza de forma bastante inteligente as transformações geométricas de forma a simplificar o cálculo de interseções. É bastante razoável que apliquemos esta mesma idéia para efetuar o teste de interferência de primitivas contra o cubo.

Este teste é mais complexo que o de raio contra primitiva. Caso decida-se por se fazer o teste no SCCena, serão necessários rotinas que calculem a interferência de cubo contra

elipsóides, cilindróides, romboedros, sendo necessário implementar uma rotina de interferência de quádrica contra semi–espaço plano. No caso deste teste ser executado no SCPrimtiva, então o voxel da Octree é levado em um romboedro, e será necessário também resolver a interferência de uma quádrica contra um semi–espaço plano (Figura VI.3).

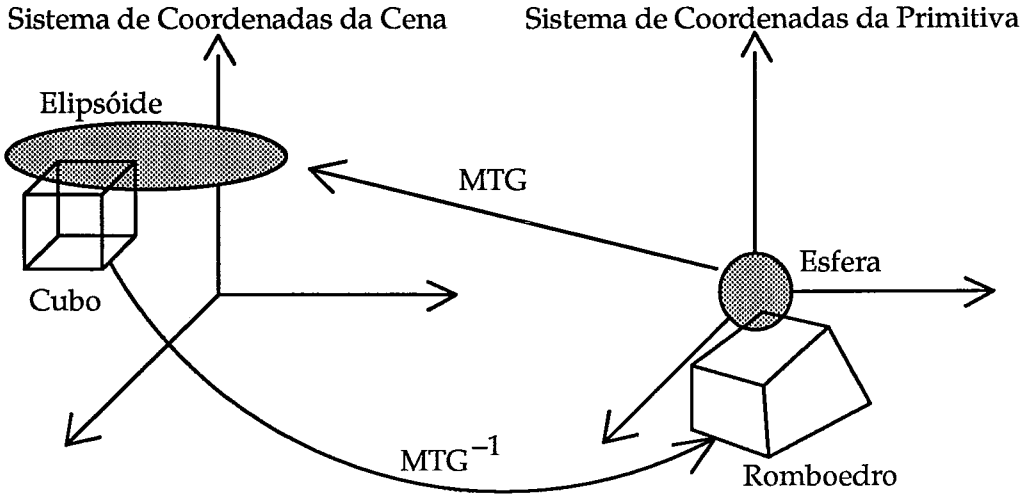


FIGURA VI.3 – Cálculo de Interseções Cubo x Primitiva

A conclusão deste estudo é que torna–se necessário testar a interferência de planos contra quádricas. Este método não pareceu ser eficiente e pesquisou–se outras alternativas que permitissem a realização de classificações com testes apenas lineares.

A idéia básica deste novo tipo de alternativa seria associar a cada primitiva um volume limitante que o representasse (de preferência constituído de semi–espaços planos). Assim, o teste complexo contra quádricas poderia ser substituído por testes lineares contra semi–espaços planos. SAMET et allii (1985) propôs um algoritmo bastante eficiente para classificar semi–espaços planos contra cubos. A estrutura utilizada por SAMET era originariamente uma Bintree (ver definição em SAMET (1990a, 1990b), mas pode ser facilmente estendida para lidar com uma Octree. Este algoritmo será utilizado para classificar os volumes limitantes de cada primitiva contra os cubos.

Os volumes limitantes associados a cada primiva serão descritos a seguir. O algoritmo de SAMET será apresentado logo após, com sua utilização no processo de classificação de cada semi–espaço pertencente ao volume limitante contra os voxels da Octree.

VI.3.1.2 – A escolha dos volumes limitantes

O uso de volume limitantes tem por objetivo reduzir a complexidade dos testes de interferência. No modelador CSG que usamos é necessário a criação de volumes limitantes para as primitivas esfera, cilindro e bloco.

Para que tivéssemos um volume limitante que mais se aproximasse da primitiva decidiu–se por criar volumes limitantes baseados em poliedros convexos (criados através da interseção de semi–espaços planos). Esta escolha é adequada pois o volume limitante pode aproximar bem a primitiva, e os testes de interseção tem sua complexidade reduzida (pois o teste de interseção contra semi–espaços é mais simples).

A especificação destes volumes limitantes pode ser feita em CSG, visto que a primitiva semi–espaço é uma primitiva disponível no modelador. Os seguintes volumes limitantes foram os escolhidos (Figura VI.4):

- **Esfera**: Foi escolhido um Icosaedro regular
- **Bloco** : Foi escolhido um cubo
- **Cilindro**: Foi utilizado um prisma com base hexagonal

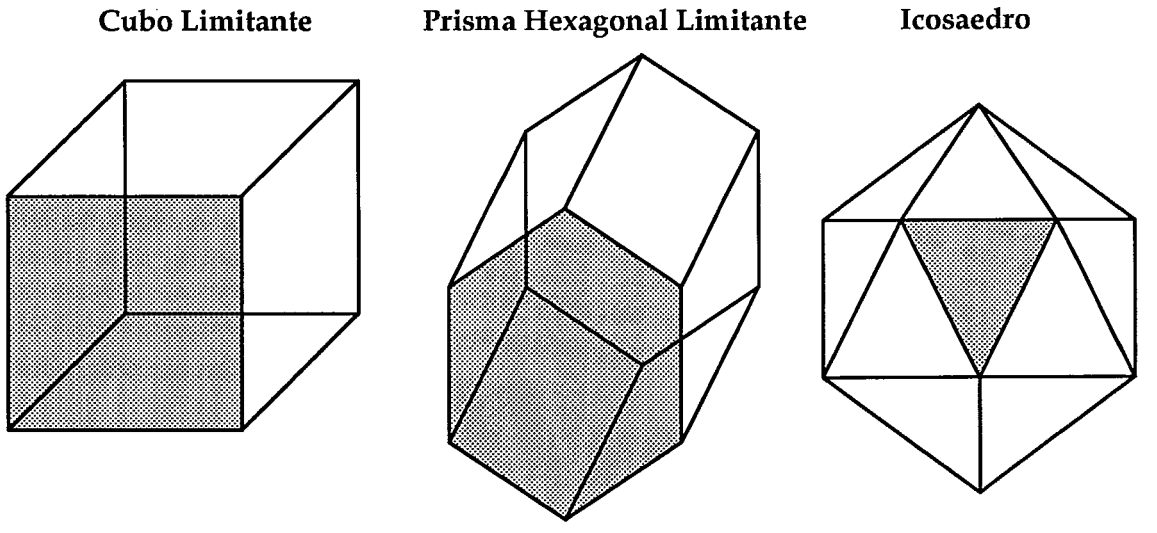


FIGURA VI.4 – Volumes Limitantes

VI.3.1.3 – Algoritmo para avaliação de CSG através de conversão para Bintrees

SAMET (1985) apresentou um algoritmo para avaliação de uma árvore CSG constituída de semi–espaços planos através de um conversão para uma Bintree. O objetivo do algoritmo é

montar uma Bintree, a fim de melhor avaliar sólidos CSG (exibição, cômputo de propriedades geométricas, etc).

A idéia básica deste algoritmo consiste em verificar o tipo de interferência existente entre um dado semi–espaço plano e uma célula da Bintree. SAMET argumentou que este teste pode ser feito de forma rápida e eficiente, através de uma soma e uma multiplicação.

Vamos analisar a proposta de SAMET. Um semi–espaço pode ser descrito pela seguinte inequação:

$$\sum_{i=0}^{d} a_i \cdot x_i \geq 0 \qquad \textbf{(Equação VI.1)}$$

Um semi–espaço exemplo pode ser visto na figura VI.5.

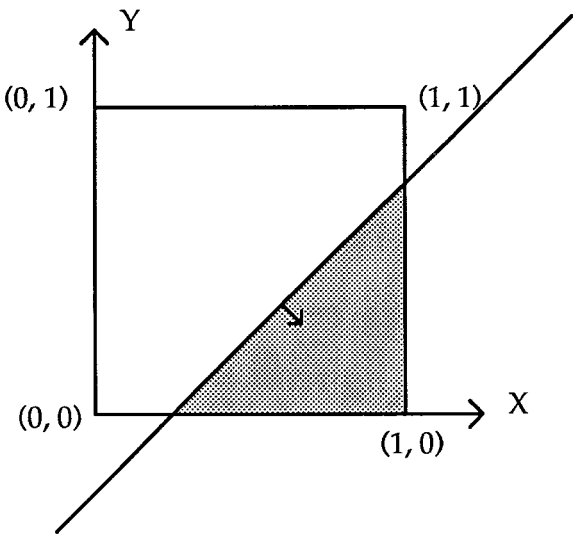


FIGURA VI.5 – Semi–Espaço referente a 4x – 4y >= 0

Para avaliar o tipo de interferência que um semi–espaço faz com um quadrado (no caso do plano) ou um cubo (no caso do espaço) basta analisar os valores mínimos e máximos dos produtos dos coeficientes (a_i) pelos valores dos diversos pontos (x_i) dentro destas regiões. Quando o máximo for menor ou igual a zero o semi–espaço não faz interseção. Quando o mínimo for maior ou igual a zero o semi–espaço contém a região. Nos demais casos, existe uma interseção parcial.

A determinação destes valores de mínimo e máximo pode ser feita de maneira bastante eficiente se a região for definida como um quadrado unitário (0 <= x, y <= 1) ou um cubo

unitário (0 <= x, y, z <= 1). Neste caso, os extremos da forma linear podem ser calculados da seguinte forma:

$$\mathbf{Minimo} = a_0 + \sum_{i=1}^{n} a_i^- \qquad \textbf{(Equação VI.2a)}$$

$$\mathbf{Maximo} = a_0 + \sum_{i=1}^{n} a_i^+ \qquad \textbf{(Equação VI.2b)}$$

Onde a_i^- e a_i^+ representam os coeficientes negativos e positivos do semi–espaço

No caso da figura VI.5 o mínimo vale –5 e o máximo 3, o que significa dizer que o semi–espaço faz interseção parcial com a região.

Para efeitos de subdivisão, quando o semi–espaço faz interseção parcial com o voxel, o espaço correspondente é subdividido de forma binária (o que irá originar a estrutura hierárquica conhecida como Bintree). SAMET apontou que quando o espaço é subdividido em dois, ou o valor do mínimo, ou o do máximo, é alterado (embora não simultaneamente).

SAMET observou que os sucessivos valores de máximos e mínimos decorrentes das subdivisões binárias poderiam ser obtidos pela soma dos valores dos máximos (ou mínimos, dependendo do caso) da subdivisão ancestral. O pseudocódigo deste algoritmo é apresentado a seguir.

```
PROCEDURE LocalizaSemiEspaco
    (
        SemiEspaco   : DadosSemiEspaco;   (* Dados referentes ao semi–espaco *)
        Nivel        : CARDINAL;          (* Nivel de subdivisoes da Bintree *)
        Direcao      : DirecaoSubdivisao  (* Indica a direcao da subdivisao *)
    ):
        TipodeInterferencia ;             (* Inclusao, Exclusao Ou Interferência *)

    BEGIN
        NovoMinimo := SemiEspaco.Minimo;
        NovoMaximo := SemiEspaco.Maximo;
        NivelOctree := (Nivel DIV 3) + 1 ;
        Coordenada  := (Nivel MOD 3) + 1 ;
        Delta := SemiEspaco.Equacao [Coordenada] * Largura [NivelOctree] ;
        IF Direcao = Esquerda THEN
            IF Delta > 0.0 THEN
                NovoMaximo := NovoMaximo – Delta
            ELSE
                NovoMinimo := NovoMinimo – Delta
            END
        ELSE
```

```
            IF Delta > 0.0 THEN
                NovoMinimo := NovoMinimo + Delta
            ELSE
                NovoMaximo := NovoMaximo + Delta
            END
        END;
        IF (NovoMaximo < 0.0) THEN
            (* O semi espaço nao esta no voxel  *)
            RETURN EXCLUSAO
        ELSE
            IF (NovoMinimo > 0.0)  THEN
                (* O semi-espaco contem o voxel *)
                RETURN INCLUSAO
            ELSE
                RETURN INTERFERENCIA
            END ;
        END
    END
END LocalizaSemiEspaco;
```

No exemplo da figura VI.6, dividindo o espaço em dois em direção perpendicular ao eixo Y e a seguir em direção a X teremos quatro subdivisões com valores de máximo e mínimo obtidos a partir dos valores de máximo e mínimo inicial.

O uso de volume limitante faz com que alguns casos de exclusão sejam facilmente descartados (evitando assim a necessidade do teste de interferência contra a primitiva). Entretanto, em casos que o volume limitante engloba o voxel nada se pode afirmar, pois existe a possibilidade da primitiva não o conter. Este caso é mostrado na figura VI.7.

No caso do volume limitante conter o voxel, nada se pode afirmar se a primitiva também o contém, pois como vimos na figura VI.7 existem casos onde este fato pode não ocorrer. Uma solução trivial para este problema seria dois volumes limitantes por primitiva: um circunscrito e outro inscrito. Caso o teste com o volume circunscrito indicasse que este engloba o voxel, o volume inscrito seria testado para interseção. Caso ele também englobasse o voxel, então poderíamos afirmar com certeza que a primitiva também englobava o voxel.

Esta alternativa, embora resolva o problema, parece um tanto quanto custosa, visto que serão necessários dois volumes limitantes por primitiva. Nos casos de dúvidas devem-se fazer dois testes de interferência, o que pode inviabilizar o uso de volumes limitantes (caso o tempo de cálculo de interferência contra o volume limitante fosse maior que a metade do tempo do cálculo de interseção contra a primitiva).

Usando de forma conveniente os semi-espaços que delimitam o volume circunscrito, temos condições de resolver o caso de inclusão sem a necessidade de especificação explícita de

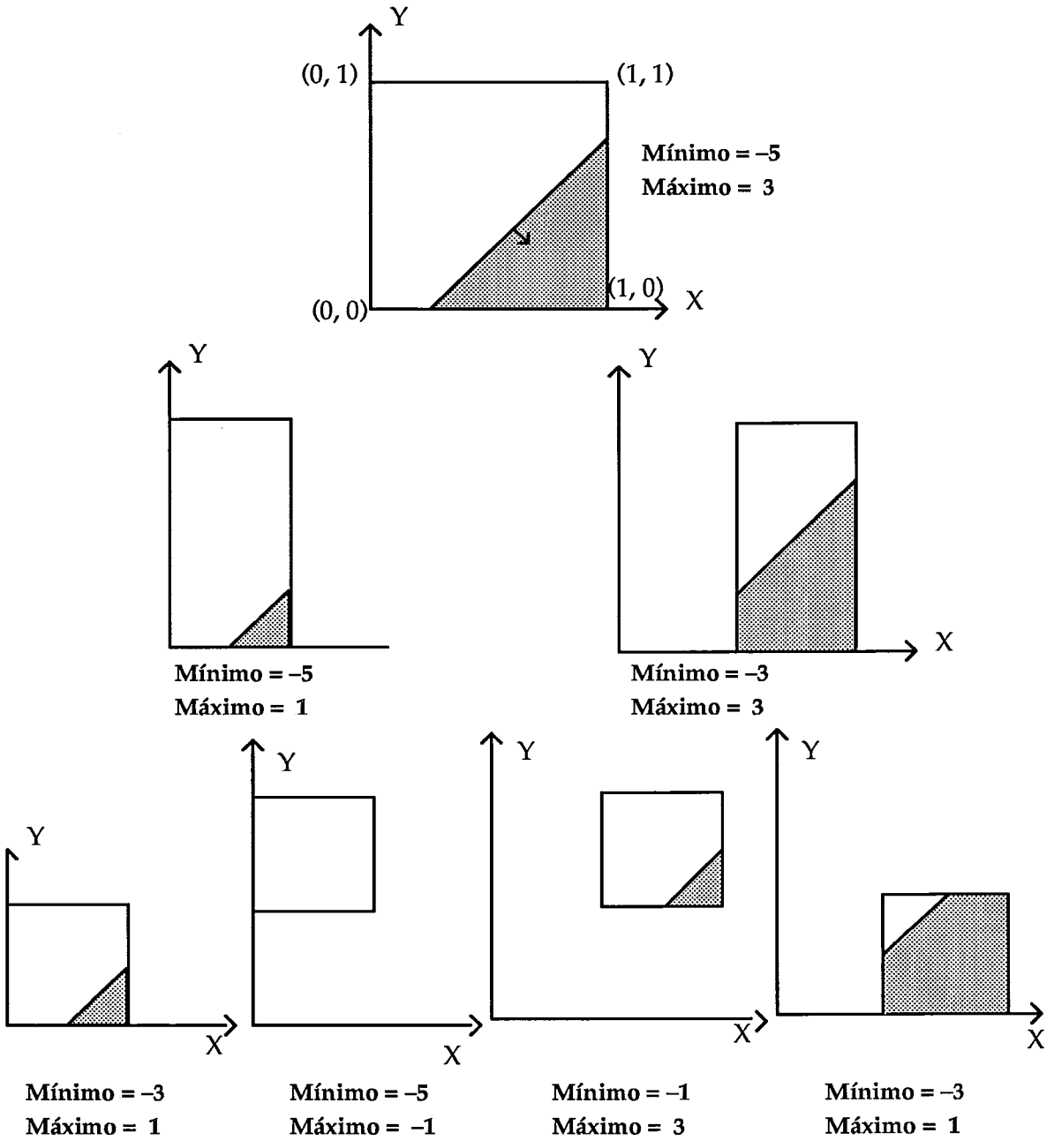


FIGURA VI.6 – Alteração dos valores de máximo e mínimo

um volume inscrito à primitiva. A idéia consiste em definir os semi–espaços que compõem o volume circunscrito de forma que uma determinada translação radial destes definam um volume inscrito à primitiva. Na prática o que se faz é multiplicar todos os coeficientes que definem os semi–espaços do volume circunscrito por uma constante adequada. Desta forma, os semi–espaços que definem o volume inscrito são obtidos através de uma translação da unidade sobre os semi–espaços que definem o volume circunscrito. As constantes utilizadas revelam proporções entre os raios das esferas circunscritas e inscritas aos poliedros utilizados como volumes limitantes. (Ver RANGEL (1982) (Figura VI.8))

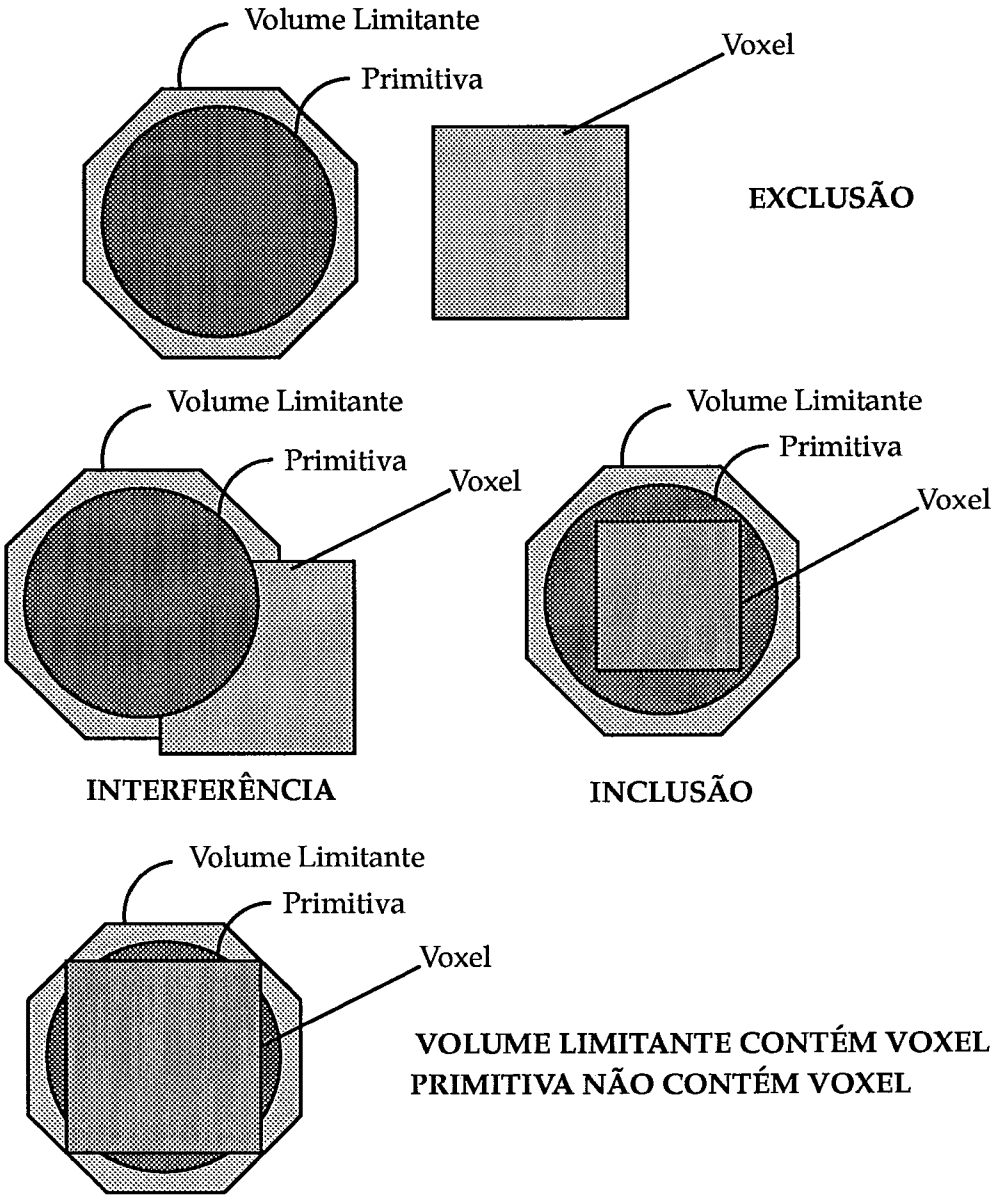


FIGURA VI.7 – Tipos de Interferências

VI.3.2 – Simplificação da Árvore CSG

Esta etapa preocupa-se com a geração de simplificações da árvore CSG que estão associadas com os diversos voxels na Octree. Na seção anterior vimos como classificar uma primitiva contra um voxel. Agora, devemos fazer uso destas informações de forma a gerar a estrutura híbrida entre Octree e CSG (árvore OCSG).

O algoritmo, como proposto por SAMET, é extremamente adequado para a geração de uma estrutura Bintree, visto que a cada iteração do algoritmo o espaço é subdividido de forma

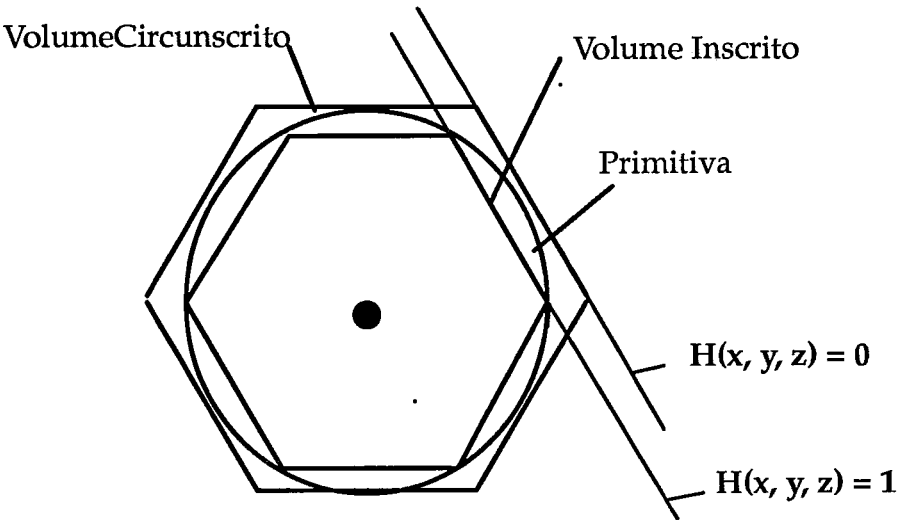


FIGURA VI.8 – Obtenção do volume inscrito

binária. Como a estrutura hierárquica que desejamos gerar é uma Octree, o algoritmo deve ser adaptado para esta nova estrutura.

No algoritmo de SAMET as classificações obtidas quando da subdivisão do espaço em X ou em Y referiam–se a porções retangulares do espaço. Quando ocorria a subdivisão em Z, a classificação obtida referia–se a uma porção cúbica do espaço. O algoritmo continuava seu processamento tendo como critério de parada obter uma interferência do tipo inclusão ou exclusão, ou atingir o limite máximo de profundidade da árvore.

No caso de aplicarmos este mesmo algoritmo para Octrees, o critério acima não é mais satisfatório, pois em uma Octree todas as células possuem um formato cúbico. Por isso este critério deve ser alterado, de forma a fazer obrigatoriamente após cada corte em X, cortes também em Y e Z. Assim, é garantido que o resultado do teste de interferência será sempre referente a uma porção cúbica do espaço. A figura VI.9 ilustra este processo.

Mas como tratar os casos que com o critério anterior pararíam, e com o novo prossiguiriam na subdivisão ? Por exemplo, se uma dada subdivisão binária for vazia, os voxels referentes a esta subdivisão (podendo estes ser quatro, dois ou apenas um) também serão vazios, não sendo necessário em alguns casos a subdivisão nos demais eixos coordenados. Para evitar continuar o processo de subdivisão, mantendo a condição de não gerar classificações referentes a partes não cúbicas, deve–se alterar o algoritmo de forma a gerar cada um dos subvoxels (quatro, dois ou um) com resultado igual ao obtido neste ponto do algoritmo.

Vamos analisar um exemplo. Seja um corte em X: a subdivisão à esquerda irá determinar o conteúdo dos octantes 0, 2, 4, 6 e à direita os octantes 1, 3, 5, 7. Caso a subdivisão à

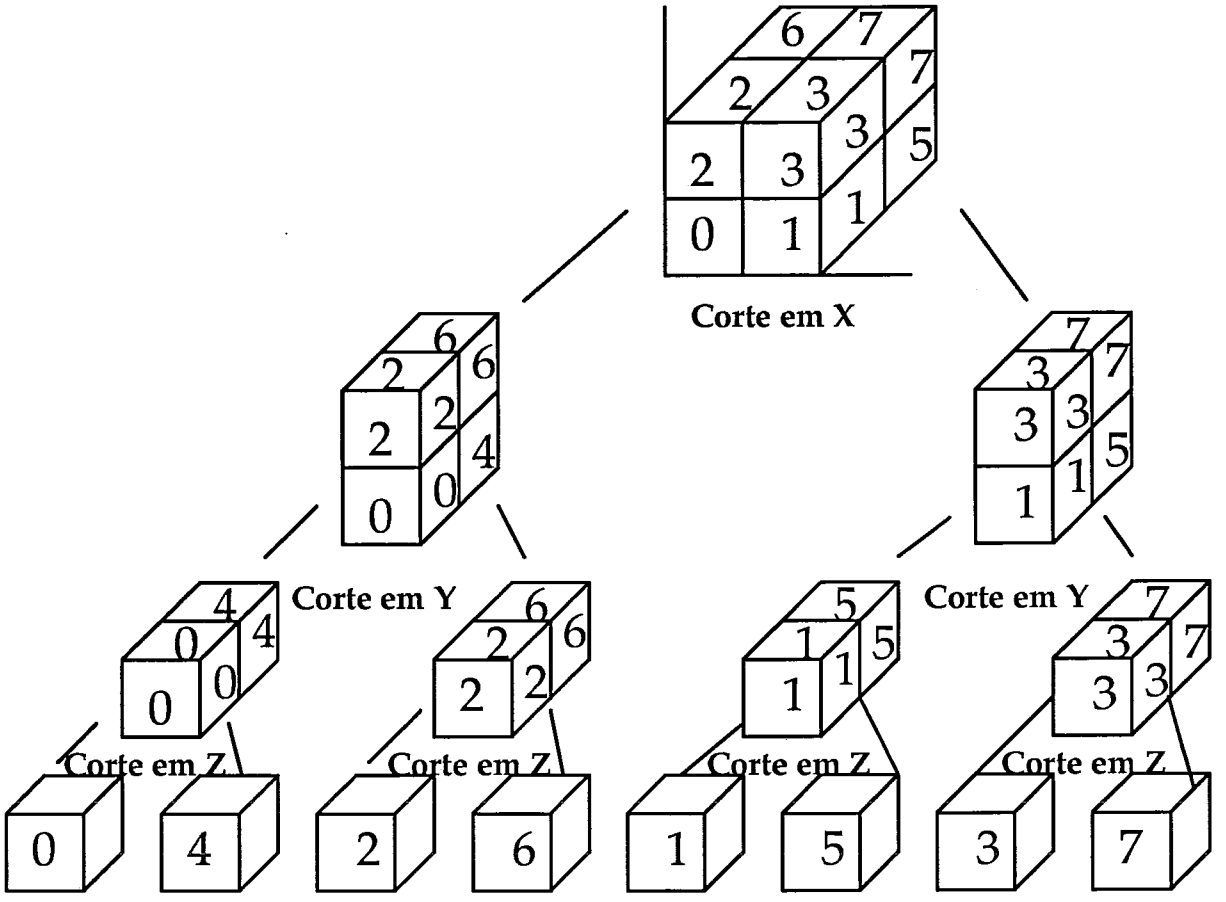


FIGURA VI.9 – Subdivisão binária do espaço de forma a obter uma Octree

esquerda tenha como resultado do teste de interferência exclusão, podemos associar aos nodos referentes aos octantes 0, 2, 4, 6 o tipo vazio, sem a necessidade de continuar o processo. O processo de subdivisão pode, portanto, parar no mesmo ponto que o processo parava segundo o critério anterior, desde que haja o cuidado necessário de associar a todos octantes interessados na subdivisão o resultado da classificação. Para auxiliar a determinar os octantes de interesse a uma determinada subdivisão, foram criadas três tabelas, referentes a primeira subdivisão (em X), a segunda (em Y) e a terceira (em Z). A tabela é acessada pelo lado de interesse da subdivisão (esquerdo ou direito). Como a subdivisão em Y vem logo após a subdivisão em X a tabela referente a esta subdivisão possui duas chaves de acesso (referente ao lado da subdivisão em X e em Y). O acesso da tabela referente à subdivisão em Z possui 3 chaves de forma análoga. (Ver figura VI.10).

Usando este critério, podemos otimizar bastante o processo de subdivisão, pois o número de subdivisões necessárias tende a ser bem menor.

Na seção seguinte iremos apresentar a adaptação do algoritmo de SAMET para a geração da árvore OCSG.

O Pseudocódigo usa as tabelas descritas na figura VI.10 para otimização do algoritmo.

Primeira Subdivisão (X)

Esquerda	0 2 4 6
Direita	1 3 5 7

Segunda Subdivisão (Y)

Esquerda Esquerda	0 4
Esquerda Direita	2 6
Direita Esquerda	1 5
Direita Direita	3 7

Terceira Subdivisão (Z)

Esquerda Esquerda Esquerda	0	Direita Direita Direita	7
Esquerda Esquerda Direita	4	Direita Direita Esquerda	3
Esquerda Direita Esquerda	2	Direita Esquerda Direita	5
Esquerda Direita Direita	6	Direita Esquerda Esquerda	1

FIGURA VI.10 – Tabelas de auxílio à subdivisão

```
PROCEDURE GeraArvoreOCSG
    (
        Raiz                    : ArvoreBBox;    (* Arvore CSG com volumes limitantes *)
    )   :
        OctreeCSG ;                              (* Retorna arvore hibrida entre Octree e CSG *)

    VAR
        RaizOCSG : OctreeCSG ;

    BEGIN
        RaizOCSG := AlocaNodoMistoOCSG () ;
        (*    Corta o espaco de forma binaria *)
        (*    O vetor Direcao ira conter o lado da subdivisao. A primeira posicao
              refere-se a subdivisao em X, a segunda a Y e a terceira a Z *)
        Direcao [1] := Esquerda ;
        SubdivideEspaco (Raiz, RaizOCSG, 1, Direcao) ;
        Direcao [1] := Direita ;
        SubdivideEspaco (Raiz, RaizOCSG, 1, Direcao) ;
        RETURN RaizOCSG ;
    END GeraArvoreOCSG;
```

```
PROCEDURE SubdivideEspaco
    (
        Raiz            : ArvoreBBox ;      (* Arvore CSG com volumes limitantes *)
        VAR RaizOCSG    : OctreeCSG;        (* Nodo da arvore OCSG *)
        Nivel           : CARDINAL;         (* Nivel na subdivisao binaria *)
        Direcao         : TipoDirecao ;     (* Esquerda ou Direita *)
    );

    BEGIN
        (* Obtem a localizacao da arvore referente a uma subdivisao *)
        ArvoreTemp := PodaArvore (Raiz, Nivel, Direcao) ;
        (* Avalia o tipo do nodo da arvore localizada: Universo, Vazio, CSGReduzido ou Indeciso *)
        TipoNodo := AvaliaTipoNodo (ArvoreTemp) ;
        IF (TipoNodo = Indeciso) THEN
            Coordenada := (Nivel MOD 3) + 1 ;
            IF (Coordenada = 3) THEN
                (* Descer mais um nivel na Octree *)
                Octante := TerceiraSubdivisao [Direcao[1], Direcao[2], Direcao[3]] ;
                RaizOCSG.Filho [Octante] := AlocaNodoMistoOCSG () ;
                Direcao [1] := Esquerda ;
                SubdivideEspaco (Raiz, RaizOCSG.Filho [Octante] , Nivel + 1, Direcao) ;
                Direcao [1] := Direita ;
                SubdivideEspaco (Raiz, RaizOCSG.Filho [Octante], Nivel + 1, Direcao) ;
            ELSE
                (* Prossegue subdivisao binaria *)
                Direcao [Nivel + 1] := Esquerda ;
                SubdivideEspaco (Raiz, RaizOCSG, Nivel + 1, Direcao) ;
                Direcao [Nivel + 1] := Direita ;
                SubdivideEspaco (Raiz, RaizOCSG, Nivel + 1, Direcao) ;
            END
        ELSE
            CASE (Nivel MOD 3) + 1 OF
            (*  Gerar os filhos deste lado da subdivisao com o mesmo tipo de
                TipoNodo: Universo, Vazio ou CSGReduzido
            |   1 : (* Usar a tabela da primeira subdivisao *)
                FOR i := 1 TO 4 DO
                    RaizOCSG.Filho [
                        PrimeiraSubdivisao [Direcao [1]]
                    ]   := AlocaFolhaOCSG (TipoNodo, ArvoreTemp) ;
                END ;
            |   2 : (* Usar a a tabela da segunda subdivisao *)
                FOR i := 1 TO 2 DO
                    RaizOCSG.Filho [
                        SegundaSubdivisao [Direcao [1], Direcao [2]]
                    ]   := AlocaFolhaOCSG (TipoNodo, ArvoreTemp) ;
                END ;
```

```
            |   3 : (* Usar a tabela da terceira subdivisão *)
                    RaizOCSG.Filho [
                        TerceiraSubdivisao [Direcao [1], Direcao [2], Direção [3]]
                    ]   := AlocaFolhaOCSG (TipoNodo, ArvoreTemp) ;
                END
            END ;
            LiberaArvoreBBox (ArvoreTemp) ;
        END SubdivideEspaco ;

PROCEDURE PodaArvore
    (
        Raiz            : ArvoreBBox ;      (* Arvore CSG com volumes limitantes *)
        Nivel           : CARDINAL;         (* Nivel na subdivisao binaria *)
        Direcao         : TipoDirecao ;     (* Esquerda ou Direita *)
    ) :
        ArvoreBBox ;                        (* Arvore podada *)

BEGIN
    IF Folha (Raiz) THEN
        (*  Aplica o algoritmo de SAMET para obter a localizacao da arvore CSG
            Para cada semi-espaco pertencente ao volume limitante e aplicado o
            algoritmo, mantendo aqueles que fazem interferencia com a subdivisao (ativos) *)
        ListaSemiEspacosAtivos :=
            CalculaSemiEspacosAtivos (Raiz.VolumeLimitante, Nivel, Direcao) ;
        (* Avalia o resultado desta classificacao: Interferencia, Inclusao ou Exclusao *)
        ResultadoClassificacao := AvaliaClassificacao (ListaSemiEspacosAtivos) ;
        CASE ResultadoClassificacao OF
        |   INTERFERENCIA:
                ArvorePodada := AlocaArvoreBBox () ;
                ArvorePodada.VolumeLimitante := ListaSemiEspacosAtivos ;
                RETURN ArvorePodada ;
        |   EXCLUSAO:
                RETURN ArvoreVazia ;
        |   INCLUSAO:
                RETURN ArvoreUniverso ;
        END
    ELSE
        Esquerda :=  PodaArvore (Raiz^.Esquerda, Nivel, Direcao) ;
        Direita := PodaArvore (Raiz^.Direita, Nivel, Direcao) ;
        RETURN OperacaoBooleana (Esquerda, Direita, Arvore^.Operador) ;
    END ;
END PodaArvore;
```

Capítulo VII

Algoritmos de Progressão Celular

VII.1 – INTRODUÇÃO

Neste capítulo iremos discutir os fundamentos de uma nova proposta de acompanhamento de raios por uma Octree: o algoritmo de Progressão Celular. Inicialmente o algoritmo terá seus fundamentos apresentados. Em uma primeira etapa, este algoritmo será projetado para enumerar as células pelas quais uma reta passa numa malha de mesma resolução. A extensão para malhas de resolução variável é estudada na seqüência, onde aparecem com destaques as operações de Detalhamento e Integração. A extensão do algoritmo para 3D, importante por determinar as células pelas quais uma reta passa em um malha multiresolução no espaço, é apresentada logo após.

VII.2 – O ALGORITMO DE PROGRESSÃO CELULAR

O algoritmo de Progressão Celular tem por objetivo principal enumerar (encontrar) aquelas células em uma malha retangular que são interceptadas por um segmento de reta que une dois pontos de coordenadas inteiras.

Sua implementação pode ser feita de diversas maneiras, mas a mais adequada é utilizando aritmética inteira e incremental. A apresentação do algoritmo será feita de forma formal, pois este formalismo irá assegurar o correto tratamento de todos casos possíveis, além do que será útil quando mais tarde formos discutir o algoritmo de Progressão Celular para Multi-Resoluções.

Definição 1 – Célula: Uma célula pertencente a uma subdivisão do plano é quadrada e é delimitada da seguinte forma:

$$C(i, j, L) = \{(x, y) \mid i.L <= x < (i+1).L,\ j.L <= y < (j+1).L\}$$

onde:

i, j referem-se aos índices da célula

L refere-se ao tamanho do lado da célula

A figura VII.1 ilustra o conceito de célula.

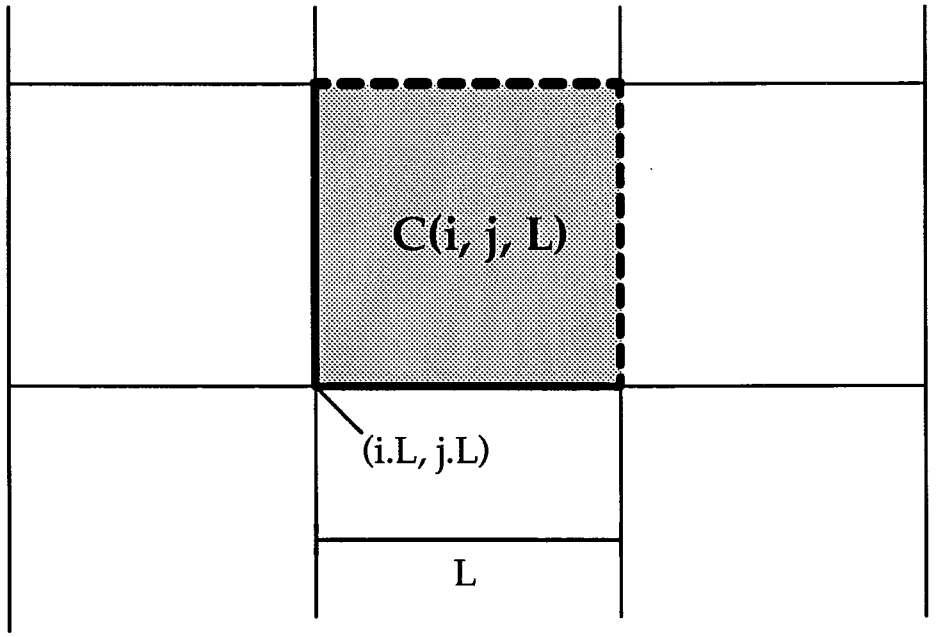


FIGURA VII.1 – Exemplo de célula

O problema que nos propomos a solucionar pode ser enunciado da seguinte forma:

Problema: Encontrar as células que são interceptadas por um segmento de reta R que une dois pontos p1 e p2 de coordenadas inteiras (x1, y1) e (x2, y2) respectivamente.

A reta que passa por estes dois pontos pode ser expressa implicitamenta na seguinte forma:

$$F(x, y) = -(x - x1)\,(y2 - y1) + (y - y1)\,(x2 - x1) = 0$$

Por simplicidade de exposição assumiremos que o coeficiente angular da reta é positivo (à semelhança de FOLEY (1990), por exemplo). Assim, podemos estabelecer que:

Condição 1: (x1 <= x2) e (y1 <= y2)

Esta condição fica garantida por:

Condição 1a: F(x+1, y) <= F(x, y) <= F(x, y+1)

Esta última condição revela que um avanço no eixo do X provoca um decréscimo em F(x, y), enquanto que um avanço no eixo Y provoca um acréscimo em F(x, y). A figura VII.2 ilustra esta condição.

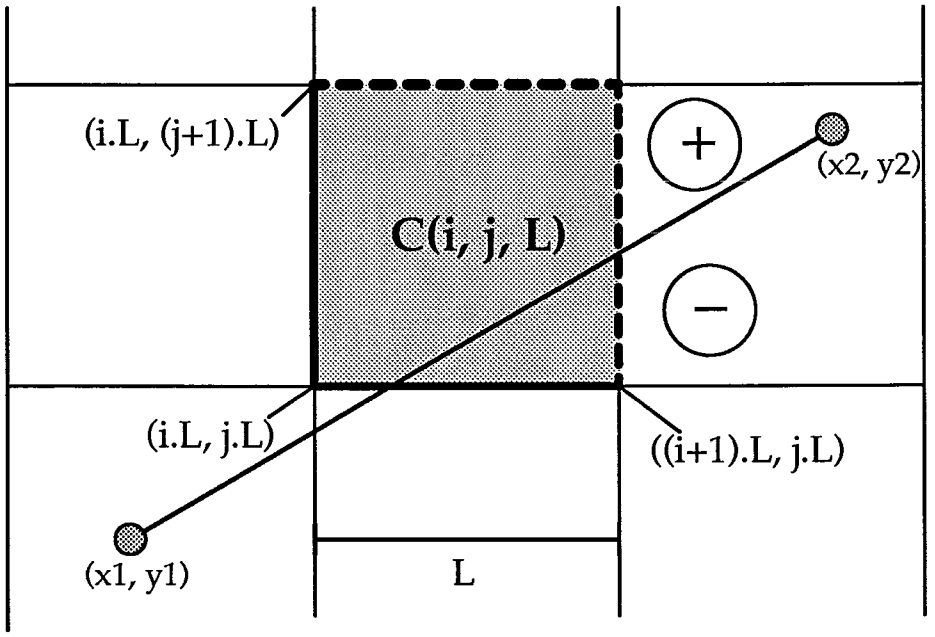


FIGURA VII.2 – Restrições à reta R

Como conseqüência da continuidade de F podemos estabelecer as seguintes condições para que uma dada célula seja interceptada pela reta:

Propriedade 1: Uma célula C(i, j, L) é interceptada por uma reta R se e somente se:

(1) F(i.L, j.L) = 0 ou

(2) F(i.L, (j+1).L) > 0 e F((i+1).L, j.L) < 0

Esta condição pode ser melhor entendida pela figura VII.3. Caso a função se anule em (i,j), então ela seguramente passa pela célula, mesmos nos casos de reta horizontal e vertical.

Caso isto não aconteça, basta que os valores F (i+1, j) e F(i, j+1) tenham sinais opostos, significando lados opostos da reta.

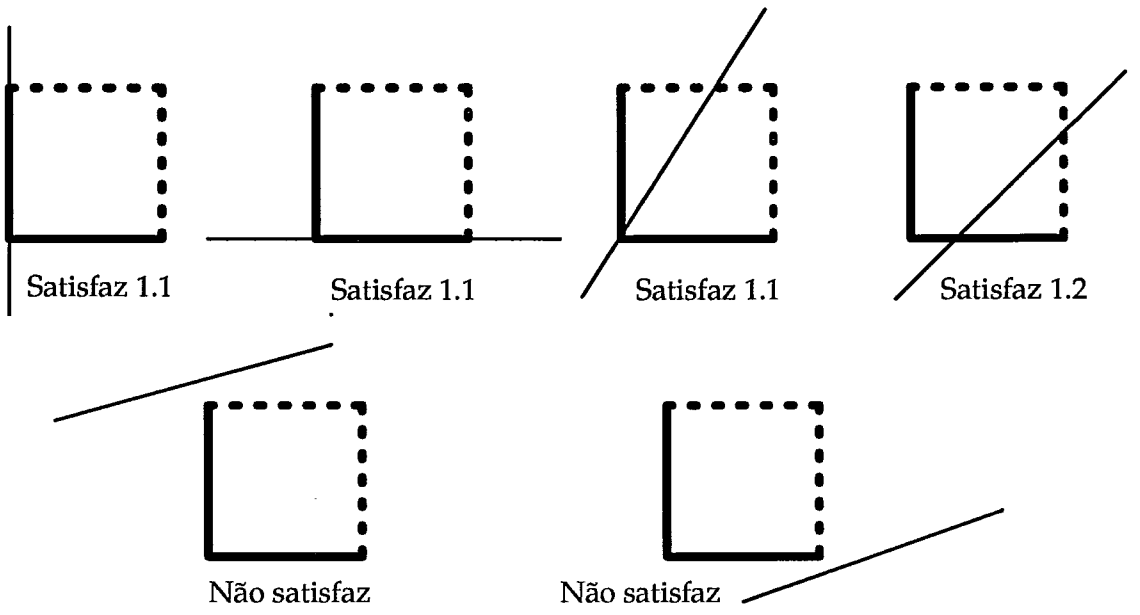


FIGURA VII.3 – Análises da Propriedade 1 em várias células

Baseados nas condições de interseção de uma célula C(i, j, L), podemos estabelecer as condições para que uma de suas vizinhas seja interceptada. Dessa forma, temos como progredir desde a primeira célula contendo, digamos, (x1, y1) até as demais.

Critério de Progressão: Dada uma célula C(i, j, L) que faz interseção com o segmento de reta R, então a célula seguinte será determinada pelas seguintes condições:

(1) Se F((i+1).L, (j+1).L) > 0 então a próxima célula será C(i+1, j, L)

(2) Se F((i+1).L, (j+1).L) < 0 então a próxima célula será C(i, j+1, L)

(3) Se F((i+1).L, (j+1).L) = 0 então a próxima célula será C(i+1, j+1, L)

A validade deste critério é consequência imediata da Propriedade 1.

O Critério de Progressão nos permite evoluir pela malha de subdivisão, passando de uma célula à sua vizinha. No entanto, para darmos início à progressão, é necessário partir de uma célula sabidamente interceptada pelo segmento de reta.

Para isso basta calcular a célula que contém o extremo (x1, y1): a célula C(i, j, L) tal que:

(1) $i = \lfloor x1/L \rfloor$ e

(2) $j = \lfloor y1/L \rfloor$ onde $\lfloor x \rfloor$ denota o truncamento de x.

Estabelecido como obter a primeira célula, e como passar para as células seguintes, podemos apresentar a primeira versão do algoritmo:

```
PROCEDURE ProgressaoCelular (
    x1,        (* Coordenadas do Ponto Inicial *)
    y1,
    x2,        (* Coordenadas do Ponto Final *)
    y2
)
    BEGIN        (* Determinar a celula inicial *)
        i := TRUNC (x1 / L) ;
        j := TRUNC (y1 / L) ;
        VisitaCelula (i, j) ;
        WHILE (i * L)  <= x2 DO
            f := F((i+1).L, (j+1).L) ;
            IF (f > 0) THEN
                i := i + 1
            ELSE
                IF (f < 0) THEN
                    j := j + 1;
                ELSE i := i + 1 ; j := j + 1 ;
                END
            END
        END
        VisitaCelula (i, j) ;
    END ProgressaoCelular  ;
```

O algoritmo proposto acima pode ser aperfeiçoado de forma a se tornar incremental, sem a necessidade de avaliar a cada iteração a expressão linear F(x, y). Para tanto, é preciso identificarmos uma invariante no algoritmo (isto é, uma condição que mantenha–se constante antes e ao fim de cada iteração). Por meio da invariante poderemos usar suas propriedades de forma a eliminar o cálculo repetido de F(x, y).

Antes de determinarmos a invariante, é necessário apresentarmos a seguinte definição:

Definição 2 – Conjunto E: Seja E o conjunto de todas as sextuplas (i, j, L, f, incx, incy) tais que valem as seguintes propriedades:

(a) i, j, L são inteiros

(b) f = F((i+1).L, (j+1).L)

(c) incx = F(L, 0) – F(0, 0) = –L (y2 – y1)

(d) incy = F(0, L) – F(0, 0) = L (x2 – x1)

Uma propriedade pode ser enunciada sobre este conjunto E:

Propriedade 2: Se a sextupla (i, j, L, f, incx, incy) ∈ E então:

(1) (i+1, j, L, f+incx, incx, incy) ∈ E

(2) (i, j+1, L, f+incy, incx, incy) ∈ E

(3) (i+1, j+1, L, f+incx+incy, incx, incy) ∈ E

Prova da Propriedade 2:

- Propriedade 2.1:

  Condição a: i+1, j e l sao inteiros

  Condição c e d: incx = F(L, 0) – F(0, 0) e incy = F(0, L) – F(0, 0) valem

  Condição b:

  f = F((i+1).L, (j+1).L)
  f = –((i+1).L–x1).(y2–y1)+(x2–x1).((j+1).L–y1)
  f = –((i+2).L–x1–L).(y2–y1)+(x2–x1).((j+1).L–y1)
  f – (L * (y2 – y1)) = –((i+2).L–x1).(y2–y1)+(x2–x1).((j+1).L–y1)
  f + incx = F((i+2).L, (j+1).L)
- Propriedades 2.2 e 2.3 –> De forma análoga

Estabelecemos como invariante do algoritmo a condição da sextupla (i, j, L, f, incx, incy) ∈ E e o fato de C(i, j, L) ser interceptada por R. Para garantir a validade da invariante somente aplicaremos alterações sobre a sextupla que satisfaçam a propriedade 2.

Assim, o pseudocódigo do algoritmo incremental é o seguinte:

```
PROCEDURE ProgressaoCelular (
    x1,        (* Coordenadas do Ponto Inicial *)
    y1,
    x2,        (* Coordenadas do Ponto Final *)
    y2
)

    BEGIN      (* Determinar a celula inicial *)
        i := TRUNC (x1 / L) ;
        j := TRUNC (y1 / L) ;
        incx = –L * (y2 – y1) ;
        incy = L * (x2 – x1) ;
        f = –((i+1)*L–x1)*(y2–y1)+((j+1)*L–y1)*(x2–x1)
        VisitaCelula (i, j) ;
```

```
WHILE (i * L) <= x2 DO
(* Invariante (i, j, L, f, incx, incy ∈ E) e C(i, j, L) é interceptada*)
    IF (f > 0) THEN
        i := i + 1 ;
        f := f + incx ;
    ELSE
        IF (f < 0) THEN
            j := j + 1;
            f := f + incy ;
        ELSE
            i := i + 1 ;
            j := j + 1 ;
            f := f + incx + incy ;
        END
    END
    VisitaCelula (i, j) ;
  END
END ProgressaoCelular ;
```

VII.3 – O ALGORITMO DE PROGRESSÃO CELULAR EM QUADTREES

Na seção anterior mostramos os conceitos referentes ao algoritmo de Progressão Celular em malhas regulares. Suponha agora a subdivisão do plano no formato de uma quadtree. A Quadtree subdivide o espaço em células de tamanho variável, onde podemos estabelecer as seguintes definições:

Definição 3 – Célula Filha: Para cada célula C(i, j, L), as células C1(2i, 2j, L/2), C2(2i+1, 2j, L/2), C3(2i, 2j+1, L/2) e C4(2i+1, 2j+1, L/2) são denominadas células filhas de C. (Ver figura VII.4).

Definição 4 – Célula Pai: Para cada célula C(i, j, L), a célula C'(⌊i/2⌋, ⌊j/2⌋, 2L) é denominada célula pai de C.

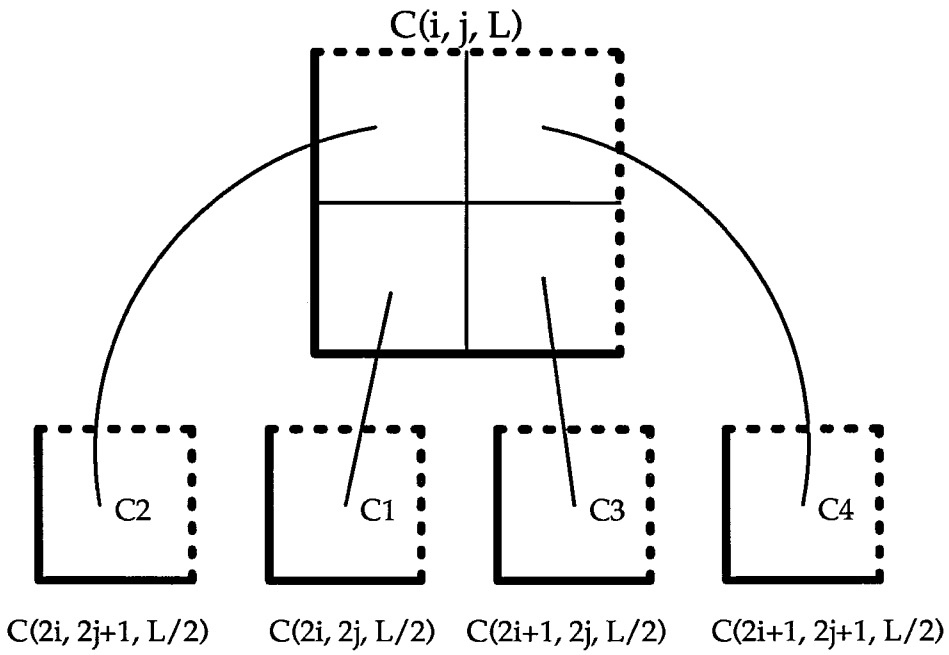


FIGURA VII.4 – Células Filhas

O processo de progressão utilizado anteriormente foi destacado em um procedimento à parte chamado algoritmo de progressão. Este algoritmo será igual ao do caso da malha regular, mesmo com multi-resolução. Seu pseudocódigo é o seguinte:

```
PROCEDURE Progressão (
    sextupla:     DadosSextupla     (* i, j, L, f, incx e incy *)
) :
    DadosSextupla ;                 (* Sextupla na celula obtida pela progressao *)
    BEGIN
        WITH sextupla DO
            IF f > 0 THEN
                i := i + 1;
                f := f + incx ;
            ELSE
                IF f < 0 THEN
                    j := j + 1;
                    f : f + incy ;
                ELSE
                    i := i + 1; j := j + 1 ;
                    f := f + incx + incy ;
                END
            END
        END ;
        RETURN sextupla ;
    END Progressao ;
```

O problema de enumerar as células que façam interseção com a reta torna-se agora mais complicado, pois as células podem ter tamanhos diferentes. Anteriormente a progressão era efetuada entre células de tamanhos iguais. Agora, dois novos casos devem ser levados em conta:

- **Transição de Integração**: Passagem para uma célula de lado maior.
- **Transição de Detalhamento**: Passagem para uma célula de lado menor.

A figura VII.5 ilustra as transições de Detalhamento e Integração.

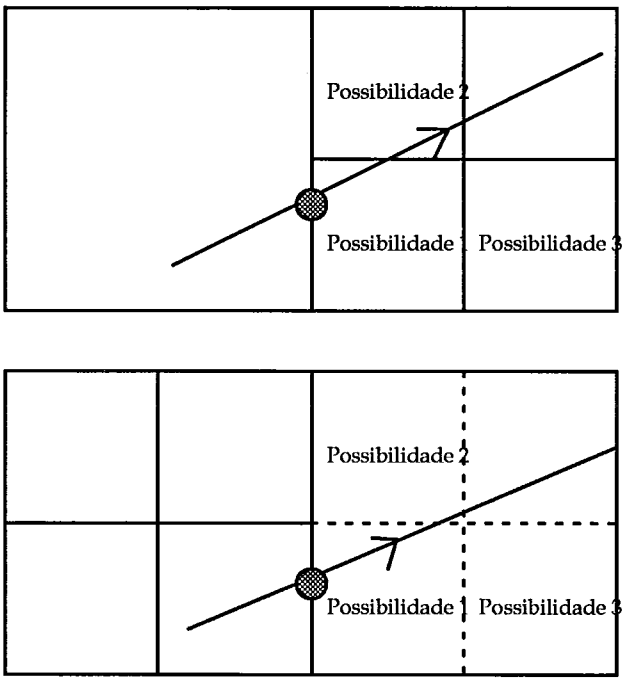


FIGURA VII.5 – Transições entre células de tamanhos diferentes

Os dois tipos de transição podem ser formalmente especificados por:

Definição 5 – Detalhamento: Consiste em encontrar a primeira filha C'(i', j', L') (e os correspondentes parâmetros da progressão (i', j', L', f', incx', incy') ∈ E)) de uma determinada célula C(i, j, L) que seria obtida na progressão com malha de largura L' = L/2 sobre a reta R.

Definição 6 – Integração: Consiste em determinar a célula pai C'(i', j', L') (e os correspondentes parâmetros da progressão (i', j', L', f', incx', incy') ∈ E)) de uma determinada célula C (i, j, L) que seria obtida na progressão com malha de largura L' = 2L sobre a reta R.

Observe que um procedimento de Integração é exatamente o inverso de um processo de Detalhamento. Entretanto, o Detalhamento da Integração pode ser distinto desta. Por conta desta observação iremos tratar inicialmente do processo de Detalhamento, enquanto o de Integração será deduzido por inversão daquele.

Tanto o procedimento de Detalhamento quanto no de Integração são aplicados após a realização da progressão. No Detalhamento de uma célula C(i, j, k, L) há apenas 3 filhas passíveis de serem geradas (vide Figura VII.5). Destas filhas, aquela que possui por origem a origem do pai é a que tende a ser primeiramente encontrada pelo raio. Portanto, se o raio interceptá–la então ela é certamente a solução. Por outro lado, as duas outras filhas são alternativas, isto é, uma e apenas uma delas é interceptada pelo raio.

Levando em conta as condições de interseção raio–célula estabelecidas previamente, podemos determinar a filha solução por meio do seguinte critério:

Critério de Detalhamento:

(a) A filha C(2i+1, 2j, L') é a solução se:

(a.1) $L' = L/2$ e

(a.2) Aplicando a função F sobre a célula vem:

$F((2i+1).L', 2j.L') >= 0$

Substituindo L' por L vem:

$F((i+1).L - 1/2L, (j+1)L - L) >= 0$

Expandindo, vem:

$F((i+1)L, (j+1)L) - 1/2\,(F(L, 0) - F(0, 0)) - (F(0, L) - F(0, 0)) >= 0$

Agrupando os termos usando a definição 2 vem:

$f - incx/2 + incy >= 0$

Além disto:

$i' = 2i + 1,\ j' = 2j,\ f' = f - incy/2,\ incx' = incx/2,\ incy' = incy/2$

(b) A filha C(2i, 2j + 1, L') é a solução se:

(b.1) $L' = L/2$ e

(b.2) $f - incy/2 + incx >= 0$

Além disto:

$i' = 2i,\ j' = 2j + 1,\ f' = f - incx/2,\ incx' = incx/2,\ incy' = incy/2$

(c) Nos demais casos a filha C(2i, 2j, L/2) é a solução, e:

$i' = 2i,\ j' = 2j,\ f' = f - incx/2 - incy/2,\ incx' = incx/2,\ incy' = incy/2$

O procedimento que se segue é uma simples codificação deste critério:

```
PROCEDURE Detalhamento (
    sextupla:      DadosSextupla     (* i, j, L, f, incx e incy *)
)   :
    DadosSextupla ;                  (* Sextupla na celula integrada *)

    BEGIN
        WITH sextupla DO
            L := L / 2 ;
            incx := incx / 2 ;
            incy := incy / 2 ;
            IF (f – incx – 2 * incy) >= 0 THEN
                i := 2*i + 1 ;
                j := 2*j ;
                f := f – incy
            ELSE
                IF (f – 2 * incx – incy) <= 0 THEN
                    i := 2*i ;
                    j := 2*j + 1 ;
                    f := f – incx ;
                ELSE
                    i := 2*i ;
                    j := 2*j ;
                    f := f – incx –incy ;
            END ;
        END
        RETURN sextupla ;
    END Detalhamento ;
```

O processo de Integração é inverso ao de Detalhamento, e por conta disto pode ser escrito quase que simplesmente pela execução na ordem inversa das operações inversas as do Detalhamento. Para identificarmos qual dos 3 casos de Detalhamento devemos inverter basta testarmos a paridade de i e j. O pseudocódigo do algoritmo de Integração é o seguinte:

```
PROCEDURE Integração (
    sextupla:      DadosSextupla     (* i, j, L, f, incx e incy *)
)   :
    DadosSextupla ;                  (* Sextupla na celula integrada *)

    BEGIN
        WITH sextupla DO
            IF Par (i) THEN
                f := f + incx
            END ;
            IF Par (j) THEN
                f := f + incy ;
            END ;
            L := 2*L ;
            incx := incx * 2 ;
            incy := incy * 2 ;
```

```
            i := TRUNC (i/2) ;
            j := TRUNC (j/2) ;
        END ;
        RETURN sextupla ;
    END Integracao ;
```

O procedimento ProgressãoCelularQuadtree, redigido em sequência, inicia pela determinação da primeira célula folha da quadtree Q interceptada pelo raio emanando de (x0,y0) na direção (dx, dy). Para esta primeira célula é calculada a sextupla em E. Visitada esta célula o algoritmo progride em um ciclo iterativo.

No ciclo, executado enquanto a progressão não escapa ao espaço de Q, se a célula é uma folha de Q, então ela é visitada e passa-se à seguinte na progressão. Senão, antes da progressão, promove-se uma série de integrações até que ela seja um nó de Q, ou promove-se uma série de detalhamentos para que se torne uma folha de Q. Ao fim dessa sequência, a invariante é estabelecida, a célula pode ser visitada e nova progressão pode se dar.

```
PROCEDURE ProgressaoCelularQuadtree (
    Q       (* Quadtree *)
    x0,     (* Coordenadas do Ponto Inicial *)
    y0,
    dx,     (* Direcao do raio *)
    dy
)

    BEGIN
        sextupla := PrimeiraCelula (Q, x0, y0, dx, dy) ;
        IF (L <= 0) AND (L > 2^n) THEN EXIT END ;
        Visita (i, j) ;
        sextupla := Progressao (sextupla) ;
        WHILE (i.L < 2^n) AND (j.l < 2^n) DO
            (* Enquanto no espaço de Q *)
            WHILE C(i, j, L) ∉ Q DO
                sextupla := Integracao (sextupla) ;
            END ;
            WHILE C(i, j, L) ∉ Folha(Q) DO
                sextupla := Detalhamento (sextupla) ;
            END ;
            (* Invariante:     CijL é interceptada pelo raio
                               CijL é folha de Q
                               sextupla ∈ E *)
            VisitaCelula (i, j) ;
            sextupla := Progressao (sextupla) ;
        END ;
    END ProgressaoCelularQuadtree ;
```

A determinação da Primeira Célula inicia-se testando se a origem do raio (x0, y0) está dentro do espaço de Q. No caso afirmativo, a célula C(x0, y0, 1) seguramente será interceptada

pelo raio e estará dentro do espaço da Quadtree (e os parâmetros da sextupla podem ser facilmente avaliados). Para chegar-se a uma folha de Q promove-se uma série de integrações. No caso de (x0, y0) não pertencer ao espaço de Q, determina-se a sextupla para a célula C(0, 0, 2^n), que é todo espaço de Q, e ali busca-se uma folha, por meio de detalhamentos. No evento do espaço de Q não ser interceptado pelo raio, uma sextupla nula é retornada.

```
PROCEDURE PrimeiraCelula (
    Q       (* Quadtree *)
    x0,     (* Coordenadas do Ponto Inicial *)
    y0,
    dx,     (* Direcao do Raio *)
    dy
)
    BEGIN
        L = 2^n ;  (* Largura do espaço de Q *)
        IF (x0 >= 0) AND (y0 >= 0) THEN
            IF (x0 <= L) AND (y0 <= L) THEN
                WITH sextupla DO
                    i := x0 ;
                    j := y0 ;
                    f := -dy + dx ;
                    incx := dy ;
                    incy := dx ;
                    L := 1 ;
                END ;
            (* Estabelecem parametros para C(x0, y0, 1 *)
                WHILE C(i, j, L) ∉ Q DO
                    sextupla := Integracao (sextupla) ;
                END ;
            ELSE
                RETURN sextuplanula ;
        ELSE
            WITH sextupla DO
                i := 0 ;
                j := 0 ;
                f := x0 * dy - yo * dx ;
                incx := -L * dy ;
                incy := L * dx ;
            END ;
            (* Estabelecem parametros para C(0, 0, 2^n), o espaço de Q *)
            IF (f - incx >= 0) AND (f - incy <= 0) THEN
                WHILE C(i, j, L) ∉ Folha(Q) DO
                    sextupla := Detalhamento (sextupla) ;
                END
            ELSE
                RETURN sextuplanula ;
            END
        RETURN sextupla ;
    END PrimeriaCelula ;
```

VII.4 – O ALGORITMO DE PROGRESSÃO CELULAR EM OCTREES

Nesta seção iremos discutir os fundamentos da extensão do algoritmo de Progressão Celular de Quadtrees para Octrees. Muitos dos conceitos apresentados pelo algoritmo 2D serão utilizados, pois como veremos mais adiante o projeto do algoritmo 3D será baseado em dois algoritmos 2D. Este fato é importante, pois o corpo do algoritmo de Progressão Celular tanto para Quadtrees como para Octrees não sofrerá alterações. As modificações serão efetuadas somente sobre o processo de iniciação e os procedimentos invocados.

Uma reta S no R^3 pode ser expressa algebricamente por duas equações reais da forma:

$$F(x, y) = 0$$

$$G(y, z) = 0$$

Por exemplo, a reta que passa por (x1, y1, z1) e (x2, y2, z2) pode ser expressa de diversas formas. Em particular por:

$$F(x, y) = (y - y1)(x2 - x1) - (x - x1)(y2 - y1) = 0$$

$$G(y, z) = (z - z1)(y2 - y1) - (y - y1)(z2 - z1) = 0$$

Essa alternativa foi escolhida por conveniência, pois uma combinação linear simples de F e G torna–se independente de y. Assim podemos enunciar a seguinte propriedade:

Propriedade 3: A função H definida da seguinte forma:

$$H(z, x) = (x - x1)(z2 - z1) - (z - z1)(x2 - x1)$$

satisfaz:

$$\mathbf{H(z, x) = -(z2 - z1)\, F(x, y) - (x2 - x1)\, G(y, z)}$$

Uma conclusão imediata da Propriedade 3 é que a reta S fica caracterizada por três equações não independentes:

$$F(x, y) = 0$$

$$G(y, z) = 0$$

$$H(z, x) = 0$$

Podemos interpretar geometricamente estes resultados. Cada equação acima exprime a equação da reta S projetada sobre os planos cartesianos aos quais elas se referem. A figura VII.6 ilustra esta interpretação.

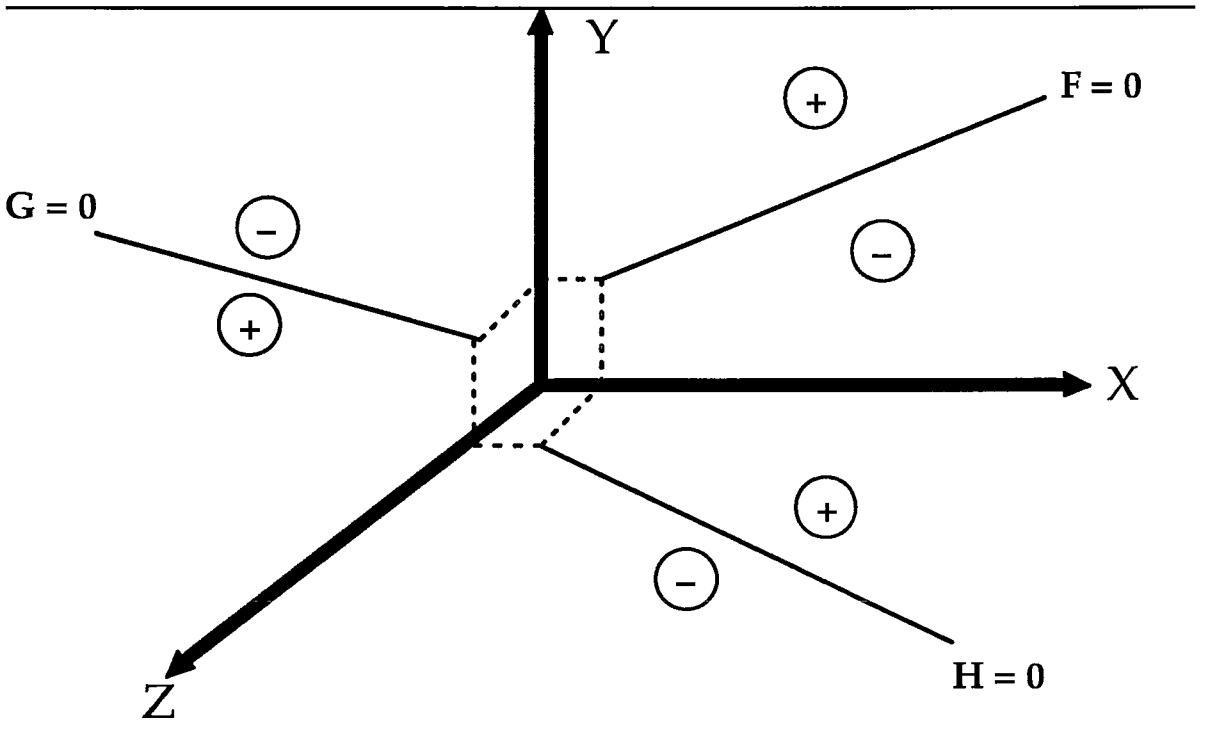


FIGURA VII.6 – Interpretação Geométrica de F, G e H

Da mesma forma que no algoritmo 2D (ver Condição 1), assumimos por simplicidade que os coeficientes angulares dessas projeções são positivos, o que nos leva a:

Condição 2: $(x2 > x1)$, $(y2 > y1)$ e $(z2 > z1)$

Esta condição é garantida por:

Condição 2a:

(1) $F(x + 1, y) < F(x, y) < F(x, y + 1)$

(2) $G(y + 1, z) < G(y, z) < G(y, z + 1)$

(3) $H(z + 1, x) < H(z, x) < H(z, x + 1)$

Na figura VII.6 foram registrados os sinais dos valores de F, G e H nos planos coordenados. Estabelecidas estas condições podemos dar o conceito de célula 3D:

Definição 5 – Célula 3D: Uma célula pertencente a uma subdivisão do espaço é cúbica e é delimitada da seguinte forma:

$$C(i, j, k, L) = \{(x, y, z) \mid i.L <= x < (i+1).L,\ j.L <= y < (j+1).L,\ k.L <= z <= (k+1).L\}$$

onde:

i, j, k referem-se aos índices da célula

L refere-se ao tamanho do lado da célula

A figura VII.7 ilustra o conceito de célula.

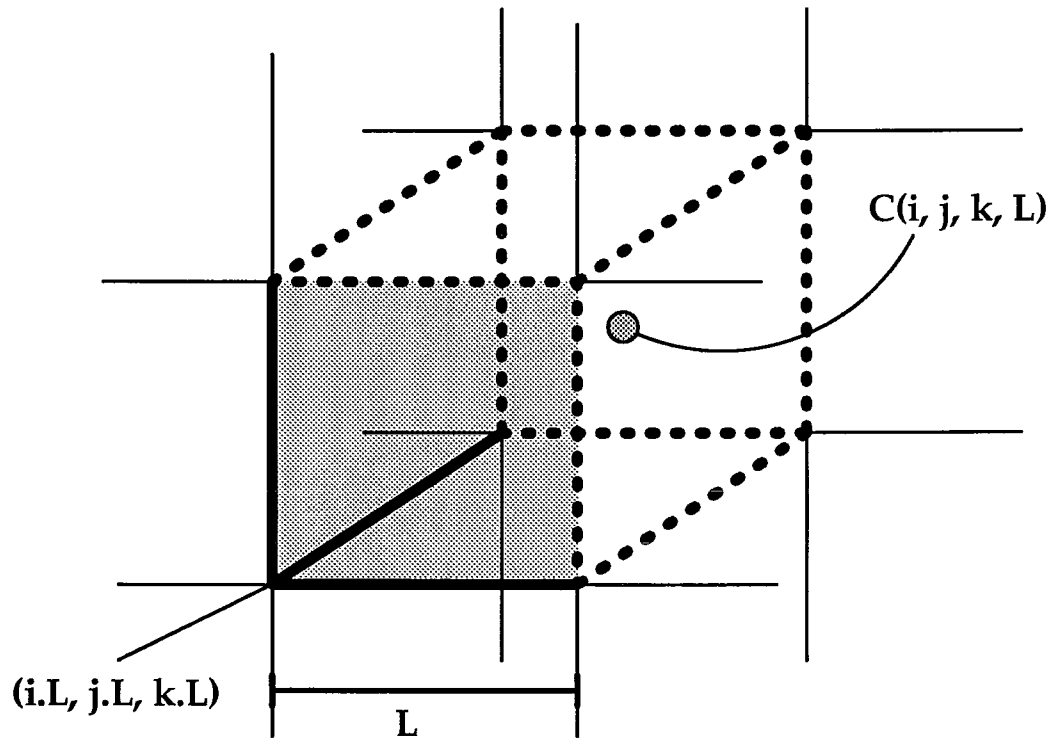


FIGURA VII.7 – Exemplo de célula cúbica

Como consequência da continuidade de F, G e H, e das definições e propriedades enunciadas anteriormente podemos afirmar que:

Propriedade 4: Uma célula C(i, j, k, L) é interceptada por uma reta S se e somente se cada projeção de S interceptar a respectiva projeção da célula, isto é:

(1) $F_{i\ j+1} > 0$

(2) $F_{i+1\ j} < 0$

(3) $G_{j\ k+1} > 0$

(4) $G_{j+1\ k} < 0$

(5) $H_{k\ i+1} > 0$

(6) $H_{k+1\ i} < 0$

onde $F_{i\,j} = F(i.L, j.L)$, $G_{j\,k} = G(j.L, k.L)$ e $H_{k\,i} = H(k.L, i.L)$

A Propriedade 4 é similar à Propriedade 1, que estabelecia as condições para interseção no caso 2D, guardadas as devidas proporções. Apresentadas as condições de interseção de uma reta com uma célula cúbica C(i, j, k, L), podemos estabelecer as condições para que uma de suas vizinhas seja interceptada. No caso 2D a célula vizinha poderia ser uma entre 3 células, referente aos dois casos de vizinhos de aresta e um caso de vizinho de vértice. No caso 3D a célula vizinha pode ser uma entre 7 células, referentes a três vizinhos de face, três vizinhos de aresta e um vizinho de vértice.

Vamos inferir o critério de progressão 3D. Para tanto, inicialmente vamos analisar um caso de transição, e avaliar as condições necessárias. Por avaliações similiares são deduzidos todos os outros casos. Logo após, o critério de progressão será enunciado.

Supondo C(i, j, L) interceptada por S, a partir da Propriedade 4 podemos estabelecer que C(i+1, j, k, L) é interceptada se:

(1) $F_{i+1\ j+1} > 0$

(2) $F_{i+2\ j} < 0$

(3) $G_{j\ k+1} > 0$

(4) $G_{j+1\ k} < 0$

(5) $H_{k\ i+2} > 0$

(6) $H_{k+1\ i+1} < 0$

As propriedades (3) e (4) são satisfeitas imediatamente aplicando a Propriedade 4 a C(i,j, k, L), enquanto que as propriedades (2) e (5) são decorrentes das Propriedades 3a.1 e 3a.3 respectivamente.

Portanto, para garantirmos que C(i + 1, j, k) também seja interceptada basta estabelecermos as condições (1) e (6).

Por dedução similar podemos estabelecer as condições para as demais possibilidades de células vizinhas.

Critério de Progressão 3D: Dada uma célula C(i, j, k, L) que faz interseção com a segmento de reta S, então a célula seguinte interceptada será uma entre as seguintes, de acordo com as condições estabelecidas:

- C (i + 1, j, k, L) é a próxima se:

  $F_{i+1\ j+1} > 0$ e

  $H_{k+1\ i+1} < 0$

- C (i, j + 1, k, L) é a próxima se:

  $G_{j+1\ k+1} > 0$ e

  $F_{i+1\ j+1} < 0$

- C (i, j, k + 1, L) é a próxima se:

  $H_{k+1\ i+1} > 0$ e

  $G_{j+1\ k+1} < 0$

- C (i + 1, j + 1, k, L) é a próxima se:

  $F_{i+1\ j+1} = 0$ e

  $G_{j+1\ k+1} > 0$ e

  $H_{k+1\ i+1} < 0$

- C (i, j+1, k+1, L) é a próxima se:

  $G_{j+1\ k+1} = 0$ e

  $H_{k+1\ i+1} > 0$ e

  $F_{i+1\ j+1} < 0$

- C (i + 1, j, k + 1, L) é a próxima se:

  $H_{k+1\ i+1} = 0$ e

  $F_{i+1\ j+1} > 0$ e

  $G_{j+1\ k+1} < 0$

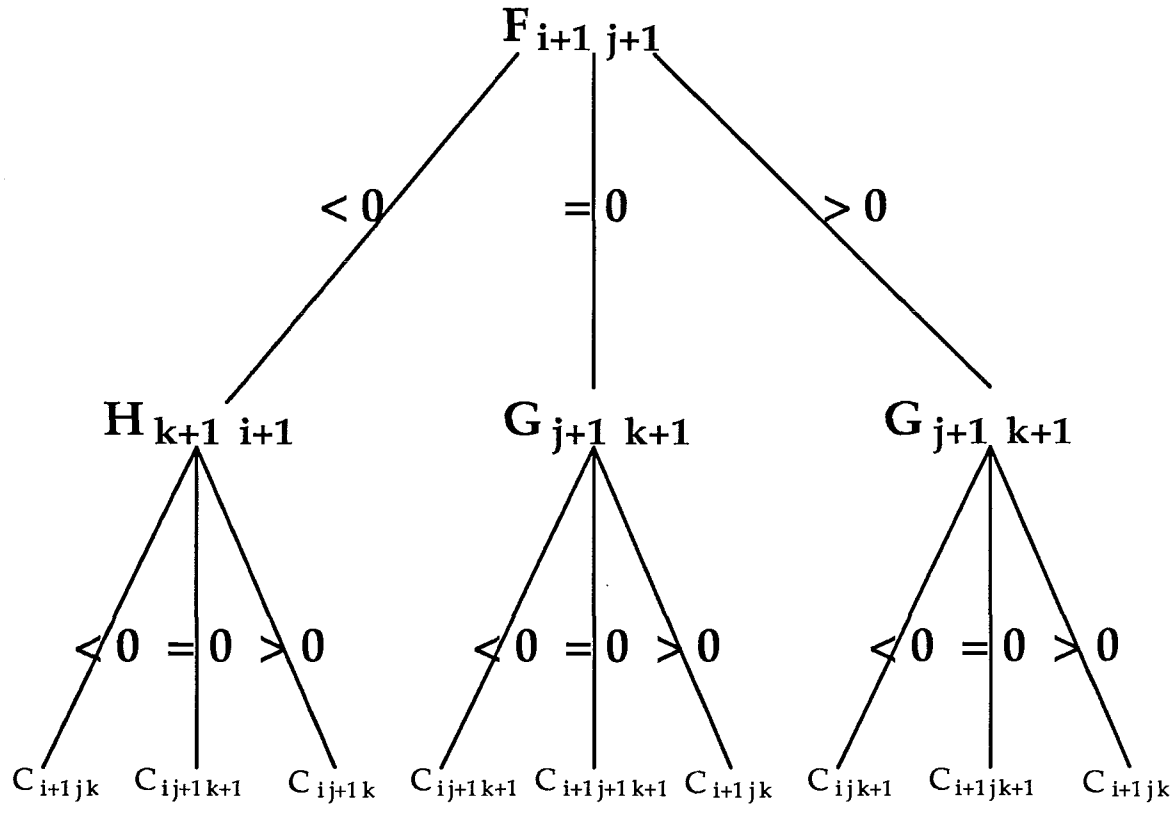
- C (i + 1, j + 1, k + 1, L) é a próxima se:

  $F_{i+1\ j+1} = 0$ e

  $G_{j+1\ k+1} = 0$ e

$$H_{k+1\ i+1} = 0$$

Baseados no Critério de Progressão 3D podemos montar uma árvore de decisão para auxiliar no estabelecimento das transições de progressões válidas, que será útil quando do projeto do algoritmo. Muitos casos impossíveis foram agregados no sentido de simplificá-la. Por exemplo, em virtude da Propriedade 3 a condição $F>0, G>0$ e $H>0$ nunca será verificada, mas na árvore ela foi agregada à progressão de C(i+1, j, k, L). Simplificando a árvore, obtemos nove casos possíveis, que nos indicam uma entre as sete células possíveis de ser a próxima na progressão. Assim, em mais de um caso podemos concluir que uma determinada célula seja a próxima. Esta árvore encontra-se ilustrada na figura VII.8.

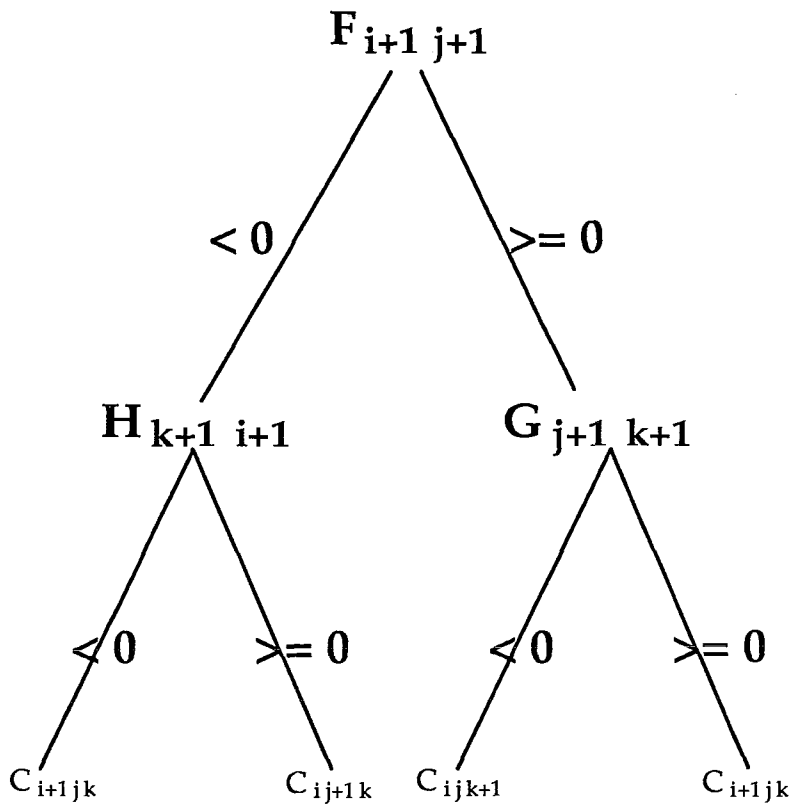


Notação: C_{ijk} refere-se a célula C(i, j, k, L)

FIGURA VII.8 – Árvore de Decisão referente ao Critério de Progressão 3D

Esta árvore admite não só transições para vizinhos de face, mas também para células vizinhas de aresta, ou mesmo de vértice. Transições diagonais deste tipo podem ser obtidas por duas ou três transições por face. Na nossa aplicação isto apenas aumentaria o número de testes desnecessários. Já que a probabilidade do acontecimento de vizinhos de aresta e vértice é muito inferior à probabilidade do acontecimento de vizinhos de face, resolvemos simplificar o algoritmo de forma a nos restringirmos somente a vizinhos de face.

Com esta simplificação a árvore de decisão deve também ser alterada. A nova versão desta, onde somente temos transições por face, é representada na figura VII.9.



Notação: C_{ijk} refere-se a célula C(i, j, k, L)

FIGURA VII.9 – Árvore de Decisão Simplificada

Usando a árvore de decisão simplificada, podemos então elaborar o algoritmo de progressão 3D, que possui o seguinte pseudocódigo:

```
PROCEDURE Progressão3D (
    VAR i, j, k, L : INTEGER
)
    BEGIN
        IF F ((i+1).L, (j+1).L) < 0 THEN
            IF H((k+1).L, (i+1).L) < 0 THEN
                i := i + 1 ;
            ELSE
                j := j + 1;
        ELSE
            IF G((j+1).L, (k+1).L) < 0 THEN
                k := k + 1
            ELSE
                i := i + 1 ;
            END
```

```
        END
    END Progressao3D ;
```

Este algoritmo não se utiliza do cálculo incremental, tornando necessária a avaliação das funções F, G e H. À semelhança do caso 2D, podemos calcular os valores das funções F, G, H de forma incremental, estabelecendo uma invariante similar. Como (pela Propriedade 3) H é linearmente dependente de F e G, bastaria calcularmos F e G.

Antes de determinarmos a invariante, é necessário apresentarmos a seguinte definição:

Definição 6 – Conjunto I: Seja I o conjunto de todas as 9–uplas (i, j, k, L, f, incx, incy, incz) tais que valem as seguintes propriedades:

(a) i, j, k, L são inteiros

(b) $f = F((i+1).L, (j+1).L)$

(c) $g = G((j+1).L, (k+1).L)$

(d) $incx = F(L, 0) - F(0, 0) = -L\ (y2 - y1)$

(e) $incy = F(0, L) - F(0, 0) = L\ (x2 - x1)$

(f) $incz = G(L, 0) - G(0, 0) = -L\ (z2 - z1)$

De forma análoga ao caso 2D podemos estabelecer a seguinte propriedade:

Propriedade 5: Se a 9–upla (i, j, k, L, f, g, incx, incy, incz) $\in$ I então:

(1) (i+1, j, k, L, f+incx, g, incx, incy, incz) $\in$ I

(2) (i, j+1, k, L, f+incy, g+incy, incx, incy, incz) $\in$ I

(3) (i, j, k+1, L, f, g+incz, incx, incy, incz) $\in$ I

A prova desta propriedade é similar à prova da propriedade 2. Assim, estabelecemos como invariante do algoritmo a condição da 9–upla (i, j, k, f, g, incx, incy, incz) $\in$ I. Para garantir a validade da invariante aplicaremos alterações somente sobre a 9–upla que satisfaçam a Propriedade 5. O pseudocódigo do algoritmo de Progressão fica reescrito da seguinte forma:

```
PROCEDURE Progressão3D (
    NoveUpla : DadosNoveUpla ;
)   :
```

```
NoveUpla ;

BEGIN
    WITH NoveUpla DO
        IF f < 0 THEN
            IF (–dz.f – dx.g) < 0 THEN (* H = –dz.F – dx.G *)
                i := i + 1 ;
                f := f + incx ;
            ELSE
                j := j + 1;
                f := f + incy ;
                g := g + incy ;
        ELSE
            IF g < 0 THEN
                k := k + 1 ;
                g := g + incz ;
            ELSE
                i := i + 1 ;
                f := f + incx ;
            END
        END
    END
    RETURN NoveUpla ;
END Progressao3D ;
```

No caso de lidarmos com Octrees devemos também realizar operações de Detalhamento e Integração. Como o algoritmo trabalha com as funções F e G que são projeções da reta sobre os planos coordenados, ao realizar uma transição de face estaremos aplicando uma transição sobre uma destas funções. Portanto, basta executar os procedimentos correspondentes ao caso 2D sobre F e G.

Para obtenção do algoritmo de ProgressãoCelulaOctree basta usar o corpo do algoritmo de ProgressãoCelularQuadtree, substituindo as rotinas ligadas ao tratamento de Quadtrees pelas rotinas de tratamento de Octrees.

VII.5 – GENERALIZAÇÃO PARA QUALQUER TIPO DE RETA

A apresentação das idéias do algortimo feita até aqui parte do pressuposto que a reta possui coeficiente angular positivo. Este pode não ser o caso na maioria das vezes, pois a existência de coeficientes angulares negativos ou nulos é muito freqüente.

O uso de retas com coeficientes angular de qualquer valor é feito aplicando–se uma função de mapeamento, que faz com que qualquer tipo de reta seja tratada como se tivesse

coeficiente angular positivo. Esta função faz com que os octantes reais sejam levados em octantes lógicos dentro de um espaço Octree lógico, onde as retas possuem coeficiente angular do sinal desejado. A função de mapeamento tem como argumentos os sinais dos coeficientes angulares dos eixos X, Y e Z respectivamente, e estabelece um relacionamento 1x1 entre o octante real e o octante lógico.

Octante Real / Inclinações	0	1	2	3	4	5	6	7
Positiva Positiva Positiva	0	1	2	3	4	5	6	7
Negativa Positiva Positiva	1	0	3	2	5	4	7	6
Positiva Negativa Positiva	2	3	0	1	6	7	4	5
Positiva Positiva Negativa	4	5	6	7	0	1	2	3
Negativa Negativa Positiva	3	2	1	0	7	6	5	4
Negativa Positiva Negativa	5	4	7	6	1	0	3	2
Positiva Negativa Negativa	6	7	4	5	2	3	0	1
Negativa Negativa Negativa	7	6	5	4	3	2	1	0

TABELA VII.1 – Função de Mapeamento

O algoritmo irá trabalhar sobre um octante lógico, mas ao acessar a estrutura de dados o octante real será utilizado (obtido através da função de mapeamento).

Caso a reta possua coeficiente angular nulo, uma solução seria aplicar uma perturbação, de forma a eliminar o incômodo paralelismo. Outra alternativa seria a implementação de uma rotina específica para enumerar as células pelas quais uma reta paralela a um dos eixos passa.

VII.6 – ESPECIFICAÇÃO INTEIRA DO RAIO

A utilização do algoritmo de Progressão Celular não é imediata. O principal problema é que as retas com que iremos lidar podem possuir origem e direção que não possam ser expressas através de números racionais.

O problema da precisão ocorre principalmente nos raios refratados e refletidos. Este problema pode ser contornado para o raio inicial, onde podemos assumir que na especificação do sistema de projeção o observador situa-se em uma posição descrita por números racionais. Entretanto, o mesmo não acontece nos raios refletidos e refratados, que podem ter origem em qualquer lugar do espaço.

A solução adotada consiste no acompanhamento do raio sobre um novo espaço octree, não mais o espaço real, mas um espaço inteiro onde os números reais são aproximados por números inteiros. Agora, todo o algoritmo de Progressão Celular será aplicado sobre um espaço inteiro, assim como os raios serão descritos neste sistema. Para os cálculos de interseções e demais operações que manipulam com reais o espaço real será utilizado (Figura VII.10).

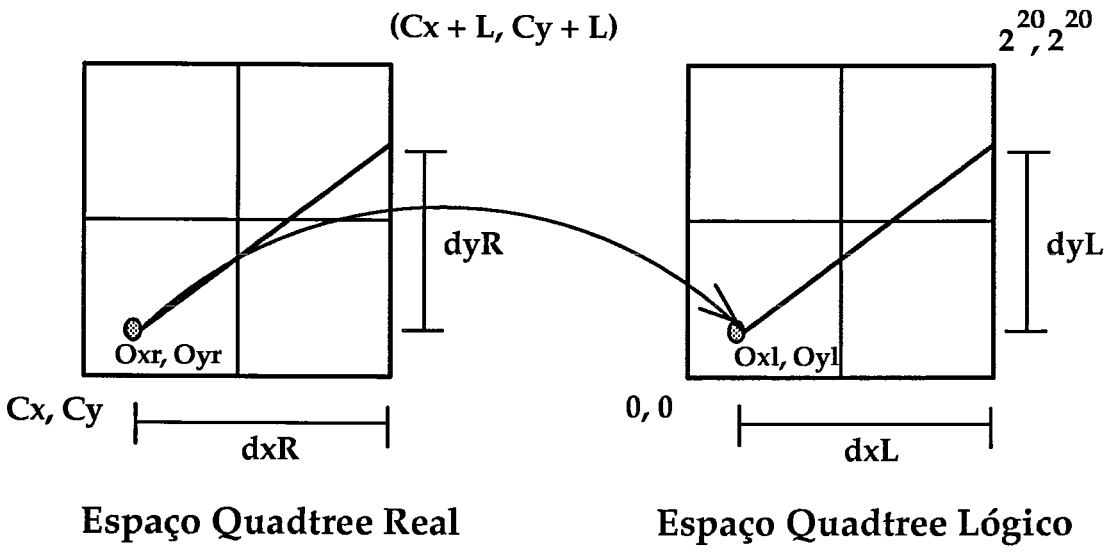


FIGURA VII.10 – Utilização do Espaço Lógico

Capítulo VIII

Avaliações e Conclusões

VIII.1 – INTRODUÇÃO

Neste capítulo iremos analisar os resultados obtidos com a implementação do algoritmo de acompanhamento de raios para sólidos CSG. Foram implementadas as seguintes versões do algoritmo:

- Acompanhamento de Raios Canônico: Refere–se ao algoritmo de acompanhamento de raios na sua versão simples, onde todas as primitivas são testadas para verificação de interseção com o raio.

- Acompanhamento de Raios sobre a OctreeCSG (Primeira Variante): Refere–se ao algoritmo de acompanhamento de raios otimizado pela busca proposta por Glassner.

- Acompanhamento de Raios sobre a OctreeCSG (Segunda Variante): Refere–se ao algoritmo de acompanhamento de raios otimizado pela busca de Progressão Celular.

Primeiramente os resultados obtidos com dois testes serão mostrados. A seguir estes resultados serão comentados de forma a avaliar os novos métodos propostos. Ao final do capítulo faremos uma breve conclusão das experiências e resultados obtidos, propondo possíveis melhoramentos e trabalhos futuros.

VIII.2 – RESULTADOS DOS TESTES

São os seguintes os resultados aplicados a duas cenas:

- Cena 1:
  - Número de Primitivas: 84 (2 icosaedros, 2 octaedros, 2 dodecaedros, 2 hexaedros e 2 blocos – No total 80 planos e 4 blocos)
  - Resolução da Imagem: 300 x 300
  - Número de Raios Incidentes: 90000
  - Número de Raios Refratados: 0
  - Número de Raios Refletidos: 19861
  - Execuções (vide tabela VIII.1)
- Cena 2:
  - Número de Primitivas: 18 (16 esferas e 2 blocos)
  - Resolução da Imagem: 300 x 300
  - Número de Raios Incidentes: 90000
  - Número de Raios Refratados: 0
  - Número de Raios Refletidos: 18258
  - Execuções (vide tabela VIII.2)

Nomenclatura:

Variante: : Tipo de algoritmo utilizado

Profundidade da Octree : Profundidade máxima para geração da OCSG

Cálc. Parâmetro da Reta : Número de cálculos do parâmetro da reta

Interseções : Número de interseções primitiva x raio computadas

Vizinhos : Contabilização dos tipos de vizinhos (Face, Aresta e Vértice)

Tempo

Geração OCSG : Tempo de geração da OctreeCSG

Acomp. Raios : Tempo do algoritmo de Acompanhamento de Raios

Variante	Prof Octree	# Cálculos do Parâmetro da Reta	# Interseções	# Vizinhos			Tempo	
				Face	Aresta	Vértice	Geração OCSG	Acomp. Raios
Canônico	–	–	5509793	–	–	–	–	40m22.0s
Otimizado Glassner	1	583192	3487304	407826	922	1	1.3s	34m41.1s
Otimizado Prog. Cel.	1	486627	3492300	–	–	–	1.3s	28m45.6s
Otimizado Glassner	2	830164	2186321	654494	1026	1	2.0s	19m2.5s
Otimizado Prog. Cel.	2	703563	2188536	–	–	–	2.0s	20m44.1s
Otimizado Prog. Cel.	3	875218	1395122	–	–	–	2.5s	13m57.1s

TABELA VIII.1 – Execuções do Teste1

Variante	Prof Octree	# Cálculos do Parâmetro da Reta	# Interseções	# Vizinhos			Tempo	
				Face	Aresta	Vértice	Geração OCSG	Acomp. Raios
Canônico	–	–	2943000	–	–	–	–	13m25.0s
Otimizado Glassner	1	552194	887007	387790	902	2	1.2s	5m9.2s
Otimizado Prog. Cel.	1	454254	888195	–	–	–	1.2s	5m26.1s
Otimizado Glassner	2	817129	778674	652661	967	1	1.3s	5m56.7s
Otimizado Prog. Cel.	2	692717	779282	–	–	–	1.3s	6m2.3s
Otimizado Glassner	3	601464	914213	437118	844	2	1.5s	6m6.1s
Otimizado Prog. Cel.	3	564364	914589	–	–	–	1.5s	6m0.4s

TABELA VIII.2 – Execuções do Teste2

VIII.3 – AVALIAÇÕES DOS TESTES

Os resultados apresentados na seção anterior permitiram chegar as seguintes avaliações:

- O algoritmo otimizado obteve larga vantagem em relação ao canônico.

- O tempo de geração da árvore híbrida entre Octree e CSG é satisfatório, viabilizando ainda mais a performance do algoritmo otimizado.

- A Octree CSG revela–se de fundamental importância quando formos aplicar o algoritmo otimizado. A escolha do correto posicionamento desta no espaço, de forma a separar ao máximo as primitivas em voxels distintos, é uma tarefa de difícil realização. A utilização de métodos empíricos se faz necessária, pois embora não levem em geral ao particionamento ótimo, podem produzir resultados razoáveis com pouca demanda de tempo.

- A variação da profundidade máxima da Octree–CSG leva a resultados diferentes de tempo. Este fato é decorrente de como a subdivisão conseguiu particionar o espaço. Na maioria das vezes é interessante termos uma Octree mais profunda possível, se pudermos assegurar que a cada subdivisão há mais e mais separação das primitivas em voxels distintos. Este é o caso do teste 1, onde temos resultados melhores a medida que aumentamos a profundidade da Octree. Entretanto, as vezes não é interessante continuar a subdivisão, pois existe a possibilidade que as árvores de todos subvoxels de um determinado voxel V sejam iguais. Este fato não seria problema se tivessemos um algoritmo de condensação (o que não é o caso). Por isto, no teste 2 temos um acréscimo de tempo a medida que aumentamos a profundidade da Octree. Para concluir, vale dizer que não é necessário termos Octrees de profundidade muito grande. Estabelecer que a profundidade máxima da Octree seria 5 não é nenhum absurdo, visto que bons resultados são obtidos com Octrees até este limite.

- A comparação entre o algoritmo de Progressão Celular e a abordagem de GLASSNER revela–se interessante. Na maioria das vezes os resultados são equivalentes, mas em certos casos existe grande vantagem para o algoritmo de Progressão Celular (conforme o teste 1). Conjectura–se que o algoritmo de Progressão Celular é mais rápido quando o número de células vazia a ser ultrapassado na Octree é grande, pois nestes casos o cálculo do parâmetro da reta é efetuado somente no algoritmo de GLASSNER, enquanto que no algoritmo de Progressão Celular somente cálculos inteiros são realizados. Em células que contém primitivas é preciso calcular para ambos os algoritmos o intervalo do parâmetro referente a entrada e saída do segmento de reta dentro do voxel, de forma a restringir a interseções dentro do voxel. Por ourto lado, se o número de células vazias for pequeno o algoritmo de Glassner é mais vantajoso, pois tende a visitar menos nodos que o algoritmo de Progressão Celular,

porque gera vizinhos de aresta, vértice e face, enquanto que o algoritmo de Progressão Celular transita somente por vizinhos de face.

VIII.4 – CONCLUSÕES FINAIS

A Implementação de um algoritmo de condensação revela–se importante como forma de aperfeiçoar o processo de subdivisão executado. A dificuldade principal neste processo consiste na verificação da igualdade entre árvores. Caso todas as árvores filhas de um nodo N forem iguais, podemos realizar uma condensação, associando somente uma árvore ao nodo N, tornando este folha. A maneira pensada para realizar este tipo de teste seria representar uma árvore por um rótulo, de forma a simplificar a operação de comparação. Esta abordagem seria interessante também pelo ponto de vista de memória. Usando este algoritmo de condensação, resultados ainda melhores poderão ser obtidos.

A experiência com o uso de estruturas híbridas foi satisfatória. Estas estruturas tornam–se a cada dia mais importantes no campo da modelagem de sólidos. Podemos sugerir como trabalho futuro o uso de Octrees aliadas a B–Rep. Deve–se definir um novo algoritmo para subdividir B–Rep em Octrees, e implementar uma rotina que calcule a interseção de raios contra B–Rep. Feito isto, basta acoplar estes módulos ao presente trabalho, e teremos um algoritmo otimizado para acompanhamento de raios em sólidos B–Rep.

O desenvolvimento teórico do algoritmo de Progressão Celular permitiu a criação de um algoritmo extremamente simples como variante para a solução do problema de acompanhamento de raios sobre uma Octree. O potencial de algoritmos deste estilo (incrementais e inteiros) foi um estímulo para o desenvolvimento do mesmo. Estima–se que tal algoritmo possa ser de grande utilidade no futuro em outras aplicações.

Como resultado prático desta tese, temos a implementação de um algoritmo de acompanhamento de raios que nos permite gerar muitas imagens chamadas realistas. Este algoritmo tornou–se viável com a implementação de uma otimização ao método, que reduz sensivelmente os tempos de execução do algoritmo.

Referências Bibliográficas

AMANATIDES, JOHN (1984). ***Ray Tracing with Cones***. ACM Computer Graphics, Vol 18, Julho 1984.

AMANATIDES, JOHN E WOO, ANDREW (1987). ***A Fast Voxel Traversal Algorithm for Ray Tracing***. Anais do Eurographics 1987.

APPEL, ARTHUR (1968). ***Some Techniques for Shading Machine Rendering of Solids***. SJCC, 1968, Thompson Books, Washington, D. C.

CHEN, HOMER H. E HUANG, THOMAS S (1988). ***A Survey of Construction and Manipulation of Octrees***. Computer Vision, Graphics and Image Processing. Nro 43 (1988).

COMBA, JOÃO L. D. ; SANTOS, CARLOS L. N. E PERSIANO, R. C. MARINHO (1989). ***Um Sistema para Manipulação e Visualização de Octrees***. Anais do IX Congresso da SBC, Uberlândia, 1989.

COMBA, JOÃO L. D. ; ESPERANÇA, CLÁUDIO E PERSIANO, R. C. MARINHO. (1990a). ***Algoritmos para Exibição de Imagens Coloridas***. Anais do III Sibgrapi, Gramado, 1990.

COMBA, JOÃO L. D. E PERSIANO, R. C. MARINHO (1990b). ***"Ray Tracing" Otimizado de Sólidos CSG usando Octrees***. Anais do III Sibgrapi, Gramado, 1990.

COOK, R. L. E TORRANCE, K. E (1981). ***A Reflectance model for Computer Graphics***. Anais do SIGGRAPH, 1981.

COQUILLART, SABINE (1985). ***An Improvement of The Ray Tracing Algorithm***. Anais do Eurographics 1985.

ESPERANÇA, CLÁUDIO E COMBA, JOÃO L. D (1990a). ***Imprimindo Imagens Coloridas em um Dispositivo Monocromático***. Anais do III Sibgrapi, Gramado, 1990.

ESPERANÇA, CLÁUDIO (1990b). ***Técnicas para Visualização Direta de Modelos CSG***. Tese de Mestrado, COPPE–UFRJ, Junho de 1990.

FIUME, EUGENE L (1989). *The Mathematical Structure of Raster Graphics.* Academic Press, 1989

FOLEY, JAMES D. ; VAN DAM, ANDRIES ; FEINER, STEVEN K. E HUGHES, JOHN, F (1990). *Computer Graphics: Principles and Practice.* Segunda Edição. Addison–Wesley, 1990.

FUJIMOTO, AKIRA ET ALL (1986). *ARTS: Accelerates Ray Tracing System.* IEEE Computer Graphics & Applications, Abril 1986.

GARGANTINI, I, ET ALL (1986). *Viewing Transformations of Voxel–Based Objects via Linear Octrees.* IEEE Computer Graphics & Applications, Outubro, 1986.

GLASSNER, ANDREW S (1984). *Space Subdivision for Fast Ray Tracing.* IEEE Computer Graphics & Applications, Outubro 1984.

GLASSNER, ANDREW S (1988). *Spacetime Ray Tracing for Animation.* IEEE Computer Graphics & Applications, Março 1988.

GLASSNER, ANDREW S (1989a). *3D Computer Graphics.* Segunda edição. Design Press, 1989.

GLASSNER, ANDREW S (1989b). *An Introduction to Ray Tracing.* Academic Press, 1989.

GLASSNER, ANDREW S (1990). *Graphics Gems.* Academic Press, 1990.

GREENBERG, D. P. E VERBECK, C. P (1984). *A Comprehensive Light–Source Description for Computer Graphics.* IEEE , 4(7) 1984.

HAINES, ERIC A. E GREENBERG, DONALD P (1986). *The Light Buffer: A Shadow Testing Accelerator.* IEEE Computer Graphics & Applications. Setembro 1986.

HEARN, DONALD E BAKER, M. PAULINE (1986). *Computer Graphics.* Prentice–Hall, 1986.

HECKBERT, PAUL S. E HANRAHAN, PAT (1984). *Beam Tracing Polygonal Objects.* ACM Computer Graphics, Vol 18, Nro 3, Julho 1984.

HOFMANN, GEORG R (1990). *Who Invented Ray Tracing ?* The Visual Computer, vol 6 pp 120–124, 1990.

JANSEN, F. W (1986). *Data Structures for Ray Tracing.* Artigo pertencente ao livro referenciado em KESSENER (1986).

KAPLAN, M. R (1987). *The Use of Spatial Coherence in Ray Tracing.* Artigo pertencente ao livro referenciado em ROGERS (1987).

KESSENER, L. R. A. ET ALL (1986). *Data Structures for Raster Graphics.* Springer–Verlag, 1986.

MANDELBROT, B (1982). *The Fractal Geometry of Nature.* W.H. Freeman, San Francisco, 1982.

MÄNTYLÄ, MARTTI (1988). *An Introduction to Solid Modeling.* Computer Science Press, 1988.

MEAGHER, DONALD (1982). *Geometric Modeling Using Octree Encoding.* Computer Graphics and Image Processing. Nro 19 (1982).

MORRIS, DAVID T (1985). *An Algorithm for Direct Display of CSG Objects by Spatial Subdivision.* Fundamental Algorithms for Computer Graphics. NATO ASI Series. Springer–Verlag, 1985.

MORTENSON, M. E (1985). *Geometric Modeling.* Jonh Wyley & Sons, 1985.

MORTENSON, M. E (1989). *Computer Graphics: An Introduction to the Mathematics and Geometry.* Industrial Press, 1989.

PENNA, MICHAEL A. E PATTERSON, RICHARD (1986). *Projective Geometry and its Applications to Computer Graphics.* Prentice–Hall, 1986.

NEWMANN, WILLIAN M. E SPROULL, ROBERT F (1981). *Principles of Interactive Computer Graphics.* McGraw–Hill, 1981.

RANGEL, ALCYR PINHEIRO (1982). *Poliedros.* Livros Técnicos e Científicos, 1982.

REQUICHA, A. A (1980). ***Representations for Rigid Solids: Theory, Methods and Systems.*** Computing Surveys, Vol 12, Nro 4, Dezembro 1980.

ROGERS, DAVID F (1985). ***Procedural Elements for Computer Graphics.*** McGraw–Hill, 1985.

ROGERS, DAVID F. E EARNSHAW, RAE A (1987). ***Techniques for Computer Graphics.*** Springer–Verlag, 1987.

ROGERS, DAVID F. E ADAMS, J. ALAN (1990). ***Mathematical Elements for Computer Graphics.*** Segunda edição. McGraw–Hill, 1990.

ROTH, SCOTT D (1982). ***Ray Casting for Modeling Solids.*** Computer Graphics and Image Processing, Nro 18, 1982.

SAMET, HANAN (1982). ***Neighbor Finding Techniques for Images Represented by Quadtrees.*** Computer Graphics and Image Processing. Nro 18 (1982).

SAMET, HANAN E TAMMINEM, MARKKU (1985). ***Bintrees, CSG Trees and Time.*** ACM Computer Graphics, Vol 19, Nro 3, 1985.

SAMET, HANAN E WEBER, ROBERT E (1988a). ***Hierarchical Data Structures and Algorithms for Computer Graphics. Part I: Fundamentals.*** IEEE Computer Graphics & Applications, Maio 1988.

SAMET, HANAN E WEBER, ROBERT E (1988b). ***Hierarchical Data Structures and Algorithms for Computer Graphics. Part II: Applications.*** IEEE Computer Graphics & Applications, Julho 1988.

SAMET, HANAN (1990a). ***The Design and Analysis of Spatial Data Structures.*** Addison–Wesley, 1990.

SAMET, HANAN (1990b). ***Applications of Spatial Data Structures.*** Addison–Wesley, 1990.

SPEER, L.R.; DE ROSE T.D. E BARSKY, B. A (1985). ***A Theorectical and Empirical Analysis of Coherent Ray Tracing.*** Computer Generated Images, Springer Verlag, 1985.

STOLFI, JORGE E SALESIN, DAVID (1990). ***Rendering CSG Models with a ZZ–Buffer***. Anais do Siggraph, 1990.

SUTHERLAND, IVAN E. ET ALL (1974). ***A Characterization of Ten Hidden–Surface Algorithms***. Computing Surveys, Vol 6, Nro 1, Março 1974.

TAMMINEN, MARKKU ET ALL (1984). ***Ray Casting and Block Model Conversion Using a Spatial Index***. Vol 16, Nro 4, Julho 1984.

TANIMOTO, STEVEN L. E JACKINS, CHRIS L (1980). ***Oct–Trees and Their Use in Representing Three–Dimensional Objects***. Computer Graphics and Image Processing, Nro 14 (1980).

THALMANN, N. M. E THALMANN, D (1985). ***Computer Animation***. Springer–Verlag, 1985.

THALMANN, N. M. E THALMANN, D (1987). ***Image Synthesis***. Springer–Verlag, 1987.

WATT, ALAN (1989). ***Fundamentals of Three Dimensional Computer Graphics***. Addison–Wesley, 1989.

WEGHORST, HANK ET ALL (1984). ***Improved Computational Methods for Ray Tracing***. ACM Transactions On Graphics, Vol 3, Nro 1, Janeiro 1984.

WHITTED, TURNER (1980). ***An Improved Ilumination Model for Shaded Display***. Communications of the ACM, Junho 1980.

WYVILL, GEOFF E KUNII, TOSIYASU L. (1985). ***A Functional Model for Constructive Solid Geometry***. The Visual Computer, Vol 1, 1985.

WYVILL, GEOFF E KUNII, TOSIYASU L (1986). ***Space Division for Ray Tracing in CSG***. IEEE Computer Graphics & Applications, Abril 1986.

YAMAGUCHI, K. ET ALL (1984). ***Octree–Related Data Structures and Algorithms***. IEEE Computer Graphics & Applications, Janeiro 1984.

Índice Remissivo

S

V