UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA

Este exemplar corresponde à redação final da tese defendida por Ari de Moura Villaça e aprovada pela Comissão Julgadora em 28/09/89.

SÍNTESE DE IMAGENS REALÍSTICAS

UMA ABORDAGEM COM A METODOLOGIA RAY-TRACING

ARI DE MOURA VILLAÇA
ORIENTADOR : PROF. DR. CLÉSIO LUÍS TOZZI

DISSERTAÇÃO APRESENTADA À FACULDADE DE ENGENHARIA ELÉTRICA DA UNIVERSIDADE ESTADUAL DE CAMPINAS - UNICAMP - COMO PARTE DOS REQUISITOS EXIGIDOS PARA OBTENÇÃO DO TÍTULO DE MESTRE EM ENGENHARIA ELÉTRICA.

AGRADECIMENTOS

A Celso Hiroshi Yamaguchi, graduando da FEE, e a Hélio Azevedo, pesquisador do CTI, pelo desenvolvimento do software básico para controle da placa gráfica utilizada no LCG / DCA,

Ao pessoal do LEA / DCA - Laboratório de Engenharia Auxiliada por Computador - pelas discussões a respeito da metodologia ray-tracing e sistemas de cores,

A Armando L. N. Delgado, doutorando da FEE, por tornar disponíveis os equipamentos do LCG / DCA.

Em especial, quero agradecer ao orientador deste trabalho, o Doutor Clésio Luiz Tozzi, pelo incentivo recebido em todas as fases do desenvolvimento desta dissertação de mestrado.

Este trabalho foi desenvolvido com o auxílio financeiro e técnico da UNICAMP, CAPES e SID INFORMÁTICA através do Projeto Estra.

RESUMO

ESTE TRABALHO DESCREVE A UTILIZAÇÃO DA METODOLOGIA RAY-TRACING COM O OBJETIVO DE SINTETIZAR IMAGENS REALÍSTICAS COM TRANSPARÊNCIAS, REFLEXOS ESPECULARES E SOMBRAS.

O CONCEITO DE RAY-TRACING E ALGORITMOS PARA SÍNTESE DE IMAGENS DE CENAS MODELADAS COM O MÉTODO CONSTRUCTIVE SOLID GEOMETRY SÃO DISCUTIDOS DETALHADAMENTE.

É APRESENTADO O AMBIENTE PARA SÍNTESE DE IMAGENS REALÍSTICAS COMPOSTO POR PROGRAMAS PARA MODELAGEM DE CENAS, SÍNTESE DE IMAGENS PROPRIAMENTE DITA, ESCOLHA DE LOOKUP-TABLES E APRESENTAÇÃO DE UMA IMAGEM NO MONITOR COLORIDO. O AMBIENTE É TOTALMENTE BASEADO NA METODOLOGIA RAY-TRACING.

ABSTRACT

This work describes the utilization of ray-tracing in the synthesis of realistic images with transparencies, specular reflections and shadows.

The concept of ray-tracing and algorithms to the synthesis of images of constructive solid geometry modeled scenes are discussed in detail.

Here is presented the realistic images synthesis environment which is formed by programs to model scenes, ray-tracing images, choice lookup-tables and image presentation.

ÍNDICE

CAP. 1- INTRODUÇÃO

A metodologia ray-tracing foi utilizada nas imagens sintetizadas em computador que causaram maior impacto até hoje devido à qualidade dos resultados obtidos, conseqüência do poder de simulação de efeitos luminosos do método. Ray-tracing é um método capaz de sintetizar imagens com reflexos especulares, transmissão de luz e sombras.

É importante também destacar que, além de simular efeitos luminosos interessantes, é inerente à metodologia ray-tracing uma grande flexibilidade no que concerne aos métodos de modelagem de cenas aos quais é possível aplicar ray-tracing. O método pode ser aplicado às cenas modeladas com malhas de polígonos, constructive solid geometry (CSG), superfícies definidas parametricamente, superfícies definidas algebricamente por polinômios, sólidos de revolução, fractais, etc.

A qualidade das imagens sintetizadas e a simplicidade conceitual de ray-tracing estimularam um grande número de pesquisadores a trabalhar com essa metodologia. As áreas de pesquisa que se destacam no que concerne ao método ray-tracing são: utilização de ray-tracing com vários métodos de modelagem de cenas, técnicas para aumento do desempenho computacional do método e a exploração do paralelismo embutido no conceito de ray-tracing.

No Laboratório de Computação Gráfica (LCG) da Faculdade de Engenharia Elétrica (FEE) da UNICAMP desenvolvemos um ambiente para síntese de imagens realísticas baseado na metodologia ray-tracing. O objetivo principal da implementação do ambiente foi criar condições básicas de pesquisa em síntese de imagens, fornecendo ao pesquisador ferramentas que possam ser modificadas e expandidas com facilidade para estudo de novas técnicas de produção de imagens em computador.

O ambiente desenvolvido é formado por um conjunto de programas com o objetivo de modelar uma cena, sintetizar imagens realísticas de uma cena, escolher a lookup-table mais indicada para uma determinada imagem e mostrar uma imagem no monitor colorido.

O desenvolvimento desse ambiente já nos permitiu comparar alguns métodos para escolha de lookup-tables para uma imagem, assunto bastante raro na literatura especializada.

O capítulo 2 desta dissertação é uma introdução à metodologia ray-tracing. Nesse capítulo o método ray-tracing é conceituado, um algoritmo ray-tracing de cenas modeladas em CSG é estudado detalhadamente e assuntos relacionados com ray-tracing como modelos de iluminação, mapeamento de texturas e aliasing são abordados sucintamente, dando uma idéia do montante de assuntos envolvidos e a complexidade da síntese de imagens realísticas com ray-tracing.

O ambiente para síntese de imagens realísticas desenvolvido no LCG é discutido no capítulo 3. As características principais do modelador de cenas, sintetizador de imagens e programas para escolha de lookup-tables são descritas. A comparação dos métodos de escolha de lookup-tables é tratada com certa ênfase nesse capítulo devido à escassez de informações relativas ao assunto na literatura.

No capítulo 4 comentamos os resultados obtidos com nosso trabalho e fazemos sugestões quanto a possíveis esforços futuros de pesquisas na área.

O capítulo 5 é dedicado à apresentação das referências bibliográficas. Fazemos um comentário a cerca do assunto principal tratado em cada referência. Esperamos que os comentários sirvam como um guia para o pesquisador ou o estudante no seu primeiro contato com o método ray-tracing.

CAP. 2- A METODOLOGIA RAY-TRACING

2.1- INTRODUÇÃO

A metodologia ray-tracing foi introduzida por Goldstein e Nagel [14], os quais a utilizaram para produzir imagens em tons de cinza de objetos opacos iniciando uma pesquisa sobre animação gráfica.

Whitted, em seu trabalho clássico [38], estendeu com sucesso a metodologia ray-tracing conseguindo simular transparências, reflexos especulares e sombras.

Desde então, a metodologia ray-tracing vem sendo exaustivamente pesquisada com resultados excelentes. De fato, esta metodologia pode ser aplicada a diversos métodos de modelagem de cenas tais como CSG, malhas de polígonos, superfícies definidas parametricamente, fractais, etc. Técnicas que aceleram os algoritmos ray-tracing, tais como subdivisão espacial e hardware especializado, também foram desenvolvidas, visando diminuir o tempo necessário para síntese de imagens através do método.

O conceito da metodologia ray-tracing é dado na seção 2.2. Um algoritmo ray-tracing para cenas modeladas com CSG visando a síntese de imagens de objetos opacos é apresentado na seção 2.3. Na seção 2.4 uma extensão ao algoritmo dado é discutida com o objetivo de simular transparências, reflexões especulares e sombras. Na seção 2.5 apresentam-se algumas técnicas que melhoram o desempenho de algoritmos ray-tracing.

2.2- O CONCEITO DE RAY-TRACING

2.2.1- UM MODELO DE CÂMERA FOTOGRÁFICA

A síntese de imagens realísticas em computador simula o processo fotográfico para obter imagens de uma cena.

O modelo de câmera fotográfica utilizado em Computação Gráfica é uma simplificação das câmeras reais, porém bastante eficiente para ser utilizado em computadores.

Dada uma tela, onde é gravada a imagem da cena, e a posição do olho do observador como ponto focal, o modelo de câmera pode ser estabelecido. A partir do olho do observador saem planos que passam pelos lados da tela formando uma pirâmide no espaço. A câmera grava apenas a imagem dos objetos que estejam no interior da pirâmide e à frente da tela, no lado oposto à posição do olho do observador. A figura 2.1 mostra o modelo de câmera utilizado em Computação Gráfica.

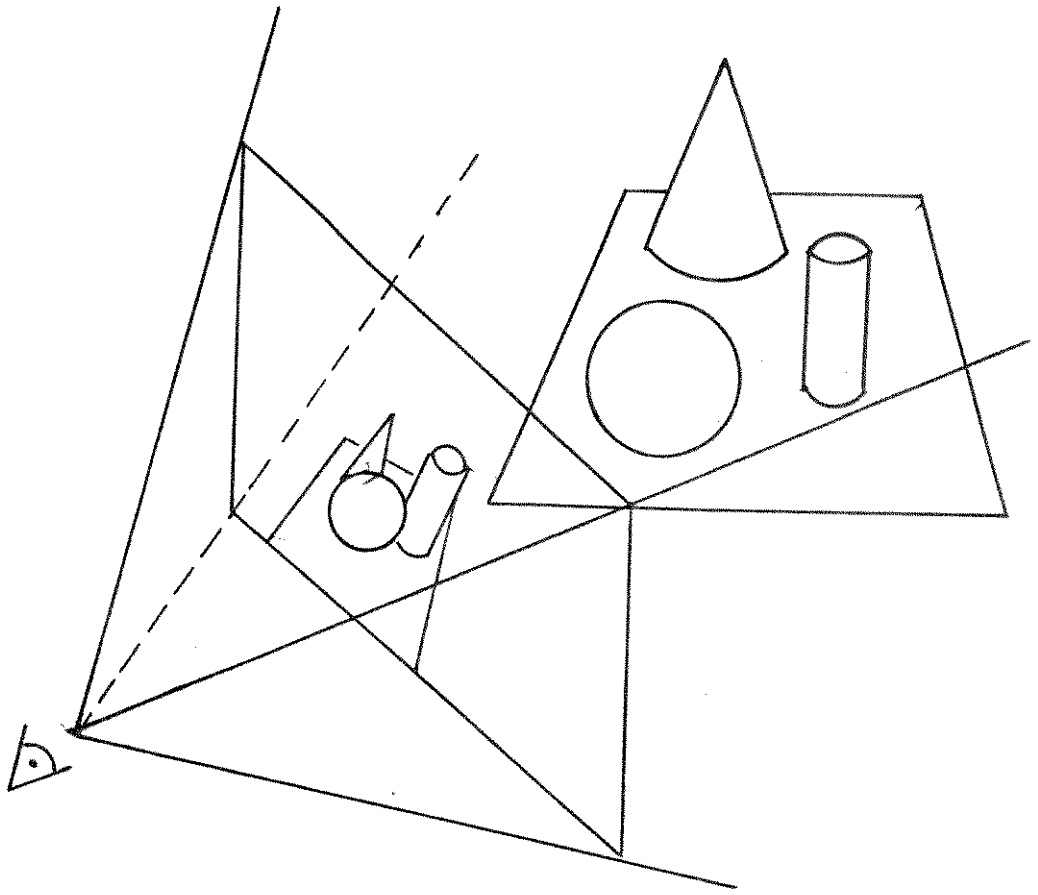

Fig. 2.1- Modelo de câmera.

Nesse modelo, a tela é composta por uma matriz de pixels (abreviatura de picture elements). A cada pixel da tela associamos uma cor. A cor de cada pixel, computada levando-se em conta o modelo de câmera, deve corresponder à parte da cena que é projetada no pixel. Um problema importante em Computação Gráfica é descobrir, dada uma cena, qual ou quais são as partes da cena projetadas no pixel. Em nosso trabalho abordaremos a metodologia ray-tracing visando resolver esse problema.

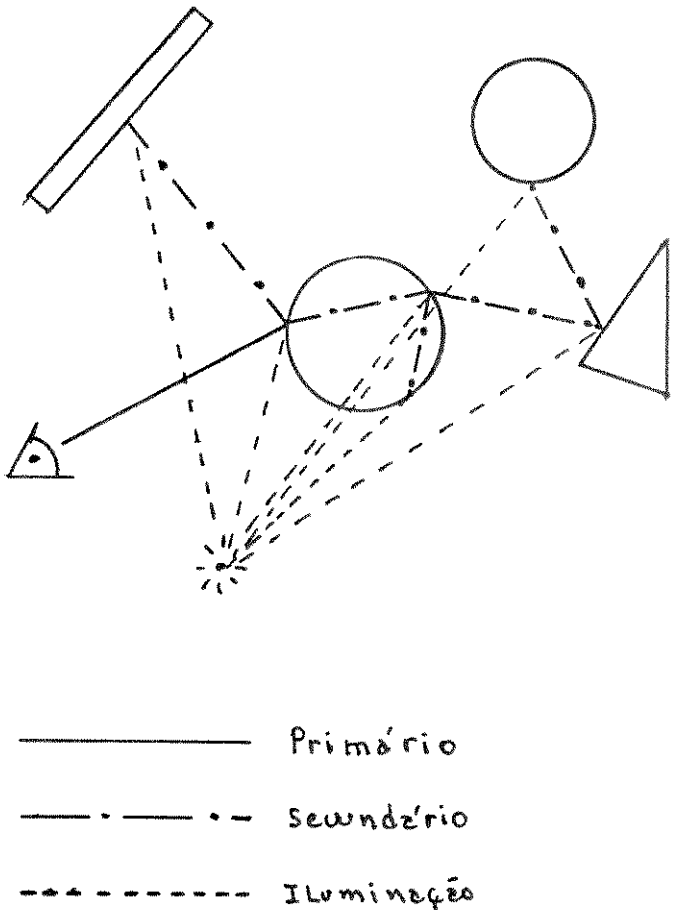

Fig. 2.2- Caminho de um raio de luz na cena.

2.2.2- O CAMINHO DE UM RAIO DE LUZ

O conceito de ray-tracing para cálculo da cor de um pixel é bastante simples e ao mesmo tempo poderoso pois permite simular transparências de luz com refração, reflexões especulares e sombras.

A idéia central do método é o lançamento de raios de luz através de uma cena. Um raio de luz é sempre classificado com um dos seguintes tipos: primário, secundário ou de iluminação.

O raio de luz é primário quando parte do olho do observador e passa pelo interior de um pixel da tela na direção da cena até encontrar uma superfície ou atravessar totalmente a cena sem intersectar nenhum objeto.

O raio de luz é secundário quando nasce no ponto de interseção entre um raio primário ou secundário lançado anteriormente, e uma superfície da cena. O raio secundário é uma decorrência de reflexão especular ou de transmissão de luz numa superfície da cena.

O raio de iluminação nasce no ponto de interseção entre um raio primário ou secundário com uma superfície da cena e vai até uma das fontes de luz que iluminam a cena. O raio de iluminação é utilizado no cálculo da intensidade de luz que chega ao ponto de interseção a partir da fonte de luz.

Definidos os tipos de raios lançados numa cena, vamos agora definir o conceito de ray-tracing.

Partindo do olho do observador e passando pelo centro de um pixel da tela, o raio primário percorre a cena. Ao atingir a superfície do objeto mais próximo do observador, calcula-se a cor do ponto de interseção entre o raio primário e a superfície do objeto. A cor do ponto de interseção chega ao olho do observador através do raio primário. Esta cor é, então, atribuída ao pixel por onde o raio primário passou antes de entrar na cena.

O conceito estabelecido acima é suficiente para a simulação de objetos opacos. Esse conceito pode ser facilmente estendido para simular transparências e reflexos especulares. Para tanto, no ponto de interseção estabelecido pelo raio primário e a superfície traçamos raios secundários, decorrentes de reflexão e transmissão de luz. Esses novos raios de luz percorrem a cena encontrando novos pontos de interseção com superfícies onde o processo de criação de raios secundários se repete como decorrência de reflexão e de transmissão de luz.

A partir de cada ponto de interseção encontrado entre raios primários e secundários com superfícies da cena, um raio de iluminação é lançado na direção de cada fonte de luz para verificar se o ponto de interseção está iluminado ou em sombra. Se existe alguma superfície entre o ponto de interseção e a fonte de luz, a intensidade de luz que chega até o ponto de interseção através do raio de iluminação é

atenuada de acordo com as características de transparência de luz da superfície intermediária. Se esta superfície é opaca, a intensidade de luz que chega ao ponto de interseção é nula. Se a superfície intermediária é transparente, a intensidade de luz que chega ao ponto de interseção é atenuada. A intensidade de luz que chega ao ponto de interseção através de cada raio de iluminação é utilizada para calcular a cor do ponto, e contribue para o cálculo da cor do pixel.

A figura 2.2 ilustra o conceito de ray-tracing mostrando o caminho de um raio de luz na cena.

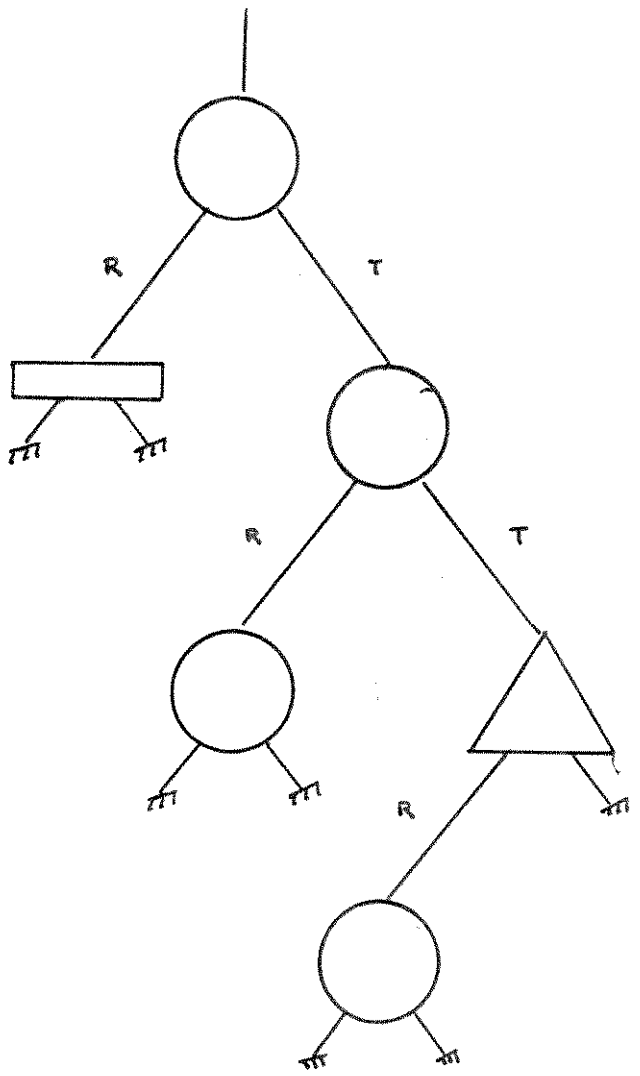

Fig. 2.3- Árvore de interseções raio-superfícies.

2.2.3- A ÁRVORE DE INTERSEÇÕES RAIO-SUPERFÍCIES

O caminho traçado pelo raio de luz na cena gera uma árvore binária de interseções entre raios e superfícies como mostra a figura 2.3. A árvore de interseções nessa figura está relacionada com o caminho percorrido pelo raio de luz da figura 2.2. A árvore de interseções raio-superfícies é implementada como uma árvore binária porque de cada nó da árvore podemos derivar 0, 1 ou, no máximo, 2 nós filhos. Um nó da árvore de interseções tem nós filhos somente se a superfície da cena correspondente ao nó refletir ou transmitir luz. O cálculo da cor vista pelo observador através do raio primário é feito percorrendo-se a árvore de interseções e combinando, segundo um modelo de iluminação, as cores dos pontos de interseção encontrados.

Vários modelos de iluminação foram criados para o cálculo da cor de um ponto numa superfície de uma cena [12,16,25,27,30,38]. O modelo que utilizamos em nosso trabalho é discutido na seção 2.4.3.

2.3- UM ALGORITMO RAY-TRACING

Nos últimos anos, algoritmos ray-tracing foram estabelecidos por vários pesquisadores para produzir imagens de cenas modeladas com métodos diversos. Na verdade, ray-tracing é muito flexível quanto ao método de modelagem de cenas utilizado porque o raio de luz é definido como uma reta no espaço tridimensional. Esse fato torna relativamente fácil encontrar a interseção entre raios de luz e o objeto modelado. Essa simplicidade inerente a ray-tracing é que permitiu o desenvolvimento de algoritmos para sintetizar imagens de cenas modeladas com CSG, superfícies definidas parametricamente, superfícies de revolução, fractais, malhas de polígonos, etc.

Todos esses algoritmos trabalham segundo a mesma idéia básica: um raio de luz é lançado e os pontos de interseção entre o raio e as superfícies da cena são encontrados. Podemos utilizar essa característica dos algoritmos ray-tracing para gerar a imagem de uma cena composta por objetos modelados por métodos diferentes. A única preocupação de alguém que queira utilizar ray-tracing para sintetizar

tal imagem é a de chavear os algoritmos ray-tracing específicos para cada tipo de objeto na hora de calcular as interseções de um raio de luz com um objeto da cena.

O algoritmo ray-tracing estabelecido por Roth [31] utiliza CSG como método de modelagem de cenas e tornou-se um padrão devido a sua simplicidade, elegância e flexibilidade. Esse algoritmo foi o nosso escolhido para implementação tornando-se a base dos programas que compõem o ambiente para síntese de imagens realísticas desenvolvido no Laboratório de Computação Gráfica.

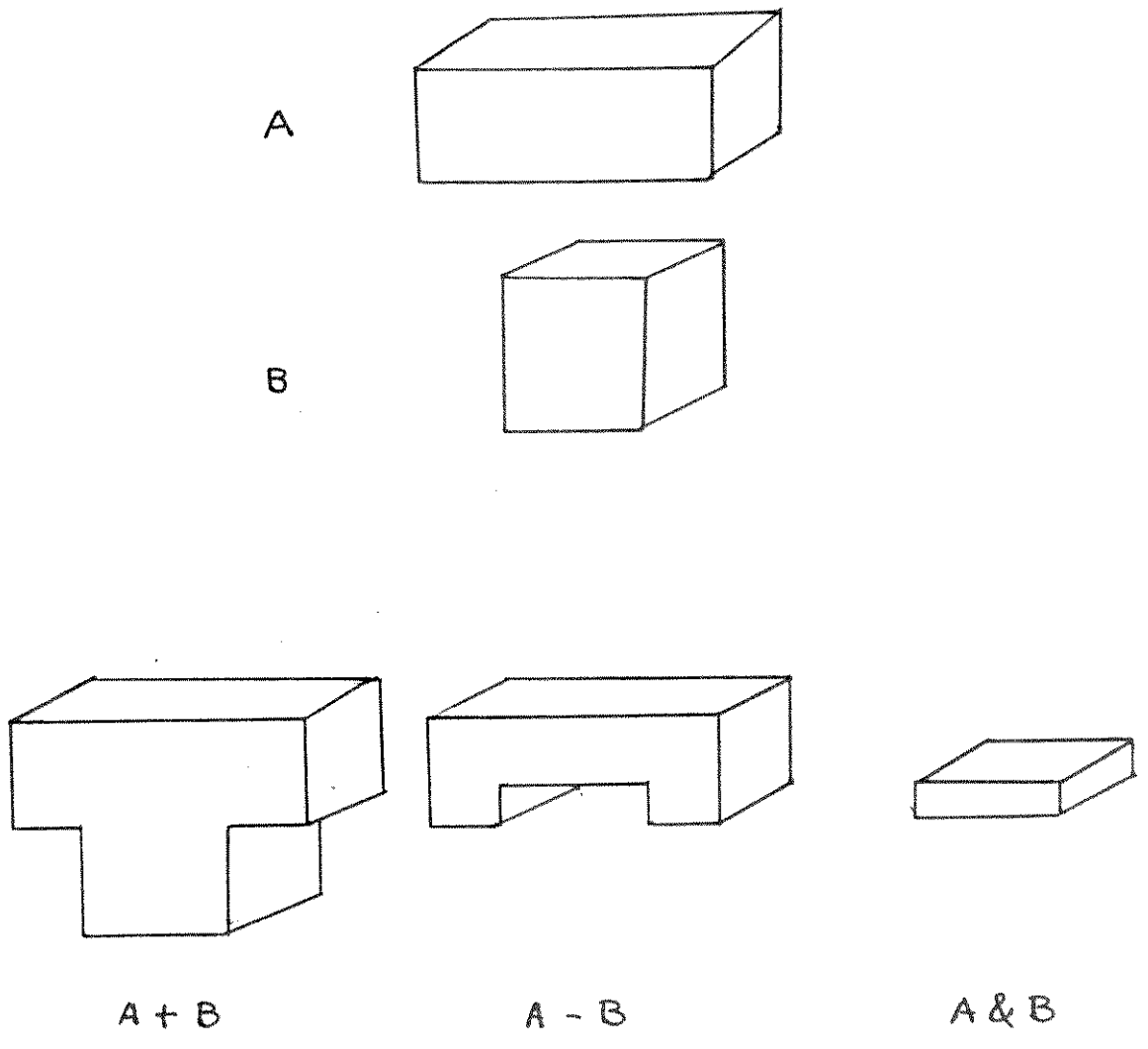

Fig. 2.4- Aplicação de operadores sobre sólidos primitivos.

O algoritmo de Roth apresentado nesta seção é suficiente para sintetizar imagens de objetos opacos apenas. Na seção 2.4 veremos como

estender este algoritmo para simular transparências, reflexos especulares e sombras.

2.3.1- O MÉTODO DE MODELAGEM CSG

O método de modelagem de sólidos Constructive Solid Geometry (CSG), um dos mais utilizados atualmente [29], representa um sólido como a combinação, através dos operadores união (+), diferença (-) e interseção (&), de sólidos primitivos ou sólidos modelados anteriormente.

O efeito da aplicação dos operadores sobre dois sólidos pode ser visto na figura 2.4, com a produção de três sólidos compostos diferentes, um sólido por operador.

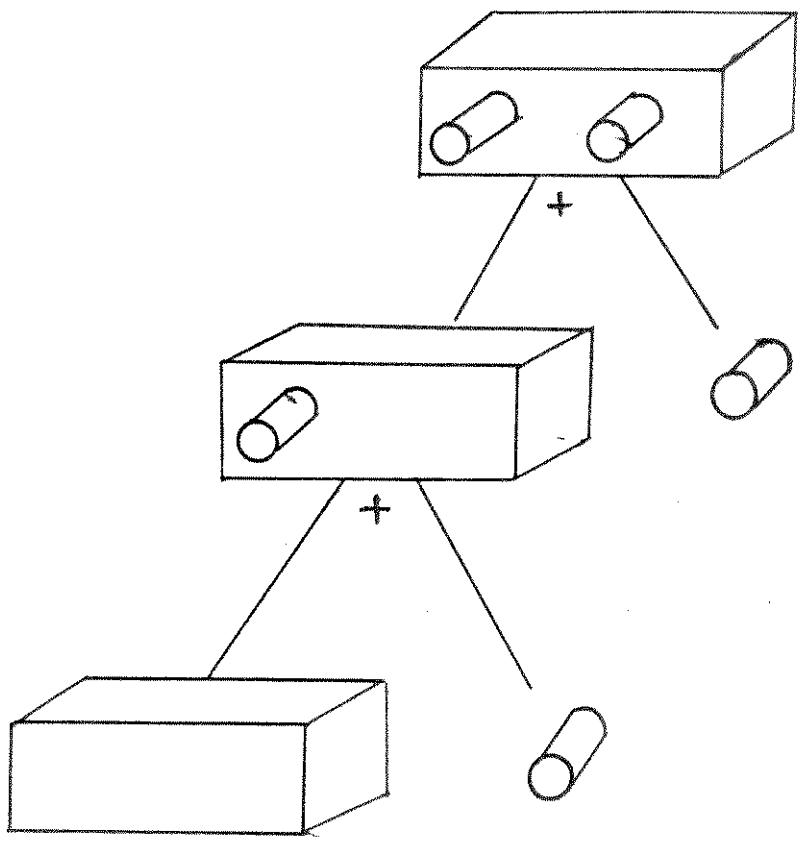

Fig. 2.5- Árvore de composição de sólidos.

A composição de um sólido em CSG é representada por uma árvore binária, chamada árvore de composição de sólidos, cujos nós não-terminais representam sólidos compostos e nós terminais (folhas)

representam sólidos primitivos. A figura 2.5 mostra uma árvore de composição de um sólido a partir de sólidos primitivos. Se a árvore de composição contém n nós primitivos, então essa árvore contém $n - 1$ nós compostos, totalizando $2 * n - 1$ nós na árvore.

O conjunto dos sólidos primitivos de um sistema CSG é importante porque pode facilitar a composição de sólidos. Um possível conjunto básico de sólidos primitivos é formado por cubo, esfera, cone e cilindro, sendo que outros tipos de sólidos primitivos podem ser adicionados ao conjunto, notadamente os sólidos que podem ser modelados por quádricas.

2.3.2- Definição do raio de luz

Um raio de luz é modelado como uma reta no espaço tridimensional. A reta é definida de forma paramétrica, iniciando num pixel da tela e seguindo na direção da cena. A equação da reta é:

$$\begin{cases} X = Xo + tDx \\ Y = Yo + tDy \\ Z = Zo + tDz \end{cases}$$

onde (Xo,Yo,Zo) é o ponto inicial da reta, (Dx,Dy,Dz) é o vetor de direcionamento da reta e t é o parâmetro que, ao variar, define pontos no espaço por onde a reta passa. Por definição, o parâmetro t só pode ter valores positivos, iniciando com 0. Os valores negativos do parâmetro t indicam pontos situados atrás da tela e, portanto, desprezados. O vetor de direcionamento é definido com base na posição relativa entre um pixel da tela e o olho do observador. Na próxima seção veremos como é calculado o vetor de direcionamento do raio de luz.

2.3.3- Os sistemas de coordenadas

Os sistemas de coordenadas utilizados pelo algoritmo de Roth são três: o do observador (SC_OBS), da cena (SC_CENA) e o sistema de coordenadas local (SC_LOCAL), onde são definidos os sólidos primitivos.

O sistema de coordenadas do observador (SC_OBS) é criado com a finalidade de descrever a cena como vista através da camêra fotográfica e facilitar a criação de raios primários pelo algoritmo ray-tracing. O SC_OBS é posicionado através da criação da matriz de transformação CENA_OBS, que permite a transformação de objetos entre o sistema de coordenadas da cena (SC_CENA) e o SC_OBS. O processo de criação da matriz CENA_OBS, a partir do posicionamento da câmera na cena, está bem definido em [25] e não será discutido aqui. O resultado dessa transformação é que um ponto no SC_CENA é transformado para o SC_OBS através da multiplicação do ponto pela matriz CENA_OBS.

A figura 2.6 mostra o modelo de câmera fotográfica estabelecido pelo SC_OBS. O foco da câmera fica na parte negativa do eixo Z enquanto que a cena fica na parte positiva. A janela de visualização definida é utilizada para criar a pirâmide que delimita a região da cena vista pelo observador. O plano XY do SC_OBS é a tela da câmera onde a imagem é projetada. A tela é formada por uma matriz de pixels.

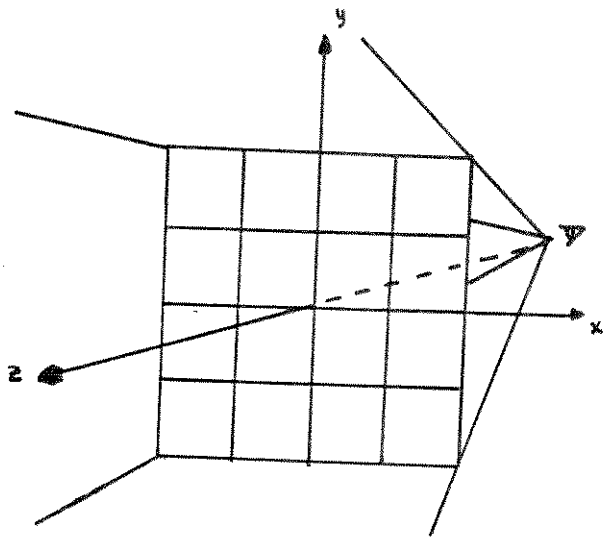

Fig. 2.6- O modelo de câmera estabelecido por SC_OBS.

Os raios de luz primários são definidos no SC_OBS. No conceito de ray-tracing vimos que o raio primário começa no olho do observador, porém em tempo de implementação estabelecemos que o raio primário começa sempre de um pixel da tela. Cada raio de luz inicia num pixel da tela e segue para a cena na direção dada pelo vetor de direcionamento.

Os raios de luz são definidos no SC_OBS da seguinte forma, supondo que o modelo de câmera utiliza uma projeção perspectiva, dados:

a- um ponto focal na parte negativa do eixo Z do SC_OBS: (0,0,-Zf);
b- a tela no plano Z = 0 do SC_OBS;
c- um pixel da tela é definido como (Xp,Yp,0).

O raio de luz é, então, definido como:

$$(Xp,Yp,0) + t * [(Xp,Yp,0) - (0,0,-Zf)],$$

ou seja,

$$(Xp,Yp,0) + t * (Xp,Yp,Zf),$$

onde (Xp,Yp,0) é o ponto inicial do raio e (Xp,Yp,Zf) é o vetor de direcionamento do raio.

Se temos uma projeção paralela, o raio de luz é definido no SC_OBS da seguinte forma, dados:

a- ponto focal no infinito;
b- um pixel da tela definido como na projeção perspectiva.

O raio de luz numa projeção paralela é:

$$(Xp,Yp,0) + t * (0,0,1),$$

onde (Xp,Yp,0) é o ponto inicial do raio e (0,0,1) é o vetor de direcionamento do raio.

O SC_CENA é o único com o qual o usuário precisa realmente se preocupar. Os outros sistemas de coordenadas são criados e processados internamente pelo algoritmo ray-tracing.

O SC_CENA é utilizado para:

a- definir o modelo da câmera fotográfica; todos os parâmetros que definem a câmera são dados no SC_CENA;
b- modelar a cena; a cena, composta por um sólido modelado em CSG, ocupa uma posição definida no SC_CENA;

c- posicionar as fontes de luz que iluminam a cena;

d- o cálculo da cor de um ponto numa superfície (os pontos e os vetores utilizados pelo modelo de iluminação devem ser posicionados e orientados no SC_CENA);

e- definir o raio secundário e o raio de iluminação (será visto na seção 2.4).

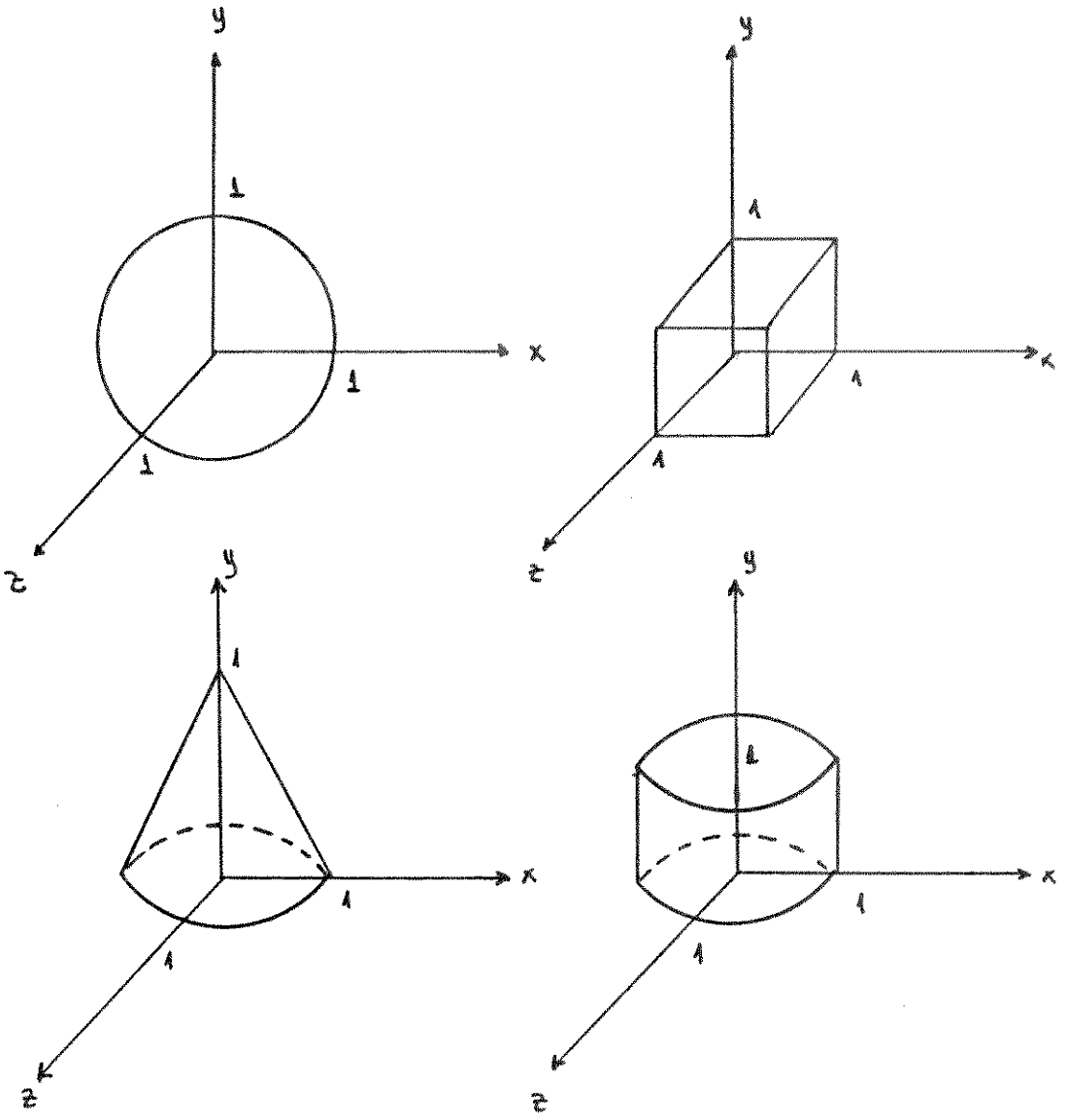

Fig. 2.7- Sistema de coordenadas local de sólidos primitivos.

Cada tipo de sólido primitivo, (cubo, esfera, cone e cilindro), é definido num sistema de coordenadas local próprio (SC_LOCAL) onde o sólido primitivo está situado numa posição padrão. O posicionamento de um sólido primitivo na cena é feito através de uma transformação entre o SC_LOCAL e o SC_CENA.

O posicionamento dos sólidos primitivos cubo, esfera, cone e cilindro em seu SC_LOCAL é mostrado na figura 2.7. O sólido primitivo definido no SC_LOCAL é situado na sua posição na cena através da matriz de transformações LOCAL_CENA. A matriz LOCAL_CENA, bem como sua inversa CENA_LOCAL, é armazenada na árvore de composição de sólidos no nó correspondente ao sólido primitivo que LOCAL_CENA (CENA_LOCAL) transforma.

A utilização do conceito de sistemas locais para sólidos primitivos facilita a implementação das rotinas relacionadas com cada sólido primitivo. Essas rotinas são: cálculo da interseção entre raios de luz e o sólido, cálculo da normal à superfície do sólido e mapeamento de texturas na superfície do sólido. Como um sólido primitivo está sempre na mesma posição no SC_LOCAL, parâmetros importantes como o centro e o raio, no caso de uma esfera, não precisam ser passados a uma rotina de cálculo de interseção porque são conhecidos de antemão. O preço a pagar por essa facilidade é o armazenamento da matriz CENA_LOCAL no nó correspondente a cada sólido primitivo na árvore de composição de sólidos e a transformação do raio de luz do SC_CENA para o SC_LOCAL de cada sólido primitivo antes de executar as rotinas relacionadas com o sólido.

Portanto, para calcular a interseção entre um raio de luz e um sólido primitivo, é preciso transformar o raio de luz, inicialmente criado no SC_OBS, para o SC_CENA e daí para o SC_LOCAL de cada sólido primitivo.

Como Roth notou em seu trabalho, um raio de luz definido como

$$P(t) = Po + tD,$$

onde Po é o ponto inicial do raio, t é o parâmetro que define pontos do raio e D é o vetor de direcionamento do raio, num sistema de coordenadas A, é transformado para o sistema de coordenadas B através da matriz de transformações M_{AB} da seguinte forma:

$$[Po + tD]_A * M_{AB} =$$

$$Po_A * M_{AB} + (tD_A) * M_{AB} =$$

$$Po_A * M_{AB} + t(D_A * M_{AB}) =$$

$$Po_B' + tD_B'.$$

Deste modo, o parâmetro t que define pontos do raio de luz é o mesmo em ambos sistemas de coordenadas, ou seja,

$$P_B'(t) = P_A(t) * M_{AB}.$$

Veremos adiante que o algoritmo de Roth não encontra os pontos onde o raio intersecta um sólido mas, os parâmetros t que definem esses pontos.

2.3.4- O CÁLCULO DE INTERSEÇÕES

Um raio de luz pode passar por um sólido de quatro formas diferentes:

a- o raio não intersecta o sólido;
b- o raio é tangente a um ponto do sólido;
c- o raio é tangente a uma face do sólido;
d- o raio intersecta o sólido.

Nos casos a e b o raio é classificado como OUT, ou seja, não há interseção entre o raio e o sólido. O caso c é considerado como um caso especial de a e b, ou seja, o raio é classificado como OUT.

No caso d dois pontos de interseção são definidos pelo raio e o sólido primitivo. Nesse caso o raio é classificado como OUT_IN_OUT, ou seja, o raio é dividido em três regiões definidas como fora do sólido, dentro do sólido e, novamente, fora do sólido.

Supondo que as equações que definem as superfícies de um sólido primitivo são conhecidas e que um raio de luz seja dado no SC_LOCAL do sólido primitivo, o algoritmo que calcula os pontos de interseção entre raio e sólido e classifica o raio é:

```
PROCEDIMENTO calcula_pontos_de_interseção_e_classifica_raio;
INÍCIO
  PARA CADA superfície do sólido primitivo FAÇA
    INÍCIO
      resolve as equações do raio e da superfície simultaneamente;
      SE os pontos de interseção encontrados estão nos limites da
         superfície
       ENTÃO classifica o raio como OUT_IN_OUT
       SENÃO classifica o raio como OUT
    FIM
FIM;
```

As equações das superfícies dos sólidos primitivos são dadas a seguir, estando os sólidos primitivos definidos nos SC_LOCAL mostrados na figura 2.7:

equação	limites
$X = 0$	$0 <= Y <= 1$ e $0 <= Z <= 1$
$X = 1$	$0 <= Y <= 1$ e $0 <= Z <= 1$
$Y = 0$	$0 <= X <= 1$ e $0 <= Z <= 1$
$Y = 1$	$0 <= X <= 1$ e $0 <= Z <= 1$
$Z = 0$	$0 <= X <= 1$ e $0 <= Y <= 1$
$Z = 1$	$0 <= X <= 1$ e $0 <= Y <= 1$

Tabela 2.1- Definição do cubo.

a- cubo

O cubo é definido por seis planos limitados, cujas equações estão definidas na tabela 2.1;

b- esfera

A esfera está centrada na origem do SC_LOCAL e tem raio unitário. Sua equação é $X^2 + Y^2 + Z^2 = 1$;

equação	limites
$X^2 + Z^2 - (Y - 1)^2 = 0$ $Y = 0$	$0 <= Y <= 1$ $X^2 + Z^2 <= 1$

Tabela 2.2- Definição do cone.

c- cone

O ápice do cone está situado no ponto (0,1,0) e seu eixo está alinhado com a parte positiva do eixo Y do SC_LOCAL. O cone é limitado pelo plano Y = 0, originando um círculo. As equações e limites que definem o cone no SC_LOCAL são dadas na tabela 2.2;

d- cilindro

O cilindro está situado entre os planos Z = 0 e Z = 1 do SC_LOCAL. Suas equações e limites estão na tabela 2.3.

equação	limites
Y = 0	$X^2 + Z^2 <= 1$
Y = 1	$X^2 + Z^2 <= 1$
$X^2 + Z^2 = 1$	$0 <= Y <= 1$

Tabela 2.3- Definição do cilindro.

Por exemplo, seja um raio de luz definido como

$$\begin{cases} X = Xo + tDx \\ Y = Yo + tDy \\ Z = Zo + tDz \end{cases}$$

definido no SC_LOCAL da esfera. Para verificar se há interseção entre ambos é preciso resolver simultaneamente as equações do raio de luz e da esfera:

$$(Xo + tDx)^2 + (Yo + tDy)^2 + (Zo + tDz)^2 = 1.$$

Desenvolvendo essa equação temos:

$$\begin{aligned} &Xo^2 + 2XotDx + t^2Dx^2 + \\ &Yo^2 + 2YotDy + t^2Dy^2 + \\ &Zo^2 + 2ZotDz + t^2Dz^2 = 1 \end{aligned}$$

Colocando t em evidência temos:

$$t^2(Dx^2 + Dy^2 + Dz^2) +$$
$$t(2Xo + 2Yo + 2Zo) +$$
$$(Xo^2 + Yo^2 + Zo^2 - 1) = 0.$$

Fazendo

$$a = (Dx^2 + Dy^2 + Dz^2),$$
$$b = (2Xo + 2Yo + 2Zo) \text{ e}$$
$$c = (Xo^2 + Yo^2 + Zo^2 - 1),$$

resolvemos a equação quadrática como segue.

Se a é igual a 0 apenas uma raiz pode ser encontrada para a equação, ou seja, a esfera tem raio 0 e o raio de luz é classificado como OUT.

Se a é diferente de 0 continuamos a resolução calculando o delta. Se o delta da equação é menor que 0 nenhum ponto de interseção é encontrado e o raio é classificado como OUT. Se o delta é igual a 0, o raio é tangente à esfera e nenhum ponto de interseção é encontrado. Nesse caso o raio é classificado OUT. Se o delta é maior que 0 duas raizes são encontradas: t_1 e t_2. As duas raízes são na verdade os parâmetros que definem os pontos onde o raio e o sólido intersectam. Se substituirmos o parâmetro t na equação que define o raio pelas raízes encontraremos os pontos de interseção. Nesse caso o raio é classificado OUT_IN_OUT.

2.3.5- A ESTRUTURA DE DADOS PARA O ALGORITMO DE ROTH

A estrutura de dados utilizada pelo algoritmo de Roth é composta pela árvore de composição de sólidos que define a cena em CSG e uma lista onde são armazenados os diversos parâmetros calculados pelo algoritmo ray-tracing juntamente com as classificações do raio e pela árvore de interseções raio-superfície.

A árvore de composição de sólidos é composta por dois tipos de nós que representam sólidos compostos ou sólidos primitivos. O conteúdo de cada tipo de nó é:

a- Sólido composto:

- nome do sólido,
- operação de composição,
- ponteiro para sub-árvore direita,
- ponteiro para sub-árvore esquerda;

b- Sólido primitivo:

- nome do sólido,
- tipo do sólido primitivo,
- matriz LOCAL_CENA,
- matriz CENA_LOCAL,
- ponteiro para características da superfície do sólido primitivo.

A lista com a classificação de um raio com relação ao sólido representado pela árvore de composição de sólidos é modelada como um vetor onde cada posição contém um conjunto de dados calculados pelo algoritmo ray-tracing. Essas informações são:

a- o parâmetro t que define cada ponto de interseção encontrado,
b- a classificação IN_OUT do raio no ponto de interseção e
c- o ponteiro para as características da superfície no ponto de interseção.

As informações contidas na lista de classificação são processadas pelo algoritmo de duas formas:

a- os parâmetros t e as classificações IN_OUT são utilizadas pelo algoritmo de combinação de classificações na busca do ponto de interseção entre raio de luz e sólido mais próximo do observador, ou do início do raio;

b- o ponteiro para as características da superfície intersectada permite o cálculo da cor no ponto de interseção.

Como o algoritmo de Roth prevê apenas o lançamento do raio primário, a estrutura de dados da árvore de interseções

raio-superfície será dada na seção 2.4.2, onde veremos a extensão do algoritmo de Roth para produzir transparências, reflexos especulares e sombras.

2.3.6- A COMBINAÇÃO DE CLASSIFICAÇÕES

O algoritmo de Roth, dado um raio de luz, percorre a árvore de composição de sólidos recursivamente, classificando o raio com relação a cada sólido primitivo e combinando as classificações em cada sólido composto segundo a operação de composição, produzindo a lista que contém a classificação final do raio com relação ao sólido.

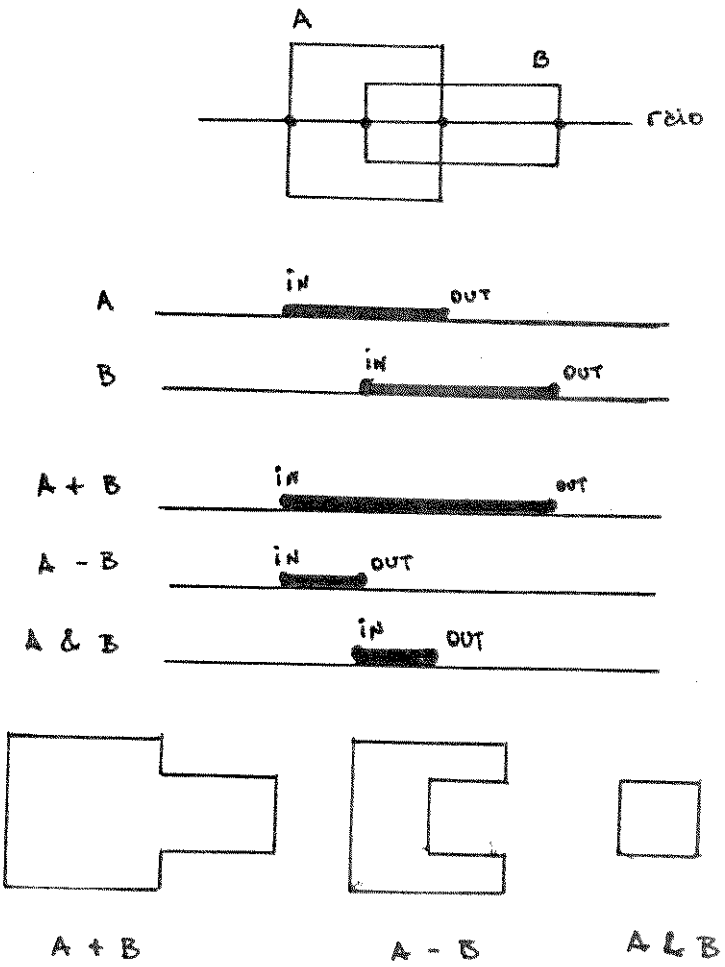

Fig. 2.8- Combinação de classificações de um raio de luz.

A combinação de classificações é feita levando-se em consideração as operações de composição em cada sólido composto. A figura 2.8

ilustra a combinação de classificações de um raio e dois sólidos primitivos para cada operação.

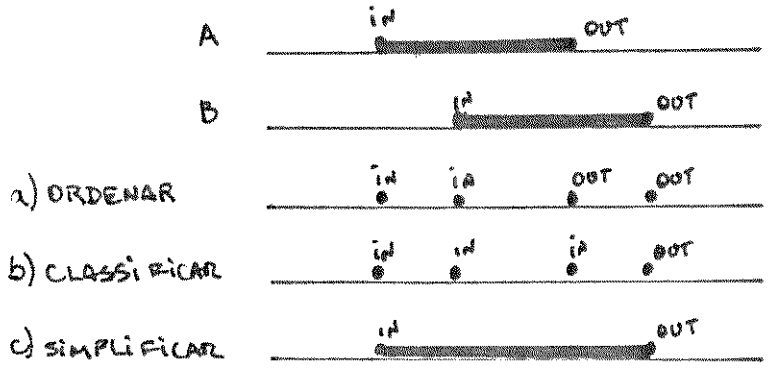

Fig. 2.9- Combinação de classificações.

OPERAÇÃO	ESQUERDA	DIREITA	COMBINAÇÃO
UNIÃO	IN	IN	IN
	IN	OUT	IN
	OUT	IN	IN
	OUT	OUT	OUT
DIFERENÇA	IN	IN	OUT
	IN	OUT	IN
	OUT	IN	OUT
	OUT	OUT	OUT
INTERSEÇÃO	IN	IN	IN
	IN	OUT	OUT
	OUT	IN	OUT
	OUT	OUT	OUT

Tabela 2.4- Classificação de segmentos.

A combinação de classificações é feita em três passos:

a- os pontos de interseção dos sólidos esquerdo e direito são ordenados estabelecendo segmentos IN e OUT no raio de luz;

b- os segmentos são classificados de acordo com a posição do parâmetro t, de sua classificação IN_OUT e da operação de composição;

c- a classificação do raio composto é simplificada unindo-se segmentos adjacentes com a mesma classificação IN_OUT.

A figura 2.9 ilustra o processo de combinação das classificações de um raio e dois sólidos primitivos combinados pela operação união.

A tabela 2.4 mostra as regras para classificação dos segmentos de um raio de luz para as três operações de composição.

2.3.7- O PROCEDIMENTO RAY-TRACING

Supondo que são dados uma árvore de composição de sólidos e um raio de luz no SC_CENA, o único parâmetro fornecido à rotina ray-tracing é um ponteiro para o nó da árvore de composição de sólidos sendo acessado no momento. Esse parâmetro é fornecido devido à estrutura recursiva do procedimento.

A informação retornada por ray-tracing é uma lista com os parâmetros t que definem pontos de interseção entre o raio de luz e o sólido, a classificação IN_OUT de cada ponto de interseção e ponteiros para uma lista com as características de cada superfície intersectada pelo raio de luz. Esta lista, chamada CLASSIFICA, é manipulada em cada chamada de ray-tracing pela combinação de classificações nos nós de combinação.

A seguir o procedimento ray-tracing é dado:

```
PROCEDIMENTO ray_tracing(P_SÓLIDO^);
INÍCIO
  SE P_SÓLIDO^.tipo = COMPOSTO ENTÃO
    INÍCIO
      C_ESQ := ray_tracing( P_SÓLIDO^.P_SÓLIDO_ESQUERDO^ );
      C_DIR := ray_tracing( P_SÓLIDO^.P_SÓLIDO_DIREITO^ );
      combina_classificações( C_ESQ, C_DIR, CLASSIFICA );
    FIM
  SENÃO { o sólido é PRIMITIVO }
    INÍCIO
      transforma o raio para o SC_LOCAL do sólido primitivo;
      CASO tipo_do_sólido_primitivo É:
      CUBO: testa 6 interseções raio-superfícies;
      ESFERA: testa 1 interseção raio-quádrica;
      CONE: testa 1 interseção raio-superfície e
                  1 interseção raio-quádrica;
      CILINDRO: testa 2 interseções raio-superfícies e
                      1 interseção raio-quádrica;
      FIM_DE_CASO;
      CLASSIFICA := resultado do teste de interseção;
    FIM
FIM;
```

O ponto definido pelo primeiro parâmetro t retornado na lista CLASSIFICA é visto pelo observador. Nesse ponto deve ser calculada a quantidade de luz refletida pela superfície encontrando assim a cor que deve ser vista no pixel correspondente ao raio de luz. Na seção 2.4.3 o modelo de iluminação utilizado para o cálculo da cor de um ponto numa superfície será visto.

O algoritmo de Roth é suficiente para sintetizar imagens de objetos opacos mas, como será visto na próxima seção, pode ser estendido para explorar todo o potencial inerente à metodologia ray-tracing.

2.4- A SIMULAÇÃO DE TRANSPARÊNCIAS, ESPELHOS E SOMBRAS

Whitted em [38] foi o primeiro a utilizar com sucesso a metodologia ray-tracing para simular cenas com transparências, reflexões especulares e sombras.

Whitted desenvolveu um método para, dado um raio de luz incidente, gerar raios decorrentes de reflexão e transmissão de luz (figura 2.2). A aplicação dessa metodologia produz uma árvore de interseções raio-superfícies, correspondente ao caminho percorrido pelo raio de luz na cena (figura 2.3).

Em cada nó da árvore de interseções, informações úteis para o cálculo da cor do pixel ao qual a árvore corresponde são armazenadas.

Em paralelo ao método de geração de raios de luz, Whitted desenvolveu um modelo de iluminação para cálculo da cor do pixel explorando as características do método ray-tracing. Esse modelo leva em consideração a contribuição luminosa dos raios de transmissão e reflexão de luz na definição da cor do pixel.

O modelo de iluminação é alimentado com parâmetros que indicam a quantidade de luz refletida em cada superfície da cena. É simples associar os parâmetros de iluminação através da técnica de mapeamento de texturas. Em Computação Gráfica a noção de textura está intimamente

ligada com a aparência de um objeto. De fato, em nosso sistema para síntese de imagens realísticas utilizamos modelos procedurais para definir texturas, as quais são mapeadas nas superfícies dos sólidos presentes na cena. Com essa técnica podemos sintetizar a imagem de um sólido com cores diversas.

2.4.1- Raios especulares, de transmissão e de iluminação

Para estender o algoritmo de Roth com o objetivo de simular sombras, transparências e reflexões é preciso criar novos raios de luz decorrentes de reflexão especular, transmissão com refração de luz e raios na direção das fontes d luz.

Whitted [38] estabeleceu um método para calcular os raios de transmissão e reflexão dados o raio incidente ao ponto de interseção, a normal à superfície no ponto de interseção e os índices de refração entre os meios que o raio de luz atravessa.

A figura 2.10 ilustra o processo de criação de raios de Whitted.

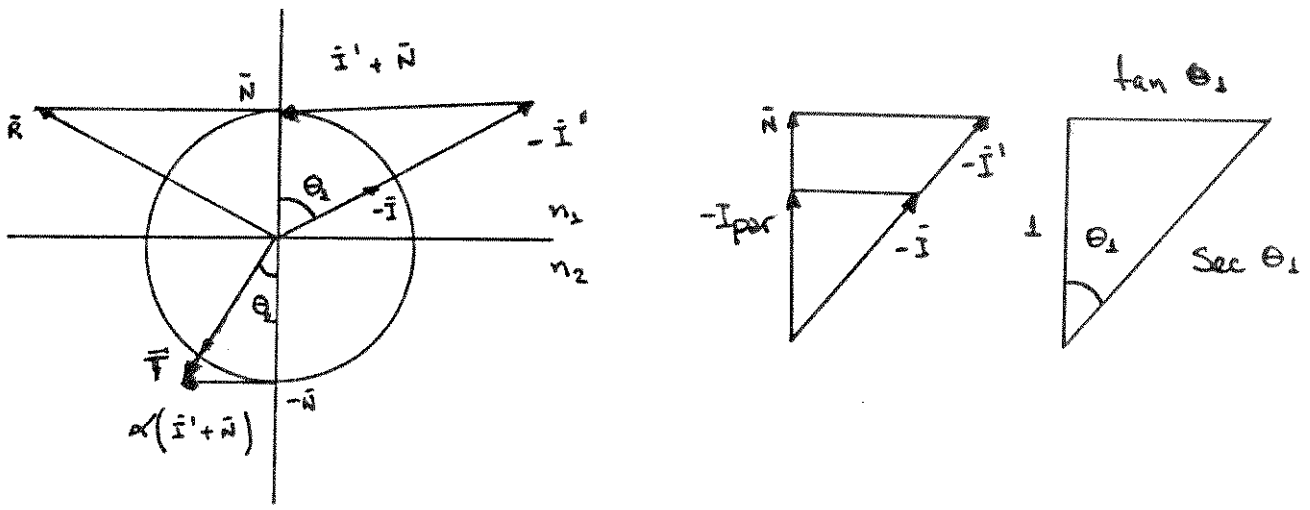

Fig. 2.10- Derivação de raios de luz.

A seguir mostramos como obter raios de reflexão e de transmissão segundo Whitted.

Dado o vetor normalizado N a uma superfície no ponto de interseção entre o raio incidente e o sólido, o vetor raio incidente I e os índices de refração entre os meios n_1 e n_2, calculam-se os vetores raios de transmissão T e de reflexão R de luz como segue. Acompanhe a demontração utilizando a figura 2.10.

$$-I_{par} = \frac{N * (-I . N)}{(N . N)} = N * (-I . N) \quad ==>$$

$$==> \quad I_{par} = (I . N) * N$$

$$\frac{-I}{-I'} \quad \frac{-I_{par}}{N} = I . N \quad ==>$$

$$==> \quad I' = \frac{I}{(-I . N)}$$

Em todas as equações dadas acima o operador . denota produto escalar entre vetores.

O vetor R é dado por:

$$R = I' + 2 * N.$$

O vetor $I' + N$ é paralelo à superfície. Utilizando esse vetor podemos escrever o vetor T como:

$$T = \alpha * (I' + N) - N.$$

Agora é preciso escrever α em termos de N, I e I'.

Como mostrado na figura 2.10,

$$| I' | = \sec \theta_1 ,$$

$$| I' + N | = \tan \theta_1 \text{ e}$$

$$\alpha * | I' + N | = \tan \theta_2 .$$

Assim, dada a lei de Snell,

$$n_1 * \cos \theta_1 = n_2 * \operatorname{sen} \theta_2 ,$$

temos:

$$\alpha = \frac{\tan \theta_2}{\tan \theta_1} = \frac{\operatorname{sen} \theta_2}{\operatorname{sen} \theta_1} * \frac{\cos \theta_1}{\cos \theta_2} =$$

$$= \frac{\dfrac{n_1}{n_2} * \cos \theta_1}{\sqrt{(1 - \operatorname{sen}^2 \theta_2)}}$$

$$= \frac{\dfrac{n_1}{n_2} * \cos \theta_1}{\sqrt{1 - \left(\dfrac{n_1}{n_2}\right)^2 * \operatorname{sen}^2 \theta_1}}$$

$$= \frac{1}{-\sqrt{1 - \frac{n_1^2}{n_2^2} * \operatorname{sen}^2 \theta_1}{\frac{n_1^2}{n_2^2} * \cos^2 \theta_1}}$$

$$= \frac{1}{-\sqrt{\frac{n_2^2}{n_1^2} * \frac{1}{\cos^2 \theta_1} - \tan^2 \theta_1}}$$

$$= \frac{1}{-\sqrt{n^2 \sec^2 \theta_1 - \tan^2 \theta_1}}, \quad n = \frac{n_2}{n_1}$$

Os termos trigonométricos podem ser eliminados resultando:

$$\alpha = \frac{1}{-\sqrt{n^2 * |I'|^2 - |I' + N|^2}}, \quad n = \frac{n_2}{n_1}$$

Reflexão interna ocorre quando o termo dentro da raiz quadrada é negativo. Isso acontece quando a luz passa de um meio mais refringente para um meio menos refringente com um ângulo maior que um ângulo limite determinado pela lei de Snell: $\theta_1 > \theta_L = \arcsen(n)$.

2.4.2- A ESTRUTURA DA ÁRVORE DE INTERSEÇÕES RAIO-SUPERFÍCIES

Ao percorrer uma cena, o raio de luz gera uma árvore binária com interseções raio-superfícies (fig. 2.3). Cada nó dessa árvore contém informações necessárias para o cálculo da cor do pixel correspondente à árvore.

As informações contidas em cada nó da árvore de interseções são:

a- ponto inicial do raio de luz;
b- ponto de interseção raio-superfície;
c- vetor normal à superfície no ponto de interseção;
d- um vetor na direção de cada fonte de luz;
e- a intensidade de luz que chega ao ponto de interseção a partir de cada fonte de luz;
f- um ponteiro para o conjunto de parâmetros de iluminação da superfície;
g- ponteiro para nó de transmissão;
h- ponteiro para nó de reflexão.

Todos os pontos e vetores presentes num nó da árvore de interseções devem ser posicionados e orientados no SC_CENA. Veremos na próxima seção que os dados contidos em cada nó da árvore de interseções são utilizados pelo modelo de iluminação para calcular a cor do ponto de interseção.

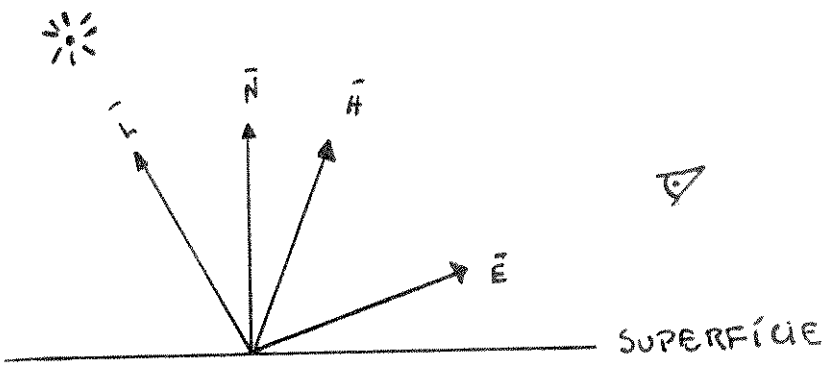

Fig. 2.11- Posição dos vetores do modelo de Phong.

2.4.3- O MODELO DE ILUMINAÇÃO UTILIZADO POR RAY-TRACING

Phong [27] estabeleceu um modelo de iluminação que possibilita calcular a intensidade de luz refletida por uma superfície de objeto opaco simulando reflexão difusa e especular.

O modelo de iluminação de Phong é definido por:

$$I = I_a K_a + K_d \sum_{i=1}^{j} (N \cdot L_i) I_{f_i} + K_e \sum_{i=1}^{j} (N \cdot H_i)^n I_{f_i} \quad , \text{ onde}$$

I é a intensidade de luz refletida pela superfície,
I_a é a intensidade de luz ambiente na cena,
K_a é o coeficiente de reflexão de luz ambiente da superfície,
K_d é o coeficiente de reflexão de luz difusa da superfície,
N é o vetor normal à superfície,
L_i é o vetor na direção da i-ésima fonte de luz,
I_{f_i} é a intensidade luminosa da i-ésima fonte de luz,
K_e é o coeficiente de reflexão especular da superfície,
H_i é o vetor que fica entre o vetor L_i e o vetor E na direção do observador,
n é o fator de reflexão especular da superfície.

A figura 2.11 mostra a posição e o relacionamento dos vetores do modelo de Phong.

O modelo de Phong foi estendido por Whitted em [38] com o objetivo de calcular a intensidade de luz que chega ao olho do observador através de reflexão especular e transmissão de luz.

O modelo de Whitted, aplicado em cada ponto de interseção da árvore de interseções, é basicamente o modelo de Phong, porém acrescido de dois fatores para o cálculo das intensidades de luz que chegam ao ponto de interseção como resultado de reflexão especular e transmissão de luz:

$$I = I_a K_a + K_d \sum_{i=1}^{j} (N \cdot L_i) I_{f_i} + K_e \sum_{i=1}^{j} (N \cdot H_i)^n I_{f_i} + k_r R + k_t T, \qquad [2.1]$$

onde:

R é a intensidade luminosa que chega ao ponto de interseção como decorrência de reflexão especular,
k_t é o coeficiente de transmissão de luz da superfície onde está o ponto de interseção e
T é a intensidade luminosa que chega ao ponto de interseção como decorrência de transmissão de luz.

O cálculo da intensidade de luz que chega ao olho do observador é feito percorrendo-se a árvore de interseções recursivamente, aplicando a equação 2.1 em cada nó. Quando o nó não apresenta raio de reflexão nem de transmissão, as contribuições a intensidade de luz R e T são definidas como 0 ou como a intensidade do fundo se for mais conveniente.

A árvore de interseções raio-superfícies contém, em cada nó, os dados necessários para a aplicação do modelo de iluminação de Whitted no cálculo da cor do ponto de interseção encontrado por um raio lançado na cena. Os coeficientes de reflexão de luz da superfície intersectada pelo raio são acessados através do ponteiro para as características de reflexão de luz da superfície. O vetor na direção do observador é definido como a subtração vetorial do ponto inicial do raio pelo ponto de interseção do raio com a superfície. O vetor na direção de cada fonte de luz que ilumina a cena é definido como a subtração vetorial do ponto onde está situada a fonte de luz pelo ponto de interseção.

O vetor H do modelo de iluminação de Whitted é definido através da equação:

$$H = \frac{L + E}{|L + E|}, \text{ onde}$$

L é o vetor na direção da fonte de luz,
E é o vetor na direção do observador e
| | denota módulo do vetor.

O modelo de Whitted como apresentado é utilizado para gerar imagens em tons de cinza. Para sintetizar uma imagem colorida em

computador, onde cada cor é constituída por três primárias no sistema RGB, a equação é repetida para cada primária. Assim, para calcular a componente R, ou seja, a cor vermelha que chega ao olho do observador num pixel, é preciso definir coeficientes de reflexão de luz de cada superfície exclusivamente para a cor vermelha. Procedemos da mesma forma com as componentes G e B, ou seja, as cores verde e azul. Na seção 2.4.4 um procedimento que associa os parâmetros de iluminação a um sólido, definido com superfícies de cor vermelha, é comentado.

O modelo de Whitted adaptado para o cálculo da componente vermelha de uma cor é definido como:

$$I_r = I_{a_r} K_{a_r} + K_{d_r} \sum_{i=1}^{j} (N \cdot L_i) I_{i_r} + K_s \sum_{i=1}^{j} (N \cdot H_i)^n I_{i_r} + k_s R + k_t T ,$$

onde:

I_r é a intensidade de luz vermelha refletida pela superfície,
I_{a_r} é a intensidade da componente vermelha da luz ambiente na cena,
K_{a_r} é o coeficiente de reflexão de luz ambiente vermelha da superfície,
K_{d_r} é o coeficiente de reflexão de luz difusa vermelha da superfície,
N é o vetor normal à superfície,
L_i é o vetor na direção da i-ésima fonte de luz,
I_{i_r} é a intensidade luminosa da componente vermelha da i-ésima fonte de luz,
K_s é o coeficiente de reflexão especular da superfície,
H_i é o vetor que fica entre o vetor L_i e o vetor na direção do observador,
n é o fator de reflexão especular da superfície,
R é a intensidade luminosa que chega ao ponto de interseção como decorrência de reflexão especular,
k_t é o coeficiente de transmissão de luz da superfície onde está o ponto de interseção e
T é a intensidade luminosa que chega ao ponto de interseção como decorrência de transmissão de luz.

O modelo de Whitted é adaptado para o cálculo das componentes verde e azul de uma cor de modo similar.

Cada superfície de uma cena tem um conjunto de parâmetros de reflexão de luz associados, os quais modelam sua cor. Os parâmetros de uma superfície são acessados através do ponteiro para as características de reflexão da superfície contido na árvore de interseções raio-superfícies. Esse ponteiro aponta para um registro com os parâmetros de reflexão da superfície, definido a seguir:

```
TYPE CAR_REF_LUZ_SUP : RECORD
        kar, kag, kab,
        kdr, kdg, kdb,
        ks, n,
        kt, índice_refração : REAL;
     END;
```

onde,

kar é o coeficiente de reflexão de luz ambiente vermelha,
kag é o coeficiente de reflexão de luz ambiente verde,
kab é o coeficiente de reflexão de luz ambiente azul,
kdr é o coeficiente de reflexão de luz difusa vermelha,
kdg é o coeficiente de reflexão de luz difusa verde,
kdb é o coeficiente de reflexão de luz difusa azul,
ks é o coeficiente de reflexão especular,
n é o fator de reflexão especular,
kt é o coeficiente de transmissão de luz da superfície e
índice_refração é o índice de refração da superfície.

Uma exposição mais aprofundada sobre modelos de iluminação está além do escopo de nosso trabalho. Modelos de iluminação são discutidos em [12,16,25,27,30,38].

2.4.4- MAPEAMENTO DE TEXTURAS

Em Computação Gráfica, a expressão *textura de uma superfície* é entendida como uma referência à aparência da superfície de um objeto. Mais especificamente: quando falamos da textura de uma superfície, estamos falando de sua cor e das pequenas imperfeições observadas, como ranhuras, rugas, etc.

A técnica de mapeamento de texturas é utilizada para atribuir uma determinada aparência a uma superfície de duas formas:

a- pela definição da cor da superfície;

b- através da modelagem de pequenas irregularidades como ondulações e rugas na superfície.

Um estudo aprofundado das técnicas de mapeamento de texturas está além do escopo de nosso trabalho, porém, utilizamos mapeamento de texturas para definir as cores das superfícies de uma cena. As técnicas de mapeamento de texturas são discutidas em [2,3,12,15,19,25,30].

Um padrão de textura é definido no plano bidimensional definido por um sistema de coordenadas com eixos U e V onde:

$$0 <= U <= 1 \quad e$$

$$0 <= V <= 1 \ .$$

O padrão mais simples de se definir é a textura de uma superfície monocromática. Por exemplo, se queremos modelar uma superfície vermelha, todo par (u,v) que define um ponto no sistema de coordenadas de definição de texturas é associado com um conjunto de parâmetros que definem a superfície como vermelha. Por exemplo, uma superfície vermelha pode ser definida com os seguintes parâmetros de reflexão de luz, para todo par (u,v), através do seguinte procedimento:

```
PROCEDIMENTO vermelho( u, v : REAL; VAR sup : CAR_REF_LUZ_SUP);
INÍCIO
  COM sup^ FAÇA
    INÍCIO
      Kar := 1;     Kag := 0;  Kab := 0;     { luz ambiente }
      Kdr := 1;     Kdg := 0;  Kdb := 0;     { luz difusa   }
      Ks  := 0.2;   n := 50;                 { luz especular }
      Kt  := 0;     índice_refração := 0;    { luz transmitida }
   FIM
FIM;
```

O procedimento acima associa a uma superfície, independentemente dos parametros u e v fornecidos, os parâmetros que a definem como vermelha.

A estrutura CAR_REF_LUZ_SUP é um registro que agrupa as caractorísticas de reflexão do luz de uma superfície. Para cada superfície contida numa cena há uma estrutura como CAR_REF_LUZ_SUP associada, com os parâmetros que definem a cor da superfície. Essa estrutura é acessada pelo ponteiro para as características de iluminação de uma superfície armazenado no num nó da árvore de interseções raio-superfícies quando queremos calcular a cor do ponto de interseção entre um raio e a superfície.

O padrão xadrez é um pouco mais complicado de se definir. Inicialmente construímos uma matriz booleana com 8x8 posições. Essas posições recebem valores VERDADEIRO e FALSO de forma alternada formando um padrão xadrez. Ao valor VERDADEIRO associamos a cor vermelha e ao valor FALSO a cor amarela, atribuindo a cada cor os parâmetros de reflexão de luz adequados.

Dados um par (u,v), 0 <= u <= 1 e 0 <= v <= 1, que definem um ponto da superfície onde o padrão xadrez será mapeado, a cor do ponto é definida segundo o procedimento abaixo:

```
VAR matriz_booleana : ARRAY [0..7,0..7] OF BOOLEAN;

PROCEDIMENTO xadrez( u, v : REAL; VAR sup : CAR_REF_LUZ_SUP );
VAR iu, iv : INTEIRO;
INÍCIO
  iu := TRUNC( u * 7 );
  iv := TRUNC( v * 7 );
  SE matriz_booleana[iu,iv] ENTÃO vermelho(u,v,sup)
  SENÃO amarelo(u,v,sup);
FIM;
```

O procedimento amarelo é similar ao procedimento vermelho visto acima. TRUNC é a rotina que devolve o valor inteiro de uma expressão real.

O mapeamento de valores u e v sobre a superfície de um sólido primitivo é feito no SC_LOCAL do sólido. O cubo definido conforme a figura 2.7 é mapeado da seguinte forma.

Considere a face do cubo contida no plano Z = 1, definindo um quadrado contido entre 0 <= X <= 1 e 0 <= Y <= 1. Os pontos dessa face

são definidos genericamente como (x,y,1). Nesse caso fazemos a associação direta entre os eixos X e Y do SC_LOCAL do cubo e os eixos U e V do sistema de coordenadas da textura. Dessa forma, o ponto (x,y,1) é mapeado como (u=x,y=v) no sistema de coordenadas da textura. As outras faces do cubo são mapeadas de forma similar.

Em [15] é feita a descrição de como mapear pontos no SC_LOCAL de sólidos primitivos como esfera, cone e cilindro para o sistema de coordenadas UV.

2.4.5- O PROBLEMA DE "ALIASING"

"Aliasing" é o nome que damos ao aparecimento de certos efeitos nas imagens sintetizadas em computador. Esses efeitos ocorrem devido à discretização de uma imagem contínua numa tela definida como uma matriz de pontos.

O aliasing, numa imagem, é notado principalmente nas bordas de objetos, onde se vê a formação de "escadas". Outro efeito é o desaparecimento de objetos pequenos da imagem.

Para resolver esse problema e gerar imagens livres de "aliasing", ou seja, nas quais as bordas dos objetos são linhas contínuas, Whitted propôs o lançamento de 4 raios por pixel, cada raio passando por um dos cantos do pixel. So as intensidades calculadas pelos raios diferem por um valor superior a um limite então o pixel é subdividido e, para cada subpixel, quatro outros raios são lançados. Esse processo continua até que as intensidades dos raios de luz fiquem dentro do limite estabelecido ou que o número máximo de subdivisões de pixel seja alcançado. A cor do pixel calculada dessa forma é a média das cores de cada raio de luz. O nome desta técnica é "supersampling", super amostragem em português.

Uma outra técnica, cujo objetivo é eliminar o "aliasing", é lançar vários raios aleatoriamente através de um pixel e calcular a cor do pixel como a média da cor dos raios lançados.

O resultado da aplicação de técnicas anti-aliasing num algoritmo ray-tracing é a eliminação ou, no mínimo, a suavização do aliasing encontrado numa imagem. O preço a pagar pela utilização desta técnica agregando-a ao algoritmo ray-tracing, é um aumento extraordinário no tempo de processamento necessário para sintetizar uma imagem devido ao grande número de raios lançados na direção da cena.

2.5- MELHORAMENTOS NA METODOLOGIA RAY-TRACING

O esforço computacional de um algoritmo ray-tracing é muito grande. Uma imagem com resolução 480x640 pixels de uma cena complexa pode levar horas para ser gerada, mesmo em computadores de grande porte.

As características que tornam lentos os algoritmos ray-tracing são o grande número de raios lançados e o número de interseções calculadas em ponto flutuante para cada raio de luz lançado.

Roth [31] ilustrou a complexidade computacional de seu algoritmo com o exemplo dado a seguir.

Dada uma cena com 300 sólidos primitivos, queremos sintetizar uma imagem numa resolução de 500x500 pixels. Portanto, a árvore de composição de sólidos da cena é composta por 600 nós (na verdade 599) e 250.000 raios primários são lançados (apenas 1 raio por pixel). Roth faz as seguintes considerações:

a- para cada raio lançado o procedimento ray-tracing é chamado recursivamente 600 vezes, ou seja:

CUSTO1: 600 * 250.000 = 150.000.000 de execuções do procedimento ray-tracing;

b- em cada sólido composto o procedimento combina classificações é chamado, definindo:

CUSTO2: 300 * 250.000 = 75.000.000 de execuções do procedimento combina classificações;

c- em cada sólido primitivo o procedimento ray-tracing transforma o raio do SC_CENA para o SC_LOCAL do sólido primitivo:

CUSTO3: 300 * 250.000 = 75.000.000 de transformações de raios entre sistemas de coordenadas;

d- em cada sólido primitivo o cálculo de interseções entre o raio e o sólido é feito mesmo que o raio não atinja o sólido. Supondo que a média de superfícies dos sólidos presentes na cena seja 4, temos:

CUSTO4: 4 * 300 * 250.000 = 300.000.000 de testes de interseção entre raios e superfícies.

É difícil modelar um exemplo que leve em consideração raios lançados devido a transmissão, reflexão especular e iluminação mas, tendo em mente que os custos se repetem para cada raio lançado é fácil entender a lentidão dos algoritmos ray-tracing.

Esse elevado custo computacional motivou a pesquisa de técnicas cujo objetivo é melhorar o desempenho dos algoritmos ray-tracing, diminuindo seu tempo de execução.

As técnicas desenvolvidas são classificadas em dois grupos e discutidas nas próximas seções:

a- técnicas visando a diminuição do número de raios lançados;
b- técnicas visando a diminuição do número de testes de interseções.

A aplicação dessas técnicas de melhoramento de desempenho permite um ganho notável no tempo de execução de programas ray-tracing. Em nossa pesquisa, não fizemos uma comparação minuciosa dos tempos de execução do programa ray-tracing, mas observamos um desempenho bastante melhor, da ordem de até 50% de redução no tempo de execução do programa, a partir do momento que aplicamos a técnica de envoltórios.

2.5.1- Diminuição do número de raios lançados

A diminuição do número de raios lançados é feita em dois níveis: a nível de raio primário e a nível de raio secundário.

A nível de raio primário utiliza-se o princípio de coerência de imagem para diminuir o número de raios lançados.

O princípio de coerência de imagem baseia-se no fato de que pixels vizinhos de uma imagem têm a mesma cor, ou cores bastante parecidas.

O processo de síntese de uma imagem utilizando esse princípio, inicia-se com a divisão do plano de projeção da câmera em regiões. Por exemplo, cada região pode ser formada por um quadrado com 16x16 pixels. Um raio primário é lançado através de cada região e a cor encontrada para o raio é atribuída a todos os pixels que compõem a região.

Bronsvoort [6] sugere, dada a cor de uma região, verificar a cor das regiões vizinhas e, se houver uma diferença notável entre as cores, subdividir a região em 4 novas regiões e calcular as cores de cada nova região, refinando a imagem.

Os problemas decorrentes do uso desta técnica são o "aliasing" extremamente acentuado devido à subamostragem e o desaparecimento de objetos pequenos da imagem.

Esta técnica é normalmente utilizada nas primeiras imagens sintetizadas de uma cena, visando a calibração dos parâmetros associados à câmera e a cada uma das superfícies que compõem a cena.

O número de raios secundários lançados como decorrência de reflexão especular e transmissão de luz numa cena pode ser muito grande. A técnica de controle adaptativo é utilizada para diminuir esse número, levando em consideração a contribuição do raio a ser lançado para a cor final do pixel.

A criação da árvore de interseções entre raios de luz e superfícies, definindo o caminho percorrido por um raio na cena é um dos objetivos principais do algoritmo ray-tracing.

Em cenas que contenham muitos objetos reflexivos e transparentes, a árvore de interseções construída para um raio pode tornar-se muito grande, como decorrência do lançamento de raios de reflexão e de transmissão. Como a criação de cada nó da árvore de interseções envolve o lançamento de um raio e o cálculo das interseções entre o raio e as superfícies, o tempo de execução de um programa ray-tracing para uma cena muito reflexiva é proibitivo.

Hall [16] sugere a utilização de controle adaptativo para diminuir a árvore de interseções, evitando o lançamento de raios secundários desnecessariamente.

O controle adaptativo é feito verificando se a contribuição luminosa de cada raio secundário a ser lançado é significativa para a cor final do pixel. Se a contribuição é importante então o raio é lançado, caso contrário o raio é desprezado.

Outra forma de se limitar o crescimento da árvore de interseções de sólidos é estabelecer, a princípio, uma profundidade máxima de crescimento, além da qual nenhum nó pode ser criado, limitando o número máximo de raios que podem ser lançados por pixel.

2.5.2- Diminuição do número de testes de interseção

Whitted [38] observou que programas que executam o método ray-tracing podem gastar até 90% de tempo de execução calculando interseções entre raios e superfícies.

Revendo o exemplo dado anteriormente ao analisarmos os custos do algoritmo de Roth, percebemos que um raio deve ser testado contra todos os sólidos primitivos da cena em busca de interseções mesmo que o raio passe fora do sólido.

É possível diminuir o número de testes de interseção integrando aos programas ray-tracing o uso de envoltórios e do princípio de coerência de objetos na forma de subdivisão espacial.

Clark [7] sugeriu o emprego de envoltórios para acelerar algoritmos para síntese de imagens. Um envoltório é um sólido como cubo ou esfera que envolve vários sólidos no espaço. A contribuição dos envoltórios é que se um raio não intersecta o envoltório, certamente não intersecta os sólidos que estão no interior do envoltório. Vale a pena gastar um tempo extra calculando a interseção de um raio com o envoltório formado por um cubo ou uma esfera, uma operação bastante simples, visando eliminar o tempo gasto com o cálculo de intersecções com todos os sólidos no interior do envoltório.

Roth utiliza envoltórios cúbicos para acelerar seu algoritmo. Os envoltórios são calculados inicialmente nos sólidos primitivos que fazem parte da cena. Os envoltórios dos sólidos compostos são calculados segundo a operação booleana que define o sólido.

Mais fácil que explicar o que é coerência de objetos é explicar seu uso: dados dois sólidos bastante próximos no espaço, se um raio não intersecta um deles, provavelmente não intersectará o outro.

Técnicas para subdivisão 2D e 3D do espaço, baseadas em coerência de objetos foram estabelecidas. O ponto comum de ambas é o objetivo de diminuir a árvore de composição de sólidos e, conseqüentemente, o número de interseções que devem ser calculadas por um raio.

A subdivisão 2D [6.31] é empregada para acelerar o cálculo de interseções de raios primários somente. O algoritmo para subdivisão 2D verifica quais são os sólidos primitivos que projetam-se numa área limitada da tela e constrói uma árvore de composição de sólidos simplificada, eliminando os sólidos primitivos que não estão contidos na área. Esta técnica pode ser utilizada com excelentes resultados em cenas que contenham sólidos opacos, sem reflexão ou transmissão de luz. Caso sejam lançados raios secundários é preciso contar com, além da árvore simplificada, a árvore de composição de sólidos completa para a cena.

A subdivisão espacial 3D [4,7,13,36] divide a cena em subespaços. Cada subespaço contém apenas alguns sólidos primitivos, resultando árvores de composição de sólidos pequenas. Quando um raio é lançado, calcula-se qual é o primeiro subespaço atravessado pelo raio e, se nesse subespaço há interseções entre o raio e os sólidos contidos no subespaço. Se não há interseção continua-se a pesquisa no próximo subespaço atravessado pelo raio.

Esta técnica, apesar do tempo gasto na procura do próximo subespaço percorrido pelo raio, permite acelerar bastante a execução de programas ray-tracing.

CAP. 3- O AMBIENTE PARA SÍNTESE DE IMAGENS REALÍSTICAS

3.1- INTRODUÇÃO

A produção de imagens realísticas em computador envolve aspectos diversos de Computação Gráfica tais como modelagem de cenas, algoritmo para visualização de cenas, utilização de modelos de iluminação, uso de texturas e de sistemas de cor. A construção de um ambiente para a síntese de imagens realísticas em computador é, portanto, tarefa bastante complexa e, de fato, tornou-se o esforço principal no desenvolvimento deste trabalho.

O ambiente para síntese de imagens realísticas em computador desenvolvido no LCG/DCA tem como objetivo preencher todos os requisitos necessários para a produção de imagens realísticas e fornecer ao usuário um conjunto de programas que possam ser modificados com relativa facilidade para fins de pesquisa.

O ambiente básico para síntese de imagens realísticas foi desenvolvido com o emprego da metodologia ray-tracing e utilizando o algoritmo de Roth [31] como base. Este ambiente é composto por um conjunto de programas com o objetivo de:

a- modelar cenas com o método CSG;
b- síntese da imagem de uma cena propriamente dita;
c- construir lookup-tables para cada imagem produzida;
d- mostrar a imagem no monitor.

A arquitetura do ambiente para produção de imagens realísticas é mostrada na figura 3.1.

O programa responsável pela modelagem de cenas é descrito na seção 3.2. O programa para síntese de imagens realísticas é discutido na seção 3.3. Na seção 3.4, discutimos métodos para escolha das cores que melhor representam uma imagem e o programa montador de lookup-tables é apresentado. O programa para mostrar imagens no monitor é mostrado na seção 3.5. Os formatos de arquivos usados pelo ambiente são descritos na seção 3.6.

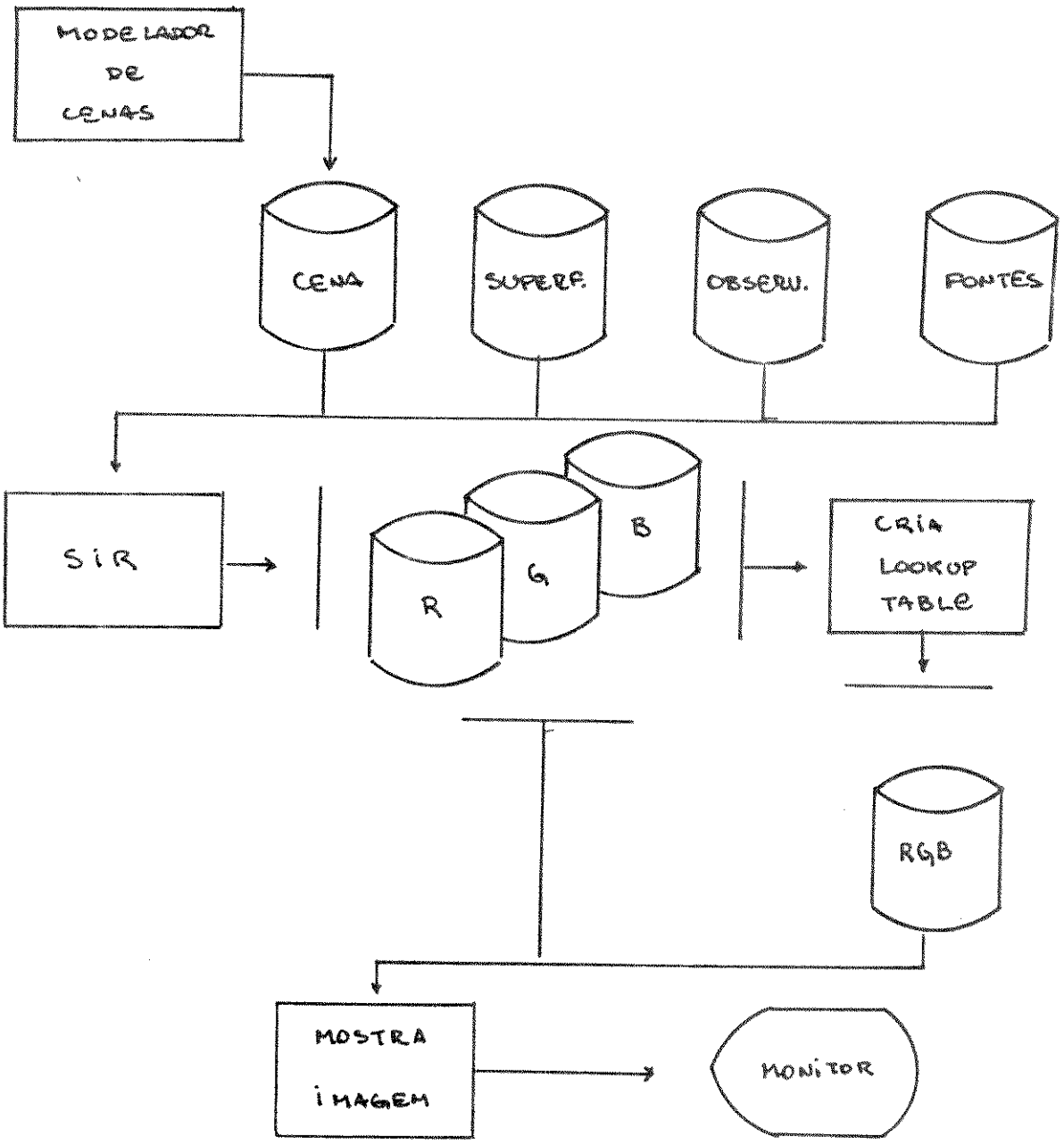

Fig. 3.1- Arquitetura do ambiente para síntese de imagens.

3.2- O MODELADOR DE CENAS

O programa responsável pelo modelamento de cenas com o método CSG foi implementado em linguagem Pascal em ambiente PC, conta com aproximadamente 4000 linhas na versão atual, e é similar ao sugerido por Roth [31].

As funções do modelador são:

a- criar instâncias de sólidos primitivos;
b- transformar sólidos no sistema de coordenadas da cena;
c- combinar sólidos formando sólidos mais complexos;
d- deletar sólidos;
e- copiar o conteúdo de um sólido fonte num sólido destino;
f- desfazer um sólido composto gerando dois novos sólidos;
g- salvar um sólido em disco;
h- carregar um sólido do disco;
i- mostrar um sólido no monitor de vídeo;
j- verificar o conteúdo de um sólido;
k- listar os sólidos disponíveis ao usuário.

Este programa baseia-se na utilização de menus e é operado interativamente pelo usuário com o objetivo de modelar uma cena. Uma cena pode ser composta por um ou mais sólidos. O processo de criação de um sólido pelo usuário é o seguinte:

a- o usuário cria nós de sólidos primitivos;
b- transforma cada sólido primitivo do SC_LOCAL para o SC_CENA, situando cada sólido primitivo em sua posição na cena;
c- cria nós de sólidos compostos através da composição de sólidos primitivos e de sólidos compostos anteriormente;
d- visualiza o sólido durante a fase de composição se for necessário.

3.2.1- Estrutura de dados do modelador de sólidos

O modelador de sólidos trabalha com o método CSG. Em CSG um sólido é modelado como a composição de dois outros sólidos através de uma das operações união, diferença ou interseção. O sólido é modelado como uma árvore binária onde os nós são sólidos compostos e as folhas são sólidos primitivos fornecidos pelo modelador.

A árvore binária que define um sólido, chamada árvore de composição de sólidos, é formada por dois tipos de nós: o nó composto e o nó primitivo.

O nó composto contém os seguintes dados:

a- nome do sólido;
b- operação de composição (união, diferença ou interseção);
c- envoltório no SC_OBS;
d- envoltório no SC_CENA;

e- ponteiro para o nó esquerdo;
f- ponteiro para o nó direito.

O nó primitivo contém os seguintes dados:

a- nome do sólido;
b- tipo do sólido (cubo, esfera, cone ou cilindro);
c- envoltório no SC_OBS;
d- envoltório no SC_CENA;
e- matriz de transformação LOCAL_CENA;
f- matriz de transformação CENA_LOCAL;
g- números que identificam as superfícies componentes do sólido.

O modelador permite ao usuário verificar uma lista de sólidos em uso (LSU), onde estão disponíveis os sólidos modelados anteriormente. A LSU pode ser modelada como um vetor onde cada posição corresponde a um sólido e contém os seguintes dados:

a- nome do sólido;
b- tipo do sólido (primitivo ou composto);
c- ponteiro para a árvore de composição do sólido.

3.2.2- AS FUNÇÕES DO MODELADOR

Os seguintes procedimentos são de uso geral de todas as funções do modelador:

a- insere_sólido_na_LSU - esta função é responsável pela inserção de um sólido na LSU se houver espaço; seu único parâmetro é o ponteiro para o sólido que deve ser inserido;

b- delete_sólido_da_LSU - esta função é responsável pela deleção de um sólido e liberação da posição ocupada pelo sólido na LSU; seu único parâmetro é o nome do sólido a ser deletado;

c- recupera_sólido_na_LSU - esta função é responsável pela busca de sólidos na LSU; seu parâmetro de entrada é o nome do sólido e os de saída são a posição do sólido na LSU e o tipo do sólido (primitivo ou composto);

d- existe_sólido_na_LSU - esta função booleana é responsável pela verificação da existência ou não de um sólido na LSU; seu parâmetro é o nome do sólido;

e- espaço_na_LSU - esta função verifica se há espaço disponível na LSU para inserção de sólidos. Retorna o número de posições disponíveis;

f- LSU_vazia - esta função booleana verifica se a LSU está vazia ou não;

g- libera_posição_na_LSU - esta função torna disponível uma posição da LSU sem deletar o sólido correspondente; seu parâmetro é o índice da posição na LSU.

A seguir as funções do modelador são apresentadas. A apresentação das funções é feita através da sua descrição, do algoritmo e de observações a respeito da função se for o caso.

3.2.2.1- CRIA SÓLIDO PRIMITIVO

A- Descrição

Cria sólido primitivo é a função utilizada para criar uma instância de sólido primitivo inserindo-a na LSU. Os dados relativos ao sólido primitivo fornecidos pelo usuário são o nome do sólido, o tipo do sólido (cubo, esfera, cone ou cilindro). Para cada superfície que constitue o sólido, um número que a identifica é fornecido pelo usuário. Este número é associado a uma textura específica dentre as fornecidas pelo programa de visualização de cenas.

B- Algoritmo

```
PROCEDIMENTO cria_sólido_primitivo;
INÍCIO
  SE espaço_na_LSU > 0 ENTÃO
    INÍCIO
      LE( nome_do_sólido );
      SE NÃO existe_sólido_na_LSU(nome_do_sólido) ENTÃO
        INÍCIO
```

```
            cria_sólido_primitivo(SÓLIDO^);
            LE(tipo_do_sólido_primitivo);
            LE(número_das_súperfícies_que_constituem_o_sólido);
            inicia_sólido_primitivo;
            insere_sólido_na_LSU(SÓLIDO^);
          FIM
        SENÃO ESCREVE("O SÓLIDO JÁ EXISTE");
      FIM
    SENÃO ESCREVE("NÃO HÁ ESPAÇO NA LSU");
FIM;
```

C- Observações

A rotina cria_sólido_primitivo é responsável pela criação de um nó de sólido primitivo. Esta rotina solicita ao sistema operacional da máquina onde o modelador foi implementado a memória para tal fim e retorna um ponteiro para o nó (SÓLIDO^).

A rotina inicia_sólido_primitivo é responsável pela inicialização de todos os campos do sólido primitivo, ou seja, o nome e o tipo dados pelo usuário, as matrizes LOCAL_CENA E CENA_LOCAL com a matriz identidade e o envoltório do sólido de forma padrão.

3.2.2.2- TRANSFORMA SÓLIDOS ENTRE OS SISTEMAS DE COORDENADAS

A- Descrição

Transforma sólidos é responsável pelo processamento de transformações entre sistemas de coordenadas. Esta função é utilizada quando se quer transformar um sólido primitivo definido em seu sistema de coordenadas local para o sistema de coordenadas da cena ou quando se quer mover um sólido já situado na cena.

B- Algoritmo

```
PROCEDIMENTO transforma_sólido;
INÍCIO
  SE NÃO LSU_vazia ENTÃO
    INÍCIO
      LE(nome_do_sólido);
      SE existe_sólido_na_LSU(nome_do_sólido) ENTÃO
        INÍCIO
          recupera_sólido_na_LSU( nome_do_sólido, posição_do_sólido,
                                  tipo_do_sólido );
          SE tipo_do_sólido é PRIMITIVO ENTÃO
```

```
            INÍCIO
              tipo_do_sólido_primitivo := LSU[posição_do_sólido].
                                          SOLIDO^.tipo_do_sólido_primitivo;
              CASO tipo_do_sólido_primitivo É
                CUBO: cria_transformação(CUBO,matriz);
                ESFERA: cria_transformação(ESFERA,matriz);
                CONE: cria_transformação(CONE,matriz);
                CILINDRO: cria_transformação(CILINDRO,matriz);
              FIM_DE_CASO;
              LOCAL_CENA := matriz;
              inverte_matriz(LOCAL_CENA,CENA_LOCAL);
              inicia_envoltórios_do_sólido_primitivo;
            FIM
          SENÃO  { o tipo_do_sólido é COMPOSTO }
            INÍCIO
              centraliza_sólido(matriz);
              processa_transformação(matriz)
              atualiza_os_envoltórios_do_sólido;
            FIM
        FIM
      SENÃO ESCREVE("O SÓLIDO NÃO EXISTE");
    FIM
  SENÃO ESCREVE("A LSU ESTÁ VAZIA");
FIM;
```

C- Observações

A rotina cria_transformação é responsável pelo posicionamento de um sólido primitivo na cena. O modelador de sólidos oferece ao usuário um conjunto padrão de sólidos primitivos, cubo, esfera, cilindro e cone, situados numa posição padrão no sistema de coordenadas local de cada tipo de sólido primitivo (SC_LOCAL). É importante notar que, devido à utilização de diferentes tipos de sólidos primitivos, algumas operações de transformação resultam interessantes. Por exemplo, se uma esfera sofrer uma operação de escala com valores diferentes em cada eixo do sistema de coordenadas, ela se transformará num elipsóide. O modelador tem capacidade para tratar individualmente transformações em cada eixo de coordenadas permitindo a criação de sólidos diferentes dos sólidos primitivos iniciais.

A rotina inverte_matriz simplesmente inverte a matriz LOCAL_CENA que situa um sólido na cena, criando a matriz CENA_LOCAL que situa o sólido da cena no sistema local novamente.

As rotinas inicializa_os_envoltórios_do_sólido_primitivo e atualiza_os_envoltórios_do_sólido operam sobre os envoltórios do sólido. Os envoltórios são importantes tanto para modelagem, o

envoltório no SC_CENA é usado para centralizar o sólido no SC_CENA, como para visualização, o envoltório no SC_OBS é usado para melhorar o desempenho da função responsável pela visualização do sólido no monitor.

A rotina centraliza_sólido situa um sólido composto na posição padrão de transformação de sólidos compostos. O sólido é colocado na posição padrão com o auxílio do envoltório no SC_CENA. Utilizando as coordenadas do envoltório pode-se criar uma matriz de transformação com a operação translação posicionando o sólido de tal forma que o centro do envoltório situe-se no centro do SC_CENA.

Há várias formas de se implementar a rotina para o processamento de transformações. O algoritmo utilizado pelo modelador de sólidos é:

```
PROCEDIMENTO processa_transformação( VAR matriz);
INÍCIO
  inverte_matriz( matriz,matriz_inv);
  mt1 := matriz;
  REPETE
    identidade(mt2);
    LE(opção_de_transformação);
    CASO opção_de_transformação É
      TRANSLAÇÃO: INÍCIO
                    LE(dados_da_translação);
                    cria_matriz(mt2,TRANSLAÇÃO,dados_da_translação);
                  FIM;
      ESCALA    : INÍCIO
                    LE(dados_da_escala);
                    cria_matriz(mt2,ESCALA,dados_da_escala);
                  FIM;
      ROTAÇÃO   : INÍCIO
                    LE(dados_da_rotação);
                    cria_matriz(mt2,ROTAÇÃO,dados_da_rotação);
                  FIM;
    FIM_DO_CASO;
    mt1 := mt1 * mt2;
  ATÉ opção_de_transformação = FIM;
  matriz := mt1 * matriz_inv;
FIM;
```

A matriz de entrada deste procedimento é a matriz de translação que coloca o sólido na posição padrão de transformação. Nesta posição o sólido é centralizado no SC_CENA.

A rotina identidade simplesmente atribui à matriz dada como parâmetro os valores correspondentes a uma matriz identidade.

A multiplicação final da matriz de transformação do sólido por matriz_inv tem o objetivo de situar o sólido que foi inicialmente colocado na posição padrão de transformação na posição em que o sólido encontrava-se inicialmente.

3.2.2.3- Combina sólidos

A- Descrição

Combina sólidos é responsável pela criação de sólidos compostos através da combinação de sólidos primitivos ou de sólidos compostos anteriormente através das operações de conjunto união, diferença ou interseção. Os sólidos que serão combinados devem estar na LSU. O resultado é um único sólido na LSU, tornando uma posição da LSU disponível.

B- Algoritmo

```
PROCEDIMENTO combina_sólidos;
INÍCIO
  SE espaço_na_LSU > 0 ENTÃO
    INÍCIO
      SE NÃO LSU_vazia ENTÃO
        INÍCIO
          LE(nome_do_sólido_a_ser_composto);
          SE  NÃO  existe_sólido_na_LSU(nome_do_sólido_a_ser_composto)
          ENTÃO
            INÍCIO
              LE(nome_do_sólido_esquerdo);
              SE existe_sólido_na_LSU(nome_do_sólido_esquerdo) ENTÃO
                INÍCIO
                  LE(nome_do_sólido_direito);
                  SE     existe_sólido_na_LSU(nome_do_sólido_direito)
                  ENTÃO
                    INÍCIO
                      LE(tipo_de_operação_de_conjunto);
                      SE (tipo_de_operação_de_conjunto = UNIÃO) OU
                         ( tipo_de_operação_de_conjunto ESTÁ_EM
                             [DIFERENÇA,INTERSEÇÃO]
                           E há_interseção_espacial(sólido_esquerdo,
                                                    sólido_direito))
                      ENTÃO
                        INÍCIO
                          cria_sólido_composto(SÓLIDO_COMPOSTO^);
                          insere_sólido_na_LSU(SÓLIDO_COMPOSTO^);
```

```
                        SÓLIDO_COMPOSTO^.NÓ_ESQUERDO          :=
                          sólido_esquerdo;
                        SÓLIDO_COMPOSTO^.NÓ_DIREITO           :=
                          sólido_direito;
                        combina_envoltórios_no_sólido_composto;
                        recupera_sólido_na_LSU(
                          nomo_do_sólido_esquerdo,
                          posição_do_sólido_esquerdo,
                          tipo_do_sólido_esquerdo);
                        recupera_sólido_na_LSU(
                          nome_do_sólido_direito,
                          posição_do_sólido_direito,
                          tipo_do_sólido_direito);
                        libera_posição_na_LSU(
                          posição_do_sólido_esquerdo);
                        libera_posição_na_LSU(
                          posição_do_sólido_direito);
                      FIM
                    SENÃO  ESCREVE("NÃO  HÁ  INTERSEÇÃO  ENTRE  OS
                                    SÓLIDOS");
                  FIM
                SENÃO ESCREVE("O SÓLIDO NÃO EXISTE");
              FIM
            SENÃO ESCREVE("O SÓLIDO NÃO EXISTE");
          FIM
        SENÃO ESCREVE("O SÓLIDO JÁ EXISTE");
      FIM
    SENÃO ESCREVE("A LSU ESTÁ VAZIA");
  FIM
SENÃO ESCREVE("NÃO HÁ ESPAÇO NA LSU");
FIM;
```

C- Observações

Inicialmente deve haver uma posição livre na LSU onde o sólido composto é inserido.

O teste de interseção espacial é uma forma de auxílio ao usuário pois impede a criação de sólidos compostos vazios, já que o resultado da interseção ou diferença de dois sólidos que não intersectam no espaço é vazio.

A rotina cria_sólido_composto cria um nó de sólido composto solicitanto memória ao sistema operacional onde o modelador foi implementado e retornando um ponteiro para o nó composto (SÓLIDO_COMPOSTO^)

A rotina combina_envoltórios atualiza os envoltórios do sólido composto através da operação de combinação do sólido composto aplicada aos envoltórios dos sólidos esquerdo e direito.

3.2.2.4- DELETA SÓLIDO

A- Descrição

A função deleta sólido é responsável pela deleção de cada nó do sólido, devolução da memória correspondente a cada nó ao sistema operacional onde o modelador foi implementado e pela liboração da posição da LSU correspondente ao sólido deletado.

B- Algoritmo

```
PROCEDIMENTO deleta_sólido;
INÍCIO
  SE NÃO LSU_vazia ENTÃO
    INÍCIO
      LE(nome_do_sólido);
      SE existe_sólido_na_LSU(nome_do_sólido) ENTÃO
        INÍCIO
          recupora_sólido_na_LSU( nome_do_sólido,  posição_do_sólido,
            tipo_do_sólido);
          percorre_o_sólido_deletando_nós;
          libera_posição_da_LSU(posição_do_sólido);
        FIM;
      SENÃO ESCREVE("NÃO EXISTE ESSE SÓLIDO");
    FIM
  SENÃO ESCREVE("A LSU ESTÁ VAZIA");
FIM;
```

C- Observações

A rotina percorre_o_sólido_deletando_nós é responsável pela devolução da memória relativa a cada nó ao sistema operacional.

3.2.2.5- COPIA SÓLIDO

A- Descrição

Copia sólido copia o conteúdo de um sólido fonte num sólido destino e insere o sólido destino na LSU. O sólido fonte deve ser um sólido composto.

B- Algoritmo

```
PROCEDIMENTO copia_sólido;
INÍCIO
  SE NÃO LSU_vazia ENTÃO
    INÍCIO
      SE espaço_na_LSU > 0 ENTÃO
        INÍCIO
          LE(nome_do_sólido_fonte);
          SE existe_sólido_na_LSU(nome_do_sólido_fonte) ENTÃO
            INÍCIO
              LE(nome_do_sólido_destino);
              SE NÃO existe_sólido_na_LSU(nome_do_sólido_destino)
               ENTÃO
                INÍCIO
                   cria_sólido_composto(SÓLIDO_DESTINO^);
                   percorre_o_sólido_fonte_copiando_nós;
                   insere_sólido_na_LSU(SÓLIDO_DESTINO^);
                FIM
              SENÃO ESCREVE("O SÓLIDO DESTINO JÁ EXISTE");
            FIM
          SENÃO ESCREVE("O SÓLIDO FONTE NÃO EXISTE");
        FIM
      SENÃO ESCREVE("A LSU ESTÁ CHEIA");
    FIM
  SENÃO ESCREVE("A LSU ESTÁ VAZIA");
FIM;
```

C- Observações

A rotina percorre_sólido_fonte_carregando_nós é responsável pela criação dos nós do sólido destino e cópia do conteúdo dos nós do sólido fonte no sólido destino.

3.2.2.6- Desfaz sólido

A- Descrição

A função desfaz sólido divide um sólido composto em dois novos sólidos. O sólido composto original é deletado e os dois novos sólidos são inseridos na LSU.

B- Algoritmo

```
PROCEDIMENTO desfaz_sólido;
INÍCIO
  SE NÃO LSU_vazia ENTÃO
```

```
    INÍCIO
      SE espaço_na_LSU > 0 ENTÃO
        INÍCIO
          LE(nome_do_sólido_a_ser_desfeito);
          SE existe_sólido_na_LSU(
               nome_do_sólido_a_ser_desfeito) ENTÃO
            INÍCIO
              recupera_sólido_na_LSU( nome_do_sólido_a_ser_desfeito,
                                      posição_do_sólido,
                                      tipo_do_sólido );
              SE tipo_do_sólido = COMPOSTO ENTÃO
                INÍCIO
                  SÓLIDO_COMPOSTO := LSU[posição_do_sólido].SÓLIDO;
                  sólido_esquerdo^ := SÓLIDO_COMPOSTO^.NÓ_ESQUERDO^;
                  sólido_direito^ := SÓLIDO_COMPOSTO^.NÓ_DIREITO^;
                  recupera_sólido_na_LSU(
                     nome_do_sólido_a_ser_desfeito,
                     posição_do_sólido_a_ser_desfeito,
                     tipo_do_sólido_a_ser_desfeito);
                  libera_posição_na_LSU(
                    posição_do_sólido_a_ser_desfeito);
                  deleta_nó_cabeça_do_sólido_composto;
                  insere_sólido_na_LSU( sólido_esquerdo^);
                  insere_sólido_na_LSU( sólido_direito^);
                FIM
              SENÃO ESCREVE("O SÓLIDO NÃO É COMPOSTO");
            FIM
          SENÃO ESCREVE("O SÓLIDO NÃO EXISTE");
        FIM
      SENÃO ESCREVE("A LSU ESTÁ CHEIA");
    FIM
  SENÃO ESCREVE("A LSU ESTÁ VAZIA");
FIM;
```

C- Observações

A rotina deleta_nó_cabeça_do_sólido_composto devolve ao sistema operacional a memória correspondente ao nó composto.

3.2.2.7- SALVA SÓLIDO

A- Descrição

Salva sólido é responsável pelo salvamento de um sólido modelado em arquivo listável em disco. O uso desta função permite a criação de bibliotecas de sólidos.

B- Algoritmo

```
PROCEDIMENTO salva_sólido;
INÍCIO
  SE NÃO LSU_vazia ENTÃO
    INÍCIO
      LE(nome_do_sólido);
      SE existe_sólido_na_LSU( nome_do_sólido) ENTÃO
        INÍCIO
          LE( nome_do_arquivo);
          abre_arquivo( nome_do_arquivo);
          percorre_salvando_sólido;
          fecha_arquivo( nome_do_arquivo);
        FIM
      SENÃO ESCREVE("O SÓLIDO NÃO EXISTE");
    FIM
  SENÃO ESCREVE("A LSU ESTÁ VAZIA");
FIM;
```

C- Observações

abre_arquivo e fecha_arquivo são responsáveis pela abertura, fechamento e tratamento de erros na manipulação de arquivos.

percorre_salvando_sólido é responsável pela leitura e salvamento de cada nó que forma o sólido que está sendo salvo em disco. O formato do arquivo em que o sólido é salvo é mostrado na seção 4.6.

3.2.2.8- CARREGA SÓLIDO

A- Descrição

Carrega sólido é a função que permite ao usuário carregar um sólido modelado numa sessão de trabalho anterior.

B- Algoritmo

```
PROCEDIMENTO carrega_sólido;
INÍCIO
  SE espaço_na_LSU > 0 ENTÃO
    INÍCIO
      LE(nome_do_sólido);
      SE NÃO existe_sólido_na_LSU( nome_do_sólido) ENTÃO
        INÍCIO
          LE( nome_do_arquivo);
          abre_arquivo( nome_do_arquivo);
          percorre_carregando_sólido;
```

```
          fecha_arquivo( nome_do_arquivo);
          insere_sólido_na_LSU(SÓLIDO^);
        FIM
      SENÃO ESCREVE("O SÓLIDO JÁ EXISTE");
    FIM
  SENÃO ESCREVE("A LSU ESTÁ CHEIA");
FIM;
```

C- Observações

percorre_carregando_sólido é responsável pela leitura do arquivo onde se encontra a descrição do sólido e inserção do sólido na LSU.

3.2.2.9- MOSTRA SÓLIDO

A- Descrição

Mostra sólido é a função que permite ao usuário visualizar o estado do modelamento de um sólido. A silhueta do sólido é mostrada na tela do monitor de vídeo dando ao usuário uma noção do posicionamento do sólido na cena.

B- Algoritmo

```
PROCEDIMENTO mostra_sólido;
INÍCIO
  SE NÃO LSU_vazia ENTÃO
    INÍCIO
      LE(nome_do_sólido);
      SE existe_sólido_na_LSU( nome_do_sólido) ENTÃO
        INÍCIO
          recupera_sólido_na_LSU( nome_do_sólido, posição_do_sólido,
                                  tipo_do_sólido );
          percorre_transformando_sólido;
          PARA y := ymax ATÉ ymin PASSO -1 FAÇA
            PARA x := xmin ATÉ xmax PASSO 1 FAÇA
              INÍCIO
                cria_raio(x,y,raio);
                ray_tracing(raio,LSU[posição_do_sólido],
                            superfície[x,y]);
                SE superfície[x,y] <> superfície[x,y - 1] ENTÃO
                  acende_pixel(x,y - 0.5)
                SENÃO
                SE superfície[x,y] <> superfície[x - 1,y] ENTÃO
                  acende_pixel(x - 0.5,y);
              FIM
        FIM
      SENÃO ESCREVE("O SÓLIDO NÃO EXISTE");
```

```
    FIM
  SENÃO ESCREVE("A LSU ESTÁ VAZIA");
FIM;
```

C- Observações

percorre_transformando_sólido é responsável pela atualização dos envoltórios do sólido no SC_OBS segundo o modelo de câmera utilizado pelo usuário. O envoltório no SC_OBS é usado com a finalidade de melhorar o desempenho do algoritmo ray-tracing.

cria_raio gera um raio de luz que inicia no pixel (x,y) e vai na direção do cena.

ray_tracing é a rotina responsável pelo cálculo do ponto de interseção entre o raio e o sólido mais próximo do observador. Seus parâmetros de entrada são o raio e o ponteiro para um sólido da LSU, o qual será visualizado. ray-tracing retorna o número de identificação da superfície atingida pelo raio, ou 0 se nenhuma superfície foi intersectada.

acende_pixel é a rotina responsável por setar um pixel da tela do monitor de vídeo.

A variável superfície é uma matriz que recebe o número da superfície atingida pelo raio de luz em cada pixel. O teste realizado no final de mostra_sólido tem a finalidade de descobrir quais são os pixels localizados na silhueta do sólido. Esses pixels devem ser setados para dar ao usuário a visão do sólido na cena.

3.2.2.10- VERIFICA SÓLIDO

A- Descrição

Verifica sólido permite ao usuário conhecer dados importantes a respeito de um sólido como nome, o conteúdo dos envoltórios, tipo do sólido, tipo de sólido primitivo, os números de identificação de cada superfície que constitue um sólido primitivo, a árvore de composição do sólido, etc.

B- Algoritmo

```
PROCEDIMENTO verifica_sólido;
INÍCIO
  SE NÃO LSU_vazia ENTÃO
    INÍCIO
      LE( nome_do_sólido);
      SE existe_sólido_na_LSU( nome_do_sólido) ENTÃO
        INÍCIO
          LE( opção_de_verificação );
          mostra_dados( opção_de_verificação );
        FIM
      SENÃO ESCREVE("O SÓLIDO NÃO EXISTE");
    FIM
  SENÃO ESCREVE("A LSU ESTÁ VAZIA");
FIM;
```

C- Observações

A rotina mostra_dados exibe na tela os dados do sólido conforme requeridos pelo usuário.

3.2.2.11- LISTA SÓLIDOS

A- Descrição

Lista sólidos tem a função de listar os nomes dos sólidos inseridos na LSU que estão, portanto, disponíveis para uso do usuário.

B- Algoritmo

```
PROCEDIMENTO lista_sólidos;
INÍCIO
  SE NÃO LSU_vazia ENTÃO
    INÍCIO
      lista_sólidos_da_LSU;
    FIM
  SENÃO ESCREVE("A LSU ESTÁ VAZIA");
FIM;
```

C- Observações

lista_sólidos_da_LSU percorre a LSU recolhendo os nomes dos sólidos disponíveis e listando esses nomes na tela do monitor.

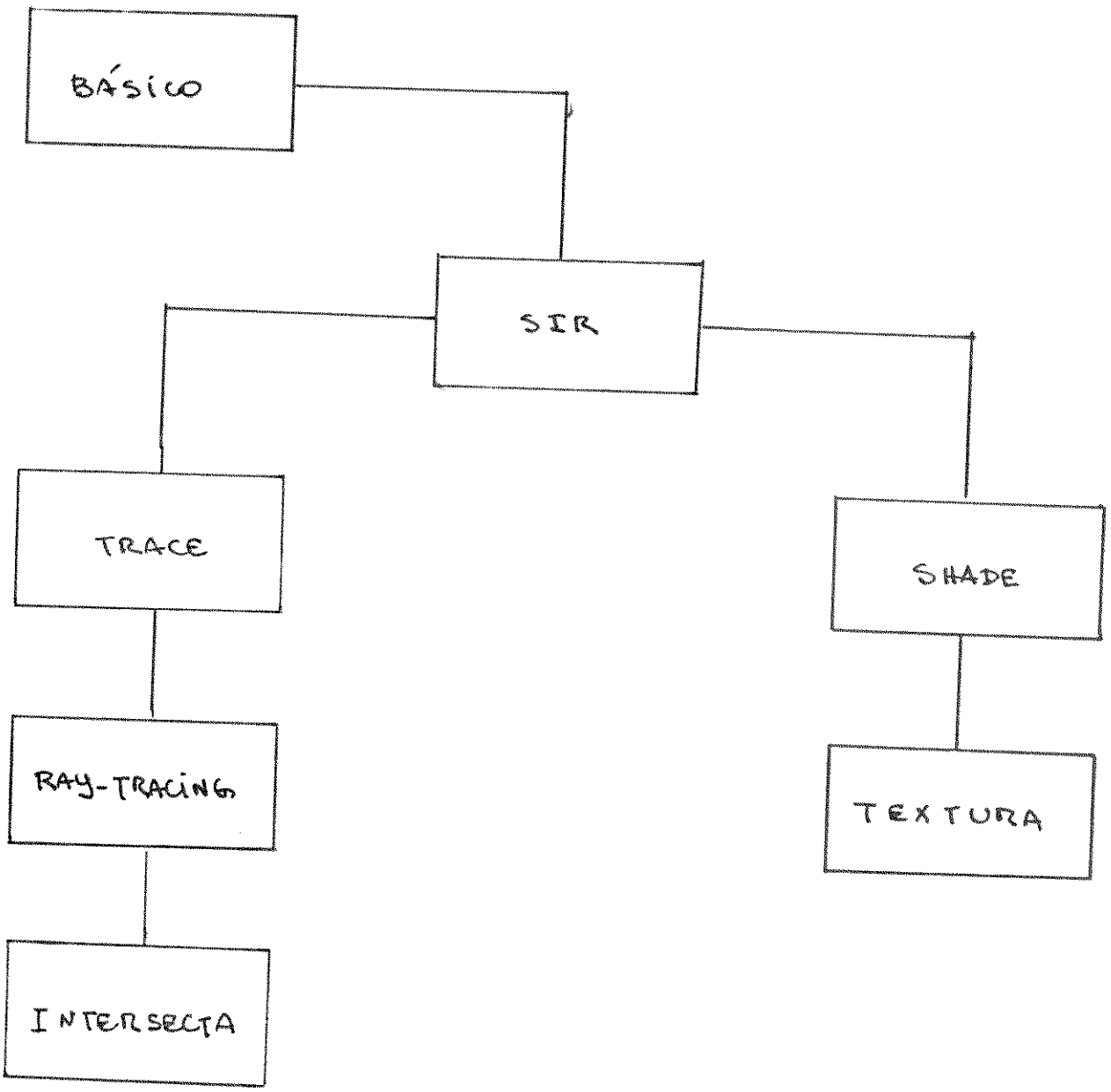

Fig. 3.2- Estrutura modular do SIR.

3.3- O SINTETIZADOR DE IMAGENS REALÍSTICAS

O sintetizador de imagens realísticas (SIR) é o programa responsável pela produção de imagens coloridas com transparências, reflexões especulares e sombras de uma cena.

Para que o SIR possa criar uma imagem é necessário que o usuário forneça ao programa o banco de dados que modela a cena, a posição do observador e a posição e intensidade das fontes de luz que iluminam a cena. O banco de dados da cena contém os sólidos que compõem a cena e o número que identifica cada uma das superfícies de cada sólido da cena. Essas informações são criadas com o modelador de sólidos apresentado na seção 3.2. O posicionamento do observador da cena, necessário para se construir o modelo de camêra fotográfica e as informações a respeito da iluminação da cena, devido à sua simplicidade, são criados com um editor de textos.

O SIR utiliza essas informações para produzir uma imagem colorida da cena. A imagem é armazenada em três arquivos, sendo que cada arquivo armazena uma das primárias RGB que contribue para a cor de um pixel da imagem.

Os formatos dos arquivos utilizados pelo SIR são descritos na seção 3.6.

O SIR é um programa construído com a utilização de módulos. Cada módulo do SIR tem uma função bem definida no programa, tornando mais fáceis as tarefas de depuração e modificações de código. A estrutura modular do SIR é mostrada na figura 3.2.

O módulo BÁSICO compreende as rotinas de uso geral do SIR. Aqui estão as rotinas para leitura dos arquivos onde as informações relativas à cena, descrição de superfícies, posição do observador e das fontes de luz estão armazenadas. Este módulo também inicializa as estruturas de dados internas do SIR e contém rotinas de uso geral do SIR como manipulação de matrizes, cálculo do módulo, normalização de vetores e cálculo da normal de um sólido.

O módulo SIR é responsável pela manipulação do modelo de câmera criando e lançando raios de luz primários na direção da cena. A versão atual gera apenas um raio primário por pixel e não faz nenhum tratamento anti-aliasing. SIR gera uma imagem criando três arquivos com as primárias que formam as cores de cada um dos pixels da imagem.

Os raios secundários e de iluminação são criados e lançados no módulo TRACE. Esses raios simulam transparências, reflexões especulares e sombras. O módulo TRACE é chamado recursivamente em cada ponto de interseção entre raio de luz e superfície da cena encontrado.

Em cada ponto de interseção encontrado cria-se um nó da árvore de interseções raio-superfícies, a qual representa o caminho percorrido pelo raio de luz na cena.

Dois mecanismos que limitam o número de raios de luz gerados foram implementados em TRACE: limite de recursão e controle adaptativo [16].

O limite de recursão estabelece a profundidade máxima da árvore de interseções raio-superfícies e é definido pelo usuário de acordo com a cena. Se a cena contém apenas objetos opacos o limite de recursão é definido como 1. Se a cena é composta por objetos transparentes e reflexivos o limite de recursão pode ser maior e deve ser definido de forma apropriada.

O controle adaptativo da árvore de interseções impede que novos raios sejam lançados verificando se a contribuição luminosa do raio de luz é importante para o cálculo da cor do pixel ou não. A contribuição luminosa limite também é definida pelo usuário. Podemos definir, por exemplo, o limite de contribuição luminosa de um raio como 5% da cor total do pixel.

Raios são lançados a partir dos pontos de interseção na direção de cada fonte de luz da cena possibilitando o cálculo da intensidade de luz que chega ao ponto de interseção e o cálculo da cor do pixel.

O módulo RAY-TRACING é responsável por percorrer a árvore de composição de sólidos que modela a cena e pela combinação dos pontos de interseção encontrados para cada raio de luz. Neste módulo utiliza-se o conceito de envoltórios para melhorar o desempenho do programa. Em RAY-TRACING, o raio de luz é testado contra o envoltório do sólido e, se houver interseção, o raio é passado para o módulo INTERSECTA, onde as interseções entre raios e sólidos primitivos são calculadas.

O módulo INTERSECTA é responsável pelo cálculo da interseção entre um raio de luz no SC_LOCAL e um sólido primitivo. Este módulo retorna os parâmetros que indicam os pontos de interseção calculados, a classificação OUT_IN_OUT que define os pontos de entrada e saída do raio do sólido primitivo e o número da superfície do sólido primitivo.

O módulo SHADE é responsável por percorrer a árvore de interseções raio-superfícies e calcular a intensidade de luz que chega ao olho do observador através do raio de luz, definindo a cor do pixel correspondente. O modelo de iluminação utilizado em SHADE é o modelo de Whitted [38].

O módulo TEXTURA é responsável pela definição de texturas de cada sólido da cena. O número da superfície de um sólido primitivo define a textura que corresponde ao sólido primitivo. Na versão atual, apenas texturas que definem a cor de um sólido podem ser usadas, ou seja, as superfícies dos sólidos modelados no ambiente são sempre lisas. A cor de um sólido primitivo é definida através das constantes de reflexão de luz de sua superfície.

Algumas texturas simples como xadrez, letras e outras que possam ser criadas através de modelos procedurais são facilmente agregadas ao módulo TEXTURA enriquecendo o ambiente.

3.3.1- Estrutura de dados e algoritmos do SIR

Tendo definido a estrutura modular do SIR de tal forma que o programa possa ser modificado e aumentado com facilidade, vejamos agora sua estrutura de dados e seus algoritmos principais.

As estruturas de dados mais importantes do SIR são a árvore de composição de sólidos que define a cena, a árvore de interseções raio-superfícies que define o caminho de um raio na cena, a lista com a classificação de um raio com relação à cena e a definição do raio de luz. Essas estruturas são definidas abaixo.

```
TYPE
     (* tipos básicos *)
     VETOR  = ARRAY [1..4] OF REAL;
     MATRIZ = ARRAY [1..4,1..4] OF REAL;


     (* definição do raio de luz *)
     RAIO = RECORD
       p : VETOR;  (* início do raio *)
       d : VETOR;  (* direção do raio *)
     END;


     (* ponteiros para nós de sólidos *)
     P_SLD_PRMT = ^SLD_PRMT;  (* ponteiro para sólido primitivo *)
     P_SLD_CMP  = ^SLD_CMP;   (* ponteiro para sólido composto  *)


     (* definição de sólido primitivo *)
     SLD_PRMT = RECORD
       tipo     : ( CUBO, ESFERA, CONE, CILINDRO );
       loc_cena,
       cena_loc : MATRIZ;    (* matrizes de transformação entre
                                sistemas de coordenadas          *)
       nro_sld  : INTEGER;   (* identidade do sólido na cena     *)
       id_sup   : INTEGER;   (* ponteiro para características das
                                superfícies do sólido           *)
     END;


     (* definição de sólido composto *)
     SLD_CMP = RECORD
       op       : ( UNIÃO, DIFERENÇA, INTERSEÇÃO );
                                          (* operação de
                                             composição          *)
       no_esq   : ( PRIMITIVO, COMPOSTO );  (* tipo do nó esquerdo *)
       p_esq    : ( P_SLD_ESQ, P_SLD_DIR );  (* ponteiro para sólido
                                             esquerdo            *)
       no_dir   : ( PRIMITIVO, COMPOSTO );  (* tipo do nó direito  *)
       p_dir    : ( P_SLD_ESQ, P_SLD_DIR );  (* ponteiro para sólido
                                             direito             *)
       nro_sld : INTEGER;                 (* identidade do sólido
                                             na cena             *)
     END;


     (* lista com a classificação de um raio *)
     CLASSIFICA = ARRAY [1..NINTER] OF RECORD
       t        : REAL;          (* parâmetro que define um ponto     *)
       id_sup   : INTEGER;       (* número da superfície do
                                    sólido intersectada pelo raio     *)
       io       : ( IN, OUT );   (* indica se o ponto de interseção é
                                    IN ou OUT                         *)
       p_sld    : P_SLD_PRMT;    (* ponteiro para o nó do sólido
                                    primitivo intersectado pelo raio  *)
     END;
```

A constante NINTER define o número máximo de interseções que um raio podo ter com a cena.

```
(* ponteiro para árvore de interseções raio-superfícies *)
P_ARV_INTER : ^ARV_INTER;

(* árvore de interseções raio-superfícies *)
ARV_INTER : RECORD
  pi      : VETOR;     (* ponto de interseção raio-superfície *)
  n       : VETOR;     (* vetor normal à superfície no pi     *)
  nro_sld : INTEGER;   (* identidade do sólido intersectado   *)
  int     : ARRAY [1..NFONTE] OF VETOR;
                       (* intensidade que chega ao pi de cada
                          fonte de luz da cena                *)
  u, v    : REAL;      (* parâmetros para cálculo de textura  *)
  kar, kag, kab,       (* parâmetros de iluminação do pi      *)
  kdr, kdg, kdb,
  ks, expn,
  kt, indr: REAL;
  p_esp   : P_ARV_INTER;
                       (* ponteiro para nó de reflexão de luz *)
  p_tra   : P_ARV_INTER;
                       (* pont. para nó de transmissão de luz *)
END;
```

A constante NFONTE define o número máximo de fontes de luz presentes na cena.

As rotinas principais do programa SIR são:

a- sir - leitura dos arquivos de dados de uma cena, lançamento de raios primários e montagem do primeiro nó da árvore de interseções;
b- trace - lançamento de raios de iluminação, de raios secundários e continuação da montagem da árvore de interseções;
c- shade - cálculo da cor do pixel segundo o modelo de Whitted.

Essas rotinas são dadas a seguir. A maior parte do "código" dessas rotinas é apenas uma linha auto-explicativa. Onde sentimos a necessidade de maiores detalhes, escrevemos em estilo Pascal.

São variáveis acessadas globalmente pelas rotinas sir, trace e shade:

```
VAR
  P_lst_clas : INTEGER;       (* ponteiro para a lista CLASSIFICA *)
  classifica : CLASSIFICA;    (* lista de classificações          *)
  p_sólido   : ( P_SLD_CMP, P_SLD_PRMT );
                              (* ponteiro para a cena             *)
```

A rotina ray_tracing é responsável pelo cálculo de interseções entre um raio de luz e a cena modelada com o método CSG e pelo retorno dessas interseções devidamente classificadas na lista classifica. p_lst_clas controla se foram encontradas interseções ou não entre o raio e a cena.

A rotina calcula_pi é responsável pelo cálculo do ponto de interseção entre o raio e a cena. É preciso notar que o raio é definido de forma paramétrica e o cálculo do ponto de interseção feito com auxílio do parâmetro t armazenado na lista classifica.

A rotina calcula_normal é responsável pelo cálculo da normal à superfície do sólido no ponto de interseção encontrado com o raio.

A rotina calcula_uv é responsável pela parametrização da superfície do sólido intersectada pelo raio. Os parâmetros u e v encontrados são utilizados pela rotina calcula_ctes_iluminação para calcular os coeficientes de reflexão de luz do ponto de interseção entre raio e sólido.

```
(* sir é responsável pela leitura dos arquivos de dados que definem a
cena, pela criação dos raios primários e pela montagem do primeiro nó
da árvore de interseções raio-superfícies                              *)
(* A constante FUNDO define a cor de fundo da cena                     *)
PROCEDURE sir;
VAR robs   : RAIO;                  (* raio no SC_OBS                 *)
    rcena  : RAIO;                  (* raio no SC_CENA                *)
    x, y   : INTEGER;               (* coordenadas de um pixel da tela *)
    cor    : VETOR;                 (* cor de um pixel                *)
    linha  : ARRAY [XMIN..XMAX] OF VETOR;
                                    (* registra as cores dos pixels de
                                       uma linha da imagem            *)
    p_arv      : P_ARV_INTER;       (* ponteiro para nó da árvore de
                                       composição de sólidos          *)
INÍCIO
  Lê os arquivos de dados da cena;
  PARA y := YMAX ATÉ YMIN PASSO -1 FAÇA  (* linha da imagem *)
    INÍCIO
      PARA x := XMIN ATÉ XMAX PASSO 1 FAÇA  (* colunas da imagem *)
        INÍCIO
          Cria raio primário robs com visão perspectiva no SC_OBS;
          Transforma robs em rcena;
          p_lst_clas := 0;
          ray_tracing( p_sólido, rcena, p_lst_clas );
          SE p_lst_clas <> 0 ENTÃO  (* encontrou interseção *)
```

```
            INÍCIO
              (* Monta nó da árvore de interseções *)
              calcula_pi(p_arv^.pi,rcena);
              calcula_normal(p_arv^.n);
              calcula_uv(p_arv^.u,p_arv^.v);
              calcula_ctes_iluminação(p_arv);
              trace( 1, 1, rcena, p_arv );
              cor := [0,0,0];
              shade( p_arv, cor);
              linha[x] := cor;
            FIM
          SENÃO  (* não há ponto de interseção *)
            linha[x] := FUNDO;
        FIM;
      Salva linha no arquivo imagem;
    FIM;
FIM;



(* trace é responsável pelo lançamento de raios de iluminação e
secundários e pela continuação da montagem da árvore de interseções.*)
PROCEDURE trace( profundidade : INTEGER;
                      (* prof. da árvore de interseções *)
                 ind_ref      : REAL;
                      (* índice de refração do meio     *)
                 rcena        : RAIO;
                      (* raio no SC_CENA                *)
                 p_arv        : P_ARV_INTER );
                      (* ponteiro para nó da árvore
                         de interseções raio-sup.       *)
VAR  i : INTEGER;
     rfonte,          (* raio de iluminação  *)
     ref,             (* raio de reflexão    *)
     tra      : RAIO; (* raio de transmissão *)
INÍCIO
  (* lança raios de iluminação *)
  PARA i := 1 ATÉ NFONTE FAÇA
    INÍCIO
      Cria raio rfonte a partir de p_arv^.pi na direção da i-ésima
      fonte de luz;
      p_lst_clas := 0;
      ray_tracing(p_sólido, rfonte, p_lst_clas);
      SE p_lst_clas <> 0 ENTÃO  (* há interseções *)
        INÍCIO
          Verifica se a intensidade de luz que parte da fonte e chega
          ao ponto de interseção deve ser atenuada ou não;
          Armazena a intensidade de luz que chega ao ponto de
          interseção a partir da i-ésima fonte de luz;
        FIM;
    FIM;
  (* lança raio secundários *)
  SE profundidade < LIMITE_DE_PROFUNDIDADE_DA_ÁRVORE_DE_INTERSEÇÕES
  ENTÃO
    INÍCIO
      (* raio de reflexão *)
      SE p_arv^.ks > 0 ENTÃO  (* o sólido é reflexivo *)
        INÍCIO
```

```
          Cria raio de reflexão ref na direção de reflexão de luz;
          p_lst_clas := 0;
          ray_tracing( p_sólido,ref,p_lst_clas);
          SE p_lst_clas <> 0 ENTÃO  (* encontrou interseção *)
            INÍCIO
              (* Monta nó da árvore de interseções *)
              calcula_pi(p_arv^.pi,rcena);
              calcula_normal(p_arv^.n);
              calcula_uv(p_arv^.u,p_arv^.v);
              calcula_ctes_iluminação(p_arv);
              trace( profundidade + 1, ind_ref, ref, p_arv^.p_esp );
            FIM
        FIM;
      SE p_arv^.kt > 0 ENTÃO  (* o sólido transmite luz *)
        INÍCIO
          Cria raio de reflexão tra na direção de reflexão de luz;
          p_lst_clas := 0;
          ray_tracing( p_sólido,tra,p_lst_clas);
          SE p_lst_clas <> 0 ENTÃO  (* encontrou interseção *)
            INÍCIO
              (* Monta nó da árvore de interseções *)
              calcula_pi(p_arv^.pi,rcena);
              calcula_normal(p_arv^.n);
              calcula_uv(p_arv^.u,p_arv^.v);
              calcula_ctes_iluminação(p_arv);
              trace( profundidade + 1, p_arv^.p_tra^.indr, tra,
                     p_arv^.p_tra );
            FIM
        FIM;
    FIM
FIM;



(* shade é responsável pelo cálculo da cor de um ponto de interseção
segundo modelo de Whitted.
   o ponto de interseção é apontado por p_arv e a cor é retornada na
variável cor.
   ( l . p_arv^.n ) e ( h . p_arv^.n ) denotam o produto escalar entre
os vetores.
   As constantes IAMB_x definem a luz ambiente na cena                *)
PROCEDURE shade( p_arv   : P_ARV_INTER;
                   (* ponteiro para nó da árvore de interseções *)
                 VAR cor : VETOR );
                   (* cor do ponto de interseção *)
VAR  i : INTEGER;
     l : VETOR;       (* vetor na direção da fonte de luz          *)
     h : VETOR;       (* vetor entre o observador e a fonte de luz *)
     soma_r,          (* auxiliares p/ cálculo da luz difusa       *)
     soma_g,
     soma_b : REAL;
     soma_sr,         (* auxiliares p/ cálculo de luz especular    *)
     soma_sg,
     soma_sb : REAL;
     cor_ref,         (* luz vinda da direção especular            *)
    cor_tra : VETOR;(* luz vinda da direção de transmissão de luz*)
INÍCIO
  (* calcula a parcela de reflexão de luz ambiente                 *)
```

```
  cor[1] := IAMB_R * p_arv^.kar;  (* luz ambiente vermelha      *)
  cor[2] := IAMB_G * p_arv^.kag;  (* luz ambiente verde         *)
  cor[3] := IAMB_B * p_arv^.kab;  (* luz ambiente azul          *)
  (* calcula as parcelas de reflexão de luz difusa e especular *)
  soma_r := 0; soma_g := 0; soma_b := 0;     (* luz difusa      *)
  soma_sr := 0; soma_sg := 0; soma_sb := 0;  (* luz especular   *)
  PARA i := 1 ATÉ NFONTE FAÇA
    INÍCIO
      Calcula o vetor l na direção da i-ésima fonte de luz;
      soma_r := soma_r + ( l . p_arv^.n ) * int[i,1];
      soma_g := soma_g + ( l . p_arv^.n ) * int[i,2];
      soma_b := soma_b + ( l . p_arv^.n ) * int[i,3];
      calcula o vetor h entre a fonte e o observador;
      COM p_arv^ FAÇ
        INÍCIO
          soma_sr := soma_sr + ( h . p_arv^.n )^expn * int[i,1];
          soma_sg := soma_sg + ( h . p_arv^.n )^expn * int[i,2];
          soma_sb := soma_sb + ( h . p_arv^.n )^expn * int[i,3];
        FIM;
    FIM;
  cor[1] := cor[1] + soma_r * p_arv^.kdr;  (* luz difusa vermelha  *)
  cor[2] := cor[2] + soma_g * p_arv^.kdg;  (* luz difusa verde     *)
  cor[3] := cor[3] + soma_b * p_arv^.kdb;  (* luz difusa azul      *)
  cor[1] := cor[1] + soma_sr * p_arv^.ks;  (* luz espec. vermelha  *)
  cor[2] := cor[2] + soma_sg * p_arv^.ks;  (* luz espec. verde     *)
  cor[3] := cor[3] + soma_sb * p_arv^.ks;  (* luz espec. azul      *)
  (* contribuição de reflexão especular *)
  SE p_arv^.p_esp <> NIL ENTÃO  (* há nó de reflexão *)
    INÍCIO
      cor_ref := [0,0,0];
      shade(p_arv^.p_esp, cor_ref);
      cor[1] := cor[1] + cor_ref[1] * p_arv^.ks;
                                          (* luz espec. vermelha*)
      cor[2] := cor[2] + cor_ref[2] * p_arv^.ks;
                                          (* luz espec. verde   *)
      cor[3] := cor[3] + cor_ref[3] * p_arv^.ks;
                                          (* luz espec. azul    *)
    FIM;
  (* contribuição de transmissão de luz*)
  SE p_arv^.p_tra <> NIL ENTÃO  (* há nó de transmissão *)
    INÍCIO
      cor_tra := [0,0,0];
      shade(p_arv^.p_tra, cor_tra);
      cor[1] := cor[1] + cor_tra[1] * p_arv^.kt;
                                          (* luz trans. vermelha*)
      cor[2] := cor[2] + cor_tra[2] * p_arv^.kt;
                                          (* luz trans. verde   *)
      cor[3] := cor[3] + cor_tra[3] * p_arv^.kt;
                                          (* luz trans. azul    *)
    FIM;
FIM;
```

3.4- A CRIAÇÃO DE LOOKUP-TABLES PARA UMA IMAGEM

O LCG conta com um monitor Eltek-MA120 com resolução de 1024x780 pixels com 8 planos de memória e uma lookup-table de 256 cores. A intensidade de cada primária RGB da lookup-table varia entre 0 e 63, ou seja, o monitor torna disponível ao usuário um conjunto de 256k cores diferentes, dentre as quais 256 podem ser vistas simultaneamente na tela.

O SIR cria uma imagem numa resolução padrão de 480x640 pixels. Toda imagem criada por SIR é composta por três arquivos diferentes, sendo que cada arquivo é composto por uma das três primárias RGB que definem uma cor. Cada um desses arquivos é uma matriz de 480x640 bytes, onde cada byte representa um pixel. Com essa resolução uma imagem pode contar com milhares de cores diferentes (no máximo 480x640 = 307.200, o que é muito mais que a capacidade de representação de cores do monitor), com uma cor diferente por pixel. Como a lookup-table do monitor permite a visualização de apenas 256 cores ao mesmo tempo, é necessário selecionar as 256 cores que melhor representam a imagem para definir a lookup-table que será usada quando se quer mostrar a imagem no monitor.

A escolha da lookup-table mais adequada para cada imagem é fundamental para a apresentação da imagem com sucesso no monitor. A escolha das cores relaciona-se com as características de reflexão de luz de cada superfície da cena. Por exemplo, se uma cena contém superfícies amarelas e vermelhas a lookup-table deve ser preenchida com tons de amarelo e vermelho apenas, porque não é conveniente desperdiçar nenhuma posição da lookup-table com tons de outra cor.

A dificuldade de se reconhecer uma cor, dadas as três componentes primárias RGB que a definem, inviabiliza a utilização extensiva do sistema RGB por usuários. Por exemplo, qual é a tripla RGB que deve ser escolhida para representar a cor laranja clara ? A dificuldade de utilização do sistema RGB de cores nesta tarefa é o fato desse modelo ser orientado para o hardware. Em outras palavras, placas gráficas e monitores de vídeo são construídos utilizando diretamente esse sistema de cores.

O sistema HSV de cores, orientado para o usuário, permite reconhecer e controlar as cores que fazem parte da imagem. O sistema HSV foi criado com o objetivo de ser uma interface amigável entre usuários e sistemas gráficos. De fato, com um pouco de treino e experiência é fácil definir cores nesse sistema.

Heckbert [18] sugeriu dois métodos para escolha de lookup-tables para uma imagem. Ambos trabalham com o sistema de cores RGB. No LCG, aplicamos os métodos de Heckbert no sistema de cores HSV e, nas imagens produzidas até aqui, conseguimos uma apresentação de melhor qualidade no monitor colorido que a aplicação direta em RGB.

3.4.1- Os métodos de Heckbert para escolha de lookup-tables

Heckbert [18] discutiu dois métodos para seleção de cores de uma imagem com o objetivo de mostrar a imagem em monitores coloridos com lookup-tables. Ambos os métodos trabalham com o sistema RGB e, em ambos, uma amostragem da imagem é feita inicialmente para contagem de freqüencia das cores, utilizando-se para isso de uma matriz tridimensional que representa o sistema de cores RGB. Essa matriz pode ser definida com a declaração em Pascal dada abaixo, supondo que o valor de cada primária varie entre 0 e 31:

```
var rgb : array [0..31,0..31,0..31] of real;
```

A matriz acima ocupa $2^{15} * m$ posições de memória, onde m é o tamanho de um registro do tipo real. Se o registro do tipo real ocupa 4 bytes numa determinada implementação, a matriz de contagem ocuparia 128k de memória.

Heckbert desenvolveu seus métodos de seleção de cores partindo do princípio que o olho humano distingüe aproximadamente 50.000 cores. Heckbert, então, supôs que bastam 5 bits para cada primária para representar cores num monitor colorido, o que totaliza 32k cores diferentes, sem perda notável de qualidade nas imagens. Se a imagem é formada por primárias que podem variar além de 0 a 31, Heckbert faz um mapeamento da primária original para a faixa apropriada, onde cada

primária varia entre 0 e 31. Por exemplo, para mapear uma primária que pode receber valores entre 0 e 63 na faixa definida por Heckbert, basta dividir o valor da primária por 2.

O primeiro método de Heckbert, seleção por freqüência absoluta, simplesmente escolhe as cores que mais aparecem na imagem, ou seja, as cores que aparecem com maior freqüência na matriz de contagem, para montar a lookup-table. A desvantagem desse método é que cores e até mesmo subespaços do sistema RGB ocupados por cores que aparecem na imagem podem ficar sem representação na lookup-table. Por exemplo, dada uma imagem onde tons das cores vermelho e amarelo apareçam com grande freqüência e tons da cor azul tenha baixa freqüência, provavelmente poucos tons da cor azul serão representados na lookup-table e a qualidade da imagem deixará a desejar.

O segundo método, seleção por subdivisão pela mediana, mais elaborado e interessante, subdivide o espaço RGB formado pelas cores da imagem em um número de subespaços igual ao número de posições da lookup-table, de tal forma que pelo menos uma cor, das que são limitadas por um subespaço, está presente na imagem. Em cada subespaço encontra-se a cor média das cores limitadas pelo subespaço e que estão presentes na imagem. Essa cor média é então inserida na lookup-table da imagem. O processo de subdivisão pela mediana permite construir lookup-tables mais adequadas que o método anterior porque todas as cores que aparecem na imagem participam do cálculo da cor média de cada subespaço. Dado um subespaço no sistema RGB limitado pelas coordenadas (r_{min}, g_{min}, b_{min}) e (r_{max}, g_{max}, b_{max}), a subdivisão acontece da seguinte forma:

a- descobre qual é o maior eixo do subespaço verificando qual módulo de uma das seguintes subtrações é maior: $r_{max} - r_{min}$, $g_{max} - g_{min}$, $b_{max} - b_{min}$;

b- percorre o maior eixo do subespaço contando a freqüência de aparecimento das cores em cada coordenada do eixo, criando uma distribuição de freqüências;

c- encontra a freqüência total de cores do subespaço;

d- percorre o maior eixo do subespaço até encontrar a coordenada onde a freqüência acumulada de freqüências é mais próxima da mediana da distribuição de freqüências de cores no eixo maior do subespaço;

e- subdivide o subespaço na coordenada encontrada no passo anterior gerando dois novos subespaços.

O processo de subdivisão espacial mostrado acima, dado um subespaço inicial, continua até que sejam gerados 256 subespaços, um subespaço por posição da lookup-table. Procede-se então, em cada subespaço, ao cálculo da média das cores do subespaço que estão presentes na imagem. As médias encontradas são as cores que devem inicializar a lookup-table para a imagem.

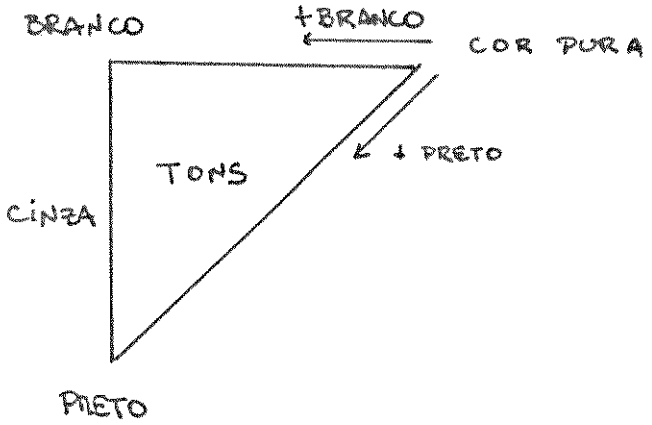

Fig. 3.3- O método de obtenção de tons de uma cor.

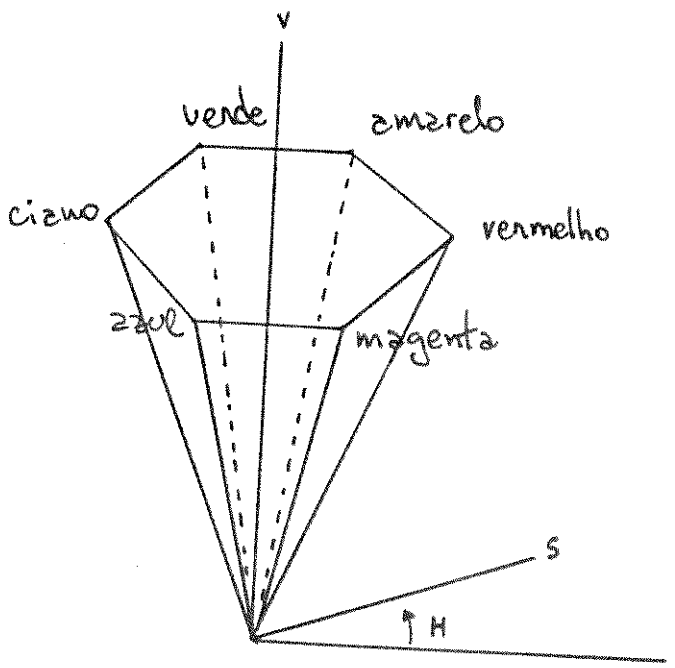

Fig. 3.4- O sistema HSV de cores.

3.4.2- O SISTEMA HSV DE CORES

O sistema HSV de cores foi proposto por Smith [32] e é orientado para facilitar o trabalho do usuário com cores, sendo baseado no processo de manipulação de cores por um artista.

Dada uma cor pura, o artista produz tons dessa cor adicionando a cor branca, a cor preta, ou ambas. A figura 3.3 ilustra esse processo. O triângulo da figura mostra como obter os tons de apenas uma cor. Colocando ao lado desse triângulo outros triângulos que representam outras cores é possível criar um sistema tridimensional de cores.

O sistema HSV (hue, saturation, value) é representado por um cone hexagonal cujo eixo representa os tons de cinza. A dimensão H define cores distintas e é orientada no sentido anti-horário, começando em 0^{o}, onde está a cor vermelho, fazendo a volta no eixo do cone até atingir 360^{o}. Fixando o valor H da tripla que representa uma cor no sistema HSV e variando S e V obtemos os tons da cor definida na dimensão H. A dimensão S indica a saturação do tom de uma cor. No eixo do cone a saturação é 0 indicando tom de cinza. Nas faces do cone a saturação é 1, o que indica cor pura. A dimensão V indica a intensidade luminosa da cor, sendo 0 no ápice do cone e 1 no topo. A figura 3.4 mostra o cone que representa o sistema HSV de cores.

Os tons originais de uma cor são definidos com S = 1 e V = 1, ou seja, são cores puras com intensidade máxima. Todos os outros tons de uma cor são obtidos a partir do tom original com a adição de branco ou preto. Adicionar branco a um tom de uma cor significa diminuir o valor de S desse tom, criando um novo tom da cor. Para adicionar preto a um tom de uma cor basta diminuir o valor de V desse tom , criando também um novo tom da cor.

Smith [32] e Foley [12] apresentam as rotinas que convertem uma cor dada no sistema RGB para o sistema HSV e vice-versa. Na versão de Smith e de Foley, quando uma cor tem valor V = 0, seu valor S pode ser qualquer entre 0 e 1 e seu H é indefinido; quando S = 0, V varia entre 0 e 1 e H é indefinido.

Em nossa versão dessas rotinas simplificamos essas relações com a finalidade de facilitar sua implementação. Quando V = 0, S é 0 e H é 0; e quando S = 0, V é 0 e H é 0. Assim, uma cor com saturação 0 é sempre um tom de cinza.

Os algoritmos para conversão RGB-HSV e HSV-RGB são apresentados a seguir.

```
PROCEDIMENTO rgb_para_hsv( r, g, b : REAL; VAR h, s, v : REAL );
   { entrada r, g, b : valores entre 0 e 1
     saída   h, s, v : h entre 0 e 360; s e v entre 0 e 1 }
INÍCIO
  max := MÁXIMO(r,g,b); { maior valor entre r, g, b }
  min := MÍNIMO(r,g,b); { menor valor entre r, g, b }
  v := max;
  SE max = 0 ENTÃO
    INÍCIO
      h := 0;  s := 0;  v := 0;
    FIM
  SENÃO
    INÍCIO
      s  := ( max - min ) / max;
      rc := ( max - r   ) / ( max - min );
      gc := ( max - g   ) / ( max - min );
      bc := ( max - b   ) / ( max - min );
      SE r = max ENTÃO h := bc - gc
      SENÃO
      SE g = max ENTÃO h := 2 + rc - bc
      SENÃO
      SE b = max ENTÃO h := 4 + gc - rc;
      h := h * 60;
      SE h < 0 ENTÃO h := h + 360;
    FIM
FIM; { rgb_para_hsv }



PROCEDIMENTO hsv_para_rgb( h, s, v : REAL; VAR r, g, b : REAL);
   { entrada h, s, v : h entre 0 e 360; s e v entre 0 e 1;
     saída   r, g, b : entre 0 e 1 }
INÍCIO
  SE s = 0 ENTÃO
    INÍCIO
      (r,g,b) := (v,v,v);  { tom de cinza }
    FIM
  SENÃO
    INÍCIO
      SE h = 360 ENTÃO h := 0;
      h := h / 60;
      i := FLOOR( h );  { maior inteiro <= h }
      f := h - i;
      p := v * ( 1 - s );
      q := v * ( 1 - ( s * f ) );
      t := v * ( 1 - ( s * ( 1 - f ) ) );
```

```
        CASO i É
        0: (r,g,b) := (v,t,p);
        1: (r,g,b) := (q,v,p);
        2: (r,g,b) := (p,v,t);
        3: (r,g,b) := (p,q,v);
        4: (r,g,b) := (t,p,v);
        5: (r,g,b) := (v,p,q);
        FIM_DE_CASO;
    FIM
FIM; { hsv_para_rgb }
```

3.4.3- MÉTODOS PARA SELEÇÃO DE CORES EM HSV

Os métodos desenvolvidos para selecionar cores que representem uma imagem na lookup-table do monitor colorido baseiam-se nos métodos sugeridos por Heckbert e utilizam o sistema de cores HSV para contagem da freqüência de cores de uma imagem.

A estrutura de dados do método é uma matriz tridimensional chamada FHSV utilizada para contar a freqüência de aparecimento de cores na imagem. FHSV representa 36 cores possíveis (dimensão H) e 900 tons de cada cor, ou seja, 30 saturações (dimensão S) e 30 intensidades diferentes (dimensão V), o que permite representar um total de 32.400 tons diferentes. Partindo da mesma suposição de Heckbert, ou seja, que, bastam 5 bits por cor primária para representar todas as cores distingüíveis pelo olho humano, a matriz FHSV parece ser o suficiente para escolher lookup-tables de qualidade. A matriz FHSV pode ser declarada em Pascal da seguinte forma:

```
var fhsv : array [0..35,0..29,0..29] of real;
```

A quantidade de memória ocupada por essa matriz é aproximadamente 128k bytes.

É preciso notar que as dimensões da matriz FHSV podem ser aumentadas se isso se fizer necessário para melhorar a qualidade das lookup-tables.

No sistema HSV de cores os valores de S e V variam entre 0 e 1. Porém, no monitor colorido, uma cor com S menor que 0.2 parece um tom de cinza e uma cor com V menor que 0.2 é muito escura. Devido a esse

fato, uma cor da imagem com S e V maiores que 0.2 incrementa a posição apropriada da matriz FHSV mas uma cor com S e V menores que esses limites contada como um nível de cinza.

À matriz FHSV é adicionado um plano específico para contagem do número de tons de cinza da imagem onde H = 0, S = 0, e V é variável. Nesse plano, portanto, apenas uma coluna é utilizada pois S é sempre constante.

Esta particularidade é uma das vantagens da utilização do sistema HSV para seleção de cores porque elimina de antemão um conjunto de cores que, por serem muito escuras ou acinzentadas, podem ser substituídas por tons de cinza sem perda de qualidade da imagem. Naturalmente, se for o caso, os limites de saturação e valor podem ser modificados de acordo com a necessidade e a imagem, aumentando ou diminuindo o subespaço HSV que entra em cogitação na hora da seleção das cores. A possibilidade de controlar cores com uma característica específica também pode ser utilizada para destacar áreas de interesse de uma imagem. Por exemplo, todas as cores com determinada saturação de uma imagem podem ser associadas à cor vermelho. Dessa forma, com a associação de padrões de cor a certas áreas da imagem, tratamentos interessantes podem ser feitos na imagem.

A dimensão H da matriz FHSV foi dividida de forma que uma cor represente 10 graus da dimensão H do sistema HSV. Os intervalos começam em 355^{o} e a cor média de cada intervalo é escolhida para representar todo o intervalo. A figura 3.5 mostra a divisão da dimensão H em intervalos de 10^{o}.

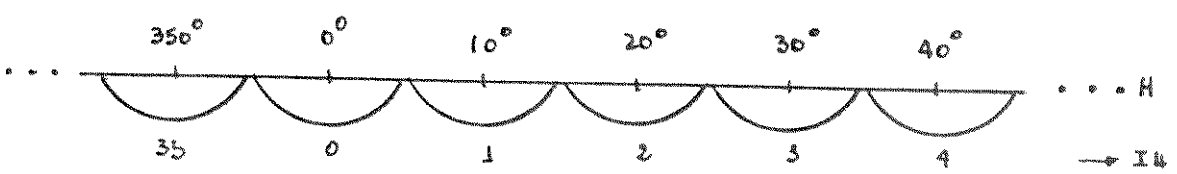

Fig. 3.5- Mapeamento da dimensão H do sistema HSV.

A principal vantagem da aplicação dos métodos de Heckbert no sistema de cores HSV é a possibilidade do emprego dos métodos em cada plano H que define uma cor ao invés de aplicá-los no espaço

tridimensional de cores. Com HSV é possivel calcular a porcentagem de aparecimento de uma cor definida por uma das coordenadas do eixo H da matriz FHSV numa imagem e construir a lookup-table de acordo com essa porcentagem, o que não pode ser feito no sistema RGB. Por exemplo, se 20% das cores que aparecem numa imagem são tons da cor azul, então 20% da lookup-table é inicializada com tons de azul. Essa característica do sistema HSV permite escolher lookup-tables de melhor qualidade que as escolhidas com o sistema RGB.

O procedimento que cria a lookup-table para uma imagem por freqüência absoluta em HSV é dado abaixo:

a- a cor de cada pixel da imagem é lido dos arquivos que representam a imagem no sistema RGB de cores; faz-se a conversão de cada tripla (r,g,b) que define a cor do pixel numa tripla (h,s,v);

b- os valores h, s e v da cor do pixel são utilizados como índices para incrementar a freqüência do tom de uma cor ou de um nível de cinza na matriz FHSV, indicando o número de vezes que o tom de uma cor aparece na imagem e o número de vezes que a cor (somatória das freqüências dos tons da cor) aparece na imagem;

c- o número de tons diferentes na imagem é calculado verificando-se na matriz FHSV as posições com freqüência diferente de 0; basta que um tom de uma cor tenha freqüência diferente de 0 na FHSV para que a cor esteja presente na imagem;

d- calcula-se a porcentagem de aparecimento de cada cor na imagem dividindo-se o número de tons diferentes da cor com freqüência diferente de 0, pelo número total de tons diferentes presentes na imagem;

e- multiplica-se a porcentagem de aparecimento de cada cor por 256 (número de posições da lookup-table) calculando-se o número de posições da lookup-table que devem ser inicializadas com tons dessa cor;

f- os tons de cada cor são escolhidos para preencher as posições da lookup-table alocadas para a cor de acordo com sua freqüência na matriz FHSV;

g- transforma a tripla (h,s,v) correspondente ao tom escolhido para fazer parte da lookup-table para o sistema RGB;

A desvantagem desse método é que, da mesma forma que o método de seleção por freqüência de Heckbert em RGB, cores esparsas e subspaços do sistema HSV podem ficar sem representante selecionado.

A aplicação do método de subdivisão pela mediana de Heckbert no sistema HSV produz lookup-tables de melhor qualidade que o método anterior por dois motivos:

a- a lookup-table é produzida de acordo com a porcentagem de aparecimento de cada cor na imagem, ou seja, o processo de subdivisão acontece no plano que define uma cor e não no sistema tridimensional como sugerido por Heckbert;

b- os tons de uma mesma cor são produzidos como a média de uma região do sistema HSV e não escolhidas de acordo com uma distribuição local como no caso da escolha por freqüência absoluta.

3.5- Mostrar imagens no monitor

O programa desenvolvido para mostrar imagens no monitor é bastante simples e utiliza a lookup-table gerada pelo programa de criação de lookup-tables, e os arquivos que representam a imagem.

O procedimento para mostrar a imagem no monitor é o seguinte:

a- inicializa a lookup-table do monitor com a lookup-table no sistema RGB criado para a imagem pelo programa de criação de lookup-tables;

b- cada pixel é lido dos arquivos que representam a imagem; pesquisa se a lookup-table criada para a imagem procurando a cor mais próxima da cor que representa o pixel;

c- o índice da cor mais próxima da cor do pixel, encontrada na lookup-table, é atribuído ao pixel e enviado para o monitor.

O procedimento acima foi utilizado para mostrar no monitor de vídeo imagens com a resolução padrão de 480x640 pixels.

Dada uma cor representada pela tripla (r, g, b) de um pixel da imagem, a busca da cor mais próxima na lookup-table é feita através do cálculo em cada posição da lookup-table da seguinte equação:

$$D_i = (r_i - r)^2 + (g_i - g)^2 + (b_i - b)^2, \quad \text{onde}$$

i é um índice da lookup-table, (r_i, g_i, b_i) é uma tripla que representa uma cor da lookup-table e D_i é a distância entre a cor do pixel e a i-ésima cor da lookup-table.

Após ter sido feito o cálculo das distâncias entre a cor do pixel e cada cor da lookup-table, basta procurar a menor distância encontrada. O índice correspondente a essa distância aponta a cor da lookup-table que deve representar o pixel no monitor colorido.

3.6- O FORMATO DOS ARQUIVOS UTILIZADOS NO AMBIENTE

Vários arquivos são manipulados pelos programas que compõem este ambiente com o objetivo de sintetizar uma imagem.

Uma característica comum entre os arquivos é o fato de serem relativos a uma única imagem. Devido a esse fato todos os arquivos têm o mesmo nome, sendo que os sufixos são diferentes.

Os arquivos utilizados na síntese de uma imagem são dados a seguir, por sufixo:

a- ".CSG";
b- ".CAM";
c- ".LUZ";
d- ".R";
e- ".G";
f- ".B";
g- ".LUT".

A descrição desses arquivos é feita nas seções seguintes.

3.6.1- Arquivo ".CSG"

Este arquivo contém a descrição da cena modelada em CSG e também o número que identifica cada uma das superfícies que constituem os sólidos primitivos que compõem a cena. Todas estas informações podem ser conferidas pelo usuário porque o arquivo é listável.

Este arquivo é produzido pelo modelador de sólidos e é lido pelo programa SIR.

O formato deste arquivo é o seguinte:

a- número de nós que compõem a árvore de composição de sólidos;
b- (linha em branco);
c- número do nó;
d- nome do nó;
e- tipo do nó (COMPOSTO ou PRIMITIVO).

Se o tipo do nó é COMPOSTO o formato é o seguinte:

a- tipo da operação de composição (UNIÃO, DIFERENÇA ou INTERSEÇAO);
b- tipo do nó esquerdo (PRIMITIVO ou COMPOSTO);
c- tipo do nó direito (PRIMITIVO ou COMPOSTO);
d- número do nó esquerdo;
e- número do nó direito;
f-(linha em branco).

Se o tipo do nó é PRIMITIVO o formato é o seguinte:

a- tipo do nó primitivo (CUBO, ESFERA, CONE ou CILINDRO);
b- tipo da superfície do sólido primitivo;
c- matriz de transformação LOCAL_CENA;
d- (linha em branco).

3.6.2- Arquivo ".CAM"

Este arquivo contém as informações necessárias para o programa SIR montar o modelo de câmera para visualização da cena. Essas informações são o tamanho da janela de visualização, o ponto alvo da câmera, a distância da câmera, a direção normal da câmera e o vetor

inclinação da câmera. A obtenção do modelo de câmera fotográfica, bem como cada um dos parâmetros listados acima são explicados detalhadamente em [25] e não serão abordados aqui.

Atualmente este arquivo é montado por um editor de textos comum devido a sua simplicidade.

O formato de ".CAM" é:

a- janela de visualização;
b- alvo da câmera;
c- direção normal da câmera;
d- distância entre a câmera e o alvo;
e- vetor de inclinação da câmera.

3.6.3- Arquivo "LUZ"

Este arquivo contém informações relativas à iluminação da cena. Essas informações são: cor do fundo, intensidade da luz ambiente e o número, posição e intensidade das fontes de luz.

Este arquivo é montado com o auxílio de um editor de textos e tem o seguinte formato:

a- cor do fundo;
b- intensidade da luz ambiente;
c- número de fontes de luz que iluminam a cena.

Para cada fonte de luz temos:

a- posição e intensidade da fonte de luz.

3.6.4- Os arquivos que representam a imagem

Os arquivos ".R", ".G" e ".B" armazenam as três componentes primárias que formam a cor de um pixel da imagem. Cada um desses arquivos é uma matriz com 480x640 bytes (resolução padrão).

3.6.5- O arquivo lookup-table

O ambiente cria um arquivo para representar a lookup-table da imagem: ".LUT. O arquivo ".LUT" contém a lookup-table no sistema RGB e

é utilizado para inicializar a lookup-table do monitor de vídeo em tempo de mostrar a imagem.

O formato do arquivo ".LUT" é uma lista de 256 linhas com três valores inteiros representando, cada tripla, uma cor.

CAP. 4- CONCLUSÕES E COMENTÁRIOS FINAIS

4.1- INTRODUÇÃO

O esforço principal de nosso programa de mestrado foi o desenvolvimento, no LCG / DCA, de um ambiente para síntese de imagens realísticas baseado na metodologia ray-tracing.

O ambiente desenvolvido compõe-se de programas para:

a- modelagem de cenas com o método CSG,
b- visualização de cenas com algoritmo ray-tracing,
c- escolher lookup-tables para uma imagem e
d- mostrar a imagem no monitor colorido.

O ambiente desenvolvido mostrou-se eficaz no sentido de permitir ao pesquisador cumprir todos os passos necessários para a síntese de imagens com o método ray-tracing. No anexo 1 vemos uma amostra das imagens produzidas até aqui.

O ambiente foi construído de forma modular com a previsão de modificações nos programas básicos que o compõem para aumentar sua capacidade de simulação da realidade.

Nas próximas seções fazemos comentários a cerca dos aspectos principais dos programas que compõem o ambiente e sugerimos modificações para a continuidade das pesquisas na área de síntese de imagens realísticas com o método ray-tracing.

4.2- O MODELADOR DE SÓLIDOS

O modelador de sólidos é responsável pelo modelamento de cenas com o método CSG e utiliza as funções sugeridas por Roth em seu excelente artigo sobre ray-tracing [31].

O modelador mostrou-se adequado para modelar cenas simples, com poucos sólidos primitivos, onde não há a necessidade de muita visualização durante as diversas etapas de modelagem.

A função de visualização do modelador deixa muito a desejar por apresentar uma resposta muito lenta às requisições do usuário. A lentidão no fornecimento de apoio visual à modelagem por parte do modelador acontece por duas razões:

a- a função de visualização utiliza o algoritmo ray-tracing para cenas modeladas com o método CSG, o qual é um algoritmo lento;
b- o modelador foi implementado em microcomputador PC-XT, o qual não tem uma capacidade de processamento à altura das exigências do método ray-tracing.

Além da lentidão, a função de visualização deixa a desejar por mostrar ao usuário apenas as silhuetas dos sólidos, o que não permite um entendimento claro da posição e do relacionamento espacial dos diversos sólidos presentes numa cena.

O modelador de sólidos ocupa uma posição estratégica num ambiente para síntese de imagens realísticas. É o modelador que determina se uma cena é viável para ser modelada e que fornece o ferramental necessário para a modelagem. Um modelador com uma visualização de cenas que deixa a desejar, seja pela lentidão de resposta, seja pela qualidade da imagem produzida, inviabiliza a criação de cenas complexas e mais interessantes pelo usuário.

Embora o modelador de sólidos não seja o centro de nosso trabalho, é bastante clara a necessidade de aumentá-lo com uma função de visualização mais eficiente. Essa função poderia, por exemplo, transformar inicialmente cada sólido primitivo que faz parte da cena num conjunto de polígonos. A função de visualização utilizaria, então, um algoritmo de eliminação de superfícies escondidas para mostrar a cena ao usuário.

4.3- O PROGRAMA PARA SÍNTESE DE IMAGENS

O programa para síntese de imagens realísticas (SIR) utiliza cenas modeladas com o método CSG para produzir imagens com objetos transparências, reflexos especulares e sombras.

Este programa pode ser facilmente modificado, como decorrência da sua estrutura modular bem definida, para aumentar a sua capacidade de

simulação de efeitos luminosos e melhorar seu desempenho computacional, no sentido de diminuir o tempo de execução do programa.

Entre as modificações podemos citar:

a- lançamento de mais de um raio por pixel como uma forma de tratar o aliasing,
b- aumento do número de tipos de sólidos primitivos que o programa é capaz de reconhecer,
c- inclusão de esquemas para aumentar o tempo de resposta do programa como subdivisão espacial e
d- inclusão da técnica de perturbação da normal para simular ranhuras nas superfícies da cena.

O anexo 1 mostra um conjunto de imagens sintetizada com o modelador. Essas imagens mostram o potencial do método ray-tracing embutido no SIR.

4.4- Escolha de lookup-tables

Quando se fala em imagens realísticas em computador, subentende-se uma imagem com milhares, e até milhões, de cores diferentes, a qual pode até mesmo ser confundida com uma imagem de uma cena real. Na verdade, sintetizar imagens com essa qualidade é o nosso objetivo. Porém, os equipamentos disponíveis atualmente no LCG / DCA limitam seriamente nosso trabalho.

O LCG / DCA dispõe de um monitor colorido cuja placa de controle permite a visualização de apenas 256 cores simultaneamente na lookup-table. Essa limitação impõe a escolha das 256 cores que melhor representem a imagem no monitor, embora as imagens sintetizadas com o SIR sejam formadas por milhares de cores diferentes. Tal escolha prejudica seriamente a qualidade das imagens produzidas pelo SIR, as quais deveriam apresentar mudanças contínuas nas cores de uma superfície e, no entanto, mostram mudanças de tonalidades notáveis como pode ser observado nas imagens do anexo 1.

O ambiente desenvolvido no LCG / DCA permitiu-nos aplicar o método de escolha de lookup-tables por subdivisão pela mediana sugerido por Heckbert [18] no sistema HSV e compará-lo com a aplicação original no sistema RGB.

A tabela 4.1 mostra o resultado dessa comparação para as imagens vistas no anexo 1. As imagens foram produzidas com a resolução padrão de 480x640 pixels. O tempo de execução dos programas de escolha de lookup-tables com aplicação em cada sistema de cores é dado em minutos e segundos.

IMAGEM	RGB NÚMERO DE CORES	RGB TEMPO	HSV NÚMERO DE CORES	HSV TEMPO
WHITTED	867	1:55	489	1:25
CALICE	723	1:03	410	1:19
LUMINAR	400	1:36	206	1:12
PIRÂMIDE	7619	2:44	3950	1:26
LCG	1398	2:08	865	1:24

Tabela 4.1- Comparação de métodos de escolha de lookup-tables.

O tempo de execução do método de Heckbert para escolha de lookup-tables aplicado ao sistema HSV é quase sempre menor que o tempo do método quando aplicado em RGB.

A qualidade das imagens no monitor, o ponto que mais nos interessa, é muito parecida. Porém, como pode ser visto na imagem Whitted no anexo 1, a lookup-table escolhida para o sistema RGB descaracterizou completamente a imagem, enquanto que a lookup-table escolhida com a aplicação do sistema HSV permitiu uma visualização correta.

Levando em consideração que a aplicação do método de Heckbert no sistema HSV pode ter a matriz de contagem da freqüência de cores aumentada, o que permite a verificação de outras cores e tons de cores escondidos pelo mapeamento que é feito com esse sistema, parece-nos mais interessante empregar o método de Heckbert com HSV para escolher a lookup-table para uma imagem.

4.5- Considerações finais

O desenvolvimento do ambiente para síntese de imagens realísticas baseado na metodologia ray-tracing no LCG / DCA gerou um sistema

básico que deixa espaço para muitos melhoramentos e modificações com o objetivo de se conseguir um maior realismo nas imagens produzidas.

Esperamos que o ambiente aqui desenvolvido sirva como um incentivo para outros pesquisadores interessados em ray-tracing e em síntese de imagens de modo geral.

CAP. 5- REFERÊNCIAS BIBLIOGRÁFICAS COMENTADAS

A metodologia ray-tracing é uma área de intensas pesquisas atualmente. Devido à qualidade das imagens geradas com esta técnica, um grande número de pesquisadores em todo o mundo se dispôs a pesquisar e a propor soluções para os problemas inerentes ao método.

Neste capítulo listamos uma coleção de referências que tratam da metodologia ray-tracing e de assuntos relacionados, tais como: modelos de iluminação, metodologias anti-aliasing, texturas, etc.

Fazemos um pequeno comentário abaixo de cada referência, onde descrevemos seu conteúdo principal. Esse comentário servirá ao pesquisador ou estudante interessado como um guia numa coleta inicial de artigos para estudo sobre a matéria.

A coleção de referências que listamos não é completa, isto é, não contém a totalidade do que foi publicado a respeito da metodologia ray-tracing. No entanto, acreditamos que esta lista de referências é representativa no sentido de abarcar os principais artigos publicados e de cobrir os principais aspectos técnicos da metodologia ray-tracing já discutidos.

Tomamos o cuidado de marcar com um * o início do comentário das referências não comentadas no corpo desta dissertação.

A seguir, as referências.

[1] Amanatides, J. Ray tracing with cones. Computer Graphics 18(3):129-135,jul., 1984.

* Ray-tracing de polígonos e esferas com raios em forma de cone, e não de reta como na metodologia usual. Esta técnica é voltada para o tratamento anti-aliasing da imagem.

[2] Blinn, J.F. Simulation of wrinkled surfaces. Computer Graphics 12(3):286-292,ago., 1978.

Este é um artigo clássico a respeito da modelagem de rugosidades em superfícies através da perturbação da normal à superfície no ponto onde o raio de luz incide.

[3] Blinn, J.F. e Newell, M.E. Texture and reflection in computer generated images. Communications ACM 19(10):542-547,oct., 1976.

Outro artigo clássico sobre a aplicação de texturas em superfícies.

[4] Bouatouch, K. e outros A new algorithm of space tracing using a CSG model. Anais da Eurographics Amsterdam, Holanda, 1987.

Subdivisão espacial tridimensional da cena em subespaços. Cada subespaço contém uma pequena parte do banco de dados que representa a cena. Inicialmente, calcula-se qual subespaço é atravessado primeiro pelo raio. Nesse subespaço calcula-se a interseção do raio com a cena. Se não houver interseção do raio com a cena nesse subespaço o próximo subespaço atravessado pelo raio é verificado e, dessa forma até que o raio intersecte uma superfície ou atravesse todos os subespaços. A utilização desta técnica aumenta a eficiência da metodologia ray-tracing quando aplicada à síntese de imagens de cenas complexas. Esta técnica de subdivisão espacial foi utilizada em cenas modeladas com CSG.

[5] Bouville, C., Dubois, J.L. e Marchal, I. Generating high quality pictures by ray-tracing. Computer Graphics Forum 4(2):87-99,jun., 1985.

* Um modelo de iluminação e um algoritmo para interseção de superfícies de revolução.

[6] Bronsvoort, W.F., Wijk, J.J.V. e Jansen, F.W. Two methods for improving the efficiency of ray casting in solid modelling. Computer Aided Design 16(1):51-55,jan., 1984.

Técnica 1: diminuição de uma árvore de composição de sólidos em árvores temporárias de acordo com a projeção dos sólidos primitivos na tela da cena. Técnica 2: refinamento passo a passo da imagem, isto é, inicialmente são lançados poucos raios e, nas vizinhanças entre raios onde uma diferença de cores é notável, raios intermediários são lançados.

[7] Clark, J.H. Hierarchical geometric models for visible surface algorithms. Communications ACM 19(10):547-554,oct., 1976.

Proposta de utilização de hierarquias e de volumes de encaixotamento para melhorar o desempenho de algoritmos para eliminação de superfícies escondidas em geral.

[8] Cleary, J.G. e outros Multiprocessor ray tracing. Computer Graphics Forum 5(1):3-12, 1986.

Proposta de um algoritmo ray-tracing baseado em multiprocessador com estudo de desempenho.

[9] Cook, R.L., Porter, T. e Carpenter, L. Distributed ray tracing. Computer Graphics 18(3):137-145,jul., 1984.

* Distribuição probabilística da direção dos raios de reflexão e de transmissão para simular borrão de movimento, reflexões aleatórias, penumbra, etc.

[10] Crow, F.C. The aliasing problem in computer-generated shaded images. Communications ACM 20(11):799-805,nov., 1977.

* Artigo clássico sobre aliasing e anti-aliasing.

[11] Dippé, M. e Swensen, J. An adaptative subdivision algorithm and parallel architecture for realistic image synthesis. Computer Graphics 18(3):149-158,jul., 1984.

Subdivisão tridimensional de uma cena e carga de subespaços em processadores paralelos visando melhorar o desempenho de ray-tracing.

[12] Foley, J.D. e Vam Dam, A. Fundamentals of interactive computer graphics. Reading, Mass., Addison-Wesley, 1982.

Livro introdutório sobre a Computação Gráfica. Discute diversos aspectos da matéria incluindo modelos de iluminação e sistemas de cores.

[13] Glassner, A.S. Space subdivision for fast ray tracing. IEEE Computer Graphics and Applications 4(10):15-22,oct., 1984.

Uma técnica de subdivisão espacial de cenas. Procura diminuir o número de cálculos de interseções entre o raio e as superfícies que compõem a cena.

[14] Goldstein, R.A e Nagel, R. 3-D visual simulation. Simulation 16(1):25-31,jan., 1971.

Primeira implementação de ray-tracing e sugestão do método CSG. Essa implementação só suportava objetos opacos.

[15] Haines, E. Essential ray tracing algorithms. Introduction to ray tracing course notes Curso 13, Siggraph, 1987.

Apresenta algorimos detalhados para cálculo da interseção entre raio e primitivas básicas como plano, esfera, cone e cilindro. No mesmo artigo, algoritmos para mapeamento das primitivas num sistema de coordenadas bidimensional visando a aplicação de texturas.

[16] Hall, R.A. e Greenberg, D.P. A testbed for realistic image synthesis. IEEE Computer Graphics and Applications 3(8):10-20,nov., 1983.

Discussão do ambiente para síntese de imagens realísticas implementado na Universidade Cornell. Lista um conjunto de melhoramentos técnicos conseguidos através do ambiente, incluindo a proposta de um modelo de iluminação.

[17] Hanrahan, P. Ray tracing algebraic surfaces. Computer Graphics 17(3):83-90,jul., 1983.

* Derivação automática das equações polinomiais de interseção entre um raio e superfícies definidas como polinômios.

[18] Heckbert, P. Color image quantization for frame buffer display. Computer Graphics 16(3):297-307, jul., 1982.

Discussão e comparação de técnicas para escolha de lookup-tables para uma imagem.

[19] Heckbert, P. Survey of texture mapping. IEEE Computer Graphics and Applications 6(11):56-67,nov., 1986.

Um apanhado geral de técnicas de mapeamento de texturas e de técnicas de filtragem para anti-aliasing.

[20] Heckbert, P. Ray tracing bibliography. Introduction to ray tracing course notes Curso 13, Siggraph, 1987.

* Uma colotânea de referências publicadas sobre ray-tracing até maio de 1987. Também é comentada e mais completa que a nossa.

[21] Heckbert, P. Derivation of refraction formulas. **Introduction to ray tracing course notes** Course 13, Siggraph, 1987.

Mostra como computar as fórmulas de refração de três modos diferentes, enfatizando métodos mais interessantes do ponto de vista computacional.

[22] Kajiya, J.T. Ray tracing parametric patches. **Computer Graphics** 16(3):245-254,jul., 1982.

* Algoritmo para encontrar a interseção de um raio com superfícies definidas parametricamente.

[23] Kajiya, J.T. New techniques for ray tracing procedurally defined objects. **Computer Graphics** 17(3):91-102,jul., 1983.

* Interseção de raio com superfícies definidas por fractais, prismas e superfícies de revolução.

[24] Kay, T.L. e Kajiya, J.T. Ray tracing complex scenes. **Computer Graphics** 20(4):269-278,ago., 1986.

* Volume de encaixotamento definido por planos arbitrariamente distanciados. Tentativa de sucesso para melhorar o desempenho do método ray-tracing.

[25] Newman, W.M. e Sproull, R.F. **Principles of interactive computer graphics.** New York, McGraw-Hill, 1979.

Livro introdutório sobre Computação Gráfica. Inclui a discussão de algoritmos para eliminação de superfícies escondidas e modelos de iluminação.

[26] Peachey, D.R. Portray - an image synthesis system for realistic computer graphics. **Introduction to ray tracing course notes** Curso 13, Siggraph, 1987.

* Excelente descrição do sistema para síntese de imagens realísticas desenvolvido por Peachey, baseado no algoritmo de Roth.

[27] Phong, B.T. Illumination for computer generated pictures. **Communications ACM** 18(6):311-317,jun., 1975.

Proposta de um modelo de iluminação para simular reflexão difusa e brilhos intensos de luz numa superfície. O modelo de iluminação de

Phong foi posteriormente aumentado por Whitted para simular reflexos especulares e transmissão de luz.

[28] Plunket, D.J. e Bailey, M.J. The vectorization of a ray tracing algorithm for improved execution speed. IEEE Computer Graphics and Applications 5(8):52-60,ago., 1985.

Implementação do algoritmo de Roth em computador vetorial.

[29] Requicha, A.A.G. Representations for rigid solids: theory, methods and systems. Computing Surveys 12(4):437-464,dec., 1980.

Artigo clássico que discute vários métodos existentes para modelagem de sólidos, incluindo CSG.

[30] Rogers, D.F. Procedural elements for computer graphics. New York, McGraw-Hill, 1985.

Livro dedicado à síntese de imagens realísticas. Inclui a discussão de técnicas raster, algoritmos para eliminação de superfícies escondidas, ray-tracing, modelos de iluminação, texturas, aliasing e anti-aliasing e sistemas de cor entre outros tópicos de interesse da área.

[31] Roth, S.D. Ray casting for modeling solids. Computer Graphics and Image Processing 18(2):109-144, fev., 1982.

Artigo clássico e básico a respeito de ray-tracing. Propõe e especifica detalhadamente o algoritmo que discutimos nesta dissertação.

[32] Smith, A.R. Color gamut transform pairs. Computer Graphics 12(3):12-19,ago., 1978.

Definição dos modelos HSV e HLS para trabalho com cores. Algoritmos de conversão entre os sistemas RGB e HSV.

[33] Sweeney, M.A.J. e Bartels, R.H. Ray tracing free-form B-spline surfaces. IEEE Computer Graphics and Applications 6(2):41-49,fev., 1986.

* Algoritmo para cálculo de interseção entre raio e superfícies definidas por B-splines.

[34] Toth, D.L. On ray tracing parametric surfaces. Computer Graphics 19(3):171-179,jul., 1985.

* Outro algoritmo para cálculo de interseções entre raio e superfícies paramétricas.

[35] Villaça, A.M. Um ambiente para síntese de imagens realísticas baseado na metodologia ray-tracing. Anais do II Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens (II SIGGRAPI) Águas de Lindóia, São Paulo, abril de 1989.

Descreve o sistema para síntese de imagens realísticas desenvolvido no LCG da FEE/UNICAMP. Ênfase no processo de escolha de lookup-tables.

[36] Weghorst, H.G., Hooper, G. e Greenberg, D.P. Improved computational methods for ray tracing. ACM Transactions on Graphics (1):52-69,jan., 1984.

Técnicas para melhorar o desempenho da metodologia ray-tracing. Inclui volumes de encaixotamento, hierarquias e preprocessamento de superfícies visíveis pelo observador.

[37] Wijk, J.J.V. Ray tracing objects defined by sweeping a sphere. Computer and Graphics 9(3):283-290, 1985.

* Cálculo da interseção de raio com objeto definido como uma esfera de raio variável numa trajetória definida por polinomial.

[38] Whitted, T. An improved illumination model for shaded display. Communications ACM 23(6):343-349,jun., 1980.

Neste artigo clássico, os princípios básicos da metodologia ray-tracing foram definidos. Whitted aumenta o modelo de iluminação de Phong incluindo o cálculo das contribuições luminosas devido a reflexão especular e transmissão de luz. Derivação de fórmulas que permitem a criação de raios de luz refletidos especularmente e transmitidos com refração.