Introdução ao Ray Tracing

Fernando Wagner Serpa Vieira da Silva
Laboratório de Computação Gráfica - LCG
COPPE / UFRJ - Engenharia de Sistemas e Computação
e-mail: nando@lcg.ufrj.br

Abstração

Os estudos avançados na área de Computação Gráfica têm demonstrado bastante interesse no que diz respeito ao foto-realismo, ou seja, geração de imagens com alto grau de realismo. A técnica de Ray Tracing oferece uma ferramenta simples e poderosa para esse estudo.

Introdução

A Computação Gráfica se divide, basicamente, em duas áreas: Modelagem e Visualização. A primeira está relacionada com a construção dos objetos da cena a ser visualizada, utilizando bases matemáticas para isso. A segunda busca uma forma de representação visual dos objetos construídos. Neste artigo apresentaremos um poderoso método de visualização chamado Ray Tracing.

Nas primeiras seções deste artigo introduziremos os conceitos de Ray Tracing. Nas seções seguintes, falaremos sobre reflexão, transparência, sombras e outros efeitos produzidos pelo Ray Tracing. Falaremos também sobre modelos de iluminação e antialiasing.

Seção 1 - Os Fundamentos do Ray Tracing

Atualmente, existem diversos algoritmos de visualização de superfícies e sólidos, porém nenhum deles é tão simples e tão poderoso quanto o Ray Tracing. O algoritmo se baseia numa idéia muito simples: um observador se senta em frente a uma tela plana transparente. De seus olhos partem diversos "raios visuais" que vão atravessar os pontos da tela e bater nos objetos tridimensionais, que foram definidos utilizando-se alguma técnica de modelagem[1]. Pintamos, então, o ponto da tela que foi atravessado pelo raio com a cor do objeto que foi atingido por este. Esta é a forma mais simples de se apresentar o algoritmo de Ray Tracing. Variações mais avançadas que incluem sombras, reflexão e transparência serão apresentadas adiante. Em termos algorítmicos, poderíamos apresentar o Ray Tracing da seguinte forma:

Algoritmo básico de Ray Tracing :

```
Para cada ponto da tela
• Calcule uma linha reta unindo o olho do observador a este
  ponto
• Descubra as interseções desta reta com os objetos 3D que
  estão atrás da tela
• Pinte o ponto com a cor do objeto mais próximo
```

Alguns aspectos interessantes devem ser analisados:

[1] Nos exemplos mostrados neste artigo foi utilizada a modelagem CSG (Constructuve Solid Geometry).

1. O algoritmo de Ray Tracing gasta entre 75% e 95% [Whitted 80] de seu tempo determinando as interseções com os objetos, por isso, a eficiência da rotina de interseção raio-objetos afeta significativamente a eficiência do algoritmo. Atualmente existem hardwares que fazem estes cálculos com extrema rapidez.

2. Os objetos da cena a ser visualizada são descritos sob a forma de estruturas de dados. Essas estruturas têm uma certa forma, dependendo do método de modelagem[2] a ser utilizado.

3. Quando o raio visual atinge um objeto visível[3], o ponto da tela a ser pintado possui características do ponto do objeto que foi atingido, mas não é necessariamente da cor do objeto. Diversos fatores influem no cálculo da cor do ponto, como a iluminação, por exemplo. Se girarmos um objeto, a impressão visual da cor de um ponto deste muda, ficando mais escuro à medida que a luz que incide sobre ele fica mais perpendicular em relação à normal no ponto. A textura e rugosidade do objeto são fatores que também influem no cálculo da sua cor.

4. O observador (ou câmera[4]) é constituído de um ponto focal (ou olho) e de uma tela (ou plano de projeção) composta de pequenos retângulos. Esta tela de retângulos geralmente é a própria tela do monitor e os retângulos são os seus pixels. Os objetos visíveis, ou cena, estão em frente à câmera, dentro da pirâmide de visão da figura 2. Os raios visuais que são lançados passam pelos retângulos do plano de projeção e ligam o ponto focal à cena. Quando o ponto focal está localizado no infinito, os raios tornam-se paralelos e então a pirâmide de visão torna-se o paralelepípedo de visão da figura 1.

5. Como podemos perceber, os raios lançados no método de Ray Tracing seguem o caminho inverso ao da luz. Daí surge a pergunta: por que não lançá-los na direção tradicional? A resposta é simples: de uma fonte luminosa partem bilhões de fótons, que são os formadores dos raios luminosos. Porém apenas uma pequeníssima parte deles chega aos olhos de um observador. Simular tal mecanismo seria impraticável para o computador, fazendo que uma simples cena levasse anos para ser renderizada. Lançando raios a partir do observador, garantimos que estaremos calculando apenas os raios que efetivamente serão captados pelo observador.

[2] B-Rep, CSG e Octree são exemplos.

[3] No algoritmo de Ray Tracing, o objeto é visível quando o raio que o atinge não interceptou nenhum objeto opaco antes dele.

[4] Este modelo de câmera também é conhecido como *pinhole camera*.

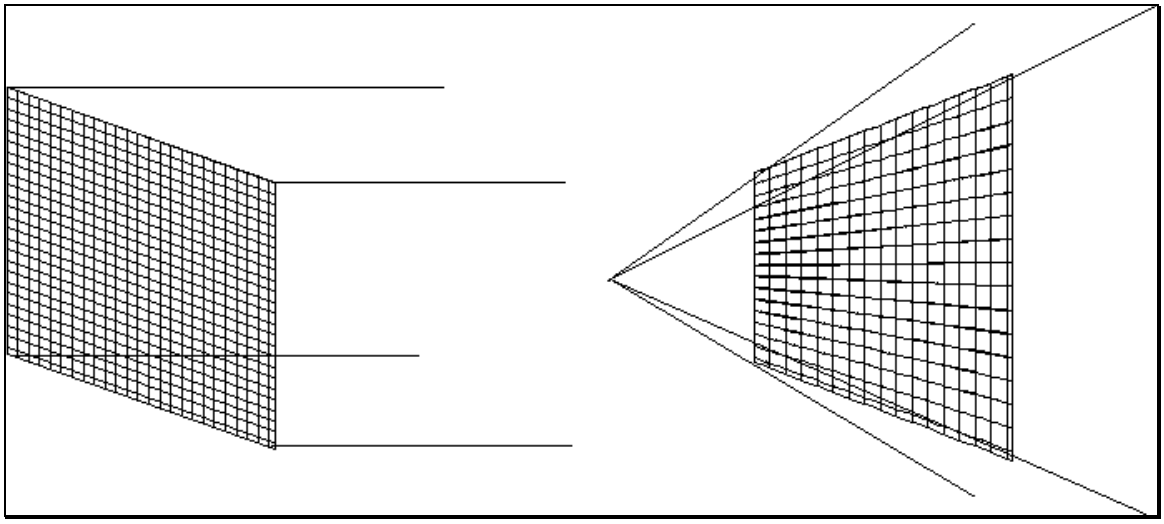


Figura 1 - Paralelepípedo de visão

Figura 2 - Pirâmide de visão

A figura 3 nos mostra imagens geradas por um ray tracer que utiliza o algoritmo descrito acima. A versão simplificada do algoritmo de Ray Tracing é conhecida também como Ray Casting[5].

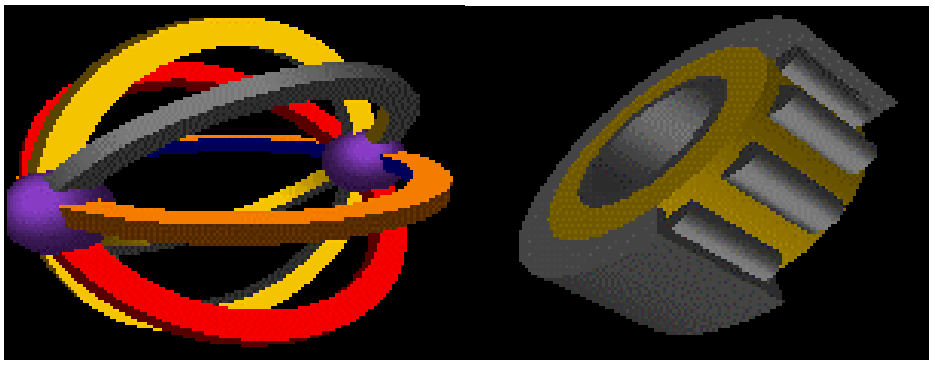


Figura 3 - Cenas visualizadas por Ray Tracing simples (Ray Casting)

1.1 - O Mecanismo de Interseção Raio-Cena

O raio visual é simplesmente uma linha reta no espaço 3D do observador. Para fins de implementação é descrito na sua forma paramétrica de um ponto inicial (x_0 , y_0 , z_0) e um ponto final (x_1 , y_1 , z_1), e um vetor diretor[6] (D_x , D_y , D_z) = (x_1 - x_0 , y_1 - y_0 , z_1 - z_0). Desta maneira, os pontos (x, y, z) desta linha podem ser acessados via um parâmetro t , ou seja:

[5] Uma variante 2D do Ray-Casting é amplamente utilizada em jogos que simulam ambientes 3D.

[6] Para simplificar os cálculos no computador, costuma-se utilizar o vetor diretor na sua forma normalizada, ou seja, Norma = Sqrt ($D_x^2 + D_y^2 + D_z^2$) = 1 .

$x = x_0 + t \,.\, D_x$
$y = y_0 + t \,.\, D_y$
$z = z_0 + t \,.\, D_z$

Para calcularmos a interseção do raio com uma superfície *S*, basta exprimir a superfície na sua forma algébrica convencional, substituir a equação do raio na equação de *S* e calcular as raízes desta equação segundo a variável *t*. Uma vez obtido *t*, pode-se facilmente calcular o ponto de interseção (x_i , y_i ,z_i) usando o método descrito anteriormente.

Nota : Alguns exemplos de interseção de raios com sólidos (ou primitivas) e superfícies podem ser encontrados em [Roth82].

Alguns sólidos são particularmente interessantes de serem utilizados pelos usuários de Ray Tracing, pelos efeitos bonitos de reflexão e refração que produzem. Citamos abaixo alguns deles, com suas respectivas equações algébricas:

Esfera : $X^2 + Y^2 + Z^2 = 1$
Cone : $X^2 + Y^2 - Z^2 = 0$ com $X^2 + Y^2 \leq 1$ e $0 \leq Z \leq 1$
Toro : $(X^2 + Y^2 + Z^2 + 1 - r^2) - 4 \,.\, (X^2 + Y^2) = 0$ com $0 \leq r \leq 1$

No caso de uma esfera, por exemplo, o cálculo de interseção com o raio pode nos fornecer três resultados distintos:

1. O raio não interceptou a esfera.
2. O raio interceptou a esfera em um só ponto, ou seja, encontramos apenas a raiz t_0 na equação da variável *t* (nesse caso o raio é tangente à esfera)[7].
3. O raio interceptou a esfera em 2 pontos, ou seja, encontramos duas raízes t_0 e t_1.

Observe agora os seguintes fatos:

- O intervalo [t_0 , t_1] se localiza no interior da esfera.
- O ponto visível é aquele que tem o valor de *t* menor, ou seja, no caso do item 3, o Min(t_0 , t_1). Isso significa que este ponto é o mais próximo do observador.
- Se um dos valores de *t* for menor que zero, isto significa que a interseção ocorreu em um ponto antes do ponto de origem do lançamento do raio e, portanto, deve ser descartado. Este caso ocorre, por exemplo, quando o observador está localizado no interior da esfera.
- Para calcular a coordenada de interseção, basta substituir o valor da raiz *t* na equação paramétrica do raio.

[7] Nesse caso descartamos esta interseção, para facilitar os cálculos. Essa é uma situação muito rara e que pode causar erros de ponto flutuante.

Se os objetos da cena forem descritos por meio de polígonos, o processo é semelhante. Por exemplo, se o plano que contém o polígono for da forma $ax + by + cz + d = 0$, então sua interseção com o raio paramétrico será dada por:

$$t = -\frac{ax_0 + by_0 + cz_0 + d}{aD_x + bD_y + cD_z} \qquad (1.1)$$

Agora basta determinar se a interseção com o plano se deu no interior do polígono. Um método simples é verificar se a soma dos ângulos entre as linhas traçadas do ponto a cada vértice do polígono for 360^0. Caso afirmativo, o ponto de interseção ocorreu no interior do polígono. Note que se o denominador em (1.1) for zero, então o raio e o plano são paralelos, não havendo então interseção.

Em termos algorítmicos, a rotina de interseção raio-sólido poderia ter o seguinte formato:

```
      .
      .
      .
Para cada superfície do sólido
{
Resolva a equação de interseção raio-superfície
Se achar pontos de interseção com a superfície
    Então guarde os parâmetros do raio que interceptou a
superfície
}
```

Devemos ter em mente que a estrutura de lançamento dos raios visuais deve possuir, basicamente, 2 listas:

1. Parâmetros do raio t[1] , ... , t[n]
2. Apontadores para superfícies s[1] , ... , s[n]

Onde *n* é o número de interseções raio-sólido. A primeira lista (ordenada) contém os pontos de entrada e saída do raio no sólido. O raio entra no ponto t[1], sai em t[2], entra em t[3], sai em t[4], e assim por diante até, finalmente, sair em t[n]. O ponto t[1] é o mais próximo da câmera e o ponto t[n] o mais distante. Em associação com os parâmetros do raio, existe uma lista de ponteiros para as superfícies as quais o raio intercepta. A lista de superfícies contém informações importantes para a etapa de cálculo da iluminação da cena.

Existem ainda algumas aplicações bastante interessantes dos mecanismos de lançamento de raios e de interseções. Por exemplo, o volume da cena modelada pode ser facilmente calculado pela seguinte fórmula:

$$\text{Volume} = \sum_{j=1}^{j=n} Vraio(j)$$

Onde $Vraio(j) = S1 \cdot S2 \cdot \left(t[2] - t[1] + t[4] - t[3] + \ldots + t[n] - t[n-1]\right) \cdot L$

Os raios que foram lançados irão particionar a cena em elementos de volume. Duas das dimensões destes elementos são constantes, definidas pelas dimensões S1 e S2 dos pixels do plano de projeção. A terceira dimensão será definida pelos parâmetros de entrada e saída dos raios (t[1], t[2], ... , t[n]), definidos na lista.

L é o comprimento do vetor diretor do raio traçado em questão, ou seja, $L = \sqrt{Dx \cdot Dx + Dy \cdot Dy + Dz \cdot Dz}$. Vale a pena ressaltar que *n* é o número de interseções do raio com o sólido e que cada $\left(t[i] - t[i-1]\right) \cdot L$ é o comprimento de um segmento de raio no interior do sólido.

Na verdade, o que estamos fazendo é aproximar (ou discretizar) a cena modelada por um conjunto de paralelepípedos retangulares. O cálculo de outras propriedades físicas como centro de massa e momentos de inércia também seguem esta mesma linha de raciocínio.

OBS: Por razões matemáticas, o algoritmo para cálculo de volume descrito acima só terá resultados satisfatórios quando estivermos utilizando a projeção ortográfica.

1.1.1 - Sólidos Limitantes (Bounding Volumes)

Como vimos anteriormente, a interseção do raio com os objetos da cena é bastante trabalhosa para o computador. Muitas vezes (na verdade, na maioria das vezes), quando é lançado um raio, este não intercepta nenhum dos objetos da cena, mas mesmo assim são feitos todos os cálculos de interseção do raio com as superfícies dos objetos.

Para evitar estas interseções desnecessárias, fazemos a interseção do raio com um sólido limitante, antes de interceptá-lo com a superfície do objeto. Este sólido limitante geralmente é uma esfera (bounding sphere) ou um bloco (bounding box), pois possuem equações simples, que não são muito trabalhosas para o cálculo no computador.

A idéia é simples: se o raio não intercepta o sólido limitante do objeto, então este raio não interceptará o objeto e, então, deve ser descartado. Assim, eliminamos as interseções desnecessárias. O uso de sólidos limitantes pode aumentar em até 40% a velocidade de renderização das cenas.

Note que podem também ser definidas hierarquias entre os sólidos limitantes. Por exemplo: dois toros, cada um contido em seu sólido limitante, podem ser envolvidos por um outro sólido limitante que engloba os dois anteriores. Assim, se um raio não interceptar o sólido limitante que engloba os dois toros, não haverá a necessidade de interceptá-lo com os sólidos limitantes de cada um dos toros. Observe que o processo é recursivo, ou seja, se para um raio *r* o teste com o nó *x* da árvore de hierarquia dos sólidos

limitantes falhar, então todos os filhos de *x* na árvore não precisarão ser testados com relação a *r*.

Seção 2 - Ray Tracing Recursivo

O algoritmo completo de Ray Tracing é formado por diversas chamadas recursivas. Tal recursão é necessária para produzir os efeitos de reflexão, sombra e transparência. A implementação desses efeitos não requer grandes esforços de programação. Para um estudo mais profundo veja [Rogers85] e [Kay79].

2.1 - Sombra

Quando trabalhamos com fontes pontuais de luz, o efeito de sombra aparece em um objeto quando existe um outro objeto opaco entre o primeiro e a fonte de luz.

A idéia é lançar um outro raio, chamado "raio de sombra", que une o ponto do objeto que foi atingido ao ponto de luz. Se entre o ponto e a luz existir um outro objeto, este ponto estará na sombra, ou seja, será pintado com a intensidade de luz ambiente.

Note que neste caso estamos considerando somente uma única fonte de luz. Se existirem diversas fontes de luz na cena, o processo deve ser repetido para cada uma delas antes de decidir se um objeto está completamente na sombra[8]. Veja abaixo a descrição algorítmica do processo.

```
Para cada ponto da tela
```
- `Calcule uma linha reta unindo o olho do observador a este ponto`
- `Descubra as interseções desta reta com os objetos 3D que estão atrás da tela`
- `Calcule o ponto mais próximo`
- `Calcule uma linha reta unindo este ponto à fonte de luz`
- `Se algum objeto opaco é interceptado pela linha que ligou o ponto à fonte de luz, então pinte o ponto com a intensidade de luz ambiente, senão pinte o ponto com a cor normal do objeto`

[8] Neste caso podem aparecer as regiões de umbra e penumbra.

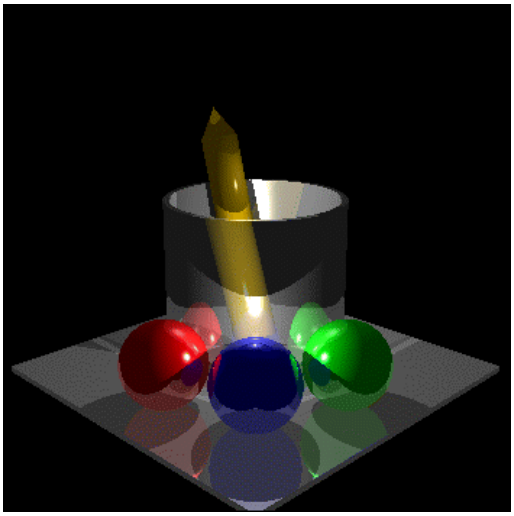


Figura 4 - Sombra

Devemos analisar também o caso de entre a luz e o ponto da superfície do objeto interceptado pelo raio existir um objeto transparente. Nesse caso, devemos "atenuar" a sombra pelo fator de transparência do objeto no caminho da luz. Fazemos isto pois o objeto transparente deixa passar parcelas de luz, que devem ser levadas em consideração no cálculo da iluminação.

2.2 - Reflexão

Quando executamos Ray Tracing em uma superfície refletora (um espelho, por exemplo), o raio que foi lançado bate nesta superfície e é refletido segundo a lei da ótica, que diz que o raio refletido tem a mesma direção do raio que incidiu sobre a superfície. Assim, em termos de algoritmo, é como se fosse lançado um novo raio visual a partir deste ponto, só que na direção de reflexão. Este ponto terá a cor calculada a partir do raio refletido.

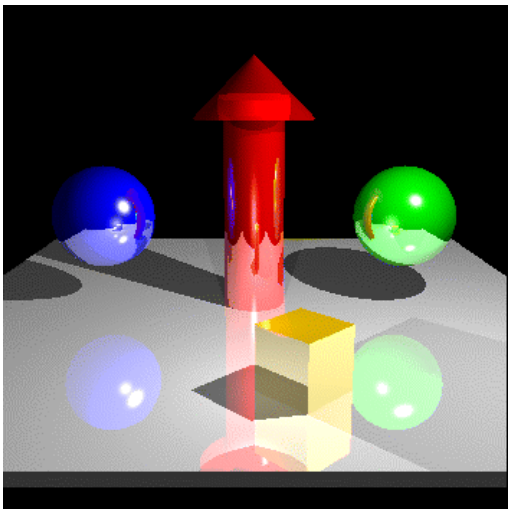


Figura 5 - Reflexão

2.3 - Transparência

Ao atingir um objeto transparente com índice de refração N_1, a luz atravessa a superfície e é desviada num fenômeno chamado refração, que é regulado pela lei de Descartes-Snell:

$$\frac{N1}{N2} = \frac{sen\theta 2}{sen\theta 1}$$

Onde N_1 é o índice de refração do objeto atravessado pelo raio, N_2 é o índice de refração do meio ou objeto de onde o raio se originou, θ_1 é o ângulo de incidência e θ_2 é o ângulo de refração.

A cor do ponto é calculada a partir da cor obtida por um novo raio lançado com a direção refratada (T). O cálculo do vetor T é mostrado na seguinte fórmula:

$$T = \left(N_{21} \cdot (N \cdot I) - \sqrt{1 - \left(N_{21}\right)^2 \cdot \left(1 - (N \cdot I)^2\right)} \cdot N - N_{21} \cdot I \right)$$

Onde :

$$N_{21} = \frac{N_2}{N_1}.$$

N = Normal à superfície no ponto atingido pelo raio.

I = Vetor de incidência da luz.

Devemos ficar atentos à reflexão interna total, fenômeno que ocorre quando a raiz quadrada utilizada no cálculo de *T* se torna um número complexo. Nesse caso, devemos lançar um raio de reflexão na direção de *T*, que será a direção de reflexão interna total.

Objetos transparentes funcionam como filtros. Por exemplo, um objeto verde transparente iluminado por uma fonte de luz deixará passar apenas luz verde, absorvendo todas as componentes de vermelho e azul. Por isso, se tal objeto estiver diante de uma parede, sua sombra na parede aparecerá esverdeada. Para conseguir tal efeito, basta atenuar a luz que incidiu sobre o objeto levando em conta as componentes RGB da cor do objeto.

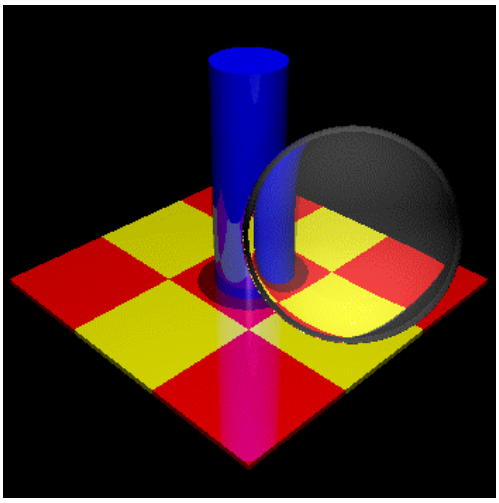


Figura 6 - Transparência

Seção 3 - Iluminação

Um grande problema na geração de imagens foto-realistas é o cálculo da interação da luz com os objetos existentes na cena. Até hoje, não conseguimos desenvolver um modelo computacional de iluminação que consiga descrever perfeitamente esse fenômeno. Quando conseguirmos isso, teremos dado um grande passo rumo ao realismo total.

Dentre os diversos modelos de iluminação existentes, destacamos o de Phong [Phong73]. Nele, consideramos três componentes de iluminação: Ambiente, difusa e especular. A componente ambiente (I_a) é constante em toda a cena. A componente difusa segue a lei de Lambert, que relaciona a intensidade de luz refletida pela superfície ao coseno do ângulo α, formado pela normal à superfície (*N*) e pelo vetor (*L*), que liga o ponto a ser iluminado com a fonte luminosa (ver figura 7). Observe a equação:

$$I_d = I_{fonte} \cdot k_d \cdot \cos\alpha$$

Onde :

I_{fonte} = Intensidade da fonte luminosa.
k_d = Coeficiente de reflexão difusa (varia de material para material).

A componente especular é dada pela "contribuição de Phong", uma fórmula empírica que relaciona a intensidade especular a uma potência do coseno do ângulo β, formado pela direção de reflexão (*R*) e pelo vetor (*V*), que liga o ponto da superfície ao observador (figura 7).

Com todas essas informações, podemos calcular a intensidade luminosa de um ponto da cena através da seguinte equação de luminosidade:

$$I = I_a \cdot k_a + \sum_{i=1}^{i=m} I_{fonte\_i} \cdot \left[k_d \cdot \cos\alpha + k_{esp} \cdot \cos^n \beta \right]$$

Onde

k_a = Coeficiente de iluminação ambiente.
m = Número total de fontes luminosas presentes na cena.
$I_{fonte\text{-}i}$ = Intensidade da *i-ésima* fonte luminosa.
k_{esp} = Coeficiente de reflexão especular (varia de material para material).
n = Potência do cosseno. Quanto maior n, mais refletora é a superfície.

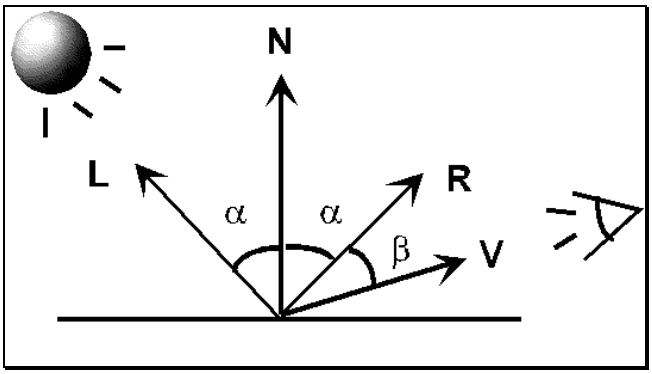


Figura 7 - O modelo de iluminação de Phong

Em termos vetoriais, a equação de luminosidade poderia ser escrita da seguinte maneira:

$$I = I_a \cdot k_a + \sum_{i=1}^{i=m} I_{fonte\_i} \cdot \left[k_d \cdot \left(N \cdot L_i \right) + k_{esp} \cdot \left(R_i \cdot V \right) \right]$$

É importante notar que não estamos incluindo a cor do objeto no calculo da equação de luminosidade. Para um objeto cuja cor possua componentes *Or*, *Og* e *Ob* teremos três equações, uma para cada componente de cor. Veja o exemplo para a componente vermelha:

$$I_r = I_a \cdot k_a \cdot Or + \sum_{i=1}^{i=m} I_{fonte\_i} \cdot \left[k_d \cdot \left(N \cdot L_i \right) \cdot Or + k_{esp} \cdot \left(R_i \cdot V \right) \right]$$

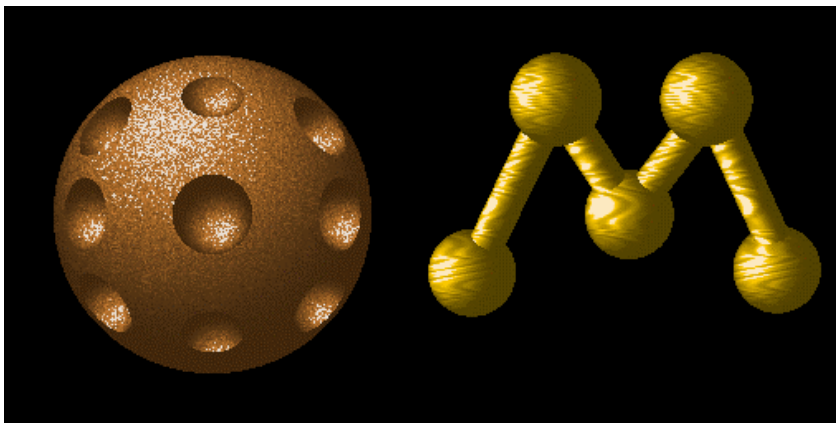
3.1 - Bump Mapping

Este interessante e elegante método de simular superfícies rugosas foi desenvolvido por Blinn [Blinn78]. Ele partiu da observação de que o aspecto rugoso de uma superfície dependia da variação da direção da normal em relação a esta. A idéia é simples: sabendo a forma da normal **N** de uma superfície, utilizamos um vetor de perturbação **D**, que irá "sacudir" a direção da normal, dando-lhe o aspecto rugoso. Teremos então uma normal **N'**, que é descrita por:

$$\mathbf{N'} = \mathbf{N} + \mathbf{D}$$

O vetor de perturbação **D**, é conseguido através da utilização de uma função (Bump Map), que dará a característica da deformação da superfície. Uma visão mais ampla (e matemática) do assunto pode ser encontrada em [Watt93]. É possível conseguir resultados fantásticos com Bump Mapping. Atualmente existem funções que simulam até pêlos na superfície de objetos!

Veja, na figura 8, alguns exemplos de objetos com diferentes tipos de "texturas" conseguidas com Bump Mapping.



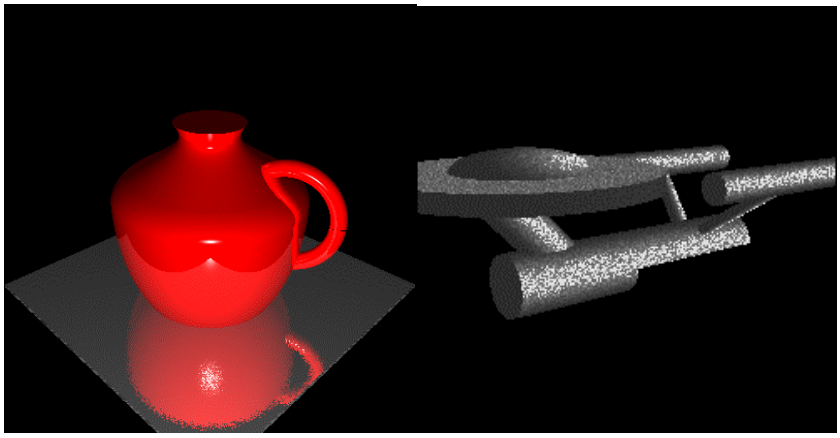


Figura 8 - Bump Mapping

Seção 4 - Antialiasing

Ao gerarmos imagens no computador, notamos o desagradável efeito "escada". Este fenômeno, conhecido como aliasing, acontece principalmente porque nossos dispositivos de vídeo e nossos computadores possem uma resolução limitada. Por exemplo: quando desenhamos um círculo na tela, o que estamos fazendo na verdade é aproximar por um conjunto limitado de pixels uma entidade geométrica que possui infinitos pontos, ou seja, estamos fazendo uma discretização do círculo.

Um outro exemplo: quando gravamos uma música no computador, estamos transformando um sinal contínuo (o som) em uma sequência de números (os bits). Como a precisão do computador é finita, temos que selecionar uma quantidade finita de números (fazer uma amostragem) para representar o som. Se o número for grande, haverá uma boa representação digital da música e, consequentemente, uma melhor qualidade de reprodução. Se não conseguirmos uma boa amostragem, a música apresentará falhas e ruídos.

Lançar um raio é, essencialmente, um procedimento de amostragem. Os raios lançados têm largura zero, e assim, atingem os objetos em pontos mais ou menos espalhados. Além disso os raios lançados não se espalham, mas continuam, de interseção em interseção, com a mesma "finura". Como resultado, surge o aliasing[9].

Para minimizar os efeitos de aliasing, devemos lançar mais raios por pixel, calculando intensidades de "sub-pixels", e depois calcular uma média aritmética ou ponderada destes valores. Fica claro que esse processo aumenta geometricamente o tempo de processamento.

Para acelerar o processo, podemos selecionar aqueles pixels que precisam ser melhor calculados. Uma idéia, portanto, é calcular todos os pixels com apenas um raio e, imaginando que os pixels que devem ser melhorados são aqueles em cuja vizinhança ocorra uma grande mudança de cor, varrer a imagem descobrindo os pixels com essas características e então lançar mais raios.

Após o lançamento desses novos raios, é interessante executar algum procedimento de filtragem na imagem, para dar-lhe uma aparência mais uniforme. O processo descrito acima é geralmente chamado de super-amostragem (*supersampling*) e é facilmente incorporado a uma programação simples de Ray Tracing.

Em animações, o efeito aliasing nos objetos é pouco percebido pelo fato de os mesmos estarem em movimento na cena. Mas as animações sofrem outro tipo de aliasing, o aliasing temporal. Um exemplo: em filmes, quando a câmera focaliza uma roda girando, a impressão que se tem é de que ela está rodando mais lento do que deveria, e às vezes parece até que ela roda ao contrário. Isso se deve porque a frequência em que a roda gira é bem maior do que a frequência de amostragem do filme (cerca de 30 quadros por segundo). Para minimizar o efeito de aliasing temporal é utilizada a técnica de Motion Blur, que será descrita adiante.

[9] Um método muito interessante de antialiasing para Ray Tracing pode ser encontrado em [Wyvill85].

Seção 5 - Técnicas Avançadas de Ray Tracing

Nesta seção apresentaremos uma coletânea de técnicas mais avançadas de Ray Tracing, que possuem uma implementação mais trabalhosa.

5.1 - Ray Tracing Distribuído

A intensidade de um pixel na tela é uma função analítica que pode envolver integrais bastante complexas. O cálculo destas integrais são fundamentais para a geração de certos fenômenos como Motion Blur (integral do tempo), Antialiasing (integral da região do pixel), Deph of Field (integral da área da lente) e outros. Os algoritmos de rendering tradicionais não levam isso em conta e portanto só conseguem produzir *sharp shadows*, *sharp reflections*, câmeras do tipo *pinhole*, etc.

A técnica de Ray Tracing distribuído avalia as integrais descritas acima lançando vários raios numa região determinada, segundo uma distribuição estocástica. Na verdade o que estamos fazendo é uma avaliação de Monte Carlo das integrais pelo conjunto de raios lançados.

A intensidade *I* de um ponto em uma superfície é dada pela integral sobre o hemisfério acima da superfície de uma função de iluminação *L* e de uma função de reflexão *R*.

$$I(\phi_r, \theta_r) = \int_{\phi_i} \int_{\theta_i} L(\phi_i, \theta_i) \cdot R(\phi_i, \theta_i, \phi_r, \theta_r) d\phi_i d\theta_i$$

Onde (ϕ_i, θ_i) é o ângulo de incidência da luz na superfície e (ϕ_r, θ_r) é o ângulo de reflexão da luz na superfície. Os algoritmos tradicionais evitam avaliar esta integral fazendo as seguintes suposições:

- *L* é uma função delta (δ), ou seja, *L* é zero exceto para as direções da fonte de luz, que é tratada como sendo pontual. A integral é então substituída por uma soma. Esta suposição causa as sombras cerradas.
- Todas as direções que não são relativas à fonte de luz são agrupadas formando uma intensidade ambiente constante na cena. Assim, *L* se torna independente de θ_i e ϕ_i e pode ser retirada da integral. Deste modo, a integral de *R* pode ser substituída por uma reflectância ambiente. Esta suposição não permite o cálculo de interação da luz entre objetos. O método de Radiosidade é um dos poucos que trata corretamente este caso.
- *R* é uma função (δ), ou seja, a superfície é um espelho perfeito e reflete a luz somente na direção de reflexão. Esta suposição dá as cenas um aspecto demasiadamente reflexivo e causa reflexões cerradas.

O método da Ray Tracing distribuído evita estas simplificações tomando amostras da função através da distribuição dos raios. Os raios de iluminação não são mais traçados somente na direção da fonte de luz, mas são distribuídos de acordo com a função de iluminação *L*. E os raios não são mais refletidos somente na direção especular, mas também são distribuídos, de acordo com a função de reflectância *R*. A seguir veremos alguns efeitos conseguidos com Ray Tracing distribuídos.

5.1.1 - Penumbra

O efeito de penumbra ocorre quando uma fonte de luz extensa é parcialmente obstruída. Seu cálculo exato se faz por meio do ângulo sólido da porção visível da luz, porém achar este ângulo sólido é muito complicado. Como vimos anteriormente, as sombras podem ser calculadas lançando-se raios secundários ligando os pontos da superfície à fonte de luz. Para o cálculo de penumbras, estes raios secundários devem ser distribuídos e lançados de acordo com a área relativa da fonte de luz.

5.1.2 - Reflexão Borrada

Em Ray Tracing, a reflexão é obtida traçando-se raios na direção de reflexão. Para obter o efeito de reflexão borrada (blurry reflection) basta distribuir os raios de reflexão sobre a direção de reflexão. A distribuição é ponderada de acordo com a mesma função de distribuição que determina os highlights. Os highlights são as regiões de maior intensidade luminosa nos objetos, e são produzidos por raios que refletem a própria fonte luminosa.

5.1.3 - Translucência

Para obter a transparência, novos raios devem ser lançados na direção de refração da luz no objeto. Na translucência, os raios devem ser distribuídos de acordo com a direção principal da luz transmitida. Tal distribuição é definida pela função de transmissão especular.

5.1.4 - Antialiasing

Antialiasing pode ser obtido através do lançamento de diversos raios por pixel, distribuídos estocasticamente pela região do pixel. Ao final do lançamento dos raios para cada pixel, deve-se calcular uma média das intensidades e então pintar o pixel de acordo com o resultado obtido.

5.2 - Ray Tracing Parametrizado

O que torna o algoritmo de Ray Tracing lento é o cálculo das interseções dos raios primários e secundários (reflexão, transparência e sombra) com os objetos da cena. Tais cálculos consomem cerca de 90% do tempo de processamento da cena no computador. Qualquer esforço para reduzir ou evitar tais cálculos acarretará em grande aumento de velocidade de renderização.

A técnica de Ray Tracing parametrizado baseia-se na observação de que os caminhos percorridos pelos raios luminosos traçados pelo algoritmo tradicional dependem apenas de características geométricas dos objetos da cena e das luzes[10]. Deste modo, se a forma e posição dos objetos, e a posição da câmera e das luzes não forem alteradas, então podemos evitar o cálculo das interseções, pois elas são independentes de modificações como cor do objeto, coeficientes de superfície, cor da textura, intensidade das fontes de luz, etc.

A idéia é renderizar a cena, guardando, para cada pixel da imagem, a árvore de raios que contém todas as informações geométricas utilizadas. Após esta primeira renderização, podemos modificar os parâmetros ópticos dos objetos (cor, textura, coeficientes especular, difuso, transparente, etc.), obtendo então uma nova imagem apenas reavaliando na árvore de raios de cada pixel as equações de shading. Fica claro que um dos problemas deste método é a grande quantidade de memória necessária para armazenar as árvores de raios de cada pixel da cena.

O uso de Ray Tracing parametrizado é útil para depurar interativamente uma cena, ajustando as intensidades de luzes, cor dos objetos e qualquer outro parâmetro óptico que se queira sem precisar calcular novamente as interseções dos raios com os objetos. Isto aumenta a velocidade de renderização em até 150 vezes [Santos94].

5.3 - Radiosidade com Ray Tracing

Considerado, juntamente com o Ray Tracing, como o melhor método para gerar imagens realistas, o método de Radiosidade[11] consegue tratar de forma correta a reflexão da luz em superfícies difusas, coisa que não é possível com o Ray Tracing tradicional. Deste modo, efeitos como penumbras e fontes de luz com área são avaliadas

[10] Neste caso não estamos considerando o índice de refração, que pode alterar a trajetória do raio.
[11] Nesta seção assumimos que o leitor possui alguma afinidade com os conceitos básicos de Radiosidade.

corretamente. No entanto, este método não consegue tratar superfícies especulares, o que é feito em Ray Tracing. A equação geral da radiosidade é dada por:

$$b_i = e_j + p_j \cdot \sum_{i=1}^{n} b_i \cdot F_{ij} \quad \text{com j = 1.. n}$$

onde:

b_j é a intensidade correspondente à radiosidade da superfície *j*.
e_j é a intensidade correspondente à energia emitida pela superfície *j*.
p_j é a reflectividade da superfície *j*.
F_{ij} é a fração do fluxo de energia que sai de *j* e chega em *i*, mais conhecido como fator de forma.

O fator de forma depende da geometria das superfícies envolvidas no processo, e é dado pela seguinte equação:

$$F_{ij} = \frac{1}{A_i} \cdot \int_{A_i} \int_{A_j} \frac{\left(\cos\theta_i \cdot \cos\theta_j\right)}{\pi \cdot r^2} dA_j \, dA_i$$

onde:

θ_i é o ângulo entre a normal da superfície *i* e a direção da superfície *j*.
dA_i é a área do elemento diferencial *i*.
R é a distância entre as superfícies *i* e *j*.

Existem diversos métodos para calcular os fatores de forma. E entre eles destacamos o método de hemicubo, que consiste em um semi-cubo posicionado no patch *i*, onde todos os outros patches devem ser projetados (figura 9) para saber qual patch contribuirá para o cálculo de *F*. Este método possui algumas desvantagens, dentre elas o alto consumo de memória (precisa manter um *item buffer*) e a impossibilidade de paralelização do algoritmo.

Uma alternativa para o cálculo dos fatores de forma é usar Ray Tracing. Uma semi-esfera subdividida em elementos de área é posicionada no patch *i* e então os raios são lançados a partir do seu centro, passando pelos elementos de sua superfície (figura 10). Para cada raio, o patch *k* mais próximo é aquele que contribuirá com seu fator de forma para o cálculo de *F*. As vantagens deste método são a não utilização de buffers de memória, solução imediata de patches ocultos e possibilidade de paralelização em hardware.

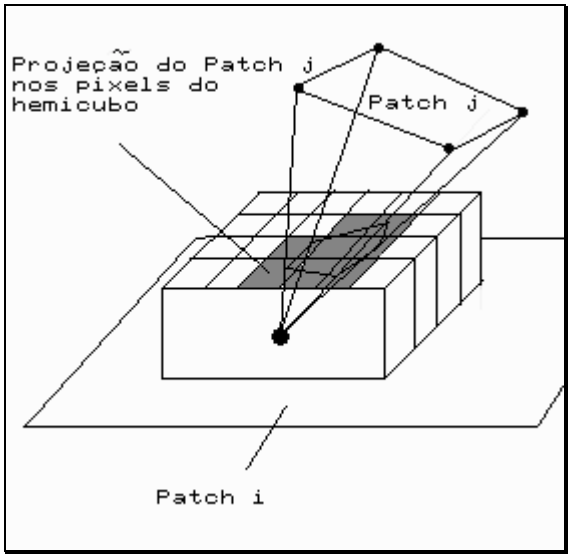


Figura 9 - Projeção do patch *j* sobre as faces do hemicubo centrado no patch *i* , durante o cálculo do fator de forma

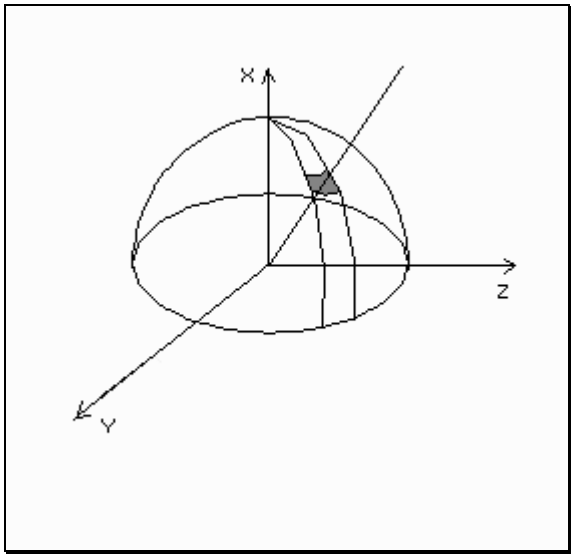


Figura 10 - hemisfério para o cálculo do fator de forma por Ray Tracing

Seção 6 - Considerações Finais

Neste artigo, tivemos uma visão geral das técnicas de Ray Tracing, um método para a geração de imagens com alto grau de realismo. Em termos de implementação, pode-se pensar no seguinte esquema recursivo para uma rotina de Ray Tracing:

Para cada ponto da tela

- Lance um raio a partir do observador.
- Ao atingir um objeto, calcule as 3 contribuições para a cor final do pixel na tela: a local (proveniente de fontes de luz), a do raio refletido e a do raio refratado (Kloc, Krefl e Krefr, respectivamente).
- Calcule a cor local (utilizando algum modelo de iluminação[12]), verificando também se o ponto está na região de sombra (lançando um raio de sombra).
- Lance dois novos raios: um na direção refletida e outro na direção refratada. O lançamento destes raios retornará a cor dos pontos atingidos.
- Calcule a média ponderada das três cores, usando Kloc, Krefl e Krefr.

O processo descrito acima é recursivamente infinito, porém existem situações em que o processo deve ser cancelado:

1. Quando o raio não atinge nenhum objeto (o ponto deverá ser pintado com a cor de fundo).
2. Quando um objeto tiver coeficientes de reflexão ou refração muito pequenos.
3. Quando o raio apresentar uma contribuição muito pequena para o cálculo da iluminação.
4. Quando houver uma recursão excessiva (por exemplo, mais de 10 chamadas recursivas).

[12] Não necessariamente o de Phong. Existem outros como Gouraud, Cook-Torrance e Hall. Eles podem ser encontrados nas referências no final do artigo.

Referências

[Rogers85] - Rogers, D.F., "Procedural Elements For Computer Graphics", *McGRAW-HILL,* 1985.

[Roth82] - Roth, S. D., "Ray Casting for Modeling Solids", *Computer Graphics and Image Processing, 18, 109-144 (1982).*

[Kay79] - Kay, Douglas S., "Transparency, Refraction and Ray Tracing for Computer Synthesized Images", *Master Thesis, Program of Computer Graphics, Cornell University, Jan 1979.*

[Phong73] - Bui-Tuong, Phong, "Ilumination for Computer Generated Surfaces", *Doctoral Thesis, University of Utah, 1973.*

[Blinn78] - Blinn, James F., "Simulation of Wrinkled Surfaces", *Computer Graphics, Vol. 12 , pp. 286-292 , 1978 (Proc. SIGGRAPH 78).*

[Cook et Alii 94] - "Distributed Ray Tracing", *Computer Graphics, July 1984.*

[Wagner94] - Wagner, F. S. V. S., CSG-Ray Versão 2.0 - "Sistema de Modelagem e Visualização de Sólidos", Manual do Usuário, *Instituto de Matemática - UFRJ,* 1994.

[Watt93] - Watt, A., "Advanced Rendering Techniques", *2nd. edition, Addison Wesley, 1993.*

[Glassner91] - Glassner, A., "An Introduction to Ray Tracing", Academic Press Limited, 1991.

[Wyvill85] - Wyvill ,G. , Sharp , P. , "Fast Antialiasing of Ray Traced Images", IEEE CG & A.

[Nigri92] - Nigri , H. ,"Um Algoritmo Híbrido de Radiosidade e Ray Tracing para a Geração de Imagens Realísticas", *Anais do SIBGRAPI 1992.*

[Santos94] - Santos , E. T. ,"Changing Some Geometric Parameters in Parametrized Ray Tracing", *Anais do SIBGRAPI 1994.*

[Borges93] - Apostila para o curso de Introdução às Técnicas de Ray Tracing da Universidade de Buenos Aires, 1993.

[Whitted80] - Whitted, T., "An Improved Illumination Model for Shaded Display", *CACM*, 23(6), June 1980, 343-349.