UM AMBIENTE INTEGRADO DE SÍNTESE DE IMAGENS REALISTAS

Aluno: Ícaro Bouças.
Orientador: Marcelo Dreux.

Introdução

A Síntese de Imagens por computador visa gerar, a partir da modelagem matemática, objetos no computador. A Síntese de Imagens pode ter como objetivo a geração realista de um objeto, a geração de objetos que não são visíveis pelo olho humano, cenas da natureza etc. Há diversos métodos para geração de imagens por computador, não existindo um método ótimo, dependendo do tipo de objetos que deseja-se sintetizar.

Objetivos

O objetivo deste projeto é desenvolver um ambiente integrado para síntese de imagens por computador a fim de disponibilizar uma ferramenta amigável para o ensino desta área nos cursos de graduação e pós-graduação.

Devem fazer parte deste ambiente os algoritmos mais difundidos para Síntese de Imagens, a saber: Ordenação por profundidade, Subdivisão de áeras, z-buffer, ***Scanline, Ray Tracing e Ray Tracing Distribuído***. Os modelos de iluminação Phong-Shading e Gouraud-Shading, assim como o tratamento de texturas e anti-aliasing também devem estar disponíveis.

Nessa nova fase de otimização do RISE, foi integrado ao seu ambiente o algoritmo de Ray Tracing Distribuído [Cook 01]. Atualmente o ambiente do RISE oferece ao usuário grande flexibilidade para gerar cenas com os três diferentes algoritmos citados. Permite ao usuário carregar cenas via arquivo ou então através de entrada de dados pelo teclado.

Foi implementado para o RISE um help on-line através do WebBook desenvolvido pelo Tecgraf/PUC-Rio, onde o usuário encontrada um manual completo com todas as funções do RISE executadas passo a passo de maneira clara e objetiva.

Metodologia

O ambiente computacional em desenvolvimento (RISE – Realistic Image Synthesis Environment) se propõe a integrar os principais algoritmos de síntese de imagens. Optou-se por iniciar pelos mais difundidos a saber: Ray Tracing [Kuchkuda 87] e [Whitted 70], Scanline [Dreux 88] e [Bouknight 70] e Ray Tracing Distribuído [Cook 01].

O projeto está sendo desenvolvido na linguagem C, utilizando o compilador Visual C++ versão 6.0. A interface com o usuário foi desenvolvida a partir de bibliotecas do próprio Windows, Glut e Glui.(Figura 1)

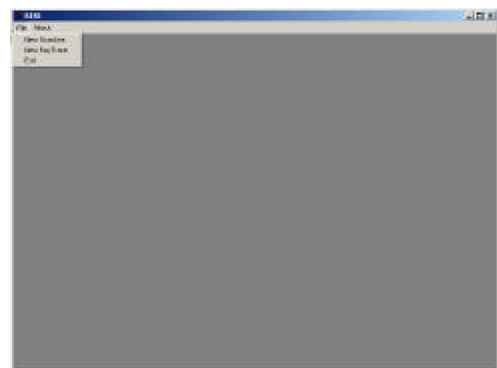

Figura 1 Interface do RISE

Ray Tracing Distribuído [Cook01]

O algoritmo de Ray Tracing Distribuído possui em linhas gerais, a mesma idéia do algoritmo de Ray Tracing visto anteriormente, porém apresenta algumas características próprias no que diz respeito ao lançamento dos raios no ambiente. Sua idéia é utilizar múltiplos raios distribuídos no tempo (borramento por movimento ou “Motion Blur”) e no espaço (profundidade de campo ou “Depth of Field”). Gerando o chamado hiper-realismo, ou seja, técnicas utilizadas na renderização de imagens que conferem a cena um realismo semelhante ao fotorealismo.

Na técnica de Ray Tracing Distribuído, é utilizado o modelo de câmera com lentes ao invés do modelo de “pinhole”, conforme ilustrado na figura 2 abaixo:

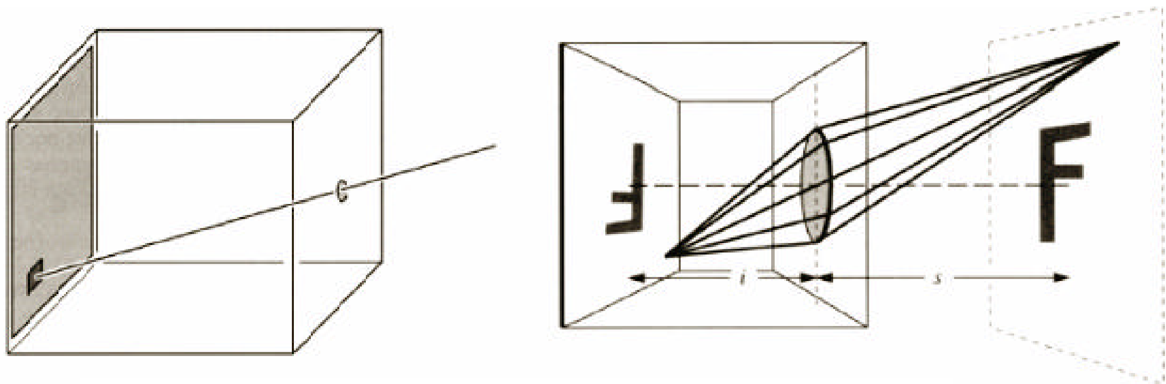

Figura 2 Modelo de Câmera Pinhole versu*s* Modelo de câmera com lente.

Com está técnica é possível implementar o efeito de “Depth of Field”, onde somente os objetos no plano de focalização estão em foco ao contrário do modelo de “pinhole” onde todos estão em foco. É possível ajustar a abertura e a distância focal da lente virtual, de forma análoga a uma maquina fotográfica convencional. Conforme ilustrado na figura 3 abaixo:

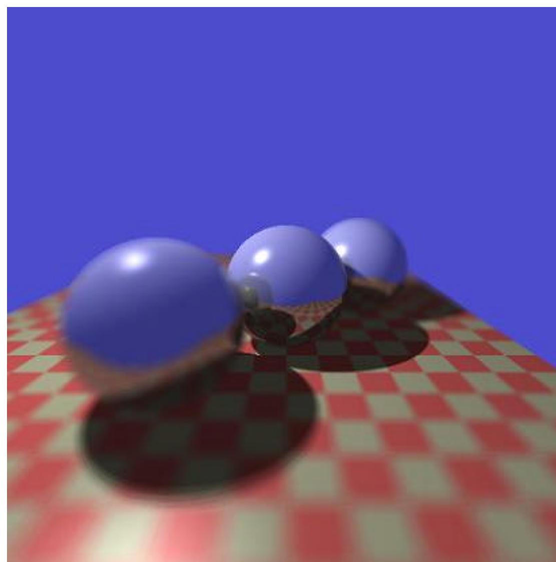

Figura 3. Cena com efeito de Profundidade de Campo(“Distributed Ray Tracing” Robert . L Cook)

Também é possível implementar o borramento por movimento ou “Motion Blur”. Essa técnica consiste em dar a cena um efeito semelhante de uma fotografia com objetos em movimento. Pois quando fotografamos um determinado objeto em deslocamento, percebemos que ele aparece desfocado ou até mesmo embaçado em relação aos outros, esse efeito de borramento é proporcional à velocidade do objeto no momento em que está sendo fotografado. Conforme a figura 4 abaixo:

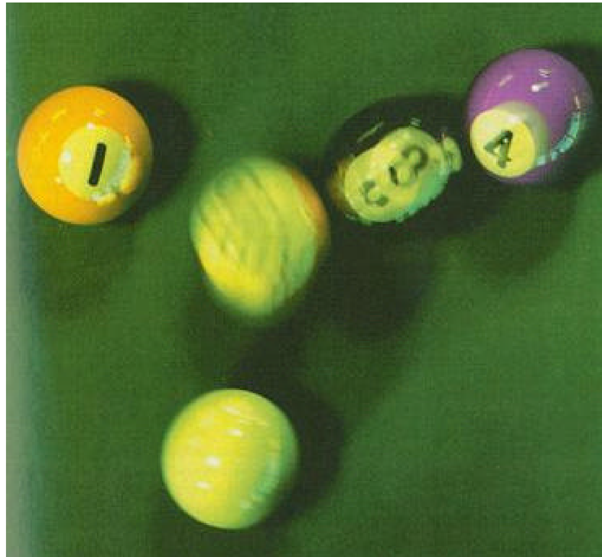

Figura 4. Cena com efeito de motion blur (“Distributed Ray Tracing” Robert . L .Cook)

Ray Tracing: [Kuchkuda 87] e [Whitted 70]

O algoritmo de Ray Tracing é usado na renderização de imagens com alto grau de realismo.A idéia fundamental por trás do algoritmo é lançar raios do observador em todas as direções e determinar se há ou não a interseção de cada raio com todos os objetos no ambiente (Figura 5).

Figura 5 Esquema do Ray Tracing

Havendo interseção, calculam-se os raios refratados e refletidos. As propriedades da superfície do material atingido são levadas em consideração para o cálculo da cor de cada pixel, usa-se o modelo de Iluminação de Phong (Figura 2.b). I_a é a componente da luz ambiente. $\mathbf{I_d}$ é a componente difusa, característica de superfícies foscas. Cabe ressaltar que se ? for 0° temos a *contribuição máxima* de $\mathbf{I_d}$, o que significa que a luz está perpendicular a superfície. $\mathbf{I_s}$ é a componente especular da luz, ou seja , a cor do brilho da luz, característica de superfícies brilhosas. $\mathbf{k_d}$ e $\mathbf{k_s}$ são os coeficientes de reflexão difusa e especular, respectivamente e n é o coeficiente especular responsável pelo controle do tamanho do brilho a ser gerado na superfície. $\mathbf{I_a}$, $\mathbf{I_d}$ e $\mathbf{I_s}$ estão no formato RGB. A cor resultante é a soma da contribuição de todas as fontes luminosas.

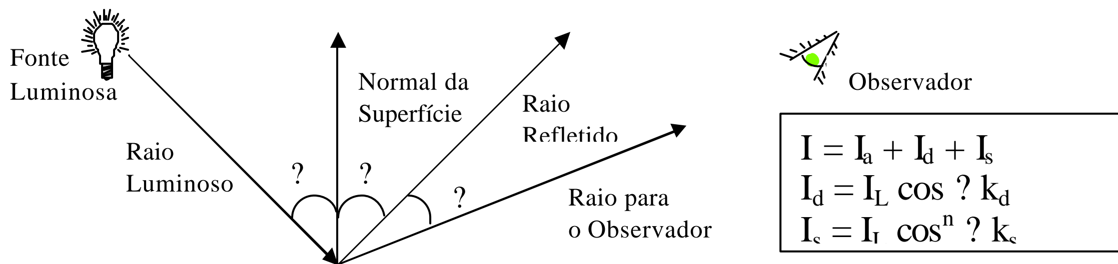

Figura 6 Modelo de Iluminação de Phong

Podemos representar todo o ambiente matematicamente. A luz possui coordenadas (x,y,z) e intensidade luminosa. Os objetos são modelados como formas geométricas primitivas: esferas, cilindros, cones, polígonos etc. A câmera pode ser definida pela sua posição no espaço (coordenadas (x,y,z)), orientação (para onde ela está apontada), e raio de ação.

Por se tratar de muitos raios luminosos, inter-reflexão e inter-refração entre os objetos, o tempo computacional do algoritmo torna-se elevado.

Segue um pseudo código do algoritmo de Ray Tracing:

```
Begin RayTracing:
    carregar os objetos
    posicionar camera
    calcular a direção e o espaçamento de cada raio a ser traçado.

    Para cada pixel da tela
    {
        Determinar o raio que passa por cada pixel.
        Determinar se o raio atinge algum objeto ( Intersect( ) )
        Se a interseção for maior que zero
        {
            /* Raio atingiu algum objeto */
            Cor do pixel = Cor do raio (Color)
        }
        Se não
        {
```

```
                /* Raio não atingiu nenhum objeto */
                Cor do pixel = Cor de fundo
            }
    }
```

End RayTracing:

Begin Intersect

```
    /* checar interseção com todos os objetos */

    Para cada Objeto
    {
            Verificar se raio intercepta o Objeto.
            Se raio atingir Objeto
            {
                    S  = Retornar a interseção mais próxima
            }
    }
    Se (S <= 0 || S > MAX_DEEP)
    {       /* raio não  atingiu objeto algum */
            retornar 0.0
    }

     Achar ponto de interseção
    Calcular o vetor Normal no ponto de interseção
    Color = Calcular a cor correspondente do ponto (Shade( ) )

    Retornar S;
}
```

End Interseção:

Begin Shade:

```
    Color  = Ambient Color

    Para cada Luz
    {
            lRay = Raio da superfície do objeto até a  fonte luminosa
            diffuse =  produto escalar(normal do objeto , lRay)
            Se (diffuse  > 0 )
            {
                    diffuse = diffuse * brilho da Luz
                    Color = Ambient Color + Surface Color*diffuse
            }
            Spec = produto escalar (raio refletido , lRay)
            Se (Spec > 0 )
```

{

*Spec = brilho da Luz * Spec* ^{coeficiente}

*Color = Ambient Color + Surface Color*diffuse + Specular* Spec*

}

}

Retornar Color;

End Shade:

Basicamente o algoritmo de Ray Tracing se baseia nessas três rotinas. Pode-se modificá-las para acrescentar novos efeitos e melhorias visuais como por exemplo a inclusão de **sombreamento, reflexão e refração.** No Ray Tracing desenvolvido há todos esses recursos / efeitos (Figura 7) .

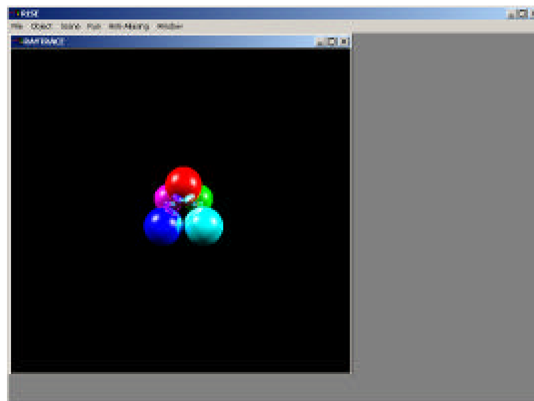

Figura 7 Imagem gerada pelo Ray Tracing

Usualmente objetos são representados por malhas de polígonos e na maioria das vezes esses polígonos são triângulos. Sendo o triângulo uma das primitivas básicas do RayTracing pode-se renderizar então malhas de triângulos. Foi incluído no RISE um carregador de malha de triângulos no formato MD2 [Hawkings 01]. O formato MD2 foi usado como modelador de objetos no jogo Quake2©.

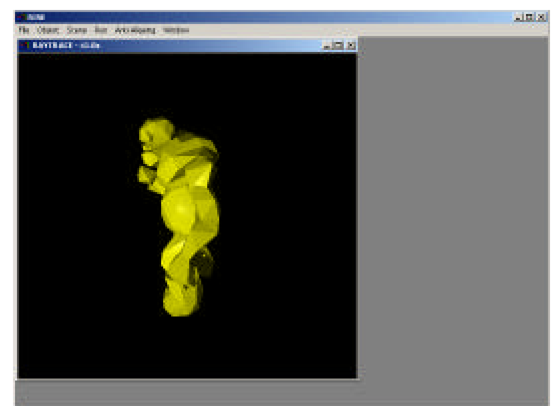

Figura 8 Cena gerada a partir de uma malha MD2

A figura 8 mostra um objeto formado por 670 triângulos. Com o aumento do número de objetos a ser renderizado, o tempo computacional aumenta consideravelmente, já que para cada raio 670 cálculos de interseção serão feitos. Para reduzir este problema existem diversas soluções. Optou-

se por usar uma técnica chamada de *bounding Volume* que consiste em criar um volume (ex. Esferas,cilindros,caixas...) envolvendo (*bounding*) um conjunto de objetos. O volume que usualmente é utilizado é a esfera, por possuir um cálculo de interseção bastante rápido.

Segue um pseudo código do algoritmo de *bounding sphere* [Ritter 90]

Begin *bounding sphere* [Ritter 90]

Para todos os vértices dos objetos a serem incluídos na esfera
{
/ Encontrar os seguintes vertices: */*
O ponto com a ***menor*** *coordenada x , o ponto com a* ***maior*** *coordenada x*
O ponto com a ***menor*** *coordenada y , o ponto com a* ***maior*** *coordenada y*
O ponto com a ***menor*** *coordenada z , o ponto com a* ***maior*** *coordenada z*
}

*/*Dado esses 3 pares de vértices achar o par que possui a* ***maior*** *distância entre eles.Esta distancia será o diâmetro da esfera inicial.*/*
Seja $P1 = Par_{min}$ e $P2 = Par_{max}$

/ O centro da esfera é dado por:*/*
$Centro.x = (Par_{min}.x + Par_{max}.x) / 2$
$Centro.y = (Par_{min}.y + Par_{max}.y) / 2$
$Centro.z = (Parmin.z + Par_{max}.z) / 2$

/ Raio é dado por */*
$Raio.x = Par_{max}.x – Centro.x$
$Raio.y = Par_{max}.y – Centro.y$
$Raio.z = Par_{max}.z – Centro.z$

/ Tamanho do raio é dado como */*
$Raio_dist = \sqrt{Raio.x * Raio.x + Raio.y * Raio.y + Raio.z * Raio.z}$
/ Pode acontecer de após calculada a esfera , algum ponto ficar fora do volume. Para isto não acontecer tem-se que proceder conforme o pseudo código abaixo*/*
Para todos os vértices
{
CheckPoint(Ponto,Centro,Raio,Raio_dist)
}

End *bounding sphere*

Begin *CheckPoint*

$dx = x – Centro.x$
$dy = y – Centro.y$
$dz = z – Centro.z$

```
Old_to_p = Ö dx*dx + dy*dy + dz*dz
Se Old_to_p > Raio_dist
{
        /* ponto está fora da esfera */
        Novo_raio_dist = (Raio_dist + Old_to_p) / 2
        Old_to_new = Old_to_p – Novo_raio_dist
        Centro.x = ( Novo_raio_dist*Centro.x + Old_to_new * Ponto.x) /
        Old_to_p
        Centro.x = ( Novo_raio_dist*Centro.y + Old_to_new * Ponto.y) /
        Old_to_p
        Centro.z = ( Novo_raio_dist*Centro.z + Old_to_new * Ponto.z) /
        Old_to_p
}
```

End *Check Point*

Após a segunda passagem pelos vértices, é garantido que tenha uma *bounding sphere* de tamanho ótimo com nenhum vértice fora da mesma.

Na geração da imagem da figura 9 os tempos de processamento foram 41 e 83 segundos, do com e sem *bounding sphere*, respectivamente.

Um problema encontrado em imagens sintetizadas por computador é o problema de *aliasing* que consiste em uma falha na geração da imagem devido à natureza discreta do monitor, no qual tenta-se representar uma figura contínua. Podemos ativar no RayTracing a opção para o tratamento de *aliasing* que é conhecido como *anti-aliasing*.

Solucionar o problema é impossível, já que corresponde a um problema físico (a discretização do monitor), mas podemos usar técnicas para diminuir de forma muito satisfatória o problema de *aliasing*.

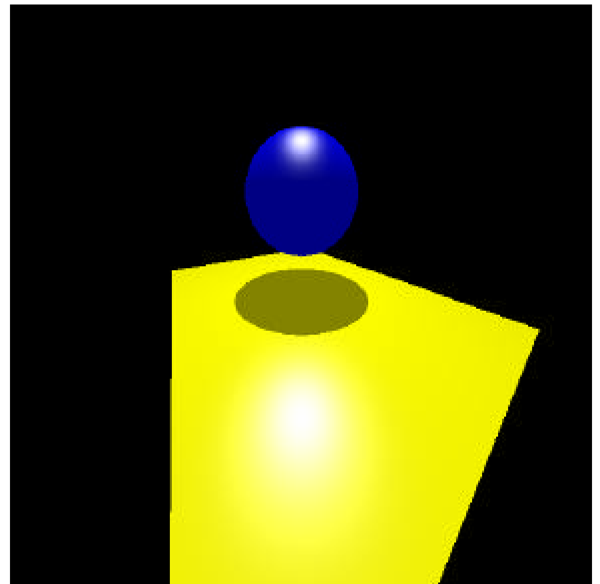

Figura 9 Imagem sem tratamento de aliasing

A técnica usada para diminuir o problema de *aliasing* é conhecida como *jittering*.[Haines]

A idéia da técnica é dividir a "área" do pixel em *n* regiões de áreas iguais.Para cada região, defina uma amostra aleatória.A cor final de cada pixel, será dada através de uma média entre todas as amostras coletadas.Para um resultado satisfatório é necessário que as amostras sejam aleatórias e que não obedeçam a nenhum tipo de *pattern*.O algortimo de jittering implementado é simplificado, o pixel é divido em 16

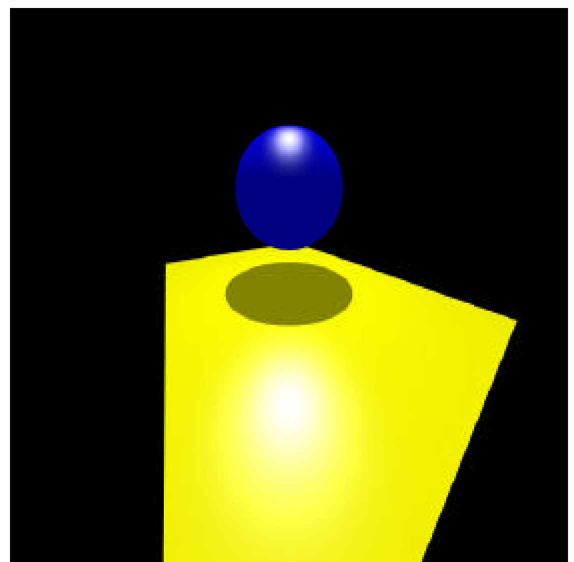

Figura 10 Imagem com tratamento de aliasing

regiões.Note que o tempo computacional aumenta proporcinalmente ao número de regiões a ser subdividido o pixel.

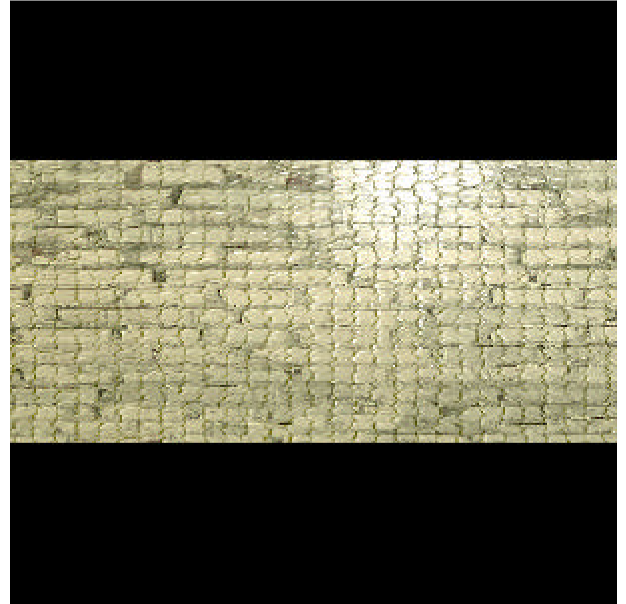

Figura 11 Um plano com mapeamento de textura junto com um mapa de bump aplicada

Além das técnicas citadas acima, é possível usar mapeamento de textura para gerar cenas com grau de realismo ainda maior.No momento , há somente mapeamento de texturas para as primitivas esféricas, triangulares e planares (plano,caixas).
Junto com a possibilidade de mapeamento de textura, está implementado também a técnica de bump mapping.A técnica de bump mapping consiste fazer com que a superfície do objeto se torne não uniforme (rugosa por exemplo) de alguma maneira. Bump mapping simula numa superfície o que seria possível somente com muitos polígonos.A idéia básica é usar um mapa de textura (bump mapping) para ao invés de variar a cor da superfície, variar a normal da superfície num determinado ponto. A normal geométrica da superfície permanece intacta(figura 11).

O RISE teve o seu código fonte do RayTracing parcialmente transformado a fim de adotar os princípios de orientação a objetos. Um objeto é uma variável que armazena dados semelhantes a uma struct, porém ele pode efetuar determinadas operações especificas, ou seja, um objeto possui então atributos (dados) e comportamentos (métodos, procedimentos, funções que atuam sobre ele).

Para uma melhor visualização do funcionamento do código do RayTracing orientado a objetos utilizaremos "UML" que é uma abreviação de Linguagem de Modelagem Unificada (Unified Modeling Languagem), uma notação (principalmente diagramática) para modelagem de sistemas, usando conceitos orientados a objetos.

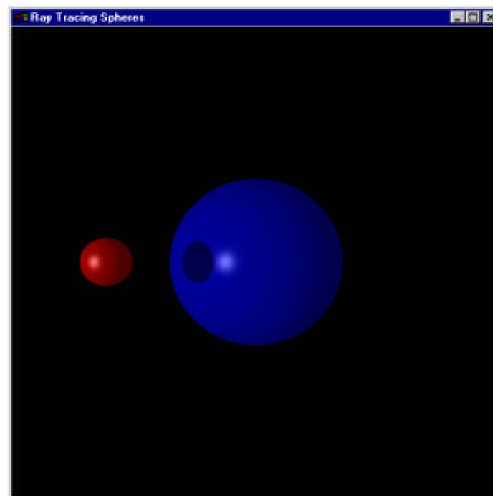

Figura 12 Imagem gerada apartir do código do Raytracing parcialmente orientado a objetos.

A análise orientada a objetos se preocupa com a criação de uma especificação do domínio do problema e dos requisitos, segundo uma perspectiva de classificação por objetos e segundo a perspectiva de compreensão dos termos usados no domínio do problema. Uma decomposição do domínio do problema envolve uma identificação dos conceitos, dos atributos e das associações, no domínio, que são considerados importantes. O resultado pode ser expresso através de um **modelo conceitual (Figura 14),** o qual a ilustrado em um conjunto de diagramas que mostram conceitos (objetos).

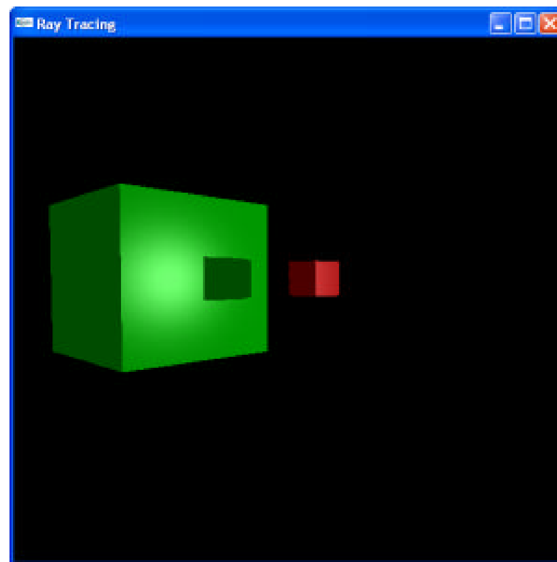

Figura 13 Cena gerada com o código do RayTracing parcialmente orientado a objetos.

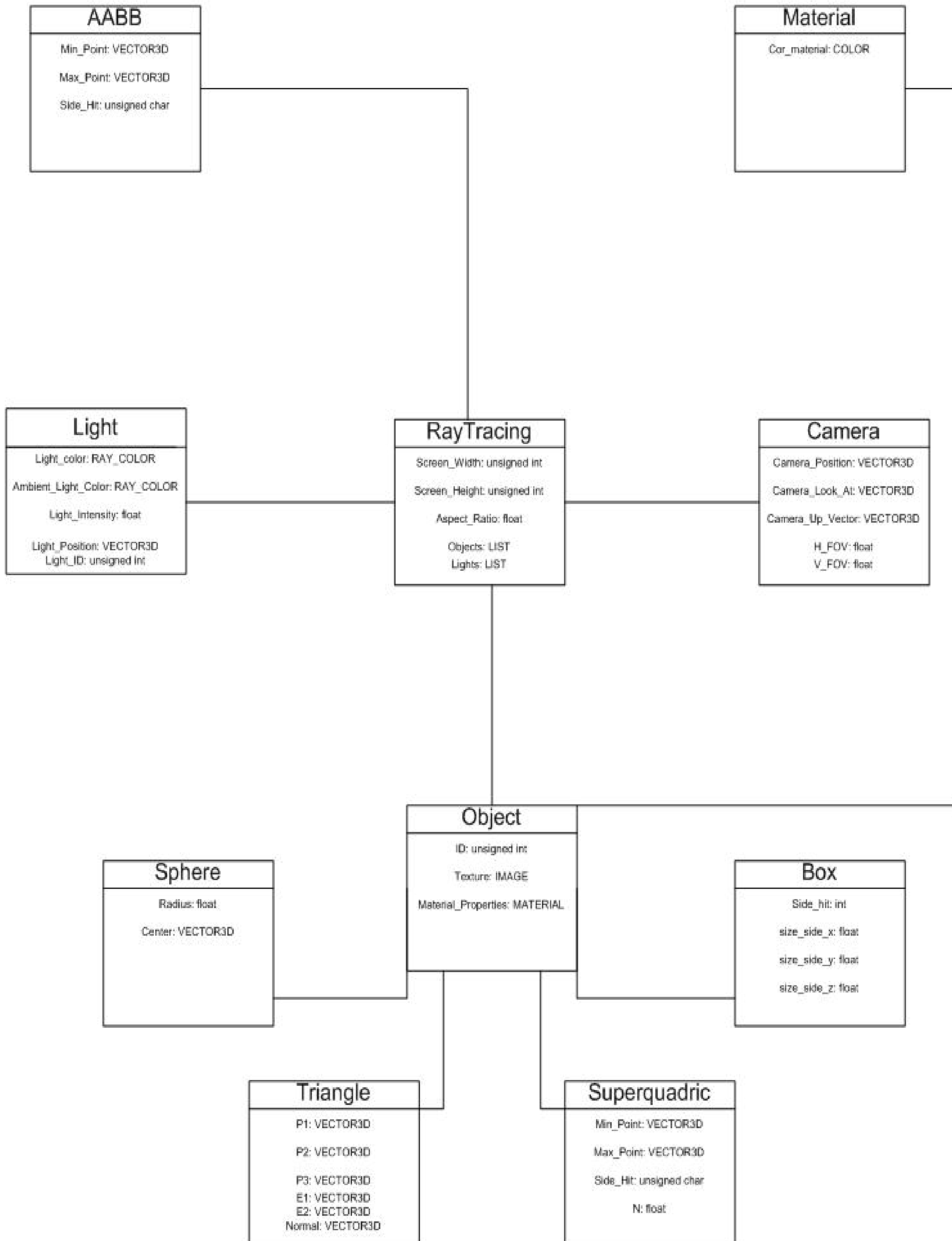

Figura 14 Modelo contendo as principais classes do RayTracing.

Scanline [Bouknight 70] [Dreux 88]

Este algoritmo é adequado para visualização de objetos poligonais, com um modelo de iluminação local, não levando em consideração a inter-reflexão de objetos vizinhos. Sua grande vantagem reside no fato de seu tempo de processamento ser bem menor do que o do algoritmo de Ray Tracing.

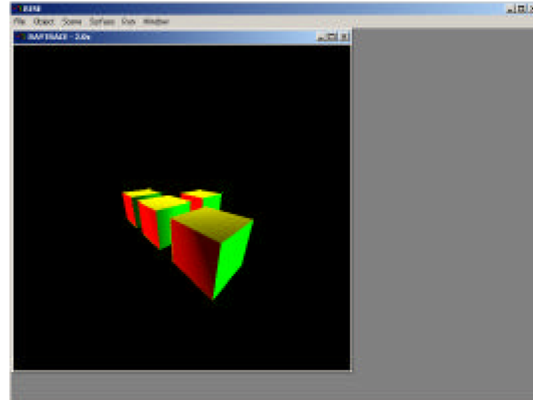

Figura 15 Exemplo de imagem gerada com o

Conceitualmente o algoritmo é simples. Há necessidade de inicialmente projetar todos os objetos na tela. Para cada scanline, verifica-se quais são as arestas interceptadas e suas faces correspondentes.

Calculam-se os valores da coordenada z (sua profundidade) entre os limites de cada face, armazenando, para cada pixel, a coordenada z mais próxima do observador (z-buffer). Junto com esta informação é necessário também armazenar a face a que estes pontos pertencem (face-buffer).

Como a imagem (figura 15) vai sendo gerada a partir de uma scanline por vez, os buffers só precisam ter tamanho correspondente à resolução x da imagem, o que garante uma economia de memória significativa, em relação ao tradicional algoritmo de z-buffer [Catmull75]

Segue um pseudo código do algoritmo de Scanline [Dreux 88]

Begin *Scanline*
{

Entrar Dados( ) / Leitura de vértices arestas e faces */*

Entrar Info ( ) / informações sobre a tela, observador e luzes */*

Calcular i_j ( ) / Cálculo da relação entre o valor do pixel(i,j) e suas coordenadas (x,y) */*

Matriz3D ( ) / Matriz de transforma o sistema de coordenadas original para outro cuja origem é o observador , os eixos X e Y são paralelos as bordas da tela e Z fura a tela no seu centro */*

Transforma 3D ( ) / transforma todas as coordenadas para o novo sistema e faz calculo da projeção perpectiva para os vértices */*

Normais ( ) / Calcula a normal de cada face */*

```
        Faces_Visivel ( ) /* Processar Backface_Culling para todas as faces e descobrir
                            quais faces não são visiveis para o observador */

        Faces _da _Aresta ( ) /* Achar faces a esquerda e a direita de cada aresta */

        Achar_Min_Max ( ) /* Achar menor e maior scanline cortada por cada aresta */

        Init_Bucket_Sort ( ) /* ordenar as arestas em ordem decrescente das maiores
                                coordenadas */

        Inicializar_Buffer ( ) /* para cada i = xres , Z_Buffer[i] = Z_Max e
                                  face_Buffer[i] = 0 */

        Para Scanline = 0 ; Scanline = Yres ; Scanline = Scanline + 1
        {
                Remover_Arestas ( ) /* remover arestas não cortadas pela Scanline
                                       Corrente caso aresta ainda cortada pela Scanline
                                       então atualizar _aresta( ) */
                Inserir_Arestas ( ) /* Inserir novas arestas sendo cortadas pela Scanline
                                       e calcu lar as interseções aresta / Scanline */

                Construir_Lista ( ) /* Construir lista de arestas ativas */
                Limites_da_Face ( ) /* Determinar o começo e o fim de cada face ativa */

                Para cada aresta na lista de arestas ativas
                {
                        Processar_Faces_Ativas ( ) /* Atualizar Z_Buffer e Face_Buffer
                                                      ao processar cada face ativa */

                }

        gerar_scanline( ) /* arquivar os valores X,Y,Z,face para cada ponto visível */

        Atualizar_Buffers( )

        }
}
END
```

Segue um pseudo código para transformação de dados .

```
Begin Matriz3D
{
        Obs    /* Vetor posição do observador */
        VRP  /* Vetor para onde o observador está olhando */
        Vup  /* Vetor View up */
```

N / Vetor normal ao plano de visualização que corresponde ao vetor ObsVRP*/*

/ Direção Y_v */*
V = Vup – (Vup.N)N

/ Direção X_v */*
U = N x *V*

/ Matriz de translação da origem para posição do observador */*

```
T = | 1  0  0  -Obs.x |
    | 0  1  0  -Obs.y |
    | 0  0  1  -Obs.z |
    | 0  0  0     1   |
```

/ Rotação dos eixos para coincindir com os eixos (U , V , N) do sistema de coordenada de visão */*

```
R = | U.x  U.y  U.z  0 |
    | V.x  V.y  V.z  0 |
    | N.x  N.y  N.z  0 |
    | 0    0    0    1 |
```

retornar Matriz RT;
}
END

Também está implementado o módulo de carregamento da malha tipo MD2 [Hawkings 01] idêntica a do algoritmo de RayTracing. Podemos ver pela comparação do tempo de renderização das duas malhas, a grande vantagem que o Scanline tem em relação ao RayTracing. O tempo de processamento do Scanline foi de apenas 2 segundos enquanto o processamento do RayTracing com *bounding sphere* foi de 41 segundos (figura 16).

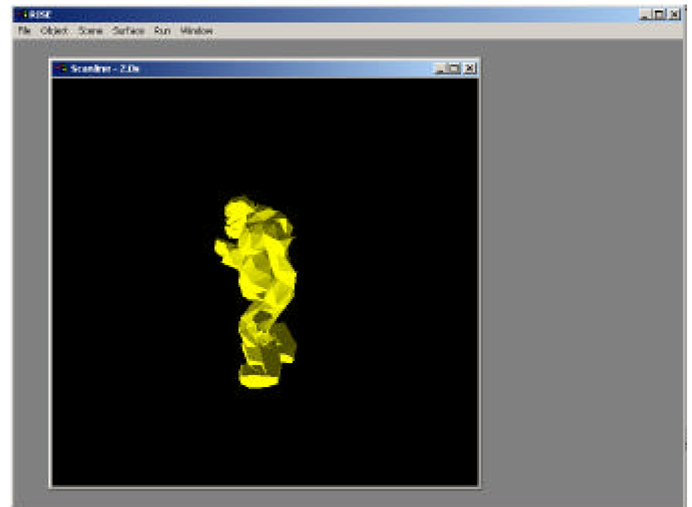

Figura 16 Malha MD2 renderizada através do algoritmo de Scanline

No presente momento a única otimização feita para o Scanline é a técnica de *backface culling* que consiste em descobrir se a normal da face está ou não apontando para o observador. Está técnica é muito simples, basta
calcular o vetor que sai do observador e vai até a face (para isto toma-se um ponto qualquer da face) e calcular o produto escalar desse vetor com a normal da face. Se o

resultado for menor ou igual a zero a face é visível, caso contrário a face não é visível e pode-se descartar esta face.

Help do usuário – WebBook. [Scuri 04]

O WebBook é um é um toolkit para criar manuais online em HTML. Já que a grande maioria dos manuais possui uma estrutura hierárquica simples, implementamos algumas funções em JavaScript para ajudar na criação de vários manuais do Tecgraf - mas não é necessário conhecer JavaScript.

Utiliza-se a metáfora de um livro: ele é composto por várias páginas, arrumadas em seqüência, com um sumário para localizar determinadas informações. O usuário pode ir para a primeira página, para a seguinte, para a anterior ou para uma página qualquer localizada pelo índice de páginas. A mesma metáfora é utilizada por sistemas comerciais como o HTML Help da Microsoft, usado nas distribuições do MSDN e do TechNet.

O toolkit oferece todo o suporte necessário para se navegar pelas diversas páginas do livro e permite alternar entre os diferentes idiomas disponíveis. Mas o diferencial do WebBook é sua portabilidade, pois ele depende apenas de um navegador, podendo ser utilizado para qualquer tipo de documentação online. E o melhor: está disponível de graça para todos.

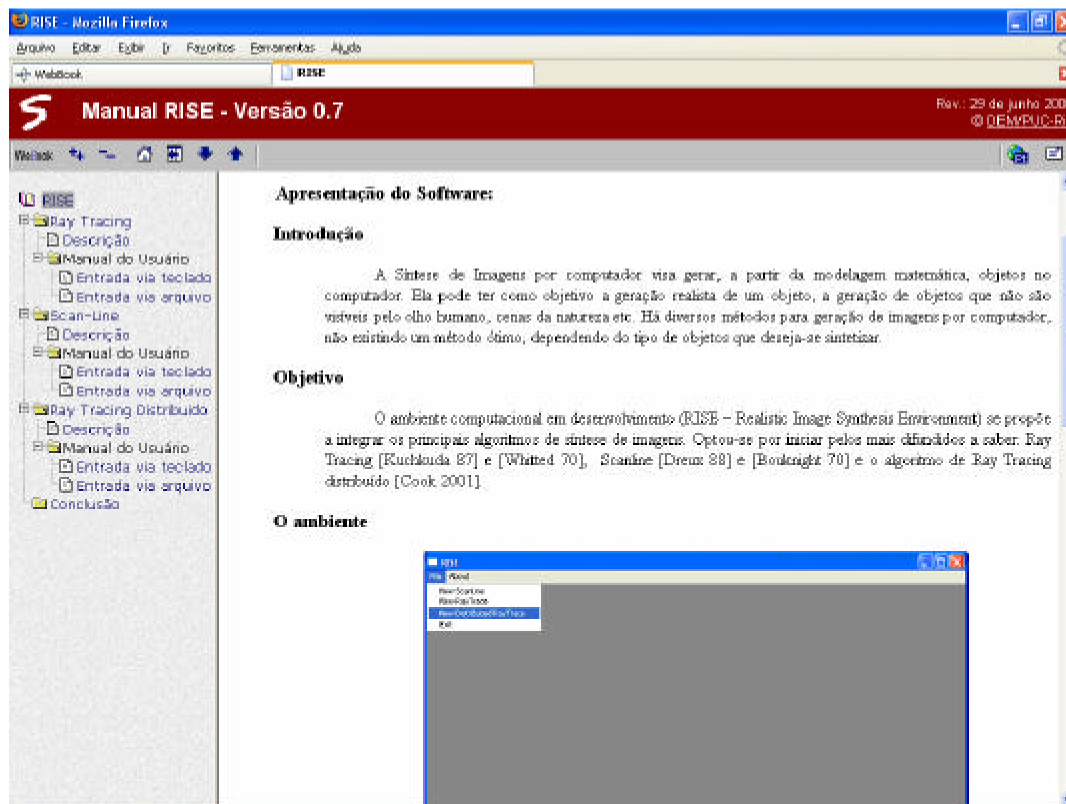

Figura 17 Help do RISE em WebBook .

O template acima(figura 17) ilustra de forma clara como está organizado os informações no manual on-line do RISE.
Na sua parte esquerda o manual conta com uma arvore de pastas que contem os seguintes campos:

RISE – Traz um overview do software. Algoritmos que o compõem e objetivo geral com relação ao desenvolvimento do ambiente .

Pasta Ray Tracing - conta com outras duas sub-pastas:
I – Descrição: onde a idéia geral do algoritmo é detalhada para o usuário com fotos ilustrativas e descrições detalhadas sobre todos os passos do algoritmo de Ray Tracing.
II – Manual do Usuário: onde ilustramos para o usuário as duas formas de entrada de dados que o RISE comporta, por arquivos e manualmente via teclado.

Pasta Scanline - conta com outras duas sub-pastas:
I – Descrição: onde a idéia geral do algoritmo é detalhada para o usuário com fotos ilustrativas e descrições detalhadas sobre todos os passos do algoritmo de Scanline.
II – Manual do Usuário: onde ilustramos para o usuário as duas formas de entrada de dados que o RISE comporta, por arquivos e manualmente via teclado.

Pasta Ray Tracing Distribuído - conta com outras duas sub-pastas:
I – Descrição: onde a idéia geral do algoritmo é detalhada para o usuário com fotos ilustrativas e descrições detalhadas sobre todos os passos do algoritmo de Pasta Ray Tracing Distribuído .
II – Manual do Usuário: onde ilustramos para o usuário as duas formas de entrada de dados que o RISE comporta, por arquivos e manualmente via teclado.

Pasta de Conclusão – Descreve a conclusão geral do projeto e detalha o estágio atual dos trabalhos de otimização do RISE.

Conclusão

Já há até o momento algumas técnicas e otimizações implementadas no **RISE**. Uma outra otimização que já está em fase de estudo é a junção do algoritmo de RayTracing com o algoritmo de Scanline que será denominado como “A Hybrid ScanLine / Ray Tracing Algorithm” [Dreux 89]. Segue abaixo uma tabela comparativa entre os dois algoritmos já implementados com algumas otimizações (Tabela 1).

	Ray Tracing	Scanline
bounding sphere	**41,0s**	**X**
S / bounding sphere	**83,0s**	**2,0 s**
Anti-aliasing	**834,0s**	**X**
Anti-aliasing e bounding sphere	**415,0s**	**X**

Tabela 1 Estudo comparativo entre Ray Tracing e Scanline para a cenas apresentadas nas figuras 4 e 7

Através da tabela acima, observa-se o efeito produzido no tempo de renderização com a inclusão de otimizações no cálculo das interseções e com técnicas para a melhoria da qualidade das imagens.

No estágio atual o RISE já se constitui numa ferramenta importante para o ensino e pesquisa na área de Síntese de Imagens por computador.

Referências

[Bouknight 70] - W.J. Bouknight, A Procedure for Generation of Three-dimensional Half-toned Computer Graphics Representations, Communications of the ACM, v13, n9, September 1970, pp 527-536.

[Cook 01] – Robert L. Cook, Stochastic Sampling and Distributed Ray Tracing, In An Introduction to Ray Tracing, Academic Press, London, U.K, 1989, pp 161-199.

[Dreux 88] - M. Dreux, A Simple Effective Linear-Growth Scanline Algorithm, ICONCG 88, International Conference on Computer Graphics, Singapore, September 1988, 209-218.

[Dreux 89] – M. Dreux, A Hybrid Scan-Line / Ray-Tracing Algorithm, PhD Thesis, Brunel University, London, UK, 1989.

[Haines] – Eric Haines, Real Time Rendering Second Edition , Screen-Based Antialiasing.

[Hawkings 01] - Kevin Hawkings and Dave Astle, OpenGL Game Programming, Ed. Prima Tech's Game Development, 2001.

[Kuchkuda 87] - R. Kuchkuda, An Introduction to Ray Tracing, Theoretical Foundations of Computer Graphics and CAD, Italy, 1987.

[Ritter 90] – Jack Ritter , An efficient Bounding Sphere , in Andrew S. Glassner, ed. , Graphic Gems , pp 301-303, 1990.

[Scuri 04] – Antonio E. Scuri e Mark Stroetzel Glasberg, WebBook – Toolkit para criar manuais on-line em HTML, Tecgraf/PUC-Rio, 2004, http://www.tecgraf.puc-rio.br/webbook/ acessado em 15/08/06.

[Whitted 80] -T. Whitted, An Improved Illumination Model for Shaded Display, Communications of the ACM, v23, n6, June 1980, pp 343-349.