CARLOS HENRIQUE DA SILVA SANTOS

MILENA ALEXANDRE DOS SANTOS

“MAPEAMENTO DE AMBIENTE”

UNIVERSIDADE ESTADUAL DE CAMPINAS
UNICAMP

Campinas
-2004-

CARLOS HENRIQUE DA SILVA SANTOS RA: 992697
MILENA ALEXANDRE DOS SANTOS RA: 00224

"MAPEAMENTO DE AMBIENTE"

Trabalho de aproveitamento do curso Computação Gráfica I da Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, ministrado pela Profª. Wu Shin Ting

UNIVERSIDADE ESTADUAL DE CAMPINAS
UNICAMP

Campinas
-2004-

“Grandes realizações são possíveis,
quando se dá importância aos pequenos começos”
Lao - Tsé

Sumário

Lista de Figuras

RESUMO

A Texturização é uma forma de variar as propriedades de uma superfície ponto-a-ponto possibilitando que esta simule detalhes que não estão de fato presentes na sua geometria. Existem algumas técnicas que possibilitam o emprego desse recurso como, por exemplo, o mapeamento de textura e a textura procedimental.

Uma forma de realização de diferentes tipos de mapeamentos é modular os parâmetros da função de iluminação (normal, coeficientes de reflexão difusa e especular, entre outros), numa função de textura.

Neste trabalho, é apresentado o mapeamento de ambiente (*envinronment mapping*), recorrente da modulação dos coeficientes de reflexão especular e difusa. Também será apresentado como trabalhar este tipo de mapeamento na *API OpenGL*.

ABSTRACT

The Texturization is a form of the variate the point-to-point surface properties to possibilite to simulate the details who not are present in fact in your geometry. It exists some techniques that they make possible the job of this resource like, per example, the texture mapping and procedimental texture.

One accomplishment form of differents types of mapped is modular the parameters of the ilumination function (normal, diffuse and specular reflexion coefficients, and others), in a texture function.

In this work, is showed the environment mapping, recurrent of the modulation of the specular and diffuse reflexion coefficient. Also it will be presented as to work this mapping type in the OpenGL API.

INTRODUÇÃO

A computação gráfica, de forma bem simplificada, prima pela transformação de dados em imagens, o que pode acarretar a necessidade da resolução de problemas de modelagem e visualização desses dados.

A modelagem se ocupa da criação e representação dos objetos, também conhecidos como modelos, no computador. Na modelagem a tarefa é de tentar representar no computador o mundo físico real. Já a visualização estuda os métodos e técnicas utilizadas para obter–se, a partir do modelo, uma imagem como produto final da Computação Gráfica.

O objetivo inverso, ou seja, a recuperação dos dados a partir de uma imagem (análise de imagem) corresponde à área de Visão Computacional, que é muito importante, por exemplo, em Robótica. Por fim, existe a necessidade de manipulação de imagens com o objetivo de processar, de alguma forma, uma imagem, para produzir uma nova imagem, a partir de operações de filtragem e de deformação, além disso, pode-se trabalhar com a compactação. Essas atividades são tratadas na área de Processamento de Imagens. A Figura I-1 esquematiza os relacionamentos entre as áreas citadas [2].

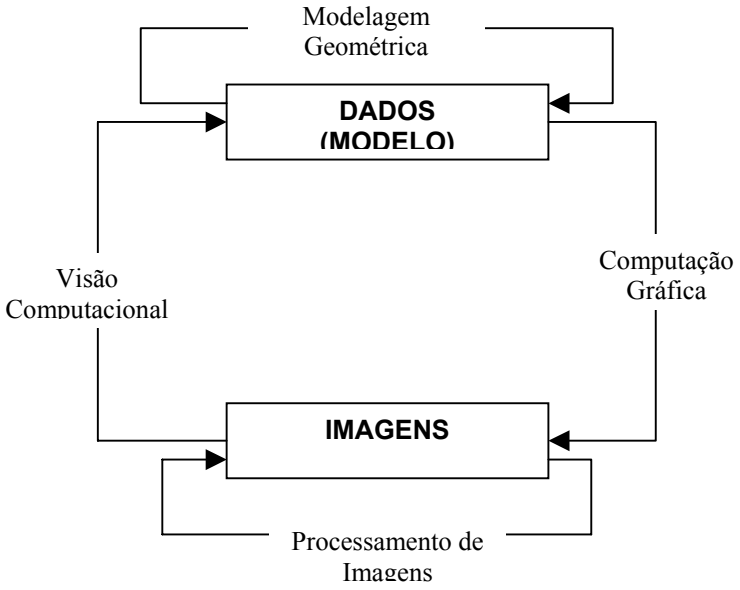

Figura I-1: Relacionamento entre as Áreas

Este trabalho visa apresentar conceitos referentes à área de computação gráfica, mais especificamente no mapeamento de ambiente com texturas. Para isso, é necessário identificar os possíveis universos distintos de serem abstraídos na computação gráfica [1], sendo:

1. O universo físico *F*, que contém os objetos do mundo real que pretende–se estudar;
2. O universo matemático *M*, que contém uma descrição matemática abstrata dos objetos do universo *F*;
3. O universo de representação *R*, que é constituído por descrições simbólicas e finitas associadas aos objetos do universo *M*;
4. E o universo *C* de codificação, que é constituído pelas estruturas de dados utilizadas na codificação do universo *R* em uma dada linguagem de programação.

O paradigma dos quatro universos, parte do pressuposto de que para estudar, com o auxílio do computador, um determinado fenômeno ou objeto do mundo real, associa - se ao mesmo um modelo matemático, para em seguida, procurar – se uma representação finita do modelo associado.

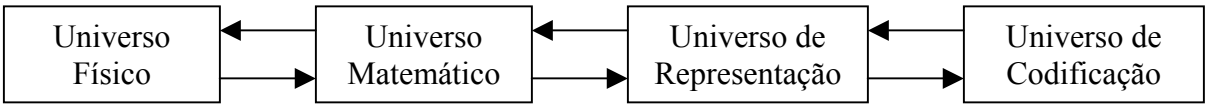

Figura I-2: Paradigma dos Universos

As áreas de aplicação da Computação Gráfica são diversas, podendo ser citadas as seguintes:

- Entretenimento: televisão, filmes, jogos;
- CAD/CAM: engenharia, arquitetura, design;
- Visualização Científica: medicina, biologia, matemática.

Técnicas de tratamento computacional aplicadas em modelagem de sólidos, com o objetivo de criar um ambiente 3D o mais próximo da realidade, são freqüentemente estudadas e propostas na Computação Gráfica.

O mapeamento de ambiente pode ser visto como uma técnica de tratamento de texturas, que simula os resultados do traçado de raio (*Ray Tracing*),

e produz uma reflexão do ambiente que circunda o objeto cuja imagem está sendo gerada.

CAPÍTULO 1

1.1 Texturas

A incorporação de “realismo” em imagens sintetizadas em computador foi fortemente auxiliada pela introdução da textura de superfície, em meados de 1970. Técnicas de texturização de superfície permitiram criar imagens visualmente interessantes e ricas sem a necessidade de produzir descrições complexas das superfícies dos objetos. Desde então, muitas técnicas de texturização foram criadas e exploradas, e a textura passou a ser uma característica essencial na qualidade de uma imagem sintética.

1.1.1 Importância da Síntese de Texturas

Métodos para a geração sintética de texturas de aparência natural são necessários para a geração de cenas de aparência realística, semelhantes a um padrão previamente escolhido, ou textura fonte.

Pode-se indicar como recomendável que os parâmetros para a síntese sejam automaticamente derivados das fontes de textura, sendo que este tipo pode ser extremamente útil na armazenagem de imagens, já que guardar um conjunto de parâmetros e limites é muito menos custoso que uma cena integral digitalizada. Outro fator interessante a ser considerado é que, sem as redundâncias existentes nas imagens brutas, é muito menos complexo o estudo e análise da cena resultante. As características dos parâmetros de textura são também utilizáveis para a discriminação e reconhecimento de padrões.

Como destaque é importante mencionar o uso da texturização em aplicações de *renderização* (obtenção da imagem a partir do modelo. É neste processo que se adiciona, por exemplo, sombreamento, cores e iluminação à cena.) de imagens de computação gráfica, para utilização em simuladores de cenas naturais, como em aplicações militares ou jogos de computador.

Ainda, consegue-se através da manipulação dos parâmetros, especificando-se intervalos de valores, gerar famílias de texturas eficientemente

para diversas aplicações, ampliando ainda mais a utilidade dos métodos de geração de texturas através de síntese.

1.1.2 O que é Textura?

Embora não exista uma definição universalmente aceita para textura, pode-se referenciá-la como sendo o conjunto de estruturas detalhadas existentes nas superfícies físicas perceptíveis ao olho humano, e que trazem grande quantidade de informações sobre a natureza da superfície. Outrossim, a definição de textura é fortemente intuitiva, o que torna uma definição incisiva alvo suscetível de contestação. Dessa forma, é importante a busca, na literatura referente a Processamento de Imagens, Computação Gráfica e Visão Computacional, de outras formas de conceituação:

- "Regiões de Textura são padrões espacialmente estendidos baseados na maior ou menor repetição precisa de alguma unidade celular *(texton* ou subpadrão)" [3].
- "O termo textura geralmente se refere à repetição de elementos básicos de textura chamados *texel*s. O *texel* contem vários pixels, cuja colocação pode ser aleatória, quasi-periódica ou, periódica. Texturas Naturais são geralmente aleatórias, ao passo que texturas artificiais são freqüentemente determinísticas ou periódicas. Textura pode ser áspera, fina, suave, granulada, ondulada, irregular, regular, ou linear" [4].
- "Textura é um atributo representando o arranjo espacial dos níveis de cinza dos *pixels* em uma região" [5].
- "Nós intuitivamente vemos este descritor como provedor de uma medida de propriedades tal como suavidade, asperidade, e regularidade" [6].

Pode-se inferir que, a despeito da falta de uma definição universalmente aceita, os pesquisadores concordam em dois pontos:

- Existe uma variação significativa na intensidade dos níveis da coloração entre *pixels* próximos, ou seja, no limite da resolução, não existe homogeneidade;

- A textura é uma propriedade homogênea a certa resolução espacial maior que a resolução da imagem.

O estudo das texturas é aplicável para o aumento do realismo das imagens geradas pela Computação Gráfica, uma vez que os métodos de remoção de superfícies escondidas e de *shading* não conseguem, por si próprios, conferir alto grau de verossimilhança às imagens, tendo em vista a complexidade do mundo real. Em termos de Computação Gráfica, textura nada mais é que uma função espacial do tipo *F(x,y*), em um espaço bidimensional, ou *F(x,y,z*), em um espaço tridimensional.

Seguindo tal definição, é possível imaginar a textura como uma matriz de valores discretos, os quais são indexados pelas coordenadas discretas *x*, *y* e *z*, comumente chamadas de espaço de textura.

Existem, ainda, duas abordagens principais para a definição de textura, em termos computacionais: a abordagem estatística e a abordagem estrutural.

A primeira considera que os valores das texturas são gerados por um campo aleatório bidimensional [7][8]. Supõe-se também que existe uma certa estrutura espacial no campo aleatório [8], o qual preceitua que o nível de cinza a um certo ponto da imagem é altamente dependente do nível de cinza nos pontos da vizinhança, a não ser que a imagem seja um campo totalmente aleatório.

A abordagem estrutural adota a premissa que a textura é composta de primitivas que se repetem ao longo da mesma. O posicionamento relativo das primitivas no padrão é determinado pela chamada “regra de distribuição” *(placement rul*e).

“O ponto de vista regra de distribuição considera a textura como sendo composta por primitivas. Estas primitivas podem variar em sua forma determinística, como círculos, hexágonos ou padrões de pontos. Macrotexturas tem primitiva grande, e microtexturas são compostas por primitivas pequenas. Estes termos são relativos à resolução da imagem. A imagem texturizada é formada por primitivas cuja orientação é especificada pelas regras de distribuição, tanto no contexto global da imagem quanto em respeito a cada uma em particular.

Exemplos destas texturas incluem ladrilhos em um plano, estruturas celulares como amostras de tecido, e a foto de uma parede de tijolos".[8]

1.1.3 Textura de Superfície

A textura de superfície baseia-se no armazenamento de valores de textura em uma tabela (que pode ser uma imagem digital) e sua aplicação sobre uma superfície. É mais fácil produzir imagens 2D para serem usadas como textura do que construir funções que produzam resultados satisfatórios quando se quer replicar uma certa textura real. [12]

No caso de modelos paramétricos esta estratégia utiliza a relação natural que existe entre o espaço de endereçamento da imagem de textura e o espaço paramétrico usado para criar e posicionar os objetos, sendo que existem diversos métodos para o mapeamento.

Atualmente, o mapeamento de textura está presente em um grande número de placas gráficas, possibilitando a modificação das informações do *pixel* durante o procedimento de *renderização*, mesmo após o término do processo de tonalização. O tempo de *renderização* em imagens em computação Gráfica depende especialmente do hardware e de como ele é utilizado: o nível de entrada e interação do usuário pode reduzir significativamente o desempenho do sistema. Assim, a utilização de hardware gráfico para mapeamento de textura pode ajudar a maximizar o desempenho. Os três componentes básicos necessários a um procedimento de mapeamento de textura são:

- A textura, definida em um espaço de textura;
- A geometria 3D, em geral definida por uma malha de vértices;
- Uma função de mapeamento que associa a textura ao objeto geométrico 3D.

1.1.4 Espaço de Textura

O espaço de textura é um espaço de coordenadas paramétricas que pode ser definido sobre um domínio de 1, 2 ou 3 dimensões. Análogo ao *pixel (picture*

element) no espaço da tela, cada elemento no espaço de textura é chamado de *texel* (*texture element*). As placas gráficas atuais oferecem flexibilidade para a interpretação das informações armazenadas em cada *texel*. Múltiplos canais de cores, intensidades, transparências, ou índices para tabela de cores são implementados em hardware.

Em uma concepção mais abstrata, o espaço de textura vai além de ser apenas uma imagem definida em um sistema de coordenadas paramétricas, podendo ser visto como um segmento especial de memória em que as variações dos valores podem ser armazenadas para serem conectadas à representação no espaço 3D.

O procedimento de mapeamento associa uma coordenada do espaço da textura a cada vértice do objeto 3D. É importante lembrar que a dimensão do espaço da textura é independente da dimensão do espaço do objeto a ser mostrado.

1.1.5 Aplicações de Valores de Texturas

A textura é, geralmente, aplicada por uma função de textura calculada para cada *pixel* exibido que pertence à superfície a ser texturizada. O valor da textura pode ser usado para modificar a iluminação da superfície de diferentes maneiras.

Uma grande porcentagem de objetos manufaturados tem pinturas ou padrões impressos em sua superfície. Desta maneira, muitos objetos são bem representados apenas variando – se suavemente suas cores ao longo da superfície. Objetos como metal e plástico apresentam padrões de reflexão suaves.

Por outro lado, objetos naturais, como madeira, exibem variações de reflexão ao longo de toda a superfície, além das variações de cor. Em funções de iluminação que incluem um componente de brilho especular, a contribuição de brilho pode ser modificada pelo valor de textura antes de ser combinada com as demais componentes da função de iluminação para implementar as variações de reflexão.

Vidros transparentes e objetos plásticos freqüentemente são exibidos com variações de transparência que podem ser produzidas por pinturas ou pela

impressão de padrões na superfície, ou pela utilização de materiais com diferentes propriedades. Um valor de textura pode ser utilizado para variar o coeficiente de reflexão da luz na superfície de forma a obter efeitos visuais melhores.

Muitos objetos manufaturados exibem uma textura de baixo relevo. Uma superfície pode ficar mais realística, por exemplo, com a adição de irregularidades ou rugosidades. Em outros casos as texturas de baixo relevo podem ser um artefato introduzido pelo processo de manufatura, como um muro com sulcos entre os tijolos ou um carpete trançado. Este tipo de superfície pode ser simulado usando os valores de textura para perturbar a orientação local da superfície antes de calcular a função de iluminação. Assim, a superfície parece distorcida, sem que tenha sido alterada geometricamente.

É necessário deslocar a superfície na região onde se deseja representar alto relevo. Isto pode ser feito usando valores armazenados na textura como um coeficiente de deslocamento da normal, podendo ser utilizada para modular qualquer parâmetro que possa variar sobre a superfície.

CAPÍTULO 2

2.1 Mapeamentos

Dentre as propriedades, ou parâmetros que podem ser reproduzidos a partir de mapas, pode-se citar: [1]

- Coeficientes de Reflexão Difusa;
- Coeficientes de Reflexão Especular e Difusa;
- Perturbação do Vetor Normal;
- Transparência/Opacidade.

Alguns dos muitos usos de mapeamentos incluem: [10]

- Simulação de materiais como madeira, tijolos ou granito;
- Redução da complexidade (número de polígonos) de um objeto geométrico;
- Técnicas de processamento de imagens como deformação e retificação, rotação e escala;
- Simulação de superfícies refletivas.

2.2 Formas de Mapeamentos

Antes de a textura ser aplicada à superfície do objeto, deve-se determinar como ela preencherá esta superfície, sendo que isso depende da função de mapeamento.

Uma função de mapeamento corresponde à forma com que a textura é usada para “embrulhar” o objeto. Ela retorna o ponto do objeto correspondente a cada ponto do espaço de textura, $(x, y, z) = F(u,v)$. Se a superfície do objeto pode ser descrita em forma paramétrica, esta pode servir como base para a função de mapeamento.

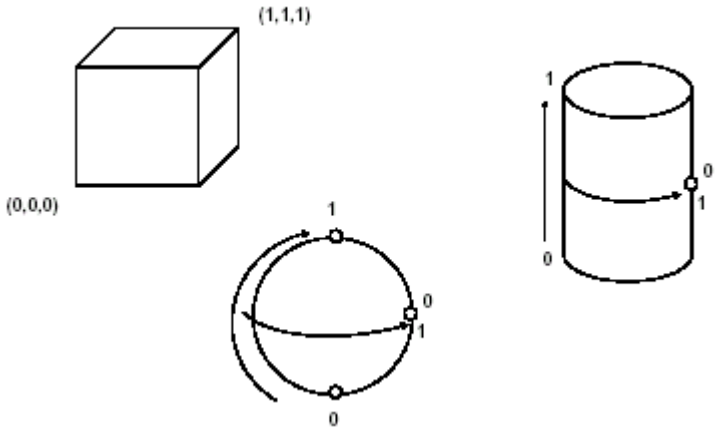

Figura 2.1: Formas de Mapeamento

2.2.1 Mapeamento Cilíndrico

Considerando primeiro o mapeamento cilíndrico; dada uma definição paramétrica do cilindro *(θ,z)*, a função de projeção que obtém as coordenadas de textura é trivial. Qualquer ponto C sobre a superfície do cilindro de raio *r* e altura *h* é representado como:

$$C(\theta,z) = (r\cos\theta,\ r\sin\theta,\ hz),\ \text{onde } 0 < \theta < 2\pi \text{ e } 0 < z < 1.$$

Pode-se, assim, associar valores de textura *(u,v)* a um ponto do cilindro pela equação:

$$(u,v) = (\theta/2\pi,\ z) \qquad u,v \in [0,1]$$

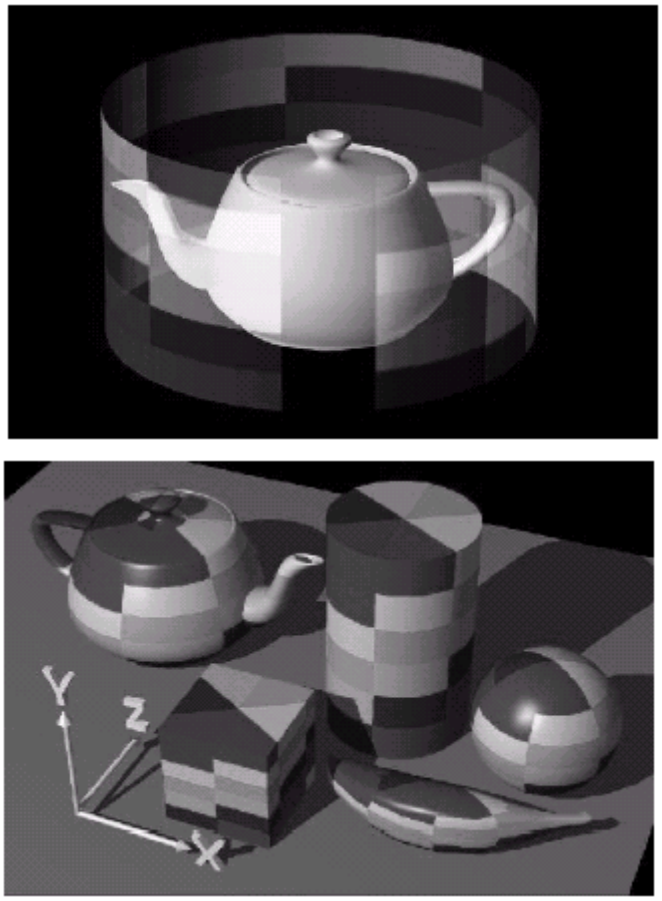

Figura 2.2: Mapeamento Cilíndrico

2.2.2 Mapeamento Esférico

O mapeamento esférico apresenta alguns problemas, pois mapear um plano sobre uma esfera produz distorções na textura próximas aos pólos da esfera. Assim, considera–se o mapeamento de uma textura sobre parte de uma esfera. A parametrização da esfera é dada por:

$C(\theta,\phi) = (r\cos\theta\sin\phi, r\sin\theta\sin\phi, r\cos\phi)$, onde $0 \leq \theta \leq \pi/2$ e $\pi/4 \leq \phi \leq \pi/2$.

A função de mapeamento é definida da mesma forma que para o cilindro:

$$(u,v) = (\theta/(\pi/2), ((\pi/2) - \phi)/(\pi/4)) \quad u,v \in [0,1]$$

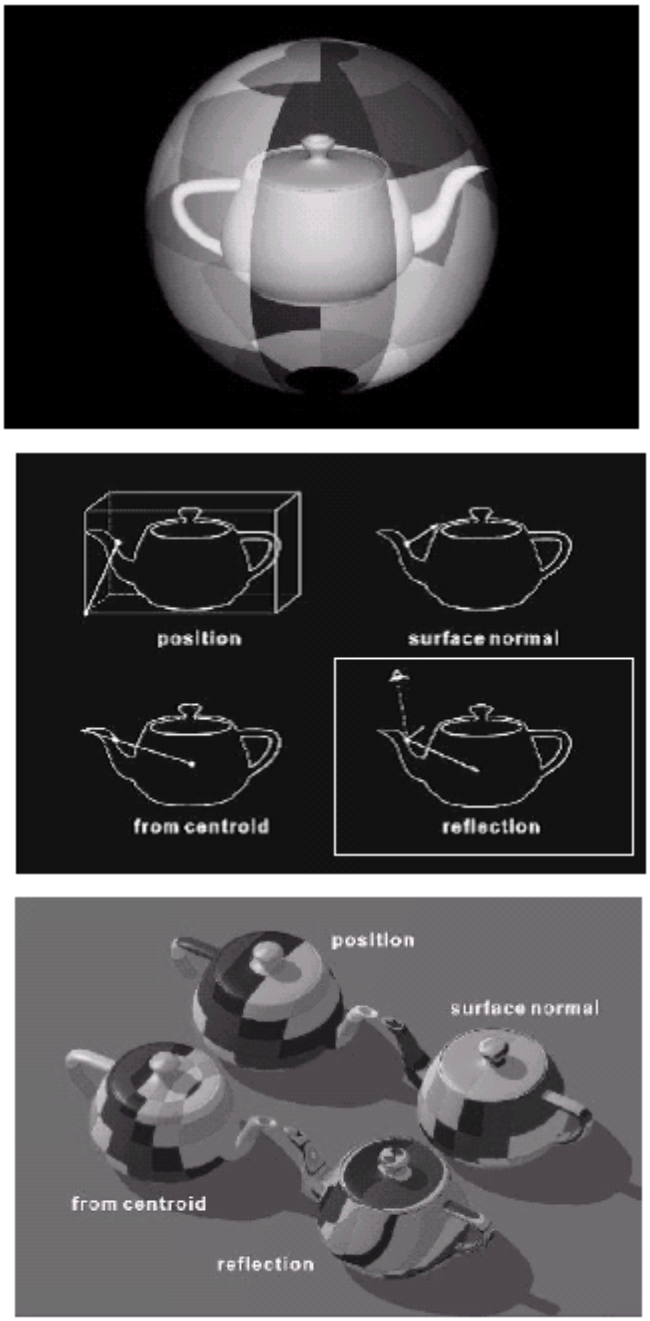

Figura 2.3: Mapeamento Esférico

Ambas as definições são transformações canônicas que mapeiam as unidades do domínio da textura no espaço do objeto. Todas as transformações 2D padrão, como escala, rotação e translação, podem ser aplicadas ao espaço da textura.

2.2.3 Mapeamento Cúbico

No caso do mapeamento sobre um cubo, a textura plana é aplicada a cada face do mesmo.

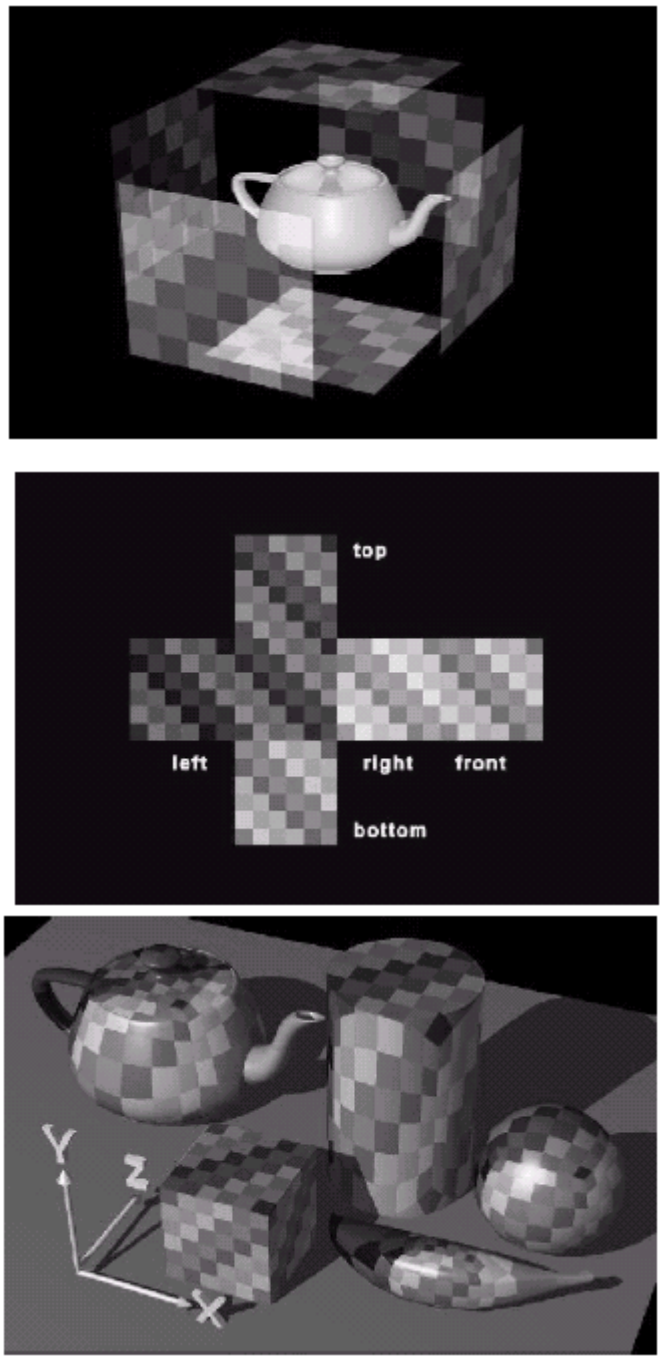

Figura 2.4: Mapeamento Cúbico

CAPÍTULO 3

3.1 Mapeamento de Ambiente

Imagens *renderizadas* não são, provavelmente, completamente o que visualizamos. Vimos que a textura é uma técnica que pode melhorar drasticamente essas imagens. Agora veremos técnicas baseadas no Mapeamento de Ambiente, onde ocorre a simulação de um objeto refletindo o seu arredor e supõe-se que o ambiente de um objeto (isto é, tudo que o cerca) está infinitamente distante dele e, pode conseqüentemente ser codificado em uma imagem conhecida, como mapa do ambiente [19].

3.1.1 Mapa do Ambiente

O mapeamento de ambiente pode ser obtido através do mapeamento da textura de duas maneiras [19].

A primeira maneira requer seis imagens da textura, cada uma corresponderá a uma faceta de um cubo, que representa o ambiente circunvizinho. Juntas, estas seis imagens dão forma a uma imagem que nós chamamos mapa do ambiente. Em cada vértice do polígono que representará o ambiente mapeado, é calculado um vetor da reflexão do observador da superfície. Este vetor da reflexão posiciona uma das seis imagens da textura. Quando todos os vértices do polígono gerarem reflexões na mesma imagem, ela será mapeada no polígono através da projeção da textura. Se um polígono tiver reflexões em mais de uma faceta do cubo, o polígono será subdividido em partes, onde cada uma das partes gera reflexões em somente uma faceta. Visto que o vetor de reflexão não é calculado em cada *pixel*, este método não é exato, mas os resultados satisfazem completamente quando os polígonos são pequenos. A Figura 3.1 mostra um exemplo de um mapa do cubo que captura um ambiente que consiste em um céu nublado e em um terreno montanhoso.

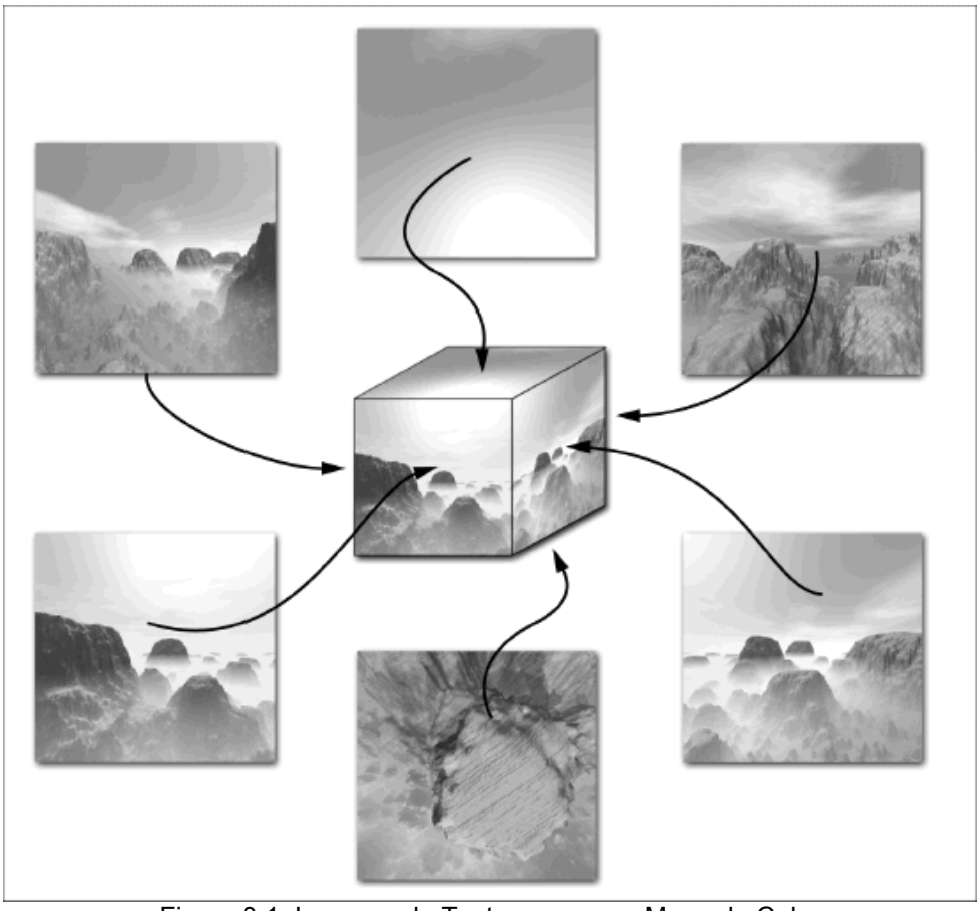

Figura 3.1: Imagens da Textura para um Mapa do Cubo

Cada faceta do mapa do cubo codifica 1/6 do panorama do ambiente em torno de um objeto. Uma textura do mapa do cubo fornece uma maneira rápida de determinar o que o objeto centrado dentro desse ambiente estaria visualizando.

O segundo método deve gerar uma única imagem da textura que corresponderá a uma esfera que simula o ambiente refletido. Esta imagem consiste em um círculo que representa o hemisfério do ambiente atrás do observador, cercado por um ângulo que representa o hemisfério na frente do observador. A imagem é aquela de uma esfera perfeita refletindo o ambiente no qual ela está situada. Em cada vértice do polígono, uma função de geração da coordenada da textura gera as coordenadas que posicionam esta imagem da textura, e são interpolados através do polígono. Se o vetor (normalizado) da reflexão em um vértice for $r = [x \quad y \quad z]$, e $m = \sqrt{2 * (z+1)}$, então as coordenadas geradas são $\frac{x}{m}$ e $\frac{y}{m}$ quando a imagem da textura é posicionada pelas

coordenadas que variam de -1 a 1 (o cálculo é exemplificado na Figura 3.2). Este método tem a desvantagem de que a imagem da textura deve ser re-calculada sempre que ocorrerem mudanças de sentido do observador, mas requerem somente uma única imagem da textura com nenhuma subdivisão especial do polígono (Figura 3.3).

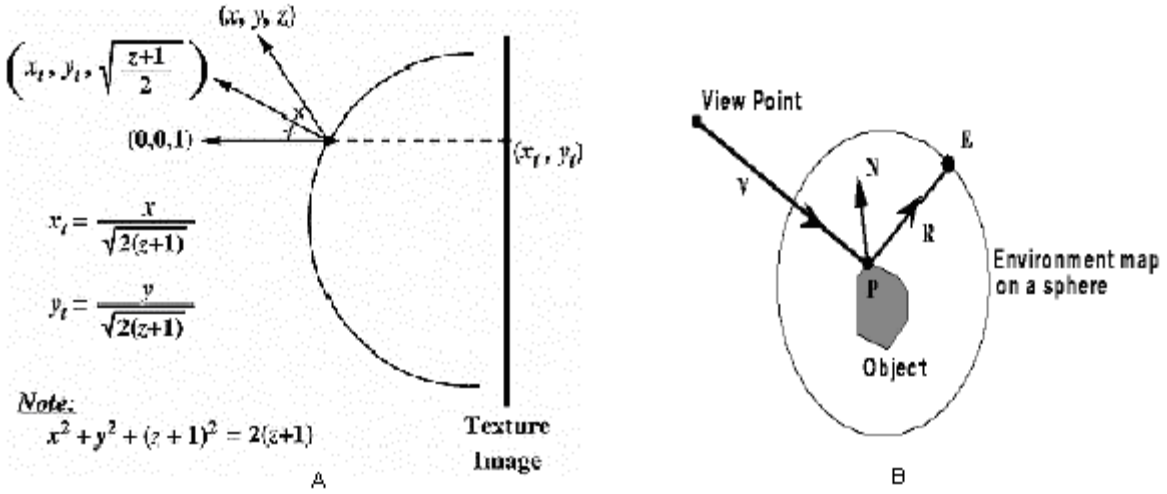

Figura 3.2: Montando um mapa esférico do ambiente (A); Aplicando o mapa esférico do ambiente num objeto (B).

Para aplicar a textura deve-se olhar cada texel por vez e para cada um determinar o sentido do raio de reflexão correspondente.

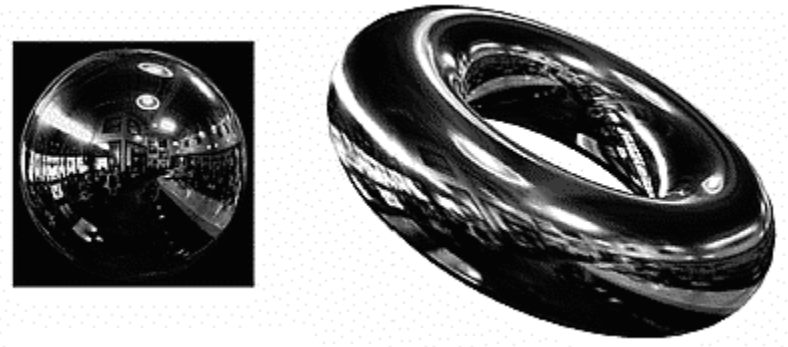

Figura 3.3: Mapeamento do Ambiente

3.1.2 Mapeamento de Ambiente – Conceito

Quando olhamos um objeto altamente reflexivo tal como uma esfera de cromo, o que vemos não é o objeto, mas como ele reflete seu ambiente. Quando olhamos em algum ponto em uma superfície altamente reflexiva, a superfície nesse ponto reflete, o raio de visão que viaja do olho ao ponto no interior do ambiente. As características do raio refletido dependem do raio original de visão e da normal da superfície no ponto onde o raio de visão alcança a superfície. O que vemos não é a superfície, mas o que o ambiente “olha” no sentido do raio refletido.

Para utilizar um mapa do cubo para codificar o que o objeto “olha” em todos os sentidos, é *renderizado* um ponto em uma superfície reflexiva, isto é calculamos o sentido refletido de visão para esse ponto na superfície. Assim, é possível alcançar o mapa do cubo, baseado no sentido refletido da visão, para determinar a cor do ambiente para o ponto na superfície [19].

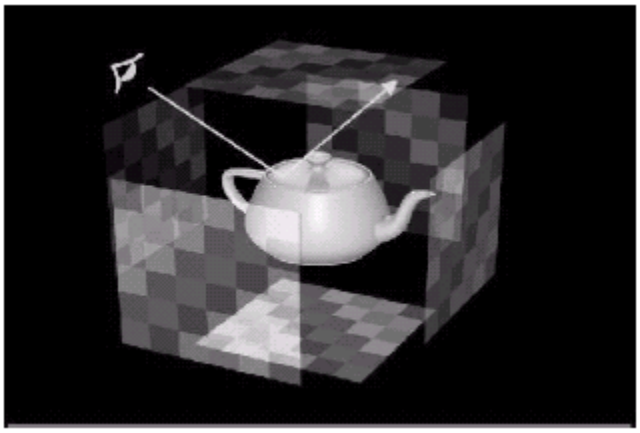

Figura 3.4: Mapa do Cubo

3.1.3 História do Mapeamento de Ambiente

Em 1976, Jim Blinn apresentaram um trabalho onde foi utilizado o mapa do ambiente. O primeiro objeto apresentado, com uso do mapeamento de ambiente foi um *Teapot* de *Utah* (Figura 3.6), o mapa do ambiente era a imagem de um quarto (Figura 3.5):

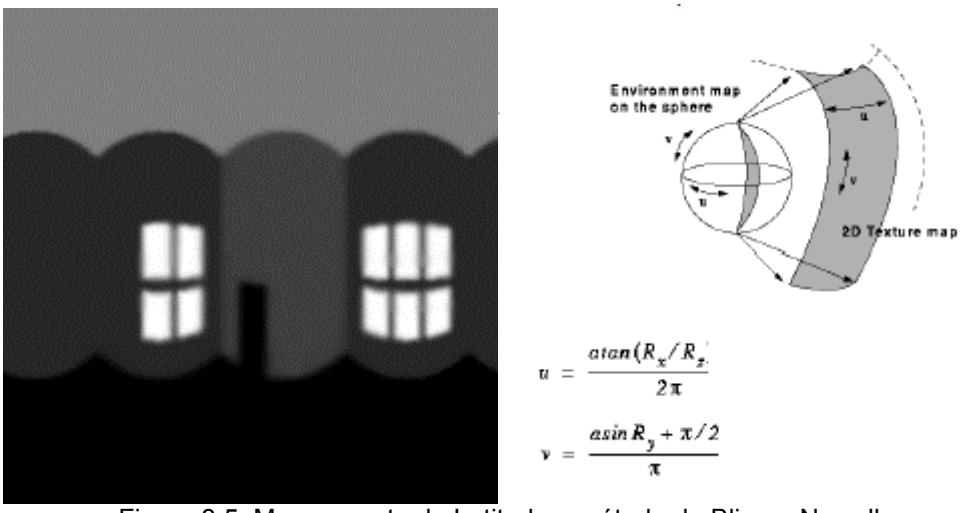

Figura 3.5: Mapeamento de Latitudes, método de Blinn e Newell.

Figura 3.6: Teapot de Utah

No mesmo artigo, Blinn incluiu a imagem de um satélite, com mapeamento de ambiente, onde o mapa do ambiente era uma imagem da terra e do sol (Figura 3.7).

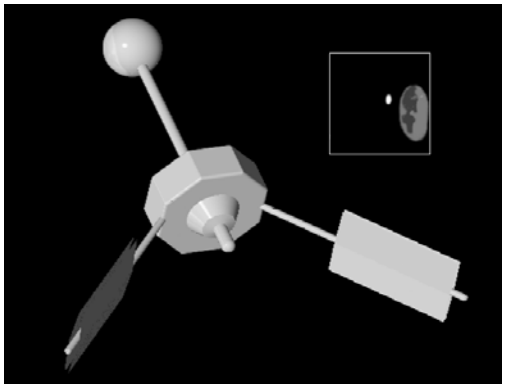

Figura 3.7: Composição de Satélite

O uso de fotografias reais como mapa de reflexão, foi desenvolvida, independentemente, por Gene Moleiro em conjunto com Ken Perlin, e também por

Michael Chou que trabalha com Lance Williams, por volta de 1982 ou 1983. O robô ao lado de Michael Chou (Figura 3.8), foi uma das primeiras imagens com uso de mapeamento de reflexão usada para colocar objetos numa cena.

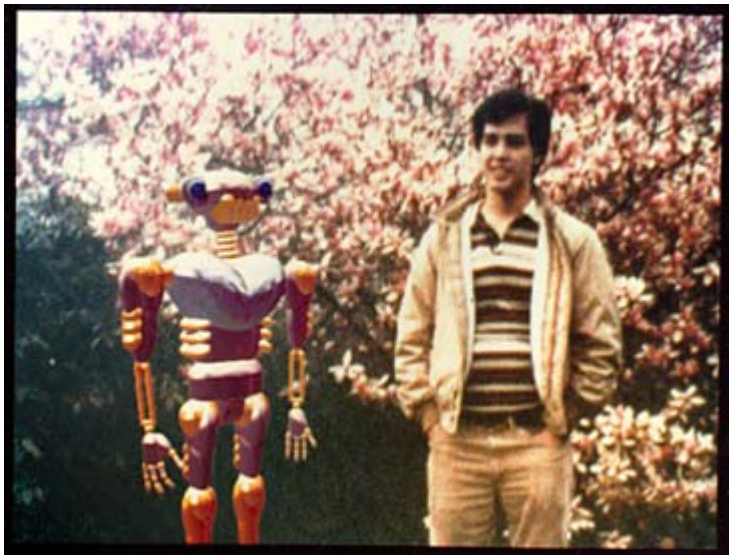

Figura 3.8: Mapeamento de Reflexão

A imagem de Chou com o robô apareceu no artigo "*Pyramidal Parametrics*" de Williams em 1983. O artigo introduz o *Mipmapping*, um esquema para pre-filtragem para evitar o *aliasing* em algoritmos de mapeamento de textura. A imagem de mapeamento de reflexão do robô era apenas um exemplo usado para demonstrar a técnica.

Em 1985, Williams fazia parte de uma equipe no *New York Institute of Technology* que usou o mapeamento de reflexão em numa cena em movimento com um elemento animado por Computação Gráfica. A "*Interface*" caracterizou uma mulher beijando um robô brilhante. Na realidade, ela foi filmada beijando uma esfera brilhante, e o mapa da reflexão foi feito através da reflexão da esfera. Para fazer a animação, o mapa da reflexão foi aplicado ao robô, e o robô foi adicionado na cena para substituir a esfera.

O primeiro filme a utilizar a técnica foi *Flight of the Navigator* de Randal Kleiser em 1986. C. Robert Hoffman fazia parte da equipe de efeitos que rendeu um vôo brilhante da nave espacial refletindo aeroportos, campos, e oceanos. A

técnica foi revisada recentemente para render a nave espacial reflexiva de Naboo Spacecraft em Guerra nas Estrelas: Episodio I.

Figura 3.9: Nave do filme de Randal Kleiser

Também em 1986, Ned Greene publicou um artigo onde foi desenvolvida e formalizada a técnica de mapeamento de reflexão. Mostrou-se que os mapas do ambiente poderiam ser pré-filtrados e indexados numa tabela com a soma das áreas, afim de executar uma aproximação boa para corrigir o *anti-aliasing*. Greene combinou uma imagem de 180 graus real do céu com uma imagem gerada no computador de um terreno do deserto para criar um cubo do ambiente.

Figura 3.10: Mapa do Ambiente – Greene

O mapeamento de reflexão teve maior êxito em 1991 num filme de James Cameron. Inspirado pelo uso do mapeamento de reflexão em "*Flight of the Navigator*", a luz em acréscimo com as mágicas industriais usaram a técnica para criar o olhar surpreendente do robô *T1000* no "*Terminator II*".

Figura 3.11: Robô – “Terminator II”

Em 1993, Paul Haeberli e a marca Segal publicaram usos inovadores de mapeamento de textura. Mapeamento de reflexão era uma das aplicações, e demonstraram a técnica aplicando um mapa do ambiente de um restaurante num torus. Paul Haeberli fotografou o mapa do ambiente do "restaurante" no Cafe Verona in Palo Alto, CA. usando uma câmera Nikon com a lente de 180 graus. Para criar o ambiente completo de 360 graus, Paul espelhou a imagem de 180 graus, e colocou então as duas imagens juntas para produzir um ambiente completo.

Figura 3.12: Café Verona 180 graus – Imagem Original

Figura 3.13: Café Verona 360 graus – Imagem Montada

3.1.4 Calculando Vetores de Reflexão

A Figura 3.14 ilustra um objeto, uma posição do olho, e uma textura do mapa do cubo que captura o ambiente que cerca o objeto. Como a figura está descrevendo uma cena 3D no plano 2D, o objeto é mostrado enquanto um trapezóide, e o ambiente é mostrado como o quadrado circunvizinho.

O vetor *I* é chamado raio incidente do olho à superfície do objeto. Quando *I* alcança a superfície, reflete-se no sentido *R* baseado na normal da superfície *N*. Este segundo raio é o raio refletido. A Figura 3.15 mostra a geometria da situação [19].

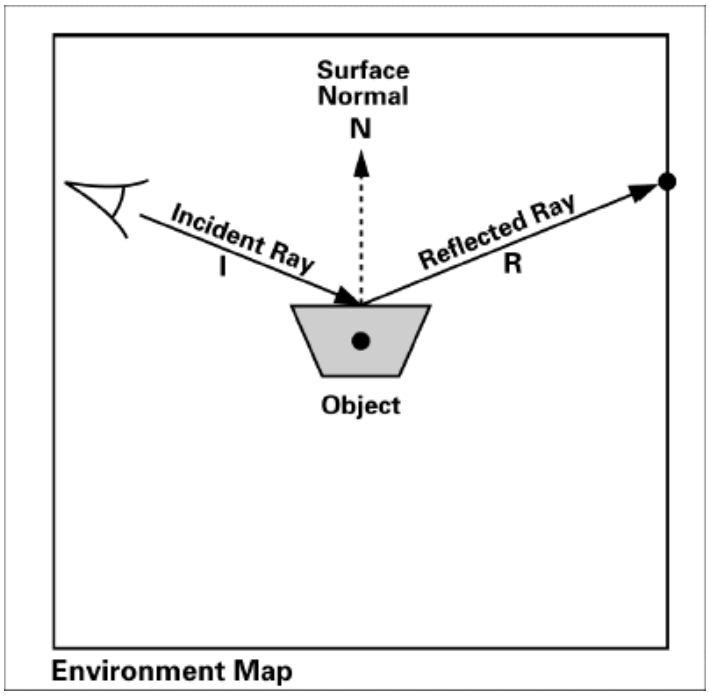

Figura 3.14: Descrição de uma cena 3D no 2D

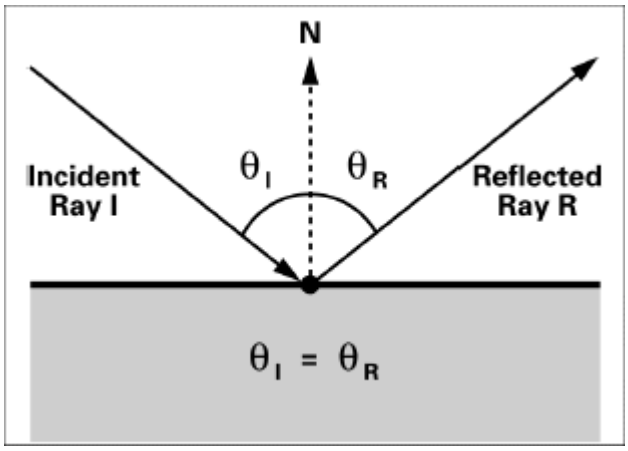

Figura 3.15: Calculando o Raio Refletido

O ângulo de incidência (θ_I) é o mesmo que o ângulo de reflexão (θ_R) para um refletor perfeito tal como um espelho. Podemos calcular o vetor refletido *R* nos termos dos vetores *I* e *N*, como mostra a equação:

$$R = I - 2N(N \bullet I)$$

Figura 3.16: Mapeamento de Ambiente de Reflexão

3.1.5 Suposições para o Mapeamento de Ambiente

A discussão precedente mencionou que o mapeamento do ambiente supõe que o ambiente está infinitamente distante do objeto. A razão para a suposição é que os mapas do ambiente são traçados baseando-se em um único sentido 3D. O mapeamento do ambiente não tem nenhuma permissão para variações na posição para afetar a aparência refletida das superfícies. Se tudo no ambiente for suficientemente distante da superfície, esta suposição é aproximadamente verdadeira.

Na prática, os artefatos visuais que resultam quando o ambiente não é suficientemente distante são tipicamente despercebidos. As reflexões, particularmente em superfícies curvadas, são sutis o bastante que a maioria das pessoas não observam quando uma reflexão não é fisicamente exata. Como as reflexões combinam a coloração do ambiente e o mudam apropriadamente com a curvatura da superfície, as superfícies *renderizadas* com mapeamento do ambiente parecem reais.

Idealmente, cada objeto mapeado no ambiente de uma cena deve ter seu próprio mapa do ambiente. Na prática, os objetos podem freqüentemente compartilhar os mapas do ambiente como se ninguém os observassem.

Na teoria, devemos regenerar um mapa do ambiente quando os objetos no ambiente se movem ou quando o objeto reflexivo que usa o mapa do ambiente se move significativamente relativo ao ambiente. Na prática, conhecendo reflexões é possível obter mapas estáticos do ambiente.

Com um mapa do ambiente, um objeto pode refletir somente o ambiente; não podendo refletir-se. Similarmente, não ocorre reflexão múltipla, como quando dois objetos brilhantes se refletem, pois um objeto mapeado no ambiente pode refletir somente seu ambiente e não a si próprio. O mapeamento do ambiente é melhor representado em objetos convexos do que em objetos mais côncavos.

Como o mapeamento do ambiente depende unicamente do sentido e não da posição, para o seu funcionamento, ele não é bem sucedido em superfícies reflexivas planas tais como os espelhos, onde as reflexões dependem

significativamente da posição. Resumindo, o mapeamento do ambiente trabalha melhor em superfícies curvas.[19]

3.1.6 A Física da Refração

Quando a luz passa entre dois materiais com densidade diferente (ar e água, por exemplo), ocorre mudança de sentido da luz. Esta mudança no sentido acontece, pois a luz viaja mais lentamente em materiais mais densos. Por exemplo, a luz viaja rapidamente no ar, porém é mais lenta na água. O exemplo clássico da refração é a "curvatura" que aparece em uma palha quando você a coloca em um vidro com água.

Como referência é importante citar a Lei de Snell que descreve o que acontece à luz entre dois meios, como mostrado na Figura 3.17. O vetor refratado é representado por *T*, que está "transmitido". A lei de Snell é expressa matematicamente pela equação $n_1 sin\theta_I = n_2 sin\theta_T$, a equação tem quatro variáveis: *o* $\emptyset_I$ do ângulo incidente*, o* $\emptyset_T$ do ângulo refratado, e um índice de refração para cada meio, n_1 e n_2.

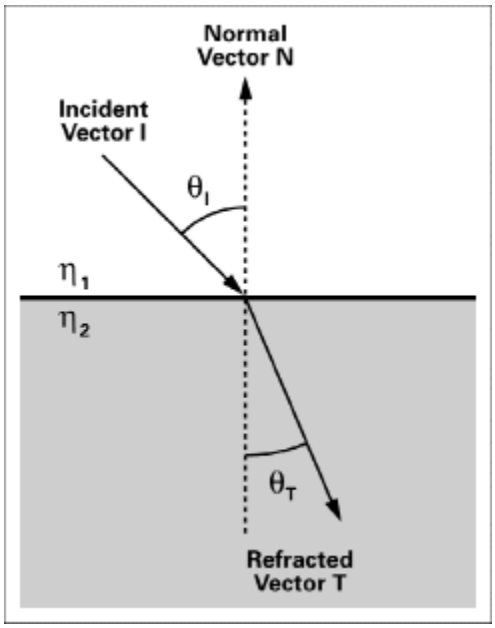

Figura 3.17: Lei de Snell

O índice de um meio de refração mede como o meio afeta a velocidade da luz. Quanto mais elevado o índice de refração de um meio, mais lentamente a luz

viaja por ele. Na Tabela 3.1 são listados alguns materiais comuns com seus respectivos índices de refração aproximados.

Material	Índice de Refração
Vácuo	1,0
Ar	1,0003
Água	1,3333
Vidro	1,5
Plástico	1,5
Diamante	2,417

Tabela 3.1 – Índices de Refração de alguns Materiais

Na Figura 3.18, observa-se que cada raio incidente no olho refrata e cada raio refratado é usado para “olhar” o mapa do ambiente.

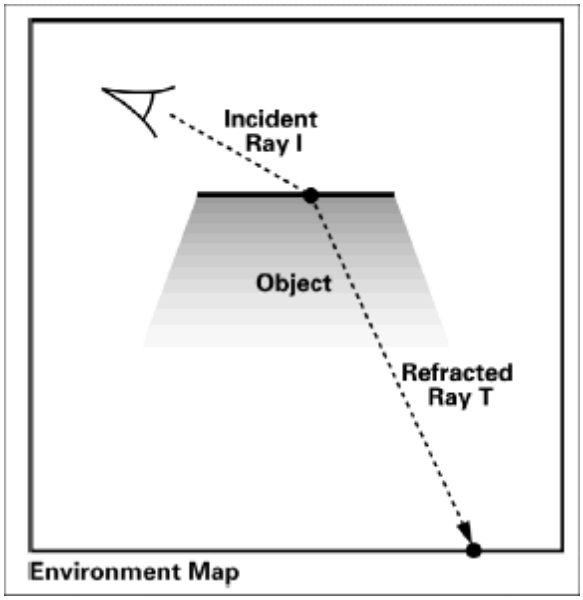

Figura 3.18: Refração em um Mapa do ambiente

Figura 3.19: Mapeamento de Ambiente de Refração

CAPÍTULO 4

4.1 OpenGL

A interface destinada a aplicações gráficas 2D ou 3D deve satisfazer diversos critérios como, por exemplo, ser implementável em plataformas com capacidades distintas sem comprometer o desempenho gráfico e sem sacrificar o controle sobre as operações de *hardware* [13].

Atualmente, a *OpenGL* ("*GL*" significa *Graphics Library*) é uma *API* de grande utilização no desenvolvimento de aplicações em computação gráfica [14]. Este padrão é o sucessor da biblioteca gráfica conhecida como *IRIS GL*, desenvolvida pela *Silicon Graphics* como uma interface gráfica independente de hardware [15]. A maioria das funcionalidades da *IRIS GL* foi removida ou reescrita na *OpenGL* e as rotinas e os símbolos foram renomeados para evitar conflitos (todos os nomes começam com gl ou GL_). Na mesma época, foi formado o *OpenGL Architecture Review Board*, um consórcio independente que administra o uso da *OpenGL*, formado por diversas empresas da área.

A *OpenGL* é uma interface que disponibiliza um controle simples e direto sobre um conjunto de rotinas, permitindo ao programador especificar os objetos e as operações necessárias para a produção de imagens gráficas de alta qualidade. Para tanto, esta biblioteca funciona como uma máquina de estados, onde o controle de vários atributos é realizado através de um conjunto de variáveis de estado que inicialmente possuem valores padrão, podendo ser alterados caso seja necessário. Como exemplo, pode-se citar que todo objeto será traçado com a mesma cor até que seja definido um novo valor para a variável.

Por ser um padrão destinado somente a *renderização* [13], a *OpenGL* pode ser utilizada em qualquer sistema de janelas (por exemplo, *X Window System* ou *MSWindows*), aproveitando-se dos recursos disponibilizados pelos diversos *hardwares* gráficos existentes. No *X Window System*, ela é integrada através da GLX (*OpenGL Extension for X*), um conjunto de rotinas para criar e gerenciar um contexto de *renderização* da *OpenGL* no *X* [15]. Além da *GLX*, existem outras bibliotecas alternativas para *interfaceamento* no *X*, tais como *GLUT* (*OpenGL*

Utility Toolkit [16]) e *GTK* [17]. Estas bibliotecas possuem um conjunto de ferramentas que facilita a construção de programas utilizando a *OpenGL*. Podemos citar, por exemplo, funções para gerenciamento de janelas, rotinas para geração de vários objetos gráficos 3D ou dispositivos de entrada de dados.

Uma vantagem em se utilizar a *GLUT* é que esta biblioteca é compatível com quase todas as implementações *OpenGL* em *Windows* e *X*. Em aplicações que requerem uma maior utilização dos recursos do *X*, pode-se utilizar a *GLUT* juntamente com a *GLX*.

4.2 Texturas no OpenGL

Para realizarmos um mapeamento de textura na *OpenGL*, o procedimento utilizado segue um padrão básico, conforme descrito a seguir:

- Especificar a textura;
- Indicar como a textura será aplicada para cada *pixel*;
- Habilitar o mapeamento de textura;
- Desenhar a cena, fornecendo as coordenadas geométricas e as coordenadas de textura;

Na *OpenGL*, quando um mapeamento de textura é realizado, cada *pixel* do fragmento a ser mapeado referencia uma imagem, gerando um *texel*. O *texel* é um elemento de textura que representa a cor que será aplicada em um determinado fragmento, tendo entre um (uma intensidade) e quatro componentes (RGBA) [18]. Uma imagem de textura é disponibilizada pelas funções *glTextImage*()* podendo, caso necessário, ser especificada em diferentes resoluções através de uma técnica denominada *mipmapping*. O uso de uma textura com multiresolução é recomendado em cenas que possuam objetos móveis. A medida que estes objetos se movem para longe do ponto de visão, o mapa de textura deve ser decrementado em seu tamanho na mesma proporção do tamanho da imagem projetada. Desta maneira, o mapeamento sempre utilizará a resolução mais adequada para o fragmento.

Para indicar como a textura será aplicada para cada *pixel*, é necessário escolher uma das três possíveis funções que combinam a cor do fragmento a ser

mapeado com a imagem da textura, de modo a calcular o valor final para cada *pixel*. Pode-se utilizar os métodos *decal*, *modulate* ou *blend*, de acordo com a necessidade do usuário. O controle do mapeamento da textura na área desejada é especificado através das rotinas *glTextEnv*()*, enquanto as rotinas *glTextParameter*()* determinam como a textura será organizada no fragmento a ser mapeado e como os *pixels* serão "filtrados" quando não há um exato casamento entre os *pixels* da textura e os da tela.

Para desenhar a cena é necessário indicar como a textura estará alinhada em relação ao fragmento desejado, ou seja, é necessário que sejam especificadas as coordenadas geométricas e as coordenadas de textura. Para um mapeamento de textura bidimensional, o intervalo válido para as coordenadas de textura será de 0.0 a 1.0 em ambas direções, diferentemente das coordenadas do fragmento a ser mapeado onde não há esta restrição. No caso mais simples, por exemplo, o mapeamento é feito em um fragmento proporcional às dimensões da imagem de textura. Nesta situação, as coordenadas de textura são *(0,0), (1,0), (1,1)* e *(0,1)*. Entretanto, em situações onde o fragmento a ser mapeado não é proporcional à textura, deve-se ajustar as coordenadas de textura de modo a não distorcer a imagem. Para definir as coordenadas de textura é utilizado as rotinas *glTextCoord*()*.

Para habilitar o mapeamento de textura é necessário utilizar a rotina *glEnable()*, utilizando a constante *GL_TEXTURE_1D* ou *GL_TEXTURE_2D*, respectivamente para um mapeamento unidimensional ou bidimensional.

Muito embora a *OpenGL* provê suporte para o mapeamento de texturas, este ainda é um recurso bastante limitado, pois não existem facilidades para mapear imagens de outras fontes (é necessário um programa auxiliar para converter uma imagem em uma representação aceita pela *OpenGL*).

4.3 Mapeamento de Ambiente no OpenGL

Nesta seção, é apresentado como aplicar o Mapeamento de Ambiente, com a OpenGL utilizando o Mapa da Esfera. O padrão da API não suporta o Mapa do Cubo, mas ele se encontra disponível em suas extensões.

Dentre as opções para se fazer um Mapa da Esfera, cita-se utilizar os comandos abaixo, para geração automática de coordenadas de textura.

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
```

O próximo passo é habilitar o uso do mapa, através dos comandos:

```
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
```

Recomenda-se a leitura do Anexo A, para visualização de código fonte e telas capturadas de um programa exemplo do uso de Mapeamento de Ambiente, na API OpenGL.

CONCLUSÃO

A dificuldade do Mapeamento de Ambiente está na formação da própria textura. Criar o Mapa Esférico da textura depende:

- Do ambiente.
- Da posição do observador.

Assim, conclui-se que a textura necessita ser recalculada sempre que estes mudam, mas não necessita ser computada quando um objeto se move num ambiente mapeado. Pode-se dizer que o Mapeamento de Ambiente é bom quando um observador estacionário vê objetos em movimentos num ambiente estático.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] CAVALCANTI, P.R.; ESPERANÇA, C. Apostila de Introdução à Computação Gráfica. UFRJ – 2000.

[2] TING, W.S. Notas de Aula da disciplina Computação Gráfica I. FEEC – Unicamp – 2004.

[3] WILSON, R. SPANN, M. Image segmentation and uncertainty. In: Kittler, J. ed. Pattern Recognition and Image Processing Series, Herts: RSP, 1988. v. 9, p. 354-385.

[4] JAIN, A.K. Fundamentals of digital image processing. Englewood Cliffs: Prentice Hall, 1988, 592 p.

[5] Institute of Electrical and Electronics Engineers (IEEE). IEEE standard glossary of image processing and pattern recognition terminology. New York, 1990. 16 p. (IEEE Standard 610.4-1990).

[6] GONZALEZ, R.C.; WOODS, R.E. Digital image processing. Reading: Addison-Wesley, 1992, 716 p.

[7] FAUGERAS, O.D.; PRATT, W.K. Decorrelation methods of texture feature extraction. IEEE Transactions on Pattern Analysis and Machine Inteligence, v. 2, n. 4, p. 323-332, 1980.

[8] CROSS, G.R.; JAIN, A.K. Markov randon fields texture models. In: PRIP' 81 IEEE Computer Society Conference on Pattern Recognition and Image Processing, Dallas, 1981. Proceedings. Piscataway: IEEE, 1981, p. 597-602.

[10] MENDONÇA, M.B. Dissertação de Mestrado: Aplicação de Texturas em Visualização Científica. USP – 2001.

[11] BLINN, J.; NEWELL, M. Texture and Reflection in Computer Generated Images. Communications of the ACM 19:10 (1976), 542—547.

[12] CROW, F.C. “Texture” in ROGERS, D.F.;EARNSHAW,R.A. (eds), Computer Graphics Techniques – Teory and Pratice pp. 159-187, Springer-Verlag, 1990.

[13] SEGAL, M.; AKELEY, K. The Design of the OpenGL Graphics Interface.Technical report, Silicon Graphics Inc, [on-line] http://www.opengl.org/developers/documentation/white_papers/opengl/index.html, 1996.

[14] NEIDER, J.; DAVIS, T.; MASON, W. OpenGL Programming Guide (AddisonWesley, 1993).

[15] KILGARD, M.J. OpenGL and X, Part 1:An Introduction. Technical report, SGI,http://www.sgi.com/software/opengl/glandx/intro/intro.html, 1994.

[16] KILGARD, M.J. The OpenGL Utility Toolkit (GLUT) Programming Interface API Version 3. Technical report, SGI, Http://www.opengl.org/developers/documentation/glut/spec3/spec3.html, 1996.

[17] GTK+ The GIMP Toolkit. http://www.gtk.org/, 2002.

[18] SEGAL, M.; AKEEY, K. The OpenGL Graphics Interface. Technical report, Silicon Graphics Inc, http://www.opengl.org/developers/documentation/white_papers/oglGraphSys/opengl.html, 1994.

[19] RANDIMA, F.; KILGARD, M.J. The Cg Tutorial: The Definitive Guide to Programmable Real-time Graphics Cap. Sete (Addison Wesley)

[20] AZEVEDO, E.; CONCI, A. Computação Gráfica: Teoria e Prática p.314-323, Campus – 2003.

[21] WILLIAMS, L. "Pyramidal Parametrics," Computer Graphics (SIGGRAPH), vol. 17, No. 3, Jul. 1983 pp. 1-11.

[22] GREENE, N. Environment Mapping and Other Applications of World Projections. IEEE Computer Graphics and Applications, Vol 6. No. 11. Nov. 1986.

[23] HAERBERLI, P.; SEGAL, M.Texture Mapping as a Fundamental Drawing Primitive. Fourth Eurographics Workshop on Rendering. June 1993, pp. 259-266.

APÊNDICE A

Exemplos de Cargas de Texturas no OpenGL

Não existem funções específicas no OpenGL que permitem a carga direta de um arquivo de imagem para o mapeamento de texturas. Em muitos casos a biblioteca auxiliar do OpenGL GLAUX é utilizada. Será apresentada aqui a utilização das bibliotecas padrão do C/C++ aliadas às funcionalidades do GLU/GLUT.

Carga de texturas através de arquivos

A carga de textura através de arquivos consiste na leitura física do arquivo armazenado e armazenamento do mesmo em um vetor que será tratado pelas funções de criação e mapeamento de texturas.

Passos básicos utilizando-se um raw bitmap:

1. Definição do ponteiro para o vetor que conterá a imagem a ser carregada bem como o tipo de dados:

   *GLubyte *raw_bitmap;*

2. Abertura do Arquivo:

   *FILE *file;*

   file = fopen(file_name, "rb")

3. Alocação do espaço de memória necessário para armazenamento da imagem (width representa a largura da imagem, height a altura da imagem e depth a profundidade da imagem):

   *raw_bitmap = (GLubyte *)malloc(width * height * depth * (sizeof(GLubyte)));*

4. Leitura do arquivo diretamente para memória e fechamento do mesmo:

   *fread (raw_bitmap , width * height * depth, 1 , file);*

   fclose (file);

5. Definição do modo de armazenamento e criação do objeto de textura:

   GLuint texture_id; glPixelStorei (GL_UNPACK_ALIGNMENT, 1);

   glGenTextures (1, texture_id);

   glBindTexture (GL_TEXTURE_2D, texture_id);

6. Definição dos tipos de filtros a serem aplicados na textura:

Comando: glTexParameter()*

7. Definição do ambiente de textura:

Comando : glTexEnvf ()

8. Construção dos mipmaps exemplo:

gluBuild2DMipmaps (GL_TEXTURE_2D, colour_type, width, height, colour_type, GL_UNSIGNED_BYTE, raw_bitmap);

9. Liberação do espaço de memória reservado para carga:

free (raw_bitmap);

Uma vez que os passos foram seguidos a textura está pronta para ser utilizada pelo programa.

O formato *RAW*, utilizado nos passos descritos, é basicamente um tipo de formato de importação e exportação ao invés de um formato de armazenamento contendo os valores “brutos” da imagem. Este tipo de formato pode ser gerado através de ferramentas gráficas. Qualquer outro tipo de formato pode ser carregado como textura no OpenGL. No entanto funções específicas de cargas devem ser criadas. Para os arquivos do tipo bmp e jpg o tamanho da imagem (altura e largura) devem ser do tipo 2 elevado a n em modo RGB.

ANEXO A

Para exemplificar a aplicação do mapeamento de ambiente utilizando a *OpenGL*, com suas devidas funções, é apresentado a seguir o código fonte que foi adaptado para utilizar esses recursos. Na Figura A-1, é mostrada o *Teapot* que recebeu a textura, sem a ocorrência do mapeamento para o ambiente, como é apresentado na Figura A-2.

Figura A-1 – *Teapot* utilizando mapeamento de textura

Figura A-2 – *Teapot* com texturização baseada no mapeamento de ambiente

Nas Figuras A-3 e A-4, são apresentadas as imagens texturizadas com mapeamento de ambiente estando rotacionadas diferentemente na cena, para melhor visualização dos resultados do trabalho.

Figura A-3 – *Teapot* rotacionado com texturização utilizando mapeamento de ambiente

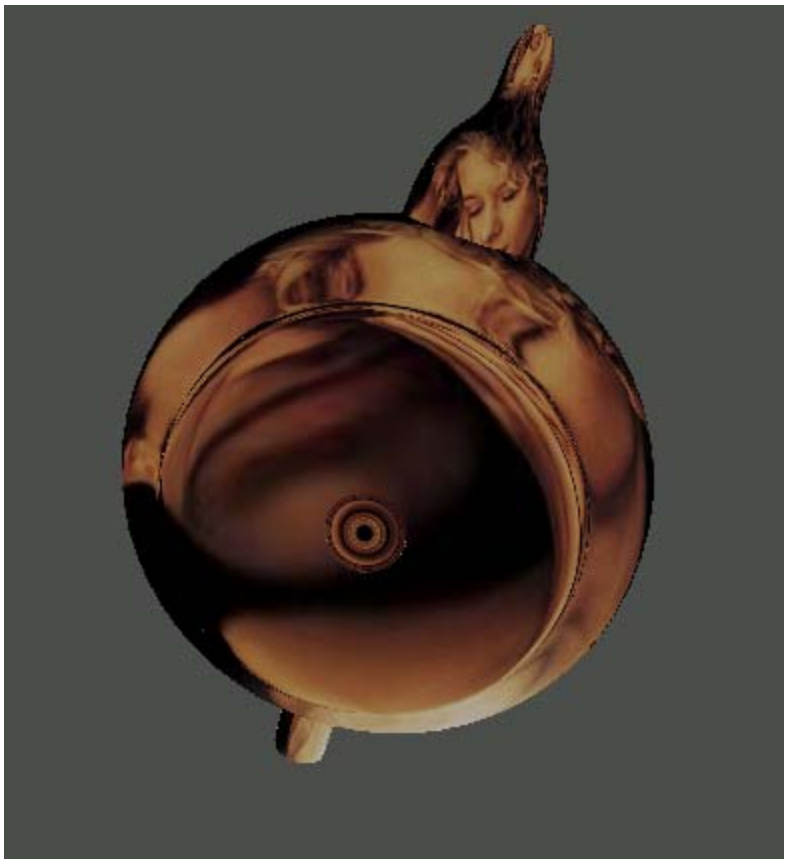

Figura A-4 – *Teapot* rotacionado diferentemente utilizando mapeamento de ambiente

A seguir é apresentado o código fonte que gerou as imagens com mapeamento de ambiente, tendo sido desenvolvido no ambiente *Microsoft*

Windows utilizando o compilador *Borland C++* juntamente com a biblioteca gráfica OpenGL.

```
/*
*     Data...........: 22/06/2004
*     Aplicativo.....: Environment Mapping
*     Objetivo.......: Apresentar os resultados do uso de mapeamento de
*                      ambiente  utilizando texturas
*     Desenvolvedores:
*                 Carlos Henrique da Silva Santos
*                 Milena Alexandre dos Santos
*/

#include <windows.h>
#include <math.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <stdio.h>

static int year = 0, day = 0;
int posicaoluz = 0;
const int img_width = 256;
const int img_height = 256;
GLubyte the_image[img_height][img_width][3];

int tx;
GLuint  texture_id[1];

/*
  Função responsável pela carga de
  um arquivo BMP

  Esta função utiliza leitura direta do BMP sem
  a necessidade de outras bibliotecas assim
  segue abaixo a descrição de cada deslocamento
  do Header.
  Referencia :http://www.fastgraph.com/help/bmp_header_format.html

 Formato do header de arquivos BMP (Windows)
 Windows BMP files begin with a 54-byte header:
 offset  size   description
   0       2     signature, must be 4D42 hex
   2       4     size of BMP file in bytes (unreliable)
   6       2     reserved, must be zero
   8       2     reserved, must be zero
  10       4     offset to start of image data in bytes
  14       4     size of BITMAPINFOHEADER structure, must be 40
  18       4     image width in pixels
  22       4     image height in pixels
  26       2     number of planes in the image, must be 1
  28       2     number of bits per pixel (1, 4, 8, or 24)
  30       4     compression type (0=none, 1=RLE-8, 2=RLE-4)
  34       4     size of image data in bytes (including padding)
  38       4     horizontal resolution in pixels per meter (unreliable)
  42       4     vertical resolution in pixels per meter (unreliable)
  46       4     number of colors in image, or zero
  50       4     number of important colors, or zero

*/

int  LoadBMP(char *filename)
 {

    #define SAIR        {fclose(fp_arquivo); return -1;}
    #define CTOI(C)     (*(int*)&C)

    GLubyte     *image;
    GLubyte     Header[0x54];
```

```
        GLuint      DataPos, imageSize;
        GLsizei     Width,Height;

        // Abre o arquivo e efetua a leitura do Header do arquivo BMP
        FILE * fp_arquivo = fopen(filename,"rb");
        if (!fp_arquivo)
        {
           printf("1--");
            return -1;
        }
        if (fread(Header,1,0x36,fp_arquivo)!=0x36)
        {
              printf("2--");
            SAIR;
        }
        if (Header[0]!='B' || Header[1]!='M')
        {
                  printf("3--");
            SAIR;
        }
        if (CTOI(Header[0x1E])!=0)
        {
                  printf("4--");
            SAIR;
        }
        if (CTOI(Header[0x1C])!=24)
        {
                  printf("5--");
            SAIR;
        }

        for (int j=0; j < 54; j++)
        {
          printf ("Header[%i]=%i \n", j, Header[j]);
        }

        // Recupera a informação dos atributos de
        // altura e largura da imagem

        Width   = CTOI(Header[0x12]);
        Height  = CTOI(Header[0x16]);
        (  CTOI(Header[0x0A])  ==  0  )  ?  (  DataPos=0x36  )  :  (  DataPos  =
CTOI(Header[0x0A]) );

        imageSize=Width*Height*3;
        printf("ImageSize=%i\n", imageSize);
        // Efetura a Carga da Imagem
        image = (GLubyte *) malloc ( imageSize );
        int retorno;
        retorno = fread(image,1,imageSize,fp_arquivo);

        if (retorno !=imageSize)
         {
            free (image);
            SAIR;
         }
        // Inverte os valores de R e B
        int t, i;
        for ( i = 0; i < imageSize; i += 3 )
         {
            t = image[i];
            image[i] = image[i+2];
            image[i+2] = t;
         }
        // Tratamento da textura para o OpenGL

        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT);
        glTexEnvf ( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE );
```

```
         // Geraçao da textura na memória
         glTexImage2D(GL_TEXTURE_2D,   0,   GL_RGB,   Width,   Height,   0,   GL_RGB,
GL_UNSIGNED_BYTE, image);

         fclose (fp_arquivo);
         free (image);
         return 1;
      }


     void init(void)
     {

        tx=0;
        /* Cria as matrizes responsáveis pelo
           controle de luzes na cena */

        GLfloat ambiente[]   = { 1.0, 1.0, 1.0, 1.0 };
        GLfloat difusa[]     = { 1.0, 1.0, 1.0, 1.0 };
        GLfloat especular[]  = { 0.5, 0.5, 0.5, 1.0 };
        GLfloat posicao[]    = { 0.0, 3.0, 2.0, 1.0 };
        GLfloat lmodelo_ambiente[] = { 0.2, 0.2, 0.2, 1.0 };

        glClearColor(0.3, 0.3, 0.3, 1.0);
        glEnable(GL_DEPTH_TEST);
        glShadeModel(GL_SMOOTH);

        /*
          Habilita a Texturizacao.
          Criacao inicial das texturas.
        */
        glEnable ( GL_TEXTURE_2D );
        glPixelStorei ( GL_UNPACK_ALIGNMENT, 1 );
        glGenTextures ( 1, texture_id );
        glBindTexture ( GL_TEXTURE_2D, texture_id[0] );
        LoadBMP ("amb2.bmp");

        /* Cria e configura a Luz para a cena */
        glLightfv(GL_LIGHT0, GL_AMBIENT, ambiente);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, difusa);
        glLightfv(GL_LIGHT0, GL_POSITION, posicao);
        glLightfv(GL_LIGHT0, GL_SPECULAR, especular);
        glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodelo_ambiente);

        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glEnable(GL_COLOR_MATERIAL);

     }

     void display(void)
     {
       /* Variáveis para definição da capacidade de brilho do material */
       GLfloat semespecular[4]={1.0,1.0,1.0,0.0};
       GLfloat especular[] = { 1.0, 1.0, 1.0, 1.0 };

       /* Posição da luz */
       GLfloat posicao[] = { 0.0, 0.0, 0.0, 0.0 };

       /*
         Limpa o buffer de pixels e
         determina a cor padrão dos objetos.
       */
       glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
       glColor3f (1.0, 1.0, 1.0);

       /* Armazena o estado anterior para
          rotação da posição da luz */

       glPushMatrix () ;
        glRotated ((GLdouble) posicaoluz, 1.0, 0.0, 0.0);
        glLightfv (GL_LIGHT0, GL_POSITION, posicao);
```

```
  glPopMatrix(); // Posição da Luz

  /* Armazena a situação atual da pilha de matrizes */
  glPushMatrix ();
  glRotatef (tx, 1.0, 0.0, 0.0);
  glTranslatef( 0.0, 0.0, 2.0);
  glPushMatrix ();
  glTranslatef (0.0, 0.0, -3.0);
  glPushMatrix ();
  glRotatef (9, 0.0, 0.0, 1.0);
  glPushMatrix();
  glRotatef ((GLfloat) year, 1.0, 0.0, 0.0);
  glRotatef ((GLfloat) day, 0.0, 1.0, 0.0);
  glColor3f (1.0, 1.0, 1.0);

  /* Define a propriedade do material */
  //refletância do material
  glMaterialfv(GL_FRONT,GL_SPECULAR, semespecular);
  // Define a concentração do brilho
  glMateriali(GL_FRONT,GL_SHININESS,20);

  /* Habilita o mapeamento do ambiente */
  glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
  glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
  glEnable(GL_TEXTURE_GEN_S);
  glEnable(GL_TEXTURE_GEN_T);

   /* Criar o objeto  */
   glColor3f(0.6,0.6,0.6);
   glutSolidTeapot(1.0);

   glPopMatrix();
   glPopMatrix();
   glPopMatrix();
   glPopMatrix();

   // Executa os comandos
   glutSwapBuffers();
}

void reshape (int w, int h)
{

  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt (0.0, 1.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
}

void keyboard (unsigned char key, int x, int y)
{
  switch (key) {
      case 'd':
      case 'D':
           glDisable(GL_TEXTURE_2D);
           glutPostRedisplay();
           break;
      case 'A':
      case 'a':
           glEnable(GL_TEXTURE_2D);
           glutPostRedisplay();
           break;
  }
}

/*
  Esta função é chamada quando o botão esquerdo do
  mouse é pressionado, a mesma irá calcular um novo
  valor para os valores dos ângulos contidos em year e day
```

```
*/

void spinDisplay(void)
{
  year = (year + 1) % 360;
  day  = (day + 2 ) % 360;
  tx = (tx + 1) % 360 ;

  glutPostRedisplay();
}

void mouse(int button, int state, int x, int y)
{
  switch (button) {
      case GLUT_LEFT_BUTTON:
           if (state == GLUT_DOWN)
              glutIdleFunc(spinDisplay);
           break;
      case GLUT_MIDDLE_BUTTON:
           if (state == GLUT_DOWN)
              glutIdleFunc(NULL);
           break;
      case GLUT_RIGHT_BUTTON:
           posicaoluz = (posicaoluz + 1) % 360;
           glutPostRedisplay();
           break;
      default:
           break;
  }
}

/*
 Função principal do programa.
*/
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (800, 600);
  glutInitWindowPosition (100, 100);
  glutCreateWindow ("Mapeamento de Textura");
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMouseFunc(mouse);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```