MAPA DE TEXTURA: MIP-MAPPING

Renata Corrêa
Silmara Pedretti Gomes

Monografia apresentada à Faculdade de Engenharia Elétrica e da Computação da Universidade de Campinas, Departamento de Engenharia da Computação e Automação Industrial, para a avaliação da disciplina IA725 - computação gráfica, ministrada pela Professora Doutora Wu Shin Ting.

Campinas / SP
2004

Renata Corrêa
Silmara Pedretti Gomes

MAPA DE TEXTURA: MIP-MAPPING

Monografia apresentada à Faculdade de Engenharia Elétrica e da Computação da Universidade de Campinas, Departamento de Engenharia da Computação e Automação Industrial, para a avaliação da disciplina IA725 - computação gráfica, ministrada pela Professora Doutora Wu Shin Ting.

Campinas / SP
2004

SUMÁRIO

Resumo

Este trabalho visa descrever o uso de texturas em computação gráfica, detalhando especificamente o mapeamento de textura conhecido como mip-mapping. Essa técnica é de grande utilidade na correção de alguns efeitos indesejados que podem ocorrem em imagens sintéticas. Uma etapa desta pesquisa consiste em apresentar exemplos práticos da implementação de mip-mapping em programas que suportam essa técnica, focando principalmente seu uso em OpenGL.

Palavras-chave: textura, mip-mapping, OpenGL

Introdução

Criar imagens sintéticas realistas é o objetivo final da computação gráfica [1].

Conseguir que uma imagem sintética se aproxime da perfeição fotográfica não é uma tarefa trivial. Isso envolve um conjunto de características que devem ser definidas para um objeto, mas talvez a cor e a textura sejam as mais perceptíveis intuitivamente, elas acabam sendo o diferencial entre eles, um mesmo prato, por exemplo, pode ser de vidro, metal ou madeira.

Com isso podemos notar que o estudo da aplicação de texturas é uma área importante em Computação Gráfica, com muitos assuntos ainda a serem conhecidos, pesquisados e desenvolvidos.

Algumas técnicas de textura foram criadas com o objetivo de caracterizar os materiais de cada detalhe de uma cena, outras surgiram para solucionar problemas e melhorar o desempenho gráfico, como é o caso do Mip-Mapping – detalhado no capítulo 4. Veremos também alguns exemplos práticos de texturas utilizadas em programas, filmes e jogos.

Concluímos este trabalho com a apresentação de texturas em OpenGL – uma biblioteca gráfica bastante conhecida atualmente e que suporta o uso de mip-mapping.

1. Realismo visual na sintetização de imagens

1.1. Computação Gráfica

A Computação Gráfica é uma área da ciência da computação, que segundo a definição da ISO (International Standarts Organization) consiste num "*conjunto de ferramentas e técnicas para converter dados para ou de um dispositivo gráfico através do computador"*.

O processo de conversão de dados em uma imagem é também conhecido pelo nome de *visualização* [2].

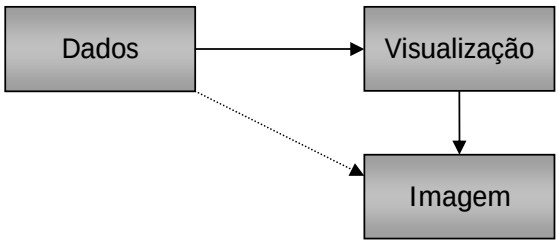


FIGURA 2.1 – COMPUTAÇÃO GRÁFICA: CONVERSÃO DE DADOS EM IMAGEM.

I

Atualmente a computação gráfica é utilizada nas mais diversas áreas, aplicações e setores da sociedade como: indústria, governo, educação, entretenimento, área médica, porém ela tem origem relativamente recente, começou a surgir na década de 50.

Seu desenvolvimento se deve muito à evolução do hardware computacional, como o surgimento da tecnologia dos circuitos integrados na década de 70, permitindo o barateamento das máquinas. Ao mesmo tempo, foram sendo desenvolvidos os algoritmos que até hoje são utilizados, aproximando cada vez mais as imagens geradas sinteticamente do realismo fotográfico.

1.1.1. Áreas correlatas

O grande número de aplicações das técnicas de computação gráfica a coloca algumas vezes em um posicionamento em relação a áreas tão próximas que chegam a ser confundidas [2].

[S1] Comentário: [2] pág. 2

Podemos relacionar a computação gráfica com três grandes sub-áreas: a **Síntese de Imagens**, o **Processamento de Imagens** e a **Análise de Imagens**, conforme o diagrama:

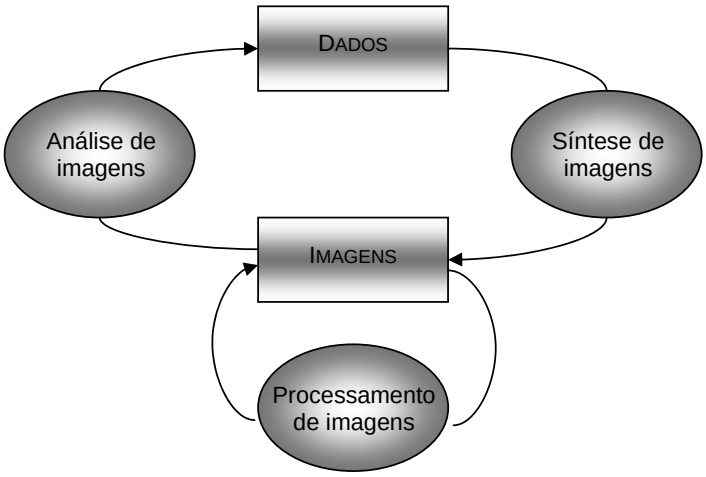


FIGURA 2.2 – RELACIONAMENTO ENTRE SUB-ÁREAS CORRELATAS

Essas três áreas são a base na qual se fundamentam os processos computacionais que envolvem a imagem. A análise de imagens é tratada pela visão computacional, onde a imagem está presente na fase inicial do processo. A síntese de imagens é abordada pela computação gráfica, neste caso temos a imagem como produto final. Há também o processamento de imagens que envolve as técnicas de transformação visando melhorar as características visuais da imagem.

1.2. Síntese de imagens

A computação gráfica trata da síntese de imagens através do computador. Sintetizar uma imagem não é mostrá-la no computador digitalmente através da captura de algo existente. Isso é tratado no Processamento de Imagens [1].

Criar uma imagem computacionalmente significa definir sua geometria e todos os demais atributos, sejam eles diretamente relacionados com o objeto (como a propriedade dos materiais), quanto indiretamente (como condições de luz do ambiente e posição do observador da cena).

1.3. Realismo Visual

Em geral queremos que a imagem criada seja convincente, isto é, mais próxima possível de sua figura real. Isso nem sempre é uma tarefa fácil, porém utilizamos todos os recursos possíveis para aproximar a imagem sintética do realismo visual.

Podemos classificar o realismo em duas etapas: a dinâmica – relacionada com o movimento da cena e presente especialmente em animações – e a estática – que tenta conseguir o realismo fotográfico ou "fotorrealismo".

Algumas vezes simplesmente desejamos alterar intencionalmente a realidade para por exemplo gerar cenas de ficção científica, ou enfatizar detalhes mudando a iluminação normal de um objeto e neste caso não queremos efeitos fotorrealistas.

Uma dificuldade fundamental em se obter o realismo visual por inteiro é a complexidade do mundo real [3]. A riqueza no detalhe do nosso ambiente, a singularidade de cada item, com suas cores, texturas, formatos, acaba sendo um grande problema para simular uma realidade "perfeita". Por outro lado, como a visão humana é de certa forma falha, conseguimos obter resultados

satisfatórios simplificando a modelagem da realidade e “enganando” nossos olhos, discretizando cores e efeitos luminosos e criando um fantástico mundo de ilusão na tela do computador.

1.3.1. Fases do realismo visual

Para exibir imagens 3D numa superfície 2D apropriadamente as ações necessárias são divididas em algumas etapas. Como o processamento computacional tem um custo alto, o nível do realismo deve estar adequado ao seu uso para não despender recursos desnecessariamente, neste caso podemos utilizar apenas algumas etapas e não necessariamente todas ao mesmo tempo.

a) construção do modelo geométrico;

b) aplicação de transformações lineares – projeção paralela ou perspectiva para dar uma aparência tridimensional;

c) eliminação de arestas e superfícies ocultas (relativo a posição do observador);

d) recorte do volume de visualização (desconsiderando as partes da cena que estão fora do alcance da janela de visão do observador);

e) rasterização (representar os dados em pixels);

f) tratamento de partes escondidas (sobrepostas por outro objeto, por exemplo);

g) colorização dos pixels (levando em conta luzes, sombras, textura, transparência, brilho, etc).

2. Texturas

Observamos que praticamente tudo que está a nossa volta possui um certo padrão em sua superfície. Pele, tecidos, madeira, metal, folhagens, padrões mais ou menos detalhados, tudo isso pode se resumir numa palavra: textura.

Perceber uma imagem tridimensional pode ser definida como a capacidade que temos de distinguir a forma, as cores, a textura e a relação espacial entre os objetos[1].

Através da luz refletida numa superfície conseguimos reconhecer sua forma e curvatura, porém este não é um modelo apropriado para descrever todas as suas propriedades como rugosidade e padronagem. Detalhes podem ser modelados na geometria do objeto ou através da diferenciação do material, porém quando se deseja criar padrões mais complexos ou repetitivos essa solução se torna cara e consome muito processamento, a saída mais simples é obter esses efeitos através do uso de mapas de texturas.

As texturas, juntamente com a luz, auxiliam até mesmo na percepção do movimento. Se rotacionarmos continuamente duas esferas, uma “lisa” e outra devidamente texturizada notaremos muito mais facilmente o movimento desta última.

2.1. Tipos de mapeamentos

Por ser a textura a interação da luz com a superfície do objeto, os parâmetros da função de iluminação (normal, luz difusa, luz especular) podem ser modulados por uma função de textura[4].

Com isso podemos citar tipos diferentes de mapeamento, de acordo com o parâmetro modulado:

a) mapeamento de textura: é a modulação da luz difusa, ou seja, a própria cor do objeto;

b) mapeamento do ambiente (environment mapping): é a modulação da luz especular;

c) mapeamento de rugosidade (bump mapping): é a modulação da normal (perturbando-a);

d) mapeamento de transparência: modulação do coeficiente de transparência;

e) mapeamento de refração: modulação do coeficiente de refração.

2.1.1.Mapeamento de textura

O mapeamento de textura corresponde à projeção de uma imagem (usualmente 2D), digitalizada ou sintetizada, sobre uma superfície 3D, uma técnica criada por Catmull[5] e refinada por Blinn e Newell[6]. A imagem é chamada de mapa de textura, ela forma uma matriz retangular e cada elemento é chamado de *texel*. Um pixel é freqüentemente coberto por um número maior de texels [3].

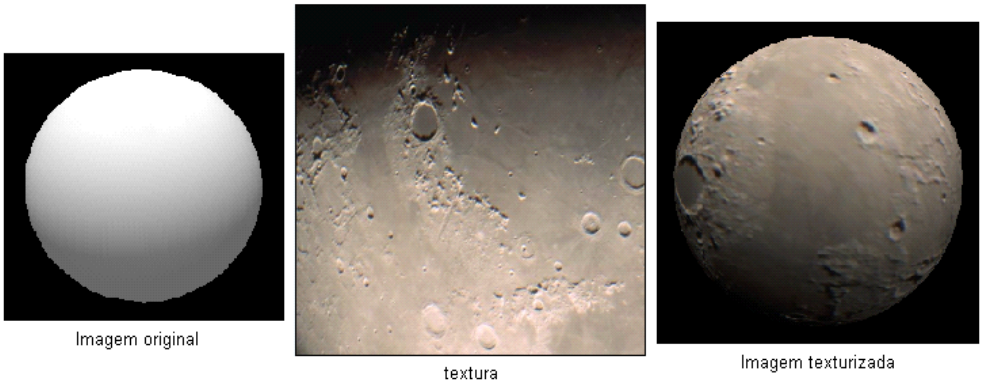


FIGURA 3.1: MAPEAMENTO DE TEXTURA NUMA ESFERA

No mapeamento de texturas, uma imagem é usada essencialmente como uma tabela de consulta: se um pixel da projeção está para ser desenhado, uma consulta é feita na imagem. A cor que estiver na imagem na posição consultada, torna-se a cor que será usada no modelo. A cor descoberta checando-se a imagem pode ser usada sem modificações, ou pode ser misturada com a cor da iluminação do modelo.

Mas como criar uma correspondência entre o objeto e a imagem?

Bem, primeiro criamos coordenadas para o mapa de textura, representadas aqui como (u, v). Depois, se a superfície puder ser descrita com em forma paramétrica, ela pode servir de base para o mapeamento (exemplo: planos, cilindros, esferas).

Para objetos poligonais ou mais complexos, que não suportam uma parametrização natural, Blinn sugeriu um mapeamento em duas etapas:

1) a textura 2D é mapeada para um objeto 3D simples como um plano, cubo, cilindro ou esfera;

2) a nova textura 3D (criada através do passo 1) é mapeada num objeto final.

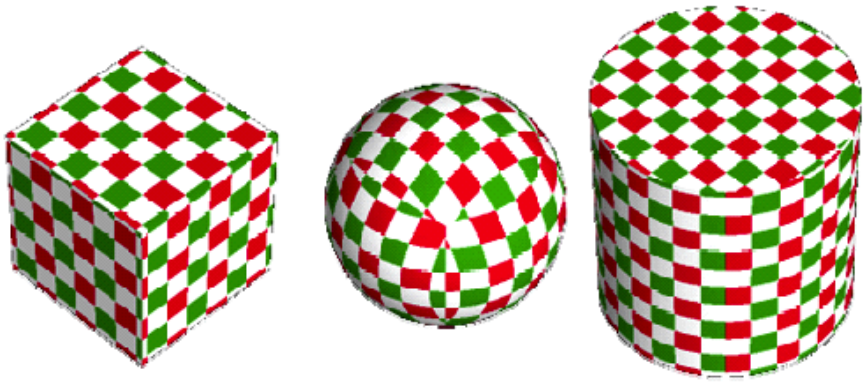


FIGURA 3.2: PARAMETRIZAÇÃO CÚBICA E DUAS PROJEÇÕES – NUMA ESFERA E NUM CILINDRO [7]

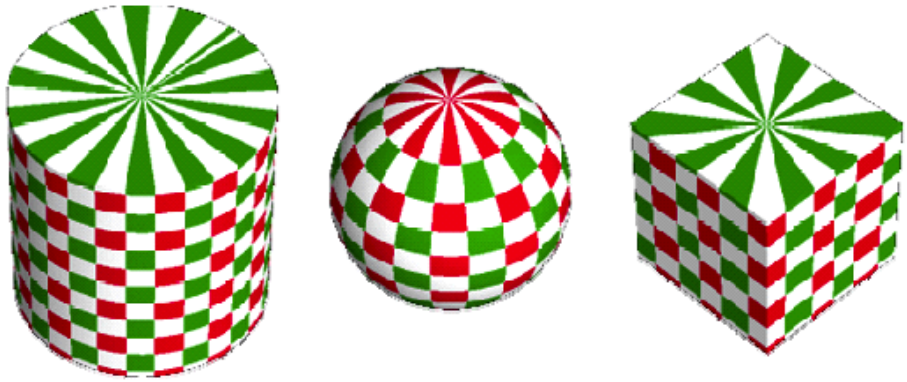


FIGURA 3.3: PARAMETRIZAÇÃO CILÍNDRICA E DUAS PROJEÇÕES – NUMA ESFERA E NUM CUBO [7]

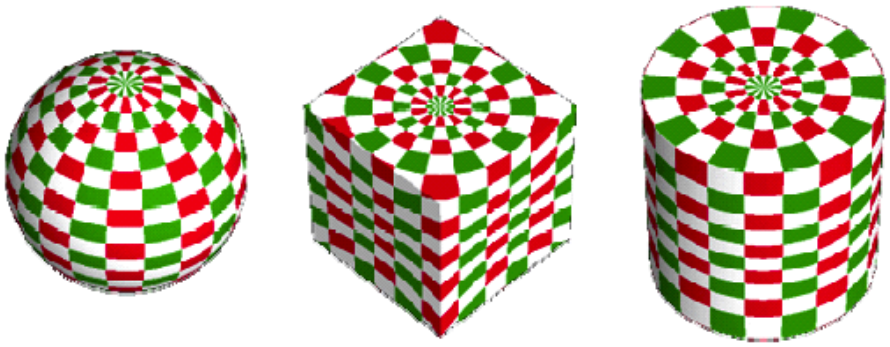


FIGURA 3.4: PARAMETRIZAÇÃO ESFÉRICA E DUAS PROJEÇÕES – NUM CUBO E CILINDRO [7]

2.2. Magnificação e minificação

Os mapas de textura são quadrados ou retangulares e depois de serem mapeados para polígonos ou superfícies e transformados em coordenadas de tela, cada texel raramente corresponderá a exatamente um pixel.

Quando uma grande área da textura é mapeada para uma pequena área da imagem, um grande número de texels será mapeado para um único pixel. Se usarmos a cor de um só texel para colorir o pixel, um padrão indesejado será criado na imagem (aliasing). A esse problema chamamos de minificação.

Quando ocorre o inverso, ou seja, quando um único pixel é maior que um texel, o problema é chamado de magnificação e pode ser corrigido através de filtros que fazem a interpolação bilinear de vários texels para se encontrar o valor apropriado para exibir um pixel.

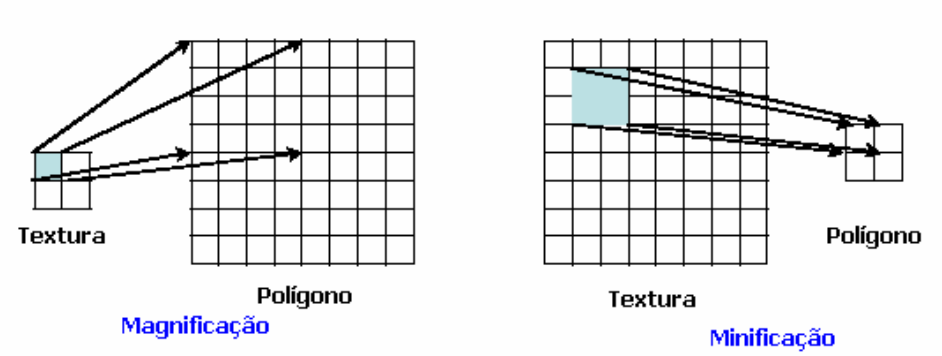


FIGURA 3.5: PROBLEMAS GERADOS POR MAPEAMENTO DE TEXTURA

3. Mip-Mapping

3.1. Definição

MIP vêm das iniciais de “Multum In Parvo” e significa: muitas coisas num lugar pequeno. É uma técnica de filtragem utilizada para melhorar a qualidade visual de imagens sintéticas, criada por Williams [9].

3.2. Problemas que o mip-mapping visa resolver

Como vimos nos capítulos anteriores, a exibição de imagens 3D realistas na tela de um computador depende de vários fatores e um deles é a impressão de profundidade nas mesmas. Num computador, a profundidade é obtida exibindo-se objetos menores conforme se aumenta a distância da câmera. O primeiro problema ocorre justamente com o afastamento dos objetos em relação ao observador (*minificação*), causando ruídos indesejados (*moire*) na imagem.

Além disso, quando os objetos numa cena ficam muito pequenos, a tela do computador não consegue mais mostrar todos os seus detalhes, mesmo quando nos aproximamos de tais objetos, perdendo uma parte de sua informação visual.

O mip-mapping é utilizado para solucionar esses dois problemas no mapeamento de objetos, melhorando a qualidade visual de gráficos e animações geradas por computador.

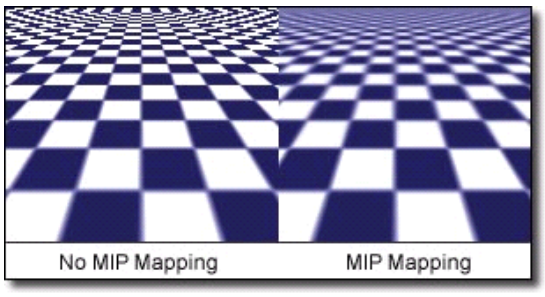


FIGURA 4.1: IMAGEM GERADA SEM O USO DE MIP-MAPPING E COM O MESMO.

3.3. *Como funciona*

A técnica consiste em pré-filtrar a imagem, criando múltiplas cópias de um mapa de textura, todas derivadas da primeira, com resolução cada vez menor, geralmente na proporção de metade do tamanho anterior, até chegar a 1x1 pixel. Em distâncias próximas a textura de tamanho original é exibida, para distâncias maiores os mapas menores vão sendo mostrados. Cada estágio desses é conhecido como “mip map level”.

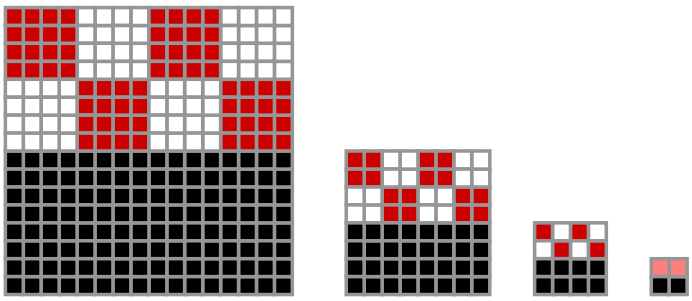


FIGURA 4.2: IMAGEM EM DIFERENTES RESOLUÇÕES [8]

Se a imagem original possui uma resolução de 64X64 pixels então a do nível seguinte possui 32X32, a próxima 16X16, 8x8, 4x4 e, finalmente, 1X1 pixel.

O mapa resultante ocupa 4/3 da memória de um mapa comum. Os canais RGB da imagem preenchem ¾ do MIP map. Cada quartil sucessivo restante é preenchido com versões filtradas da imagem.

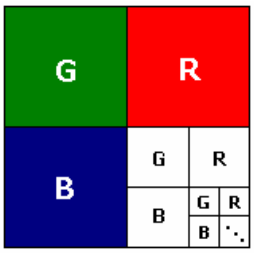


FIGURA 4.3: MIP MAP. CADA NOVO NÍVEL DO MAPA OCUPA ¾ DO ESPAÇO RESULTANTE.[3]

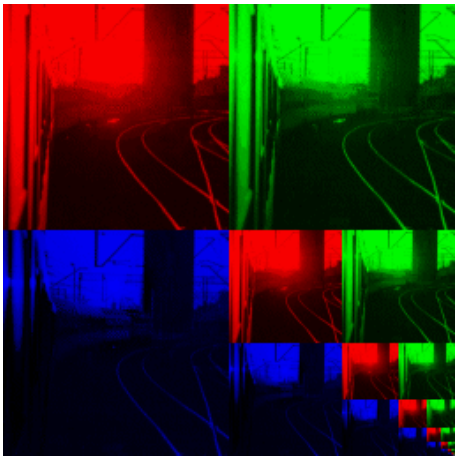


FIGURA 4.4: UMA REPRESENTAÇÃO GRÁFICA DA ESTRUTURA DE DADOS UM MIP MAP[13]

3.4. O fator de compressão

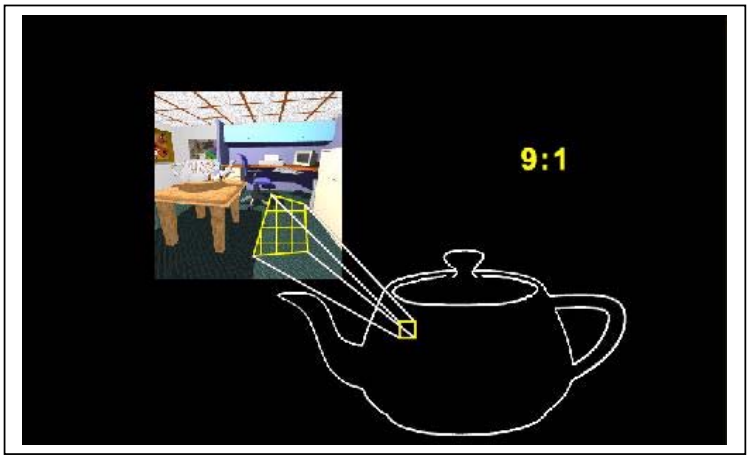
Para decidir qual nível de mapa utilizar, temos que calcular o fator de compressão. Calcular esse fator para cada pixel é custoso, então costuma-se fazer isso através de áreas. O fator de compressão é a área retangular da textura dividida pela área do espaço retangular que o pixel ocupa.

O fator de compressão nos diz quantos texels cabem num pixel, na média. Quando acessamos o MIP map, o seguinte cálculo é feito: dados u, v e o fator de compressão “d”, encontre a cor do pixel.

Se d <=1, nós temos menos de um texel por pixel. O mapa de maior resolução é sempre usado para retornar a cor do texel correspontente a u, v.

Se d >= tamanho do bitmap, sabemos que o bitmap inteiro cabe dentro de um único pixel. É retornada a cor do texel do canto inferior direito (o bitmap inteiro é aproximado por um único texel).

Se d>1 e < tamanho do bitmap, encontramos as duas potências de 2 entre as quais d se encontra. Esses serão os tamanhos dos bitmaps usados para o cálculo da cor do pixel. Um deles tem muitos detalhes, o outro poucos. Através dos valores de u e v são calculadas as duas cores possíveis. A interpolação entre essas duas cores na proporção onde d fica entre os dois mapas resulta na cor final do pixel.



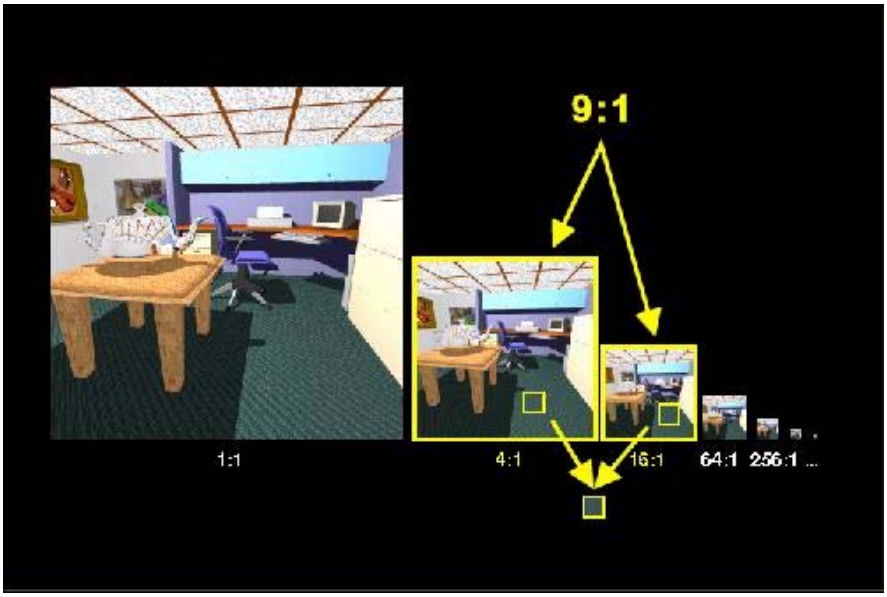


FIGURA 4.5: ESCOLHA DOS DOIS MAPAS MAIS APROPRIADOS PARA O PIXEL[14]

A cor é calculada num nível específico através de uma interpolação trilinear. O ponto (u,v) cai entre o centro de quatro texels. Baseados no valor atual de u, primeiro interpolamos dos dois texels superiores, depois repetimos para os dois valores inferiores. Finalmente interpolamos os dois novos valores obtidos usando o valor atual de v.

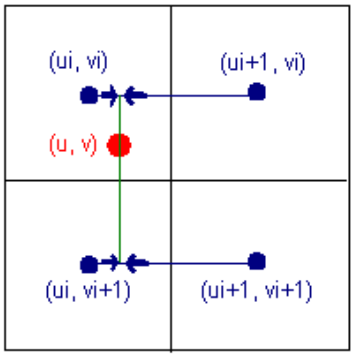


FIGURA 4.6: A INTERPOLAÇÃO LINEAR ENTRE CORES NUM MAPA DE PIXEL[13]

Interpolações lineares entre os níveis de filtragem são utilizadas para conseguir uma transição mais suave de valores.

3.5. Combinação de Mip-mapping com outras técnicas

Mip-mapping assume que a região de textura é quadrada. Para regiões retangulares, é aconselhado o uso de outras técnicas como “summed-area tables”, proposta por Crow [10].

3.5.1. Summed-area tables

É um algorítimo de pré-filtragem que pode melhorar muito a velocidade do cálculo médio do pixel em áreas retangulares. Ao invés de se armazenar a textura como um bitmap, é criada uma tabela de soma de áreas, ou seja, cada entrada na tabela de soma “S” representa a soma dos valores de todos os

componentes RGB dos texels no retângulo formado pelos pontos (0,0) e (u,v). Nós podemos computar o valor de outras áreas baseados na identidade algébrica:

$$ab= (u+a)(v+b) – (u+a)v – u(v+b) + uv.$$

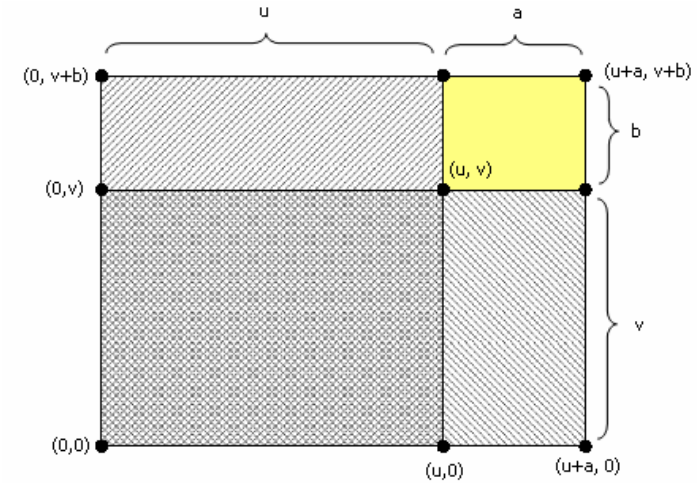


FIGURA 4.6: SUMMED-AREA TABLE- INTERPRETAÇÃO GEOMÉTRICA

Dividindo a soma pelo número de texels contidos no retângulo de interesse chegamos ao valor médio do texel.

As tabelas de soma de áreas faz com que a o valor médio de cada texel seja encontrado rapidamente porém requer quatro vezes mais memória do que a exigida pela textura original da imagem.

Um efeito indesejado é que a soma de áreas pode, em alguns casos, “borrar” o resultado final excessivamente [11].

3.5.2. Proposta de Heckbert

Heckbert [12] propõem um sistema utilizando a o cálculo do valor médio do pixel combinado com MIP maps.

Assumindo que um pixel é circular, sua projeção no espaço de textura irá produzir uma região elíptica de texels. Dependendo do tamanho dessa região, um nível apropriado de MIP map é selecionado e os texels contidos

nessa área serão utilizados para o cálculo do valor médio do pixel. Essa combinação de técnicas produz resultados melhores para as imagens sintéticas.

3.6. Utilização prática de Mip-mapping

O Nintendo 64 causou um enorme rebuliço no mercado, devido ao seu propalado poder gráfico e a capacidade de criar personagens, cenários e efeitos muito mais realistas do que a concorrência. A primeira coisa que se nota em um jogo de N64, é o aspecto "clean" das texturas, cenário e caracteres. Efeitos especiais como o mip-mapping e anti-aliasing (correção dos contornos da imagem) foram usados pela primeira vez em um videogame.

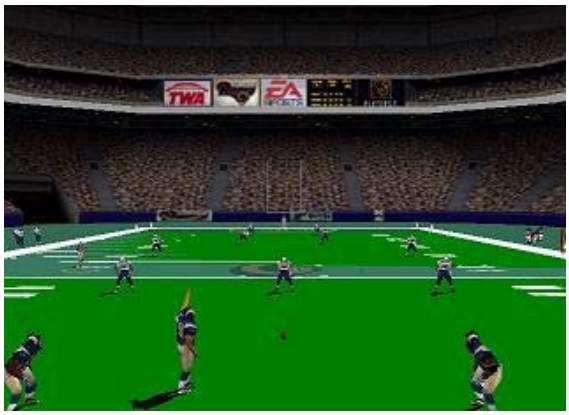


FIGURA 4.7: MADDEN NFL 2001

FIGURA 4.8: BUG'S LIFE

FIGURA 4.9: MARIO 64

Especificações técnicas do Nintendo 64

O Nintendo 64 ostenta um potente processador derivado de modelos utilizados em estações Silicon Graphics. O seu processador gráfico ajuda na renderização das imagens e geração de sons, centralizando todas as tarefas importantes em um jogo.

Abaixo, as especificações técnicas em inglês, retiradas do site da Nintendo:

CPU: MIPS 64-bit RISC CPU (customized R4000 series);
Clock Speed: 93.75 MHz;
MEMORY: RAMBUS D-RAM 36M bit;
Transfer Speed: maximum 4,500M bit/sec;
CO-PROCESSOR:
RCP: SP (sound and graphics processor) and DP (pixel drawing processor) incorporated;
Clock Speed: 62.5MHz;
RESOLUTION: 256 x 224 ~ 640 x 45 dots;
Flicker-free interlace mode support;
COLOR: 32-bit RGBA pixel color frame buffer support;
21-bit color video output;
GRAPHICS PROCESSING:
Z buffer;
Anti-aliasing;

- Realistic texture mapping:
- Tri-linear filtered MIP-map interpolation;
- Perspective correction;
- Environment mapping.

4. Aplicando texturas em OpenGL

4.1. OpenGL

"OpenGL é uma biblioteca de rotinas gráficas e de modelagem, 2D e 3D, que estabelece para o programador uma interface com o hardware gráfico, independente da plataforma. OpenGL não é uma linguagem de programação, é uma poderosa e sofisticada API (*Application Programming Interface*) para criação de gráficos 3D" [15].

4.2. Texturas

"O padrão OpenGL exige que as dimensões das imagens de texturas devam ser em potências de 2. As imagens de textura podem também ter 1 ou dois pixels de borda sobre suas extremidades para definir a cor dos polígonos que estão fora da imagem de textura" [16].

"O uso de texturas requer a execução de dois passos distintos: a carga e a aplicação da textura. Em OpenGL cada textura recebe um número de identificação através da função glGenTextures que é usado para definir a textura corrente. Sempre que for necessário trabalhar com uma textura (para carga ou aplicação) deve-se chamar a função glBindTexture que define a textura corrente através do número de identificação" [16].

4.3. Carga de uma textura

Para realizar a tarefa de carga de uma textura é necessário seguir os seguintes passos:

- Identificar o objeto de textura (através da criação de um nome) para representar a imagem de textura.

glGenTextures(GLsize n, GLuint *textures);

onde, n contém o número de nomes de textura a ser criado e textures é um array que contém os nomes das texturas armazenadas.

- Criar um objeto de textura propriamente dito associando a um nome de textura criado anteriormente.

glBindTexture(GLenum target, GLuint textureName);

sendo, textureName o nome de um objeto de textura do tipo especificado por target, que pode ser GL_TEXTURE_2D, define que será usada uma textura 2D ou GL_TEXTURE_1D, define que será usada uma textura de uma dimensão.

- Especificar a imagem para o objeto de textura criado anteriormente

glTextImage*D(GLenum target, GLint level, GLint internalformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *pixels);

onde,

target → GL_TEXTURE_2D ou GL_TEXTURE_1D.

level → dever ser 0 para uma única imagem de textura. Será diferente de zero quando usando a técnica de mip-mapping para obter níveis de detalhe diferente nas texturas.

Internalformat → descreve a representação interna da textura (GL_RGBA, GL_ALPHA, GL_LUMINANCE, etc).

Tamanho → especificado por width ou width e height.

border → indica se deseja uma borda de 0 ou mais pixels.

Fomat e type → correspondem aos mesmos parâmetros vistos para imagem (GL_RGBA, GL_RED, etc, e GL_INT, GL_SHORT_INT, etc, respectivamente).

Pixels → corresponde ao array de pixels (unidimensional, 1D ou bidimensional, 2D).

- Especificar a forma de mapeamento da textura ao objeto.

glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_MODE, modo);

sendo, modo, GL_MODULATE (default), que modula (multiplica) a cor do pixel obtido do objeto com as informações da textura; GL_DECAL, onde cor atual do pixel obtido do objeto não influenciam na imagem final (ou seja, a imagem de textura é colada); GL_BLEND, que usa a cor da textura e faz uma mistura com a cor do objeto e uma cor especificada num atributo apropriado; GL-REPLACE, quando o valor do pixel é apenas substituído da textura.

- Habilitar o uso de texturas, isso é feito através do comando:

glEnable (GLenum <contante>);

onde, o argumento constante pode ser GL_TEXTURE_2D, define que será usada uma textura 2D ou GL_TEXTURE_1D, define que será usada uma textura de uma dimensão.

4.4. Aplicando uma textura

Para a aplicação da textura é preciso criar uma relação entre os vértices da textura e os vértices dos polígonos sobre os quais se deseja mapear a textura escolhida. O processo de mapeamento de texturas em OpenGL consiste em "aplicar" a imagem 2D sobre o polígono 3D. Para permitir a construção desta correspondência entre a imagem 2D e o polígono 3D usa-se a função glTexCoord2f(x, y) antes da definição do ponto 3D [1].

4.5. Uso de mip-mapping em OpenGL

A técnica mipmapping seleciona a imagem de textura para um polígono mais próximo da resolução da tela. A carga da texturas mipmapped leva um pouco mais de tempo do que uma textura padrão, mas os resultados visuais são melhores. Além disso, as texturas mipmapped podem melhorar a performance de exibição reduzindo a necessidade de filtros de imagem GL_LINEAR.

As texturas mipmapped são definidas fornecendo um parâmetro específico de nível para cada imagem. Os níveis de imagem são especificados no primeiro parâmetro da chamada glTexImage*D(). A imagem nível 0 é a primária, com a maior resolução para a textura. A imagem nível 1 é metade do tamanho da imagem primária e assim por diante. Quando se desenha polígonos com uma textura mipmapped, é necessário o uso de filtros redutores [16].

4.6. Algoritmos de Filtros baseados em Mip-Mapping [17]

Um único pixel na tela pode corresponder a vários texels (redução) ou ser menor que um texel (ampliação). Em qualquer caso, não é claro qual valor do texel deveria ser utilizado e como deve ser calculada a sua media ou interpolada. Conseqüentemente, OpenGL permite que sejam especificados várias opções de filtros para determinar estes cálculos.

Os filtros são aplicados na textura com a utilização do comando :

glTexParameter*(GLenum target, GLenum pname,GLfloat param)

O parâmetro target deverá ser: GL_TEXTURE_1D ou GL_TEXTURE_2D, depende do tipo de mapeamento desejado unidimensional ou bidimensional.

O parâmetro pname deve seguir a tabela:

pname	**Param**
GL_TEXTURE_MAG_FILTER	GL_NEAREST ou GL_LINEAR
GL_TEXTURE_MIN_FILTER	GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_LINEAR

GL_TEXTURE_MIN_FILTER :

A função de redução é usada sempre que o pixel do mapa de textura para uma área é maior que um elemento de textura (texel).

GL_TEXTURE_MAG_FILTER :

A função de ampliação é usada quando o pixel do mapa de textura para uma área é menor ou igual a um elemento de textura (texel).

GL_NEAREST :

Retorna o valor do elemento de textura que esta mais próximo do centro do pixel a ser texturizado.

GL_LINEAR:

Retorna a média dos pesos dos quarto elementos de textura que estão mais próximos ao centro do pixel texturizado

GL_NEAREST_MIPMAP_NEAREST:

Escolhe o mipmap (conjunto ordenado de arrays que representam uma mesma imagem em resoluções progressivamente mais baixas) que é mais próxima ao tamanho do pixel a ser aplicado a textura e usa o critério de GL_NEAREST (o elemento de textura mais próximo ao centro do pixel) para produzir o valor de textura.

GL_LINEAR_MIPMAP_NEAREST:

Escolhe o mipmap que mais se aproxima do tamanho do pixel a ser aplicado a textura e utilize GL_LINEAR (o peso médio dos quatro elementos de textura que estão mais próximos ao centro do pixel) como critério para produzir o valor de textura.

GL_NEAREST_MIPMAP_LINEAR:

Escolhe dois mipmaps que mais se aproximam ao tamanho do pixel a ser mapeado. Utiliza GL_NEAREST como critério para produzir um valor de cada mipmap. O valor final de textura é o peso médio desses dois valores.

GL_LINEAR_MIPMAP_LINEAR:

Escolhe dois mipmaps que mais se aproximam ao tamanho do pixel a ser mapeado. Utiliza GL_LINEAR como critério para produzir um valor de cada mipmap. O valor final de textura é o peso médio desses dois valores.

5. Conclusão

O mip-mapping é um filtro que traz bons resultados no mapeamento de texturas e ainda evita o cálculo do valor filtrado para cada pixel, sendo especialmente útil nos casos de uma projeção perspectiva de um plano, quando um único pixel da imagem final corresponde a centenas de texels na textura.

Ele utiliza apenas 1/3 a mais de memória que o mapeamento comum e a interpolação linear entre os níveis do mapa podem ser utilizadas para suavizar as transições de valores.

Embora ele não resolva todos os problemas de aliasing, se for utilizado com outras técnicas como a summed-area table pode trazer resultados ainda melhores. Neste caso, o custo computacional não é tão alto comparado com a qualidade da imagem obtida, sendo muito utilizado atualmente em jogos e diversos softwares gráficos.

Apêndice

A.1. Programa exemplo 1 – utilizando mip-mapping em OpenGL

```
/* Copyright (c) Mark J. Kilgard, 1994. */
/*
 * (c) Copyright 1993, Silicon Graphics, Inc.
 * ALL RIGHTS RESERVED
 * Permission to use, copy, modify, and distribute this software for
 * any purpose and without fee is hereby granted, provided that the above
 * copyright notice appear in all copies and that both the copyright notice
 * and this permission notice appear in supporting documentation, and that
 * the name of Silicon Graphics, Inc. not be used in advertising
 * or publicity pertaining to distribution of the software without specific,
 * written prior permission.
 *
 * THE MATERIAL EMBODIED ON THIS SOFTWARE IS PROVIDED TO YOU "AS-IS"
 * AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE,
 * INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR
 * FITNESS FOR A PARTICULAR PURPOSE.  IN NO EVENT SHALL SILICON
 * GRAPHICS, INC.  BE LIABLE TO YOU OR ANYONE ELSE FOR ANY DIRECT,
 * SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY
 * KIND, OR ANY DAMAGES WHATSOEVER, INCLUDING WITHOUT LIMITATION,
 * LOSS OF PROFIT, LOSS OF USE, SAVINGS OR REVENUE, OR THE CLAIMS OF
 * THIRD PARTIES, WHETHER OR NOT SILICON GRAPHICS, INC.  HAS BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH LOSS, HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE
 * POSSESSION, USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 * US Government Users Restricted Rights
 * Use, duplication, or disclosure by the Government is subject to
 * restrictions set forth in FAR 52.227.19(c)(2) or subparagraph
 * (c)(1)(ii) of the Rights in Technical Data and Computer Software
```

```
 * clause at DFARS 252.227-7013 and/or in similar or successor
 * clauses in the FAR or the DOD or NASA FAR Supplement.
 * Unpublished-- rights reserved under the copyright laws of the
 * United States.  Contractor/manufacturer is Silicon Graphics,
 * Inc., 2011 N.  Shoreline Blvd., Mountain View, CA 94039-7311.
 *
 * OpenGL(TM) is a trademark of Silicon Graphics, Inc.
 */
/*  mipmap.c
 *  This program demonstrates using mipmaps for texture maps.
 *  To overtly show the effect of mipmaps, each mipmap reduction
 *  level has a solidly colored, contrasting texture image.
 *  Thus, the quadrilateral which is drawn is drawn with several
 *  different colors.
 */
#include <stdlib.h>
#include <GL/glut.h>

GLubyte mipmapImage32[32][32][3];
GLubyte mipmapImage16[16][16][3];
GLubyte mipmapImage8[8][8][3];
GLubyte mipmapImage4[4][4][3];
GLubyte mipmapImage2[2][2][3];
GLubyte mipmapImage1[1][1][3];

void makeImages(void)
{
   int i, j;

   for (i = 0; i < 32; i++) {
         for (j = 0; j < 32; j++) {
            mipmapImage32[i][j][0] = 255;
            mipmapImage32[i][j][1] = 255;
            mipmapImage32[i][j][2] = 0;
         }
```

```
    }
    for (i = 0; i < 16; i++) {
            for (j = 0; j < 16; j++) {
                mipmapImage16[i][j][0] = 255;
                mipmapImage16[i][j][1] = 0;
                mipmapImage16[i][j][2] = 255;
            }
    }
    for (i = 0; i < 8; i++) {
            for (j = 0; j < 8; j++) {
                mipmapImage8[i][j][0] = 255;
                mipmapImage8[i][j][1] = 0;
                mipmapImage8[i][j][2] = 0;
            }
    }
    for (i = 0; i < 4; i++) {
            for (j = 0; j < 4; j++) {
                mipmapImage4[i][j][0] = 0;
                mipmapImage4[i][j][1] = 255;
                mipmapImage4[i][j][2] = 0;
            }
    }
    for (i = 0; i < 2; i++) {
            for (j = 0; j < 2; j++) {
                mipmapImage2[i][j][0] = 0;
                mipmapImage2[i][j][1] = 0;
                mipmapImage2[i][j][2] = 255;
            }
    }
    mipmapImage1[0][0][0] = 255;
    mipmapImage1[0][0][1] = 255;
    mipmapImage1[0][0][2] = 255;
}

void myinit(void)
```

```
{
   glEnable(GL_DEPTH_TEST);
   glDepthFunc(GL_LESS);
   glShadeModel(GL_FLAT);

   glTranslatef(0.0, 0.0, -3.6);
   makeImages();
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
   glTexImage2D(GL_TEXTURE_2D, 0, 3, 32, 32, 0,
                GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage32[0][0][0]);
   glTexImage2D(GL_TEXTURE_2D, 1, 3, 16, 16, 0,
                GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage16[0][0][0]);
   glTexImage2D(GL_TEXTURE_2D, 2, 3, 8, 8, 0,
                GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage8[0][0][0]);
   glTexImage2D(GL_TEXTURE_2D, 3, 3, 4, 4, 0,
                GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage4[0][0][0]);
   glTexImage2D(GL_TEXTURE_2D, 4, 3, 2, 2, 0,
                GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage2[0][0][0]);
   glTexImage2D(GL_TEXTURE_2D, 5, 3, 1, 1, 0,
                GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage1[0][0][0]);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
         GL_NEAREST_MIPMAP_NEAREST);
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
   glEnable(GL_TEXTURE_2D);
}

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glBegin(GL_QUADS);
   glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
   glTexCoord2f(0.0, 8.0); glVertex3f(-2.0, 1.0, 0.0);
```

```
    glTexCoord2f(8.0, 8.0); glVertex3f(2000.0, 1.0, -6000.0);
    glTexCoord2f(8.0, 0.0); glVertex3f(2000.0, -1.0, -6000.0);
    glEnd();
    glFlush();
}

void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, 1.0*(GLfloat)w/(GLfloat)h, 1.0, 30000.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

static void
key(unsigned char k, int x, int y)
{
  switch (k) {
  case 27:  /* Escape */
    exit(0);
    break;
  default:
    return;
  }
  glutPostRedisplay();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize (500, 500);
    glutCreateWindow (argv[0]);
```

```
    myinit();
    glutReshapeFunc (myReshape);
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    glutMainLoop();
    return 0;           /* ANSI C requires main to return int. */
}
```

Bibliografia

1. AZEVEDO, Eduardo, CONCI, Aura. *Computação gráfica - teoria e prática*. Rio de Janeiro: Elsevier, 2003.

2. GOMES, Jonas, VELHO, Luiz. *Computação gráfica: imagem*. Rio de Janeiro: IMPA/SBM, 1994.

3. Foley, James D., ... [et al.]. *Computer graphics – principles and practice. 2nd ed. In C*, Addison Wesley Publishing Company,1996.

4. Cavalcanti, Paulo Roma. Apostila: *Introdução à computação gráfica I*. Rio de Janeiro: UFRJ, 2000.

5. CATMULL, E., ROM, R. *A class of local interpolating splines*. In Barnhill, R and R. Riesenfeld, eds., *Computer Aided Geometric Design*, Academic Press, San Francisco, 1974, 317-326.

6. BLINN, J. F., NEWELL, M. E. *Texture and reflection in computer generated images*. CACM, 19(10), October 1976, 542-547.

7. Esperança, Cláudio, Cavalcanti, Paulo Roma. *Introdução à computação gráfica – texturas*. Disponível em: Site ???? texture2.pdf

8. LUEBKE, David. Site: http://www.cs.virginia.edu/~gfx/Courses/2003/ Intro.spring.03/lecture27.ppt. Acessado em: junho/2004.

9. WILLIAMS, L. *Pyramidal Parametrics*. SYSGRAPH 83, 1-11.

10. CROW, F. C. *Summed-area tables for texture mapping*. SIGGRAPH 84, 207-212

11. Glassner, A. S. *Adaptive precision in texture mapping*. SIGGRAPH 86, 297-306

12. Heckbert, P.S. *Filtering by repeate integration*. SIGGRAPH 86, 315-321.

13. http://www.relisoft.com/Science/Graphics/mip.html. Acessado em: junho/2004.

14. http://escience.anu.edu.au/lecture/cg/Texture/MIPmapping1.en.html. Acessado em: junho/2004.

15. ROCHA, Lorena K. de M. Disponível em: http:// w3.impa.br/~lvelho/i3d01/ demos/lourena/Introducao.htm#A. Acessado em: junho/2004.

16. CONCI, Aura. Disponível em: http://www.ic.uff.br/~aconci/CG.html. Acessado em: junho/2004.

17. SOBRINHO, Marcionílio Barbosa. Tutorial de OpenGl. Disponível em: http://www.ingleza.com.br/opengl/9-1-2.html. Acessado em: junho/2004.