IA725 – Computação Gráfica I

Mapeamento de textura

29/05/2008

Shirley, capítulo 11

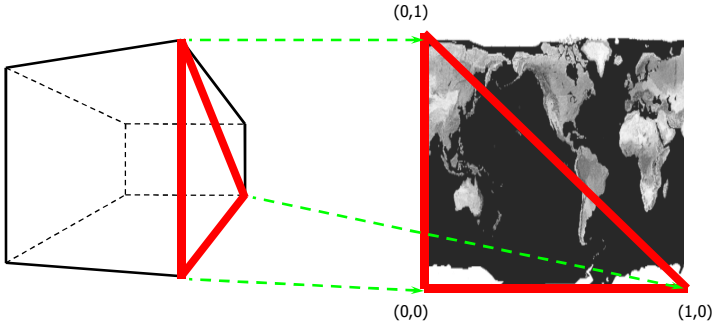
Red Book, capítulo 9

Visão geral

- Texturização com perspectiva correta.
- Mapeamento de textura em triângulos rasterizados.
- *Aliasing* e *mipmapping*.
- Efeitos especiais com mapeamento de textura:
  - *Bump mapping* (mapeamento de rugosidade)
  - *Displacement mapping* (mapeamento de deslocamento).
  - *Environment mapping* (mapeamento de ambiente).
  - *Shadow mapping* (mapeamento de sombra).
- Utilizando texturas em OpenGL.

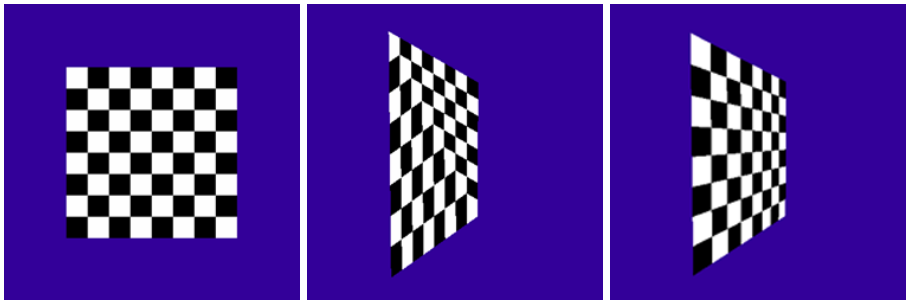
Interpolação e perspectiva

- Mapeamento de textura em malhas triangulares:
  - Interpolação das coordenadas baricêntricas define coordenadas *uv* no interior do triângulo.
  - Sob transformação perspectiva, as coordenadas uv podem ser interpoladas linearmente no espaço da tela?



Interpolação e perspectiva

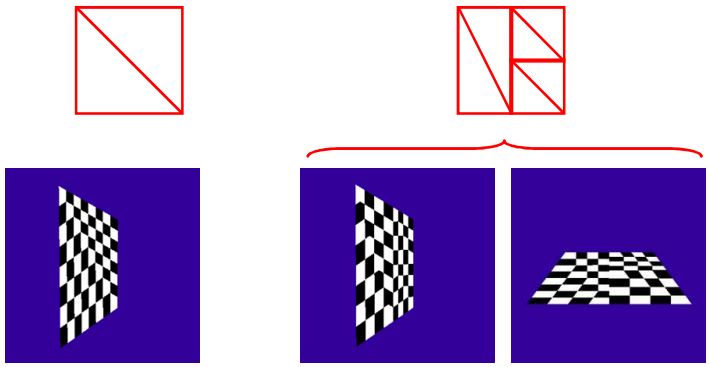
- Interpolação linear no espaço da tela:



textura fonte | o que obtemos | o que queremos

Interpolação e perspectiva

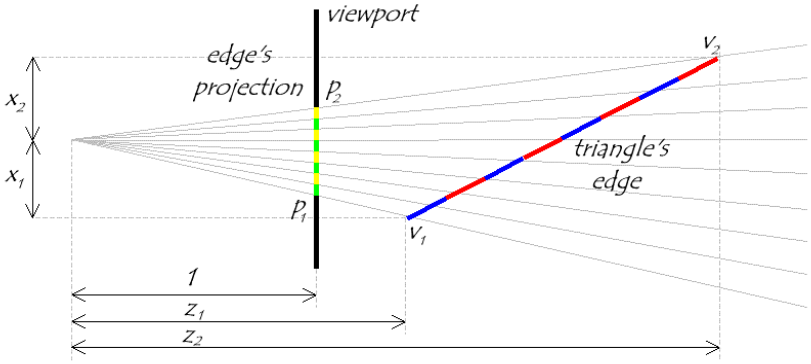
- Problema pode ser reduzido se a malha for subdivida:



- Requer muitas subdivisões para obter bons resultados.

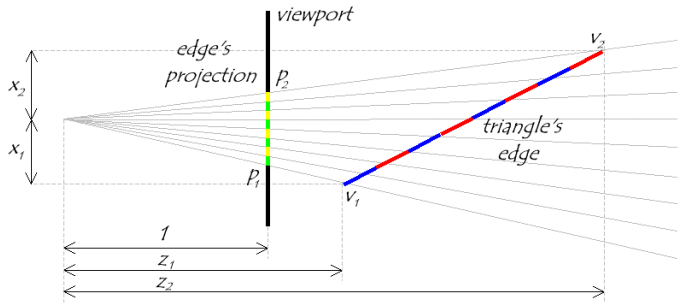
Interpolação e perspectiva

- Entendendo o problema:



- Passos uniformes no plano da imagem não correspondem a passos uniformes ao longo da aresta.

Interpolação e perspectiva



interpolação linear no espaço da tela

$$p(t) = p_1 + t(p_2 - p_1) = \frac{x_1}{z_1} + t\left(\frac{x_2}{z_2} - \frac{x_1}{z_1}\right)$$

interpolação linear em 3D

$$\begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} x_1 \\ z_1 \end{bmatrix} + s\left(\begin{bmatrix} x_2 \\ z_2 \end{bmatrix} - \begin{bmatrix} x_1 \\ z_1 \end{bmatrix}\right) \qquad P\left(\begin{bmatrix} x \\ z \end{bmatrix}\right) = \frac{x_1 + s(x_2 - x_1)}{z_1 + s(z_2 - z_1)}$$

Interpolação e perspectiva

Precisamos de um mapeamento entre valores de *t* e valores de *s*:

$$\frac{x_1}{z_1} + t\left(\frac{x_2}{z_2} - \frac{x_1}{z_1}\right) = \frac{x_1 + s(x_2 - x_1)}{z_1 + s(z_2 - z_1)}$$

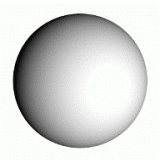
Resolvemos para *s* em termos de *t*:

$$s = \frac{t\, z_1}{z_2 + t\,(z_1 - z_2)}$$

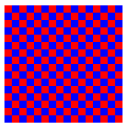
Textura e iluminação

- Mapeamento de textura pode ser utilizado para alterar quaisquer coeficientes de reflectância do modelo de iluminação:

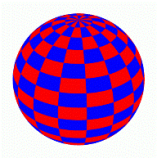
$$I_{total} = \underbrace{k_a\, I_a}_{\text{ambiente}} + \sum_{j=1}^{\text{fontes de luz}} [(f_{att_j}\, I_{i_j})(\underbrace{k_d\, (\vec{N}\bullet\vec{L})}_{\text{difusa}} + \underbrace{k_s\, (\vec{N}\bullet\vec{H})^n}_{\text{especular}})]$$



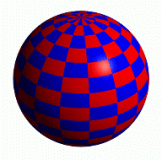
Cor difusa constante



Cor difusa da textura



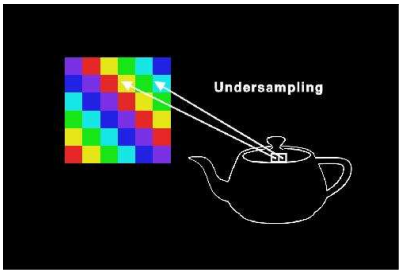
Textura como cor ambiente

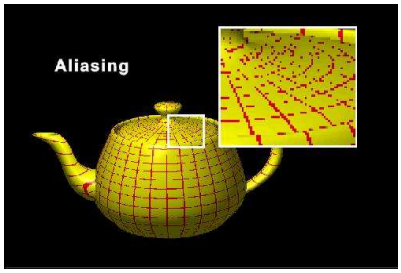


Textura como cor difusa

Aliasing

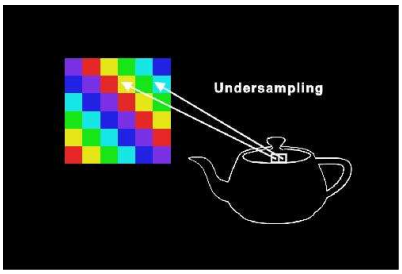
- Resultado da subamostragem da textura.
  - Em mapeamento de textura, *aliasing* ocorre quando dois *pixels* adjacentes do objeto são mapeados em *texels* diferentes, não-adjacentes, do mapa de textura.

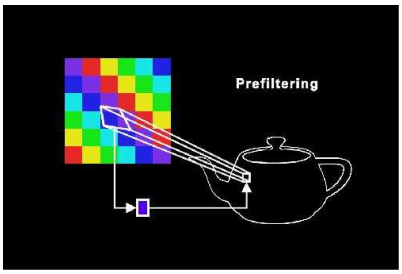




Aliasing

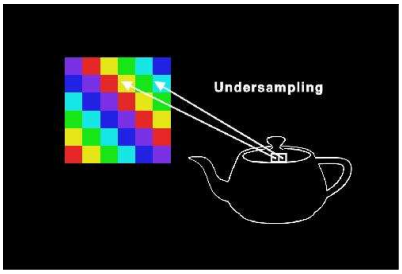
- Métodos de redução de *aliasing* (*antialiasing*):
  - Pré-filtragem (Catmull, 1978): Cada *pixel* do objeto é tratado como uma área. A área do *pixel* é mapeada na textura. A cor média é calculada para os *texels* contidos nesta área.

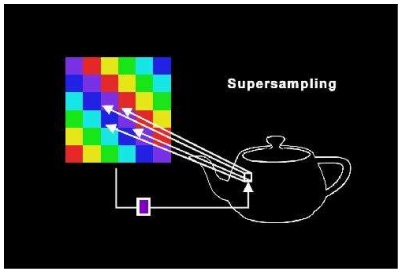




Aliasing

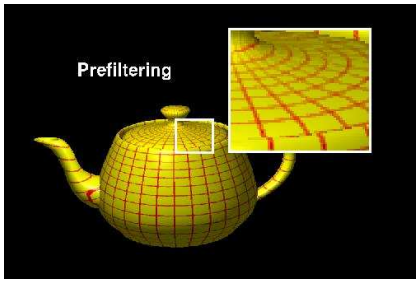
- Métodos de redução de *aliasing* (*antialiasing*):
  - Superamostragem (Crow, 1981): Também calcula uma cor média. No exemplo abaixo, cada canto da área do pixel é mapeado na textura. A média dos valores obtidos produz a cor final do objeto.

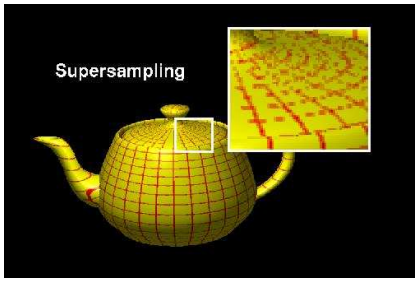




Aliasing

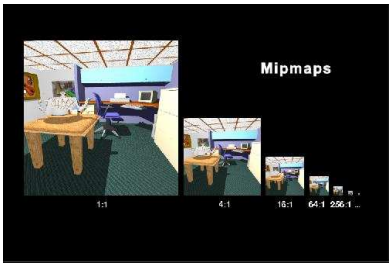
- Exemplos de redução de *aliasing*:

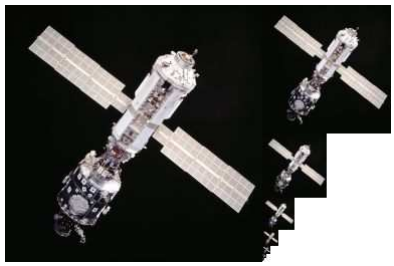




Mipmapping

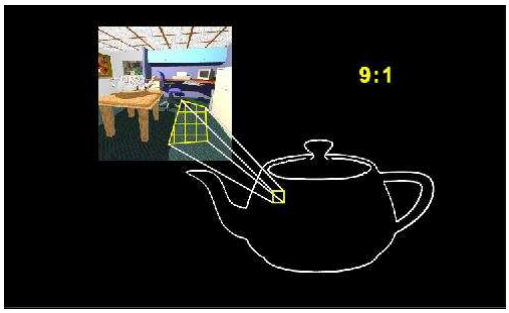
- MIP (*Multum In Parvo* – muito em pouco espaço).
- Método de aceleração do cálculo da cor média para áreas de amostragem da textura (Williams, 1983).
  - Várias versões da textura são criadas (*mipmaps*).
  - Cada *texel* de uma versão contém a cor média de 4 *texels* da versão anterior.





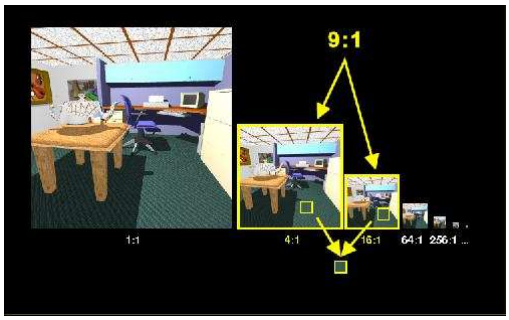
Mipmapping

- Na texturização, a área de cada pixel do objeto é mapeada no mapa de textura original.
- Cada *mipmap* é associado a uma medida de quantos *texels* da textura original estão na área do *texel* do mipmap. No exemplo abaixo, a razão desses *texels* é de 9:1.



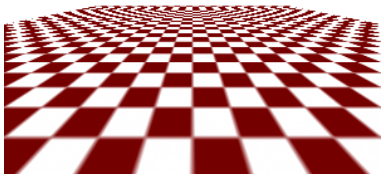
Mipmapping

- Para calcular a cor final, encontramos os dois mapas de textura cujas razões de *texels* é a mais próxima da razão do *pixel* atual.
- A cor resultante é a média das cores dos *pixels* amostrados nos dois mapas.

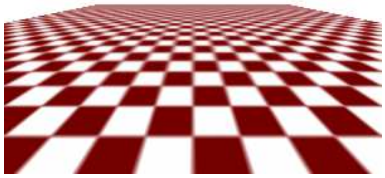


Mipmapping

- Exemplo de superamostragem, com e sem *mipmapping*.



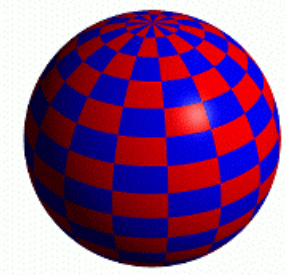
Sem *mipmapping*



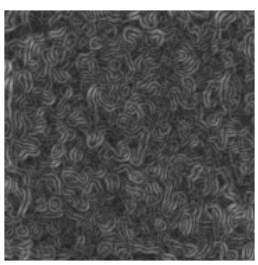
Com *mipmapping*

Bump mapping

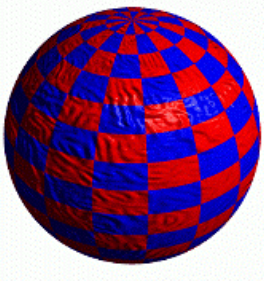
- Utiliza texturas para perturbar a direção do vetor normal de cada ponto da superfície (Blinn, 1978).
  - Não modifica a forma da superfície.
  - Modelo de iluminação usa o vetor normal modificado.



Esfera com textura difusa

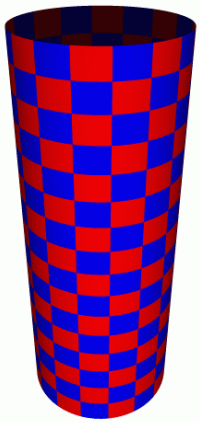


Bump map

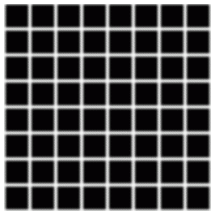


Esfera com textura difusa
e *bump mapping*

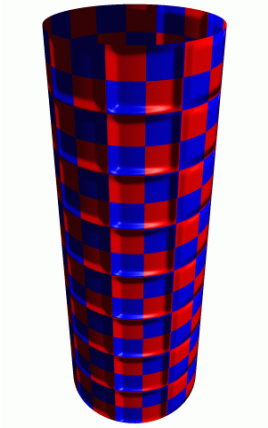
Bump mapping



Cilindro com textura difusa



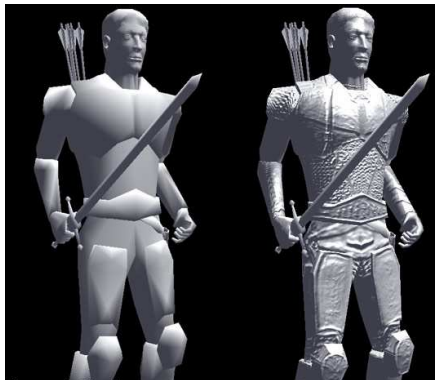
Bump map



Cilindro com textura difusa e *bump mapping*

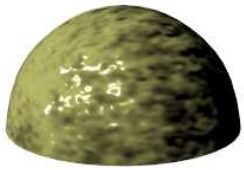
Bump mapping

- Simula detalhes na superfície sem a necessidade de criar geometria.
- Por outro lado:
  - Não produz silhuetas corretas.
  - Não simula oclusão entre os detalhes.
  - Não simula sombras entre os detalhes.



Bump mapping

- Simula detalhes na superfície sem a necessidade de criar geometria.
- Por outro lado:
  - Não produz silhuetas corretas.
  - Não simula oclusão entre os detalhes.
  - Não simula sombras entre os detalhes.

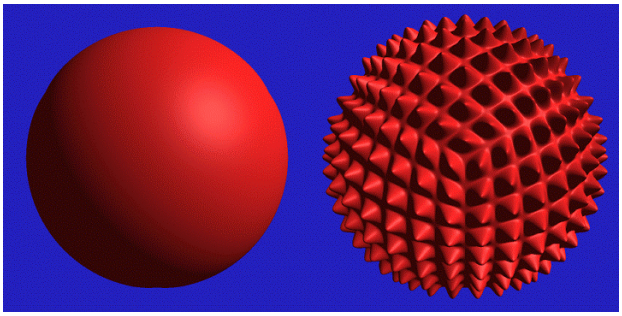


Bump mapping

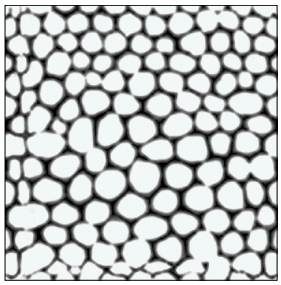
Resultado desejado

Displacement mapping

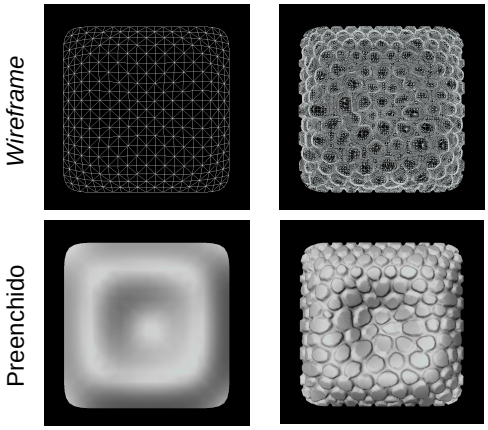
- Semelhante ao *bump mapping*, porém modifica a geometria.
  - Cada *texel* do *displacement map* é um valor de deslocamento do vértice ao longo do vetor normal.



Displacement mapping

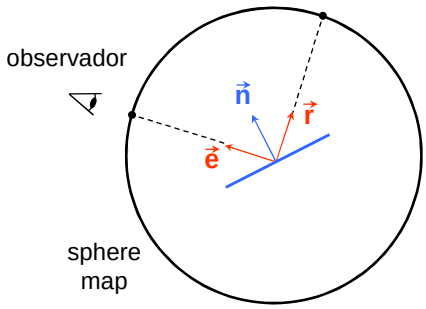


Displacement map



Environment mapping

- Técnica simples e eficaz de simular reflexos produzidos por superfícies espelhadas. Requer a posição da câmera.
- *Sphere mapping*:
  - Similar ao mapeamento esférico.



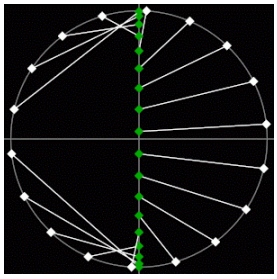
$$\vec{r} = 2\,(\vec{e} \bullet \vec{n})\,\vec{n} - \vec{e}$$

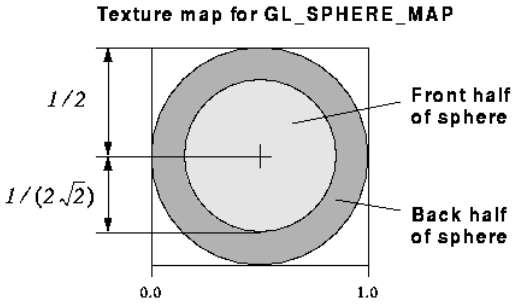
$$u = \frac{\operatorname{atan}(\vec{r}_y / \vec{r}_x) + \pi}{2\,\pi}$$

$$v = \frac{\pi - \operatorname{acos}(\vec{r}_z)}{\pi}$$

Environment mapping

- *Sphere mapping* no OpenGL.





$m = 2\ \mathrm{sqrt}(\vec{r}_x^{\,2} + \vec{r}_y^{\,2} + (\vec{r}_z + 1)^2)$

$u = \vec{r}_x/m + 0.5$

$v = \vec{r}_y/m + 0.5$

Environment mapping

- *Sphere mapping* no OpenGL.

Mapa de textura

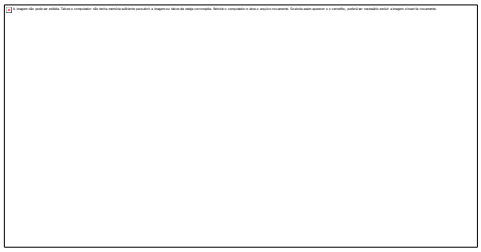
Modelo texturizado

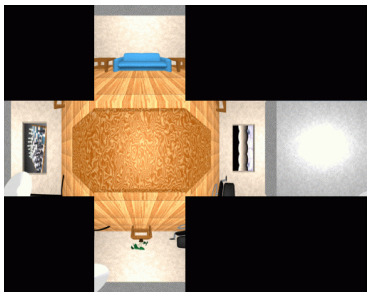
Environment mapping

- *Sphere mapping* no OpenGL.
  - Ligue a textura contendo o *sphere map*.
  - Habilite o estado de geração de coordenadas de textura para *sphere mapping*:
    - `glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);`
    - `glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);`
  - Habilite a texturização e geração de coordenadas de textura.
    - `glEnable(GL_TEXTURE_GEN_S);`
    - `glEnable(GL_TEXTURE_GEN_T);`
    - `glEnable(GL_TEXTURE_2D);`
  - Desenhe o objeto.

Environment mapping

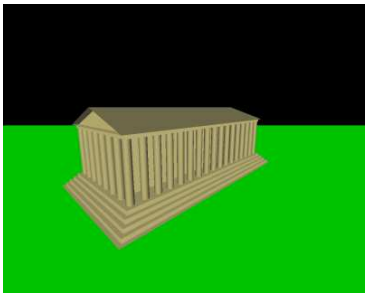
- *Cube mapping*.
  - Utiliza o mapeamento cúbico para simular reflexão.
  - O *cube map* pode ser facilmente gerado em tempo real.



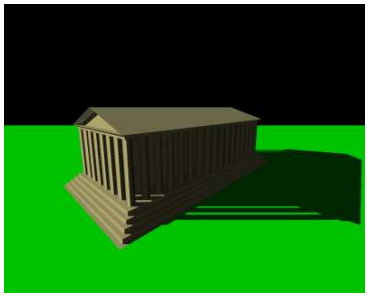


Shadow mapping

- Sombras são obtidas verificando se um *pixel* está sendo visto pela fonte de luz através de um teste com o *z-buffer* do ponto de vista da fonte de luz (Williams, 1978).



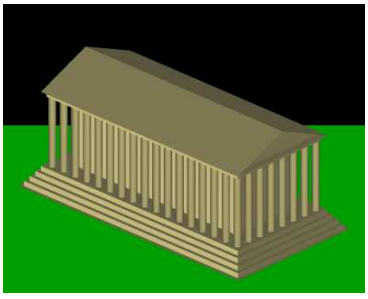
Cena sem sombras



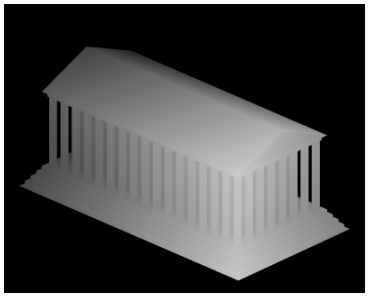
Cena com *shadow mapping*

Shadow mapping

- Primeiro passo: Visualizar a cena do ponto de vista da fonte de luz e armazenar o *z-buffer* correspondente em uma textura. Essa textura é o *shadow map*.



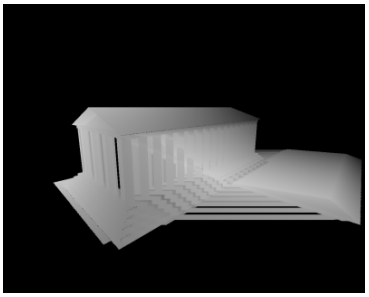
Cena do ponto de vista da fonte de luz



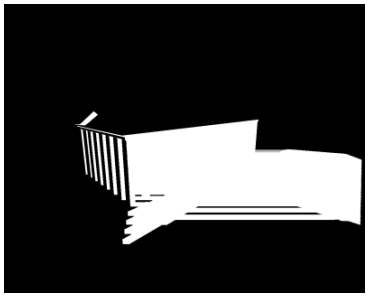
Z-buffer correspondente (*shadow map*)

Shadow mapping

- Segundo passo: Desenhar a cena do ponto de vista atual e comparar o valor z de cada ponto desenhado com o valor z correspondente no *shadow map*. Se o valor z atual é maior que o valor z do *shadow map*, então está na sombra.



Z-buffer projetado na cena



Onde o teste de profundidade falhou

Referências da aula prática

- http://jerome.jouvie.free.fr/OpenGl/Tutorials/Tutorial11.php (tutorial sobre *sphere mapping* e *cube mapping* no OpenGL).
- http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=23 (tutorial sobre *sphere mapping* em C com OpenGL).
- http://www.paulsprojects.net/tutorials/simplebump/simplebump.html (tutorial sobre *normal mapping* no OpenGL).
- http://www.paulsprojects.net/tutorials/smt/smt.html (tutorial sobre *shadow mapping* no OpenGL).