

Realismo: como implementar detalhes?

- Até agora vimos como iluminação e sombreamento podem produzir objetos mais realistas
 - Modelo de Phong
 - Essa técnica é boa para objetos lisos com cor uniforme.
 - um copo de plástico
 - uma tampa de metal
- Foley: Chapters 16.3 e 17.4
- Mount: Lectures 9 e 10

Mapeamento de Texturas

O que fazer para superfícies mais complexas?

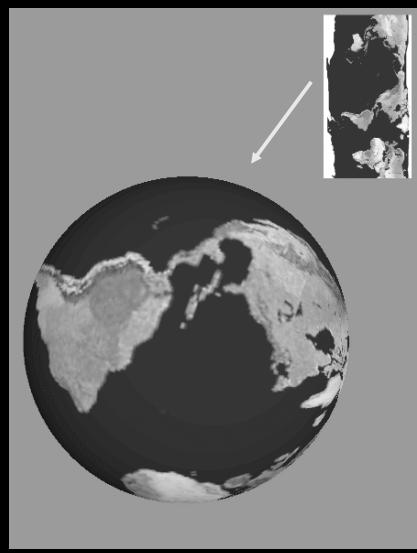
- Podemos aumentar o nível de detalhes da superfície
 - ou definir uma função matemática
 - como gradiente ou quadriculado

Mas essas alternativas podem ser inviáveis para superfícies imprevisíveis e muito complexas

Exemplo: como modelar a ...



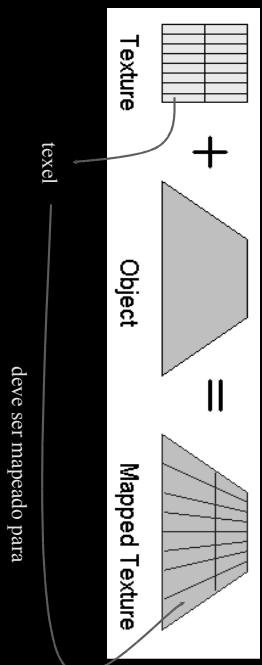
Exemplo: mapa para esfera



Mapeamento de Texturas

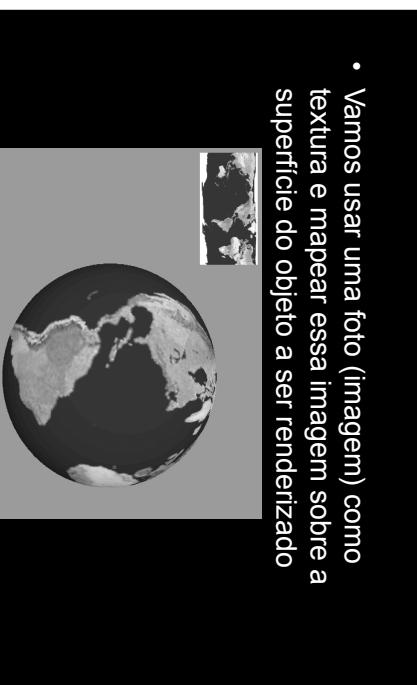
- Processo de mapear (ou embrulhar) uma imagem ao redor de um objeto.
 - exemplo: a partir de uma foto de um gramado, digitalizar a foto em algum formato e mapear essa imagem sobre um retângulo.
- Representação de imagens:
 - formatos: gif, jpeg, tiff, bmp, ppm, etc.
 - OpenGL: imagem é uma matriz NxN de pixels com valores de RGB (N é potência de 2).
- **Texels:** texture elements (ao invés de pixels)

Mapeamento de texturas



Ideia: mapeamento de textura

- Vamos usar uma foto (imagem) como textura e mapear essa imagem sobre a superfície do objeto a ser renderizado

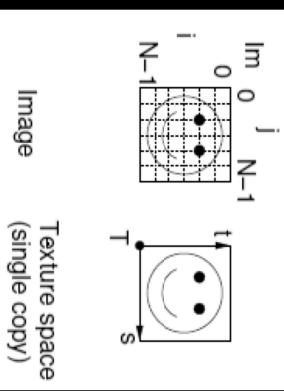


Espaço de texturas

- Ao invés de pensar que a imagem é uma matriz 2D, podemos considerá-la como sendo uma função que mapeia um ponto (s,t) no espaço de textura para um valor RGB.

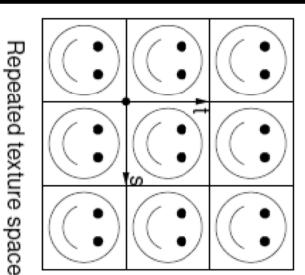
- Dado um par (s,t) , $0 \leq s,t \leq 1$, a textura define um valor $T(s,t)$, que é um valor de RGB (uma cor).

Espaço de textura: cópia única



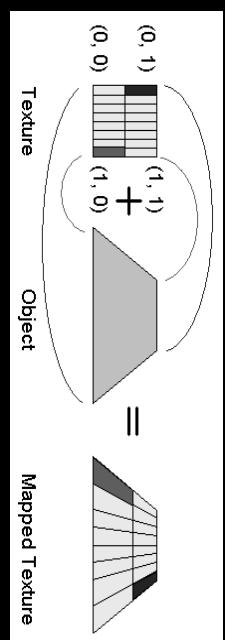
$$T(s, t) = Im[(1-t)N, \lfloor sN \rfloor], \quad \text{for } s, t \in (0, 1)$$

Textura: múltiplas cópias



$$T(s, t) = Im[(1-t)N] \bmod N, \lfloor sN \rfloor \bmod N, \quad \text{for } s, t \in \mathbb{R}.$$

Textura como função 2d



Vamos ignorar o formato e considerar uma função:

$$T(s, t) \rightarrow \text{RGB}$$

$$0 \leq s, t \leq 1$$

Como embrulhar a textura no objeto?

- devemos definir uma função de mapeamento
- Mas o objeto é 3D -> um ponto 2D (s, t) precisa ser mapeado para um ponto 3D (x, y, z)
- Isso é feito por meio da parametrização da superfície do objeto

Exemplo: esfera

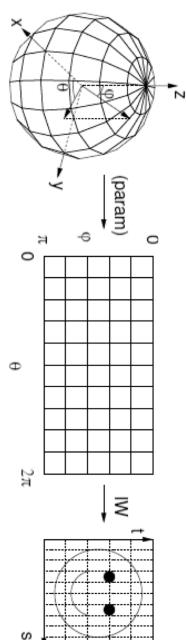
- Um ponto sobre a esfera de raio unitário pode ser representado por 2 ângulos θ e φ .

- coordenadas esféricas
- $0 \leq \varphi \leq \pi$ e $0 \leq \theta \leq 2\pi$

Qual a coordenada (x, y, z) em função de (θ, φ) ?

Parametrizando uma esfera

$$z(\phi, \theta) = \cos \phi, \quad x(\phi, \theta) = \cos \theta \sin \phi, \quad y(\phi, \theta) = \sin \theta \sin \phi.$$



Poderíamos ter: $u = 1 - \varphi/\pi$ e $v = \theta/(2\pi)$

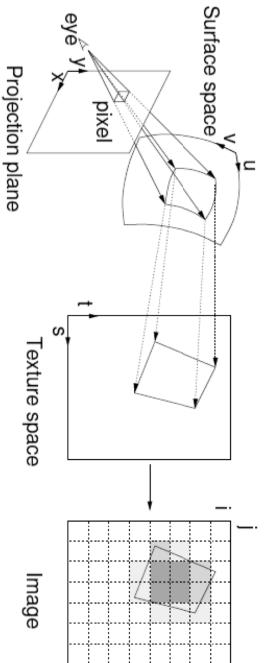
Parametrizações

- Desejamos uma função que mapeie um ponto (s, t) da textura para um ponto (x, y, z) na superfície do objeto.
- Isso é tipicamente implementado através de uma função paramétrica
- cada ponto da superfície do objeto é associado a uma coordenada (u, v) local a superfície.
- obtemos assim 3 funções que mapeiam (u, v) para (x, y, z) :
- $x(u, v), y(u, v)$ e $z(u, v)$.
- uma vez obtida uma função paramétrica (u, v)
- mapeamos cada (u, v) para um texel (s, t)

Mapeamento inverso

- Em geral é mais útil considerar o mapeamento inverso, da textura para a superfície
 - Seja $IW(u, v) = \text{Inverse Wrapping function}$ que:
 - mapeia um ponto (u, v) para um ponto (s, t)
 - A partir das funções $x(u, v)$, $y(u, v)$, e $z(u, v)$ sabemos que:
 - $\varphi = \arccos(z)$ e $\theta = \arctan(y/x)$
 - Normalizando para $(0, 1)$:
 - $u = 1 - \arccos(z)/\pi$ e $v = \arctan(y/x)/(2\pi)$
 - podemos usar (u, v) diretamente como (s, t)

O processo de mapeamento



O processo de mapeamento

- Suponha que conhecemos $IW(u, v)$ e a parametrização. O mapeamento se faz da seguinte forma:
- projetar os cantos do pixel a ser renderizado para a superfície do objeto, pontos P_i
 - calcular os parâmetros (u, v) p/ cada P_i
 - Desenbrulhar: aplicar o mapamento inverso dos P_i 's
 - Tirar uma média local dos texels que "cobrem" o pixel

Textura no OpenGL

- O OpenGL modela os objetos por polígonos
 - isso simplifica o processo de mapeamento.
 - exemplo: suponha um triângulo. Quando os vértices são especificados, o usuário também define os pontos (s, t) no espaço de textura, o que define um mapeamento linear entre a textura e a superfície.

```
glBegin(GL_POLYGON)
glNormal3f(nx, ny, nz)
glTexCoord2f(tx, ty)
 glVertex3f(vx, vy, wz)
 ...
glEnd();
```

Observe que a textura é definida antes do vértice

Como tratar a textura: Quick and Dirty

Como interpolar as coordenadas da textura no interior do polígono?

- Implementação simples (rápida e eficiente):
 - Sejam P_0, P_1, P_2 os vértices de um triângulo
 - Sejam Q_0, Q_1, Q_2 as 3 coordenadas de textura correspondentes

Implementação

- Um pixel P do triângulo pode ser definido por:
$$P = \alpha_0 P_0 + \alpha_1 P_1 + \alpha_2 P_2$$
 onde $\alpha_0 + \alpha_1 + \alpha_2 = 1$
- Uma vez calculados os coeficientes α , para determinar a textura de P basta calcular:
$$Q = \alpha_0.Q_0 + \alpha_1.Q_1 + \alpha_2.Q_2$$

Problemas

- Aliasing
 - o método não calcula a média das cores em uma região
 - quando um pixel cobre uma região da textura, esse método não produz bons resultados
- perspectiva
 - combinações afins não são preservadas sob projeção perspectiva
 - projeção -> wrapping != wrapping -> projeção

Quick and Dirty

Forma correta

Problemas

- Não considera perspectiva

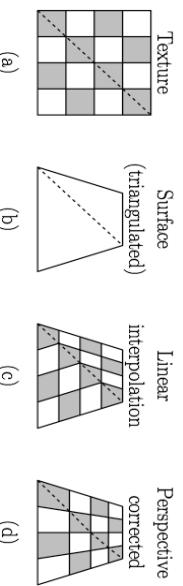


Fig. 49: Perspective correction.

```
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST)
```

Problemas

- Aliasing

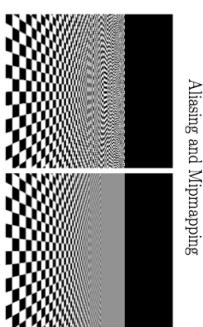


Fig. 50: Aliasing and mipmaping.

Mipmapping

- Cria uma série de texturas, com resoluções menores

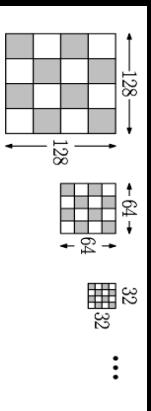


Fig. 51: Mipmapping.

Texture mapping in OpenGL

```
glEnable(GL_TEXTURE_2D)  
  
# gera n texturas, em geral de 1 a n  
texID = glGenTextures(n)  
# torna a textura texID ativa  
 glBindTexture(GL_TEXTURE_2D, texID)
```

Problemas

- Não considera perspectiva

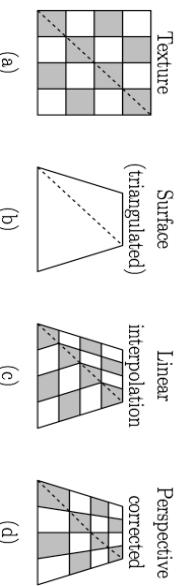


Fig. 49: Perspective correction.

```
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST)
```

Problemas

- Aliasing

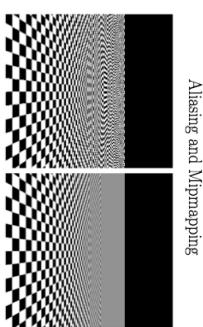


Fig. 50: Aliasing and mipmaping.

Mipmapping

- Cria uma série de texturas, com resoluções menores

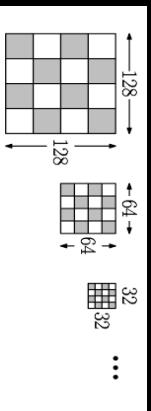


Fig. 51: Mipmapping.

Texture mapping in OpenGL

```
glEnable(GL_TEXTURE_2D)  
  
# gera n texturas, em geral de 1 a n  
texID = glGenTextures(n)  
# torna a textura texID ativa  
 glBindTexture(GL_TEXTURE_2D, texID)
```

Gerando a textura

```
glTexImage2d(GL_TEXTURE_2D, level,  
internalFormat, width, height, border, format, type,  
image)
```

level – mipmap (em geral 0)
format – GL_RGB ou GL_RGBA
width, height: dimensão
border = 0
type = GL_UNSIGNED_BYTE

Veja os exemplos

Filtro e mipmapping

```
par_name = GL_TEXTURE_MAG_FILTER  
par_name = GL_TEXTURE_MIN_FILTER  
  
par_value = GL_NEAREST ou GL_LINEAR
```

Textura e iluminação

- A cor pode ser combinada com a textura

```
glTexEnvf(GL_TEXTURE_ENV,  
GL_TEXTURE_ENV_MODE, GL_MODULATE)
```

- default: multiplica as cores

```
glTexEnvf(GL_TEXTURE_ENV,  
GL_TEXTURE_ENV_MODE, GL_REPLACE)
```

- a cor do pixel é a cor da textura

Outras opções

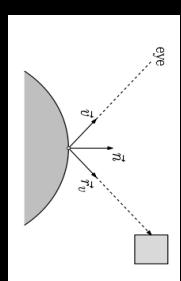
```
glTexParameteri(target, par_name, par_value)  
glTexParameterf(target, par_name, par_value)
```

- O que fazer quando s (ou t) é < 0 ou > 1?

```
target = GL_TEXTURE_2D  
par_name = GL_TEXTURE_WRAP_S  
par_value = GL_CLAMP ou GL_REPEAT
```

Mapeamento da cena

- Em superfícies altamente polidas, é comum observarmos outros objetos da cena sendo refletidos.
- o modelo de Phong, por ser local, não modela esse fenômeno
- pode ser gerado de forma precisa através de "ray-tracing"
 - pode ser feita de forma rápida (porém não muito precisa) através de uma técnica chamada de *environment mapping* (EM) ou *reflection mapping*.



Exemplo

- Chaleira na cozinha
 - A chaleira é pequena em relação as paredes. A mesa e os objetos da mesa no entanto estão muito perto e...



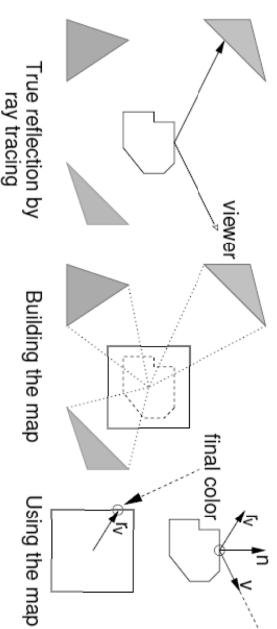
Reflexão x Textura

- Quando observamos uma superfície com textura, e movemos a cabeça, a textura permanece constante sobre a superfície.
- Porém, reflexões dependem da posição e orientação da superfície do objeto e do observador, e portanto os padrões de reflexões variam com o movimento da cabeça.
- Em que condições podemos aproximar reflexões como padrões de textura?

Processo de mapeamento

- 1) criação do mapa do ambiente:
 - coloque uma pequena esfera ou cubo ao redor do objeto
 - assuma que não há intersecção com outros objetos da cena
 - projete a cena sobre as faces desse objeto (6 faces do cubo), usando o seu centro como centro de projeção.
 - todas as imagens são armazenadas para permitir reflexões de qualquer posição.
 - Obs: Precisão pode não ser necessária
 - cromados
 - espelhos precisam de mais precisão

Environment mapping



Processo de mapeamento

- 2) cálculo do vetor de reflexão r , a partir da normal n e do vetor de observação v
 - de forma semelhante ao cálculo utilizado no método de Phong.
 - usando esse vetor, podemos traçar um raio do centro do cubo e determinar o ponto sobre o cubo que é interceptado pelo raio. Esse ponto determina a cor a ser desenhada.

Bump Mapping

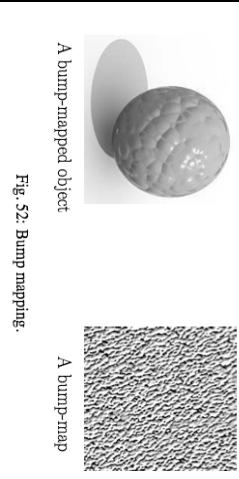


Fig. 32: Bump mapping.

Uma imagem pode não ser suficiente para dar ideia de textura devido a rugosidades da superfície

Rugosidades

- Rugosidades podem ser muito pequenas para se notar à grande distância
- Porém elas afetam a normal da superfície
- *Bump mapping* procura reproduzir esse efeito

Espaço de rugosidades

- Similar ao espaço de textura.
- Precisamos definir uma função 2D $b(s,t)$
- $b(s,t) = \text{bump displacement function}$ mapeia um ponto (s,t) para o *bump space* (uma altura h)
- vamos considerar também a função inversa $W(u,v)$, que mapeia um ponto sobre a superfície do objeto para (s,t)

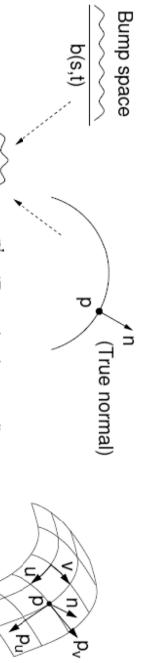
Alternativas

- 1) uma forma seria armazenar um "mapa de normais"
 - necessário utilizar 3 canais: $n = (nx, ny, nz)$
- 2) ou o valor de cada "pixel" representa a altura do "monte".
 - basta utilizar um canal: h
 - mais fácil de armazenar e calcular que um mapa de normais
 - Basicamente, é uma imagem em nível de cinza ou Bump Image

Perturbação dos vetores normais

- Superfície do objeto é representada por uma função paramétrica em (u,v)
 - $x(u,v), y(u,v), z(u,v)$
 - seja P um ponto sobre a superfície em (u,v)
 - n o vetor normal em (u,v)
- Problema: dado n , como calcular o vetor perturbado n' em P ?
 - uma vez conhecido n' , basta utilizar esse valor no modelo de Phong

Descrição do método



(a)

(b)

Fig. 46: The bump-mapping process.

Ideia: "embrulhar" o mapa de rugosidades ao redor da superfície

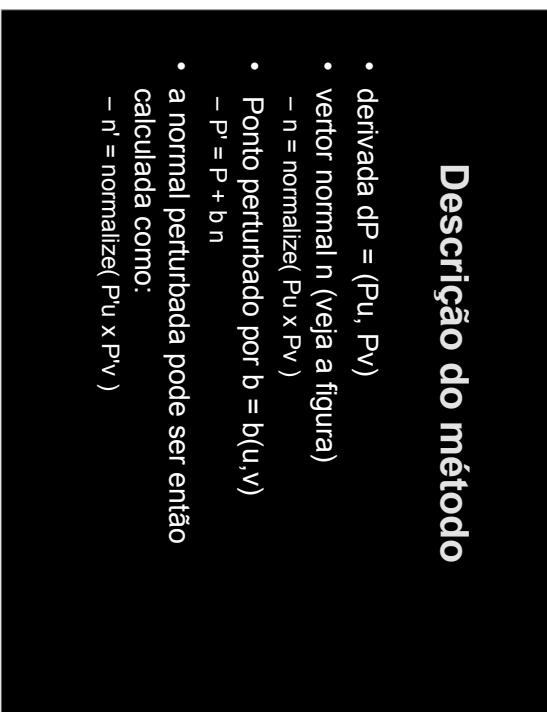
Descrição do método

- P é uma função de (u, v) .
- A derivada em P nos fornece um vetor tangente a P

$$p_u = \begin{pmatrix} \frac{\partial x}{\partial u} \\ \frac{\partial y}{\partial u} \\ \frac{\partial z}{\partial u} \end{pmatrix} \quad p_v = \begin{pmatrix} \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial v} \\ \frac{\partial z}{\partial v} \end{pmatrix}.$$

Vetor n'

- derivada $dP = (P_u, P_v)$
- vetor normal n (veja a figura)
 - $n = \text{normalize}(P_u \times P_v)$
- Ponto perturbado por $b = b(u, v)$
 - $P' = P + b n$
 - a normal perturbada pode ser então calculada como:
 - $n' = \text{normalize}(P'_u \times P'_v)$



Vetor n'

$$\begin{aligned} n' &\sim (P_u + b_u n) \times (P_v + b_v n) \\ &\sim (P_u \times P_v) + b_v (P_u \times n) + b_u (n \times P_v) \\ &\quad + b_u b_v (n \times n) \\ &\quad \xrightarrow{\text{cancelar}} 0 \\ &\quad - (n \times P_u) \end{aligned}$$

e assim temos:

$$n' \sim n + b_s (n \times P_v) - b_t (n \times P_u)$$

Como calcular as derivadas

- A partir das funções de $P(u, v)$ e $b(s, t)$
 - Use cálculo caso tenha as funções
 - Se P estiver sobre um polígono então P_u e P_v são constantes
 - b_s e b_t podem ser por diferenças finitas

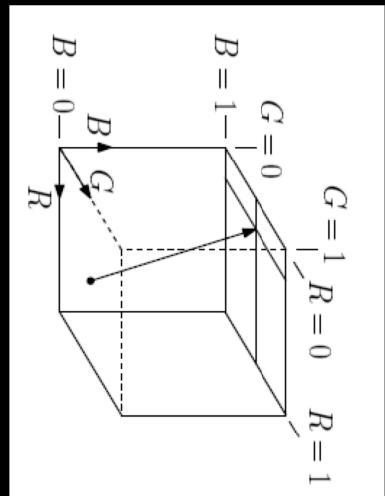
Mapa de normais (normal map)

- As equações diferenciais apresentadas anteriormente podem ser difíceis de serem realizadas pela GPU
- Ao invés de usar apenas a altura, podemos utilizar as 3 componentes RGB de uma imagem colorida para representar a perturbação da normal diretamente.

Método

- Um método possível é utilizar a tripla RGB para representar um vetor da origem ao topo de um cubo unitário
- Cada valor RGB pode ser normalizado para o intervalo $[0, 1]$ (dividindo por 256)
 - B representa $z \Rightarrow B = 1$
 - R representa $x \Rightarrow x = 2R - 1$
 - G representa $y \Rightarrow y = 2G - 1$
 - onde $0 \leq R, G \leq 1$
 - é um vetor normal $(x, y, z) = (2R - 1, 2G - 1, B)$

Normal maps



O que você deve saber

- Mapeamento de texturas
 - superfícies paramétricas
 - espaço de texturas
 - função de mapeamento (embrulho x desembrulho)
- Mapeamento da cena ou de reflexões
 - diferença entre textura e reflexão
- Mapeamento de rugosidades
 - bump maps
 - normal maps

Bump x Normal maps

