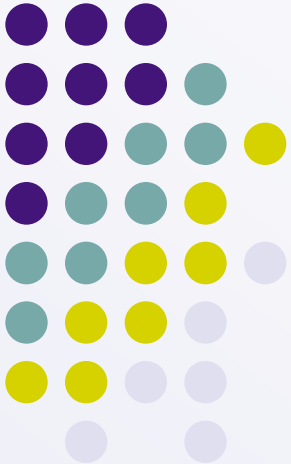
Universidade de São Paulo
Instituto de Matemática e Estatística

Introdução à Computação Gráfica

Marcel P. Jackowski
mjack@ime.usp.br

Aula #9



Objetivos

- Iluminação em OpenGL
- Tonalização de polígonos
  - Flat (faceteada)
  - Smooth (suave ou Gouraud)
  - Phong
- Introdução à mapeamentos
  - Textura
  - Ambiente
  - Bump (“rugosidade”)

Tonalização em OpenGL

1. Abilitar tonalização e selecionar o modelo;
2. Especificar vetores normais;
3. Especificar propriedades dos materiais;
4. Especificar luzes;

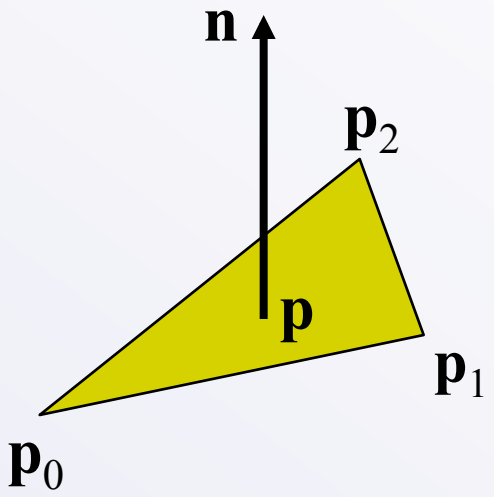
Normais

- Especificado através de `glNormal*()`
  - `glNormal3f(x, y, z);`
  - `glNormal3fv(p);`
- Comprimento deve ser unitário, |**n**|=1
  - Transformações podem afetar o comprimento
  - `glEnable(GL_NORMALIZE)` resulta na degradação na performance
- O vetor normal é parte do estado;

Normal de um triângulo

$$\vec{n} = (p_2 - p_0) \times (p_1 - p_0)$$

$$\vec{n} = \frac{\vec{n}}{|\vec{n}|}$$

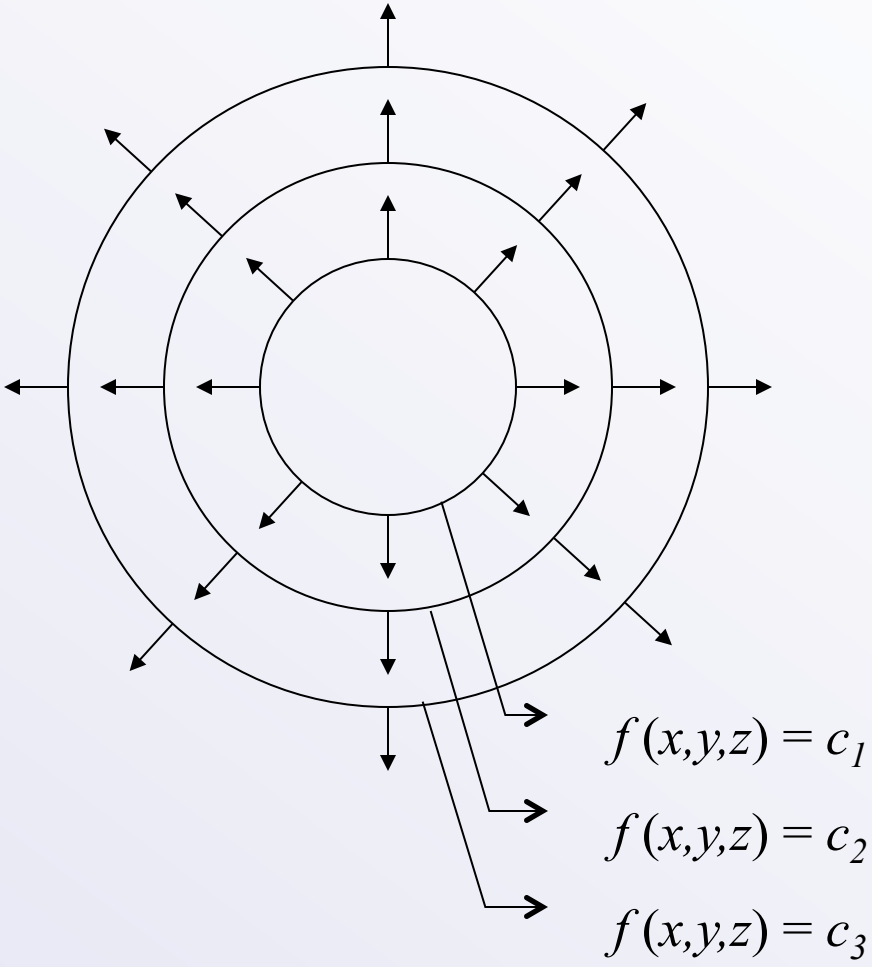


Calculando o Vetor Normal de Superfícies Implícitas

- Normal é dada pelo vetor gradiente

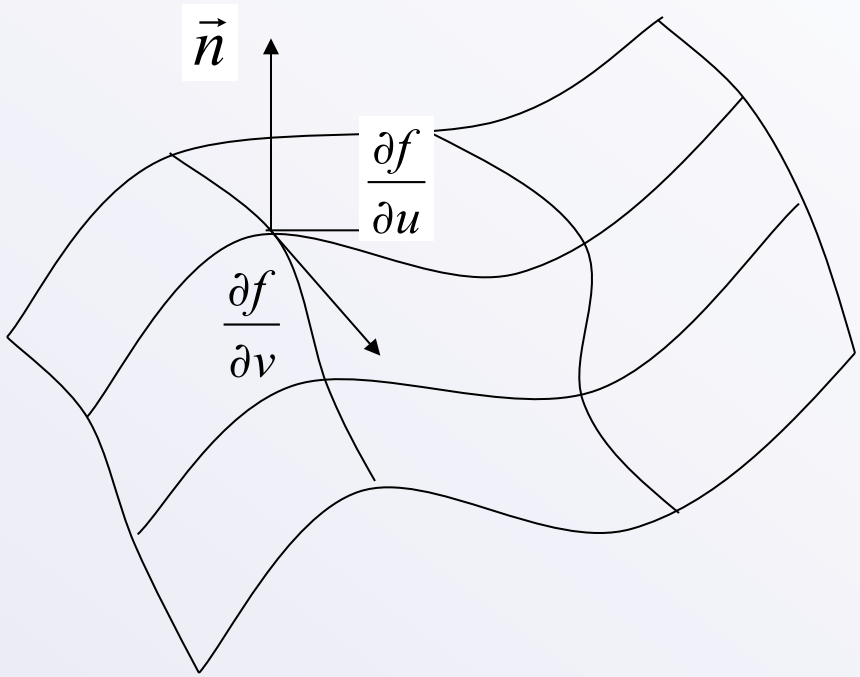
$$f(x,y,z) = 0$$

$$\vec{n} = \begin{pmatrix} \partial f / \partial x \\ \partial f / \partial y \\ \partial f / \partial z \end{pmatrix}$$



Calculando o Vetor Normal de Superfícies Paramétricas

- Normal é dada pelo produto vetorial dos gradientes em relação aos parâmetros *u* e *v*



$$P = \begin{pmatrix} f_x(u,v) \\ f_y(u,v) \\ f_z(u,v) \end{pmatrix}$$

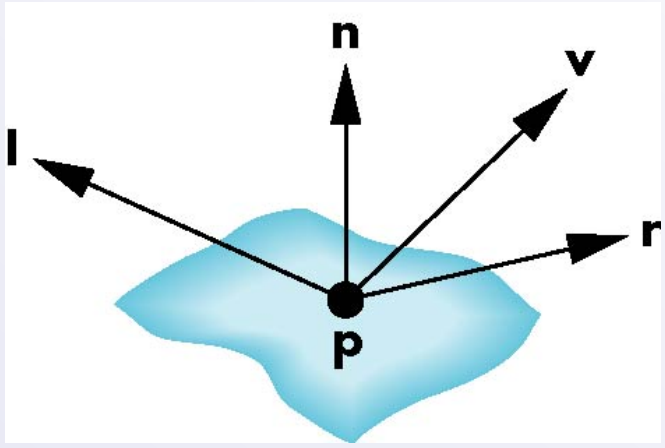
$$\vec{n} = \frac{\partial f}{\partial u} \times \frac{\partial f}{\partial v} = \begin{pmatrix} \partial f_x / \partial u \\ \partial f_y / \partial u \\ \partial f_z / \partial u \end{pmatrix} \times \begin{pmatrix} \partial f_x / \partial v \\ \partial f_y / \partial v \\ \partial f_z / \partial v \end{pmatrix}$$

Modelo de Phong

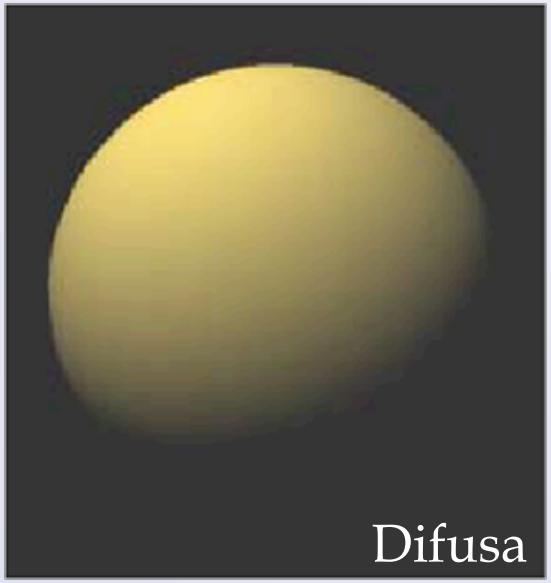
Para cada fonte de luz e componente de cor, o modelo Phong pode ser descrito como:

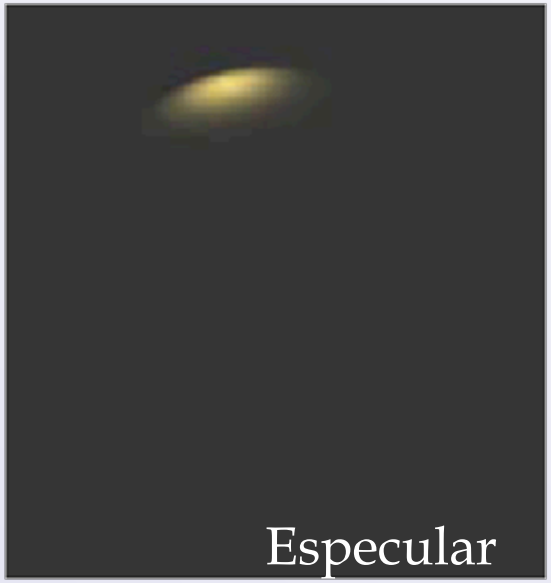
$$I = k_d I_d \, \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{v} \cdot \mathbf{r})^{\alpha} + k_a I_a$$

Para cada componente de cor, adicionamos contribuições de todas as fontes



Componentes do Modelo Phong





Tonalização

- Habilitação através do comando
  - `glEnable(GL_LIGHTING)`
  - Comandos `glColor()` são ignorados
- Cada luz deve ser ligada individualmente
  - `glEnable(GL_LIGHTi)` i=0,1,…,7
- Parâmetros do modelo de iluminação
  - `glLightModeli(parameter, GL_TRUE)`
    - `GL_LIGHT_MODEL_LOCAL_VIEWER` observador não está posicionado no infinito;
    - `GL_LIGHT_MODEL_TWO_SIDED` tonaliza ambos os lados dos polígonos independentemente

Ponto de luz

- Para cada fonte de luz, especificamos intensidades RGBA para cada um dos componentes: difuso, especular e ambiente, e sua posição/direção

```
GLfloat luz0difusa[]={1.0, 0.0, 0.0, 1.0};
GLfloat luz0ambiente[]={1.0, 0.0, 0.0, 1.0};
GLfloat luz0espec[]={1.0, 0.0, 0.0, 1.0};
Glfloat luz0posicao[]={1.0, 2.0, 3,0, 1.0};

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightv(GL_LIGHT0, GL_POSITION, luz0pos);
glLightv(GL_LIGHT0, GL_AMBIENT, luz0ambiente);
glLightv(GL_LIGHT0, GL_DIFFUSE, luz0difusa);
glLightv(GL_LIGHT0, GL_SPECULAR, luz0espec);
```

Distância

- O OpenGL também permite especificar a rapidez com que a intensidade diminui em relação a distância da fonte

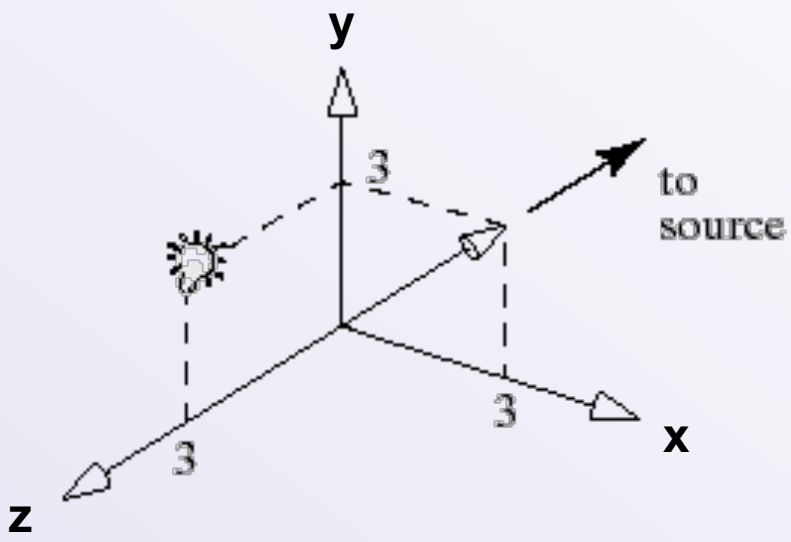
$$atenuação = \frac{1}{a + bx + cx^2}$$

- Onde a,b e c são os coeficientes e x é distância entre posição da luz e do vértice em questão (default: a=1, b=c=0)

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, a);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, b);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, c);
```

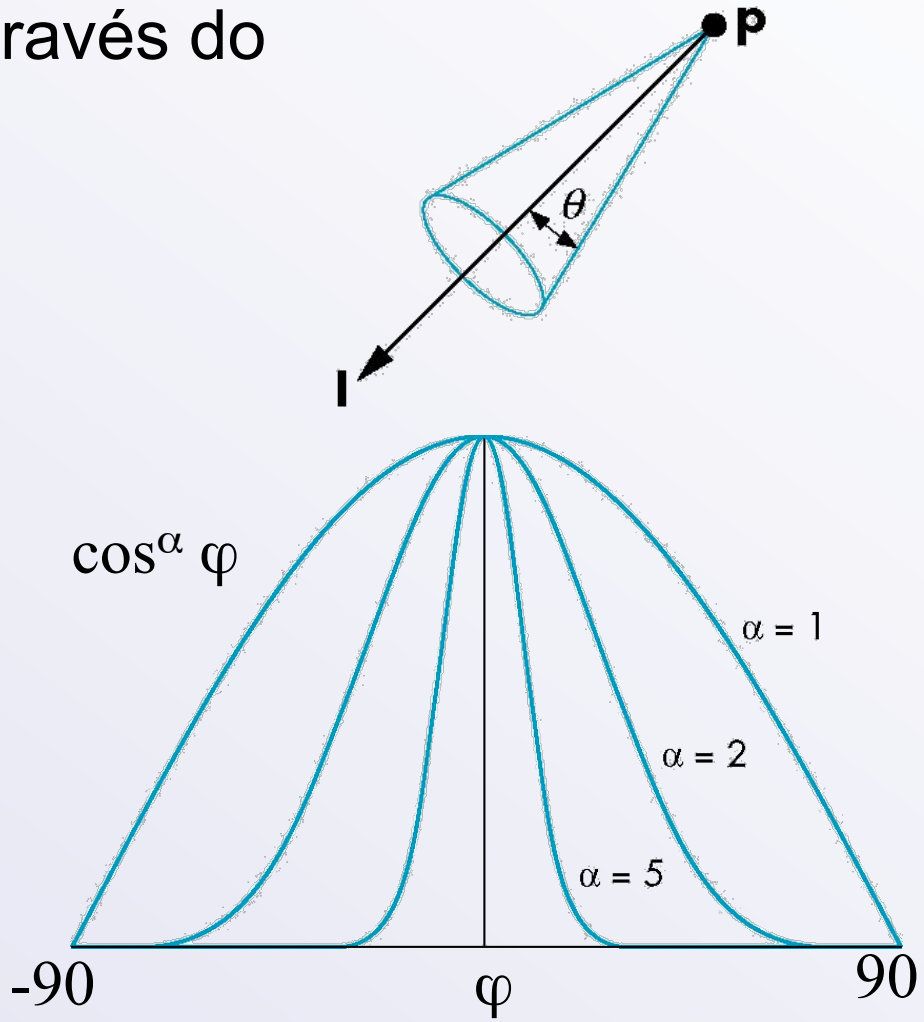
Luzes direcionais

- A figura mostra uma fonte local em (0, 3, 3, 1) e uma fonte distante apontada pelo vetor (3, 3, 0, 0).
- Fontes infinitamente distantes são chamadas “direcionais”
- A direção **l** nos cálculos dos termos difusivo e especular se mantém constantes para todos os vértices da cena.
- As luzes direcionais nem sempre são a melhor opção para conseguir os melhores efeitos visuais.



Luzes spot

- Também é especificada através do comando `glLightv()`
- Direção:
  - `GL_SPOT_DIRECTION`
- Ângulo de corte:
  - `GL_SPOT_CUTOFF`
  - Default: 180º
- Concentração:
  - `GL_SPOT_EXPONENT`
  - Proporcional a $\cos^{\alpha} \varphi$



Usando luzes spot

- Valores default **d** = {0, 0, 0, 1}, φ = 180º, α = 0
- Exemplo:
  - `glLightf (GL_LIGHT_0, GL_SPOT_CUTOFF, 45.0); // (45.0 é` φ `em graus)`
  - `glLightf (GL_LIGHT_0, GL_SPOT_EXPONENT, 4.0); // (4.0 é` α`)`
  - `glLightfv (GL_LIGHT_0, GL_SPOT_DIRECTION, d);`

Luz ambiente

- Um termo global de luz ambiente está presente mesmo se nenhuma luz for criada. A sua cor default é {0.2, 0.2, 0.2, 1.0}.
- Não depende da geometria
- Em OpenGL podemos especificar um termo global de ambiente através da função
  - **`glLightModelfv(GL_LIGHT_MODEL_AMBIENT, global_ambient)`**
- Os valores default para termos ambiente são {0, 0, 0, 1} para todas as luzes

Posição do observador

- OpenGL calcula as reflexões especulares utilizando o vetor médio **h**
- As direções verdadeiras do observador **v** e da luz **l** serão diferentes para cada vértice
- Para aumentar a velocidade de renderização, o OpenGL utiliza o vetor **v** = (0, 0, 1)
- Para utilizar o vetor verdadeiro, devemos executar

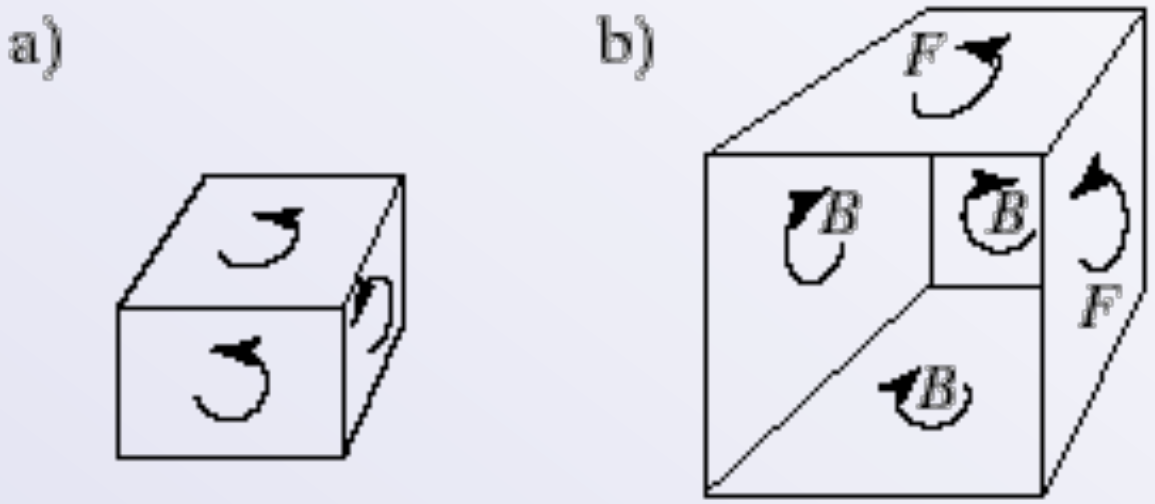
```
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER,
GL_TRUE);
```

Modelo de iluminação

- Cada face poligonal possui dois lados
- O OpenGL não entende o que é “interior” ou “exterior”. Pode somente distinguir entre faces frontais e faces traseiras
- Assumimos que os vértices são definidos no sentido anti-horário
- Uma face é dita como face frontal se os seus vértices são definidos no sentido anti-horário em relação ao observador

Modelo de iluminação

- `glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);`



Transformações

- Fontes de luz também são afetadas pela matriz de modelo;
- Dependendo do tipo da transformação, podemos
  - Mover as luzes com os objetos
  - Mover somente as luzes
  - Mover somente os objetos
  - Mover as luzes e os objetos independentemente

Movimento independente

```
void display()
{
  GLfloat position[] = {2,1,3,1}; // posição inicial
   // limpa buffers
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glPushMatrix();
   glRotated(...); glTranslated(...); // move luz
   glLightfv(GL_LIGHT0, GL_POSITION, position);
   glPopMatrix();
   gluLookAt(...); // seleciona câmera
    // Desenha objeto
    glutSwapBuffers();
}
```

Movimento dependente

```
GLfloat pos[ ] = {0,0,0,1};
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
// luz posicionada em (0,0,0)
glLightfv(GL_LIGHT0, GL_POSITION, position);
// movimenta luz e câmera
gluLookAt(...);
// desenha objeto
```

Demo lightposition.c

Demo movelight.c

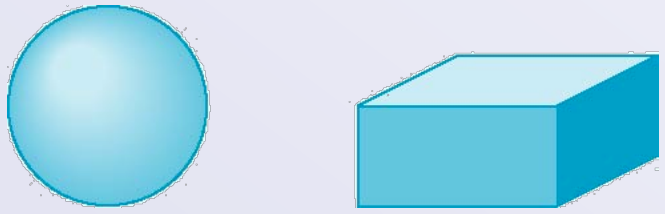
Propriedades de materiais

- Materiais são parte do estado do OpenGL e correspondem aos termos no modelo Phong modificado
- Especificados através de **`glMaterialv()`**

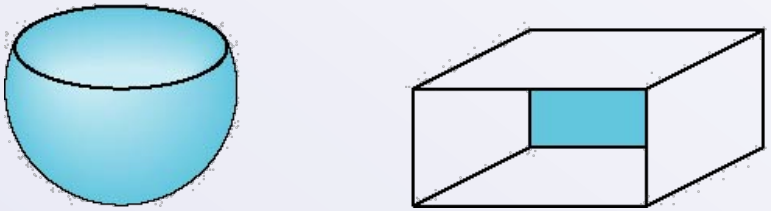
```
GLfloat ambiente[] = {0.2, 0.2, 0.2, 1.0};
GLfloat difuso[] = {1.0, 0.8, 0.0, 1.0};
GLfloat especular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat brilho = 100.0
glMaterialf(GL_FRONT, GL_AMBIENT, ambiente);
glMaterialf(GL_FRONT, GL_DIFFUSE, difuso);
glMaterialf(GL_FRONT, GL_SPECULAR, especular);
glMaterialf(GL_FRONT, GL_SHININESS, brilho);
```

Faces frontais e traseiras

- O default é tonalizar somente faces frontais
- O OpenGL tonalizará ambos os lados da superfície se usado o flag **`GL_FRONT_AND_BACK`**
- Cada lado poderá ter propriedades diferentes de materiais especificados através de **`GL_FRONT`**, **`GL_BACK`**, or **`GL_FRONT_AND_BACK`** na função **`glMaterialf`**



faces traseiras invisíveis



faces traseiras visíveis

Termo emissivo

- Podemos simular uma fonte de luz em OpenGL adicionando um termo emissivo ao material
- Este termo não está sujeito a transformações ou é afetado por outras fontes de luz

```
GLfloat emissao[] = {0.0, 0.3, 0.3, 1.0};
glMaterialf(GL_FRONT, GL_EMISSION, emissao);
```

Transparência

- Materiais são especificados através de valores RGBA
- O valor A (Alpha) pode ser usado para deixar as superfícies translúcidas
- O default é todas as superfícies serem opacas independentemente do valor de A

Tonalização de polígonos

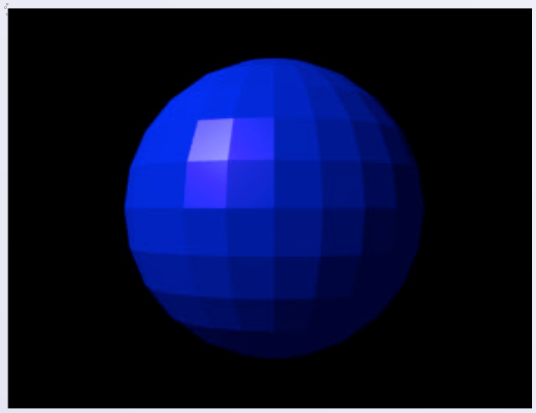
- Cálculos de tonalização são feitos por vértice
  - Cores de vértices se tornam tonalizações
- Por default, tons dos vértices são interpolados dentro do polígono
  - **`glShadeModel(GL_SMOOTH)`**
- Caso usemos **`glShadeModel(GL_FLAT)`** a cor do primeiro vértice determinará a cor de todo o polígono.

Normais

- Polígonos possuem somente uma normal
  - Tonalidades nos vértices calculados pelo modelo Phong são quase os mesmos
  - Idêntico para um observador distante (default) ou se não existe componente especular
- Gostaríamos de diferentes normais em cada vértices

Tonalização faceteada

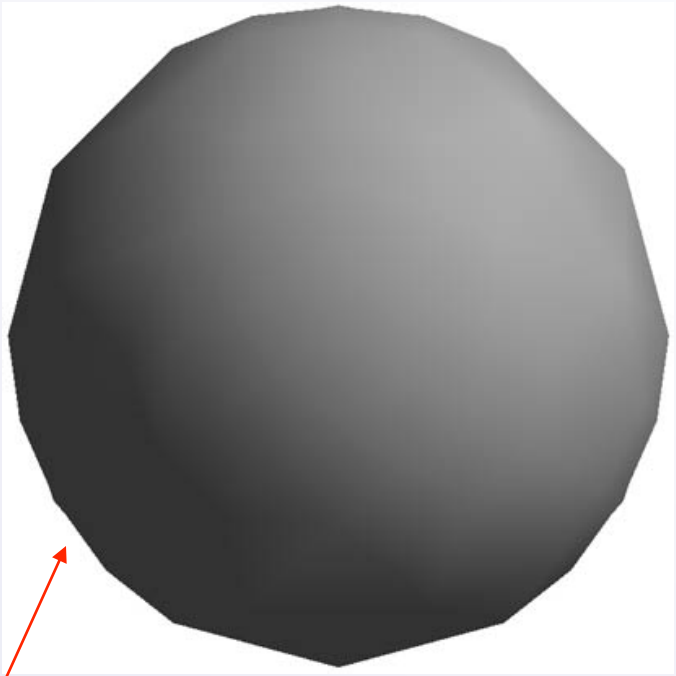
- Problemas com brilhos especulares:
  - Se existir um grande componente especular em um vértice, este brilho será dividido uniformemente através da face.
  - Caso o brilho especular não caia em um ponto representativo, ele é totalmente perdido.
- Consequentemente, normalmente não incluimos componentes especulares no cálculo da tonalização faceteada.



FLAT SHADING

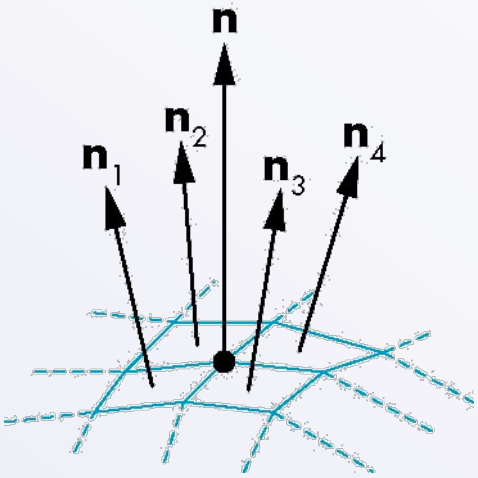
Tonalização suave

- Uma normal diferente para cada vértice
- Fácil para a esfera, já que podemos determinar a normal analiticamente
  - Se centrada na origem $\mathbf{n} = \mathbf{p}$
- Aspecto de suavização
- Note o artefato de silhueta



Tonalização de malhas

- No exemplo anterior, podemos determinar a normal analiticamente
- Não pode ser generalizado para objetos poligonais arbitrários
- Para modelos poligonais, Gouraud propôs usar a média das normais ao redor de cada vértice



$$\mathbf{n} = (\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4) \,/\, |\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4|$$

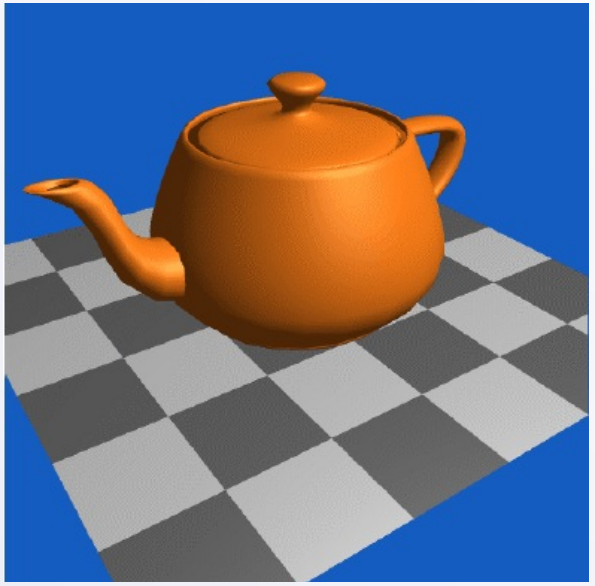
Gouraud e Phong

- Tonalização **Gouraud**
  - Determine a normal média em cada vértice
  - Aplicar o modelo modificado de Phong em cada vértice
  - Interpolação de tons dentro do polígono
- Tonalização **Phong**
  - Determina normais dos vértices
  - Interpolação das normais nas arestas e entre as arestas
  - Aplicar o modelo Phong em cada fragmento

Comparação

flat

Gouraud



Phong

Comparação

- Caso o polígono possua altas curvaturas, a tonalização Phong parecerá mais suave, mas a tonalização Gouraud mostrará artefatos nas arestas
- A tonalização Phong requer mais esforço computacional que Gouraud
  - Até recentemente não era disponível em sistemas de tempo real
  - Pode ser implementada através de tonalizadores de fragmentos (“fragment shaders”)
- Ambas as técnicas dependem de estruturas de dados para representar as as malhas e determinar as normais em cada vértice

Exemplo

http://www.hlc-games.de/forum/viewtopic.php?f=10&t=56

Modelagem geométrica

- Placas gráficas podem renderizar cerca de 10 milhões de polígonos por segundo
- Todo esse poder é insuficiente para representar
  - Nuvens
  - Vegetação
  - Pele
  - Pelos, cabelos

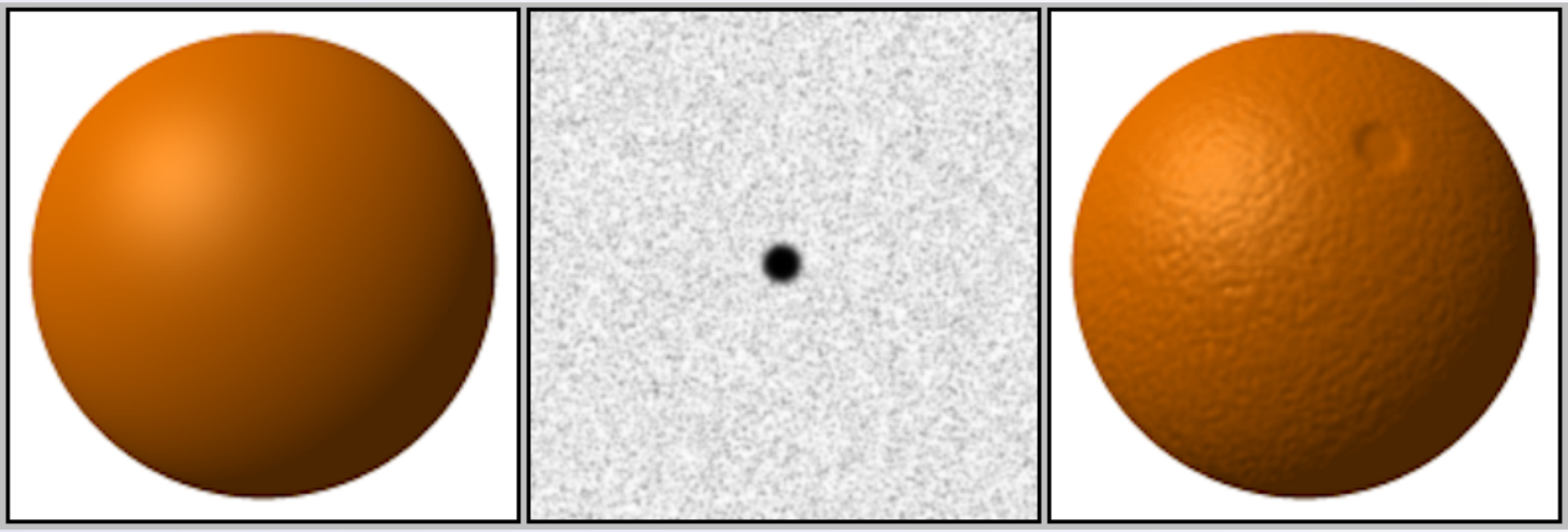
Modelando uma laranja

- Começamos como uma esfera de cor laranja
  - Simples demais
- Trocar a esfera por uma forma mais complexa
  - Ainda não captura todas as características da superfície da fruta
  - Muitos polígonos são necessários para modelar as pequenas rugosidades

Modelando uma laranja

- Tiramos uma foto digital de uma laranja e “colamos” sobre a esfera
  - Mapeamento de textura
- Ainda pode não ser suficiente, pois a aparência final é suave demais
  - Precisamos alterar localmente a forma
  - Bump mapping (“rugosidade”)

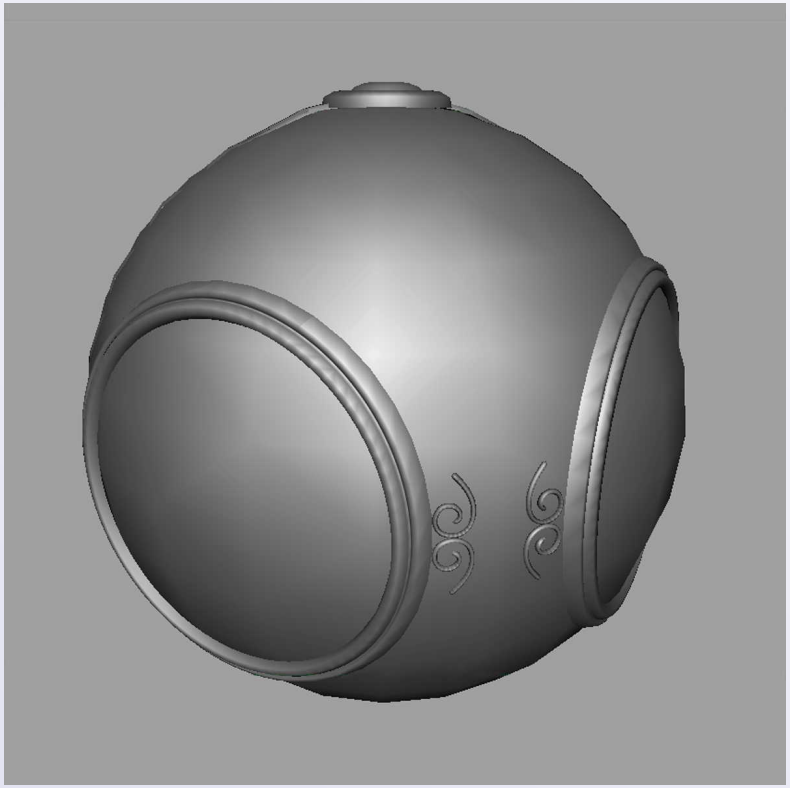
Exemplo



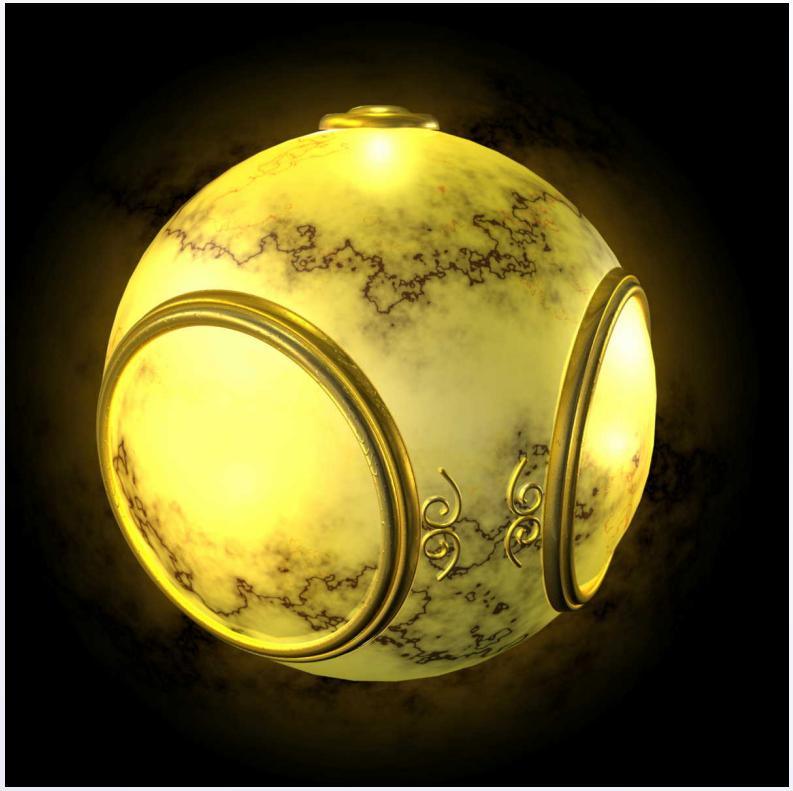
Três tipos de mapeamento

- Mapeamento de textura
  - Utilizam-se imagens para o preenchimento dos polígonos
- Ambiente (reflexão)
  - Uma foto do ambiente é usada como textura
  - Permite a simulação de superfícies altamente especulares
- Bump mapping/Displacement mapping
  - Alteração dos vetores normais durante o processo de renderização/coordenadas dos pixels

Mapeamento de textura



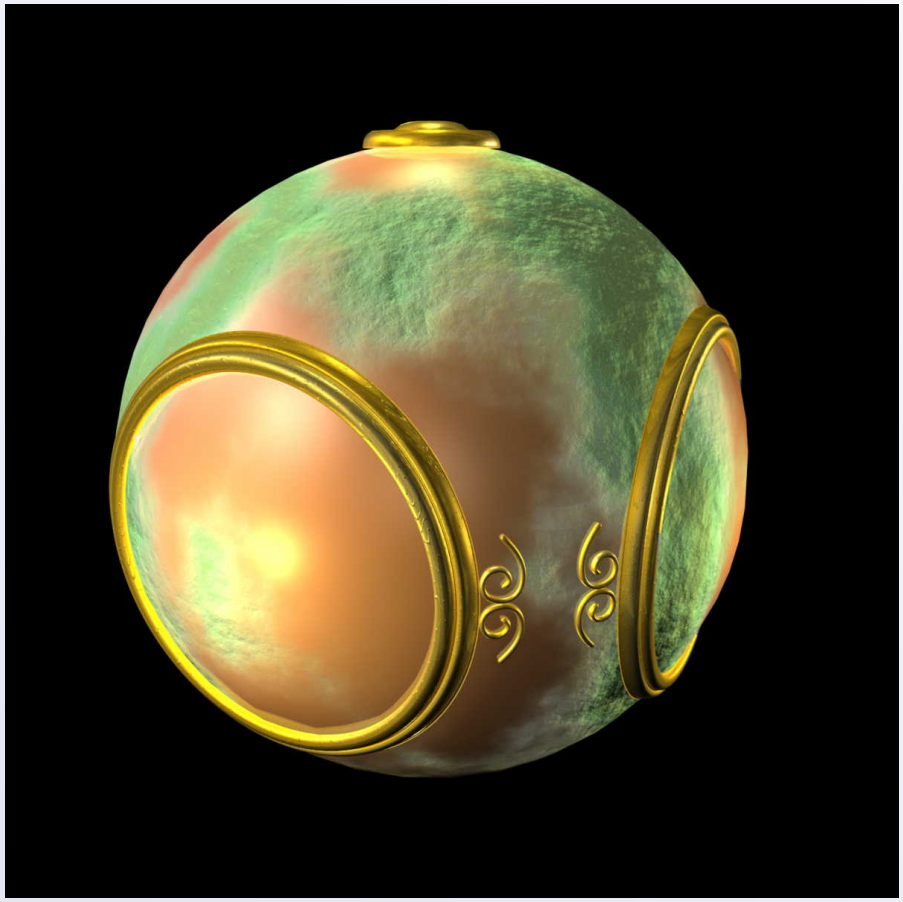
modelo geométrico



mapeamento de textura

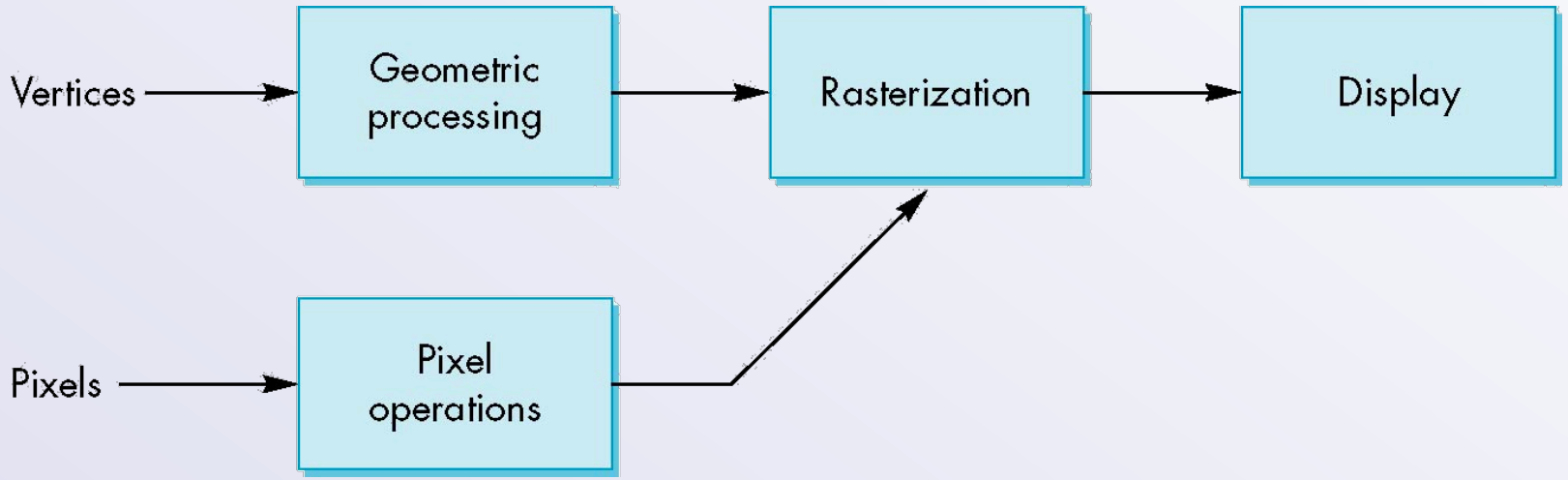
Mapeamento de ambiente

Bump Mapping



Mapeamento no pipeline

- Técnicas de mapeamento são implementadas no final do pipeline
  - Eficiente pois poucos polígonos sobrevivem ao processo de recorte



Mapeamento de textura

- Embora a idéia seja simples, ela envolve a transformação entre 3 ou 4 diferentes sistemas de coordenadas

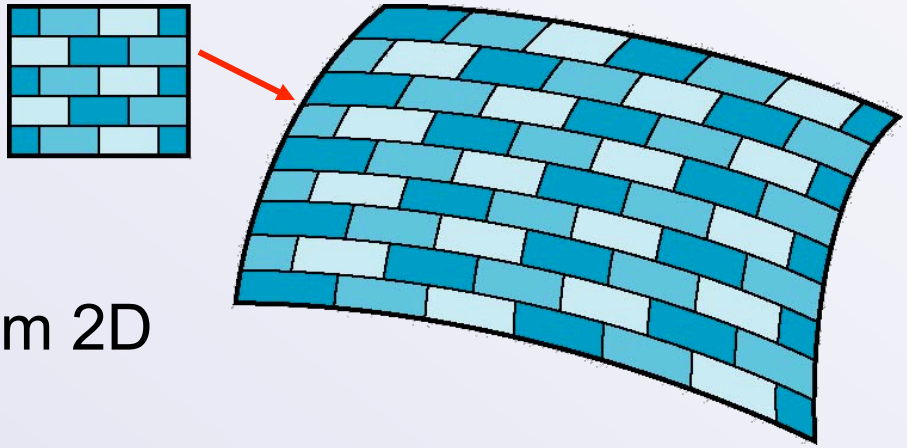


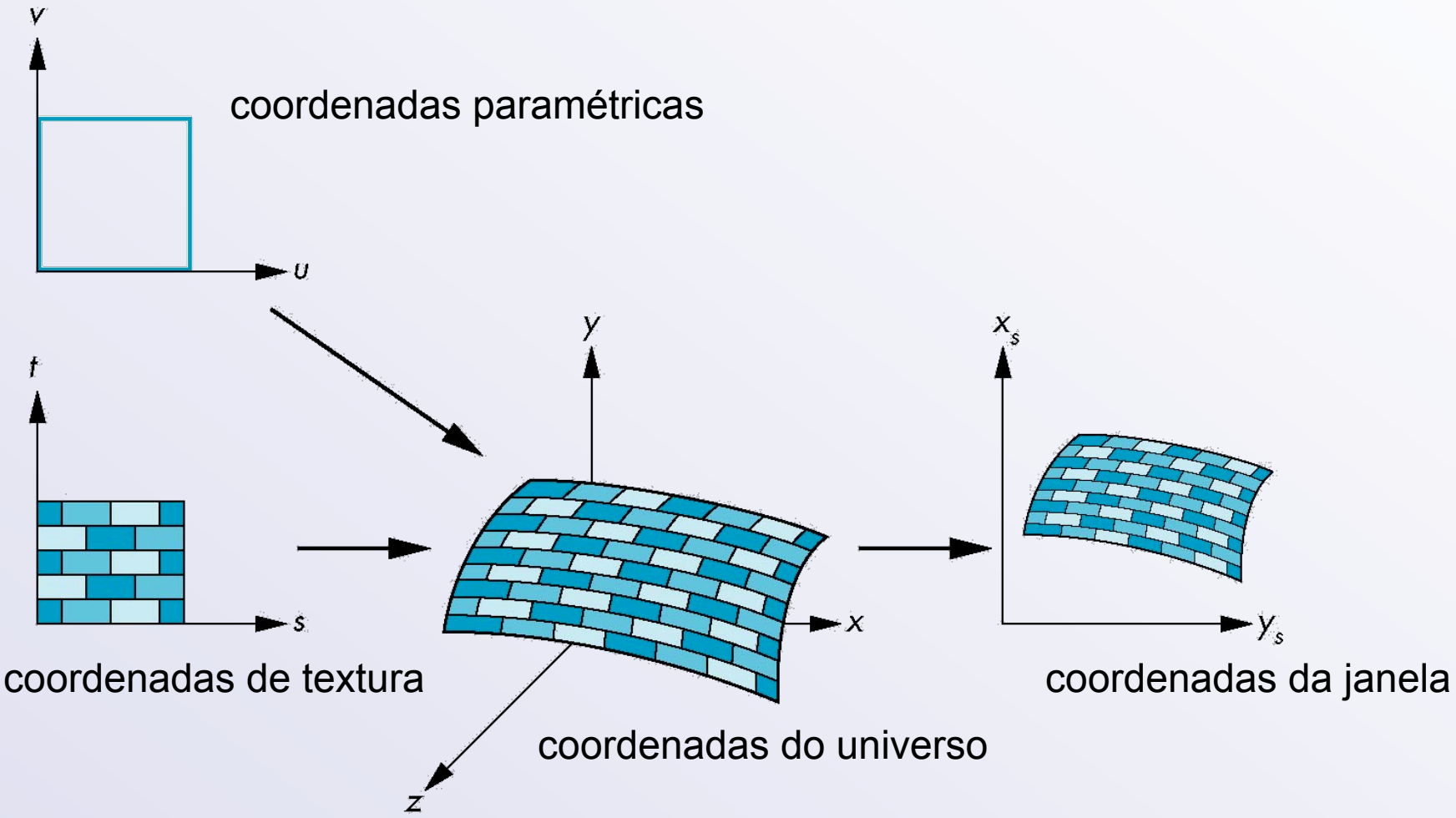
Imagem 2D

superfície 3D

Sistemas de coordenadas

- Coordenadas paramétricas
  - Modelagem de curvas e superfícies
- Coordenadas de textura
  - Identificação de pontos da imagem a ser mapeada
- Coordenadas do universo ou do objeto
  - Onde o mapeamento acontece
- Coordenadas da janela (viewport)
  - Onde a imagem final é produzida

Mapeamento de textura



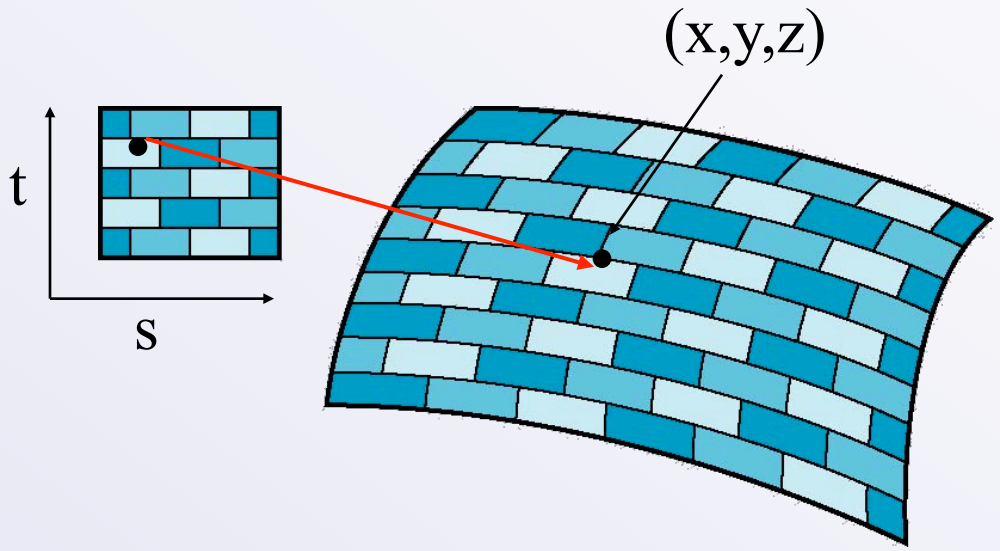
Funções de mapeamento

- Como achar as funções de mapeamento ?
- Aparentemente precisaremos de três funções

$$x = Tx(s,t)$$
$$y = Ty(s,t)$$
$$z = Tz(s,t)$$

- Mas desejamos o caminho inverso

  Como descobrir *s* e *t* ?

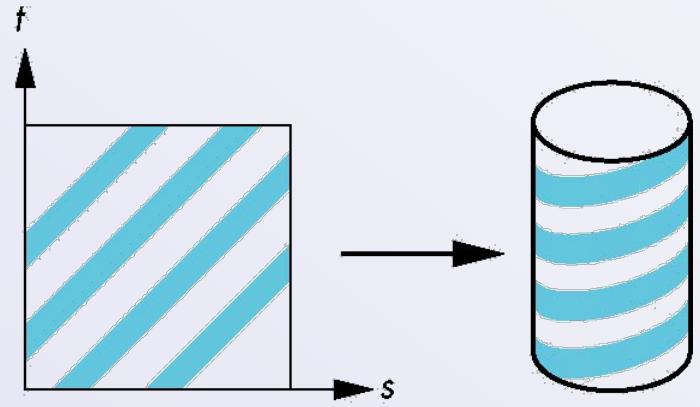


Mapeamento inverso

- Dado um pixel, queremos saber qual o ponto no objeto a que ele corresponde
- Dado um ponto no objeto, queremos saber que ponto na textura ele corresponde
- Precisamos de um mapeamento
  - $s = T_s(x,y,z)$
  - $t = T_t(x,y,z)$
- De um modo geral, são difíceis de achar

Mapeamento intermediário

- Uma solução ao problema é primeiro mapear a textura para uma superfície intermediária mais simples
- Exemplo: cilindro



Mapeamento cilíndrico

Equações paramétricas de um cilindro

$$x = r \cos(2\pi u)$$
$$y = r \operatorname{sen}(2\pi u)$$
$$z = v\, h$$

Mapeia um retângulo no espaço u,v em [0,1] para um cilindro de raio *r* e altura *h* em coordenadas do universo

$$s = u$$
$$t = v$$

Mapeamento esférico

Podemos usar uma esfera paramétrica

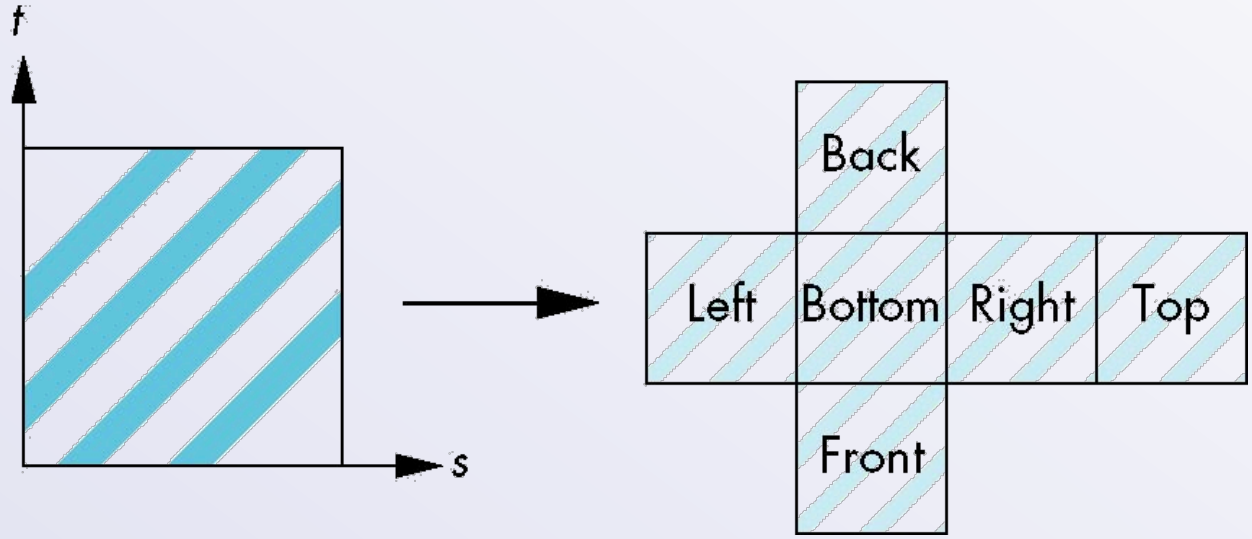
$$x = r \cos(\pi u)$$
$$y = r \operatorname{sen}(\pi u) \cos(2\pi v)$$
$$z = r \operatorname{sen}(\pi u) \operatorname{sen}(2\pi v)$$

de forma similar ao cilindro, mas temos que decidir aonde inserir a distorção

Esferas são usadas em mapeamento de ambiente

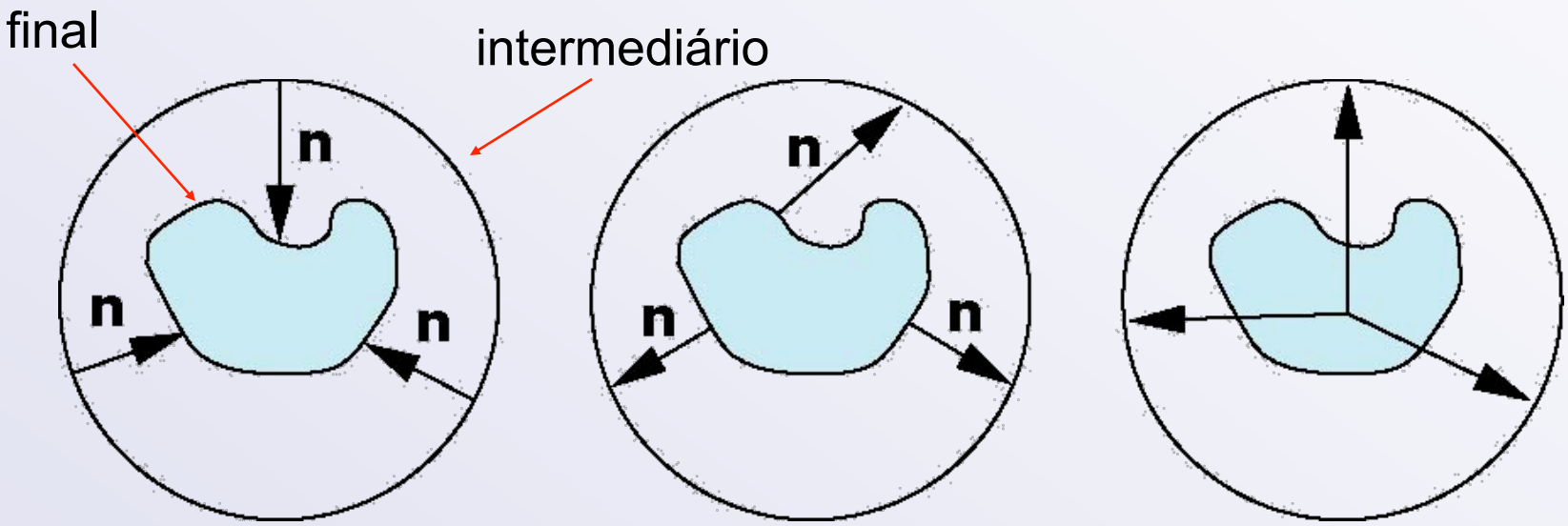
Mapeamento cúbico

- Fácil de usar com projeções ortográficas simples
- Usado em mapeamentos de ambiente



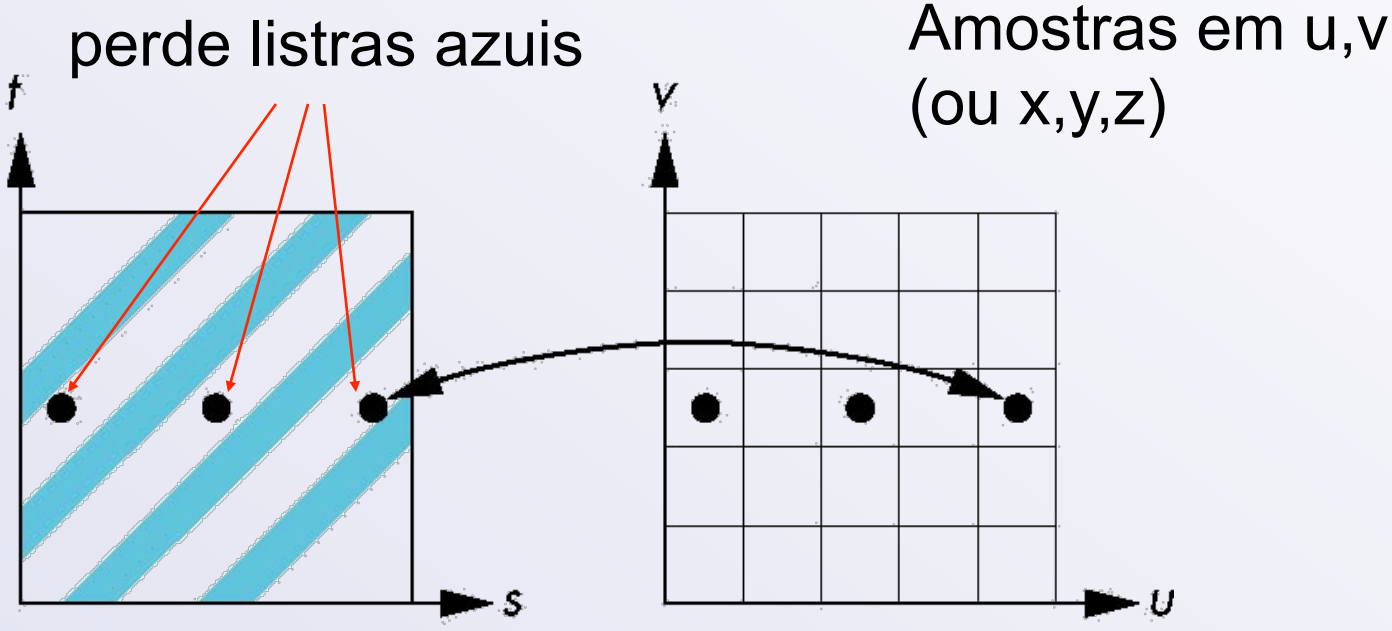
Mapeamento final

- Mapear objeto intermediário para o objeto final



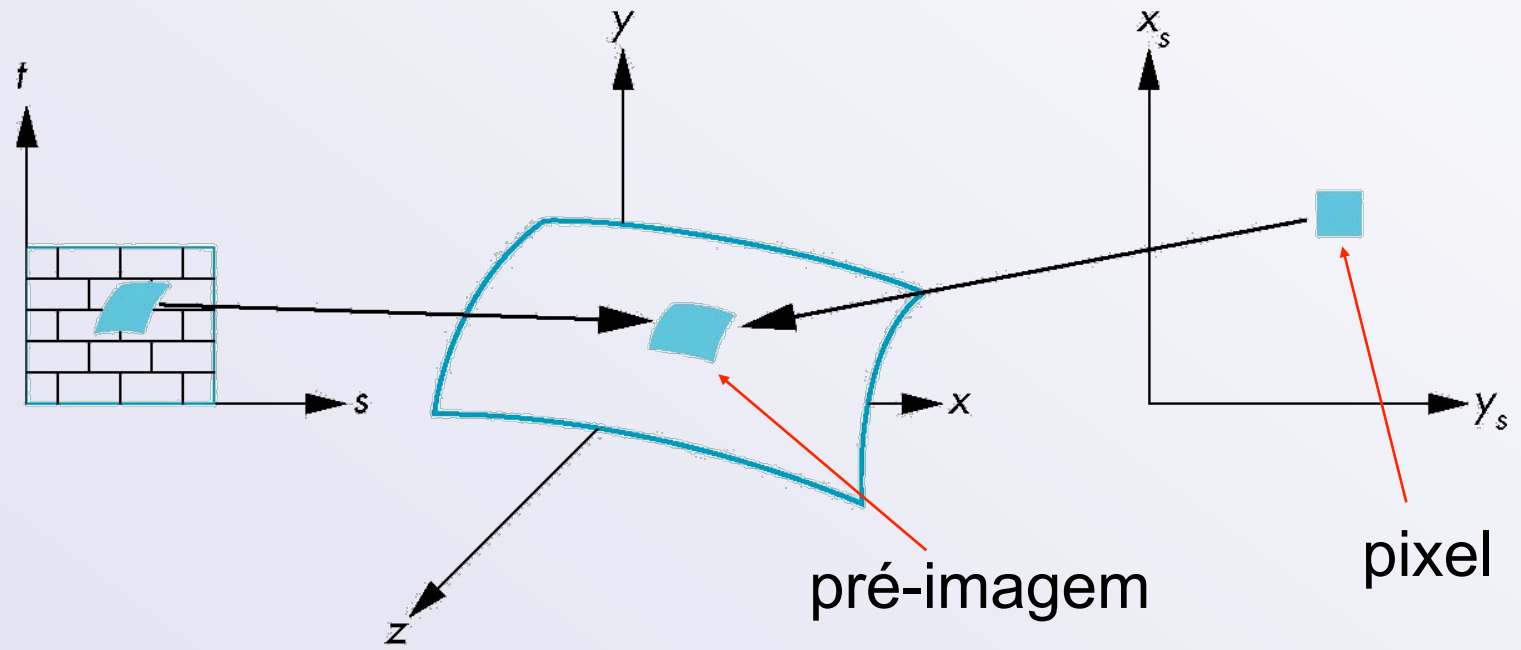
Subamostragem

- A amostragem de textura por pontos pode acarretar em erros de aliasing



Área

Uma melhor solução, porém menos eficiente, é utilizar uma área de amostragem

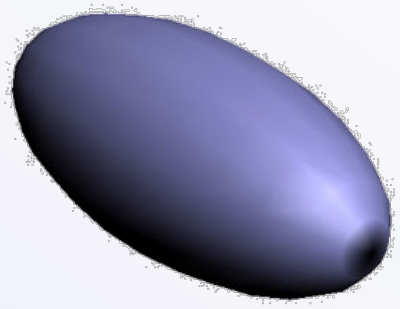


Demo Nate Robins

- texture.c

Tarefa de casa

Uma elipsóide é descrita pelas seguintes equações paramétricas:



- $P_x(\theta, \phi) = a \cos(\theta) \text{ sen}(\varphi)$
- $P_y(\theta, \phi) = b \text{ sen}(\theta) \text{ sen}(\varphi)$
- $P_z(\theta, \phi) = c \cos(\varphi)$

Onde $\theta \in [0, 2\pi]$ e $\varphi \in [0,\pi]$, e a,b,c são os comprimentos de seus semi-eixos

- Crie uma função que desenhe uma elipsóide dados a, b, c e o número de subdivisões de θ e φ
- Calcule o vetor normal em cada ponto P utilizando os vetores componentes de cada face
  - Ou diretamente da equação paramétrica!