



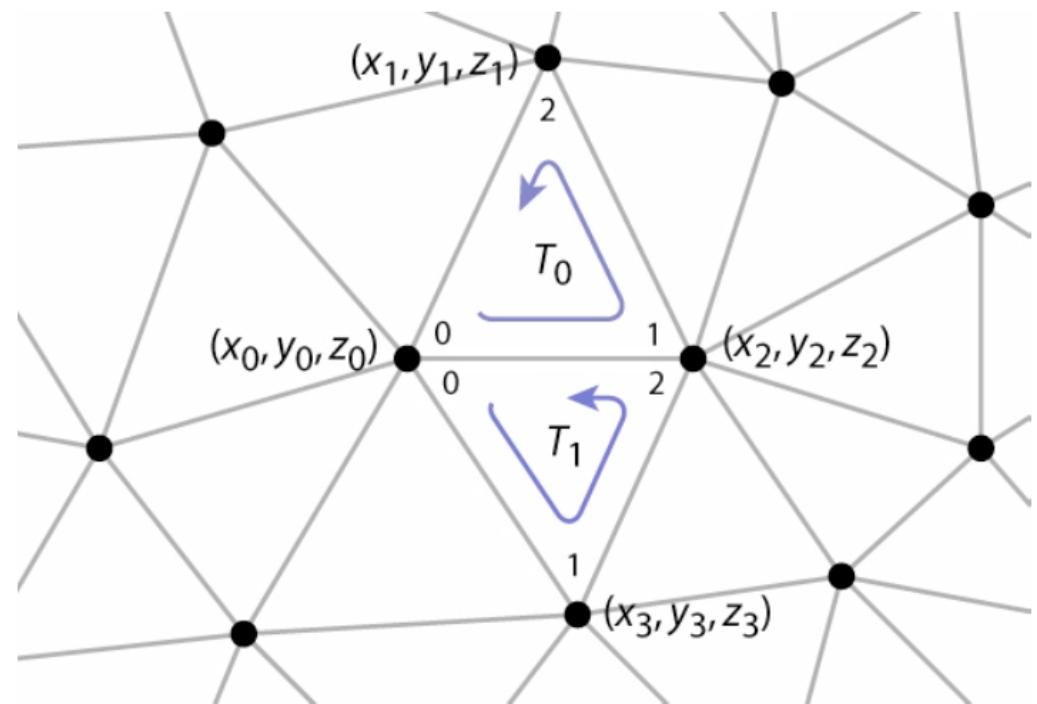
MAC420/5744: Introdução à Computação Gráfica

Marcel P. Jackowski
mjack@ime.usp.br

Aula #6: Tonalização (Parte II)

Triângulos

	[0]	[1]	[2]
tris[0]	x_0, y_0, z_0	x_2, y_2, z_2	x_1, y_1, z_1
tris[1]	x_0, y_0, z_0	x_3, y_3, z_3	x_2, y_2, z_2
\vdots	\vdots	\vdots	\vdots



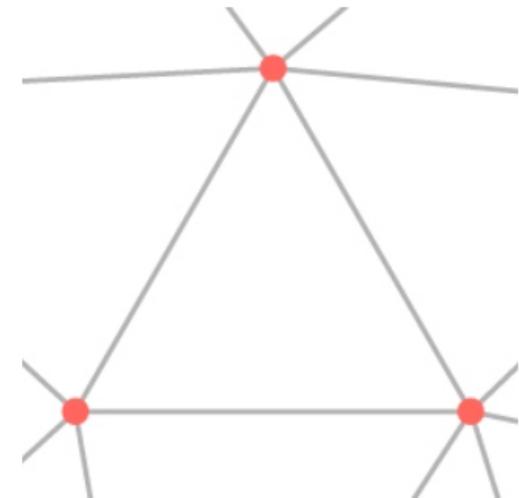
Indexação de triângulos

- Store each vertex once
- Each triangle points to its three vertices

```
Triangle {  
    Vertex vertex[3];  
}
```

```
Vertex {  
    float position[3]; // or other data  
}  
// ... or ...
```

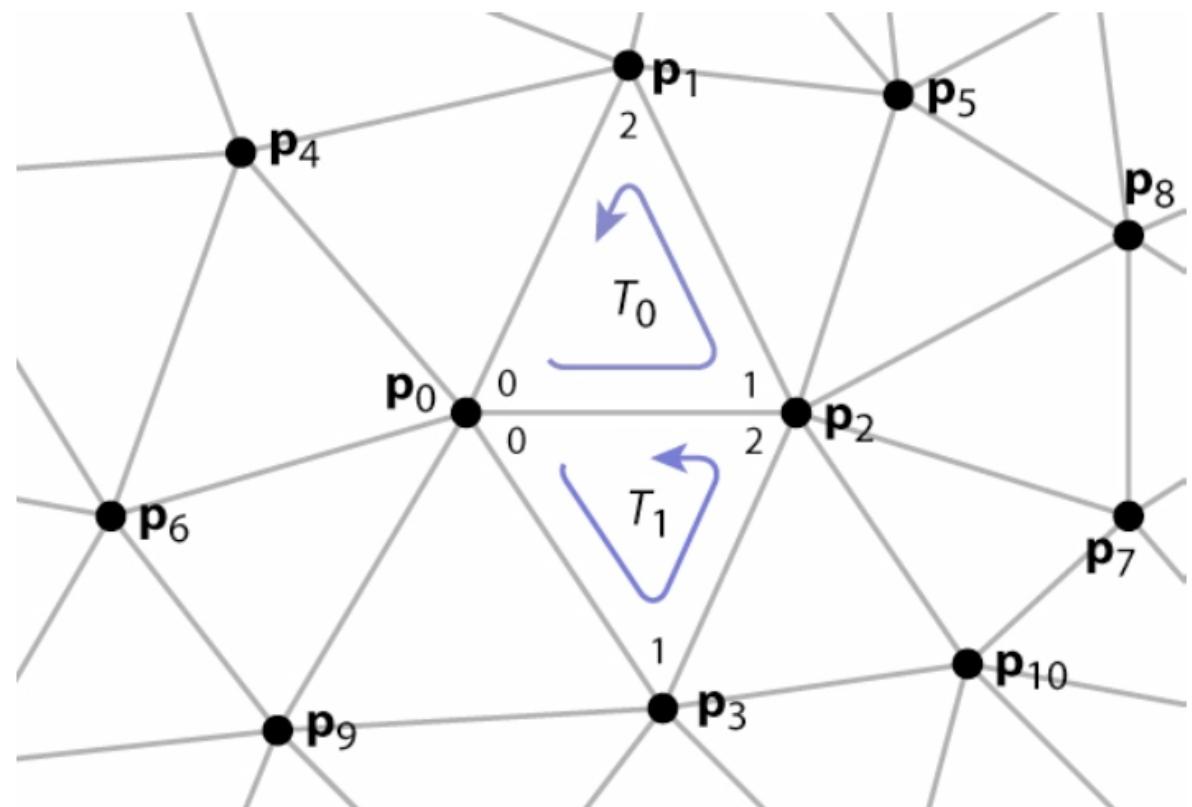
```
Mesh {  
    float verts[nv][3]; // vertex positions (or other data)  
    int tInd[nt][3]; // vertex indices  
}
```



Indexação de triângulos

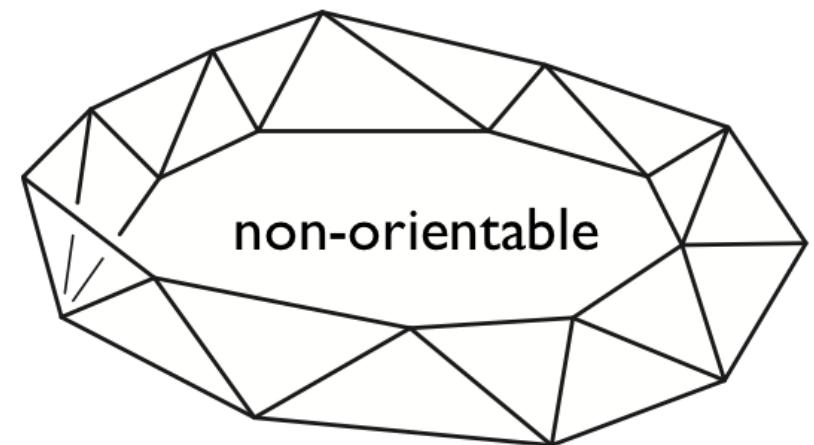
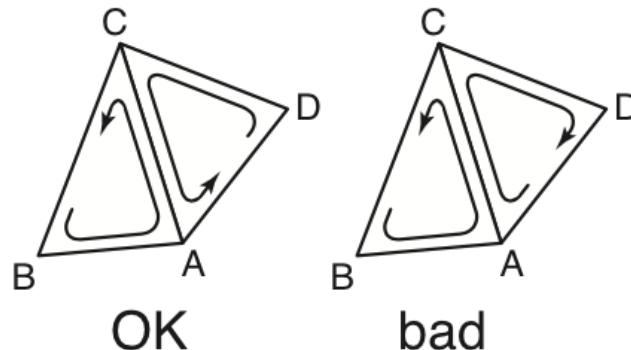
verts[0]	x_0, y_0, z_0
verts[1]	x_1, y_1, z_1
	x_2, y_2, z_2
	x_3, y_3, z_3
:	

tInd[0]	0, 2, 1
tInd[1]	0, 3, 2
:	



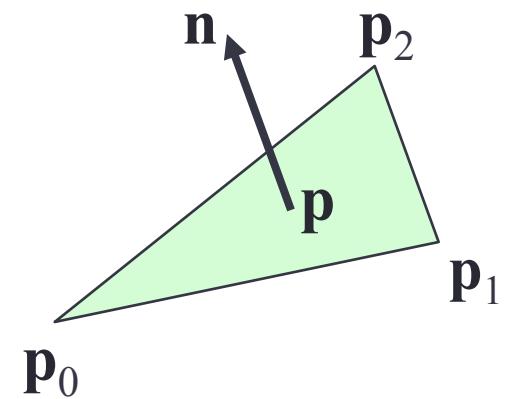
Orientação consistente

- Consistent orientation
 - Which side is the “front” or “outside” of the surface and which is the “back” or “inside?”
 - rule: you are on the outside when you see the vertices in counter-clockwise order
 - in mesh, neighboring triangles should agree about which side is the front!
 - caution: not always possible



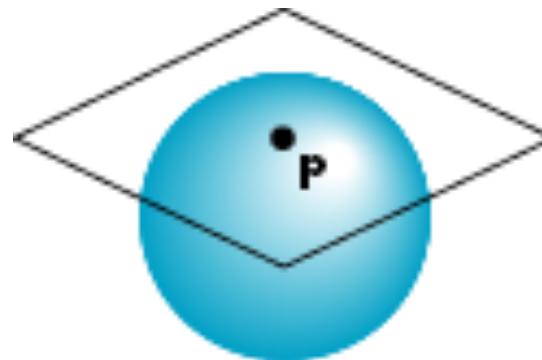
Normal de um triângulo

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



Vetor normal à esfera

- Função implícita $f(x,y,z)=0$
 - $f(x,y,z) = x^2+y^2+z^2-1 = 0$
 - $f(p) = p^2 - 1 = 0$
- Normal dada pelo gradiente de f
 - $n = [\partial f / \partial x, \partial f / \partial y, \partial f / \partial z]^T$

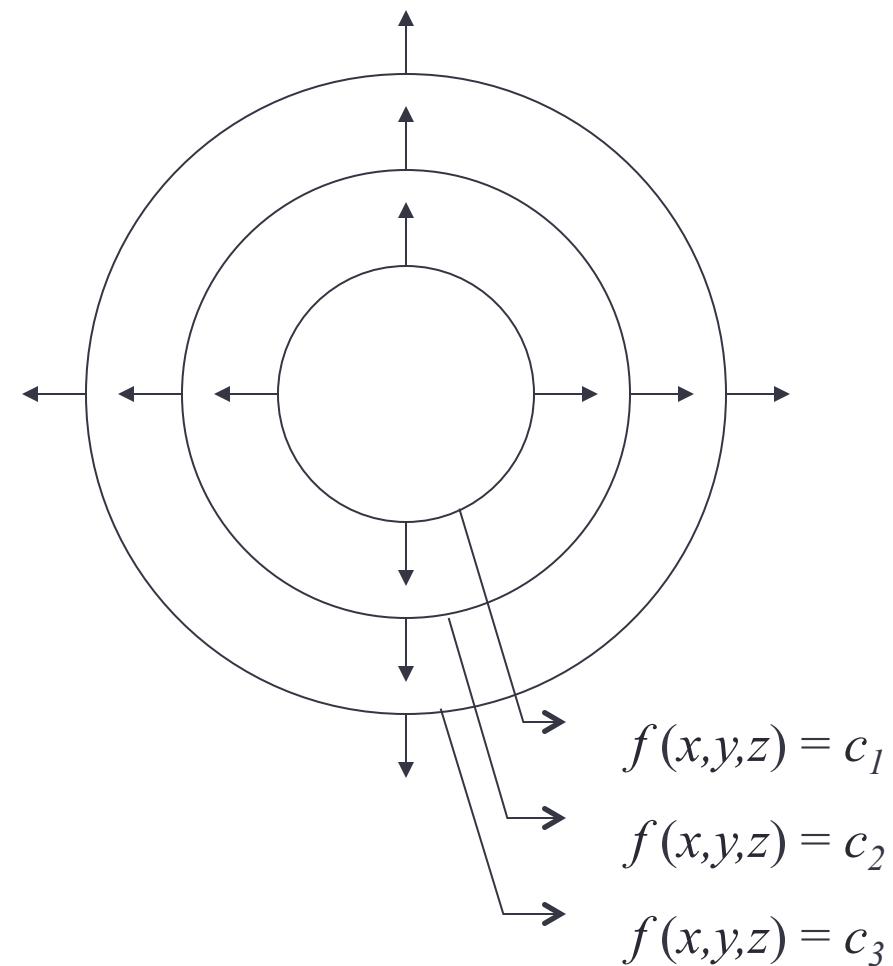


Superfícies implícitas

- Vetor normal é dado pelo gradiente da função

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

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



Equação paramétrica da elipsóide

- Uma elipsóide é descrita pelas seguintes equações paramétricas:
 - $x = a \sin(v) \cos(u)$
 - $y = b \sin(v) \sin(u)$
 - $z = c \cos(v)$

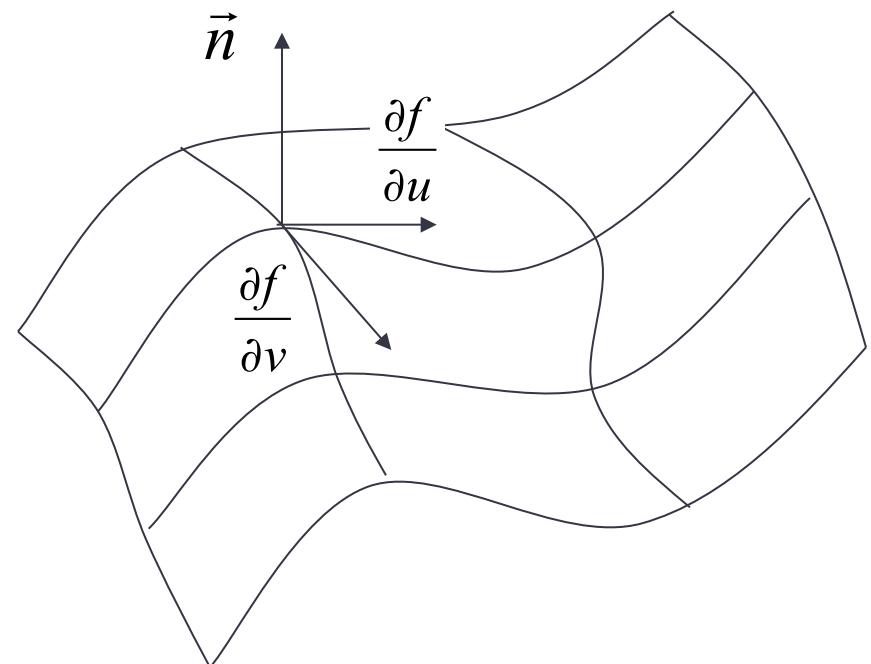
Onde $0 \leq u \leq 2\pi$, representa a longitude,
 $0 \leq v \leq \pi$ representa a latitude e a, b, c são os comprimentos de seus semi-eixos
- Calcular a forma paramétrica do vetor normal

Superfícies paramétricas

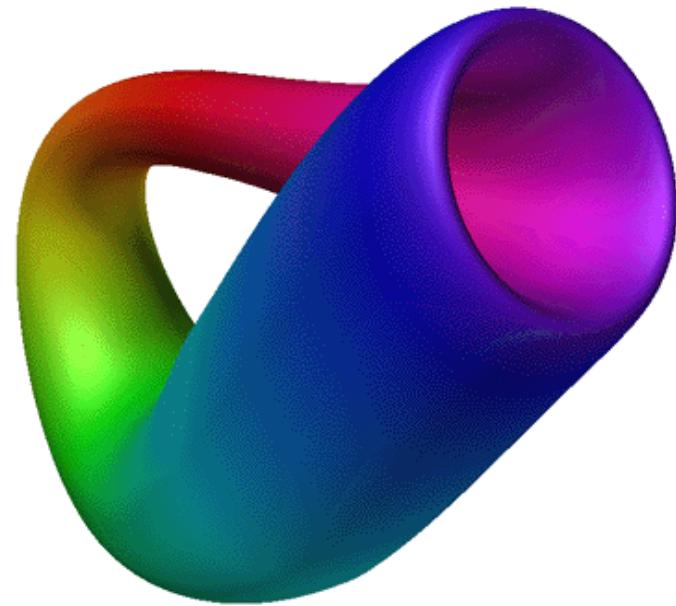
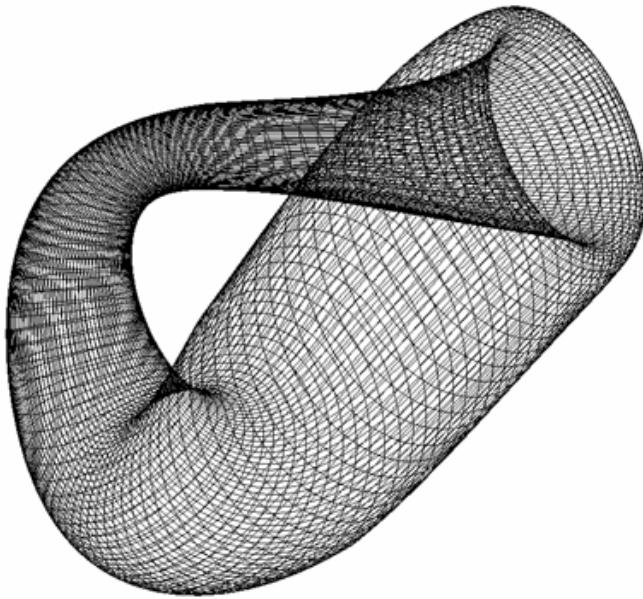
- Normal é dada pelo produto vetorial dos gradientes em relação aos parâmetros u e v

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

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

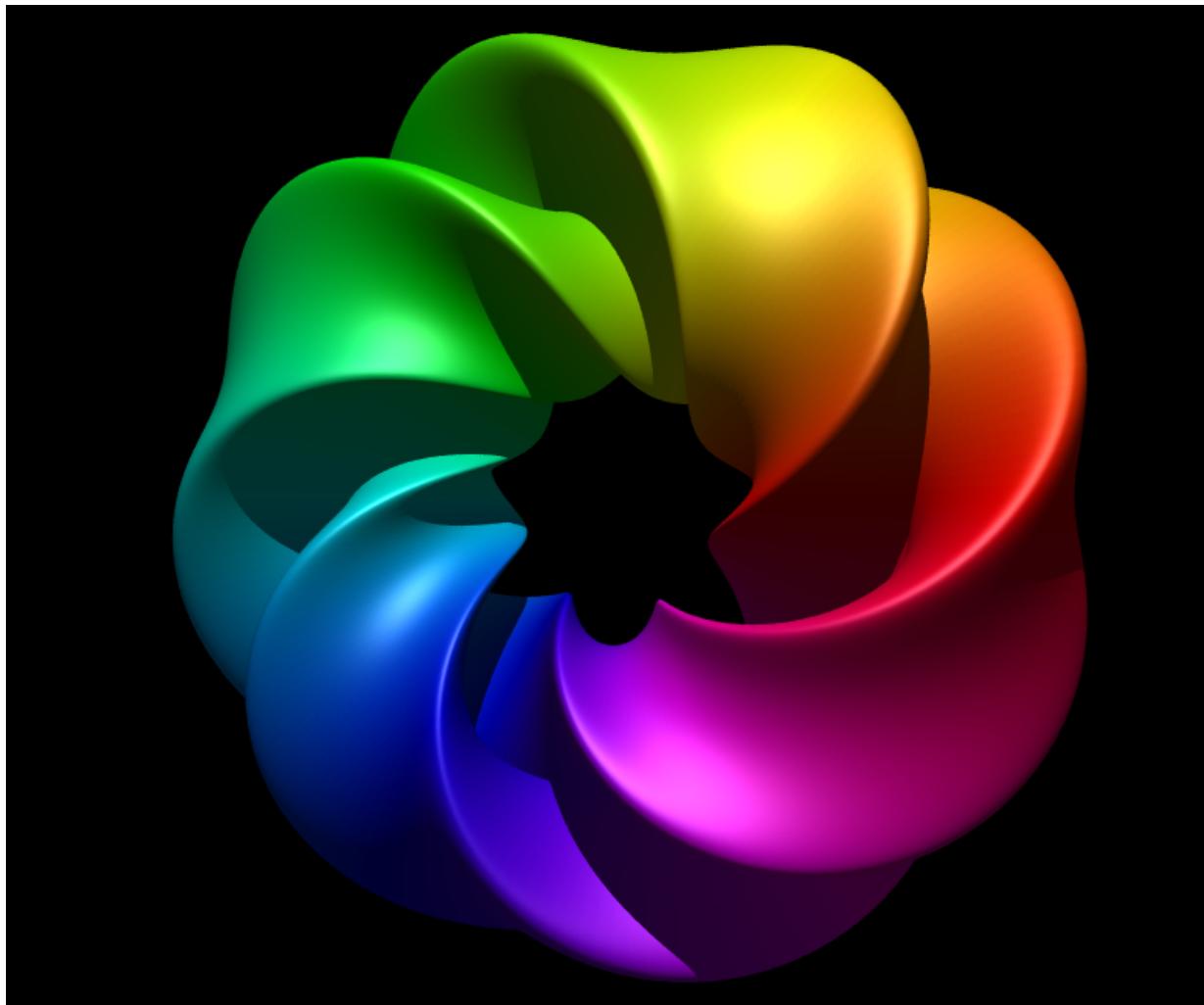


Klein bottle

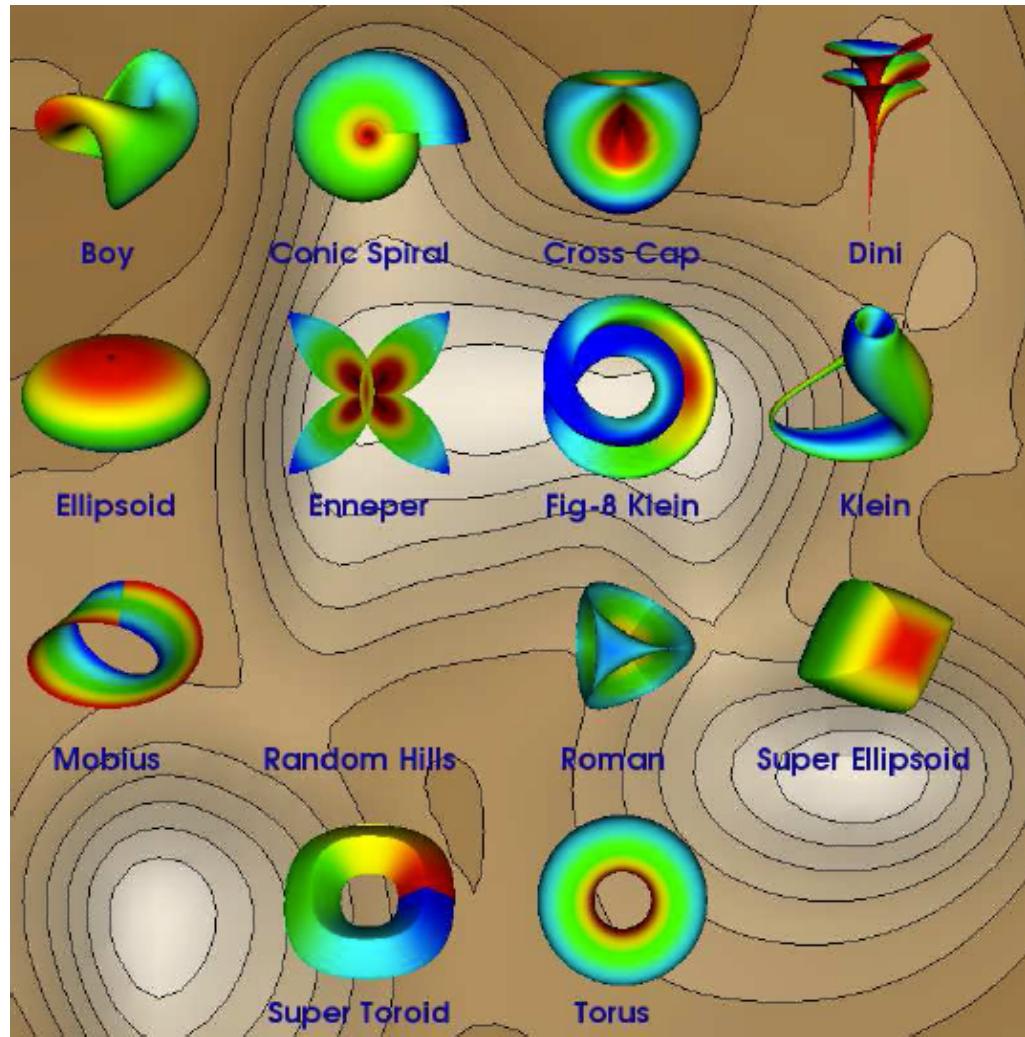


```
r = 4 * (1 - cos(u) / 2);
if (u < PI) {
    x = 6*cos(u) * (1+sin(u)) + r*cos(u)*cos(v);
    y = 16*sin(u) + r*sin(u)*cos(v);
} else {
    x = 6*cos(u) * (1+sin(u)) + r*cos(v + PI);
    y = 16*sin(u);
}
z = r * sin(v);
```

Gray's Kleinbottle



Outros exemplos



Supertoróides

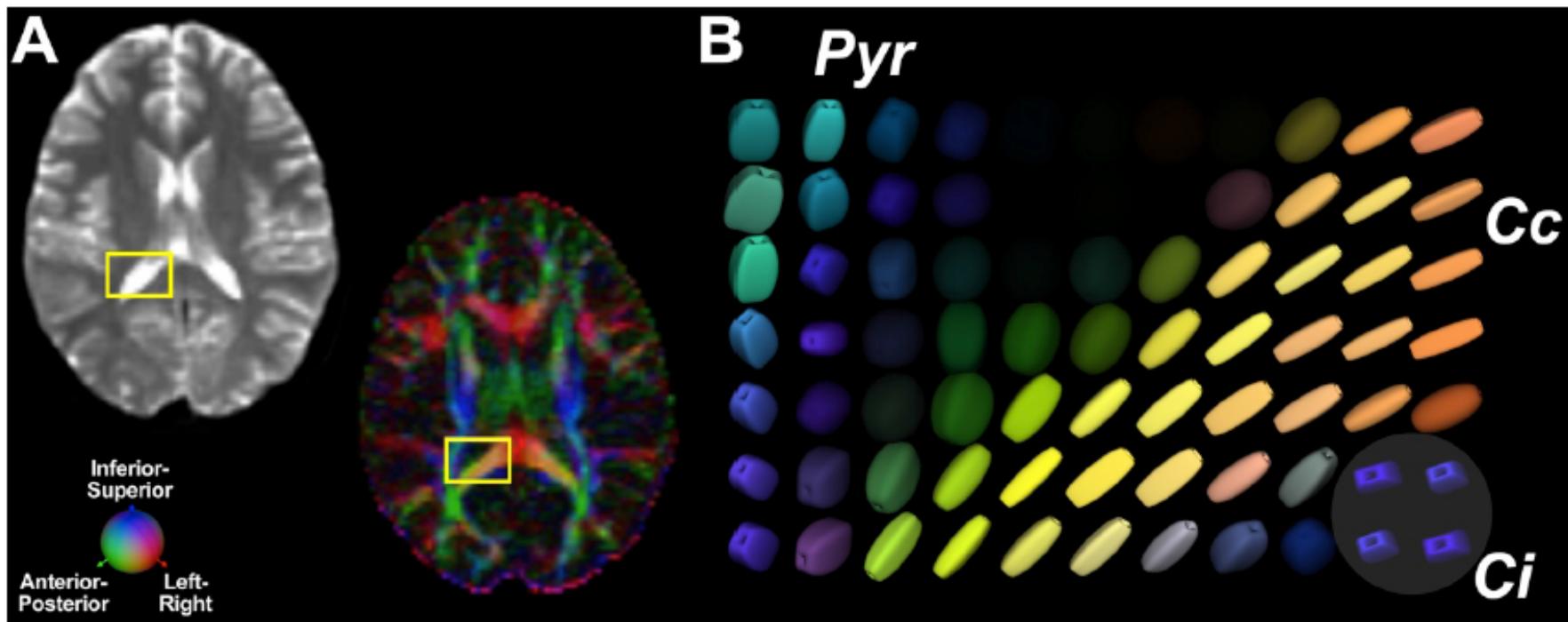


Figura 1. (A) Corte axial T2 e mapa de direcionalidade de fibras a partir de imagem DT-MRI de um indivíduo saudável. Retângulo amarelo denota a região de interesse para a visualização do campo supertorial. (B) Ilustração do campo supertorial indicando os tratos piramidal (Pyr), corpo caloso (Cc) e cíngulo (Ci).

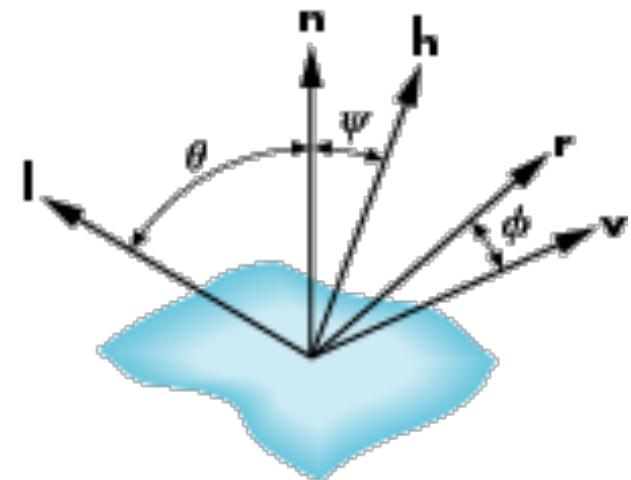
Modelo *Blinn-Phong*

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

$$L = k_d I_d \max(0, \mathbf{l} \cdot \mathbf{n}) + k_s I_s \max(0, \mathbf{n} \cdot \mathbf{h})^\beta + k_a I_a$$

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

$$\mathbf{h} = (\mathbf{l} + \mathbf{v}) / |\mathbf{l} + \mathbf{v}|$$



Atenuação por distância

- Na atenuação por distância, a , b e c são os coeficientes e d é distância entre posição da luz e do vértice em questão (default: $a=1$, $b=c=0$)

$$\text{atenuação} = \frac{1}{a + bd + cd^2}$$

Especificando fontes de luz

- Para cada fonte de luz, especificamos arrays RGBA para descrever seus componentes

```
var diffuse = vec4(1.0, 0.0, 0.0, 1.0);
var ambient = vec4(1.0, 0.0, 0.0, 1.0);
var specular = vec4(1.0, 0.0, 0.0, 1.0);
var light_pos = vec4(1.0, 2.0, 3.0, 1.0);
```

Especificando materiais

- As propriedades materiais devem estar associadas com o modelo de luz escolhido e de acordo com cada objeto

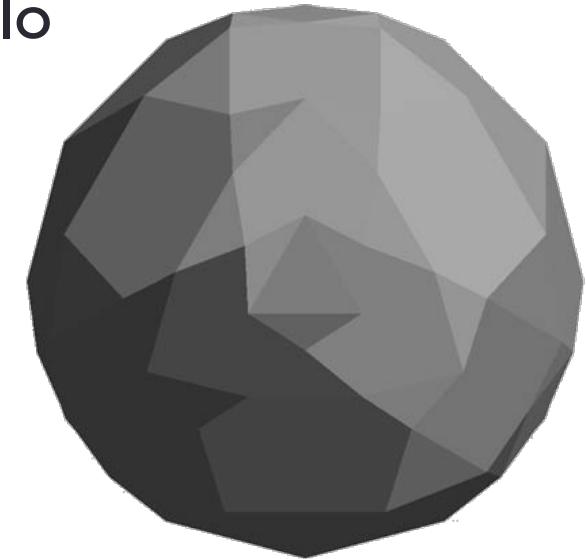
```
var materialAmbient = vec4( 1.0, 0.0, 1.0, 1.0 );
var materialDiffuse = vec4( 1.0, 0.8, 0.0, 1.0);
var materialSpecular = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialShininess = 100.0;
```

Processo de tonalização

- A tonalização por vértice requer que cálculos sejam feitos vértice a vértice
 - Cores dos vértices são enviados ao *vertex shader* como atributos
 - Ou podemos enviar parâmetros da luz ao *vertex shader* e deixá-lo calcular a cor do vértice
- Por default, cores dos vértices são interpolados se passados ao *fragment shader* utilizando **varying**

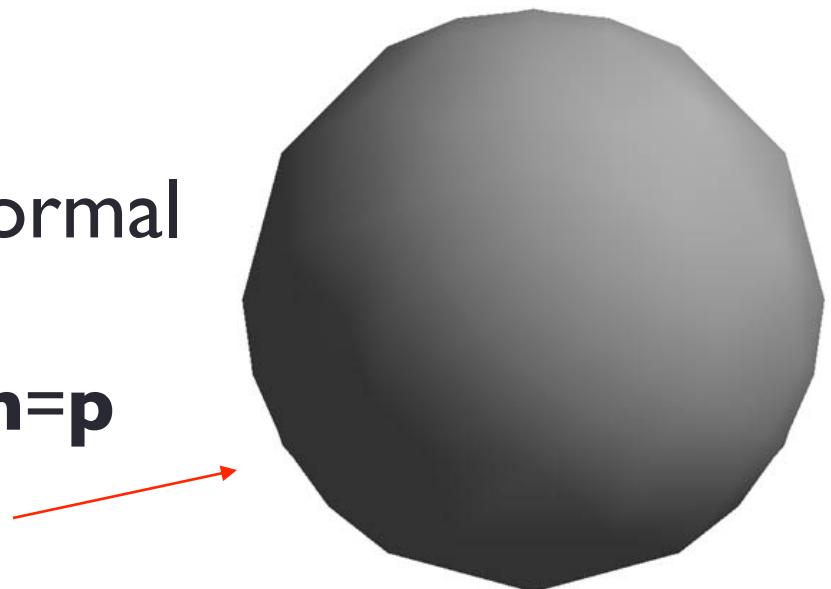
Vetor normal

- Triângulos possuem somente uma normal
 - Tonalidades nos vértices calculados pelo modelo *Blinn-Phong* são quase os mesmos
 - Idêntico para um observador distante (default) se não existir componente especular
- Gostaríamos de especificar diferentes normais para cada vértice
 - Embora não seja matematicamente correto



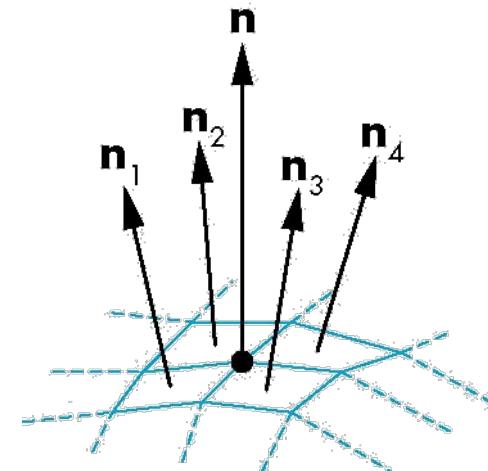
Tonalização suave

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



Tonalização de malhas

- No exemplo anterior, podemos determinar a normal analiticamente
- Mas 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 *Blinn-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 *Blinn-Phong* em cada fragmento

Comparação



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

Comparação

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

Shaders de tonalização por vértice

```
// vertex shader
attribute vec4 vPosition;
attribute vec4 vNormal;
varying vec4 fColor;
uniform vec4 ambientProduct, diffuseProduct,
           specularProduct;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;
uniform float shininess;

...
```

Shaders de tonalização por vértice

```
vec3 pos = modelViewMatrix * vPosition.xyz;  
vec3 light = lightPosition.xyz;  
vec3 L = normalize( pos - light );  
vec3 E = normalize( pos );  
vec3 H = normalize( L + E );  
  
// Transform vertex normal into eye coordinates  
vec3 N = normalize( (modelViewMatrix *  
    vNormal).xyz );  
  
// Compute terms in the illumination equation
```

Shaders de tonalização por vértice

```
// Compute terms in the illumination equation
vec4 ambient = AmbientProduct;

float Kd = max( dot(L, N), 0.0 );
Vec4 diffuse = Kd * DiffuseProduct;
float Ks = pow( max(dot(N, H), 0.0), Shininess );
Vec4 specular = Ks * SpecularProduct;

gl_Position = projectionMatrix * modelViewMatrix *
              vPosition;
fColor = ambient + diffuse + specular;
fColor.a = 1.0;
}
```

Shaders de tonalização por vértice

```
// fragment shader
precision mediump float;

varying vec4 fColor;

void main()
{
    gl_FragColor = fColor;
}
```

Shaders de tonalização por fragmento

```
// vertex shader

attribute vec4 vPosition;
attribute vec4 vNormal;
varying vec3 N, L, E;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;
```

...

Shaders de tonalização por fragmento

```
// vertex shader

void main()
{
    vec3 pos = modelViewMatrix * vPosition.xyz;
    vec3 light = lightPosition.xyz;
    L = normalize( pos - light );
    E = normalize( pos );
    N = normalize( (modelViewMatrix * vNormal).xyz );
    gl_Position = projectionMatrix * modelViewMatrix *
                  vPosition;
};
```

Shaders de tonalização por fragmento

```
// fragment shader
precision mediump float;

uniform vec4 ambientProduct;
uniform vec4 diffuseProduct;
uniform vec4 specularProduct;
uniform float shininess;
varying vec3 N, L, E;

void main()
{
```

Shaders de tonalização por fragmento

```
vec4 fColor;
vec3 H = normalize( L + E );
vec4 ambient = ambientProduct;
float Kd = max( dot(L, N), 0.0 );
vec4 diffuse = Kd * diffuseProduct;
float Ks = pow( max(dot(N, H), 0.0), shininess );
vec4 specular = Ks * specularProduct;

fColor = ambient + diffuse +specular;
fColor.a = 1.0;

gl_FragColor = fColor;
```

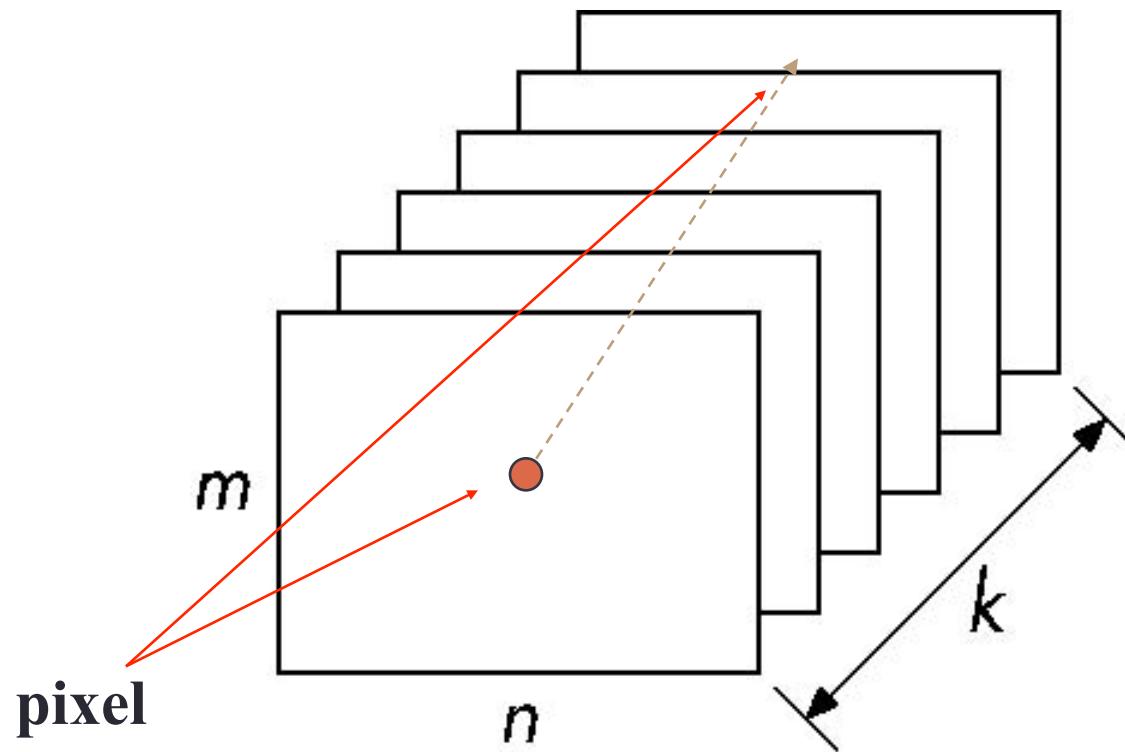
}

Exemplo com *teapot*

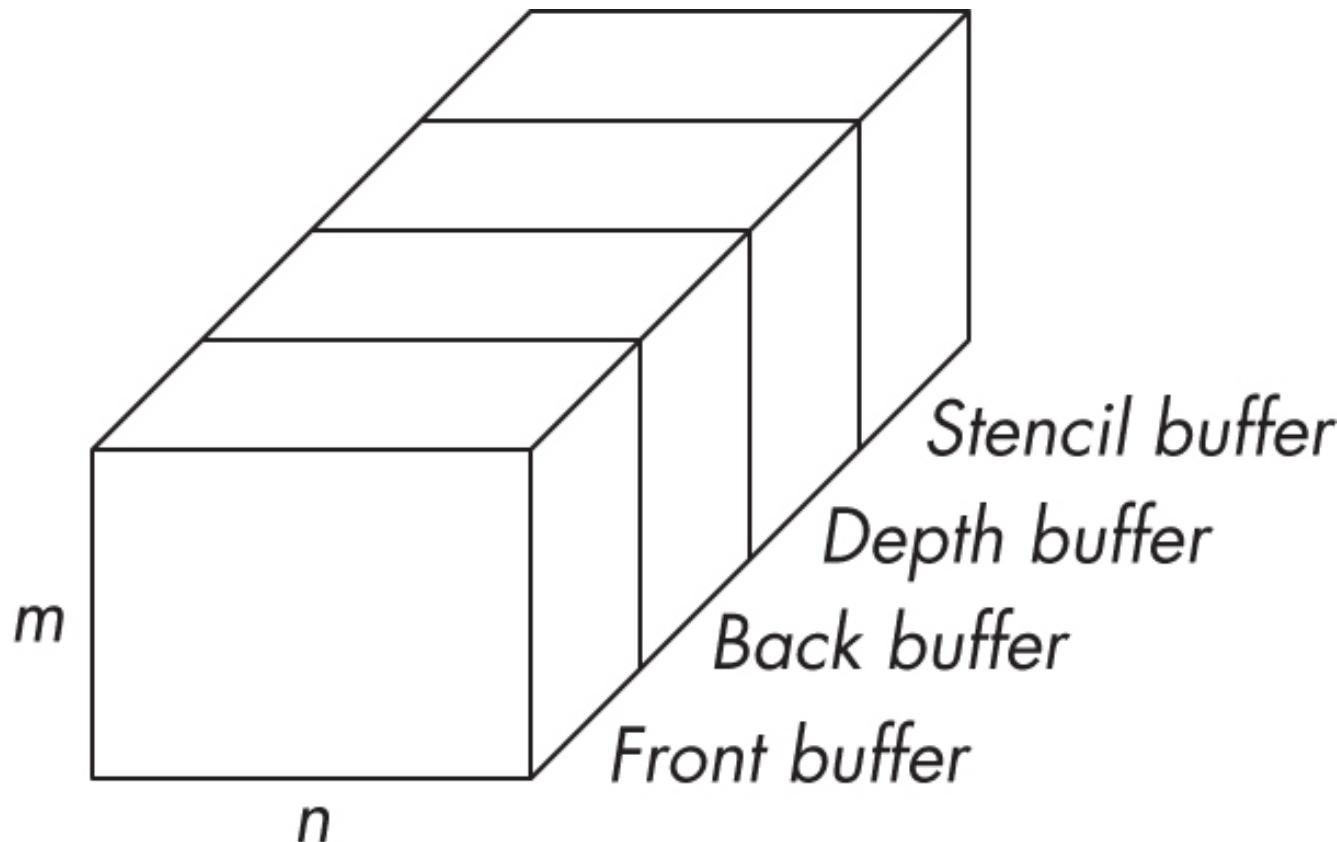


Buffer

Definimos um buffer por sua resolução espacial ($n \times m$) e sua profundidade (ou precisão) k , número de bits/pixel



WebGL Frame Buffer



Onde estão estes buffers ?

- HTML5 Canvas
 - Inclui por default buffers *front* e *back*
 - Sob controle do sistema de janelas
 - Fisicamente na placa gráfica
- *Depth buffer* também está na placa gráfica
- *Stencil buffer*
 - Guarda máscaras
- A maioria dos buffers RGBA possui 8 bits por componente
 - Os mais recentes são do tipo *float* (IEEE)

Outros Buffers

- O OpenGL desktop tinha outros buffers
 - *Auxiliary color buffers*
 - *Accumulation buffer*
 - Agora deprecados
- As GPUs possuem memória localizada na placa de vídeo
 - *Texture buffers*
 - *Off-screen buffers*
 - Não estão sob o controle do sistema de janelas e podem ser *floating point*

Imagens

- Conteúdo do *frame buffer* não possui formato
 - Usualmente RGB ou RGBA
 - Um byte por componente
 - Sem compressão
- Padrões de formatos de imagem
 - jpeg, gif, png
- WebGL não possui funções de conversão
 - Mas entende estes formatos para utilização como texturas

Renderizar em textura

- As GPUs agora incluem uma grande quantidade de memória de textura que podemos utilizar
- Vantagem: rápida (e não está sob controle do SO/sistema de janelas)
- Utilizando *frame buffer objects* (FBOs) podemos renderizar uma cena em uma memória de textura e depois lê-la
 - Processamento de imagens
 - GPGPU

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

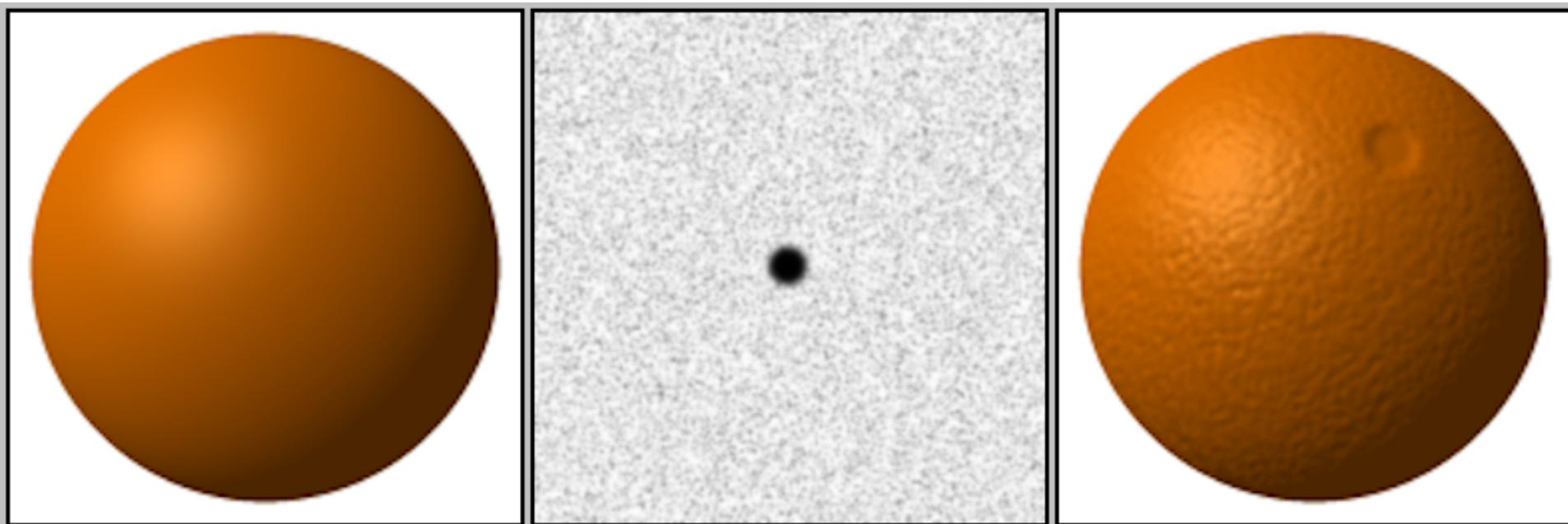
Modelando uma laranja

- 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 objeto real
 - 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”)

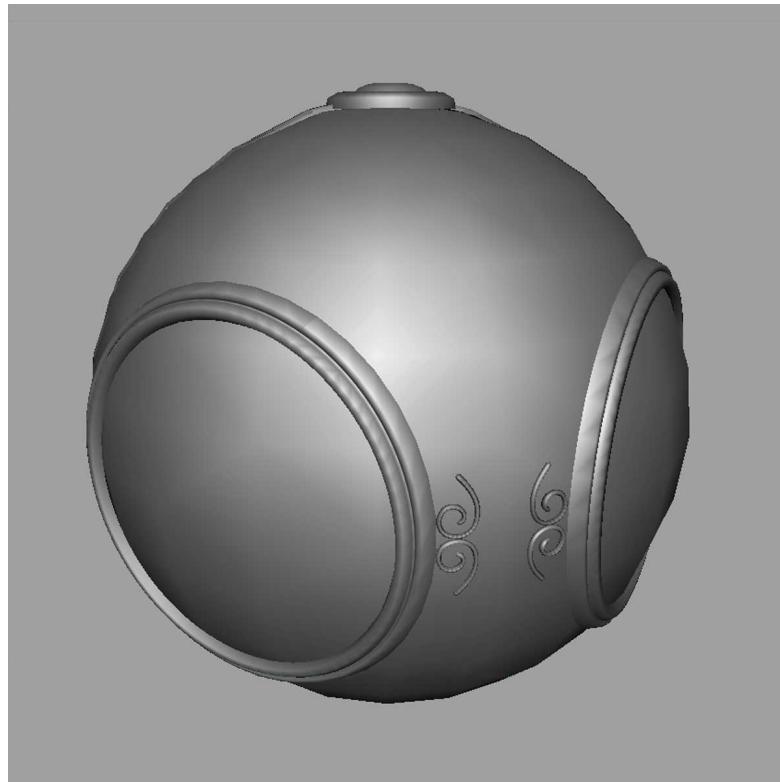
Exemplo



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

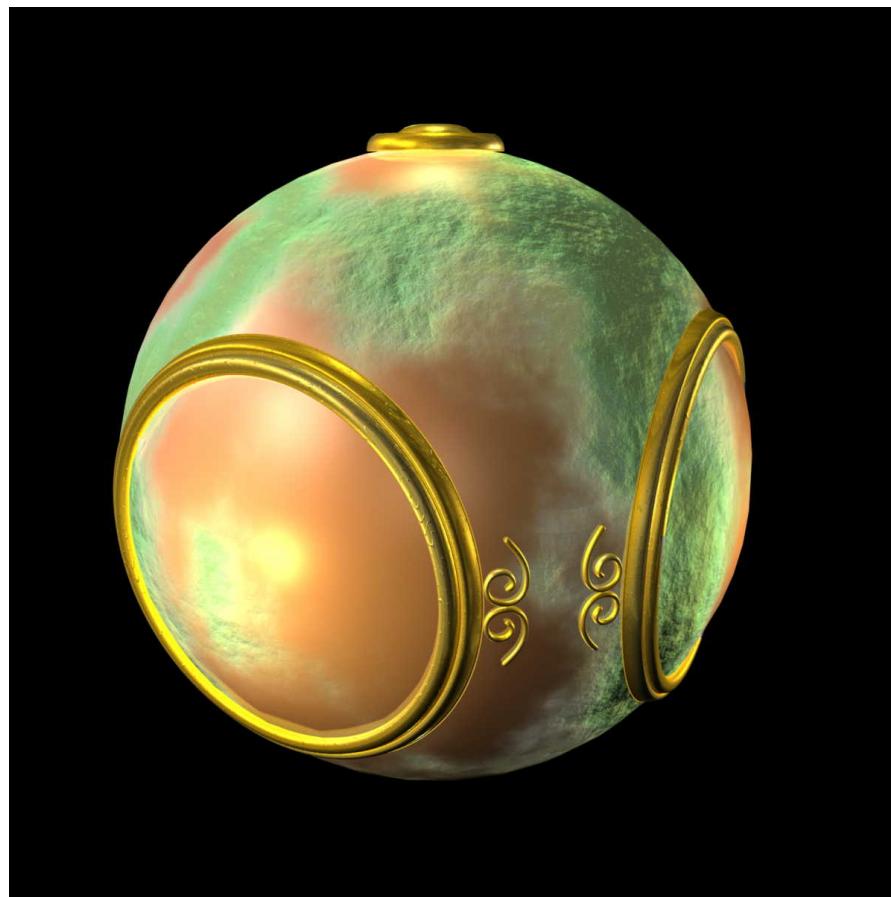


mapeamento de textura

Mapeamento de ambiente

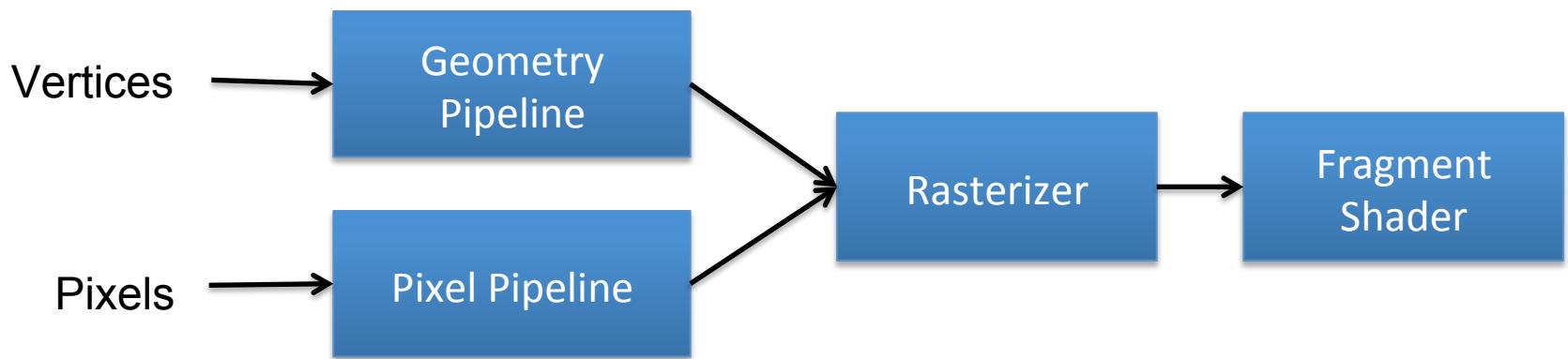


Mapeamento de rugosidade



Mapeamento no pipeline

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



Tarefa

- Leitura livro-texto
 - Shirley and Marschner. Fundamentals of Computer Graphics, CRC Press, 3rd Ed. 2010
 - Capítulos 2-4
- Exercício-programa I