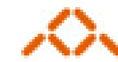
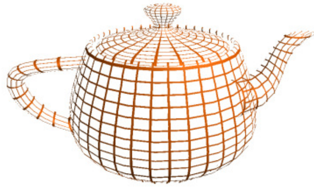


Computação Gráfica

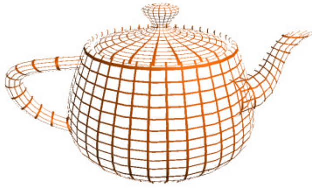
GLSL - Programação de Shaders
Iluminação

António Ramires Fernandes - Computação Gráfica 07/08



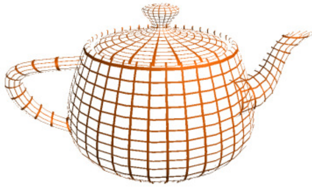
Resumo

- Tipos de Dados, Funções e Controle de Fluxo
- GLSL - Minimal
- GLSL - Cores
- GLSL - Iluminação
  - Direcional
  - Direcional por pixel
  - Posicional por pixel
  - Spot Light por pixel



Tipos de Dados

- Escalares
  - `float` — `float a,b=2.0;`
  - `int` — `int c,d=1;`
  - `bool` — `bool e,f= true;`
- Vectores
  - `vec{2,3,4}` — `vec3 v = vec3(1,2,3);`
  - `ivec{2,3,4}` — `vec3 a = v, b = vec3(4);`
  - `bvec{2,3,4}`
- Matrizes
  - `mat{2,3,4}` — `mat3 m = mat3(1.0); // matriz identidade`
  - `mat3 n = mat3(v,a,b);`



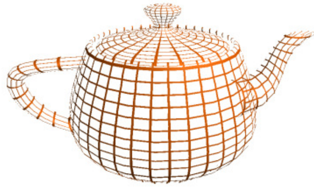
Tipos de Dados

- Selectores para Vectores
  - x,y,z,w
  - r,g,b,a
  - s,t,p,q

```
– vec3 v = vec3(1.0,2.0,3.0);
– float f = v.x;                    // f = 1.0
– float y = v[1];                   // y = 2.0
– vec3 u = v.zyx                    // u = (3.0,2.0,1.0)
```

- Selectores para Matrizes

```
– mat2 m = mat2(1.0);        // m = matriz identidade
– vec2 v = m[0];             // v = (1.0,0.0) primeira coluna
– float f = m[0][0]          // f = 1.0
```



Tipos de Dados

- Estruturas
  - Definição

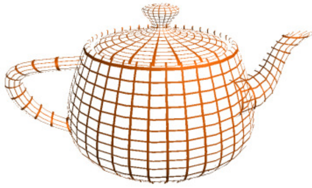
```
struct luz {
     vec3 pos;
     vec3 cor;
};
```

  - Declaração e inicialização

```
luz luz1;
luz luz2 = luz(vec3(1.0,2.0,3.0),vec3(1.0,0.0,0.0));
```

  - Utilização

```
luz1.pos = vec3(1.0,2.0,3.0);
```



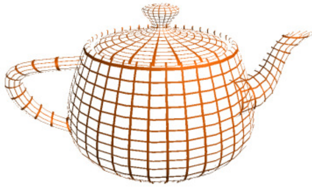
Tipos de Dados

- *Arrays*

  - `vec3 pontos[5];`
  - `vec4 vertices[];`

  - No caso de um array não definir a sua dimensão, o compilador procura o índice mais elevado no shader.

  - Neste caso, os índices têm de ser constantes.



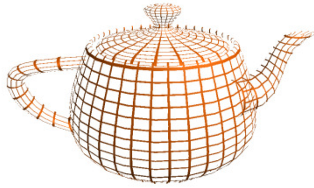
Tipos de Dados

- *Casting*
  - Realizado através de construtores

```
float a = 3.0;
int b = int(a);
```

- Constantes

```
const int texturas = 10;
```



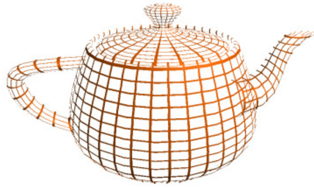
Funções

- Semelhante ao C

```
vec3 f() {
     ...
}
```

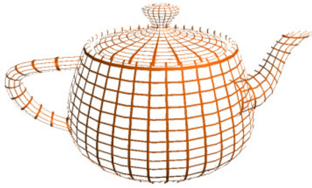
- Tipos dos parâmetros
  - in
  - out
  - inout

```
vec3 f(in vec3 a, out vec3 b) {
     ...
}
```



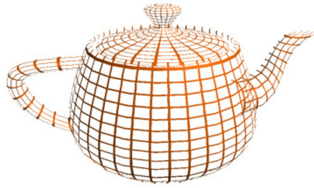
Controle de fluxo

- Instruções iguais a C
  - if else
  - for
  - while
  - do while



Resumo

- Tipos de Dados, Funções e Controle de Fluxo
- GLSL - Minimal
- GLSL - Cores
- GLSL - Iluminação
  - Direcional
  - Direcional por pixel
  - Posicional por pixel
  - Spot Light por pixel



GLSL - Minimal - Vértices

- A funcionalidade fixa, *ff*, determina que um vértice deve ser transformado através da seguinte operação:
  - Vres = Matriz PROJECTION * Matriz MODELVIEW * vertex
- O vertex shader recebe os vértices através da variável:

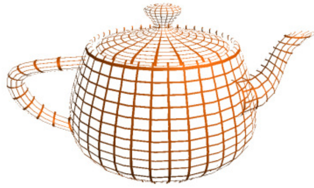
```
attribute vec4 gl_Vertex:
```

- O vertex shader deve transformar o vértice recebido e escrever o resultado em:

```
vec4 gl_Position;
```

- Variáveis de estado do OpenGL acessiveis:

```
uniform mat4 gl_ModelViewMatrix;
uniform mat4 gl_ProjectionMatrix;
uniform mat4 gl_ModelViewProjectionMatrix;
```

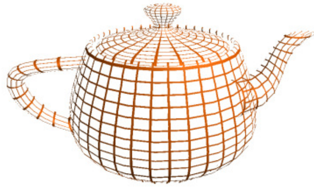


GLSL - Minimal - Vértices

- Para obter uma funcionalidade semelhante à ff, o vertex shader deverá então calcular `gl_Position` utilizando uma das seguintes opções:

```
gl_Position = gl_ProjectionMatrix *
                  (gl_ModelViewMatrix * gl_Vertex);

gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;

gl_Position = ftransform();
```

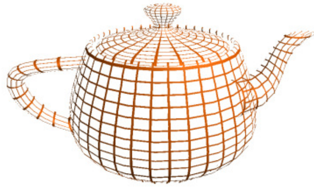
- Esta última alternativa garante a funcionalidade exacta da ff.



GLSL - Minimal - Vértices

- Vertex Shader

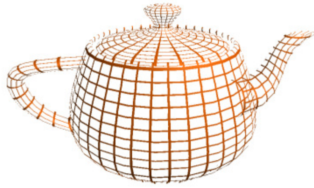
```
void main (void) {

    gl_Position = ftransform();
}
```



GLSL - Minimal - Cor Constante

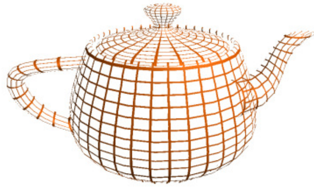
- O Fragment shader é responsável por escrever a variável gl_FragColor.

```
void main(void) {
   gl_FragColor = vec4(1,0,0,1);
}
```



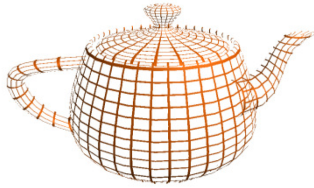
Resumo

- Tipos de Dados, Funções e Controle de Fluxo
- GLSL - Minimal
- **GLSL - Cores**
- GLSL - Iluminação
  - Direcional
  - Direcional por pixel
  - Posicional por pixel
  - Spot Light por pixel



GLSL - Cores

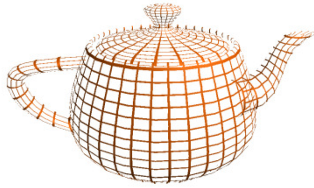
- Um vertex shader calcula valores por vértice.
- Dentro de um Fragment shader precisamos de ter acesso aos valores interpolados para realizar os cálculos para cada fragmento.
- O pipeline gráfico utiliza os valores processados para cada vértice no vertex shader, e interpola-os de acordo com a construção de primitivas (utilizando a informação de conectividade).
- Estas variáveis interpoladas têm o qualificador *varying*.



GLSL - Cores

- Código OpenGL

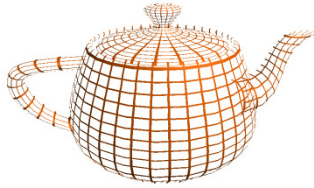
```
glBegin(GL_QUADS);
     glColor3f(1,0,0);
     glVertex3f(-1,-1,0);

     glColor3f(0,1,0);
     glVertex3f(1,-1,0);

     glColor3f(0,0,1);
     glVertex3f(1,1,0);

     glColor3f(1,1,0);
     glVertex3f(-1,1,0);
glEnd();
```



GLSL - Cores

- Variáveis de estado acessíveis nos shaders:

```
attribute vec4 gl_Color; // só vertex shader

struct gl_MaterialParameters {
    vec4 emission;   // Ecm
    vec4 ambient;    // Acm
    vec4 diffuse;    // Dcm
    vec4 specular;   // Scm
    float shininess; // Srm
};
uniform gl_MaterialParameters gl_FrontMaterial;
uniform gl_MaterialParameters gl_BackMaterial;
```



GLSL - Cores

- Vertex Shader obtem cor especificada na aplicação (glColor) em:

```
attribute vec4 gl_Color;
```

- Vertex Shader escreve em:

```
varying vec4 gl_FrontColor;
varying vec4 gl_BackColor;
```

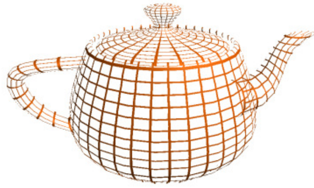
- Fragment Shader lê cor interpolada em:

```
varying vec4 gl_Color;
```

  - Nota: não é a mesma variável que está acessível no vertex shader. Esta variável é interpolada pelo OpenGL a partir das variáveis `gl_FrontColor` e `gl_BackColor`.

- Fragment Shader escreve em:

```
vec4 gl_FragColor
```



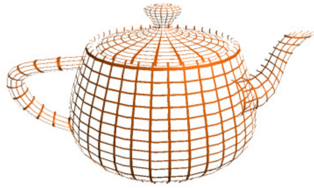
GLSL - Cores

- Exemplo

```
// Aplicação OpenGL

glBegin(GL_QUADS);
        glColor3f(1,0,0);
        glVertex3f(-1,-1,0);
        glColor3f(0,1,0);
        glVertex3f(1,-1,0);
        glColor3f(0,0,1);
        glVertex3f(1,1,0);
        glColor3f(1,1,0);
        glVertex3f(-1,1,0);
glEnd();
```

```
// Vertex Shader

void main(void) {

        gl_FrontColor = gl_Color;
        gl_Position = ftransform();
}
```

```
// Fragment Shader

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



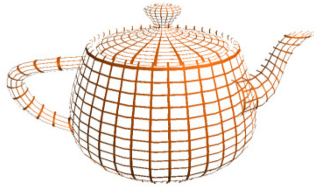
GLSL - Cores

- Exemplo

```
// Aplicação OpenGL
glMaterialfv(GL_FRONT, GL_AMBIENT,
                   blueMaterial);

glBegin(GL_QUADS);
          glColor3f(1,0,0);
          glVertex3f(-1,-1,0);
          glColor3f(0,1,0);
          glVertex3f(1,-1,0);
          glColor3f(0,0,1);
          glVertex3f(1,1,0);
          glColor3f(1,1,0);
          glVertex3f(-1,1,0);
glEnd();
```

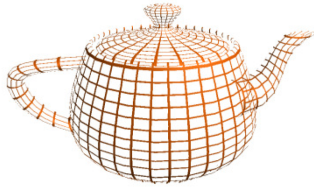
```
// Vertex Shader

void main(void) {

          gl_FrontColor =
                    gl_FrontMaterial.ambient;
          gl_Position = ftransform();
}
```

```
// Fragment Shader

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



Resumo

- Tipos de Dados, Funções e Controle de Fluxo
- GLSL - Minimal
- GLSL - Cores
- GLSL - Iluminação
  - Direcional
  - Direcional por pixel
  - Posicional por pixel
  - Spot Light por pixel



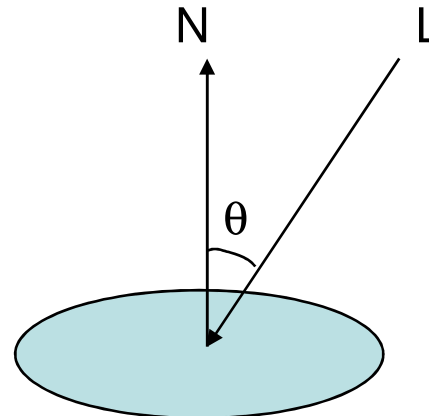
GLSL - Iluminação

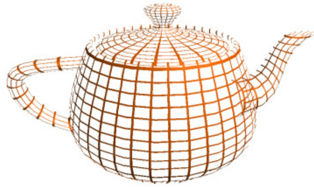
- Reflexão difusa (Lambert)
  - A intensidade de um objecto é proporcional ao ângulo entre a direcção da luz e a normal do objecto.

Intensidade da luz

$$I = I_p * K_d * \cos(\theta)$$

coeficiente de reflexão difusa





GLSL - Iluminação

- Adicionando a componente ambiente:

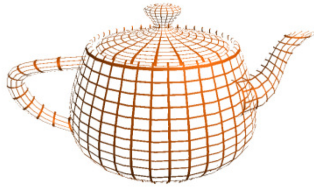
$$I = I_a * K_a + I_p * K_d * \cos(\theta)$$

- Caso ambos os vectores estejam normalizados pode-se substituir o coseno pelo produto interno,

$$I = I_a * K_a + I_p * K_d * (N . L)$$

Produto Interno N.L = $n_x * l_x + n_y * l_y + n_z * l_z$

Nota: só se consideram valores positivos do produto interno ou coseno



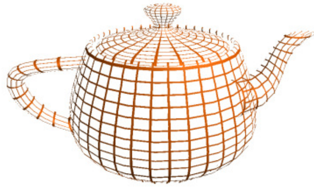
GLSL - Iluminação

- Código OpenGL:

```
glLoadIdentity();
gluLookAt(0.0, 0.0, 5.0,    0.0,0.0,0.0,      0.0,1.0,0.0);
glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
glutSolidTeapot(1);
```

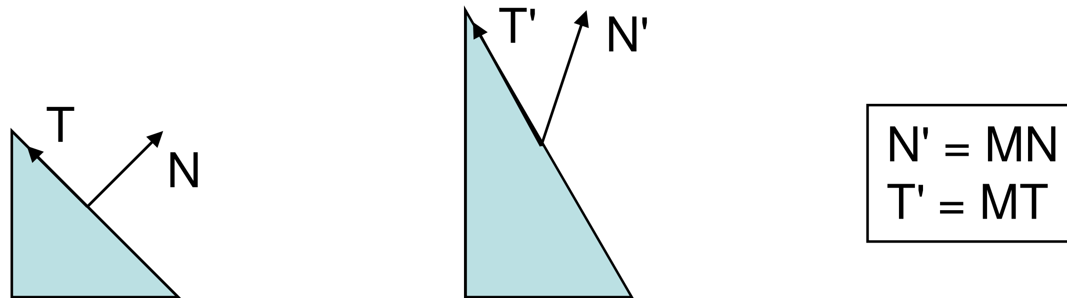
- A especificação do OpenGL diz que a posição da luz é transformada pela matriz ModelView na altura da sua especificação, ou seja quando um shader acede à posição da luz, está a aceder à posição da luz no espaço da câmara.

- É necessário transformar as normais para o espaço câmara para poder calcular o coseno do ângulo entre a normal e a direcção da luz.

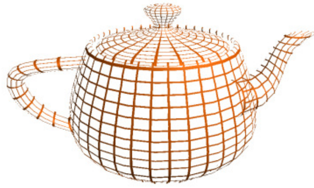


GLSL - Iluminação

- Transformação da Normal
  - Caso a matriz ModelView não seja ortogonal, a normal pode ser incorrectamente transformada.



  - Precisamos de encontrar uma matriz G que transforme N de forma a que N'.T' = 0 (prod. interno)



GLSL - Iluminação

- Transformação da Normal

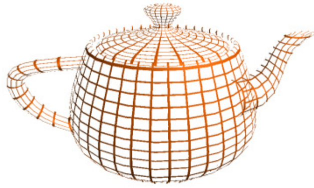
  $N'.T' = (GN).(MT) = 0$

  $(GN).(MT) = (GN)^T (MT)$
  $= N^T G^T M T$

  - Sabemos que $N^T T = 0$, logo a equação é satisfeita quando $G^T M = I$, ou seja

  $G = (M^{-1})^T$

  - Se M for ortogonal $M^{-1} = M^T$, logo $G = M$

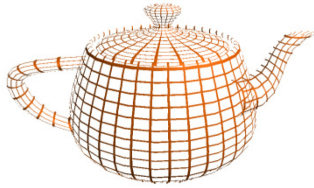


GLSL - Iluminação

- GLSL fornece a matriz G: gl_NormalMatrix
- Sendo assim o cálculo da normal dentro de um vertex shader é realizado através da seguinte operação:

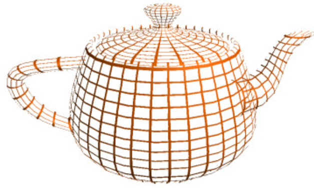
```
vec3 n;
n = normalize(gl_NormalMatrix * gl_Normal);
```

- Sendo `gl_Normal` a normal especificada para o vértice na aplicação OpenGL através da função glNormal.
- A normalização da normal num vertex shader pode ser dispensada caso haja garantia de que ambas as condições se verifiquem:
  - a normal especificada na aplicação OpenGL está normalizada
  - a matriz ModelView é ortogonal, i.e., que só foram aplicadas rotações e translações, mas não escalas.



Resumo

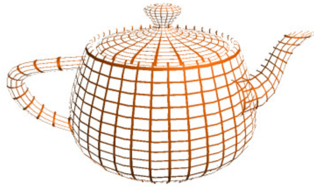
- Tipos de Dados, Funções e Controle de Fluxo
- GLSL - Minimal
- GLSL - Cores
- GLSL - Iluminação
  - Direccional
  - Direccional por pixel
  - Posicional por pixel
  - Spot Light por pixel



GLSL - Iluminação Direcional

- Variáveis de estado relevantes

```
struct gl_LightSourceParameters {
    vec4 ambient; // Acli
    vec4 diffuse; // Dcli
    vec4 specular; // Scli
    vec4 position; // Ppli
    vec4 halfVector; // Derived: Hi
    ...
};

uniform gl_LightSourceParameters gl_LightSource[gl_MaxLights];

struct gl_LightModelParameters {
   vec4 ambient; // Acs
};

uniform gl_LightModelParameters gl_LightModel;
```



GLSL - Iluminação Direcional

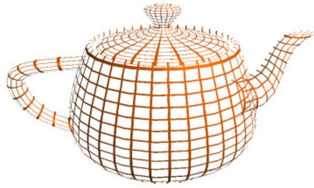
- $I = I_a * K_a + I_p * K_d * (N . L)$

Vertex Shader

```
void main(){

    vec3 normal,lightDir;
    vec4 diffuse,ambient;
    float intensity;

    lightDir = normalize(vec3(gl_LightSource[0].position));
    normal = normalize(gl_NormalMatrix * gl_Normal);

    intensity = max(dot(lightDir,normal),0.0);
    diffuse = gl_LightSource[0].diffuse * gl_FrontMaterial.diffuse;
    ambient = gl_LightSource[0].ambient * gl_FrontMaterial.ambient;

    gl_FrontColor = ambient + diffuse*intensity;
    gl_Position = ftransform();
}
```

Fragment Shader

```
void main(){

      gl_FragColor = gl_Color;
}
```

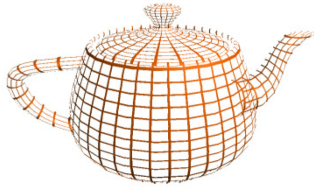


GLSL - Iluminação Direcional

- Incorporar o termo ambiente global e emissivo

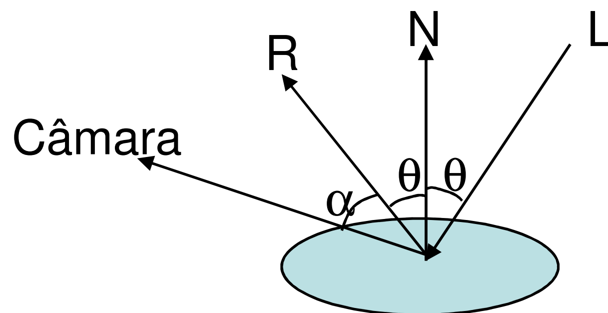
$$I = I_a * K_a + I_p * K_d * (N . L) + \mathbf{K_e} + \mathbf{G_a} * \mathbf{K}$$

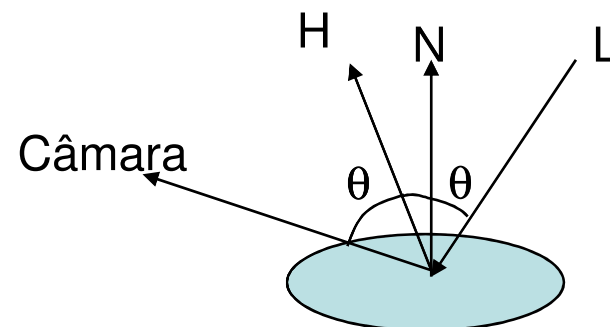
```
void main(){



    ...
    gl_FrontColor = ambient + diffuse*intensity;
    gl_FrontColor += gl_FrontMaterial.ambient * gl_LightModel.ambient;
    gl_FrontColor += gl_FrontMaterial.emissive;
    gl_Position = ftransform();
}
```

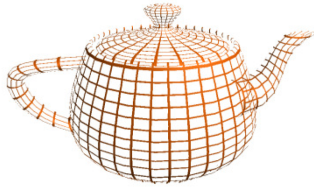


GLSL - Iluminação Direcional

- Incorporar o termo especular
  - Especular = $I_s * K_s * (R.Câmara)^s$
  - Especular = $I_s * K_s * (H.N)^s$





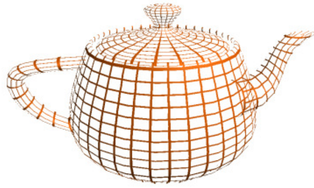


GLSL - Iluminação Direcional

- Termo Especular no Vertex Shader

  - Especular = $I_s * K_s * (H.N)^s$

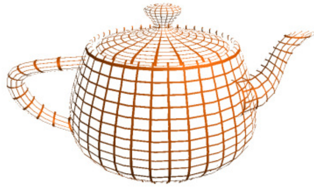
```
if (intensity > 0) {
          aux = max(dot(vec3(gl_LightSource[0].halfVector),normal),0.0);
          shininess = pow(aux,gl_FrontMaterial.shininess);
          specular = gl_LightSource[0].specular * gl_FrontMaterial.specular;
          specular *= shininess;
          gl_FrontColor += specular;
}
```



GLSL - Iluminação Direcional

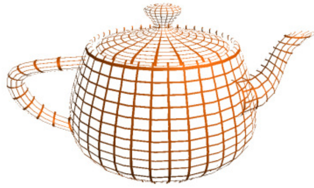
Vertex Shader

```
void main(){

    vec3 normal,lightDir;
    vec4 diffuse,ambient,specular;
    float intensity, shininess,aux;

    lightDir = normalize(vec3(gl_LightSource[0].position));
    normal = normalize(gl_NormalMatrix * gl_Normal);

    intensity = max(dot(lightDir,normal),0.0);
    diffuse   = gl_LightSource[0].diffuse * gl_FrontMaterial.diffuse;
    ambient = gl_LightSource[0].ambient * gl_FrontMaterial.ambient;

    gl_FrontColor = ambient + diffuse*intensity;
    gl_FrontColor += gl_FrontMaterial.ambient * gl_LightModel.ambient;
    gl_FrontColor += gl_FrontMaterial.emission;

    if (intensity > 0.0) {
          aux = max(dot(vec3(gl_LightSource[0].halfVector),normal),0.0);
          shininess = pow(aux,gl_FrontMaterial.shininess);
          specular = gl_LightSource[0].specular * gl_FrontMaterial.specular;
          specular *= shininess;
          gl_FrontColor += specular;
    }
    gl_Position = ftransform();
}
```



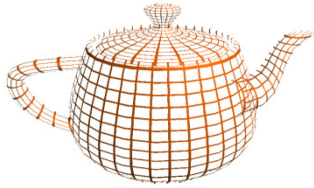
Resumo

- Tipos de Dados, Funções e Controle de Fluxo
- GLSL - Minimal
- GLSL - Cores
- GLSL - Iluminação
  - Direcional
  - Direcional por pixel
  - Posicional por pixel
  - Spot Light por pixel



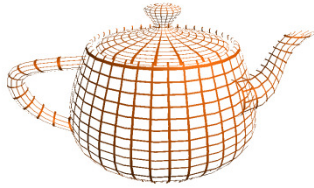
GLSL - Iluminação Direcional

- Shaders Apresentados:
  - Vertex Shader calcula cores baseado na iluminação por vértice
  - Fragment Shader utiliza os valores interpolados das cores para cada vértice
- Passo Seguinte:
  - Cor calculada por pixel



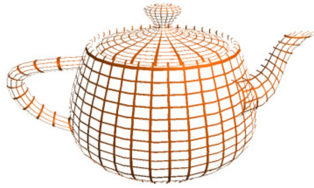
GLSL - Iluminação Direcional

- Para calcular a cor da superfície iluminada por pixel precisamos para cada ponto da superfície (não só para os vértices) :
  - Normal
  - Vector com direcção do "olhar"
  - Direcção da luz
- Estes valores podem ser calculados por vértice e passados ao fragment shader que acederá à interpolação dos valores para cada primitiva.



GLSL - Iluminação Direcional

- A comunicação entre o vertex shader e o fragment shader realiza-se através de duas formas:
  - variáveis pré-definidas (caso das cores);
  - variáveis definidas nos shaders pelo programador e declaradas como varying.



GLSL - Iluminação Direcional

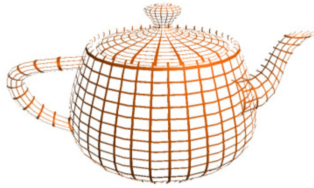
- Vertex Shader

```
varying vec3 normal, lightDir, ecPos;

void main()
{
    lightDir = normalize(vec3(gl_LightSource[0].position));
    normal = normalize(gl_NormalMatrix * gl_Normal);
    ecPos = vec3(gl_ModelViewMatrix * gl_Vertex) ;

    gl_Position = ftransform();
}
```

Variáveis para interpolação

Porque não normalizar a normal somente no Fragment Shader?

- O vector com a direcção do olhar é calculado transformando o vértice para o espaço câmara. Neste espaço a câmara encontra-se na origem, e portanto o vector é gl_Vertex - camPos = gl_Vertex.



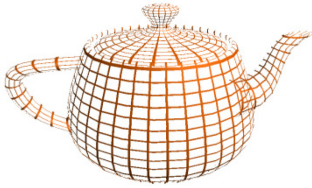
GLSL - Iluminação Direcional

- Fragment Shader

```
varying vec3 normal, lightDir, ecPos;

void main()
{
   vec4 diffuse,ambient,specular,color;
   float intensity,shininess;
   vec3 n,hv,ec;

   n = normalize(normal);
   intensity = max(dot(lightDir,n),0.0);
       ...
```

Porquê normalizar novamente?

Porque não normalizar a direcção da luz?

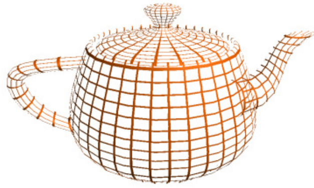


GLSL - Iluminação Direcional

- Fragment Shader (2)

```
...
diffuse = gl_LightSource[0].diffuse * gl_FrontMaterial.diffuse;
ambient = gl_LightSource[0].ambient * gl_FrontMaterial.ambient;

color = ambient + diffuse*intensity;
color += gl_FrontMaterial.ambient * gl_LightModel.ambient;
color += gl_FrontMaterial.emission;
...
```

- Aqui realiza-se o cálculo da cor com as componentes difusa, ambiente e emissiva.

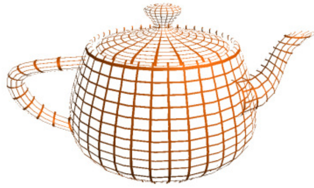


GLSL - Iluminação Direcional

- Fragment Shader (3)

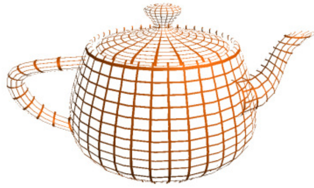
```
    ...
    if (intensity > 0.0) {
           ec = normalize(-ecPos);
           hv = normalize(ec+lightDir);
           shininess = pow(max(dot(hv,n),0.0),gl_FrontMaterial.shininess);
           specular = shininess * gl_LightSource[0].specular * gl_FrontMaterial.specular;
           color += specular;
    }

    gl_FragColor = color;
}
```

- É necessário normalizar o vector com a direcção do olhar pela mesma razão que se normalizou o vector normal.



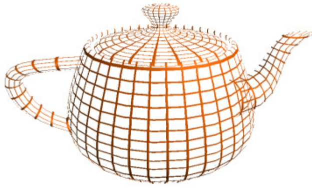
Resumo

- Tipos de Dados, Funções e Controle de Fluxo
- GLSL - Minimal
- GLSL - Cores
- GLSL - Iluminação
  - Direcional
  - Direcional por pixel
  - Posicional por pixel
  - Spot Light por pixel



GLSL - Iluminação Posicional

- A diferença entre a luz direcional e a luz posicional reduz-se aos seguintes termos:
  - Existência de um factor de atenuação
  - Direcção da luz varia por vértice (e por pixel)



GLSL - Iluminação Posicional

- Equação OpenGL

$$cor = m_e + lm_a + m_a + f_{att} * (Amb + Diff + Spec)$$

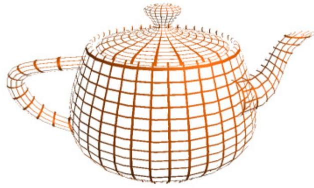
$$fatt = \frac{1}{c + c_l d + c_q d^2}$$

$$Amb = m_a * l_a$$

$$Diff = \max(L \bullet N, 0) * m_d * l_d$$

$$Spec = \max(0, H \bullet N)^{shininess} * ms * ls)$$

- onde:
  - m representa o material
  - l representa as propriedades da luz
  - f_{att} é o factor de atenuação relativamente à distância da luz ao objecto
  - lm indica as propriedades do modelo de luz (ambiente)
  - L é o vector unitário que aponta do ponto do objecto à posição da luz
  - N é a normal do ponto do objecto
  - H representa o half-vector

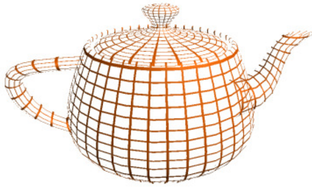


GLSL - Iluminação Posicional

- Vertex Shader (iluminação por pixel)

```
varying vec3 normal, ecPos;

void main()
{

   normal = normalize(gl_NormalMatrix * gl_Normal);
   ecPos = vec3(gl_ModelViewMatrix * gl_Vertex) ;

   gl_Position = ftransform();
}
```

- O cálculo da direcção da luz passa para o fragment shader pois esta depende da localização do fragmento



GLSL - Iluminação Posicional

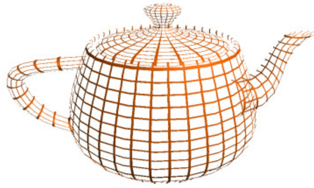
- Fragment Shader
  - Cálculo do factor de atenuação

$$att = \frac{1}{c + c_l d + c_q d^2}$$

Variáveis de estado relevantes

```
struct gl_LightSourceParameters {
    ...
    float constantAttenuation;
    float linearAttenuation;
    float quadraticAttenuation;
};
```

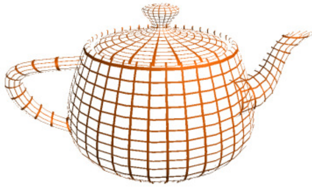
```
lightDir = vec3(gl_LightSource[0].position)-ecPos;
d = length(lightDir);
att = 1 / (gl_LightSource[0].constantAttenuation +
                gl_LightSource[0].linearAttenuation * d +
                gl_LightSource[0].quadraticAttenuation * d * d);
```



GLSL - Iluminação Posicional

- Fragment Shader

```
varying vec3 normal, ecPos;

void main()
{
   vec4 diffuse,ambient,specular,color;
   float intensity,shininess,d,att;
   vec3 n,hv,ec , lightDir;

   lightDir = vec3(gl_LightSource[0].position)-ecPos;
   n = normalize(normal);

       ...
```

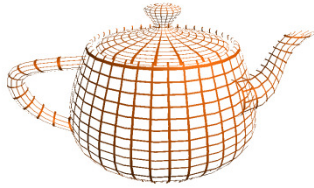


GLSL - Iluminação Posicional

- Fragment Shader (2)

```
d = length(lightDir);
att = max(1.0,1.0 / (gl_LightSource[0].constantAttenuation +
        gl_LightSource[0].linearAttenuation * d +
        gl_LightSource[0].quadraticAttenuation * d * d));

diffuse = gl_LightSource[0].diffuse * FrontMaterial.diffuse;
ambient = gl_LightSource[0].ambient * gl_FrontMaterial.ambient;

lightDir = normalize(lightDir)
intensity = max(dot(lightDir,n),0.0);
color = (ambient + diffuse*intensity) * att;
color += gl_FrontMaterial.ambient * gl_LightModel.ambient;
color += gl_FrontMaterial.emission;
...
```

- *Aqui realiza-se o cálculo da cor com as componentes difusa, ambiente e emissiva, agora com a atenuação a influenciar a componente difusa e ambiente.*

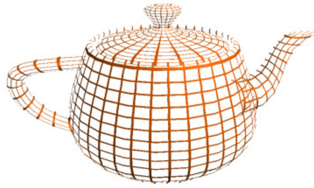


GLSL - Iluminação Posicional

- Fragment Shader (3)

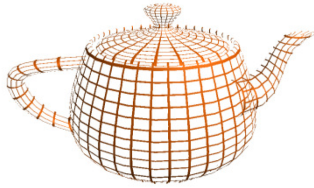
```
    ...
    if (intensity > 0.0) {
        ec = normalize(-ecPos);
        hv = normalize(ec+lightDir);
        shininess = pow(max(dot(hv,n),0.0),gl_FrontMaterial.shininess);
        specular = shininess * gl_LightSource[0].specular * gl_FrontMaterial.specular;
        color += specular * att;
    }

    gl_FragColor = color;
}
```

- O cálculo da cor especular também é influenciado pela atenuação.



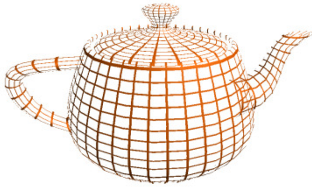
Resumo

- Tipos de Dados, Funções e Controle de Fluxo
- GLSL - Minimal
- GLSL - Cores
- GLSL - Iluminação
  - Direcional
  - Direcional por pixel
  - Posicional por pixel
  - Spot Light por pixel



GLSL - Iluminação Spot Light

- A iluminação com spot lights traz as seguintes novidades:
  - A luz tem uma direcção (para alem da posição)
  - Existe um ângulo que define o cone de luz
- O vertex shader é igual ao caso da luz posicional!



GLSL - Iluminação Spot Light

- Equação OpenGL

$$cor = m_e + lm_a + m_a + f_{att} * \mathbf{spotEffect} * (Amb + Diff + Spec)$$

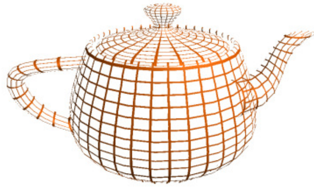
$$f_{att} = \frac{1}{c + c_l d + c_q d^2}$$

$$\mathbf{spotEffect = max(0, L \cdot spotDirection)^{spotExponent}}$$

$$Amb = m_a * l_a$$

$$Diff = \max(L \cdot N, 0) * m_d * l_d$$

$$Spec = \max(0, H \cdot N)^{shininess} * ms * ls)$$



GLSL - Iluminação Spot Light

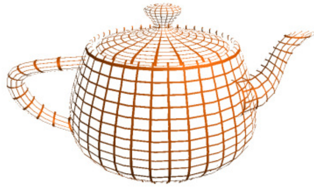
- Fragment Shader
  - Cálculo de spotEffect

s = max(0, -lightDir . spotDirection)
Se (acos(s) > cutoff)
    spotEffect = 0
senão
    spotEffect = $s^{exponent}$

Variáveis de estado relevantes

```
struct gl_LightSourceParameters {
      ...
      vec3 spotDirection; // Sdli
      float spotExponent; // Srli
      float spotCutoff; // Crli
      // (range: [0.0,90.0], 180.0)
      float spotCosCutoff; // Derived: cos(Crli)
      // (range: [1.0,0.0],-1.0)
      ...
};
```

```
spotDot = max(0.0,dot(-lightDir,gl_LightSource[0].spotDirection));
if (spotDot < gl_LightSource[0].spotCosCutoff)
  spotAtt = 0.0;
else
  spotAtt = pow(spotDot,gl_LightSource[0].spotExponent);
```

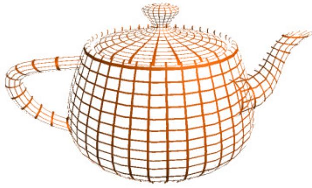


GLSL - Iluminação Spot Light

- Fragment Shader

```
varying vec3 normal, ecPos;

void main()
{
   vec4 diffuse,ambient,specular,color;
   float intensity,shininess,d,att,spotAtt,spotDot;
   vec3 n,hv,ec,lightDir;

   lightDir = vec3(gl_LightSource[0].position)-ecPos;
   n = normalize(normal);

       ...
```

- A primeira parte do fragment shader é igual ao caso anterior das luzes posicionais.

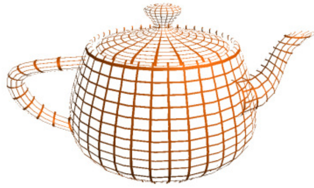


GLSL - Iluminação Spot Light

- Fragment Shader (2)

```
d = length(lightDir);
att = max(1.0,1.0 / (gl_LightSource[0].constantAttenuation +
         gl_LightSource[0].linearAttenuation * d +
         gl_LightSource[0].quadraticAttenuation * d * d));

lightDir = normalize(lightDir);
spotDot = max(0.0,dot(-lightDir,gl_LightSource[0].spotDirection));
if (spotDot < gl_LightSource[0].spotCosCutoff)
      spotAtt = 0.0;
else
      spotAtt = pow(spotDot,gl_LightSource[0].spotExponent);

att *= spotAtt;

diffuse = gl_LightSource[0].diffuse * gl_FrontMaterial.diffuse;
ambient = gl_LightSource[0].ambient * gl_FrontMaterial.ambient;
 ...
```

- *Aqui realiza-se o cálculo da cor com as componentes difusa, ambiente e emissiva, agora com a atenuação a influenciar a componente difusa e ambiente.*

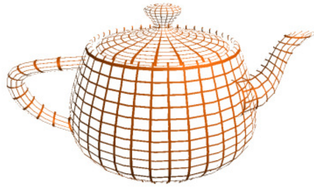


GLSL - Iluminação Spot Light

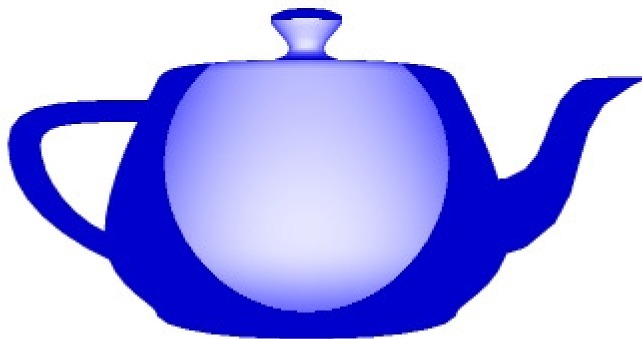
- Fragment Shader (3)

```
    ...
    intensity = max(dot(lightDir,n),0.0);
    color = (ambient + diffuse*intensity) * att;
    color += gl_FrontMaterial.ambient * gl_LightModel.ambient;
    color += gl_FrontMaterial.emission;
    if (intensity > 0.0) {
          ec = normalize(-ecPos);
          hv = normalize(ec+lightDir);
          shininess = pow(max(dot(hv,n),0.0),gl_FrontMaterial.shininess);
          specular = shininess * gl_LightSource[0].specular * gl_FrontMaterial.specular;
          color += specular * att;
    }

    gl_FragColor = color;
}
```

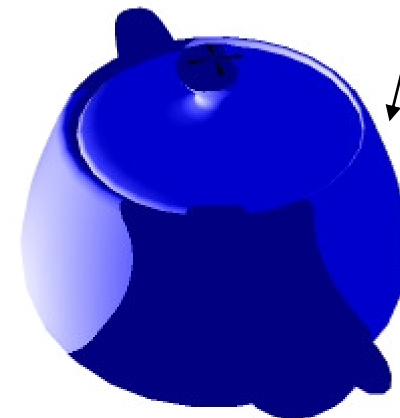
- O cálculo da cor é idêntico ao caso das points lights, tendo em atenção que o factor de atenuação agora também considera o efeito da spotlight.

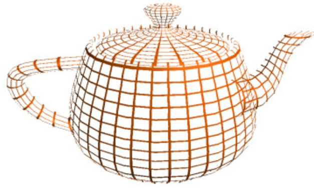


GLSL - Iluminação Spot Light



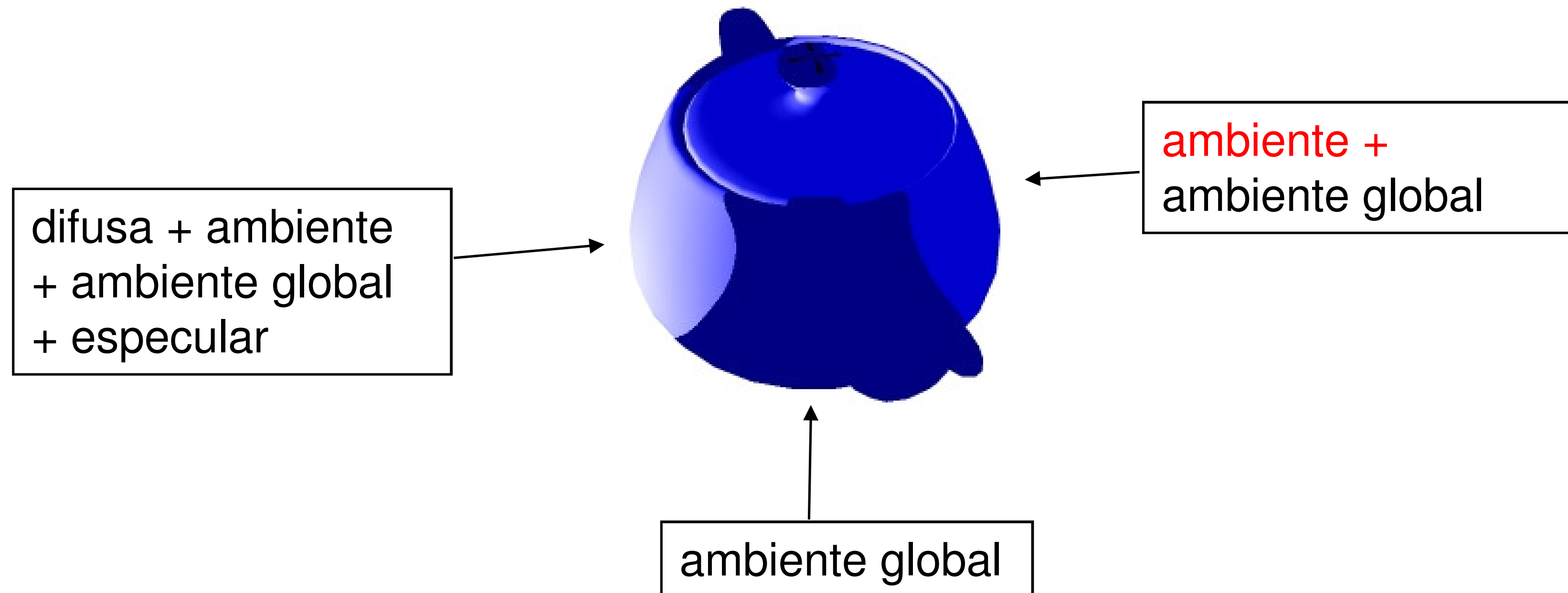
Ooops!

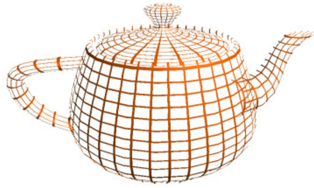




GLSL - Iluminação Spot Light

- Análise das contribuições de luz





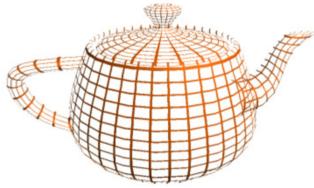
GLSL - Iluminação Spot Light

- O problema deve-se à forma como a luz ambiente é aplicada.

```
intensity = max(dot(lightDir,n),0.0);
color = (ambient + diffuse*intensity) * att;
```

- O valor de att depende de:
  - factor de atenuação relativo à distância da luz ao ponto
  - o ponto estar dentro do cone de luz

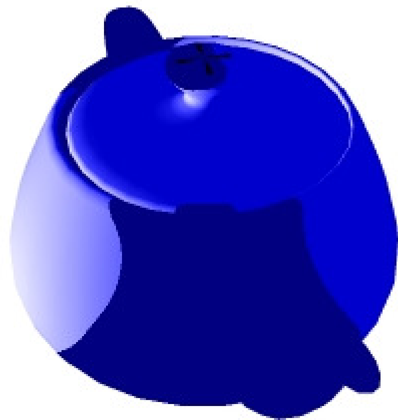
- Pelo código acima vemos que a cor ambiente está a ser aplicada sempre que att não seja zero.
- Os pontos da parte de trás do teapot estão dentro do cone de luz logo têm uma contribuição da luz ambiente, para alem da luz ambiente global.

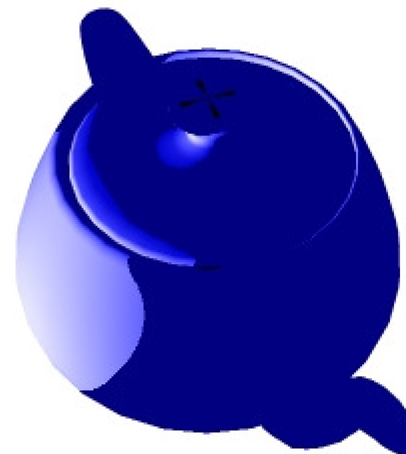


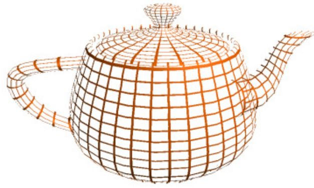
GLSL - Iluminação Spot Light

Código para eliminar o problema:

```
color = (diffuse*intensity)*att;
if ((intensity > 0.0) && (spotAtt != 0))
        color += ambient;
```







Referências

- OpenGL 2.0 Specification, www.opengl.org
- GLSL Specification, www.opengl.org
- "OpenGL Shading Language", Randi Rost, Addison Wesley