Texturas
WebGL

2019-2020

Fernando Birra

Objetivos

- Aprender a usar texturas em WebGL

- Perceber as utilizações que se podem dar às texturas para obter efeitos distintos

Passos para usar texturas*

- Adaptar o programa GLSL para receber as coordenadas que definem o mapeamento da textura
- Adaptar o fragment shader para aceder à textura
- Adaptar o programa para “carregar” ou “criar” a textura e enviá-la ao GPU
- Adaptar o programa para:
  - definir o mapeamento através das coordenadas de textura
  - definir qual a textura a usar para esse mapeamento

*Os passos indicados refletem apenas um dos vários cenários de uso

Vertex shader

- A versão corrente de WebGL apenas suporta texturas 2D
- O vertex shader passará a emitir um novo output (varying), contendo as coordenadas (s,t) que definem o mapeamento do vértice na textura 2D. Normalmente, este valor é fornecido pela aplicação (via atributo)
- Nada impede que vários conjuntos de coordenadas de textura sejam associados a cada vértice (por exemplo para definir mapeamentos distintos para diferentes texturas a serem combinadas)
- Também nada impede que as coordenadas de textura sejam determinadas pelo próprio vertex shader, não havendo assim necessidade as definir como atributos dos vértices (ver mapeamentos clássicos)
- O fragment shader usará as coordenadas interpoladas para aceder às texturas e determinar a cor final do fragmento.

Vertex shader
(mapeamento definido pela aplicação)

```
attribute vec4 vPosition;
attribute vec2 vTexCoord;

uniform mat4 mModelView;
uniform mat4 mProjection;

varying vec2 fTexCoord;

void main() {
    gl_Position = mProjection * mModelView * vPosition;
    fTexCoord = vTexCoord;
}
```

coordenadas de textura associadas ao vértice e passadas pela aplicação

coordenadas de textura que irão ser interpoladas em cada fragmento

Passagem das coordenadas de textura ao rasterizer

Vertex shader
(mapeamento definido pelo shader)

```
attribute vec4 vPosition;

uniform mat4 mModelView;
uniform mat4 mProjection;

varying vec2 fTexCoord;

void main() {
    gl_Position = mProjection * mModelView * vPosition;
    fTexCoord = vPosition.xy;
}
```

coordenadas de textura que irão ser interpoladas em cada fragmento

Passagem das coordenadas de textura ao rasterizer

Mapeamento ortogonal clássico. No caso geral, usa-se uma função da posição.

Fragment shader*

- O fragment shader recebe as coordenadas de textura de cada fragmento sob a forma duma variável varying

- O fragment shader receberá a textura a usar sob a forma duma variável uniform (podem usar-se várias texturas em simultâneo)

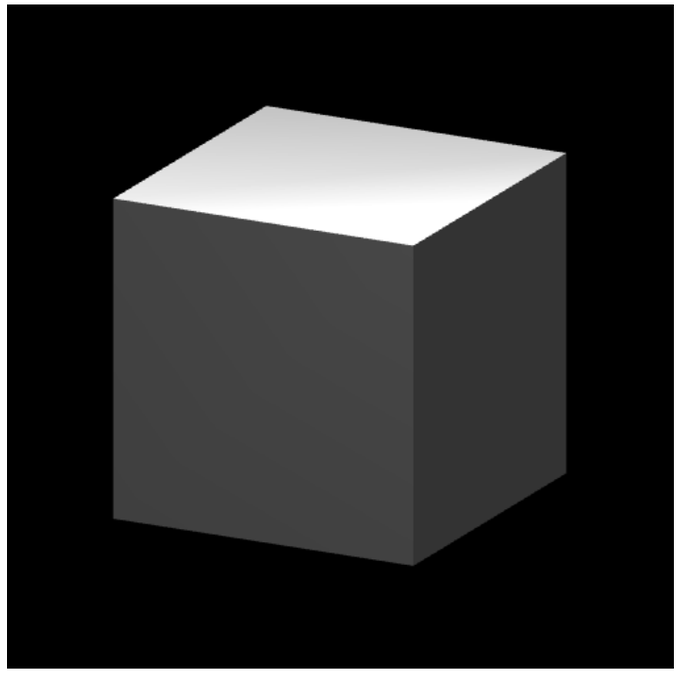
- O fragment shader acederá à textura e utilizará o valor encontrado para recriar o efeito pretendido

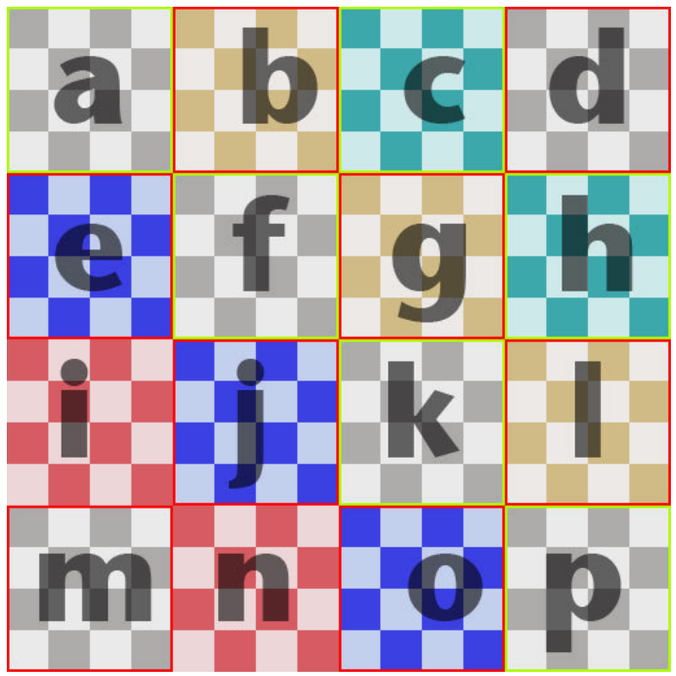
*Os passos indicados refletem apenas um dos vários cenários de uso

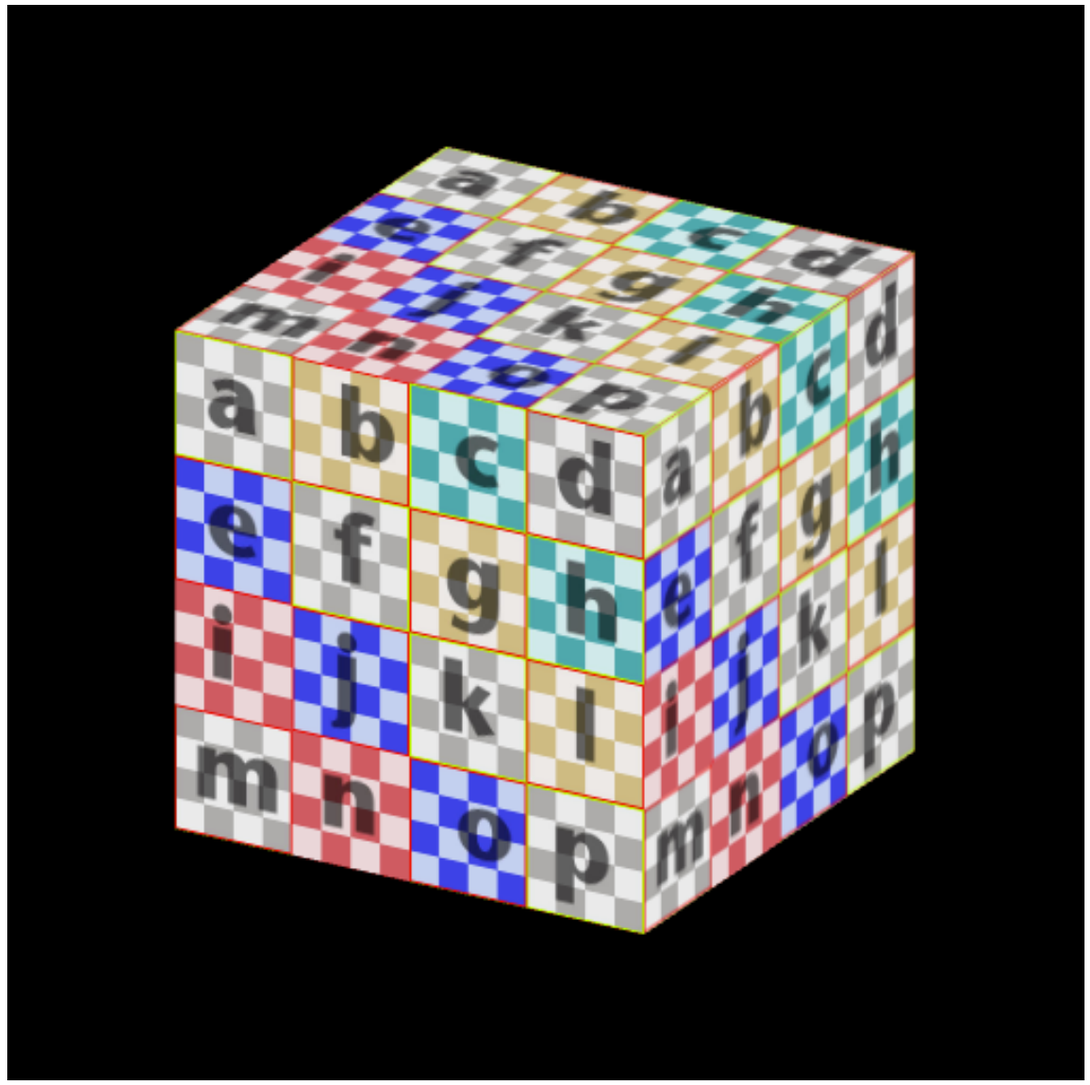
Utilizações típicas dos valores das texturas

- Substituição da cor do fragmento pela cor encontrada na textura (decalque)

Decalque





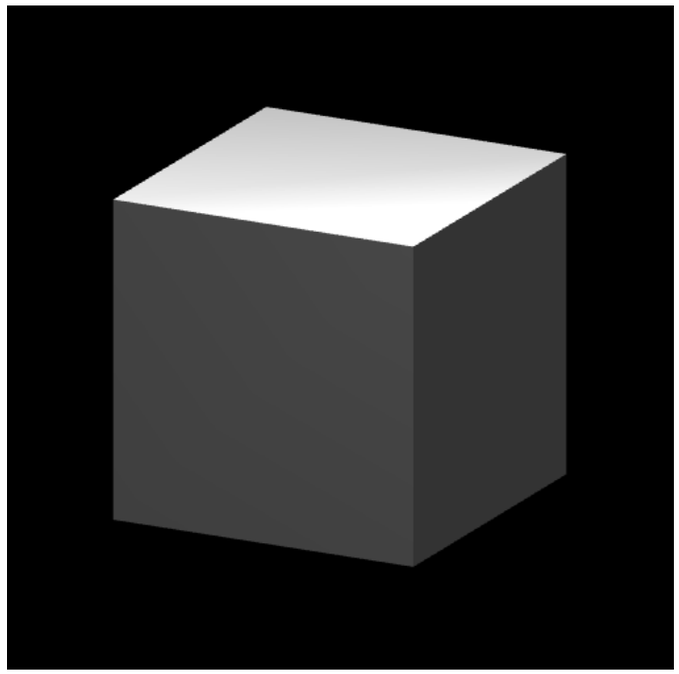


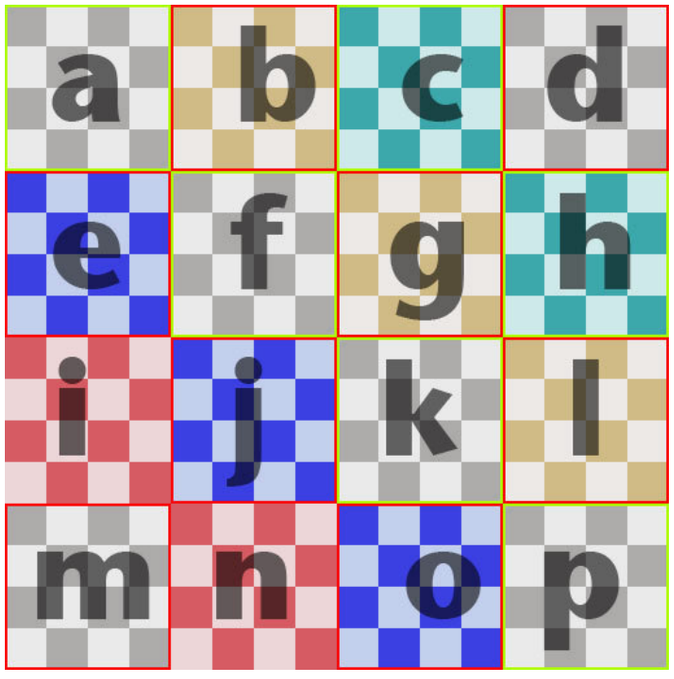
Utilizações típicas dos valores das texturas

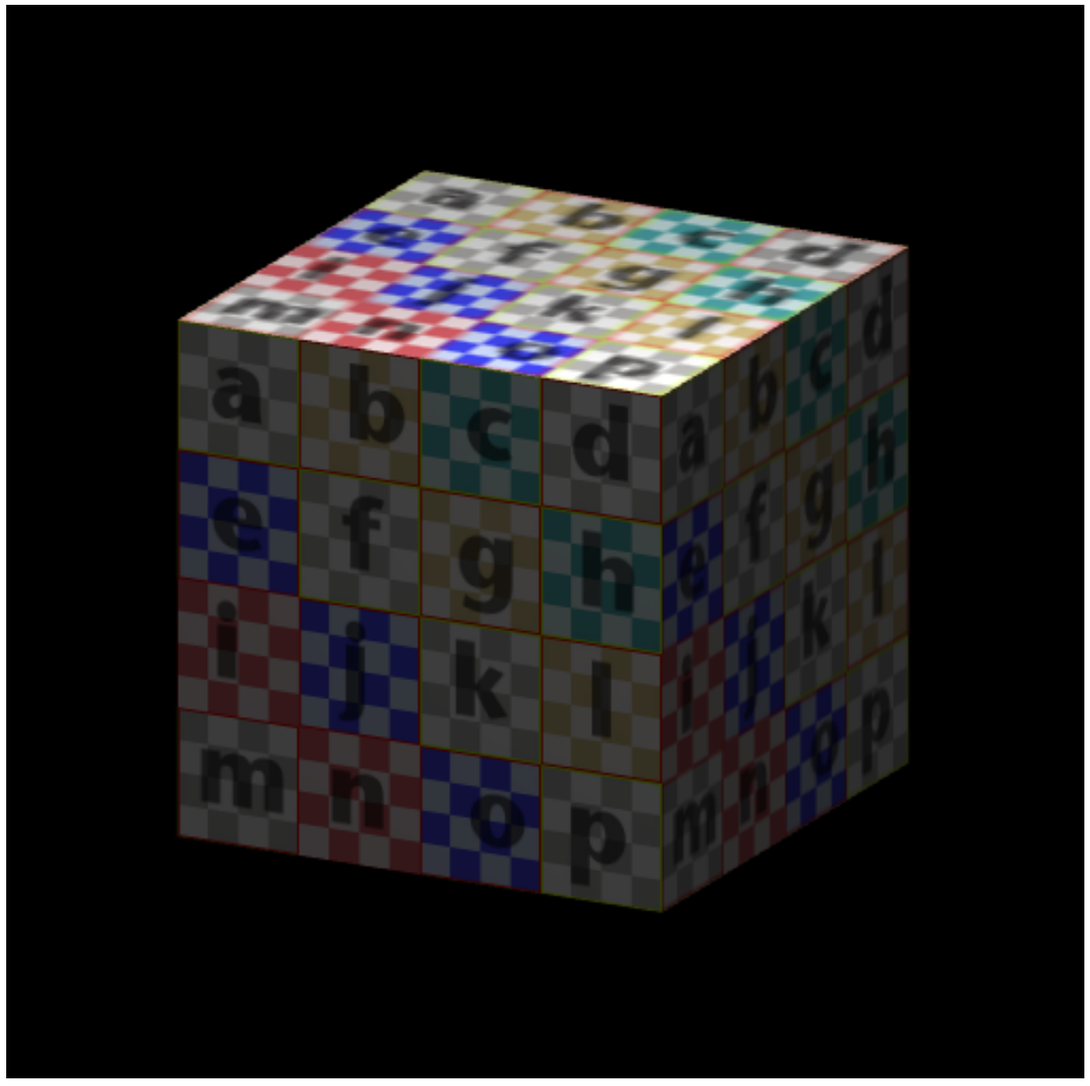
- Substituição da cor do fragmento pela cor encontrada na textura (decalque)

- Combinação/modulação da cor definida para o pixel (por exemplo pela aplicação dum modelo de iluminação) com a cor da textura.

Modulação da cor







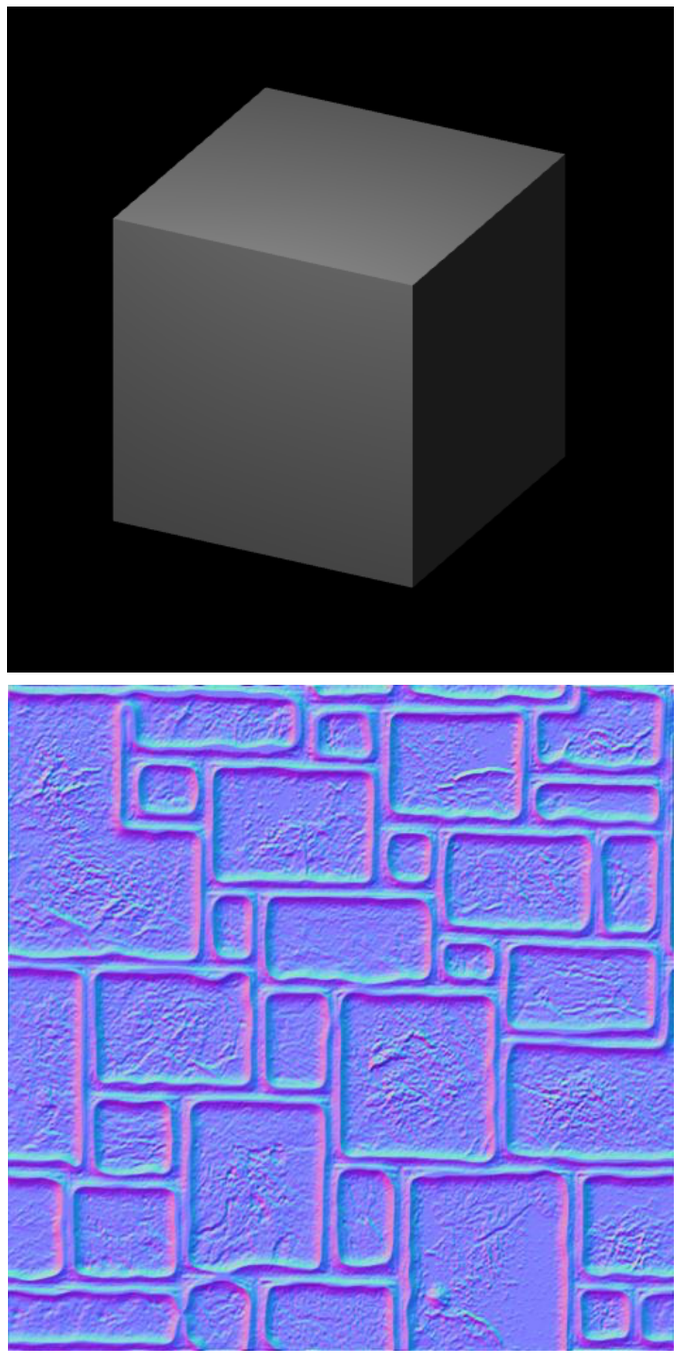
Utilizações típicas dos valores das texturas

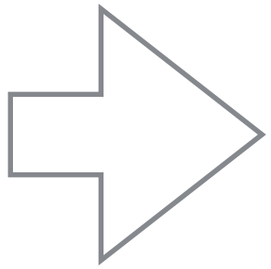
- Substituição da cor do fragmento pela cor encontrada na textura (decalque)
- Combinação/modulação da cor definida para o pixel (por exemplo pela aplicação dum modelo de iluminação) com a cor da textura.
- Utilização da textura para adicionar ou remover intensidade à cor dos pixels (mapas de luzes ou sombras projetadas)

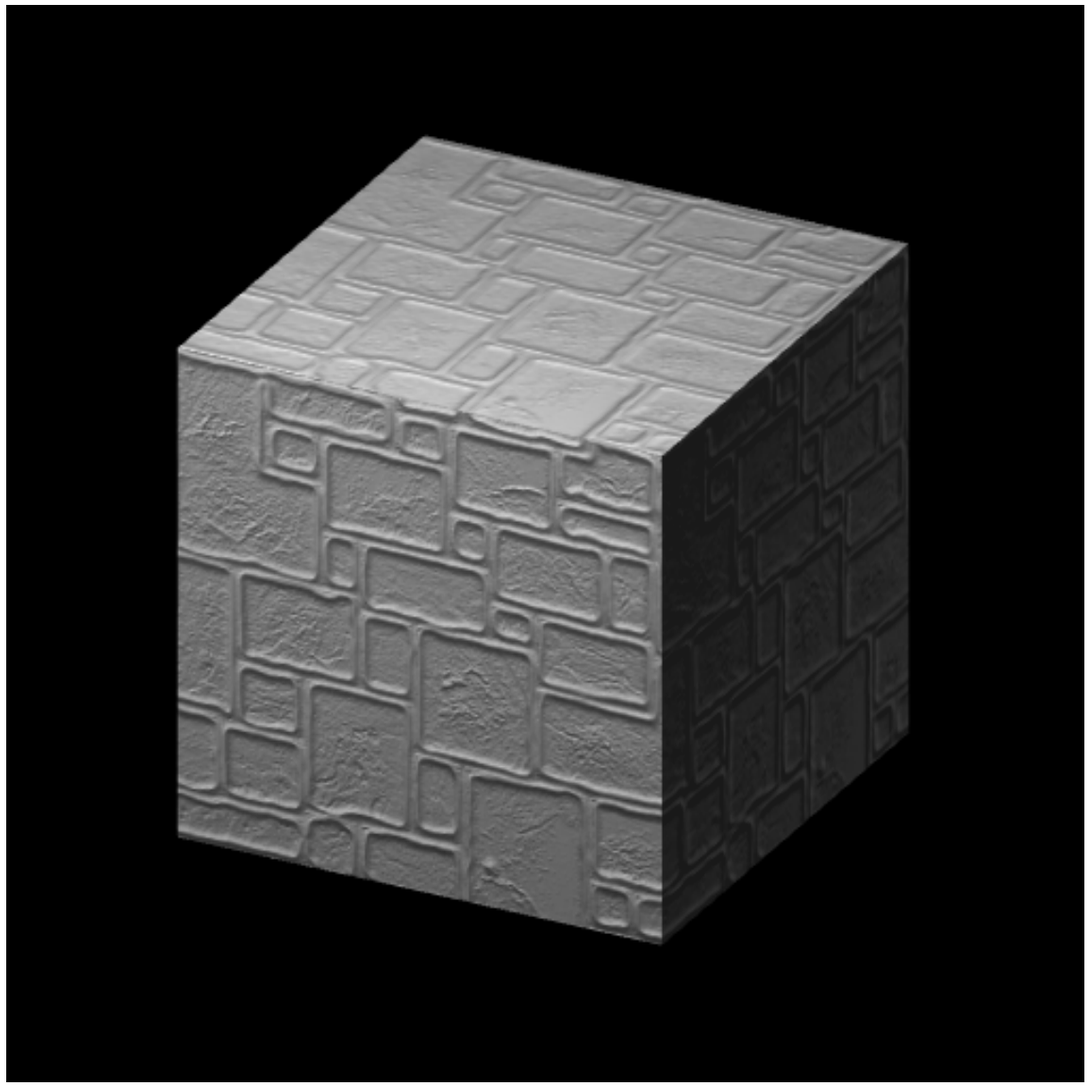
Utilizações típicas dos valores das texturas

- Substituição da cor do fragmento pela cor encontrada na textura (decalque)
- Combinação/modulação da cor definida para o pixel (por exemplo pela aplicação dum modelo de iluminação) com a cor da textura.
- Utilização da textura para adicionar ou remover intensidade à cor dos pixels (mapas de luzes ou sombras projetadas)
- Utilização da textura para substituir ou alterar a normal da superfície (bump mapping)

Bump mapping







Utilizações típicas dos valores das texturas

- Substituição da cor do fragmento pela cor encontrada na textura (decalque)
- Combinação/modulação da cor definida para o pixel (por exemplo pela aplicação dum modelo de iluminação) com a cor da textura.
- Utilização da textura para adicionar ou remover intensidade à cor dos pixels (mapas de luzes ou sombras projetadas)
- Utilização da textura para substituir ou alterar a normal da superfície (bump mapping)
- Variante do primeiro exemplo onde se usa a reflexão do vetor que aponta para observador em relação à normal e assim indexar uma textura envolta numa esfera ou nas faces dum cubo (environment mapping)

Utilizações típicas dos valores das texturas

- Substituição da cor do fragmento pela cor encontrada na textura (decalque)
- Combinação/modulação da cor definida para o pixel (por exemplo pela aplicação dum modelo de iluminação) com a cor da textura.
- Utilização da textura para adicionar ou remover intensidade à cor dos pixels (mapas de luzes ou sombras projetadas)
- Utilização da textura para substituir ou alterar a normal da superfície (bump mapping)
- Variante do primeiro exemplo onde se usa a reflexão do vetor que aponta para observador em relação à normal e assim indexar uma textura envolta numa esfera ou nas faces dum cubo (environment mapping)
- muitos, muitos outros…

Fragment shader (decalque)

```
varying vec2 fTexCoord;
```

coordenadas de textura que foram interpoladas em cada fragmento

tipo específico para representar texturas 2D no shader

```
uniform sampler2D texture;

void main() {
    gl_FragColor = texture2D(texture, fTexCoords);
}
```

variável uniforme que identifica a textura (imagem) a usar

função de amostragem da textura

Fragment shader (modulação)

```
varying vec2 fTexCoord;

uniform sampler2D texture;

void main() {
    ...
    color = ...; // computed color of the fragment
    gl_FragColor = color * texture2D(texture, fTexCoords);
}
```

final color modulated by texture

Programa

Programa

- O programa deverá:
  - carregar ou definir as imagens a usar como texturas
  - enviar as imagens para o GPU
  - especificar todos os detalhes da textura, tais como modo de repetição, filtragem, tipo de textura (RGB, Grayscale, etc.)
  - Enviar as coordenadas de textura para o GPU, juntamente com os demais atributos dos vértices
  - Durante o desenho:
    - ativar a aplicação de texturas
    - enviar para o shader a informação acerca das texturas a usar

Criação duma textura

```
function setupTexture() {
    // Create a texture.
    texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);

    // Fill the texture with a 1x1 blue pixel.
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE,
                  new Uint8Array([0, 0, 255, 255]));

     // Asynchronously load an image
     var image = new Image();
    image.src = “textures/image.jpg";
    image.onload =  function() {
       gl.bindTexture(gl.TEXTURE_2D, texture);
       gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
       // setup of texture parameters
       // ...
       gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,gl.UNSIGNED_BYTE, image);
       gl.bindTexture(gl.TEXTURE_2D, null);
     };
}
```

create a texture object

Criação duma textura

```
function setupTexture() {
    // Create a texture.
    texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);

    // Fill the texture with a 1x1 blue pixel.
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE,
                  new Uint8Array([0, 0, 255, 255]));

     // Asynchronously load an image
     var image = new Image();
    image.src = “textures/image.jpg";
    image.onload =  function() {
       gl.bindTexture(gl.TEXTURE_2D, texture);
       gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
       // setup of texture parameters
       // ...
       gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,gl.UNSIGNED_BYTE, image);
       gl.bindTexture(gl.TEXTURE_2D, null);
     };
}
```

make the texture the current one

Envio da textura para o GPU

```
function setupTexture() {
    // Create a texture.
    texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);

    // Fill the texture with a 1x1 blue pixel.
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE,
                  new Uint8Array([0, 0, 255, 255]));

     // Asynchronously load an image
     var image = new Image();
    image.src = “textures/image.jpg";
    image.onload =  function() {
       gl.bindTexture(gl.TEXTURE_2D, texture);
       gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
       // setup of texture parameters
       // ...
       gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,gl.UNSIGNED_BYTE, image);
       gl.bindTexture(gl.TEXTURE_2D, null);
     };
}
```

Send a dummy blue texture to the GPU and after asynchronous loading completes, replace it

Envio da textura para o GPU

```
function setupTexture() {
    // Create a texture.
    texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);

    // Fill the texture with a 1x1 blue pixel.
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE,
                  new Uint8Array([0, 0, 255, 255]));

     // Asynchronously load an image
     var image = new Image();
    image.src = “textures/image.jpg";
    image.onload =  function() {
       gl.bindTexture(gl.TEXTURE_2D, texture);
       gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
       // setup of texture parameters
       // ...
       gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,gl.UNSIGNED_BYTE, image);
       gl.bindTexture(gl.TEXTURE_2D, null);
     };
}
```

texture upload function: a 1x1 RGBA texture (4 bytes per texel), filled with a single blue texel.

Carregamento assíncrono

```
function setupTexture() {
    // Create a texture.
    texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);

    // Fill the texture with a 1x1 blue pixel.
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 1, 1, 0,
                  new Uint8Array([0, 0, 255, 255]));

    // Asynchronously load an image
    var image = new Image();
    image.src = “textures/image.jpg";
    image.onload =  function() {
      gl.bindTexture(gl.TEXTURE_2D, texture);
      gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
      // setup of texture parameters
      // ...
      gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,gl.UNSIGNED_BYTE, image);
      gl.bindTexture(gl.TEXTURE_2D, null);
    };
}
```

asynchronous loading of the image resource to be used as a texture.

Carregamento assíncrono

```
function setupTexture() {
    // Create a texture.
    texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);

    // Fill the texture with a 1x1 blue pixel.
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE,
                  new Uint8Array([0, 0, 255, 255]));

    // Asynchronously load an image
    var image = new Image();
    image.src = “textures/image.jpg";
    image.onload =  function() {
      gl.bindTexture(gl.TEXTURE_2D, text
      gl.pixelStorei(gl.UNPACK_FLIP_Y_WE
      // setup of texture parameters
      // ...
      gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,gl.UNSIGNED_BYTE, image);
      gl.bindTexture(gl.TEXTURE_2D, null);
    };
}
```

create an Image object

Carregamento assíncrono

```
function setupTexture() {
    // Create a texture.
    texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);

    // Fill the texture with a 1x1 blue pixel.
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE,
                  new Uint8Array([0, 0, 255, 255]));

    // Asynchronously load an image
    var image = new Image();
    image.src = “textures/image.jpg";
    image.onload =  function() {
      gl.bindTexture(gl.TEXTURE_2D, texture);
      gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, tr
      // setup of texture parameters
      // ...
      gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,gl.UNSIGNED_BYTE, image);
      gl.bindTexture(gl.TEXTURE_2D, null);
    };
}
```

define its source url

Carregamento assíncrono

```
function setupTexture() {
    // Create a texture.
    texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);

    // Fill the texture with a 1x1 blue pixel.
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.                                BYTE,
                  new Uint8Array([0, 0,

    // Asynchronously load an image
    var image = new Image();
    image.src = “textures/image.jpg";
    image.onload =  function() {
      gl.bindTexture(gl.TEXTURE_2D, texture);
      gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
      // setup of texture parameters
      // ...
      gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,gl.UNSIGNED_BYTE, image);
      gl.bindTexture(gl.TEXTURE_2D, null);
    };
}
```

specify callback to be executed upon finishing loading the resource

Parametrização e envio

```
function setupTexture() {
    // Create a texture.
    texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);

    // Fill the texture with a 1x1 blue pixel.
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE,
                  new Uint8Array([0, 0, 255, 255]));

    // Asynchronously load an image
    var image = new Image();
    image.src = “textures/image.jpg";
    image.onload =  function() {
      gl.bindTexture(gl.TEXTURE_2D, texture);
      gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
      // setup of texture parameters
      // ...
      gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,gl.UNSIGNED_BYTE, image);
      gl.bindTexture(gl.TEXTURE_2D, null);
    };
}
```

Make the texture the current one, flip it vertically, specify rest of parameters and send it to the GPU

Especificação do mapeamento

Exemplo de aplicação em cube.js

```
function cubeAddFace(a, b, c, d, n)
{
    var offset = cube_points.length;

    cube_points.push(cube_vertices[a]);
    cube_points.push(cube_vertices[b]);
    cube_points.push(cube_vertices[c]);
    cube_points.push(cube_vertices[d]);

    cube_texCoords.push(vec2(0.0, 0.0));
    cube_texCoords.push(vec2(1.0, 0.0));
    cube_texCoords.push(vec2(1.0, 1.0));
    cube_texCoords.push(vec2(0.0, 1.0));

    for(var i=0; i<4; i++)
        cube_normals.push(n);

    // Add 2 triangular faces (a,b,c) and
(a,c,d)
    cube_faces.push(offset);
    cube_faces.push(offset+1);
    cube_faces.push(offset+2);

    cube_faces.push(offset);
    cube_faces.push(offset+2);
    cube_faces.push(offset+3);

    // Add first edge (a,b)
    cube_edges.push(offset);
    cube_edges.push(offset+1);

    // Add second edge (b,c)
    cube_edges.push(offset+1);
    cube_edges.push(offset+2);
}
```

Envio das coordenadas para o GPU

```
...
cube_texCoords_buffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, cube_texCoords_buffer);
gl.bufferData(gl.ARRAY_BUFFER, flatten(cube_texCoords), gl.STATIC_DRAW);
...
```

Em tudo semelhante à forma como os demais atributos são passados colocados em buffers e enviados ao GPU

Durante o desenho…

The GPU has several texture units. Each with a unique identifier gl.TEXTUREx

```
...
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, texture);
gl.uniform1i(gl.getUniformLocation(program, "texture"), 0);
...
```

```
...
gl.bindBuffer(gl.ARRAY_BUFFER, cube_texCoords_buffer);
var vTexCoords = gl.getAttribLocation(program, "vTexCoords");
gl.vertexAttribPointer(vTexCoords, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vTexCoords);
...
```

Durante o desenho…

```
...
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, texture);
gl.uniform1i(gl.getUniformLocation(program, "texture"), 0);
...
```

Our texture is now bound to texture unit 0

```
...
gl.bindBuffer(gl.ARRAY_BUFFER, cube_texCoords_buffer);
var vTexCoords = gl.getAttribLocation(program, "vTexCoords");
gl.vertexAttribPointer(vTexCoords, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vTexCoords);
...
```

Durante o desenho…

```
...
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, texture);
gl.uniform1i(gl.getUniformLocation(program, "texture"), 0);
...

...
gl.bindBuffer(gl.ARRAY_BUFFER, cube_texCoords_buffer);
var vTexCoords = gl.getAttribLocation(program, "vTexCoords");
gl.vertexAttribPointer(vTexCoords, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vTexCoords);
...
```

Inside the shader each texture unit is referred by its number.

Durante o desenho…

```
...
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, texture);
gl.uniform1i(gl.getUniformLocation(program, "texture"), 0);
...
```

```
...
gl.bindBuffer(gl.ARRAY_BUFFER, cube_texCoords_buffer);
var vTexCoords = gl.getAttribLocation(program, "vTexCoords");
gl.vertexAttribPointer(vTexCoords, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vTexCoords);
...
```

Activate the attribute containing the texture coordinates.