Universidade Lusófona de Humanidades e Tecnologias

ULHT260-2267
Licenciatura em Engenharia Informática
Computação Gráfica
Nuno Cruz Garcia – p6040@ulusofona.pt

Aula Prática 04a

Temas
Texturas

Usar texturas para adicionar detalhe aos objectos é mais simples e eficaz que especificar muitos vértices com diferentes propriedades. Uma textura 2D é uma imagem que pode ser vista como um autocolante que é estampado numa superfície. A ideia base de texturas 2D é precisamente esta: usar uma imagem para determinar a cor de um pixel do objecto. Também existem texturas 1D – por exemplo padrões aplicados a uma linha – e 3D – por exemplo relevo de terrenos.

Para fazer *load* de texturas iremos usar a biblioteca *stb_image.h* . O download pode ser feito aqui. A biblioteca é composta apenas por este ficheiro *header*, pelo que para a utilizar basta adicionar o ficheiro ao projecto e fazer o include:
#include “your/path/to/stb_image.h”

Nota: existem outras bibliotecas que poderão eventualmente ter mais funcionalidades úteis para usar no projecto, e que são referidas no anexo no final desta ficha.

A utilização de texturas é feita nos *shaders*. Tanto pode ser feita no *vertex shader* como no *fragment shader*. A diferença na maioria das vezes é que no *vertex shader* o resultado é mais rápido mas não tão satisfatório e vice-versa em relação ao *fragment shader*. De grosso modo, a razão prende-se com a interpolação de cor feita em vértices ou *fragments*.
Vejamos: depois do *vertex shader*, o *raster* do *OpenGL* transforma os vértices em “candidatos a pixéis” que são os *fragments* – ou seja, no caso de um simples triângulo, três vértices dão origem a muitos *fragments*. Se a aplicação de uma textura complexa for feita no *vertex shader* a interpolação de cores da textura é feita com base nos três vértices, enquanto que se for feita no *fragment shader* é feita em muitos *fragments* e por isso de forma mais contínua e suave.

Parte I
Os passos para gerar uma textura são semelhantes aos de gerar um VBO ou VAO: (1) gerar o ID, (2) fazer Bind do objecto, (3) aplicar as operações adequadas.

Para gerar o ID da textura:
```
GLuint tex;
glGenTextures(1, &tex);
```

Para fazer *Bind* à textura:
```
glBindTexture(GL_TEXTURE_2D, tex);
```

A um pixel da textura dá-se o nome de *texel*. Importa definir as coordenadas da textura 2D: o canto inferior esquerdo corresponde à coordenada (0,0) e o superior direito à coordenada (1,1).

Considere-se o exemplo de uma textura 2D que é aplicada ao triângulo tipicamente usado nas fichas. Devemos especificar qual a coordenada da textura 2D (imagem) que corresponde a cada um dos vértices do triângulo, como mostra a Fig.1.

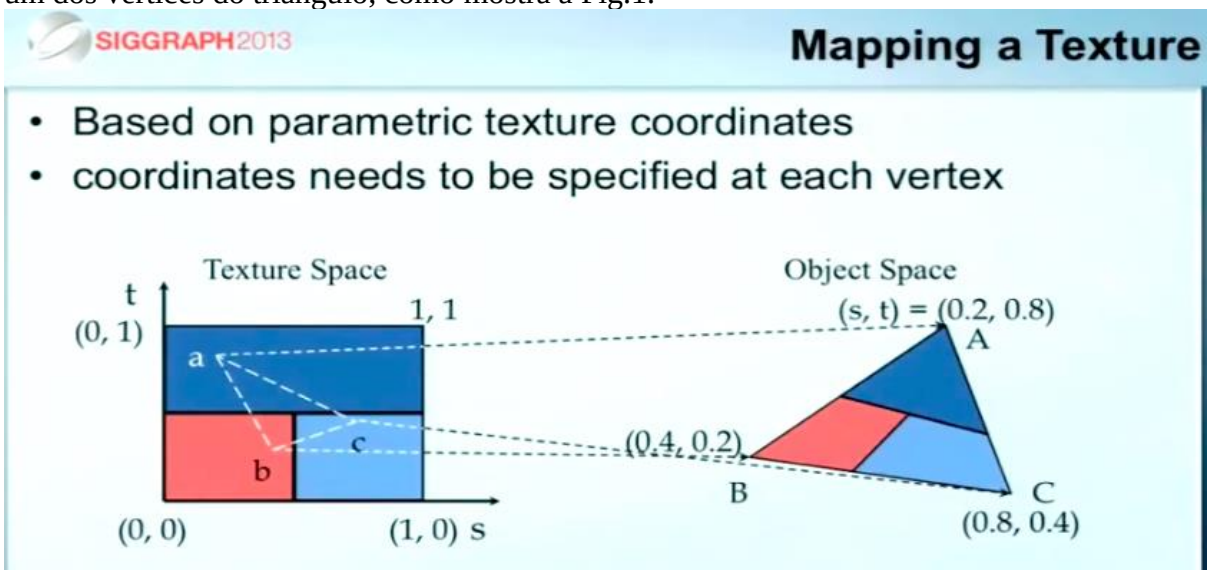

Figura 1 – https://www.youtube.com/watch?v=6-9XFm7XAT8

Por exemplo, podemos definir que a coordenada bottomLeft=(-1,-1,0) do triângulo corresponde à coordenada bottomLeft=(0,0) da imagem textura 2D.
As coordenadas do *texel* são adicionadas ao vector que contém as coordenadas dos vértices, como anteriormente foi feito com a cor.

```
float vertices[] = {
          //x, y, z,       coord. textura
        -1.0f, -1.0f, 0.0f, 0.0f, 0.0f,  //left
        1.0f, -1.0f, 0.0f, 1.0f, 0.0f,  //right
        0.0f,  1.0f, 0.0f, 0.5f, 1.0f, //top
  };
```

À operação que determina a cor de um pixel considerando a textura a aplicar dá-se o nome de *sampling*. O *shader sampla* a textura e atribui um cor ao pixel.

Se as coordenadas da textura estiverem fora do intervalo [0,1] existem 4 modos de *sampling* que descrevem como a textura deve funcionar:
GL_REPEAT, GL_MIRRORED_REPEAT, GL_CLAMP_TO_EDGE, GL_CLAMP_TO_BORDER.
Eles são definidos através da seguinte chamada para o eixo S e T - nome dado aos eixos X e Y da textura.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

Figura 2 https://open.gl/textures

A função *glTexImage2D* gera a textura. Os seus argumentos são:
. target – tipo de textura.
. level – nível de detalhe da textura; um objecto mais distante deve ter um nível de detalhe mais reduzido.
. internalformat – número de componentes de cor;
. width, height – dimensões;
. border – argumento legacy, deve ser 0;
. format – formato de dados dos pixéis;
. type – tipo de dados dos pixeís;
. data – a imagem em memória, *array* de valores de intensidade.

A função *glGenerateMipmap* gera as texturas nas suas várias dimensões / *levels* como especificado anteriormente no argumento *level*.

Ou seja, há essencialmente 3 coisas a fazer para se usar uma textura:
1 – Especificar a textura (ler uma imagem ou gerar através de um algoritmo, etc.) + construir a textura a partir dessa imagem (*glTexImage2D*).
2 – Definir a correspondência entre vértices do objecto e pontos da textura (pode ser directamente no array *vertices[]* e especificado no VBO).
3 – Definir parâmetros da textura (*glTexParameter*).

O código para declarar e usar uma textura tem o seguinte aspecto (código na aplicação c++):

```
unsigned int texture;
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);

// valores a alterar para verificar os vários modos.
// se as coord. fora de range [0,1]
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

// sampling method - GL_LINEAR neste caso = média dos 4 pixeis vizinhos
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

// load and generate the texture
int width, height, nrChannels;
unsigned char *data = stbi_load("yourImage.jpg", &width, &height,
&nrChannels, 0);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
                                        GL_UNSIGNED_BYTE, data);
```

```
glGenerateMipmap(GL_TEXTURE_2D);
stbi_image_free(data);
```

Exercício 1

Utilize um .jpg à escolha como textura, e experimente com os 4 modos de sampling da Fig2. Use a classe shader que foi implementada no último exercício da ficha 3.

Segue algum código exemplo, que pode ser adaptado para diferentes polígonos, etc.:
- O vector de vértices do triângulo deve ter 5 *floats* por vértice, que correspondem às 3 coordenadas do ponto e às duas correspondentes da textura: x y z s t. . Por exemplo

```
    float vertices[] = {
        -1.0f, -1.0f, 0.0f, 0.0f, 0.0f,  //left
        1.0f, -1.0f, 0.0f, 1.0f, 0.0f,  //right
        0.0f,  1.0f, 0.0f, 0.5f, 1.0f, //top
    };
```

- O VBO deve ser alterado de modo a reflectir esta organização das propriedades, nomeadamente a introdução dos valores das coordenadas da textura. Continuando o exemplo de cima:

```
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
                                        GL_STATIC_DRAW);
//position
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 5 * sizeof(float),
                                                  (void*)0);
glEnableVertexAttribArray(0);

//texture coordinates
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 5 * sizeof(float),
                                        (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);
```

Os shaders também devem reflectir essas alterações. As alterações são semelhantes ao que foi feito nas fichas práticas anteriores relativamente a cor.

- shader.vs

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec2 aTexCoord;
out vec2 TexCoord;
void main(){
   gl_Position = vec4(aPos.x, aPos.y, aPos.z, 1.0);
   TexCoord = aTexCoord;
}
```

O Fragment Shader declara um *uniform sampler2D* que é um tipo de dados especial do GLSL relacionado com texturas. Esta variável recebe o valor do ID da textura através da chamada BindTexture() feita no while loop da função main() da aplicação c++.

- shader.fs

```
#version 330 core
out vec4 FragColor;
in vec2 TexCoord;
uniform sampler2D ourTexture;   // recebe o ID da textura gerada, através do BindTexture()
void main(){
   FragColor = texture(ourTexture, TexCoord);
}
```

Repare-se como o FragColor é o resultado da função *texture*, existente na linguagem de Shading do OpenGL. Esta é a função que efectivamente *sampla* a cor para atribuir a cada pixel.

Dentro do *while loop* deve haver algo do género:

```
//bind textura - goes to uniform sampler variable.
glBindTexture(GL_TEXTURE_2D, texture);
//bind VAO
glBindVertexArray(vertexArrayID);
//use shader program
ourShader.use();
//draw
glDrawArrays(GL_TRIANGLES, 0, 3);
```

Atendendo a este código, deve conseguir verificar que a textura é aplicada triângulo.

Exercício 2

Modifique o seu código da seguinte forma:
1 – Adicione uma propriedade de cor a cada vértice, red para um vértice, green para outro, blue para outro. Isto é semelhante ao feito na última ficha.
2 – Adicione a tradicional chamada glVertexAttribPointer(...) e glEnableVertexAttribArray(x) de forma a satisfazer a nova propriedade.
3 – Adicione no *vertex shader* o *layout* para a nova coordenada ***layout*** *(location = 2)* ***in*** *vec3 aColor;* assim como uma variável de output adequada, e no main : *ourColor = aColor;*
4 – Substitua a linha de atribuição de valor da variáel *FragColor* no *Fragment Shader* por: *FragColor = texture(ourTexture, TexCoord) * vec4(ourColor, 1.0);*

A textura deverá aparecer colorida pelo espectro de cores RGB.

References:

https://learnopengl.com/
https://open.gl/

Anexo A – outras libs

Existem outras bibliotecas para carregar imagens e definir texturas, como o SOIL2 (fork da SOIL, mais recente – instalação aqui e/ou aqui), GLI (OpenGL Image), DevIL (Developers Image Library). O uso é livre para o projecto.

Por exemplo, a SOIL – Simple OpenGL Image Library
Para instalar:

- Download de zip aqui https://www.lonesock.net/soil.html
- Em sistemas Ubuntu e semelhantes:
    - sudo apt-get install libsoil-dev
    - compilar com -lSOIL
- Em Mac:
    - // ignore
    - // No terminal, $ cd pasta/do/SOIL
    - // $ mkdir build
    - // $ cd build
    - // $ cmake ..
    - // $ make.
    - Como de costume, No Xcode, click project name,
        - adicionar o include path em "Header Search Path"
        - adicionar o lib path em "Library Search Path"
    - Build Phases -> link binary with libraries
        - adicionar o ficheiro *.a*
        - adicionar Core.Foundation
- Em Windows:
    - Siga este vídeo (3min) https://www.youtube.com/watch?v=O9qrm5ZelD8