

Tutorial 5 – Texturização

1. Introdução

De forma a aumentar o realismo dos objetos 3D, é possível fazer uso de imagens e aplicá-las às superfícies dos objetos geométricos. Este processo é denominado de texturização, sendo que as imagens utilizadas são designadas de texturas. À semelhança dos objetos 3D, as texturas também têm um espaço de coordenadas, mas de duas dimensões (UV) com valores entre 0 e 1 em que as coordenadas das texturas são designadas como *texels*.

Para aplicar uma textura a um objeto, é necessário mapear cada vértice do objeto 3D a um ponto da textura que tem como referência o referencial UV . Tendo a Figura 1 como exemplo, cada ponto da geometria final tem uma representação no referencial UV . O *fragment shader* é responsável por mapear cada ponto da textura na geometria.

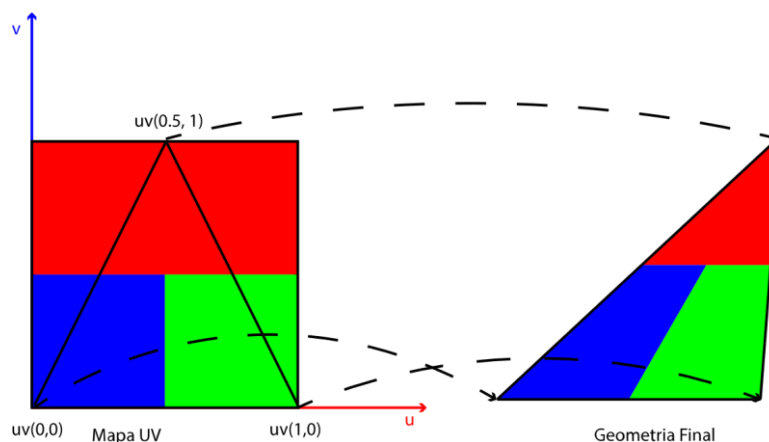


Figura 1 – Mapeamento de texturas

2. Tutorial

2.1. Ficheiros necessários

1. À semelhança dos tutoriais anteriores, cria uma pasta com o nome “*Tutorial 5*” e copia para dentro dessa pasta todos os ficheiros do tutorial anterior. Apaga todos os comentários existentes no código para uma melhor organização e compreensão da realização deste tutorial. Para além disso, deves também apagar as alterações feitas relativas aos desafios propostos no tutorial anterior.
2. Dirige-te à pasta “*Tutorial 5*” e cria uma pasta com o nome “*Images*”.
3. Accede ao link “<https://opengameart.org/content/simple-toon-wooden-crate-texture>”. Assim que a página carregar, vais deparar-te com uma página web com o aspeto semelhante à Figura 2. Descarrega a imagem, utilizando o link assinalado a vermelho na imagem abaixo, e guarda-a com o nome “*boxImage*” na pasta que criaste anteriormente com o nome “*Images*” dentro da pasta “*Tutorial 5*”.

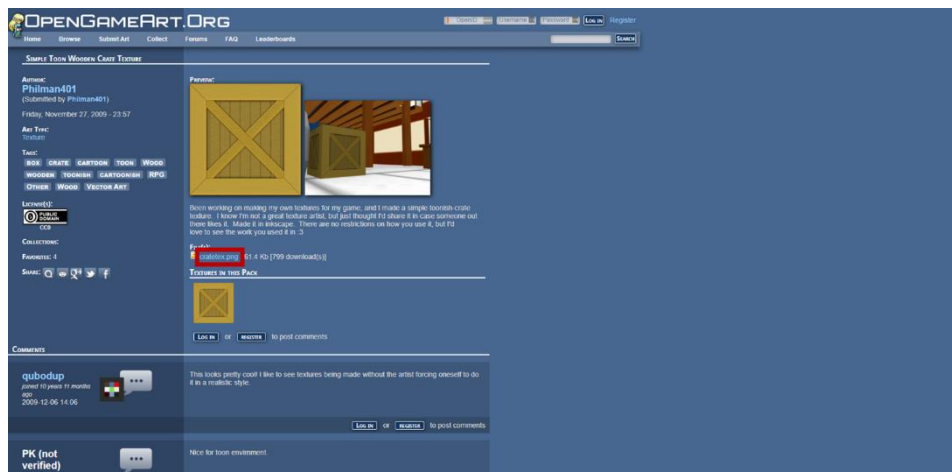


Figura 2 Download da textura.

2.2. Texturização

1. Abre o VSCode e, no programa, abre o ficheiro “*index.html*” localizado na pasta “*Tutorial 5*”
2. Importa a imagem que descarregaste com recurso ao código assinalado a vermelho:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>WebGL - Primeiro Cubo</title>
5   </head>
6   <body onload="Start()">
7     <!--
8       A linha seguinte adiciona a imagem que descarregaste ao documento HTML,
9       dá-lhe um id para ser mais fácil a sua procura e ainda dá-lhe o comprimento
10      e altura de 0 unidades para não aparecer quando a página é carregada.
11    -->
12    </img>
13    <script src="https://cdnjs.cloudflare.com/ajax/libs/mathjs/6.6.0/math.js"></script>
14    <script src="./JavaScript/matriz.js"></script>
15    <script src="./JavaScript/camara.js"></script>
16    <script src="./JavaScript/shaders.js"></script>
17    <script src="./JavaScript/app.js"></script>
18  </body>
19 </html>
```

3. Nos tutoriais anteriores estavam a utilizar código *RGB* para cada um dos vértices do cubo. Agora, ao aplicar a textura ao cubo criado, iremos utilizar o referencial *UV* de forma a mapear a textura ao cubo. Para isso, abre o ficheiro “*shaders.js*” e altera o *vertex shader* tal como se segue:

```
1 var codigoVertexShader = [  
2   'precision mediump float;  
3  
4   'attribute vec3 vertexPosition;',  
5   // Uma vez que agora vamos receber coordenadas UV já não vamos precisar da linha  
6   // 'attribute vec3 vertexColor;', e vamos substituí-la pela linha abaixo  
7   'attribute vec2 texCoords;',  
8  
9   // Pela mesma explicação da variável acima, agora já não vamos precisar da linha  
10  // 'varying vec3 fragColor;', e vamos substituí-la pela linha abaixo.  
11  'varying vec2 fragTexCoords;',  
12  
13  'uniform mat4 transformationMatrix;',  
14  'uniform mat4 visualizationMatrix;',  
15  'uniform mat4 projectionMatrix;',  
16  'uniform mat4 viewportMatrix;',  
17  
18  'void main(){',  
19  // Uma vez que agora não estamos a utilizar cores, temos que adaptar a linha abaixo  
20  // para que o fragmentShader receba as coordenadas UV.  
21  '  fragTexCoords = texCoords;',  
22  '  gl_Position = vec4(vertexPosition, 1.0) * transformationMatrix * visualizationMatrix * projectionMatrix * viewportMatrix;',  
23  '}',  
24 ].join('\n');  
25
```

4. Para além de alterar o *vertex shader* temos também que alterar o *fragment shader* para receber as informações que provêm do *vertex shader*. Para isso altera o código do *fragment shader* da seguinte forma:

```
25  
26 var codigoFragmentShader = [  
27   'precision mediump float;  
28  
29   // Uma vez que agora vamos receber coordenadas UV já não vamos precisar da linha  
30   // 'varying vec3 fragColor;', e vamos substituí-la pela linha abaixo  
31   'varying vec2 fragTexCoords;',  
32  
33   // Vamos adicionar uma variável que guarde a texture que vai ser utilizada para  
34   // aplicar ao cubo.  
35   'uniform sampler2D sampler;',  
36  
37   'void main(){',  
38   // Uma vez que temos as coordenadas UV temos que utilizar a função abaixo para ir buscar  
39   // a cor de cada pixel à textura fornecida.  
40   '  gl_FragColor = texture2D(sampler, fragTexCoords);',  
41   '}',  
42 ].join('\n');
```

5. Agora que tens os shaders prontos para receber a textura e aplicá-la corretamente ao objeto é necessário alterarmos o nosso programa para passar essas mesmas informações. A primeira coisa que vais fazer é criar uma variável que guarde a textura, para isso transcreve o código abaixo para o ficheiro “*app.js*”

```
17  
18 // Variável que guarda na memória da GPU a Textura que será utilizada.  
19 var boxTexture = GL.createTexture();  
20
```

6. Agora, na função *PrepareTriangleData()*, onde definimos cada vértice e a cor desse mesmo vértice, teremos que adaptar esse mesmo *array* para que, em vez de receber a cor, receba as coordenadas *UV* da textura. Para isso adapta o *array* para ficar igual ao seguinte:

```
72     ... vertexPosition = [  
73     ... // Em vez de termos 3 valores para as cores RGB vamos ter apenas  
74     ... // 2 valores, que são as coordenadas UV  
75     ... // X, Y, Z, U, V  
76     ... // Frente  
77     ... 0, 0, 0, 0, 0,  
78     ... 0, 1, 0, 0, 1,  
79     ... 1, 1, 0, 1, 1,  
80     ... 1, 0, 0, 1, 0,  
81  
82     ... // Direita  
83     ... 1, 0, 0, 0, 0,  
84     ... 1, 1, 0, 1, 0,  
85     ... 1, 1, 1, 1, 1,  
86     ... 1, 0, 1, 0, 1,  
87  
88     ... // Trás  
89     ... 1, 0, 1, 1, 0,  
90     ... 1, 1, 1, 1, 1,  
91     ... 0, 1, 1, 0, 1,  
92     ... 0, 0, 1, 0, 0,  
93  
94     ... // Esquerda  
95     ... 0, 0, 1, 0, 1,  
96     ... 0, 1, 1, 1, 1,  
97     ... 0, 1, 0, 1, 0,  
98     ... 0, 0, 0, 0, 0,  
99  
100    ... // Cima  
101    ... 0, 1, 0, 0, 0,  
102    ... 0, 1, 1, 0, 1,  
103    ... 1, 1, 1, 1, 1,  
104    ... 1, 1, 0, 1, 0,  
105  
106    ... // Baixo  
107    ... 1, 0, 0, 0, 0,  
108    ... 1, 0, 1, 0, 1,  
109    ... 0, 0, 1, 0, 1,  
110    ... 0, 0, 0, 0, 0,  
111    ... ];
```

7. Agora que já tens a informação *UV* para cada vértice, é necessário configurar a *GPU* e dar-lhe a imagem que queremos que ela use para texturizar o cubo. Para isso copia o código abaixo para o fim da função *PrepareTriangleData()*:

```
152
153     ....// Agora é necessário configurar os parâmetros da GPU.
154     ....// Primeiro temos que fazer bind à textura...
155     ....GL.bindTexture(GL.TEXTURE_2D, boxTexture);
156
157     ....// Parâmetros necessários para a GPU saber como interpretar a rasterização
158     ....// Faz Clamp à borda no eixo do U
159     ....GL.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_WRAP_S, GL.CLAMP_TO_EDGE);
160
161     ....// Faz Clamp à borda no eixo do V
162     ....GL.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_WRAP_T, GL.CLAMP_TO_EDGE);
163
164     ....// As duas linhas a baixo indicam como deve ser escalada a textura tanto
165     ....// para diminuí-la (TEXTURE_MIN_FILTER) como para aumentá-la (TEXTURE_MAG_FILTER)
166     ....GL.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_MIN_FILTER, GL.LINEAR);
167     ....GL.texParameteri(GL.TEXTURE_2D, GL.TEXTURE_MAG_FILTER, GL.LINEAR);
168
169     ....// Passamos a imagem que está no documento "escondida" através do id da imagem
170     ....GL.texImage2D(
171         ....GL.TEXTURE_2D, // Tipo da Textura
172         ....0, // Detalhe da imagem (0 é o valor por defeito)
173         ....GL.RGBA, // Tipo de imagem
174         ....GL.RGBA, // Tipo de textura que vai ser aplicada a imagem
175         ....GL.UNSIGNED_BYTE, // Tipo de valores da textura
176         ....document.getElementById('boxImage') // Imagem que deve ser passada para o sampler
177     ....);
178 }
```

8. Depois disso, é necessário corrigir a transferência da informação para a *GPU* uma vez que ela ainda está a utilizar o *array* como se ainda estivesse com as cores. Altera o código da função *SendDataToShaders()* para ficar igual à código da imagem abaixo.

```
180 function SendDataToShaders() {
181     ....var vertexPositionAttributeLocation = GL.getAttribLocation(program, "vertexPosition");
182     ....// Agora em vez de irmos buscar a localização da variável "vertexColor" vamos buscar
183     ....// a localização da variável "texCoords"
184     ....var texCoordAttributeLocation = GL.getAttribLocation(program, 'texCoords')
185
186     ....GL.vertexAttribPointer(
187         ....vertexPositionAttributeLocation,
188         ....3,
189         ....GL.FLOAT,
190         ....false,
191         ....// Agora o conjunto apenas tem 5 valores.
192         ....5 * Float32Array.BYTES_PER_ELEMENT,
193         ....0 * Float32Array.BYTES_PER_ELEMENT
194     ....);
195
196     ....GL.vertexAttribPointer(
197         ....// Mudar a variável de vertexColorAttributeLocation
198         ....// para texCoordAttributeLocation
199         ....texCoordAttributeLocation,
200         ....// Agora só enviamos um conjunto de 2 valores para a variável
201         ....// texCoords
202         ....2,
203         ....GL.FLOAT,
204         ....false,
205         ....// Agora o conjunto apenas tem 5 valores.
206         ....5 * Float32Array.BYTES_PER_ELEMENT,
207         ....3 * Float32Array.BYTES_PER_ELEMENT
208     ....);
209
210     ....GL.enableVertexAttribArray(vertexPositionAttributeLocation);
211     ....// Substituímos "vertexColorAttributeLocation" por "texCoordAttributeLocation"
212     ....GL.enableVertexAttribArray(texCoordAttributeLocation);
213
214     ....finalMatrixLocation = GL.getUniformLocation(program, 'transformationMatrix');
215     ....visualizationMatrixLocation = GL.getUniformLocation(program, 'visualizationMatrix');
216     ....projectionMatrixLocation = GL.getUniformLocation(program, 'projectionMatrix');
217     ....viewportMatrixLocation = GL.getUniformLocation(program, 'viewportMatrix');
218
219 }
```

Agora já tens tudo pronto para abrires a página *web* e veres o cubo texturizado com a imagem que descarregamos no início deste tutorial. Nota que, se abrires a página *web* fazendo duplo clique no ficheiro "*index.html*" não vais ver nada, uma vez que o *CHROME*

e o **FIREFOX** vão dar um erro por questões de segurança visto que a imagem tem origem noutra *site*. De forma a ultrapassar isto, podes usar o plugin *Live server* do *VSCode* (instalado no primeiro tutorial) para emular um servidor *web* de forma a que ambos os ficheiros tenham a mesma origem. A Figura 3 ilustra o botão através do qual podes fazer uso do plugin *Live Server*.

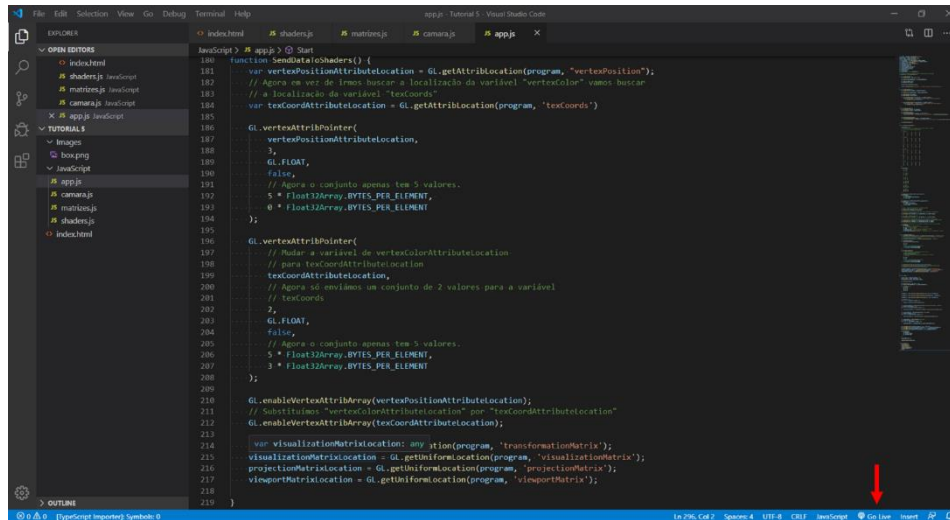


Figura 3 – Utilização do Live Server.

2.3. Desafios

Desafio 1. Altera a textura do cubo para a textura disponível no link: <https://opengameart.org/sites/default/files/4%20signs-sml.jpg>. O teu cubo deverá ficar com o aspeto ilustrada na figura Figura 4.



O canvas encontra-se acima deste texto!

Figura 4 - Captura de ecrã ilustrativas do resultado esperado do Desafio 1.

Desafio 2. Configura o teu cubo para que cada uma das faces laterais corresponda a cada uma das diferentes imagens da textura, como ilustra a Figura 5¹.



Figura 5 - Capturas de ecrã ilustrativas do resultado esperado do Desafio 2.

ENTREGA

O trabalho deve ser submetido no MOODLE até dia **23/04/2021**.

A submissão é individual e deve conter todos os ficheiros necessários à correta execução da aplicação: uma pasta contendo o ficheiro .html e a pasta JavaScript com os respetivos ficheiros.

¹ *Dica:* se tiveres dificuldades em encontrar os valores para mapear a textura corretamente para cada uma das faces, desenhar o cubo e fazer essa correspondência primeiro no papel pode ser muito útil.