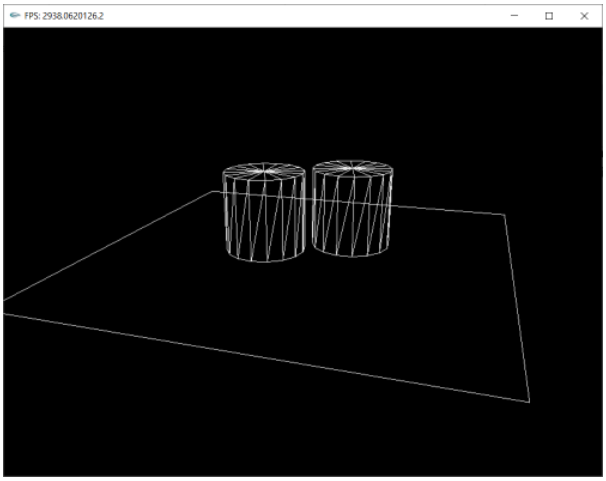
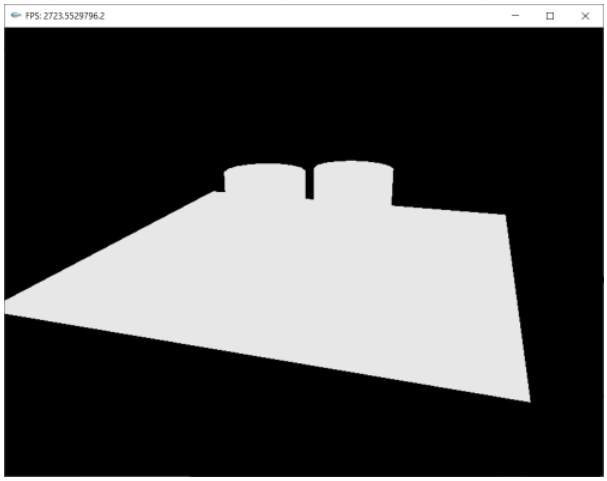
PL11 - Textured Cylinder

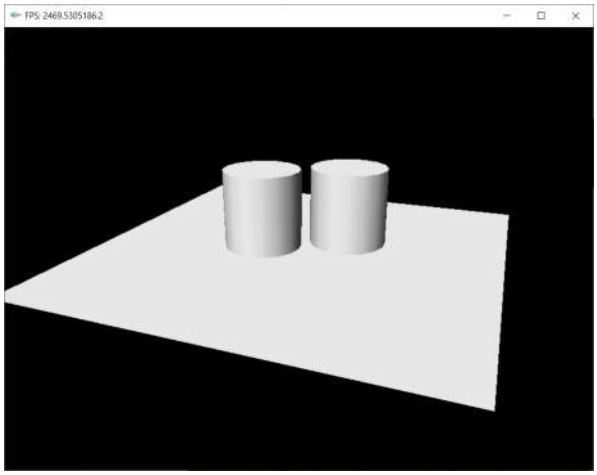
Notas para a componente prática de Computação Gráfica
Universidade do Minho
Carlos Brito and António Ramires

1. Introdução

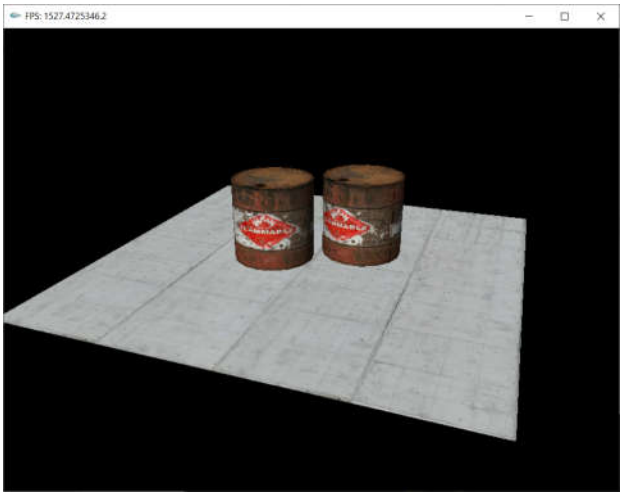
Com este guião pretende-se explorar a adição de uma textura ao cilindro iluminado da aula 09, completando a cena que se encontra pronta no código desta aula prática. À semelhança do guião anterior, a sequência de renders a seguir será o desenho da cena sem normais ou texturas:

Ao ativar as normais da aula 09, ficamos com:



E com as texturas da tarefa de hoje ficamos com:

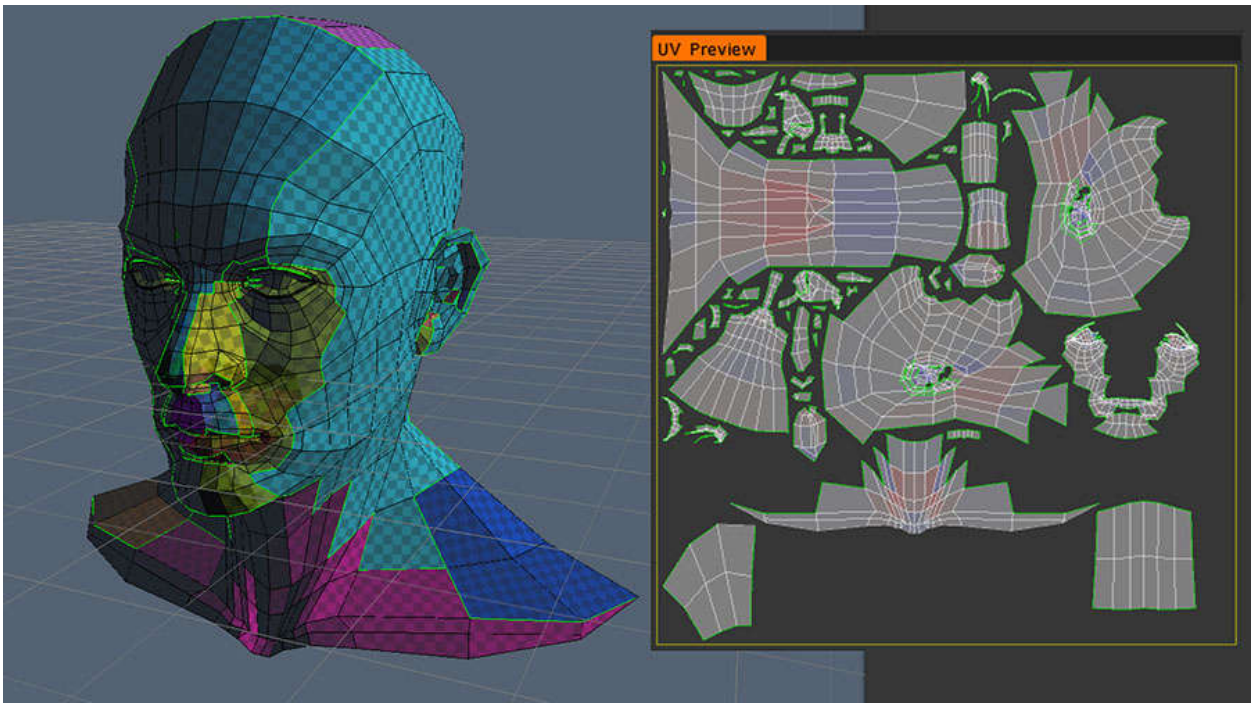


2. Coordenadas de textura (Texture maps ou texture Atlas)

Texturas são uma componente fundamental da computação gráfica e consistem em imagens que que atribuem cor (e várias outras propriedades) aos nossos modelos 3D.

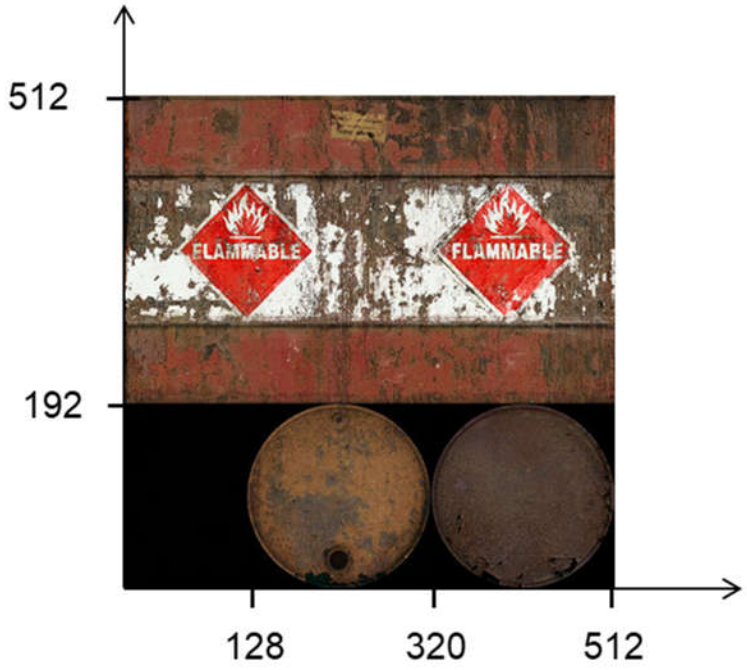
Um texture atlas da personagem Heavy do jogo Team Fortress 2

Isto é feito a partir de um processo denominado de UV mapping, que consiste em estabelecer o **mapeamento entre localizações numa uma imagem e vértices de um modelo**. U e V vêm do nome que é comum atribuir às coordenadas 2D em texturas, sendo que x, y e z já se encontram reservados para a posição.



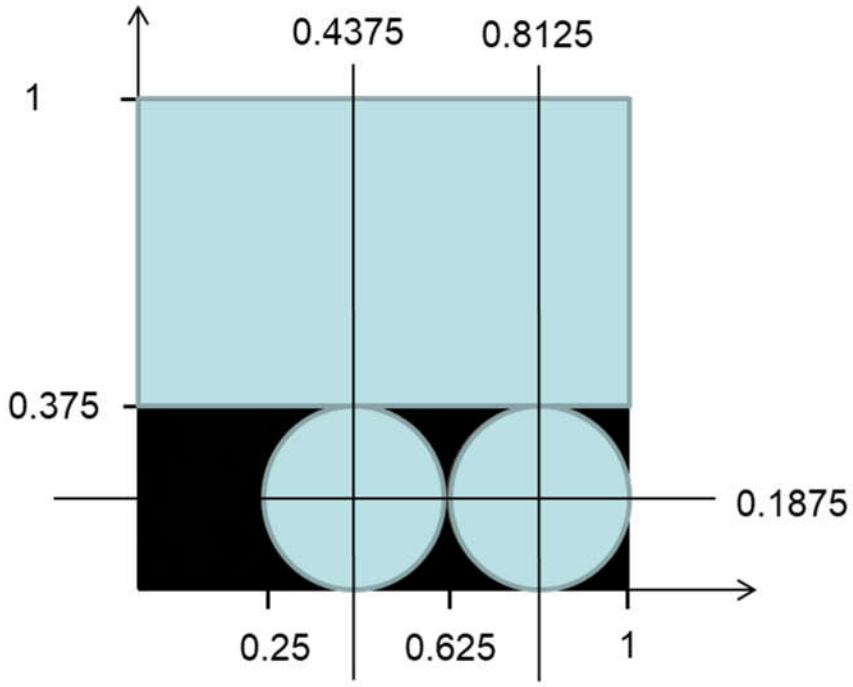
A produção deste tipo de mapas de textura está normalmente associada a artistas 3D, que ao esculpirem os modelos também definem como projetar uma imagem neles, sendo que qualquer textura que respeite o formato escolhido produz resultados corretos.

Nesta tarefa, pela primeira vez estamos perante um modelo que necessita de **regiões diferentes da textura para regiões diferentes da geometria**, nomeadamente os dois tampos do barril e o seu corpo. Para tal já definimos um formato de atlas que será utilizado:

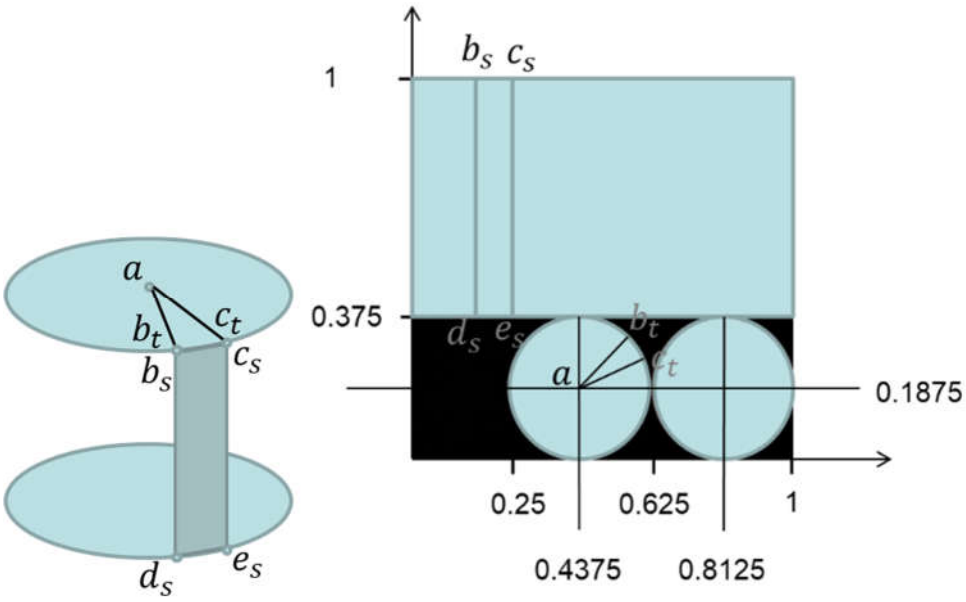


Regra geral, o espaço U-V de texturas **não utiliza pixéis como unidade**, mas sim um valor adimensional, que varia entre 0 e 1 onde as coordenadas (0, 0) e (1, 1) correspondem a cantos opostos na imagem. Por omissão (0, 0) corresponde ao canto superior esquerdo da imagem e (1,1) corresponde ao canto inferior direito, no entanto podemos e iremos alterar este comportamento através do DevIL, de forma a colocar a origem da textura no canto inferior esquerdo.

Isto significa que passamos a indexar os pixéis da nossa imagem de forma **independente da resolução**, assim, por exemplo, o ponto (0.5, 0.5) será sempre o centro da imagem quer ela tenha uma resolução de 256x256 ou 1920x1080. A imagem anterior passa a ser indexada da seguinte forma:



Para realizar este mapeamento correctamente, temos que atribuir a um vértice do cilindro um par de coordenadas u, v da textura relativas ao pedaço da imagem correcta. No seguinte esquema podemos ver o mapeamento necessário, por exemplo ao vértice a, tem de corresponder o par de coordenadas (0.4375,0.1875).



Para o cálculo das coordenadas b_t e c_t teremos de usar coordenadas polares, tal como usámos no cálculo das posições dos vértices respectivos. A corpo do cilndro é a zona mais fácil e as coordenadas variam de forma linear com a iteração utilizada para construir as posições do corpo do cilndro.

Para o chão iremos utilizar o mapeamento Streched da última aula, ou seja, uma correspondência directa entre os 4 vértices do plano e os 4 cantos da imagem.

3. Implementação com DevIL e OpenGL

Carregamento de uma imagem para uma textura RGB na placa gráfica:

```
unsigned int t,tw,th;
unsigned char *texData;
unsigned int texID;

// Iniciar o DevIL
ilInit();

// Colocar a origem da textura no canto inferior esquerdo
ilEnable(IL_ORIGIN_SET);
ilOriginFunc(IL_ORIGIN_LOWER_LEFT);

// Carregar a textura para memória
ilGenImages(1,&t);
ilBindImage(t);
ilLoadImage((ILstring)s.c_str());
tw = ilGetInteger(IL_IMAGE_WIDTH);
th = ilGetInteger(IL_IMAGE_HEIGHT);

// Assegurar que a textura se encontra em RGBA (Red, Green, Blue, Alpha) com um byte (0 -
255) por componente
ilConvertImage(IL_RGBA, IL_UNSIGNED_BYTE);
texData = ilGetData();
```

```
// Gerar a textura para a placa gráfica
glGenTextures(1,&texID);

glBindTexture(GL_TEXTURE_2D,texID);
glTexParameteri(GL_TEXTURE_2D,     GL_TEXTURE_WRAP_S,          GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D,     GL_TEXTURE_WRAP_T,          GL_REPEAT);

glTexParameteri(GL_TEXTURE_2D,     GL_TEXTURE_MAG_FILTER,      GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,     GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// Upload dos dados de imagem
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, tw, th, 0, GL_RGBA, GL_UNSIGNED_BYTE, texData);
glGenerateMipmap(GL_TEXTURE_2D);

glBindTexture(GL_TEXTURE_2D, 0);


return texID;
```

Desenho do chão em modo imediato

```
glBegin(GL_QUADS);

glNormal3f(0, 1, 0);
glTexCoord2f(1, 0);
glVertex3f(xmax, 0, zmin);

...

glEnd();
```

Desenho do cilindro com VBO's

Dado que as coordenadas de textura são mais uma vez informação **por vértice**, o processo para a adição de coordenadas de textura é idêntico ao das normais: na altura de setup precisamos de criar mais um buffer e ativar mais um client state e na altura de desenho definir semântica.

```
// Gerar um buffer para conter as coordenadas de textura e fazer upload para a GPU
glBindBuffer(GL_ARRAY_BUFFER, texCoord);
glBufferData(GL_ARRAY_BUFFER, sizeof(float) * vertexCount * 2, t, GL_STATIC_DRAW);

// Configurar o OpenGL para aceitar um buffer adicional com coordenadas de textura
glEnableClientState(GL_TEXTURE_COORD_ARRAY);

// Definir a semântica na altura de desenho
glBindBuffer(GL_ARRAY_BUFFER, texCoord);
glTexCoordPointer(2, GL_FLOAT, 0, 0);
```