Universidade do Minho
Escola de Engenharia
Mestrado Integrado em Engenharia Informática

Unidade Curricular de Computação Gráfica

Ano Letivo de 2016/2017

Computação Gráfica: Fase IV

Bruno Pereira (a75135), Maria Ana de Brito (a73580), Diogo Silva (a76407)

Maio, 2017

CG

Índice

1. Introdução 1
2. Generator 2
2.1. Normais das Primitivas Gráficas 2
2.1.1 Plano 2
2.1.2 Caixa 3
2.1.3 Anel 3
2.1.4 Cone 4
2.1.4.1 Base 4
2.1.4.2 Topo da pirâmide 4
2.1.4.3 Parte Lateral 5
2.1.5 Esfera 6
2.1.6 Teapot 7
2.2. Coordenadas de Textura das Primitivas Gráficas 8
2.2.1 Plano 8
2.2.2 Caixa 8
2.2.3 Anel 9
2.2.4 Cone 10
2.2.4.1 Base 10
2.2.4.2 Parte Lateral 11
2.2.5 Esfera 12
2.2.6 Teapot 12
3. Motor 14
3.1. Ficheiro XML 14
3.1.1 Estrutura do XML 14
3.1.2 Leitura do Ficheiro XML 15
3.2. Alteração da estrutura Model 16
3.3. Luzes 17
3.3.1 Point 17
3.3.2 Directional 18
3.3.3 Spotlight 18
3.4. Texturas 19

4. Cenas Demo 21
4.1. Sistema Solar 21
4.2. Final do Campeonato Europeu de Futebol de 2016 22
4.3. Farol 22
5. Conclusões e Trabalho Futuro 23

Índice de Figuras

Figura 1-Normais do plano 2
Figura 2-Box e os três eixos 3
Figura 3 – Plano 8
Figura 4 - Caixa 9
Figura 5 – Anel 9
Figura 6 - Estrutura da Imagem da Textura do Cone 10
Figura 7 - Base do Cone 11
Figura 8 - Parte Lateral do Cone 11
Figura 9 - Esfera 12
Figura 10 - Coordenadas de Textura do Teapot 13
Figura 11 - XML da luz POINT 14
Figura 12 - XML da luz DIRECTIONAL 14
Figura 13 - XML da luz SPOTLIGHT 14
Figura 14 - XML da emissividade do material 15
Figura 15 - XML da cor ambiente e difusa do material 15
Figura 16 - XML da câmara 15
Figura 17 - XML do ângulo de rotação 15
Figura 18 - Luz do tipo POINT 17
Figura 19 - Luz do tipo DIRECTIONAL 18
Figura 20 - Luz do tipo SPOTLIGHT 19
Figura 21-Sphere com a textura do Planeta Terra 20
Figura 22 - Sistema Solar 21
Figura 23 - Final do Campeonato Europeu de Futebol de 2016 22
Figura 24 - Farol 22

Índice de Tabelas

Tabela 1 - Vetores normais dos pontos dos triângulos 5

1. Introdução

A quarta e última fase do projeto prático consistiu em definir as normais das várias primitivas gráficas, assim como as suas coordenadas de textura. Além disso, implementaram-se, ainda, vários tipos de luzes que são representados nas cenas *demo* que o grupo criou. Deste modo, procurou-se ao máximo conceber cenas *demo* significantes, que mostrassem os aspetos mais relevantes do motor. Nesta fase também foram aplicadas várias texturas às primitivas geradas, de forma a tornar os modelos o mais realista possível.

Desta forma, o grupo, primeiramente, tratou de definir as normais para todas as primitivas geradas pelo *Generator*. De seguida, foram aplicadas as luzes, visando a correta definição das normais. A seguir, foram definidas as coordenadas de textura, assim como a aplicação da própria textura.

Assim, este relatório tem a seguinte estrutura: na secção 2, procede-se à explicação da definição das normais e das coordenadas de textura, na secção 3, explica-se as alterações feitas em algumas *structs*, as mudanças na estrutura do ficheiro XML, assim como a leitura do mesmo. Ainda, se refere o tipo de luzes usado e a sua implementação, bem como o processo de aplicação de texturas.

2. *Generator*

2.1. Normais das Primitivas Gráficas

Nesta quarta fase do trabalho prático, um dos objetivos era implementar iluminação. A iluminação não se reflete unicamente em colocar uma luz num determinado ponto, também tem de ter algum efeito sobre as primitivas gráficas. É através desta premissa que surgiu a motivação para implementar os vetores normais para cada ponto da primitiva gráfica, visto que através das normais é possível recriar a forma como a luz se comporta quando incide sobre um objeto.

2.1.1 Plano

O vetor normal de qualquer ponto do plano xOz é igual, independentemente do ponto, e perpendicular ao plano, ocorrendo uma alteração no seu sentido dependendo da posição do ponto no plano. Por exemplo, se o triângulo gerado for visível numa posição superior ao plano (utilizando a regra da mão direita), o vetor normal dos seus três pontos terá de apontar para cima. O contrário tem de se verificar para os triângulos gerados utilizando o sentido contrário ao da regra da mão direita, em que a normal dos seus pontos tem de apontar para baixo.

De seguida apresenta-se um plano xOz com a representação de dois vetores normais a esse mesmo plano.

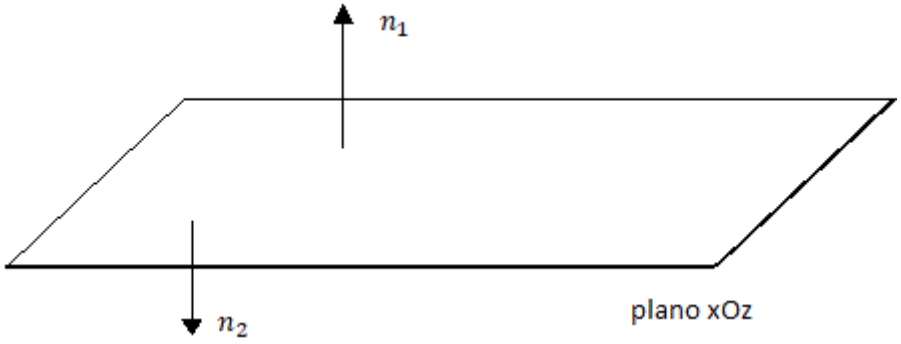

Figura 1-Normais do plano

- **Coordenadas do vetor** $\overrightarrow{n_1} = (0, 1, 0)$.
- **Coordenadas do vetor** $\overrightarrow{n_2} = (0, -1, 0)$.

2.1.2 Caixa

O vetor normal a um ponto na superfície da primitiva gráfica *box* depende da face da *box* em que o ponto se situa. Como o objetivo do cálculo das normais é aplicar uma luz à superfície da primitiva gráfica, só os triângulos das faces exteriores da *box* é que estão a ser gerados. Assim, para cada face, queremos calcular o vetor normal que aponta para fora da *box*.

Cada face do cubo é paralela a um plano do eixo tridimensional, logo o vetor normal de uma certa face do cubo será igual o vetor normal ao plano paralelo a essa face. No entanto, duas faces paralelas do cubo têm de apresentar sentidos contrários para os vetores normais à face. Assim, através da seguinte imagem que representa um cubo e os três eixos, conseguimos determinar as coordenadas do vetor normal para cada face do cubo.

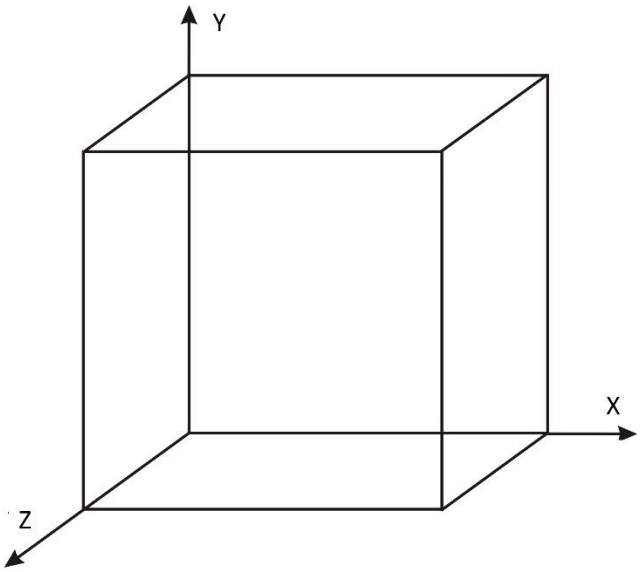

Figura 2-Box e os três eixos

Coordenadas dos vetores normais nas várias faces da *box*:

- **Face superior (paralela ao plano xOz):** $\vec{n} = (0,\ 1,\ 0)$
- **Face inferior (paralela ao plano xOz**): $\vec{n} = (0, -1,\ 0)$
- **Face da direita (paralela ao plano yOz):** $\vec{n} = (0,\ 0,\ 1)$
- **Face da esquerda (paralela ao plano yOz):** $\vec{n} = (0,\ 0, -1)$
- **Face da frente (paralela ao plano xOy**): $\vec{n} = (1,\ 0,\ 0)$
- **Face de trás (paralela ao plano xOy):** $\vec{n} = (-1,\ 0,\ 0)$

2.1.3 Anel

A primitiva gráfica *ring* situa-se no eixo xOz, logo qualquer vetor normal a esse plano tem dois valores possíveis: (0, 1, 0) ou (0, -1, 0). Assim sendo, temos que o vetor normal que aponta para cima tem as coordenadas (0, 1, 0), enquanto que o vetor normal que aponta para baixo possui as coordenadas (0, -1,0).

2.1.4 Cone

O cálculo dos vetores normais varia dependendo da zona do cone em que o ponto se situa. Assim, para facilitar a compreensão, analisar-se-ão as três situações em que o cálculo dos vetores normais varia.

2.1.4.1 Base

A base da primitiva gráfica *cone* é paralela ao eixo xOz, logo qualquer vetor normal a esse plano tem dois valores possíveis: (0, 1, 0) ou (0, -1, 0). Como a parte interior do cone não é visível, nós queremos que o vetor normal aponte para baixo. Assim sendo, para qualquer ponto da base, temos que a sua normal é $\overrightarrow{\boldsymbol{n}} = (0, -1, 0)$.

2.1.4.2 Topo da pirâmide

O topo da pirâmide é formado através de triângulos semelhantes aos da figura 3:

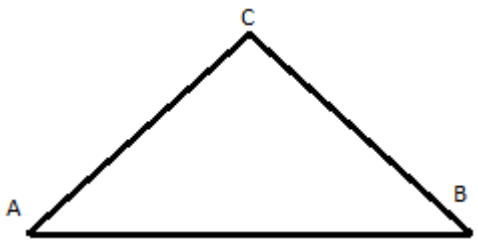

Figura 3-Triângulo que constitui o topo da pirâmide

Para cada ponto do triângulo é necessário determinar o seu vetor normal, que será obtido através do produto externo de dois vetores. Por exemplo, o vetor normal ao ponto C será determinado pela seguinte fórmula:

$$\overrightarrow{n_c} = \overrightarrow{CA} * \overrightarrow{CB}.$$

Neste caso, queremos que o vetor normal aponte para fora da primitiva gráfica *cone*, mas se quiséssemos que apontasse para dentro da primitiva, seria necessário inverter o sentido dos dois vetores, como exemplificado na seguinte fórmula

$\overrightarrow{n_c} = \overrightarrow{AC} * \overrightarrow{BC}.$

no entanto, esta fórmula será desconsiderada, visto que não tem relevância neste caso.

Mediante a fórmula mencionada, podemos determinar as normais para qualquer ponto do topo da pirâmide, através das seguintes equações.

- **Normal do ponto A:** $\overrightarrow{n_A} = \overrightarrow{AB} * \overrightarrow{AC}.$
- **Normal do ponto B:** $\overrightarrow{n_B} = \overrightarrow{BA} * \overrightarrow{BC}.$

- **Normal do ponto C:** $\overrightarrow{n_c} = \overrightarrow{CA} * \overrightarrow{CB}.$

2.1.4.3 Parte Lateral

A superfície lateral da primitiva gráfica *cone* é gerada através de uma sequência de triângulos, estando uma parte dessa sequência ilustrada na figura 4.

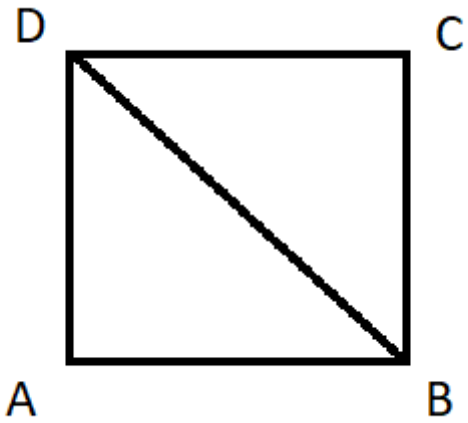

Figura 4- Secção lateral do cone

A estratégia para o cálculo do vetor normal para um determinado ponto da superfície da lateral do cone é a mesma que se utilizou no topo da pirâmide, ou seja, utiliza-se o produto externo de dois vetores cujo ponto inicial seja o ponto em que se deseje calcular o vetor normal. De seguida, apresenta-se uma tabela em que, mediante o triângulo em que o ponto se encontre, apresenta-se a fórmula utilizada para o cálculo do vetor normal desse ponto. Os pontos considerados são os pontos exemplificados na figura 4.

Tabela 1 - Vetores normais dos pontos dos triângulos

Ponto\Triângulo	[ABD]	[DBC]
A	$\vec{n} = \overrightarrow{AB} * \overrightarrow{AD}$	----
B	$\vec{n} = \overrightarrow{BD} * \overrightarrow{BA}$	$\vec{n} = \overrightarrow{BC} * \overrightarrow{BD}$
C	----	$\vec{n} = \overrightarrow{CD} * \overrightarrow{CB}$
D	$\vec{n} = \overrightarrow{DA} * \overrightarrow{DB}$	$\vec{n} = \overrightarrow{DB} * \overrightarrow{DC}$

2.1.5 Esfera

Para qualquer ponto P na superfície esférica e para o ponto C no centro da esfera, o vetor normal do ponto P pode ser calculado através do cálculo do vetor $\overrightarrow{CP} = P - C$. Como C é o ponto no centro da esfera e a esfera está centrada na origem do referencial, obtemos a seguinte fórmula: $\overrightarrow{CP} = P$

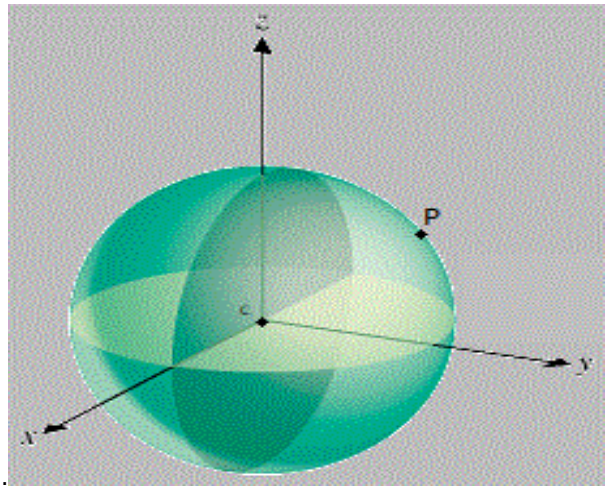

Figura 5-Sphere centrada na origem e os pontos P e C

Assim, o vetor $\overrightarrow{CP}$ é igual ao ponto P, e através da seguinte figura, podemos encontrar as coordenadas do ponto P mediante a um ângulo alfa e beta e o raio da esfera.

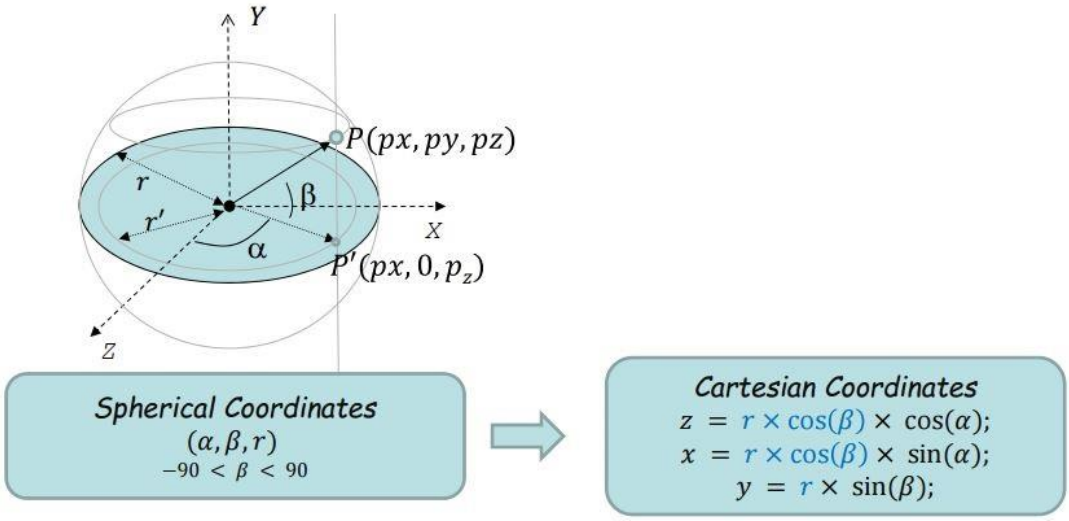

Figura 6- Representação de dois pontos P e C, os seus ângulos e a transformação de coordenadas esféricas em cartesianas

Coordenadas do ponto P:

$$\begin{cases} P_x = raio * \cos(beta) * \sin(alfa) \\ P_y = raio * \sin(beta) \\ P_x = raio * \cos(beta) * \cos(alfa) \end{cases}$$

Normalizando o vetor $\overrightarrow{CP}$, obtemos as seguintes coordenadas para o vetor normal:

$$\begin{cases} P_x = \cos(beta) * \sin(alfa) \\ P_y = \sin(beta) \\ P_x = \cos(beta) * \cos(alfa) \end{cases}$$

Assim, através destas coordenadas pode-se calcular o vetor normal de qualquer ponto, desde que a esfera se encontre centrada na origem.

2.1.6 Teapot

O cálculo dos vetores normais da primitiva gráfica *teapot* é feito através do produto externo dos vetores resultantes da derivação da matriz B por u e v. Assim sendo, temos que

$$\overrightarrow{n_{teapot}} = \frac{\partial B(u,v)}{\partial u} * \frac{\partial B(u,v)}{\partial v} ,$$

em que

$$\frac{\partial B(u,v)}{\partial u} = [3u^2 \quad 2u \quad 1 \quad 0] M \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} M^T V^T$$

e

$$\frac{\partial B(u,v)}{\partial v} = UM \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} M^T \begin{bmatrix} 3v^2 \\ 2v \\ 1 \\ 0 \end{bmatrix}$$

Sabemos que o vetor u e v são criados através do valor de u e de v passados como parâmetro aquando o *teapot* é desenhado (sendo eles as divisões de cada *patch*), M é a seguinte matriz:

$$M = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

sendo M^T, M transposta e P são os pontos de uma determinada *patch.*

Denote-se que para um determinado ponto, é necessário calcular três vezes as derivadas de B para cada uma das coordenadas.

2.2. Coordenadas de Textura das Primitivas Gráficas

2.2.1 Plano

No caso do plano, interessa que as coordenadas de textura coincidam com os cantos da figura que contem a textura em si. Deste modo, como o plano é assim desenhado:

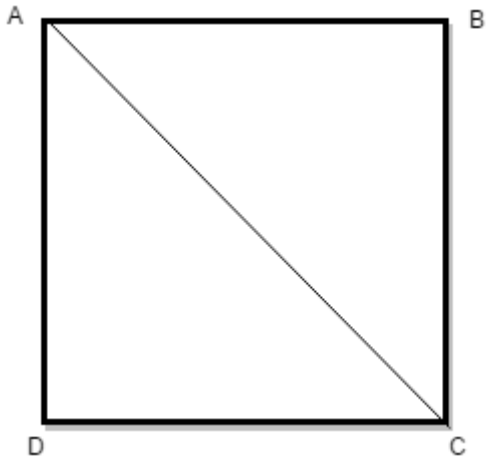

Figura 3 – Plano

As coordenadas da textura possuem os seguintes valores: para o vértice A, (0,1), para o B, (1,1), para o C (1,0) e, finalmente, para o D, (0,0). Como se pode verificar, desta maneira, a imagem é mapeada de acordo com a forma do plano definido.

2.2.2 Caixa

Para a caixa, ou *box*, o grupo decidiu que a mesma imagem da textura se iria repetir ao longo das seis faces. Assim, se tivermos uma imagem com uma determinada textura, podemos verificar que esta é repetida em todas as faces da caixa. Deste modo, atendendo ao número de

divisões de cada uma das faces, tivemos que dividir o comprimento da imagem (que é 1) pelo número de divisões, de forma a fazer o mapeamento corretamente. Tal como se pode verificar, as coordenadas de textura dependem do valor da divisão, assim como do j e do i, que representam a largura (parte vertical) e o comprimento (parte horizontal). Deste modo, o vértice A é $(j * rt, i * rt)$, o B é $(j * rt, (i + 1) * rt)$, o C é $((j + 1) * rt, i * rt)$ e o D é $((j + 1) * rt, (i + 1) * rt)$.

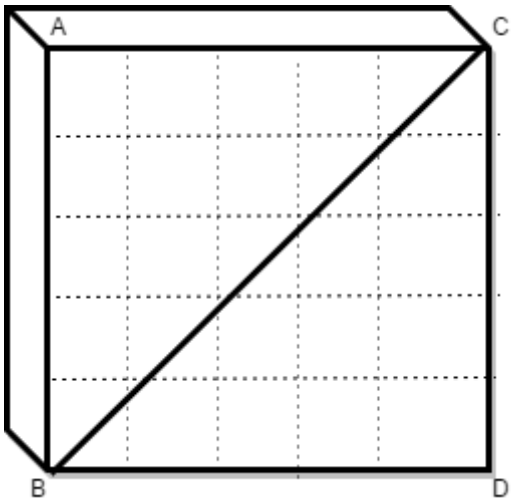

Figura 4 - Caixa

2.2.3 Anel

A primitiva gráfica anel é um círculo sem o interior, tal como se pode verificar na figura 5. Deste modo, a textura será mapeada para cada dois triângulos (tal como foi feito para o plano), repetidamente ao longo do comprimento total do anel.

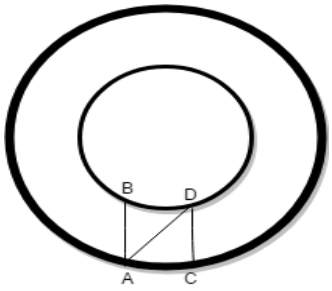

Figura 5 – Anel

De forma a dar continuidade à imagem, as coordenadas de textura são:

A = (0, 0)

B = (0, 1)

C = (1, 0)

D = (1, 1)

2.2.4 Cone

A imagem de textura do cone difere das restantes, uma vez que tem de diferenciar a textura da base e a textura da parte lateral. Deste modo, foi decidido pelo grupo que a estrutura dessa imagem seria a seguinte:

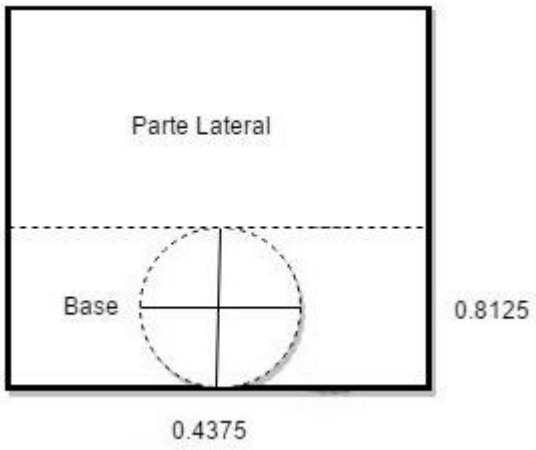

Figura 6 - Estrutura da Imagem da Textura do Cone

2.2.4.1 Base

Para o caso em que o número de *slices* é 1, a textura da base é mapeada como se fosse um triângulo circunscrito numa circunferência, uma vez que a forma da base é também um triângulo. As coordenadas são calculadas através das fórmulas:

$$s = 0.4375 + 0.1875 * \sin(i * \alpha)$$

$$t = 0.8125 + 0.1875 * \cos(i * \alpha)$$

Por outro lado, quando o número de *slices* é 2, temos que a base tem a forma de um quadrado, logo é mapeada como se fosse um quadrado circunscrito numa circunferência. As coordenadas são calculadas através das fórmulas:

$$A = (0.4376 + 0.8125 * \sin(i * \alpha)\,, 0.8125 + 0.8125 * \cos(i * \alpha))$$

$$B = (0.4376 + 0.8125 * \sin(i * \alpha - \alpha)\,, 0.8125 + 0.8125 * \cos(i * \alpha - \alpha))$$

$$C = (0.4376 + 0.8125 * \sin(i * \alpha - 2 * \alpha)\,, 0.8125 + 0.8125 * \cos(i * \alpha - 2 * \alpha))$$

Por fim, quando o número de *slices* é igual ou maior do que 3, temos que percorrer a circunferência da textura da base, segundo as fórmulas:

$$A = (0.8125 + 0.4375 * sin(i * \alpha - \alpha), 0.4375 + 0.4375 * cos(i * \alpha - \alpha)$$

$$B = (0.8125 + 0.4376 * sin(i * \alpha), 0.4375 + 0.4375 * cos(i * \alpha)$$

$$C = (0.8125, 0.4375)$$

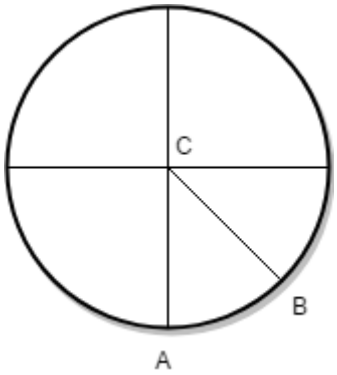

Figura 7 - Base do Cone

2.2.4.2 Parte Lateral

Como a origem da imagem encontra-se no canto superior esquerdo, o seu mapeamento é feito tendo em conta o número de *slices* e *stacks*. Deste modo, ao contrário do que acontece na base, em que é preciso ter em conta a posição da textura da base dentro da própria imagem (uma vez que não se encontra na origem), não é necessário ter esses cuidados.

Assim, as coordenadas são calculadas através das fórmulas:

$$A = (j * rslices, i * rstacks)$$

$$B = (j * rslices, (i + 1) * rstacks)$$

$$C = ((j + 1) * rslices, (i + 1) * rstacks)$$

$$D = ((j + 1) * rslices, i * rstacks)$$

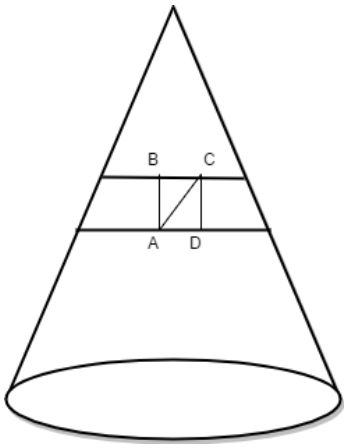

Figura 8 - Parte Lateral do Cone

2.2.5 Esfera

De forma a fazer o mapeamento da imagem da textura, foi preciso atender ao número de *slices* e *stacks* e, assim, dividir o comprimento da imagem (que é 1) por esses valores. É através dos valores obtidos desta divisão que a aplicação da textura é feita corretamente. Deste modo, as fórmulas das coordenadas de textura são as seguintes:

$$A = (j * rslices, i * rstacks)$$

$$B = ((j + 1) * rslices, i * rstacks)$$

$$C = ((j + 1) * rslices, (i + 1) * rstacks)$$

$$D = (j * rslices, (i + 1) * rstacks)$$

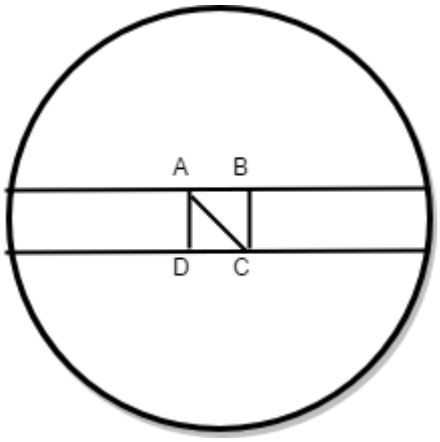

Figura 9 - Esfera

2.2.6 *Teapot*

O *teapot* está dividido em divisões ao longo do *u* e do *v*, que vão permitir calcular as coordenadas de textura. Assim, temos quatro variáveis *uu* e *vv,* inicializadas a zero e que são incrementadas no fim de cada ciclo pelas variáveis *u* e *v*, que, por sua vez, contêm o resultado da divisão de 1 pelo número de divisões em *u* e *v*, respetivamente. Deste modo, as coordenadas de textura são calculadas do seguinte modo:

A = (uu, vv)

B = (uu + u, vv)

C = (uu, vv + v)

D = (uu + u, vv + v)

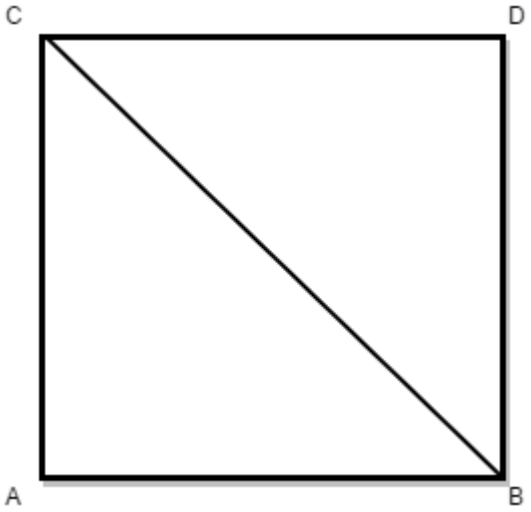

Figura 10 - Coordenadas de Textura do *Teapot*

3. Motor

3.1. Ficheiro XML

3.1.1 Estrutura do XML

Para esta quarta fase do projeto prático, foi necessária a adição de novas etiquetas de XML de modo a implementar as novas funcionalidades do programa, tais como a aplicação de texturas e a existência de vários tipos de luzes.

Desta forma, em relação à iluminação, temos um grupo chamado *lights*, onde são declaradas todas as luzes da cena *demo*. Para a luz do tipo *POINT* é preciso indicar as coordenadas x, y e z da luz. Da mesma forma, para uma luz *DIRECTIONAL* é preciso indicar, igualmente, a sua posição. Por outro lado, para uma luz do tipo *SPOTLIGHT*, enquanto também é necessário indicar a posição em que se encontra, também é requerido que sejam referidas a direção do feixe de luz, assim como o seu ângulo e o seu expoente.

```
<lights>
    <light type="POINT" posX=0 posY=5 posZ=0 ambR=0 ambG=0 ambB=0 difR=1 difG=1 difB=1/>
</lights>
```

Figura 11 - XML da luz *POINT*

```
<lights>
    <light type="DIRECTIONAL" posX=20 posY=15 posZ=0 ambR=0 ambG=0 ambB=0 difR=1 difG=1 difB=1/>
</lights>
```

Figura 12 - XML da luz *DIRECTIONAL*

```
<lights>
    <light type="SPOTLIGHT" posY=5 dirX=1 dirZ=0 angle=45 exp=10 ambR=0 ambG=0 ambB=0 difR=1 difG=1 difB=1/>
</lights>
```

Figura 13 - XML da luz *SPOTLIGHT*

A textura e caracterização do material das primitivas gráficas também são indicadas no ficheiro XML, logo a seguir ao nome do ficheiro que contém os pontos a serem lidos. Deste modo, se o material for emissivo, é preciso indicar os valores de R, G e B da intensidade de luz emitida. Caso se pretenda indicar a cor ambiente ou difusa do material é também preciso referir os seus valores de R, G e B, tal como se pode verificar nos seguintes exemplos:

```
<models>
    <model file="sun.3d" texture="sun.jpg" emiR=1 emiG=1 emiB=1/>
</models>
```

Figura 14 - XML da emissividade do material

```
<models>
    <model file="mercury.3d" texture="mercury.jpg" ambR=1 ambG=1 ambB=1 difR=1 difG=1 difB=1/>
</models>
```

Figura 15 - XML da cor ambiente e difusa do material

A criação de novas cenas *demo* levou à necessidade de introduzir um novo grupo chamado *camera*, que indica a posição da câmara em coordenadas polares. Assim, armazena-se aqui os ângulos α e β, assim como o seu raio.

```
<camera>
    <position alpha=0 beta=22 radius=560/>
</camera>
```

Figura 16 - XML da câmara

Relativamente à rotação, anteriormente o ângulo por defeito era 360º, ou seja, uma rotação era feita ao longo dos 360º, agora cabe ao utilizador decidir qual o ângulo que pretende para a rotação. Caso não seja indicado nenhum ângulo, o ângulo por defeito continua em vigor.

```
<rotate time=10 rotationAngle=90 axisZ=1/>
```

Figura 17 - XML do ângulo de rotação

3.1.2 Leitura do Ficheiro XML

A leitura do tipo da luz, assim como a dos seus atributos, é feita na função *loadLights*. Primeiramente, é inicializada uma *struct* chamada *Light*, que armazenará o tipo da luz e as suas características (posição, direção do feixe, ângulo e expoente, quando aplicável). De seguida, são lidos os vários atributos (como, por exemplo, *type, posX, posY, posZ, dirX, dirY, dirZ, angle* e *exp)*, caso existam. Se o tipo de luz for *POINT* ou *SPOTLIGHT*, a quarta

coordenada será um 1 (indicando que é um ponto), caso contrário (o tipo for *DIRECTIONAL*), a quarta coordenada será um 0 (indicando que é uma direção).

Já o ficheiro que contém a textura e as características da cor do material (cor de ambiente, cor difusa e cor emissiva) são lidas na função *loadGroups*, juntamente com o ficheiro que contém os pontos da primitiva a ser desenhada. Caso não sejam indicados estes atributos, o seu valor por defeito é 0. O novo ângulo de rotação acima mencionado também é lido nesta função, sendo o seu valor por defeito 360º.

Por fim, posição da câmara é lida na função *loadCamera*, que trata de recolher os dados acerca dos ângulos e do raio. Logo de seguida, é chamada uma nova função *calculateCam*, que calcula os novos valores da posição da câmara, através das formulas:

$$x = r * \sin\left(\propto * \frac{3.14}{180.0}\right) * \cos(\beta * \frac{3.14}{180.0})$$

$$y = r * \cos\left(\propto * \frac{3.14}{180.0}\right) * \cos(\beta * \frac{3.14}{180.0})$$

$$z = r * \sin(\beta * \frac{3.14}{180.0})$$

3.2. Alteração da estrutura Model

```
struct Model
{
	string texture;
	int texID;
	int numberPoints[20];
	Buffers buffers[20];
	LightComp lc[20];
};
```

A estrutura *Model*, contida na estrutura *Group*, sofreu alterações relativamente à fase anterior. A primeira alteração foi a mudança da estrutura *Buffers*, que continha unicamente um *GLuint*(que continha os pontos de uma primitiva gráfica), para um *array* de três *GLuint*: um para os pontos da primitiva, o segundo para as coordenadas dos vetores normais e o terceiro para as coordenadas das texturas.

A segunda alteração surgiu devido à adição da iluminação nesta fase e foi a introdução de um *array,* de 20 elementos, com a estrutura *LightComp*. O *array* possui vinte unidades, porque foi definido como limite máximo de desenho de modelos.

```
struct LightComp{
	float ambR, ambG, ambB;
	float difR, difG, difB;
	float emiR, emiG, emiB;
};
```

A estrutura *LightComp* apresenta as componentes da luz para um determinado modelo, sendo ela constituída por nove float's:

- valores para descrever a cor de ambiente (RGB);
- valores para descrever a cor difusa (RGB);
- valores para descrever a cor emissiva (RGB).

Relativamente às texturas, a estrutura *Model* apresenta uma *string* com o nome da textura, que foi lido no ficheiro XML, e que indica qual o ficheiro onde se deve ler a textura. Além da *string*, foi adicionado também um valor inteiro que representa o identificador da textura correspondente ao modelo.

3.3. Luzes

3.3.1 Point

A luz do tipo *POINT* caracteriza-se por ter um ponto e não uma direção (ao contrário das luzes do tipo *DIRECTIONAL*), ou seja, a direção da luz não é constante em todos os vértices das primitivas gráficas desenhadas, uma vez que o ponto irradia luz por todas as suas direções.

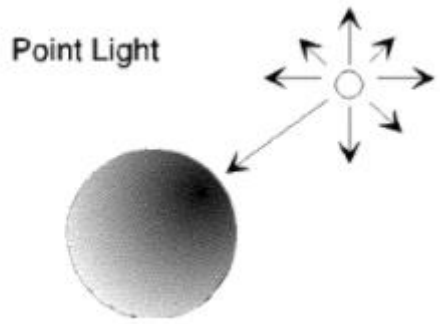

Figura 18 - Luz do tipo *POINT*

Deste modo, podemos caracterizar uma luz deste tipo com a sua posição, a cor de ambiente e cor difusa. Estes atributos são ativos na função *enableLighting*, que trata de chamar

a função *glLightfv* para cada uma destas características (*GL_POSITION, GL_AMBIENT, GL_DIFFUSE*). É de salientar que a quarta coordenada do vetor que indica a posição da luz é um 1, ou seja, é um ponto.

3.3.2 Directional

As luzes do tipo *DIRECTIONAL* são caracterizadas por terem uma direção, ou seja, a sua direção é constante para todos os vértices das primitivas.

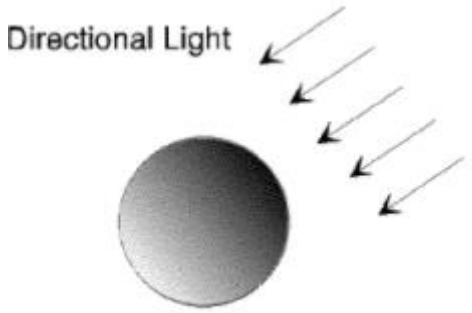

Figura 19 - Luz do tipo *DIRECTIONAL*

As características desta luz são ativas do mesmo modo que a luz do tipo *POINT*. É de referir que a quarta coordenada do vetor da posição da luz é um 0, uma vez que se trata de uma direção.

3.3.3 Spotlight

Uma luz do tipo *SPOTLIGHT* é semelhante a uma luz do tipo *POINT*, com a diferença que apenas é irradiada num conjunto de direções muito restrito. Tem como atributos a sua posição, que indica o local onde a luz se encontra, a direção do *spot*, que se trata de um vetor que define a direção do feixe de luz, o ângulo, que representa a “abertura” da luz (se virmos este tipo de luz como se fosse um cone, onde o vértice do topo é a posição da luz, podemos considerar este ângulo como se fosse a “abertura” do cone) e o expoente, que indica a intensidade da distribuição da luz (se o seu valor for 0, a distribuição é uniforme, caso seja um valor mais alto, mais concentrada será a luz).

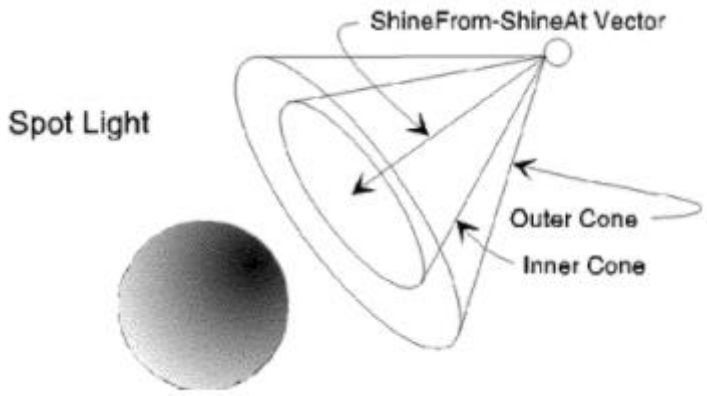

Figura 20 - Luz do tipo *SPOTLIGHT*

As características desta luz são ativas na função *enableLighting*, através da função *glLightfv*, que utiliza as *flags*: *GL_POSITION, GL_SPOT_DIRECTION, GL_SPOT_CUTOFF* e *GL_SPOT_EXPONENT*. A quarta coordenada da posição da luz é um 1, uma vez que se trata de um ponto.

3.4. Texturas

Após as coordenadas das texturas terem sido calculadas pelo *Generator*, o *Motor* deverá aplica-las à primitiva gráfica correspondente.

O primeiro passo será armazenar as coordenadas lidas do ficheiro cujo nome se encontra no ficheiro XML num VBO especificado para os pontos das texturas. De seguida, de acordo com o nome contido no ficheiro XML, far-se-á *bind* à textura, através da função *bindTexture*, do ficheiro que contém a imagem que se quer aplicar à primitiva gráfica. Esta função retornará um identificador da textura que será armazenada no modelo correspondente para ser utilizada futuramente.

Antes de ocorrer o desenho das primitivas gráficas, é necessário ativar as texturas de duas dimensões (*glEnable(GL_TEXTURE_2D)*) e, antes da primitiva gráfica ser desenhada, é necessário ligar a primitiva gráfica à textura correspondente e isto será efetuado através do identificador da textura armazenada para cada modelo. Com a imagem e as coordenadas da textura, é possível aplicar a textura à primitiva gráfica através do método *glTexCoordPointer*.

Para finalizar, após ter sido desenhada a primitiva gráfica, é necessário ligar a textura atual a um identificador que representa um valor nulo, para que a próxima primitiva gráfica, caso não tenha textura, não surja com a textura da primitiva anterior. O valor do identificador nulo é zero.

Figura 21-Sphere com a textura do Planeta Terra

4. Cenas Demo

4.1. Sistema Solar

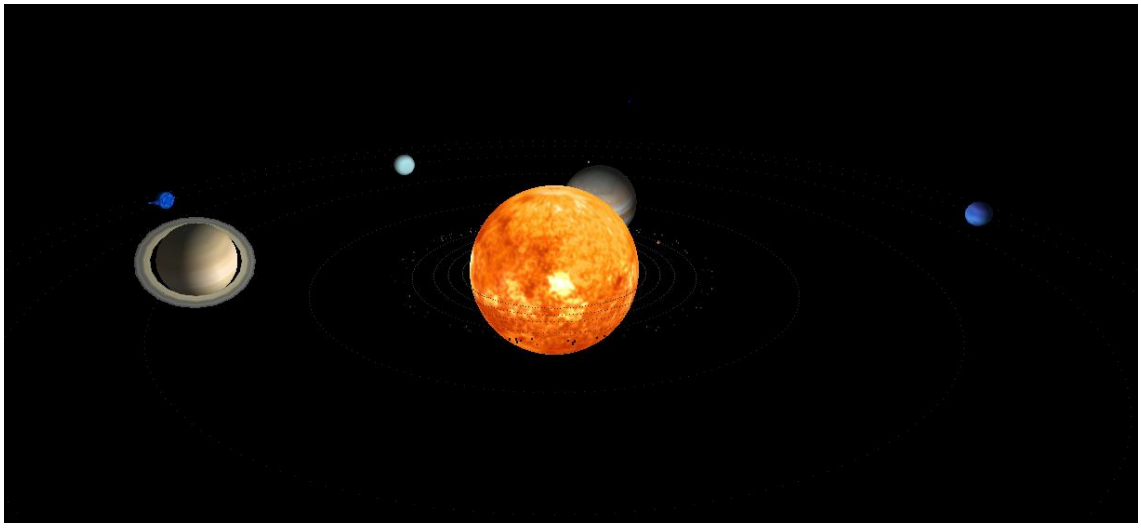

Figura 22 - Sistema Solar

4.2. Final do Campeonato Europeu de Futebol de 2016

Figura 23 - Final do Campeonato Europeu de Futebol de 2016

4.3. Farol

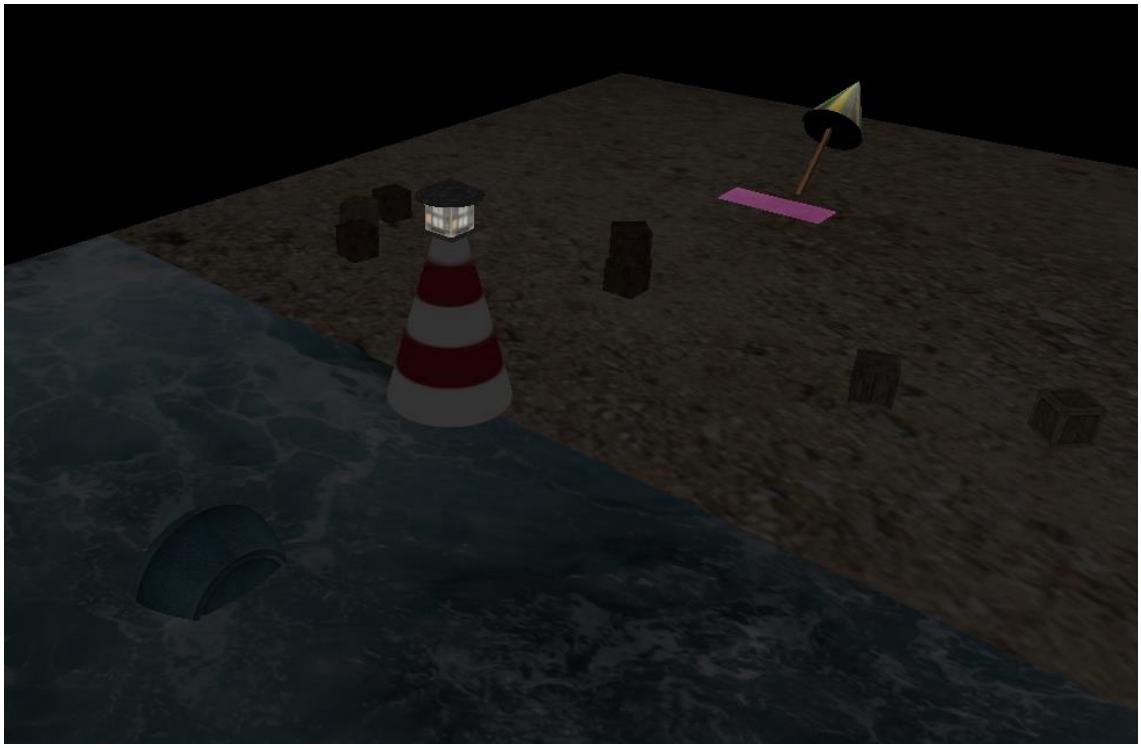

Figura 24 - Farol

5. Conclusões e Trabalho Futuro

Terminada a quarta e última fase do projeto prático da Unidade Curricular de Computação Gráfica, o grupo considera que os objetivos propostos pela equipa docente para esta fase e para as anteriores foram atingidos com sucesso.

O grupo conseguiu implementar as funcionalidades requeridas e, ainda, procurou melhorá-las, de modo a enriquecer o projeto prático. Nesta fase, foram aplicadas as texturas às primitivas geradas e foram criadas as luzes para iluminar os elementos das cenas *demo*. Isto foi possível graças à definição das normais dos vértices, assim como às coordenadas de textura das primitivas. É de referir que foram definidas para todas as primitivas geradas pelo *Generator*.

Além de compor a demo *scene* proposta, o Sistema Solar, o grupo criou, ainda, mais duas cenas adicionais: a final do Campeonato Europeu de Futebol de 2016 e a *Lighthouse*, que demonstram claramente várias texturas aplicadas e os tipos de luzes usados.

Referências

Ramires, António B., 2015. GLSL Tutorial – Point Lights. *Lighthouse3d.com Computer Graphics blog*. Available at: < http://www.lighthouse3d.com/tutorials/glsl-tutorial/point-lights/> [Accessed 20 May 2017]

Ramires, António B., 2015. GLSL Tutorial - Spotlights. *Lighthouse3d.com Computer Graphics blog*. Available at: < http://www.lighthouse3d.com/tutorials/glsl-tutorial/spotlights/> [Accessed 20 May 2017]

Available at: https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/glMaterial.xml [Accessed 13 May 2017]

Available at: < https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/glLight.xml> [Accessed 13 May 2017]