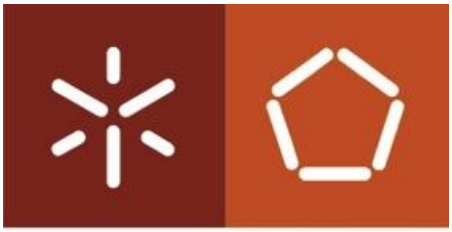


Universidade do Minho

Computação Gráfica

Relatório Projeto Prático – Parte IV

Mestrado Integrado em Engenharia Informática

Ano Letivo 2016/2017

Docente: António José Borba Ramires Fernandes

Elementos do Grupo:

Daniel Teixeira Militão – A74557

Hugo Alves Carvalho – A74219

João Ismael Barros Dos Reis – A75372

Luís Miguel Da Cunha Lima – A74260

Índice

1. Introdução

O presente relatório descreve a segunda etapa do projeto prático da unidade curricular de Computação Gráfica.

Nesta etapa, para além dos vértices que compõem os vários modelos, foi pedido que o *generator* produzisse, as normais e as coordenadas de textura correspondentes a cada um dos vértices das figuras.

Ainda nesta etapa, o *engine* deve ativar as funcionalidades criadas pelo *generator*, como a iluminação, textura e a leitura das normas e coordenadas de textura, de forma a serem produzidas cenas com figuras iluminadas e que possuam texturas e cores. Para isto, foi importante adaptar a leitura do ficheiro XML e dos ficheiros 3d de maneira a cumprir os objetivos desta etapa.

Neste relatório vamos explicar todo o raciocínio para a realização dos novos aspetos acima apresentados.

2. Generator

2.1. Normais

2.1.1. Plano

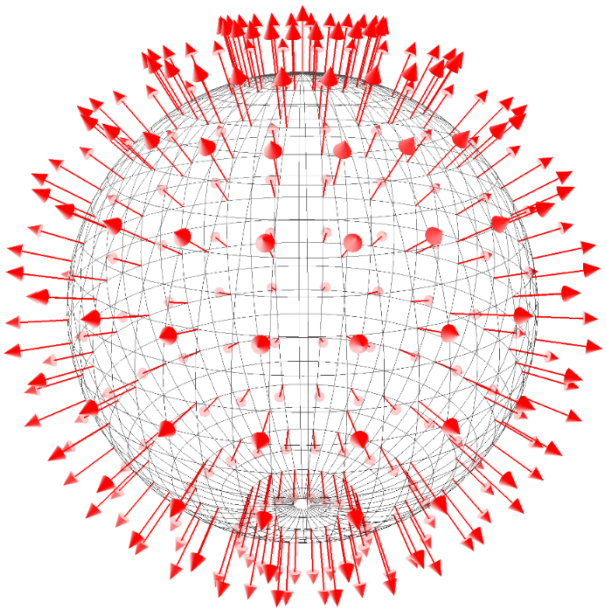
No nosso caso, as normais para qualquer ponto do plano são sempre as mesmas: (0,1,0).

2.1.2. Esfera

Nesta fase, a cada descoberta de um ponto da esfera, foi igualmente descoberto a sua normal relacionada. Inicialmente, foi reconhecida a normal do vértice da base da esfera. Uma vez que a esfera tem a base no eixo negativo do eixo Y, a normal para este ponto é (0,-1,0).

Em relação á parte do meio da esfera, as normais de cada vértice correspondem à normalização de cada coordenada do respetivo ponto. Por exemplo, dado um ponto p=(1,2,3) e o raio da esfera (raio), a normal relacionada corresponde a n=(1/raio,2/raio,3/raio) .

Relativamente ao ultimo vértice da esfera, correspondente ao topo da esfera, a normal correspondente é (0,1,0) uma vez apontar para o eixo Y positivo. Na figura seguinte são demonstradas todas as normais de uma esfera.



2.1.3. Teapot

Para calcular aas normais para os pontos do *teapot* foi necessário, primeiramente, calcular dois vetores tangentes a esse ponto. Seguidamente, ao calcularmos o produto externo desses dois vetores, obtemos o vetor pretendido (que corresponde à normal).

De modo a obter as duas tangentes ao ponto pretendido, procedemos ao cálculo das derivadas em ordem a *u* e a *v* da função que permite calcular os pontos num patch de *Bezier*.

Assim, foram utilizadas as fórmulas dos polinómios de Bernstein:

$$
\begin{aligned}
k'_1(t) &= -3(1-t)^2 \\
k'_2(t) &= 3(1-t)^2 - 6t(1-t) \\
k'_3(t) &= 6t(1-t) - 3t^2 \\
k'_4(t) &= 3t^2
\end{aligned}
$$

De seguida, foram então utilizadas as fórmulas para calcular o vetor tangentes a *u* e a *v*.

$$\sum_j k'_j(u) \times \sum_i P_{ji} \times k_i(v)$$

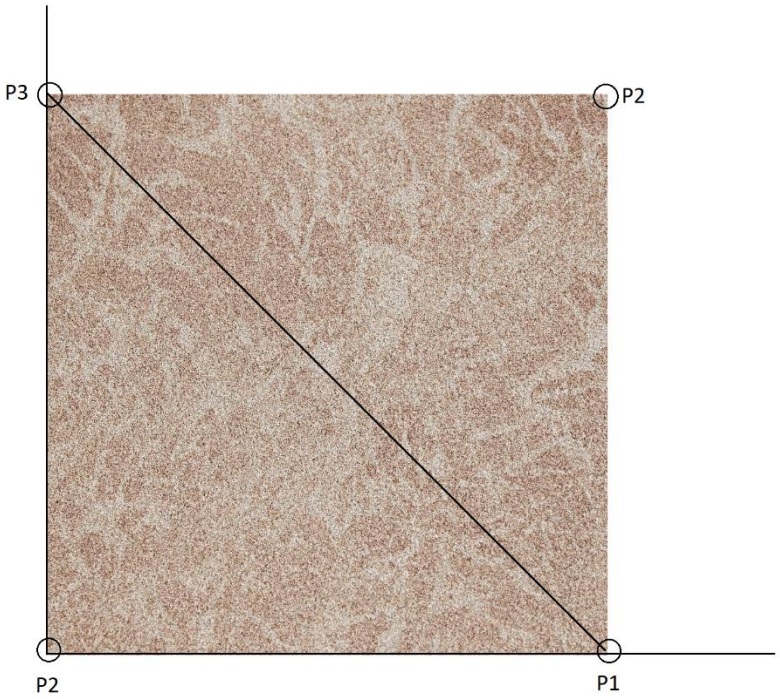
$$\sum_j k'_j(v) \times \sum_i P_{ij} \times k_i(u)$$

Depois de calculados os dois vetores tangentes, aplicamos o produto externo e obtemos a normal pretendida.

2.2. Coordenadas de Textura

2.2.1. Plano

Um plano tem apenas quatro coordenadas de textura sendo demonstrado na seguinte figura essas coordenadas.



2.2.2. Esfera

De maneira a encontrar as coordenadas de textura para a esfera foi desenvolvido um esboço de uma textura. A textura é repartida no número de *stacks* e *slices* que são desejadas, sendo que a parte do meio as coordenadas de textura têm um comportamento distinto uma vez que as texturas são retangulares.

As coordenadas de textura da parte do meio são simples de identificar, basta percorrer uma dada *stack* e a cada *slice* registar a sua coordenada de textura.

A parte de baixo e de cima da esfera é composta somente por triângulos, sendo necessário que as correspondentes coordenadas de textura não sigam o mesmo padrão da parte do meio. Estas partes podem ser consideradas como a primeira *stack* e a última, em cada *slice* destas *stacks* é calculado o ponto medio de forma a poder ter um triângulo.

Na seguinte figura é demonstrada uma textura com todos os pontos de textura assinalados, salientar a base e o topo que são divididos em triângulos de maneira a poder ter uma esfera com o menor erro possível.

A próxima fase passou pela descoberta da sequência correta das coordenadas de textura a fim de calcular os diversos triângulos com as porções de textura correspondentes.

2.2.3. Teapot

O método para conseguir as coordenadas de textura para um ponto do teapot consiste no uso dos valores de u e de v do ponto em questão. Estes valores correspondem às coordenadas na imagem da textura (x e y) que devem ser utilizadas para esse ponto.

Posto isto, o pseudo-código seguinte ilustra a geração dos pontos do teapot, das normais e das coordenadas de textura, bem como a sua escrita no ficheiro do modelo.

```
for (int patch_number = 0; patch_number < 32; patch_number++) {
    for (int tesselation_v = 0; tesselation_v < tesselationLevel; tesselation_v++) {
        float v = (float)tesselation_v / tesselationLevel;
        for (int tesselation_u = 0; tesselation_u < tesselationLevel; tesselation_u++) {
            float u = (float)tesselation_u / tesselationLevel;
            // first triangle
            points.push_back(bezierPoint(patch_number, u, v));
            points.push_back(bezierPoint(patch_number, (u + (1.0f / tesselationLevel)), (v + (1.0f / tesselationLevel))));
            points.push_back(bezierPoint(patch_number, (u + (1.0f / tesselationLevel)), v));

            // second triangle
            points.push_back(bezierPoint(patch_number, (u + (1.0f / tesselationLevel)), (v + (1.0f / tesselationLevel))));
            points.push_back(bezierPoint(patch_number, u, v));
            points.push_back(bezierPoint(patch_number, u, (v + (1.0f / tesselationLevel))));

            // first normal point
            normal.push_back(bezierPointNormal(patch_number, u, v));
            normal.push_back(bezierPointNormal(patch_number, (u + (1.0f / tesselationLevel)), (v + (1.0f / tesselationLevel))));
            normal.push_back(bezierPointNormal(patch_number, (u + (1.0f / tesselationLevel)), v));

            // second normal point
            normal.push_back(bezierPointNormal(patch_number, (u + (1.0f / tesselationLevel)), (v + (1.0f / tesselationLevel))));
            normal.push_back(bezierPointNormal(patch_number, u, v));
            normal.push_back(bezierPointNormal(patch_number, u, (v + (1.0f / tesselationLevel))));

            // first texture Points
            texture_points.push_back(TexturePoint(u, v));
            texture_points.push_back(TexturePoint(u + (1.0f / tesselationLevel), (v + (1.0f / tesselationLevel))));
            texture_points.push_back(TexturePoint(u + (1.0f / tesselationLevel), v));

            // second texture point
            texture_points.push_back(TexturePoint(u + (1.0f / tesselationLevel), (v + (1.0f / tesselationLevel))));
            texture_points.push_back(TexturePoint(u, v));
            texture_points.push_back(TexturePoint(u, (v + (1.0f / tesselationLevel))));
        }
    }
}
```

```
// Print the number of triangles into the file
fprintf(f, "%d\n", points.size() / 3);

// Print every point into the file
for (int point = 0; point < points.size() && point < normal.size() && point < texture_points.size(); point++) {
    // Triangle point
    fprintf(f, "%f %f %f\n", points[point].getX(), points[point].getY(), points[point].getZ());
    // Normal point
    fprintf(f, "%f %f %f\n", normal[point].getX(), normal[point].getY(), normal[point].getZ());
    // Texture point
    fprintf(f, "%f %f\n", texture_points[point].getX(), texture_points[point].getY());
}
```

3. Engine

3.1. Estruturas de Dados

Na última fase da engine será necessário trabalhar com luzes, cores e texturas de uma figura o que levou a uma aprimoração das estruturas já existentes. Todas estas componentes estarão especificadas no ficheiro XML anteriormente criado.

No que diz às luzes diz respeito, foi criada uma classe que contem as variáveis que a si lhe caracterizam:

```
class Color {
public:
	Color() : diffuseColor(NULL), specularColor(NULL), emissiveColor(NULL),
	ambientColor(NULL), empty(true){}
	bool empty;
	float *diffuseColor,
		*specularColor,
		*emissiveColor,
		*ambientColor;
};
```

Sabendo que as quatro componentes que definem uma cor são *diffuse*, specular, *emissive* e *ambient* estas foram representadas por Array de `float` representado os quatro valores RGBA, tendo valor inicial nulo. Consequentemente, visto que uma figura poderá ter cor, passa a ser uma das suas componentes a classe *Color*.

Passando para estruturação duma luz, foi também criada uma classe com as diversas componentes:

```
class Light {
public:
	Light() :cons(NULL), quad(NULL), linear(NULL), spotCutOff(NULL),
spotDirection(NULL),spotExponent(NULL) {}
	string type;
	float pos[4];
	Color color;
	float cons, quad, linear
		,spotCutOff, *spotDirection, spotExponent;
	unsigned int id;
};
```

A variável `type` indica o tipo de luz (*point*, *spotlight* ou *directional*), `pos` a posição no referencial, `color` a sua cor, `cons`, `quad` e `linear` a sua atenuação de uma luz *pointlight* e as três caractristicas de uma luz *spotlight* (`spotCutOff`, `spotDirection`, `spotExponent`). Finalmente, id refere-se a número da luz.

De modo a representar uma textura, uma figura tem na sua estrutura o índice VBO na sua estrutura, da mesma que contem as normais e os vértices:

```
class Figure {
public:
	Figure() : vboIndex(-1), triangles(0), textureIndexVBO(-1) {}
	string name;
	int vboIndex, normalIndex, textureIndex, textureIndexVBO, triangles;
	Color color;
	Scale scale;
	Rotation rotate;
	Translate translate;
};
```

3.2. Ciclo de Leitura

O ciclo de leitura, como anteriormente implementado nas anteriores fases faz-se de maneira recursiva ao longo das várias *tags* existentes ao longo do ficheiro XML. Uma luz no ficheiro XML pode indicar a posição, cores e respetivas características previamente mencionadas.

```
<lights>
	<light type="POINT" posX="0" posY="0" posZ="0" diffR="10" diffG="10"
diffB="10" ambR="0.2" ambG="0.2" ambB="0.2" />
</lights>
```

Será também importante referir que existe uma variável global `light_number` que regista a quantidade de luzes existentes, e também para possibilitar que não passe de 8.

Assim que entrar numa *tag* `lights`, ativa-se as luzes e itera-se sobre a(s) `light` presentes:

1. Ative-se a luz a que está irá corresponder (`GL_LIGHT0 + light_number`)
2. Cria-se um novo objecto `Light`;
3. Adiciona o tipo de luz;
4. Adiciona a posição;
5. Se tiver cores, adiciona-se os diferentes tipos de cores;
6. Se tiver as características próprias a uma luz;
7. Adiciona-se essa luz como subnodo da árvore;

No que diz respeito as cores que se atribui a uma figura, este deixa-se explicito no ficheiro XML. Por outro lado, se nada especificado, deixa-se as cores `ambient` e `diffuse` a preto, e as restantes `specular` e `emissive` a branco.

Finalmente, no que diz respeito à leituras das texturas, esta divide-se em duas fases – primordialmente verifica-se se a textura em questão já está carregada, recorrendo ao

`map` que contem os nomes das texturas e o respetivo id. Se tal não se verificar recorre-se à função `getTexture()` que recebe como paramento a diretoria do ficheiro. Esta função que abre então o ficheiro, recolhe as informações necessárias e veicula o slot GL_TEXTURE_2D, define-se os parâmetros para a textura e envia-se para OpenGL. No fim é retornado o id e adiciona-se a textura ao `map`.

De seguida é preciso ler o ficheiro 3D de modo a obter as coordenadas de textura, podendo também obter as normais. Analogamente a obter as texturas, também procuramos num `map` a existência de um VBO para essa mesma textura/normal – se existir, colocamos o seu valor no nodo – se não, recorre-se à função `loadfigure()`, enviando como parâmetros o endereço dos arrays para preencher. Assim que preenchidos, são realizados o `GenBuffers` e o `BindBuffers` e obtem-se os índices VBOs, adicionando aos `map` esses mesmos valores.

3.3. Representação

Tendo finalmente uma arvore com diversas figuras e luzes

Mais uma vez a função drawScene() é responsável pela representação do ficheiro XML. Ao que às luzes diz respeito verifica-se se nodo actual da arvore contem luzes – se tiver itera-se sobre elas e recorrendo `glLightfv` aplica-se todas as componentes a ela associadas, como posição, atenuações e cores.

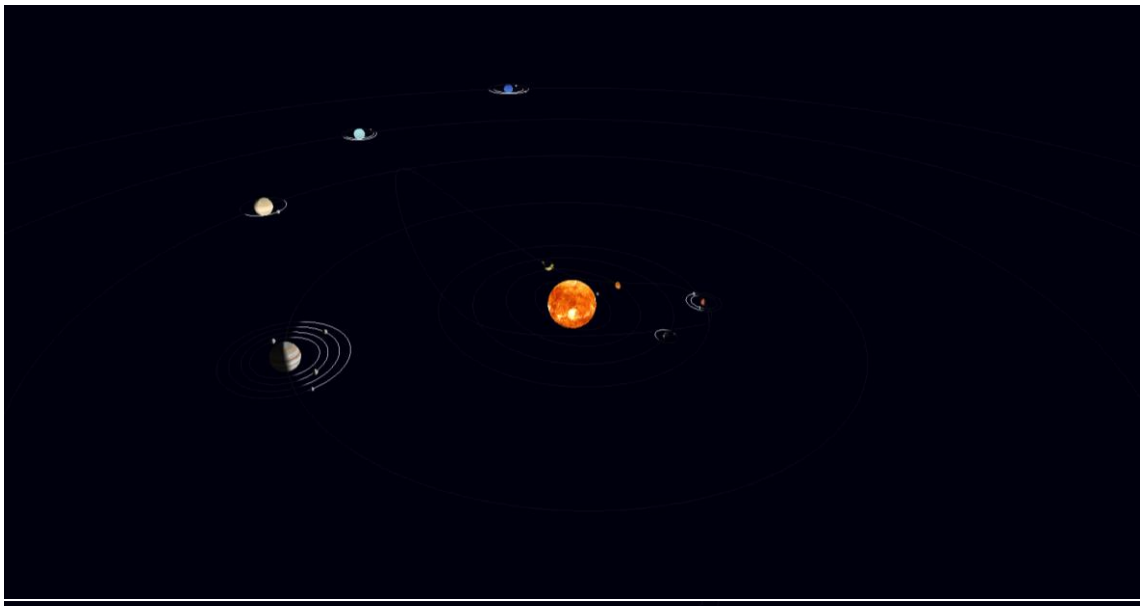
Voltando à personalizações de figuras, verifica-se se o actual da arvore nodo contem cor, e caso tenha, recorre-se a `glMaterialfv` de modo a representar. Finalmente, representa-se todas as figuras, texturas e normais usando a sequência de comandos:

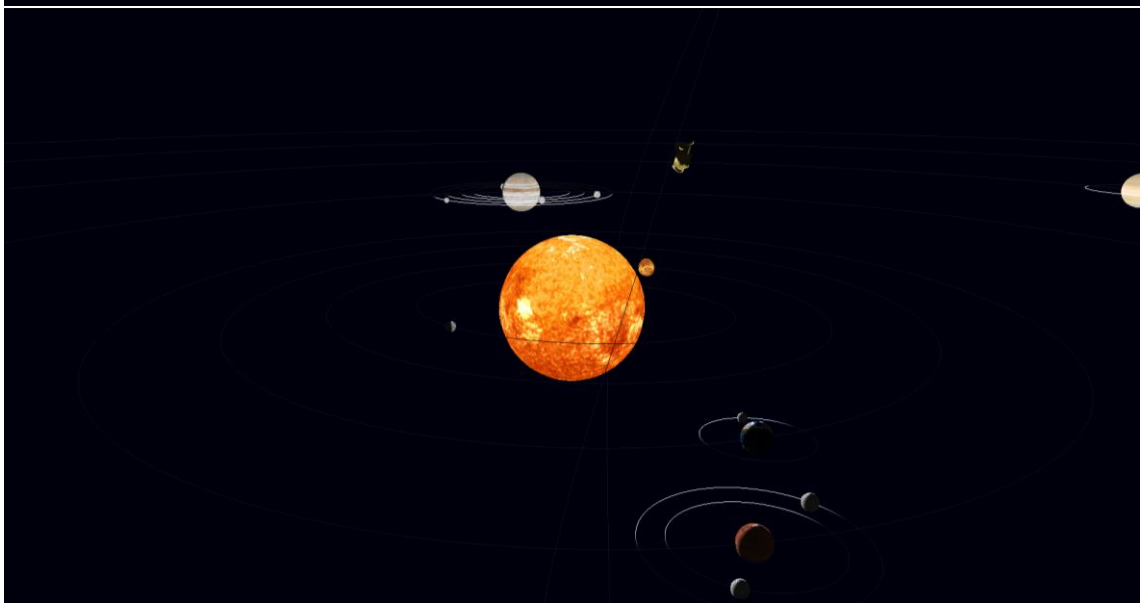
```
if (t.figure.vboIndex != -1) {

                    glBindBuffer(GL_ARRAY_BUFFER, t.figure.vboIndex);
                    glVertexPointer(3, GL_FLOAT, 0, 0);

                    glBindBuffer(GL_ARRAY_BUFFER, t.figure.normalIndex);
                    glNormalPointer(GL_FLOAT, 0, 0);

                    glBindBuffer(GL_ARRAY_BUFFER, t.figure.textureIndexVBO);
                    glTexCoordPointer(2, GL_FLOAT, 0, 0);
                    glBindTexture(GL_TEXTURE_2D, t.figure.textureIndex);

                    glDrawArrays(GL_TRIANGLES, 0, t.figure.triangles *  3);
                    glBindTexture(GL_TEXTURE_2D, 0);
            }
```

4. Scene

O Sistema Solar é a *scene* que é pedida no enunciado. Esta consiste:

- numa luz do tipo *POINT* localizada no centro do Sistema Solar (ou seja, no local do Sol);
- num conjunto de planetas representados por esferas, a orbitar em torno em do Sol, com as respetivas texturas;
- nos vários conjuntos de luz que orbitam em torno dos respetivos planetas;
- num cometa (figura do teapot) a viajar pelo Sistema Solar.





5. Conclusão

Este trabalho prático foi realizado com a intenção de solidificar os conhecimentos da Unidade Curricular de Computação Gráfica. Concluída esta etapa, conseguimos aprimorar as técnicas de programação em C++ bem como rever conceitos matemáticos e geométricos.

Através do cálculo das normais de cada sólido e colocação das texturas foram revistos conceitos geométricos. Com o uso de luzes foi possível conseguir um melhor conhecimento sobre os distintos tipos considerados no nosso projeto, bem como a influência delas nos modelos utilizados.