Universidade do Minho
Escola de Engenharia
Mestrado Integrado em Engenharia Informática

Unidade Curricular de Computação Gráfica

Ano Letivo de 2018/2019

Sistema Solar

Henrique Faria (A82200), João Marques (A81826),

José Pereira (A82880), Ricardo Petronilho (A81744)

Maio, 2019

CG

Data de Recepção	
Responsável	
Avaliação	
Observações	

Sistema Solar

Henrique Faria (A82200), João Marques (A81826),

José Pereira (A82880), Ricardo Petronilho (A81744)

Maio, 2019

Índice

Introdução

1.1. Contextualização

O presente relatório refere-se à quarta fase do projeto da Unidade Curricular de Computação Gráfica do 3º ano 2º semestre do curso Mestrado Integrado em Engenharia Informática da Universidade do Minho, onde o objetivo é a construção de um sistema solar.

Na quarta fase, começou-se por modificar a aplicação do generator, fazendo a geração das normais dos vértices para cada sólido e também das coordenadas de textura dos mesmos, tendo sido retirado, devido a complicações de código, o filtro para criação dos índices dos vértices dos sólidos.

De seguida, adaptou-se o parser do *XML*, pois foram adicionadas as luzes ao sistema solar e as texturas aos modelos a desenhar, podendo as texturas ser uma imagem predefinida ou um conjunto de três cores. Tendo também sido alterado o modo de escrita e leitura em ficheiro pois para além dos pontos existem agora coordenadas de textura e normais.

Na aplicação *engine* colocou-se os modelos a serem desenhados com *VBOs*, onde os pontos, as coordenadas de textura e as normais são preenchidos a partir de ficheiros.

Por fim, ainda no *engine*, adicionou-se as funções necessárias para a colocação das texturas nos modelos a desenhar.

Generator

2.1 Bezier

Para gerar as normais e as coordenadas de textura de Bezier é necessário, primeiramente, ler o ficheiro **.patch.** As normais e as coordenadas de textura são geradas ao mesmo tempo que os vértices da superfície de Bezier.

2.1.1. Leitura do ficheiro .patch

A leitura do ficheiro foi bastante facilitada devido ao formato do mesmo, criou-se a função **readPatchFile()**, desta forma o procedimento foi o seguinte:

1. ler o número de patches - N
2. ler N patches
3. ler o número de pontos – M
4. ler M pontos

À medida que se faz a leitura tanto dos patches como dos pontos, os mesmos são migrados para as respetivas estruturas em memória.

2.1.2. Gerar Normais de Bezier

De seguida utilizou-se a função **getPointsOfBezier()** da fase anterior que, após a leitura do ficheiro .patch, segue o seguinte procedimento:

1. Percorrer cada patch e a respetiva tesselation associada ao mesmo.
2. Para o procedimento anterior calcular os 6 pontos de Bezier.
3. Armazenar os pontos calculados num vetor para os pontos.
4. Para o procedimento anterior calcular as 6 normais de Bexier.
5. Armazenar os pontos calculados num vetor para as normais.

Após a geração dos pontos e das normais escreve-se os mesmos num ficheiro .3d evocando a função **list2file().**

2.1.3. Gerar uma normal de Bezier

Para calcular uma normal de bezier é utilizado o procedimento seguinte:

1. Gerar o polinómio de Bernstein para o ponto U
2. Gerar o polinómio de Bernstein para o ponto V
3. Gerar a derivada do polinómio de Bernstein para o ponto U

4. Gerar a derivada do polinómio de Bernstein para o ponto V
5. Implementar a seguinte fórmula: dBu(u,v) = dBi(u) * Pij * Bj(v)
6. Implementar a seguinte fórmula: dBv(u,v) = Bi(u) * Pij * dBj(v)
7. Aplicar o cross entre dBu(u,v) e dBv(u,v)
8. Aplicar o normalize ao vetor gerado no tópico anterior

Sendo Bi(u) o polinómio para o ponto U, Bj(v) o polinómio para o ponto V, o dBi(u) a derivada do polinómio para o ponto U, o dBj(v) a derivada do polinómio para o ponto V e Pij o ponto de controlo calculado e previamente lido do ficheiro .patch.

2.1.4. Gerar Coordenadas de textura de Bezier

Para gerar as coordenadas de textura de bezier utilizou-se uma fórmula bastante simples, como é impossível saber qual a figura que vai ser produzida através do ficheiro .patch as coordenadas de textura foram calculadas da seguinte formula:

- (u,v)
- (u + (1.0f / tLevel), v)
- (u + (1.0f / tLevel), v + (1.0f / tLevel))
- (u + (1.0f / tLevel), v + (1.0f / tLevel)
- (u + (1.0f / tLevel), v)
- (u,v)

As variáveis u e v tem o valor de k a dividir por tLevel e w a dividir por tLevel, respetivamente, e k e w são duas vaiáveis que variam entre 0 e tLevel, em que tLevel é o nível de tesselation pretendido.

2.2 Plano

2.2.1 Gerar Normais

Sendo esta uma superfície plana e contida no plano XoZ as normais são iguais para todos os vértices e com o valor (0,1,0).

2.2.2 Gerar Coordenadas de Textura

Com a mesma simplicidade que se calculou as normais para os vértices do plano calcula-se as coordenadas de textura. O plano é constituído por dois triângulos sendo que um é constituído pelos cantos superior direito, superior esquerdo e inferior esquerdo pelo que em coordenadas de textura se resume a (1,1), (0,1) e (0,0), respetivamente. O outro triângulo é formado pelos cantos inferior esquerdo, inferior direito e superior direito que em coordenadas corresponde a (0,0), (1,0) e (1,1).

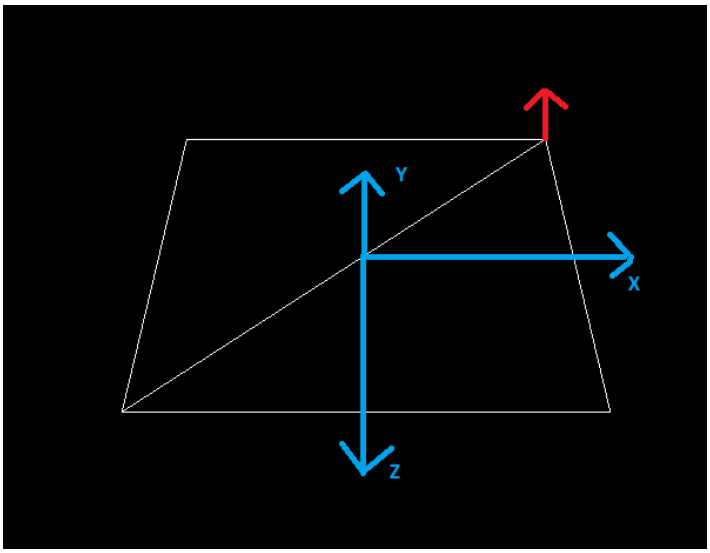

Figura 1: Plano com uma normal (seta a vermelho).

2.3 Box

2.3.1 Gerar Normais

A geração de normais na box é muito idêntica à do plano, pois do mesmo modo que o plano está perpendicular ao eixo Y as faces da box estão perpendiculares aos eixos ordenados. Os vértices da face frontal (face que interseta o eixo Z na parte positiva) terão todos a mesma normal de valor (0,0,1), pois essa face sendo perpendicular ao eixo Z e às suas normais perpendiculares este tem a mesma direção, o sentido é definido da origem para o exterior porque a box encontra-se centrada na origem. A face traseira, paralela à frontal, mas que interseta o eixo Z na parte negativa, tem como normais o seguinte valor (0,0,-1). As faces direita e esquerda que interseção o eixo do X na parte positiva e negativa, respetivamente, tem como valor das normais (1,0,0) e (-1,0,0), respetivamente. O mesmo se aplica às faces inferior e superior da box que intersetam o eixo Y, sendo que a face superior interseta na parte positiva daí as suas normais serem (0,1,0) e a face inferior interseta na parte negativa pelo que as suas normais tomam o valor (0,-1,0).

2.3.2 Gerar Coordenadas de Textura

Para o desenho das coordenadas de textura da box foi primeiro necessário definir quais as posições da textura iriam corresponder às faces da mesma, pelo que foi definido que a imagem seria dividida a meio em termos de altura e dividida em três partes em termos de largura. Tendo sido feita esta divisão ficam assim atribuídas as partes da imagem a cada face:

- Face frontal – parte superior esquerda da textura
- Face direita – parte superior central da textura
- Face superior – parte superior direita da textura
- Face traseira – parte inferior esquerda da textura
- Face esquerda – parte inferior central da textura
- Face inferior – parte inferior direita da textura

Para a geração das coordenadas de textura da face superior foram utilizados os algoritmos que serão demonstrados a seguir, para tal foram precisos definir seis coordenadas de textura (para poder formar um plano), mas a face de uma box pode ter mais do que 6 coordenadas de textura, isso vai depender do número de divisões que seja definido, para tal utilizaram-se então os seguintes algoritmos:

- (0.66f + 0.33f * (j+1) / divisions, 0.5f + 0.5f * i / divisions)
- (0.66f + 0.33f * (j+1) / divisions, 0.5f + 0.5f * (i+1) / divisions)
- (0.66f + 0.33f * j / divisions, 0.5f + 0.5f * i / divisions)
- (0.66f + 0.33f * j / divisions, 0.5f + 0.5f * i / divisions)
- (0.66f + 0.33f * (j+1) / divisions, 0.5f + 0.5f * (i+1) / divisions)
- (0.66f + 0.33f * j / divisions, 0.5f + 0.5f * (i+1) / divisions)

Nestes algoritmos apresentados a variável divisions é o número de divisões aplicadas à box, que por omissão é 1 gerando assim as 6 coordenadas de textura por face referidas), as variáveis i e j são as variáveis que vão percorrendo o número de divisões em altura e comprimento, respetivamente. Os valores 0.66f e 0.5f servem para posicionar as coordenadas de textura no espaço de textura correspondente, cada face tem o seu conjunto de valores:

- Face frontal - (0, 0.5f)
- Face direita - (0.33f, 0.5f)
- Face superior - (0.66f, 0.5f)
- Face traseira - (0, 0)
- Face esquerda - (0.33f, 0)
- Face inferior - (0.66f, 0)

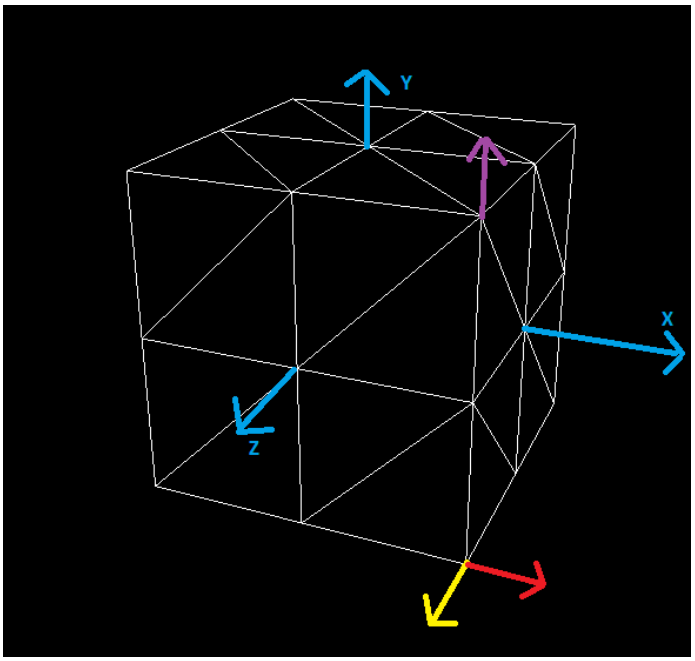

Figura 2: Box com uma normal em cada face visível (setas a vermelho, amarelo e roxo).

2.4 Cone

2.4.1 Gerar Normais

A geração das normais do cone é realizada ao mesmo tempo da geração das coordenadas dos vértices, pois são usados 3 vértices para calcular dois vetores para se realizar a função **cross()** entre eles obtendo um vetor normal ao triângulo criado, de seguida tem de ser aplicada a função **normalize()** para normalizar o vetor (o comprimento desse vetor passa a ser 1), este procedimento foi apenas tomado para a face lateral do cone pois a base está contida no plano XoZ pelo que as normais desses vértices serão (0,-1,0).

Os vértices utilizados para criar as normais são os vértices gerados da seguinte forma:

- p1 = (tmp1*sin(angle*(j + 1)), fraction_height, tmp1*cos(angle*(j + 1)))
- p2 = (tmp2*sin(angle*(j+1)) , fraction_height+((height/stacks)) , tmp2*cos(angle*(j+1)))
- p3 = (tmp1*sin(angle*j), fraction_height, tmp1*cos(angle*j))

As variáveis tmp1 e tmp2 são os raios de duas stacks consecutivas, o fraction_height é a altura da stack e o angle é o ângulo entre duas slices consecutivas.

Depois dos três pontos calculados (p1, p2 e p3) são calculados os vetores v1 e v2 da seguinte forma, v1 = (p2.x-p1.x,p2.y-p1.y,p2.z-p1.z) e v2 = (p3.x-p1.x,p3.y-p1.y,p3.z-p1.z), de seguida faz se a função **cross()** que calcula o novo vetor fazendo:

- v3.x = v1.y*v2.z - v1.z*v2.y)
- v3.y = v1.z*v2.x - v1.x*v2.z)
- v3.z = v1.x*v2.y - v1.y*v2.x)

Despois desta função é apenas necessário normalizar o vetor e adicioná-lo ao conjunto das normais do cone.

2.4.2 Gerar Coordenadas de Textura

Para gerar as coordenadas de textura para o cone foi assumido que a imagem da textura irá conter um retângulo na parte superior e uma circunferência na parte inferior, sendo que a circunferência irá ter 0.30 de diâmetro numa escala de 0 a 1. Depois do pressuposto assumido a geração das coordenadas de textura para a base é executado da seguinte maneira:

- (0.15f + 0.15f * sin(angle*j),0.15f + 0.15f * cos(angle*j))
- (0.15f,0.15f)
- (0.15f + 0.15f * sin(angle*(j+1)),0.15f + 0.15f * cos(angle*(j+1)))

Como foi assumido que a circunferência teria 0.30 de diâmetro na imagem de textura, o seu centro será (0.15f,0.15f). O angle é o ângulo entre duas slices consecutivas.

Para a parte lateral do cone as coordenadas foram calculadas da seguinte maneira:

- (0.5f + 0.5f * sin(angle*(j+1)),0.3f + 0.7f * (j + 1) / stacks)
- (0.5f + 0.5f * sin(angle*(j+1)),0.3f + 0.7f * (j + 1) / stacks)
- (0.5f + 0.5f * sin(angle*j),0.3f + 0.7f * j / stacks)
- (0.5f + 0.5f * sin(angle*(j+1)),0.3f + 0.7f * (j+1) / stacks)
- (0.5f + 0.5f * sin(angle*j),0.3f + 0.7f * j / stacks)
- (0.5f + 0.5f * sin(angle*j),0.3f + 0.7f * j / stacks)

A constante 0.3f serve para elevar as coordenadas de textura até ao retângulo referido anteriormente e a constante 0.7f a variação possível, juntando as duas as coordenadas de textura a face lateral variam entre 0.3 e 1 em altura. Em largura irão variar entre 0 e 1 pelo que se definiu como o centro da largura o 0.5 pois como o **sin()** varia entre –1 e 1 este valor é multiplicado por 0.5 para variar entre –0.5 e 0.5 e soma-se o valor por omissão de 0.5 para variar entre 0 e 1 como pretendido.

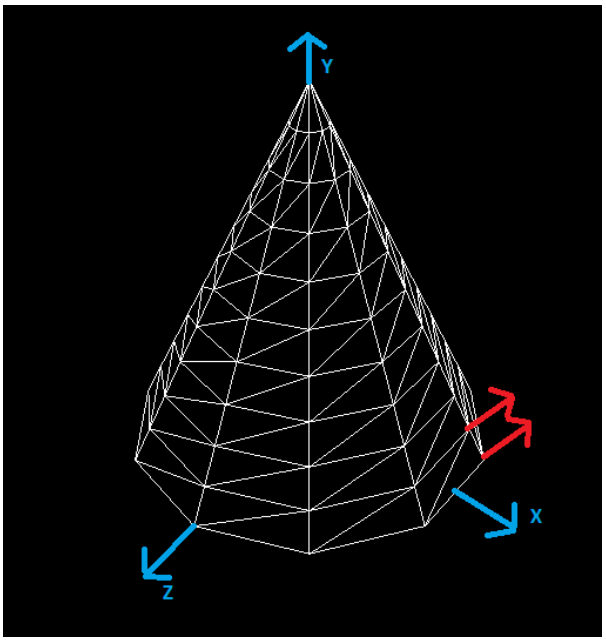

Figura 3: Cone com duas normais paralelas por pertencerem à mesma face lateral (setas vermelhas).

2.5 Esfera

2.5.1 Gerar Normais

A geração das normais da esfera é realizada ao mesmo tempo da geração das coordenadas dos vértices pois são usados 3 vértices para calcular dois vetores para se realizar a função **cross()** entre eles obtendo um vetor normal ao triângulo criado, de seguida tem de ser aplicada a função **normalize()** para normalizar o vetor (o comprimento desse vetor passa a ser 1).

Os vértices utilizados para criar as normais são os vértices gerados da seguinte forma:

- p0 = POINT(r2*cos(m2*alfa),y2,r2*sin(m2*alfa));
- p1 = POINT(r1*cos(m2*alfa),y1,r1*sin(m2*alfa));
- p2 = POINT(r1*cos((m2+1)*alfa),y1,r1*sin((m2+1)*alfa));

As variáveis y2 e y1 são respetivamente a altura da stack superior e inferior na malha lateral da esfera a ser desenhada.

Alfa é o angulo entre slices consecutivas e r1 e r2 são respetivamente os raios das stacks superior e inferior numa mesma malha lateral da esfera.

Depois dos três pontos calculados (p0, p1 e p2, respetivamente) são calculados os vetores v1 e v2, fazendo uso da estrutura Point da seguinte forma:

- p3 = POINT(getX(p1)-getX(p0),getY(p1)-getY(p0),getZ(p1)-getZ(p0));
- p4 = POINT(getX(p2)-getX(p0),getY(p2)-getY(p0),getZ(p2)-getZ(p0));

Em seguida calculamos a normal, e guardamos o resultado em p5 fazendo:

- cross(p3,p4,p5);

Despois desta função é apenas necessário normalizar o vetor e adicioná-lo ao conjunto das normais da esfera.

2.5.2 Gerar Coordenadas de Textura

Para gerar as coordenadas de textura para a esfera foi assumido que a imagem da textura está uniformemente distribuida por toda a esfera.

Depois do pressuposto assumido a geração das coordenadas de textura para a malha do topo da esfera, malhas laterais e malha da base da esfera são as seguintes:

Malha superior da esfera:

- p0 = POINT((float)m2*slice_texture+(slice_texture/2),1.0,0.0);
- p1=POINT((float)m2*slice_texture,(float)1-(stack_texture*m1+stack_texture),0.0);
- p2=POINT((float)(m2*slice_texture+slice_texture),(float)1-(stack_texture*m1+stack_texture),0.0);

Malha lateral da esfera:

- triângulo inferior

- p0 = POINT(m2*slice_texture,1-(m1*stack_texture),0.0);
- p1 = POINT(m2*slice_texture,1-(m1*stack_texture+stack_texture),0.0);
- p2 = POINT(m2*slice_texture+slice_texture,1-(m1*stack_texture+stack_texture),0.0);

- triângulo superior

- p0 = POINT(m2*slice_texture+slice_texture,1-(m1*stack_texture),0.0);
- p1 = POINT(m2*slice_texture,1-(m1*stack_texture),0.0);
- p2=POINT(m2*slice_texture+slice_texture,1-(m1*stack_texture+stack_texture),0.0);

Malha inferior da esfera:

- p0 = POINT(m2*slice_texture+(slice_texture/2),0.0,0.0);
- p1 = POINT(m2*slice_texture+slice_texture,1-(stack_texture*m1),0.0)
- p2 = POINT(m2*slice_texture,1-(stack_texture*m1),0.0);

Slice_texture e stack_texture são respetivamente a distancia entre dois pontos da imagem consecutivos correspondentes a 2 vertices da esfera também consecutivos,

as variáveis m1 e m2 são utilizadas num ciclo for para percorrer todos os pontos da esfera sendo m1 usada para percurrer stack a stack e m2 para percorrer as slices. Nota: Como começamos a mapear a esfera na imagem do topo da mesma e m1 é incremental temos de subtrair a 1, m1*stack_texture e m1*stack_texture + stack_texture para mapear a imagem corretamente e não de pernas para o ar.

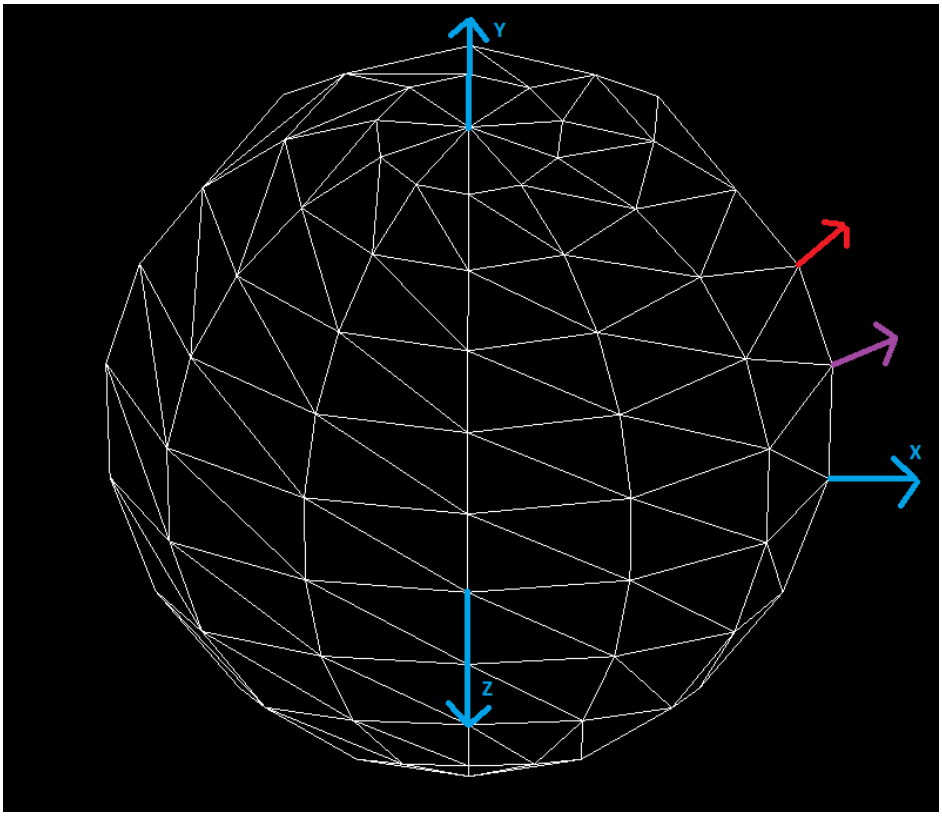

Figura 4: Esfera com duas normais de mesma slice mas em stacks consecutivas (setas a vermelho e roxo).

Engine

3.1 Implementação de VBOs

Na fase anterior foi utilizada a função *filter*, que filtrava os pontos repetidos gerados pelo generator e calcula os índices respetivos. Devido a complicações de implementação do código esta mesma função foi retirada sendo assim escritos para ficheiros todos os pontos, normais e coordenadas de textura com valores repetidos.

Com a alteração explicitada anteriormente a função *file2list* teve de ser modificada para ler todos os tipos de coordenadas presentes no ficheiro sem o uso de índices. Depois de lidos todos os pontos guardam-se em 3 buffers na gráfica, um para os pontos, outro para as normais e outro para as coordenadas de textura.

Para a criação dos buffers referidos usaram-se as funções:

- glGenBuffers(nFiguras, buffers);
- glGenBuffers(nFiguras, normals);
- GlGenBuffers(nFiguras, texturas);

No qual a variável nFiguras corresponde ao número de figuras diferentes encontradas no file.xml.

Nota: Os buffers gerados variam com o número de figuras diferentes no projeto pois, a bem da eficiência, existem 3 buffers para cada figura diferente evitando que existam vários buffers de pontos, normais ou coordenadas de textura para a mesma figura.

Após a leitura, usaram-se as seguintes funções:

- glBindBuffer(GL_ARRAY_BUFFER, buffers[x]);
- glBufferData(GL_ARRAY_BUFFER,sizeof(float)*(it->second.pointsTAM), it->second.points, GL_STATIC_DRAW);
- glBindBuffer(GL_ARRAY_BUFFER,normals[x]);
- glBufferData(GL_ARRAY_BUFFER,sizeof(float)*(it->second.normalsTAM), (it->second.normals), GL_STATIC_DRAW);
- glBindBuffer(GL_ARRAY_BUFFER, texturas[x]);
- glBufferData(GL_ARRAY_BUFFER,sizeof(float)*(it->second.texCoordsTAM), (it-> second.texCoords), GL_STATIC_DRAW);

Para saber que buffer preencher usou-se a função GlBindBuffer.

No primeiro campo desta temos GL_ARRAY_BUFFER o qual possibilita a escrita para o buffer de pontos.

Estas funções foram usadas num ciclo for no qual a variável x variava de 0 a nFiguras para a cada posição dos buffers para poder preencher um buffer na gráfica com o conteúdo dos pontos, normais e coordenadas de textura com o conteúdo dos mesmos necessários para desenhar uma figura, respetivamente.

Posteriormente fez-se uso da função glBufferData, para preencher cada um dos buffers selecionados com recurso á função glBindBuffer.

Esta função recebe o tipo de array a ser preenchido, o tamanho do array a ser copiado, o array a ser copiado e a forma como este vai ser usado na aplicação.

A variável it é um iterador sobre um map de <string,Figura> o qual é percorrido para garantir a unicidade de cada buffer gerado para cada figura diferente.

Após estes procedimentos, percorremos a class Group que contem o sol, os planetas, respetivas luas e o cometa.

Para desenhar estes, foi primeiro necessário verificar se se tratava de uma figura com uma textura predefinida através de uma imagem ou se utilizava em vez de uma textura um conjunto de cores especular, ambiente, emissiva ou difusa, para a parte comum (pontos e normais) foram utilizadas as seguintes funções:

- glBindBuffer(GL_ARRAY_BUFFER, buffers[count]);
- glVertexPointer(3, GL_FLOAT, 0, 0);
- glBindBuffer(GL_ARRAY_BUFFER, normals[count]);
- glNormalPointer(GL_FLOAT,0,0);
- glDrawArrays(GL_TRIANGLES, 0,tam);

No caso de se tratar de uma figura com uma textura predefinida por imagem são usadas as funções:

- glEnable(GL_TEXTURE_2D);
- glBindTexture(GL_TEXTURE_2D, figTex);
- glBindBuffer(GL_ARRAY_BUFFER, texturas[count]);
- glTexCoordPointer(2, GL_FLOAT, 0, 0);
- glBindTexture(GL_TEXTURE_2D, 0);
- glDisable(GL_TEXTURE_2D);

A primeira função habilita a utilização das coordenadas de textura para colocar uma textura sobre uma figura. A segunda diz qual a imagem a utilizar (figTex), estas foram previamente carregadas para a gráfica durante o parse do ficheiro XML. A terceira e quarta indicam o buffer e quantas valores a ler por vértice, respetivamente. De seguida é realizada a função glDrawArrays() referida anteriormente. As duas últimas colocam a textura por defeito e depois desabilita as texturas, este procedimento teve de ser tomado porque existiam figuras a aparecer com a textura errada.

No caso de a figura não ter uma textura mas sim um conjunto de cores verificasse qual o tipo de cor da figura (emissiva, difusa, especular ou ambiente), utilizando respetivamente as seguintes funções:

- glMaterialfv(GL_FRONT, GL_EMISSION, arr)
- glMaterialfv(GL_FRONT, GL_DIFFUSE, arr)
- glMaterialfv(GL_FRONT, GL_SPECULAR, arr)
- glMaterialfv(GL_FRONT, GL_AMBIENT, arr)

Nas funções anteriores *arr* é o vetor com os valores das cores. Depois de executada uma das funções anteriores é executado o procedimento comum já referido.

Estas funções usadas por esta ordem indicam:

1. Qual o buffer de pontos a ser utilizado.
2. Quantos elementos do buffer são usados para definir 1 ponto e qual o tipo do elemento.
3. Qual o buffer de normais a ser utilizado.
4. Quantos elementos do buffer são usados para definir 1 ponto e qual o tipo do elemento.
5. Quais as figuras base a ser desenhadas, qual a posição em que se começa a percorrer os buffers e o número de vértices a desenhar.

3.2 Carregamento das Texturas

Para o carregamento das texturas foi utilizada a função fornecida pelo docente da Unidade Curricular, **loadTexture()**.

Esta função recebe como parâmetro o nome do ficheiro com a imagem para ser convertida para textura. Primeiramente tem de se iniciar as bibliotecas do devIL utilizando para isso a função **ilInit()** da biblioteca referida. De seguida invocan-se as funções **ilEnable(IL_ORIGIN_SET)** e **ilOriginFunc(IL_ORIGIN_LOWER_LEFT)** para definir como a posição (0,0) o canto inferior esquerdo da imagem. Note-se que as imagens e as respetivas coordenadas de textura variam entre 0 e 1.

Após a definição de qual a posição (0,0) a imagem é carregada gerando um buffer de imagem para a mesma e fazendo o seu carregamento, para a converter converter um bytes e de seguida gerar um buffer de textura onde será guardada. Por fim preenche o buffer de textura com o auxílio das seguintes funções:

- 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_MIPMAP_LINEAR)
- 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)

A primeira função diz qual o buffer a utilizar e a última recoloca a textura por defeito como textura ativa. Os parâmetros th e tw são, respetivamente, a altura e a largura da imagem.

3.3 Classes modificadas

A classe Figura representa o plane, box, cone, sphere e/ou uma superfície de Bezier, a mesma estava predefinida na fase precedente, no entanto devido à necessidade de inserção das normais e das coordenadas de textura, decidiu-se definir os arrays normals e texCoords, e os respetivos tamanhos, nesta classe para se guardarem os mesmos após a leitura do ficheiro correspondente, e ser mais fácil o acesso dos mesmos. Foi, no entanto, removido desta classe os array de índices dos pontos e o seu tamanho, devido à não utilização de índices nesta fase.

```
class Figura{
public:
    Figura(float*, int, float*, int, float*, int);
    float* points;
    int pointsTAM;
    float* normals;
    int normalsTAM;
    float* texCoords;
    int texCoordsTAM;
};
```

Figura 5: Class Figura.

A classe Group foi também alterada. Como nesta fase foram adicionadas as texturas era necessário guardar a informação de quais texturas ou conjunto de cores a utilizar para uma figura, sendo que foram para o efeito criados um vector<string> texturas para guardar os nomes das texturas e vector<TAD_POINT> materials para guardar os conjuntos de cores, TAD_POINT armazena 3 floats, como o último componente da cor tem sempre o valor 1.0f não é necessário guarda-lo pelo que se reutilizou a classe Point para armazenar então os 3 floats relevantes.

```
class Group{
public:
    Group();
    vector<Operation> operacoes;
    vector<string> ficheiros;
    vector<string> texturas;
    vector<TAD_POINT> materials;
    vector<Group> filhos;
};
```

Figura 6: Class Group.

3.4 Classes adicionadas

Para esta última fase foi necessário criar duas novas classes, a Light para armazenar informação sobre as luzes e a Textura que guarda os identificadores das texturas carregadas para a gráfica.

A classe Light armazena um *char* tipo para identificar se uma luz é um ponto, se é direcional ou se é um spotlight, todos estes tipos de luzes têm dois vetores em comum que são o pos e o diff (quatro floats para cada um), para o ponto de luz e o spotlight o pos é a sua posição sendo que para a luz direcional é a direção para onde este aponta a luz, o diff tem está relacionado com a difusão da luz, depois o ponto de luz e a luz direcional tem um outro vetor de quatro floats (amb) que especifica a intensidade das cores da luz e o spotlight tem um vetor de 3 floats que indicam para onde aponta a luz.

```
class Light
{
public:
    Light();
    Light(char,float,float,float,float, float, float, float, float, float, float, float, float, float, float, float);
    char tipo;
    float posX;
    float posY;
    float posZ;
    float posD;
    float diffX;
    float diffY;
    float diffZ;
    float diffD;
    float ambX;
    float ambY;
    float ambZ;
    float ambD;
    float spotX;
    float spotY;
    float spotZ;
};
```

Figura 7: Class Light.

A classe Textura já como referido acima apenas guarda o identifcador de uma textura guardada na gráfica pois assim é guardado um map de <string,Textura> onde são guardadas todos os identificadores das texturas para depois serem utilizadas no desenho das figuras.

```
class Textura{
public:
    Textura(GLuint);
    GLuint tex;
};
```

Figura 8: Class Textura.

Resultado Final

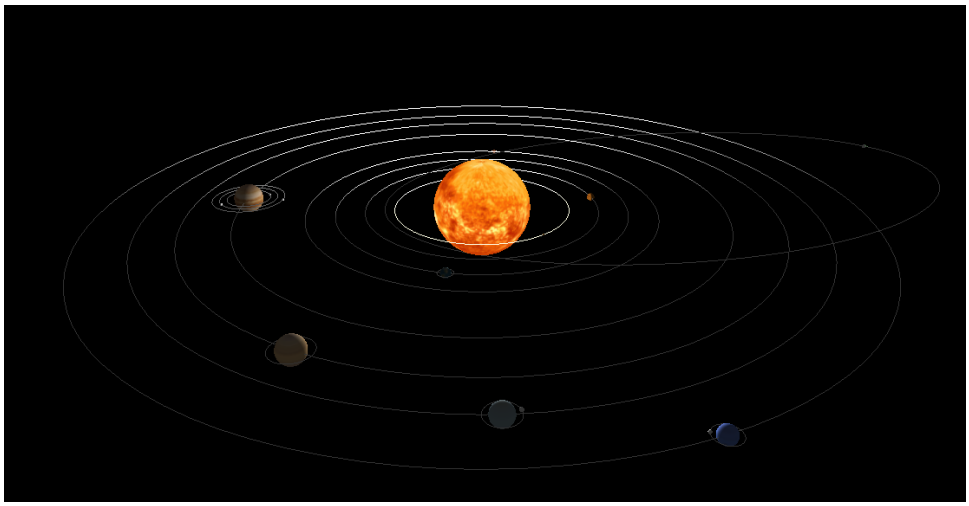

Figura 9: Demo Scene do Sistema Solar.

Conclusões

Em suma, a quarta fase, foi mais complexa comparado com as fases precedentes, pois necessitou-se de gerar as normais e coordenadas de textura sendo que para o cálculo das normais da superfície de Bezier foram utilizadas as fórmulas disponibilizadas pelo docente, que ao início suscitaram algumas dúvidas, mas que acabaram por sumir.

Por fim, os objetivos propostos pelo enunciado foram atingidos, dos quais o *generator* capaz de gerar ficheiros com normais e coordenadas de textura, sendo também possível a geração de normais a partir de ficheiros *(.patch)*. O *engine* desenha os pontos com *VBOs* aplicando ainda as texturas e as luzes obtidas através do parse do XML para a iluminação de todo o sistema solar.

Lista de Siglas e Acrónimos

XML Extensible Markup Language

VBOs Vertex Buffer Objects