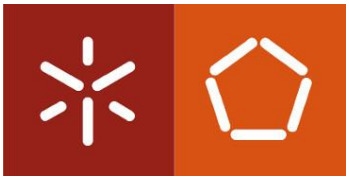

UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

TRABALHO PRÁTICO – Fase 4

Normals and Texture Coordinates

Computação Gráfica
Maio de 2021

Autores:

Ana Filipa Pereira A89589

Carolina Santejo A89500

Raquel Costa A89464

Sara Marques A89477

Índice

Introdução e Principais Objetivos

Nesta quarta e última fase do projeto prático da unidade de Computação Gráfica, foi pedido que tornássemos o sistema solar desenvolvido nas fases passadas mais realista através da adição de texturas e de iluminação. Para tal, foi necessário modificar o *generator* e o *motor.* No gerador, para cada primitiva, passaram a ser calculados os pontos de textura e os vetores normais. O *engine* foi alterado de forma a conseguir ler os ficheiros gerados. Além disto, foi preciso modificar também os *XML*.

Assim sendo, ao longo deste relatório será descrito todo o processo de desenvolvimento da fase além de serem explicadas todas as decisões tomadas.

Arquitetura do Programa

Nesta fase do projeto, foi adicionada a classe model, responsável por gerar VBOs, carregar imagens de textura e gerir os diferentes tipos de luz existentes (difusa, ambiente, especular, emissiva) e as suas cores.

Foi também adicionada em Demos a pasta texImages, que contém todas as imagens a serem utilizadas como texturas pelo nosso programa.

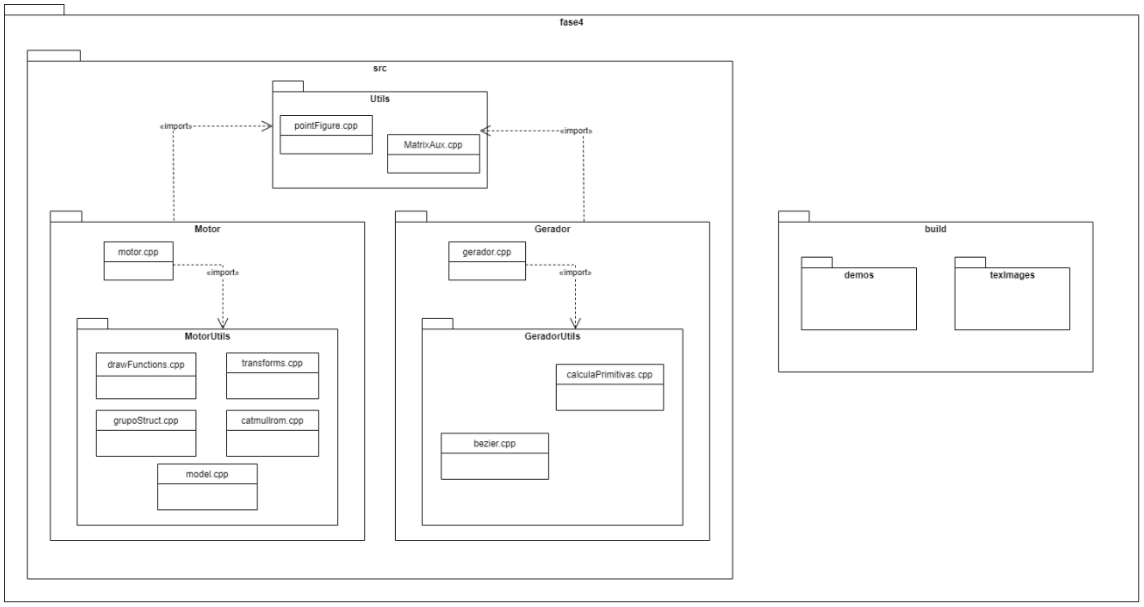

Figura 1 - Diagrama de Packages

Motor

VBO'S com índice

Tendo implementadas as estruturas VBO's na fase anterior, o grupo chegou à conclusão que, por um lado, a leitura dos ficheiros .3d estava a ser demorada devido à existência de um número elevado de pontos por cada figura e por outro, estava a ser gasto desnecessariamente demasiado espaço na memória da placa gráfica. Nesta fase, a necessidade da inclusão de vetores normais e pontos de textura justificou a conveniência da implementação de VBO's com índices.

Com esta alteração, todos os pontos passam a ser identificados pelos seus índices, o que leva à redução do número de vértices e consequentemente, do número de vetores normais e pontos de textura a guardar. Desta forma, apesar de atualmente se armazenar mais informação foi notável uma maior rapidez principalmente na leitura dos ficheiros .3d.

Para gerar os buffers respetivos para cada modelo lido foi utilizada a função generateVBOs da classe model que para cada conjunto de valores utiliza as funções glGenBuffers, glBindBuffer e glBufferData para alocar e copiar os dados para a memoria da placa gráfica.

Iluminação

Para definir a iluminação de uma cena gerada pelo nosso programa foi necessário caraterizar, não só a luz que ilumina todo o espaço, como também a iluminação que cada uma das figuras desenhadas pode possuir.

Para a luz principal, foi admitido que pode ser de três tipos: posicional, direcional ou ser um foco de luz. Para a primeira só foi necessário definir a sua posição, para o caso da segunda é estabelecido o vetor da direção e para a última são necessários ambos os parâmetros das duas primeiras. Para ativar a iluminação foram utilizadas as funções do openGL: glEnable(GL_LIGHTING) e glEnable(GL_LIGHT0). Consequentemente, para definir o tipo de luz utilizou-se a função glLightfv.

Quanto à iluminação dos materiais admitiu-se que podem possuir todas as quatro componentes da cor: difusa, especular, ambiente e emissiva. Para cada uma delas apenas foi necessário definir a sua cor com recurso ao método glMaterialfv, tendo em conta que para a especular se definiu por *default* o valor do *shininess* a 128.

Texturas

Dada a necessidade de aplicação de texturas às figuras desenhadas foi necessário carregar as todas imagens necessárias para a memória da placa gráfica. Com recurso à função loadTexture da classe model, foi carregada a imagem de cada um dos models dado o seu nome indicado previamente no ficheiro XML.

Para o seu desenho, inicialmente foram ativadas as funcionalidades das texturas do openGL chamando as funções glEnable(GL_TEXTURE_2D) e glEnableClientState(GL_TEXTURE_COORD_ARRAY) e logo a seguir foram ativados tanto o buffer com os dados da imagem como também os respetivos pontos de textura previamente lidos no ficheiro .3d que relacionam as posições do referencial e da imagem.

Estrutura de Dados – Atualização

A adição de iluminação e texturas e a alteração para estruturas *VBO's* com índices nesta fase do projeto implicou a alteração da estrutura de dados que armazena a todas as informações necessárias ao desenho de todas as figuras.

Ao contrario da fase anterior onde era possível agrupar todos os vértices das figuras no mesmo array e ir desenhando parcialmente de acordo com o grupo que estivéssemos a ler, nesta foi necessário, dentro de cada *group* separar os dados constituintes de cada modelo.

Deste modo foi criada uma nova classe *model* que guarda em primeiro lugar todos os valores dos vértices, índices, vetores normais e pontos de textura lidos a partir do ficheiro .3d, assim como o nome da imagem da textura (se não possuir textura é uma string vazia) referenciado no XML.

Uma vez que estamos perante a utilização de *VBO's* foi também necessário gerar e copiar todos os dados para a placa gráfica, logo a classe *model* constitui também 5 buffers do tipo *GLuint*, em que os primeiros quatro incluem os dados previamente guardados já referidos e o ultimo é a imagem da textura já carregada para a memória. A geração dos buffers e respetiva copia dos dados é feita apenas uma vez no momento em que for desenhada a cena pela primeira vez.

Assim sendo, a classe *group* é constituída pelos vetores das transformações (estáticas e dinâmicas) e dos grupos filhos tal como na fase anterior e é agora adicionado também um vetor de elementos da classe *model*.

Leitura e Parse do Ficheiro XML – Atualização

Uma vez que foram realizadas alterações ao ficheiro XML, foi necessário, consequentemente, alterar a função responsável por ler e fazer parse do XML.

Agora que foram adicionados os atributos de iluminação e textura ao elemento <model> do ficheiro, foi preciso incluir uma solução que conseguisse extrair esses atributos e guardá-los na estrutura de dados.

Além disso, com o surgimento do elemento <lights>, responsável por indicar qual a posição da fonte de luz e o seu tipo, podendo ser "POINT", "DIRECTIONAL" e "SPOT", e iremos ter de guardar numa variável global o tipo da mesma. Sendo assim, criamos uma nova função que irá ler essa secção do XML e guardar na variável global lightSource, onde as 3 primeiras posições são as posições da fonte de luz, no caso do tipo ser "POINT". Já no caso da "DIRECTIONAL" guarda as coordenadas do vetor da direção, e no caso de "SPOT" guarda tanto a direção como a posição.

Leitura do Ficheiro tipo 3D

Devido à mudança da estrutura dos ficheiros *3d*, foi também necessário efetuar mudanças ao nível da leitura dos mesmos.

Desta forma, foi feita uma leitura por partes, isto é, em primeiro lugar foi feito o parse da primeira linha do ficheiro e preenchido o vetor dos índices da classe *model*. Para o resto do ficheiro serão preenchidos os vetores correspondentes aos vértices, normais e coordenadas de textura sendo que cada linha irá corresponder a três ou dois elementos (para o caso das texturas).

Keybinds

Com recurso ao método *glutKeyboardFunc* foram atribuídas algumas funcionalidades a certas teclas do teclado quando o motor é executado. Cada uma delas pode ser especificada na tabela seguinte:

Tecla(s)	Ação
1	Ativar camara 1
2	Ativar camara 2
x	Mostrar/ocultar eixos do referencial
o	Mostrar/ocultar órbitas das curvas de *catmull-rom*
b	Mostrar/ocultar imagem de fundo
Camara 1	
Botão esquerdo do rato	Pressionar e arrastar os lados para rodar a camara
Botão direito do rato	Pressionar e arrastar para cima/baixo para aproximar/afastar a camara
Camara 2	
w	Mover para a frente da direção do olhar
s	Mover para trás da direção do olhar
a	Mover para a esquerda da direção do olhar
d	Mover para a direita da direção do olhar
Botão esquerdo do rato	Pressionar e arrastar para rodar a camara

Gerador

Nesta fase o gerador foi alterado para, além de fornecer as coordenadas de cada ponto das primitivas (*plane*, *box*, *sphere*, *cylinder*, *torus*) e os índices dos *VBOs*, gerar também as coordenadas normais e as coordenadas de textura de cada ponto. Todos estes dados serão então guardados num ficheiro que será lido pelo motor.

As normais dos pontos serão necessárias para que o *OpenGl* possa desenhar sombras correspondente às fontes de luz presentes em cada primitiva. A normal de cada ponto é calculada por face, logo, pontos que pertencem a múltiplas faces (como vértices ou pontos em arestas) terão as suas coordenadas registadas n vezes, sendo n o número de faces ao qual o ponto pertence.

As coordenadas de textura serão utilizadas para aplicar texturas às primitivas, sendo estas representadas por um ponto (x,y,z). As coordenadas (0,0,0) correspondem ao canto inferior esquerdo de uma imagem de textura. Ao percorrer a imagem horizontal ou verticalmente, x e y aumentam respetivamente até atingirem o valor máximo 1, sendo que z se mantém sempre 0, sendo uma coordenada inútil nestas circunstâncias. O *OpenGl* irá, depois, utilizar estas coordenadas para saber que ponto da imagem de textura corresponde a que ponto da primitiva em questão.

Vetores Normais

Nesta fase, cada uma das primitivas, passou a ter um vetor, no qual são colocadas todas as suas normais calculadas.

Plano

O cálculo das normais num plano, é bastante simples. Sendo que esta primitiva se encontra no plano XZ, os vetores normais serão (0,1,0) para todos os pontos da face voltada para cima e (0,-1,0) para todos os pontos da face voltada para baixo.

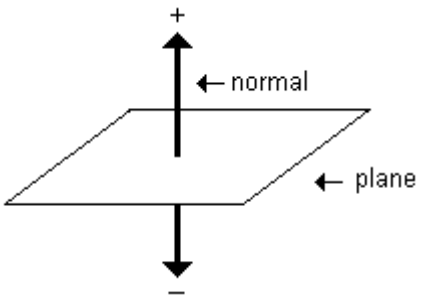

Figura 2 - Vetores normais ao plano

Caixa

Visto que o cubo possui seis faces planas e que para todos os pontos de uma face o vetor normal é o mesmo, concluímos o seguinte:

- o vetor normal dos pontos da face da frente é (0,0,1)
- o vetor normal dos pontos da face de trás é (0,0,-1)
- o vetor normal dos pontos da face da direita é (1,0,0)
- o vetor normal dos pontos da face da esquerda é (-1,0,0)
- o vetor normal dos pontos da face de cima é (0,1,0)
- o vetor normal dos pontos da face de baixo é (0,-1,0)

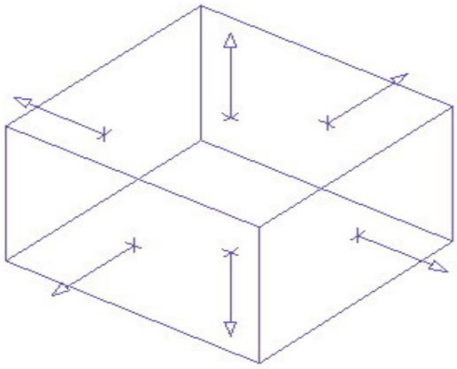

Figura 3 - Vetores normais às faces do cubo

Esfera

No caso da esfera, o vetor normal num dado ponto é igual ao vetor normalizado do vetor que vai do centro da esfera até ao ponto em questão. Assim sendo, a normal em qualquer ponto é a seguinte: $(\cos(\theta) * \sin(\varphi), \sin(\varphi), \cos(\varphi) * \cos(\theta))$;

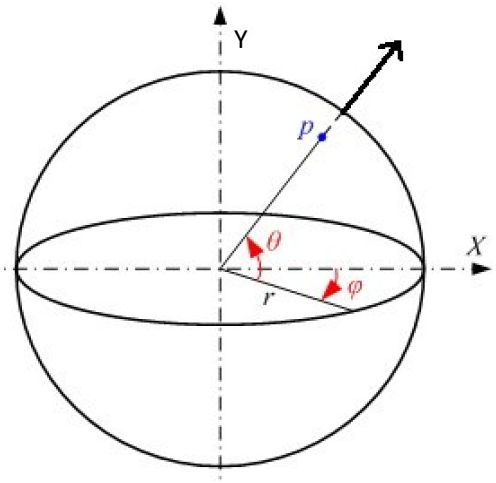

Figura 4-Vetores normais a um ponto na esfera

Cone

Para a base do cone, que se encontra no plano XZ, o vetor normal é o mesmo para todos os pontos e é igual a (0,-1,0). Por outro lado, o vetor normal para qualquer ponto da superfície lateral do cone pode ser obtido da seguinte forma:$(\sin(\alpha), \cos(\text{atan}(h/R)), \cos(\alpha))$;

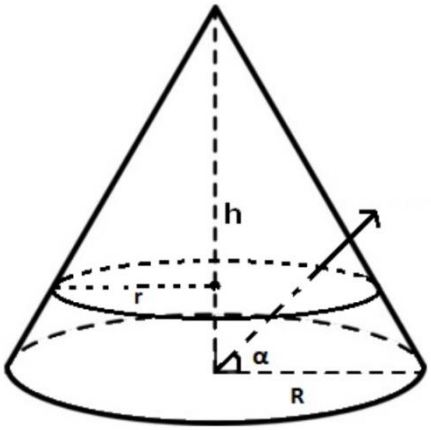

Figura 5 - Vetores normais à base e lateral do cone

Torus

No caso do *Torus*, o vetor normal para qualquer ponto, pode ser obtido da seguinte forma: $(cos(\varphi) * cos(\theta), cos(\varphi) * sin(\theta), sin(\varphi))$;

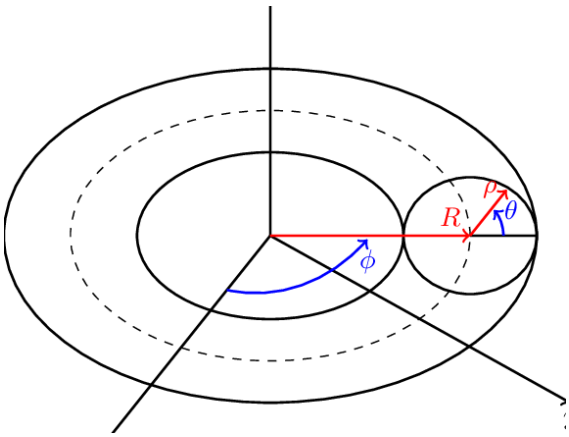

Figura 6 - Vetores normais ao Torus

Bezier Patches

No caso dos *patches* de *Bezier* as normais podem ser calculadas através o produto externo entre as derivadas de *u* (segunda fórmula) e *v* (terceira fórmula) e normalizando o resultado obtido. A matriz M corresponde à matriz de *Bezier*, e P aos pontos de controlos.

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

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

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

Figura 7 - Equações para o cálculo dos vetores normais de um Bezier Patch

Coordenadas de Textura

Nesta fase, cada primitiva passou a ter também um vetor no qual são armazenadas as coordenadas de textura que lhe poderão ser aplicadas.

Plano

No plano, o cálculo das coordenadas de textura é bastante simples. Para tal basta considerar o seguinte:

- O canto superior esquerdo corresponde ao ponto de textura com as coordenadas (0,1);
- O canto superior direito corresponde ao ponto de textura com as coordenadas (1,1);
- O canto inferior esquerdo corresponde ao ponto de textura com as coordenadas (0,0);
- O canto inferior direito corresponde ao ponto de textura com as coordenadas (1,0);

Box

Para o cálculo dos pontos de textura na caixa, inicialmente foi preciso considerar os valores de proporções apresentados na figura a baixo, sendo que cada número corresponde à parte da imagem que ficará numa determinada face do cubo.No entanto, é necessário ter em conta que a *box* é desenhada em camadas. Assim temos o seguinte:

- Uma divisão é a "interceção" entre uma camada vertical e horizontal;
- *textH* = (1/3)/(num_camadas), que simboliza o "comprimento" de uma divisão numa face;
- *textV* = (1/2)/(num_camadas), que simboliza a "altura" de uma divisão numa face;

- À medida que calculamos os pontos de uma face calculamos também as coordenadas de textura (três por triângulo);
- A cada iteração pelas camadas horizontais o x aumenta *textH* e a cada iteração pelas camadas verticais o y aumenta *textV*;
- Na face 4, o iniciamos o cálculo das coordenadas de textura no ponto (0,0);
- Na face 5, o iniciamos o cálculo das coordenadas de textura no ponto (1/3,0);
- Na face 6, o iniciamos o cálculo das coordenadas de textura no ponto (2/3,0);
- Na face 1, o iniciamos o cálculo das coordenadas de textura no ponto (0,1/2);
- Na face 2, o iniciamos o cálculo das coordenadas de textura no ponto (1/3,1/2);
- Na face 3, o iniciamos o cálculo das coordenadas de textura no ponto (2/3,1/2);
- Os valores das coordenadas de textura variam entre 0 e 1 (inclusive);

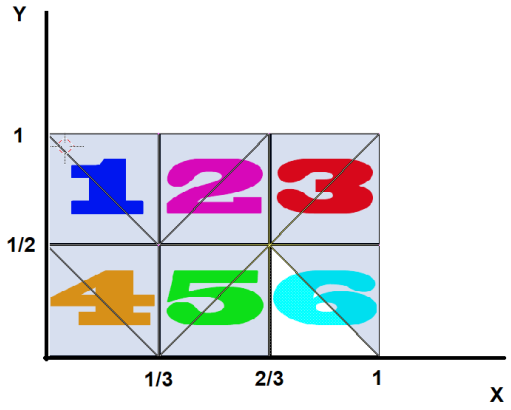

Figura 8 - Formato de uma textura para o cubo

Esfera

No caso da esfera, foi necessário dividir a imagem 2D, pelo número *stacks* e *slices* que a primitiva possui, considerando que o valor máximo da sua largura e comprimento é 1. Assim sendo temos o seguinte:

- Uma divisão da esfera é a "interceção" entre uma *stack* e uma *slice*;
- Para cada divisão da esfera, calculamos 6 coordenadas de textura (3 por triângulo);
- O comprimento de uma divisão da esfera é *textH* = 1/*slices*;
- A largura de uma divisão da esfera é *textV* = 1/*stacks*;
- A cada iteração pelas *stacks* a coordenada *y* varia *textV*;
- A cada iteração pelas *slices* a coordenada *x* varia *textH*;

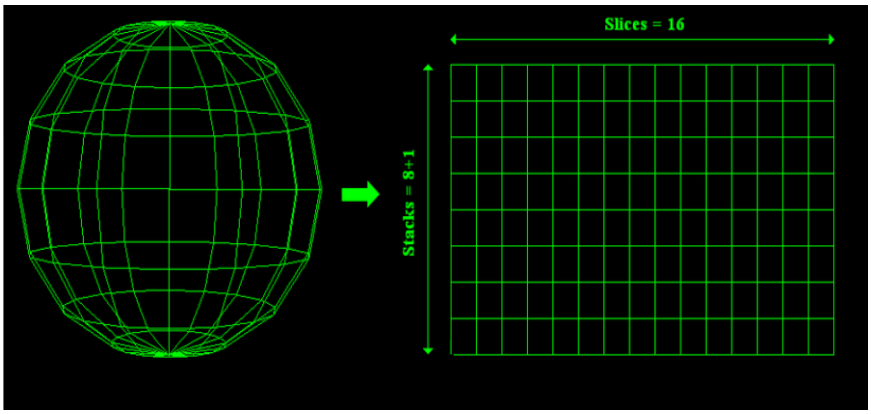

Figura 9 - Formato de uma textura para a esfera

Cone

No caso do cone, o cálculo das coordenadas de texturas é dividido em duas partes: uma dedicada à base da primitiva e outra à lateral;
Para base do cone, podemos considerar que a "decomposição" de uma circunferência em vários triângulos é aproximada da forma como é mostrado na figura 1. É de realçar que o comprimento da base de cada um dos triângulos é igual ao perímetro da circunferência a dividir pelo número de *slices*, ou seja, *dist* = 2**PI**Raio/*slices*.
Assim sendo, basta para cada iteração pelas *slices*, determinar as coordenadas de textura, que são: (i*dist,0,0), ((i+1)*dist,0,0) e (dist/2,1,0), onde *i* é o número da *slice* em questão.

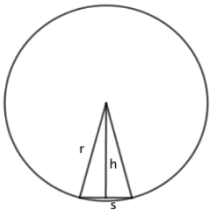

Figura 10 - Divisão aproximada de uma circunferência a vários triângulos

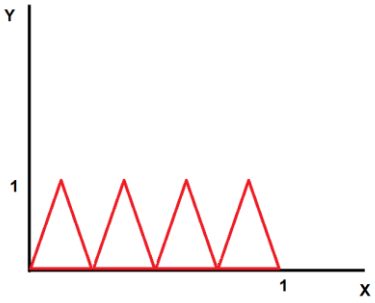

Figura 11 - Triângulos de uma circunferência

Por outro lado, o raciocínio utilizado para o cálculo das coordenadas de textura, é semelhante ao da esfera.

Torus

Para o caso desta primitiva, o raciocínio é semelhante ao da esfera.

Bezier Patches

Para o caso dos *patches* de *Bezier*, as coordenadas dos pontos de textura são dadas por (1-*v*,1-*u*), onde *v* e *u* são valores de tecelagem.

Geração do Ficheiro do tipo 3D – Atualização

A adição de vetores normais e pontos de textura e a alteração para estruturas *VBO's* com índices nesta fase do projeto implicou a alteração da estrutura dos ficheiros *.3d* criados pelo gerador.

Nesta fase, cada ficheiro gerado possui uma estrutura bem definida onde a primeira linha representa todos os índices dos pontos a desenhar separados por vírgula. De seguida temos uma linha constituída apenas pelo número de vértices (N) sendo que para cada uma das N linhas seguintes temos os três valores das coordenadas (x, y e z) de cada ponto. De modo análogo, estão listados também tanto os vetores normais de cada ponto como os respetivos pontos de textura de cada figura sendo que para os últimos cada linha possui apenas dois valores (x e y).

```
0,1,2,2,3,0,4,5,6,6,7,4,
8
1 0 1
1 0 -1
-1 0 -1
-1 0 1
-1 0 -1
1 0 -1
1 0 1
-1 0 1
8
0 1 0
0 1 0
0 1 0
0 1 0
0 -1 0
0 -1 0
0 -1 0
0 -1 0
8
1 0
1 1
0 1
0 0
1 1
0 1
0 0
1 0

```

Figura 12 - Ficheiro 3d de um plano

Demos

Sistema Solar

Com a adição de texturas e iluminação nesta fase do projeto foram feitas alterações ao anterior ficheiro do sistema solar.

Em primeiro lugar foi adicionada uma fonte de luz pontual no centro do referencial para simular a iluminação por parte do sol. Por outro lado, isto implicou a que a superfície correspondente ao desenho do sol nunca ficasse iluminada. Para resolver este problema, foi feita uma pequena alteração ao nível do gerador no que toca à geração dos pontos da esfera, isto é, as coordenadas dos vetores normais foram alterados para o seu simétrico. Desta forma, o sentido das normais aponta para o interior da esfera por isso apesar da luz estar no interior da figura, esta ficará iluminada na mesma.

Para todos os modelos desenhados foram adicionadas texturas retiradas dos sites http://planetpixelemporium.com/planets.html e http://www.solarsystemscope.com/textures/ que tornaram o desenho cada vez mais realista. Por outro lado, foi adicionada também a rotação dos planetas e luas sobre o seu próprio eixo.

Por ultimo, alterou-se também o código do script em python responsável por desenhar a cintura de asteroides de modo a ser mais geral para que, por um lado fosse possível desenhar mais do que uma cintura de asteroides, e por outro se pudesse escolher tanto o raio da circunferência que delimitam como o número de asteroides que constituem. Sendo assim, foi adicionada uma nova cintura exterior a todos os planetas.

```python
from lxml import etree
import random

mytree = etree.parse('novoSS.xml')
myroot = mytree.getroot()


def draw(radiusMin, radiusMax,numA):
    for i in range(numA):

        angle = random.uniform(0, 360)
        radius = random.uniform(radiusMin, radiusMax)
        altura = random.uniform(-0.1, 0.1)
        r = random.uniform(0.48, 0.53)
        g = random.uniform(0.48, 0.53)
        b = random.uniform(0.48, 0.53)

        scale = random.uniform(0.15, 0.25)
        time = random.uniform(20,30)

        global myroot
        myroot[8].tail = '\n\t'
        doc = etree.SubElement(myroot, "group")
        doc.tail = '\n\t'
        doc.text ='\n\t\t'


        field1 = etree.SubElement(doc, "rotate")
        field1.set("angle", str(angle))
        field1.set("axisX", "0")
        field1.set("axisY", "1")
        field1.set("axisZ", "0")
        field1.tail = '\n\t\t'

        field4 = etree.SubElement(doc, "group")
        field4.tail = '\n\t'
        field4.text = '\n\t\t'

        field5 = etree.SubElement(field4, "translate")
        field5.set("X", str(radius))
        field5.set("Y", str(altura))
        field5.set("Z", "0")
        field5.tail = '\n\t\t\t'

        field6 = etree.SubElement(field4, "scale")
        field6.set("X", str(scale))
        field6.set("Y", str(scale))
        field6.set("Z", str(scale))
        field6.tail = '\n\t\t\t'

        field7 = etree.SubElement(field4, "models")
        field7.tail = '\n\t\t'
        field7.text ='\n\t\t\t\t'

        field8 = etree.SubElement(field7, "model")
        field8.set("file","asteroid.3d")
        field8.set("diffR", str(r))
        field8.set("diffG", str(g))
        field8.set("diffB", str(b))
        field8.tail ='\n\t\t\t'
        i+=1
    doc.tail='\n'


draw(5.5,6.6,500)
draw(15,17.7,450)

mytree.write('SistemaSolar.xml')
```

Figura 13 – Novo código python para o desenho dos asteroides

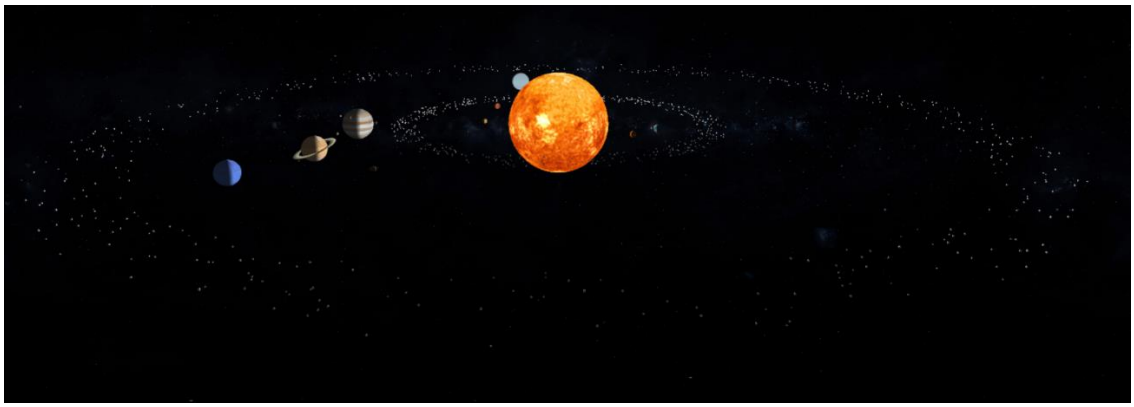

Figura 14 - Output do ficheiro sistemaSolar.xml

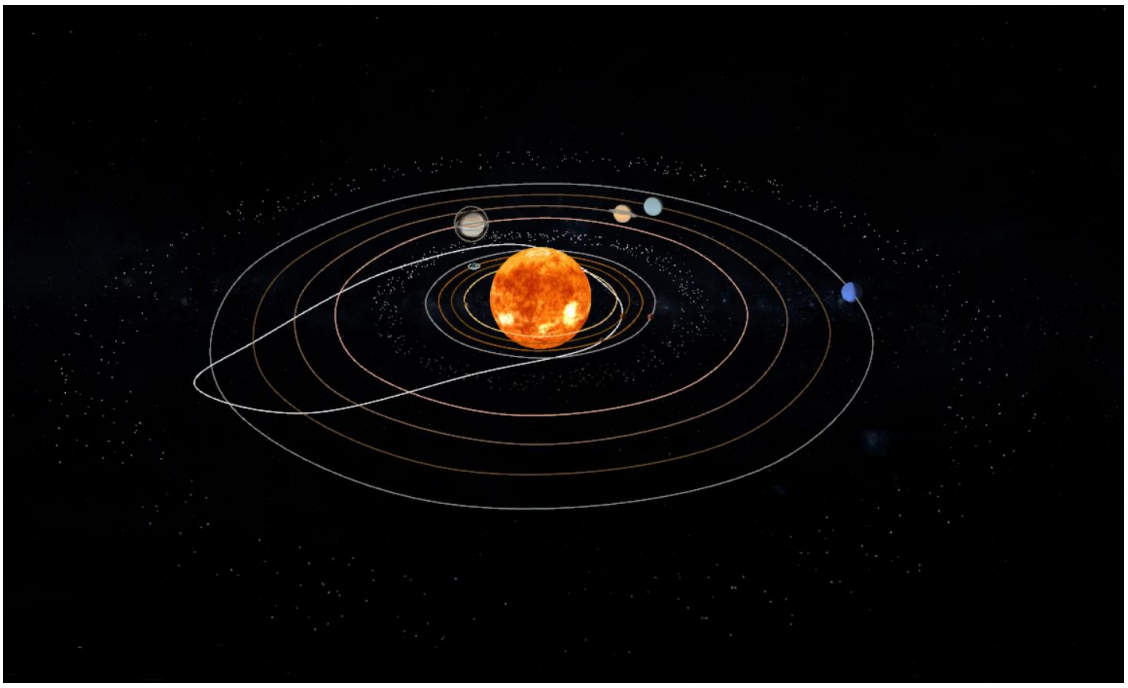

Figura 15 - Output do ficheiro sistemaSolar.xml com o desenho das órbitas

Plano

```
<scene>
    <lights>
        <light type="POINT" posX="0" posY="1" posZ="1" />
    </lights>
    <group>
        <models>
            <model file="plane.3d" texture="de.png"/>
        </models>
    </group>
</scene>
```

Figura 16 – demo2.xml

Figura 17 - Output demo2

Esfera

```
<scene>
    <lights>
        <light type="POINT" posX="0" posY="10" posZ="10" />
    </lights>
    <group>
        <rotate angle="-90" axisX="1" axisY="0" axisZ="0"/>
        <models>
            <model file="sphere.3d" diffR="1" diffG="0.8" diffB="0" ambR="0.7" ambG="0.8" ambB="0"/>
        </models>
    </group>
</scene>
```

Figura 18 - demo3.xml

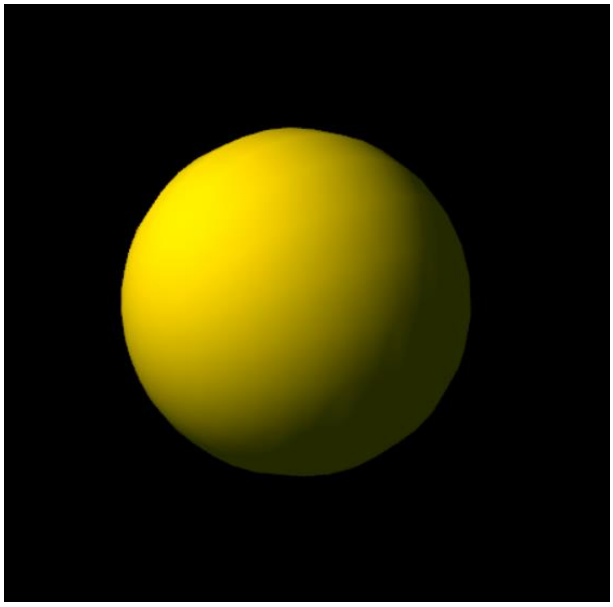

Figura 19 - Output demo3

Torus

```
<scene>
    <lights>
        <light type="DIRECTIONAL" dirX="0" dirY="10" dirZ="10" />
    </lights>
    <group>
        <rotate angle="-90" axisX="1" axisY="0" axisZ="0"/>
        <models>
            <model file="torusDemo4.3d" texture="demo4Img.png"/>
        </models>
    </group>
</scene>
```

Figura 20 - demo4.xml

Figura 21 - Output demo4

Cone

```
<scene>
    <lights>
        <light type="POINT" posX="0" posY="0" posZ="10" />
    </lights>
    <group>
        <rotate angle="-60" axisX="1" axisY="0" axisZ="1"/>
        <models>
            <model file="cone.3d" texture="demo5Img.png"/>
        </models>
    </group>
</scene>
```

Figura 22 - demo5.xml

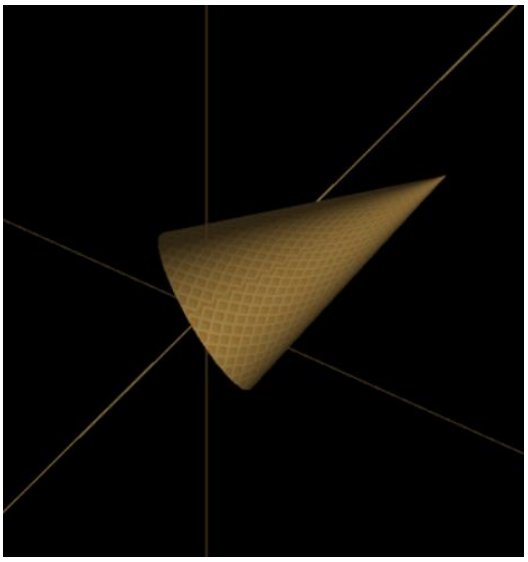

Figura 23 - Output demo5

Caixa

```
<scene>
    <lights>
        <light type="SPOT" posX="0" posY="0" posZ="10" dirX="0" dirY="0" dirZ="-1"/>
    </lights>
    <group>
        <rotate angle="-20" axisX="0" axisY="1" axisZ="0"/>
        <models>
            <model file="box.3d" texture="img5.png"/>
        </models>
    </group>
</scene>
```

Figura 24 - demo6.xml

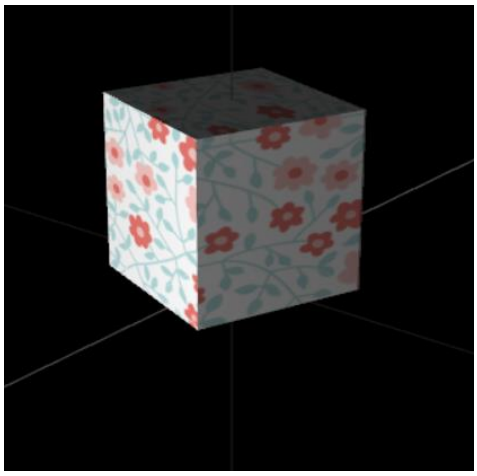

Figura 25 - Output demo6

Teapot

```xml
<scene>
    <lights>
        <light type="POINT" posX="0" posY="10" posZ="10" />
    </lights>
    <group>
        <rotate angle="-90" axisX="1" axisY="0" axisZ="0"/>
        <models>
            <model file="teapot.3d" texture="teapot.jpg" ambR="1" ambG="1" ambB="1"/>
        </models>
    </group>
</scene>
```

Figura 26 - demo1.xml

Figura 27 - Output demo1

Conclusão

Ao longo desta quarta e última fase do trabalho prático, foi possível aplicar e consolidar todo o conhecimento adquirido nas aulas práticas e teóricas da unidade curricular de Computação Gráfica.

O grupo considera que realizou tanto esta fase como o projeto na totalidade com sucesso, uma vez que apresentamos um sistema solar dinâmico e realista, com todas as funcionalidades pedidas no enunciado.

No entanto, há sempre espaço para melhorias, sendo que como trabalhos futuros poder-se-ia implementar a *View Frustrum Culling*.