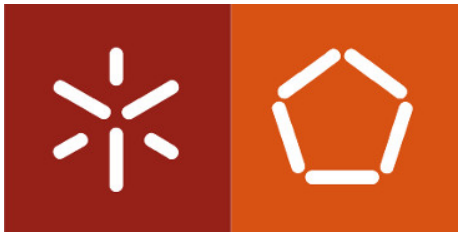
Computação Gráfica

Trabalho Prático - Fase 4

Grupo nr.	38
a83899	André Morais
a84577	José Pedro Silva
a85954	Luís Ribeiro
a84783	Pedro Rodrigues



Mestrado Integrado em Engenharia Informática
Universidade do Minho

Conteúdo

1 Introdução

Nesta quarta e última fase do trabalho prático, procuramos desenvolver e finalizar os trabalhos desenvolvidos previamente nas fases anteriores.

Para isto, foi implementado os conceitos estudados previamente nas aulas teóricas, sobre a Iluminação e Texturas. Então, o Generator passou a gerar normais (para a parte da iluminação) e coordenadas de texturas para cada um dos pontos gerados. Além das mudanças no Generator, a Engine passou a suportar funcionalidades relativas à iluminação e à aplicação de texturas aos modelos.

Os ficheiros XML passarão a possuir informação relativa à iluminação do cenário, assim o parser responsável por ler esses ficheiros é alterado e também é alterada a forma de processamento da informação para construir o cenário.

No presente relatório descrevem-se com detalhe cada uma das componentes referidas em cima, que engloba o trabalho feito nesta fase, desde a descrição das novas funcionalidades do Generator e Engine, até às estratégias usadas no cálculo das normais de cada modelo e das coordenadas de textura.

2 Generator

Nesta fase do trabalho, de modo a adicionar as luzes e as texturas, foi necessário calcular as normais em cada ponto gerado também como as coordenadas da textura. Posto isto, o ficheiro que anteriormente continha os pontos gerados de cada figura, passará a ter o ponto, a sua normal e as coordenadas da textura.

2.1 Adição das Normais

2.1.1 Cone

No caso do cone, as tarefas foram divididas em duas partes distintas: cálculo das normais na base e cálculo das normais no corpo do cone.

Base:

$$\vec{n} = (0, -1, 0)$$

Corpo:

Como podemos observar na Figura 1, se imaginarmos uma linha entre um ponto da base e o ponto do topo do cone, as normais em cima dessa linha, serão todas iguais, então, sendo α o ângulo que varia ao longo da circunferência da base e dh a diferença de alturas entre duas stacks:

$$\forall_{0 \leq \alpha \leq 2 \times \pi} \vec{n} = (sin(\alpha), dh, cos(\alpha))$$

A equação acima representa as normais dos pontos pertencentes à linha entre o ponto com ângulo $alfa$ e o ponto do topo do cone.

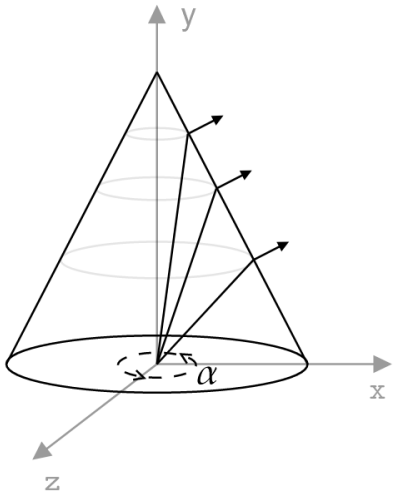


Figura 1: Representação da normal num ponto

2.1.2 Sphere

No caso da esfera, como podemos reparar pela Figura 2, as normais em qualquer ponto da superfície esférica pode ser representada como a extensão do vetor entre o ponto e a origem. Então, sendo *alpha* e *beta* os ângulos representados na figura:

$$\forall_{0\leq\alpha\leq 2\times\pi, 0\leq\beta\leq\frac{\pi}{2}} \vec{n} = (sin(\alpha), sin(\beta), cos(\alpha))$$

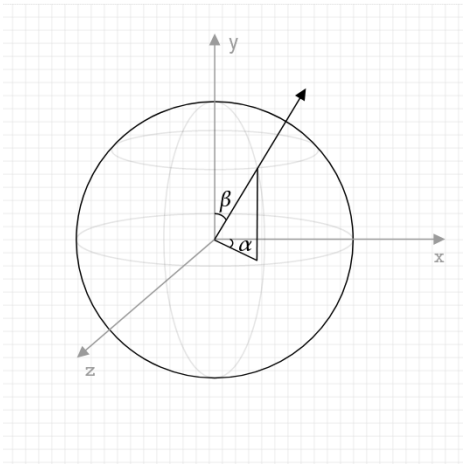


Figura 2: Representação da normal num ponto

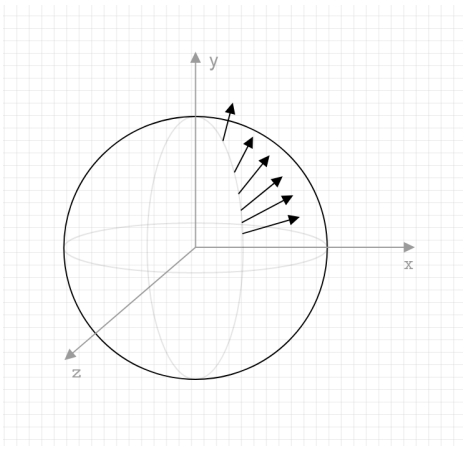


Figura 3: Normais de uma esfera

2.1.3 Plane

Como podemos observar na Figura 4, as normais de um plano são triviais, para a componente superior do plano, teremos $\vec{n} = (0, 1, 0)$ e para a parte inferior $\vec{n} = (0, -1, 0)$.

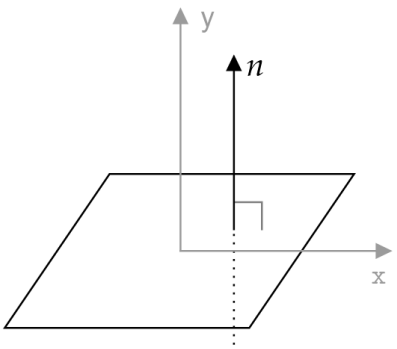


Figura 4: Representação da normal num ponto

2.1.4 Box

Se pensarmos bem, uma caixa não é nada mais nada menos que um conjunto de planos. Então, as normais são todas iguais em cada face.

- **Face Direita** - $\vec{n} = (1, 0, 0)$
- **Face Esquerda** - $\vec{n} = (-1, 0, 0)$
- **Face Cima** - $\vec{n} = (0, 1, 0)$
- **Face Baixo** - $\vec{n} = (0, -1, 0)$
- **Face Frente** - $\vec{n} = (0, 0, 1)$
- **Face Atrás** - $\vec{n} = (0, 0, -1)$

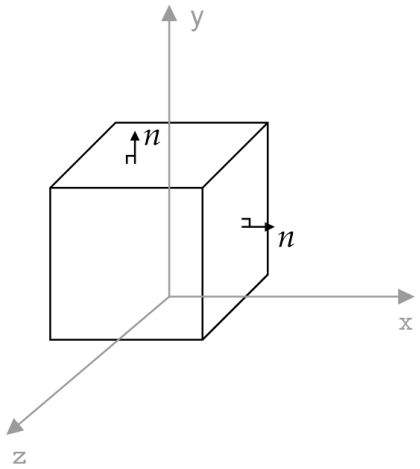


Figura 5: Representação da normal num ponto

2.1.5 Patch Bezier

Como estudado nas aulas teóricas da cadeira, o vetor normal em cada ponto da superfície é definido pelo resultado normalizado do produto cruzado entre os vetores tangentes.

Para calcular os vetores tangentes, é necessário calcular as derivadas parciais em u e v. Então, foi utilizada a mesma fórmula para calcular o ponto, alterando apenas o valor de U e de V respetivamente.

$$\vec{u} = [3u^2 \ 2u \ 1 \ 0]MPM^TV^T$$
$$\vec{v} = UMPM^T[3v^2 \ 2v \ 1 \ 0]^T$$
$$\vec{n} = \vec{u} \times \vec{v}$$

2.2 Adição de Texturas

2.3 Cone

Para calcular as coordenadas das texturas de um cone é necessário diferenciar a base do corpo do cone, então, é calculada a proporção entre o raio da base e altura do cone e dividimos a imagem tal como indicada a Figura 6.

Assim, a textura da base será apenas relacionar o raio representado na imagem com o $cos(alpha)$ e $sin(alpha)$. Na parte do corpo, basta calcular as proporções dadas a cada stack e slice e relacionar consoante iteramos sobre as mesmas.

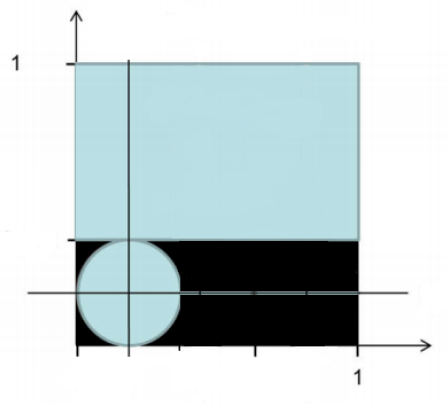


Figura 6: Formato Textura Cone

2.4 Sphere

Neste caso, basta calcular as proporções correspondentes a cada stack e slice e relacionar as coordenadas consoante iteramos sobre as slices e stacks.

2.5 Plane

Como um plano é representado por apenas 4 pontos que correspondem às extremidades, as coordenadas nas texturas irão também corresponder às extremidades, basta relacionar as mesmas.

2.6 Box

Como referido anteriormente, uma caixa é um conjunto de planos, então, é aplicada a mesma ideia dos planos em cada face.

2.7 Bezier

Para cada u e v que variam consoante o valor da tesselagem:

$$\forall_{0\leq u\leq 1, 0\leq v\leq 1} texCoords = (u, v)$$

3 Engine

Nesta fase implementamos o conceito de luz e de textura. Para isso era necessário o cálculo das normais e das coordenadas das texturas, que foram referido na secção anterior.

Como agora os ficheiros dos models contém pontos, normais e coordenadas de textura, foi necessário alterar o modo como lemos cada ficheiro de um model.

3.1 Estrutura

Foram adicionadas variáveis de instância à classe Group. Esta classe passou a guardar uma classe Model em vez de um vetor de pontos. Esta classe Model conterá toda a informação necessária para desenhar o model posteriormente.
Classe Model:

- vector<Point> pontos - guarda todos os pontos de cada triângulo
- vector<Point> normais - guarda as normais de cada ponto
- vector<Point> texturas - guarda as coordenadas da textura de cada ponto
- GLuint vertex - guarda o VBO gerado com os pontos
- GLuint normal - guarda o VBO gerado com as normais
- GLuint texCoords- guarda o VBO gerado com as coordenadas de textura
- GLuint texID - guarda o ID da textura correspondente
- MaterialLight - guarda as informações de luzes da figura

Classe MaterialLight:

Esta classe contém todas as informações acerca da luz difusa, ambiente, emissiva, especular e shininess de cada model.

- difR
- difG
- difB
- ambR
- ambG
- ambB
- emiR
- emiG
- emiB
- speR
- speG
- speB
- shi

3.2 Luzes

Nesta fase, o programa passa a aceitar a adição de luzes nos sistema, do seguinte modo.

```
<light type="POINT" posX="0" posY="0" posZ="0" />
<light type="DIRECTIONAL" X="1" Y="2" Z="3" />
<light type="SPOT" X="1" Y="2" Z="3"
    posX="4" posY="5" posZ="6" angle="45"/>
```

Para isso foi criada uma classe **Light**, que contém:

- x - direção x
- y - direção y
- z - direção z
- posx - posição x
- posy - posição y
- posz - posição z
- angle - ângulo
- type - tipo de luz

Assim, ao ler o ficheiro xml, as luzes são guardadas num vetor presente no sistema, mais tarde, ao desenhar a cena, as luzes serão representadas consoante o seu tipo.

Foi também adicionada a possibilidade de acrescentar luz aos models, da seguinte forma.

```
<model file = "bezier.3d" diffR="0" diffG="0" diffB="1"
    ambiR="0" ambiG="0" ambiB="0.2"/>
```

Podem também ser definidas as componentes especulares, emissivas e shininess. Estas informações são guardadas em cada model utilizando a classe **MaterialLight** apresentada acima.

Ao desenhar os VBO's de cada model, é adicionada estas mesmas componentes.

3.3 Texturas

Para implementar esta componente, procuramos por texturas no ficheiro xml, caso não tenha, desenha a figura sem textura, caso contrário, lê o ficheiro correspondente à textura e guarda o ID da mesma no model. Ao desenhar os vbo's, este model será desenhado com a textura presente no ID guardado.

4 Resultado Final Sistema Solar

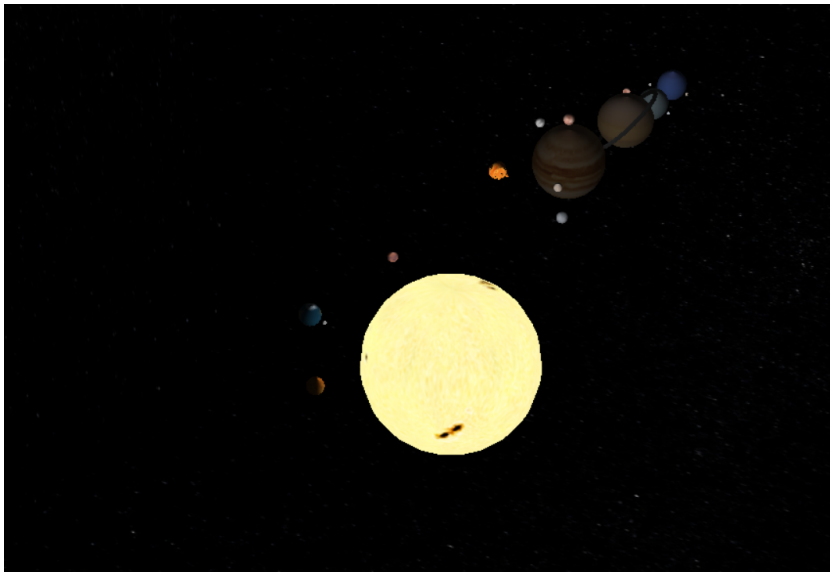


Figura 7: Sistema solar implementado em OpenGL

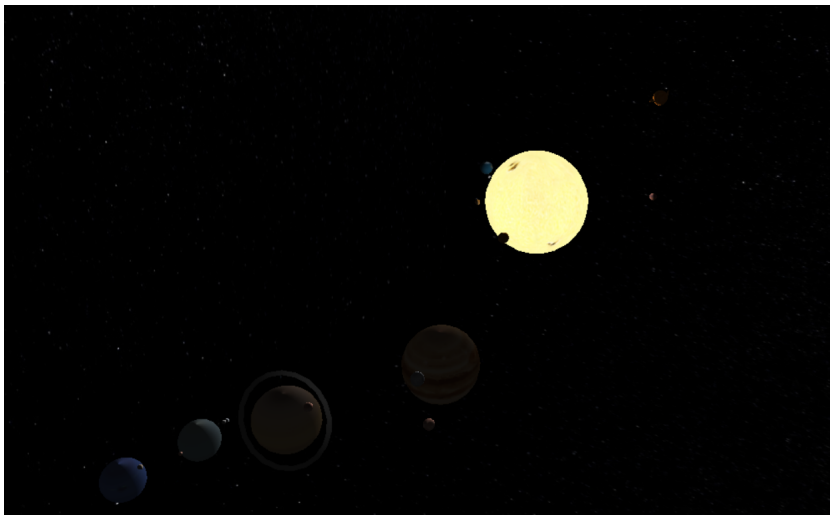


Figura 8: Sistema solar implementado em OpenGL

5 Conclusão

Como já foi referido, esta última fase permitiu-nos explorar os conceitos de Textura e Iluminação. Assim, esta última fase consistiu na geração adicional das normais do modelos e de coordenadas para as texturas. Assim, foi possível a implementação de vários tipos de luzes para a iluminação da cena e a aplicação de texturas aos modelos.

Para concluir, o grupo reconhece que os objetivos globais foram alcançados com sucesso pois desenvolvemos um Sistema Solar animado, minimamente realista, como esperado. O método de aplicação fase a fase do conhecimento permitiu que o nosso conhecimento relativamente às ferramentas e utilidades do OpenGl e do GLUT crescesse bastante com o desenvolvimento destas, consolidando todos os conceitos previamente estudados ao longo desta Unidade Curricular.