Universidade do Minho
Escola de Engenharia

Computação Gráfica

Fase 4 – Coordenadas normais e de texturas

Grupo 52

31 de Maio de 2020

MiEI - 3ºAno - 2ºSemestre

Beatriz Rocha A84003

Filipe Guimarães A85308

Gonçalo Ferreira A84073

José Mendes A75481

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Nesta 4ª fase e última do trabalho é proposto a aplicação de normais e coordenadas de textura para as primitivas já desenvolvidas. Modificamos o nosso formato de ficheiro para abranger estes novos vetores e pontos para que a engine os consiga ler. Aplicamos também novas funcionalidades à engine para que possa aplicar luz bem como texturas recebidas no ficheiro xml.

Capítulo 2

Generator

Para esta fase tivemos de adicionar duas funcionalidades no *generator*: gerar normais e gerar texturas para cada uma das primitivas. Tanto um processo como o outro variam de primitiva para primitiva

2.1 Gerar as normais

2.1.1 Plano

Para gerar as normais no plano temos de apenas atribuir um vetor normal como se pode ver na figura seguinte, ou seja, todos os pontos com o vetor (0,1,0).

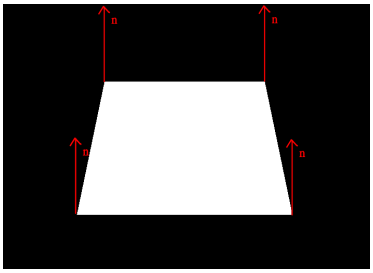

Figura 2.1: Normais no plano

2.1.2 Caixa

A caixa terá uma abordagem similar ao plano. Apenas termos de perceber para que eixo teremos a face virada como se vê na imagem.

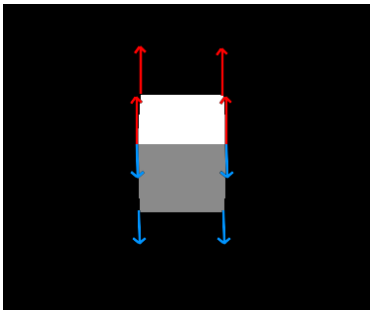

Figura 2.2: Normais na caixa

2.1.3 Esfera

No caso da esfera percebemos que os vetores nomais em cada ponto são os mesmos pontos como podemos ver na imagem.

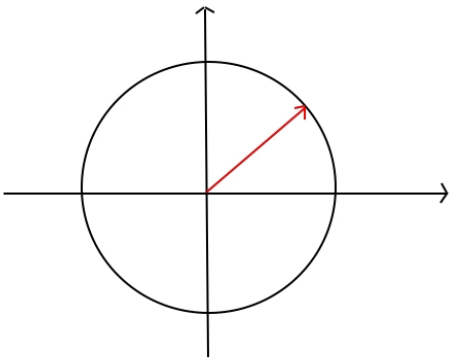

Figura 2.3: normais na esfera

2.1.4 Cone e patch Bezier

No caso do cone e do patch usamos o conceito que vimos no guião 10 que pega em 4 pontos e calcula a secante da interceção dos vetores por eles gerados.

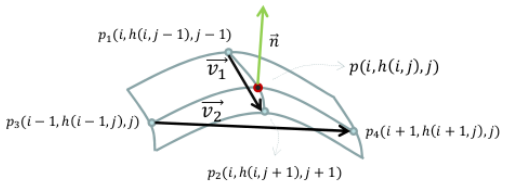

Figura 2.4: Normais apartir de 4 pontos

2.2 Texturas

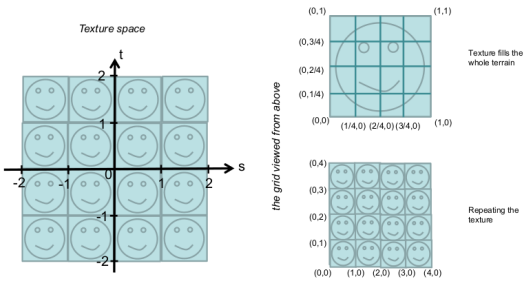

Figura 2.5: Aplicação de texturas

2.3 Plano

Para o plano aplicamos a textura para que fique a cobrir o plano todo como na figura no lado superior direito.

2.4 caixa

Para a caixa consideramos uma caixa desmontada como vemos na imagem e aplicamos a respetiva coordenada de textura a cada ponto em que verticamente é dividida em 3 e horizontalmente em 4.

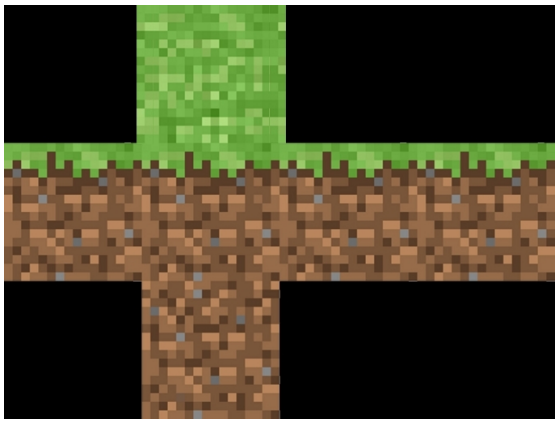

Figura 2.6: Textura exemplo para caixa

2.5 Cone

Para o cone aproveitamos a nossa implementação e pegamos nos angulos e associamos ás coordenadas de textura horizontais que varia de 0 a 1. Verticamente dividimos a altura atual pela máxima e obtemos a coordenada vertical na textura.

2.6 Esfera

Para a esfera aplicamos uma estratégia muito parecida com o cone. Fazendo:

```
Coordenada Horizontal = (angulo_horizontal_atual) / (2 * M_PI);

Coordenada Vertical = (angulo_vertical_atual + (M_PI/2)) / (M_PI);
```

2.7 Patches Bezier

Neste caso aplicamos apenas os pontos x e y como coordenadas de textura fazendo a textura se repetir.

Capítulo 3

Ficheiro xml

3.1 Texturas

Um objetivo nesta fase era a criação de texturas para as primitivas. No caso do sistema solar colocamos as texturas de todos os elementos dentro de uma pasta associando cada uma ao respetivo astro. Como exemplo:

```
<model file="primitives/sphere.3d" texture="planets/sun.jpg"/>
```

Figura 3.1: Aplicação da textura respetiva ao sol

3.2 Sem luz

Para testar textura criamos um xml sem luzes. Colocamos também uma esfera a envolver o nosso sistema solar para criar um efeito de espaço.

```
<group>
    <!-- Sky -->
    <scale X="300" Y="300" Z="300" />
    <models>
        <model file="primitives/sphere.3d" texture="planets/sky.jpg" />
    </models>
</group>
```

Figura 3.2: Espaço com textura

3.3 Com luzes

Nesta fase foram introduzidas as luzes. Para obtermos uma luz uniforme em todo o sistema solar decidimos colocar um ponto de luz em cada eixo logo após onde o sol está desenhado.

```
<lights>
      <light type="POINT" posX="3" posY="0" posZ="0" />
      <light type="POINT" posX="-3" posY="0" posZ="0" />
      <light type="POINT" posX="0" posY="3" posZ="0" />
      <light type="POINT" posX="0" posY="-3" posZ="0" />
      <light type="POINT" posX="0" posY="0" posZ="3" />
      <light type="POINT" posX="0" posY="0" posZ="-3" />
</lights>
```

Figura 3.3: Pontos de luz

Capítulo 4

Engine

4.1 Processar o ficheiro xml

Com o objetivo de satisfazer as novas capacidades do motor gráfico, iluminação, cor, sombra e texturas, houve a necessidade de especificar novas condições de *parsing*. Como demonstra a figura abaixo, estes casos foram subdivididos pelas *tags* existentes nas passadas fases e a nova inserida.

```
struct model {
    std::vector<float> * pontos;
    std::vector<float> * pontos_textura;
    std::vector<float> * pontos_normais;
    GLuint vertexBuffer[1];
    GLuint normalBuffer[1];
    GLuint textureBuffer[1];


    std::string * textura_file;
    GLuint idTextura;
    bool temTextura;
    bool temCor;

    float diffuse[4];
    float specular[4];
    float emissive[4];
    float ambient[4];
};
```

Figura 4.1: Model struct

De acordo ao *parser* previamente desenvolvido com a ferramenta *tinyXML* acrescentou-se ainda novas variáveis para conter os diversos parâmetros para os cálculos.

4.2 Cor

Para a cor foi necessário adicionar a *sturct model* 4 novas variáveis, as quais contêm os valoras representativos das componentes difusa,especular, emissiva e ambiente. Estas são representadas pela escala RGBA, como *default* são inicializadas com os valores predefinidos do *OpenGL*. Para pode alterar os diversos valores para atribuir cor ao solido foi adicionado o *parsing* dos novos campos

diffR, diffG, diffB, specR, specG, specB, ambR, ambG, ambB, emisR, emisG e emisB.

4.3 Textura & Normais

A adição de texturas e normais ao trabalho teve lugar na *struct model* que possui os pontos importados do *parse* dos demais ficheiros passados por parâmetros no XML. Ambas são importadas para vetores de *floats*, que posteriormente são utilizados para inicializar os VBO's correspondente aos mesmos. Isto permite garantir uma boa performance como também fácil manipulação dos dados, mantendo assim um arquitetura simples e versátil. Como tal é adicionado o *parse* de um campo *texture*.

4.4 Luz

No âmbito de completar a *engine* com as ferramentas disponibilizadas pelo *OpenGL*, a sua documentação indica que uma luz possui uma posição no espaço, componentes de ambiente difusa e especular, como também um tipo expoente, direção e corte. Estes componentes permitem dar a preceção de que material é que está a refletir a cor, já o tipo modela a luz para por exemplo ser direcional como uma lanterna.

```
struct light {
    GLenum number;  // number of the light
    float pos[4];   // position
    float amb[4];   // ambient component
    float dif[4];   // diffuse component
    float spec[4];  // specular component
    float spotD[3]; // direction
    float spotExp;
    float spotCut;
};
```

Figura 4.2: Light struct

Os campos adicionados para possibilitar a personlização da luz foram type, posX, posY, posZ, diffR, diffG, diffB, specR, specG, specB, ambR, ambG, ambB, dX, dY, dZ, exp e cut, os quais têm correspondência á documentação do *OpenGL*.

Capítulo 5

Teste do Sistema Solar

5.1 Manual

À semelhança das outras fases criamos uma pasta com os ficheiros necessários para ver o sistema solar. Para correr temos apenas de entrar na pasta.

```
$ cd solarSystem
```

Compilar a engine na mesma com os seguintes comandos.

```
$ cmake ../engine/
$ make
```

Agora basta passar como argumento o ficheiro xml do sistema solar. Colocamos um modelo sem luz:

```
$ ./engine xml/SolarSystem_noLight.xml
```

E um com a respetiva iluminação:

```
$ ./engine xml/SolarSystem.xml
```

5.2 Imagens do Sistema Solar

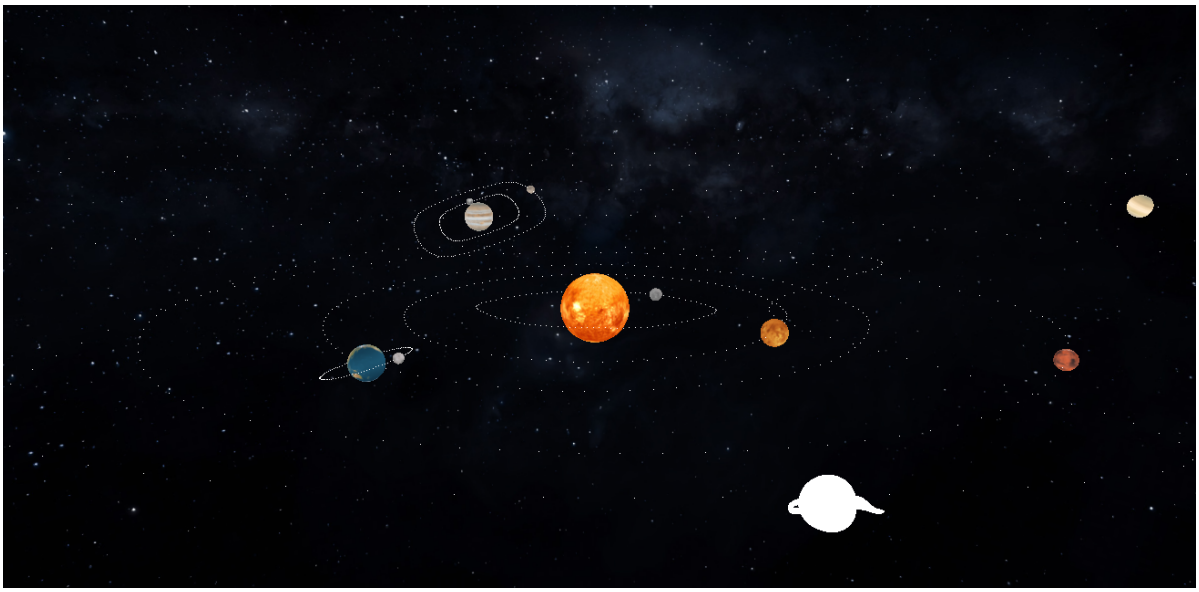

Figura 5.1: Sistema solar sem luz e com fundo

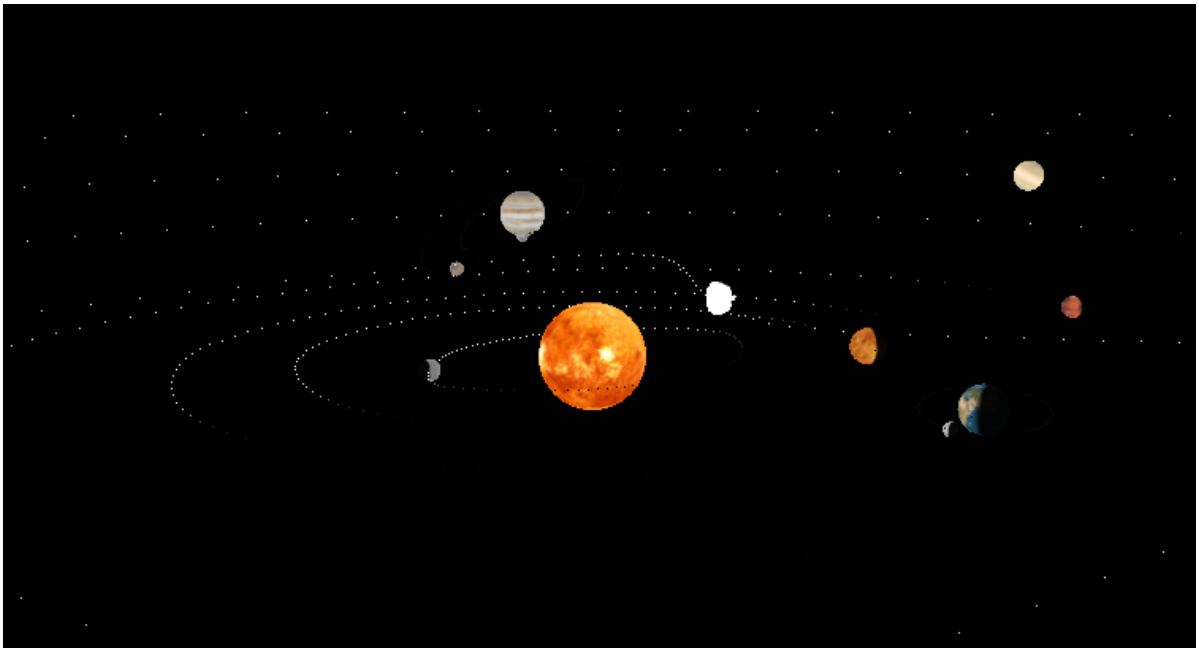

Figura 5.2: Sistema solar com luz

Capítulo 6

Demos

Criamos para efeitos de demonstração uma pasta *demos* para efeitos de demonstração de texturas e luz nas primitivas. Para compilar o programa apenas tem de se executar o comando:

```
$ cmake ../engine/
$ make
```

Para selecionar o ficheiro xml apenas fazemos:

6.1 Plano

Exemplo de plano com uma textura de concreto aplicada.

```
$ ./engine xml/plano.xml
```

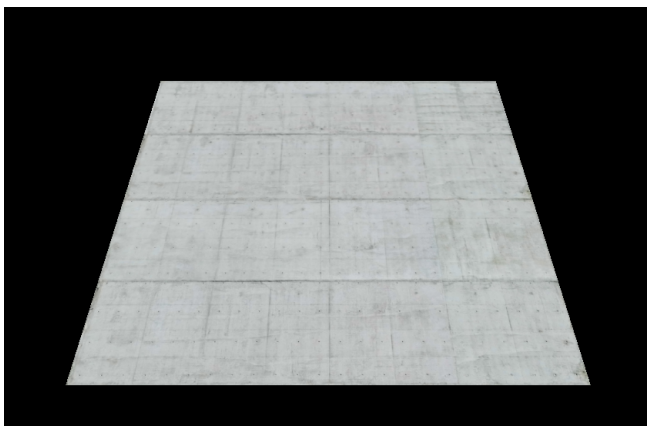

Figura 6.1: Plano com textura

6.2 Caixa

Exemplo de uma caixa com textura aplicada, emissividade a 50% e um ponto de luz acima dela.

```
$ ./engine xml/caixa.xml
```

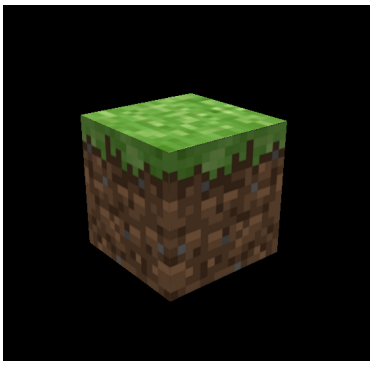

Figura 6.2: Caixa com luz e textura

6.3 Cone

Exemplo de um cone com textura aplicada, luz difusa produminante na componente verde e vermelha e um ponto de luz ao seu lado.

```
$ ./engine xml/cone.xml
```

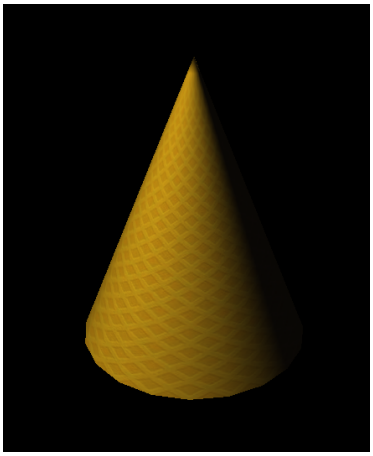

Figura 6.3: Cone com luz e textura

6.4 Esfera

Exemplo de uma esfera com a textura da Terra e um ponto luz ao seu lado.

```
$ ./engine xml/sphere.xml
```

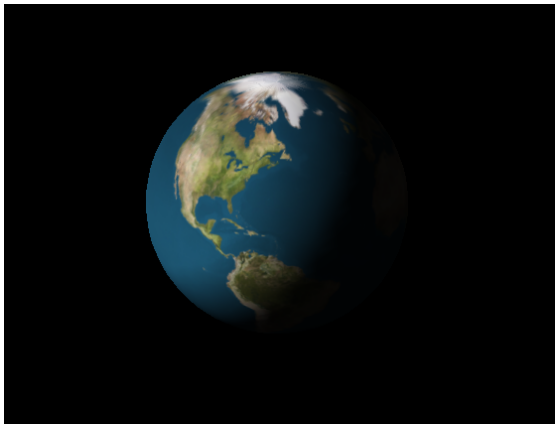

Figura 6.4: Cone com luz e textura

6.5 Teapot

Exemplo do teapot com textura e sem textura com um ponto de luz.

```
$ ./engine xml/teapot.xml
```

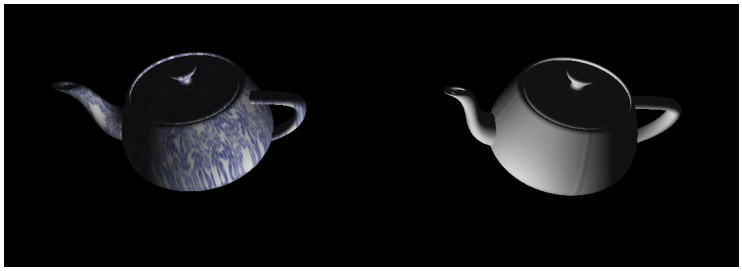

Figura 6.5: Teapots com luz e textura

Capítulo 7

Conclusão e Análise de Resultados

Nesta fase aprendemos a gerar normais para qualquer primitiva que geramos na 1ª fase do projeto, compreendendo o conceito por detrás das mesmas, Aprendemos a mapear texturas tanto como um só como repetindo a mesma. Percebemos, também, os diferentes tipos de luz nos objetos. Pensamos que conseguimos satisfazer os requisitos com os resultados obtidos.