Universidade do Minho

Licenciatura em Engenharia Informática

Computação Gráfica

Phase 4 - Normals and Texture Coordinates

Grupo 12

Ana Murta (A93284)

Ana Henriques (A93268)

Leonardo Freitas (A93281)

Rui Coelho (A58898)

junho, 2022

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

A fase final do trabalho prático, realizado no âmbito da disciplina de Computação Gráfica, procura finalizar o desenvolvimento do modelo anteriormente construído para o Sistema Solar, introduzindo iluminação e texturas, a fim de obter uma reprodução mais realista do modelo.

Como tal, o trabalho desenvolvido na fase final recorreu a conceitos como a vetores normais e coordenadas de textura para a implementação das funcionalidades pretendidas. Estas funcionalidades exigiram alterações ao código previamente desenvolvido, quer a nível do `GENERATOR`, quer a nível do `ENGINE`. Assim sendo, ao longo do presente relatório, é dado a conhecer o trabalho desenvolvido, assim como as estratégias adotadas para o efeito.

Capítulo 2

Generator

Um dos objetivos estipulados para a presente fase centra-se na aplicação de iluminação e de texturas aos corpos celestes do modelo desenvolvido para o Sistema Solar. A aplicação de texturas e de iluminação foi possível através do cálculo das coordenadas de textura e do cálculo das normais de cada vértice. Assim sendo, para as primitivas usadas na cena do Sistema Solar, foram alteradas as diversas funções de cálculo dos pontos das formas geométricas, a fim de introduzir, também, a determinação das normais dos vértices e das coordenadas de textura.

2.1 Formato *.3d*

Atendendo à necessidade de calcular as normais e pontos de textura dos pontos, o formato dos ficheiros *.3d* foi estendido de modo a acomodar esta nova informação. Deste modo, e tal como se pode confirmar pelo excerto apresentado a seguir, o ficheiro primeiramente indica o número de pontos calculados para a figura geométrica a gerar e, para cada ponto, apresenta, pela seguinte ordem, as coordenadas do ponto, as coordenadas normalizadas e as coordenadas de textura.

```
2400
0.148778,0.987688,0.048341,0.147327,0.978054,0.147327,0.050000,0.050000
0.156434,0.987688,0.000000,0.154555,0.975820,0.154555,0.000000,0.000000
0.000000,1.000000,0.000000,0.000000,1.000000,0.000000,0.050000,0.000000
```

2.2 Plano

A geração de normais para os vértices do plano é um procedimento trivial. Para cada ponto do plano, a normal pode ser definida como sendo o vetor vertical `(0,1,0)`. Assim sendo, aquando do cálculo dos pontos que definem o plano é, também, introduzida a normal desse vértice. A Figura 2.1 apresenta um breve esquema dos vetores normais para os pontos do plano.

Os extremos do plano tomarão as seguintes coordenadas de textura: `(0,0)`, `(0,1)`, `(1,1)` e `(1,0)`. No entanto, para calcular as coordenadas de textura associadas a cada ponto que define a primitiva plano, aplicou-se a tática usada para determinar esses mesmos pontos na Fase 1 – representada na Figura 2.2, em que o comprimento de uma divisão do plano é `1/divisions`.

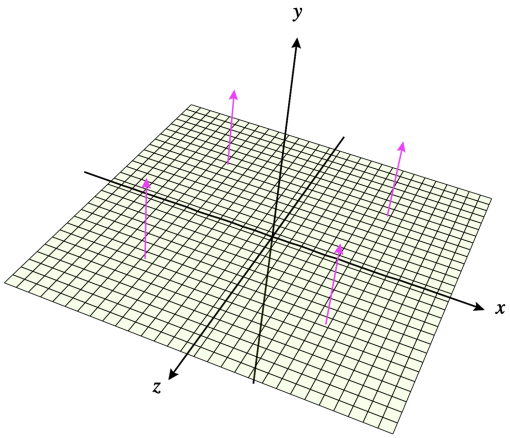

Figura 2.1: Vetores normais do plano.

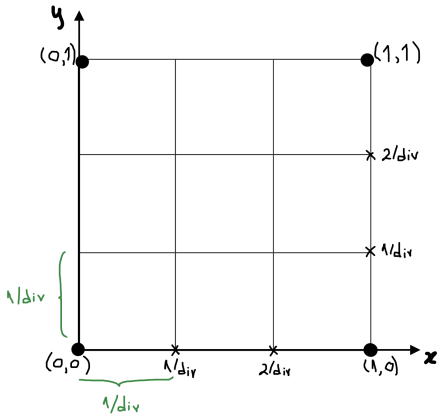

Figura 2.2: Coordenadas de textura do plano.

2.3 Cubo

De modo a acrescentar o cálculo das coordenadas de textura e das normais do cubo, foi alterada a função *generateBox*, que anteriormente determinava somente os pontos necessários para a representação dos pontos do cubo. Cada face do cubo representa, no fundo, uma porção de um plano – o que permite estender a trivialidade do raciocínio efetuado para o plano em termos da determinação da normal dos pontos. Assim sendo, para cada plano, e tal como se pode verificar na Figura 2.3, consideraram-se os seguintes vetores normais para as faces:

- Baixo: `(0,-1,0)`
- Cima: `(0,1,0)`
- Direita: `(1,0,0)`
- Esquerda: `(-1,0,0)`
- Frente: `(0,0,1)`
- Trás: `(0,0,-1)`

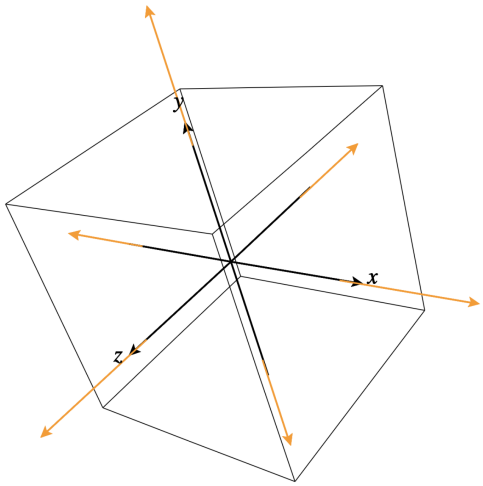

Figura 2.3: Vetores normais do cubo.

No que diz respeito às coordenadas de textura, adotou-se a seguinte estratégia: desdobrar as faces do cubo em 6 partes que constituem a imagem ilustrada na Figura 2.4. No entanto, é preciso ter em conta que o cubo é desenhado com uma determinada dimensão e, como tal, o comprimento e a altura da divisão de uma face são calculados, respetivamente, a partir das seguintes fórmulas:

```
textureHoriz = 1.0f / (3.0f * divisions)
textureVert = 1.0f / (2.0f * divisions)
```

Claramente que, à medida que se cresce no eixo do x, o valor de `textureHoriz` aumenta, assim como, ao crescer no eixo do y, o valor de `textureVert` também aumenta.

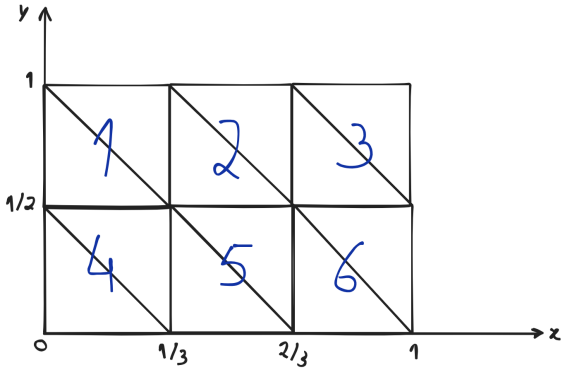

Figura 2.4: Coordenadas de textura do cubo.

Deste modo, ao calcular os pontos que definem cada face, determinamos também as coordenadas de textura do cubo, verificando-se ser as seguintes:

- Face 1: `(0,1/2)`
- Face 2: `(1/3,1/2)`
- Face 3: `(2/3,1/2)`
- Face 4: `(0,0)`
- Face 5: `(1/3,0)`
- Face 6: `(2/3,0)`

2.4 Cone

A determinação das normais do cone foi efetuada com recurso à função *normalize*[1], cujo código se encontra abaixo apresentado. Com o seu auxílio, cada um dos pontos do cone é normalizado.

```
void normalize(float* a) {
    float l = sqrt(a[0] * a[0] + a[1] * a[1] + a[2] * a[2]);
    if (l != 0) {
        a[0] = a[0] / l;
        a[1] = a[1] / l;
        a[2] = a[2] / l;
    }
}
```

[1] A mesma função foi, posteriormente, usada para normalizar as coordenadas esféricas da esfera e semi-esfera.

Relativamente à base do cone, a normal a cada um dos seus pontos está sempre apontada para "baixo"; logo, é `(0,-1,0)`. Já para a lateral do cone, a normal a cada um dos seus pontos é determinada através da seguinte fórmula, sendo que `normalized_y` igual a `cos(atan(height/radius))`:

- Ponto 1: `(cos(alpha), normalized_y, sin(alpha))`
- Ponto 2: `(cos(alpha2), normalized_y, sin(alpha2))`
- Ponto 3: `(cos(alpha), normalized_y, sin(alpha))`
- Ponto 4: `(cos(alpha2), normalized_y, sin(alpha2))`

Na Figura 2.5, temos, então, a representação dos vetores normais à base e à lateral do cone, tal como previamente apresentados e calculados.

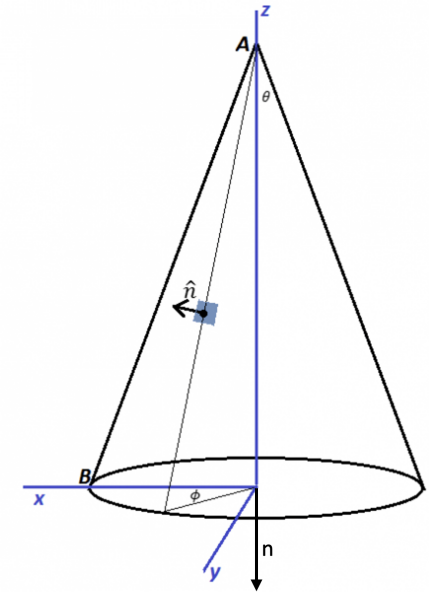

Figura 2.5: Vetores normais do cone.

2.5 Esfera

O cálculo das normais da esfera é idêntico ao aplicado no cone, recorrendo-se, para ambos, à função *normalize*, introduzida anteriormente na Secção 2.3, para normalizar cada um dos pontos que define esta primitiva. Deste modo, para determinar o vetor normal em qualquer ponto da esfera, recorre-se à seguinte expressão: `(cos(`θ`)*cos(`φ`), sin(`φ`), cos(`θ`)*cos(`φ`))`.

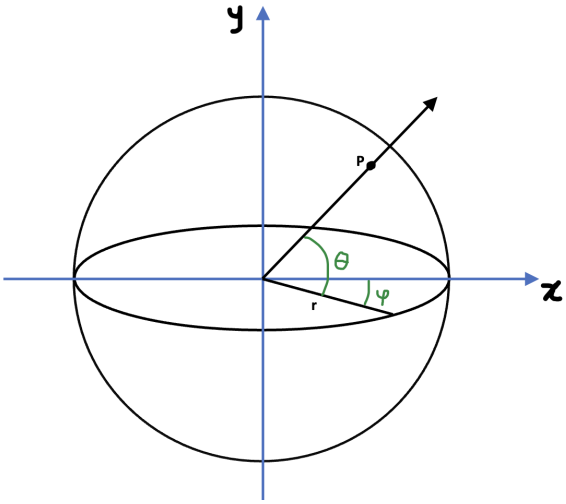

Figura 2.6: Vetores normais da esfera.

Para determinar as coordenadas de textura, adotou-se, também, a estratégia de desdobrar a esfera numa imagem – ilustrada na Figura 2.7, sendo o seu comprimento igual ao número de `totalSlices` e a sua largura igual ao número de `totalStacks` atribuídos à esfera. Tal como para o cubo, tanto o comprimento como a largura não podem ultrapassar o valor de 1. Posto isto, sabe-se que o comprimento e a largura de uma divisão da esfera são calculados, respetivamente, a partir das seguintes fórmulas:

```
textureHoriz = 1.0 / totalSlices
textureVert = 1.0 / totalStacks
```

Assim como averiguado para o cubo, à medida que se percorrer o eixo do x, o valor de `textureHoriz` varia e, ao percorrer o eixo do y, o valor de `textureVert` também varia.

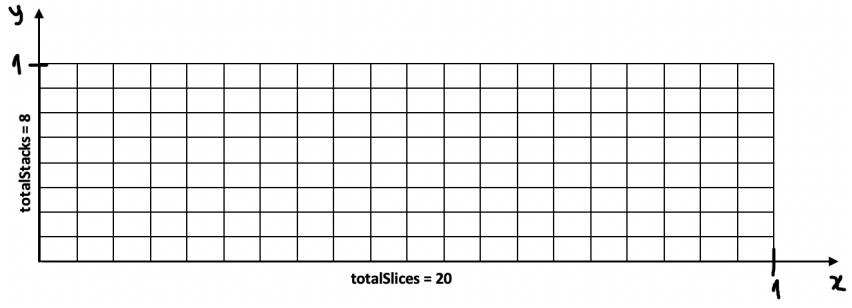

Figura 2.7: Coordenadas de textura da esfera.

2.6 Toro

Para determinar as normais a cada vértice da primitiva toro, vulgo *donut*, seguiu-se a estratégia adotada para a esfera, em que, para cada ponto, o vetor normal é calculado pela seguinte expressão: `(cos(`θ`)*cos(`φ`), sin(`θ`), cos(`θ`)*sin(`φ`))`.

Em relação às coordenadas de textura, seguiu-se o mesmo procedimento para calcular as coordenadas de textura da esfera.

2.7 Cilindro

Para determinar os vetores normais do cilindro, foi tido em conta a posição dos triângulos gerados. Tal como podemos ver pela Figura 2.8, no topo do cilindro, as normais têm direção vertical e apontam para cima – `(0,1,0)`; porém, na base do cilindro, as normais, apesar de continuarem a ter direção vertical, já apontam para baixo – `(0,-1,0)`.

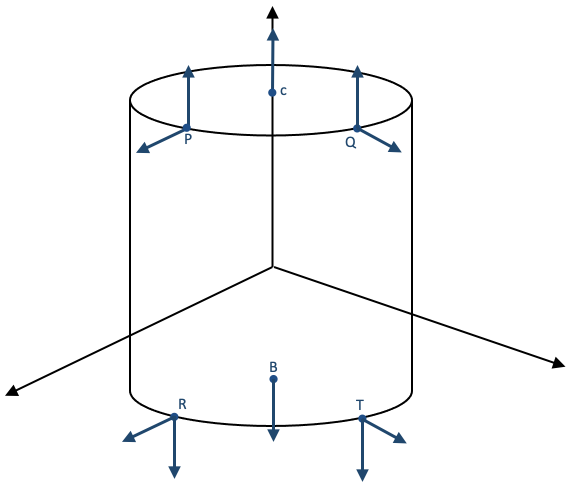

Figura 2.8: Vetores normais do cilindro.

Capítulo 3

Ficheiro XML

Comparativamente com as fases anteriores, e com o objetivo de permitir a reprodução de um modelo mais realista através de iluminação e de texturas, foram introduzidos novos identificadores no ficheiro XML. Esses identificadores, que remetem, respetivamente, para texturas e iluminação, são `texture` e `color`. Dentro do campo da iluminação, é possível atribuir componentes de cor difusa (atributo `diffuse`), especular (atributo `specular`), ambiente (atributo `ambient`), emissiva (atributo `emissive`) e brilho (atributo `shininess`).

```
<!-- Sol -->
<group>
    <transform>
        <translate x="0" y="0" z="0"/>
        <scale x="1.1" y="1.1" z="1.1"/>
        <!--color x="255" y="170" z="0"/-->
    </transform>
    <models>
        <model file="sphere.3d">
            <texture file="sun.jpg"/>
            <color>
                <diffuse R="200" G="200" B="200"/>
                <ambient R="50" G="50" B="50"/>
                <specular R="0" G="0" B="0"/>
                <emissive R="0" G="0" B="0"/>
                <shininess value="0"/>
            </color>
        </model>
    </models >
</group>
```

Figura 3.1: Excerto do ficheiro XML.

Capítulo 4

Engine

Para a seguinte fase, de modo a incorporar as novas funcionalidades em termos de textura e iluminação, foi necessário estender as funcionalidades do ENGINE. A primeira alteração efetuada permitiu incorporar movimentação da câmera com o rato do computador. Assim sendo, foram introduzidas funções para o efeito: *processMouseButtons* e *processMouseMotion* para o efeito.

Uma vez que o ficheiro modelo do Sistema Solar, previamente apresentado, integra informação adicional, comparativamente com a fase anterior, foi necessário alterar a função *parseGroup* de modo a ser possível processar, corretamente, a nova informação do ficheiro XML. Estas alterações centraram-se na leitura dos componentes referentes à textura dos objetos celestes e da informação acerca da iluminação – para a leitura da informação referente à iluminação foi introduzida a função *parseLights*. O carregamento das texturas foi efetuado com recurso à função *loadTextura*, cujo código se encontra abaixo apresentado.

```
int loadTextura(std::string s) {
    unsigned int t, tw, th;
    unsigned char* texData;
    unsigned GLuint;

    ilInit();
    glEnable(GL_TEXTURE_2D);
    ilEnable(IL_ORIGIN_SET);
    ilOriginFunc(IL_ORIGIN_LOWER_LEFT);
    ilGenImages(1, &t);
    ilBindImage(t);
    ilLoadImage((ILstring)s.c_str());
    tw = ilGetInteger(IL_IMAGE_WIDTH);
    th = ilGetInteger(IL_IMAGE_HEIGHT);
    ilConvertImage(IL_RGBA, IL_UNSIGNED_BYTE);
    texData = ilGetData();
    glGenTextures(1, &GLuint);
    glBindTexture(GL_TEXTURE_2D, GLuint);
    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);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, tw, th, 0, GL_RGBA, GL_UNSIGNED_BYTE, texData);
```

```
    glBindTexture(GL_TEXTURE_2D, 0);

    return GLuint;
}
```

Capítulo 5

Conclusão

A quarta fase do trabalho prático procurou concluir o desenvolvimento do modelo representativo do Sistema Solar, que tem vindo a ser construído/preparado desde a primeira fase do trabalho prático. Particularmente, a fase final do projeto visou inserir textura e iluminação no modelo previamente apresentado na fase anterior.

No que toca aos objetivos estipulados para a fase final de entrega do modelo do Sistema Solar, pode dizer-se que os mesmos não foram alcançados com sucesso. A inserção de textura nos diversos corpos celestes que integram o Sistema Solar foi algo que o grupo não foi capaz de executar como pretendia. O modelo final apresentado para a corrente fase incorpora a inserção de texturas, contudo as aplicação das mesmas demonstra-se pouco correta, uma vez que as imagens geradas não correspondem ao objetivo pretendido. Tal efeito verifica-se pela incerteza no cálculo das coordenadas de textura para os diversos pontos que integram as figuras da cena apresentada, o que se traduziu numa reprodução errada das imagens dos corpos.

Adicionalmente, a inserção de iluminação no modelo foi, também, algo que o grupo não foi capaz de trabalhar. Embora tenham sido desenvolvida uma estrutura, em termos de dados e de funções, para trabalhar em termos de iluminação, a mesma não foi implementada. No decurso do desenvolvimento da componente de iluminação o grupo deparou-se com obstáculos que não foi capaz de ultrapassar.

Posto isto, o trabalho desenvolvido nesta fase permitiu: (i) inserir movimentação da câmera em função de input recebido com o rato do computador; (ii) extensão do modelo XML desenvolvido para o Sistema Solar, incorporando agora informação acerca de textura e ilumincação; (iii) cálculo de normais e coordenadas de textura. Embora tenham sido inseridas estas novas componentes, o grupo efetua uma avaliação pouco positiva do trabalho desenvolvido, face aos objetivos esperados.