Computação Gráfica

Trabalho Prático CG - 2017/18

Fase 4 - Normais e Coordenadas de Textura

Relatório de Desenvolvimento

André Pereira (A79196)
Filipe Miranda (A78992)
José Alves (A78178)

21 de Maio de 2018

Conteúdo

Capítulo 1

Introdução

O trabalho prático **Normais e Coordenadas de Textura** surge inserido na disciplina de Computação Gráfica e corresponde à quarta e última fase do trabalho prático desta unidade curricular. Versa sobre o uso do **TinyXML** e do **OpenGL** e corresponde, essencialmente, em atualizar o gerador para gerar as coordenadas de textura e as normais para cada vértice de um ficheiro modelo e ativar as funcionalidades de iluminação e texturização no motor. Ao longo deste documento podemos encontrar a exposição do problema (a sua descrição informal, os dados do problema e o que foi pedido que concebessemos), o desenho da resolução do problema e alguns resultados/testes. Por fim, apresenta-se uma breve conclusão e o código do programa.

Estrutura do Relatório

Este relatório inicializa-se com uma breve introdução ao trabalho prático. No capítulo 2 é feita uma análise do problema proposto, onde se apresenta o enunciado do trabalho e a especificação dos seus requisitos. De seguida, são apresentadas as estratégias para a resolução do mesmo, onde no capítulo 4 são especificados os resultados obtidos. Finalizando, no capítulo 5 encontram-se as conclusões a que chegamos e algumas reflexões.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

Esta última fase do trabalho prático corresponde a atualizar o gerador de maneira a que gere as coordenadas de textura e as normais para cada vértice e ativar as funcionalidades de iluminação e texturização no motor, bem como ler e aplicar as coordenadas normais e de textura dos ficheiros modelo. Como exemplo, podemos observar em baixo um modelo com textura e de seguida um modelo colorido (no que diz respeito à componente difusa):

```
<model file="sphere.3d" texture="earth.jpg"/>
<model file="bla.3d" diffR=0.8 diffG=0.8 diffB=0.15 />
```

2.2 Especificação dos Requisitos

Permitir a definição das componentes de cor difusa, especular, emissiva e ambiental no parser do XML, bem como definir as fontes de luz, que podem ser do tipo PONTO, DIRECIONAL ou SPOT. De notar que para estes dois últimos tipos, podem ser necessários argumentos diferentes. Por exemplo, uma luz direcional requer uma direção e não uma posição. O demo requisitado para esta fase corresponde a um model do sistema solar animado com texturas e iluminação.

2.3 Exemplo

Exemplo de um ficheiro XML com as novas funcionalidades:

```
<scene>
    <lights>
        <light type="POINT" posX=0 posY=10 posZ=0 />
    </lights>
    <group>
        <translate X=5 Y=0 Z=2 />
        <rotate angle=45 axisX=0 axisY=1 axisZ=0 />
        <models>
            <model file="sphere.3d" />
        </models>
    </group>
</scene>
```

Capítulo 3

Concepção/Desenho da Resolução

3.1 Texturas

Tal como os VBOs, as texturas são objetos que têm de ser gerados inicialmente, invocando uma função. Tipicamente são usadas imagens para decorar modelos 3D e podem armazenar tipos diferentes de dados. Para o caso do sistema solar, são aplicadas diferentes texturas e as imagens de textura são provenientes do endereço web fornecido no enunciado do trabalho. Para carregar uma textura no motor, abre-se o ficheiro da imagem, define-se a origem do espaço da imagem no DevIL e posteriormente converte-se a imagem para RGBA.

```
ilInit();
ilEnable(IL_ORIGIN_SET);
ilOriginFunc(IL_ORIGIN_LOWER_LEFT);
ilGenImages(1, &t);
ilBindImage(t);
ilLoadImage((ILstring)s.c_str());
(...)
ilConvertImage(IL_RGBA, IL_UNSIGNED_BYTE);
```

Obtém-se as informações necessárias (altura da imagem, comprimento da imagem, etc), cria-se um slot de textura e faz-se bind, define-se os parâmetros das texturas e manda-se os dados das texturas para o OpenGL.

```
tw = ilGetInteger(IL_IMAGE_WIDTH);
th = ilGetInteger(IL_IMAGE_HEIGHT);
texData = ilGetData();

glGenTextures(1, &texID);

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_MIPMAP_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);
```

Para cada vértice, é definido a sua coordenada de textura (cujo método é abordado em 3.2), são ativados os arrays no initGL do motor e adiciona-se um array com as coordenadas de textura na classe dos modelos.

```
void initGL() {

    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_NORMAL_ARRAY);
    glEnableClientState(GL_TEXTURE_COORD_ARRAY);

}

class Model : public Group {

    (...)

    glGenBuffers(1, &texCoords);
    glBindBuffer(GL_ARRAY_BUFFER, texCoords);
    glBufferData(GL_ARRAY_BUFFER, sizeof(float) * 2 * count, t, GL_STATIC_DRAW);

}
```

3.2 UV Mapping

Para definir as coordenadas de textura das esferas, usamos o mapeamento UV. O UV mapping é um processo de modelação 3D que consiste em projetar uma imagem 2D na superfície de um modelo 3D para mapeamento de textura.

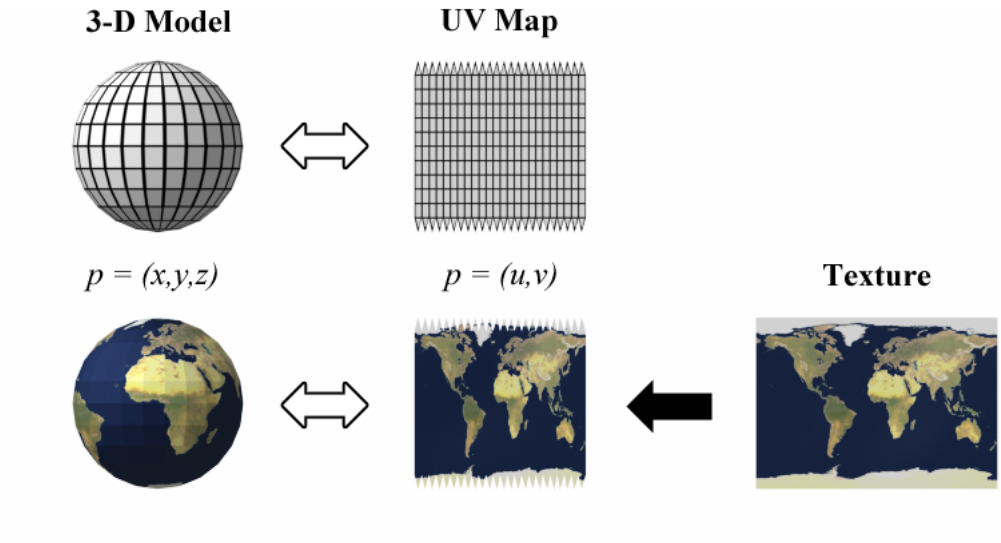

Figura 3.1: Aplicação de uma textura no espaço UV em relação ao efeito em 3D.

Para cada ponto P da esfera, é calculado $\hat{d}$, que é o vetor unitário desde P até à origem da esfera. Assumindo que os polos da esfera estão alinhados com o eixo Y, as coordenadas UV, que se encontram no intervalo [0,1], podem ser calculadas da seguinte forma:

$$u = 0.5 + \frac{\arctan 2(d_z, d_x)}{2\pi}$$

$$v = 0.5 - \frac{\arcsin(d_y)}{\pi}$$

De seguida podemos averiguar um extrato do código que determina as coordenadas de textura de uma esfera. O processo acaba por ser semelhante quer para as slices, quer para as stacks.

```
void sphere(float radius, int slices, int stacks) {

    float alfa = M_PI / stacks;
    float alfaux = M_PI / stacks;
    float altura = 0;
    float altura2 = sin(alfa) * radius;
    float angulo = (2 * M_PI) / slices;
    float raioaux = radius;
    float raioaux2 = sqrtf((radius*radius) - (altura2 * altura2));

    long int num_vertices = ((stacks - 2)*(6 * slices)) + (6 * slices);

    int vertex = 0;

    float *n, *t;
    n = (float *)malloc(sizeof(float) * num_vertices * 3);
    t = (float *)malloc(sizeof(float) * num_vertices * 2);

    fprintf(ficheiro, "%ld\n", num_vertices);

    if (stacks % 2 == 1) {

        alfa = alfa / 2;
        altura = sin(alfa) * radius;
        altura2 = sin(alfa + alfaux) * radius;
        raioaux = sqrtf((radius*radius) - (altura * altura));;
        raioaux2 = sqrtf((radius*radius) - (altura2 * altura2));

        for (int i = 0; i < slices; i++) {

            fprintf(ficheiro, "%f %f %f\n", raioaux*(sin(angulo*i)), -altura, raioaux*((cos(angulo*i))));
            fprintf(ficheiro, "%f %f %f\n", raioaux*(sin(angulo*(i + 1))), -altura,
                    raioaux*((cos(angulo*(i + 1)))));
            fprintf(ficheiro, "%f %f %f\n", raioaux*(sin(angulo*(i + 1))), altura,
                    raioaux*((cos(angulo*(i + 1)))));

            n[vertex * 3 + 0] = raioaux * sin(angulo*i) / radius;
            n[vertex * 3 + 1] = -altura / radius;
            n[vertex * 3 + 2] = raioaux * cos(angulo*i) / radius;
            t[vertex * 2 + 0] = atan2(n[vertex*3+2]/radius, n[vertex*3+0]/radius) / (2*M_PI)+0.5;
            t[vertex * 2 + 1] = 0.5-(asin(n[vertex*3+1])/M_PI);
            vertex++;

            n[vertex * 3 + 0] = raioaux * sin(angulo*(i + 1)) / radius;
            n[vertex * 3 + 1] = -altura / radius;
            n[vertex * 3 + 2] = raioaux * cos(angulo*(i + 1)) / radius;
            t[vertex * 2 + 0] = atan2(n[vertex * 3 + 2]/radius, n[vertex*3+0]/radius) / (2*M_PI)+0.5;
            t[vertex * 2 + 1] = 0.5-(asin(n[vertex*3+1])/M_PI);
            vertex++;

            n[vertex * 3 + 0] = raioaux * sin(angulo*(i+1)) / radius;
            n[vertex * 3 + 1] = altura / radius;
```

```
        n[vertex * 3 + 2] = raioaux * cos(angulo*(i+1)) / radius;
        t[vertex * 2 + 0] = atan2(n[vertex*3+2]/radius,
        n[vertex*3+0]/radius) / (2*M_PI)+0.5;
        t[vertex * 2 + 1] = 0.5-(asin(n[vertex*3+1])/M_PI);
        vertex++;

        fprintf(ficheiro, "%f %f %f\n", raioaux*
                (sin(angulo*i)), -altura, raioaux*((cos(angulo*i))));
        fprintf(ficheiro, "%f %f %f\n",
                raioaux*(sin(angulo*(i + 1))), altura, raioaux*((cos(angulo*(i + 1)))));
        fprintf(ficheiro, "%f %f %f\n",
                raioaux*(sin(angulo*(i))), altura, raioaux*((cos(angulo*(i)))));

        n[vertex * 3 + 0] = raioaux * sin(angulo*i) / radius;
        n[vertex * 3 + 1] = -altura / radius;
        n[vertex * 3 + 2] = raioaux * cos(angulo*i) / radius;
        t[vertex * 2 + 0] = atan2(n[vertex * 3 + 2]/radius, n[vertex*3+0]/radius) / (2*M_PI)+0.5;
        t[vertex * 2 + 1] = 0.5-(asin(n[vertex*3+1])/M_PI);
        vertex++;

        n[vertex * 3 + 0] = raioaux * sin(angulo*(i + 1)) / radius;
        n[vertex * 3 + 1] = altura / radius;
        n[vertex * 3 + 2] = raioaux * cos(angulo*(i + 1)) / radius;
        t[vertex * 2 + 0] = atan2(n[vertex * 3 + 2]/radius, n[vertex*3+0]/radius) / (2*M_PI)+0.5;
        t[vertex * 2 + 1] = 0.5-(asin(n[vertex*3+1])/M_PI);
        vertex++;

        n[vertex * 3 + 0] = raioaux * sin(angulo*(i)) / radius;
        n[vertex * 3 + 1] = altura / radius;
        n[vertex * 3 + 2] = raioaux * cos(angulo*(i)) / radius;
        t[vertex * 2 + 0] = atan2(n[vertex * 3 + 2]/radius, n[vertex*3+0]/radius) / (2*M_PI)+0.5;
        t[vertex * 2 + 1] = 0.5-(asin(n[vertex*3+1])/M_PI);
        vertex++;
    }

    alfa += alfaux;
    stacks--;
}

(...)
```

Capítulo 4

Codificação e Testes

4.1 Testes realizados e Resultados

O demo requisitado para esta fase corresponde a um model do sistema solar animado com texturas e iluminação. Mostram-se a seguir alguns testes feitos e os respectivos resultados obtidos, começando pelo ficheiro de configuração para o Sistema Solar já com as novas implementações:

```
<scene>
    <group>
        <translate X=0 Y=0 Z=0 />
        <models>
            Sol
            <model file="sphere.3d"  texture="sun.jpg" />
        </models>
        <group>
            Mercurio
            <rotate time=504  axisX=0 axisY=1 axisZ=0 />
            <translate X=1.5236 Y=0 Z=0 />
            <scale X=0.02439 Y=0.024439 Z=0.02439/>
            <models>
                <model file = "sphere.3d "  texture="mercury.jpg"/>
            </models>
        </group>
        <group>
            Venus
            <rotate time=502  axisX=0 axisY=1 axisZ=0 />
            <translate X=2.0472 Y=0 Z=0 />
            <scale X=0.06051 Y=0.06051 Z=0.06051/>
            <models>
                <model file = "sphere.3d"  texture="venus.jpg"/>
            </models>
        </group>
        <group>
            Terra
            <rotate time=506  axisX=0 axisY=1 axisZ=0 />
            <translate X=2.49 Y=0 Z=0 />
            <scale X=0.06372 Y=0.06372 Z=0.06372/>
            <models>
                <model file = "sphere.3d"  texture="earth.jpg"/>
            </models>
```

```
        Lua
        <rotate time=450  axisX=0 axisY=1 axisZ=0 />
        <translate X = 1.06872 Y=0 Z= 1.06872/>
        <scale X=0.418 Y=0.418 Z=0.418/>
        <models>
            <model file = "sphere.3d" texture="moon.jpg"/>
        </models>
    </group>
    <group>
        Marte
        <rotate time=510  axisX=0 axisY=1 axisZ=0 />
        <translate X=3.3188 Y=0 Z=0 />
        <scale X=0.03402 Y=0.03402 Z=0.03402/>
        <models>
            <model file = "mars.3d" />
        </models>
    </group>
    <group>
        Jupiter
        <rotate time=506  axisX=0 axisY=1 axisZ=0 />
        <translate X=4.7 Y=0 Z=0 />
        <scale X=0.68366 Y=0.68366 Z=0.68366/>
        <models>
            <model file = "sphere.3d" texture="jupiter.jpg"/>
        </models>
    </group>
    <group>
        Saturno
        <rotate time=508  axisX=0 axisY=1 axisZ=0 />
        <translate X=6.62 Y=0 Z=0 />
        <scale X=0.60268 Y=0.60268 Z=0.60268/>
        <models>
            <model file = "sphere.3d" texture="saturn.jpg"/>
        </models>
    </group>
    <group>
        Urano
        <rotate time=5016  axisX=0 axisY=1 axisZ=0 />
        <translate X=9.66 Y=0 Z=0 />
        <scale X=0.25559 Y=0.25559 Z=0.25559/>
        <models>
            <model file = "sphere.3d" texture="uranus.jpg"/>
        </models>
    </group>
    <group>
        Neptuno
        <rotate time=5014  axisX=0 axisY=1 axisZ=0 />
        <translate X=12.85 Y=0 Z=0 />
        <scale X=0.24622 Y=0.24622 Z=0.24622/>
        <models>
            <model file = "sphere.3d" texture="neptune.jpg"/>
        </models>
    </group>
</group>
```

```
</scene>
```

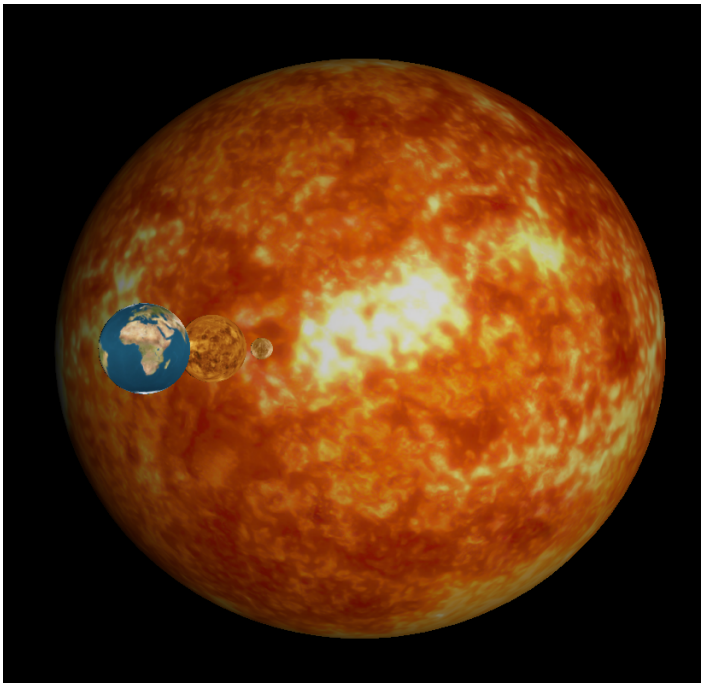

Figura 4.1: Perspetiva do Sol, já com as texturas aplicadas aos corpos celestes.

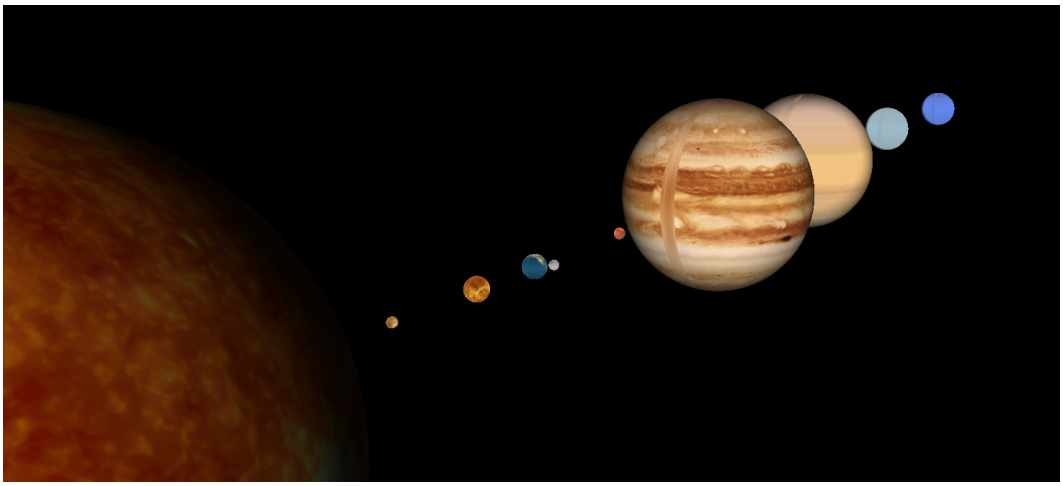

Figura 4.2: Perspetiva do sistema solar inteiro, já com as texturas aplicadas aos corpos celestes.

Capítulo 5

Conclusão

O objetivo desta quarta e última fase do trabalho prático de Computação Gráfica consistiu em atualizar o gerador de maneira a que gerasse as coordenadas de textura e as normais para cada vértice de um determinado ficheiro modelo e inserir as funcionalidades de iluminação e texturização no motor. De maneira a definir as coordenadas de textura, foi usado, essencialmente, o mapeamento UV, que consiste em projetar uma imagem 2D na superfície de um modelo 3D para mapear texturas em determinados polígonos. Juntamente com o que aprendemos na aula prática referente as texturas do cilindro, foi possível chegar ao resultado pretendido e aplicar as texturas ao nosso Sistema Solar, tornando-o mais apelativo. Achamos que podiamos ter apresentado uma última fase mais completa, visto que não nos foi possível adicionar as componentes da luz, mas achamos que atingimos o nosso objetivo nesta última fase do trabalho e assim exprimimos o nosso contentamento em o termos finalizado, devido aos problemas de implementação que foram surgindo ao longo da concepção da resolução (problemas de uso excessivo de memória, anomalias com a aplicação de texturas, etc).