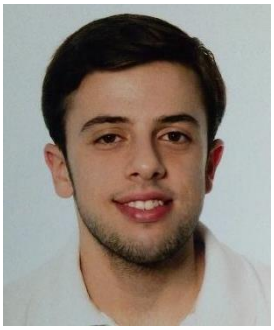
Universidade do Minho

Departamento de Informática

Normals and Texture Coordinates

Mestrado Integrado em Engenharia Informática

Computação gráfica



Jaime Leite (A80757)

Bruno Veloso (A78352)

Índice

Índice de figuras

Introdução

Partindo do enunciado que nos foi proposto, apresentaremos neste relatório o trabalho desenvolvido na quarta fase do projeto relativo à unidade curricular de *Computação Gráfica*.

Apresentaremos primeiramente as alterações efetuadas nas estruturas de armazenamento de dados, bem como no ficheiro XML, por forma a poderem ser cumpridos os requisitos propostos no enunciado.

Numa segunda fase, irá ser explicado o processo de implementação e normais e de coordenadas de textura, que auxilia no processo de iluminação do cenário e de texturas às diferentes figuras do sistema solar.

Para finalizar, é feita uma análise a todo o trabalho desenvolvido neste projeto.

Fase 4

Normais e coordenadas de textura

Apresentaremos de seguida os passos para a construção das figuras propostas no enunciado.

1. Alteração de estruturas e do ficheiro XML

Para conseguir responder às exigências desta fase foi necessário efetuar umas alterações na estrutura de dados.

De forma a ser possível guardar toda a informação relativamente às luzes, tivemos de criar uma estrutura de dados chamada “lights”. É um array de structs “Light”, uma vez que o ficheiro XML pode ter mais que uma luz. A “struct Light” guarda informação relativamente ao tipo de luz (POINT, SPOT ou DIRECTIONAL) e também guarda outros 3 parâmetros que serão usados para posicionar a luz.

```
struct Light {
    char *type;
    float posX;
    float posY;
    float posZ;

};


static struct Xml * documents = NULL;
static struct Light * lights = NULL;
```

Figura 1: struct Light

Quanto ao novo tipo que informação que uma tag “model” pode conter, foi necessário criar-se uma “struct Model”, em que para a tag “group” iremos ter um array de “struct Model”, e por sua vez, cada parte do array corresponde a uma tag “model”.

Nesta “struct Model” guardam-se informações como: as componentes de cor do modo diffuse (diffR, diffG, diffB e diffA), do modo specular (specR, specG, specB e specA), modo ambient (ambR, ambG, ambB e ambA) e o modo emissive (emiR, emiG, emiB e emiA); para além disto guarda o tipo de modo de que se trata, o nome da textura(caso haja), o nome do ficheiro que contém os triângulos a desenhar, tem também um array de “struct Normal”, em que cada “struct Normal” corresponde às normais da figura, e por fim um array de “struct PointText”, que guardará os pontos de textura do modelo.

Cada ficheiro modelo agora passará a ter numa linha 24 valores, em que destes 24: 9 representam o triângulo, seguido de 6 que representam as coordenadas textura para cada vértice do triângulo e por fim 9 que representam a normal.

```
struct Model {
    struct PointText *text;
    struct Normal *normals;
    int textura;
    char *file;
    char *texture;
    float diffR;
    float diffG;
    float diffB;
    float diffA;
    float specR;
    float specG;
    float specB;
    float specA;
    float emiR;
    float emiG;
    float emiB;
    float emiA;
    float ambR;
    float ambG;
    float ambB;
    float ambA;
};
```

Figura 2: struct Model

2. Normal

Para descobrir as normais para a esfera bastou para cada ponto tirar a multiplicação pelo raio, por exemplo: tendo ponto A(radius*cos(10)*sin(10), radius * sin(10), radius *cos(10)*cos(10)), para saber o normal unitário basta tirar o raio, como foi dito, ficando (cos(10)*sin(10), sin(10), cos(10)*cos(10)). Efetuando isto para todos os pontos existentes na esfera tem-se então as normais da esfera todas definidas. Os valores vão sendo guardados no ficheiro modelo da esfera.

```
pontoText pontoTexA;
pontoTexA.x = (((float) j) / slices);
pontoTexA.y = 0.5 + (((float)i) / stacks);

vetorNormal vecNormalA;
vecNormalA.x = cos(nivel)*sin(angulo);
vecNormalA.y = sin(nivel);
vecNormalA.z = cos(nivel)*cos(angulo);

ponto pontoA;
pontoA.x = radius * cos(nivel)*sin(angulo);
pontoA.y = radius * sin(nivel);
pontoA.z = radius * cos(nivel)*cos(angulo);

pontoText pontoTexB;
pontoTexB.x = ((((float)j) + 1) / slices) ;
pontoTexB.y = 0.5 + (((float)i) / stacks);

vetorNormal vecNormalB;
vecNormalB.x = cos(nivel)*sin(angulo2);
vecNormalB.y = sin(nivel);
vecNormalB.z = cos(nivel)*cos(angulo2);

ponto pontoB;
pontoB.x = radius * cos(nivel)*sin(angulo2);
pontoB.y = radius * sin(nivel);
pontoB.z = radius * cos(nivel)*cos(angulo2);
```

Figura 3: ponto, ponto normal e ponto textura

3. Coordenadas de textura

Apresenta-se de seguida o processo de cálculo das coordenadas de textura relativas ao plano, bem como às figuras do sistema solar(esferas).

A. Plano

Para além das figuras do sistema solar, o grupo aplicou também texturas a um plano. Sabendo que este é definido por dois triângulos, as coordenadas de textura assumem apenas os valores (0,0), (0,1), (1,0), (1,1), como se mostra de seguida.

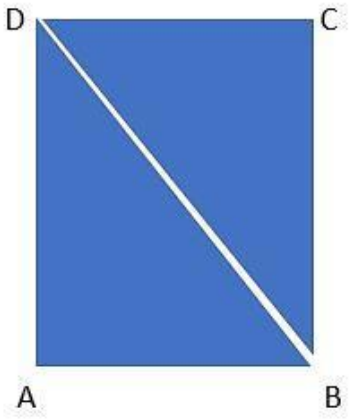


Figura 4: pontos de um plano

A corresponde à coordenada de textura (0,0)

B corresponde à coordenada de textura (1,0)

C corresponde à coordenada de textura (1,1)

D corresponde à coordenada de textura (0,1)

Tendo estes valores armazenados, o processo de aplicação da textura ao plano fica simplificado.

B. Esfera

Para definir as coordenadas de textura da esfera foi efetuado o seguinte método.

Uma vez que a forma como é desenhada a esfera é ir a uma stack (até stacks/2) e desenhar as slices todas da face superior e inferior na mesma iteração, então na parte da face superior o primeiro vértice terá coordenadas de textura de (j/slices , 0.5 + (i/stacks)), em que "j" é a slice e "i" é a stack em que se está, seguido de ((j+1)/slices , 0.5 + (i/stacks)), que representa o ponto ao seu lado direito. Se depois se pretender obter o ponto acima deste último será ((j+1)/slices , 0.5 + ((i+1)/stacks)), etc.

Quanto à parte inferior da esfera foi efetuado o mesmo método, no entanto muda um pouco a maneira de pensar, pois em vez de desenhar as slices da esquerda para a direita, são desenhadas as slices da direita para a esquerda, então o x vai ter de começar no fim (1, uma vez que o x e y vão de 0 a 1), ficando assim para o primeiro ponto a desenhar da parte inferior (1 - ((j+1) / slices), 0.5 - i/stacks), sendo o segundo ponto a desenhar (1 - (j / slices), 0.5 - i/stacks) (passou de j+1 a j pelo facto de ser da direita para a esquerda).

Tendo estes valores todos definidos, estes são guardados no ficheiro modelo da esfera, sendo possível no engine estes serem carregados para as estruturas.

4. Desenho do sistema solar

Depois de tudo armazenado corretamente nas structs, passou-se à criação de mais 2 tipos de buffers. Um dos buffers irá guardar as normais de cada modelo e o outro irá guardar as coordenadas de textura de cada modelo.

Para desenhar, primeiro é feita a verifica-se se este tem textura ou não. Se tiver, desenhamos o modelo e coloca-se a textura. Se não tiver, então passa-se a desenhar o modelo e a ativar as componentes de cor que este tenha presente.

```
if (document.modelss[k].texture != NULL) {
    glBindTexture(GL_TEXTURE_2D, document.modelss[k].textura);
    glBindBuffer(GL_ARRAY_BUFFER, buffers[0]);
    glVertexPointer(3, GL_FLOAT, 0, 0);

    glBindBuffer(GL_ARRAY_BUFFER, normals[0]);
    glNormalPointer(GL_FLOAT, 0, 0);

    glBindBuffer(GL_ARRAY_BUFFER, texturas[0]);
    glTexCoordPointer(2, GL_FLOAT, 0, 0);

    glDrawArrays(GL_TRIANGLES, 0, size * 3);
    glBindTexture(GL_TEXTURE_2D, 0);
}
```

Figura 5: desenhar figura com textura

```
else {
    glPushAttrib(GL_LIGHTING_BIT);

    if ((document.modelss[k].ambR != 0) || (document.modelss[k].ambG != 0) || (document.modelss[k].ambB != 0) ||
        (document.modelss[k].ambA != 0)) {
        GLfloat color[4] = { document.modelss[k].ambR, document.modelss[k].ambG,document.modelss[k].ambB, document.modelss[k].ambA };
        glMaterialfv(GL_FRONT, GL_AMBIENT, color);
    }
    if ((document.modelss[k].diffR != 0) || (document.modelss[k].diffG != 0) || (document.modelss[k].diffB != 0) ||
        (document.modelss[k].diffA != 0)) {
        GLfloat color[4] = { document.modelss[k].diffR, document.modelss[k].diffG,document.modelss[k].diffB, document.modelss[k].diffA };
        glMaterialfv(GL_FRONT, GL_DIFFUSE, color);
    }
    if ((document.modelss[k].specR != 0) || (document.modelss[k].specG != 0) || (document.modelss[k].specB != 0) ||
        (document.modelss[k].specA != 0)) {
        GLfloat color[4] = { document.modelss[k].specR, document.modelss[k].specG,document.modelss[k].specB, document.modelss[k].specA };
        glMaterialfv(GL_FRONT, GL_SPECULAR, color);
    }
    if ((document.modelss[k].emiR != 0) || (document.modelss[k].emiG != 0) || (document.modelss[k].emiB != 0) ||
        (document.modelss[k].emiA != 0)) {
        GLfloat color[4] = { document.modelss[k].emiR, document.modelss[k].emiG,document.modelss[k].emiB, document.modelss[k].emiA };
        glMaterialfv(GL_FRONT, GL_EMISSION, color);
    }
    glBindBuffer(GL_ARRAY_BUFFER, buffers[0]);
    glVertexPointer(3, GL_FLOAT, 0, 0);

    glBindBuffer(GL_ARRAY_BUFFER, normals[0]);
    glNormalPointer(GL_FLOAT, 0, 0);

    glDrawArrays(GL_TRIANGLES, 0, size * 3);
    glBindTexture(GL_TEXTURE_2D, 0);
    glPopAttrib();
}
```

Figura 6: desenhar figura com componentes de cor

Para ativar todas as luzes presentes na tag “lights” foi definido um ciclo na main que para cada luz existente a ativa.

```
for (int i = 0; i < size2; i++) {
    turnOnStaticLight(lights, i);
}

// enter GLUT's main cycle
glutMainLoop();

return 1;
```

Figura 7: *loop para ativar todas as luzes*

```
void turnOnStaticLight(struct Light *lights, int i) {
    int w = 0;
    if (!strcmp(lights[i].type, "POINT")) w = 1;
    if (!strcmp(lights[i].type, "DIRECTIONAL")) w = 0;
    if (!strcmp(lights[i].type, "SPOT")) w = 2;

    glEnable(GL_LIGHT0 + i);
    GLfloat light[4] = { lights[i].posX, lights[i].posY, lights[i].posZ, w };
    GLfloat colour[4] = { 1,1,1,1 };

    glLightfv(GL_LIGHT0 + i, GL_POSITION, light);
    glLightfv(GL_LIGHT0 + i, GL_AMBIENT, colour);

}
```

Figura 8: função que ativa uma luz

Apresenta-se de seguida o sistema solar, com a aplicação de todos os requisitos pretendidos para esta fase do projeto.

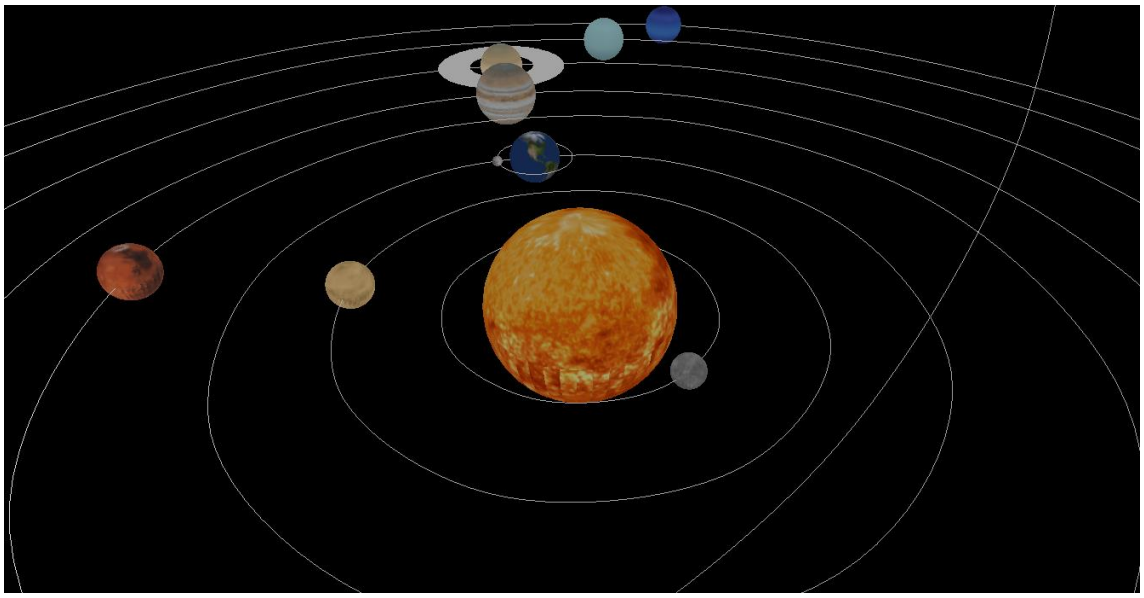


Figura 9: sistema solar

Conclusão

O grupo considera que alcançou os objetivos propostos relativamente à componente prática da unidade curricular de *Computação Gráfica*. Com a execução desta fase do projeto, a análise ao trabalho desenvolvido é bastante positiva.

Contudo existem aspetos que poderiam ter sido melhorados, como por exemplo, o facto de nesta fase poderem ter sido aplicadas as texturas nas figuras geométricas como a caixa e o cone.

Neste projeto, o parser de ficheiros XML, e respetiva definição de estruturas de dados, assumiram bastante importância, visto que foi necessário manipular pontos, nomes de ficheiros e informação sobre translações, rotações, tipos de luz...

Concluindo, apesar desta ser uma área inicialmente pouco explorada pelos elementos do grupo, este projeto consciencializou o mesmo acerca das bases sobre as quais o mundo do OpenGl assenta.