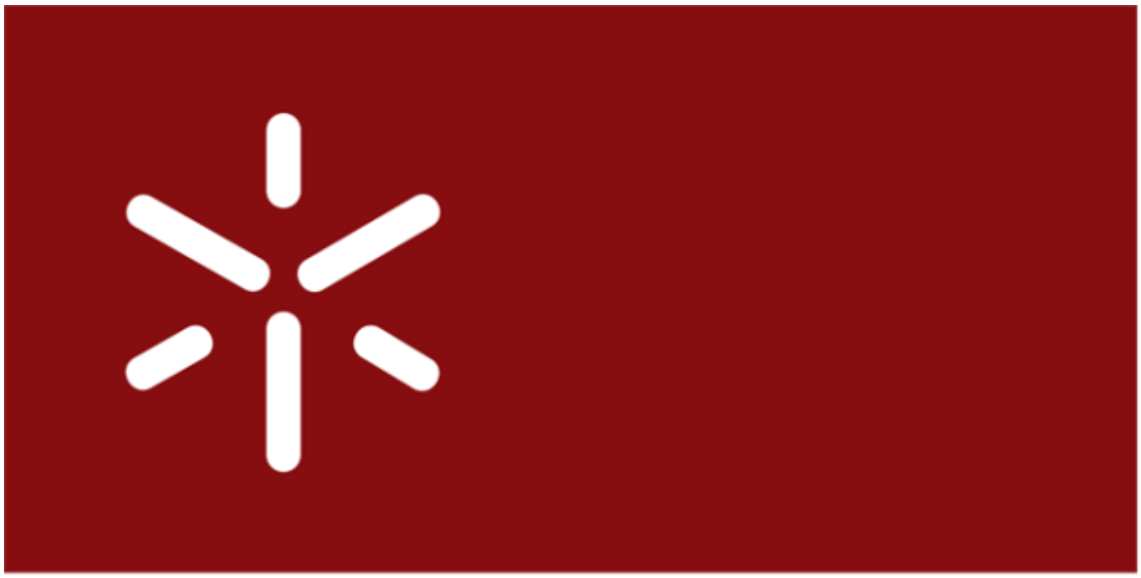
Relatório de Computação Gráfica

Fase 4- Normais e coordenadas de Textura



Universidade do Minho

2019/2020

Carlos Afonso A82529, Gonçalo Nogueira A86617, Luís Fernandes A76712

Índice

Introdução

No âmbito da cadeira de Computação Gráfica foi-nos proposto, na quarta e última fase, a continuação do trabalho realizado na fase anterior, recorrendo a novos métodos que foram lecionados durante o semestre, com o objetivo de molhar o projeto.

Nesta fase foram adicionadas texturas e iluminação à fase anterior, tendo como objetivo final a criação de um Sistema Solar animado.

Ao longo deste relatório irão ser apresentadas as alterações feitas ao projeto, relativamente à terceira fase, e as observações do grupo de trabalho

Estruturação do projeto

De forma a corresponder ao pedido nesta fase, para além de gerar os pontos referentes às formas geométricas requisitadas, também são geradas as respetivas normais e as coordenadas de texturas desses mesmos pontos.

O motor foi alterado, com alterações feitas nos métodos de parsing do ficheiro XML para agora contemplar a iluminação e dos ficheiros .3d de forma a guardar os vértices gerados num “vector” bem como as suas normais e as coordenadas das texturas, e por fim, alterações nos métodos de rendera ação e nos VBO´s de forma a implementar as texturas nos planetas.

Motor

1. Leitura do XML

Em relação ao parsing do ficheiro XML a função xmlRead foi alterada para contemplar o elemento “lights” e os seus valores e atributos.

```
void parseLights(TiXmlElement* lights)
{

    for (TiXmlElement* light = lights->FirstChildElement(); light ;light = light->NextSiblingElement()) {

        std::string lightName(toLower(light->Value()));

        if (lightName == "light") {

            xmlreadLight(light);

        }
        else {

            cout << "problema no xml com as luzes" << endl;
        }
    }
}
```

Figura 1-Parsing das luzes

```
void xmlreadLight(TiXmlElement* light)
{
    Ponto pos(0.0f, 0.0f, 1.0f);       // Posição da luz
    Ponto diff(1.0f, 1.0f, 1.0f);      // Cor difusa da luz
    Ponto spec(1.0f, 1.0f, 1.0f);      // Cor especular da luz
    Ponto ambt(0.0f, 0.0f, 0.0f);      // Cor ambiente da luz
    Ponto spotDir(0.0f, 0.0f, -1.0f); // Direção da spot light



    bool isPoint = false;
    bool isDirectional = false;
    bool isSpot = false;
```

Figura 2-Parsing das variáveis da luz

```
for (TiXmlAttribute* a = light->FirstAttribute(); a ; a = a->Next()) {



    if (!strcmp(a->Name(), "posX")) {
        pos.setX(readFloat(a->Value()));

    }
    else if (!strcmp(a->Name(), "posY")) {
        pos.setY(readFloat(a->Value()));
    }
    else if (!strcmp(a->Name(), "posZ")) {
        pos.setZ(readFloat(a->Value()));
    }
    else if (!strcmp(a->Name(), "diffR")) {
        diff.setX(readFloat(a->Value()));
    }
    else if (!strcmp(a->Name(), "diffG")) {
        diff.setY(readFloat(a->Value()));
    }
    else if (!strcmp(a->Name(), "diffB")) {
        diff.setZ(readFloat(a->Value()));
    }
    else if (!strcmp(a->Name(), "specR")) {
        spec.setX(readFloat(a->Value()));
    }
    else if (!strcmp(a->Name(), "specG")) {
        spec.setY(readFloat(a->Value()));
    }
    else if (!strcmp(a->Name(), "specB")) {
        spec.setZ(readFloat(a->Value()));
    }
```

Figura 3-Parsing das variavies de luz

```
else if (!strcmp(a->Name(), "ambtR")) {
    ambt.setX(readFloat(a->Value()));
}
else if (!strcmp(a->Name(), "ambtG")) {
    ambt.setY(readFloat(a->Value()));
}
else if (!strcmp(a->Name(), "ambtB")) {
    ambt.setZ(readFloat(a->Value()));
}
else if (!strcmp(a->Name(), "spotDirX")) {
    spotDir.setX(readFloat(a->Value()));
}
else if (!strcmp(a->Name(), "spotDirY")) {
    spotDir.setY(readFloat(a->Value()));
}
else if (!strcmp(a->Name(), "spotDirZ")) {
    spotDir.setZ(readFloat(a->Value()));
}

else if (!strcmp(a->Name(), "type")) {


    if (!strcmp(a->Value(),"POINT")) {
        isPoint = true;


    }
    else if (!strcmp(a->Value(), "DIRECTIONAL")){
        isDirectional = true;

    }
    else if (!strcmp(a->Value(), "SPOT")) {
        isSpot = true;

    }

}
```

Figura 4-Parsing das variaveis de luz

A função “parseLights” lê cada elemento “light” dentro de “lights”
A função “xmlreadlights” percorre todos os atributos do valor luz e conforme o tipo e as variáveis indicadas guarda nas suas componentes respetivas.

Para além da alteração mencionada anteriormente foi também criada forma de ler do XML para qualquer a textura presente no ficheiro que pretendemos associar a cada planeta. Esta menção é feita no atributo “model” do elemento “scenes” e guardamos assim qual o ficheiro de textura associado a cada planeta.

2. VBOs e texturas

Nesta fase além dos VBOS para os vértices, foram também criados VBOs, quando os mesmos pontos se encontravam no ficheiro 3d, para a textura e para as normais. Ao mesmo tempo, é também feito load à textura de cada conjunto de pontos de textura, para depois na classe Desenhar se proceder ao desenho dos vértices e texturas.

```
void prepareVBOs(){
    //vertices
    buffersVerts = (GLuint*) malloc( size: (sizeof(GLuint))*nVBOVert);
    //normais
    buffersNorms = (GLuint*) malloc( size: (sizeof(GLuint))*nVBONorm);
    //textures
    buffersTextsCoords  = (GLuint*) malloc( size: (sizeof(GLuint))*nVBOTex);

    //vbos dos vertices
    glGenBuffers(nVBOVert, buffersVerts);
    for(int i = 0; i < nVBOVert; ++i){

        glBindBuffer(GL_ARRAY_BUFFER,buffersVerts[i]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(float) * vertVBOs[i].size(), vertVBOs[i].data(), GL_STATIC_DRAW);

    }
```

Figura 6-Preparar VBO's

```
//vbos das normais
glGenBuffers(nVBONorm, buffersNorms);
for(int i = 0; i < nVBONorm; ++i){

    glBindBuffer(GL_ARRAY_BUFFER,buffersNorms[i]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(float) * normVBOs[i].size(), normVBOs[i].data(), GL_STATIC_DRAW);

}

//vbos das texturas

glGenBuffers(nVBOTex, buffersTextsCoords);
for(int i = 0; i < nVBOTex; ++i){

    glBindBuffer(GL_ARRAY_BUFFER,buffersTextsCoords[i]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(float) * texVBOs[i].size(), texVBOs[i].data(), GL_STATIC_DRAW);

}
```

Figura 5-Preparar VBO's(2)

3. Camara

As funcionalidades da camara foram alteradas, de forma a ser possível percorrer o espaço global de uma forma simples. A maioria das alterações foram feitas aos botões do “mouse”, sendo que o botão dois permite ampliar ou minguar a distância ao ponto para onde a camara está a olhar. O botão um permite rodar a camara em todos os eixos. De forma complementar, usamos teclas do teclado para andar no eixos do X e do Z de forma a podermos ter controlo total sobre o espaço.

Gerador

1. Normais e Texturas

Para obter as normais de uma figura geométrica, é necessário descobrir o vetor perpendicular a cada ponto que constitui a mesma. Relativamente às texturas, teremos de mapear uma imagem 2D para uma 3D, resultando no revestir da figura geométrica com o resultado deste mapeamento

1.1. Plano

Neste caso, a obtenção da normal é simples, uma vez que, como o plano está no eixo xOz, concluímos que a normal terá de ser (0,1,0), uma vez que está é a perpendicular entre estes eixos.

As coordenadas de texturas foram obtidas fazendo a relação de coordenada textura, com os vértices do triangulo.

1.2. Cubo

Para obter as várias normais do cubo, aplicamos a mesma estratégia usada no plano, mas desta vez, teremos de pensar em todas as faces, ou seja, teremos de fazer a perpendicular com os “6 planos”, ou faces, que formam o cubo:

Face Topo = (0,1,0).

Face Baixo = (0,-1,0).

Face Direita = (1,0,0).

Face Esquerda = (-1,0,0).

Face Frontal = (0,0,1).

Face Traseira = (0,0,-1).

Quanto à obtenção das coordenadas de textura, o problema pode ser melhor percebido olhando para a seguinte imagem.

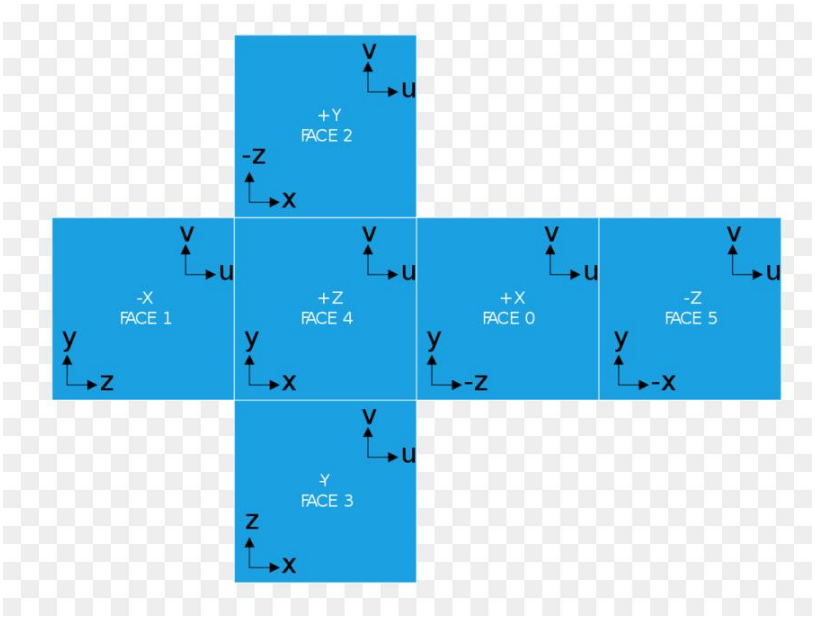


Figura 7-Cubo

Tendo esta imagem o valor máximo de coordenadas (1,1), iremos dividir o eixo do Y em três partes, dividir o eixo X em quatro partes, e de seguida, desenvolver o mapeamento das texturas.

1.3. Esfera

As normais da esfera serão calculadas usando o método seguinte. Ao longo da geração dos triângulos que formam a esfera, apos os três pontos de um triangulo serem gerados, eles são normalizados, e então, colocados no vetor de normais

```
//file << "glColor3f(1.0f, 0.0f, 0.0f);\n";
file << polarToCart(radius, alpha, beta, p1);
file << polarToCart(radius, alpha, beta + stackSize, p2);
file << polarToCart(radius, alpha + sliceSize, beta, p3);
//normais
normalize(p1);
normalize(p2);
normalize(p3);
p.x=p1[0];
p.y=p1[1];
p.z=p1[2];
normais.push_back(p);
p.x=p2[0];
p.y=p2[1];
p.z=p2[2];
normais.push_back(p);
p.x=p3[0];
p.y=p3[1];
p.z=p3[2];
normais.push_back(p);
```

Figura 8-Normais da esfera

Os pontos são normalizados usando esta função:

```
void normalize(float *a) {

    float l = sqrt(a[0]*a[0] + a[1] * a[1] + a[2] * a[2]);
    a[0] = a[0]/l;
    a[1] = a[1]/l;
    a[2] = a[2]/l;
}
```

Figura 9-Normalização de um ponto

Todo este processo ocorre cada vez que for gerado um triangulo.

Quanto às texturas, os vértices da esfera são gerados por camadas. Considerando que cada camada da esfera corresponde à camada da textura, desta forma, a cada três pontos gerados, associa-se a estes uma textura.

1.4. Cone

Por último, quanto ao cone, as normais serão geradas da seguinte forma. Quanto à base do cone, a normal será (0,-1,0).

Quanto às normais da lateral, a estratégia usada é igual à da esfera.

Quanto às texturas, as texturas, devemos primeiro olhar para a seguinte imagem.

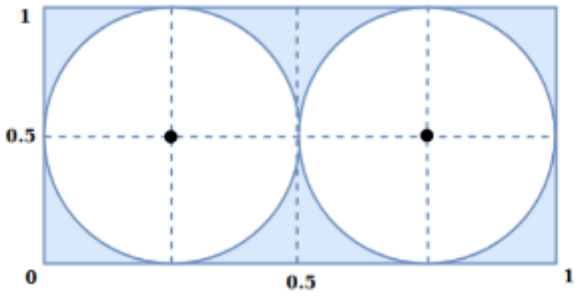


Figura 10-Cone

Assumindo que a parte da esquerda representa a base do cone, e a da direita a sua lateral. Para formar o vetor de texturas, procedemos da seguinte forma.

```
t.x=0.25f; t.y=0.5f;
texturas.push_back(t);
t.x=(0.25f+cos(alpha+sliceSize)/4.0f); t.y=(0.5f + sin(alpha+sliceSize) / 2.0f);
texturas.push_back(t);
t.x=(0.25f + cos(alpha) / 4.0f); t.y=(0.5f + sin(alpha) / 2.0f);
texturas.push_back(t);
```

Figura 11-Texturas

De forma a calcular as texturas laterais iremos recorrer aos seguintes valores, res e resplus, permitindo-nos calcular assim as coordenadas de texturas laterais.

```
float res = (float) (stacks-i) / stacks;
float resplus = (float) (stacks-(i+1)) / stacks;

t.x=(0.75f + 0.25f * cos(alpha) * resplus); t.y=(0.5f + 0.5f * sin(alpha) * resplus);
texturas.push_back(t);
t.x=(0.75f + 0.25f * cos(alpha) * res); t.y=(0.5f + 0.5f * sin(alpha) * res);
texturas.push_back(t);
t.x=(0.75f + 0.25f * cos(alpha + sliceSize) * res); t.y=(0.5f + 0.5f * sin(alpha + sliceSize) * res);
texturas.push_back(t);
t.x=(0.75f + 0.25f * cos(alpha) * resplus); t.y=(0.5f + 0.5f * sin(alpha) * resplus);
texturas.push_back(t);
t.x=(0.75f + 0.25f * cos(alpha + sliceSize) * res); t.y=(0.5f + 0.5f * sin(alpha + sliceSize) * res);
texturas.push_back(t);
t.x=(0.75f + 0.25f * cos(alpha + sliceSize) * resplus); t.y=(0.5f + 0.5f * sin(alpha + sliceSize) * resplus);
texturas.push_back(t);
```

Figura 12-Texturas laterais

Configuração ficheiro XML

```
<lights>
    <light type="POINT" posX=-1.0 posY=1.0 posZ=-1.0 ambtR=1.0 ambtG=1.0 ambtB=1.0 />
</lights>
<!--SOL-->
<group>
    <color R="1.0" G="1.0" B="0.0" />
    <translate X="0.0" Y="0.0" Z="0.0" />
    <rotate time="45" X="0" Y="1" Z="0" />
    <scale X="1.2" Y="1.2" Z="1.2" />
    <models>
        <model file="sphere.3d" textura="sun.jpg" />
    </models>
</group>
<!--MERCURIO-->
<group>
    <color R="0.3764" G="0.3764" B="0.3764" />
    <translate time="60">
        <point X="30.0" Y="0" Z="0.0"/>
        <point X="27.406363729278027" Y="0" Z="12.202099292274005"/>
        <point X="20.073918190765745" Y="0" Z="22.294344764321824"/>
        <point X="9.270509831248424" Y="0" Z="28.531695488854606"/>
        <point X="-3.1358538980296" Y="0" Z="29.835656861048204"/>
        <point X="-14.999999999999993" Y="0" Z="25.98076211353316"/>
        <point X="-24.27050983124842" Y="0" Z="17.633557568774197"/>
        <point X="-29.344428022014167" Y="0" Z="6.237350724532792"/>
        <point X="-29.34442802201417" Y="0" Z="-6.237350724532772"/>
        <point X="-24.270509831248425" Y="0" Z="-17.63355756877419"/>
        <point X="-15.000000000000014" Y="0" Z="-25.980762113533153"/>
        <point X="-3.1358538980296279" Y="0" Z="-29.8356568610482"/>
        <point X="9.270509831248416" Y="0" Z="-28.53169548885461"/>
        <point X="20.073918190765735" Y="0" Z="-22.29434476432184"/>
        <point X="27.406363729278016" Y="0" Z="-12.202099292274028"/>
    </translate>
    <rotate time="15" X="0" Y="1" Z="0"/>
    <scale X="0.150" Y="0.150" Z="0.150" />
    <models>
        <model file="sphere.3d" textura="mercury.jpg" />
    </models>
</group>
<!--VENUS-->
```

Figura 13-Excerto xml

Resultados

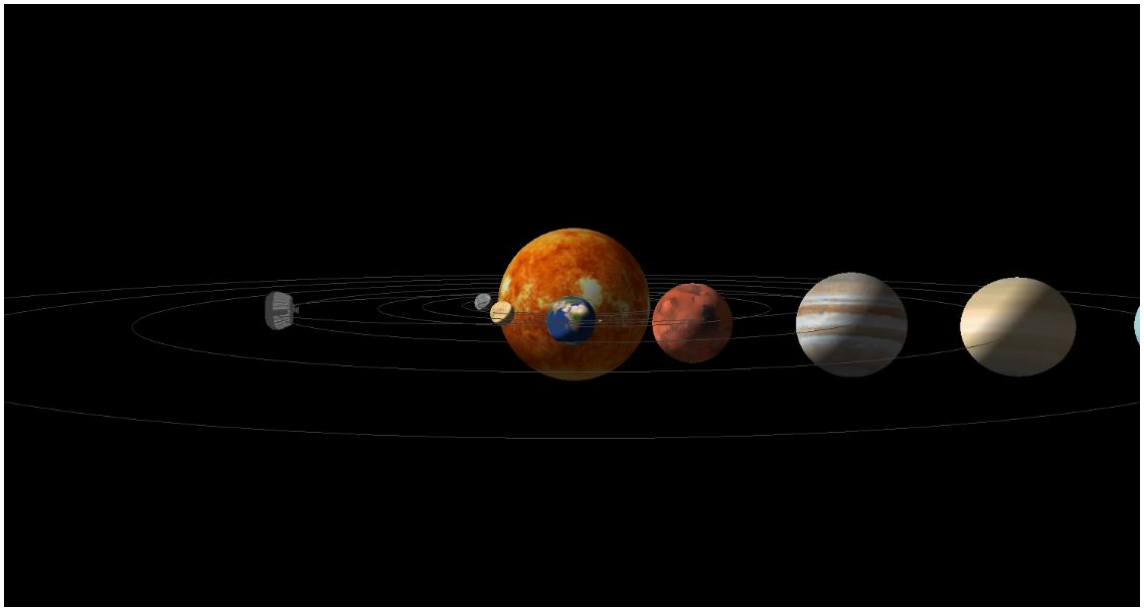


Figura 14-Resultado 1

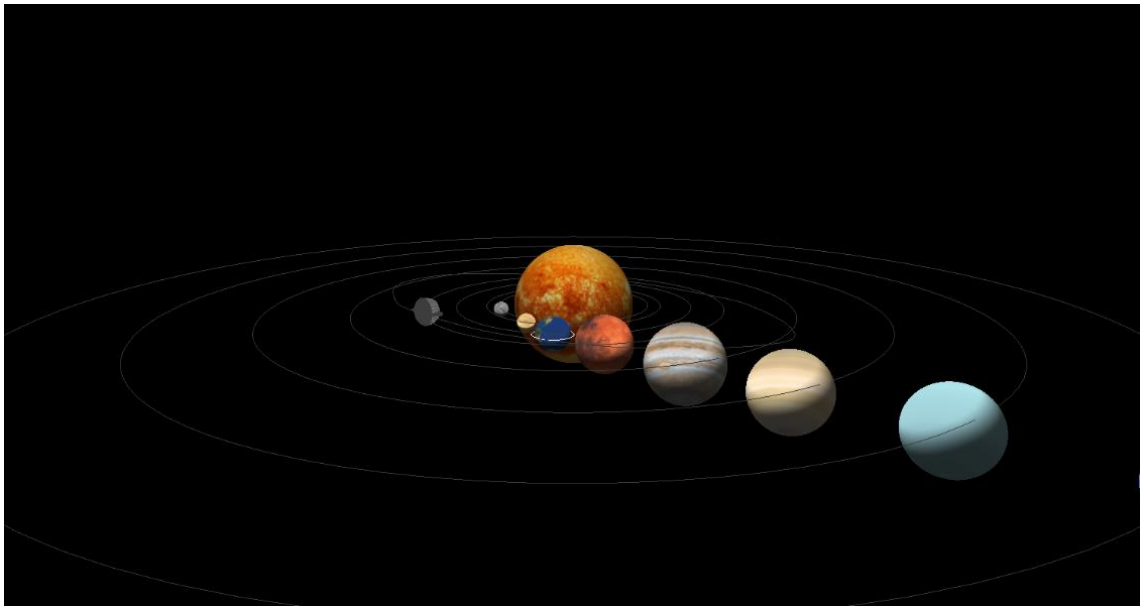


Figura 15-Resultado 2

Conclusão

Esta fase do trabalho foi importante pois conseguimos apresentar uma melhoria do Sistema Solar, usufruindo da introdução de texturas e iluminação (resultado da introdução de normais).

Após uma análise critica ao resultado deste projeto, achamos que os objetivos globais foram atingidos, pois desenvolvemos um sistema solar animado realista, tal como pedido.

Desta forma, consideramos que aprendemos bastante sofre as ferramentas que são o OpenGL e o Glut, sendo esse o grande objetivo deste trabalho, que foi concluído com sucesso.