Computação Gráfica

Relatório

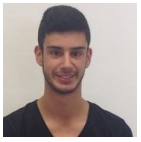
4º Fase - Normais e Coordenadas de Textura

Ana Paula Carvalho
a61855

Bruno Manuel Arieira
a70565

João Miguel Palmeira
a73864

Pedro Manuel Almeida
a74301



20 de Maio de 2018

Resumo

Foi proposto, no âmbito da Unidade Curricular de Computação Gráfica, o desenvolvimento de um mini mecanismo 3D, baseado num cenário gráfico. Para isso surge a necessidade da utilização de várias ferramentas apresentadas nas aulas práticas - tais como C++ e *OpenGL*.

Este trabalho foi dividido em quatro partes, sendo esta a quarta e última fase que, essencialmente, tem como objetivo a inclusão de texturas e iluminação ao trabalho anteriormente desenvolvido, concluindo a criação do modelo animado do Sistema Solar.

Conteúdo

Lista de Figuras

1 Introdução

Visto se tratar esta da última parte do projeto prático, é natural que se mantenham grande parte das funcionalidades criadas nas fases anteriores e, por outro lado, algumas delas sejam alteradas e melhoradas, de modo a cumprir com os requisitos necessários.

Assim, esta fase traz consigo duas novidades que irão levar a várias alterações, tanto ao nível do motor como do gerador.

Começando pelo gerador, nesta fase, este passará a conseguir, a partir de uma dada imagem, ser capaz de obter as normais e os pontos de textura para os vários vértices das primitivas geométricas anteriormente criadas.

Passando para o motor, este sofrerá algumas modificações e, ao mesmo tempo, receberá também novas funcionalidades. Os ficheiros *XML* passarão a conter as informações relativas à iluminação do cenário. De maneira que terá que ser alterado não só o *parser* responsável por ler esses mesmos ficheiros, assim como, o modo como é processada toda a informação recebida com o intuito de gerar todo o cenário pretendido.

A última modificação que o motor sofrerá, está relacionada com a preparação dos *VBOs* e das texturas durante o processo de leitura de informação proveniente do ficheiro de configuração.

Tudo isto tem como finalidade conseguirmos gerar eficazmente um modelo do Sistema Solar ainda mais realista do que o elaborado na fase anterior, uma vez que este passará a ter as texturas e a iluminação necessárias.

2 Objetivos

Como objetivos, nesta fase, e como já foi referido anteriormente, temos uma evolução da fase anterior através da criação de Normais e Coordenadas de textura. Na aplicação geradora, teremos de adicionar os seus vetores normais bem como adicionar as coordenadas de textura de cada vértice.
Por sua vez, nos modelos 3D, tanto as coordenadas de textura como as normais dos ficheiros .3d serão utilizadas para aplicar texturas e iluminar os mesmos.
Por fim, para que seja possível iluminar os modelos, serão especificadas as diferentes iluminações no ficheiro *XML*.

3 Contextualização

Para a obtenção das normais e dos pontos de textura para as diferentes figuras geométricas desenvolvidas foi necessário estudar as faces e os respetivos vértices que as constituem.
O estudo das normais passa por conseguir obter um **vetor** normal para cada vértice (perpendicular a este), que constitui a figura geométrica.

Define-se o vetor normal a um plano como sendo um vetor cujo a direção

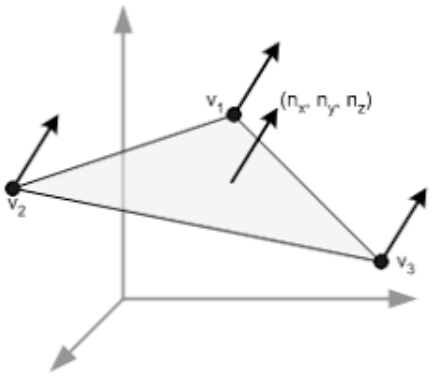


Figura 1: Exemplo de vetores normais

é ortogonal a qualquer reta pertencente a esse plano. É através deste mesmo vetor, que, juntamente com um ponto, se pode definir uma dada reta.
Quanto às **coordenadas de textura**, foi necessário estudar o mapeamento de um plano **2D (imagem) para as figuras geométricas 3D**, servindo como revestimento das mesmas. Para tal, o mapeamento é feito, por exemplo, da seguinte forma:

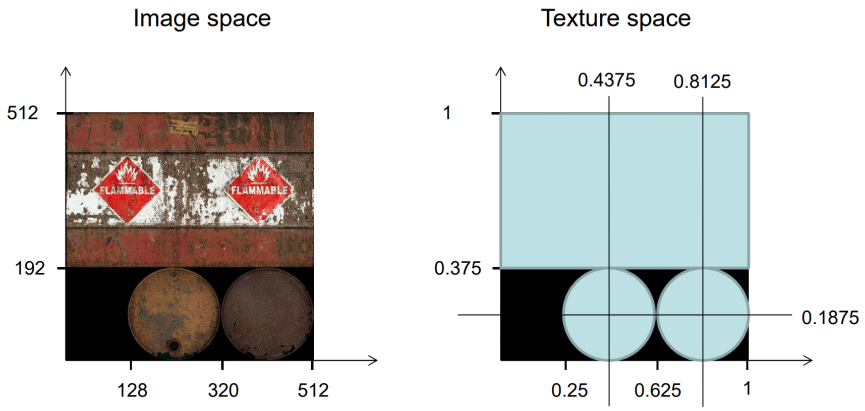


Figura 2: Espaço da imagem real e espaço da textura (mapeamento)

Como é visível, é necessário estabelecermos desde início a origem do eixo cartesiano da textura, que é limitada de **X=0 até X=1 e Y=0 até Y=1**. Estes limites correspondem aos limites da imagem real, no entanto, mapeados para valores entre **0 e 1**. Em último, a origem encontra-se no canto inferior esquerdo de todas as imagens.

3.0.1 Plano

Para obter o conjunto dos vetores normais, de cada vértice, é necessário verificar o plano cartesiano em que este plano geométrico será desenhado. Como desenhamos este no plano **x0z**, todos os vértices possuirão a mesma normal, como tal todos os vértices partilham com o normal o vetor (0,0,1).

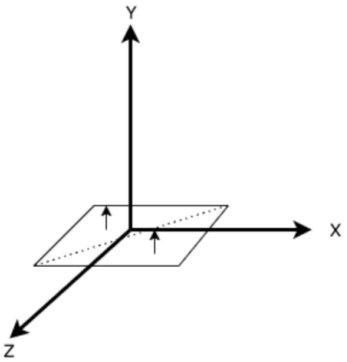


Figura 3: Referencial cartesiano, com os vetores normais do plano

O processo de obtenção dos pontos de textura é também simples no caso desta figura geométrica. Como é de esperar, o formato é o mesmo que o da **imagem 2D**, sendo apenas necessário fazer a correspondência direta de cada vértice do plano com os vértices da **imagem 2D**.

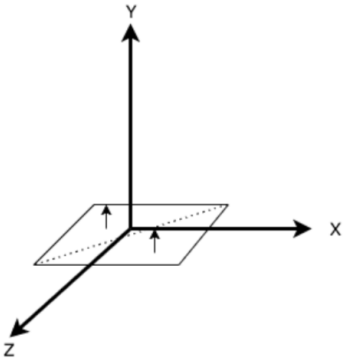


Figura 4: Referencial cartesiano, com o mapeamento dos vértice com a imagem 2D

3.0.2 Paralelepípedo

Da mesma forma que o plano, para obter o conjunto de vetores normais para cada vértice, é indispensável verificar o plano cartesiano de cada face do paralelepípedo. Facilmente reconhecemos os correspondentes vetores normais a cada uma das faces do mesmo:

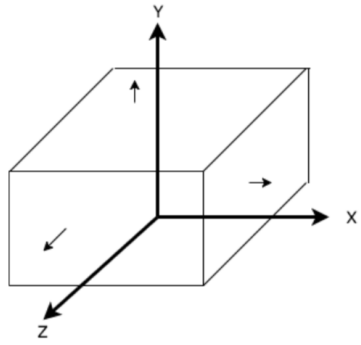


Figura 5: Referencial cartesiano, com os vetores normais de 3 faces

1. **Topo** - vetor (0,0,1)
2. **Base** - vetor (0,0,-1)
3. **Face Frontal** - vetor (0,0,1)
4. **Face Traseira** - vetor (0,0,-1)
5. **Face Direita** - vetor (1,0,0
6. **Face Esquerda** - vetor (-1,0,0)

O cálculo dos pontos de textura passa por obter a posição da imagem a que corresponde a face em questão e iterar no mesmo sentido que o paralelepípedo é desenhado, atribuindo a cada vértice o ponto de textura correspondente. Para uma melhor compreensão, segue um exemplo:

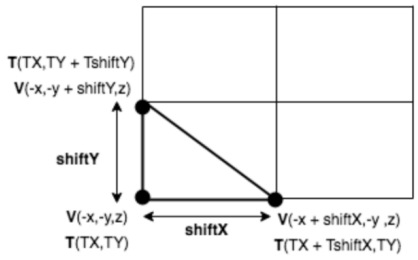


Figura 6: Processo de cálculo dos pontos de textura para a face frontal

Os valores de **TX e TY** correspondem à posição onde se encontra a face frontal, dentro da **imagem 2D**, isto é, o **ponto (TX, TY)** corresponde ao canto inferior esquerdo da face frontal, na **imagem 2D**.

3.0.3 Esfera

Para obter o conjunto dos vetores normais da esfera é necessário ter em conta a orientação da origem do referencial até ao ponto em questão. Uma vez que o processo de desenho da esfera já nos fornece essa informação, isto é, como durante o processo de desenho é referenciada pela origem do referencial, quando definimos as coordenadas de um vértice, as mesmas podem corresponder à do vetor normal.
Assim interessa-nos apenas saber a direção a partir da origem até ao ponto e não a distância até este. Para tal, obtemos o seguinte:

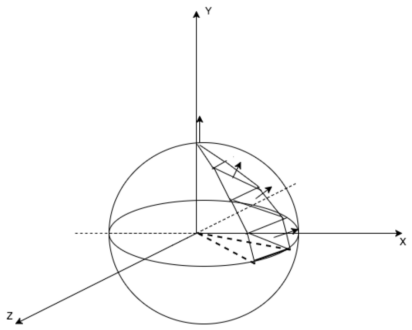


Figura 7: Referencial cartesiano, com os vetores normais de uma fatia da esfera

Para um dado vértice V(x,y,z), o vetor normal respetivo é dado por:

$$\mathbf{N(sin(PH),y/raio,cos(PH))\ -\ PH} \tag{1}$$

Corresponde ao passo (desvio) horizontal, que se faz para iterar a circunferência que compõe o centro da esfera.
Em contrapartida, para obter o ponto de textura de um dado vértice o grupo recorreu à seguinte solução. Para um dado vértice V(x,y,z), o ponto de textura T(t1,t2) correspondente é dado por:

$$\mathbf{t1\ =\ (atan2(x,z)+)/2} \tag{2}$$

$$\mathbf{t2 = 1 \quad ((-y/raio)+1)/2} \tag{3}$$

Desta forma, conseguimos mapear os pontos de textura de uma imagem 2D normal, para os vértices de uma esfera, revestindo a mesma com a imagem.

3.0.4 Cone

O cone pode ser dividido em 2 partes distintas para o cálculo dos vetores normais: normais da base e do corpo. Desta forma, a normal de cada vértice é obtida da seguinte forma:

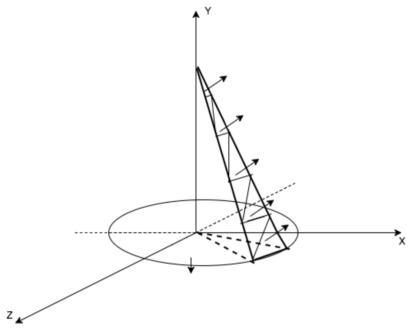


Figura 8: Referencial cartesiano, com os vetores normais de uma fatia do cone

1. **Base** - vetor (0,0,-1)
2. **Corpo** - vetor (sin(alfa), CC/stack, cos(alfa)) - **CC** corresponde ao comprimento do corpo, ou seja, a distância da ponta do cone, até a um ponto da circunferência da base. Como é de esperar o valor da coordenada **Y** do vetor normal, é dada pelo **CC** a dividir pelas *stacks* definidas. As coordenadas **X e Z**, dão orientação restante ao vetor normal (seno e cosseno para andar no sentido da circunferência). O valor de **alfa** é a amplitude a que se encontra o vértice.

3.0.5 Cilindro

De uma forma muito idêntica à da figura anterior, o cilindro pode ser dividido em 3 partes diferentes para o cálculo dos vetores normais: normais da base, do topo e do corpo. Desta forma, a normal de cada vértice é obtida da seguinte forma:

1. **Topo** - vetor (0,0,1)
2. **Base** - vetor (0,0,-1)
3. **Corpo** - vetor (sin(alfa), 0, cos(alfa)) - sendo alfa a amplitude a que se encontra o vértice.

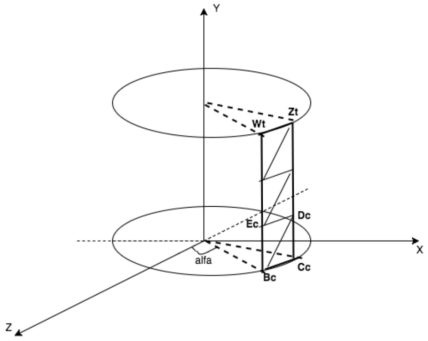


Figura 9: Processo de cálculo dos pontos de textura para a face frontal

Quanto aos pontos de textura, primeiro temos que estabelecer um padrão das imagens 2D a serem mapeadas. Por isso, retomamos ao primeiro exemplo, onde o padrão estabelecido é:

Figura 10: Exemplo de padrão de imagens 2D para o cilindro

Com isto, para mapear os pontos da imagem para os vértice do cilindro, estabelecemos a seguinte correspondência:

1. **Topo** - Circunferência da esquerda na imagem 2D. Para obter os pontos apenas é preciso posicionar no centro da circunferência, e obter os pontos da circunferência, como por exemplo, **T(0.4375 + 0.1875*sin(alfa), 0.1875 + 0.1875 * cos(alfa))**, para um dado alfa;

2. **Base** - Circunferência da direita na imagem 2D. Para obter os pontos recorre-se ao mesmo processo do Topo, isto é, **T(0.8125 + 0.1875*sin(alfa), 0.1875 + 0.1875 * cos(alfa))**, para um dado **alfa**;

3. **Corpo** - Plano de cima. Para obter os pontos é apenas preciso integrar o plano, aos pedaços, como por exemplo, **T(1/slice, 0.375 + (0.625/stack))**;

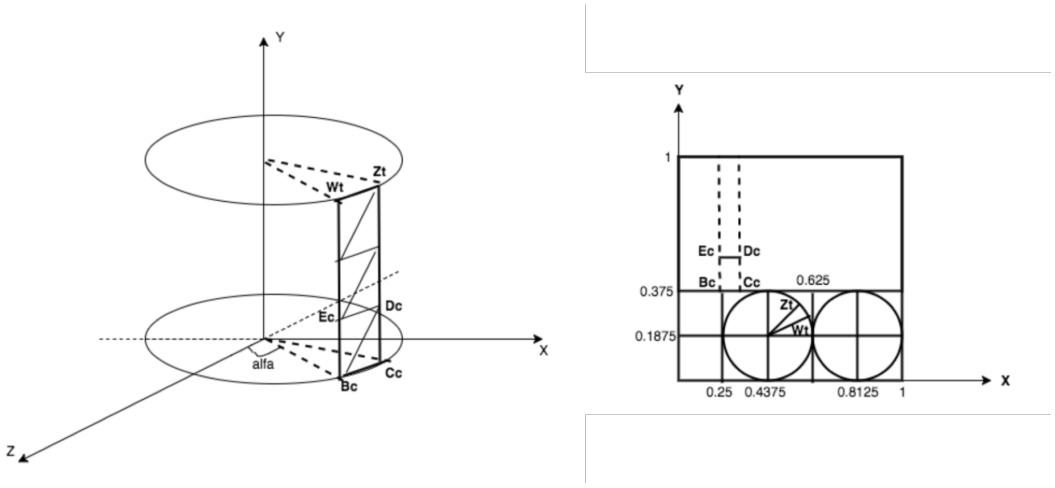


Figura 11: Vértices do Cilindro e pontos de textura da imagem 2D a mapear

4 Estruturas

Relativamente as estruturas criadas anteriormente e, de maneira a que seja possível implementar as novas funcionalidades pretendidas (iluminação e texturas), foi necessário adicionar novos elementos a estruturas elaboradas nas fases anteriores e construir novas, nomeadamente a classe para a iluminação. De seguida iremos apresentar as alterações efetuadas bem como as novidades inseridas.

4.1 Modelo

A classe Modelo foi uma das classes alteradas relativamente a fase anterior. Nesta fase foram adicionadas todas as variáveis necessárias para que seja possível definir a iluminação de cada objeto bem como as texturas de cada um.

```
class Modelo{
public:
    string ficheiro;
    int pontos;
    float ambR,ambG,ambB;
    float diffR,diffG,diffB;
    float specR,specG,specB;
    float emiR,emiG,emiB;
    string textura;
    unsigned int t,width,height,texID;
    unsigned char *texData;
    GLuint buffers[3];
};
```

4.2 Luz

Esta é uma classe nova que nos permite introduzir as luzes, isto é, construirmos esta estrutura de modo a podermos definir uma luz através do posicionamento das suas coordenadas e o seu tipo.

```
class Light{
public:
```

```
    float posX,posY,posZ,tipo;
};
```

4.3 Figura

Em relação a classe Figura apenas introduzimos uma lista de luzes.

```
class Figura{
public:
    list<Grupo> grupos;
    list<Light> luzes;
    void load(const char* pFilename);
};
```

Já na função que realiza o *load* da classe Figura foi adicionado um *parsing* da informação recebida para as luzes tal como se verifica abaixo.

```
void Figura::load(const char* pFilename){
    TiXmlDocument doc(pFilename);
    if (!doc.LoadFile()) return;
    TiXmlNode* nRoot = TiXmlHandle(doc.RootElement()).ToNode();

    TiXmlNode* lNode = nRoot->FirstChild("luzes");
    if(lNode){
        TiXmlElement* lElem = lNode->ToElement();
        TiXmlNode* testNode = lElem->FirstChild("luz");
        if(testNode)
            for(TiXmlElement* luzElem = lElem->FirstChild("luz")->ToElement() ; luzElem;
            luzElem = luzElem->NextSiblingElement()){
                Light l;
                const char *tipo = luzElem->Attribute("tipo"), *posX =
                luzElem->Attribute("posX"), *posY = luzElem->Attribute("posY"),
                *posZ = luzElem->Attribute("posZ");
                if(tipo) l.tipo = atof(tipo); else l.tipo = 0;
                if(posX) l.posX = atof(posX); else l.posX = 0;
                if(posY) l.posY = atof(posY); else l.posY = 0;
                if(posZ) l.posZ = atof(posZ); else l.posZ = 0;
```

```
                this->luzes.push_back(l);
            }
        }
        GRload(nRoot,&this->grupos);
    }
```

5 Gerador

Em relação á fase anterior deste trabalho, foram necessárias algumas alterações relativamente ao formato dos ficheiros **.3d**.

- O número inteiro n que se apresenta na primeira linha (número de pontos) alteramos para n = *número de triângulos * 3*. É de reparar que foi retirada uma multiplicação por 3, pois este número será necessário para a implementação das texturas.
- Em vez de 3 valores, por cada linha, passamos a ter 8;

  –3 valores para a normal;

  –3 valores para os vértices;

  –2 valores correspondentes á textura;

  De seguida, demonstramos um excerto do ficheiro .3d gerado para o Sol.

```
2400
 0.000000 -0.987688 0.156434 0.000000 -0.987688 0.156434 0.000000 -0.050000
-0.000000 -1.000000 -0.000000 -0.000000 -1.000000 -0.000000 0.000000 -0.000000
-0.000000 -1.000000 -0.000000 -0.000000 -1.000000 -0.000000 0.050000 -0.000000
 0.000000 -0.987688 0.156434 0.000000 -0.987688 0.156434 0.000000 -0.050000
-0.000000 -1.000000 -0.000000 -0.000000 -1.000000 -0.000000 0.050000 -0.000000
 0.048341 -0.987688 0.148778 0.048341 -0.987688 0.148778 0.050000 -0.050000
 0.048341 -0.987688 0.148778 0.048341 -0.987688 0.148778 0.050000 -0.050000
-0.000000 -1.000000 -0.000000 -0.000000 -1.000000 -0.000000 0.050000 -0.000000
-0.000000 -1.000000 -0.000000 -0.000000 -1.000000 -0.000000 0.100000 -0.000000
 0.048341 -0.987688 0.148778 0.048341 -0.987688 0.148778 0.050000 -0.050000
-0.000000 -1.000000 -0.000000 -0.000000 -1.000000 -0.000000 0.100000 -0.000000
 0.091950 -0.987688 0.126558 0.091950 -0.987688 0.126558 0.100000 -0.050000
 0.091950 -0.987688 0.126558 0.091950 -0.987688 0.126558 0.100000 -0.050000
-0.000000 -1.000000 -0.000000 -0.000000 -1.000000 -0.000000 0.100000 -0.000000
-0.000000 -1.000000 -0.000000 -0.000000 -1.000000 -0.000000 0.150000 -0.000000
 0.091950 -0.987688 0.126558 0.091950 -0.987688 0.126558 0.100000 -0.050000
-0.000000 -1.000000 -0.000000 -0.000000 -1.000000 -0.000000 0.150000 -0.000000
```

```
0.126558 -0.987688 0.091950 0.126558 -0.987688 0.091950 0.150000 -0.050000
```

5.1 Cálculo da normal

Por forma a calcular o vetor normal ao triângulo dado, houve a necessidade de implementar a função *normal*, que recebe como parãmetro três vértices (double) e devolve um array com três valores. Esta função apenas é invocada para o desenho do cone e do patch, visto que para as outras figuras o calculo é direto.

```
double *normal(double ponto1[3], double ponto2[3], double ponto3[3]){
    double *normal = NULL;
    normal = (double*) malloc(3*sizeof(double));

    double vetor1[3] = { ponto2[0]-ponto1[0], ponto2[1]-ponto1[1], ponto2[2]-ponto1[2] };
    double vetor2[3] = { ponto3[0]-ponto1[0], ponto3[1]-ponto1[1], ponto3[2]-ponto1[2] };

    double vetor[3];
    vetor[0] = vetor1[1]*vetor2[2] - vetor1[2]*vetor2[1];
    vetor[1] = vetor1[2]*vetor2[0] - vetor1[0]*vetor2[2];
    vetor[2] = vetor1[0]*vetor2[1] - vetor1[1]*vetor2[0];

    double vn = sqrt(vetor[0]*vetor[0] + vetor[1]*vetor[1] + vetor[2]*vetor[2]);

    normal[0] = vetor[0]/vn;
    normal[1] = vetor[1]/vn;
    normal[2] = vetor[2]/vn;

    return normal;
}
```

- **Plano:** Em relação ao calculo dos vetores do plano é efetuado de forma imediata, devido á superfície ser plana, encontrando-se no plano XZ, o que indica que não há inclinação. Relativamente á textura, é mapeada uma vez para cada lado.

- **Paralelepípedo:** Neste caso o calculo dos vetores são feitos de forma imediata, pois são faces planas e sem inclinação, e a sua textura é mapeada uma vez para cada lado.

- **Esfera:** Como é constituido por coordenadas esféricas, os vetores normais são iguais aos vértices, sendo que para normalizá-los basta não multiplicar por *radius*. Para a textura, para mapear os vértices num intervalo de 0 a 1, basta dividir por pi e 2pi.

- **Cone:** Usa a função *normal* para calcular os vetores normais. As coordenadas relativas á textura são calculadas como no caso da esfera.

- **Patch:** Relativamente aos vetores normais, são calculados da mesma forma que no caso do cone. Em relação ás coordenadas da textura, são calculadas uma vez para cada patch.

6 Motor

6.1 XML

Nos respetivos ficheiros XML acrescentamos as seguintes tags/atributos:

- ```
  <luzes>
      <luz tipo=t posX=pX posY=pY posZ=pZ />
  </luzes>
  ```

- ```
  ambR=aR;
  ```

- ```
  ambG=aG;
  ```

- ```
  diffR=dR;
  ```

- ```
  diffG=dG;
  ```

- ```
  diffB=dB;
  ```

- ```
  textura="./SistemaSolar/'respetivoPlaneta'.jpg"
  ```

6.2 Parsing

Relativamente á última fase, como já foi referido em cima, foi adicionado (conforme os requisitos enunciados para esta última etapa):

- A classe Light, retrata um ponto de luz a ser inserido, sendo que consequentemente, na classe Figura é adicionado uma lista de pontos de luzes, correspondentes da classe Light. A classe grupo não contém referências algumas da classe Ligth pois são logo definidas no ínicio da Cena.
- Na classe Modelo o tamanho do buffer é aumentado para 3, porque terão de ser desenhados mais três VBO's por cada modelo, um que retrata as
  ```

texturas, outro que faz referência aos vértices e outro para as normais. O carregamento das respetivas figuras é feito através da função *GRload*. Ainda nesta classe, foram acrescentados os atributos imprescindíveis á iluminação e textura de um dado objeto.

A seguir é apresentado um excerto de código da função *GRload*, com as modificações e adições necessárias. Aqui é onde é carregada a imagem e toda a informação relativa aos vetores normais e às coordenadas relativas á textura.

```
//modelos
TiXmlNode* mNode = gNode->FirstChild("modelos");
if(mNode){
    TiXmlElement* mElem = mNode->ToElement();
    TiXmlNode* testNode = mElem->FirstChild("modelo");
    if(testNode){
        for(TiXmlElement* modElem = mElem->FirstChild("modelo")->ToElement() ; modElem;
        modElem = modElem->NextSiblingElement()){
            Modelo m;
            const char *ficheiro = modElem->Attribute("ficheiro");
            if(ficheiro) m.ficheiro = ficheiro;

                const char *ambR = modElem->Attribute("ambR"), *ambG =
                modElem->Attribute("ambG"), *ambB = modElem->Attribute("ambB");
                if(ambR) m.ambR = atof(ambR); else m.ambR = 0;
                if(ambG) m.ambG = atof(ambG); else m.ambG = 0;
                if(ambB) m.ambB = atof(ambB); else m.ambB = 0;

                const char *diffR = modElem->Attribute("diffR"), *diffG =
                modElem->Attribute("diffG"), *diffB = modElem->Attribute("diffB");
                if(diffR) m.diffR = atof(diffR); else m.diffR = 0;
                if(diffG) m.diffG = atof(diffG); else m.diffG = 0;
                if(diffB) m.diffB = atof(diffB); else m.diffB = 0;

                const char *specR = modElem->Attribute("specR"), *specG =
                modElem->Attribute("specG"), *specB = modElem->Attribute("specB");
                if(specR) m.specR = atof(specR); else m.specR = 0;
                if(specG) m.specG = atof(specG); else m.specG = 0;
                if(specB) m.specB = atof(specB); else m.specB = 0;
```

```
const char *emiR = modElem->Attribute("emiR"), *emiG =
modElem->Attribute("emiG"), *emiB = modElem->Attribute("emiB");
if(emiR) m.emiR = atof(emiR); else m.emiR = 0;
if(emiG) m.emiG = atof(emiG); else m.emiG = 0;
if(emiB) m.emiB = atof(emiB); else m.emiB = 0;

const char *textura = modElem->Attribute("textura");
if(textura) m.textura = textura;

//parse iluminação e textura
        //carregamento da imagem
ilInit();
ilGenImages(1,&m.t);
ilBindImage(m.t);
ilLoadImage((ILstring)m.textura.c_str());
ilConvertImage(IL_RGBA,IL_UNSIGNED_BYTE);

m.width = ilGetInteger(IL_IMAGE_WIDTH);
m.height = ilGetInteger(IL_IMAGE_HEIGHT);
m.texData = ilGetData();

glGenTextures(1,&m.texID);
glBindTexture(GL_TEXTURE_2D,m.texID);
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,m.width,m.height,0,GL_RGBA,
GL_UNSIGNED_BYTE,m.texData);


fstream f;

//f.open(m.ficheiro,ios::in);
f.open(m.ficheiro.c_str());

if(f.is_open()){
```

```
string line;
int pontos=0,j=0,k=0,l=0;
float *vertexB = NULL, *normalB = NULL, *textureB=NULL;

if(getline(f,line)){
    sscanf(line.c_str(),"%d\n",&pontos);
    m.pontos = pontos;
}

vertexB = (float*) malloc(3*pontos*sizeof(float));
normalB = (float*) malloc(3*pontos*sizeof(float));
textureB = (float*) malloc(2*pontos*sizeof(float));

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

while(getline(f,line)){
    float x,y,z,xIl,yIl,zIl,xText,yText;
    sscanf(line.c_str(),"%f %f %f %f %f %f %f %f\n",&x,&y,&z,&xIl,&yIl,
    &zIl,&xText,&yText);
    vertexB[j++] = x; vertexB[j++] = y; vertexB[j++] = z;
    normalB[k++] = xIl; normalB[k++] = yIl; normalB[k++] = zIl;
    textureB[l++] = xText; textureB[l++] = yText;
}

glGenBuffers(3,m.buffers);

glBindBuffer(GL_ARRAY_BUFFER,m.buffers[0]);
glBufferData(GL_ARRAY_BUFFER,3*m.pontos*sizeof(float),vertexB,
GL_STATIC_DRAW);

glBindBuffer(GL_ARRAY_BUFFER,m.buffers[1]);
glBufferData(GL_ARRAY_BUFFER,3*m.pontos*sizeof(float),normalB,
GL_STATIC_DRAW);

glBindBuffer(GL_ARRAY_BUFFER,m.buffers[2]);
```

```
                    glBufferData(GL_ARRAY_BUFFER,2*m.pontos*sizeof(float),textureB,
                    GL_STATIC_DRAW);


                    free(vertexB);
                    free(normalB);
                    free(textureB);

                    f.close();
                }

                g.modelos.push_back(m);
            }
        }
    }
```

6.3 Rendering

Com estas novas implementações, é necessário na main ativar as respetivas iluminações assim como as texturas:

```
int main(int argc, char **argv) {
            (...)
    glEnable(GL_LIGHT0);
    glEnable(GL_LIGHTING);
    glEnable(GL_TEXTURE_2D);
            (...)
}
```

Á posteriori, também se torna necessário ativar todas as luzes carregadas:

```
void renderScene(void) {
        (...)
    for(list<Light>::iterator itl = figura.luzes.begin();
        itl != figura.luzes.end();
        itl++){
            GLfloat pos[4] = { itl->posX, itl->posY, itl->posZ, itl->tipo };
            glLightfv(GL_LIGHT0,GL_POSITION,pos);
```

```
            glLightf(GL_LIGHT0,GL_CONSTANT_ATTENUATION,0.4);
    }
        (...)
}
```

Relativamente ao desenho de cada grupo, primeiro as transformações são colocadas por ordem e só depois as respetivas funções de iluminação e textura são ativas.

```
    void renderGrupo(list<Grupo>::iterator g){
glPushMatrix();

    if(!strcmp(g->tipo.c_str(),"orbita")){
        glRotatef(g->rotacao.angulo, g->rotacao.eixox,
        g->rotacao.eixoy, g->rotacao.eixoz);
        glTranslatef(g->translacao.coordx, g->translacao.coordy, g->translacao.coordz);
    }
        else{

        if(g->translacao.tempo == -1) glTranslatef(g->translacao.coordx,
        g->translacao.coordy, g->translacao.coordz);
        else getTranslation(g->translacao.tempo,g->translacao.pontos);

        if(g->rotacao.tempo == -1) glRotatef(g->rotacao.angulo, g->rotacao.eixox,
        g->rotacao.eixoy, g->rotacao.eixoz);
        else getRotation(g->rotacao.tempo,g->rotacao.eixox,g->rotacao.eixoy,
        g->rotacao.eixoz);
    }

    glScalef(g->escala.escala_x, g->escala.escala_y, g->escala.escala_z);
    glColor3f(g->cor.r,g->cor.g,g->cor.b);

    glDisable(GL_LIGHTING);
    drawOrbit(g->orbita.raioX,g->orbita.raioZ,g->orbita.nControl);
    glEnable(GL_LIGHTING);


    for(list<Modelo>::iterator itm = g->modelos.begin();
```

```
    itm != g->modelos.end(); itm++){
        GLfloat amb[3] = {itm->ambR,itm->ambG,itm->ambB}; glLightfv(GL_LIGHT0,GL_AMBIENT,
        amb);
        GLfloat diff[3] = {itm->diffR,itm->diffG,itm->diffB}; glLightfv(GL_LIGHT0,
        GL_DIFFUSE,diff);
        GLfloat spec[3] = {itm->specR,itm->specG,itm->specB}; glMaterialfv(GL_FRONT,
        GL_SPECULAR,spec);
        glLightfv(GL_LIGHT0,GL_SPECULAR,spec); glMaterialf(GL_FRONT,GL_SHININESS,120);
        GLfloat emi[3] = {itm->emiR,itm->emiG,itm->emiB}; glMaterialfv(GL_FRONT,
        GL_EMISSION,emi); glLightfv(GL_LIGHT0,GL_EMISSION,emi);

        glBindTexture(GL_TEXTURE_2D,itm->texID);

        glBindBuffer(GL_ARRAY_BUFFER,itm->buffers[0]);
        glVertexPointer(3,GL_FLOAT,0,0);
        glBindBuffer(GL_ARRAY_BUFFER,itm->buffers[1]);
        glNormalPointer(GL_FLOAT,0,0);
        glBindBuffer(GL_ARRAY_BUFFER,itm->buffers[2]);
        glTexCoordPointer(2,GL_FLOAT,0,0);

        glDrawArrays(GL_TRIANGLES,0,itm->pontos);
        glBindTexture(GL_TEXTURE_2D,0);
    }


    for(list<Grupo>::iterator itg = g->grupos.begin();
    itg != g->grupos.end();
    itg++)
    renderGrupo(itg);

    glPopMatrix();
}
```

7 Resultados

Incorporou-se, ao modelo alcançado na fase antecedente, as componentes de textura e iluminação. Nesta secção demonstra-se então os resultados finais desta adição.

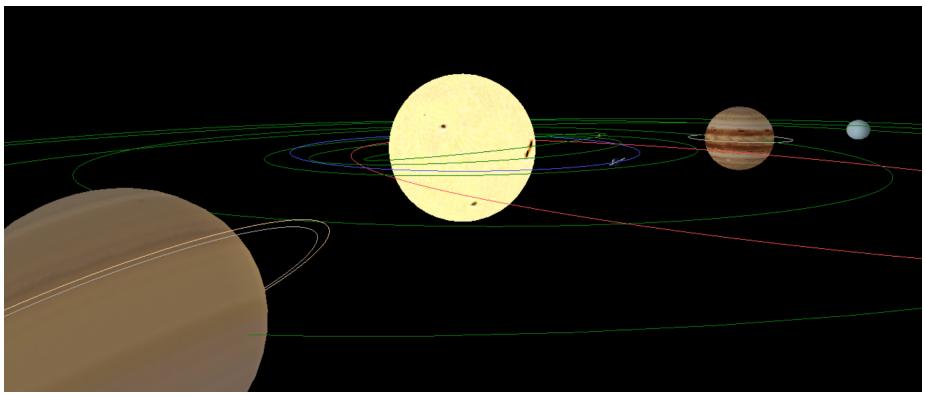


Figura 12: Perspetiva lateral do modelo

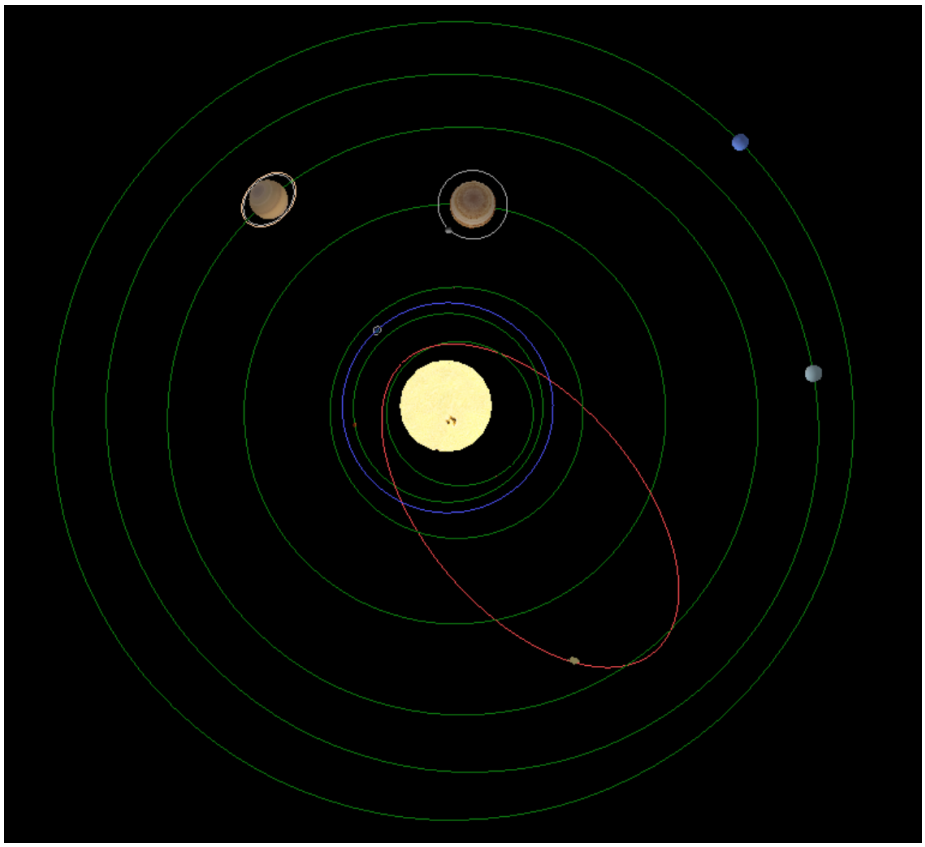


Figura 13: Perspetiva superior do modelo

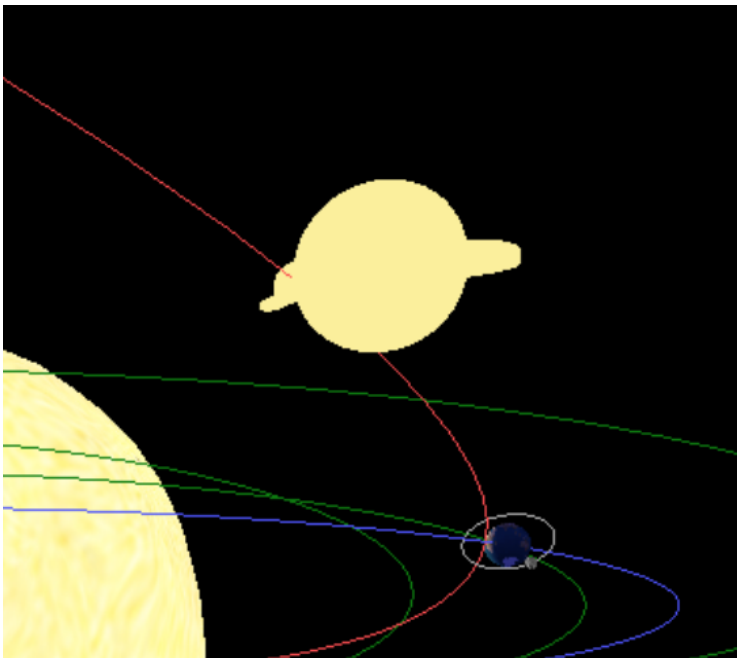


Figura 14: Pormenor - Bule

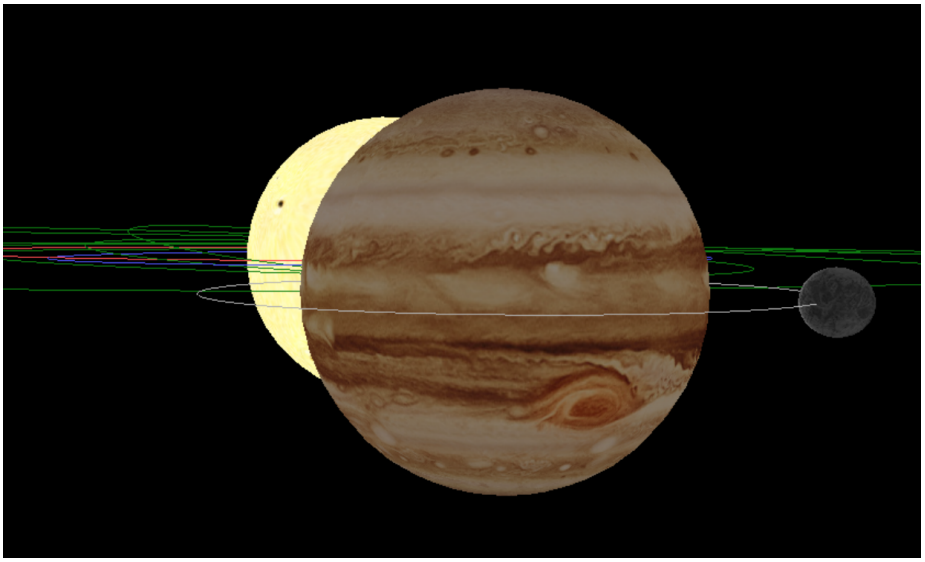


Figura 15: Pormenor - Júpiter

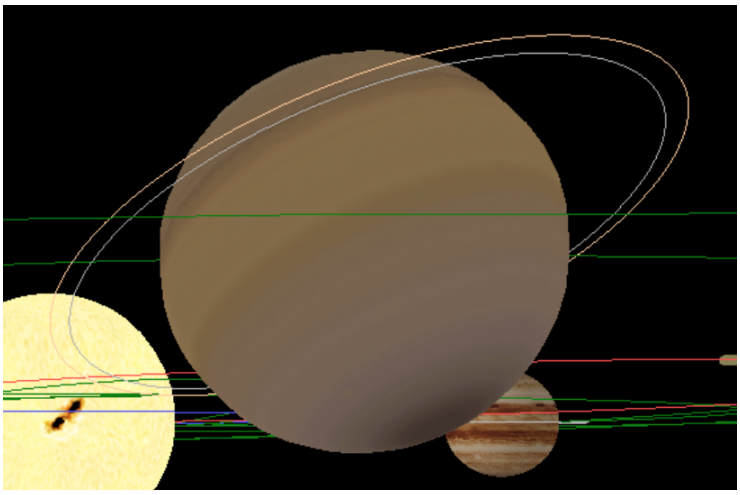


Figura 16: Pormenor - Saturno

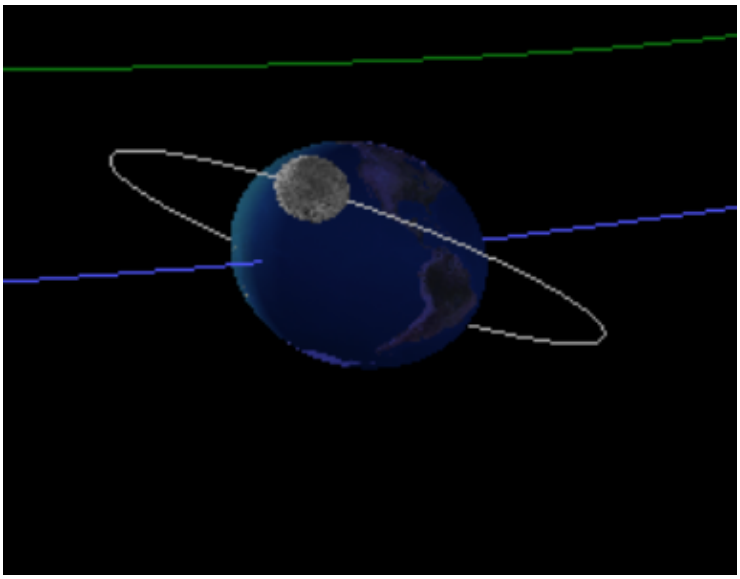


Figura 17: Pormenor - Terra

8 Extras

8.1 Anel

–**Vetores Normais:** Sao usadas coordenadas polares, sendo que os vetores são iguais aos respetivos vértices. Assim para normaliza-los apenas basta não multiplicar por *radius*.
–**Coordenadas da textura:** De forma a mapear os vértices em intervalos correspondentes de 0 a 1, é necessário dividir por 2pi.

```
void drawAnel(float in_r , float out_r ,int slices, char* filename){
        FILE* f;
        char* aux = (char*)malloc(sizeof(char)*64);
        strcpy(aux , filename);
        strcat(aux , ".3d");
        f = fopen(aux,"w");
        int pontos = 4*slices*3;
        float doisPi = 2*M_PI;
        float slice = doisPi/slices;

        if (f!=NULL) {
            fprintf(f, "%d\n", pontos);

            for (int i=0; i < slices; i++) {
                fprintf(f, "%f %f %f %f %f %f %f %f\n", sin(i*slice)*in_r, 0.0,
                        cos(i*slice)*in_r, 0.0, 1.0, 0.0, 0.0, (i*slice)/doisPi);
                fprintf(f,"%f %f %f %f %f %f %f %f\n",sin(i*slice)*out_r, 0.0,
                        cos(i*slice)*out_r, 0.0, 1.0, 0.0, -1.0, (i*slice)/doisPi);
                fprintf(f,"%f %f %f %f %f %f %f %f\n",sin((i+1)*slice)*in_r, 0.0,
                        cos((i+1)*slice)*in_r, 0.0, 1.0, 0.0, 0.0,
                        ((i+1)*slice)/doisPi);
                fprintf(f,"%f %f %f %f %f %f %f %f\n",sin(i*slice)*out_r, 0.0,
                        cos(i*slice)*out_r, 0.0, 1.0, 0.0, -1.0, (i*slice)/doisPi);
                fprintf(f,"%f %f %f %f %f %f %f %f\n",sin((i+1)*slice)*out_r, 0.0,
                        cos((i+1)*slice)*out_r, 0.0, 1.0, 0.0, -1.0,
                        ((i+1)*slice)/doisPi);
                fprintf(f,"%f %f %f %f %f %f %f %f\n",sin((i+1)*slice)*in_r, 0.0,
```

```
                cos((i+1)*slice)*in_r, 0.0, 1.0, 0.0, 0.0,
                ((i+1)*slice)/doisPi);
        fprintf(f,"%f %f %f %f %f %f %f %f\n",sin(i*slice)*out_r, 0.0,
                cos(i*slice)*out_r, 0.0, -1.0, 0.0, -1.0, (i*slice)/doisPi);
        fprintf(f,"%f %f %f %f %f %f %f %f\n",sin(i*slice)*in_r, 0.0,
                cos(i*slice)*in_r, 0.0, -1.0, 0.0, 0.0, (i*slice)/doisPi);
        fprintf(f,"%f %f %f %f %f %f %f %f\n",sin((i+1)*slice)*in_r, 0.0,
                cos((i+1)*slice)*in_r, 0.0, -1.0, 0.0, 0.0,
                ((i+1)*slice)/doisPi);
        fprintf(f,"%f %f %f %f %f %f %f %f\n",sin((i+1)*slice)*out_r, 0.0,
                cos((i+1)*slice)*out_r, 0.0, -1.0, 0.0, -1.0,
                ((i+1)*slice)/doisPi);
        fprintf(f,"%f %f %f %f %f %f %f %f\n",sin(i*slice)*out_r, 0.0,
                cos(i*slice)*out_r, 0.0, -1.0, 0.0, -1.0, (i*slice)/doisPi);
        fprintf(f,"%f %f %f %f %f %f %f %f\n",sin((i+1)*slice)*in_r, 0.0,
                cos((i+1)*slice)*in_r, 0.0, -1.0, 0.0, 0.0,
                ((i+1)*slice)/doisPi);
    }
  }
}
```

8.2 Cilíndro

–**Vetores Normais:** Cálculo é feito de forma imediata.
–**Coordenadas da textura:** Cálculo é feito de forma imediata.

```
void drawCilindro(float radius , float height , int slices , char*filename){
    FILE* f ;
    int pontos = 4*slices*3;
    char* aux = (char*)malloc(sizeof(char)*64) ;
    strcpy(aux , filename) ;
    strcat(aux , ".3d") ;

    f = fopen(aux,"w") ;

    float doisPi = 2*M_PI ;
```

```
float slice = doisPi/slices;

if (f!=NULL){
fprintf(f,"%d\n" , pontos) ;
        for(int i=0; i < slices; i++){
    fprintf(f,"%f %f %f %f %f %f %f %f\n",0.0, -height/2, 0.0, 0.0, -1.0, 0.0,
    0.5, 0.5);
    fprintf(f,"%f %f %f %f %f %f %f %f\n",sin((i+1)*slice)*radius, -height/2,
    cos((i+1)*slice)*radius, 0.0, -1.0, 0.0, ((i+1)*slice)/doisPi, 0.0);
    fprintf(f,"%f %f %f %f %f %f %f %f\n",sin(i*slice)*radius, -height/2,
    cos(i*slice)*radius, 0.0, -1.0, 0.0, (i*slice)/doisPi, 0.0);

    fprintf(f,"%f %f %f %f %f %f %f %f\n",0.0, height/2, 0.0, 0.0, 1.0, 0.0,
    0.5, 0.5);
    fprintf(f,"%f %f %f %f %f %f %f %f\n",sin(i*slice)*radius, height/2,
    cos(i*slice)*radius, 0.0, 1.0, 0.0, (i*slice)/doisPi, 0.0);
    fprintf(f,"%f %f %f %f %f %f %f %f\n",sin((i+1)*slice)*radius, height/2,
    cos((i+1)*slice)*radius, 0.0, 1.0, 0.0, ((i+1)*slice)/doisPi, 0.0);

    fprintf(f,"%f %f %f %f %f %f %f %f\n",sin(i*slice)*radius, height/2,
    cos(i*slice)*radius, sin(i*slice), 0.0, cos(i*slice), (i*slice)/doisPi, -1.0);
    fprintf(f,"%f %f %f %f %f %f %f %f\n",sin(i*slice)*radius, -height/2,
    cos(i*slice)*radius, sin(i*slice), 0.0, cos(i*slice), (i*slice)/doisPi, 0.0);
    fprintf(f,"%f %f %f %f %f %f %f %f\n",sin((i+1)*slice)*radius, -height/2,
    cos((i+1)*slice)*radius, sin((i+1)*slice), 0.0, cos((i+1)*slice),
    ((i+1)*slice)/doisPi, 0.0);

    fprintf(f,"%f %f %f %f %f %f %f %f\n",sin((i+1)*slice)*radius, height/2,
    cos((i+1)*slice)*radius, sin((i+1)*slice), 0.0, cos((i+1)*slice),
    ((i+1)*slice)/doisPi, -1.0);
    fprintf(f,"%f %f %f %f %f %f %f %f\n",sin(i*slice)*radius, height/2,
    cos(i*slice)*radius, sin(i*slice), 0.0, cos(i*slice), (i*slice)/doisPi, -1.0);
    fprintf(f,"%f %f %f %f %f %f %f %f\n",sin((i+1)*slice)*radius, -height/2,
    cos((i+1)*slice)*radius, sin((i+1)*slice), 0.0, cos((i+1)*slice),
    ((i+1)*slice)/doisPi, 0.0);
}
}
```

```
        fclose(f) ;
}
```

9 Conclusão

Terminando, por fim, esta quarta e última fase do projeto prático no âmbito de Computação Gráfica, o grupo conseguiu ultrapassar, com sucesso, a tarefa proposta e aplicar os conhecimentos lecionados nestas últimas aulas.

A matéria que assinala o término desta Unidade Curricular remete-se à implementação de vetores normais, iluminação e texturas ao modelo do Sistema Solar desenvolvido até ao momento. Considerámos que estes conceitos foram bem aplicados no nosso projeto e congratulámo-nos do resultado final obtido.

Reconhecemos, claramente, a importância do desenvolvimento deste projeto na assimilação dos conteúdos desta área de estudos e, com esta construção fase-a-fase ao longo do semestre, alcançámos uma melhor competência e agilidade na abordagem destes.

Em suma, o grupo sente-se realizado com a sua prestação e na resolução das diversas dificuldades que foram surgindo.