Universidade do Minho
Escola de Engenharia

Normais e Coordenadas de Textura

Mestrado Integrado em Engenharia Informática

Computação Gráfica
2º Semestre\2016-2017

A70938 Diogo Meira Neves
A70676 Marcos de Morais Luís
A71625 Nelson Arieira Parente
A70951 Pedro Miguel Lopes Pereira

21 de Maio de 2017
Braga

Conteúdo

Capítulo 1

Introdução

Sendo esta a quarta e última fase do projeto de Computação Gráfica esta fase irá dar como terminado o projeto contendo todo o trabalho elaborado ao longo do semestre.

O grupo deparou-se com a dificuldade da instalação do Devil no ambiente MacOS tendo este problema sido resolvido mudando a área de trabalho para uma distribuição Linux.

Capítulo 2

Objetivos

Como objetivos, nesta fase, 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 luzes no ficheiro XML.

Capítulo 3

Contextualização

3.1 Vetor Normal

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

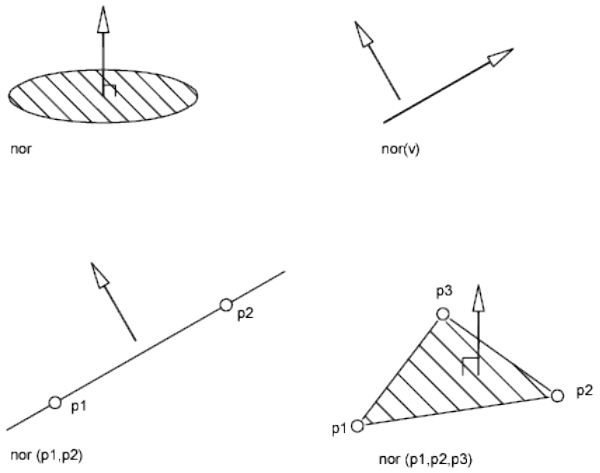

Figura 3.1: Exemplo vetores normais

3.2 Coordenadas de Textura

Sendo uma textura uma imagem com componentes RGB e alpha, o seu mapeamento consiste na aplicação de uma imagem sobre as faces de um objeto 3D, isto é, este processo requer que uma imagem em 2D seja associada à face do objeto. Para aplicação da textura é preciso criar uma relação entre os vértices da textura e os vértices de um polígono sobre os quais se deseja mapear a textura escolhida. Imaginemos que temos uma imagem quadrada em que os vértices são designados por A,B,C,D e um cubo em que os vértices de um face são designados por A1,B2,C3,D4. Com este processo de mapeamento iremos encaixar o ponto A com o A1, o ponto B com o B1 e assim sucessivamente.

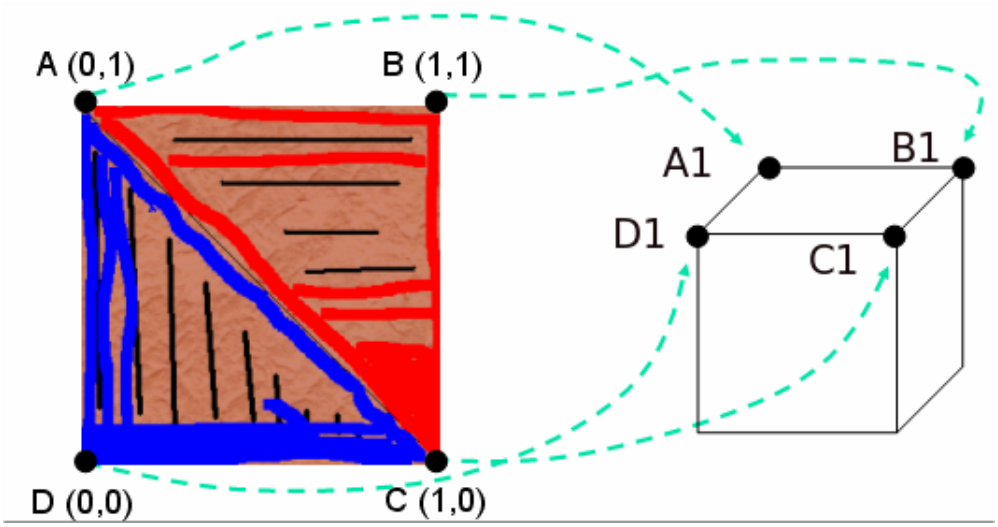

Figura 3.2: Exemplo de mapeamento

Capítulo 4

Gerador

Em relação à ultima fase, o formato de ficheiros 3D foi alterado.

- O numero inteiro 'n' da primeira linha passou a ser *n = número de triângulos * 3*. Repare-se que foi retirada uma multiplicação por 3, devido ao facto de este número ser necessário para as texturas;
- Por cada linha, passamos a ter 8 valores por linha em detrimento de 3;
  - 3 valores para os vértices;
  - 3 valores para a normal;
  - 2 valores para a textura;

Mostramos em baixo um excerto do ficheiro .3d gerado para a Terra.

```
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
```

4.1 Cálculo da normal

De modo a calcular o vetor normal ao triângulo dado, implementámos a função 'normal' que recebe três vértices (double) e devolve um array com três valores. Esta função apenas é invocada no desenho do cone e do patch, uma vez que para as outras figuras o cálculo é imediato.

```
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**: O cálculo dos vetores para o plano é realizado de forma imediata, pois a superficie é plana, e como tal, encontra-se no plano XZ, sem "inclinação". A textura é mapeada uma vez para cada lado.
- **Paralelepípedo**: Tal como o plano, o seu cálculo dos vetores normais é imediato pois as faces são planas e sem "inclinação". A textura é mapeada uma vez para cada face.
- **Esfera**: Por sua vez, a esfera, como é desenhada com coordenadas esféricas, os seus vetores normais são iguais aos vértices. De forma a normaliza-los, simplesmente basta não multiplica-lo pelo radius. Para as suas coordenadas de textura, de modo a mapear os vertices num intervalo de zero a um, é necessário dividir por pi e 2pi.
- **Cone**: Já o cone utiliza a função normal para o cálculo automático dos seu vetores normais. As coordenadas de textura são cálculadas como no caso da esfera.
- **Patch**: O patch, tal como o cone, também utiliza a função normal para calcular os vetores normais. As coordenadas de textura são calculadas uma vez para cada patch.

Capítulo 5

Motor

5.1 XML

No ficheiro XML adicionamos as seguintes tags/atributos:

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

- `ambR=aR`
- `ambG=aG`
- `diffR=dR`
- `diffG=dG`
- `ambR=aR`
- `textura="./SistemaSolar/planeta.jpg" />`

5.2 Parsing

Relativamente à fase anterior deste trabalho foi adicionado:

- A classe Light, que representa um ponto de luz a ser adicionado, sendo que desta forma, a classe Figura passa a ter uma lista de pontos de luzes. De reparar também que a class Grupo nao têm Light's porque estas são definidas no ínicio da Cena.;
- O tamanho do buffer na classe Modelo foi aumentado para 3 pois terão de ter de ser desenhados mais 3 VBO's por modelo, um para os vértices, um para as normais e um para as texturas. O carregamento das figuras é feito na fase de parsing.
- Novamente, na classe Modelo, foram adicionados os atributos necessários à iluminação e textura de um objeto.

```
class Light
{
public:
    float posX,posY,posZ,tipo;
};
```

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

```
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];
};
```

Posteriormente, é carregada a imagem e a informação relativa aos vetores normais e às coordenadas de textura.

```
void loadGrupos(TiXmlNode *currentNode, list<Grupo> *grupos)
{
    for(TiXmlNode* gNode = currentNode->FirstChild("grupo") ;
                        gNode; gNode = gNode->NextSiblingElement())
    {
        Grupo g;

        Translacao t;
        TiXmlNode* tNode = gNode->FirstChild("translacao");
        if(tNode)
        {
            //faz parse dos pontos se tiver atribuido tempo

            if(t.tempo == -1)
            {//parse x,y,z}
        else {
            for(pontos)
                {
                    Ponto ponto;
                    //parse
                    t.pontos.push_back(ponto);
                }
            }
            else t.tempo = -1;
            }
        }
        else { t.tempo = -1; t.x = 0; t.y = 0; t.z = 0; }
        g.translacao = t;

        //rotacao
        Rotacao r;
        TiXmlNode* rNode = gNode->FirstChild("rotacao");
        if(rNode)
        { //parse }
        else { r.angulo = 0; r.eixox = 0; r.eixoy = 0; r.eixoz = 0;
                                                    r.tempo = -1; }
```

```
        g.rotacao = r;

        //escala
        Escala e;
        TiXmlNode* eNode = gNode->FirstChild("escala");
        if(eNode)
        { //parse }
        else { e.x = 1; e.y = 1; e.z = 1; }
        g.escala = e;

        Cor c;
        TiXmlNode* cNode = gNode->FirstChild("cor");
        if(cNode)
        {
            //parse
        }
        else { c.r = 1; c.g = 1; c.b = 1; }
        g.cor = c;

        Orbita o;
        TiXmlNode* oNode = gNode->FirstChild("orbita");
        if(oNode)
        {
            //parse
        }
        else { o.raioX = 0; o.raioZ = 0; o.nControl = 4; g.tipo = "planeta"; }
        g.orbita = o;

        for(modelos){

                    Modelo m;

//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.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(o ficheiro tem linhas)
                {
                    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);
        }
        ...
}
```

5.3 Rendering

Com estas novas alterações, é necessário ativar as iluminações e as texturas...

```
int main( int argc, char **argv){
    ...

    glEnable(GL_LIGHT0);
    glEnable(GL_LIGHTING);
    glEnable(GL_TEXTURE_2D);

    ...
}
```

...sendo que, posteriormente, é 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);
    }

...
}
```

Inicialmente, no desenho de cada grupo, as transformações são colocadas por ordem, sendo que, depois, as funções de txtura e iluminação são ativadas.

```
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.x, g->translacao.y, g->translacao.z);
    }
    else
    {

        if(g->translacao.tempo == -1)
            glTranslatef(g->translacao.x, g->translacao.y, g->translacao.z);
        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.x, g->escala.y, g->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();
}
```

Capítulo 6

Sistema Solar

Relativamente à fase anterior foram adicionadas as componentes de textura e iluminação. Colocamos em baixo exemplos deste upgrade.

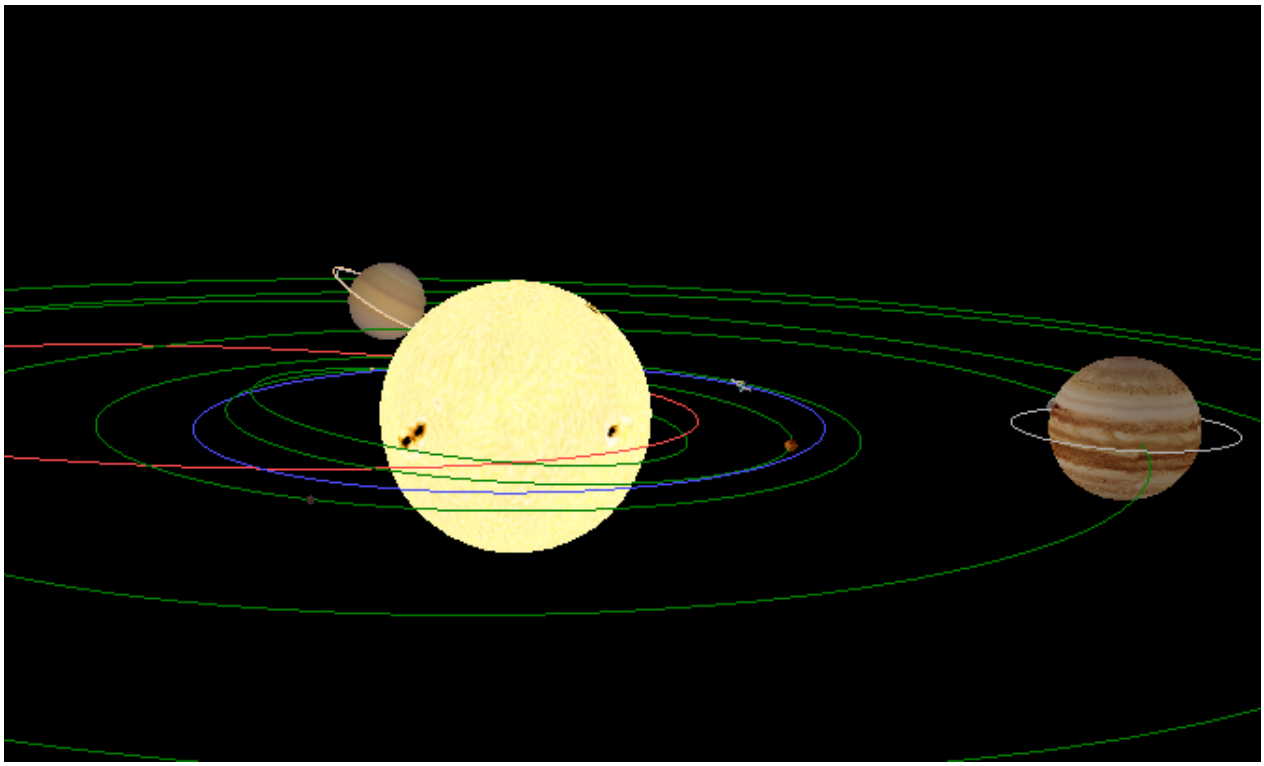

Figura 6.1: Sistema Solar - Horizontal

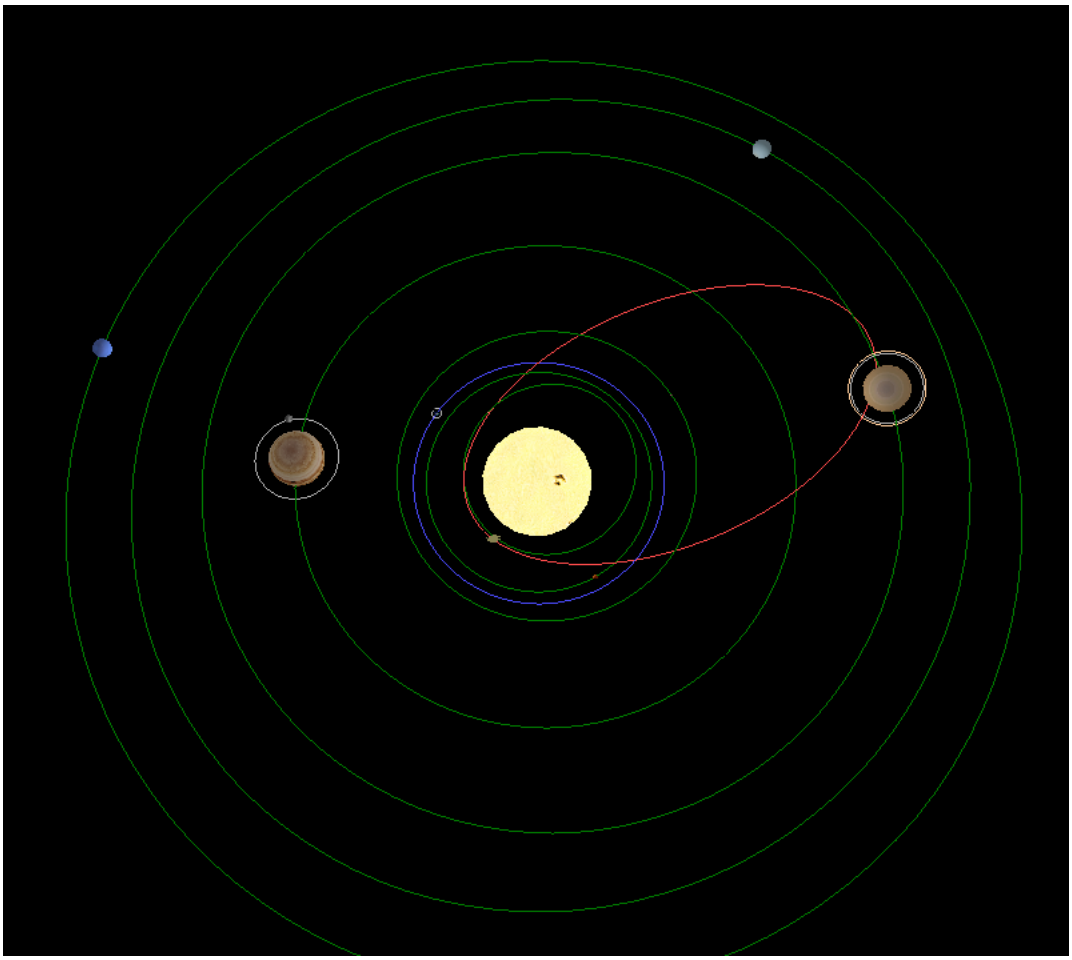

Figura 6.2: Sistema Solar - Vertical

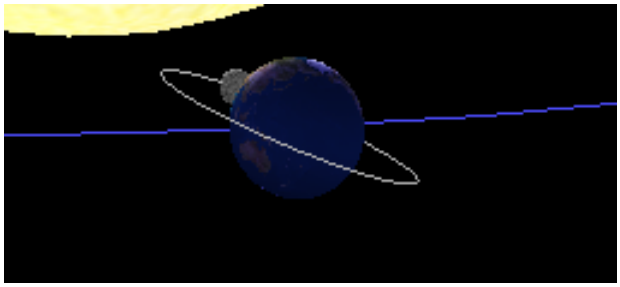

Figura 6.3: Planeta Terra

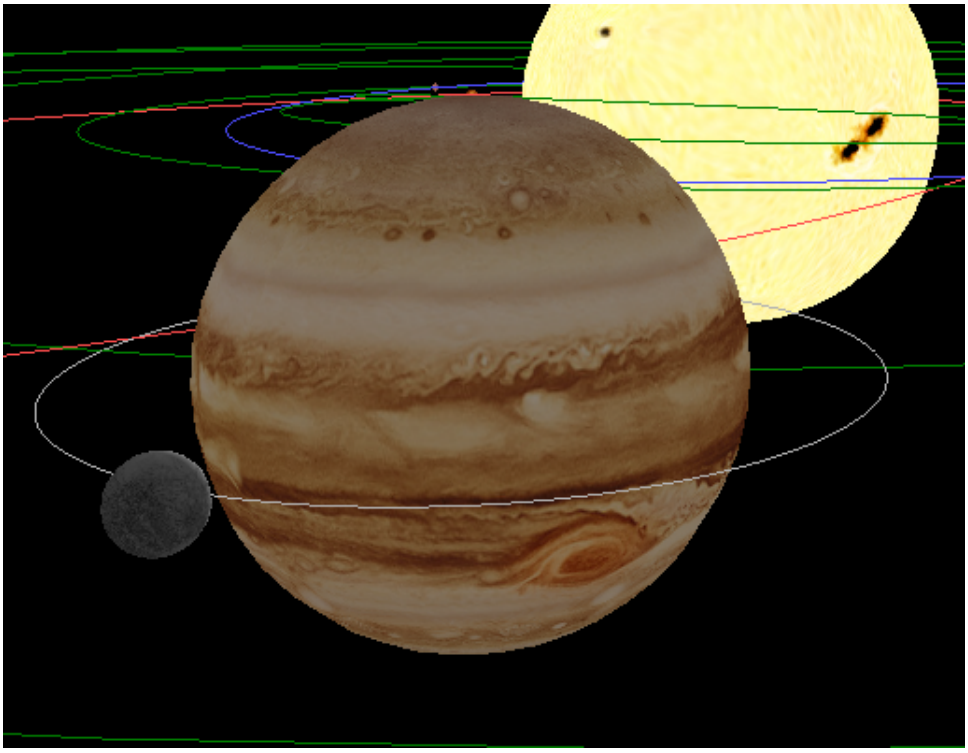

Figura 6.4: Planete Jupiter

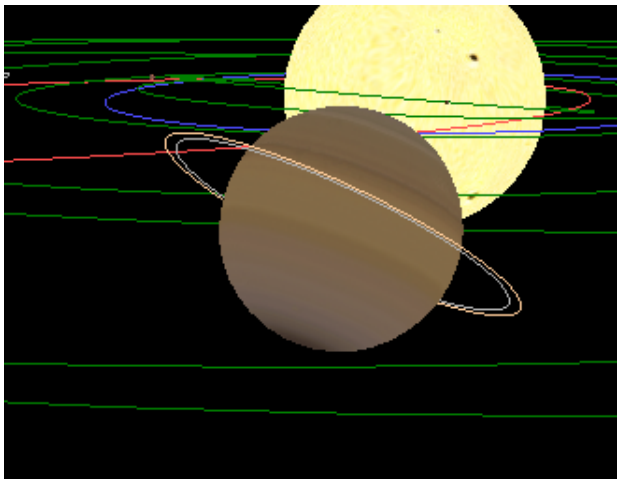

Figura 6.5: Planete Saturno

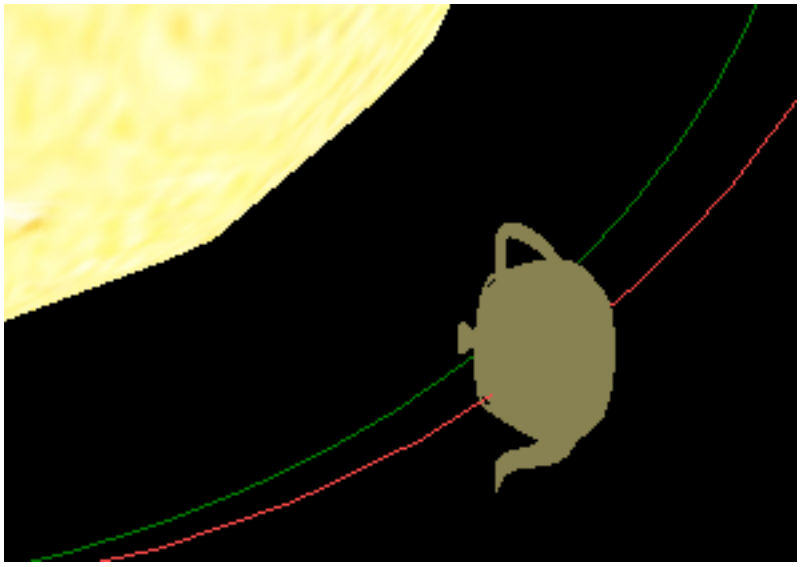

Figura 6.6: Bule de Chá

Capítulo 7

Extras

7.1 Cilindro

- **Vetores Normais**: são utilizadas coordenadas polares, e, em consequência disso, os vetores são iguais aos vértices. Como tal, para normaliza-los, basta não multiplicar pelo radius.
- **Coordenadas de Textura**: De modo a mapear os vértices em valores dentro de intervalos de 0 a 1, divide-se por 2pi.

```
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) ;
}
```

7.2 Anel

- **Vetores Normais**: Cálculo é feito de imediato;
- **Coordenadas de Textura**: Cálculo é feito de imediato;

```
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);
        }
 }

}
```

Capítulo 8

Conclusão

O objetivo deste trabalho prático é no fundo aplicar de uma forma global os conhecimentos adquiridos nas aulas práticas nas ultimas semanas, sendo por isso esperada a implementação com sucesso das tarefas pedidas, tais como, vetores normais, iluminação e texturas.

Uma das principais dificuldade prende-se com a implementação das texturas devido ao sistema operativo MacOS, sendo posteriormente mudada a máquina para uma em sistema operativo Linux o problema foi facilmente resolvido.

Concluímos então com esta quarta e última fase do trabalho prático de Computação Gráfica, na opinião do grupo todos os objetivos requisitados foram cumpridos tendo ajudado imenso no desenvolvimento dos conhecimentos lecionados ao longo do semestre.