Universidade do Minho
Mestrado Integrado em Engenharia Informática
Departamento de Informática

Computação Gráfica

Coordenadas das Texturas e Normais

Isabel Sofia da Costa Pereira A76550
José Francisco Gonçalves Petejo e Igreja Matos A77688
Maria de La Salete Dias Teixeira A75281
Tiago Daniel Amorim Alves A78218

20 de Maio de 2018

Conteúdo

1 Introdução

Neste projeto, desenvolvido no âmbito da UC de Computação Gráfica, foi proposto a realização de uma mini cena gráfica 3D. Para tal, foi necessário utilizar certos recursos tais como C++ e OpenGL.

O trabalho está dividido em quatro fases, estando presente neste relatório uma explicação da abordagem tomada para o desenvolvimento da quarta que consiste na implementação de texturas, materiais e iluminação dos objetos.

Para uma demonstração das funcionalidades desenvolvidas, elaborou-se um protótipo dinâmico do Sistema Solar.

1.1 Alterações e Conservações

Tendo em conta que esta é a quarta fase do projeto, existem, naturalmente, algumas mudanças na estrutura do código, assim como conservações, ou seja, funcionalidades implementadas na primeira, segunda e terceira fases que se mantêm.

Na fase anterior, o XML continha o nome dos ficheiros 3d com as coordenadas a serem representadas, as transformações necessárias a aplicar sobre a figura em questão, uma variável *time* capaz de representar a velocidade destas mesmas transformações e ainda um conjunto de pontos para a construção de curvas *Catmull-Rom*.

Relativamente à quarta fase, visto que é necessário representar texturas e materiais, o ficheiro XML poderá conter, associado ao ficheiro 3d a desenhar, o caminho para uma imagem que constitui uma textura e a definição das componentes de cor, ou seja, `diffuse`, `ambient`, `diffuse and ambient`, `emissive`, `specular` e `shininess`. Além disso, o ficheiro XML também poderá possuir fontes de iluminação, isto é, `POINT`, `SPOT` e `DIRECTIONAL`.

Tal como na fase anterior, de forma a otimizar o projeto, foram utilizados *VBOs* para o desenho das figuras, normais e texturas.

Com as mudanças do ficheiro XML, foi necessário alterar as seguintes classes: *struct*, *Parser* e *Engine*. Além disso, foram criadas as classes *Material*, *Light* e *Scene*.

Quanto à implementação das texturas e iluminação, foi necessário modificar outros ficheiros, tal como a classe *Vertex* e o ficheiro *Generator*, para que fosse possivel imprimir nos ficheiros 3d a informação relativa às normais e texturas.

2 Desenvolvimento do Projeto

Para o desenvolvimento do trabalho foi conveniente a utilização dos ficheiros *generator* e o *engine* que representam as duas aplicações requeridas.

2.1 Gerador

O gerador, *generator*, tal como nas outras fases, é responsável pelo cálculo dos vértices de uma figura primitiva (`plane`, `box`, `cone`, `sphere` e `torus`) ou `patch`, guardando todos esses num ficheiro 3d passado como paramêtro. Para tal, o *generator* utiliza a classe *vertex* como auxílio.

Nesta fase, é acrescentada a funcionalidade de gerar e imprimir num ficheiro 3d os vértices das normais e das texturas, tendo sido prestada atenção extra a esta última, visto que tem de ser gerada num contexto 2D.

2.2 Engine

O *engine*, semelhante às fases anteriores, tem como objetivo apresentar uma janela exibindo os resultados processados através da leitura de um ficheiro XML. Devido às mudanças neste ficheiro, o *engine* foi sujeito a alterações que se encontram explicadas mais pormenorizadamente adiante.

2.3 Classes

Tal como referido anteriormente, devido às novas funcionalidades do programa, várias classes foram modificadas e outras adicionadas. Sendo assim, todas as classes utilizadas neste projeto encontram-se descritas pormenorizadamente nesta secção.

2.3.1 Point

Esta classe, tal como nas fases anteriores, representa um ponto num referencial a três dimensões, com as coordendas `X`, `Y` e `Z`. Esta torna-se bastante útil na representação dos vértices utilizados para o desenho dos triângulos que elaboram as figuras primitivas, as normais ou as texturas.

```
class Point{

        float x;
        float y;
        float z;

public:
        Point();
        Point(float,float,float);
        float getX();
        float getY();
        float getZ();
        void setX(float);
        void setY(float);
        void setZ(float);
};
```

2.3.2 Transform

Tal como na fase anterior, esta classe representa toda a informação de uma determinada transformação geométrica. Desta forma, a classe *Transform* armazena a designação da transformação, que pode ser `translate`, `rotate` ou `scale`. Esta também possui a variavel `time`, adicionada na fase anterior, referente à velocidade a que figura realiza determinada transformação. Além do que foi mencionado anteriormente, armazena igualmente os restantes dados associados à transformação presentes no ficheiro XML, como o ângulo, as coordenadas e os pontos para representar a curva de *Catmull-Rom*.

```
class Transform{

        string name;
        float timeT, ang;
        vector<Point*> pointsL;

public:
        Transform();
        Transform(string,float,float,vector<Point*>);
        string getName();
```

```
11          float getTime();
12          float getAngle();
13          vector<Point*> getPoints();
14          Point* getPoint();
15          void setName(string);
16          void setTime(float);
17          void setAngle(float);
18          void setPoint(vector<Point*>);
19          Transform* clone();
20  };
```

2.3.3 Material

A classe *Material* foi criada com o intuito de armazenar a informação obtida do ficheiro XML, mais especificamente, informação relativa aos materiais das figuras.

Os materiais, por sua vez, podem possuir as seguintes propriedades:

- *Diffuse*
- *Ambient*
- *Diffuse and Ambient*
- *Specular*
- *Emission*
- *Shininess*

Além disso, esta classe é responsável por gerar uma cor aleatória caso nenhum material ou textura tenha sido associado à figura.

```
1   if(notDiffuse && notAmbient && notDiffuseANDambient &&
    ↪  notEmission && notSpecular && !texture){
2           srand(time(NULL));
3           //geração da cor aleatória
4           red = (float) rand() / (float) RAND_MAX;
5           green = (float) rand() / (float) RAND_MAX;
6           blue = (float) rand() / (float) RAND_MAX;
```

```
7           if (red <= 0.1 && green <= 0.1 && blue <= 0.1) red = 1;
8           diffuseANDambient[0]=red;
9           diffuseANDambient[1]=green;
10          diffuseANDambient[2]=blue;
11      }
```

Esta classe possui ainda a função `draw()` que ativa o material de acordo com os pontos obtidos do ficheiro XML.

```
1   void Material::draw() {
2       glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
3       glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, diffuse);
4
5       if(diffuseANDambient[0]!=0 || diffuseANDambient[1]!=0 ||
        ↪  diffuseANDambient[2]!=0)
6           glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE,
            ↪  diffuseANDambient);
7
8       glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, shininess);
9       glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, specular);
10      glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, emission);
11  }
```

Desta forma a classe apresenta o seguinte formato:

```
1   class Material{
2
3           float diffuse[4], ambient[4], diffuseANDambient[4],
            ↪  specular[4], emission[4], shininess;
4
5   public:
6           Material();
7           Material(Point*, Point*, Point*, Point*, Point*, float,
            ↪  bool);
8           float* getDiffuse();
9           float* getAmbient();
10          float* getDiffuseANDambient();
11          float* getSpecular();
```

```
12          float* getEmission();
13          float getShininess();
14          void draw();
15  };
```

2.3.4 Struct

Tal como sugere o nome, esta classe possui como objetivo guardar os dados de uma figura retirados do ficheiro XML, cumprindo assim o mesmo propósito que nas fases anteriores. Desta forma, é possível armazenar nesta estrutura todas as informações da figura apresentadas de abaixo.

- O caminho do ficheiro 3d;
- O caminho do ficheiro da sua textura;
- As várias transformações a serem aplicadas;
- O material que a constitui;
- A textura;
- Os vértices e as coordenadas normais e de textura que a constituem;
- O buffer;
- Os arrays relativos à utilização de *VBOs* sobre os vértices, normais e texturas.

Sendo assim, esta classe possui a seguinte estrutura:

```
1   class Struct{
2
3           string file3d, fileTexture;
4           vector<Transform*> refit;
5           Material material;
6           GLuint texture;
7           vector<Point*> points, normals, textures;
8           GLuint buffer[3];
9           float *points_array, *normals_array, *textures_array;
10
```

```
11 public:
12         Struct();
13         string getFile3d();
14         string getFileTexture();
15         vector<Transform*> getRefit();
16         Material getMaterial();
17         GLuint getTexture();
18         vector<Point*> getPoints();
19         vector<Point*> getNormals();
20         vector<Point*> getTextures();
21         GLuint getBuffer();
22         float* getPointsArray();
23         float* getNormalsArray();
24         float* getTexturesArray();
25         void setFile3d(string);
26         void setFileTexture(string);
27         void setRefit(vector<Transform*>);
28         void setMaterial(Material);
29         void setTexture(GLuint);
30         void setPoints(vector<Point*>);
31         void setNormals(vector<Point*>);
32         void setTextures(vector<Point*>);
33         void setBuffer(GLuint);
34         void setPointsArray(float*);
35         void setNormalsArray(float*);
36         void setTexturesArray(float*);
37         void addTransform(Transform*);
38         void addTransform(vector<Transform*>);
39         void prepareTexture(string s);
40         void fillBuffer();
41         void draw();
42 };
```

Como se pode verificar, apareceram novas variáveis sendo uma delas a textura. De forma a que o *engine* seja capaz de considerar texturas, julgou-se conveniente criar o método `prepareTexture(textura)`. Este método aparece de modo a que se possa carregar a textura a partir de um ficheiro, utilizando os metódos presentes na biblioteca *il.h*, tal como foi leccionado nas aulas práticas. É também nesta função que se ativa o *buffer* das texturas.

```
1  void Struct::prepareTexture(string s) {
2          unsigned int t,tw,th;
3          unsigned char *texData;
4
5          //inicializar o devIL
6          ilInit();
7          ilEnable(IL_ORIGIN_SET);
8          ilOriginFunc(IL_ORIGIN_LOWER_LEFT);
9          ilGenImages(1,&t);
10         ilBindImage(t);
11         ilLoadImage((ILstring)s.c_str());
12         tw = ilGetInteger(IL_IMAGE_WIDTH);
13         th = ilGetInteger(IL_IMAGE_HEIGHT);
14
15         ilConvertImage(IL_RGBA, IL_UNSIGNED_BYTE);
16         texData = ilGetData();
17         //inicialização do buffer relativo à textura
18         glGenTextures(1,&texture);
19
20         //associar a textura à sua variavel
21         glBindTexture(GL_TEXTURE_2D,texture);
22         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
           ↪ GL_REPEAT);
23         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
           ↪ GL_REPEAT);
24
25         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
           ↪ GL_LINEAR_MIPMAP_LINEAR);
26         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
           ↪ GL_LINEAR_MIPMAP_LINEAR);
27
```

```
28          glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, tw, th, 0,
            ↪  GL_RGBA, GL_UNSIGNED_BYTE, texData);
29          glGenerateMipmap(GL_TEXTURE_2D);
30
31          glBindTexture(GL_TEXTURE_2D, 0);
32  }
```

Para além da variável texture, apareceram outras novas variáveis que associam à figura o VBO correspondente às normais e às texturas. Sendo assim, quando é executado o *Parser* sobre o ficheiro XML, é conveniente preencher o `buffer` com os pontos da figura, os pontos das normais e os pontos da textura. Desta forma, extendeu-se o método `fillBuffer()` de modo a guardar a informação extra em `normals_Array` e em `textures_Array`. De seguida, os *buffers* são preparados de forma a carregar toda a informação contida em cada posiçao do buffer (posição 0 contém os pontos, 1 contém as normais e 2 contém a texturas).

```
1   void Struct::fillBuffer(){
2           Point p;
3           int index = 0;
4
5           //existirá tantos floats quanto o número de pontos*3
6           //pois cada ponto é constituido por um X, um Y e um Z
7           points_array = (float*) malloc(sizeof(float) *
            ↪  points.size() * 3);
8           normals_array = (float*) malloc(sizeof(float) *
            ↪  normals.size() * 3);
9           textures_array = (float*) malloc(sizeof(float) *
            ↪  textures.size() * 3);
10
11          //preencher o vertex_array com os pontos do ficheiro
            ↪  3d correspondentes aos vértices
12          for (vector<Point *>::const_iterator i =
            ↪  points.begin(); i != points.end(); ++i) {
13                  p = **i;
14                  points_array[index] = p.getX();
15                  points_array[index+1] = p.getY();
16                  points_array[index+2] = p.getZ();
```

```
17              index+=3;
18          }
19
20          //preencher o normals_array com os pontos do ficheiro
            ↪ 3d correspondentes às normais
21          index = 0;
22          for (vector<Point *>::const_iterator i =
            ↪ normals.begin(); i != normals.end(); ++i) {
23              p = **i;
24              normals_array[index] = p.getX();
25              normals_array[index+1] = p.getY();
26              normals_array[index+2] = p.getZ();
27              index+=3;
28          }
29
30          //preencher o textures_array com os pontos do ficheiro
            ↪ 3d correspondentes às texturas
31          //neste caso só a coordena X e Y é que importam pois a
            ↪ textura é um ficheiro 2D
32          index = 0;
33          for (vector<Point *>::const_iterator i =
            ↪ textures.begin(); i != textures.end(); ++i) {
34              p = **i;
35              textures_array[index] = p.getX();
36              textures_array[index+1] = p.getY();
37              index+=2;
38          }
39
40          //geração de 3 buffers
41          glGenBuffers(3, buffer);
42
43          //ativação do buffer 0 associado aos vertices
44          glBindBuffer(GL_ARRAY_BUFFER,buffer[0]);
45          //preenchimento do buffer com os pontos do
            ↪ vertex_array e escolha do padrão de desenho
46          glBufferData(GL_ARRAY_BUFFER, sizeof(float) *
            ↪ points.size() * 3, points_array, GL_STATIC_DRAW);
47
```

```
48          //ativação do buffer 1 associado às normais
49          glBindBuffer(GL_ARRAY_BUFFER,buffer[1]);
50          //preenchimento do buffer com os pontos do
            ↪ normal_array e escolha do padrão de desenho
51          glBufferData(GL_ARRAY_BUFFER, sizeof(float) *
            ↪ normals.size() * 3, normals_array, GL_STATIC_DRAW);
52
53          //ativação do buffer 2 associado às texturas
54          glBindBuffer(GL_ARRAY_BUFFER, buffer[2]);
55          //preenchimento do buffer com os pontos do
            ↪ texture_array e escolha do padrão de desenho
56          glBufferData(GL_ARRAY_BUFFER, sizeof(float) *
            ↪ textures.size() * 2, textures_array,
            ↪ GL_STATIC_DRAW);
57
58          free(points_array); free(normals_array);
            ↪ free(textures_array);
59  }
```

Por fim, com os buffers já preenchidos, a função `draw` pode ser invocada no ficheiro *engine* permitindo um desenho mais rápido e eficiente da figura através dos triângulos constituidos pelos pontos que já se encontram preparados.

```
1   void Struct::draw(){
2       //ativação do buffer 0
3       glBindBuffer(GL_ARRAY_BUFFER,buffer[0]);
4       //especificação do formato dos pontos a ler do buffer,
5       //neste caso tratam-se de 3 floats por vértice
6       glVertexPointer(3,GL_FLOAT,0,0);
7
8       if(normals.size()!=0){
9           //ativação do buffer 1
10          glBindBuffer(GL_ARRAY_BUFFER, buffer[1]);
11          //como são normais, utilizar glNormalPointer
12          glNormalPointer(GL_FLOAT, 0, 0);
13      }
14
```

```
15      if(textures.size()!=0){
16          //ativação do buffer 2
17          glBindBuffer(GL_ARRAY_BUFFER, buffer[2]);
18          //como são texturas, utilizar glTexCoordPointer
19          //(atenção: apenas se utilizam 2 valores em vez de 3)
20          glTexCoordPointer(2, GL_FLOAT, 0, 0);
21          //associar a textura à sua respetiva variável
22          glBindTexture(GL_TEXTURE_2D, texture);
23      }
24
25      //especificação do método de desenho e do início e fim do
        ↪  buffer (triângulos, tal como proposto no enunciado)
26      glDrawArrays(GL_TRIANGLES, 0,
        ↪  (points.size()+normals.size()+textures.size()) * 3);
27      //fazer reset à textura.
28      glBindTexture(GL_TEXTURE_2D, 0);
29  }
```

2.3.5 Light

Uma das classes criadas foi a *Light* fundamental para guardar a informação relativa às origens de luz. Esta classe possui duas variáveis, uma para identificar se a luz é pontual, e outra que guarda a posição ou direção da luz.

```
1   class Light{
2
3           int isPoint;
4           Point* point;
5
6   public:
7           Light();
8           Light(int, Point*);
9           int getIsPoint();
10          Point* getPoint();
11          void draw();
12  };
```

Esta classe possui ainda a função `draw()` que ativa as luzes de acordo com os pontos obtidos do ficheiro XML, sendo convertido o valor de *isPoint* para 1 caso se trate de uma luz pontual, como referido anteriormente.

```
void Light::draw() {
    GLfloat pos[4] = {point->getX(), point->getY() ,
     ↪  point->getZ(), isPoint};

    // light position
    glLightfv(GL_LIGHT0, GL_POINT, pos);

    // light colors
    GLfloat amb[4] = {0.1, 0.1, 0.1, 1.0};
    GLfloat diff[4] = {1.0, 1.0, 1.0, 1.0};

    glLightfv(GL_LIGHT0, GL_AMBIENT, amb);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diff);
}
```

2.3.6 Scene

A classe *Scene* junta toda a informação da cena, ou seja, todas as estruturas e luzes presentes no ficheiro XML. Desta forma, esta é a variável utilizada no ficheiro *engine* para aceder a toda a informação necessária para a elaboração da cena.

```
class Scene{

        vector<Light> luzes;
        vector<Struct> estruturas;

public:
        Scene();
        vector<Light> getLuzes();
        vector<Struct> getEstruturas();
        void setLuzes(vector<Light>);
        void setEstruturas(vector<Struct>);
};
```

2.3.7 Patch

A classe patch, que tem como objetivo armazenar os pontos de controlo de um *patch*, foi uma das poucas classes a não sofrer qualquer alteração nesta fase, por isso mantém a mesma estrutura.

```
class Patch{

    vector<Point*> controlPoints;

public:
    Patch();
    Patch(vector<Point*>);
    vector<Point*> getControlPoints();
    void setCtrlPoints(vector<Point*>);
};
```

2.3.8 Vertex

A classe *Vertex* sofreu bastantes alterações, passando a gerar não só os vértices para o desenho das figuras, como também os pontos para o desenho das normais e das texturas. Sendo assim, foram acrescentadas duas novas variáveis que guardam esses pontos: `normalsList` e `texturesList`.

```
class Vertex{

        vector<Point*> pointsList;
        vector<Point*> normalsList;
        vector<Point*> texturesList;

public:
        Vertex();
        Vertex(vector<Point*>,vector<Point*>,vector<Point*>);
        vector<Point*> getPointsList();
        vector<Point*> getNormalsList();
        vector<Point*> getTexturesList();
        void setPointsList(vector<Point*>);
        void setNormalsList(vector<Point*>);
        void setTexturesList(vector<Point*>);
```

```
16          void makePlane(float);
17          void makeBox(float, float, float, int);
18          void makeCone(float, float, int, int);
19          void makeSphere(float, int, int);
20          void makeTorus(float, float, int, int);
21          Point* bezierCurve(float, Point*, Point*, Point*,
            ↪ Point*);
22          Point* bezierPatch(float, float, vector<Point*>);
23          void bezierPatchTriangles(int, vector<Patch*>);
24  };
```

As modificações aplicadas a esta classe apresentam-se descritas na secção *Gerador* que se encontra mais à frente.

2.4 Outros Ficheiros

2.4.1 tinyxml2

O *tinyxml2* é uma ferramenta que processa ficheiros XML, sendo de extrema importância para o funcionamento do *Parser*.

```
1  namespace tinyxml2 {
2  class XMLDocument;
3  class XMLElement;
4  class XMLAttribute;
5  class XMLComment;
6  class XMLText;
7  class XMLDeclaration;
8  class XMLUnknown;
9  class XMLPrinter;
10 ... }
```

2.4.2 Parser

Este ficheiro é crucial para o bom funcionamento do *engine* devido ao facto de ser este que efetua o parsing do ficheiro XML. Desta forma, o *Parser* é responsável por inserir toda a informação encontrada no documento XML numa *Scene*, em vez de guardar num vetor de *Structs*, como nas fases

anteriores. Sendo assim, é natural que a função principal, `lookFiles`, foi modificada para devolver uma *Scene*.

Além disso, como o XML sofreu alterações, outras funções do *Parser* também se encontram modificadas relativamente à fase anterior.

Pela primeira vez, desde a primeira fase, os ficheiros 3d encontram-se alterados, logo a função `readFile` foi alterada de modo a conseguir ler os pontos da figura, das normais e das texturas.

Para além disso, foi adicionada a função `lookLight`, responsável por identificar o tipo de luz (se se trata de uma luz pontual ou não) e as suas respetivas posições ou direções.

De forma semelhante, foi adicionada a funçao `lookUpMaterial` que identifica os tipos de material, e utiliza um *Point* para preencher os seus valores de cor (vermelho, verde e azul). É nesta função que também é obtida a informação sobre a shininess.

Criou-se também a função `lookUpModel` que obtem o ficheiro 3d (anteriormente obtido pela função `lookAux`) e o ficheiro da textura, preparando este último através da função `prepareTexture`, descrita na secção *2.3.4*. Além disso, também obtem o material através da função `lookUpMaterial` explicada acima.

Como deu para entender, a função `lookAux` foi modificada. Agora esta é capaz de identificar igualmente os novos parâmetros do XML, como *lights* não existindo mais o parâmetro *color* devido à implementação dos materiais.

Sendo assim, as unicas funções que não sofreram alterações foram a `lookUpTranslate` e a `lookUpTransformation`, responsáveis por extrair a informação das transformações `translate`, `rotate` e `scale`.

```
1 Struct readFile(string, Struct);
2 Struct lookUpTranslate(XMLElement*, Struct);
3 Struct lookUpTransformation(XMLElement*, Struct);
4 void lookUpLight(XMLElement*);
5 Struct lookUpMaterial(bool, XMLElement*, Struct);
6 Struct lookUpModel(XMLElement*, Struct);
7 vector<Struct> lookAux(XMLElement*);
8 Scene lookFiles(char*);
9 int parseXML(char*);
```

3 Gerador

Tal como mencionado anteriormente, o *generator* é responsável pela criação dos ficheiros 3d que contêm não só os vertíces para desenho, como também, nesta fase, os pontos das normais e das texturas. Sendo assim, as funções geradoras sofreram alterações.

Nas secções seguintes serão abordados os raciocínios para a geração das normais e das texturas.

3.1 Plano

No desenho de um plano, é de esperar que o vetor normal tenha uma direção no sentido positivo do eixo dos Y para qualquer ponto, isto é, `(0,1,0)`.

Quanto às texturas, estas irão acompanhar os vertices a ser desenhados, ou seja, o valor `h` corresponde a 1 e o valor `-h` corresponde ao valor 0. Sendo assim, é possível percorrer toda a textura através de dois triângulos e aplicá-la corretamente ao plano.

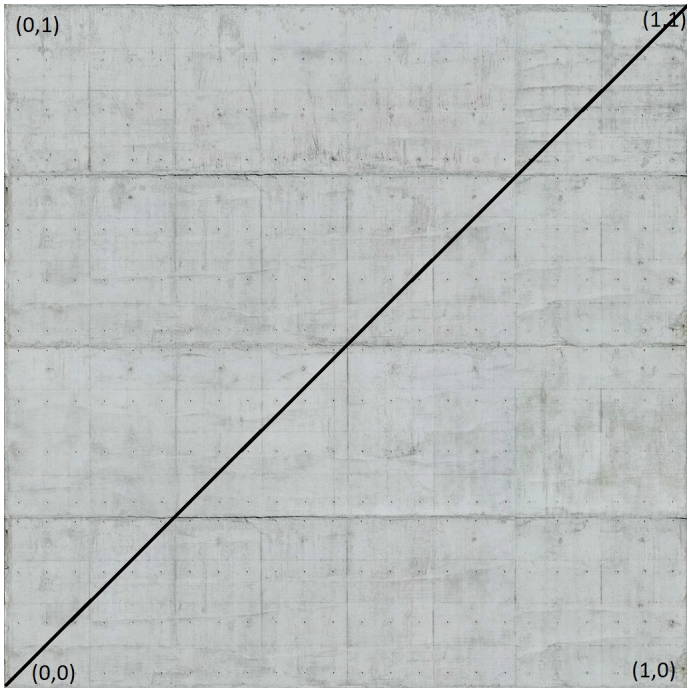

Figura 1: Textura do plano

```
void Vertex::makePlane(float size){
    //centrar o plano
    float h = size/2;

    pointsList.push_back(new Point(h, 0, h));
    pointsList.push_back(new Point(h, 0, -h));
    pointsList.push_back(new Point(-h, 0, -h));
    normalsList.push_back(new Point(0,1,0));
    normalsList.push_back(new Point(0,1,0));
    normalsList.push_back(new Point(0,1,0));
    texturesList.push_back(new Point(1,1,0));
    texturesList.push_back(new Point(1,0,0));
    texturesList.push_back(new Point(0,0,0));

    pointsList.push_back(new Point(h, 0, h));
    pointsList.push_back(new Point(-h, 0, -h));
    pointsList.push_back(new Point(-h, 0, h));
    normalsList.push_back(new Point(0,1,0));
    normalsList.push_back(new Point(0,1,0));
    normalsList.push_back(new Point(0,1,0));
    texturesList.push_back(new Point(1,1,0));
    texturesList.push_back(new Point(0,0,0));
    texturesList.push_back(new Point(0,1,0));
```

3.2 Paralelipipedo

Para gerar um paralelipipedo teve-se por base uma textura desdobrada desta figura como a apresentada de seguida.

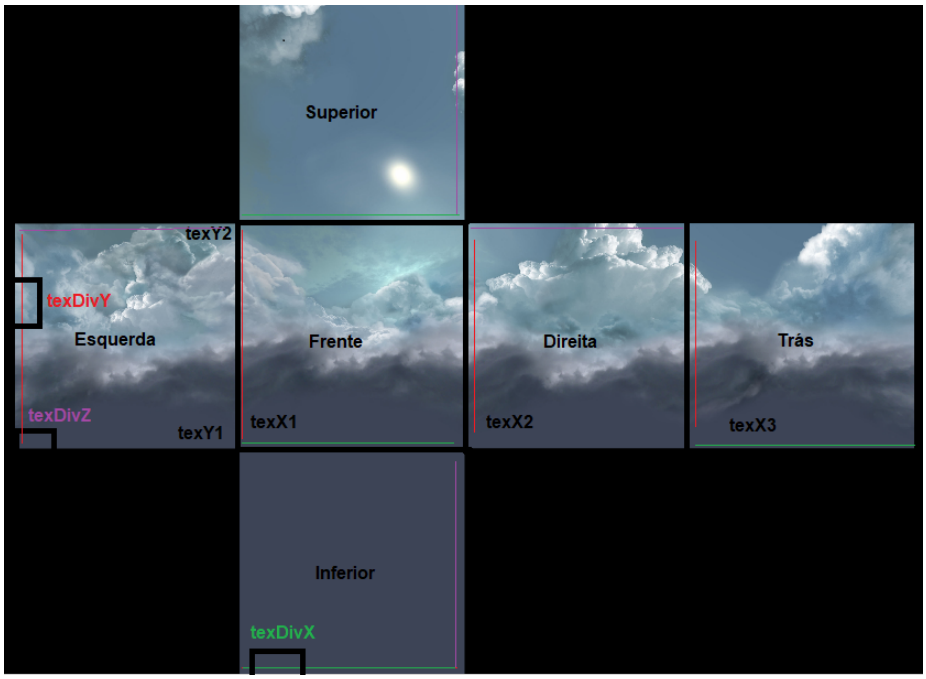

Figura 2: Textura do paralelipipedo

Tal como se pode ver pela figura 2, considerou-se cinco pontos base (`texY1`, `texY2`, `texX1`, `texX2` e `texX3`) que representam as posições fundamentais para identificar as diferentes faces à medida que se desenha a figura.

Quanto às variaveis `texDivX`, `texDivY` e `texDivZ`, estas representam a divisão numa determinada aresta da figura. A escolha de cada uma destas depende de qual medida a aresta é representada, `X` (identificado a verde), `Y` (identificado a vermleho) ou `Z` (identificado a roxo).

Com esta informação e sabendo que o paralelipipedo desenvolvido na primeira fase é desenhado da esquerda para a direita e de cima para baixo, elaborou-se o seguinte código.

```
void Vertex::makeBox(float x,float y,float z,int divisions) {
    (...)

    // Calculo dos pontos que identificam as faces da textura
    float texY1 = z/((z*2)+y);
    float texY2 = (z+y)/((z*2)+y);

    float texX1 = (z)/((z*2)+(x*2));
    float texX2 = (z+x)/((z*2)+(x*2));
    float texX3 = ((z*2)+x)/((z*2)+(x*2));

    // Calculo das divisões da textura
    float texDivX = (x/((z*2)+(x*2)))/float(divisions);
    float texDivY = (y/((z*2)+y))/float(divisions);
    float texDivZ = (z/((z*2)+(x*2)))/float(divisions);

    for(int i=0; i<divisions; i++){
       for(int j=0; j<divisions; j++){

            // Face frontal
            (... cálculo dos pontos ...)
            // Visto que se trata da face frontal, o vetor
             ↪  normal ao longo desta face será sempre
             ↪  (0,0,1)
            normalsList.push_back(new Point(0,0,1));
            normalsList.push_back(new Point(0,0,1));
            normalsList.push_back(new Point(0,0,1));
            // Começar na posição de texX1 e ir aumentando um
             ↪  "passo" de cada vez. Por outro lado começa-se
             ↪  no texY2 e diminui-se um "passo"
            texturesList.push_back(new Point(texX1+(j*texDivX),
             ↪  texY2 - (i*texDivY),0));
            texturesList.push_back(new Point(texX1+(j*texDivX),
             ↪  (texY2 - texDivY)-(i*texDivY),0));
            texturesList.push_back(new
             ↪  Point((texX1+texDivX)+(j*texDivX),
             ↪  (texY2-texDivY)-(i*texDivY),0));
```

```
30
31              (... cálculo dos pontos ...)
32              normalsList.push_back(new Point(0,0,1));
33              normalsList.push_back(new Point(0,0,1));
34              normalsList.push_back(new Point(0,0,1));
35              // Triângulo complementar ao anterior
36              texturesList.push_back(new Point(texX1+(j*texDivX),
                ↪  texY2-(i*texDivY),0));
37              texturesList.push_back(new
                ↪  Point((texX1+texDivX)+(j*texDivX),
                ↪  (texY2-texDivY)-(i*texDivY),0));
38              texturesList.push_back(new
                ↪  Point((texX1+texDivX)+(j*texDivX),
                ↪  texY2-(i*texDivY),0));
39
40
41              // Face traseira
42              (... cálculo dos pontos ...)
43             // Visto que se trata da face traseira, o vetor
                ↪  normal ao longo desta face será sempre
                ↪  (0,0,-1)
44              normalsList.push_back(new Point(0,0,-1));
45              normalsList.push_back(new Point(0,0,-1));
46              normalsList.push_back(new Point(0,0,-1));
47              // Começar na posição final, 1, e ir diminuindo um
                ↪  "passo" de cada vez. Por outro lado começa-se
                ↪  no texY2 e diminui-se um "passo"
48              texturesList.push_back(new
                ↪  Point((1-texDivX)-(j*texDivX),
                ↪  (texY2-texDivY)-(i*texDivY),0));
49              texturesList.push_back(new Point(1-(j*texDivX),
                ↪  (texY2-texDivY)-(i*texDivY),0));
50              texturesList.push_back(new Point(1-(j*texDivX),
                ↪  texY2-(i*texDivY),0));
51
52              (... cálculo dos pontos ...)
53              normalsList.push_back(new Point(0,0,-1));
```

```
54              normalsList.push_back(new Point(0,0,-1));
55              normalsList.push_back(new Point(0,0,-1));
56             // Triângulo complementar ao anterior
57              texturesList.push_back(new
                ↪  Point((1-texDivX)-(j*texDivX),
                ↪  (texY2-texDivY)-(i*texDivY),0));
58              texturesList.push_back(new Point(1-(j*texDivX),
                ↪  texY2-(i*texDivY),0));
59              texturesList.push_back(new
                ↪  Point((1-texDivX)-(j*texDivX),
                ↪  texY2-(i*texDivY),0));
60
61
62              // Face lateral esquerda
63              (... cálculo dos pontos ...)
64              // Visto que se trata da face esquerda, o vetor
                ↪  normal ao longo desta face será sempre
                ↪  (-1,0,0)
65              normalsList.push_back(new Point(-1,0,0));
66              normalsList.push_back(new Point(-1,0,0));
67              normalsList.push_back(new Point(-1,0,0));
68              // Começar na posição inicial, 0, (tendo em conta
                ↪  que esta face utiliza a variavel Z) e ir
                ↪  aumentando um "passo" de cada vez. Por outro
                ↪  lado começa-se no texY2 e diminui-se um
                ↪  "passo".
69              texturesList.push_back(new Point((j*texDivZ),
                ↪  texY2-(i*texDivY),0));
70              texturesList.push_back(new Point(j*texDivZ,
                ↪  (texY2-texDivY)-(i*texDivY),0));
71              texturesList.push_back(new
                ↪  Point(texDivZ+(j*texDivZ),
                ↪  (texY2-texDivY)-(i*texDivY),0));
72
73              (... cálculo dos pontos ...)
74              normalsList.push_back(new Point(-1,0,0));
75              normalsList.push_back(new Point(-1,0,0));
```

```
76              normalsList.push_back(new Point(-1,0,0));
77              // Triângulo complementar ao anterior
78              texturesList.push_back(new Point((j*texDivZ),
                ↪  texY2-(i*texDivY),0));
79              texturesList.push_back(new
                ↪  Point(texDivZ+(j*texDivZ),
                ↪  (texY2-texDivY)-(i*texDivY),0));
80              texturesList.push_back(new
                ↪  Point(texDivZ+(j*texDivZ),
                ↪  texY2-(i*texDivY),0));
81
82
83              // Face lateral direita
84              (... cálculo dos pontos ...)
85              // Visto que se trata da face direita, o vetor
                ↪  normal ao longo desta face será sempre
                ↪  (0,0,1)
86              normalsList.push_back(new Point(1,0,0));
87              normalsList.push_back(new Point(1,0,0));
88              normalsList.push_back(new Point(1,0,0));
89              // Começar na posição de texX2 e ir aumentando um
                ↪  "passo" de cada vez, em Z. Por outro lado
                ↪  começa-se no texY2 e diminui-se um "passo".
90              texturesList.push_back(new Point(texX2+(j*texDivZ),
                ↪  (texY2-texDivY)-(i*texDivY),0));
91              texturesList.push_back(new
                ↪  Point((texX2+texDivZ)-(j*texDivX),
                ↪  (texY2-texDivY)-(i*texDivY),0));
92              texturesList.push_back(new Point(texX2+(j*texDivZ),
                ↪  texY2-(i*texDivY),0));
93
94              (... cálculo dos pontos ...)
95              normalsList.push_back(new Point(1,0,0));
96              normalsList.push_back(new Point(1,0,0));
97              normalsList.push_back(new Point(1,0,0));
98              // Triângulo complementar ao anterior
```

```
99          texturesList.push_back(new
            ↪  Point((texX2+texDivZ)+(j*texDivZ),
            ↪  (texY2-texDivY)-(i*texDivY),0));
100         texturesList.push_back(new
            ↪  Point((texX2+texDivZ)+(j*texDivX),
            ↪  texY2-(i*texDivY),0));
101         texturesList.push_back(new Point(texX2+(j*texDivZ),
            ↪  texY2-(i*texDivY),0));
102
103
104         // Base inferior
105         (... cálculo dos pontos ...)
106         // Visto que se trata da face inferior, o vetor
            ↪  normal ao longo desta face será sempre
            ↪  (0,-1,0)
107         normalsList.push_back(new Point(0,-1,0));
108         normalsList.push_back(new Point(0,-1,0));
109         normalsList.push_back(new Point(0,-1,0));
110         // Começar na posição de texX1 e ir aumentando um
            ↪  "passo" de cada vez. Por outro lado começa-se
            ↪  no texY1 e diminui-se um "passo", em Z.
111         texturesList.push_back(new Point(texX1+(j*texDivX),
            ↪  (texY1-texDivZ)-(i*texDivZ),0));
112         texturesList.push_back(new Point(texX1+(j*texDivX),
            ↪  texY1-(i*texDivZ),0));
113         texturesList.push_back(new
            ↪  Point((texX1+texDivX)+(j*texDivX),
            ↪  texY1-(i*texDivZ),0));
114
115         (... cálculo dos pontos ...)
116         normalsList.push_back(new Point(0,-1,0));
117         normalsList.push_back(new Point(0,-1,0));
118         normalsList.push_back(new Point(0,-1,0));
119         // Triângulo complementar ao anterior
120         texturesList.push_back(new Point(texX1+(j*texDivX),
            ↪  (texY1-texDivZ)-(i*texDivZ),0));
```

```
121             texturesList.push_back(new
                ↪  Point((texX1+texDivX)+(j*texDivX),
                ↪  texY1-(i*texDivZ),0));
122             texturesList.push_back(new
                ↪  Point((texX1+texDivX)+(j*texDivX),
                ↪  (texY1-texDivZ)-(i*texDivZ),0));
123
124
125             // Base superior
126             (... cálculo dos pontos ...)
127             //Visto que se trata da face frontal, o vetor
                ↪  normal ao longo desta face será sempre
                ↪  (0,1,0)
128             normalsList.push_back(new Point(0,1,0));
129             normalsList.push_back(new Point(0,1,0));
130             normalsList.push_back(new Point(0,1,0));
131             // Começar na posição de texX1 e ir aumentando um
                ↪  "passo" de cada vez. Por outro lado começa-se
                ↪  na posiçao final e diminui-se um "passo", em
                ↪  Z.
132             texturesList.push_back(new Point(texX1+(j*texDivX),
                ↪  1-(i*texDivZ),0));
133             texturesList.push_back(new Point(texX1+(j*texDivX),
                ↪  (1-texDivZ)-(i*texDivZ),0));
134             texturesList.push_back(new
                ↪  Point((texX1+texDivX)+(j*texDivX),
                ↪  (1-texDivZ)-(i*texDivZ),0));
135
136             (... cálculo dos pontos ...)
137             normalsList.push_back(new Point(0,1,0));
138             normalsList.push_back(new Point(0,1,0));
139             normalsList.push_back(new Point(0,1,0));
140            // Triângulo complementar ao anterior
141             texturesList.push_back(new Point(texX1+(j*texDivX),
                ↪  1-(i*texDivZ),0));
142             texturesList.push_back(new
                ↪  Point((texX1+texDivX)+(j*texDivX),
                ↪  (1-texDivZ)-(i*texDivZ),0));
```

```
143                 texturesList.push_back(new
                    ↪ Point((texX1+texDivX)+(j*texDivX),
                    ↪ 1-(i*texDivZ),0));
144             }
145         }
146 }
```

3.3 Esfera

De seguida apresenta-se a esfera que para calcular as normais utiliza fórmulas relativas às coordenadas cartesianas aprendidas nas aulas:

x = r*$cos(\beta)$*sin(α)

y = r*$cos(\beta)$*cos(α)

z = r*sin(β)

Visto que se trata do cálculo das normais, apenas interessa a parte não relativa ao raio da esfera (representada em itálico), mas sim a direção do vetor.

Quanto às texturas, utilizou-se a seguinte imagem como base do racíocinio.

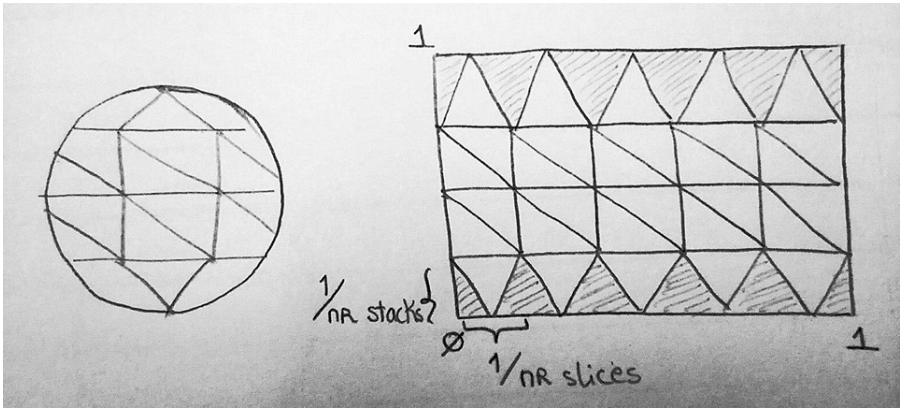

Figura 3: Algoritmo de desenho de superficies esféricas

```
void Vertex::makeSphere(float radius, int slices, int stacks){

    vector<Point*> v;
    //variáveis que representam um "passo" na textura
    float texDivY = 1.0/stacks;
    float texDivX = 1.0/slices;
    //pontos de referencia da textura
    float texYcima, texYbaixo;
    float texXesq = 0, texXdir = 0+texDivX;

    float h = (M_PI) / stacks;
    float h2 = (2 * M_PI) / slices;

    //ao mudar de slice o texXesq e texXdir aumentam uma
     ↪ divisão
    for (int i = 0; i < slices ; i++, texXesq+=texDivX,
    texXdir+=texDivX) {

        //dependendo da stack o texYcima e texYbaixo
         ↪ alteram-se
        for (int j = 0; j < stacks; j++) {

            float x2 = radius*cos((i+1)*h2)*sin(h);
            float y2 = radius*cos(h);
            float z2 = radius*sin((i+1)*h2)*sin(h);

            float x3 = radius*cos(i*h2)*sin(h);
            float y3 = radius*cos(h);
            float z3 = radius*sin(i*h2)*sin(h);

            if (j == 0) { // TOPO
                //definição dos pontos de referência nesta
                 ↪ região
                texYcima = 1.0;
                texYbaixo = 1.0-texDivY;

                pointsList.push_back(new Point(0, radius, 0));
```

```
35                  pointsList.push_back(new
                     ↪  Point(radius*cos((i+1)*h2)*sin((j+1)*h),
                     ↪  radius*cos((j+1)*h),
                     ↪  radius*sin((i+1)*h2)*sin((j+1)*h)));
36                  pointsList.push_back(new
                     ↪  Point(radius*cos(i*h2)* sin((j+1)*h),
                     ↪  radius*cos((j+1)*h),
                     ↪  radius*sin(i*h2)*sin((j+1)*h)));
37                  normalsList.push_back(new Point(0,1,0));
38                  normalsList.push_back(new Point(sin((i+1)*h2),
                     ↪  cos((j+1)*h),cos((i+1)*h2)));
39                  normalsList.push_back(new Point(sin(i*h2),
                     ↪  cos((j+1)*h),cos(i*h2)));
40                  //vertice superior do triângulo (média das
                     ↪  distâncias)
41                  texturesList.push_back(new
                     ↪  Point(texXesq+texDivX/2,texYcima,0));
42                  //base do triangulo (Y constante)
43                  texturesList.push_back(new
                     ↪  Point(texXdir,texYbaixo,0));
44                  texturesList.push_back(new
                     ↪  Point(texXesq,texYbaixo,0));
45              }
46
47              if(j == stacks-1){ // BASE
48                  //definição dos pontos de referência nesta
                     ↪  região
49                  texYcima = 0.0+texDivY;
50                  texYbaixo = 0.0;
51
52                  pointsList.push_back(new Point(0, -radius, 0));
53                  pointsList.push_back(new
                     ↪  Point(radius*cos(i*h2)*sin((j+1)*h) + x3,
                     ↪  -y2, radius*sin(i*h2)*sin((j+1)*h) + z3));
54                  pointsList.push_back(new
                     ↪  Point(radius*cos((i+1)*h2)*sin((j+1)*h) +
                     ↪  x2, -y3, radius*sin((i+1)*h2)*sin((j+1)*h)
                     ↪  + z2));
```

```
55                  normalsList.push_back(new Point(0,-1,0));
56                  normalsList.push_back(new Point(sin(i*h2),
                     ↪  -cos(h), cos(i*h2)));
57                  normalsList.push_back(new Point(sin((i+1)*h2),
                     ↪  -cos(h), cos((i+1)*h2)));
58                  //raciocínio análogo à região topo
59                  texturesList.push_back(new
                     ↪  Point(texXesq+texDivX/2, texYbaixo,0));
60                  texturesList.push_back(new
                     ↪  Point(texXesq,texYcima,0));
61                  texturesList.push_back(new
                     ↪  Point(texXdir,texYcima,0));
62              }
63
64              else{ // REGIÃO LATERAL
65                  //definição dos pontos de referência nesta
                     ↪  região que irão variar à medida que se
                     ↪  percorre a esfera
66                  texYcima = 1.0-j*texDivY;
67                  texYbaixo = 1.0-(j+1.0)*texDivY;
68
69                  pointsList.push_back(new
                     ↪  Point(radius*cos((i+1)*h2)*sin((j+1)*h),
                     ↪  radius*cos((j+1)*h),
                     ↪  radius*sin((i+1)*h2)*sin((j+1)*h)));
70                  pointsList.push_back(new
                     ↪  Point(radius*cos((i+1)*h2)*sin((j+2)*h),
                     ↪  radius*cos((j+2)*h),
                     ↪  radius*sin((i+1)*h2)*sin((j+2)*h)));
71                  pointsList.push_back(new
                     ↪  Point(radius*cos(i*h2)*sin((j+1)*h),
                     ↪  radius*cos((j+1)*h),
                     ↪  radius*sin(i*h2)*sin((j+1)*h)));
72                  normalsList.push_back(new Point(sin((i+1)*h2),
                     ↪  cos((j+1)*h), cos((i+1)*h2)));
73                  normalsList.push_back(new Point(sin((i+1)*h2),
                     ↪  cos((j+2)*h), cos((i+1)*h2)));
```

```
74                  normalsList.push_back(new Point(sin(i*h2),
                    ↪  cos((j+1)*h), cos(i*h2)));
75                  //Triângulo lateral
76                  texturesList.push_back(new
                    ↪  Point(texXdir,texYcima,0));
77                  texturesList.push_back(new
                    ↪  Point(texXdir,texYbaixo,0));
78                  texturesList.push_back(new
                    ↪  Point(texXesq,texYcima,0));
79
80
81                  pointsList.push_back(new
                    ↪  Point(radius*cos(i*h2)*sin((j+1)*h),
                    ↪  radius*cos((j+1)*h),
                    ↪  radius*sin(i*h2)*sin((j+1)*h)));
82                  pointsList.push_back(new
                    ↪  Point(radius*cos((i+1)*h2)*sin((j+2)*h),
                    ↪  radius*cos((j+2)*h),
                    ↪  radius*sin((i+1)*h2)*sin((j+2)*h)));
83                  pointsList.push_back(new
                    ↪  Point(radius*cos(i*h2)*sin((j+2)*h),
                    ↪  radius*cos((j+2)*h),
                    ↪  radius*sin(i*h2)*sin((j+2)*h)));
84                  normalsList.push_back(new Point(sin(i*h2),
                    ↪  cos((j+1)*h), cos(i*h2)));
85                  normalsList.push_back(new Point(sin((i+1)*h2),
                    ↪  cos((j+2)*h), cos((i+1)*h2)));
86                  normalsList.push_back(new Point(sin(i*h2),
                    ↪  cos((j+2)*h), cos(i*h2)));
87                  //Triângulo complementar ao anterior
88                  texturesList.push_back(new
                    ↪  Point(texXesq,texYcima,0));
89                  texturesList.push_back(new
                    ↪  Point(texXdir,texYbaixo,0));
90                  texturesList.push_back(new
                    ↪  Point(texXesq,texYbaixo,0));
91              }
92          }}}
```

3.4 Cone

Quanto ao cone, este possui um raciocínio semelhante ao da esfera em termos de normais e texturas. Desta forma, volta-se a utilizar a figura 3 como auxilio no desenho das texturas.

```
1  void Vertex::makeCone(float radius, float height, int slices,
   ↪  int stacks) {
2      double alpha = (2*M_PI)/slices;
3      double tmp = height/stacks;
4      double tanB = height/radius;
5      double h1 = 0, h2, radius2;
6      float x1, x2, x3, x4, z1, z2, z3, z4;
7
8      //variáveis que representam um "passo" na textura
9      float texDivY = 1.0/stacks , texDivX = 1.0/slices;
10
11     //pontos de referencia da textura
12     float texYcima=0+texDivY, texYbaixo=0;
13     float texXesq, texXdir;
14
15     //ao mudar de stack o texYcima e texYbaixo aumentam uma
       ↪  divisão
16     for(int j=1; j<=stacks; j++, texYcima+=texDivY,
       ↪  texYbaixo+=texDivY){
17         h2 = tmp * j;
18         radius2 = (height-h2) / tanB;
19
20                 //reiniciar os texX para a nova stack
21         texXesq = 0;
22         texXdir = 0+texDivX;
23
24         //ao mudar de slice o texXesq e texXdir aumentam uma
           ↪  divisão
25         for(int i=1; i<=slices+1; i++, texXesq+=texDivX,
26         texXdir+=texDivX){
27             x1 = radius*sin(alpha*i);
28             z1 = radius*cos(alpha*i);
```

```
29
30              x2 = radius*sin(alpha*(i+1));
31              z2 = radius*cos(alpha*(i+1));
32
33              x3 = radius2*sin(alpha*i);
34              z3 = radius2*cos(alpha*i);
35
36              x4 = radius2*sin(alpha*(i+1));
37              z4 = radius2*cos(alpha*(i+1));
38
39              if(j == 1){
40                  //BASE
41                  pointsList.push_back(new Point(0.0f,0,0.0f));
42                  pointsList.push_back(new Point(x2,0,z2));
43                  pointsList.push_back(new Point(x1,0,z1));
44                  //visto que se trata da base, a normal irá
                     ↪  apontar sempre para baixo
45                  normalsList.push_back(new Point(0,-1,0));
46                  normalsList.push_back(new Point(0,-1,0));
47                  normalsList.push_back(new Point(0,-1,0));
48                  //vertice inferior do triângulo (média das
                     ↪  distâncias)
49                  texturesList.push_back(new
                     ↪  Point(texXesq+texDivX/2,texYbaixo,0));
50                  texturesList.push_back(new
                     ↪  Point(texXdir,texYcima,0));
51                  texturesList.push_back(new
                     ↪  Point(texXesq,texYcima,0));
52
53                  //LADOS
54                  pointsList.push_back(new Point(x1,0,z1));
55                  pointsList.push_back(new Point(x2,0,z2));
56                  pointsList.push_back(new Point(x3,h2,z3));
57                  normalsList.push_back(new
                     ↪  Point(sin(alpha*i),1,cos(alpha*i)));
58                  normalsList.push_back(new
                     ↪  Point(sin(alpha*(i+1)),1,cos(alpha*(i+1))));
```

```
59                  normalsList.push_back(new
                     ↪  Point(sin(alpha*i),1,cos(alpha*i)));
60                  texturesList.push_back(new
                     ↪  Point(texXesq,texYbaixo,0));
61                  texturesList.push_back(new
                     ↪  Point(texXdir,texYbaixo,0));
62                  texturesList.push_back(new
                     ↪  Point(texXesq,texYcima,0));
63
64                  pointsList.push_back(new Point(x2,0,z2));
65                  pointsList.push_back(new Point(x4,h2,z4));
66                  pointsList.push_back(new Point(x3,h2,z3));
67                  normalsList.push_back(new
                     ↪  Point(sin(alpha*(i+1)),1,cos(alpha*(i+1))));
68                  normalsList.push_back(new
                     ↪  Point(sin(alpha*(i+1)),1,cos(alpha*(i+1))));
69                  normalsList.push_back(new
                     ↪  Point(sin(alpha*i),1,cos(alpha*i)));
70                  texturesList.push_back(new
                     ↪  Point(texXdir,texYbaixo,0));
71                  texturesList.push_back(new
                     ↪  Point(texXdir,texYcima,0));
72                  texturesList.push_back(new
                     ↪  Point(texXesq,texYcima,0));
73              }
74              else if(j == stacks){
75                  //TOPO
76                  pointsList.push_back(new Point(x1,h1,z1));
77                  pointsList.push_back(new Point(x2,h1,z2));
78                  pointsList.push_back(new Point(0,height,0));
79                  //visto que se trata do topo, a normal irá
                     ↪  apontar sempre para cima
80                  normalsList.push_back(new Point(0,1,0));
81                  normalsList.push_back(new Point(0,1,0));
82                  normalsList.push_back(new Point(0,1,0));
83                  //vertice superior do triângulo (média das
                     ↪  distâncias);
```

```
84                  texturesList.push_back(new
                    ↪  Point(texXesq,texYbaixo,0));
85                  texturesList.push_back(new
                    ↪  Point(texXdir,texYbaixo,0))
86                  texturesList.push_back(new
                    ↪  Point(texXesq+texDivX/2,texYcima,0));
87
88              }
89              else {
90                  //LADOS
91                  pointsList.push_back(new Point(x1, h1, z1));
92                  pointsList.push_back(new Point(x2, h1, z2));
93                  pointsList.push_back(new Point(x3, h2, z3));
94                  normalsList.push_back(new
                    ↪  Point(sin(alpha*i),1,cos(alpha*i)));
95                  normalsList.push_back(new
                    ↪  Point(sin(alpha*(i+1)),1,cos(alpha*(i+1))));
96                  normalsList.push_back(new
                    ↪  Point(sin(alpha*i),1,cos(alpha*i)));
97                  texturesList.push_back(new
                    ↪  Point(texXesq,texYbaixo,0));
98                  texturesList.push_back(new
                    ↪  Point(texXdir,texYbaixo,0));
99                  texturesList.push_back(new
                    ↪  Point(texXesq,texYcima,0));
100
101                 pointsList.push_back(new Point(x2, h1, z2));
102                 pointsList.push_back(new Point(x4, h2, z4));
103                 pointsList.push_back(new Point(x3, h2, z3));
104                 normalsList.push_back(new
                    ↪  Point(sin(alpha*(i+1)),1,cos(alpha*(i+1))));
105                 normalsList.push_back(new
                    ↪  Point(sin(alpha*(i+1)),1,cos(alpha*(i+1))));
106                 normalsList.push_back(new
                    ↪  Point(sin(alpha*i),1,cos(alpha*i)));
107                 texturesList.push_back(new
                    ↪  Point(texXdir,texYbaixo,0));
```

```
108                 texturesList.push_back(new
                     ↪ Point(texXdir,texYcima,0));
109                 texturesList.push_back(new
                     ↪ Point(texXesq,texYcima,0));
110             }
111         }
112
113         h1 = h2;
114         radius = radius2;
115     }
116 }
```

3.5 Torus

Mais uma vez, foi utilizada como base a figura 3, demonstrada anteriormente, para o raciocínio do desenho das texturas do torus. No entanto, só se utiliza a parte central desta, isto é, a parte dos retângulos. Além disso continua-se a utilizar as fórmulas apresentadas no *torus* para o desenvolvimento das normais.

```
1  void Vertex::makeTorus(float intRadius, float extRadius, int
   ↪  slices, int stacks){
2      double angleSlice = (2*M_PI)/stacks;
3      double angleStack = (2*M_PI)/slices;
4      double act, next, actSt, nextSt;
5      float actStR, actStZ, nextStR, nextStZ;
6
7      //variáveis que representam um "passo" na textura
8      float texX = 1.0/slices;
9      float texY = 1.0/stacks;
10
11     for(int i=0; i<stacks; i++){
12         act = angleSlice * i;
13         next = act + angleSlice;
14
15         for(int j=0; j<slices+1; j++){
16             actSt = angleStack * j;
17             actStR = intRadius * cos(actSt) + extRadius;
```

```
18              actStZ = intRadius * sin(actSt);
19
20              nextSt = (j+1) * angleStack;
21              nextStR = intRadius * cos(nextSt) + extRadius;
22              nextStZ = intRadius * sin(nextSt);
23
24              //sabendo que i representa a stack, j a slice
25              //act a divisão atual e next a divisão seguinte
26
27              pointsList.push_back(new Point(cos(act)*actStR,
                ↪  sin(act)*actStR, actStZ));
28              pointsList.push_back(new Point(cos(next)*actStR,
                ↪  sin(next)*actStR, actStZ));
29              pointsList.push_back(new Point(cos(act)*nextStR,
                ↪  sin(act)*nextStR, nextStZ));
30              normalsList.push_back(new
                ↪  Point(cos(act)*cos(actSt), sin(act)*cos(actSt),
                ↪  sin(actSt)));
31              normalsList.push_back(new
                ↪  Point(cos(next)*cos(actSt),
                ↪  sin(next)*cos(actSt), sin(actSt)));
32              normalsList.push_back(new
                ↪  Point(cos(act)*cos(actSt),
                ↪  sin(act)*cos(nextSt), sin(nextSt)));
33              texturesList.push_back(new Point(texX*i, texY*j,
                ↪  0));
34              texturesList.push_back(new Point(texX*(i+1),
                ↪  texY*j, 0));
35              texturesList.push_back(new Point(texX*i,
                ↪  texY*(j+1), 0));
36
37              pointsList.push_back(new Point(cos(act)*nextStR,
                ↪  sin(act)*nextStR, nextStZ));
38              pointsList.push_back(new Point(cos(next)*actStR,
                ↪  sin(next)*actStR, actStZ));
39              pointsList.push_back(new Point(cos(next)*nextStR,
                ↪  sin(next)*nextStR, nextStZ));
```

```
40              normalsList.push_back(new
                ↪  Point(cos(act)*cos(nextSt),
                ↪  sin(act)*cos(nextSt), sin(nextSt)));
41              normalsList.push_back(new
                ↪  Point(cos(next)*cos(actSt),
                ↪  sin(next)*cos(actSt), sin(actSt)));
42              normalsList.push_back(new
                ↪  Point(cos(next)*cos(nextSt),
                ↪  sin(next)*cos(nextSt), sin(nextSt)));
43              //triângulo complementar ao anterior
44              texturesList.push_back(new Point(texX*i,
                ↪  texY*(j+1), 0));
45              texturesList.push_back(new Point(texX*(i+1),
                ↪  texY*j, 0));
46              texturesList.push_back(new Point(texX*(i+1),
                ↪  texY*(j+1), 0));
47          }
48      }
49  }
```

3.6 Patch

Por fim, as normais es texturas do *patch* foram muito semelhantes àquelas utilizadas para o desenho dos pontos. Reutiliza-se assim o valor de *u, v, uu* e *vv*, visto que a lógica desse cálculo é análoga na determinação das texturas. Desta forma, obteve-se o seguinte código:

```
1   void Vertex::bezierPatchTriangles(int divs, vector<Patch*>
    ↪  patch_list){
2       vector<Point*> point_list;
3       float u, uu, v, vv;
4       float inc = 1.0 / divs;
5
6       for(int n_patches = 0; n_patches < patch_list.size();
        ↪  n_patches++){
7           vector<Point*> control_points =
            ↪  patch_list[n_patches]->getControlPoints();
8
```

```
9           for(int j=0; j <= divs ; j++){
10              for(int i=0; i <= divs; i++){
11                  u = i * inc;
12                  v = j * inc;
13                  uu = (i+1) * inc;
14                  vv = (j+1) * inc;
15
16                  Point* p0 = bezierPatch(u, v, control_points);
17                  Point* p1 = bezierPatch(u, vv, control_points);
18                  Point* p2 = bezierPatch(uu, v, control_points);
19                  Point* p3 = bezierPatch(uu, vv,
                     ↪  control_points);
20
21                  pointsList.push_back(p0);
22                  pointsList.push_back(p2);
23                  pointsList.push_back(p3);
24                  normalsList.push_back(p0);
25                  normalsList.push_back(p2);
26                  normalsList.push_back(p3);
27                  //pontos p0, p2 e p3 respetivamente (aplicados
                     ↪  a 2D)
28                  texturesList.push_back(new Point(u, v, 0));
29                  texturesList.push_back(new Point(uu, v, 0));
30                  texturesList.push_back(new Point(uu, vv, 0));
31
32                  pointsList.push_back(p0);
33                  pointsList.push_back(p3);
34                  pointsList.push_back(p1);
35                  normalsList.push_back(p0);
36                  normalsList.push_back(p3);
37                  normalsList.push_back(p1);
38                  //pontos p0, p3 e p1 respetivamente (aplicados
                     ↪  a 2D)
39                  texturesList.push_back(new Point(u, v, 0));
40                  texturesList.push_back(new Point(uu, vv, 0));
41                  texturesList.push_back(new Point(u, vv, 0));
42              }}}}
```

4 Engine

O ficheiro *engine*, tal como referido anteriormente, deve processar a informação de um documento XML e desenhar o seu conteúdo.

Com o auxílio do *Parser* essa ação é realizada obtendo-se assim a *Scene* completa.

```
scene = lookFiles(argv[1]);
```

Como o *engine* nesta fase deve ser capaz de desenhar as figuras atribuindo-lhes texturas e materiais e de aplicar iluminação sobre a cena este foi alterado para assim conseguir lidar com essa informação.

Desta forma, a função `renderScene` começa por percorrer as luzes da *Scene* aplicando a função `draw`, referente à classe *Light*. a cada uma destas. De seguida, percorre as estruturas e utiliza a função `figuraPrimitiva` para o desenho das figuras.

4.1 figuraPrimitiva

As alterações presentes nesta função são mínimas, mais especificamente, depois de aplicar as transformações (que já não contem *color*), elabora-se a definição do material para a figura que será desenhada. De seguida, ativa-se a imagem da textura podendo-se assim desenhar, através de *VBOs*, a figura recorrendo à função `draw` definida na classe *Struct*.

```
1  void figuraPrimitiva(Struct s){
2      (...)
3
4      s.getMaterial().draw();
5
6      glBindTexture(GL_TEXTURE_2D, s.getTexture());
7      s.draw();
8      glBindTexture(GL_TEXTURE_2D, 0);
9
10     (...)
11 }
```

4.2 Exemplos de XML

Como forma de demonstração do funcionamento do trabalho foram desenvolvidos três cenários.

4.2.1 Figuras Primitivas

Julgou-se conveniente a apresentação de dois cenário que contivessem todas as figuras desenvolvidas (`plane`, `box`, `cone`, `sphere`, `torus` e o teapot obtido através de `patches` de *Bezier*).

Os cenários são iguais diferindo que em um destes não são aplicadas texturas, materiais ou luzes para assim se poder mostrar que o progama funciona com os ficheiros de XML desenvolvidos nas outras fases e demostrar a geração da cor aleatória do material.

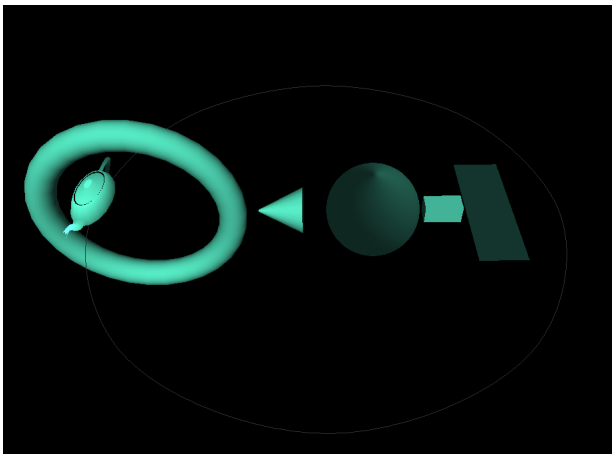

Figura 4: Figuras primitivas na terceira fase

O outro cenário possui então texturas, materiais e luzes sendo que todas as figuras nele presentes possuem diferentes tipos de materiais podendo também conter uma textura. Além disso, aplicou-se diferentes tipos de iluminação ao cenário.

O ficheiro XML que produz este cenário encontra-se em anexo. De notar a utilização de vários tipos de materiais: `diffuse`, `ambient`, `diffuse and ambient`, `emissive`, `specular` e `shininess`. Da utilização de várias iluminações: `POINT`, `SPOT` e `DIRECTIONAL`. E da utilização das várias transformações: `translate` (com e sem `time`), `scale` e `rotate` (com e sem `time`).

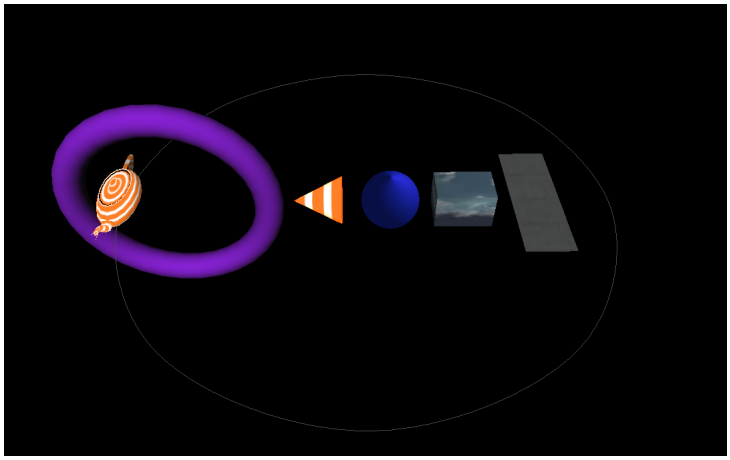

Figura 5: Figuras primitivas na quarta fase

4.2.2 Boneco de Neve

Um outro cenário desenvolvido foi o de um boneco de neve, tendo por base os conhecimentos do projeto.

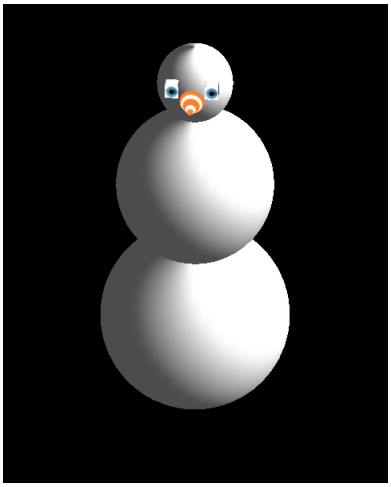

Figura 6: Boneco de Neve

O ficheiro XML responsável por esta cena encontra-se em anexo.

4.2.3 Sistema Solar

Para demonstrar o projeto em funcionamento, foi desenvolvido um Sistema Solar estático e um dinâmico, tendo por base o Sistema Solar demonstrado na fase anterior. De notar a aplicação de texturas, materiais e iluminação.

Com o XML estático elaborado obteu-se um cenário do Sistema Solar com o seguinte aspeto:

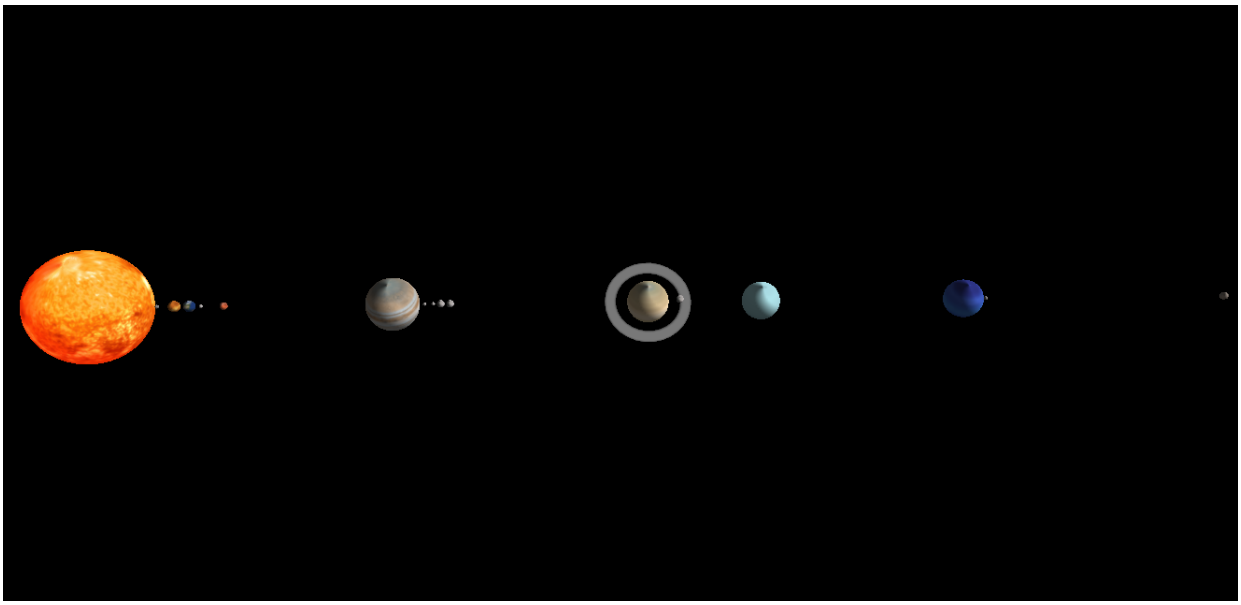

Figura 7: Sistema Solar Estático

Com o XML dinâmico elaborado, que se encontra em anexo, obteu-se um protótipo do Sistema Solar dinâmico com o seguinte aspeto:

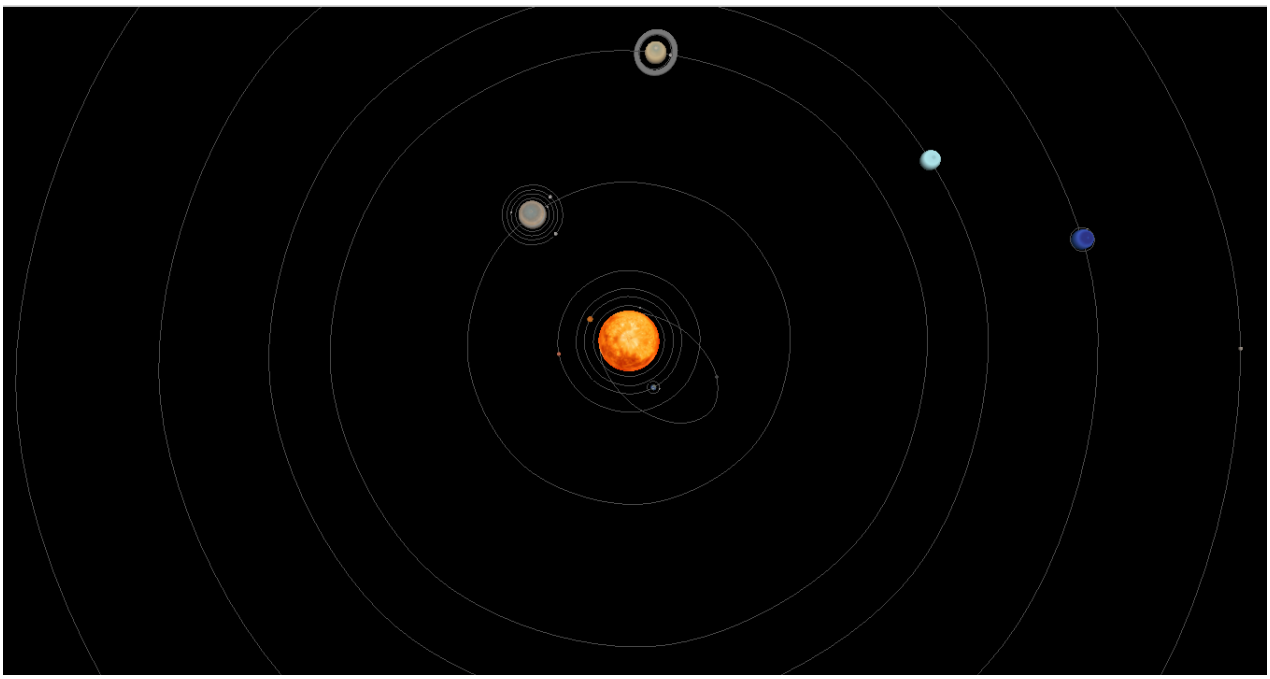

Figura 8: Sistema Solar

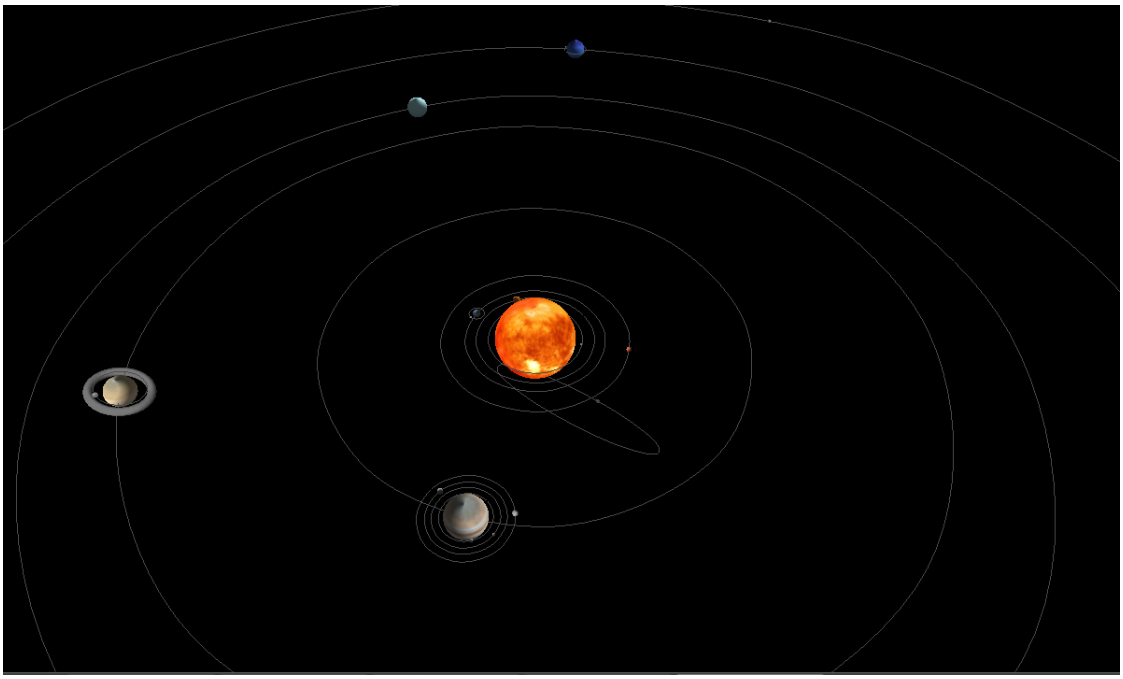

Figura 9: Sistema Solar

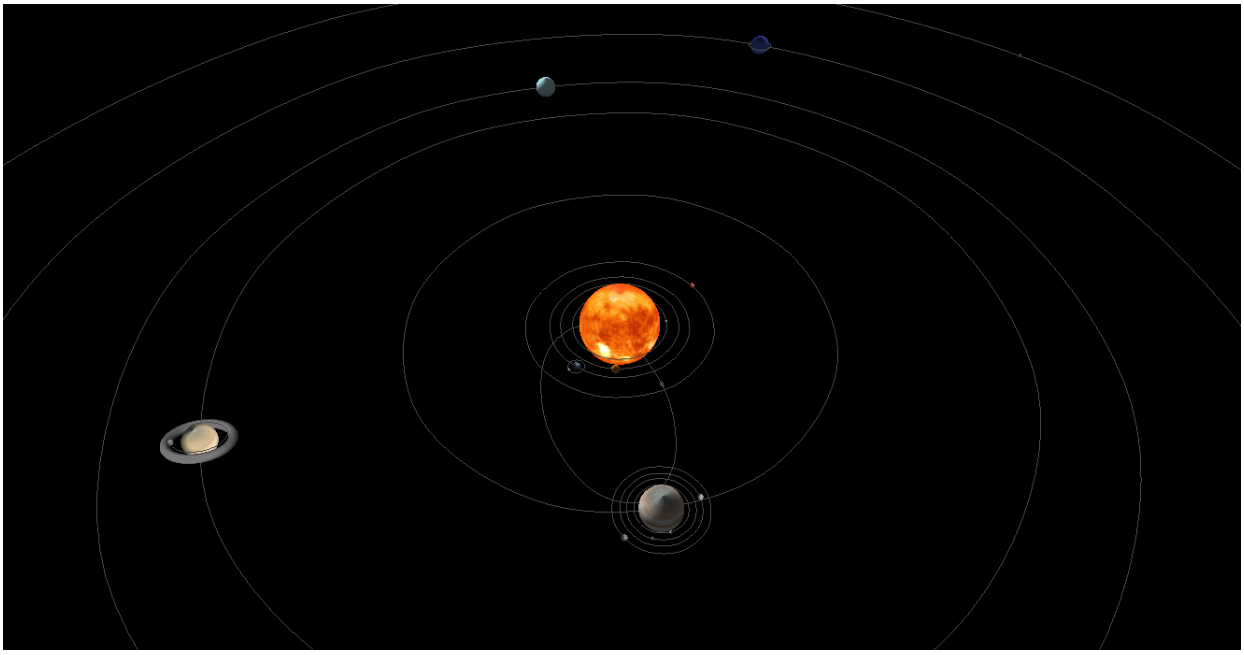

Figura 10: Sistema Solar

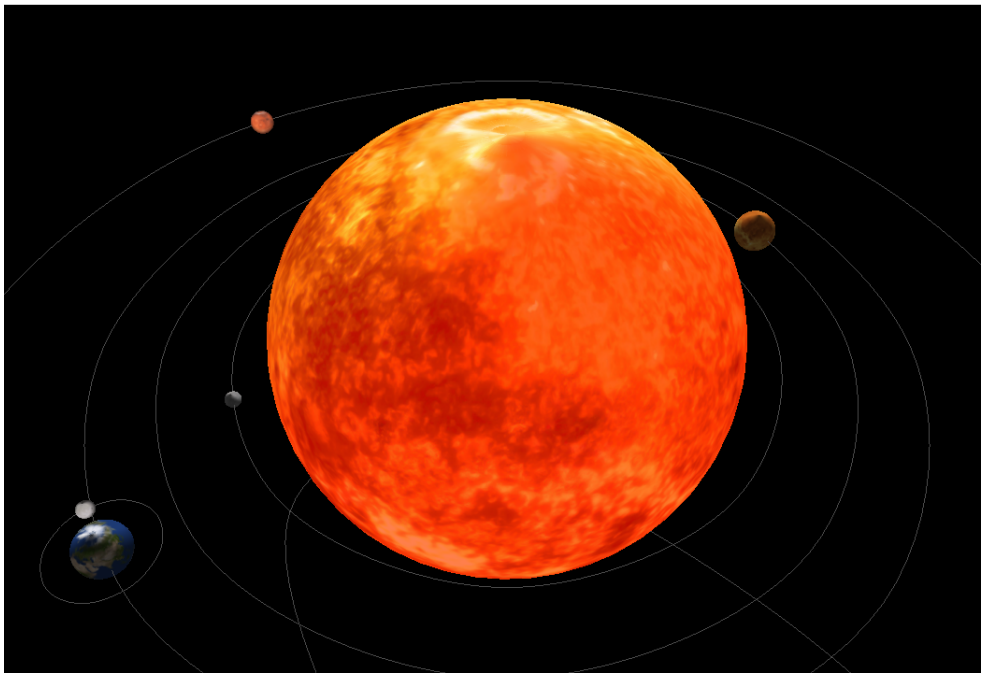

Figura 11: Planetas Rochosos e Sol

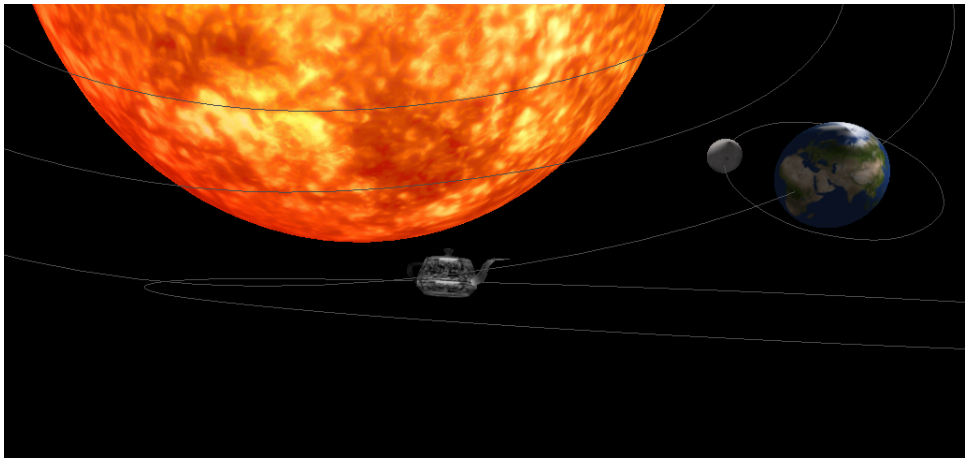

Figura 12: Cometa

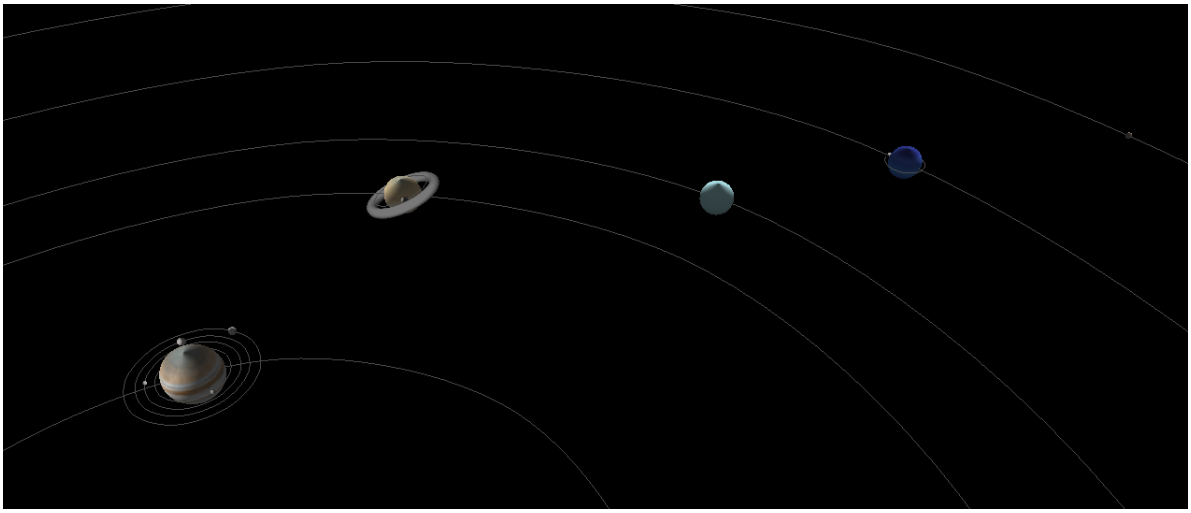

Figura 13: Planetas Gasosos

5 Conclusões

Nesta quarta fase continuou-se a desenvolver o nosso conhecimento sobre C++ e OpenGL, explorando a aplicação de texturas, materiais e iluminação.

Mais uma vez, cumpriu-se todos os requisitos propostos, apesar de todas as dificuldades encontradas, principalmente na geração dos pontos relativos às texturas e às normais de cada figura devido à geometria que acompanha tal implementação.

Com a finalização desta fase, dá-se por terminado o projeto. No entanto, ainda é possível continuar o desenvolvimento deste, podendo-se elaborar novos ficheiros XML para explorar mais cenários complexos, como o do Sistema Solar desenvolvido. Além disso, poderiam ser implementadas outras funcionalidades tal como novas formas de explorar a câmera e frustum culling.

6 Anexo

6.1 XML das figuras Primitivas

```
<scene>
    <lights>
        <light type="POINT" x="0" y="0" z="10" />
        <light type="SPOT" x="0" y="0" z="-10" />
    </lights>
    <group>
        <translate x="35" y="0" z="0" />
        <scale x="5" y="10" z="15" />
        <models>
          <model file = "plane.3d" texture="../files/plane.jpg"
          ↪ />
        </models>
    </group>
    <lights>
        <light type="DIRECTIONAL" x="0.0" y="10.0" z="0.0" />
    </lights>
    <group>
        <translate x="20" y="0" z="0" />
        <scale x="3" y="2" z="2" />
        <models>
          <model file = "box.3d" texture="../files/box.jpg"/>
        </models>
    </group>
    <group>
        <translate x="5" y="0" z="0" />
        <scale x="3" y="3" z="3" />
        <models>
          <model file = "sphere.3d" specularX="0.4"
          ↪ specularY="0.1" specularZ="0.8" shininess="2"
          ↪ diffuseANDambientX="0.1" diffuseANDambientY="0.2"
          ↪ diffuseANDambientZ="0.9"  />
        </models>
    </group>
```

```
30      <group>
31          <translate x="-5" y="0" z="0" />
32          <rotate angle="90" x="0" y="0" z="1" />
33          <models>
34            <model file = "cone.3d"
              ↪ texture="../files/coneTransito.jpg"
              ↪ emissionX="0.9" emissionY="0.3" emissionZ="0.0"
              ↪ />
35          </models>
36      </group>
37      <lights>
38          <light type="SPOT" x="10" y="0" z="0" />
39      </lights>
40      <group>
41          <translate x="-40" y="0" z="0" />
42          <scale x="1" y="1" z="1" />
43          <models>
44            <model file = "torus.3d" ambientX="0.1"
              ↪ ambientY="0.1" ambientZ="0.1" diffuseX="0.5"
              ↪ diffuseY="0.1" diffuseZ="0.8" />
45          </models>
46      </group>
47      <group>
48          <translate time="10" >
49           <point x="0" y="0" z="50" />
50           <point x="35.355" y="0" z="35.355" />
51           <point x="50" y="0" z="0" />
52           <point x="35.355" y="0" z="-35.355" />
53           <point x="0" y="0" z="-50" />
54           <point x="-35.3557" y="0" z="-35.355" />
55           <point x="-50" y="0" z="0" />
56           <point x="-35.355" y="0" z="35.355" />
57          </translate>
58          <scale x="4" y="2" z="2" />
59          <rotate angle="-90" x="1" y="0" z="0" />
60          <models>
61            <model file = "teapot.3d"
              ↪ texture="../files/coneTransito.jpg" />
```

```
        </models>
    </group>
</scene>
```

6.2 XML do Boneco de Neve

```
<scene>
    <lights>
        <light type="DIRECTIONAL" x="0" y="27" z="10" />
    </lights>
    <group>
        <models>
          <model file="bonecoCorpo1.3d"
           ↪ diffuseANDambientX="1.0" diffuseANDambientY="1.0"
           ↪ diffuseANDambientZ="1.0"/>
        </models>
    </group>
    <group>
        <translate x="0" y="17" z="0"/>
        <models>
          <model file="bonecoCorpo2.3d"
           ↪ diffuseANDambientX="1.0" diffuseANDambientY="1.0"
           ↪ diffuseANDambientZ="1.0"/>
        </models>
    </group>
    <group>
        <translate x="0" y="27" z="0"/>
        <models>
          <model file="bonecoCorpo3.3d"
           ↪ diffuseANDambientX="1.0" diffuseANDambientY="1.0"
           ↪ diffuseANDambientZ="1.0"/>
        </models>
    </group>
    <group>
        <translate x="0" y="27" z="2.75"/>
        <rotate angle="90" x="1" y="0" z="0"/>
        <scale x="0.2" y="0.2" z="0.2"/>
```

```
26          <models>
27            <model file="bonecoNariz.3d"
              ↪ texture="../files/coneTransito.jpg"/>
28          </models>
29      </group>
30      <group>
31          <translate x="1.5" y="28" z="2.75"/>
32          <models>
33            <model file="bonecoOlho.3d"
              ↪ texture="../files/boxOlho.jpg"/>
34          </models>
35      </group>
36      <group>
37          <translate x="-1.5" y="28" z="2.75"/>
38          <models>
39            <model file="bonecoOlho.3d"
              ↪ texture="../files/boxOlho.jpg"/>
40          </models>
41      </group>
42  </scene>
```

6.3 XML do Sistema Solar Dinâmico

```
1   <scene>
2       <lights>
3           <light type="POINT" x="0" y="0" z="0"/>
4       </lights>
5       <group>
6           <!--Sol-->
7           <rotate time="1" x="0" y="1" z="0" />
8           <models>
9             <model file="sol.3d" texture="../files/sun.jpg"
              ↪ emissionX="0.8" emissionY="0.2" emissionZ="0.0"
              ↪ />
10          </models>
11      </group>
12      <group>
```

```
13          <!--Cometa Halley-->
14          <rotate angle="-45" x="1" y="0" z="0" />
15          <translate x="40" y="-35" z="-40" />
16          <translate time="100">
17           <point x="0" y="0" z="70" />
18           <point x="49.497" y="0" z="49.497" />
19           <point x="70" y="0" z="0" />
20           <point x="49.497" y="0" z="-49.497" />
21           <point x="0" y="0" z="-70" />
22           <point x="-49.497" y="0" z="-49.497" />
23           <point x="-70" y="0" z="0" />
24           <point x="-49.497" y="0" z="49.497" />
25          </translate>
26          <translate x="0" y="-1" z="0" />
27          <rotate angle="270" x="1" y="0" z="0"/>
28          <models>
29            <model file="teapot.3d" texture="../files/teapot.jpg"
              ↪  diffuseX="0.8" diffuseY="0.2" diffuseZ="0.0"/>
30          </models>
31      </group>
32      <group>
33          <!--Mercurio-->
34          <translate time="10" >
35           <point x="0" y="0" z="35" />
36           <point x="24.7487" y="0" z="24.7487" />
37           <point x="35" y="0" z="0" />
38           <point x="24.7487" y="0" z="-24.7487" />
39           <point x="0" y="0" z="-35" />
40           <point x="-24.7487" y="0" z="-24.7487" />
41           <point x="-35" y="0" z="0" />
42           <point x="-24.7487" y="0" z="24.7487" />
43          </translate>
44          <rotate time="1" x="0" y="1" z="0" />
45          <models>
46            <model file="mercurio.3d"
              ↪  texture="../files/mercury.jpg" diffuseX="0.2"
              ↪  diffuseY="0.2" diffuseZ="0.2" />
47          </models>
```

```
48      </group>
49      <group>
50          <!--Venus-->
51          <translate time="20" >
52           <point x="0" y="0" z="44" />
53           <point x="31.112" y="0" z="31.112" />
54           <point x="44" y="0" z="0" />
55           <point x="31.112" y="0" z="-31.112" />
56           <point x="0" y="0" z="-44" />
57           <point x="-31.112" y="0" z="-31.112" />
58           <point x="-44" y="0" z="0" />
59           <point x="-31.112" y="0" z="31.112" />
60          </translate>
61          <rotate time="1" x="0" y="1" z="0" />
62          <models>
63            <model file="venus.3d" texture="../files/venus.jpg"
              ↪ diffuseX="0.8" diffuseY="0.5" diffuseZ="0.0" />
64          </models>
65      </group>
66      <group>
67          <!--Terra-->
68          <translate time="30" >
69           <point x="0" y="0" z="52" />
70           <point x="36.775" y="0" z="36.775" />
71           <point x="52" y="0" z="0" />
72           <point x="36.775" y="0" z="-36.775" />
73           <point x="0" y="0" z="-52" />
74           <point x="-36.775" y="0" z="-36.775" />
75           <point x="-52" y="0" z="0" />
76           <point x="-36.775" y="0" z="36.775" />
77          </translate>
78          <rotate time="1" x="0" y="1" z="0" />
79          <models>
80            <model file="terra.3d" texture="../files/mercury.jpg"
              ↪ diffuseX="0.0" diffuseY="0.0" diffuseZ="0.9" />
81          </models>
82          <group>
83              <translate time="20" >
```

```
84              <point x="0" y="0" z="6" />
85              <point x="4.242" y="0" z="4.242" />
86              <point x="6" y="0" z="0" />
87              <point x="4.242" y="0" z="-4.242" />
88              <point x="0" y="0" z="-6" />
89              <point x="-4.242" y="0" z="-4.242" />
90              <point x="-6" y="0" z="0" />
91              <point x="-4.242" y="0" z="4.242" />
92            </translate>
93            <rotate time="1" x="0" y="1" z="0" />
94            <models>
95              <model file="lua.3d" texture="../files/moon.jpg"
                ↪  diffuseX="0.5" diffuseY="0.8" diffuseZ="0.8"
                ↪  />
96            </models>
97        </group>
98    </group>
99    <group>
100       <!--Marte-->
101       <translate time="40" >
102        <point x="0" y="0" z="70" />
103        <point x="49.497" y="0" z="49.497" />
104        <point x="70" y="0" z="0" />
105        <point x="49.497" y="0" z="-49.497" />
106        <point x="0" y="0" z="-70" />
107        <point x="-49.497" y="0" z="-49.497" />
108        <point x="-70" y="0" z="0" />
109        <point x="-49.497" y="0" z="49.497" />
110       </translate>
111       <rotate time="1" x="0" y="1" z="0" />
112       <models>
113         <model file="marte.3d" texture="../files/mars.jpg"
           ↪  diffuseX="1.0" diffuseY="0.0" diffuseZ="0.0" />
114       </models>
115   </group>
116   <group>
117       <!--Jupiter-->
118       <translate time="50" >
```

```
119             <point x="0" y="0" z="159" />
120             <point x="112.43" y="0" z="112.43" />
121             <point x="159" y="0" z="0" />
122             <point x="112.43" y="0" z="-112.437" />
123             <point x="0" y="0" z="-159" />
124             <point x="-112.43" y="0" z="-112.43" />
125             <point x="-159" y="0" z="0" />
126             <point x="-112.43" y="0" z="112.43" />
127           </translate>
128           <rotate time="1" x="0" y="1" z="0" />
129           <models>
130             <model file="jupiter.3d"
                  ↪ texture="../files/jupiter.jpg" diffuseX="0.8"
                  ↪ diffuseY="0.5" diffuseZ="0.2" />
131           </models>
132           <group>
133               <translate time="15" >
134                <point x="0" y="0" z="16.8" />
135                <point x="11.879" y="0" z="11.879" />
136                <point x="16.8" y="0" z="0" />
137                <point x="11.879" y="0" z="-11.879" />
138                <point x="0" y="0" z="-16.8" />
139                <point x="-11.879" y="0" z="-11.879" />
140                <point x="-16.8" y="0" z="0" />
141                <point x="-11.879" y="0" z="11.879" />
142               </translate>
143               <rotate time="1" x="0" y="1" z="0" />
144               <models>
145                 <model file="io.3d" texture="../files/moon.jpg"
                      ↪ diffuseX="0.5" diffuseY="0.8" diffuseZ="0.8"
                      ↪ />
146               </models>
147           </group>
148           <group>
149               <translate time="20" >
150                <point x="0" y="0" z="21.3" />
151                <point x="15.061" y="0" z="15.061" />
152                <point x="21.3" y="0" z="0" />
```

```
153             <point x="15.061" y="0" z="-15.061" />
154             <point x="0" y="0" z="-21.3" />
155             <point x="-15.061" y="0" z="-15.061" />
156             <point x="-21.3" y="0" z="0" />
157             <point x="-15.061" y="0" z="15.061" />
158           </translate>
159           <rotate time="1" x="0" y="1" z="0" />
160           <models>
161             <model file="europa.3d"
                  ↪ texture="../files/moon.jpg" diffuseX="0.5"
                  ↪ diffuseY="0.8" diffuseZ="0.8" />
162           </models>
163       </group>
164       <group>
165           <translate time="25" >
166             <point x="0" y="0" z="25.3" />
167             <point x="17.889" y="0" z="17.889" />
168             <point x="25.3" y="0" z="0" />
169             <point x="17.889" y="0" z="-17.889" />
170             <point x="0" y="0" z="-25.3" />
171             <point x="-17.8891" y="0" z="-17.889" />
172             <point x="-25.3" y="0" z="0" />
173             <point x="-17.889" y="0" z="17.889" />
174           </translate>
175           <rotate time="1" x="0" y="1" z="0" />
176           <models>
177             <model file="ganymede.3d"
                  ↪ texture="../files/moon.jpg" diffuseX="0.5"
                  ↪ diffuseY="0.8" diffuseZ="0.8" />
178           </models>
179       </group>
180       <group>
181           <translate time="30" >
182             <point x="0" y="0" z="30.3" />
183             <point x="21.425" y="0" z="21.425" />
184             <point x="30.3" y="0" z="0" />
185             <point x="21.425" y="0" z="-21.425" />
186             <point x="0" y="0" z="-30.3" />
```

```
187              <point x="-21.425" y="0" z="-21.425" />
188              <point x="-30.3" y="0" z="0" />
189              <point x="-21.425" y="0" z="21.425" />
190            </translate>
191            <rotate time="1" x="0" y="1" z="0" />
192            <models>
193              <model file="callisto.3d"
                 ↪ texture="../files/moon.jpg" diffuseX="0.5"
                 ↪ diffuseY="0.8" diffuseZ="0.8" />
194            </models>
195        </group>
196    </group>
197    <group>
198        <!--Saturno-->
199        <translate time="60" >
200         <point x="0" y="0" z="294" />
201         <point x="207.889" y="0" z="207.889" />
202         <point x="294" y="0" z="0" />
203         <point x="207.889" y="0" z="-207.889" />
204         <point x="0" y="0" z="-294" />
205         <point x="-207.889" y="0" z="-207.889" />
206         <point x="-294" y="0" z="0" />
207         <point x="-207.889" y="0" z="207.889" />
208        </translate>
209        <rotate time="1" x="0" y="1" z="0" />
210        <models>
211          <model file="saturno.3d"
             ↪ texture="../files/saturn.jpg" diffuseX="0.8"
             ↪ diffuseY="0.6" diffuseZ="0.4" />
212        </models>
213        <group>
214            <translate x="0" y="-3" z="0" />
215            <rotate angle="-70" x="1" y="0" z="0" />
216            <models>
217              <model file="anel.3d" texture="../files/ring.jpg"
                 ↪ diffuseX="0.8" diffuseY="0.6" diffuseZ="0.0"
                 ↪ />
218            </models>
```

```
219         </group>
220         <group>
221             <rotate angle="-70" x="1" y="0" z="0" />
222             <rotate angle="90" x="1" y="0" z="0" />
223             <translate time="15" >
224              <point x="0" y="3" z="16.8" />
225              <point x="11.879" y="3" z="11.879" />
226              <point x="16.8" y="3" z="0" />
227              <point x="11.879" y="3" z="-11.879" />
228              <point x="0" y="3" z="-16.8" />
229              <point x="-11.879" y="3" z="-11.879" />
230              <point x="-16.8" y="3" z="0" />
231              <point x="-11.879" y="3" z="11.879" />
232             </translate>
233             <rotate time="1" x="0" y="1" z="0" />
234             <models>
235               <model file="titan.3d"
                  ↪  texture="../files/moon.jpg" diffuseX="0.5"
                  ↪  diffuseY="0.8" diffuseZ="0.8" />
236             </models>
237         </group>
238     </group>
239     <group>
240         <!--Urano-->
241         <translate time="80" >
242          <point x="0" y="0" z="354" />
243          <point x="250.316" y="0" z="250.316" />
244          <point x="354" y="0" z="0" />
245          <point x="250.316" y="0" z="-250.316" />
246          <point x="0" y="0" z="-354" />
247          <point x="-250.316" y="0" z="-250.316" />
248          <point x="-354" y="0" z="0" />
249          <point x="-250.316" y="0" z="250.316" />
250         </translate>
251         <rotate time="1" x="0" y="1" z="0" />
252         <models>
253           <model file="urano.3d" texture="../files/uranus.jpg"
              ↪  diffuseX="0.5" diffuseY="0.5" diffuseZ="1.0" />
```

```
254         </models>
255     </group>
256     <group>
257         <!--Neptuno-->
258         <translate time="90" >
259          <point x="0" y="0" z="462" />
260          <point x="326.683" y="0" z="326.683" />
261          <point x="462" y="0" z="0" />
262          <point x="326.683" y="0" z="-326.683" />
263          <point x="0" y="0" z="-462" />
264          <point x="-326.683" y="0" z="-326.683" />
265          <point x="-462" y="0" z="0" />
266          <point x="-326.683" y="0" z="326.683" />
267         </translate>
268         <rotate time="1" x="0" y="1" z="0" />
269         <models>
270           <model file="neptuno.3d"
              ↪ texture="../files/neptune.jpg" diffuseX="0.2"
              ↪ diffuseY="0.2" diffuseZ="1.0" />
271         </models>
272         <group>
273             <translate time="20" >
274              <point x="0" y="0" z="12" />
275              <point x="8.485" y="0" z="8.485" />
276              <point x="12" y="0" z="0" />
277              <point x="8.485" y="0" z="-8.485" />
278              <point x="0" y="0" z="-12" />
279              <point x="-8.485"   y="0" z="-8.485" />
280              <point x="-12" y="0" z="0" />
281              <point x="-8.485" y="0" z="8.485" />
282             </translate>
283             <rotate time="1" x="0" y="1" z="0" />
284             <models>
285               <model file="triton.3d"
                  ↪ texture="../files/moon.jpg" diffuseX="0.5"
                  ↪ diffuseY="0.8" diffuseZ="0.8" />
286             </models>
287         </group>
```

```
288     </group>
289     <group>
290         <!--Plutão-->
291         <translate time="100" >
292          <point x="0" y="0" z="602" />
293          <point x="425.678" y="0" z="425.678" />
294          <point x="602" y="0" z="0" />
295          <point x="425.678" y="0" z="-425.678" />
296          <point x="0" y="0" z="-602" />
297          <point x="-425.678" y="0" z="-425.678" />
298          <point x="-602" y="0" z="0" />
299          <point x="-425.678" y="0" z="425.678" />
300         </translate>
301         <rotate time="1" x="0" y="1" z="0" />
302         <models>
303           <model file="plutao.3d" texture="../files/pluto.jpg"
              ↪ fdiffuseX="0.7 diffuseY="0.7" diffuseZ="0.7" />
304         </models>
305     </group>
306 </scene>
```