Computação Gráfica

VBOs, superfícies cúbicas e curvas

Relatório de Desenvolvimento

João Manuel Martins Cerqueira (A65432)
Sónia Catarina Guerra Costa (A71506)
Tiago Costa Loureiro (A71191)

Ano letivo 2016/2017

Conteúdo

Capítulo 1

Introdução

Este trabalho prático incide no desenvolvimento de uma representação do sistema solar com VBOs, através de modelos dispostos hierarquicamente, compostas por figuras e transformações geométricas. Pretende-se também que leia um ficheiro do tipo patch, o qual contém informação sobre uma figura desenhada através de superfícies de Bezier. Além disso deverá desenhar curvas através da função Catmull Rom. Pretende-se que o programa leia a partir de um ficheiro XML as transformações geométricas e as figuras, guarde para as estruturas a informação e os nomes dos ficheiros onde os pontos necessários para construir os objetos estão guardados e construa os mesmos.

1.1 Estrutura do documento

A estrutura que este relatório segue, excluindo o presente capítulo onde se faz uma pequena introdução do assunto, é:

- No capítulo 2 faz-se uma análise detalhada do problema proposto especificando-se os parâmetros de entrada do programa e os resultados obtidos.
- No capítulo 3 faz-se referencia à Conceção/Desenho da Resolução dos problemas propostos, mostrando assim as estruturas de dados e os algoritmos usados durante a realização deste trabalho.
- No capítulo 4 faz-se referência a decisões e problemas de implementação que surgiram assim como os passos para executar o programa.
- No capítulo 5 faz-se uma conclusão/síntese do trabalho realizado e uma análise crítica dos resultados.
- No capítulo 6 são indicadas as fontes usadas na realização do trabalho.
- Por último, em apêndice encontra-se o código necessário para a implementação do programa.

Capítulo 2

Análise e Especificação

2.1 Especificação do Problema

Neste trabalho pretende-se desenvolver o projecto da fase anterior de forma a que os objectos sejam agora lidos para a memória da gráfica e tratados como VBOs. Estes serão mais eficientes do que do que a forma da qual estavamos a desenhar até agora. Também passamos a ler ficheiros do tipo patch para formas geométricas mais complexas, tais como o teapot que seria difícil de desenhar usando apenas as formas geométricas que tinhamos até agora. Foi então necessário implementar este novo tipo de ficheiro, o qual gera os pontos da figura através das curvas de Bezier. Também precisamos de implementar a função Catmull Rom para que seja possível dar animação aos planetas de forma a que se movam na sua trajétoria.

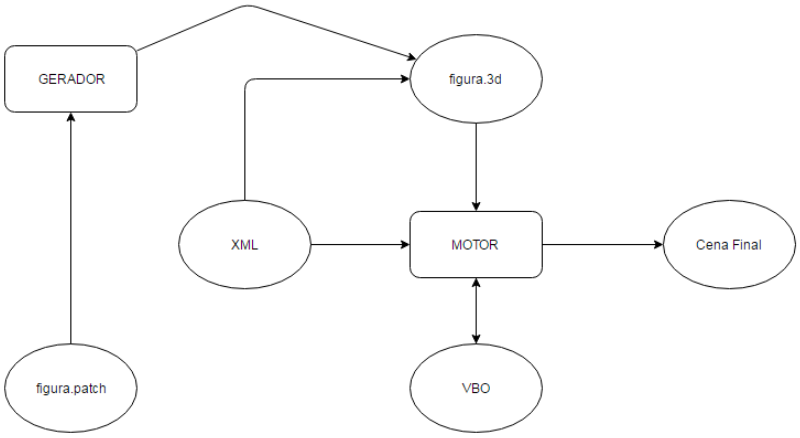

Figura 2.1: Diagrama do trabalho

Capítulo 3

Conceção/Desenho da Resolução

3.1 Leitura do ficheiro .patch

Sabemos que um ficheiro .patch tem na primeira linha o número total de patches, seguido das patches com os seus índices (16 por patch) e depois o número de pontos de controlo (seguido dos pontos de controlo, 1 por cada linha). Temos então de ler o ficheiro tendo em conta este formato, lendo primeiro o número de patches e alocando a meória necessária para as ler todas e depois lemos o número de pontos de controlo e repetimos o processo para estes. Após termos toda a informação do ficheiro na memória passamos para a função que irá gerar os pontos da figura de acordo com o nível de tesselação calculando as curvas de Bezier da figura.

```
float *pontosPatch(int *patches, int n_patches, float *control_points, int
     n_control_points, int nivel) {
float coordenadas[48], peso = 1.0 / nivel, *pontos = (float*)malloc(n_patches*(3 *
     (nivel + 1) ^ 2) * sizeof(float)), a, b, c, d; int k = 0;
for (int t = 0; t < n_patches; t++) {
    a = 0.0; b = 1.0 - a; c = 0.0; d = 1.0 - c;
    /**Excerto dos clculos efectuados para cada ponto*/
    coordenadas[0] = control_points[3 * patches[16 * t]];
    coordenadas[1] = control_points[3 * patches[16 * t] + 1];
    coordenadas[2] = control_points[3 * patches[16 * t] + 2];//(...)
    /** Excerto do clculo de um dos pontos */
    for (int i = 0; i < nivel; i++) {
        for (int j = 0; j < nivel; j++) {
            pontos[k++] = coordenadas[0] * a*a*a*c*c*c + coordenadas[3] * 3 * a*a*a*c*c*d
                 + coordenadas[6] * 3 * a*a*a*c*d*d + coordenadas[9] * a*a*a*d*d*d
                + coordenadas[12] * 3 * a*a*b*c*c*c + coordenadas[15] * 9 * a*a*b*c*c*d +
                     coordenadas[18] * 9 * a*a*b*c*d*d + coordenadas[21] * 3 * a*a*b*d*d*d
                + coordenadas[24] * 3 * a*b*b*c*c*c + coordenadas[27] * 9 * a*b*b*c*c*d +
                     coordenadas[30] * 9 * a*b*b*c*d*d + coordenadas[33] * 3 * a*b*b*d*d*d
                + coordenadas[36] * b*b*b*c*c*c + coordenadas[39] * 3 * b*b*b*c*c*d +
                     coordenadas[42] * 3 * b*b*b*c*d*d + coordenadas[45] * b*b*b*d*d*d;
```

Listing 3.1: função que gera os pontos da superfície de Bezier

3.2 Leitura das figuras .3d para VBOs

Na etapa anterior fizemos a leitura das figuras de forma a que ao passarmos para VBOs a transição fosse mais fácil. Ou seja, decidimos guardar a informação para vectores, coordenada a coordenada, como se estivessemos a trabalhar com o vertexBuffer como nas aulas. A leitura da informação sofreu poucas alterações graças a essa decisão. Precisamos contudo de criar os buffers que guardam as figuras e alterar a forma como desenhamos. Decidimos então usar a seguinte estrutura.

```
vector< tuple <int, int> > VBO; //numero de pontos, indice do VBO
GLuint *buffers = NULL; // buffers para os VBOs
// auxiliar para apontar para o prximo buffer disponivel;
int next_buffer = 0;
```

Listing 3.2: Estrutura com buffers dos VBOs e numero de pontos da figura

Tendo isto em conta, quando o nosso motor é inicializado começa por ler o ficheiro xml e guarda para os vectores respectivos as transformações efectuadas e as figuras que pretendemos que sejam desenhadas. Após lermos o ficheiro, executamos a função desenha, a qual gera os buffers necessários (visto que como já temos a lista de ficheiros temos então o número de figuras que vamos desenhar). Depois de gerar os buffers, lemos as figuras uma a uma e guardamos cada uma das suas coordenadas para um array vertexB. O passo seguinte é passar a informação da figura para o seu buffer respectivo. Este processo é efectuado até que todas as figuras sejam carregadas para um buffer. A partir deste ponto, e com toda a informação carregada para os buffers na gráfica e para a meória do PC, a função renderScene executa a função desenha2 que fica responsável desenhar as figuras a partir dos buffers e das estruturas para as transformações.

```
for (int i = 0; i <= lista_ficheiros.size()-1; ++i) {
       const char *f = lista_ficheiros[i].c_str();
       ifstream fi(f);

       if (fi.is_open()) {
           while (getline(fi, str)) { //ler todos os vertices
           //a primeira linha contem o numero de vertices, calcular espao para o vertexB;
               if (primeira_linha == 0) {
                   istringstream ss(str);
                   ss >> n_triangulos; // guardar numero de triangulos
                   n_pontos = 3 * n_triangulos; // calcular numero de pontos
                   vertexB = (float*)malloc(3 * n_pontos * sizeof(float));
                   k = 0; // reinicializar iterador do vertexB;
                   primeira_linha = 1;
               }
               else {
                   istringstream ss(str); //carregar vrtices para o vertexB;
                   ss >> vertexB[k++];
                   ss >> vertexB[k++];
                   ss >> vertexB[k++];
               }
           }
           fi.close();
```

```
            //selecionar o buffer;
            glBindBuffer(GL_ARRAY_BUFFER, buffers[next_buffer]);
            // preencher o buffer com os dados da figura;
            glBufferData(GL_ARRAY_BUFFER, n_pontos * 3 * sizeof(float), vertexB,
                 GL_STATIC_DRAW);
            // vector que guarda o numero de pontos e qual o buffer associado;
            VBO.push_back(tuple<int, int>(n_pontos, next_buffer));
            next_buffer++;// avanar para o prximo buffer;
            ::free(vertexB); // libertar o array com os dados da figura;
}
```

Listing 3.3: Leitura dos ficheiros .3d para VBOs

3.3 Desenvolvimento das figuras

Cada figura é gerada a partir da construção sucessiva de triângulos cujos vértices foram préviamente guardados em VBOs. Primeiro, foram atribuídas as cores referentes a cada astro e em seguida aplicadas as respetivas transformações, percorrendo os vetores correspondentes a cada informação, dando de seguida ínicio à construção dos VBOs a partir dos buffers. Para que estas alterações não se acumulem, é feito um $glPushMatrix()$ antes e $glPopMatrix()$ depois de cada figura. A representação do sistema solar pode ser vista no modo $fill$, $point$ e $line$, para isso bastando clicar na imagem com o botão direito do rato para mudar o modo.

```
   for (int i = 0, k = 0; i <= lista_ficheiros.size() - 1; ++i) {
       const char *f = lista_ficheiros[i].c_str();
       /** desenhar objeto */
       definir_cores();
       glPushMatrix();
       /**desenhar cores e transformaes*/
       glColor3f(get<0>(lista_cores[i]), get<1>(lista_cores[i]), get<2>(lista_cores[i]));
        get<2>(lista_translacoes[i]) << endl;
       glTranslatef(get<0>(lista_translacoes[i]), get<1>(lista_translacoes[i]),
            get<2>(lista_translacoes[i]));
       glRotatef(lista_angulos[i], get<0>(lista_rotacoes[i]), get<1>(lista_rotacoes[i]),
            get<2>(lista_rotacoes[i]));
       glScalef(get<0>(lista_escalas[i]), get<1>(lista_escalas[i]),
            get<2>(lista_escalas[i]));
       /**desenhar VBO*/
       //glBindBuffer(GL_ARRAY_BUFFER, buffers[<numero>]);
       glBindBuffer(GL_ARRAY_BUFFER, buffers[i]); //buffers[get<1>(VBO[k])]
       glVertexPointer(3, GL_FLOAT, 0, 0);
       //glDrawArrays(GL_TRIANGLES, first, count); count  o total de vertices.
       glDrawArrays(GL_TRIANGLES, 0, get<0>(VBO[k++])*3);
       glPopMatrix();
   }
```

Listing 3.4: Consctrução das figuras

Capítulo 4

Conclusão

Como previamos, a implementação de VBOs aumentou a performance do motor devido a guardarmos os dados de cada uma das figuras na meória da gráfica, sendo assim mais rápido a desenhar os objectos. Também apercebemo-nos que certas figuras não seriam possíveis de ser desenhadas sem recorrermos a algo mais complexo como curvas e superfícies de Bezier, daí a necessidade de implementar este tipo de ficheiro no gerador.Infelizmente não conseguimos implementar a função Catmull Rom para animar os planetas ao longo das suas órbitas. Esperamos que na próxima etapa tenhamos implementado esta funcionalidade, além das texturas.

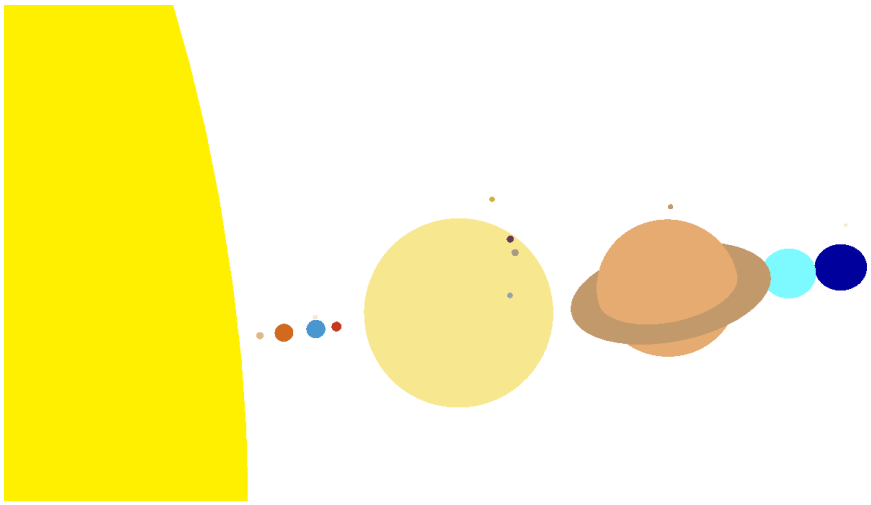

Figura 4.1: Representação do Sistema Solar

Capítulo 5

Referências

- https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/
- https://www.gamedev.net/resources/_/technical/graphics-programming-and-theory/bezier-curves-and-surfaces-r1808
- http://www.cplusplus.com/doc/tutorial/files/
- https://elearning.uminho.pt/

Apêndice A

Apêndice

A.1 Código do Motor

```
// este include tem de ser feito antes do glut caso contrario d "exit redefenition" (no
     V.Studio).
#include "tinyxml/tinyxml.h"
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glew.h>
#include <GL/glut.h>
#endif

#define _USE_MATH_DEFINES
#include <math.h>
#include <vector>

#include <iostream>
#include <fstream>
#include <sstream>
#include <cstring>
#include <string>
#include <tuple>

// para no estar sempre a escrever std::
using namespace std;

//Estrutura para guardar 3 floats
typedef vector< tuple<float, float, float> > vector3f;

// Vector que guarda a lista de ficheiros
vector3f lista_translacoes;
vector3f lista_escalas;
vector3f lista_rotacoes;
vector<float> lista_angulos;
vector<int> lista_times;

// Vector que guarda o nome de todos os ficheiros
vector<string> lista_ficheiros;

// Vector que guarda a lista das cores (1 para cada figura)
vector3f lista_cores;

//variaveis de transformacoes usadas ao ler o XML
```

```
float translate_x = 0, translate_y = 0, translate_z = 0,
scale_x = 1, scale_y = 1, scale_z = 1,
angulo = 0, rotate_x = 0, rotate_y = 0, rotate_z = 0;
/* Ainda no    usado
#define EXP 0
#define FPS 1
*/

// flag para mudar o drwing mode
int flag_drawing_mode = 1;

// ngulos para "rodar a camera"
float alfa = 0.0f, beta = 0.0f, radius = 500;
float camX = 0.0f, camY = 0.0f, camZ = 0.0f;

/**VBOs*/
// o primeiro int  o numero de pontos e o segundo  o indice do buffer;
vector< tuple <int, int> > VBO; //numero de pontos, indice do VBO
vector<tuple <int, GLuint> > VBO2; // <tuple <n_pontos, buffer> >

// buffers para os VBOs
GLuint *buffers = NULL;
// auxiliar para apontar para o prximo buffer disponivel;
int next_buffer = 0;
// frames per second
int frame = 0, fps = 0, timebase, times;
char print[20] = "";

/* Ainda no    usado
float dx = 0.0f;
float dy = 0.0f;
float dz = 0.0f;
int modo_camera = 0;
*/

/* Esta funcao vai buscar os nomes dos ficheiros .3d que esto no vector lista_ficheiros
* Desenha todos os pontos de cada ficheiro e por ficheiro atribui uma cor do vector
     lista_cores
*/

#define POINT_COUNT 5
vector<vector3f> lista_pontos_translacao; // vector com lista de todos os pontos da curva
     catmull
//int time_t = 0;
// funcoes catmull-rom da aula;
void getCatmullRomPoint(float t, int *indices, float *res) {

    int i, j, k;
    float aux[4];
    float aux_t[4];
    aux_t[0] = t*t*t; aux_t[1] = t*t; aux_t[2] = t; aux_t[3] = 1;

    // catmull-rom matrix
    float m[4][4] = { { -0.5f, 1.5f, -1.5f, 0.5f },
    { 1.0f, -2.5f, 2.0f, -0.5f },
    { -0.5f, 0.0f, 0.5f, 0.0f },
    { 0.0f, 1.0f, 0.0f, 0.0f } };

    res[0] = 0.0; res[1] = 0.0; res[2] = 0.0;
    // Compute point res = T * M * P
    // where Pi = p[indices[i]]
    for (int i = 0; i < 3; i++) {
        for (j = 0; j < 4; j++) {
            aux[j] = 0;
            for (k = 0; k < 4; k++) {
                aux[i] += (aux_t[k] * m[k][j]);
            }
        }
```

```
        /**faltam calculos aqui*/
    }
}

// given global t, returns the point in the curve
void getGlobalCatmullRomPoint(float gt, float *res) {

    float t = gt * POINT_COUNT; // this is the real global t
    int index = floor(t); // which segment
    t = t - index; // where within the segment

    // indices store the points
    int indices[4];
    indices[0] = (index + POINT_COUNT - 1) % POINT_COUNT;
    indices[1] = (indices[0] + 1) % POINT_COUNT;
    indices[2] = (indices[1] + 1) % POINT_COUNT;
    indices[3] = (indices[2] + 1) % POINT_COUNT;

    getCatmullRomPoint(t, indices, res);
}

void definir_cores() {
    lista_cores.push_back(tuple<float, float, float>(1.0, 0.94, 0.0)); //sol
    lista_cores.push_back(tuple<float, float, float>(0.87, 0.72, 0.53)); //mercurio
    lista_cores.push_back(tuple<float, float, float>(0.82, 0.41, 0.12)); //venus
    lista_cores.push_back(tuple<float, float, float>(0.29, 0.59, 0.82)); //terra
    lista_cores.push_back(tuple<float, float, float>(0.97, 0.91, 0.81)); //lua
    lista_cores.push_back(tuple<float, float, float>(0.76, 0.23, 0.13)); //marte
    lista_cores.push_back(tuple<float, float, float>(0.97, 0.91, 0.56)); //jupiter
    lista_cores.push_back(tuple<float, float, float>(0.57, 0.64, 0.69)); //lua
    lista_cores.push_back(tuple<float, float, float>(0.83, 0.69, 0.22)); //lua
    lista_cores.push_back(tuple<float, float, float>(0.66, 0.6, 0.53)); //lua
    lista_cores.push_back(tuple<float, float, float>(0.4, 0.22, 0.33)); //lua
    lista_cores.push_back(tuple<float, float, float>(0.9, 0.67, 0.44)); //saturno
    lista_cores.push_back(tuple<float, float, float>(0.76, 0.6, 0.42)); //lua
    lista_cores.push_back(tuple<float, float, float>(0.76, 0.6, 0.42)); //lua
    lista_cores.push_back(tuple<float, float, float>(0.49, 0.98, 1.0)); //urano
    lista_cores.push_back(tuple<float, float, float>(0.0, 0.0, 0.61)); //neptuno
    lista_cores.push_back(tuple<float, float, float>(0.97, 0.91, 0.81)); //lua
}

// esta funcao apenas  invocada na main para ler os ficheiros com os objectos para a
     memoria do PC.
void desenha() {
    /*
    * Variaveis
    */
    vector3f vertices; //vector< tuple<float, float, float> >
    float v1 = 0, v2 = 0, v3 = 0;
    string str;
    int primeira_linha = 0, k = 0, n_triangulos = 0, n_pontos = 0, n_vertices;
    float *vertexB = NULL; // array para os vrtices;

    /** percorrer lista com o nome dos ficheiros*/

    glEnableClientState(GL_VERTEX_ARRAY); // Enable buffer functionality;
    // Generate buffers;
    buffers = (GLuint*)malloc(sizeof(GLuint)*lista_ficheiros.size());
    // glGenBuffers(n_buffers, buffer_pointer);
    glGenBuffers(lista_ficheiros.size(), buffers);

    for (int i = 0; i <= lista_ficheiros.size()-1; ++i) {
        printf("%d ", i);
        const char *f = lista_ficheiros[i].c_str();
        //DEBUG cout << lista_ficheiros[i] << i << endl;
        ifstream fi(f);
```

```
        if (fi.is_open()) {
            while (getline(fi, str)) { //ler todos os vertices
                                    //a primeira linha contem o numero de vertices, calcular
                                        espao para o array vertexB;
                if (primeira_linha == 0) {
                    istringstream ss(str);
                    ss >> n_triangulos; // guardar numero de triangulos
                    n_pontos = 3 * n_triangulos; // calcular numero de pontos
                    vertexB = (float*)malloc(3 * n_pontos * sizeof(float));
                    k = 0; // reinicializar iterador do vertexB;
                    primeira_linha = 1;
                }
                else {
                    istringstream ss(str); //carregar cada um dos vrtices para o vertexB;
                    ss >> vertexB[k++];
                    ss >> vertexB[k++];
                    ss >> vertexB[k++];
                }
            }
            fi.close();
            // gerar o buffer;
            //glGenBuffers(1, &buffers[next_buffer]);
            // fazer bind do buffer;
            glBindBuffer(GL_ARRAY_BUFFER, buffers[next_buffer]);
            // preencher o buffer com os dados da figura;
            glBufferData(GL_ARRAY_BUFFER, n_pontos * 3 * sizeof(float), vertexB,
                 GL_STATIC_DRAW);
            // vector que guarda o numero de pontos e qual o buffer associado;
            VBO.push_back(tuple<int, int>(n_pontos, next_buffer));
            VBO2.push_back(tuple<int, GLuint>(n_pontos, buffers[next_buffer]));
            //  avanar  para o prximo buffer;
            next_buffer++;
            // libertar o array com os dados da figura, visto que j est no buffer da grfica;
            ::free(vertexB);
            primeira_linha = 0;
        }
        else {
            cerr << "Erro: No foi possvel abrir o ficheiro " << lista_ficheiros[i] << "." <<
                 endl;
            exit(1);
        }

        /** clear vector for next file*/
        vertices.clear();
    }
    printf("\n");
}

// invocada na renderScene porque puxa menos pelo pc e tenho de desenhar o VBO a seguir a
     transformao;
void desenha2() {
    for (int i = 0, k = 0; i <= lista_ficheiros.size() - 1; ++i) {
        const char *f = lista_ficheiros[i].c_str();
        /** desenhar objeto */
        definir_cores();
        glPushMatrix();

        glColor3f(get<0>(lista_cores[i]), get<1>(lista_cores[i]), get<2>(lista_cores[i]));

        glTranslatef(get<0>(lista_translacoes[i]), get<1>(lista_translacoes[i]),
             get<2>(lista_translacoes[i]));
        glRotatef(lista_angulos[i], get<0>(lista_rotacoes[i]), get<1>(lista_rotacoes[i]),
             get<2>(lista_rotacoes[i]));
        glScalef(get<0>(lista_escalas[i]), get<1>(lista_escalas[i]),
             get<2>(lista_escalas[i]));

        //drawVBO();
        //glBindBuffer(GL_ARRAY_BUFFER, buffers[<numero>]);
```

```
        glBindBuffer(GL_ARRAY_BUFFER, buffers[i]); //buffers[get<1>(VBO[k])]
        printf("n_pontos: %d buffers: %d\n", get<0>(VBO[k]),buffers[get<1>(VBO[k])]);
        glVertexPointer(3, GL_FLOAT, 0, 0);
        //glDrawArrays(GL_TRIANGLES, first, count); count  n total de vertices.
        glDrawArrays(GL_TRIANGLES, 0, get<0>(VBO[k])*3);
        glPopMatrix();
        k++;
    }
}

void spherical2Cartesian() {

    camX = radius * cos(beta) * sin(alfa);
    camY = radius * sin(beta);
    camZ = radius * cos(beta) * cos(alfa);
}

void changeSize(int w, int h) {

    // Prevent a divide by zero, when window is too short
    // (you cant make a window with zero width).
    if (h == 0)
        h = 1;

    // compute window's aspect ratio
    float ratio = w * 1.0 / h;

    // Set the projection matrix as current
    glMatrixMode(GL_PROJECTION);
    // Load Identity Matrix
    glLoadIdentity();

    // Set the viewport to be the entire window
    glViewport(0, 0, w, h);

    // Set perspective
    gluPerspective(45.0f, ratio, 1.0f, 1000.0f);

    // return to the model view matrix mode
    glMatrixMode(GL_MODELVIEW);
}

void renderScene(void) {

    // clear buffers
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // set the camera
    glLoadIdentity();
    gluLookAt(camX, camY, camZ,
              laX, laY, laZ,
              0.0f, 1.0f, 0.0f);

    /*visao lateral dos planetas*/
    /*gluLookAt(camX, camY, camZ,
    250, 50.0f, 50.0f,
    0.0f, 1.0f, 0.0f);
    */

    // put the geometric transformations here
    if (flag_drawing_mode == 0) {
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    }
    else if (flag_drawing_mode == 1) {
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    }
    else if (flag_drawing_mode == 2) {
        glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);
```

```
    }

    //glutWireTeapot(1);
    //desenha();
    //glPushMatrix();
    desenha2();
    //glPopMatrix();
    //drawVBO();

    frame++;
    times = glutGet(GLUT_ELAPSED_TIME);
    if (times - timebase > 1000)
    {
        fps = frame*1000.0 / (times - timebase);
        timebase = times;
        frame = 0;
    }
    sprintf(print, "%d", fps);
    glutSetWindowTitle(print);

    // End of frame
    glutSwapBuffers();
}

void processKeys(unsigned char c, int xx, int yy) {
    if (c == 27) exit(0);
}

void processSpecialKeys(int key, int xx, int yy) {

    switch (key) {
    case GLUT_KEY_RIGHT:
        alfa -= 0.1; break;

    case GLUT_KEY_LEFT:
        alfa += 0.1; break;

    case GLUT_KEY_UP:
        beta += 0.1f;
        if (beta > 1.5f)
            beta = 1.5f;
        break;

    case GLUT_KEY_DOWN:
        beta -= 0.1f;
        if (beta < -1.5f)
            beta = -1.5f;
        break;

    case GLUT_KEY_PAGE_UP:
        radius -= 0.8f;
        if (radius < 0.1f)
            radius = 0.8f;
        break;

    case GLUT_KEY_PAGE_DOWN:
        radius += 0.8f;
        break;
    }
    spherical2Cartesian();
    glutPostRedisplay();

}

int translacao(TiXmlElement* translate) {
    const char *aux_x = translate->Attribute("X");
    const char *aux_y = translate->Attribute("Y");
    const char *aux_z = translate->Attribute("Z");
```

```
    if (aux_x)  translate_x = atof(aux_x);
    if (aux_y)  translate_y = atof(aux_y);
    if (aux_z)  translate_z = atof(aux_z);

    return 0;
}
int rotacao(TiXmlElement* rotate) {

    const char *aux_a = rotate->Attribute("angle");
    const char *aux_x = rotate->Attribute("axisX");
    const char *aux_y = rotate->Attribute("axisY");
    const char *aux_z = rotate->Attribute("axisZ");

    if (aux_a)  angulo = atof(aux_a);
    if (aux_x)  rotate_x = atof(aux_x);
    if (aux_y)  rotate_y = atof(aux_y);
    if (aux_z)  rotate_z = atof(aux_z);

    return 0;
}
int escala(TiXmlElement* scale) {

    const char *aux_x = scale->Attribute("X");
    const char *aux_y = scale->Attribute("Y");
    const char *aux_z = scale->Attribute("Z");

    if (aux_x)  scale_x = atof(scale->Attribute("X"));
    if (aux_y)  scale_y = atof(scale->Attribute("Y"));
    if (aux_z)  scale_z = atof(scale->Attribute("Z"));

    return 0;
}
//int modelo(){}

int le_xml(char *nome) {
    int erros = 0;
    //string caminho = "xml/" + (string)nome;

    string caminho = "../";
    caminho += nome;
    TiXmlDocument doc;

    if (!doc.LoadFile(caminho.c_str())) {
        cout << "Nome do ficheiro invlido" << caminho << endl;
        return erros + 1;
    }

    //scene
    TiXmlElement* raiz = doc.FirstChildElement();
    if (raiz == NULL) return erros + 1;

    // Grupos
    TiXmlElement* grupo_ext = NULL;
    for (grupo_ext = raiz->FirstChildElement("group"); grupo_ext; grupo_ext =
         grupo_ext->NextSiblingElement("group")) {

        // TRANSLATE
        TiXmlElement* translate = grupo_ext->FirstChildElement("translate");
        if (translate != NULL) { // entrar no translate;
            translacao(translate);
            //const char* t_aux = translate->Attribute("time"); // guardar time;
            //if (t_aux) { // se time existir, converter para int e ler pontos;
            //    time_t = atoi(t_aux); // converso
            //    lista_times.push_back(time_t); // guardar
            //    while (TiXmlElement* point = translate->NextSiblingElement("point")) { //
                  enquanto tiver pontos
            //        translacao(translate); // tratar da translao;
```

```
    //                          // guardar esta  translao  no vector;
    //      lista_translacoes.push_back(tuple<float, float, float>(translate_x,
          translate_y, translate_z));
    //   }
    //   // guardar pontos da  translao ;
    //   lista_pontos_translacao.push_back(lista_translacoes);
    //   lista_translacoes.clear(); //
    //}
}

// ROTATE
TiXmlElement* rotate = grupo_ext->FirstChildElement("rotate");
if (rotate != NULL)
    rotacao(rotate);

// SCALE
TiXmlElement* scale = grupo_ext->FirstChildElement("scale");
if (scale != NULL)
    escala(scale);

TiXmlElement* models = grupo_ext->FirstChildElement("models");
if (models != NULL) {
    const char* nome_aux = NULL;

    nome_aux = models->FirstChildElement("model")->Attribute("file");
    if (nome_aux == NULL) return 0;
    std::string nome_ficheiro = "";
    nome_ficheiro += nome_aux;

    lista_ficheiros.push_back(nome_ficheiro);
    lista_rotacoes.push_back(tuple<float, float, float>(rotate_x, rotate_y,
          rotate_z));
    lista_angulos.push_back(angulo);
    lista_translacoes.push_back(tuple<float, float, float>(translate_x, translate_y,
          translate_z));
    //lista_times.push_back(time);
    lista_escalas.push_back(tuple<float, float, float>(scale_x, scale_y, scale_z));

}

TiXmlElement* grupo_int = NULL;
for (grupo_int = grupo_ext->FirstChildElement("group"); grupo_int; grupo_int =
      grupo_int->NextSiblingElement("group")) {
    // TRANSLATE
    TiXmlElement* translate = grupo_int->FirstChildElement("translate");
    if (translate != NULL) { // entrar no translate;
        translacao(translate);
        //const char* t_aux = translate->Attribute("time"); // guardar time;
        //if (t_aux) { // se time existir, converter para int e ler pontos;
        //    time_t = atoi(t_aux); // converso
        //    lista_times.push_back(time_t); // guardar
        //    while (TiXmlElement* point = translate->NextSiblingElement("point")) {
              // enquanto tiver pontos
        //        translacao(translate); // tratar da translao;
        //        // guardar esta  translao  no vector;
        //        lista_translacoes.push_back(tuple<float, float, float>(translate_x,
              translate_y, translate_z));
        //    }
        //    // guardar pontos da  translao ;
        //    lista_pontos_translacao.push_back(lista_translacoes);
        //    lista_translacoes.clear(); //
        //}
    }


    // ROTATE
    TiXmlElement* rotate = grupo_int->FirstChildElement("rotate");
    if (rotate != NULL)
```

```
                rotacao(rotate);

            // SCALE
            TiXmlElement* scale = grupo_int->FirstChildElement("scale");
            if (scale != NULL)
                escala(scale);

            TiXmlElement* models = grupo_int->FirstChildElement("models");
            if (models != NULL) {
                const char* nome_aux = NULL;

                nome_aux = models->FirstChildElement("model")->Attribute("file");
                if (nome_aux == NULL) return 0;
                std::string nome_ficheiro = "";
                nome_ficheiro += nome_aux;

                lista_ficheiros.push_back(nome_ficheiro);
                lista_rotacoes.push_back(tuple<float, float, float>(rotate_x, rotate_y,
                     rotate_z));
                lista_angulos.push_back(angulo);
                lista_translacoes.push_back(tuple<float, float, float>(translate_x,
                     translate_y, translate_z));
                //lista_times.push_back(time);
                lista_escalas.push_back(tuple<float, float, float>(scale_x, scale_y,
                     scale_z));

            }
        }
    }
    return 0;
}

void processMenuEvents(int option) {

    switch (option) {
    case 0:
        flag_drawing_mode = 0;
        break;
    case 1:
        flag_drawing_mode = 1;
        break;
    case 2:
        flag_drawing_mode = 2;
        break;
    default:
        break;
    }

    glutPostRedisplay();

}

void createGLUTMenus() {

    int menu;

    menu = glutCreateMenu(processMenuEvents);

    glutAddMenuEntry("Fill", 0);
    glutAddMenuEntry("Line", 1);
    glutAddMenuEntry("Point", 2);

    glutAttachMenu(GLUT_RIGHT_BUTTON);
}

void printInfo() {

    printf("Vendor: %s\n", glGetString(GL_VENDOR));
```

```
    printf("Renderer: %s\n", glGetString(GL_RENDERER));
    printf("Version: %s\n", glGetString(GL_VERSION));

    /*printf("\nUse Arrows to move the camera up/down and left/right\n");
    printf("Home and End control the distance from the camera to the origin");*/
}

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

    // init GLUT and the window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(glutGet(GLUT_SCREEN_WIDTH), glutGet(GLUT_SCREEN_HEIGHT));
    glutCreateWindow("MOTOR");

    // Required callback registry
    glutDisplayFunc(renderScene);
    glutReshapeFunc(changeSize);

    // Callback registration for keyboard processing
    glutKeyboardFunc(processKeys);
    glutSpecialFunc(processSpecialKeys);

    glewInit();

    // MENUS
    glutDetachMenu(GLUT_RIGHT_BUTTON);
    createGLUTMenus();


    // OpenGL settings
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    glClearColor(1, 1, 1, 1);

    /*if(argc == 2){
    if(le_xml(argv[1]) > 0){
    cout << "O ficheiro xml no foi encontrado" << endl;
    return 1;
    }
    }else{
    cout << "Nmero de argumentos invlido" << endl;
    return 1;
    }*/

    le_xml(argv[1]);
    //le_xml("universo.xml");
    //cout << lista_ficheiros.size() << endl;
    desenha();
    glutPostRedisplay();
    spherical2Cartesian();
    printInfo();
    // enter GLUT's main cycle
    glutMainLoop();

    return 0;
}
```

A.2 Código do gerador

```
#include <iostream>
#include <fstream>
#define _USE_MATH_DEFINES

#include <math.h>
#include <string>

using namespace std;

fstream file;
//fstream patch;

float *pontosPatch(int *patches, int n_patches, float *control_points, int
     n_control_points, int nivel) {
    float coordenadas[48];
    float peso = 1.0 / nivel, *pontos = (float*)malloc(n_patches*(3 * (nivel + 1) ^ 2) *
         sizeof(float));
    float a, b, c, d;
    int i = 0, j = 0, k = 0, t = 0;

    for (t = 0; t < n_patches; t++) {

        a = 0.0;
        b = 1.0 - a;
        c = 0.0;
        d = 1.0 - c;

        // 1  ponto
        coordenadas[0] = control_points[3 * patches[16 * t]];
        coordenadas[1] = control_points[3 * patches[16 * t] + 1];
        coordenadas[2] = control_points[3 * patches[16 * t] + 2];
        // 2  ponto
        coordenadas[3] = control_points[3 * patches[16 * t + 1]];
        coordenadas[4] = control_points[3 * patches[16 * t + 1] + 1];
        coordenadas[5] = control_points[3 * patches[16 * t + 1] + 2];
        // 3  ponto
        coordenadas[6] = control_points[3 * patches[16 * t + 2]];
        coordenadas[7] = control_points[3 * patches[16 * t + 2] + 1];
        coordenadas[8] = control_points[3 * patches[16 * t + 2] + 2];
        // 4  ponto
        coordenadas[9] = control_points[3 * patches[16 * t + 3]];
        coordenadas[10] = control_points[3 * patches[16 * t + 3] + 1];
        coordenadas[11] = control_points[3 * patches[16 * t + 3] + 2];
        // 5  ponto
        coordenadas[12] = control_points[3 * patches[16 * t + 4]];
        coordenadas[13] = control_points[3 * patches[16 * t + 4] + 1];
        coordenadas[14] = control_points[3 * patches[16 * t + 4] + 2];
        // 6  ponto
        coordenadas[15] = control_points[3 * patches[16 * t + 5]];
        coordenadas[16] = control_points[3 * patches[16 * t + 5] + 1];
        coordenadas[17] = control_points[3 * patches[16 * t + 5] + 2];
        // 7  ponto
        coordenadas[18] = control_points[3 * patches[16 * t + 6]];
        coordenadas[19] = control_points[3 * patches[16 * t + 6] + 1];
        coordenadas[20] = control_points[3 * patches[16 * t + 6] + 2];
        // 8  ponto
        coordenadas[21] = control_points[3 * patches[16 * t + 7]];
        coordenadas[22] = control_points[3 * patches[16 * t + 7] + 1];
        coordenadas[23] = control_points[3 * patches[16 * t + 7] + 2];
        // 9  ponto
        coordenadas[24] = control_points[3 * patches[16 * t + 8]];
        coordenadas[25] = control_points[3 * patches[16 * t + 8] + 1];
        coordenadas[26] = control_points[3 * patches[16 * t + 8] + 2];
        // 10  ponto
```

```
    coordenadas[27] = control_points[3 * patches[16 * t + 9]];
    coordenadas[28] = control_points[3 * patches[16 * t + 9] + 1];
    coordenadas[29] = control_points[3 * patches[16 * t + 9] + 2];
    // 11  ponto
    coordenadas[30] = control_points[3 * patches[16 * t + 10]];
    coordenadas[31] = control_points[3 * patches[16 * t + 10] + 1];
    coordenadas[32] = control_points[3 * patches[16 * t + 10] + 2];
    // 12  ponto
    coordenadas[33] = control_points[3 * patches[16 * t + 11]];
    coordenadas[34] = control_points[3 * patches[16 * t + 11] + 1];
    coordenadas[35] = control_points[3 * patches[16 * t + 11] + 2];
    // 13  ponto
    coordenadas[36] = control_points[3 * patches[16 * t + 12]];
    coordenadas[37] = control_points[3 * patches[16 * t + 12] + 1];
    coordenadas[38] = control_points[3 * patches[16 * t + 12] + 2];
    // 14  ponto
    coordenadas[39] = control_points[3 * patches[16 * t + 13]];
    coordenadas[40] = control_points[3 * patches[16 * t + 13] + 1];
    coordenadas[41] = control_points[3 * patches[16 * t + 13] + 2];
    // 15  ponto
    coordenadas[42] = control_points[3 * patches[16 * t + 14]];
    coordenadas[43] = control_points[3 * patches[16 * t + 14] + 1];
    coordenadas[44] = control_points[3 * patches[16 * t + 14] + 2];
    // 16  ponto
    coordenadas[45] = control_points[3 * patches[16 * t + 15]];
    coordenadas[46] = control_points[3 * patches[16 * t + 15] + 1];
    coordenadas[47] = control_points[3 * patches[16 * t + 15] + 2];

    for (i = 0; i < nivel; i++) {
        for (j = 0; j < nivel; j++) {
            pontos[k++] = coordenadas[0] * a*a*a*c*c*c + coordenadas[3] * 3 * a*a*a*c*c*d
                 + coordenadas[6] * 3 * a*a*a*c*d*d + coordenadas[9] * a*a*a*d*d*d
                + coordenadas[12] * 3 * a*a*b*c*c*c + coordenadas[15] * 9 * a*a*b*c*c*d +
                     coordenadas[18] * 9 * a*a*b*c*d*d + coordenadas[21] * 3 * a*a*b*d*d*d
                + coordenadas[24] * 3 * a*b*b*c*c*c + coordenadas[27] * 9 * a*b*b*c*c*d +
                     coordenadas[30] * 9 * a*b*b*c*d*d + coordenadas[33] * 3 * a*b*b*d*d*d
                + coordenadas[36] * b*b*b*c*c*c + coordenadas[39] * 3 * b*b*b*c*c*d +
                     coordenadas[42] * 3 * b*b*b*c*d*d + coordenadas[45] * b*b*b*d*d*d;

            pontos[k++] = coordenadas[1] * a*a*a*c*c*c + coordenadas[4] * 3 * a*a*a*c*c*d
                 + coordenadas[7] * 3 * a*a*a*c*d*d + coordenadas[10] * a*a*a*d*d*d
                + coordenadas[13] * 3 * a*a*b*c*c*c + coordenadas[16] * 9 * a*a*b*c*c*d +
                     coordenadas[19] * 9 * a*a*b*c*d*d + coordenadas[22] * 3 * a*a*b*d*d*d
                + coordenadas[25] * 3 * a*b*b*c*c*c + coordenadas[28] * 9 * a*b*b*c*c*d +
                     coordenadas[31] * 9 * a*b*b*c*d*d + coordenadas[34] * 3 * a*b*b*d*d*d
                + coordenadas[37] * b*b*b*c*c*c + coordenadas[40] * 3 * b*b*b*c*c*d +
                     coordenadas[43] * 3 * b*b*b*c*d*d + coordenadas[46] * b*b*b*d*d*d;

            pontos[k++] = coordenadas[2] * a*a*a*c*c*c + coordenadas[5] * 3 * a*a*a*c*c*d
                 + coordenadas[8] * 3 * a*a*a*c*d*d + coordenadas[11] * a*a*a*d*d*d
                + coordenadas[14] * 3 * a*a*b*c*c*c + coordenadas[17] * 9 * a*a*b*c*c*d +
                     coordenadas[20] * 9 * a*a*b*c*d*d + coordenadas[23] * 3 * a*a*b*d*d*d
                + coordenadas[26] * 3 * a*b*b*c*c*c + coordenadas[29] * 9 * a*b*b*c*c*d +
                     coordenadas[32] * 9 * a*b*b*c*d*d + coordenadas[35] * 3 * a*b*b*d*d*d
                + coordenadas[38] * b*b*b*c*c*c + coordenadas[41] * 3 * b*b*b*c*c*d +
                     coordenadas[44] * 3 * b*b*b*c*d*d + coordenadas[47] * b*b*b*d*d*d;

            c += peso;
            d = 1.0 - c;
        }
        a += peso;
        b = 1.0 - a;

        c = 0.0;
        d = 1.0 - c;
    }
    return pontos;
}
```

```
}

// esta  funo  recebe o ficheiro para o qual est a escrever e o nivel de tesselagem;
void read_patch(FILE *f_patch, int nivel) {
    int n_patches, n_control_points, i, j, m, aux, p = 0, *patches = NULL;
    float * points = NULL, *control_points = NULL, x, y, z;

    //  l  a primeira linha do ficheiro f_patch e guarda o nmero para a varivel n_patches;
    fscanf(f_patch, "%d\n", &n_patches);

    // aloca memria para as patches (n linhas, 16 por linha com tamanho int);
    patches = (int*)malloc(16 * n_patches * sizeof(int));

    // percorre cada linha
    for (i = 0; i < n_patches; i++) {
        // percorre cada indice da linha (16 no total)
        for (j = 0; j < 15; j++) {
            fscanf(f_patch, "%d, ", &aux);
            patches[p++] = aux;
        }
        fscanf(f_patch, "%d\n", &aux);
        patches[p++] = aux;
    }

    //  l  numero de pontos de controlo
    fscanf(f_patch, "%d\n", &n_control_points);
    control_points = (float*)malloc(3 * n_control_points * sizeof(float));

    for (i = 0; fscanf(f_patch, "%f %f %f\n", &x, &y, &z) != EOF; i += 3) {
        control_points[i] = x;
        control_points[i + 1] = y;
        control_points[i + 2] = z;
    }

    // funo  pontosPatch
    points = pontosPatch(patches, n_patches, control_points, n_control_points, nivel);


    //imprimir pontos para o documento
    n_control_points = n_patches * (3 * (nivel + 1)*(nivel + 1));
    file << n_control_points << endl; // guarda nmero de pontos

    for (i = 0; i < n_control_points; i += 3) {
        file << points[i] << " " << points[i + 1] << " " << points[i + 2] << endl;
    }

    n_control_points = n_patches * nivel * nivel * 3 * 2;

    file << n_control_points << endl;

    m = (nivel + 1) ^ 2;

    for (i = 0; i < n_patches; i++) {
        for (j = 0; j < nivel; j++) {
            for (p = 0; p < nivel; p++) {
                file << i*m + j*(nivel + 1) + p << " " << i*m + j*(nivel + 1) + p + 1 << " "
                     << i*m + (j + 1)*(nivel + 1) + p << endl;
                file << i*m + j*(nivel + 1) + p + 1 << " " << i*m + (j + 1)*(nivel + 1) + p +
                     1 << " " << i*m + (j + 1)*(nivel + 1) + p << endl;
            }
        }
    }

}

void plano(float comprimento) {
    float m_comp = comprimento / 2;
```

```
    //  N  DE TRIANGULOS
    file << "2" << endl;

    file << -m_comp << " 0.0 " << -m_comp << endl;
    file << -m_comp << " 0.0 " << m_comp << endl;
    file << m_comp << " 0.0 " << m_comp << endl;

    file << m_comp << " 0.0 " << m_comp << endl;
    file << m_comp << " 0.0 " << -m_comp << endl;
    file << -m_comp << " 0.0 " << -m_comp << endl;
}

void caixa(float x, float y, float z, int dimensions) {
    double dim_x = x / dimensions, dim_y = y / dimensions, dim_z = z / dimensions;
    double ori_x = -x / 2, ori_y = -y / 2, ori_z = -z / 2; // origem x, y e z
    double xx = ori_x, yy = ori_y, zz = ori_z; // ponto "origem"

    file << 4 << endl;

    for (xx = ori_x; xx < (-ori_x); xx += dim_x) {
        //x2 = x1 + dim_x;

        for (yy = ori_y; yy < (-ori_y); yy += dim_y) {
            //y2 = y1 + dim_y;

            for (zz = ori_z; zz < (-ori_z); zz += dim_z) {
                //z2 = z1 + dim_z;

                if (xx == ori_x) {
                    //glColor3f(0.09 << " " << 0.5 << " " << 0.99 << endl;
                    file << xx << " " << yy + dim_y << " " << zz << endl;
                    file << xx << " " << yy << " " << zz << endl;
                    file << xx << " " << yy + dim_y << " " << zz + dim_z << endl;

                    //glColor3f(0.18 << " " << 0.5 << " " << 0.90 << endl;
                    file << xx << " " << yy + dim_y << " " << zz + dim_z << endl;
                    file << xx << " " << yy << " " << zz << endl;
                    file << xx << " " << yy << " " << zz + dim_z << endl;

                }

                if (yy == ori_y) {
                    //glColor3f(0.27 << " " << 0.5 << " " << 0.81 << endl;
                    file << xx << " " << yy << " " << zz << endl;
                    file << xx + dim_x << " " << yy << " " << zz << endl;
                    file << xx + dim_x << " " << yy << " " << zz + dim_z << endl;

                    //glColor3f(0.36 << " " << 0.5 << " " << 0.73 << endl;
                    file << xx + dim_x << " " << yy << " " << zz + dim_z << endl;
                    file << xx << " " << yy << " " << zz + dim_z << endl;
                    file << xx << " " << yy << " " << zz << endl;
                }

                if (zz == ori_z) {
                    //glColor3f(0.45 << " " << 0.5 << " " << 0.64 << endl;
                    file << xx << " " << yy << " " << zz << endl;
                    file << xx << " " << yy + dim_y << " " << zz << endl;
                    file << xx + dim_x << " " << yy << " " << zz << endl;

                    //glColor3f(0.54 << " " << 0.5 << " " << 0.55 << endl;
                    file << xx + dim_x << " " << yy << " " << zz << endl;
                    file << xx << " " << yy + dim_y << " " << zz << endl;
                    file << xx + dim_x << " " << yy + dim_y << " " << zz << endl;
                }
            }
        }
    }
```

```
    for (xx = -ori_x; xx > ori_x; xx -= dim_x) {
        for (yy = -ori_y; yy > ori_y; yy -= dim_y) {
            for (zz = -ori_z; zz > ori_z; zz -= dim_z) {
                if (xx == -ori_x) {
                    //glColor3f(0.63 << " " << 0.63 << " " << 0.46 << endl;
                    file << xx << " " << yy << " " << zz << endl;
                    file << xx << " " << yy - dim_y << " " << zz << endl;
                    file << xx << " " << yy - dim_y << " " << zz - dim_z << endl;

                    //glColor3f(0.72 << " " << 0.72 << " " << 0.37 << endl;
                    file << xx << " " << yy - dim_y << " " << zz - dim_z << endl;
                    file << xx << " " << yy << " " << zz - dim_z << endl;
                    file << xx << " " << yy << " " << zz << endl;

                }

                if (yy == -ori_y) {
                    //glColor3f(0.81 << " " << 0.81 << " " << 0.28 << endl;
                    file << xx - dim_x << " " << yy << " " << zz << endl;
                    file << xx << " " << yy << " " << zz << endl;
                    file << xx - dim_x << " " << yy << " " << zz - dim_z << endl;

                    //glColor3f(0.9 << " " << 0.9 << " " << 0.19 << endl;
                    file << xx - dim_x << " " << yy << " " << zz - dim_z << endl;
                    file << xx << " " << yy << " " << zz << endl;
                    file << xx << " " << yy << " " << zz - dim_z << endl;
                }

                if (zz == -ori_z) {
                    //glColor3f(0.95 << " " << 0.95 << " " << 0.10 << endl;
                    file << xx << " " << yy - dim_y << " " << zz << endl;
                    file << xx << " " << yy << " " << zz << endl;
                    file << xx - dim_x << " " << yy << " " << zz << endl;

                    //glColor3f(0.99 << " " << 0.99 << " " << 0.01 << endl;
                    file << xx - dim_x << " " << yy << " " << zz << endl;
                    file << xx - dim_x << " " << yy - dim_y << " " << zz << endl;
                    file << xx << " " << yy - dim_y << " " << zz << endl;
                }
            }
        }
    }
}

/* esta  funo  (juntamente com todas as outras que geram as figuras) tem de ser alterada
     para que escreva para um ficheiro os
pontos do VBO. Mas no tenho a certeza se precisa ser feito se no usarmos VBO's com
     indices... */
void esfera(float radius, int slices, int stacks) {
    int i, j; // iteradores
    double alpha1 = 0, alpha2 = 0; // angulo de cada fatia
    double beta1 = 0, beta2 = 0; // angulo de cada corte
    double alpha = (2 * M_PI) / slices; //
    double beta = -(M_PI) / stacks;

    //   N  DE TRIANGULOS
    file << 2 * stacks*slices << endl;

    for (j = -stacks / 2; j < stacks / 2; j++) {
        beta1 = beta * j;
        beta2 = beta * (j + 1);
        double raio1 = radius * cos(beta1);
        double raio2 = radius * cos(beta2);

        for (i = 0; i < slices; i++) {
            alpha1 = alpha * i;
            alpha2 = alpha * (i + 1);
            // 1   triangulo
```

```
            file << raio1 * sin(alpha1) << " " << radius * sin(beta1) << " " << raio1 *
                 cos(alpha1) << endl;
            file << raio2 * sin(alpha1) << " " << radius * sin(beta2) << " " << raio2 *
                 cos(alpha1) << endl;
            file << raio1 * sin(alpha2) << " " << radius * sin(beta1) << " " << raio1 *
                 cos(alpha2) << endl;
            // 2  triangulo
            file << raio2 * sin(alpha2) << " " << radius * sin(beta2) << " " << raio2 *
                 cos(alpha2) << endl;
            file << raio1 * sin(alpha2) << " " << radius * sin(beta1) << " " << raio1 *
                 cos(alpha2) << endl;
            file << raio2 * sin(alpha1) << " " << radius * sin(beta2) << " " << raio2 *
                 cos(alpha1) << endl;
        }
    }
}

void cone(float radius, float height, int slices, int stacks) {
    int i, j; // iteradores
    double altura2 = 0, altura1 = 0; // altura de cada base
    double alpha1 = 0, alpha2 = 0; // angulo de cada fatia
    double alpha = (2 * M_PI) / slices; //
    double stack_height = height / stacks;
    double raio2, raio1 = radius;

    //  N  DE TRIANGULOS
    file << stacks*slices * 2 + slices << endl;

    for (j = 0; j < stacks; j++) {
        altura2 += stack_height;
        raio1 = radius - radius * ((float)j / stacks);
        raio2 = radius - radius * ((float)(j + 1) / stacks);

        for (i = 0; i < slices; i++) {
            alpha1 = alpha * i;
            alpha2 = alpha * (i + 1);


            if (j == 0) {
                file << 0.0 << " " << 0.0 << " " << 0.0 << endl;
                file << raio1 * sin(alpha2) << " " << 0.0 << " " << raio1 * cos(alpha2) <<
                     endl;
                file << raio1 * sin(alpha1) << " " << 0.0 << " " << raio1 * cos(alpha1) <<
                     endl;
            }

            file << raio1 * sin(alpha1) << " " << altura1 << " " << raio1 * cos(alpha1) <<
                 endl;
            file << raio1 * sin(alpha2) << " " << altura1 << " " << raio1 * cos(alpha2) <<
                 endl;
            file << raio2 * sin(alpha1) << " " << altura2 << " " << raio2 * cos(alpha1) <<
                 endl;

            file << raio1 * sin(alpha2) << " " << altura1 << " " << raio1 * cos(alpha2) <<
                 endl;
            file << raio2 * sin(alpha2) << " " << altura2 << " " << raio2 * cos(alpha2) <<
                 endl;
            file << raio2 * sin(alpha1) << " " << altura2 << " " << raio2 * cos(alpha1) <<
                 endl;
        }
        altura1 = altura2;
    }
}

int main(int argc, char **argv) {
    FILE *patch = NULL;
    if (argc > 1) {
        /*  S  para quando est em debug
```

```
string caminho = "../../motor/";
*/
string caminho = "../motor/";

if (argv[1] == string("plane")) {
    if (argc == 4) {
        file.open(caminho + argv[3], std::fstream::out);
        plano(atoi(argv[2]));
    }
    else {
        cout << "Faltam argumentos!" << endl;
        cout << "Os argumentos necessrios so:" << endl;
        cout << "\t- comprimento" << endl;
        cout << "\t- nome do ficheiro para guardar os vrtices" << endl;
    }

}
else if (argv[1] == string("box")) {

    if (argc == 6) {
        file.open(caminho + argv[5], std::fstream::out);
        caixa(stof(argv[2]), stof(argv[3]), stof(argv[4]), 1);
    }
    else if (argc == 7) {
        file.open(caminho + argv[6], std::fstream::out);
        caixa(stof(argv[2]), stof(argv[3]), stof(argv[4]), atoi(argv[5]));
    }
    else {
        cout << "Faltam argumentos!" << endl;
        cout << "Os argumentos necessrios so:" << endl;
        cout << "\t- Tamanho X" << endl;
        cout << "\t- Tamanho Y" << endl;
        cout << "\t- Tamanho Z" << endl;
        cout << "\t- (OPCIONAL) nmero de divises" << endl;
        cout << "\t- nome do ficheiro para guardar os vrtices" << endl;
    }

}
else if (argv[1] == string("sphere")) {

    if (argc == 6) {
        file.open(caminho + argv[5], std::fstream::out);
        esfera(stof(argv[2]), stof(argv[3]), stof(argv[4]));
    }
    else {
        cout << "Faltam argumentos!" << endl;
        cout << "Os argumentos necessrios so:" << endl;
        cout << "\t- raio" << endl;
        cout << "\t- fatias" << endl;
        cout << "\t- camadas" << endl;
        cout << "\t- nome do ficheiro para guardar os vrtices" << endl;
    }

}
else if (argv[1] == string("cone")) {

    if (argc == 7) {
        file.open(caminho + argv[6], std::fstream::out);
        cone(stof(argv[2]), stof(argv[3]), stof(argv[4]), stof(argv[5]));
    }
    else {
        cout << "Faltam argumentos!" << endl;
        cout << "Os argumentos necessrios so:" << endl;
        cout << "\t- raio da base" << endl;
        cout << "\t- altura" << endl;
        cout << "\t- fatias" << endl;
        cout << "\t- camadas" << endl;
        cout << "\t- nome do ficheiro para guardar os vrtices" << endl;
```

```
            }

        }
        else if (argv[1] == string("patch")) {
            if (argc == 5) { //generator patch <NIVEL_TES> <FILE_IN> <FILE_OUT>
                patch = fopen(caminho.c_str() + *argv[3], "r");
                if (patch) {
                    file.open(caminho + argv[4], std::fstream::out);
                    read_patch(patch, stoi(argv[2]));
                }
                else {
                    printf("ERRO: O ficheiro patch especificado no existe.\n");
                }
            }
            else {
                cout << "Faltam argumentos!" << endl;
                cout << "Os argumentos necessrios so:" << endl;
                cout << "\t- Nivel de tecelagem" << endl;
                cout << "\t- Ficheiro de Input" << endl;
                cout << "\t- nome do ficheiro para guardar os pontos" << endl;
            }
        }
        else {
            cout << "Figura invlida" << endl;
        }
    }
    else {
        cout << "No foi dado nenhum argumento" << endl;
    }

    file.close();

    return 0;
}
```

A.3 Ficheiro solarf.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<scene>

    <group>
        <scale X="300" Y="500" Z="500" /> <!-- sol -->
        <translate X="-590" />
        <models>
            <model file="sphere.3d" />
        </models>
    </group>

    <group>
        <translate X="28" />
        <rotate angle="15" axisX="0" axisY="1" axisZ="0" />
        <scale X="1.7" Y="1.7" Z="1.7" /> <!-- mercurio -->

        <models>
            <model file="sphere.3d" />
        </models>
    </group>

    <group>
        <translate X="50" />
        <rotate angle="30" axisX="0" axisY="1" axisZ="0" />
        <scale X="4.3" Y="4.3" Z="4.3" /> <!-- venus -->

        <models>
            <model file="sphere.3d" />
        </models>
    </group>

    <group>
        <translate X="80" />
        <rotate angle="0" axisX="0" axisY="1" axisZ="0" />
        <scale X="4.5" Y="4.5" Z="4.5" /> <!-- terra -->

        <models>
            <model file="sphere.3d" />
        </models>

        <group>
            <translate Y="12" />
            <rotate angle="0" axisX="0" axisY="1" axisZ="0" />
            <scale X="1.2" Y="1.2" Z="1.2" /> <!-- lua -->

            <models>
                <model file="sphere.3d" />
            </models>
        </group>
    </group>

    <group>
        <translate X="100" Y="0" Z="0"/>
        <rotate angle="75" axisX="0" axisY="1" axisZ="0" />
        <scale X="2.4" Y="2.4" Z="2.4" /> <!-- marte -->

        <models>
            <model file="sphere.3d" />
        </models>
    </group>

    <group>
        <translate X="230" />
        <rotate angle="90" axisX="0" axisY="1" axisZ="0" />
```

```
    <scale X="50.1" Y="50.1" Z="50.1" /> <!-- jupiter -->

    <models>
        <model file="sphere.3d" />
    </models>

    <group>
        <translate Y="50" Z="130"/>
        <rotate angle="15" axisX="0" axisY="1" axisZ="0" />
        <scale X="1.2" Y="1.2" Z="1.2" /> <!-- lua europa -->

        <models>
            <model file="sphere.3d" />
        </models>
    </group>

    <group>
        <translate Y="130" Z ="90"/>
        <rotate angle="30" axisX="0" axisY="1" axisZ="0" />
        <scale X="1.2" Y="1.2" Z="1.2" /> <!-- lua IO -->

        <models>
            <model file="sphere.3d" />
        </models>
    </group>

    <group>
        <translate Y="90" Z="140"/>
        <rotate angle="45" axisX="0" axisY="1" axisZ="0" />
        <scale X="1.5" Y="1.5" Z="1.5" /> <!-- lua Ganimedes -->

        <models>
            <model file="sphere.3d" />
        </models>
    </group>

    <group>
        <translate Y="100" Z="130"/>
        <rotate angle="60" axisX="0" axisY="1" axisZ="0" />
        <scale X="1.5" Y="1.5" Z="1.5" /> <!--lua calisto-->

        <models>
            <model file="sphere.3d" />
        </models>
    </group>
</group>

<group>
    <translate X="500" Y="0" Z="0"/>
    <rotate angle="45" axisX="0" axisY="1" axisZ="0" />
    <scale X="42" Y="42" Z="42" /> <!-- saturno -->

    <models>
        <model file="sphere.3d" />
    </models>

    <group>

        <rotate angle="30" axisX="0" axisY="1" axisZ="1" />
        <scale X="60" Y="1.5" Z="60" /> <!-- anel -->

        <models>
            <model file="sphere.3d" />
        </models>
    </group>

    <group>
        <translate Y="100" />
```

```
            <rotate angle="120" axisX="0" axisY="1" axisZ="0" />
            <scale X="1.5" Y="1.5" Z="1.5" /> <!-- lua tita -->

            <models>
                <model file="sphere.3d" />
            </models>
        </group>


    </group>

    <group>
        <translate X="702" Y="0" Z="0"/>
        <rotate angle="135" axisX="0" axisY="1" axisZ="0" />
        <scale X="16.8" Y="16.8" Z="16.8" /> <!-- urano -->
        <models>
            <model file="sphere.3d" />
        </models>

    </group>

    <group>
        <translate X="800" Y="0" Z="0"/>
        <rotate angle="150" axisX="0" axisY="1" axisZ="0" />
        <scale X="16.3" Y="16.3" Z="16.3" /> <!-- neptuno -->

        <models>
            <model file="sphere.3d" />
        </models>

        <group>
            <translate Y="60" />
            <rotate angle="165" axisX="0" axisY="1" axisZ="0" />
            <scale X="1.1" Y="1.1" Z="1.1" /> <!-- lua tritao -->

            <models>
                <model file="sphere.3d" />
            </models>
        </group>
    </group>
</scene>
```