Escola de Engenharia

Universidade do Minho

Computação Gráfica

Transformações Geométricas

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-se no desenvolvimento de uma representação estática do sistema solar, incluindo o sol, os planetas e algumas das respetivas luas, através de modelos dispostos hierarquimente, compostas por figuras e transformações geométricas definidas préviamente pelo utilizador.
Pretende-se que o programa leia a partir de um ficheiro XML as transformações geométricas a fazer 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 todos 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 de todo o trabalho realizado e uma análise crítica dos resultados.
- No capítulo 6 são feitas referências a fontes utilizadas durante a 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 um cenário recorrendo a figuras e transformações geométricas obtidas préviamente através da leitura de um ficheiro XML cujo o nome é dado ao programa como argumento. Neste ficheiro estão hierarquicamente definidos grupos, e subgrupos se aplicável, que contêm a informação necessária para criar um cenário em 3 dimensões, nomeadamente os nomes dos ficheiros *.3d* onde anteriormente foram guardados os vértices correspondentes à criação de cada figura. De seguida, aplicando as respetivas transformações geométricas, armazenadas também aquando da leitura do XML, é gerada a figura final.

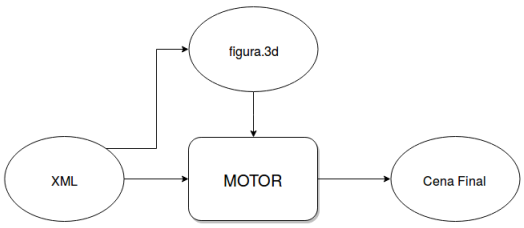

Figura 2.1: Diagrama do trabalho

Capítulo 3

Conceção/Desenho da Resolução

3.1 Leitura do XML

Sabendo que o ficheiro XML está definido hierarquicamente por grupos, pretende ler-se as informações referentes ao grupo pai e depois verificar se existe subgrupos. Cada grupo contém o nome do ficheiro .3d que será necessário ler para construir a figura pedida e as suas transformações geométricas, podendo também conter subgrupos. Estes subgrupos, se nada for dito, herdam as transformações geométricas do pai.
Tentou manter-se a leitura do XML simples mas eficaz. Foram utilizados então dois ciclos, um para os grupos externos que neste caso são os planetas e estrelas, e outro para os grupos internos que representam as luas. Desta forma, consegue-se representar qualquer sistema solar. Em cada um destes ciclos são lidos os 3 tipos de transormações obtendo assim a posição de cada planeta, estrela ou lua, sendo esta em relao à origem do referencial.

```
<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>
```

Listing 3.1: Excerto do ficheiro solar.xml

3.1.1 Estruturas de dados

Os nomes dos ficheiros *.3d* são guardados num vetor pela ordem em que são lidos, assim como as informações relativas ás transformações geométricas para que depois a leitura destas estruturas seja feita através do índice respetivo. Para mais fácil utilização de vetores, tendo em conta que se está a trabalhar num sistema de eixos tridimensional, define-se como estrutura um vetor que recebe um tuplo de 3 floats. Desta forma, são armazenadas as transformações geométricas, escalas, translações e rotações, no entanto nesta última, é necessário criar um vetor de floats para guardar os ângulos, pois a função *glRotatef* recebe 4 argumentos. Esta estrutura foi também utilizada para guardar cada cor referente aos astros do sistema solar pois, como é usado o sistema de cores RGB, são necessários 3 valores para definir determinada cor.

```
//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 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;
```

Listing 3.2: Estruturas de dados

3.2 Leitura dos ficheiros .3d

Percorrendo o vetor onde estão armazenados os nomes dos ficheiros, como já foi mencionado anteriormente, abre-se o respetivo ficheiro $.3d$. Se esta operação não for concluída com sucesso, o programa termina com uma mensagem de erro. A primeira linha de cada ficheiro contém o número de vértices da figura, por isso essa linha é ignorada, as restantes linhas são lidas e os vértices nelas contidos são armazenados numa estrutura do tipo $vector3f$, que foi mencionada no ponto anterior. Terminada a leitura de todas as linhas, o ficheiro é fechado e após terminar de construir esta figura, o vetor onde os vértices foram guardados é limpo, para ser utilizado com o próximo ficheiro.

```
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, ignorar
               if(primeira_linha==0){
                   primeira_linha=1;
               }
               else {
                   istringstream ss(str);
                   ss >> v1;
                   ss >> v2;
                   ss >> v3;
                   vertices.push_back(tuple<float,float,float>(v1,v2,v3));
               }
           }
           fi.close();
           primeira_linha = 0;
       }
       else{
           cerr << "Erro: Nao foi possivel abrir o ficheiro " << lista_ficheiros[i] << "."
                << endl;
           exit(1);
       }
(...)
       vertices.clear();
}
```

Listing 3.3: Leitura dos ficheiros .3d

3.3 Desenvolvimento das figuras

Cada figura é gerada a partir da construção sucessiva de triângulos cujos vértices foram préviamente guardados no vetor *vertices*. 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 triângulos. 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.

```
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]));

    glBegin(GL_TRIANGLES);
        for (int j = 0; j < vertices.size(); ++j){
            glVertex3f(get<0>(vertices[j]), get<1>(vertices[j]), get<2>(vertices[j]));
        }
    glEnd();

glPopMatrix();
```

Listing 3.4: Consctrução das figuras

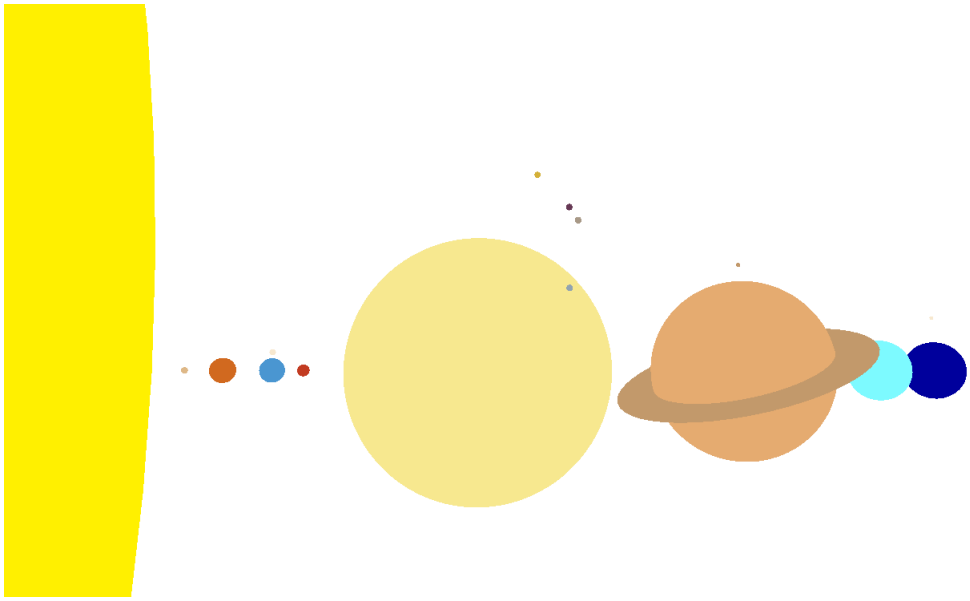

Figura 3.1: Representação do Sistema Solar

Capítulo 4

Conclusão

Pode-se concluir que a implementação das transformações geométricas exigiu o uso de estruturas de forma a conseguir gerir os dados e as transformações. Houve alguma dificuldade ao decidir que tipo de estruturas usar e qual a estratégia a ser aplicada de modo a tornar o programa mais rápido e eficiente. Contudo, a forma utilizada para ler os ficheiros será útil na implementação dos VBO's numa próxima fase. Para que o programa seja de fácil acesso a qualquer utilizador, foi criada uma makefile que permite executar todo o programa desde o input até à construção final da representação do sistema solar.

Capítulo 5

Referências

- https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/
- http://stackoverflow.com/questions/170686/what-is-the-best-open-xml-parser-for-c
- http://www.cplusplus.com/doc/tutorial/files/
- https://elearning.uminho.pt/

Apêndice A

Apêndice

A.1 Código do Motor

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

#include "tinyxml/tinyxml.h"
#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 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
int 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 = 500.0f;
float camX = 0.0f, camY = 0.0f, camZ = 0.0f;

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

/* Esta  funo  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
 */

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
}

void desenha(){
    /*
    * Variaveis
    */
    vector3f vertices; //vector< tuple<float, float, float> >
    float v1 = 0, v2=0, v3=0;
    string str;
    int primeira_linha = 0;

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

    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, passa a frente
                if(primeira_linha==0){
                    primeira_linha=1;
```

```
            }
            else {
                istringstream ss(str);
                ss >> v1;
                ss >> v2;
                ss >> v3;

                vertices.push_back(tuple<float,float,float>(v1,v2,v3));
            }
        }
        fi.close();
        primeira_linha = 0;
    }
    else{
        cerr << "Erro: No foi possvel abrir o ficheiro " << lista_ficheiros[i] << "." <<
             endl;
        exit(1);
    }

    /*
    * 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]));

        glBegin(GL_TRIANGLES);
            for (int j = 0; j < vertices.size(); ++j){
                glVertex3f(get<0>(vertices[j]), get<1>(vertices[j]), get<2>(vertices[j]));
            }
        glEnd();

    glPopMatrix();


    /*
    * clear vector for next file
    */
    vertices.clear();

  }
}

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();
    /*//visao lateral dos planetas
    gluLookAt(300, 0, 1000,
              300.0f, 0.0f, 0.0f,
              0.0f, 1.0f, 0.0f);
    */

    gluLookAt(camX, camY, camZ,
              250.0f, 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();

  // End of frame
  glutSwapBuffers();
}


void processKeys(unsigned char c, int xx, int yy) {

}


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 -= 0.1f;
            if (radius < 0.1f)
                radius = 0.1f;
            break;

        case GLUT_KEY_PAGE_DOWN:
            radius += 0.1f;
            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 = atoi(aux_x);
    if(aux_y)  translate_y = atof(aux_y);
    if(aux_z)  translate_z = atof(aux_z);

}
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);

}
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"));

}
int modelo(){}


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

    // 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_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)
            translacao(translate);

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

        // ROTATE
        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_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);
}

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);

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


//  OpenGL settings
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);

    glClearColor(1,1,1,1);

    le_xml(argv[1]);

    glutPostRedisplay();
    spherical2Cartesian();

// enter GLUT's main cycle
    glutMainLoop();

    return 0;
}
```

A.2 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>
```