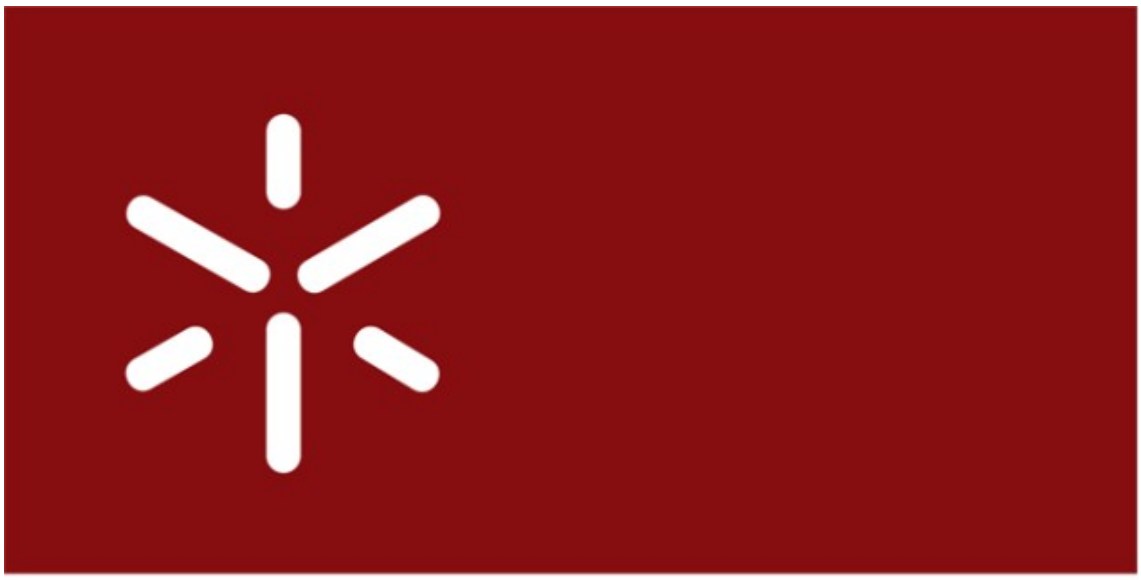
Relatório de Computação Gráfica

Fase 2- Transformações Geométricas



Universidade do Minho

2019/2020

Carlos Afonso A82529, Gonçalo Nogueira A86617, Luís Fernandes A76712,

Pedro Fernandes A84313

Índice

Introdução

No âmbito da cadeira de Computação Gráfica foi-nos proposto, na segunda fase, criar cenas hierárquicas usando transformações geométricas, neste caso um modelo estático do sistema solar, incluindo o sol, os planetas e luas definidas numa hierarquia.

Neste relatório faremos uma descrição detalhada das etapas do trabalho.

Estruturação do projeto

Nesta fase o gerador manteve-se inalterado comparativamente á primeira fase.

Apenas o motor foi alterado, com alterações feitas no parse do ficheiro XML e com a introdução de novas classes com estruturas para definir os vários tipos de transformações possíveis.

Motor

Leitura XML

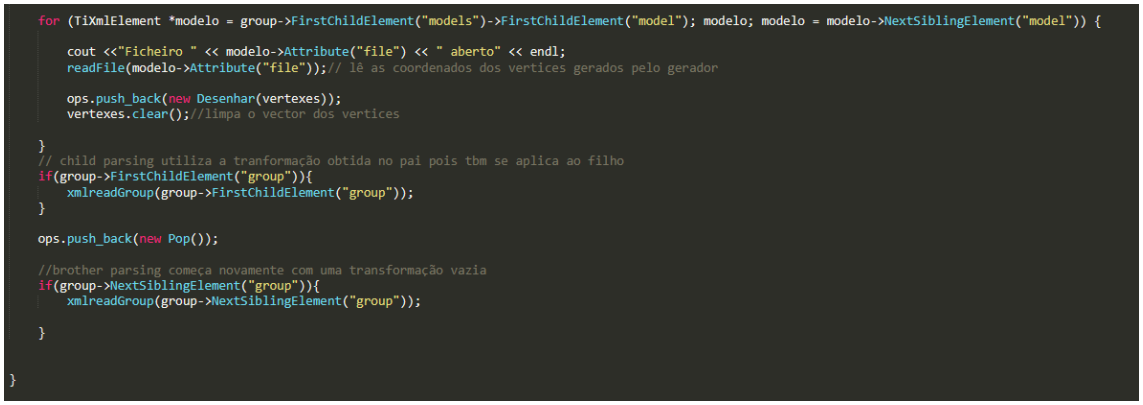
Cada leitura de XML é agora introduzida num grupo.

```
void xmlread(const char* pFilename) {
    string a = "../scenes/";
    string c = a + pFilename;
    pFilename = c.c_str();
    TiXmlDocument doc(pFilename);

    bool loadOkay = doc.LoadFile();
    if (loadOkay){

        TiXmlElement * scene = doc.FirstChildElement("scene");
        TiXmlElement * group = scene-> FirstChildElement("group");

        xmlreadGroup(group);
    }else {
        cout << "Ficheiro XML não foi encontrado" << endl;
    }
}
```

No método xmlreadGroup(TiXmlElement *group), o objetivo é ler o grupo e as operações feitas e em que ordem estão. Ou seja,

à medida que é lida a hierarquia do group/tree, são inseridas as operações juntamente com os seus argumentos no vetor de operações *ops*. Por cada leitura de grupo é inserido no vetor de operações um push e no final um pop. São também feitas recursivamente as leituras do grupos filhos (*child)* e irmãos (*siblings*).

```
//método para ler um  xml group

void xmlreadGroup(TiXmlElement *group) {

    float tlX, tlY, tlZ, angRo, esX, esY, esZ , roX, roY, roZ, coR, coG, coB;

    if (strcmp(group->FirstChildElement()->Value(), "group") == 0) {
        group = group->FirstChildElement();
    }

    ops.push_back(new Push());

    for (TiXmlElement *t = group->FirstChildElement(); (strcmp(t->Value(), "models") !=0); t = t->NextSiblingElement()) {
        if (!strcmp(t->Value(), "translate")) {
            const char *a1 = t->Attribute("X");
            const char *a2 = t->Attribute("Y");
            const char *a3 = t->Attribute("Z");

            if (a1) {
                tlX = stof(a1);
            } else tlX = 0;

            if (a2) {
                tlY = stof(a2);
            } else tlY = 0;

            if (a3) {
                tlZ = stof(a3);
            } else tlZ = 0;
            ops.push_back(new Translacao(tlX, tlY, tlZ));
        }
        if (!strcmp(t->Value(), "rotate")) {
            angRo = stof(t->Attribute("angle"));
            roX = stof(t->Attribute("axisX"));
            roY = stof(t->Attribute("axisY"));
            roZ = stof(t->Attribute("axisZ"));

            ops.push_back(new Rotacao(angRo, roX, roY, roZ));

        }
        if (!strcmp(t->Value(), "scale")) {
            esX = stof(t->Attribute("X"));
            esY = stof(t->Attribute("Y"));
            esZ = stof(t->Attribute("Z"));

            ops.push_back(new Escala(esX, esY, esZ));

        }
        if (!strcmp(t->Value(), "color")) {
            coR = stof(t->Attribute("R"));
            coG = stof(t->Attribute("G"));
            coB = stof(t->Attribute("B"));
            ops.push_back(new Cor(coR, coG, coB));
        }
    }
```



```
    for (TiXmlElement *modelo = group->FirstChildElement("models")->FirstChildElement("model"); modelo; modelo = modelo->NextSiblingElement("model")) {

        cout <<"Ficheiro " << modelo->Attribute("file") << " aberto" << endl;
        readFile(modelo->Attribute("file"));// lê as coordenados dos vertices gerados pelo gerador

        ops.push_back(new Desenhar(vertexes));
        vertexes.clear();//limpa o vector dos vertices

    }
    // child parsing utiliza a tranformação obtida no pai pois tbm se aplica ao filho
    if(group->FirstChildElement("group")){
        xmlreadGroup(group->FirstChildElement("group"));
    }

    ops.push_back(new Pop());

    //brother parsing começa novamente com uma transformação vazia
    if(group->NextSiblingElement("group")){
        xmlreadGroup(group->NextSiblingElement("group"));

    }


}
```

Classes

Foi criada a classe abstrata Operacao, com o método run que todas suas subclasses implementavam de forma a fazerem comandos GL.

Foram então criadas as seguintes subclasses:

Push: em que o método run executa o comando glPushMatrix().

Pop: em que o método run executa o comando glPopMatrix().

Translacao: em que o método run executa o comando glTranslatef().

Rotacao: em que o método run executa o comando glRotatef().

Escala: em que o método run executa o comando glScalef().

Cor: em que o método run executa o comando glColor3f().

Desenhar: em que o método run desenha o vetor de pontos da instância, recorrendo a GL_TRIANGLES e ao comando glVertex3f().

Cor.h

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

#ifndef TRABALHO_COR_H
#define TRABALHO_COR_H

#include "Operacao.h"

class Cor: public Operacao {
    float r, g, b;
public:
    Cor();
    Cor(float r, float g, float b);
    float getR() {return this->r;}
    float getG() {return this->g;}
    float getB() {return this->b;}
    void setR(float x){this->r=r;}
    void setG(float y){this->g=g;}
    void setB(float z){this->b=b;}
    void run();
};



#endif //TRABALHO_COR_H

```

Translacao.h

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

#ifndef TRABALHO_TRANSLACAO_H
#define TRABALHO_TRANSLACAO_H

#include "Operacao.h"

class Translacao: public Operacao{
    float x, y, z;

public:
    Translacao();
    Translacao(float x, float y, float z);
    float getX() {return this->x;}
    float getY() {return this->y;}
    float getZ() {return this->z;}
    void setX(float x){this->x=x;}
    void setY(float y){this->y=y;}
    void setZ(float z){this->z=z;}
    void run();
};


#endif //TRABALHO_TRANSLACAO_H

```

Rotacao.h

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

#ifndef TRABALHO_ROTACAO_H
#define TRABALHO_ROTACAO_H

#include "Operacao.h"

class Rotacao: public Operacao{
    float angle, x, y, z;

public:
    Rotacao();
    Rotacao(float angle, float x, float y, float z)
    float getAngle(){return this->angle;}
    float getX() {return this->x;}
    float getY() {return this->y;}
    float getZ() {return this->z;}
    void setAngle(float angle){this->angle=angle;}
    void setX(float x){this->x=x;}
    void setY(float y){this->y=y;}
    void setZ(float z){this->z=z;}
    void run();
};


#endif //TRABALHO_ROTACAO_H

```

Push.h

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

#ifndef TRABALHO_PUSH_H
#define TRABALHO_PUSH_H

#include "Operacao.h"

class Push: public Operacao
{

public:
    Push();
    void run();
};


#endif //TRABALHO_PUSH_H

```

Pop.h

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

#ifndef TRABALHO_POP_H
#define TRABALHO_POP_H

#include "Operacao.h"

class Pop: public Operacao {

public:
    Pop();
    void run();
};

#endif //TRABALHO_POP_H

```

Ponto.h

```
#ifndef TRABALHO_PONTO_H
#define TRABALHO_PONTO_H



class Ponto {
    float x;
    float y;
    float z;

public:
    Ponto();
    Ponto(float a, float b, float c);
    float getX() { return this->x; }
    float getY() { return this->y; }
    float getZ() { return this->z; }
    void setX( float b) { this->x = b; }
    void setY( float b) { this->y = b; }
    void setZ( float b) { this->z = b; }

};


#endif //TRABALHO_PONTO_H

```

Escala.h

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

#ifndef TRABALHO_ESCALA_H
#define TRABALHO_ESCALA_H

#include "Operacao.h"

class Escala: public Operacao{
    float x, y, z;

public:
    Escala();
    Escala(float x, float y, float z);
    float getX() {return this->x;}
    float getY() {return this->y;}
    float getZ() {return this->z;}
    void setX(float x){this->x=x;}
    void setY(float y){this->y=y;}
    void setZ(float z){this->z=z;}
    void run();
};

#endif //TRABALHO_ESCALA_H

```

Desenhar.h

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

#include <vector>
#include "Ponto.h"
#include "Operacao.h"

#ifndef TRABALHO_DESENHAR_H
#define TRABALHO_DESENHAR_H


class Desenhar: public Operacao{

    std::vector<Ponto> vertexes;

public:
    Desenhar(std::vector<Ponto> vertexes){
        this->vertexes=vertexes;
    };

    void run();

};


#endif //TRABALHO_DESENHAR_H

```

Sistema Solar

Estão representados todos os planetas, excluindo Plutão, como também o sol e a Lua. Só foi incluído o satélite terrestre devido ao número elevado que alguns planetas têm.

A proporção de distâncias entre os corpos celetes, bem como os seus diâmetros foram tidos em conta mas não totalmente cumpridos pois senão não seriam visíveis alguns deles.

Configuração ficheiro XML

```
        <!--TERRA-->
<group>
    <color R="0.0" G="0.0" B="1.0" />
    <rotate angle="130.0" axisX="0.0" axisY="1.0" axisZ="0.0" />
    <translate X="0.0" Y="0.0" Z="-10.0" />
    <scale X="0.0915" Y="0.0915" Z="0.0915" />
    <models>
        <model file="sphere.3d" />
    </models>
                    <!--LUA-->
    <group>
        <color R="1.0" G="1.0" B="1.0" />
        <rotate angle="90.0" axisX="0.0" axisY="1.0" axisZ="0.0" />
        <translate X="2.0" Y="0.0" Z="0.0" />
        <scale X="0.0649" Y="0.0649" Z="0.0649" />
        <models>
            <model file="sphere.3d" />
        </models>
    </group>
</group>
        <!--MARTE-->
<group>
    <color R="1.0" G="0.6" B="0.2" />
    <rotate angle="170.0" axisX="0.0" axisY="1.0" axisZ="0.0" />
    <translate X="0.0" Y="0.0" Z="-15.0" />
    <scale X="0.0487" Y="0.0487" Z="0.0487" />
    <models>
        <model file="sphere.3d" />
    </models>
</group>
```

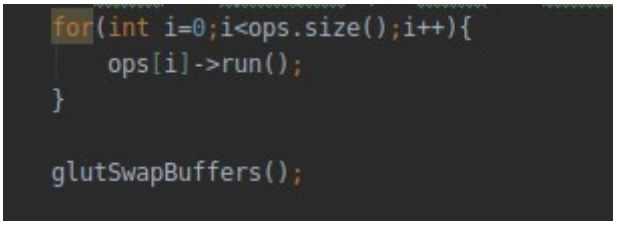
Para cada corpo celeste foi escolhida uma cor e foi definida uma rotação e translação para simular o movimento do planetas em volta do Sol.

Foi também atribuída uma escala de forma a simular o tamanho dos planetas.

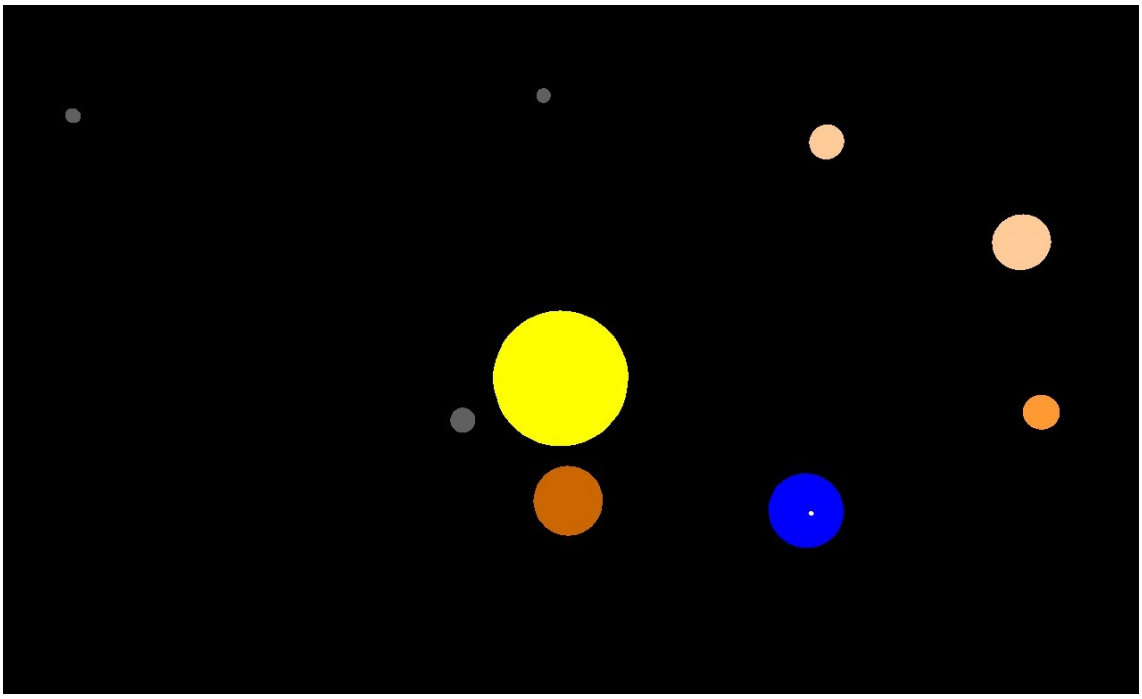
Todos eles tiveram como modelo uma esfera.

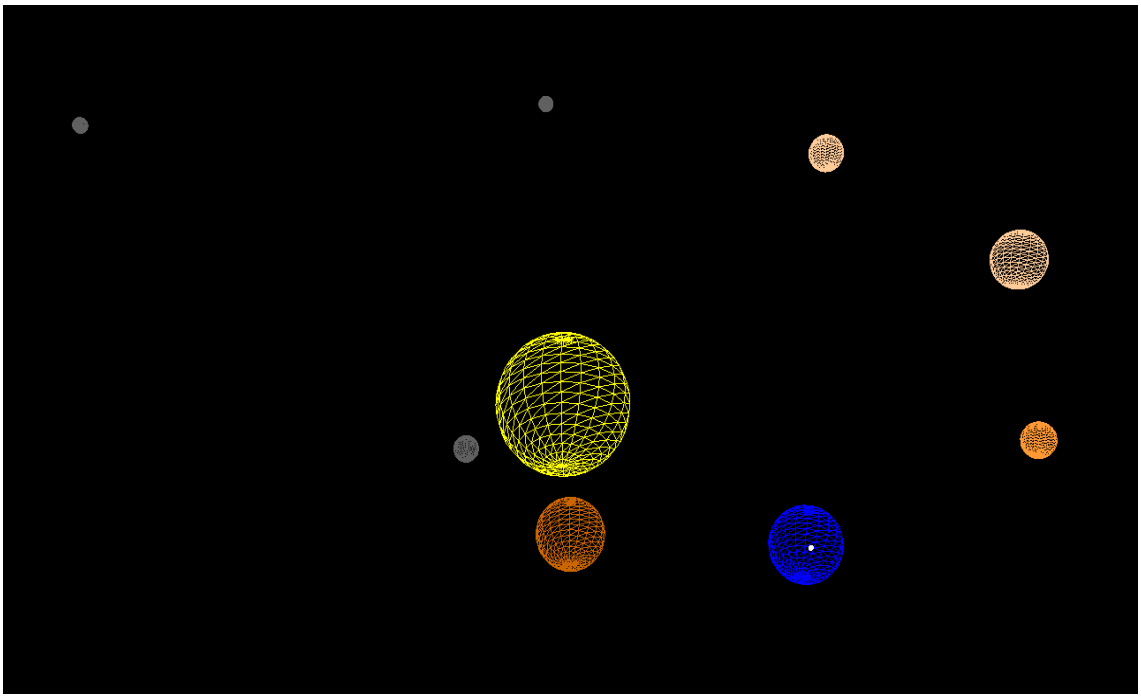
Processo de renderização

Tendo sido feita na leitura do xml a inserção das operações no vetor *ops*, *o mesmo é atravessado e para cada operação é executado o método run de forma a executar as primitivas GL*.



Resultados





Conclusão

Esta fase do trabalho foi importante pois conseguimos melhorar o trabalho realizado anteriormente com a introdução de transformações geométricas.

Nesta fase do trabalho foi-nos possível aprofundar os conhecimentos previamente adquiridos sobre a matéria em questão e consideramos que foi mais rápido o seu desenvolvimento devido á consolidação da matéria bem como um maior á vontade com a linguagem de programação em C++.

Esperamos continuar a melhorar o trabalho para as próximas fases com a introdução de novas técnicas com vista á otimização dos processos de desenho e renderizaçao bem como os FPS.