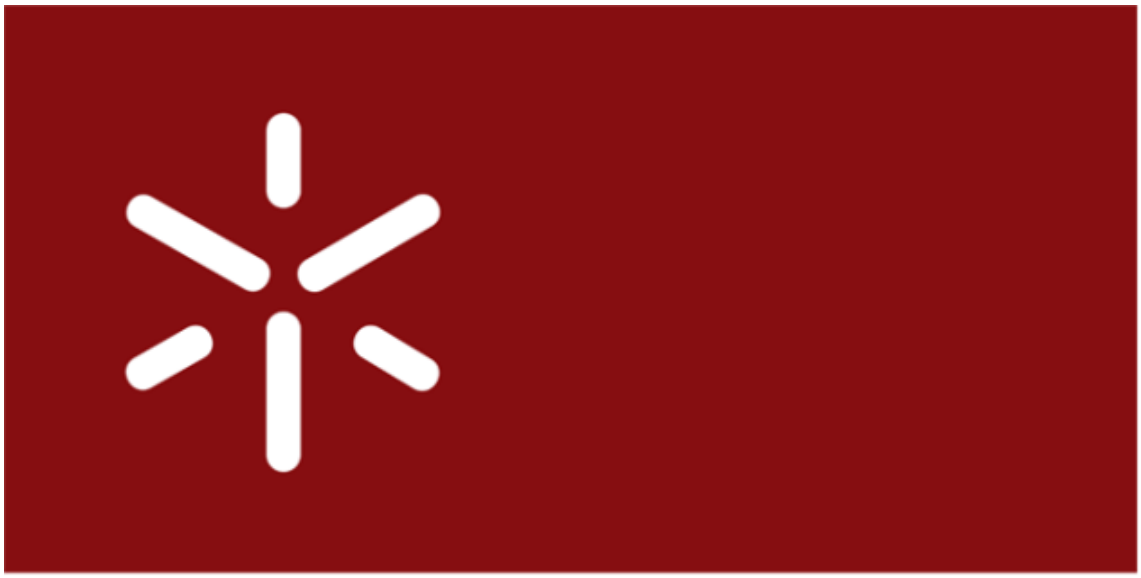
Relatório de Computação Gráfica

Fase 3- Curvas, Superfícies Cubicas e VBOs



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 terceira fase, a continuação do trabalho realizado na fase anterior, recorrendo a novos métodos que foram lecionados durante o semestre, com o objetivo de molhar o projeto.

Estes novos métodos são a implementação de curvas, superfícies cúbicas e a leitura de um ficheiro patch que contém os pontos de controlo para gerar imagens.

Será então montado um sistema solar, tal como na fase anterior, mas de forma dinâmica, ou seja, todos os objetos estarão em movimento.

Estruturação do projeto

Nesta fase, o gerador foi alterado, de forma a conseguir gerar uma superfície, consoante o ficheiro patch escolhido e o nível de “tesselation” desejado.

O motor foi alterado, com alterações feitas nos métodos de parsing do ficheiro XML, nas classes onde são armazenadas as transformações e, por fim, alterações nos métodos de renderização. Também foi implementado as curvas de Catmull Rom, e VBOs.

Motor

1. Leitura do XML

Cada leitura de XML continua a ser introduzida num grupo, no entanto, foram feitas alterações ao modo de leitura de transformações do tipo “translate” e “rotate”.

Relativamente ao parse de translações, foram introduzidas alterações para agora guardar o tempo da translação e foi introduzido um ciclo “for” para captar todos os pontos de translação do ficheiro XML e guardando tudo num vector “pTrans”.

```
if (!strcmp(t->Value(), "translate")) {
    const char *a1 = t->Attribute("X");
    const char *a2 = t->Attribute("Y");
    const char *a3 = t->Attribute("Z");
    const char *a4 = t->Attribute("time");

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

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

    if (a3) {
        tlZ = stof(a3);
    } else tlZ = 0;

    if (a4) {
        timeT = stof(a4);
    } else timeT = 0;

    std::vector<Ponto> pTrans;

    for(TiXmlElement* pt = t->FirstChildElement("point"); pt ; pt = pt->NextSiblingElement("point")){

    const char *a5 = pt->Attribute("X");
    const char *a6 = pt->Attribute("Y");
    const char *a7 = pt->Attribute("Z");

      if (a5) {
        px = stof(a5);
    } else px = 0;

    if (a6) {
        py = stof(a6);
    } else py = 0;

    if (a7) {
        pz = stof(a7);
    } else pz = 0;



    Ponto ptt = Ponto(px,py,pz);
    cout << "x=" << ptt.getX() << "y=" << ptt.getY() << "z=" << ptt.getZ() << endl;
    pTrans.push_back(ptt);

}

    ops.push_back(new Translacao(tlX, tlY, tlZ,timeT,pTrans));
```

Figura 1-Código referente ao parsing de uma translação.

Relativamente às rotações a única diferença da fase anterior foi a introdução do tempo de rotação.

```
if (!strcmp(t->Value(), "rotate")) {
    const char *a8=t->Attribute("time");
    const char *a9=t->Attribute("angle");

    if (a8) {
        timeR = stof(a8);
    } else timeR = 0;

    if (a9) {
        angRo = stof(a9);
    } else angRo = 0;

    roX = stof(t->Attribute("X"));
    roY = stof(t->Attribute("Y"));
    roZ = stof(t->Attribute("Z"));

    ops.push_back(new Rotacao(timeR,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));
}
```

Figura 2- Código referente ao parsing de uma rotação

2. Curvas de Catmull

2.1. Rotação

Para se poder implementar as novas formas de rotação foi preciso introduzir uma nova variável tempo de forma a tornar possível o cálculo de tempo que demora a fazer uma rotação completa, utilizando a seguinte fórmula:

Utilizando o glutGet com o tempo decorrido desde o início da execução do problema, no entanto, é preciso estabelecer um limite para este valor uma vez que o Glut_elapsed_time vai sempre aumentando. Conseguimos assim determinar o tempo que o objeto demora a efetuar uma rotação completa de 360º. Desta forma, é possível mandar o tempo obtido para o glRotatef que faz o objeto rodar um determinado período de tempo.

```
void Rotacao::run() {

    if((int)tempo==0){
        glRotatef(this->angle, this->x, this->y, this->z);
    }
    else{
        float r = (rAux*360) / (tempo*1000);
        glRotatef(r,x,y,z);

    }
}
```

Figura 3-Fórmula de cálculo de tempo para rotação completa

2.2. Translação

Para se poder implementar as novas formas de translação foi preciso introduzir novas variáveis como o tempo e dois vetores de pontos, um vetor “trans” que guarda os pontos lidos do ficheiro XML e outro vetor “pontosCatmull” que contém os pontos de uma curva Catmull-Rom.

A inserção dos valores no vetor “pontosCatmull” é feita usando a função contida na classe designada de “geraPontosCurva” que utiliza as funções também definidas para obter esses pontos “getCatmullRomPoint” e “getGlobalCatmullRomPoint”.

E necessário também fazer um cálculo do tempo tal como feito na rotação seguindo a seguinte formula:

Depois do cálculo do tempo, podemos chamar a função getGlobalCatmullRomPoint com o tempo determinado em cima e com os vectores “res[3]” e “deriv[3]” responsáveis por guardar os pontos e as suas derivadas.

Por fim, basta apenas chamar o glTranslate3f com os argumentos res[0], res[1] e res[2] para efetuar as translações.

```
void Translacao::run() {
    if((int)tempo==0){
        glTranslatef(x,y,z);
    }
    else {
        geraPontosCurva();
        glBegin(GL_LINE_LOOP);
        for (int i = 0; i < pontosCatmull.size(); ++i) {
            glVertex3f(pontosCatmull[i].getX(), pontosCatmull[i].getY(), pontosCatmull[i].getZ());
        }
        glEnd();

        float res[3];
        float deriv[3];
        getGlobalCatmullRomPoint(t, res, deriv);
        glTranslatef(res[0], res[1], res[2]);
    }

}
```

Figura 4-Fórmula para cálculo de tempo para translação completa

2.3. Desenho da Curva

Para desenhar as curvas que representam as orbitas dos planetas e as trajetórias que eles seguem foi criada uma função run na classe translação que chama a função “geraPontosCurva” que gera 100 pontos dessa orbita e utilizando GL_LINE_LOOP e Gl_vertex3f percorremos o vector pontosCatmull para desenhar a orbita

3. VBOs

De cada vez que é feito um parse a um model, os pontos são inseridos num vetor de vetores de floats, e é incrementada a variável que conta o número de VBOs.

Na main, é executada a função prepareVBOs, que aloca os apontadores para os VBOs e faz bind, e buffer aos arrays.

Numa ultima fase, na função RenderScene, os mesmos são desenhados quando o método run da Classe Desenhar é chamado.

```
vector<vector<float>> vertVBOs;
int nVBOS = 0;
GLuint *buffers;
int iVBO=0;

void prepareVBOs(){

    buffers = (GLuint*) malloc((sizeof(GLuint))*nVBOS);
    glGenBuffers(nVBOS, buffers);

    for(int i = 0; i < nVBOS; ++i){

        glBindBuffer(GL_ARRAY_BUFFER,buffers[i]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(float) * vertVBOs[i].size(), vertVBOs[i].data(), GL_STATIC_DRAW);

    }
}
```

Figura 5-prepareVBOs

```
extern GLuint *buffers;
extern int iVBO;

void Desenhar::run() {

    glBindBuffer(GL_ARRAY_BUFFER, buffers[iVBO]);
    iVBO++;
    glVertexPointer(3,GL_FLOAT,0,0);

    glDrawArrays(GL_TRIANGLES, 0, vertexes);
}
```

Figura 6-DesenharVBOs

Gerador

1. Patch de Bezier

No gerador, foi implementada a função generatePatch, que a partir do ficheiro patch e com o nível de "Tesselation" desejado, devolve um ficheiro 3d com os pontos dos triângulos da superfície.

Numa primeira fase, é feito o parse, ou seja, é lido o ficheiro patch, onde são retirados os o numero de patches, os índices de cada patch, que são guardados num vetor de vetores, o numero de pontos de controlo, e por fim, os pontos de controlo, que são guardados num vetor de pontos.

```
void generatePatch(string patchfile, int tesselation, string filesave){

    string path = "../models/" + patchfile + ".patch";

    ifstream f(path);

    int patches=0;
    int vertices=0;
    vector<vector<int>> indicesPatch;
    vector<Point> PControlo;

    if(f.is_open()){
        //ler patch
        string line;
        if(getline(f,line)) sscanf(line.c_str(),"%d\n",&patches);
        for(int i=0; i<patches ;i++){
            vector<int> aux;
            if(getline(f,line)){

                stringstream str_strm;
                str_strm << line;

                string temp;
                int found;
                while (!str_strm.eof()) {

                    str_strm >> temp;

                    if (stringstream(temp) >> found)
                        aux.push_back(found);

                    temp = "";
                }

            }

            indicesPatch.push_back(aux);
        }

        if(getline(f,line)) sscanf(line.c_str(),"%d\n",&vertices);
        for(int i=0; i<vertices ;i++){
            float x=0,y=0,z=0;
            if(getline(f,line)) sscanf(line.c_str(),"%f, %f, %f\n",&x,&y,&z);

            Point p;
            p.x = x; p.y = y; p.z = z;

            PControlo.push_back(p);
        }

        f.close();
    }
```

Figura 7-Codigo relativo ao parse do ficheiro patch

Numa segunda fase, é feito o cálculo dos pontos, segundo o algoritmo de Bezier, onde vão ser definidas as quatro curvas de Bezier, cada uma formada por quatro vértices.

Os pontos gerados irão ser guardados num ficheiro 3d.

```
double t;
int index, indices[4];
double q[4][tesselation][3],r[tesselation][tesselation][3],tess = 1/((double)tesselation-1);

fstream g;
string path1 = "../models/" + filesave + ".3d";
g.open(path1,std::ios::out);

if(g.is_open()){   //numero de pontos
    for(int n=0; n<patches; n++){
        //recolher os vértices do array com os pontos de controlo para o x y e z
        double p[16][3];
        for(int m=0; m<16; m++){
            p[m][0] = PControlo[indicesPatch[n][m]].x;
            p[m][1] = PControlo[indicesPatch[n][m]].y;
            p[m][2] = PControlo[indicesPatch[n][m]].z;
        }
        int j=0,k=0;
        //desenhar as 4 curvas
        for(int i=0; i<15; i+=4){
            indices[0] = i;
            indices[1] = i+1;
            indices[2] = i+2;
            indices[3] = i+3;
            //calcular a curva
            for(int a=0; a<tesselation-1; a++){
            t = a*tess;
            index = floor(t);
            t = t - index;
            q[j][k][0] = (-p[indices[0]][0] +3*p[indices[1]][0] -3*p[indices[2]][0] +p[indices[3]][0])*t*t*t +
             (3*p[indices[0]][0] -6*p[indices[1]][0] +3*p[indices[2]][0])*t*t + (-3*p[indices[0]][0] +3*p[indices[1]][0])*t + p[indices[0]][0];
            q[j][k][1] = (-p[indices[0]][1] +3*p[indices[1]][1] -3*p[indices[2]][1] +p[indices[3]][1])*t*t*t +
             (3*p[indices[0]][1] -6*p[indices[1]][1] +3*p[indices[2]][1])*t*t + (-3*p[indices[0]][1] +3*p[indices[1]][1])*t + p[indices[0]][1];
            q[j][k][2] = (-p[indices[0]][2] +3*p[indices[1]][2] -3*p[indices[2]][2] +p[indices[3]][2])*t*t*t +
            (3*p[indices[0]][2] -6*p[indices[1]][2] +3*p[indices[2]][2])*t*t + (-3*p[indices[0]][2] +3*p[indices[1]][2])*t + p[indices[0]][2];

            k++;
            }

            t = 1;

            q[j][k][0] = (-p[indices[0]][0] +3*p[indices[1]][0] -3*p[indices[2]][0] +p[indices[3]][0])*t*t*t +
            (3*p[indices[0]][0] -6*p[indices[1]][0] +3*p[indices[2]][0])*t*t + (-3*p[indices[0]][0] +3*p[indices[1]][0])*t + p[indices[0]][0];
            q[j][k][1] = (-p[indices[0]][1] +3*p[indices[1]][1] -3*p[indices[2]][1] +p[indices[3]][1])*t*t*t +
            (3*p[indices[0]][1] -6*p[indices[1]][1] +3*p[indices[2]][1])*t*t + (-3*p[indices[0]][1] +3*p[indices[1]][1])*t + p[indices[0]][1];
            q[j][k][2] = (-p[indices[0]][2] +3*p[indices[1]][2] -3*p[indices[2]][2] +p[indices[3]][2])*t*t*t +
            (3*p[indices[0]][2] -6*p[indices[1]][2] +3*p[indices[2]][2])*t*t + (-3*p[indices[0]][2] +3*p[indices[1]][2])*t + p[indices[0]][2];

            j++;
            k=0;
        }
```

Figura 8-Código relativo ao cálculo dos pontos (1)

```
        for(int j=0; j<tesselation; j++){
            for(int a=0; a<tesselation-1; a++){
                t = a*tess;;
                index = floor(t);
                t = t - index;

                r[j][k][0] = (-q[0][j][0] +3*q[1][j][0] -3*q[2][j][0] +q[3][j][0])*t*t*t + (3*q[0][j][0] -6*q[1][j][0] +3*q[2][j][0])*t*t + (-3*q[0][j][0] +3*q[1][j][0])*t + q[0][j][0];
                r[j][k][1] = (-q[0][j][1] +3*q[1][j][1] -3*q[2][j][1] +q[3][j][1])*t*t*t + (3*q[0][j][1] -6*q[1][j][1] +3*q[2][j][1])*t*t + (-3*q[0][j][1] +3*q[1][j][1])*t + q[0][j][1];
                r[j][k][2] = (-q[0][j][2] +3*q[1][j][2] -3*q[2][j][2] +q[3][j][2])*t*t*t + (3*q[0][j][2] -6*q[1][j][2] +3*q[2][j][2])*t*t + (-3*q[0][j][2] +3*q[1][j][2])*t + q[0][j][2];


                k++;
            }

            t = 1;

            r[j][k][0] = (-q[0][j][0] +3*q[1][j][0] -3*q[2][j][0] +q[3][j][0])*t*t*t + (3*q[0][j][0] -6*q[1][j][0] +3*q[2][j][0])*t*t + (-3*q[0][j][0] +3*q[1][j][0])*t + q[0][j][0];
            r[j][k][0] = (-q[0][j][1] +3*q[1][j][1] -3*q[2][j][1] +q[3][j][1])*t*t*t + (3*q[0][j][1] -6*q[1][j][1] +3*q[2][j][1])*t*t + (-3*q[0][j][1] +3*q[1][j][1])*t + q[0][j][1];
            r[j][k][0] = (-q[0][j][2] +3*q[1][j][2] -3*q[2][j][2] +q[3][j][2])*t*t*t + (3*q[0][j][2] -6*q[1][j][2] +3*q[2][j][2])*t*t + (-3*q[0][j][2] +3*q[1][j][2])*t + q[0][j][2];

            k=0;
        }

        for(int i=0; i<tesselation-1; i++)
            for(int j=0; j<tesselation-1; j++){
                g << r[i][j][0];   g << ", "; g << r[i][j][1];   g << ", "; g << r[i][j][2];   g << '\n';
                g << r[i+1][j][0]; g << ", "; g << r[i+1][j][1]; g << ", "; g <<r[i+1][j][2];  g << '\n';
                g << r[i][j+1][0]; g << ", "; g << r[i][j+1][1]; g << ", "; g << r[i][j+1][2]; g << '\n';

                g << r[i+1][j][0];   g << ", "; g << r[i+1][j][1];   g << ", "; g << r[i+1][j][2];   g << '\n';
                g << r[i+1][j+1][0]; g << ", "; g << r[i+1][j+1][1]; g << ", "; g << r[i+1][j+1][2]; g << '\n';
                g << r[i][j+1][0];   g << ", "; g << r[i][j+1][1];   g << ", "; g << r[i][j+1][2];   g << '\n';
            }
    }
    g.close();
}
else { printf("Nao e possivel abrir o ficheiro\n"); exit(0);

}
```

Figura 9-Cálculo relativo ao cálculo dos pontos (2)

Classes

Nesta fase, as seguintes classes foram atualizadas, tendo as restantes mantido a mesma definição.

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

#endif

#ifndef TRABALHO_TRANSLACAO_H
#define TRABALHO_TRANSLACAO_H

#include "Operacao.h"
#include "Ponto.h"

class Translacao: public Operacao{
    float x, y, z;
    float tempo;
    std::vector<Ponto> trans;
    std::vector<Ponto> pontosCatmull;
    float cima[3]{};

public:
    Translacao();
    Translacao(float x, float y, float z, float t, std::vector<Ponto> trans);
    float getX() {return this->x;}
    float getY() {return this->y;}
    float getZ() {return this->z;}
    float getTempo(){return this->tempo;}
    float* getCima(){return this->cima;}
    std::vector<Ponto> getTrans(){return this->trans;}
    std::vector<Ponto> getPontosCatmull(){return this->pontosCatmull;}
    void setX(float x){this->x=x;}
    void setY(float y){this->y=y;}
    void setZ(float z){this->z=z;}
    void setTempo(float x){ this->tempo = x;}
    void setTrans(std::vector<Ponto> t){ this->trans = t;}
    void setPontosCatmull(std::vector<Ponto> pC){ this->pontosCatmull = pC;}
    static void multMatrixVector(float* m, float* v, float* res);
    void getCatmullRomPoint(float t, float* p0, float* p1, float* p2, float* p3, float* pos, float* deriv);
    void getGlobalCatmullRomPoint(float gt, float *pos, float *deriv);
    std::vector<Ponto> geraPontosCurva();
    void run();
};


#endif //TRABALHO_TRANSLACAO_H
```

Figura 10-Classe Translação

```
#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{

    int vertexes;

public:
    Desenhar(int vertexes){
        this->vertexes=vertexes;
    };

    void run();

};
```

Figura 11-Classe Desenhar

```
#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 tempo,angle, x, y, z;

public:
    Rotacao();
    Rotacao(float tempo, float angle, float x, float y, float z);
    float getTempo(){return this->tempo;}
    float getAngle(){return this->angle;}
    float getX() {return this->x;}
    float getY() {return this->y;}
    float getZ() {return this->z;}
    void setTempo(float tempo){this->tempo=tempo;}
    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
```

Figura 12-Classe Rotação

Configuração ficheiro XML

Foram feitas alterações à transformação “Rotate”, uma vez que foi alterado o angulo para tempo. No translate, foi introduzido o parâmetro “tempo”, e é fornecida uma lista de pontos, por onde o planeta passa.

```
        <!--MERCURIO-->
<group>
    <color R="0.3764" G="0.3764" B="0.3764" />
    <rotate time="15" X="0" Y="1" Z="0"/>
    <translate time="60">
            <point X="30.0" Y="0" Z="0.0"/>
            <point X="27.406363729278027" Y="0" Z="12.202099292274005"/>
            <point X="20.073918190765745" Y="0" Z="22.294344764321824"/>
            <point X="9.270509831248424" Y="0" Z="28.531695488854606"/>
            <point X="-3.1358538980296" Y="0" Z="29.835656861048204"/>
            <point X="-14.999999999999993" Y="0" Z="25.98076211353316"/>
            <point X="-24.27050983124842" Y="0" Z="17.633557568774197"/>
            <point X="-29.344428022014167" Y="0" Z="6.237350724532792"/>
            <point X="-29.34442802201417" Y="0" Z="-6.237350724532772"/>
            <point X="-24.270509831248425" Y="0" Z="-17.63355756877419"/>
            <point X="-15.000000000000014" Y="0" Z="-25.980762113533153"/>
            <point X="-3.135853898029627" Y="0" Z="-29.8356568610482"/>
            <point X="9.270509831248416" Y="0" Z="-28.53169548885461"/>
            <point X="20.073918190765735" Y="0" Z="-22.29434476432184"/>
            <point X="27.406363729278016" Y="0" Z="-12.202099292274028"/>
        </translate>
    <scale X="0.0350" Y="0.0350" Z="0.0350" />
    <models>
        <model file="sphere.3d" />
    </models>
</group>
```

Figura 13-Pequeno excerto do ficheiro XML

Resultados

Após todo o trabalho, estes são os resultados apresentados.

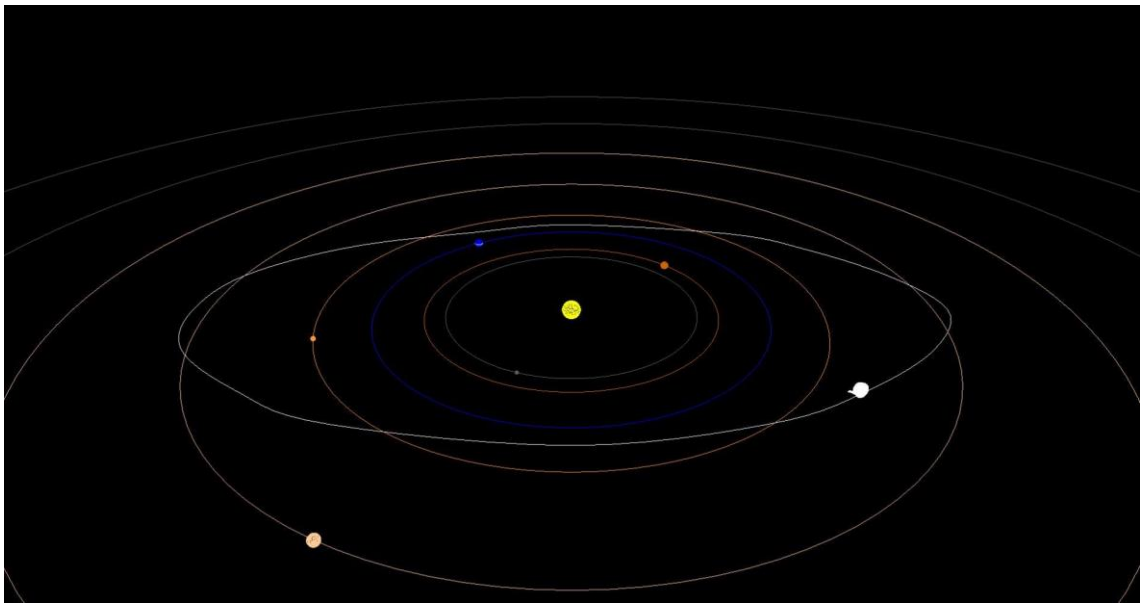


Figura 14-Sistema Solar (1)

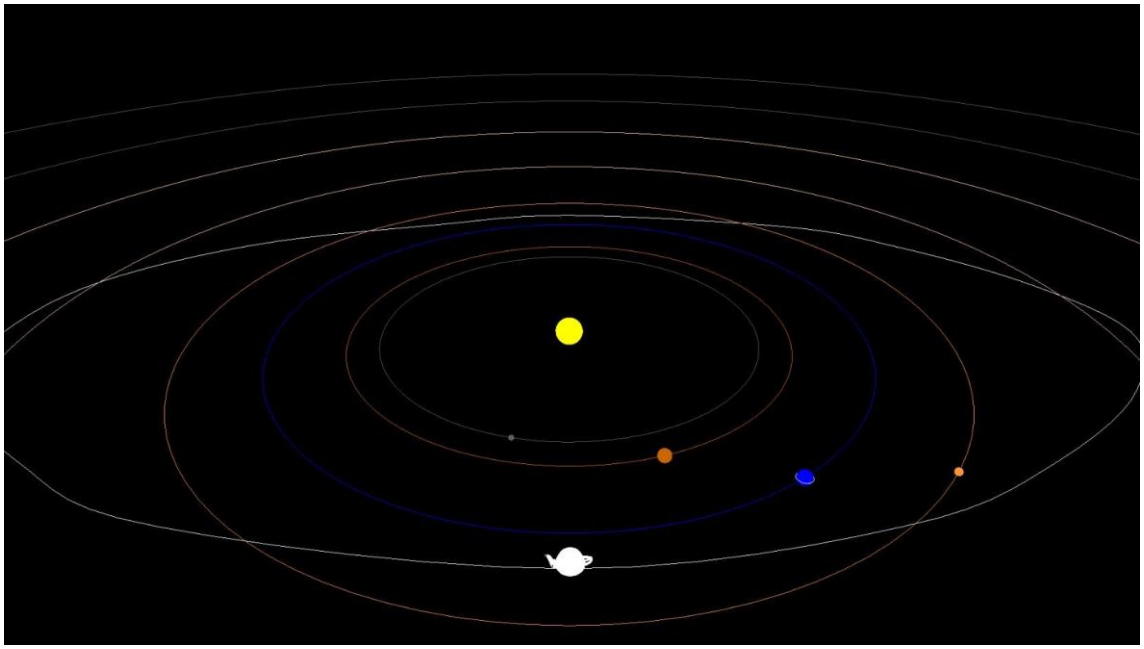


Figura 15-Sistema Solar (2)

Conclusão

Esta fase do trabalho foi importante pois conseguimos apresentar uma melhoria do Sistema Solar, usufruindo introdução de Curvas de Catmull, Patches de Bezier e VBOs.

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 a próxima fase com a introdução de novas técnicas com vista á otimização dos processos de desenho e renderizaçao bem como os FPS.