­­

**Universidade do Minho**

Escola de Engenharia

Mestrado integrado em Engenharia Informática

**Computação Gráfica**

Ano Letivo de 2016/2017

**Fase 3 – Curvas, Superficies Cúbicas e VBOs**

**A27748 - Gustavo José Afonso Andrez**

**A74634 - Rogério Gomes Lopes Moreira**

**A76507 - Samuel Gonçalves Ferreira**

**A71835 – Tiago Filipe Oliveira Sá**

Abril, 2016

7 de maio de 2017

# Índice

[Índice 2](#_Toc481948039)

[Introdução 3](#_Toc481948040)

[Gerador 4](#_Toc481948041)

[Superficíes de Bezier 4](#_Toc481948042)

[Drawer 7](#_Toc481948043)

[VBOs 7](#_Toc481948044)

[Superfícies de Catmull-Rom 7](#_Toc481948045)

[Alterações do Ficheiro XML 11](#_Toc481948046)

[Sistema Solar 12](#_Toc481948047)

[Conclusão 13](#_Toc481948048)

Introdução

O trabalho aqui apresentado, do qual este relatório diz respeito à Fase 3, tem como objetivo desenvolver competências na área dos gráficos 3D e no desenvolvimento de um motor para tal. O trabalho global está dividido em 4 fases.

Esta terceira etapa do trabalho está dividida em três partes distintas:

* **Gerador** - Implementação de um novo tipo de modelo baseado nos Bezier patches. O gerador recebe como parâmetros o ficheiro onde os pontos de controlo de Bezier estão, assim como o nível de tesselation. O ficheiro resultante contém um conjunto de triângulos para desenhar a superfície.
* **Drawer -** Extensão dos elementos translate e rotate. No caso da translação são fornecidos um número de pontos que definem uma curva do tipo cúbico Catmull-Rom e o tempo (em segundos) que o objeto tem para percorrer a curva. Na rotação o ângulo foi substituído pelo tempo que o objeto tem para fazer uma rotação completa de 360º. Para além disso, as figuras são agora desenhadas com VBOs.
* **Ficheiro XML -** No ficheiro XML foram feitas transformações para contemplar o vetor de pontos das translações e o ângulo das rotações foi substituído pelo tempo.

As aplicações foram desenvolvidas recorrendo ao Visual Studio e à linguagem de programação C++.

Gerador

Superficíes de Bezier

As superfícies de Bezier são superfícies expressas como a interpolação linear entre pontos de controlo. A principal característica destas superfícies de Bezier em geral não passa pelos pontos de controlo centrais, mas é esticada na direção de cada um deles como se cada ponto de controlo fosse um ponto de atração. A inicialização das superfícies tem como argumentos o ficheiro patch (fornecido), o grau de tesselação (*Tessellation*) e o ficheiro de output.

Assim, é possível agora a criação de formas irregulares com o objetivo de servirem como cometas neste trabalho. Para tal, usamos a função patch que, passando como argumento o ficheiro.patches, o nível de tesselação e o ficheiro .3d resultante, gera os pontos da forma. De realçar que o ficheiro.patch foi previamente fornecido pela equipa docente.

string patches(string fileIn, float tesselLvl) {

Iniciadas as variaveis locais;

Abertura do ficheiro .patch;

Leitura do número de patches a aplicar;

    for (i = 0; i < patches; i++) {

leitura de 16 indicies do patch;

    }

Leitura do número de pontos de controlo;

    for (i = 0; i < points; i++) {

        Leitura das coordenadas do control point (3 coordenadas);

    }

renderPatch(points, controlPoints, patches, indices, tesselLvl);

}

A função patch inova a função renderPatch que, com os patches e os pontos de controlo lidos pela função patches, gera os pontos da forma. Em cada ciclo da função em baixo descrita, são escritos seis pontos no ficheiro resultante. O incremento em cada ciclo depende do nível de tesselação escolhido, ou seja, este nível aproxima-se do conceito de slices e stacks na geração de, por exemplo, cones ou esferas.

string renderPatch(int points, float \*\* controlPoints, int patches, int \*\* indices, float tessel) {

Inicialização das variáveis locais;

Abertura do ficheiro .3d para escrita;

for (int patch = 0; patch < patches; patch++) {

       int \* indicesPatch = indices[patch];

       for (v = 0; v < 1; v += tesselation) {

           for (u = 0; u < 1; u += tesselation) {

//Triangulo 1

Invocação da função getBezierPoint para encontrar as respetivas coordenadas do ponto P2;

res[0] = getBezierPoint(u, v, 0, controlPoints, indicesPatch);

res[1] = getBezierPoint(u, v, 1, controlPoints, indicesPatch); res[2] = getBezierPoint(u, v, 2, controlPoints, indicesPatch); Escrita do ponto P2;

Invocação da função getBezierPoint para encontrar as respetivas coordenadas do ponto P3;

res[0] = getBezierPoint(u+ tesselation, v, 0, controlPoints, indicesPatch);

res[1] = getBezierPoint(u+ tesselation, v, 1, controlPoints, indicesPatch);

res[2] = getBezierPoint(u+ tesselation, v, 2, controlPoints, indicesPatch);

Escrita do ponto P3;

Invocação da função getBezierPoint para encontrar as respetivas coordenadas do ponto P1;

res[0] = getBezierPoint(u, v+ tesselation, 0, controlPoints, indicesPatch);

res[1] = getBezierPoint(u, v+ tesselation, 1, controlPoints, indicesPatch);

res[2] = getBezierPoint(u, v+ tesselation, 2, controlPoints, indicesPatch);

Escrita do ponto P1;

//Triangulo 2

Invocação da função getBezierPoint para encontrar as respetivas coordenadas do ponto P1;

res[0] = getBezierPoint(u, v + tesselation, 0, controlPoints, indicesPatch);

res[1] = getBezierPoint(u, v + tesselation, 1, controlPoints, indicesPatch);

res[2] = getBezierPoint(u, v + tesselation, 2, controlPoints, indicesPatch);

Escrita do ponto P1;

Invocação da função getBezierPoint para encontrar as respetivas coordenadas do ponto P3;

res[0] = getBezierPoint(u + tesselation, v, 0, controlPoints, indicesPatch);

res[1] = getBezierPoint(u + tesselation, v, 1, controlPoints, indicesPatch);

res[2] = getBezierPoint(u + tesselation, v, 2, controlPoints, indicesPatch);

Escrita do ponto P3;

Invocação da função getBezierPoint para encontrar as respetivas coordenadas do ponto P4;

res[0] = getBezierPoint(u + tesselation, v + tesselation, 0, controlPoints, indicesPatch);

res[1] = getBezierPoint(u + tesselation, v + tesselation, 1, controlPoints, indicesPatch);

res[2] = getBezierPoint(u + tesselation, v + tesselation, 2, controlPoints, indicesPatch);

Escrita do ponto P4;

ret << "FIM";

return ret.str();

}

Com base nisto e com o que foi apreendido ao longo das aulas de Computação Gráfica, desenvolvemos esta função que, dado o nível de tesselação pretendido, os pontos de controlo e os índices dos patches, retorna a coordenada de um ponto. Este processo é feito usando a multiplicação de matrizes.

float getBezierPoint(float u, float v, int coord, float \*\* vertices, int \* indices) {

float pointValue = 0;

    float bu[4][1] = { { powf(1 - u, 3) },{ 3 \* u \* powf(1 - u, 2) },{ 3 \* powf(u, 2) \* (1 - u) },{ powf(u, 3) } };

    float bv[4][1] = { { powf(1 - v, 3) },{ 3 \* v \* powf(1 - v, 2) },{ 3 \* powf(v, 2) \* (1 - v) },{ powf(v, 3) } };

    for (int i = 0; i < 4; i++) {

        for (int j = 0; j < 4; j++) {

            pointValue += vertices[indices[j + 4 \* i]][coord] \* bu[i][0] \* bv[j][0];

        }

    }

    return pointValue;

}

Drawer

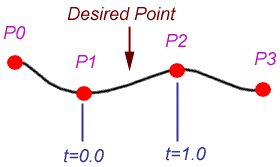
VBOs

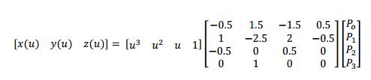
Ao contrário do que foi desenvolvido nas etapas anteriores, nesta etapa em vez do modo imediato para desenhar as primitivas foram usados VBOs (Vertex Buffer Object), proporcionando métodos de encaminhamento de dados para a placa gráfica sem que estes sejam logo desenhados. Assim, é possível obter ganhos de performance, na medida em que os dados das superfícies a serem desenhadas já se encontram na placa e não na memória.

As alterações para incluir os VBOs foram feitas seguindo o guião disponibilizado pelo professor.

Superfícies de Catmull-Rom

As órbitas dos objetos espaciais são definidas por superfícies de Catmull-Rom. A principal diferença deste tipo de superfície para as de Bezier são os pontos de controlo, que neste caso estão contidos na própria superfície. Desta forma, as órbitas tendem a ficar mais suaves e mais próximas da realidade.

****

****

Para a implementação das superfícies de Catmull-Rom foi estendida a estrutura criada na fase anterior que permite armazenar um grupo de planetas e, de maneira recursiva, os seus filhos. A estrutura é agora a seguinte:

class Group {

public:

vector<Point> points;

float translateTime;

vector<Point> controlPoints;

Point rotation;

Point scale;

float rotateTime;

vector<Group>childs;

float deltaT;

bool hasModels;

};

A obtenção dos pontos para as superfícies de Catmull-Rom está dividido em duas funções diferentes que passamos a explicar.

void getCatmullRomPoint(float t, float \*p0, float \*p1, float \*p2, float \*p3, float \*res, float \*deriv) {

//Declaração da matriz

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 } };

// Inicialização das variáveis res e deriv

//Declaração da Matriz A

// Cálculo A = M \* P

for (i<3){

Preenchimento da linha da matriz;

Multiplicação da Matriz ((float\*)m, pl, A[i]);

}

Cálculo do ponto res = T \*A

Cálculo da derivada = T' \* A

for(j<3){

for(i<4){

res[j] += v\_t[i] \* A[j][i];

deriv[j] += v\_td[i] \* A[j][i];

}}}

A função descrita em cima calcula um ponto deste tipo de superfície. Esta função é invocada pela função *getGlobalCatmullRomPoint*. Dado um t global retorna o ponto da curva a desenhar.

void getGlobalCatmullRomPoint(float gt, float \*pos, float \*deriv, vector<Point> controlPoints) {

//Declaração da variável de contagem dos pontos de controlo;

//Declaração do t global

//Segmento

// O índices guardam os pontos

indices[0] = (index + pointCount - 1) % pointCount;

indices[1] = (indices[0] + 1) % pointCount;

indices[2] = (indices[1] + 1) % pointCount;

indices[3] = (indices[2] + 1) % pointCount;

float \*\*p = new float\*[pointCount];

for (i < pointCount) {

p[i][0] = controlPoints[i].x;

p[i][1] = controlPoints[i].y;

p[i][2] = controlPoints[i].z;

}

getCatmullRomPoint(t, p[indices[0]], p[indices[1]], p[indices[2]], p[indices[3]], pos, deriv);

}

Por fim, tendo em conta os pontos calculados nas duas funções demonstradas em cima é necessário fazer o render das curvas, tendo em conta não só as translações como as rotações dos planetas. A função *drawScene* é invocada na função *renderScene*.

void drawScene(vector<Group>groups) {

for (unsigned i = 0; i < groups.size(); i++) {

Group group = groups[i];

float time = glutGet(GLUT\_ELAPSED\_TIME)\*0.001;

glPushMatrix();

//Aplica as transformações

if (group.controlPoints.size() > 3) {

//Desenha a órbita

}

glEnd();

float res[3];

float deriv[3];

float M[4][4];

M[0][3] = M[1][3] = M[2][3] = 0.0f;

M[3][3] = 1.0f;

getGlobalCatmullRomPoint(t[counterT], (float\*)res, (float\*)deriv, group.controlPoints);

t[counterT++] = time / group.translateTime;

normalize(deriv);

M[0][0] = deriv[0];

M[0][1] = deriv[1];

M[0][2] = deriv[2];

M[3][0] = res[0];

M[3][1] = res[1];

M[3][2] = res[2];

/\* Fill Z row\*/

float Z[3];

cross(deriv, up, Z);

normalize(Z);

M[2][0] = Z[0];

M[2][1] = Z[1];

M[2][2] = Z[2];

/\* up => cross product: Z x deriv\*/

cross(Z, deriv, up);

normalize(up);

M[1][0] = up[0];

M[1][1] = up[1];

M[1][2] = up[2];

glMultMatrixf((float\*)M);

}

glScalef(group.scale.x, group.scale.y, group.scale.z);

if (group.rotateTime > 0) {

glRotatef(rot[counterRot], group.rotation.x, group.rotation.y, group.rotation.z);

rot[counterRot++] = time / group.rotateTime \* 360;

}

drawScene(group.childs);

if (group.hasModels) {

glBindBuffer(GL\_ARRAY\_BUFFER, buffers[counterBuff]);

glVertexPointer(3, GL\_FLOAT, 0, 0);

glDrawArrays(GL\_TRIANGLES, 0, models[counterBuff++].size()\*3);

}

glPopMatrix();

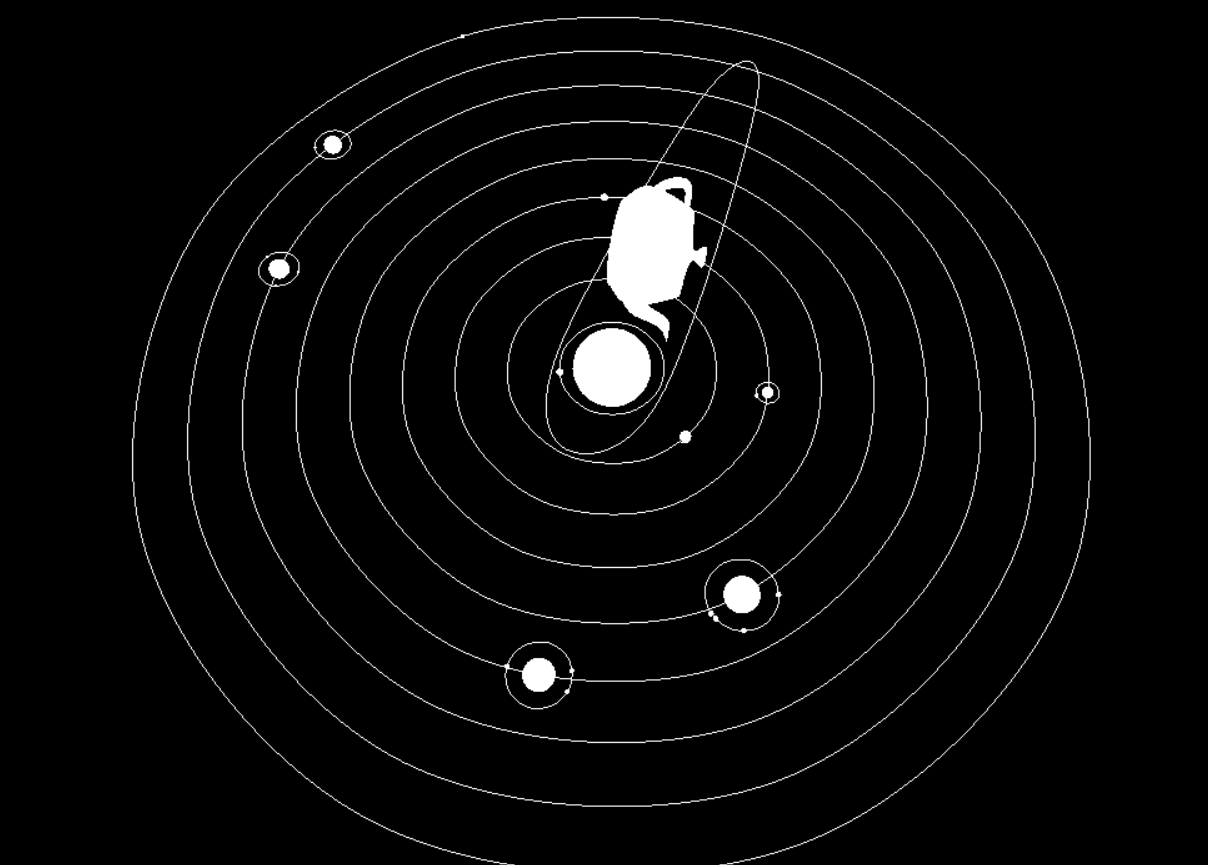
}

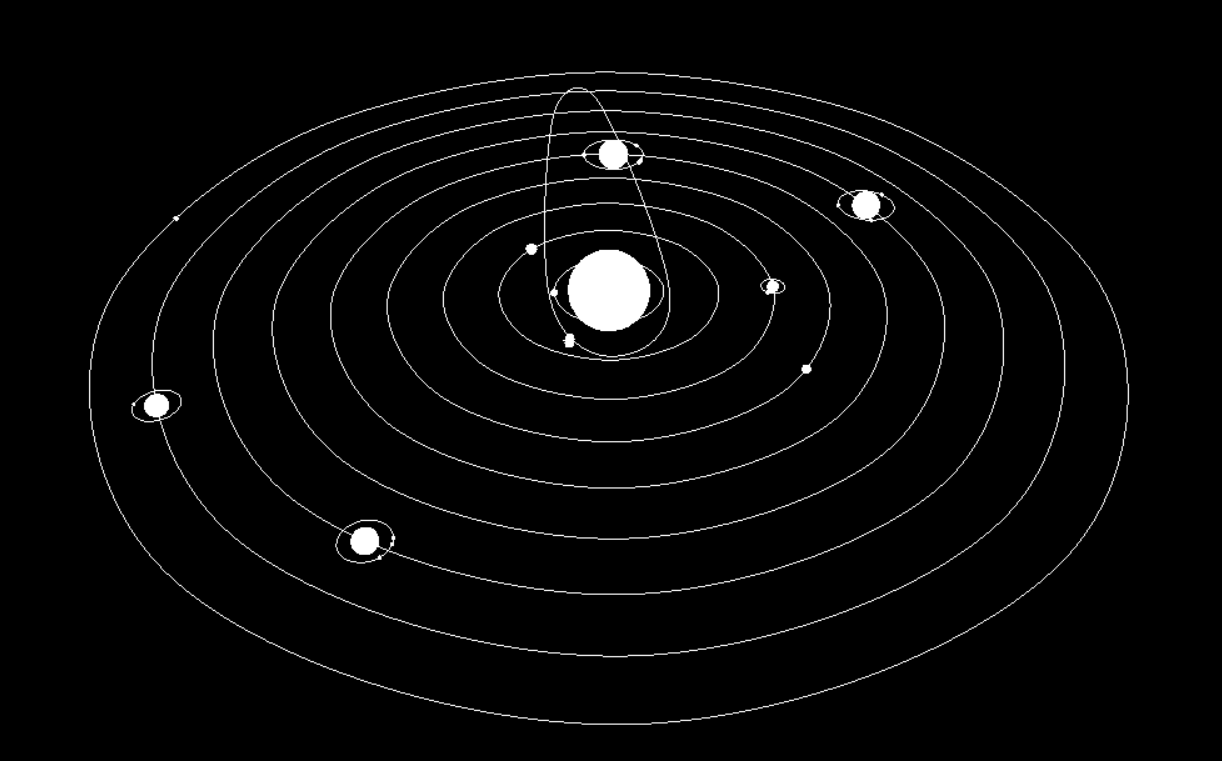
}

Alterações do Ficheiro XML

Relativamente ao ficheiro XML com os dados relativos ao Sistema Solar foram feitas alterações nos campos “translate” e “rotate”. Relativamente à translação, onde na fase anterior eram descriminadas as coordenadas dos 3 eixos, foi alterado para o seguinte um formato onde é explicitado o parâmetro tempo (time) correspondente à duração do astro efetuar a respetiva translação e também foram incluídas as coordenadas de oito pontos da trajetória de translação. Quanto ao campo de rotação, o parâmetro angulo (angle) presente na fase anterior foi alterado para o parâmetro tempo (time) correspondente ao período de uma revolução em torno do eixo do astro em questão. As coordenadas do eixo foram mantidas no mesmo formato.

Sistema Solar





Conclusão

Com elaboração desta fase do trabalho ficamos a entender a forma de aplicação das curvas de Bezier e os VBOs.

Considerando o desenvolvimento do trabalho, foram inicialmente cumpridos dois dos três objetivos. Foram implementadas as superfícies de Bezier, foram feitas as alterações relativas aos VBOs e o ficheiro XML foi também modificado. Tinham ficado por implementar as superfícies de Catmull-Rom, que implementamos agora e damos por isso concluída esta fase do trabalho. Salientamos ainda a nossa preocupação por dar como concluída a Fase 2, implementando com rigor e interesse o que faltava.