Universidade do Minho
Mestrado Integrado em Engenharia Informática

Computação Gráfica

Fase 3 - Grupo 51
Abril 2020

Figure 1: A89536
Ana Gomes

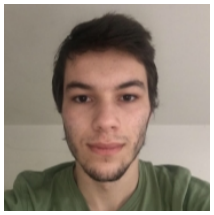

Figure 2: A89536
Francisco Franco

Figure 3: A89525
Maria Barros

Figure 4: A89535
Beatriz Lacerda

1 Introdução e Objetivos

Nesta Unidade Curricular de Computação Gráfica, foi-nos dada uma sugestão do desenvolvimento de modelos 3D, utilizando OpenGL com base na biblioteca GLUT e todo o projeto foi feito na linguagem C++.
Este foi dividido em 4 fases distintas, sendo que nesta 3ª fase o principal objetivo era, criar um sistema solar dinâmico através de translações e rotações que variassem com a passagem do tempo e a inclusão de um cometa que seria contruido utilizando patches de Bezier, com os pontos de controlo para o teapot. Todas as trajetórias seriam definidas recorrendo a curvas de Catmull-Rom.

2 Resumo

Para esta segunda fase do projeto, foi necessário alargar os elementos de translação e rotação para permitir animações com passagem de tempo.
Além disso também criamos novas funções ao gerador para permitir efetuar a leitura de um ficheiro patch, com os pontos de controle do teapot, calcular os restantes pontos e guardar num ficheiro 3d.
Para além disso passamos a desenhar o sistema solar recorrendo a VBOs.

3 Estrutura do projeto

3.1 Módulos

Anteriormente na primeira fase do projeto, já tinhamos explicado a ultilidade de cada módulo e como cada um destes funcionava, no entanto para esta segunda fase houve algumas alterações, nomeadamente a criação de uma nova estrutura XML, a serem feitas e por isso é imprescindível falarmos novamente dos mesmos.

3.1.1 Generator

O **generator.cpp**, à semelhança do que foi explicado na primeira fase, é o módulo onde estão determinadas as estruturas para cada uma das formas geométricas a representar, com o objetivo de gerar os vértices das figuras. Nesta fase, mantivemos idênticas as restantes classes que já tinhamos desenvolvido e criamos uma nova classe no Generator para leitura e calculo do patch de Bezier.

3.1.2 Engine

O **engine.cpp** contém as funcionalidades principais do projeto, como por exemplo a interpretação e leitura dos ficheiros XML.
Após a criação das extenções daas transformações no XML, tivemos a necessidade de alterar o método usado para efetuar o *parsing* e alteramos as classes Translate e Rotate para conterem as novas informações dadas no XML. Para além disso, também adicionamos algumas funções na classe Group para permitir que os modelos fossem desenhados com recurso a VBOs.

3.2 Classes

Como já foi anteriormente dito, para esta terceira fase do projeto, necessitamos de interpretar uma estrutura XML diferente. Assim, procedemos à alteração de algumas classes já existentes que nos irão ajudar nesse sentido.

3.2.1 Translação

A classe **Translate** contém tudo que é necessário para efetuar uma translação tanto estática ,como na fase anterior, como dinâmica, com recurso às variáveis de instancia time, pontos e curva, que irão descrever o trajeto que o modelo tem de percorrer no tempo definido.

```
#ifndef Engine_Translate_H
#define Engine_Translate_H

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <cstring>
#include <vector>
#include "Ponto.h"
#include <math.h>

using  namespace std;

class Translate{
        private:
        float time;
        vector<Ponto> pontos;
        vector<Ponto> curva;
        float x;
        float y;
        float z;

        public:
        Translate();
        Translate (float x, float y, float z);
        Translate (float time, vector<Ponto> p);
        float getX();
        float getY();
        float getZ();
        float getTime();
        vector<Ponto> getPontos();
        vector<Ponto> getCurva();
        void setX(float a);
        void setY(float a);
        void setZ(float a);
        void setTime(float time);
        void addPonto(Ponto p);
        void getCatmullRomPoint(float t, Ponto p0, Ponto p1, Ponto p2, Ponto p3, float *pos,float* deriv);
        void getGlobalCatmullRomPoint(float gt, float *pos,float* deriv);
        void desenhaCurvas();
};
```

Figure 5: Translate.h

3.2.2 Rotação

A classe **Rotate** contém o necessário para efetuar uma rotação estática, como na fase anterior, mas também dinâmica com a variável de instância time que vai permitir que o modelo faça uma rotação de 360º no intervalo de tempo dado.

```
#ifndef Engine_Rotate_H
#define Engine_Rotate_H

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <cstring>
#include <vector>

using  namespace std;

class Rotate{
        private:
        float time;
        float angle;
        float x;
        float y;
        float z;

        public:
        Rotate();
        Rotate (float angle, float x, float y, float z);
        float getAngle();
        float getTime();
        float getX();
        float getY();
        float getZ();
        void setAngle(float a);
        void setTime(float a);
        void setX(float a);
        void setY(float a);
        void setZ(float a);
};

#endif //Engine_Rotate_H
```

Figure 6: Rotate.h

3.2.3 Patch

A classe **Patch** contém as funções que permitem ler o ficheiro teapot.patch e calcular os pontos para gerar o ficheiro 3d.

```
#ifndef PROJETO_CG_PATCH_H
#define PROJETO_CG_PATCH_H

#define _USE_MATH_DEFINES

#include <math.h>
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>

using namespace std;

float* BezierCubica(float u, float* p0, float* p1, float* p2, float* p3);
float* Bezier(float u, float v, int* patches, float** pontos);
void parse(string file, int tess, string nome);

#endif //PROJETO_CG_PATCH_H
```

Figure 7: Patch.h

3.2.4 Grupo

À classe **Group** foram adicionadas as variáveis de instância **buffer** e **nvertices** para guardar o identificador do buffer que contém os pontos para desenhar o modelo nos VBO.

```
#ifndef Engine_Group_H
#define Engine_Group_H

#include <stdlib.h>
#include <vector>
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glew.h>
#include <GL/glut.h>
#endif

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <cstring>
#include <vector>
#include "Transformation.h"
#include "Ponto.h"

using  namespace std;

class Group{
        private:
        string nome;
        Transformation t;
        vector<Group> filhos =vector<Group>();
        vector<Ponto> pontos = vector<Ponto>();
        GLuint buffer[1];
        GLsizei nvertices;


        public:
        Group(string n, Transformation tf, vector<Group> g, vector<Ponto> p);
        vector<Ponto> getPontos();
        vector<Group> getFilhos();
        Transformation getTrans();
        void imprime();
        void prep();
        void draw();
};
```

Figure 8: Group.h

4 Primitivas Geométricas

4.1 Generator

4.1.1 Teapot

De forma a ser possivel ler o ficheiro patch foram adicionadas 3 funções. A função parse trata de fazer o parsing do ficheiro patch e guardar informações relativas aos patches e pontos de controle em vetores. Após o parsing a função contém 3 loops aninhados que funcionam da seguinte forma:

- O ciclo mais exterior trata de percorrer todos os patches;
- Os próximos 2 ciclos são repetidos dependendo do númrero de tesselagem pretendido e a cada iteração mais interior vão ser calculados 4 pontos de Bezier que são então guardados no ficheiro 3d por forma a formarem triângulos.

```
float t = 1.0/tess, u, v, u2, v2;
float*** res = new float**[npatch];

  for(int r = 0; r<npatch; r++){
    res[r] = new float*[4];

    for(int nu = 0; nu < tess; nu++){

      for(int nv = 0; nv < tess; nv++){

        cout<< nv << endl;

        u = nu * t;
        v = nv * t;
        u2 = (nu + 1) * t;
        v2 = (nv + 1) * t;

        res[r][0] = Bezier(u,v, patches[r], pontos);
        res[r][1] = Bezier(u2,v, patches[r], pontos);
        res[r][2] = Bezier(u,v2, patches[r], pontos);
        res[r][3] = Bezier(u2,v2, patches[r], pontos);

        fileo << res[r][0][0] << "," << res[r][0][1] << "," << res[r][0][2] << endl;
        fileo << res[r][2][0] << "," << res[r][2][1] << "," << res[r][2][2] << endl;
        fileo << res[r][1][0] << "," << res[r][1][1] << "," << res[r][1][2] << endl;

        fileo << res[r][1][0] << "," << res[r][1][1] << "," << res[r][1][2] << endl;
        fileo << res[r][2][0] << "," << res[r][2][1] << "," << res[r][2][2] << endl;
        fileo << res[r][3][0] << "," << res[r][3][1] << "," << res[r][3][2] << endl;
    }
  }
}
```

Figure 9: Parse.h

As outras 2 funções tratam de interpolar um ponto. Para realizar este cálculo foi usada a seguinte formula: P(t) = $(1\text{-t})^3 p0 + 3t(1-t)^2 p1 + 3t^2(1-t)p2 + 3t^3 p3$.

A função Bezier vai primeiro interpolar um ponto a cada 4 pontos de um patch passando como primeiro argumento da função Bezier a variável **v**. A função Bezier recorre então á fórmula acima para interpolar um ponto. Os 4 pontos calculados com recurso há variavel **v** são então dados como argumento á função Bezier mas desta vez o primeiro argumento desta é a variável **u**. É então desta forma que para cada patch se trata de interpolar um ponto. Utilizando o seguinte comando é então possível gerar ficheiro 3d com recurso a ficheiros patch:

./Generator Patch <input file> <tesselation> <output file>

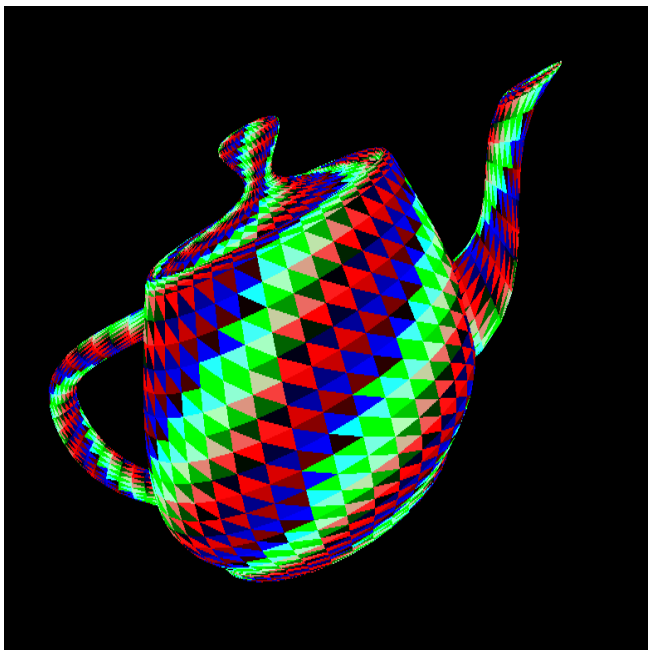

Figure 10: Teapot gerado do patch

5 Engine

5.1 Processo de leitura

Nesta fase no parsing do XML era necessário lidar com as alterações pretendidas.

5.2 Processamento dos Translates e Rotates

Os translates agora podem aparecer de 2 formas, como na fase anterior em que é dada uma coordenada para onde o modelo deve ser deslocado e uma nova forma onde é passado um tempo e um conjunto com um mínimo de 4 pontos de controle. O objetivo desta segunda forma é utilizar dos pontos de controle para calcular uma curva Catmul-Rom que deve ser percorrida pelo modelo no tempo passado como argumento, criando assim desta forma animações. A função desenhaCurvas calcula todos os pontos da curva de Catmul-Rom e guarda-os na variável de instância **curva** para mais tarde podermos desenhar as órbitas.

```
for(XMLElement* elemento = group->FirstChildElement();(strcmp(elemento->Name(),"models")!=0); elemento = elemento -> NextSiblingElement()){
  if(strcmp(elemento->Name(),"translate")==0){
    if(elemento->Attribute("time")){
      translate.setTime(stof(elemento->Attribute("time")));
      for(XMLElement* elem = elemento->FirstChildElement();elem;elem = elem->NextSiblingElement()){
        if(strcmp(elem->Name(),"ponto")==0){
          Ponto p = Ponto(stof(elem->Attribute("X")),stof(elem->Attribute("Y")),stof(elem->Attribute("Z")));
          cout << p.getX() << endl;
          translate.addPonto(p);
        }
      }
      translate.desenhaCurvas();
    }
    else{
      parseTranslate(elemento, &translate);
    }
```

Figure 11: Parsing do novo translate

O parsing do Rotate só tem uma diferença, em vez de receber um angulo, pode receber um tempo que é mais tarde utilizado para efetuar rotações constantes ao modelo com passagem do tempo.

5.3 VBOs

Para que o Engine fosse capaz de desenhar em non-immediate-mode, foi necessário incluir a biblioteca GLEW e o utlizar o método glEnableClientState(GL_VERTEX_ARRAY) na função main. Para termos a certeza que os modelos só são renderizados uma vez, realizamos a preparação do buffer aquando da criação de um **Group** no parsing do XML. Para isso recorremos á função prep, que inclui métodos para inicializar e preencher Vertex Buffer Objects que serão armazendos nas novas variáveis de instância do **Group** e posteriormente utilizadas.

```
void Group::prep(){
  int tam = pontos.size();
  int n=0;
  int i = 0;
  vector<float> v = vector<float>();

  for(i;i<tam;i++){
    v.push_back(pontos[i].getX());
    v.push_back(pontos[i].getY());
    v.push_back(pontos[i].getZ());
  }

  nvertices = v.size()/3;

  glGenBuffers(1,buffer);
        glBindBuffer(GL_ARRAY_BUFFER, buffer[0]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(float) * v.size(), v.data(), GL_STATIC_DRAW);

}
```

Figure 12: Método de preparação dos buffers

5.4 Ciclo de Rendering

Este ciclo funciona de forma idêntica á fase 2 com exceção da obtenção das transformações translate e rotate que agora variam com a passagem do tempo. A forma como os modelos são desenhados também muda, sendo que agora chamamos a função draw para renderizar os modelos em non-immediate-mode.

```
Rotate rt = t.getRotate();
Translate tl = t.getTranslate();
if(tl.getTime()>0){
    float r = glutGet(GLUT_ELAPSED_TIME) % (int)(tl.getTime() * 1000);
                            float gt = (float) r / (tl.getTime() * 1000);
                            renderCatmullRomCurve(tl.getCurva());
                            tl.getGlobalCatmullRomPoint(gt,res,deriv);
                            glTranslatef(res[0],res[1],res[2]);
    direction(deriv,y,z,m);
}
else{
  glTranslatef(t.getTranslate().getX(),t.getTranslate().getY(),t.getTranslate().getZ());
}
if(rt.getTime()>0){
  float r = glutGet(GLUT_ELAPSED_TIME) % (int)(rt.getTime() * 1000);
  float ang = r *360 / (rt.getTime() * 1000);
  glRotatef(ang, rt.getX(),rt.getY(),rt.getZ());
}
else{
glRotatef(t.getRotate().getAngle(), t.getRotate().getX(),t.getRotate().getY(),t.getRotate().getZ());
```

Figure 13: Obtenção das transformações

```
void Group::draw(){
  glBindBuffer(GL_ARRAY_BUFFER, buffer[0]);
        glVertexPointer(3 , GL_FLOAT, 0 , 0);
        glDrawArrays(GL_TRIANGLES,0, nvertices);
}
```

Figure 14: Desenho do modelo

6 Análise de resultados

Refletindo sobre o trabalho realizado, o grupo está satisfeito com os resultados visto que todos os elementos do Sistema Solar estão representados de um modo muito semelhante ao que eles são na realidade.

6.1 Visualização geral

```
<!--Cometa-->

<group>
  <scale X="0.1" Y="0.1" Z="0.1"/>
  <translate time = "30">
    <point X="50" Y="30" Z="-230" />
      <ponto X="46.194" Y="20" Z="-134.329" />
      <ponto X="35.3553" Y="10" Z="-53.2233" />
      <ponto X="19.1342" Y="0" Z="0.969883" />
      <ponto X="0" Y="-2" Z="20" />
      <ponto X="-19.1342" Y="0" Z="0.969883" />
      <ponto X="-35.3553" Y="10" Z="-53.2233" />
      <ponto X="-46.194" Y="20" Z="-134.329" />
      <ponto X="-50" Y="30" Z="-230" />
      <ponto X="-46.194" Y="40" Z="-325.671" />
      <ponto X="-35.3553" Y="50" Z="-406.777" />
      <ponto X="-19.1342" Y="60" Z="-460.97" />
      <ponto X="-0" Y="65" Z="-480" />
      <ponto X="19.1342" Y="60" Z="-460.97" />
      <ponto X="35.3553" Y="50" Z="-406.777" />
      <ponto X="46.194" Y="40" Z="-325.671" />
  </translate>
  <cor R="0.3" G="0" B="0.9"/>
  <models>
    <model file = "../3d/teapot.3d"/>
  </models>
</group>
```

Figure 15: XML do cometa

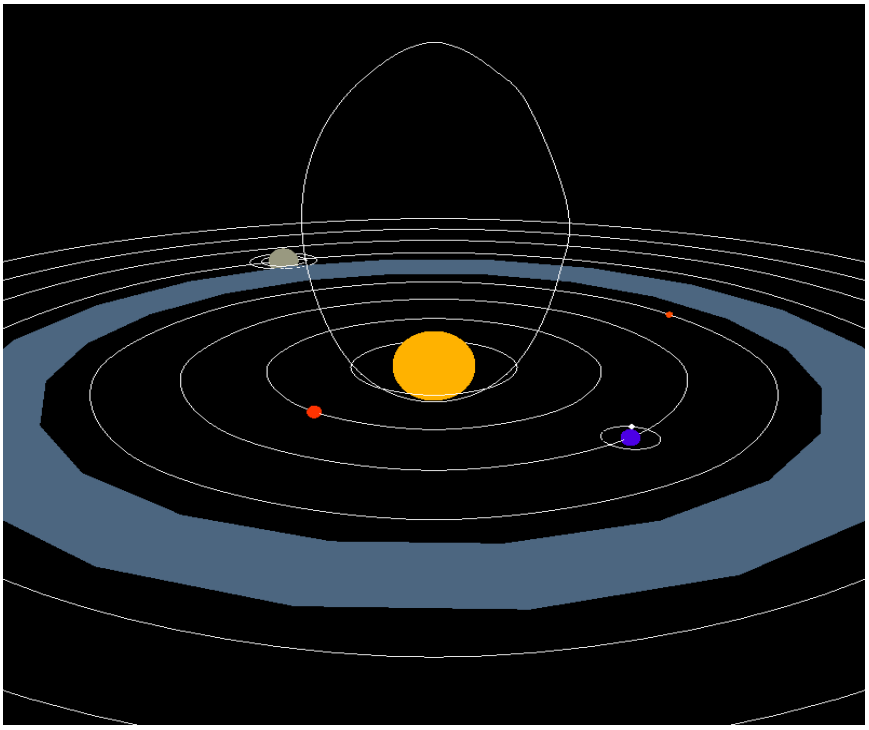

Figure 16: Perspetiva geral do Sistema Solar

7 Conclusão e Trabalho Futuro

Primeiramente, consideramos que o nosso desempenho nesta terceira fase do projeto se refletiu no facto de conseguirmos atingir todos os objetivos propostos. Conseguimos aprofundar mais os nossos conhecimentos ao nível de desenho de figuras com VBO's e implementação de curvas de Catmul-Rom. A compreensão das superfícies de Bezier foram um desafio, mas após isso tornou-se fácil aplicar as fórmulas para obtenção dos pontos.