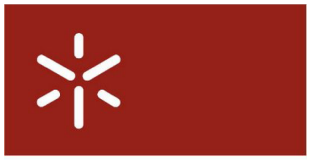
Universidade do Minho

Departamento de Informática

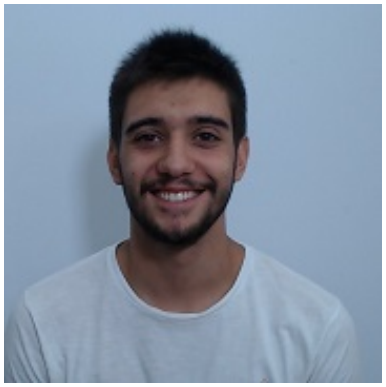


Universidade do Minho

Computação Gráfica

Fase 3 - Curvas, Superfícies Cúbicas e VBOs

Grupo 4



André Gonçalves Vieira
A90166

abril de 2022

Conteúdo

1 Introdução

Para esta terceira fase do projeto, foi necessário implementar a translação e rotação dinâmicas para permitir animações. Foram também criadas novas funções ao *generator* 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.

Nesta fase o sistema solar passou a ser desenhado recorrendo a VBO's.

2 Estrutura do Projeto

2.1 Generator

De um modo geral, o **generator.cpp** foi alterado de forma a ser capaz de converter um ficheiro com Bezier *patches* e convertê-lo num ficheiro com os pontos necessários para criar os triângulos para desenhar uma figura.

Para isso, faz-se o *parsing* do ficheiro "*.patch*", guardam-se os índices dos patches num *map* que tem um indicativo como chave e um *vector* de índices como valor e os pontos de controlo são armazenados num *vector*. Posteriormente, são calculados os pontos da superfície de Bezier, ao percorrer os *patches* guardados. Primeiro são obtidos os pontos de controlo e depois inicializam-se as matrizes de Bezier para estes pontos. De acordo com o nível de tesselação, as variáveis u e v são incrementadas e utilizam-se as matrizes calculadas para obter os pontos. Com a grelha finalmente calculada, divide-se cada quadrado em dois triângulos e armazenam-se os pontos num ficheiro "*.3d*".

Foi introduzida, também, uma nova primitiva geométrica, o *torus*.

2.2 Engine

No **engine.cpp**, foi alterarada a *drawGroup()* e a *parseXMLGroupElement()*, podendo agora realizar duas transformações geométricas novas. A partir desta fase, para desenhar as figuras acedemos à VBO correspondente. As transformações passaram a ser armazenadas num *vector* de modo a permitir realizar as por uma ordem desejada.

2.3 Classes

2.3.1 Shape

Esta classe foi completamente alterada de forma a poder guardar as informações para desenhar uma figura armazenada numa VBO, ou seja, o índice e o número de vértices. A definição da classe encontra-se em apêndice.

2.3.2 Group

A maior alteração desta classe foi a forma como as transformações são armazenadas, que passaram a ser armazenadas num *vector* de modo a permitir realizar as por uma ordem desejada. No ficheiro **group.h** (que se encontra em apêndice) é possível ver a nova estrutura das classes *Transformation*, *Rotate*, *Translate*, *Scale* e *Group*.

2.3.3 DynamicTranslate

A classe **DynamicTranslate** contém toda a informação que é necessária para efetuar uma translação dinâmica, ou seja, armazena o tempo total para percorrer a curva e os pontos de controlo de *Catmull-Rom*. Com os pontos de controloi são calculados os pontos da curva, para esta ser desenhada.

Para aplicar esta transformação é calculado o tempo passado desde o princípio da curva e, através deste, calculamos a posição da figura que pretendemos desenahar na curva. Existe também a possibilidade de orientar a figura no movimento ao longo da curva, através do calculo da derivada do ponto, caso seja pretendido. A definição da classe encontra-se em apêndice.

2.3.4 DynamicRotate

Nesta classe é armazenado o tempo necessário para realizar uma rotação completa e o eixo de rotação. De acordo com o tempo que ocorreu desde o início da rotação, a cada *frame* desenhada calcula-se o novo ângulo e efetua-se a rotação com o ângulo calculado e sobre o eixo pretendido.

A definição da classe encontra-se em apêndice.

3 Primitiva Geométrica

3.1 Torus

O ***Torus*** é uma figura geométrica que se aproxima a um *donut*. Para o definir são necessários um raio do torus *r1*, um raio da circunferência *r2*, o número de divisões da circunferência *divc* e o número de divisões angular *divh*. Calculamos os angulos *alpha* e *beta* que correspondem ao ângulo de cada divisão e calculamos os pontos das circunferências que constituem a figura.

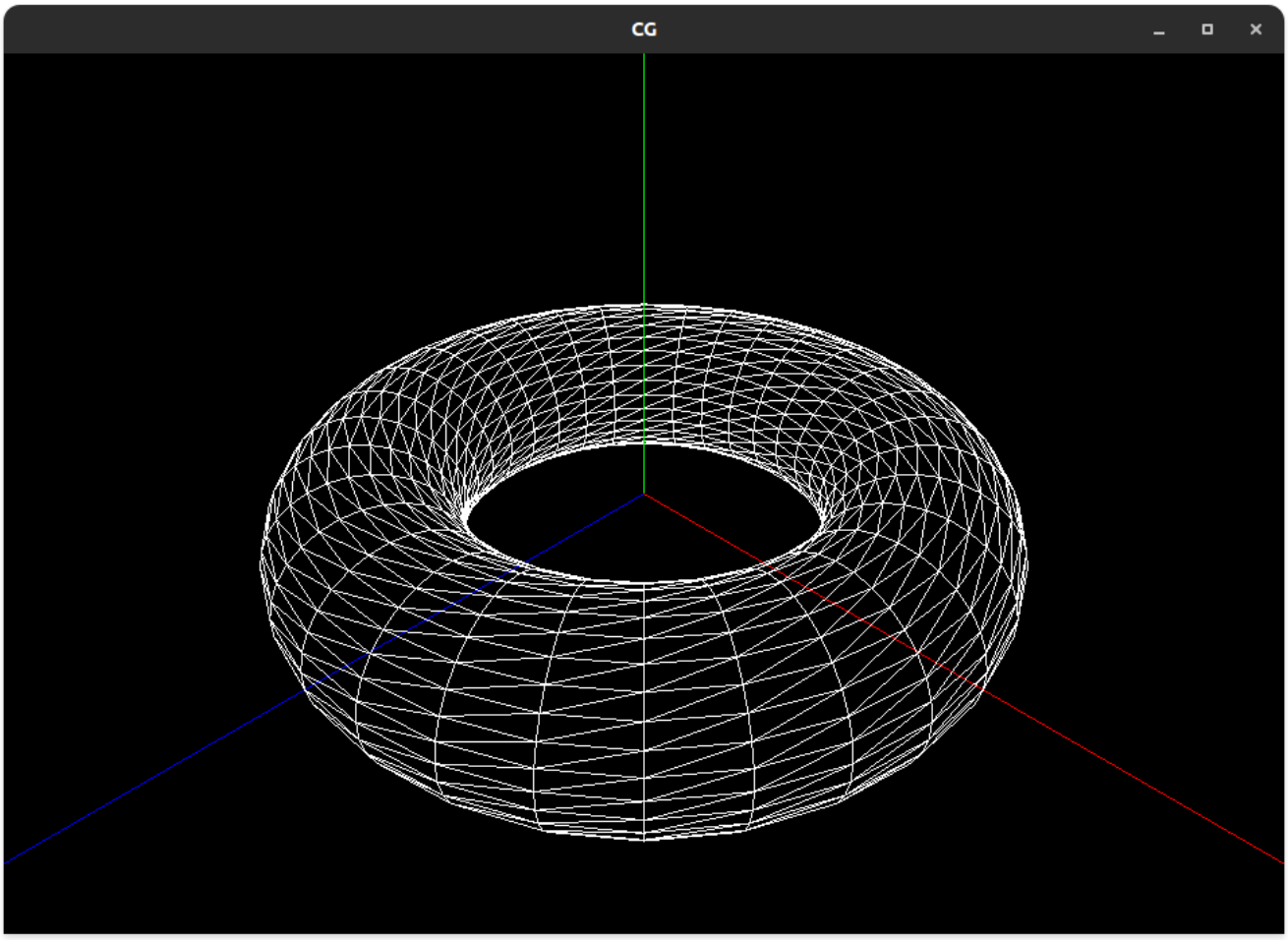


Figura 1: Exemplo de *Torus* gerado através do comando "*./gen torus 1 3 32 32 torus.3d*"

4 Resultados

No final desta fase obtemos um modelo dinâmico do Sistema Solar oposto ao modelo estático realizado na fase anterior. Na imagem abaixo é possível visualizar uma imagem com os planetas a orbitar em torno do sol. Também é possível ver um *Teapot* a fazer de cometa, também como a sua trajetória.

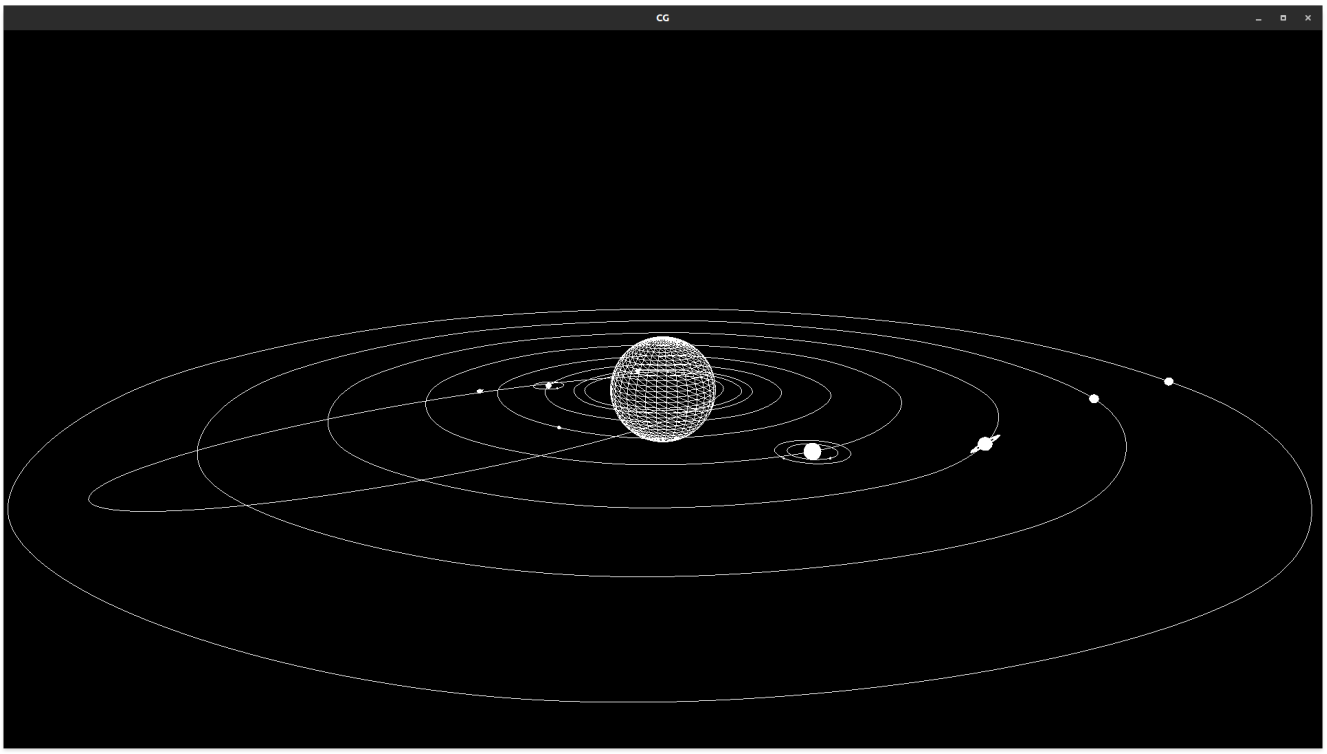


Figura 2: Perspetiva geral do Sistema Solar

5 Conclusão e Trabalho Futuro

Inicialmente, penso que foram atingidos os objetivos propostos para esta fase do trabalho, onde foram aprofundados os conhecimentos relativos à componente prática desta unidade curricular, nomeadamente ao nível do desenho de figuras através de VBO's e da implementação de curvas de *Catmul-Rom*, ambos abordados também nos guiões práticos da UC.

A Apêndice

A.1 Shape

```
class Shape {
    private:
        GLuint vbo_ind;
        GLsizei vertice_count;
    public:
        Shape() {
            this->vbo_ind = 0;
            this->vertice_count = 0;
        };
        Shape(GLuint vbo_ind, GLsizei vertice_count) {
            this->vbo_ind = vbo_ind;
            this->vertice_count = vertice_count;
        };

        GLuint getVBOInd() {return this->vbo_ind;};
        GLsizei getVerticeCount() {return this->vertice_count;};
};
```

A.2 DynamicTranslate

```
class DynamicTranslate : public Transformation {
    private:
        float total_time;  // time to run the whole curve
        float segment_time;  // time to run each segment
        float timebase;  // last time measured
        float elapsed_time;  // time elapsed since beginning of the curve
        bool align;
        vector<Ponto> points;
        vector<Ponto> render_points;

        void generateRenderPoints();
    public:
        void applyTransformation();
        void renderCatmullRomCurve();
        DynamicTranslate();
        DynamicTranslate(float total_time, bool align, vector<Ponto> points);
```

```
};
```

A.3 DynamicRotate

```
class DynamicRotate : public Transformation {
    private:
        float total_time;  // time to perform 360 degrees rotation
        float timebase;  // time at the start of the rotation
        float axisX, axisY, axisZ;  // axis of rotation
    public:
        void applyTransformation();
        DynamicRotate();
        DynamicRotate(float total_time, float axisX, float axisY, float axisZ);
};
```

A.4 group.h

```
using namespace std;

class Transformation {
public:
    virtual void smt(){};
};

class Translate : public Transformation {
private:
    float x, y, z;
public:
    float getX() {return this->x;};
    float getY() {return this->y;};
    float getZ() {return this->z;};
    Translate();
    Translate(float x, float y, float z) {
        this->x = x;
        this->y = y;
        this->z = z;
    };
};

```

```
23 class DynamicTranslate : public Transformation {
24 private:
25     float total_time;  // time to run the whole curve
26     float segment_time;  // time to run each segment
27     float timebase;  // last time measured
28     float elapsed_time;  // time elapsed since beginning of the curve
29     bool align;
30     vector<Ponto> points;
31     vector<Ponto> render_points;
32 
33     void generateRenderPoints();
34 public:
35     void applyTransformation();
36     void renderCatmullRomCurve();
37 
38     DynamicTranslate();
39     DynamicTranslate(float total_time, bool align, vector<Ponto> points);
40 };
41 
42 class Rotate : public Transformation {
43 private:
44     float angle, axisX, axisY, axisZ;
45 public:
46     float getAngle() {return this->angle;};
47     float getAxisX() {return this->axisX;};
48     float getAxisY() {return this->axisY;};
49     float getAxisZ() {return this->axisZ;};
50     Rotate();
51     Rotate(float angle, float axisX, float axisY, float axisZ) {
52         this->angle = angle;
53         this->axisX = axisX;
54         this->axisY = axisY;
55         this->axisZ = axisZ;
56     };
57 };
58 
59 class DynamicRotate : public Transformation {
60 private:
61     float total_time;  // time to perform 360 degrees rotation
62     float timebase;  // time at the start of the rotation
```

```
63      float axisX, axisY, axisZ;  // axis of rotation
64  public:
65      void applyTransformation();
66
67      DynamicRotate();
68      DynamicRotate(float total_time, float axisX, float axisY, float axisZ);
69  };
70
71  class Scale : public Transformation {
72  private:
73      float x, y, z;
74  public:
75      float getX() {return this->x;};
76      float getY() {return this->y;};
77      float getZ() {return this->z;};
78      Scale();
79      Scale(float x, float y, float z) {
80          this->x = x;
81          this->y = y;
82          this->z = z;
83      };
84  };
85
86
87  class Group {
88  private:
89      vector<Transformation*> transformations;
90      vector<Shape> pontos;
91      vector<Group> groups;
92
93  public:
94      Group();
95      void addTranslate(float x, float y, float z);
96      void addRotate(float angle, float axisX, float axisY, float axisZ);
97      void addScale(float x, float y, float z);
98      vector<Transformation*> getTransformations();
99      void addShape(Shape shape);
100     vector<Shape> getShape();
101     void addGroup(Group group);
102     vector<Group> getGroups();
```

```
103     void addDynamicTR(float time, bool align, vector<Ponto> points);
104     void addDynamicRT(float time, float axisX, float axisY, float axisZ);
105 };
```