Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Computação Gráfica

Ano Letivo de 2022/2023

Phase 3 – Curves, Cubic Surfaces and VBOs

Grupo 34

João Moreira a93326
Daniel Faria a93191
Paulo Filipe Cruz a80949

11 de junho de 2023

CG

Índice

Lista de Figuras

1 Introdução

O presente documento é alusivo à terceira fase do projeto prático desenvolvido com recurso à linguagem de programação C++, no âmbito da Unidade Curricular de Computação Gráfica que integra a Licenciatura em Engenharia Informática da Universidade do Minho. Este projeto encontra-se dividido em quatro fases de trabalho, cada uma com uma data de entrega específica. Esta divisão em fases, pretende fomentar uma simplificação e organização do trabalho, contribuindo para a sua melhor compreensão.

Pretende-se, assim, que o relatório sirva de suporte ao trabalho realizado para esta fase, mais propriamente, dando uma explicação e elucidando o conjunto de decisões tomadas ao longo da construção de todo o código fonte e descrevendo a estratégia utilizada para a concretização dos principais objetivos propostos, que surgem a seguir:

- Compreender a utilização do *OpenGL*, recorrendo à biblioteca GLUT, para a construção de modelos 3D;
- Aprofundar temas alusivos à produção destes modelos 3D, nomeadamente, em relação a transformações geométricas, curvas, superfícies, iluminação, texturas e modo de construção geométrico básico;
- Relacionar todo o conceito de construir modelos 3D com o auxílio da criação de ficheiros que guardam informação relevante nesse âmbito;
- Relacionar aspetos mais teóricos com a sua aplicação a nível mais prático.

Naturalmente, é indispensável que o conjunto de objetivos supracitados seja concretizado com sucesso e, para isso, o formato do relatório está organizado de acordo com uma descrição do problema inicial, seguindo-se o conjunto de aspetos relevantes em relação ao *Generator* e chegando aos aspetos primordiais sobre o *Engine*.

Nesta terceira fase do projeto o objetivo é desenvolver novas funcionalidades no *Generator*, nomeadamente criar um novo modelo baseado nas superfícies de Bezier. Ou seja, receber um nome de um ficheiro que possui vários pontos de controlo. O resultado final será um ficheiro com vários triângulos que representam a superfície.

Quanto ao *Engine*, as rotações e translações deixarão de poder ser apenas estáticas. Na rotação o atributo "angle"poderá ser trocado por "time", representando o tempo em segundos a fazer uma rotação completa. O campo Translação pode conter uma lista de pontos que representam uma curva Catmunll-Rom e além disso pode ter um campo "Align"para especificar

que o objeto necessita de estar alinhado com a curva.

Mais ainda, é necessário editar o ficheiro criado na fase anterior com o objetivo de desenhar um modelo do sistema solar dinâmico, incluindo um cometa com a trajetória definida usando uma curva Catmull-Rom.

2 Generator

Nesta fase, foi necessário efetuar algumas alterações de modo a efetuar a leitura dos *patches de Bezier* e a sua transformação em pontos que formam triângulos.

2.1 Leitura de ficheiros *patch*

O ficheiro que contém a informação para gerar os vértices do modelo *Bezier* pretendido, está dividido em 4 partes:

- 1ª linha -> Número de patches (number_patches)
- number_patches linhas seguintes -> *Patches*: cada linha contém 16 pontos de controlo que constitui um *patch*
- Linha seguinte -> Número de Pontos de Controlo (number_control_points)
- number_control_points linhas seguintes -> Pontos de Controlo

Tendo em conta esta organização, começamos por ler a informação relativa ao número de *patches*. De seguida, guardamos para cada *patch* os índices dos pontos de controlo num *array*. Por fim, guardamos as coordenadas dos pontos de controlo num vetor global, *controlPoints*, que serão posteriormente, acedidos para calcular os vértices do modelo que pretendemos representar.

2.2 Curvas de *Bezier*

Uma curva de *Bezier* necessita de 4 pontos, denominados de pontos de controlo e estão definidos em XYZ. Após obtermos as coordenadas dos pontos provenientes da leitura do ficheiro vamos calcular, para cada *pathc*, os pontos necessários. Para tal, vamos usar:

- divisions = 1.0 / tessellation
- u = us * divisions, us = [1..tessellation]
- v = vs * divisions, vs = [1..tessellation]

Posto isto, iremos percorrer os valores de **u** e **v** e vamos utilizar estas fórmulas para calcular os pontos dos triângulos que depois serão escritos no ficheiro de *output*. Assim, vamos calcular 4 pontos correspondentes a dois triângulos, por cada iteração do ciclo que percorre **u** e **v**.

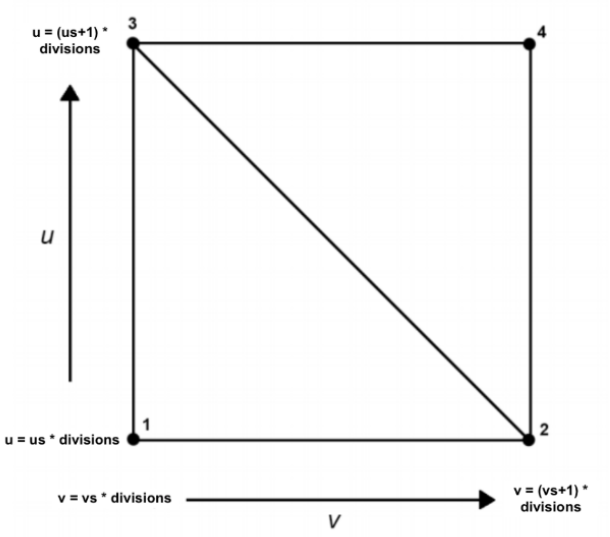

Figura 2.1: Cálculo de pontos com base em **u** e **v**.

Seguindo as fórmulas descritas, iremos calcular as coordenadas de cada um dos pontos, tendo em conta que para cada um deles apenas varia o **u** e o **v**. Começamos por escrever no ficheiro de *output* os pontos 1, 2 e 3, correspondentes ao primeiro triângulo, e seguidamente os pontos 3, 2 e 4, que correspondem ao segundo triãngulo. Para o cálculo de cada ponto, usamos os seguintes coeficientes:

- b0 = (1 - u) * (1 - u) * (1 - u)
- b1 = 3 * u * (1 - u) * (1 - u)
- b2 = 3 * u * u * (1 - u)
- b3 = u * u * u
- P(t) = (P0 * b0) + (P1 * b1) + (P2 * b2) + (P3 * b3)

3 Engine

Nesta fase do trabalho prático foi necessário atualizar o código com o principal intuito de permitir translações e rotações com tempo e de desenhar as primitivas com recurso a VBOs. Para isso, foi necessário fazer alterações tanto na leitura do ficheiro XML, como no desenho das primitivas.

A estrutura de dados relativa a transformações geométricas teve que ser alterada para guardar os novos valores das rotações e translações. Para além disso, foi preciso mudar a estrutura dos grupos para implementar VBOs. Assim, chegamos à seguinte estruturação:

```
// Representação de transformações geométricas com dados relativos para as mesmas.
struct Transform{
    float time;
    bool align;
    float angle;
    transformation type;
    std :: vector<Point> points;
};

/* Representação de um Group, do ficheiro xml, com as transformações geométricas
que são aplicadas,triângulos necessários para a construção da figura
e subgrupos que possa conter. */
struct Group{
    std :: vector<Transform> transformations;
    std :: vector<Group> subGroups;
    std :: vector<std :: pair<int,int>> vboIndexes;
};
```

3.1 Leitura do ficheiro XML

De modo a suportar a extensão das transformações de rotação e translação, foi necessário alternar a leitura do ficheiro XML. A função *parseGroups*, que trata da análise e povoamento

das estruturas de dados, possui agora a capacidade de ler e armazenar rotações e translações com tempo.

3.2 Curvas de Catmull-Rom

Um dos novos requisitos desta fase era a implementação de animações. Desta forma, os corpos celestes presentes no nosso sistema solar, podem ser alvo de translações que emulam as suas órbitas em volta do sol, no caso dos planetas, ou volta dos planetas, como é o caso das luas.

Para tal, foi necessária a implementação de uma das temáticas abordadas nas aulas: as curvas de Catmull-Rom. Estas vurvas são formadas por pelo menos 4 pontos (Po,P1,P2,P3), os quais manipulamos, em conjunto com uma iteração do valor t, através da equação abaixo apresentada:

$$p(t) = \begin{pmatrix} t^3 & t^2 & t & 1 \end{pmatrix} \times \begin{pmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1 & -2.5 & 2 & -0.5 \\ -0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$

Desta forma podemos, por pontos fornecidos no XML do sistema solar, calcular a órbita de cada corpo celeste. De forma análoga se procede ao cálculo da posição de cada um dos corpos em cada frame, considerando-se o tempo passado em relação ao tempo total que pretendemos para a rotação à volta do Sol/Planeta.

3.3 Desenho das Primitivas

Para desenhar as primitivas com a nova estrutura de dados, a função *drawGroup* foi atualizada.

A função continua a iterar por cada grupo presente na estrutura principal *world.groups* e para cada um faz *push* da matriz das transformações através da função do *glut glPushMatrix*. Posteriormente, percorremos o *array* das transformações e executamos por ordem que aparece no ficheiro e com base no tipo, seja estática ou dinâmica, através das funções *glTranslatef*, *glScalef* e *glRotatef* ou das funções *animateTranslate* e *animateRotate*. Antes de fazer *pop* voltamos a chamar recursivamente a função para aplicar às transformações dos subgrupos e só depois disso é que fazemos *glPopMatrix*. Deste modo, os subgrupos herdam as transformações do grupo pai.

```
void drawGroup(std :: vector<Group> groups){
    for(Group g : groups){
        glPushMatrix();

        for(int i = 0;i < g.transformations.size();i++){
            Transform t = g.transformations[i];
            if(t.type == translate){
                if (t.time!=-1) animateTranslate(t);
                else glTranslatef(t.points[0].x,t.points[0].y,t.points[0].z);
            }
            else if(t.type == rotate){
                if(t.time!=-1) animateRotate(t);
                else glRotatef(t.angle,t.points[0].x,t.points[0].y,t.points[0].z);
            }
            else if(t.type == scale) glScalef(t.points[0].x,t.points[0].y,t.points[0].z);
        }
        for (std :: pair<int,int> t : g.vboIndexes) {
            glBindBuffer(GL_ARRAY_BUFFER, vertices);
            glVertexPointer(3, GL_FLOAT, 0, 0);
            glDrawArrays(GL_TRIANGLES, t.first, t.second);
        }
        drawGroup(g.subGroups);
        glPopMatrix();
    }
}
```

---

4 Testes

Para mostrar em funcionamento o trabalho exposto ao longo deste relatório realizamos os seguintes testes:

4.1 Ficheiros *test*

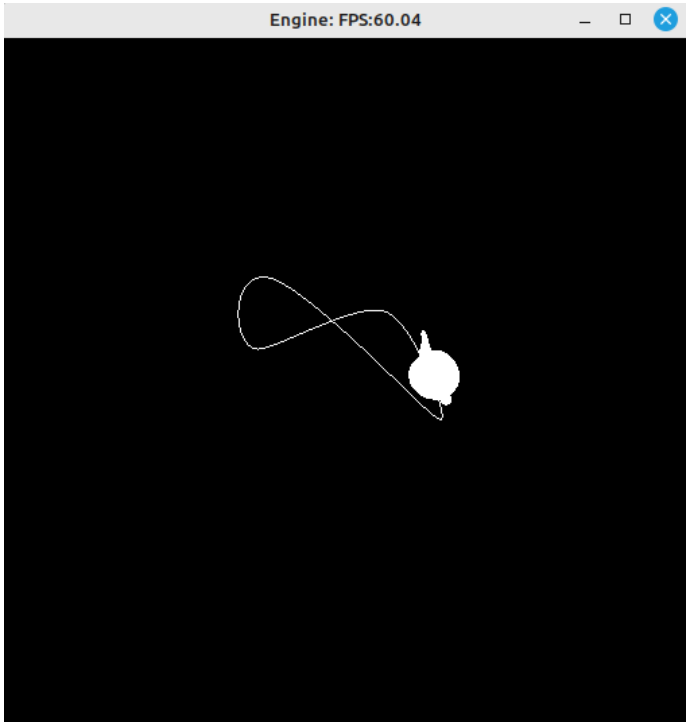

Figura 4.1: Ficheiro test_3_1.xml

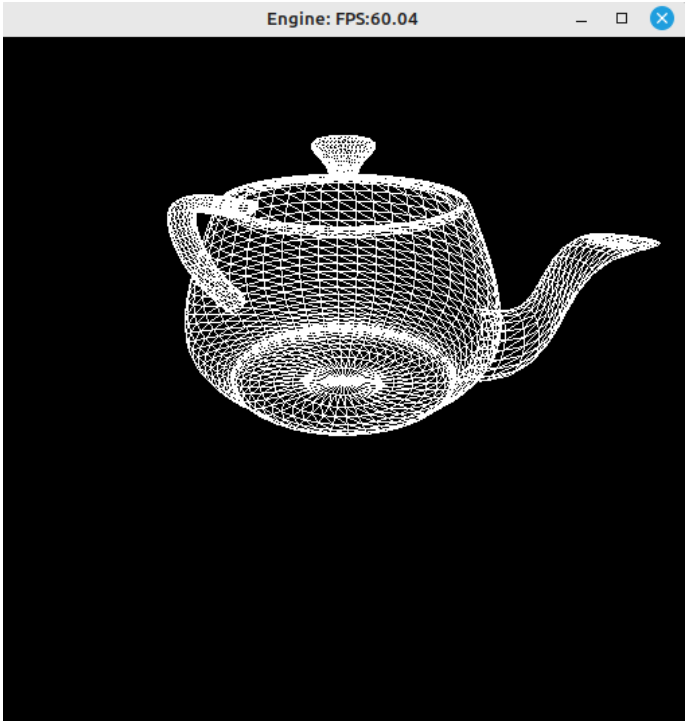

Figura 4.2: Ficheiro test_3_2.xml

5 Conclusão

Da realização desta terceira fase, o grupo considera que a mesma foi bem conseguida, já que se realizaram todas as funcionalidades requisitadas pelo próprio enunciado.

No espetro positivo, consideramos conveniente destacar o correto funcionamento do nosso programa. Além disso, as estruturas implementadas estão em concordância com a estrutura do XML permitindo uma visualização mais clara daquilo que é armazenado.

Por outro lado, também existiram algumas dificuldades, tais como a familiarização com as funções do *tinyXML2*, a implementação de funções recursivas e a implementação dos VBOs. Apesar disso, as dificuldades foram ultrapassadas.

Para concluir, consideramos que houve um balanço positivo do trabalho realizado dado que as dificuldades sentidas foram superadas a ser cumpridos todos os requisitos.