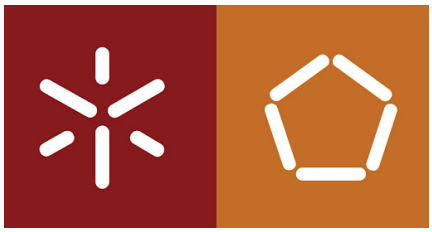

Universidade do Minho

Licenciatura em Engenharia Informática

Computação Gráfica - Fase 3
Grupo 38

Bernardo Emanuel Magalhães Saraiva - A93189

José João Cardoso Gonçalves - A93204

Mariana Rocha Marques - A93198 Rui Filipe Coelho Moreira - A93232

Ano Lectivo 2021/2022

Conteúdo

1 Introdução

1.1 Contextualização

O presente relatório tem como objetivo documentar e formalizar o procedimento efetuado para a implementação da terceira fase do Trabalho Prático de Computação Gráfica, constituinte do 2º semestre do 3º ano de Engenharia Informática.

A presente fase do Trabalho Prático dá seguimento ao que foi proposto nas fases anteriores, pelo que se trabalhou tomando como estrutura base o código anteriormente desenvolvido.

1.2 Objetivos da 3ª fase

Para a presente fase, foi proposto aos alunos uma série de objetivos, pelo que se apresenta em seguida:

- atualizar o *generator* de modo a ser possível criar modelos baseados em *patches* de *Bezier*;
- Atualizar o *engine* para desenhar os modelos com o uso de *VBOs*;
- Extender o *engine* de modo a que se torne compatível com as novas operações de rotação e translação, aceitando as tags *point* e os atributos *time* e *align*. Deste modo, aliado às curvas de Catmull-Rom, será permitido obter animações relativas às figuras desenhadas.

1.3 Alterações efectuadas na 3ª fase

Com vista a alcançar os objetivos propostos para a terceira fase, foram necessárias algumas alterações relativamente à fase anterior, centrando-se no ficheiro *tree*.

Para criar modelos baseados em *patches de Bezier* foi criado o ficheiro *bezier.cpp*, onde são calculados os pontos do modelo com base no *patch bezier*, tal como serão explicados na secção 2.1.

Uma vez que nesta fase foi introduzida a *tag point*, assim como os atributos *time* e *align*, foi necessário alterar as funções de *parsing* relativas a estes elementos, para que se possa efetuar a leitura e armazenar a informação relevante para o desenho das curvas, bem como a rotação. Para além do referido, foi acrescentada uma classe CatmullRom que guarda toda a informação necessária relativamente ao desenho das mesmas, como será descrito com maior detalhe na secção 2.3.

A nível do desenho das geometrias e armazenamento dos pontos das mesmas, foram também realizadas algumas alterações para suportar a utilização de VBOs percorridas em detalhe na secção 2.2.

2 Estrutura do código

2.1 Patches de Bezier

Um dos objetivos desta fase era o programa *Generator* passar a suportar novas pirmitivas com recurso a *Bezier Patches*, com diferentes níveis de tesselação. Para o seu desenvolvimento, é necessário passar os seguintes argumentos:

- ***InputFile:*** nome do ficheiro onde se encontram os *patches* necessários para o desenho da primitiva.
- ***Tesselation:*** representa a precisão com que a primitiva irá ser desenhada e define os incrementos a serem feitos a **u** e **v**, sendo o incremento igual a $\frac{1}{tesselation}$.
- ***OutputFile:*** nome do ficheiro onde serão guardados os vértices calculados.

Para esta nova funcionalidade, é necessário começar pela leitura dos dados presentes no *inputFile* e armazenamento dos mesmos em dois *arrays* distintos, um relativo aos *patches* e outro relativo aos pontos de controlo.

Após isto, iterar-se-á pelo número de *patches* e pela *tesselation* bidimensionalmente (dimensões u e v), calculando os pontos que geram a primitiva. Para fazer o cálculo de cada ponto, é utilizada a seguinte fórmula:

$$p(u,v) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_{00} & P_{10} & P_{20} & P_{30} \\ P_{01} & P_{11} & P_{21} & P_{31} \\ P_{02} & P_{12} & P_{22} & P_{32} \\ P_{03} & P_{13} & P_{23} & P_{33} \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

Obtém-se assim os seguintes *teapots*, verificando as diferenças gráficas com níveis de tesselação distintos:

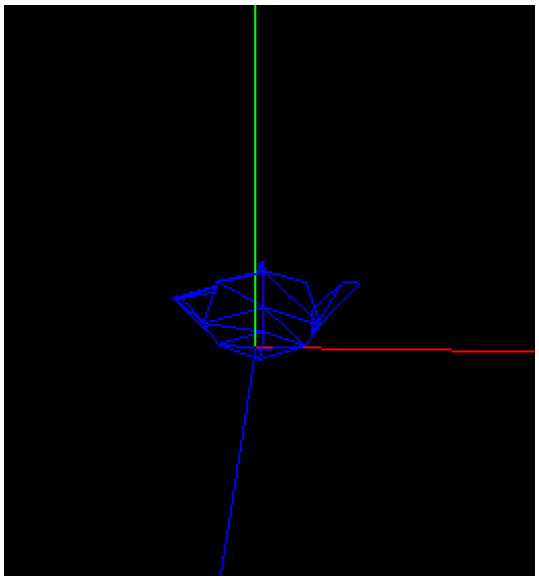

Figura 1: *Teapot* com nível de tesselação 1

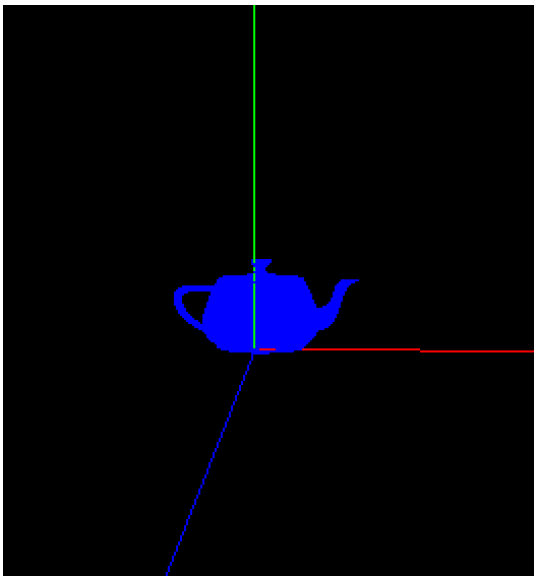

Figura 2: *Teapot* com nível de tesselação 16

2.2 VBOS

Um VBO designa-se como um array criado na memória da placa gráfica. Para a criação deste array é necessário criar um array com o conteúdo desejado em memória central, e de seguida transportá-lo para a memória gráfica. O principal objetivo deste método é desenhar a geometria pretendida com menos instruções, dado que deixa de ser necessária a chamada da função *glVertex* para cada vértice e explora, tirando partido do mesmo, o paralelismo disponiblizado pelo GPU uma vez que estamos a desenhar cada triângulo imeadiatamente ao receber cada conjunto de três vértices.

Para suportar esta nova funcionalidade, foi necessário fazer algumas alterações na estrutura de dados responsável por guardar a informação dos ficheiros '.3d'. Destas modificações resultou a nova classe *Figure*, onde é possível verificar a definição da instância *GLuint* correspondente ao *VBO* e o número de vértice existente.

```
1  class Figure
2  {
3      public:
4      GLuint vertices;
5      int verticesCount;
6
7
8      Figure(const char * name);
9
10 };
```

Para a criação do VBO, utilizar-se-á um vector para guardar os valores necessários enquanto se itera pelo ficheiro de input, e após acabar o parsing do XML, utilizar-se-á os comandos *glGenBuffers*, *glBindBuffer* e *glBufferData* para criar o VBO e armazenar os vértices

lidos.

Por fim, aquando do desenho das geometrias anteriormente armazenadas em VBOs é apenas necessário utilizar as funções *glVertexPointer* e *glDrawArrays*, logo após selecionar o VBO que se pretende representar com a função *glBindBuffer*.

2.3 Catmull Rom Curves

O desenvolvimento das curvas de Catmull Rom revela-se uma componente essencial do trabalho prático, uma vez que permite associar um movimento às figuras, quer seja de rotação ou de translação. Estas curvas são geradas através de pontos associados a matrizes, sendo posteriormente unidos e formando aproximações de curvas através de polinómios provenientes de matrizes.

Tal como na fase anterior, optou-se por recorrer a um *boolean* na classe Transform que indica se está presente o *time*, assim como uma instância da classe catmullRom, que contém a seguinte informação:

- **vector<Point> points** - vetor com os pontos por onde a curva irá passar
- **float time** - valor associado ao campo time
- **bool align** - booleano que indica se é para usar align ou não

Em seguida, passar-se-á a uma breve explicação relativamente ao procedimento para a geração destas curvas.

Tendo um conjunto de 4 pontos, através das curvas de Catmull Rom, obtém-se uma curva que expressa o segmento intermédio, como se pode ver pelo seguinte esquema.

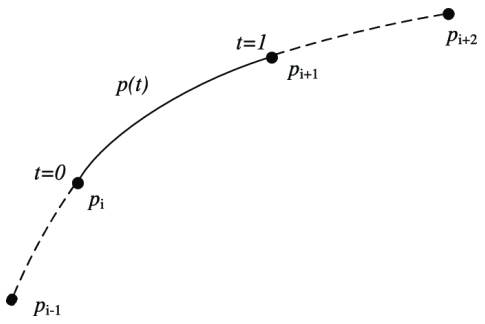

Figura 3: Imagem exemplificativa da curva relativa ao cálculo através de Catmull Rom

A partir do conjunto de pontos, é possivel obter a seguinte matriz, que irá ser extemamente útil para o cálculo de curvas. Para tal, recorre-se à expressão $P(t) = at^3 + bt^2 + ct + d$ para calcular a expressão correspondente a cada ponto.

Assim, tem-se a seguinte igualdade:

$$\begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 6 & 4 & 2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \tag{1}$$

A matriz anterior poderá ser descrita como $P = C * A$, sendo que se poderá representar de forma equivalente por $A = C^{-1} * P$:

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

Assim, obtém-se as seguintes equações que permitem calcular curvas:

$$C_0(t) = -0.5t^3 + t^2 - 0.5t$$

$$C_1(t) = 1.5t^3 - 2.5t^2 + t$$

$$C_2(t) = -1.5t^3 + 2t^2 + 0.5t$$

$$C_3(t) = 0.5t^3 - 0.5t^2$$

2.4 Curvas

De modo a implementar as curvas de Catmull Rom, começou-se por alterar a função que efetua o parsing dos ficheiros XML, para que suporte as novas terminologias a ser aplicadas no *translate* e no *rotate*, sendo elas point, align e time.

Inicialmente, começa-se por obter os pontos através das tags *point* provenientes do ficheiro XML, sendo armazenados nos índices corretos para serem utilizados no cálculo da curva, efetuando-se este processo na função *getGlobalCatmullRomPoint*. Esta função recebe um valor inserido no intervalo entre 0 e 1, que indica a percentagem do comprimento da curva onde se encontra o corpo, obtendo o t e os 4 pontos para gerar a curva.

```
void getGlobalCatmullRomPoint(float gt, Point *pos, Point *deriv, CatmullRom catR) {
    int POINT_COUNT = catR.points.size();
    float t = gt * POINT_COUNT; // this is the real global t
    int index = floor(t);  // which segment
    t = t - index; // where within  the segment

    // indices store the points
    int indices[4];
    indices[0] = (index + POINT_COUNT-1)%POINT_COUNT;
    indices[1] = (indices[0]+1)%POINT_COUNT;
    indices[2] = (indices[1]+1)%POINT_COUNT;
    indices[3] = (indices[2]+1)%POINT_COUNT;

    getCatmullRomPoint(t, catR.points[indices[0]], catR.points[indices[1]], catR.points[indices[2]], catR.points[indices[3]], pos, deriv);
}
```

Figura 4: Função getGlobalCatmullRomPoint

Em seguida, tal como se pode verificar na figura acima, passa-se à função *getCatmullRom-Point* e procede-se ao cálculo dos polinómios que auxiliam no processo de gerar as curvas, através da matriz de CatmullRom, como na Matriz C^{-1}, referente ao demonstrado em 2.3. Após isso, calcula-se as coordenadas da posição (através das fórmulas descritas anteriormente) e da derivada e procede-se ao desenho.

Na função drawModels (que é a que chama todas as referidas anteriormente), verifica-se se há a presença de align/time ou a presença de *point's* e, caso haja, recorre-se à função correspondente para efetuar a transformação desejada. É de referir que foi tido especial cuidado pelo facto de a rotação apenas ser efetuada após a translação.

2.5 Órbitas

Para desenhar as órbitas dos planetas, dado que as curvas Catmull Rom necessitam de no mínimo 4 pontos escreveu-se um pequeno script em *Python* que dado um raio calcula 10 pontos equidistantes de uma circunferência, através de coordenadas polares.

```
def shpereEquation():
    theta = 0

    while(theta < 2 * math.pi):

        x = radius * math.cos(theta)
        y = radius * math.sin(theta)
        theta = theta + math.pi/5;

        print('<point x="%f" y="0" z="%f"/>' % (x,y))
```

Desta forma, é possível colocar os planetas numa órbita circular em torno do sol, inserindo os pontos obtidos no campo translate do corpo que se pretende colocar em órbita, no XML.

No que toca à órbita dos asteroides, a função de geração dos pontos é um pouco diferente, uma vez que tem por base a equação da elipse.

2.6 Rotação

Caso haja a presença do elemento *time* num *rotate*, o *renderRotate* recorre ao tempo atual, e calcula o ângulo a partir do mesmo. Deste modo, o angulo é calculado através da fórmula

float angle = (((time / 1000) * 360) / rotate.value) - rot_angle

sendo uma regra de 3 simples que calcula a fração de ângulo que necessita ser percorrido, subtraindo-se o valor que já foi percorrido com a rotação. Deste modo, aplica-se a rotação da mesma através da função *glRotatef* com a variável *angle* referida anteriormente.

3 Sistema Solar

Após o desenvolvimento das curvas designadas, juntamente com os Patches de Bezier, atualizou-se o sistema solar gerado na fase anterior de modo a implementar uma animação no que foi previamente desenvolvido.

Deste modo, para além do movimento independente de cada planeta, adicionou-se uma lua na Terra, assim como um cometa que circula segundo uma trajetória elíptica.

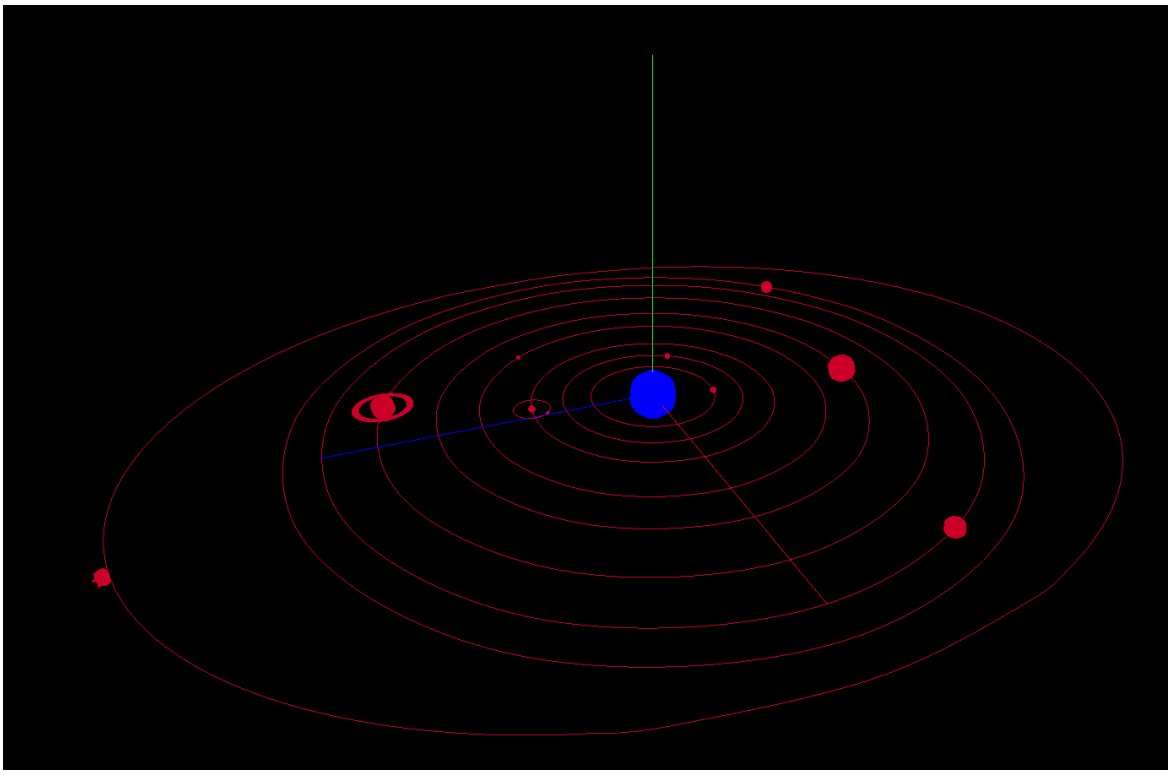

Figura 5: Resultado final do Sistema Solar para a fase 3

4 Conclusão

A elaboração desta fase do trabalho prático permitiu aplicar conhecimentos teóricos e práticos no que diz respeito às curvas de Bezier e de Catmull-Rom, de forma a melhorar o sistema solar desenvolvido nas últimas fases.

Para além de adições gráficas, foi também possível desenvolver competências relativas ao uso de VBOs de forma a tornar o desenho do sistema solar mais eficiente.

Em suma, esta fase contribuiu para aumentar não só o realismo do sistema solar ao possibilitar a inserção de órbitas, como também o desempenho do programa.