Relatório Computação Grafica - Fase 3

Marco Sousa[1,2[62608]], José Malheiro[1,2[93271]], and Miguel Fernandes[1,2[94269]]

[1] Universidade do Minho
[2] Licenciatura em Engenharia Informática, Braga, Portugal

Resumo No seguimento lógico da construção de um mundo gráfico virtual estático, vem a evolução para um modelo dinâmico, sendo todos os elementos envolventes definidos com a sua própria animação. Para tal foi necessário extender as transformações desenvolvidas na fase anterior, a rotação e a translação. Ambas devem poder ser realizadas com base num determinado tempo, sendo que a animação recomeça quando alcança o fim. No caso da translação, deve ser permitido construir curvas utilizando as estratégias de *Catmull-Rom*, *Bezier* e *Hermite*, a partir de um conjunto de pontos de controlo. Neste sentido, a transformação deve permitir mover um objeto de acordo com uma curva, ao longo de um determinado tempo. Assim, alterou-se o modelo do sistema solar construído na fase anterior para incluir animações para todos os elementos, sendo feita a adição de um cometa que segue a trajetória de uma curva definida.
Para o desenho do cometa foram utilizadas *Bezier patches* para construir as suas superfícies, sendo passados os pontos de controlo para o desenho.

Keywords: OpenGL · GLUT · Figuras Geométricas · 3D · C++ · Tesselation · Bezier Curves · Catmull-Rom Curves · Bezier Patches

1 Introdução

1.1 Contextualização

No seguimento da fase II do projeto da disciplina de Computação Gráfica da Licenciatura em Engenharia Informática da Universidade do Minho, foi proposta a aplicação de animações no esquema do sistema solar previamente definido, com a possibilidade de desenhar e movimentar objetos ao longo de curvas. Adicionalmente, deveria haver a inclusão de um novo elemento no modelo, um cometa, construído com base em *bezier patches*, para ser animado segundo uma trajetória definida.

1.2 Breve Descrição do Enunciado Proposto

O cerne da fase III do projeto implica alterações ao nível do *generator* e do *engine* previamente definidos. Deste modo, pretende-se:

generator Adição de um novo tipo de modelo, com base em *bezier patches*.
engine Alteração das transformações para permitir a animação do esquema, através da integração do **tempo** a serem realizadas e de **curvas**.

No que toca ao *generator*, o novo modelo deve criar um conjunto de pontos para desenhar os *patches*, ou superfícies, dado um ficheiro *.PATCH* e o nível de tesselação desejado. O ficheiro *.PATCH*, neste caso fornecido pela equipa docente, segue:

Número de patches Número de superfícies do objeto a desenhar.
Patches Os 16 pontos de controlo para cada *patch*. Haverá tantas linhas como o número de patches em cima definido. Os pontos de cada superfície encontram-se na forma de índices relativamente à posição dos pontos entre sí no ficheiro.
Pontos Os pontos usados na construção dos *patches*. São da forma - xyz.

O nível de tesselação é utilizado para construir o objeto com um grau de definição concreto e a sua utilização varia o número de vértices finais a serem desenhados. Novamente, as superfícies de um objeto com base neste modelo vão ser desenhadas a partir de triângulos, traduzindo-se no conceito de *triângulos de bezier*.

Relativamente à *engine*, o trabalho nesta implica evoluir as transformações de **translação** e **rotação** previamente definidas:

Rotação Permitir que uma rotação receba o tempo que demora a fazer um rotação de 360º.
Translação Permitir serem passados os pontos de controlo para constituir a curva e o tempo que demora a percorre-la. O objeto a ser movido terá de seguir esta trajetória e pode ser dada a opção de ele estar alinhado com a curva.

A partir destas alterações apresentar todo o sistema solar como um modelo animado, os planetas, luas e Sol e incluir um novo elemento no modelo, um *cometa*, que tem a sua própria trajetória.

2 Trabalho Realizado

Funcionalidades Implementadas:

1. *Generator*
   (a) Gerar triângulos para a construção de superfícies 3D a partir de pontos de controlo. (2.1)
2. *Engine*
   (a) Translação com base no tempo e em pontos de controlo,
      i. através de curvas de *Catmull-Rom/Hermite/Bezier*. (2.2)
      ii. através de uma matriz definida pelo utilizador.
   (b) Rotação em torno de eixo especificado, com base no tempo. (2.2)
   (c) *VBOs*
      i. Renderização do mundo com recurso a *VBOs*. (2.3)
3. Sistema Solar
   (a) Transição de um modelo estático para dinâmico.(2.4)

2.1 *Bezier Patches*

A construção das superfícies 3D foi dificultada por uma confusão das curvas a desenhar. Inicialmente, os *patches* estavam a ser construídos a partir da estratégia de *Catmull-Rom* e dado a ser com base num *line loop*, não estavam a ser desenhados triângulos, logo o bule final estava longe da representação desejada. Para além disso, não estavam a ser desenhadas as divisões corretas com base no **nível de tesselação**, como em *bezier patches*, só estavam a ser desenhados mais segmentos na curva.

Neste sentido, após uma análise do material fornecido pela equipa docente foram construídas 3 classes para permitir definir os triângulos das superfícies:

- `class Matrix`
- `class PointMatrix`
- `class BezierTriangles`

Classe Matrix Envés de representar as matrizes como apontadores para valores, como foi o caso das aulas práticas, procurou-se definir um módulo que facilmente representaria uma matriz.

CGDraw Source Code 1.1: Matrix

```
14  class Matrix
15  {
16  protected:
17          float* _mat;
18          int _m;
19          int _n;
```

Deste modo, de forma encapsulada encontram-se na classe todas as operações relativamente a uma matriz e entre matrizes. Não obstante a possibilidade de **transpor** e **clonar** uma matriz, foi definida a multiplicação entre a matriz local e outra; como a matriz local pode encontrar-se antes ou depois, recorreu-se ao **polimorfismo** da linguagem para construir os dois métodos considerando as possibilidades. Para a classe foram criados diversos construtores, para inicializar a matriz, sendo um deles respetivo à construção das matrizes de transformação para os tipos de curva a serem usados:

CGDraw Source Code 1.2: Matrix

```
7   enum DefMat
8   {
9           CATMULL,
10          BEZIER,
11          HERMITE
12  };
```

. . . como são sempre estáticos facilita o grupo ao usá-los nos cálculos dos pontos de uma curvas.

Class PointMatrix Apesar da `class Matrix` tratar corretamente de matrizes com valores, foram encontradas dificuldades no cálculo dos pontos de uma curva, nomeadamente na representação de **matrizes de pontos**. Como em fases anteriores foi criada a `class Point`, pretendia-se agregar numa classe a multiplicação entre uma matriz de valores e uma matriz de pontos, considerando a multiplicação à esquerda e à direita.

Surgiu a `class PointMatrix`:

CGDraw Source Code 1.3: Matrix

```
6  class PointMatrix
7  {
8          Point* _mat;
9          int _m;
10         int _n;
```

s . . . contêm ainda a possibilidade de transpor e clonar a matriz.

Class BezierTriangles Com classes criadas no âmbito de realizar operações entre matrizes, procedeu-se à criação de uma última `class BezierTriangles`. A partir dos pontos de controlo de um *patch* calcular o ponto da curva, de acordo com os vetores **u** e **v** resultante dos nível de tesselação. Neste sentido,

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

Figura 1: Cálculo de um ponto da superfício, definida com P pontos de controlo.

Assim, segue-se:

CGDraw Source Code 1.4: BezierTriangles

```
5  class BezierTriangles
6  {
7  private:
```

```
8         PointMatrix _points;        // 4 x 4
9         PointMatrix _const;               // 4 x 4 :: M . P . M^T
10        PointMatrix _u_mpm;               // 1 x 4 :: u . const
11        PointMatrix _mpm_v;               // 1 x 1 :: .. v
12        Matrix _m;                              // 4 x 4
13        Matrix _m_t;                     // 4 x 4
14        Matrix _u;                              // 1 x 4
15        Matrix _v;                              // 4 x 1
```

Para evitar o cálculo constante da matriz de transformação de *Bezier*, M, com a matriz dos pontos de controlo da superfície, P, e, consequentemente, o cálculo da resultante, MxP, pela transpost, M^T, utilizou-se variáveis locais, o que benificia no consumo de memória.

create_bezier No âmbito de calcular todas as superfícies criou-se dentro do módulo **shapes**, um método capaz de contruir um objeto, recebendo um vetor com os índices dos pontos de controlo de cada *patch*, *pacthes*, um vetor com todos os pontos, *points* e o nível de tesselação, *level*.

Deste modo, segue-se:

CGDraw Source Code 1.5: create_bezier

```
44  t_points create_bezier(vector<vector<int>> patches, vector<Point> points,
    ↪  int level);
```

Adicional A extração de informação encontra-se dentro do módulo **writer**, na função *main* do *generator*, sendo tudo feito recorrendo ao uso de **expressões regulares** e da biblioteca `<regex>`.

2.2 Animação com Transformações Geométricas

Translação

Rotação

2.3 VBOs

No seguimento de terem sido aplicados *VBOs* sem índices na fase II, a implementação dos modelos com o uso de índices era a próxima evolução lógica. Contudo, devido a impossibilidades de tempo não se encontram presentes no âmbito do presente relatório. Apresenta-se, apenas, o desenho dos triângulos a partir de *VBOs* sem índices.

2.4 Modelo do Sistema Solar

3 Conclusões