Computação Gráfica

Trabalho Prático - Fase 1

Grupo nr.	38
a83899	André Morais
a84577	José Pedro Silva
a85954	Luís Ribeiro
a84783	Pedro Rodrigues

Mestrado Integrado em Engenharia Informática
Universidade do Minho

Conteúdo

1 Introdução

O objetivo desta terceira fase do trabalho é continuar o trabalho realizado na fase anterior, através da aplicação de animações relativas a uma translação ou rotação ao nosso Sistema Solar.

Uma rotação completa passa a ser efetuada num determinado período de tempo, contrariamente à fase anterior. Já uma translação é definida à custa de uma curva de Catmull-Rom, também efetuada num determinado período de tempo. Para conseguirmos adicionar estas novas funcionalidades, usamos o algoritmo de Bézier. Para além disto, vamos também recorrer ao VBOs para nos ajudar no desenho de todos os modelos da fase anterior.

No presente relatório descrevem-se com detalhe cada uma das componentes da terceira fase, desde a descrição das novas funcionalidades dos dois programas, Generator e Engine, até aos algoritmos efetuados no cálculo dos vértices e índices de todas as primitivas, bem como de uma rotação e uma translação.

2 Curvas, Superfícies Cúbicas e VBO

2.1 Superfícies de Bezier

Foi adicionada uma nova primitiva ao módulo responsável pela geração de modelos. Esta primitiva permite a geração de superfícies de Bezier. Partindo de um ficheiro contendo pontos de controlo é possível construir triângulos que representem a superfície.

Na implementação desta primitiva foram utilizadas as fórmulas presentes no formulário disponibilizado pelos docentes, mais concretamente:

$$M = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

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

Como M é simétrica, então $M^T = M$, logo:

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

Sendo assim, é necessário realizar as multiplicações de matrizes, Primeiramente é realizada a operação $U \times M$

```
float UM[4];
for (int i = 0; i < 4; i++) {
    UM[i] = 0;
    for (int j = 0; j < 4; j++) {
        UM[i] += U[j] * M[i][j];
    }
}
```

A seguinte operação será $UM \times P$, onde a matriz P representa os pontos que fazem parte daquela superfície.

```
float UMP[3][4];
for (int i = 0; i < 4; i++) {
    UMP[0][i] = UM[0] * p[4 * i].getX()
              + UM[1] * p[4 * i + 1].getX()
              + UM[2] * p[4 * i + 2].getX()
              + UM[3] * p[4 * i + 3].getX();
    UMP[1][i] = UM[0] * p[4 * i].getY()
              + UM[1] * p[4 * i + 1].getY()
              + UM[2] * p[4 * i + 2].getY()
              + UM[3] * p[4 * i + 3].getY();
    UMP[2][i] = UM[0] * p[4 * i].getZ()
              + UM[1] * p[4 * i + 1].getZ()
              + UM[2] * p[4 * i + 2].getZ()
              + UM[3] * p[4 * i + 3].getZ();
}
```

Com a matriz UMP construída, o próximo passo é computar a multiplicação $UMP \times M$

```
float UMPM[3][4];
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) {
        UMPM[i][j] = UMP[i][0] * M[0][j]
                   + UMP[i][1] * M[1][j]
                   + UMP[i][2] * M[2][j]
                   + UMP[i][3] * M[3][j];
    }
}
```

Para obter o resultado final, apenas falta realizar a operação $UMPM \times V$

```
float UMPMV[3];
for (int i = 0; i < 3; i++) {
        UMPMV[i] = UMPM[i][0] * V[0] + UMPM[i][1] * V[1]
                 + UMPM[i][2] * V[2] + UMPM[i][3] * V[3];
}
```

Como podemos ver pela fórmula apresentada acima, a matriz $UMPMV$ é o resultado final, então, o ponto gerado, conforme o u e v recebido, será:

```
Point(UMPMV[0], UMPMV[1], UMPMV[2]);
```

Implementada a função que cálcula o ponto da superfície de Bezier, é necessário, apenas, iterar sobre o patch recebido como input e escrever os pontos gerados no ficheiro pretendido.

Este método é trivial, a maior dificuldade está em construir os triângulos a partir dos pontos, e para isso, a cada iteração, é gerado um quadrado representado por dois triângulos, tendo em conta a divisão de cada superfície.

```
float tess = (float)1 / (float)lvl;

for (int n=0;n<patches;n++){
    vector<Point> points_formula = vector<Point>(16);

    for (int k = 0; k < 16; k++) {
        points_formula[k] = points[indexes[n][k]];
    }

    for (int i = 0; i < lvl + 1; i++) {
        for (int j = 0; j < lvl + 1; j++) {
            float u = i * tess;
            float v = j * tess;

            Point p = bezier_formula(points_formula, u, v);
            file << p.getX() << "," << p.getY()
                << "," << p.getZ() << endl;
            p = bezier_formula(points_formula, u+tess, v);
            file << p.getX() << "," << p.getY()
                << "," << p.getZ() << endl;
            p = bezier_formula(points_formula, u, v+tess);
            file << p.getX() << "," << p.getY()
                << "," << p.getZ() << endl;


            file << p.getX() << "," << p.getY()
                << "," << p.getZ() << endl;
            p = bezier_formula(points_formula, u + tess, v);
            file << p.getX() << "," << p.getY()
                << "," << p.getZ() << endl;
            p = bezier_formula(points_formula, u+tess, v+tess);
            file << p.getX() << "," << p.getY()
                << "," << p.getZ() << endl;
        }
    }
}
```

3 Gerador

No módulo do gerador, foi necessário fazer várias alterações, desde a adição da possibilidade de criar uma curva de translação (*catmull-rom curve*), à possibilidade de definir um periodo de rotação até ao desenho da cena com recurso aos VBO's.

3.1 Translação com recurso ao Catmull-Rom Curve

É necessário alterar a função que dá parser do ficheiro xml para que seja possível ler uma translação com informações do tempo e dos pontos para gerar a Catmull-Rom Curve.

Com toda a informação obtida, basta-nos gerar a catmull-rom curve e calcular o ponto objeto naquele instante de tempo. Para isso, foram utilizadas as funções criadas no guião 8, mais concretamente:

- **getCatmullRomPoint** - gera o ponto e a derivada num determinado segmento da catmull-rom;
- **getGlobalCatmullRomPoint** - gera o ponto e a derivada no instante de tempo enviado como argumento, utiliza a função anterior;
- **renderCatmullRomCurve** - desenha a curva completa.

Para gerar o ponto e a derivada consoante o instante de tempo e o segmento da catmull-rom foram utilizadas as fórmulas presentes no formulário disponibilizado pelos docentes

$$M = \begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1 & -2.5 & 2 & -0.5 \\ -0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot M \cdot \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

$$P'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \cdot M \cdot \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

Assim, basta-nos fazer as multiplicações das matrizes.
$A = M \times P$

```
float a[4][3];
for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 3; j++) {
        a[i][j] = m[i][0] * p0[j] + m[i][1] * p1[j]
                + m[i][2] * p2[j] + m[i][3] * p3[j];
    }
}
```

Com a matriz A calculada, basta uma multiplicação para calcular o ponto e a derivada.

$Pos = T \times A$

```
float mt[4] = { pow(t,3),pow(t,2),t,1 };
for (int i = 0; i < 3; i++) {
    pos[i] = 0;
    for (int j = 0; j < 4; j++) {
        pos[i] += mt[j] * a[j][i];
    }
}
```

$Deriv = T' \times A$

```
float d[4] = { 3 * pow(t,2),2 * t,1,0 };
for (int i = 0; i < 3; i++) {
    deriv[i] = 0;
    for (int j = 0; j < 4; j++) {
        deriv[i] += d[j] * a[j][i];
    }
}
```

Implementada uma função que nos devolva o ponto e a derivada nesse ponto, apenas precisamos de saber o tempo passado até esse momento e realizar uma translação para esse mesmo ponto.

```
float time = (glutGet(GLUT_ELAPSED_TIME)/1000) / t.getTime();
t.renderCatmullRomCurve(c.getR(), c.getG(), c.getB());
t.getGlobalCatmullRomPoint(time, pos, deriv);
glTranslatef(pos[0], pos[1], pos[2]);
```

No entanto, surge o problema do objeto não estar alinhado com a curva, então foi utilizada a mesma técnica que no guião. É necessário contruir uma matriz rotação e multiplicar pela matriz do sistema. Para gerar essa matriz é necessário utilizar as fórmulas fornecidas pelos docentes.

$$X_i = P'(t)$$

$$Y_i = X_i \times Z_i$$

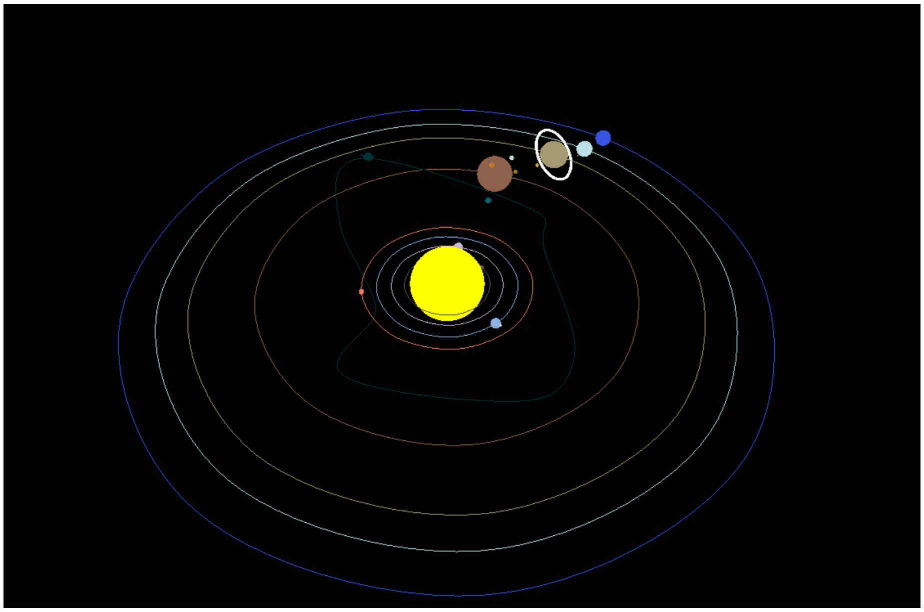
$$Z_i = X_i \times Y_{i-1}$$

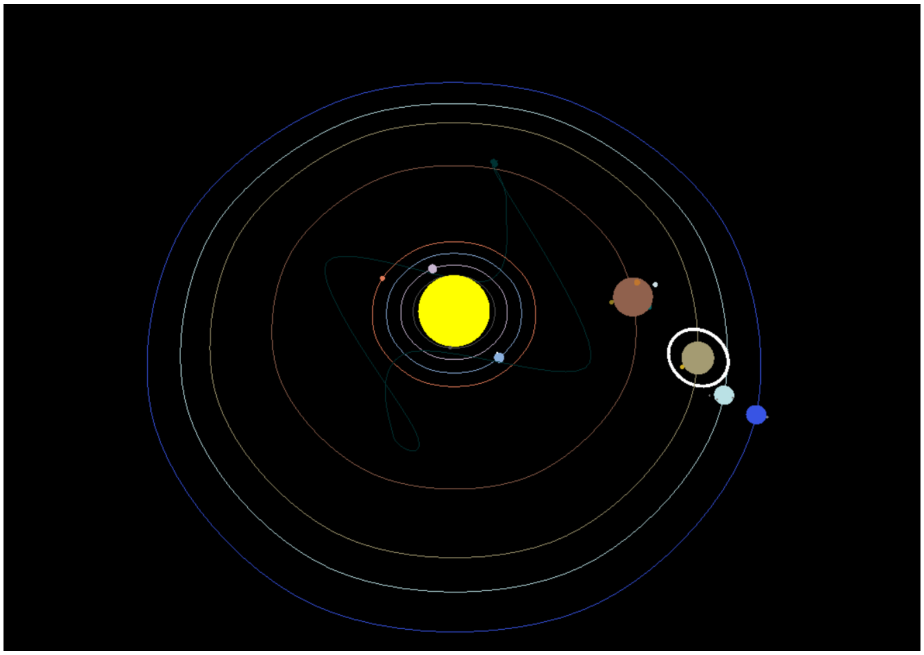
3.2 Rotação com recurso ao período

Mais uma vez, foi necessário alterar o parser de xml para ser possível receber dois tipos de rotações diferentes. Depois disto, apenas é necessário indentificar os direntes tipos e, no caso de receber o período de rotação, basta calcular o ângulo correto naquele instante de tempo e computar a rotação para lá:

```
angle = (glutGet(GLUT_ELAPSED_TIME)/1000) * 360 / r.getTime();
glRotatef(angle, r.getX(), r.getY(), r.getZ());
```

4 Sistema Solar Dinâmico





5 Conclusão

Esta terceira fase permitiu-nos explorar os conceitos de animações em translações e rotações. As curvas de Catmull-Rom representaram um método simples para o cálculo de todos os pontos que traduzem uma animação de uma translação.

As superfícies cúbicas de Bezier permitiram a construção de primitivas com bastante complexidade através de uma sequência de instruções fáceis de concretizar. Para além disto, o desenho dos modelos com recurso a VBOs facilitou-nos e permitiu uma melhoria a nível de eficiência.

Para concluir, a realização desta fase do projeto revelou-se bastante mais trabalhosa e demorada no entanto, o resultado apresentado é o esperado pois, tal como era pedido no enunciado, conseguimos criar um Sistema Solar dinâmico.