Universidade do Minho

Mestrado integrado em Engenharia Informática

Unidade Curricular de Computação Gráfica

Curves, Cubic Surfaces and VBOs

Relatório

Trabalho elaborado por:
Carlos Pedrosa - a77320
David Sousa - a78938
Manuel Sousa - a78869

Índice

Índice de ilustrações

Introdução

No âmbito da unidade curricular de Computação Gráfica foi-nos proposto o desenvolvimento de um mini cenário baseado em gráficos 3D. Assim, o projeto encontra-se dividido em 4 fases. Neste relatório pretendemos demonstrar todos os passos que foram efetuados no sentido a cumprir a terceira tarefa proposta.

Nesse sentido, a terceira fase encontra-se dividida em duas partes. A primeira prende-se com a criação de um novo modelo baseado em *patches* de Bezier. Assim, nesta parte o foco passa por gerar corretamente um teapot segundo estas fórmulas. A segunda, por sua vez, pressupõe a elaboração de um sistema solar dinâmico. Assim, o programa terá de desenhar o modelo contemplado acrescentando ainda um cometa.

Fase 3

Desenho do sistema solar dinâmico

1. Alteração do parser

Nesta fase, as alterações ao parser *XML* foram mínimas. Desta vez, teríamos que considerar que certas operações teriam um cariz dinâmico, sendo que o tempo na aplicação destas era um fator decisivo (de notar que as transformações estáticas ainda existem). No caso da operação **"rotate"**, o atributo estático **"angle"** pode agora ser substituído pelo atributo **"time"** que representa o número de segundos que um certo objeto demora a fazer uma rotação de 360º em torno do seu próprio eixo. Já para a operação **"translate"**, a alteração da estrutura desta foi mais acentuada. Assim, se for estática, tem a mesma forma que nas fases anteriores. No entanto, se for dinâmica, terá um atributo, também este nomeado de "time", da forma: **<translate time="...">**. Esta operação irá anteceder os pontos que advém da implementação da curva de *Catmull-Rom*, pelo que, o atributo "time", representa o número de segundos que um certo objeto demorará a dar uma volta completa à curva gerada. Efetivamente, nesta fase é necessária a utilização da curva de *Catmull-Rom*, pelo que o parser terá ainda de ter capacidade para extrair os pontos que advém da implementação desta.

```
char* timeStr;
if ((timeStr = (char*)child2->Attribute("time"))) {
    time = atof(timeStr);

    for (const XMLElement* child3 = child2->FirstChildElement();
         child3;
         child3 = child3->NextSiblingElement())
    {
        char* pointX = (char*)child3->Attribute("X");
        if (pointX != NULL) x = atof(pointX);

        char* pointY = (char*)child3->Attribute("Y");
        if (pointY != NULL) y = atof(pointY);

        char* pointZ = (char*)child3->Attribute("Z");
        if (pointZ != NULL) z = atof(pointZ);

        concatPoint(x, y, z, points);
    }

    tf = new TimeTransform();
    tf->time = time;
    tf->points = points;
}
```

```
else { // translate poderá também ser estático.
    char* translateX = (char*)child2->Attribute("X");
    if (translateX != NULL) x = atof(translateX);

    char* translateY = (char*)child2->Attribute("Y");
    if (translateY != NULL) y = atof(translateY);

    char* translateZ = (char*)child2->Attribute("Z");
    if (translateZ != NULL) z = atof(translateZ);
}
```

```
char* timeStr;
char* rotateAngle;
if ((timeStr = (char*)child2->Attribute("time"))) {
    time = atoi(timeStr);
    vector<coords> points;

    tf = new TimeTransform();
    tf->time = time;
    tf->points = points;
} else if ((rotateAngle = (char*)child2->Attribute("angle"))) {
    angle = atof(rotateAngle);
}
```

Figura 1 –Alteração do rotate e do translate

Em suma, a extração da informação relativa aos diversos tempos, implicou a criação de novas estruturas. Se efetivamente existisse um atributo "time" nas operações anteriormente referidas, então era criada uma ***TimeTransform*** - explicado mais à frente na secção das estruturas de dados - que tratava de guardar o tempo, bem como os pontos da curva (no caso de a operação ser um “translate”). Depois disto, a informação era adicionada às estruturas de dados principais, através da mesma metodologia seguida na fase anterior.

2. Criação de novas estruturas

Uma vez que é necessária a extração de novos elementos, foi necessária a criação de estruturas que permitam guardar estas novas alterações. Com o intuito de controlar da melhor forma as transformações dinâmicas, optamos por distingui-las das restantes transformações. Para isso, criamos uma estrutura de dados nomeada de **TimeTransform**, a qual guarda o valor de atributo "time" (**double time**), bem como um vetor de pontos que constituem uma determinada curva (**vector<coords> points**). No caso da operação "rotate", esta não tem pontos, pelo que consideramos que o **vector<coords> points** ficará vazio. A inserção desta nova estrutura de dados implicará uma pequena alteração numa das estruturas previamente criadas, **Oper**. De facto, esta passará a incluir um apontador para uma estrutura **TimeTransform**. Isto significa que cada **Oper** inserida em **FileOper** terá um apontador para essa nova estrutura. É importante referir que a escolha do apontador se deveu ao facto de nem todas as operações serem dinâmicas, e por essa mesma razão não terem um atributo "time". Com isto, atribuímos **NULL** ao respetivo apontador para **TimeTransform** de uma determinada transformação estática. Como referido nas fases anteriores, usamos sempre uma estrutura auxiliar que ia guardando as hierarquias do ficheiro *XML*, para assim aplicar as respetivas operações a um objeto de forma correta. Por essa mesma razão, a nossa estrutura de dados **LOper** também foi modificada, tendo assim também um apontador para uma **TimeTransform**.

```
typedef struct ttransform {
    double time;
    vector<coords> points;
} TimeTransform;
```

Figura 2 - Estrutura TimeTransform

3. Implementação dos VBOs

Nesta fase, foi requisitado a alteração do código para que este possa incluir VBOs. Depois de extraída toda a informação sobre as coordenadas de um ficheiro com a função ***extractCoords****,* foi necessário guardar todas estas numa matriz designada por ***vertexes,*** que por sua vez, vai ser usado na transferência de informação para um *GLuint* buffer. Assim, uma vez que ao extrair as coordenadas acrescentávamos uma marca

para identificar cada três vértices, a passagem destes para um array de VBOs aconteceu de forma bastante simplificada. É ainda importante realçar que usamos um VBO para cada ficheiro, de modo a, conseguir separar as respetivas transformações.

```
if (!isMark(aux_1, aux_2, aux_3)) {
    // Primeiro ponto do triângulo.
    vertexes[i][p]     = get<0>(aux_1);
    vertexes[i][p + 1] = get<1>(aux_1);
    vertexes[i][p + 2] = get<2>(aux_1);

    // Segundo ponto do triângulo.
    vertexes[i][p + 3] = get<0>(aux_2);
    vertexes[i][p + 4] = get<1>(aux_2);
    vertexes[i][p + 5] = get<2>(aux_2);

    // Terceiro ponto do triângulo.
    vertexes[i][p + 6] = get<0>(aux_3);
    vertexes[i][p + 7] = get<1>(aux_3);
    vertexes[i][p + 8] = get<2>(aux_3);

    p += 9;
}
```

Figura 3 - Preenchimento do vertexes, array de VBOs

4. Desenho do teapot através dos patches de Bezier

Outro dos objetivos desta fase era o de desenhar um teapot através dos patches de Bezier fornecidos pelo docente. Nesse sentido, começamos por extrair a informação contida nesse ficheiro. Para tal, criamos a função ***generateBezier***. Inicialmente guardamos os índices de cada patch numa matriz. Efetivamente, declaramos esta matriz como ***int indices[nPatches][16]*** sendo ***nPatches*** o número de patches indicados na primeira linha do documento. Posteriormente, guardamos também os pontos de controlo definidos no ficheiro, numa outra matriz. Matriz essa identificada por ***double points[nCtrlPoints][3].*** Depois de recolhida toda a informação necessária e de guardada nas respetivas estruturas preocupamo-nos com o desenho do teapot. Para tal, consultamos o formulário disponibilizado pelo professor, o que facilitou todo o processo (todo este processo foi efetuado numa função auxiliar designada por ***generateFigure***). Assim, a imagem seguinte ilustra todos os passos que efetuamos.

$$B(u,v) = \underbrace{[u^3 \quad u^2 \quad u \quad 1]}_{1} \underbrace{M}_{2} \underbrace{\begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}}_{3} \underbrace{M^T}_{4} \underbrace{\begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}}_{5}$$

1

```
double uW[1][4] = { {powf(u,3), powf(u,2), u, 1} };
```

2

```
double m[4][4] = {  {-1.0f,  3.0f, -3.0f,  1.0f},
                    { 3.0f, -6.0f,  3.0f,  0.0f},
                    {-3.0f,  3.0f,  0.0f,  0.0f},
                    { 1.0f,  0.0f,  0.0f,  0.0f}    };
```

3

```
for(int i = 0; i < 4; i++){
    for(int j = 0; j < 4; j++){
        pointsX[i][j] = points[indices[i*4+j]][0];
        pointsY[i][j] = points[indices[i*4+j]][1];;
        pointsZ[i][j] = points[indices[i*4+j]][2];;
    }
}
```

4

```
double mT[4][4] = {  {-1.0f,  3.0f, -3.0f, 1.0f},
                     { 3.0f, -6.0f,  3.0f, 0.0f},
                     {-3.0f,  3.0f,  0.0f, 0.0f},
                     { 1.0f,  0.0f,  0.0f, 0.0f}     };
```

5

```
double vH[4][1] = { {powf(v,3)},
                    {powf(v,2)},
                    {powf(v,1)},
                    {powf(v,0)}, };
```

```
/*  Computar Res = U * M    */
multMatrix14(uW, m, res);

/*  Computar ResX = Res * Px; ResY = Res * Py; ResZ = Res * pz  */
multMatrix14(res, pointsX, resX);
multMatrix14(res, pointsY, resY);
multMatrix14(res, pointsZ, resZ);

/*  Computar TransX = resX * M'; TransY = resY * M'; TransZ = resZ * M'  */
multMatrix14(resX, mT, transX);
multMatrix14(resY, mT, transY);
multMatrix14(resZ, mT, transZ);

/*  Computar x = transX * V; y = transY * V; z = transZ * V  */
multMatrix1441(transX,vH,&x);
multMatrix1441(transY,vH,&y);
multMatrix1441(transZ,vH,&z);

/*  Colocar os resultados no vetor res   */
r[0] = x;
r[1] = y;
r[2] = z;
```

Figura 4 - Algoritmo de Bezier e respetiva transformação em código

Depois de efetuarmos todos os cálculos necessários, invocamos a função *generateFigure,* conforme demonstra a figura seguinte.

```
/* Computar Todos os Pontos Necessarios para Desenhar o Teapot a Partir de Bezier. */
for (int i = 0; i < nPatches; i++) {
    for (int u = 0; u < tes; u++) {
        for (int v = 0; v < tes; v++) {
            double uD = (double)u/tes;
            double vD = (double)v/tes;
            generateFigure(uD,vD,indices[i], points, res);

            // Escrever para o Ficheiro
            if (file.is_open()) {
                file << res[0] << " " << res[1] << " " << res[2] << endl;
                lines++;
            } else {
                cout << "fileWriter not open" << endl;
            }
        }
    }
}
```

Figura 5 - Escrita no ficheiro e pontos necessários para a criação do Teapot

Assim, no final, obtemos o seguinte teapot:

Figura 6 - Teapot obtido

5. Curva de Catmull-Rom

A curva de Catmull-Rom segue o mesmo princípio aplicado para o desenho do teapot segundo a fórmula de Bezier. No entanto, a formula que aplicamos foi a seguinte:

$$[t^3 \quad t^2 \quad t \quad 1]\begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1 & -2.5 & 2 & -0.5 \\ -0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -0.5t^3 + t^2 - 0.5t \\ 1.5t^3 - 2.5t^2 + 1 \\ -1.5t^3 + 2t^2 + 0.5t \\ 0.5t^3 - 0.5t^2 \end{bmatrix}^T \qquad (24)$$

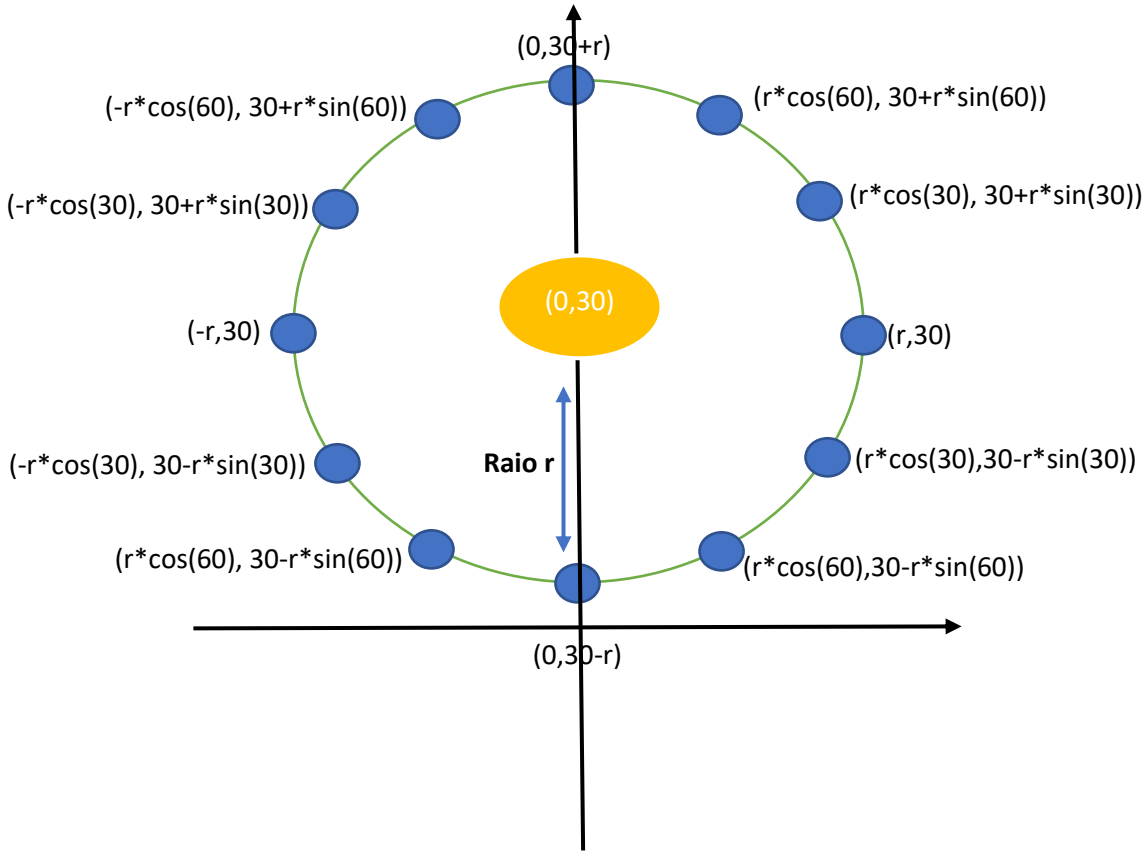
Figura 7 - Fórmula de Catmull-Rom

De resto, segue o mesmo princípio que aplicamos para desenhar o teapot. O desafio foi apenas na correta definição dos pontos. De forma a exemplificar este processo, usamos o esquema seguinte para demonstrar os passos que demos na obtenção dos pontos da curva. É importante realçar que para que a curva tivesse a forma de uma circunferência foi necessária a utilização de 12 pontos.

```
glBegin(GL_LINE_LOOP);
    int npoints = 100;
    for (int i = 0; i < npoints; i++) {
        getGlobalCatmullRomPoint((double) i / npoints, pos, deriv, curvePoints, size);
        glVertex3d(pos[0], pos[1], pos[2]);
    }
glEnd();
```

Figura 8 - Desenho de uma curva segundo Catmull-Rom

Tendo em consideração que y = 0 em todos os pontos.



6. Desenho do sistema solar dinâmico

Sendo que nesta fase o sistema solar teria de ser dinâmico, começamos por alterar o *XML* que estava associado à criação deste. Assim uma preocupação que tivemos de ter deveu-se ao facto de termos de definir corretamente o tempo de translação dos diversos planetas. No entanto, optamos por usar valores aproximados das suas translações para que fosse mais notória a diferença entre os planetas.

Planeta	**Período de translação em anos terrestres**
Mercúrio	0,241
Vénus	0,615
Terra	1,000
Marte	1,881
Júpiter	11,860
Saturno	209,460
Úrano	84,010
Neptuno	164,800
Plutão	1771,31

```
<scene>
    <group>
        <translate time="3" >
            <point X="0" Y="0" Z="6" />
            <point X="12" Y="0" Z="9.216" />
            <point X="20.784" Y="0" Z="18" />
            <point X="24" Y="0" Z="30" />
            <point X="20.784" Y="0" Z="42" />
            <point X="12" Y="0" Z="50.784" />
            <point X="0" Y="0" Z="54" />
            <point X="-12" Y="0" Z="50.784" />
            <point X="-20.784" Y="0" Z="42" />
            <point X="-24" Y="0" Z="30" />
            <point X="-20.784" Y="0" Z="18" />
            <point X="-12" Y="0" Z="9.216" />
        </translate>
        <models>
            <model file="sphere.3d" /> <!-- Earth -->
        </models>
```

Figura 9 - XML para gerar o planeta Terra

Na figura seguinte, podemos ver o sistema solar que resultou da configuração XML apresentada. É de realçar a presença do cometa, representado pelo *teapot* gerado através das curvas de Bezier.

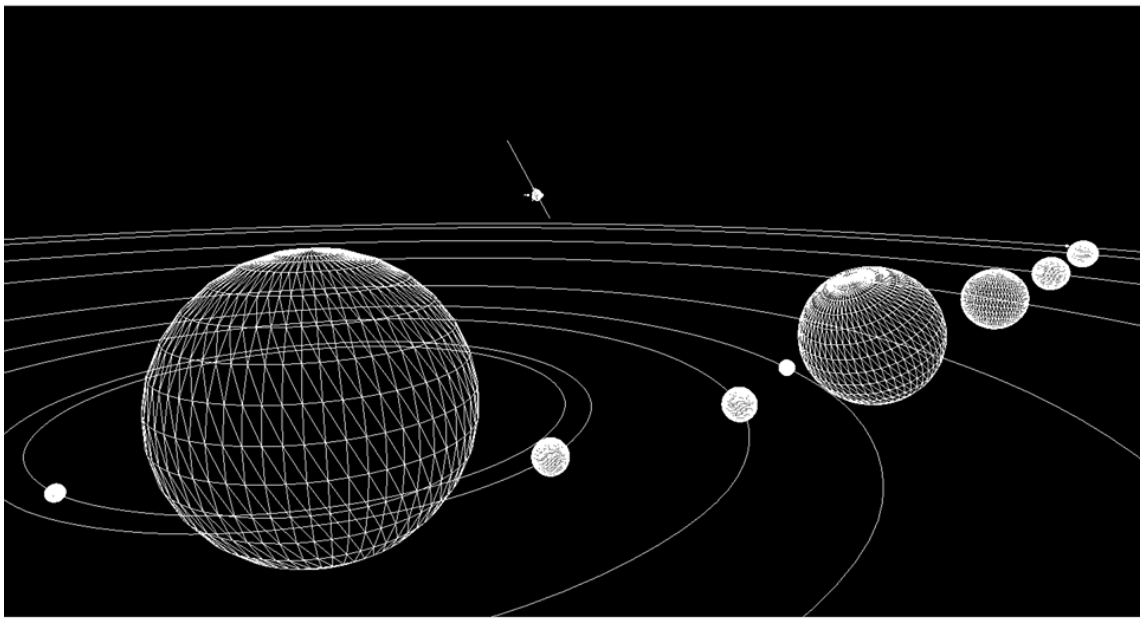


Figura 10 - Sistema solar obtido

Conclusão

De acordo com o objetivo definido na terceira fase consideramos que este foi alcançado. De facto, todos os propostos no enunciado foram conseguidos.

Assim, a presente fase do trabalho permitiu obter um conhecimento mais prático relativamente às diferentes transformações. Foi ainda possível perceber o aumento desempenho que a alteração do modelo anterior para VBOs proporcionou. O principal desafio com que nos deparamos foi numa fase inicial na perceção das formulas relacionadas com o desenho das curvas. Para além disto, o elevado consumo de memória de todo o programa também se revelou um desafio. No entanto, depois de ultrapassados todo o projeto surgiu naturalmente

Em suma, o presente trabalho tornou-nos mais objetivos e racionais sendo que foi uma mais valia para todos os elementos do grupo.

Bibliografia

- Notes for an Undergraduate Course in Computer Graphics, university of minho, António Ramires.