Universidade do Minho

Mestrado Integrado em Engenharia Informática

Computação Gráfica
Parte III - Curvas, Superfícies Cúbicas e VBOs

Grupo 44




Bernardo Mota (A77607)
Gonçalo Duarte (A77508)
Luís Neto (A77763)
João de Macedo (A76268)

Conteúdo

1 Introdução

Este relatório documenta o desenvolvimento da terceira fase do projeto de Computação Gráfica, mostrando a abordagem e as decisões tomadas ao longo desta fase. O objetivo desta fase do projeto é alterar o gerador de pontos para conseguir gerar pontos baseado em superfícies de Bezier, alterar o motor 3D para desenhar utilizando VBOs e acrescentar rotação e translação dinâmica utilizando curvas de *Catmull-Rom*.

Na secção do Gerador de Pontos é explicado o processo de interpolação de pontos através de ficheiros com superfícies de Bezier. Seguidamente são apresentadas as alterações ao motor 3D para este utilizar VBOs para o desenho dos modelos e as alterações feitas às funcionalidades de translação e rotação.

Posteriormente é apresentado um modelo do Sistema Solar dinâmico, com a translação e rotação dos planetas e de um cometa.

2 Gerador de Pontos - Superfícies de Bezier

Começando pelo Gerador de Pontos, foi adicionado a geração de modelos a partir de patches de superfícies de Bezier. Este processo consiste em gerar pontos que compõem os triângulos a ser desenhados pelo motor a partir de um ficheiro com os pontos de controlo de superfícies de Bezier e do nível de tesselação.

Este processo é feito a partir das fórmulas presentes no formulário facultado na Blackboard, mais concretamente as fórmulas das Superfícies de Bezier. Baseado nestas fórmulas, foi desenvolvida a função *generateBezier*.

```
void generateBezier(const string fileName, const string patch
    , int tesselation){
    string line;
    int cpoints;
    int npatches;
    vector<Point> points;
    vector<vector<float>> M = {
        {-1,3,-3,1},
        {3,-6,3,0},
        {-3,3,0,0},
        {1,0,0,0}
    };
    vector<vector<int>> patch_index;
    vector<vector<float>> xyzcomponent;

    std::ifstream input = patchIndexBuild(line, npatches, patch
        , patch_index);
    xyzCompBuild(cpoints, input, line, xyzcomponent);
    patchPointsBuild(M, npatches, patch_index, points, xyzcomponent
        , tesselation);
    write3DModel(fileName, points);
    return;
}
```

A matriz patch_index irá conter os índices dos pontos de todos os patches e a matriz xyzcomponent irá conter todos os pontos de controlo, com as coordenadas x,y e z.

Primeiramente é chamada a função *patchIndexBuild*, que preenche a matriz patch_index com os índices dos pontos presentes no ficheiro com as patches. Depois de construída a matriz dos índices dos pontos, a função *xyzCompBuild* preenche a matriz xyzcomponent com todos os pontos de controlo presentes no ficheiro com as patches.

Depois de termos lido toda a informação relevante do ficheiro patch e guardado esta informação em memória, passamos à interpolação dos pontos de controlo, com a função *patchPointsBuild*.

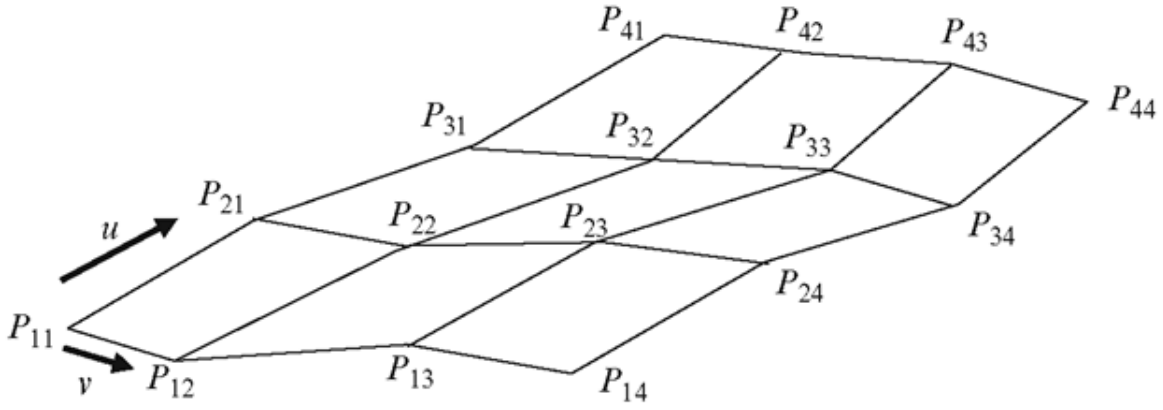
```
static void patchPointsBuild(...) {
    float step = 0.1/tesselation;
    for (int i = 0; i<npatches; i++){
        ...
        patchBuild(patch_index.at(i),xyzcomponent, &x,&y,&z);

        x = prod(M,x);
        x = prod(x,M);
        y = prod(M,y);
        y = prod(y,M);
        z = prod(M,z);
        z = prod(z,M);
        for (float u = 0; u<1; u+=step) {
            for (float v = 0; v<1; v+=step) {
                doSquare(u,v,x,y,z,&points,step);
            }
        }
    }
}
```

Dada a tesselação pretendida, a função interpola os pontos segundo a fórmula apresentada a seguir.

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

É feito o produto das matrizes com os pontos e a matriz M (e M transposta), e depois são calculados os pontos para formar todos os quadrados (formados por dois triângulos) resultantes da interpolação, através da função *doSquare*.

Segue-se uma explicação visual da interpolação da superfície de Bezier que está a ser feita, para melhor compreensão do algoritmo. Cada ponto P é um ponto de controlo da patch e a variação do u e do v depende da tesselação escolhida.



3 Motor 3D

3.1 VBOs

Nesta fase do trabalho um dos objectivos proposto era a implementação de *VBOs* que visam um aumento de desempenho do Motor 3D.

Na 2ª fase do trabalho prático foram apresentadas as seguintes funções

```
int loadModel(XMLElement* node_element, Group * group)
```

```
void Primitive::drawModel()
```

Estas funções carregam e renderizam uma *VBO*, respetivamente. Os proximos tópicos explicarão cada uma das funções acima apresentadas.

3.1.1 Leitura do Modelo

Na função *loadModels*, como explicado na 2º fase do trabalho, caso o modelo que se pretende carregar ainda não tenha sido carregado é executada a leitura do ficheiro binário. A leitura carrega os pontos lidos do ficheiro num *vector<float>* e de seguida é executada a função *genModel*.

```
void genModel(GLuint * idv, vector<float> points){
    glGenBuffers(1,idv);
    glBindBuffer(GL_ARRAY_BUFFER,*idv);
    glBufferData(GL_ARRAY_BUFFER, sizeof(float) * points.size(),
            &(points[0]), GL_STATIC_DRAW);
}
```

Esta função aloca um *buffer object* no *idv* através da função *glGenBuffers*. De seguida é executada a função *glBindBuffer* para que seja possível carregar os pontos contidos em *points* através da função *glBufferData*.

Por fim é criado um objeto *Primitive* cujas variáveis de instância são o *idv* e o tamanho do *vector<float> points*

3.1.2 Renderização do Modelo

No momento que se pretede renderizar um modelo a função *drawModel* invoca a seguinte função:

```
void Model::drawModel(){
    glBindBuffer(GL_ARRAY_BUFFER, Primitive::id);
    glVertexPointer(3,GL_FLOAT,0,0);
    glDrawArrays(GL_TRIANGLES, 0, Primitive::size);
}
```

Esta função chama *glBindBuffer* para ativar o *vertex object* que contem o modelo. De seguida é chamada a função *glVertexPointer* para definir o modo de desenho. Por fim é chamada a função *glDrawArrays* que renderiza a imagem do *vertex object* ativo.

3.2 Rotação Dinâmica

Para implementar a rotação dinâmica foi alterada a função *loadRotate* da classe Rotate.

```
int loadRotate(XMLElement* node_element, Group * group){
    const XMLAttribute * att = node_element->FirstAttribute();
    if (att == nullptr) return 0;
    float x = 0, y = 0, z = 0, angle = 0, time = 0;
    do{
        if( !strcmp(att->Name(),"axisX") ){
            x = stof(att->Value());
        }else if( !strcmp(att->Name(),"axisY") ){
            y = stof(att->Value());
        }else if( !strcmp(att->Name(),"axisZ") ){
            z = stof(att->Value());
        }else if( !strcmp(att->Name(),"ANGLE") ){
            angle = stof(att->Value());
        }else if( !strcmp(att->Name(),"time") ){
            time = stof(att->Value()), angle = (float) 360/(time*1000);
        }
        att = att->Next();
    }
    while (att != nullptr);
    Rotate *r = new Rotate(x,y,z,angle,time);
    return group->addTransformation("rotate", r);
}
```

Para além do vetor e do ângulo sobre os quais é feita a rotação foi adicionada a funcionalidade da leitura do atributo *time*. Através da leitura deste é calculado o ângulo que tem ser aplicado a cada milisegundo, possibilitando a rotação dinâmica. O atributo time apenas é utilizado caso seja lido um atributo time na leitura do XML.

Posto isto a função *transform* foi alterada:

```
void Rotate::transform(int routes){
    if (Rotate::time) {
        glRotatef(glutGet(GLUT_ELAPSED_TIME)*Rotate::angle, Rotate::x
            , Rotate::y, Rotate::z);
    }else{
        glRotatef(Rotate::angle, Rotate::x, Rotate::y, Rotate::z);
    }
}
```

A alteração consiste em verificar se a variável de instância *time* é maior que zero, caso seja, é aplicada uma rotação segundo o valor dado pela multiplicação da variável de instância *angle* pelo tempo decorrido desde a chamada da funçaõ *glutInit*.

3.3 Translação Dinâmica

Na realização desta componente do trabalho prático foram realizadas várias alterações na classe Translate e na leitura do XML que serão explicadas a seguir.

3.3.1 Leitura do XML

Para ser possível ler os pontos que definem a trajetória de uma translação foi alterada a função *loadTranslate*. A alteração efetuada foi a adição da leitura do atributo *time*. Com isto, quando o parser encontra um destes atributos lê o valor do atributo *time* e utiliza a função *loadTranslatePoints* para ler todos os pontos da trajectória. Por fim é instanciado um novo objecto Primitive e este é inserido no Group, como na fase anterior.

```
int loadTranslate(XMLElement* node_element, Group * group){
    const XMLAttribute * att = node_element->FirstAttribute();
    if (att == nullptr) return 0;
    float x = 0, y = 0, z = 0, angle = 0, time=0;
    do{
        if( !strcmp(att->Name(),"X") ){
            x = stof(att->Value());
        }else if( !strcmp(att->Name(),"Y") ){
            y = stof(att->Value());
        }else if( !strcmp(att->Name(),"Z") ){
            z = stof(att->Value());
        }else if( !strcmp(att->Name(),"angle") ){
            angle = stof(att->Value());
        }else if( !strcmp(att->Name(),"time") ){
            vector<vector<float>> points;
            time = stof(att->Value());
            loadTranslatePoints(node_element->FirstChildElement(),&points);
            Translate *t = new Translate(points,time);
            return group->addTransformation("translate", t);
        }
        att = att->Next();
    }while (att != nullptr);
    Translate *t = new Translate(x,y,z,angle);
    return group->addTransformation("translate", t);
```

3.3.2 Classe Translate

Para esta classe suportar a funcionalidade de fazer uma translação dinâmica, tendo em conta uma trajectória, foram adicionadas as seguintes variáveis de instância.

- vector< vector< float > > route, utilizada para guardar os pontos da trajectória
- vector< float > ypos, utilizado no algoritmo de Catmull-Rom
- bool time_bool, indica se o algoritmo de Catmull Rom é aplicado ou não
- float time, indica a duração da translação
- bool deriv, indica se é necessário aplicar a derivada à translação

Foi também criado um novo método de instanciação para suportar as novas variáveis.

```
Translate::Translate(vector<vector<float>> route, float time, bool deriv)
```

De referir que caso seja utilizado este constructor o *ypos* é o primeiro *vector* da *route*.

Por fim, foi alterada a função *transform* de modo a utilizar o algoritmo de Catmull-Rom caso exista uma trajectória.

```
void Translate::transform(int routes){
    if (Translate::time_bool) {
        if (routes) {
            drawRoute();
        }
        catmullRomTranslate();
    }else {
        glTranslatef(Transformation::x, Transformation::y
            , Transformation::z);
    }
}
```

A função *catmullRomTranslate* foi derivada atráves de um problema semelhante abordado e resolvido nas aulas práticas da unidade curricular, por isso não será explicada neste relatório.

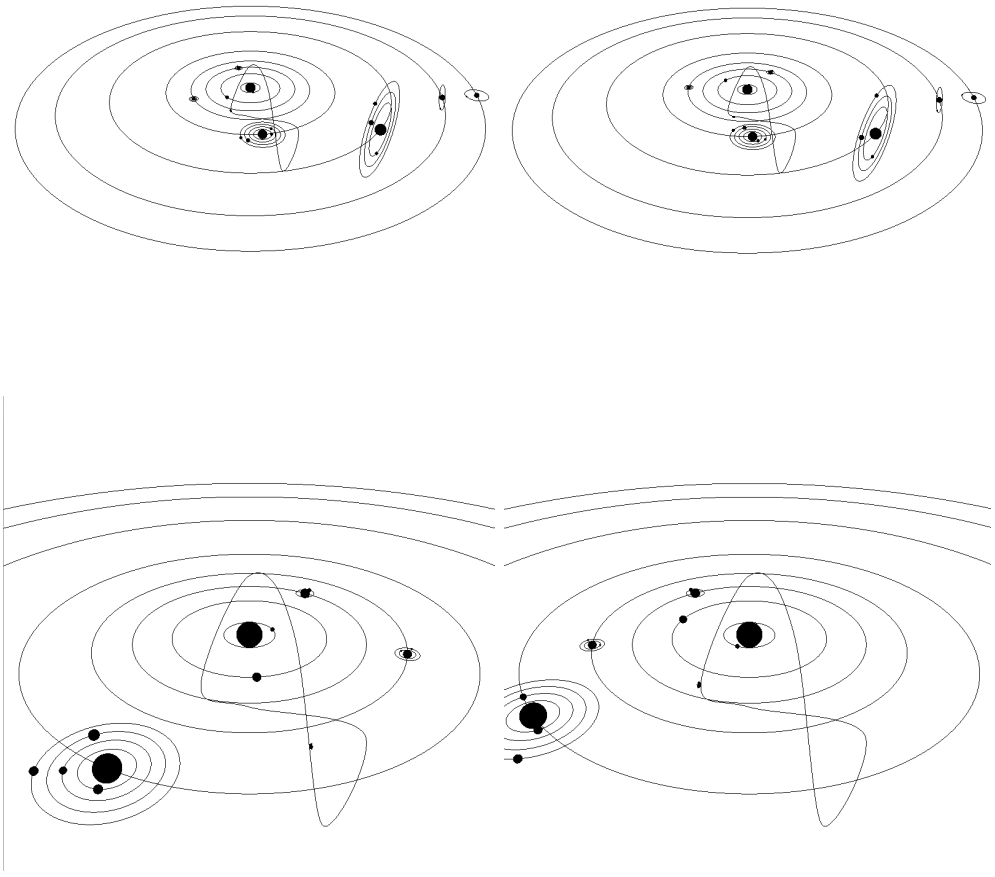
4 Demonstração do Sistema Solar

Após a implementação de todas as funcionalidades desta fase do projeto, o grupo passou à demonstração destas com o desenvolvimento do modelo do Sistema Solar concebido para a fase anterior do projeto.

Foram utilizadas rotações dinâmicas para a rotação dos planetas (e luas) e translações dinâmicas para a translações destes à volta do Sol. Para além dos planetas que já se encontravam no modelo da fase anterior, foi adicionado um cometa (utilizando o modelo *teapot* gerado a partir do ficheiro com as patches do *teapot*) com uma translação dinâmica.

Foram também adicionadas as curvas que descrevem as translações de cada planeta (e lua) e do cometa.

A seguir apresenta-se a demonstração desenvolvida.



5 Conclusões

Foram cumpridos todos os objetivos propostos nesta fase do projeto, tendo sido desenvolvido o gerador para gerar pontos a partir de superfícies de Bezier e adicionado a utilização de VBOs, translações e rotações dinâmicas ao motor 3D.

Para além dos objetivos propostos, foi adicionada a opção de mostrar as trajetórias das translação dinâmica.

A Anexos

A.1 Ficheiro XML para a Demonstração do Sistema Solar

```
<scene>
    <!--Sol-->
    <group>
        <scale X="1" Y="1" Z="1"/>
        <rotate time="10" axisY="1"/>
        <model file="sphere.3d"/>
    </group>
    <!--Mercurio-->
    <group>
        <translate time="2.4081476209">
            <point ...\>
        </translate>
        <rotate time="10" axisY="1"/>
        <scale X="0.15" Y="0.15" Z="0.15" />
        <model file="sphere.3d"/>
    </group>
    <!--Venus-->

    <group>
        <translate time="6.1512347369">
            <point ...\>
        </translate>
        <rotate time="10" axisY="1"/>
        <scale X="0.3" Y="0.3" Z="0.3" />
        <model file="sphere.3d"/>
    </group>

    <!--Terra-->
    <group>
        <translate time="10">
            <point ...\>
        </translate>
        <rotate time="10" axisY="1"/>
        <scale X="0.4" Y="0.4" Z="0.4" />
        <model file="sphere.3d"/>
        <group>
            <translate time="5">
                <point ...\>
```

```
            </translate>
            <scale X="0.4" Y="0.4" Z="0.4" />
            <rotate time="10" axisY="1"/>
            <model file="sphere.3d"/>
        </group>
    </group>
    <!--Marte-->
    <group>
        <translate time="18.8079724032" >
            <point ...\>
        </translate>
        <rotate time="10" axisY="1"/>
        <scale X="0.30" Y="0.30" Z="0.30" />
        <model file="sphere.3d"/>
        <group>
            <translate time="7" >
                <point ...\>
            </translate>
            <scale X="0.2" Y="0.2" Z="0.2" />
            <model file="sphere.3d"/>
        </group>
        <group>
            ...
        </group>
    </group>
    <!--Jupiter-->
    ...
    <!--Saturno-->
    ...
    <!--Neptuno-->
    ...
    <!--Cometa-->
    <group>
        <translate time="20" deriv="true">
            <point ...\>
        </translate>
        <rotate axisX="1" ANGLE="-90"/>
        <scale X="0.1" Y="0.1" Z="0.1"/>

        <model file="teapot.3d"/>
    </group>

</scene>
```