Universidade do Minho

Computação gráfica

Departamento de Informática

Fase 3

Grupo 40:
João Abreu — A84802
Tiago Magalhães — A84485
Hugo Matias — A85370
Sérgio Gomes — A67645

4 de Maio de 2020

Conteúdo

1. Arquitetura do código

Devido ao facto de este trabalho ser no seguimento dos anteriores, tendo sido feitas diversas alterações necessárias a cada umas das seguintes aplicações a serem cumpridos os novos requisitos.

1.1 Aplicações

No presente tópico é apresentado todas as alterações feitas às duas principais aplicações do trabalho. Foram necessárias algumas alterações ao ficheiro **XML** e consequentemente alterações no motor de processamento deste ficheiro. Foi também alterado o generator de maneira a suportar algoritmos de Bézier.

1.1.1 Generator

Indo ao encontro do que foi feito nas fases anteriores, o generator destina-se a gerar vários pontos constituintes das várias primitivas gráficas conforme os parâmetros fornecidos. Nesta fase, para além das primitivas gráficas anteriores, foi introduzido um novo método de construção de modelos com base nas curvas de Bézier, sendo que foi necessário adicionar novas funcionalidades ao gerador em relação às fases anteriores.

1.1.2 Engine

O objectivo desta aplicação continua a ser o mesmo: permitir a apresentação de uma janela e exibição dos modelos requisitados. Além disso, permite, também, a interação com os mesmos a partir de certos comandos. Tal como na versão anterior, existe um ficheiro **XML** que vai ser interpretado. No entanto, para esta fase de projeto, a arquitetura deste ficheiro evolui da maneira a cumprir requisitos. Desta forma, foram feitas algumas alterações nos métodos de *parsing* e consequentemente alterações nas medidas de armazenamento e também de renderização.

2. Generator

2.1 Superfícies de Bézier

Para esta fase do projeto tivemos de processar um ficheiro .patch, para representar um cometa a implmentação será explicada a seguir.

2.1.1 Processamento do ficheiro input

Antes de processar o ficheiro de input *(.patch)*, tivemos que entender o formato do ficheiro para posteriormente gerar a figura.O ficheiro input está organizado da seguinte forma:

- Na primeira linha surge o número de patches .
- As próximas linhas que são no total o número de patches, têm cada uma, 16 números que correspondem aos índices de cada um dos pontos de controlo que fazem parte desse patch.
- Posteriormente aparece um inteiro que representa o número de pontos de controlo.
- Por fim surgem os pontos de controlo que, no total, são o número de pontos definidos na linha anterior.

Para processar então definimos uma função no **Generator** chamada **readBezierPatches** que irá guardar num *vector* os índices e noutro *vector* os pontos de controlo.Está função irá percorrer linha a linha do ficheiro e retirar toda a informação necessário como número de patches,pontos de controlo e preencher os *vector* índices e pontos de controlo.

2.1.2 Processamento de patches

Cada patch pode ser representado por um quadrado de 4 pontos, ou seja, dois triângulos após o parse do ficheiro *.patch* tivemos de associar os pontos aos patches, isto foi obtidos atráves do seguinte código:

```
 /* Cria vetor com os pontos do patch correspondente*/
for (int patch = 0; patch < numPatches; patch++) {

    vector<Ponto> patch_points;

    float tess = (float)1 / (float)tessellation;

    for (int p = 0; p < INDEX_PER_PATCH; p++) {
        patch_points.push_back(control_points[indices[patch * INDEX_PER_PATCH + p]]);
    }
```

Como cada *patch* tem 16 índices é fácil saber que cada patch irá começar nos 16 índices seguintes ao *patch anteiror*, sendo no código anterior

```
INDEX_PER_PATCH = 16;
```

Para obter os pontos do *patch* basta associar os índices aos pontos.

```
patch_points.push_back(control_points[indices[patch * INDEX_PER_PATCH + p]]);
```

Para gerar os pontos que irão definir as superfícies usamos o seguinte:

```
bezier_patch_point = generateBezierPoint(patch_points, u, v);
bezier_points.push_back(bezier_patch_point);
bezier_patch_point = generateBezierPoint(patch_points, u + tess, v);
bezier_points.push_back(bezier_patch_point);
bezier_patch_point = generateBezierPoint(patch_points, u, v + tess);
bezier_points.push_back(bezier_patch_point);
bezier_points.push_back(bezier_patch_point);
bezier_patch_point = generateBezierPoint(patch_points, u + tess, v);
bezier_points.push_back(bezier_patch_point);
bezier_patch_point = generateBezierPoint(patch_points, u + tess, v + tess);
bezier_points.push_back(bezier_patch_point);
```

onde $u, v \in [0, 1]$, como cada superfície tem 16 pontos de controlo, terá 4 linhas e 4 colunas,ou seja u=0 e v=0 no Ponto 0 da primeira linha e u=1 e v=0 no último ponto da primeira linha, assim tem de ocorrer uma variação em u,v para desenhar-mos os triângulos, como podemos ver pela imagem , para implementar a *tessellation* fizemos 2 ciclos *for* que irão percorrer cada linha e cada coluna da superfície, ou seja a *tesselation* corresponde a sub superfícies de uma superfície, isto é subdividir cada linha e coluna, se a *tessellation* for 20 termos de passar para o parâmetro u 1/20 para ocorres a subdivisão.

```
for (int i = 0; i <= tessellation ; i++) {
        for (int j = 0; j ) <= tessellation ; j++)
```

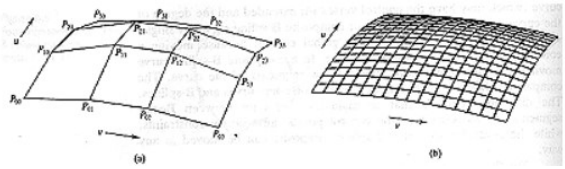

Figura 2.1: Superficie de Bezier.

A função que gera os pontos é a **generateBezierPoint** e para isso utilizamos a definição matricial, uma vez que apresenta mais vantagens computacionalmente e uma manipulção mais facilitada.

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

Figura 2.2: Formula matricial Superficies de Bezier.

3. Engine

3.1 VBOs

Uma das mudanças feitas ao nosso projeto foi implementar os *VBOs* para desenhar os diferentes modelos. Os *Virtual Buffer Objects* são uma funcionalidade oferecida pelo OpenGL, os quais nos permitem inserir informação sobre os vértices diretamente na placa gráfica do nosso dispositivo. Estes fornecem-nos uma performance bastante melhor, devido ao facto de a renderização ser feita de imediato pois a informação já se encontra na placa gráfica em vez de no sistema, diminuindo assim a carga de trabalho no processador e assim os pontos em vez de estarem a ser desenhados um a um, são passados para um buffer. Passando para a explicação do que foi alterado no nosso código, as alterações foram feitas na classe **Grupo**, cuja função de desenho foi alterada para a seguinte versão:

```
void Grupo :: draw(VBO models){
    glBindBuffer(GL_ARRAY_BUFFER, models.vertices);
    glVertexPointer(3, GL_FLOAT, 0, 0);
    glDrawArrays(GL_TRIANGLES, 0, models.size);
}
```

As suas váriáveis de instância foram também alteradas de modo a que cada ficheiro.3d lido possua o seu próprio conjunto de pontos no formato *VBO*. Assim, o nosso conjuntos de modelos passou a ficar deste modo:

```
vector<VBO> models;
```

Em que *VBO* é definido por:

```
struct VBO {
    int size;
    GLuint vertices;
};
```

Para que cada um dos ficheiros.3d tivesse o seu prórpio conjuntos de pontos VBO tivemos de alterar na classe *engine* a função que lê os ficheiros. Acrescentamos o seguinte à função *readFile*

```
// gera VBO
int size = model.size();
GLuint buffer;
VBO v;

glGenBuffers(1, &buffer);
glBindBuffer(GL_ARRAY_BUFFER, buffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model.size(),
            model.data(), GL_STATIC_DRAW);
```

```
v.size = model.size()/3 ;
v.vertices = buffer;

grupo.addModel(v);
```

3.2 Curva Catmull-Rom

3.2.1 Rotação

De modo a implementar novas formas de rotação, foi necessário adicionar uma variável tempo. Assim com esta variável é possível calcular o tempo que demora uma rotação de 360º. Recorrendo a esta variável formularam-se as seguintes formulas condicionadas:

```
if(time==0.0)
    glRotatef(angle,x,y,z);
else {
    glRotatef(tempo*(360/(time*1000)), x, y, z);
    }
```

Se a variável tempo for 0.0 entao aplica-se um glRotatef "normal"(sem recorrer à variável tempo, visto que esta não irá ter algum impacto), caso contrário é necessário calcular o angulo a partir do tempo que decorreu desde a inicialização do programa, a multiplicar pela divisão entre o angulo de uma volta completa(360º) e o tempo necessário para completar essa volta em milissegundos.

Com isto, as rotações irão depender do tempo.

3.2.2 Translação

A implementação de uma nova forma de translação foi também necessária. Para isso foram adicionadas duas novas variáveis à classe Translacao, uma variável "time"que corresponde ao tempo necessária para se fazer uma translacao, e outra variável "controlpoints"que corresponde a um vetor de Pontos que será usado para desenhar a curva de translação.

São criados também dois arrays auxiliares: res[3] que correspondem aos pontos para a próxima translação na curva, deriv[3] (derivada do ponto anterior). A inserção dos valores nos arrays é feita com recurso à função get GlobalCatmullRomPoint, onde esta última recorre também as variáveis "tm"e "controlpoints". Uma nova variável tm(tempo percentual) é calculada :

```
float x = (time * 1000);
tm = tempo / x;
getGlobalCatmullRomPoint(tm, controlpoints, res, deriv);
```

A função getGlobalCatmullRomPoint recorre a uma função getCatmullRomPoint que obtem os valores de deriv[3] e res[3] recorrendo a uma matriz "m".

Ao obtermos os resultados necessários poderemos utilizar a função glTranslatef para aplicar a translação.

Por fim é utilizada a função renderCatmullRomCurve para desenhar as curvas.

4. Resultados obtidos

4.1 Teapot

Nesta parte apresentamos o *teapot* cujo ficheiro de input é fornecido no enunciado do trabalho prático.

Figura 4.1: Demonstração do teapot.

4.2 Sistema Solar

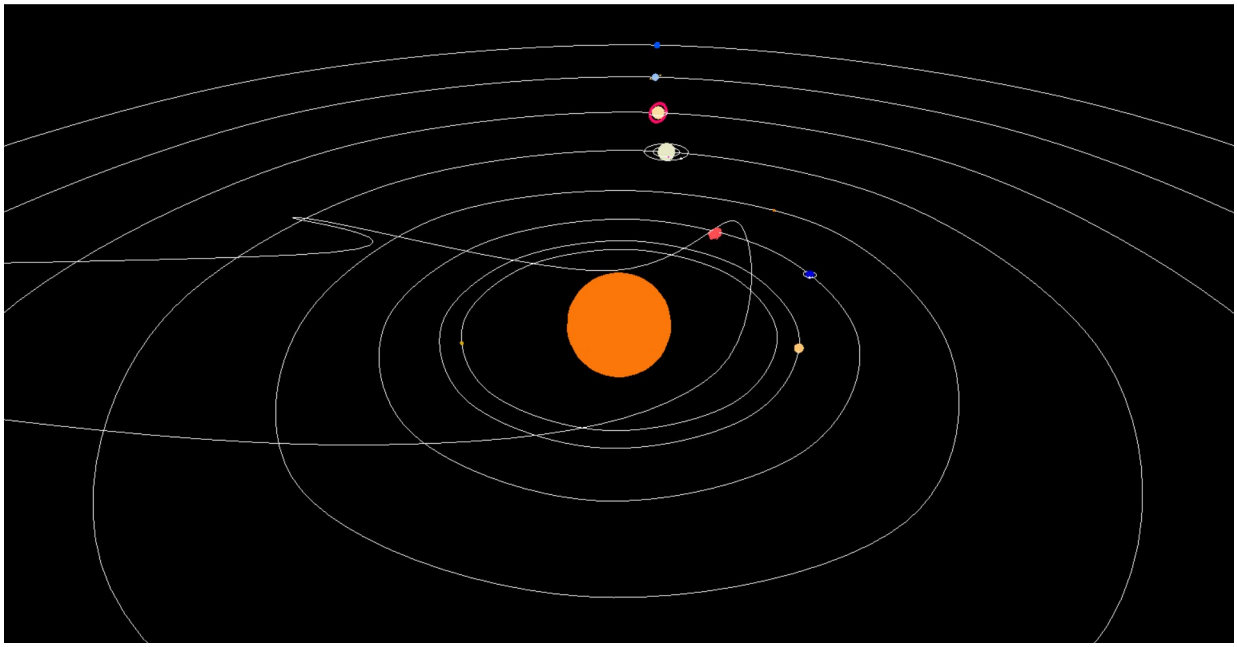

Figura 4.2: Demonstração do sistema solar (solid).

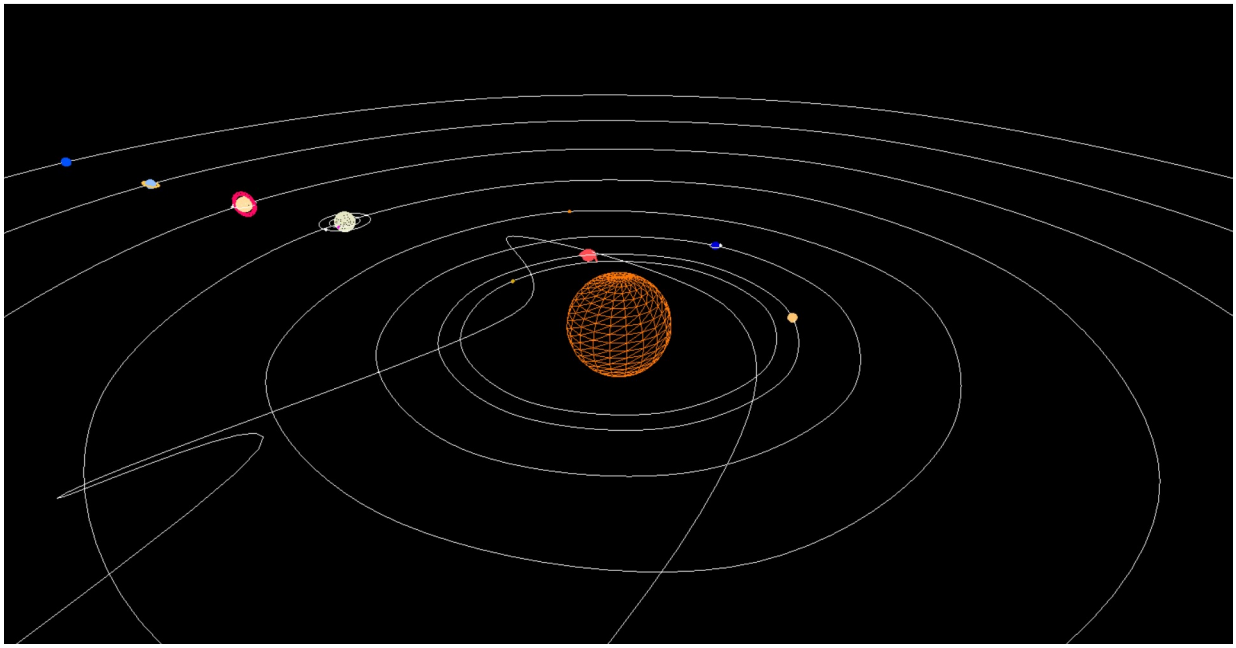

Figura 4.3: Demonstração do sistema solar (wired).

5. Alterações importantes feitas desde a última fase

O gerador passou a gerar os ficheiros para */files/3d/* e aquando a compilação do projeto, é copiado para a pasta *build* (Windows) e para a pasta *build/Debug/* (Mac) os ficheiros 3d e o ficheiro *teapot.patch*.

6. Referências

Bezier Surfaces:<http://homepages.inf.ed.ac.uk/rbf
/CVonline/LOCAL_ COPIES/AV0405/DONAVANIK/bezier.html>.Acesso em 01.mai.2020.

Bezier Curves and Surfaces:<www.cad.zju.edu.cn/home/zhx/GM/005/00-bcs2.pdf>Acesso em 01.mai.2020.