Universidade do Minho
Escola de Ciências

Computação Gráfica

Ano letivo 21/22

. Fase 3- Curvas, Superfícies Cúbicas

Grupo 10:

- Hugo Costeira - A87976
- João Silva - A87939
- João Goulart - A82643
- Pedro Martins - A87964

Índice

1. Introdução

Na terceira fase do projeto implementamos à fase anterior curvas e superfícies cúbicas. No generator geramos uma nova primitiva Teapot, baseada em Bezier Patches. Todas as primitivas são desenhadas no engine. Através de curvas de Catmull definimos animações para os planetas do Sistema Solar, sendo as translações de cada planeta definidas através de um conjunto de pontos que definem a curva, bem como um determinado tempo para percorrer a mesma, e as rotações definidas também à custa de um determinado período de tempo que caracteriza uma rotação de 360º.

2. Generator

Tal como nas fases anteriores, o Generator é responsável por criar ficheiros 3D que contêm os vértices necessários para a criação de cada uma das primitivas utilizadas, sendo estas a esfera que é utilizada para representar os planetas, o sol e as luas, o torus, utilizado para representar o anel de Saturno e agora também o teapot que será definido à custa de Bezier Patches.

2.1. TeaPot

Para tentar representar o teapot definimos a função void teapot (char* patchname, int tessellation, char* filename) que recebe como argumentos o ficheiro .patch que possui os patches de Bezier para o Teapot, o nível de tesselação e o ficheiro destino que conterá os pontos geométricos do Teapot. Assim, temos de fazer parsing e processar a informação presente no ficheiro .patch e , para isso, guardamos inicialmente os índices dos pontos de controlo do ficheiro num array de índices e os pontos de controlo em si num array de vértices. Após ter esta informação armazenada temos de a processar corretamente, no entanto para isso devemos entender como funcionam as curvas de Bézier.

Para criarmos uma curva cúbica de Bézier precisamos de 4 pontos de controlo no espaço tridimensional, após termos esses pontos é usada a fórmula seguinte:

$$B(t)=(1-t)3 \ast P0+3\ast t\ast(1-t)2 \ast P1+3\ast t2 \ast(1-t)\ast P2+t3 \ast P3$$

Onde t é uma variável que varia entre [0,1] e P0,P1,P2 e P3 são os 4 pontos de controlo mencionados anteriormente. O resultado da equação para qualquer t corresponde a uma determinada posição na curva, se resolvermos a equação para todos os t´s possíveis obtemos a curva completa.

Assim, para processarmos o ficheiro dado devemos aplicar esta mesma fórmula recursivamente, ou seja, devemos para cada conjunto de 16 vértices do ficheiro pegar em 4 de cada vez e com esses 4 definimos uma curva de Bézier. Assim, como resultado desses 16 vértices, ficamos com 4 curvas de Bézier que resultam, aplicando a fórmula de Bézier com um determinado variável v, em 4 pontos. Através desses 4 pontos definimos mais uma vez uma curva aplicando novamente a fórmula de Bézier, obtendo-se assim um vértice final que fará parte do Teapot.

Desta forma, utilizamos os vértices fornecidos e obtemos através deles os pontos de que precisamos, imprimindo-os no ficheiro final de forma a formar triângulos para posteriormente serem utilizados pelos VBOs.

3. Engine

O Engine corresponde à parte do projeto que utiliza os ficheiros 3d gerados pelo Generator e as transformações geométricas do ficheiro XML para representar a cena pretendida

3.1 Estruturas de dados

```
typedef struct caracteristicas {

        int t_time;
        int r_time;
        int align;

        float* translate;
        float* rotate;
        float* scale;
        //int* ordem;

        Buffer b;
        struct caracteristicas** luas;
        int nLuas;
}*Caracteristicas;
```

Foram feitas alterações na estrutura **caracteristicas** que permitem guardarmos informação acerca de cada transformação que é realizada no Sistema Solar, assim guardamos o tempo de cada translação e os pontos que a constituem, o tempo de cada rotação e os valores associados a ela, os valores que fazem parte da escala.

3.2 ReadXML

A implementação feita da função que faz *parsing* e interpreta o ficheiro xml fornecido é, estruturalmente, idêntica à da fase anterior. Por isso, iremos apenas nos focar nas implementações novas.

Introduzimos duas variáveis fundamentais novas: *Buffer b* e *caracteristicas c*, que utilizam as estruturas de dados explicadas anteriormente no relatório. b é usado para guardar o número de vértices a desenhar com o VBO para cada planeta ou lua. t é uma forma mais organizada para substituir *all_vertices* que usávamos na fase 1.

O ficheiro xml fornecido tem também uma estrutura ligeiramente diferente. A informação contida não é só usada para apenas manter os planetas estáticos, mas para criar a sua translação em torno do Sol e rotação sobre eles próprios. Por isso mesmo é que a variável t utilizada tem o parâmetro de *t_time* (tempo de translação) e *r_time* (tempo de rotação).

3.3 Transform

Esta função foi uma função criada para aplicar as transformações presentes no ficheiro xml fornecido. Está dividida em duas partes principais por uma condição if: transformações de planetas e transformações de luas. No caso dos planetas, são aplicadas as suas transformações, caso existam, retiradas da variável *c*. No caso das luas, começamos por aplicar uma translação estática dos planetas (isto porque estávamos a ter problemas em implementar a translação das luas em torno do Sol com os respetivos planetas), aplicamos a translação das luas iriam fazer em volta dos planetas e, em seguida, o resto das transformações que possam eventualmente ter na variável *c.*

3.4 Funções necessárias para representar as curvas de Catmull

A função **buildRotMatrix** cria uma matriz de rotação de dados 3 vetores x, y e z e devolve o resultado para um array m.

A função **normalize** normaliza um certo array que é passado como input. A função cross faz a multiplicação de dois vetores.

A função **multMatrixVector** retorna num array o resultado de multiplicar uma matriz por um vetor.

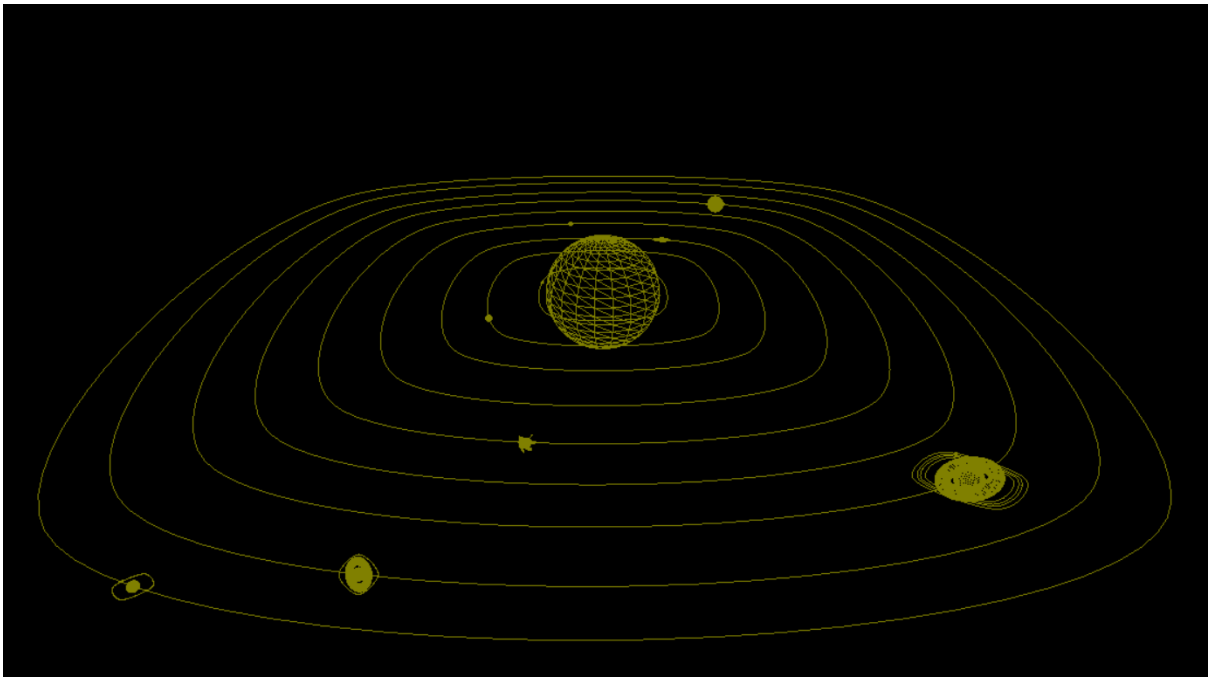
A função **getCatmullRomPoint** é responsável por computar a curva de Catmull-Rom.

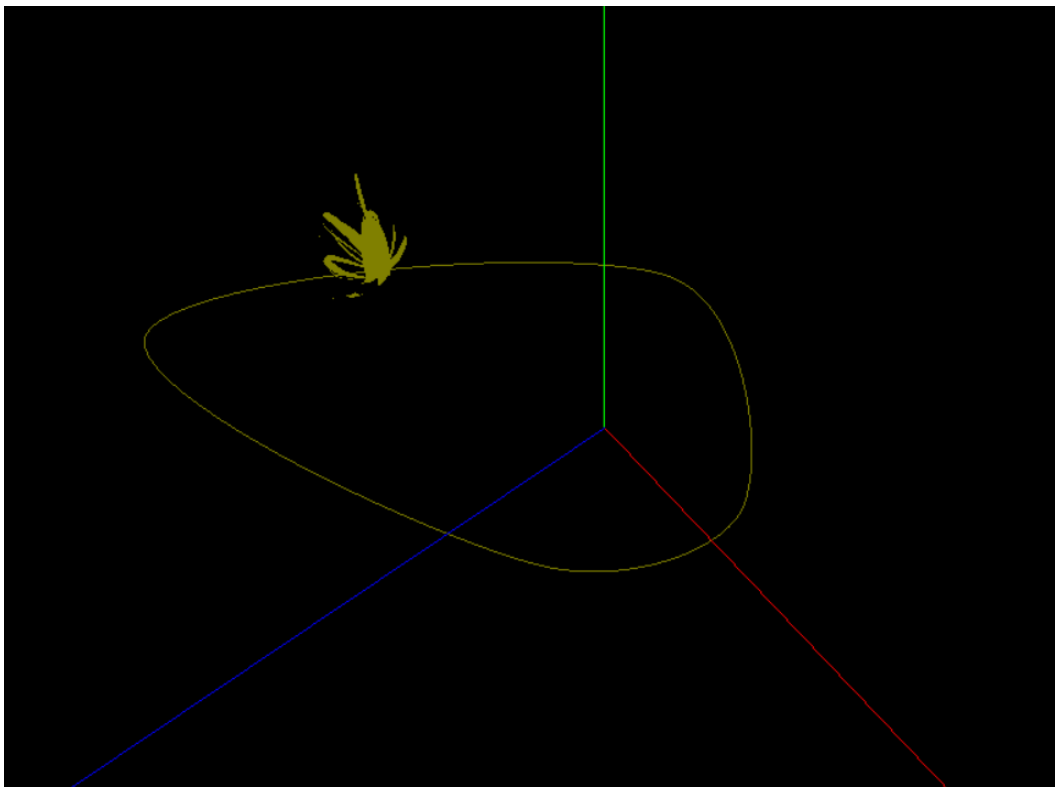
A função **getGlobalCatmullRomPoint** é necessária para obtermos os vetores p0, p1, p2 e p3 que representam as posições.

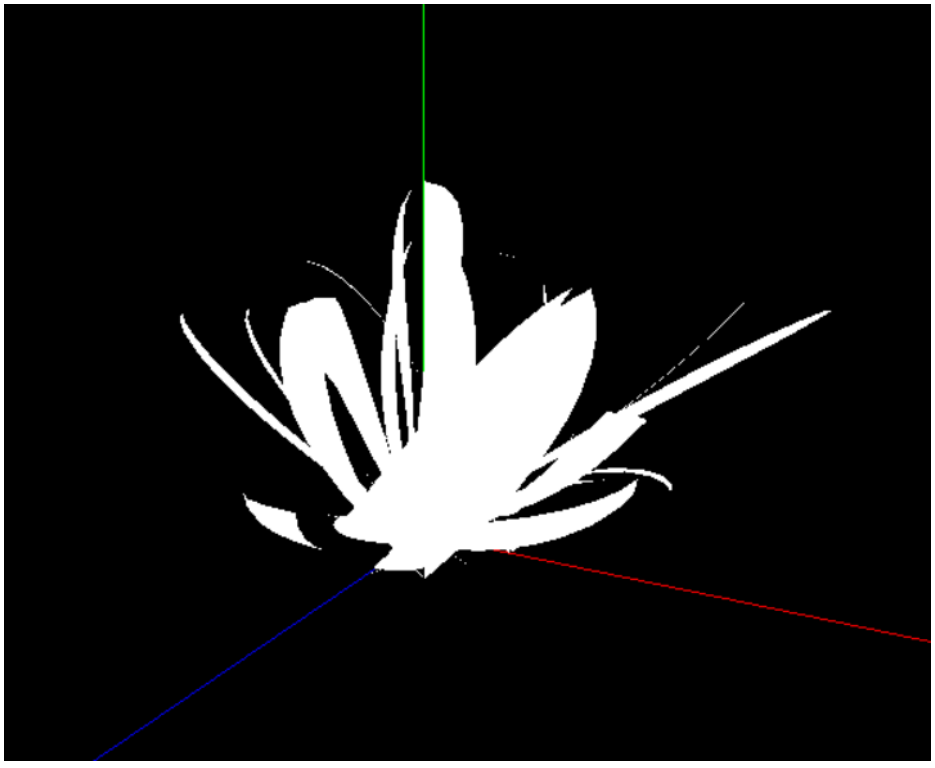
A função **renderCatmullRomCurve** vai representar as curvas em si, isto é, esta vai ser a função que vai desenhar as curvas tendo em conta o resultado da função getGlobalCatmullRomPoint, que por sua vez vai chamar a função getCatmullRomPoint.

A função **drawCatmullRomCurve** é a função que vai chamar as 3 funções enunciadas anteriormente, e vai ser responsável por fazer a translação.

4. Demonstração do resultado final







5. Conclusão

Este trabalho introduziu dois termos teóricos lecionados nesta unidade curricular sendo estes as curvas de Bezier e CatmullRom. Embora não tenhamos conseguido implementar o teapot de forma correta devido a um erro (que supomos que seja do código e não dos cálculos efetuados), o qual ainda não resolvemos.

Em relação às translações, o processo foi mais demorado e complexo, no entanto, o código está baseado no que foi feito nas aulas práticas onde se aplicou as curvas de CatmullRom sendo possível assim fazer translações.