Universidade do Minho
Escola de Engenharia

Computação Gráfica

Fase 3 - Curves, Cubic Surfaces and VBOs

Grupo 52

4 de Maio de 2020

MiEI - 3ºAno - 2ºSemestre

Beatriz Rocha A84003

Filipe Guimarães A85308

Gonçalo Ferreira A84073

José Mendes A75481

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Nesta 3ª fase do trabalho é proposto criarmos um novo tipo de modelo baseado em Bezier patches, transformando este num ficheiro de pontos que a nossa engine consiga interpretar. Mudamos também a nossa forma de desenhar, passando agora a utilizar VBO's. No lado da engine foram adicionadas novas transformções, as translações e rotações com tempo.

Capítulo 2

Introdução aos Bezier Patches

2.1 Model baseado em Bezier Patches

Nesta fase foi introduzida a possibilidade de criar models baseados em bezier patches. Para gerar este novo model foi adicionado mais uma opção de comando no generator. De seguida é apresentado um exemplo para gerar um model no ficheiro "teapotBezier.3d"a partir de um Bezier Patch de nome "teapot.patch"e em que o tesselation level é 10:

```
$ ./ generator bezierPatch teapot.patch 10 teapotBezier.3d
```

Um exemplo de um teapot é respresentado a seguir:

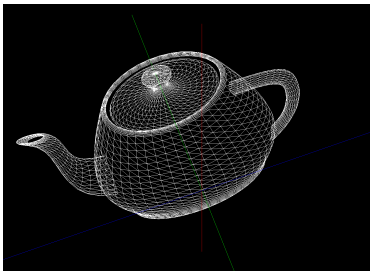

Figura 2.1: Teapot baseado num Bezier Patch

2.2 Resolução do problema

Para obtermos o model primeiro fizemos a leitura do ficheiro com o Bezier Patch e colocamos os diferentes valores lidos em estruturas adequadas. De seguida tivemos que analisar individualmente cada patch, composto por um conjunto de 16 pontos. Primeiro tive de verificar como é que lendo os pontos de um array, correspondentes aos pontos do patch, estes ficavam distribuídos no espaço. Observamos que à medida que ia lendo os pontos do array, deveria

ir colocando esses pontos numa matriz do género que é representado na figura 2.2: preencher da direita para a esquerda começando pela última linha.

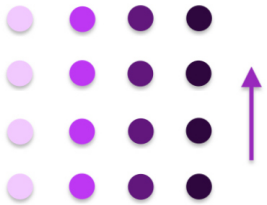

Figura 2.2: Representação abstracta de uma matriz com pontos de um patch

Ao construir a matriz desta forma descobrimos também por observação que o utilizador ao ver a figura 2.2, está a ver do lado de fora e não do lado de dentro. Esta conclusão é importante pois permite saber a ordem de escolha de pontos para o desenho dos triângulos.

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

Figura 2.3: Formula para determinar pontos de Bezier

Tendo a matriz de pontos correspondente ao patch, calculamos todos os pontos de Bezier para todos as combinações de valores de u e v. As combinações possíveis de u e v dependiam do tesselation level. Para o cálculo dos pontos de Bezier utilizamos a fórmula da figura 2.3. Fizemos este processo para os restantes patches e obtive o teapot. Convém notar que o algoritmo utilizado para desenhar os triângulos para cada patch foi em grande parte semelhante ao utilizado no desenho da esfera (feito para a fase 1).

Capítulo 3

Ficheiro xml

3.1 Transformações com tempo

Nesta frase foi introduzida as transformações com tempo usando por base as *Catmull curves*. Para as transformações precisamos, como se pode ver na imagem do tempo da mesmas bem como dos pontos que definem a trajetória.

```
...
<translate time=10 >
      <point X=1 Y=0 Z=1 />
      <point X=0.707 Y=0.707 Z=1 />
      <point X=0 Y=1 Z=1 />
      ...
      <point X=-1 Y=0 Z=1 />
</translate>
...
```

Figura 3.1: Exemplo de translação com tempo

Para conseguir criar a trajetória elíptica dos planetas desenvolvemos uma peça de software que fornecendo o *Periélio* e o *Afélio* da rota de um determinado planeta produz um output com 8 pontos pertencentes a essa trajetória já com o formato do xml. Este encontra os pontos nos eixos xOz e os pontos em que se igualam o X e o Z.

```
mercurio
<translate time="87969.000000">
    <point X="6.981690" Y="0.000000" Z="0.000000" />
    <point X="3.841274" Y="0.000000" Z="-3.841274" />
    <point X="0.000000" Y="0.000000" Z="-4.600120" />
    <point X="-3.841274" Y="0.000000" Z="-3.841274" />
    <point X="-6.981690" Y="0.000000" Z="0.000000" />
    <point X="-3.841274" Y="0.000000" Z="3.841274" />
    <point X="0.000000" Y="0.000000" Z="4.600120" />
    <point X="3.841274" Y="0.000000" Z="3.841274" />
</translate >
```

Figura 3.2: Output para Mercúrio

O código que gera estes pontos encontra-se na pasta extras e pode ser compilado com *gcc main.c* e executado com *./a.out > output.txt*

3.2 Rotações com tempo

Também nesta fase foram introduzidas rotações com o tempo. Estas são muito similares às antigas apenas passa a existir um campo tempo em vez de ângulo como podemos ver na seguinte imagem.

```
mercurio
<translate time="87969.000000">
    <point X="6.981690" Y="0.000000" Z="0.000000" />
    <point X="3.841274" Y="0.000000" Z="-3.841274" />
    <point X="0.000000" Y="0.000000" Z="-4.600120" />
    <point X="-3.841274" Y="0.000000" Z="-3.841274" />
    <point X="-6.981690" Y="0.000000" Z="0.000000" />
    <point X="-3.841274" Y="0.000000" Z="3.841274" />
    <point X="0.000000" Y="0.000000" Z="4.600120" />
    <point X="3.841274" Y="0.000000" Z="3.841274" />
</translate >
```

Figura 3.3: Output para Mercúrio

Capítulo 4

Engine

4.1 Processar o ficheiro xml

Como referido anteriormente, é adicionado um novo atributo *time* às transformações *rotate* e *translate*, que define o tempo da translação completa, ou seja, 360 graus. Mais ainda, *point* e os seus respetivos atributos *X, Y e Z*, são adicionados como representação dos pontos que constroem a curva *Catmull-Rom*.

```
<translate time="3652563.78">
	<point X="15.209824" Y="0.000000" Z="0.000000" />
	<point X="10.573764" Y="0.000000" Z="-10.573765" />
	<point X="0.000000" Y="0.000000" Z="-14.709829" />
	<point X="-10.573764" Y="0.000000" Z="-10.573765" />
	<point X="-15.209824" Y="0.000000" Z="0.000000" />
	<point X="-10.573764" Y="0.000000" Z="10.573765" />
	<point X="0.000000" Y="0.000000" Z="14.709829" />
	<point X="10.573764" Y="0.000000" Z="10.573765" />
</translate >
<rotate axisX="0.0" axisY="1.0" axisZ="0.0" time="240000" />
<scale X="0.40" Y="0.40" Z="0.40" />
<models>
	<model file="sphere.3d" />
</models>
```

Figura 4.1: Exemplo de grupo xml a processar

Recorrendo mais uma vez ao *tinyXML*, adiciona-se ao código utilizado o reconhecimento das novas funcionalidades da engine. Assim alterando algumas funções nas quais são definidas o comportamento do programa.

4.2 Transformações

Com base no desenvolvimento das estruturas desenvolvidas para solucionar a fase anterior do projeto, surge a necessidade de armazenar outro tipo de informação para calcular as transformações. Assim como proposta de resolução, adiciona-se duas variáveis *time* e *pontos*, para armazenar o tempo e para armazenar os pontos, respetivamente.

```
struct transformacao_3d {
    float var[4][4];
    float time;
    std::vector<float> * pontos;
};
```

Figura 4.2: Estrutura *transformações_3d*

Para diferenciar as transformações estáticas ou dinâmicas a variável *time* assume o valor igual a zero ou superior a zero, respetivamente. Como tal o calculo destas novas funcionalidades tiveram como auxilio novas funções para garantir o seu funcionamento.

4.2.1 Rotation

Na transformação de *rotate* passa-se a guardar os valores do vetor de rotação na matriz e o tempo em *time*. Relativamente á ultima implementação do calculo da matriz de rotação é a atribuição do ângulo, que é feito com a expressão representada abaixo.

```
angle = to_radial(glutGet(GLUT_ELAPSED_TIME) * 360.f / t->time);
```

4.2.2 Translate

Na transformação de translação, guarda-se o tempo em *time* e os pontos na variável *pontos*. Para o calculo da matriz de transformação é usada a função *calc_catmull*, que vai de encontro á resolução do Guião 8.

4.3 VBOS e Arquitetura

Esta etapa do trabalho criou novas dificuldades, assim exigindo uma nova modelação das estruturas. As dependências entre grupos e subgrupos também tem um impacto importante nas decisões apresentadas a seguir.

4.3.1 Group

Com base nos exemplos fornecidos no enunciado do trabalho, reconhece-se que as variáveis necessárias são três vetores que guardam por ordem os modelos, sub-grupos e transformações. Assim podemos ter números arbitrários destas variáveis, garantindo que as transformações são aplicadas pela ordem correta todos os seus elementos. A estrutura proposta está representada abaixo.

```
struct group {
    std::vector<MODEL> * models;
    std::vector<GROUP> * sub_group;
    std::vector<TRANSFORMACAO> * transforms;
};
```

Figura 4.3: Estrutura *group*

4.3.2 Model

Para a modelação dos modelos a construir por triângulos em VBO. É necessário guardar os pontos para a construção dos triângulos inicialmente num vetor que é designado por *pontos*. Como também uma variável para o buffer do glut, nomeado *vertexBufffer*.

```
struct model {
    std::vector<float> * pontos;
    GLuint vertexBuffer[1];
};
```

Figura 4.4: Estrutura *model*

Capítulo 5

Teste do Sistema Solar

5.1 Manual

Com vista a testar o funcionamento da engine com o xml criado para a representação do sistema solar decidimos gerar uma esfera recorrendo ao generator com raio 1 e 10 slices e stacks, bem como o ficheiro teapotBezier.3d com o ficheiro patch fornecido.

```
$ ./generator sphere 1 10 10 sphere.3d
$ ./generator bezierPatch teapot.patch 10 teapotBezier.3d
```

Criamos uma pasta que contém o ficheiro xml ,a *sphere.3d* e o *teapotBezier.3d*. Para correr temos apenas de entrar na pasta.

```
$ cd solarSystem
```

Compilar a engine na mesma com os seguintes comandos.

```
$ cmake ../engine/
$ make
```

Agora basta passar como argumento o ficheiro xml do sistema solar.

```
$ ./engine SolarSystem.xml
```

5.2 Imagens do Sistema Solar

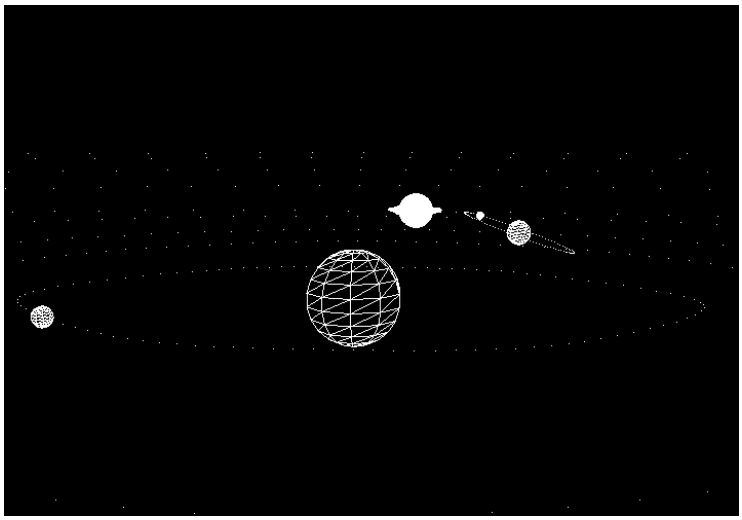

Figura 5.1: Sistema solar de perto

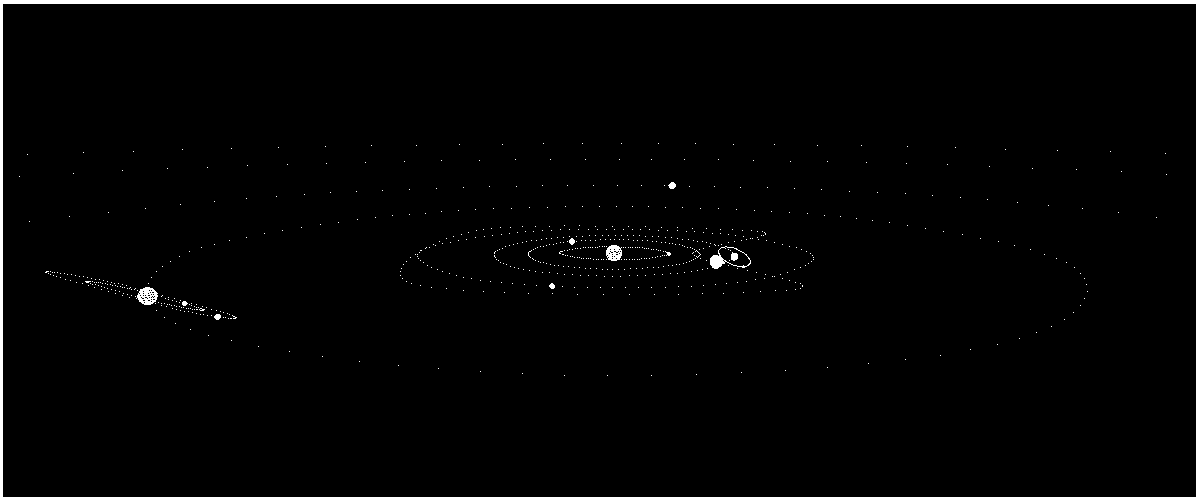

Figura 5.2: Sistema solar de longe

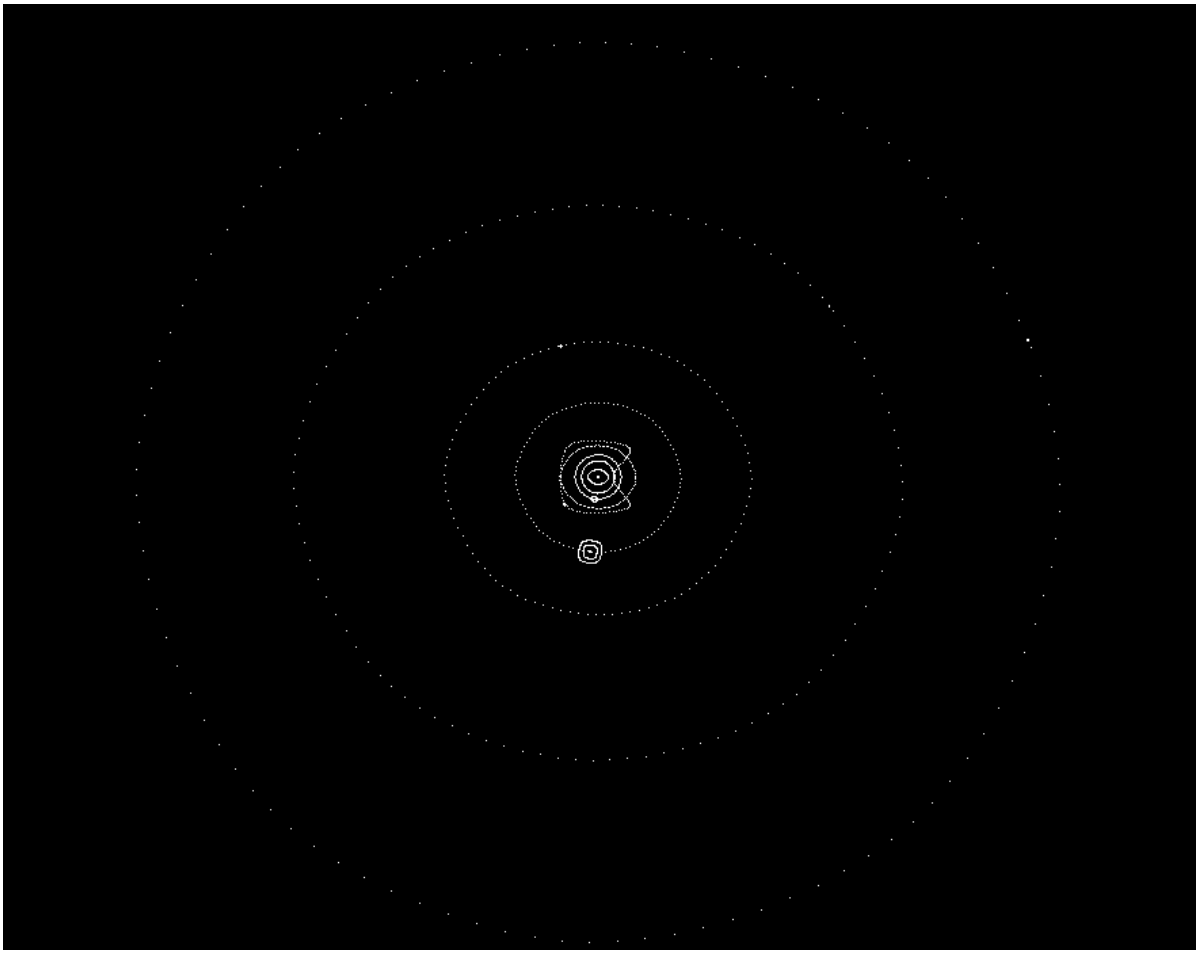

Figura 5.3: Sistema solar completo

Capítulo 6

Conclusão e Análise de Resultados

Nesta fase aprendemos a analisar Belzier Patches e a transforma-los em modelos que a nossa engine possa ler. Melhoramos também a nossa capacidade de desenhar usando VBO's. Compreendemos como funcionam as catmull curves e como aplicar transformações com tempo e desenhar as trajetória com os pontos definidos na mesma bem como as rotações com tempo. Pensamos que conseguimos satisfazer os requisitos com os resultados obtidos.