Computação Gráfica

Fase 1
Primitivas Gráficas

LEI - 2022/2023

Grupo 31

Hugo Novais
A96267

Rui Silva
A97133

Telmo Oliveira
A97018

Universidade do Minho

Índice

1 Introdução

Nesta terceira fase tivemos de realizar certas alterações tanto ao nível do generator como da engine.

Numa primeira fase, era necessário acrescentar um novo tipo de modelo baseado em Bezier patches. De seguida, seria necessário expandir os elementos de translação e rotação da engine, de maneira a permitir a criação de animações que vão mudando ao longo do tempo. Estas transformações animadas serão implementadas com recurso às curvas de Catmull-Rom.

Além disso nesta fase os diferentes modelos são desenhados utilizando VBOs, contrariamente às anteriores em que eram desenhados de modo imediato.

Como último objetivo e de maneira a mostrar o correto funcionamento das novas implementações desenvolvidas nesta fase, expandimos o modelo do Sistema Solar desenvolvido na fase anterior, de maneira que os planetas apresentem um movimento de rotação em torno do sol e em torno de si mesmos.

2 Engine

2.1 VBOs

Apesar de ser o último ponto a nos ser pedido nesta fase, decidimos que seria preferível começar pela implementação dos VBOs.

Começamos por criar duas variáveis globais. Estas são dois *maps*, um para os vértices e outro para o número de vértices. Foi decidida esta implementação para evitar guardar inúmeras vezes o mesmo ficheiro, economizando assim espaço no programa.

Criamos também uma função *prepareData* que é chamada aquando do início do programa. Esta trata-se de uma função recursiva e usa a implementação de VBOs dado nas aulas práticas. Para cada modelo é usado o nome do mesmo que será usado como chave para os *maps*. Depois, retiramos os vértices e o número de vértices do respetivo modelo para serem escritos nos *buffers*.

Com esta nova implementação, necessitamos de atualizar a função que trata desenhar os modelos. Usando novamente as funções fornecidas nas aulas, esta função agora utiliza VBOs para desenhar os mesmos.

Desta forma, conseguimos retirar peso ao CPU pois estes usam o GPU do computador, o que melhora a performance e o uso de memória do mesmo.

2.2 Curvas de Catmull-Rom

Para a criação de animações, foi nos pedido que estas fossem feitas com recurso às curvas de Catmull-Rom. Para isso, no ficheiro XML, dentro das transformações o *rotate* pode ter um novo valor de tempo que corresponde ao tempo que o objeto demora a fazer uma rotação em relação a um eixo específico. Já o *translate* pode conter também esta variável de tempo juntamente com um *align*. Assim, pontos podem ser adicionados a este que correspondem a pontos que o objeto terá que atravessar num tempo dado.

```
<group>
    <transform>
        <translate time = "10" align="true">
            <point x = "0" y = "0" z = "4" />
            <point x = "4" y = "0" z = "0" />
            <point x = "0" y = "0" z = "-4" />
            <point x = "-4" y = "10" z = "0" />
        </translate>
        <rotate time="5" x="1" y="0" z="0" />
        <scale x="0.5" y="0.5" z="0.5" />
    </transform>
    <models>
        <model file="bezier_10.3d" />
    </models>
</group>
```

Figura 1: Trecho do ficheiro XML

Para começar, começamos por aceitar estas novas variáveis dentro da respetiva função que se encarrega de fazer *parsing* ao ficheiro XML.

Na *struct* das transformações acrescentamos três novas variáveis. Um *float* para o valor de tempo, um booleano para o *align* e um vetor de *floats* que corresponde aos pontos fornecidos que o objeto poderá ter que atravessar.

De seguida criamos duas novas funções: uma para lidar com as rotações e outra para lidar com as translações.

Primeiramente fizemos a de rotação devido ao seu nível de simplicidade. Se o tempo recebido no ficheiro for nulo, a rotação é efetuada do mesmo modo que na fase anterior. Se não for calculamos uma variável *time* usando a função

$glutGet(GLUT\_ELAPSED\_TIME)$. Com este valor calculamos de seguida o ângulo de rotação usando o tempo recebido do ficheiro.

$$angle = time/fileTime * 360$$

Como na rotação, também na translação começamos por ver se o tempo é nulo. Se for, então a translação é feita de igual modo. Não sendo, calculamos o valor de uma variável t da seguinte forma:

$$t = glutGet(GLUT_ELAPSED_TIME)/(transform.time * 1000)$$

Em seguida usamos a função *fmod* para garantirmos que o valor de t fica entre 0 e o tempo recebido do ficheiro. Assim, garantimos que a animação se vai repetir depois de passar o valor de tempo.

Para efetuarmos a animação, usamos, na sua maioria, as funções fornecidas pelos docentes nas aulas práticas. A alteração que teve que ser efetuada foi nos pontos fornecidos para o cálculo do *Catmull-Rom point*. Aqui usamos os valores que agora se encontram na *struct*. Desta forma, cada objeto terá o seu vetor de pontos.

Para finalizar, com o auxílio dos diapositivos sobre as curvas Catmull-Rom, efetuamos as respetivas funções e, caso a variável *align* estivesse como *true*, efetuamos as funções necessárias para que o objeto se alinhe com a respetiva curva.

3 Generator

3.1 Bezier Patches

No que diz respeito ao *generator*, foi nos proposta a implementação de um novo tipo de modelo baseado nos *bezier patches*. Para este, aceitamos como parâmetros o ficheiro de input, o valor de *tesselation*, e o ficheiro de output.

Começamos por ler o ficheiro de input. Nesta fase, colocamos em uma lista denominada *patches* os valores dos *patches* dados pelo ficheiro. Fazemos o mesmos para os pontos de controlo, colocando todos numa lista denominada *controlPoints*.

De seguida entramos num ciclo para cada *patch* na lista de *patches*. Numa lista *arr*, adicionamos todos os *controlPoints[p]* onde p representa todos os valores dentro do *patch*.

Ainda dentro do ciclo, começamos dois novos ciclos (i,j = 0) para o valor de *tesselation* dado pelo utilizador. De começo calculamos as variáveis u e v onde $u = i * delta$ e $v = j * delta$. O valor de *delta* é dado pela seguinte expressão:

$$delta = 1/tesselation$$

Com estes valores, calculamos os valores dos quatro pontos que formam os dois triângulos que procuramos com uma função auxiliar. A cada ponto são dados diferentes valores, sendo estes:

- A - u, v
- B - u, v + delta
- C - u + delta, v
- D - u + delta, v + delta

Para respeitar a regra da mão direita, colocamos estes pontos na lista *triangles* na seguinte ordem: BCA BDC.

Para o cálculo dos pontos, como dito anteriormente criamos uma função auxiliar. Nesta começamos por criar uma matriz m com os valores dos *Bezier patches* dado nas aulas teóricas. Em seguida entramos num ciclo repetido três vezes, uma vez por cada coordenada(x, y, z). Aqui calculamos uma matriz p com os valores dados pela lista *arr*. Também criamos matrizes para os valores de u e v (U e V, respetivamente). Assim, multiplicamos as matrizes da seguinte forma:

$$pos[i] = (((U * m) * P) * m) * V$$

Esta posição pode ser descrita pela seguinte fórmula:

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

Figura 2: Trecho do ficheiro XML

Sendo M e M^T:

$$\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Depois de calculados os três valores do ponto, este é retornado.

Por fim, escrevemos no ficheiro de output a lista de valores presentes no ficheiro *triangles*.

4 Demonstração - Sistema Solar

Tal como dito anteriormente, de maneira a colocarmos em prática as novas funcionalidades da *engine* e do *generator*, desenvolvidas nesta fase, tivemos como objetivo acrescentar ao modelo do sistema solar, relativo à fase 2 do projeto, movimento aos planetas, isto é, rotação em torno do seu próprio eixo e rotação em torno do sol, ou seja, a implementação de uma órbita para cada planeta.

É importante notar que as translações alteram a origem do referencial para os diferentes objetos do modelo. Assim, de maneira a que as rotações não influenciem a trajetória dos planetas à volta do sol estas são realizadas após as translações. Em relação às luas efetuamos primeiro a rotação e de seguida a translação necessária.Os pontos necessários para as translações dos planetas foram gerados através de um programa em *python*,no qual é utilizada a função das elipses,que fornece o número de pontos que desejarmos.

Além das órbitas e das diferentes rotações, decidimos acrescentar também um cometa, o qual possui uma forma gerada a partir das curvas de Bezier e este fica também a orbitar o sol. E, assemelhando-se à realidade, a órbita deste cometa é elíptica e segue uma curva de Catmull-Rom.

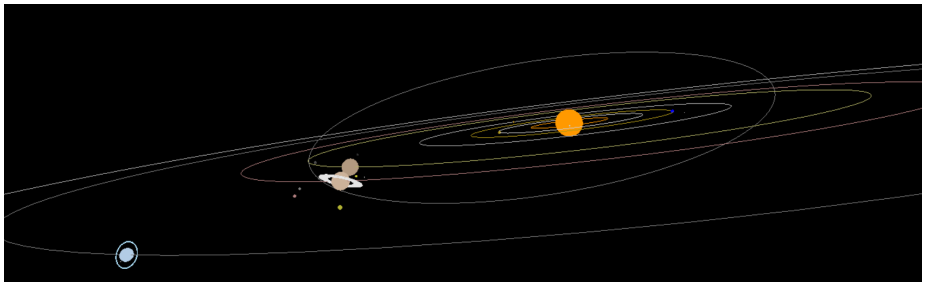

Figura 3: Sistema Solar com planetas delimitados por uma órbita.

5 Shortcuts

De forma a visualizarmos melhor o sistema solar, temos algumas teclas do teclado que nos permitem mudar a cena.

Primeiramente, para mudar a câmara de posição. Para esta se movimentar no eixo do X, usamos W e S; no eixo do Z, usamos A e D; e no eixo do Y, usamos UP_ARROW e DOWN_ARROW. Também usamos o Q e E para rodar a câmara.

Também usamos a tecla J para desenhar ou retirar as curvas de Catmull-Rom; a tecla K para desenhar ou retirar os eixos do XYZ e a tecla L para desenhar por completo ou apenas as linhas das diferentes formas.

6 Conclusão

Em suma, o desenvolvimento desta terceira fase permitiu-nos colocar em prática os conhecimentos adquiridos nas aulas, dos quais transformções de modelos 3D, VBOs, patches de Bezier e ainda curvas de Catmull-Rom. Além destas novas implementações foi ainda necessário reformular parte do código das fases anteriores,como por exemplo *structs* e a leitura do XML,no entanto estas alterações não trouxeram grandes dificuldades, uma vez que o código das últimas fases se encontra bem estruturado.

Assim, conseguimos aplicar aos planetas presentes no modelo desenvolvido do sistema solar uma órbita em torno do sol, como também, um movimento de rotação em torno do seu próprio eixo, tornando assim o nosso modelo cada vez mais fiel à realidade.