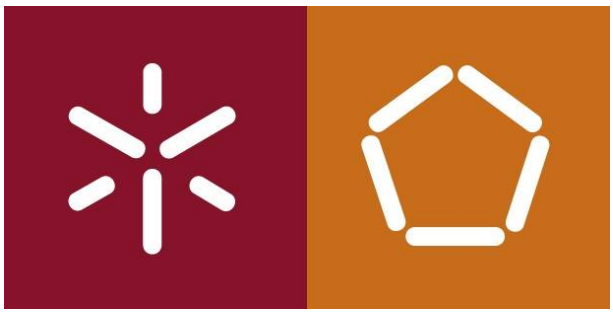

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Computação Gráfica

Sistema Solar

Joan Rodriguez (a89980)
Mário Santos (a70697)
Pedro Costa (a85700)
Rui Azevedo (a80789)
4 de Maio de 2020

Conteúdo

Lista de Figuras

1 Introdução

A terceira fase deste trabalho consiste em um modelo dinâmico do sistema solar, ao contrário da fase anterior que era estático, com base em animações relativas aos movimentos de translação e rotação.

Nesta fase surge uma nova primitiva que é desenhada através de *Bazier Patches*, fornecidos através de um ficheiro que contém toda a informação relativa ao desenho da primitiva através desta técnica de desenho. Para além disso, é pretendido que as primitivas sejam desenhadas através de *VBO's*, embora que esta funcionalidade já está presente no trabalho desde a primeira fase, mas irá ser explicada novamente neste documento. Por último, o movimento de translação, para ser dinâmico, passou a ser definido através de curvas cúbicas de *Catmull-Rom* dado também o número de segundos que demora a percorrer a curva. O movimento de rotação, passou a ser calculado através do tempo, em oposição a um ângulo.

Este relatório descreve em detalhe como foram implementadas cada uma das funcionalidades/alterações pedidas, mostrando numa primeira fase as alterações do programa *Generator*, de seguida as alterações feitas ao *Engine* e por fim o *output* obtido.

2 Generator

Como já foi referido nas fases anteriores, o programa *Generator* é responsável por gerar pontos para o desenho das primitivas. Nesta fase, é agora possível gerar pontos de uma primitiva recorrendo aos *Bezier Patches* para que uma determinada primitva seja desenhada com maior detalhe e qualidade. Toda a informação relativa aos *pacthes* estão contidas num ficheiro com uma estrutura bem definida, apresentada de seguida.

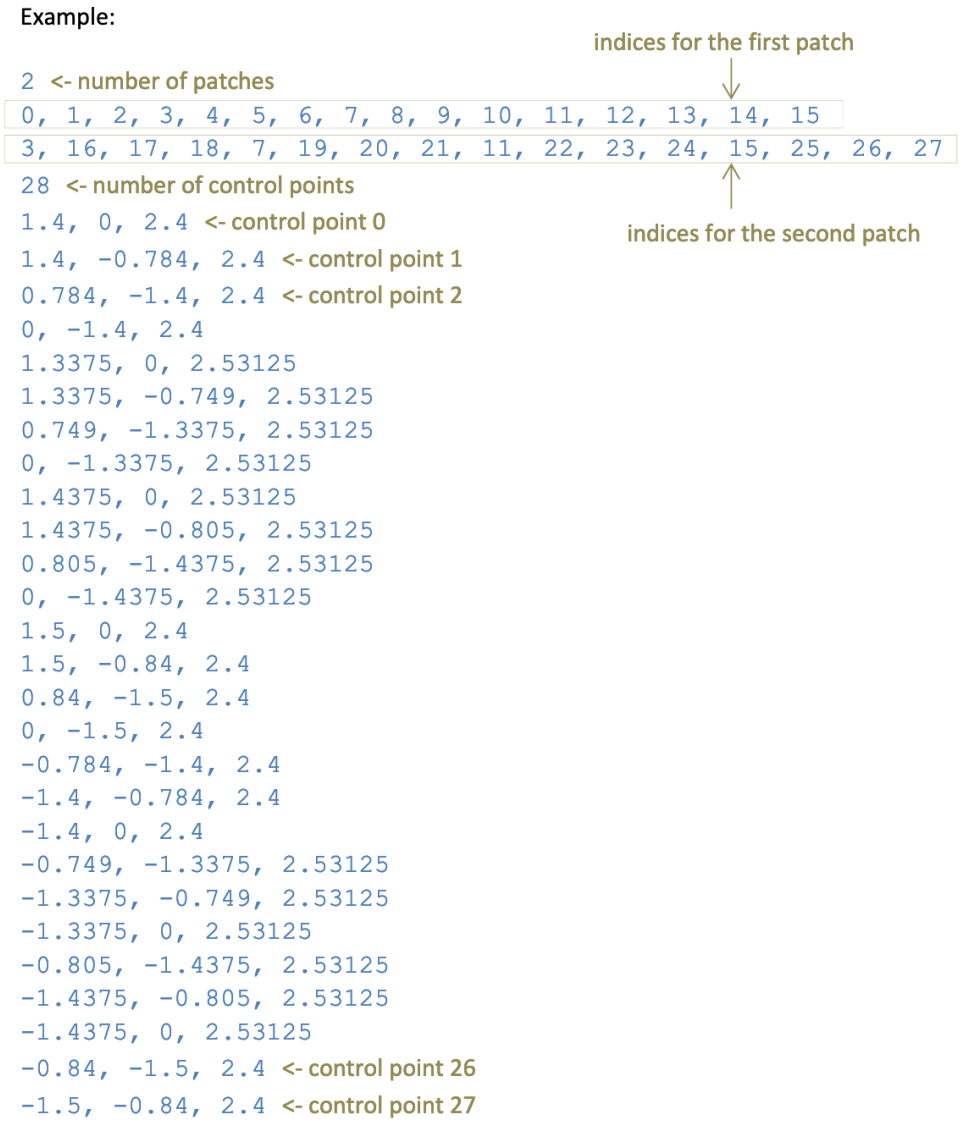

Figura 1: Formato dos *Bezier Pacthes*

2.1 Arquitectura

Para acrescentar esta nova funcionalidade de gerar pontos tendo em base a técnica de desenho anteriormente mencionada foi criada uma nova classe responsável por gerar os pontos correspondentes aos *Bezier Pacthes*. Esta classe contém três funções na sua totalidade que permitem a geração dos pontos:

- bezierCurve : função usada para calcular uma curva de *Bezier*
- calculateVertices : função auxiliar usada na função *bezierCurve*
- readBezierFile : lê o ficheiro que contém os dados dos *pacthes*, aplica a função bezierCurve, e escreve para um ficheiro os pontos gerados.

2.2 Ring

Foi introduzida uma nova primitiva designada por *ring* para desenhar o anel de saturno. Para o cálculo da figura usaram-se coordenadas esféricas devido à sua estrutura. Para o desenho de um anel precisamos de duas circunferências ligadas por triângulos. Para o cálculo do anel são precisos os seguintes parâmetros:

- *slices* : número de lados do anel. Este número, quanto maior, mais se aproxima de uma circunferência.
- *radius* : raio da circunferência mais exterior.
- *ring* : distância entre as duas circunferências.

Na imagem abaixo, é fácil visualizar a estrutura do anel com os seguintes parâmetros: $slices = 4$, $radius = 4$ e $ring = 2$. Os triângulos são formados com a seguinte ordem dos vértices, ABC e ACD.

Algorithm 1 ring

```
1: function DRAWRING(slices,radius,ring)
2:     α ← 2π/slices
3:     for i ← 0 to slices do
4:         d_x ← alpha * i
5:         A = ((radius − ring) * sin(d_x), 0, (radius − ring) * cos(d_x))
6:         B = (radius * sin(d_x), 0, radius * cos(d_x))
7:         C = (radius * sin(d_x + α), 0, radius * cos(d_x + α))
8:         D = ((radius − ring) * sin(d_x + α), 0, (radius − ring) * cos(d_x + α))
9:
```

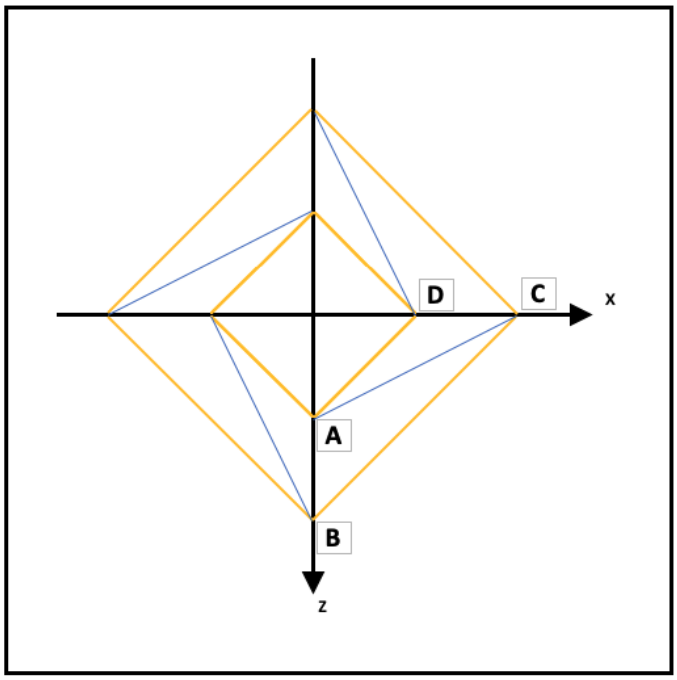

Figura 2: Ring

2.3 Bezier Patches

Nesta secção será apresentada a lógica na geração de pontos através do ficheiro de *pacthes* fornecido.

O algoritmo usado para o cálculo dos pontos segue a seguinte equação de matrizes:

$$B(u,v) = U * M * P * M^T * V$$

As matrizes U e V, referentes às variáveis u e v são:

$$U = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}$$

$$V = \begin{bmatrix} v^3 & v^2 & v & 1 \end{bmatrix}$$

A matriz M, igual à sua transposta dada a simetria da matriz, corresponde à matriz de *Bezier* e é formada da seguinte maneira:

$$M = M^T = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Por fim, a matriz P, que contém os 16 pontos de controlo dos *patches* pode ser representada da seguinte maneira:

$$M = M^T = \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}$$

Uma vez conhecida a equação a usar na geração de pontos, resta aplicar esta lógica na criação do algoritmo correspondente. De seguida, é apresentado o pseudo-código da função que vai iterar todos os *pacthes* e fazer o cálculo dos pontos. O algoritmo vai desenhar cada *patch* por linha, daí existirem três ciclos, um para iterar os *patches*, outro para iterar os $u's$ (linhas) e outro para iterar os $v's$ (colunas).

Algorithm 2 mainLoop

```
1:  function MAINLOOP(tessellation, nPatches, pacthes, controlPoints)
2:      interval = 1/tessellation
3:      for i ← 0 to nPatches do
4:          u ← −interval
5:          v ← 0.0
6:          for j ← 0 to tessellation do
7:              u ← u + interval
8:              for k ← 0 to tessellation do
9:                  bezierCurve(patches[i], controlPoints, u, v, interval)
10:                 v ← v + interval
11:
```

As funções que fazem o cálculo dos pontos em si são a *bezierCurve* e a *calculateVertices*.

Algorithm 3 bezierCurve

```
1:  function BEZIERCURVE(patches, controlPoints, u, v, interval)
2:      for i ← 0 to nPatches do
3:          p1 ← calculateVertices(patch, controlPoints, u, v)
4:          p2 ← calculateVertices(patch, controlPoints, u, v + interval)
5:          p3 ← calculateVertices(patch, controlPoints, u + interval, v)
6:          p4 ← calculateVertices(patch, controlPoints, u + interval, v + interval)
7:
8:          triangle1 ← (p1, p4, p2)
9:          triangle2 ← (p4, p1, p3)
10:
```

Na função abaixo apresentada, P_x, P_y e P_z são as matrizes que contém as coordenadas x, y e z, respectivamente, dos pontos de controlo, MV é a matriz resultante da multiplicação da matriz de *Bezier* e a matriz com os valores de v; p_x, p_y e p_z são as matrizes resultantes da multiplicação das matrizes P e M; m_x, m_y e m_z são as matrizes resultantes da multiplicação da matriz de *Bezier* e as matrizes p. Por fim, x, y e z são as coordenadas dos pontos calculados. Foi usada também uma função auxiliar, $mulMatrixVector$ que multiplica uma matriz por um vector, a fim de ter um código mais legível.

Algorithm 4 calculateVertices

function CALCULATEVERTICES(patch, controlPoints, u, v)

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

$$U = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}$$

$$V = \begin{bmatrix} v^3 & v^2 & v & 1 \end{bmatrix}$$

$mulMatrixVector(M, V, MV)$
$mulMatrixVector(P_x, MV, p_x)$
$mulMatrixVector(P_y, MV, p_y)$
$mulMatrixVector(P_z, MV, p_z)$

$mulMatrixVector(M, p_x, m_x)$
$mulMatrixVector(M, p_y, m_y)$
$mulMatrixVector(M, p_z, m_z)$

$x \leftarrow U[0] * m_x[0] + U[1] * m_x[1] + U[2] * m_x[2] + U[3] * m_x[3]$
$y \leftarrow U[0] * m_y[0] + U[1] * m_y[1] + U[2] * m_y[2] + U[3] * m_y[3]$
$z \leftarrow U[0] * m_z[0] + U[1] * m_z[1] + U[2] * m_z[2] + U[3] * m_z[3]$
=0

3 Engine

A parte do sistema relativa ao motor do sistema solar foi alterada de maneira a que o movimento de translação possa ser representado através de um curva de *Catmull-Rom*. O movimento de rotação também foi alterado, sendo que já não está dependente de um ângulo mas sim de um tempo, sendo este tempo o número de segundos necessárias para fazer uma rotação

de 360°. Para além do facto de os planetas poderem girar à volta do sol, foi implementado também o movimento de rotação à volta do próprio eixo com uma simples alteração no modelo *XML* e no sistema de *parsing*.

Para além destas alterações, e, embora já tenha sido apresentado na fase anterior, irá ser mostrado novamente o algoritmo usado para o desenho das cenas através de *VBO's*.

3.1 VBO's

Os *VBO's* são usados para aumentar a eficiência uma vez que os vértices usados para desenhar as cenas residem no dispositivo de vídeo do computador, em vez de residirem na memória do sistema e, por isso, são renderizados directamente pelo dispositivo de vídeo. Esta funcionalidade é fornecida directamente pelo *OpenGL*.

A função abaixo é a responsável por desenhar os *VBO's*. O vector *allPrimitives* contém os vértices de uma primitiva.

Algorithm 5 drawVBO

```
1:  function DRAWVBO
2:      glGenBuffers(allPrimitives.size(), buffers)
3:      for i ← 0 to allPrimitives.size() do
4:          pVertices ← allPrimitives.at(i)− > getVertices()
5:          for j ← 0 to pVertices.size() do
6:              vertices.push_back(pVertices.at(j)− > getX())
7:              vertices.push_back(pVertices.at(j)− > getY())
8:              vertices.push_back(pVertices.at(j)− > getZ())
9:              − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − −
10:             arr ← &vertices[0]
11:             glBindBuffer(GL_ARRAY_BUFFER, buffers[i])
12:             glBufferData(GL_ARRAY_BUFFER, 3 ∗ pVertices.size() ∗
    sizeof(float), &arr[0], GL_STATIC_DRAW)
13:             glVertexPointer(3, GL_FLOAT, 0, 0)
14:
15:
```

3.2 XML

O *parsing* do documento *XML* foi ligeiramente alterado para que seja possível haver rotação à volta do próprio eixo dos planetas. A alteração feita diz respeito aos grupos do ficheiro, sendo possível agora criar um grupo com transformações mas sem modelos. Os grupos aninhados dentro de um grupo deste tipo herdam as transformações do grupo pai. Isto torna

possível o facto de um planeta conseguir rodar em volta do seu próprio eixo e também à volta do sol.

3.3 Rotação

A classe de rotações foi alterada de modo a implementar a funcionalidade de rotação dinâmica. Para isto, foi acrescentada uma variável correspondente ao tempo de rotação. Esta variável, tem como base indicar o número de segundos necessários para completar uma rotação de 360°.

Para este cálculo foi usada a função *glutGet(GLUT_ELAPSED_TIME)*, com o objectivo de medir o tempo desde o início da execução da função *glutInit*. Uma vez que este tempo é estritamente crescente à medida que o programa é executado, este valor foi limitado pela variável *time* contida no ficheiro *XML*, mas em milissegundos. De seguida, este valor é usado para calcular o ângulo de rotação.

Finalmente, pode-se apresentar a fórmula desta cálculo da seguinte maneira:

$$r = elapsedTime\%(time * 1000)$$
$$gr = (r * 360)/(time * 1000)$$

O ângulo *gr* é somado ao valor inicial do ângulo contido no ficheiro *XML* e aplicado à transformação de rotação através da função do *OpenGL*, *glRotatef*.

No ficheiro de *XML*, a *tag* referente ao movimento de rotação é o seguinte:

<rotate angle=α **axisY=AXIS time=TIME/>**

3.4 Translação

Um dos objectivos desta fase do trabalho é a animação do movimento de translação. Este movimento tem duas variantes: uma vez dado um tempo no ficheiro *XML*, é lido um conjunto de pontos correspondentes aos pontos necessários para criar uma curva de *Catmull-Rom* e, se o tempo não for definido no ficheiro, a translação tem o comportamento igual ao das fase anteriores.

Para atingir este objectivo, a classe referente às translações foi alterada de maneira a implementar esta nova funcionalidade. Foram acrescentadas, à classe, mais duas variáveis, uma é um *float* correspondente ao tempo de translação e a outra é um vector de pontos, correspondente ao conjunto de pontos que definem uma curva de *Catmull-Rom*.

O cálculo das curvas de *Catmull-Rom* têm como base os seguintes procedimentos:

$$A = M * P$$
$$pos = T * A$$
$$deriv = T' * A$$

A matriz M é referente à matriz de *Catmull-Rom* e é definida da seguinte maneira:

$$M = \begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1.0 & -2.5 & 2.0 & -0.5 \\ -0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \end{bmatrix}$$

A matriz P contém os pontos de controlo das curvas de *Catmull-Rom*. A matriz T e T' são as matrizes referentes ao parâmetro t que é passado à função de cálculo das curvas, e que indica a que distância entre dois pontos é iniciado o processo recursivo da curva.

$$T = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}$$

$$T' = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix}$$

Os procedimentos apresentados acima constituem a parte mais importante no desenho das curvas de *Catmull-Rom* e tem uma tradução praticamente directa para o código usado por isso não será necessário apresentar o seu pseudo-código.

Quanto à estrutura da *tag* de translação contida no ficheiro *XML*, tem a seguinte estrutura:

<translate time=TIME/>

4 Modelo Final

O modelo final obtido é então o modelo dinâmico do sistema solar, onde cada planeta tem a sua orbita de translação e o seu movimento de rotação. Para além disso, pode-se ver o cometa, que neste caso é representado por um *teapot*, que orbita em torno do sol.

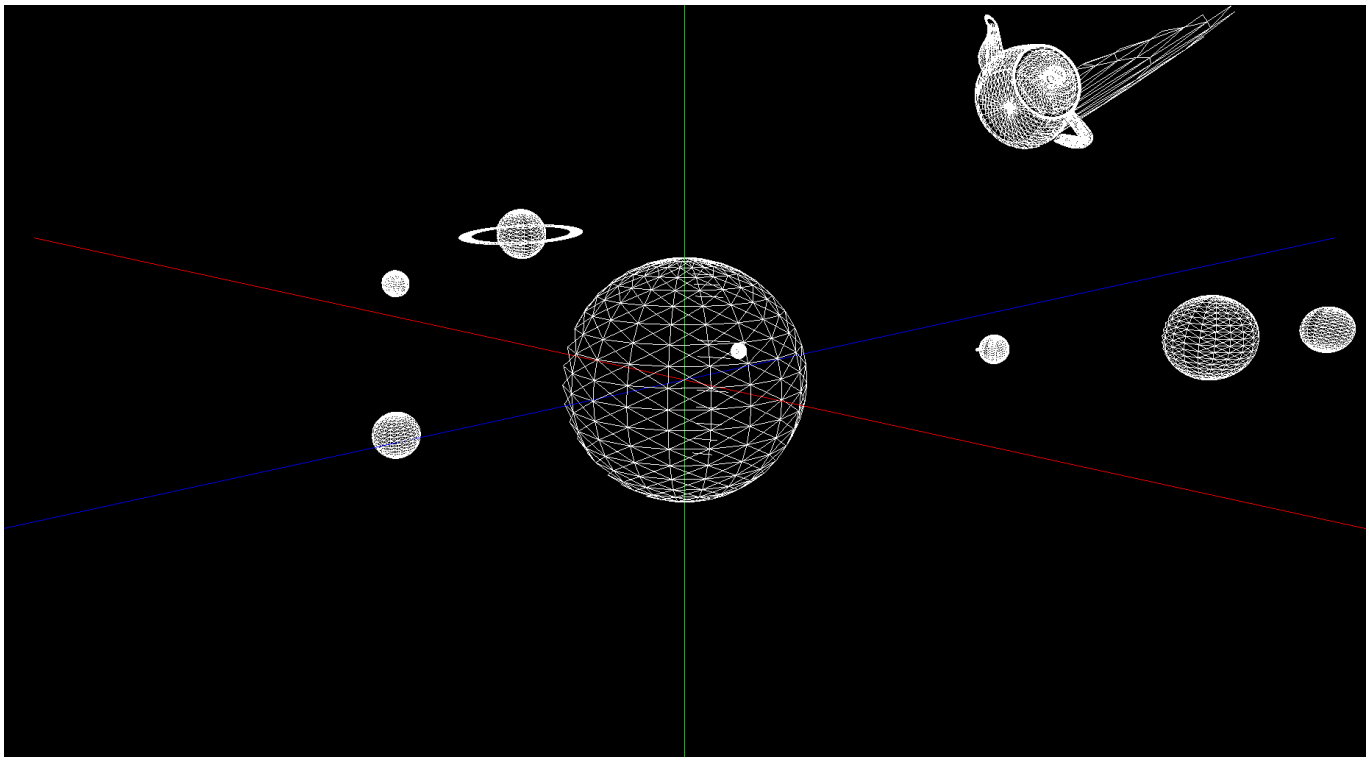

Figura 3: Ring

5 Conclusão

Foi possível explorar, nesta fase, conceitos que tornam o programa que se tem vindo a desenvolver mais interessante, pois agora tem-se um sistema solar dinâmico e com técnicas de desenho mais avançadas.

As curvas cúbicas de *Bezier* permitiram um rigor no desenho de primitivas bastante maior sendo também um processo mais complexo.

A nível das animações nos movimentos dos planetas, as curvas de *Catmull-Rom* são um método simples e eficaz para o cálculo dos pontos da translação dos planetas, e, desta maneira, os planetas ficam com movimento tornando o sistema solar bem mais interessante.

Contudo, o trabalho apresenta um erro no sistema *Engine* ao desenhar o *teapot* através dos *pacthes* de *Bezier* e que o grupo não consegui descobrir o porquê do erro. Sabe-se que o erro não está na geração de pontos e que estes estão a ser gerados correctamente pelo *Generator*.

Por fim, o recurso aos *VBO's*, já implementados na primeira fase do projecto, são um ponto bastante importante na eficiência do sistema, pois permitiram um aumento significativo nos *fps* das cenas criadas.