Relatório da fase 3

Nome: Luís Gustavo Aires Guimarães Maia
Nº Aluno: A95656
Nº Grupo: 50

1. Introdução

Nesta terceira fase, foi pedido que se atualizasse o *Generator* de modo a suportar um novo tipo de modelos baseados em *patches Bezier*. Também era pedido que se atualizasse o *Engine* de modo a desenvolver as translações e rotações e passar a utilizar *VBO's* no desenho das primitivas.

2. Decisões e abordagens

Generator

Para suportar os novos modelos, foi necessário implementar uma nova primitiva chamada ***Bezier***, que recebe como argumentos o caminho para o ficheiro *.patch* e o número de *tessellation* pretendido.
Para a leitura dos ficheiros *.patch* foi criado o ficheiro *patchReader.cpp* que preenche 2 vetores, um com as *patches* e outro com os pontos de controlo.
De seguida, são processados esses vetores de modo a calcular as coordenadas dos vértices, usando a seguinte lógica e cálculos:

Uma superfície de Bezier cúbica é uma superfície paramétrica definida por um conjunto de 16 pontos de controlo dispostos numa grelha 4×4. A posição de um ponto na superfície, para parâmetros u, v ∈ [0,1] é obtida através de uma combinação dos pontos de controlo com as funções de base de Bernstein de grau 3. As funções de Bernstein definem a influência de cada ponto de controlo na posição final do ponto na superfície. Para o grau cúbico, temos:

$$B_0(t) = (1 - t)^3$$
$$B_1(t) = 3t(1 - t)^2$$
$$B_2(t) = 3t^2(1 - t)$$
$$B_3(t) = t^3$$

Estas funções são utilizadas para ambos os parâmetros u e v.
A superfície $P(u, v) \in \Re^3$ é definida por:

$$P(u, v) = \sum_{i=0}^{3} i \sum_{j=0}^{3} j \, B_i(u) \, B_j(v) \cdot P_{ij}$$

Onde:

- $B_i(u)$ e $B_j(v)$: funções de base de *Bernstein* de grau 3
- P_{ij}: ponto de controlo na linha j, coluna i

A multiplicação $B_i(u) \cdot B_j(v)$ fornece o peso (influência) do ponto P_{ij} na posição P(u, v).

Para avaliar numericamente a superfície, discretizam-se os parâmetros u e v em intervalos uniformes. Se subdividirmos o intervalo [0,1] em n divisões, teremos n×n células (quads). Cada célula é representada por dois triângulos.
Para cada célula, calcula-se os quatro pontos da superfície correspondentes a:

$$P_{00}=P(u_0 ,v_0)$$
$$P_{10}=P(u_1, v_0)$$
$$P_{01}=P(u_0, v_1)$$
$$P_{11}=P(u_1, v_1)$$

com:

$$u_0 = i / n$$
$$u_1 = (i + 1) / n$$
$$v_0 = j / n$$
$$v_1 = (j + 1) / n$$

Para cada célula, os quatro pontos calculados são usados para formar dois triângulos que compõem a malha da superfície:

- **Triângulo 1**: P_{01}, P_{00}, P_{10}
- **Triângulo 2**: P_{01}, P_{10}, P_{11}

Engine

Para a utilização de *VBO's* foi seguida a lógica, fornecida pelos docentes:

1. Os vértices são carregados de um ficheiro *.3d* para um vetor
2. São copiados para a memória gráfica (*VBO*), libertando a *RAM*
3. São desenhados através do *VBO*.

Atualização das transformações geométricas

Para as rotações e translações poderem suportar dois tipos diferentes, foi atualizada a estrutura *Transform* de modo a que esta suporte os novos parâmetros dos ficheiros de configuração *XML*: um booleano que verifica se existe a componente *time* no *XML,* uma variável para armazenar o *time*, um booleano para verificar a existência da definição *align* e um vetor para guardar os pontos da curva cúbica de *Catmull-Rom*.

- Rotação
  Para as rotações apenas foi necessário normalizar o *time* do *GLUT* de modo a que este ficasse em segundos.

- Translação
  Para as translações, foi necessário fazer cálculos de modo a saber quais os pontos da curva cúbica de *Catmull-Rom* pela qual iria ocorrer a translação.
  A função *drawGroup* percorre a lista de transformações. Quando encontra uma translação (*TRANSLATE*) com tempo (*hasTime*) e pontos de curva (*curvePoints*), ela calcula a **posição atual do objeto ao longo da curva** usando o tempo atual da aplicação. Esse ponto é então usado com *glTranslatef* para mover o objeto.
  A ideia é: o tempo total de percurso pela curva é dividido em **vários segmentos**, e cada segmento tem o seu próprio pequeno tempo.

Se temos n pontos de controlo, então temos n segmentos (assumindo curva em *loop*). Cada segmento é percorrido no mesmo intervalo de tempo.
O parâmetro **t ∈ [0, 1)** representa **o progresso atual total ao longo da curva**.
Exemplo:

- t = 0.0 → início da curva
- t = 0.5 → meio do percurso
- t = 0.999 → quase no final

Esse t é proporcional ao tempo real da animação. É dividido em partes para saber em **qual segmento** da curva estamos.

A curva de *Catmull-Rom* usa **4 pontos consecutivos**:

- P_0: ponto antes do início do segmento
- P_1: início do segmento
- P_2: fim do segmento
- P_3: ponto após o fim

A interpolação ocorre **apenas entre P_1 e P_2**.
O objetivo é gerar uma função C(t) que retorna a posição ao longo do caminho entre P_1 e P_2, usando os quatro pontos para obter um maior nível de suavidade.
A posição na curva é dada por um **polinómio cúbico vetorial**:

$$C(t)=a_0t^3 + a_1t^2 + a_2t + a_3$$

Onde os coeficientes a_i dependem dos 4 pontos P_0, P_1, P_2, P_3 e da matriz base da *Catmull-Rom*.
Matriz base da *Catmull-Rom*:

$$\begin{vmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1.0 & -2.5 & 2.0 & -0.5 \\ -0.5 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \end{vmatrix}$$

Multiplicamos essa matriz pelos pontos P_0, P_1, P_2, P_3, para cada coordenada (x, y, z), para obter os coeficientes a_0, a_1, a_2, a_3.

A derivada da curva — ou seja, a direção em que o objeto se está a mover — é dada por:

$$C'(t) = 3a_0t^2 + 2a_1t + a_2$$

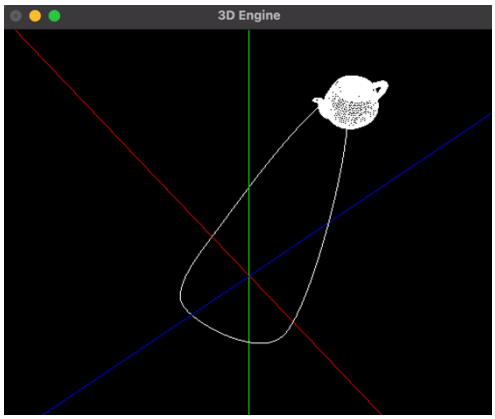
Esta derivada é usada para **quando existe a opção *align***: faz com que o objeto "aponte" na direção do movimento.

3. Testes e Sistema Solar dinâmico

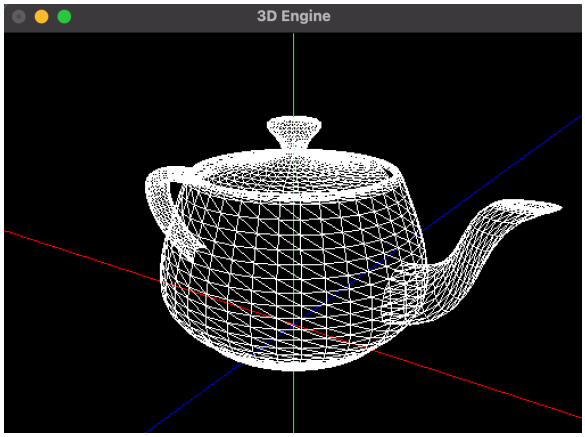
Testes

Para verificar que as novas funcionalidades estavam corretas, utilizaram-se os testes fornecidos no enunciado:

- Teste 1 – Teapot a mover-se

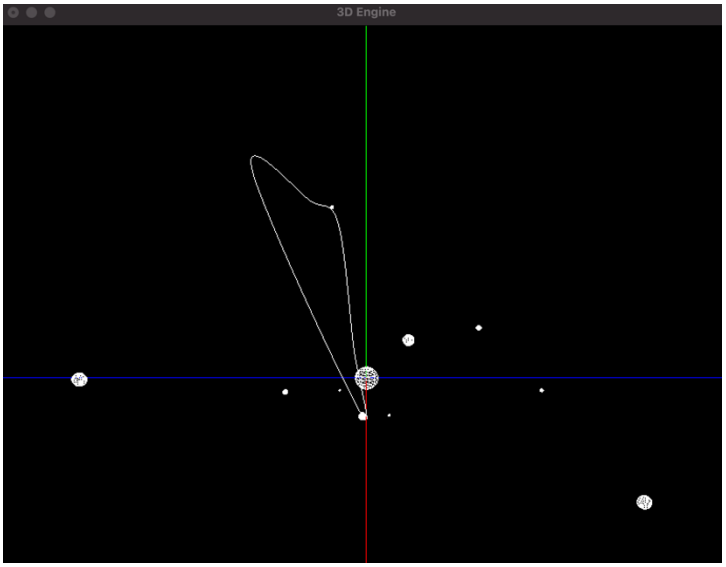


- Teste 2 – Teapot estático



Sistema Solar

Usei o modelo *XML* do Sistema Solar criado na fase anterior, adicionando o cometa cuja órbita é definida por uma curva de *Catmull-Rom* e cuja forma é criada usando os pontos de controlo do teapot mas apenas utilizando a pega da tampa.



4. Conclusões

O desenvolvimento desta fase contribuiu para três aspetos importantes: ter a possibilidade de fazer cenas animadas através da expansão das translações e rotações, gerar superfícies curvas através de ficheiros *.patch* e melhorar o desempenho do motor gráfico através da utilização de *VBO's*. Um passo futuro seria implementar índices para os *VBO's* de modo a aumentar ainda mais o desempenho. O projeto já está preparado para receber as novas funcionalidades da próxima fase, tornando cada vez mais fácil a adaptação do código para o próximo desafio.