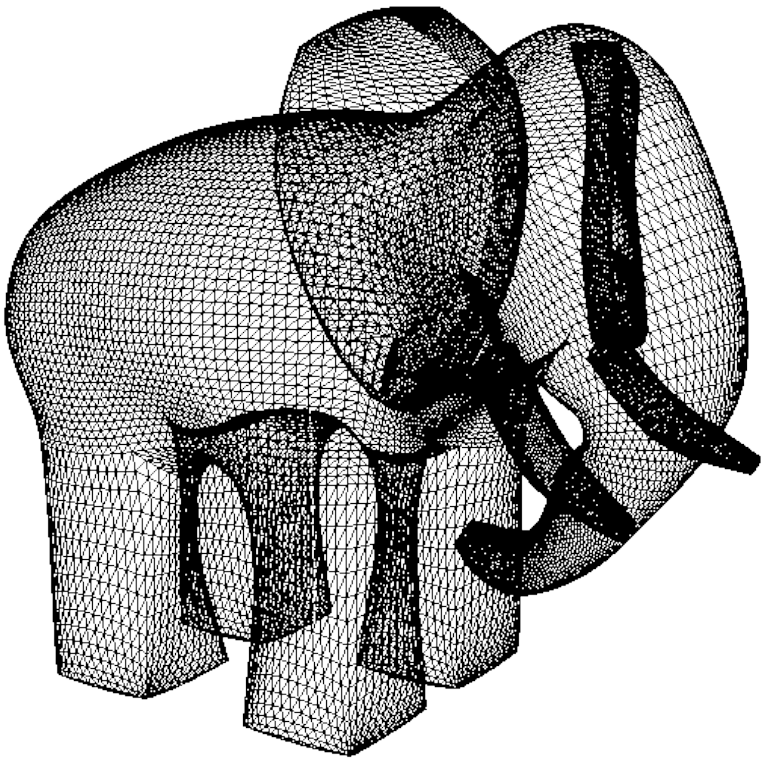

Relatório 3ª Fase | Computação Gráfica
Grupo 25 | 2024/2025

Afonso Dionísio (A104276)

João Lobo (A104356)

Rafael Seara (A104094)

Rita Camacho (A104439)

Índice

1. Introdução

O presente relatório apresenta informações relativas à **3ª Fase do Trabalho Prático** da Unidade Curricular **Computação Gráfica**, pertencente ao 2º Semestre do 3º Ano da Licenciatura em Engenharia Informática, realizada no ano letivo 2024/2025, na Universidade do Minho.

A 3ª Fase consistiu em implementar VBOs, introduzir transformações temporais - rotações e translações (*Catmull Rom*) - bem como *Bezier Patches*.

2. VBOs

Para esta fase, tornou-se necessária a utilização de VBOs de forma a otimizar a renderização da cena. Algo necessário para a sua implementação é a informação de índices dos modelos - o nosso grupo decidiu calculá-la antes de rodar o *engine* (no processo de geração), o que obrigou a mudar o formato do `.3d`.

2.1. Reformulação do formato dos modelos

Sendo assim, para o novo formato de ficheiro dos modelos gerados, decidimos utilizar o dos `.obj`, onde cada vértice é definido através de `v <x> <y> <z>`, a ordem dos mesmos está associada aos seus índices (começando em 1), e cada triângulo ou "face" é definido através de `f <índice vértice 1> <índice vértice 2> <índice vértice 3>`.

```
v 0.5 0 0
v 0 0.5 0
v 0 0 0.5
f 1 2 3
```

Listagem 1: Exemplo do novo ficheiro `.3d`

Este novo formato de modelos não contém vértices repetidos, pois os triângulos referem índices de vértices, obrigando a que fossem realizadas alterações na geração de modelos.

2.2. Geração para o novo formato

De forma a manter o código previamente escrito para a geração de modelos, construiu-se um sistema de indexação de vértices, denominado `VertexIndexer`. Este sistema permite armazenar apenas os vértices únicos e, em vez de repetir posições iguais, associa a cada vértice um índice.

O `VertexIndexer` funciona da seguinte forma:

- Sempre que é gerado um novo vértice, verifica se este já existe no conjunto de vértices previamente gerados.
- Se já existir, reutiliza-se o seu índice.
- Caso contrário, adiciona-se o vértice à lista e atribui-se-lhe um novo índice.

Internamente, este processo é realizado utilizando um `unordered_map`, permitindo procurar rapidamente vértices repetidos através de uma função de *hash* definida para pontos.

2.3. Renderização através de VBOs

Agora, quando um modelo é carregado pela primeira vez, os seus dados de vértices e composição dos triângulos (triplos de índices) são enviados para a *GPU* ficando guardados nos seus *buffers*. No momento de renderização, é feita uma chamada à GPU para renderizar o modelo diretamente com base nos dados desses *buffers* (a localização dos buffers é guardada no momento de carregamento), evitando perdas de tempo no envio de informação da memória para a GPU.

3. Transformações Temporais

Para a criação de transformações temporais, foi necessário ajustar a classe abstrata `Transformation` para o seu método `apply`, passando agora a receber como argumento o tempo atual da aplicação. Este novo argumento permite que as novas transformações utilizem o estado temporal do *engine* no cálculo da matriz a ser aplicada, algo necessário para efetuar animações.

3.1. Simulação de tempo

Antes da implementação das novas transformações temporais, foi necessário arranjar um método de calcular o tempo passado. Este é calculado em cada renderização de um novo *frame*, através da seguinte fórmula:

$$t += \Delta t$$

Onde Δt é o tempo passado desde o último *frame*. (obtido através da subtração do `current_time` deste *frame* com o do *frame* anterior) e t é o tempo passado em segundos desde a abertura do programa.

3.2. Rotações

Para a transformação de rotação temporal, o seu objetivo é animar a rotação do grupo num dado eixo durante um certo tempo. No cálculo do ângulo de rotação do grupo num dado eixo em certo tempo t, usou-se a seguinte fórmula: $\frac{\mathrm{t}*\pi*2}{\text{duração total}}$. Sendo assim, utilizamos a mesma função de rotação anterior, agora com o cálculo do ângulo baseado no tempo:

$$R\left(\frac{2*\pi*\mathrm{t}}{\text{duração total}}, x, y, z\right)$$

3.3. Translações (*Catmull Rom*)

Para a transformação de translação temporal, o seu objetivo é animar a translação do grupo numa curva de *Catmull Rom* composta por 4 ou mais pontos de controlo.
Para a interpolação da posição na curva, a trajetória calcula-se a partir dos 4 pontos de controlo mais próximos do ponto do tempo atual (t), ou seja, sempre a partir de um grupo de 4 pontos de controlo (P_0, P_1, P_2, P_3).

Para calcular o ponto atual da translação do grupo na curva, e a sua respetiva derivada (a utilização da mesma é explicada no capítulo seguinte), utilizam-se as seguintes fórmulas:

$$\mathcal{C}(P_0, P_1, P_2, P_3) = \begin{pmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1.0 & -2.5 & 2.0 & -0.5 \\ -0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$

$$\mathcal{P}(t, P_0, P_1, P_2, P_3) = \mathcal{C}(P_0, P_1, P_2, P_3)(t^3 \;\; t^2 \;\; t \;\; 1)$$

$$\mathcal{P}'(t, P_0, P_1, P_2, P_3) = \mathcal{C}(P_0, P_1, P_2, P_3)(3t^2 \;\; 2t \;\; 1 \;\; 0)$$

Tendo a posição calculada, o resultado da transformação será a matriz de translação para esse ponto.

3.3.1. Alinhamento à curva

A translação temporal entre pontos tem também o parâmetro booleano `align`, que define se o grupo deve estar alinhado com a curva em todos os momentos da transformação. Para isso, é necessário calcular uma matriz de rotação (M) através da derivada do ponto ($\mathcal{P}'$) calculada na translação.

$$\overrightarrow{X_i} = \|\mathcal{P}'(t, P_0, P_1, P_2, P_3)\|$$

$$\vec{Z}_i = \left\|\overrightarrow{X_i} \times \overrightarrow{Y_{i-1}}\right\|$$

$$\vec{Y}_i = \left\|\vec{Z}_i \times \overrightarrow{X_i}\right\|$$

$$M = \begin{pmatrix} \overrightarrow{X_i}_x & \vec{Y}_{i_x} & \vec{Z}_{i_x} & 0 \\ \overrightarrow{X_i}_y & \vec{Y}_{i_y} & \vec{Z}_{i_y} & 0 \\ \overrightarrow{X_i}_z & \vec{Y}_{i_z} & \vec{Z}_{i_z} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

i é a iteração atual.

3.3.2. Visualização da curva

De forma a permitir a visualização das curvas de *Catmull Rom*, facilitando o *debugging* e desenvolvimento da transformação, adicionou-se a possibilidade da mesma ter um campo booleano opcional `render_path`. Caso este campo seja verdadeiro, utiliza-se a funcionalidade de renderização *Line Loop* do *OpenGL* (permite a renderização de uma linha contínua que liga o seu início ao fim), desenhando um número alto de pontos, os quais representam as várias posições em momentos de tempo pelas quais o grupo irá passar na transformação.

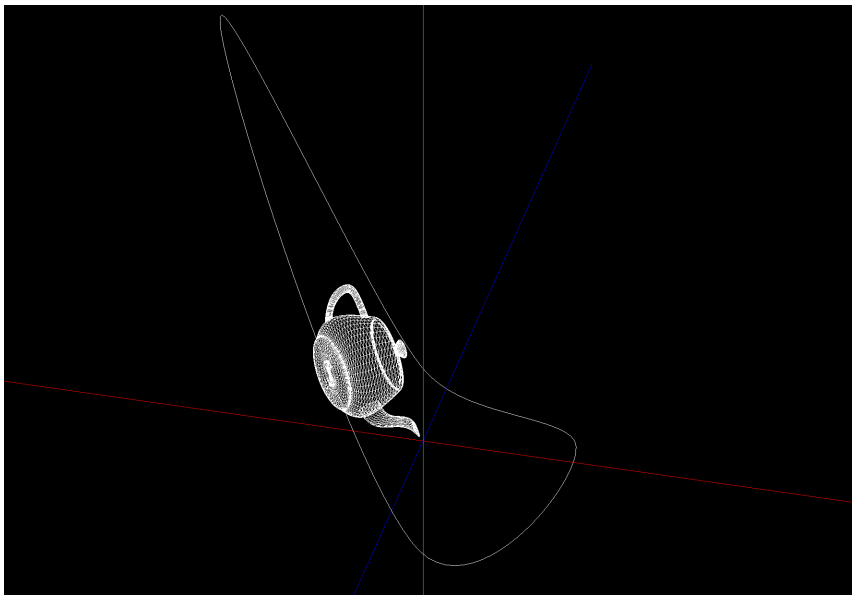

Figura 1: Renderização de um trajeto

4. *Bezier Patches*

Como nova funcionalidade foi também implementada a possibilidade de geração de modelos com *Bezier Patches*, encontrados em ficheiros `.patch`.

Ao gerar este novo tipo de modelos, é necessário passar dois parâmetros:

- O ficheiro `.patch` com os *patches* que formam o modelo, e a informação dos pontos de controlo que os compõem;
- O nível de **tesselação**: este representa o número de divisões que cada *Bezier Patch* do modelo gerado deverá ter, aumentando o seu detalhe quanto mais elevado este parâmetro for.

4.1. Geração

Na geração de modelos `.patch`, para cada um dos *patches*, são calculados os seus pontos, a partir da seguinte fórmula:

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

Sendo P_{ij} os pontos de controlo do dado *patch*, u, $v \in [0,1]$ e

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

Os pontos são calculados por cada componente x, y, z.

Após os pontos estarem todos computados, o problema reduz-se a agrupá-los de forma a criar triângulos. Em cada *patch*, um ponto forma um retângulo com os seus vizinhos diretos. Indexando os pontos com base nos valores de u e v usados para os gerar, um dado ponto P_{ij} será agrupado com os pontos $P_{(i+1)j}$, $P_{i(j+1)}$ e $P_{(i+1)(j+1)}$, formando dois triângulos.

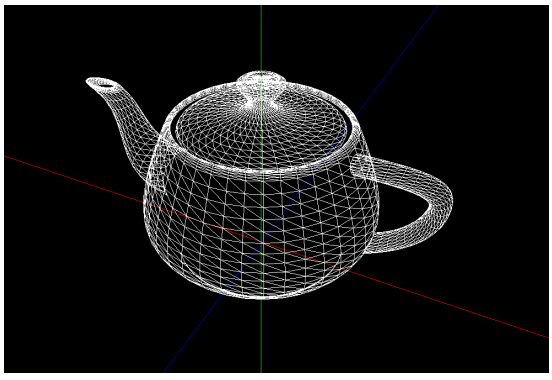

Figura 2: *Teapot*

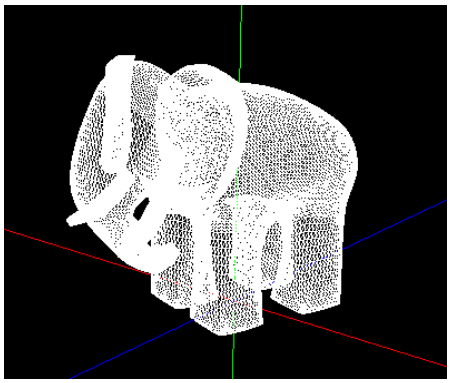

Figura 3: *Gumbo*, Ed Catmull

5. Sistema Solar

Nesta fase, o sistema solar anteriormente adicionado sofreu relevantes alterações. A par da adição de transformações temporais, os planetas, luas, etc. possuem agora animações e movimento, graças às mesmas.

5.1. Cometa

Foi adicionado um cometa à cena do sistema solar, utilizando a funcionalidade de translações temporais e modelos baseados em *patches*. O modelo utilizado como cometa foi o do elefante *Gumbo*, e o trajeto que este percorre é o de uma oval.

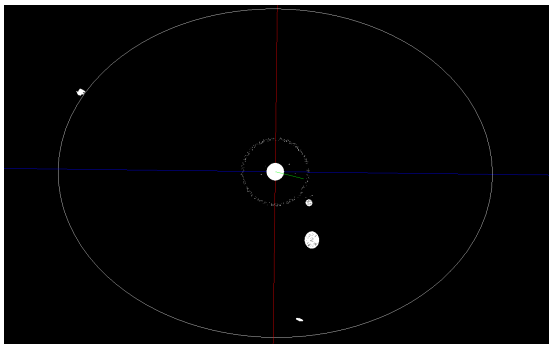

Figura 4: Trajetória do cometa

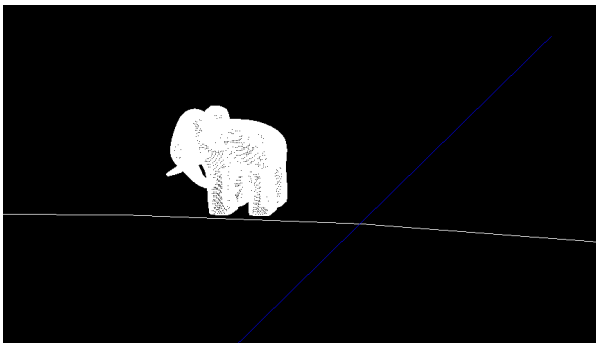

Figura 5: Cometa

6. Conclusão

Concluindo, acreditamos estar a alcançar os objetivos esperados com sucesso. Além das funcionalidades da 4ª fase, pretendemos ainda implementar mais extras, de forma a encerrar o desenvolvimento deste projeto da melhor maneira possível, aprendendo cada vez mais sobre conceitos de Computação Gráfica.