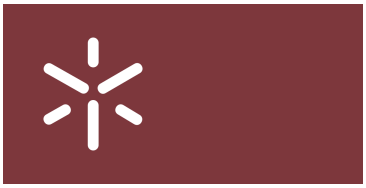


Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Computação Gráfica

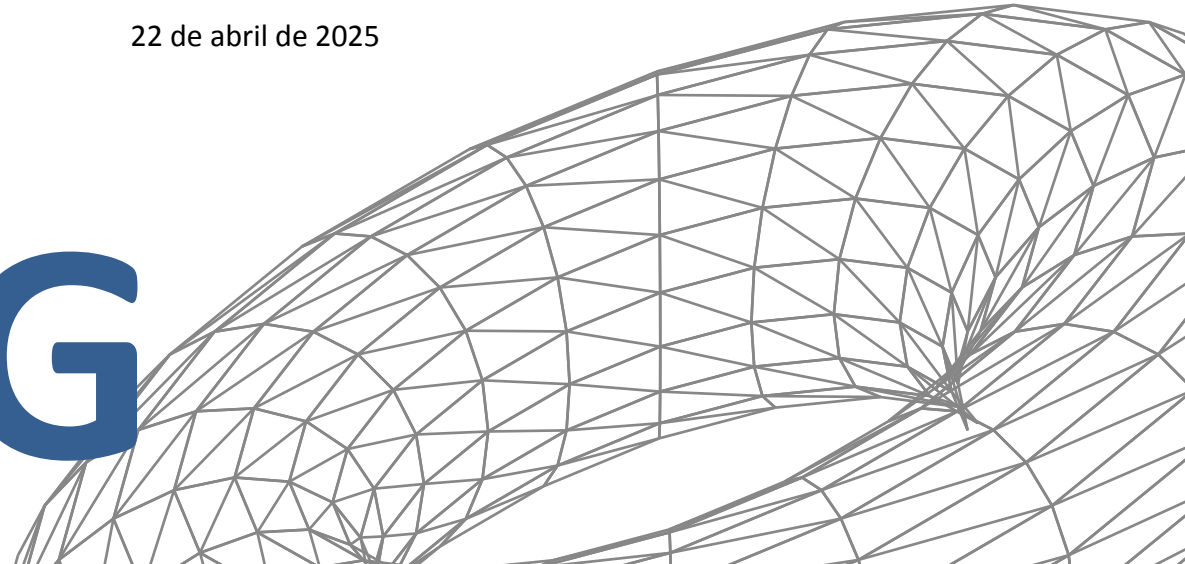
Ano Letivo de 2024/2025

Trabalho Prático - Fase 3

Eduardo Faria	**Hélder Gomes**	**Nuno Silva**	**Pedro Pereira**
a104353	a104100	a104089	a104082

22 de abril de 2025

CG



Data da Receção	
Responsável	
Avaliação	
Observações	

Trabalho Prático - Fase 3

Eduardo Faria	**Hélder Gomes**	**Nuno Silva**	**Pedro Pereira**
a104353	a104100	a104089	a104082

22 de abril de 2025

Resumo

Na terceira fase do projeto, o grupo concretizou a inclusão de curvas de *Catmull-Rom* e geração de modelos com superfícies cúbicas de *Bezier*. A expansão dos parâmetros *rotate* e *translate* foi dada pela adição da funcionalidade que permite definir o tempo total a percorrer uma rotação de 360 graus em torno do eixo especificado e da especificação de pontos a percorrer no seguimento de uma curva *Catmull-Rom*. Adicionalmente, modificou-se o *generator* do projeto de modo a poder converter ficheiros `.patch` em `.3d`, com a indicação to nível de tesselação a impor. Finalmente, realizaram-se algumas melhorias de código anterior e acrescentaram-se alguns extras para complementar o projeto.

Área de Aplicação: Computação Gráfica, Modelação Tridimensional e Motores de Renderização

Palavras-Chave: Computação Gráfica, OpenGL, Primitivas Gráficas, Modelação 3D, XML

Índice

1. Transformações **1**
1.1. Rotações em função do tempo 1
1.2. Curvas de *Catmull-rom* 1

2. *Bezier Patches* **3**
2.1. Fundamentação Teórica 3
2.2. Abordagem de Implementação 3
2.2.1. Leitura e Interpretação de um Ficheiro *.patch* 3
2.2.2. Geração do Modelo Bezier 3
2.2.3. *Tesselation* e Geração de Triângulos 4
2.2.4. Resultados 4
2.2.4.1. *Teapot* 4
2.2.4.2. Cometa 4

3. Extras **6**
3.1. VBOs sem índice — melhoria de performance 6
3.2. Fator de Tempo 6
3.3. Visualização das Curvas de *Catmull-Rom* 7
3.4. Grupos Selecionáveis 8
3.5. VBOs com Índice 8

Referências **10**

Lista de Siglas e Acrónimos **11**

Anexos **12**
Anexo 1 Logo da Universidade do Minho 12
Anexo 2 Resultado da geração e renderização de um patch do teapot 13
Anexo 3 Resultado da geração e renderização de um patch do comet 13
Anexo 4 Resultado do slider de configuração do tempo 14
Anexo 5 Resultado da configuração de uma curva Catmoll-Rom para o cometa 14
Anexo 6 Resultado da aplicação da funcionalidade de grupos selecionáveis 15

1. Transformações

Para concretizar o pretendido para esta fase, foi necessário assegurar que o nosso *engine* era capaz de interpretar um novo conjunto de parâmetros nos atributos de *rotate* e *translate*.

1.1. Rotações em função do tempo

Um campo *rotate* podia agora estar especificado com a definição de um ângulo estático sobre um eixo

```
<rotate angle="35" x="1" y="0" z="0"/>
```

ou com a definição de um tempo total para completar uma rotação de 360 graus, em torno de um eixo

```
<rotate time="100" x="0" y="1" z="0"/>
```

Com um tempo atribuído a um modelo, o *engine* necessitava agora de aplicar uma rotação iterativa sobre o objeto, cujo o valor era definido pela fórmula

$$\Delta = \mathrm{mod}(\mathrm{globalTimer}, t);$$
$$\alpha = \left(\frac{\Delta}{t}\right) * 360;$$

onde t corresponde ao tempo definido para o grupo no ficheiro de configuração, e $\mathrm{globalTimer}$ ao tempo global do programa.

Assim sendo, para finalizar a funcionalidade, restou apenas inserir a noção de tempo no projeto. O grupo escolheu defini-lo com recurso à função `glutGet(GLUT_ELAPSED_TIME)`, incluindo-a no topo da função de execução contínua `renderScene()`. Com a inserção de funcionalidade extra referida na secção Fator de tempo, foi ainda necessário assegurar que a atualização do tempo global era feita de forma iterativa, isto é, o novo tempo atual era calculado a partir da multiplicação do *delta* entre renderização de *frames* com o fator de tempo aplicado.

```
int currentRealTime = glutGet(GLUT_ELAPSED_TIME);
float deltaRealTime = (currentRealTime - lastRealTime);
float deltaTime = deltaRealTime * timeFactor;
globalTimer += deltaTime;
lastRealTime = currentRealTime;
```

1.2. Curvas de *Catmull-rom*

Um atributo de *translate* passou a poder estar definido como uma transformação estática sobre um eixo

```
<translate x="2" y="0.5" z="0"/>
```

ou como uma transformação dinâmica, que define o tempo total que um grupo tem para percorrer o caminho definido pela fórmula de *Catmull-Rom*, aplicada num conjunto de **4 ou mais** pontos, e se o grupo tem ou não de estar alinhado com o vetor *up* de um dado instante da curva

```
<translate time="10" align="True">
    <point x="1" y="0" z="1"/>
    <point x="2" y="0" z="1"/>
    <point x="1" y="0" z="2"/>
    ...
    <point x="2" y="0" z="2"/>
</translate>
```

A funcionalidade foi facilmente introduzida no *engine* através da abstração da lógica de aquisição da posição de um objeto na curva pela função `getGlobalCatmullRomPoint()`, fundamentada pela matéria abordada nas aulas práticas. De modo geral, a função retorna a nova posição de um objeto com base nos **4 pontos adjacentes** à posição atual do objeto, encapsulando todo o cálculo paramétrico de interpolação e expondo apenas uma interface de consulta de trajetória, de forma a manter o *engine* agnóstico aos detalhes matemáticos subjacentes.

Caso o objeto tenha sido definido para estar alinhado à curva, é ainda necessário gerar a matriz de rotação do ponto associada ao vetor de direção da posição em que o objeto se encontra. A matriz resultante é aplicada na função predefinida `glMultMatrixf()` para impor a rotação do objeto na matriz no topo da *stack*.

2. *Bezier Patches*

Nesta fase, abordou-se a implementação de suporte para ficheiros de *Bezier Patches* no *generator*. O objetivo foi de expandir o programa para criar modelos baseados em superfícies de *Bezier*, permitindo a geração de formas complexas a partir de pontos de controlo definidos num ficheiro.

2.1. Fundamentação Teórica

Bezier Patches são uma extensão para superfícies de *Bezier* curvas. Uma superfície de *Bezier* é definida por um conjunto de pontos de controlo organizados numa grelha e utiliza polinómios de *Bernstein* para interpolar estes pontos, gerando uma superfície suave, permitindo a definição de modelos com faces curvadas. A função de base de *Bernstein*, que originou a nossa implementação, para uma superfície de *Bezier* de grau 3 é dada por:

$$B_{i,j}(u,v) = \binom{3}{i}\binom{3}{j}(1-u)^{3-i}u^i(1-v)^{3-j}v^j$$

onde i e j variam de 0 a 3, e u e v são parâmetros que variam entre 0 e 1, controlando a posição na superfície.

O ponto $P(u,v)$ na superfície é calculado como:

$$P(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} B_{i,j}(u,v) \cdot C_{i,j}$$

onde $C_{i,j}$ são os pontos de controlo.

2.2. Abordagem de Implementação

2.2.1. Leitura e Interpretação de um Ficheiro *.patch*

O *generator* lê um ficheiro com extensão `.patch`, que contém a definição dos patches e dos pontos de controlo. A estrutura do ficheiro é a seguinte:

- Linha com um número N de *patches*.
- N linhas seguintes: definição de cada *patch* com 16 índices ($4 * 4$) referenciando pontos de controlo.
- Linha com um número K de pontos de controlo.
- K linhas seguintes: coordenadas dos pontos de controlo (x, y e z).

Derivado desta estruturação, é então possível carregar toda a informação do ficheiro fornecido para memória.

2.2.2. Geração do Modelo Bezier

Para cada patch, o *generator*:

1. Lê os 16 índices dos pontos de controlo.
2. Valida se os índices estão dentro dos limites dos pontos de controlo disponíveis.

3. Para cada passo de *tesselation* (definido pelo utilizador), calcula os pontos na superfície de *Bezier* utilizando a função de base de *Bernstein*.
4. Gera triângulos entre os pontos interpolados para formar a superfície.
5. Aplica a equação matemática da superfície de *Bezier* por transcrição direta para código usando a função `evaluateBezierPatch()`, combinando os pontos de controlo com os polinómios de *Bernstein*.

2.2.3. *Tesselation* e Geração de Triângulos

O nível de *tesselation* determina a quantidade de detalhe na superfície gerada. Para cada passo de *tesselation*, o *generator* calcula quatro pontos na superfície (p00, p01, p10, p11) e forma dois triângulos, garantindo uma representação contínua da superfície.

2.2.4. Resultados

2.2.4.1. *Teapot*

Numa primeira instância foi testada a submissão do ficheiro `teapot.patch` disponibilizado pela equipa docente, o qual resultou na correta visualização após geração do ficheiro `.3d` correspondente.

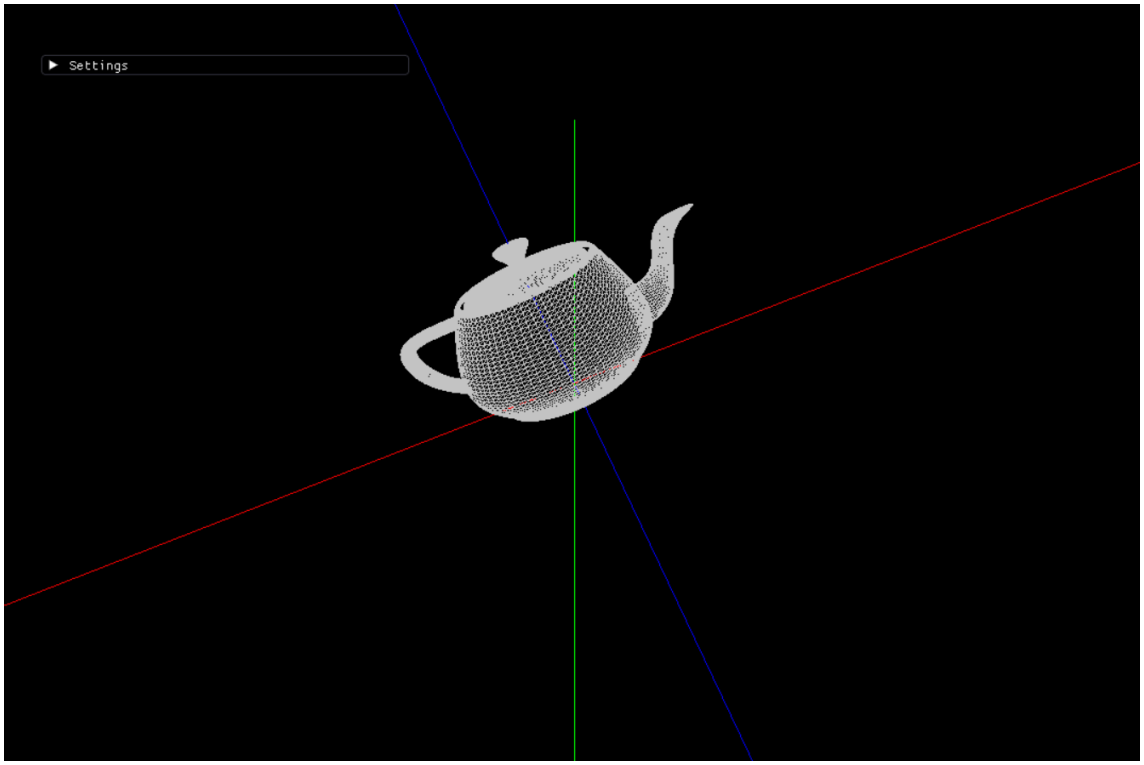


Figura 1: Resultado da geração e renderização de um patch do teapot.

2.2.4.2. Cometa

Para o cometa, o grupo procedeu à pesquisa de um ficheiro `.patch` que contivesse a representação das curvas suaves de um cometa e que encaixasse na definição previamente apresentada para *parsing*, sendo posteriormente introduzido no `.xml` de configuração do sistema solar com uma curva de *Catmull-Rom* associada para descrever a sua órbita.

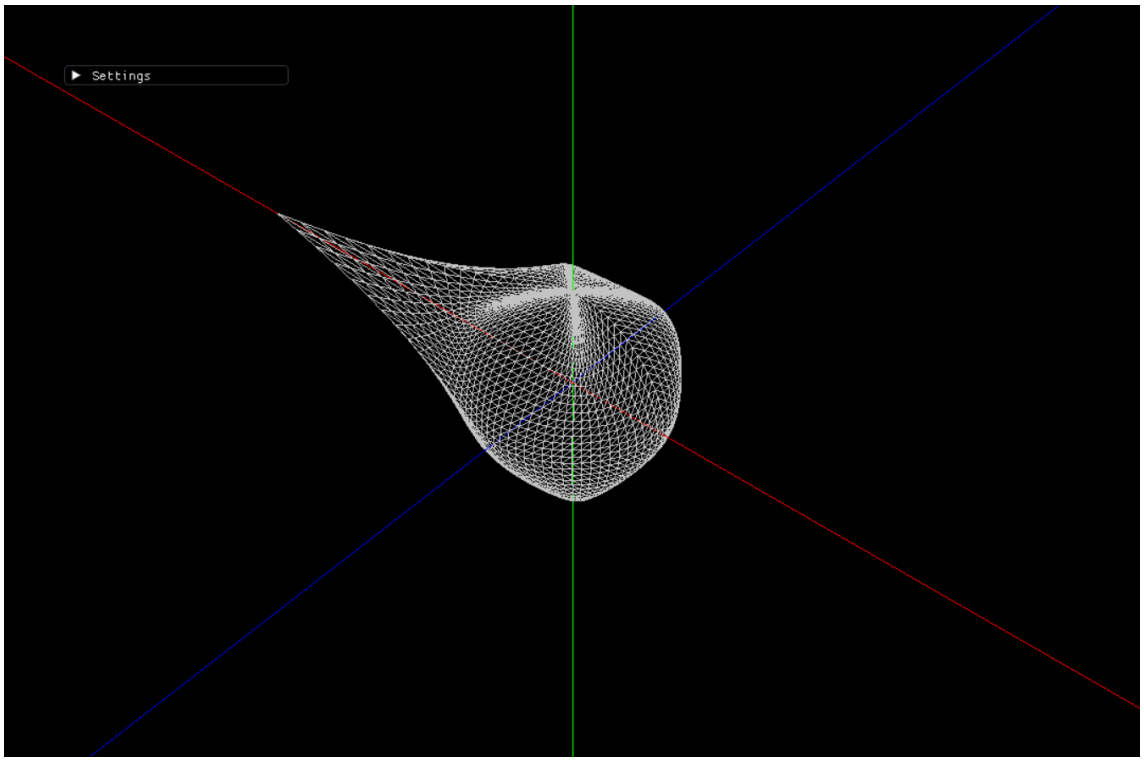


Figura 2: Resultado da geração e renderização de um patch do comet.

3. Extras

3.1. VBOs sem índice — melhoria de performance

No início da terceira fase, o grupo deparou-se com algumas redundâncias no armazenamento dos modelos nos VBOs. Em concreto, por cada ficheiro que era interpretado no ficheiro de configuração de cena, era gerada um novo VBO para esse modelo, independentemente se esse mesmo ficheiro já tivesse sido armazenado.

Para consolidar a melhoria, o grupo começou por mapear o nome do ficheiro à estrutura de dados que armazenaria o conjunto de pontos do ficheiro. Deste modo, por cada ficheiro que era interpretado no ficheiro de configuração de cena, era averiguada a sua presença no mapa de modelos. Sendo assim, o mapa resultante continha apenas os *buffers* necessários e os grupos, para posterior renderização, apenas mantinham o índice do VBO que lhes correspondia

Assumindo que uma dada cena incluía **2 modelos distintos**, mas que eram utilizados **100 vezes**, seriam gerados **100 VBOs** — **98** *buffers* iriam ser repetidos.

Com a alteração em questão, eliminou-se o armazenamento de, neste caso, **98** *buffers* redundantes. No sistema solar de geração automática, o número de FPS passou de **413 FPS**[1] para **802 FPS**[1].

3.2. Fator de Tempo

O grupo acrescentou a possibilidade de alterar o fator de tempo, isto é, acelerar ou abrandar a velocidade de acréscimo no tempo global do programa. O utilizador dispõe desta funcionalidade no menu através de um controlador deslizante. Como esperado, o fator de tempo influencia a velocidade com que os grupos realizam as suas rotações (caso definidos com rotações dinâmicas) e/ou translações sobre curvas de *Catmull-Rom* (caso definidos com translações dinâmicas). O valor do fator de tempo pode ainda ainda ser 0, tornando a cena indefinidamente estática.

[1] Testes realizados num *MacBook Air* M1 2020.

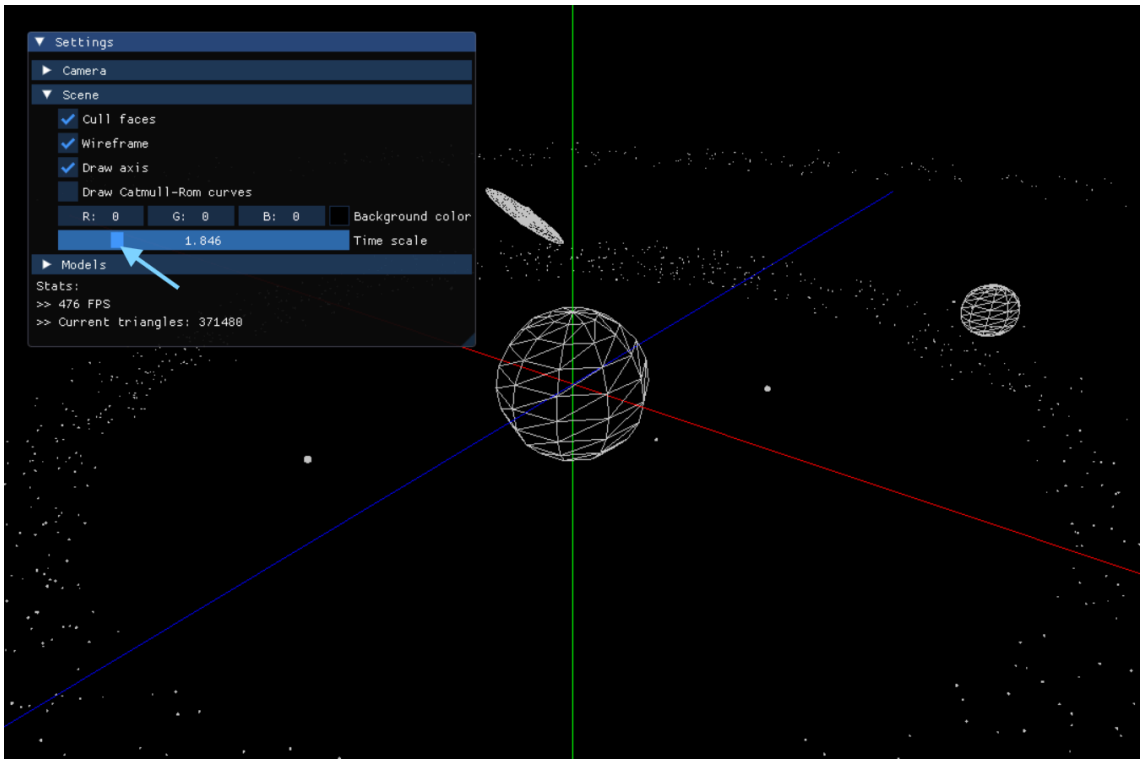


Figura 3: Resultado do slider de configuração do tempo.

3.3. Visualização das Curvas de *Catmull-Rom*

Para efeitos visuais, o utilizador dispõe da possibilidade de renderizar linhas representativas das curvas de *Catmull-Rom* definidas no ficheiro de configuração de cena.

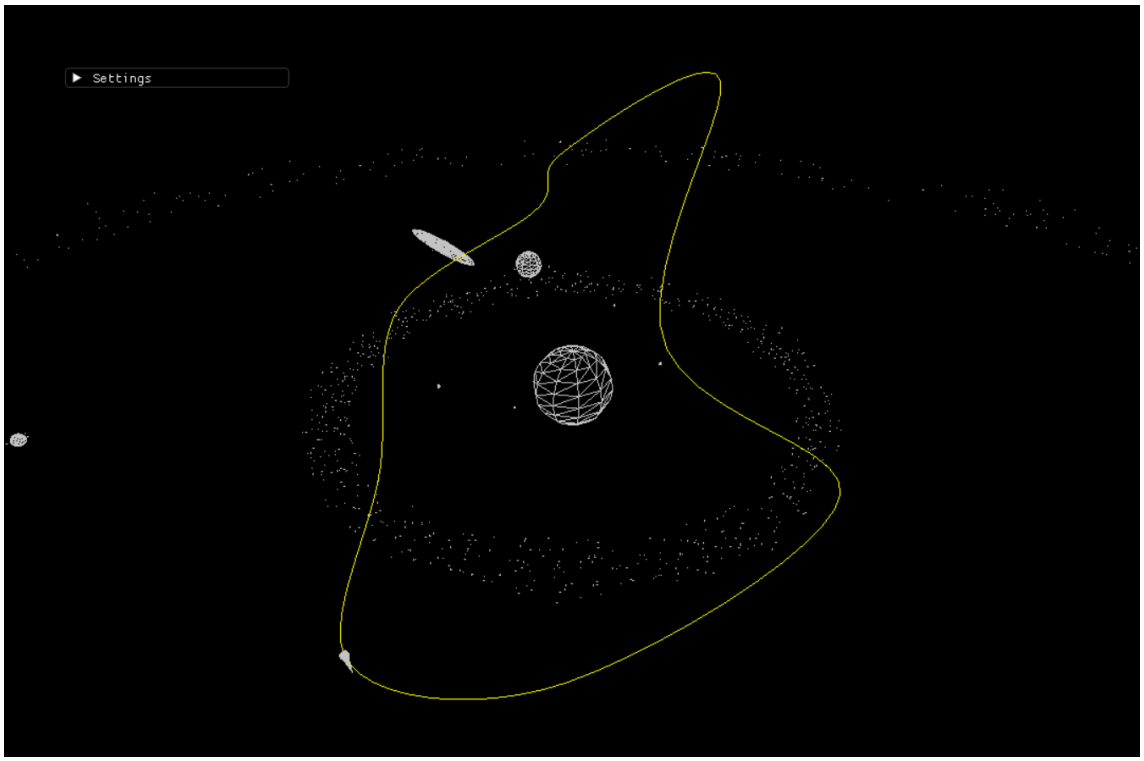


Figura 4: Resultado da configuração de uma curva Catmoll-Rom para o cometa.

3.4. Grupos Selecionáveis

Foi possível acrescentar a possibilidade de tornar os planetas clicáveis e mostrar informação adicional sobre os mesmos. Para a sua implementação, o grupo recorreu à estratégia abordada nas aulas práticas, renderizando uma cena adicional não visível com profundidade que faz uso de cores para identificação de objetos. Adicionalmente, um grupo era considerado "clicável" caso incluísse um caminho para um ficheiro com informação textual:

```
<group name="Sun" clickableInfo="../../group_info/sun.txt">
```

Na inicialização do *engine*, um identificador é atribuído a um objeto e é responsável por definir a cor do mesmo na cena não visível. Quando o utilizador clica com o **botão direito**, é desencadeada a função `picking()`, que renderiza uma cena não visível e retorna o valor da cor vermelha no *pixel* em que o cursor se encontra. Deste modo, assumindo que as cores de um modelo estão inseridas numa *grey scale*, é possível utilizar o valor retornado como identificador do objeto.

Caso um objeto tenha sido selecionado, a câmara altera o seu foco (`gluLookAt()`) para o centro do grupo e surge um menu com a informação contida no ficheiro passado como referência.

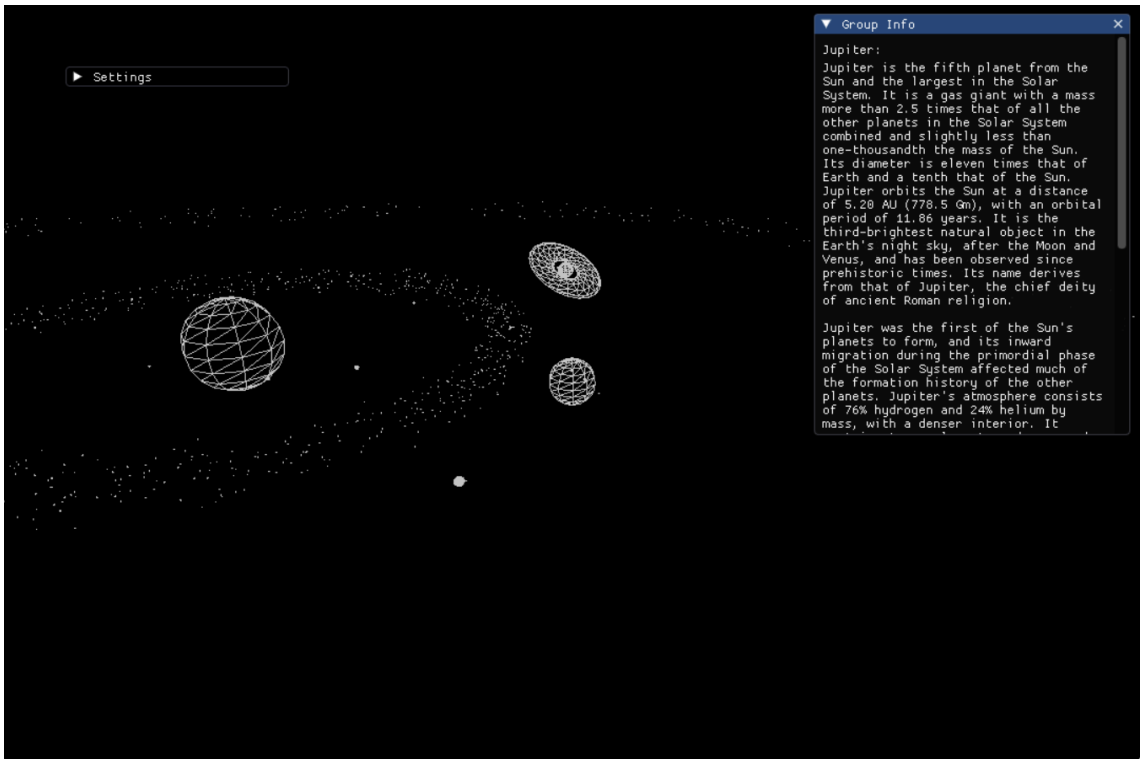


Figura 6: Resultado da aplicação da funcionalidade de grupos selecionáveis.

3.5. VBOs com Índice

Com a inclusão de cada vez mais funcionalidades, o grupo decidiu ainda alterar a sua abordagem com os VBOs. Nomeadamente, executou-se uma reestruturação do código para renderizar os objetos através de VBOs com índices.

Antes da transição para VBOs com índice, o grupo implementou uma abordagem inicial baseada em *Vertex Buffer Objects* (VBOs) sem recurso a qualquer estrutura de índices. Nesta metodologia, cada modelo armazenava sequencialmente todos os vértices necessários para a sua representação, sem reutilização explícita de vértices partilhados entre triângulos. Para desenhar os modelos, a aplicação ativava o estado de cliente de vértices com `glEnableClientState(GL_VERTEX_ARRAY)`, associava o VBO correspon-

dente através de `glBindBuffer(GL_ARRAY_BUFFER, vboID)` e configurava o apontador para os vértices com `glVertexPointer(3, GL_FLOAT, 0, 0)`. A renderização era então realizada com a chamada a `glDrawArrays(GL_TRIANGLES, 0, n)` onde n representava o número total de vértices do modelo.

Embora esta abordagem já proporcionasse melhorias face à renderização tradicional (como demonstrado no relatório da primeira fase), uma vez que os dados dos vértices eram transferidos para a memória da placa gráfica, eram visíveis algumas limitações de desempenho e de otimização de memória, sobretudo em modelos com elevada redundância de pontos. Cada triângulo duplicava a informação dos vértices partilhados, aumentando desnecessariamente a dimensão dos *buffers* e o volume de dados processados na *pipeline* gráfica.

Com a nova estratégia, passou-se a utilizar também **IBOs** (*Index Buffer Objects*), permitindo a separação entre a informação dos vértices e a informação sobre como estes se interligam para formar triângulos. Assim:

- O **VBO** contém apenas a lista única de vértices do modelo.
- O **IBO** armazena índices que especificam, de forma eficiente, como os vértices são agrupados em triângulos.

Durante o processo de desenho de um modelo passou a ser feito de forma diferente. Nomeadamente, o VBO correspondente ao modelo é inicialmente ativado com `glBindBuffer(GL_ARRAY_BUFFER, vboID)` e o formato dos vértices é igualmente definido através da função `glVertexPointer(3, GL_FLOAT, 0, 0)`. Agora, adicionalmente, define-se IBO associado ao VBO em questão com `glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, iboID)` e, por fim, invoca-se `glDrawElements(...)` para desenhar todos os triângulos, com base nos índices.

Esta abordagem trouxe uma redução do tamanho dos *buffers*, uma vez que cada vértice é armazenado apenas uma vez. Uma menor quantidade de dados transferidos para a placa gráfica leva, naturalmente, a uma melhoria de desempenho notável. Para a mesma cena de renderização do sistema solar registou-se um aumento na média de FPS, passando de **802 FPS**[2] para **907 FPS**[2], evidenciando uma melhoria relevante no desempenho.

[2] Testes realizados num *MacBook Air* M1 (2020).

Referências

CMake. (2025). CMake. https://cmake.org/

The Khronos Group. (2025). OpenGL. https://www.opengl.org/

John F. Fay, John Tsiombikas, and Diederick C. Niehorster. (2025). freeGlut. https://freeglut.sourceforge.net/

Lee Thomason. (2025). TinyXML2 [Source code]. GitHub. https://github.com/leethomason/tinyxml2

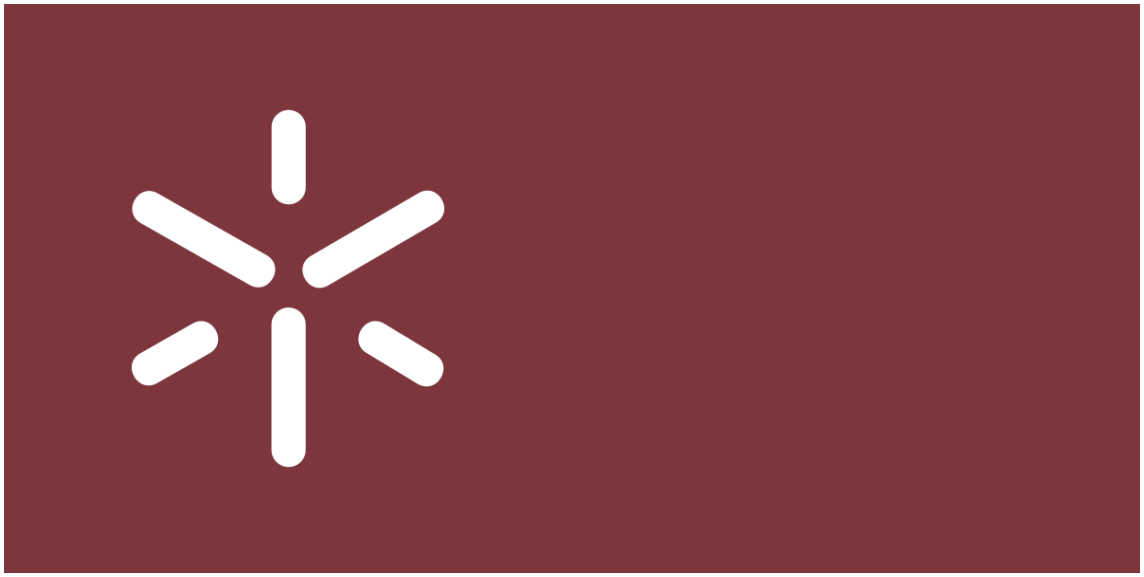
orconut. (2025). Dear imGui [Source code]. GitHub. https://github.com/ocornut/imgui

Lista de Siglas e Acrónimos

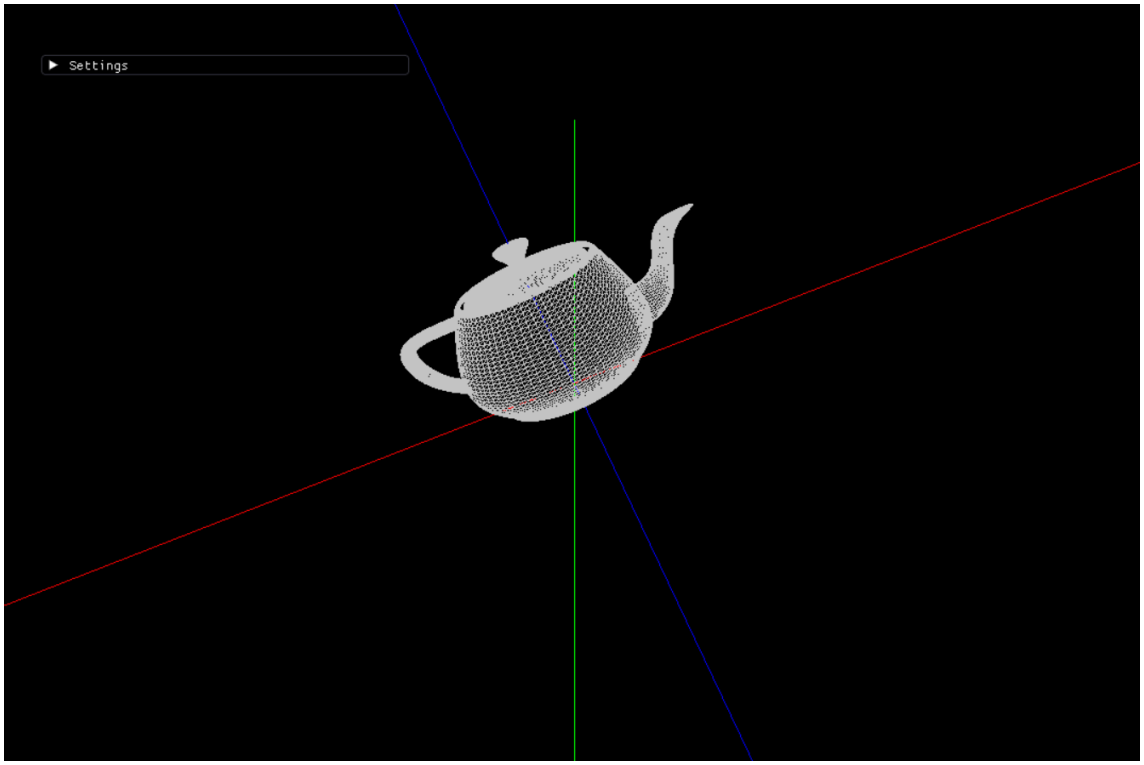
CG Computação Gráfica
XML *Extensible Markup Language*
VBO *Vertex Buffer Object*
IBO *Index Buffer Object*
FPS *Frames per Second*

Anexos

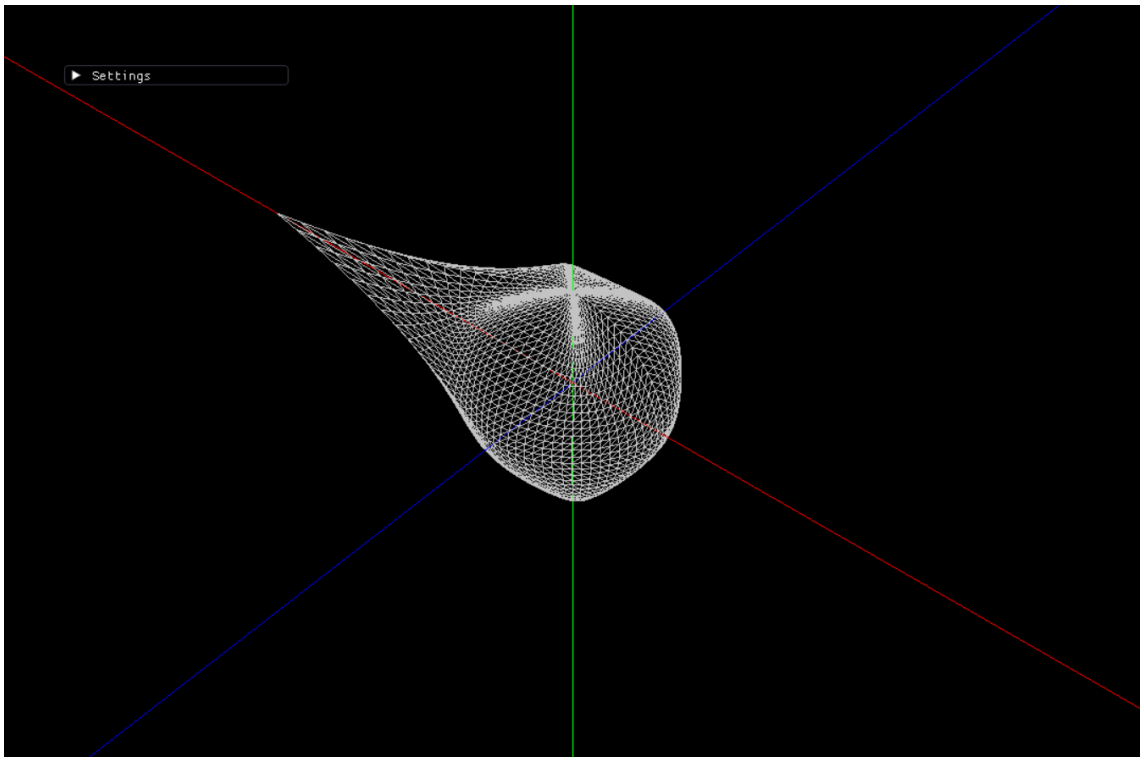
Anexo 1: Logo da Universidade do Minho



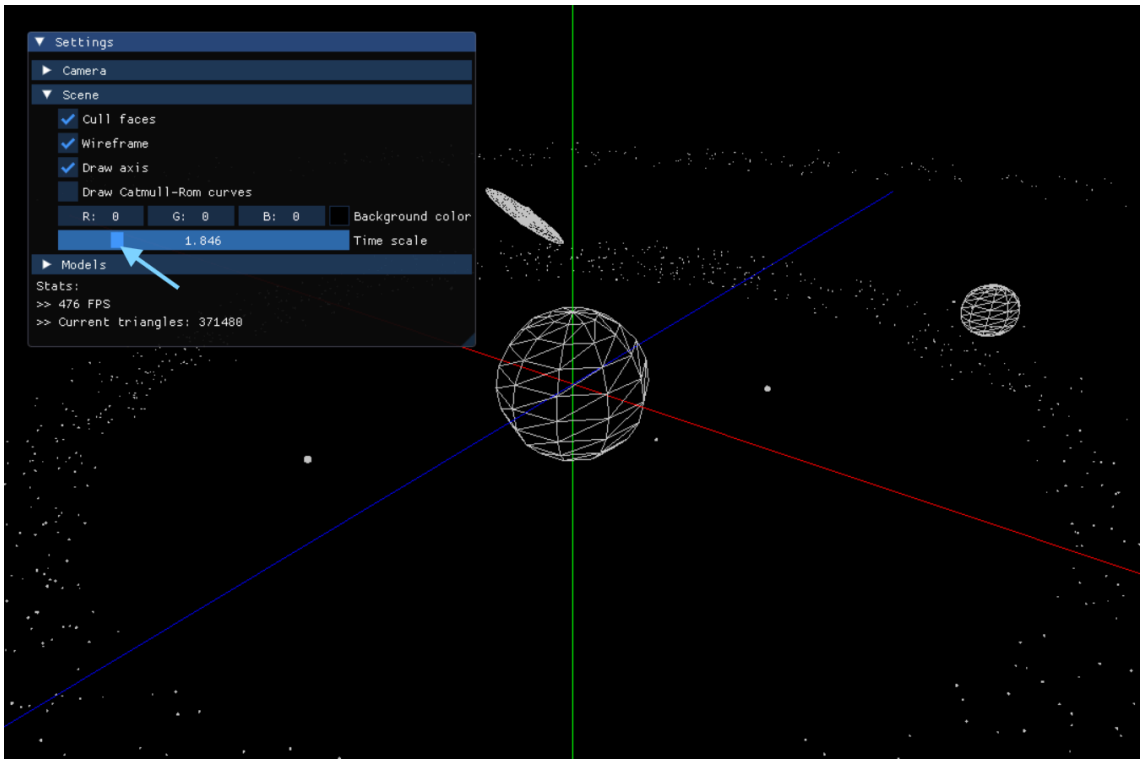
Anexo 2: Resultado da geração e renderização de um patch do teapot



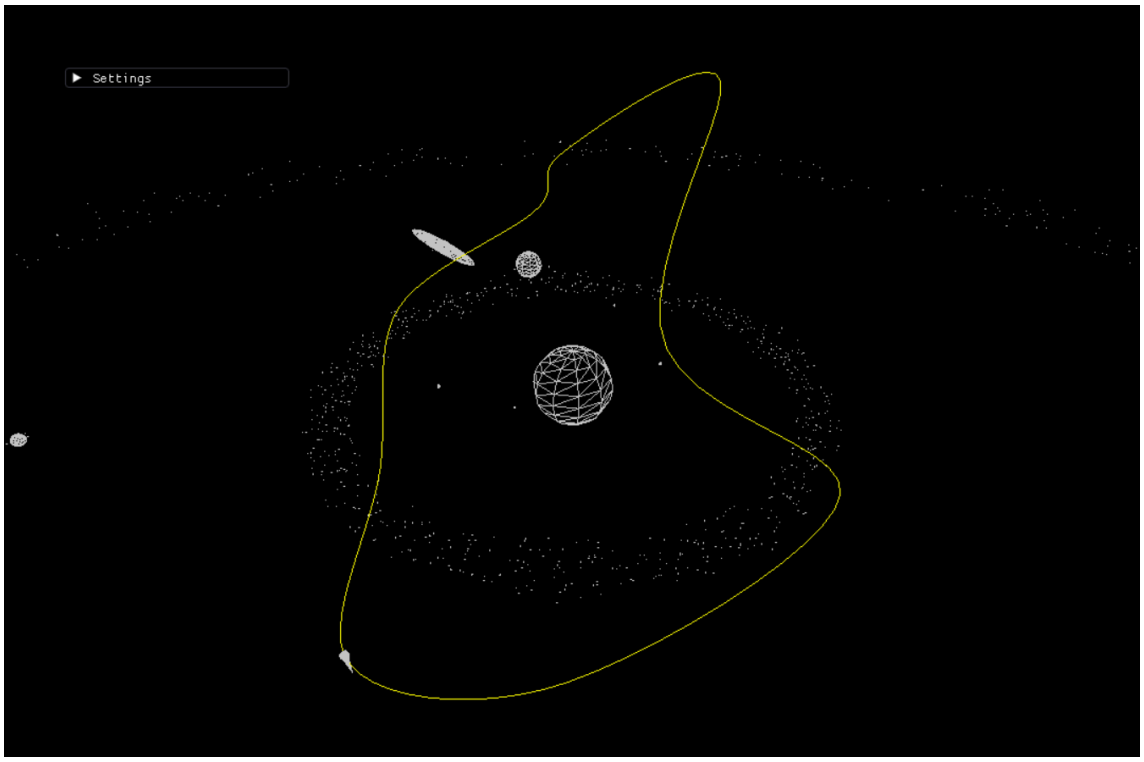
Anexo 3: Resultado da geração e renderização de um patch do comet



Anexo 4: Resultado do slider de configuração do tempo



Anexo 5: Resultado da configuração de uma curva Catmoll-Rom para o cometa



Anexo 6: Resultado da aplicação da funcionalidade de grupos selecionáveis