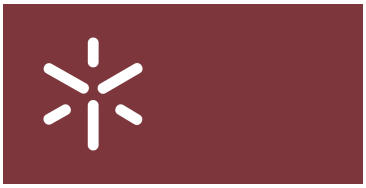

Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática
Licenciatura em Engenharia Informática

Unidade Curricular - Computação Gráfica

Ano Letivo de 2024/2025

Fase 3 - Curvas, Superfícies Cúbicas e VBOs

Grupo 36

Tomás Henrique Alves Melo a104529
Carlos Eduardo Martins de Sá Fernandes a100890
Hugo Rafael Lima Pereira a93752
Alexandre Marques Miranda a104445

CG

Abril, 2025

Resumo

Esta fase do projeto focou-se na implementação de transformações dinâmicas dentro de um ambiente 3D, especificamente nas animações de translação e rotação de objetos. A translação ao longo de curvas Catmull-Rom foi implementada, proporcionando movimentos suaves e naturais entre pontos de controlo, enquanto a rotação contínua permitiu a simulação de movimentos planetários. A alteração do motor de renderização e a manipulação dos VBOs (Vertex Buffer Objects) foram cruciais para permitir a implementação eficiente dessas animações.

Área de Aplicação: Animação de objetos ao longo de curvas (Catmull-Rom) de forma automática e suave; Geração e visualização de superfícies tridimensionais complexas a partir de patches de Bezier; Renderização eficiente utilizando VBOs para otimizar a performance gráfica.

Palavras-Chave: Computação Gráfica, OpenGL, Animação 3D, Curvas Catmull-Rom, VBOs, Transformações Geométricas, Simulação de Movimentos, Rotações Continuas.

Índice

Lista de Figuras

1. Introdução

O objetivo principal desta fase foi adicionar animações dinâmicas a um ambiente 3D, implementando movimentos contínuos de objetos, como rotação e translação ao longo de curvas. A aplicação de transformações geométricas permitiu simular movimentos realistas, como a órbita de planetas, além de facilitar o desenvolvimento de um sistema de animação robusto. Para alcançar isto, mudámos e expandimos a estrutura da engine, adicionando suporte para transformações baseadas no tempo e interpolação de curvas.

2. Objetivos Impostos

O objetivo principal desta fase do projeto foi simular movimentos complexos e dinâmicos em cenários 3D, com ênfase na animação de objetos por meio de transformações geométricas. Especificamente, procurou-se criar animações contínuas de rotação e translação, aplicadas a objetos num espaço tridimensional, com o intuito de reproduzir movimentos realistas, como o deslocamento de planetas. Além disso, a integração de VBOs (Vertex Buffer Objects) permitiu otimizar o desempenho da renderização, permitindo um processamento mais eficiente das animações e garantindo que os movimentos fossem aplicados de forma fluída e sem perdas de desempenho.

3. Arquitetura Desenvolvida

Para esta fase foi preciso criar um novo programa a que chamámos bezier.cpp localizado na diretoria src/Generator. Este programa é responsável por gerar a geometria de um modelo 3D baseado em curvas de Bezier capaz de gerar um ficheiro de saída com os vértices e índices para posterior renderização, um dos requisito para a Fase 3.

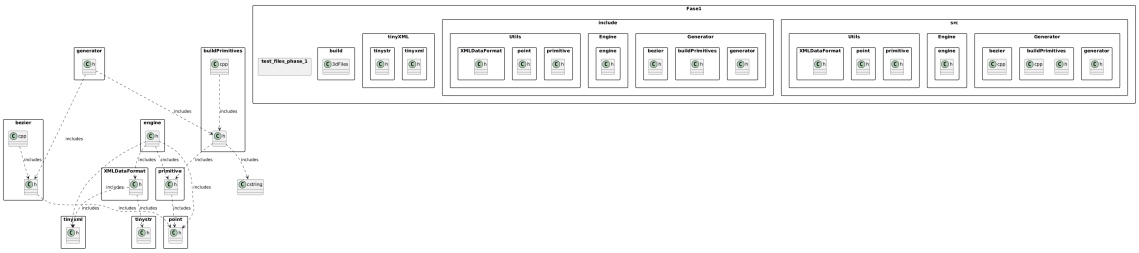

Figura 1: Arquitetura para a Fase 3

4. Generator

4.1. Criação de primitivas baseadas em patches Bezier

4.1.1. Processo

- Ler o ficheiro de patches e pontos de controlo.
- Para cada patch:
  - Dividir o patch numa grelha com `(tessellation + 1)` pontos por lado.
  - Calcular cada ponto utilizando a superfície de Bézier.
  - Gerar triângulos a partir dos pontos da grelha.
- Escrever os vértices e índices no ficheiro de saída.

4.1.2. Fórmulas Usadas

4.1.2.1. Matriz de Bézier

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

4.1.2.2. Função de Blend

Para um parâmetro t em [0,1] e linha i:

$B_{i(t)} = M_{\mathrm{i0}}t^3 + M_{\mathrm{i1}}t^2 + M_{\mathrm{i2}}t + M_{\mathrm{i3}}$

4.1.2.3. Geração dos Triângulos

Cada quadrado da grelha gera dois triângulos:

- Primeiro triângulo: $(i, j), (i, j+1), (i+1, j)$
- Segundo triângulo: $(i, j+1), (i+1, j+1), (i+1, j)$

4.1.3. Teapot

Aplicação ao ficheiro teapot.patch a partir do programa bezier.cpp desenvolvido:

Ao rodar o programa com o ficheiro teapot.patch, este é capaz de gerar a malha 3D do teapot através da lógica explicada acima, criando assim um ficheiro .3d com vértices e triângulos prontos para a renderização.

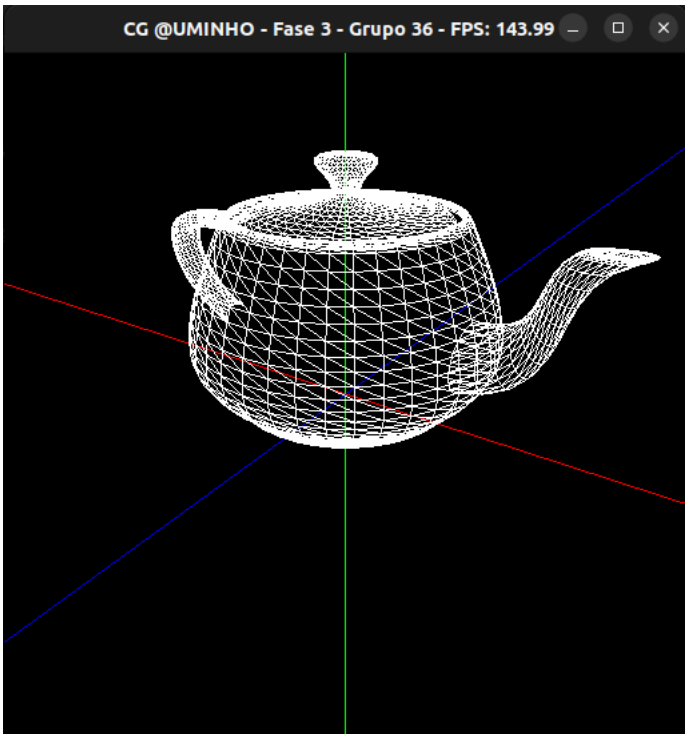

Figura 2: Teapot - A partir do test_3_2.xml

4.1.4. Cometa

4.1.4.1. Processo

Compilar e executar o ficheiro gerador do cometa.

```
gcc cometgen.c -o comet -lm ; ./comet
```

O programa cometgen.c gera um ficheiro de extensão .patch (comet.patch).

Gerar o ficheiro comet.3d que contém os vértices e índices necessários para a formação do cometa.

```
cd build ; ./generator patch ../comet.patch 10 comet.3d
```

Renderização do cometa na janela gráfica.

```
./engine ../test_files_phase_3/my_test_3_2.xml
```

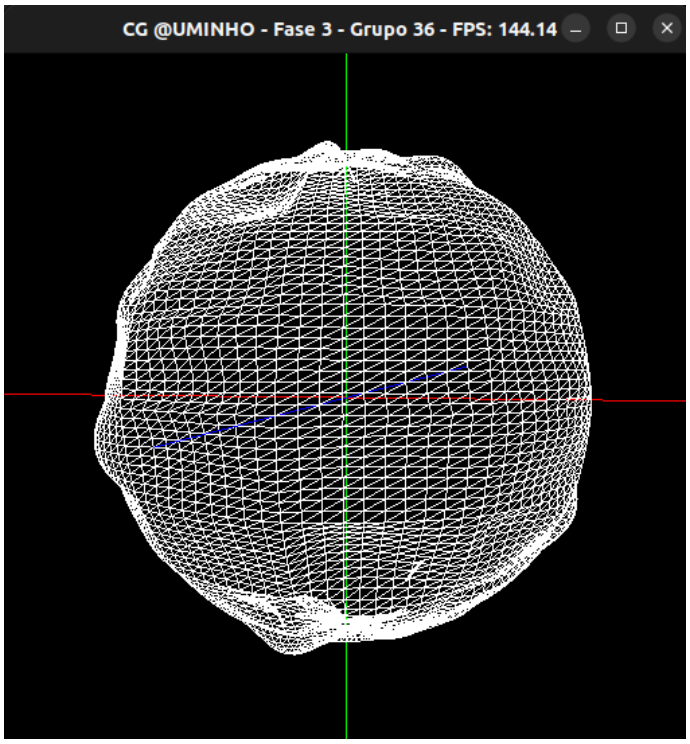

Figura 3: Cometa desenvolvido

5. Engine

A engine foi ajustado para permitir a aplicação de animações dinâmicas. Além das transformações geométricas, foi necessário ajustar a forma como os VBOs eram processados. VBOs foram otimizados para permitir a aplicação de transformações durante a renderização, o que levou a uma renderização mais eficiente. A atualização contínua dos estados das transformações foi implementada no Engine para garantir que as animações fossem aplicadas sem perdas de desempenho. A introdução de VBOs também garantiu a integração fluida entre os objetos e a aplicação de animações baseadas em tempo.

Para suportar animações dinâmicas, a arquitetura do motor foi substancialmente alterada. Inicialmente, o motor trabalhava com transformações estáticas aplicadas uma única vez. Com a introdução das animações, foi necessário implementar uma estrutura capaz de lidar com transformações que variam ao longo do tempo, especialmente com a rotação e translação contínuas.

Uma das modificações principais foi a introdução de uma abstração de transformação, representada por uma classe base `Transform`, que tem subclasses específicas para `Translate`, `Rotate` e `Scale`. Cada transformação pode agora ser calculada dinamicamente durante a execução, com base no tempo decorrido, permitindo animações suaves.

5.1. Curvas Catmull-Rom

5.1.1. Processo

- Ler os pontos de controlo para a curva de Catmull-Rom.
- Para cada segmento de curva:
  - Calcular a posição (P(t)) para cada valor de (t) no intervalo [0, 1].
  - Calcular a tangente (P'(t)) usando a derivada da curva.
  - Alinhar o objeto ao longo da curva utilizando a tangente.
  - Calcular a rotação do objeto com base na tangente da curva para o parâmetro de tempo.
- Atualizar a posição e orientação do objeto ao longo da animação.

5.1.2. Fórmulas Usadas

5.1.2.1. Curva de Catmull-Rom

Para um parâmetro (t) em [0,1], a posição ao longo da curva é dada por:

$$P(t) = \frac{1}{2}[(2P_1) + (-P_0 + P_2)t + (2P_0 - 5P_1 + 4P_2 - P_3)t^2 + (-P_0 + 3P_1 - 3P_2 + P_3)t^3]$$

5.1.2.2. Derivada de Catmull-Rom (Tangente)

A derivada (ou tangente) da curva é dada por:

$$P'(t) = \frac{1}{2}[(-P_0 + P_2) + 2(-P_0 + 2P_1 - 2P_2 + P_3)t + 3(2P_0 - 5P_1 + 4P_2 - P_3)t^2 + 4(-P_0 + 3P_1 - 3P_2 + P_3)t^3]$$

5.1.2.3. Rotação Temporal

A rotação do objeto ao longo da curva é calculada com base na tangente em:

$$\text{Rotação} = [\text{Matriz de rotação}](P'(t))$$

5.2. VBOs (Vertex Buffer Objects)

5.2.1. Processo

- Criar um VBO (Vertex Buffer Object) para armazenar os dados dos vértices.
- Carregar os dados de vértices no VBO.
- Utilizar o VAO durante a renderização para acessar os dados armazenados no VBO.

5.2.1.1. Definição de VBO

Um VBO é uma estrutura que armazena dados de vértices na memória da GPU, facilitando a renderização de objetos 3D.

A criação de um VBO envolve a alocação de memória na GPU e a cópia dos dados do CPU para a memória da GPU. O código básico para criar e carregar dados no VBO é:

```
GLuint VBO;
glGenBuffers(1, &VBO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

5.2.1.2. Estratégia adotada

1. A função loadModel começa por carregar os dados do modelo (vértices e índices).
2. Para usar o modelo com eficiência em OpenGL, a função cria VBOs e um Element Buffer Object (EBO).
   - VBO: Armazena os vértices.
   - EBO: Armazena os índices dos vértices, permitindo a renderização de triângulos com base nesses índices.
3. Criação do VBO:
   - Vértices são armazenados no VBO com glBufferData, onde são transferidos para a memória da GPU.
   - O EBO é criado de forma similar, armazenando os índices com a função glBufferData
4. Atribuição de Atributos de Vértices:
   - A função configura os atributos de vértices no VAO (Vertex Array Object). Com isso, o OpenGL sabe como interpretar os dados dos vértices que foram armazenados no VBO.
5. Cache de VBOs:
   - Para evitar a recriação de VBOs a cada renderização, uma cache pode é mantida. O vboCache verifica se o modelo já foi carregado na memória da GPU antes, e se sim, usa o VBO e EBO existentes.

6. Novas Funcionalidades Implementadas

De forma a testarmos a correta implementação de VBOs e a capacidade desta de criar um ambiente de maior performance, principalmente em gráficos 3D mais complexos, uma vez que a GPU consegue aceder muito mais rapidamente aos dados ao reduzir o número de vezes que a CPU comunica com a GPU para envio de informação, implementámos a visualização do número de FPS (Frames per Second) na janela gráfica que mostra a qualidade de desempenho do modelo gráfico gerado.

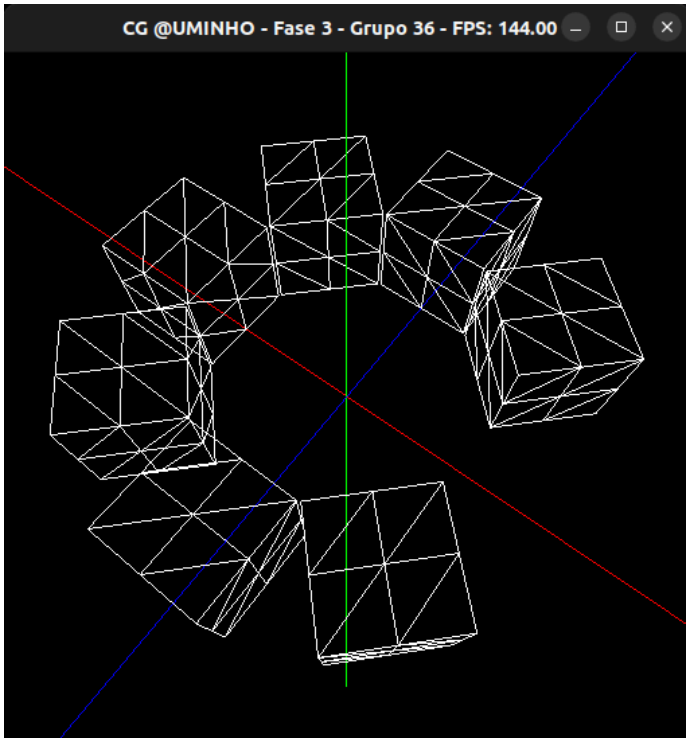

Figura 4: Visualização do número de FPS da cena

7. Cena de Demonstração (Demo Scene)

Nesta fase do projeto, implementámos a translação dos planetas ao redor do Sol e adicionámos um cometa ao sistema solar. Cada planeta agora segue o seu movimento orbital com base no período de translação real, o que aproximou significativamente a simulação da realidade.

O sistema foi ajustado para que 1 dia real corresponda a 1 segundo no XML, permitindo uma animação contínua onde o movimento de cada planeta é calculado com base no seu período real de translação. Os valores de translação dos planetas foram convertidos para dias de animação, garantindo que a simulação ocorra de forma fiel ao movimento orbital real, mas mantendo a fluidez necessária para a visualização.

A introdução do cometa trouxe mais dinamismo à simulação, já que a sua trajetória não segue uma órbita fixa, mas sim uma curva suave gerada por interpolação de Catmull-Rom, o que adiciona um movimento mais natural ao sistema.

A tabela seguinte apresenta os períodos de translação dos planetas e das suas luas principais, convertidos para dias de animação no XML (1 segundo = 1 dia real).

Astro/Lua	Período de Translação (em anos)	Período de Translação (em dias de animação)
Mercúrio	0.24	87,6 dias (0.24 * 365)
Vénus	0.615	224,5 dias (0.615 * 365)
Terra	1	365 dias
Marte	1.88	686 dias
Júpiter	11.86	4.321 dias (11.86 * 365)
Saturno	29.46	10.742 dias (29.46 * 365)
Urano	84.01	30.658 dias (84.01 * 365)
Neptuno	164.8	60.144 dias (164.8 * 365)
Plutão	248	90.520 dias (248 * 365)

Figura 5: Períodos de Translação dos Astros

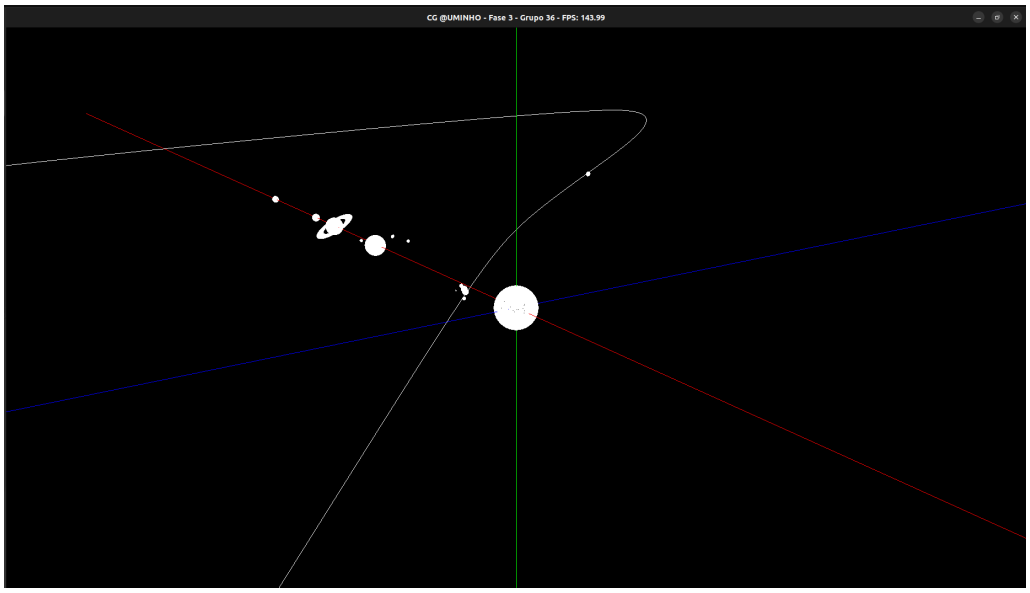

Figura 6: Sistema solar desenvolvido a partir de curvas de Catmull-Rom

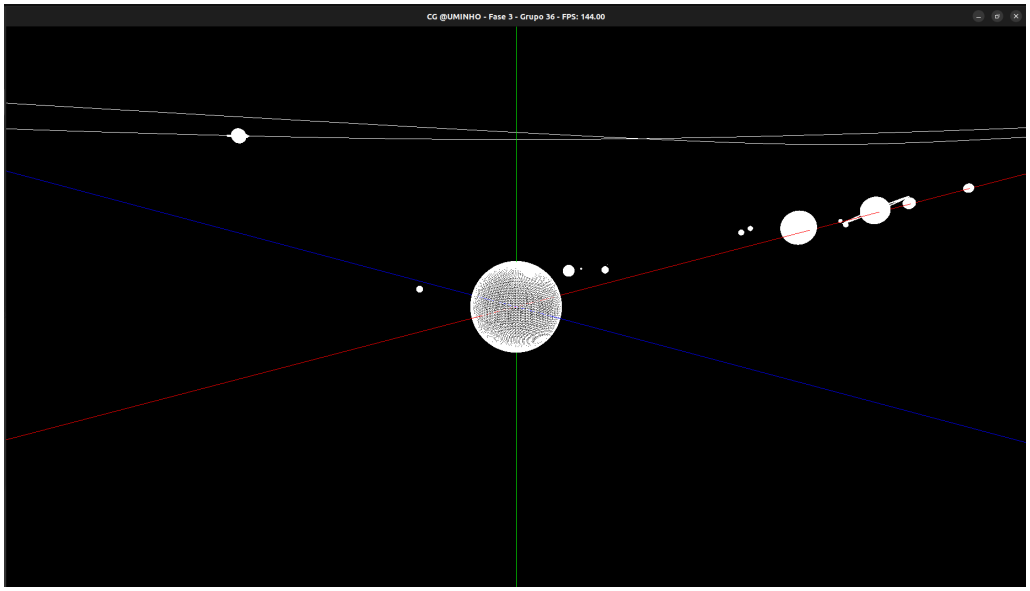

Figura 7: Sistema solar - zoomed-in

8. Conclusões e Trabalho Futuro

Durante o desenvolvimento desta terceira fase do projeto, tivemos a oportunidade de aplicar na prática diversos conceitos de computação gráfica, como as superfícies de Bézier e as curvas de Catmull-Rom, o que contribuiu significativamente para o reforço da nossa compreensão sobre o funcionamento destas técnicas. Relativamente à execução do trabalho, estamos satisfeitos, pois conseguimos implementar todas as funcionalidades exigidas nesta etapa. Em suma, acreditamos que conseguimos reunir, ao longo desta fase, todas as bases necessárias para avançar com confiança para a quarta e última fase do trabalho prático.

Referências

Catmull, E., & Rom, R. (1974). A Class of Local Interpolating Splines. Computer Aided Geometric Design. https://www.sciencedirect.com/science/article/abs/pii/B9780120790500500246

Farin, G. (2002). Curves and Surfaces for CAGD: A Practical Guide (5th ed.). Morgan Kaufmann. https://www.sciencedirect.com/book/9781558607378/curves-and-surfaces-for-cagd

Munshi, A. (2009). OpenGL ES 2.0 Programming Guide. Addison-Wesley. https://www.oreilly.com/library/view/opengl-es-20/9780321563835/

Kessenich, J., Sellers, G., & Lichtenbelt, B. (2016). OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V (9th ed.). Addison-Wesley. https://www.opengl.org/documentation/

Hughes, J. F., van Dam, A., McGuire, M., Sklar, D. F., Foley, J. D., Feiner, S. K., & Akeley, K. (2013). Computer Graphics: Principles and Practice (3rd ed.). Addison-Wesley. https://www.pearson.com/en-us/subject-catalog/p/computer-graphics-principles-and-practice/P200000004370