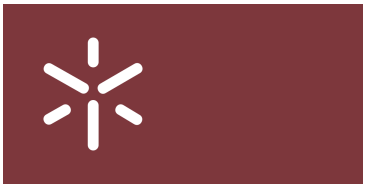

Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Computação Gráfica

Ano Letivo de 2024/2025

Motor de Mini Cenas 3D Baseado em Grafos

Alex Sá a104257
José Vasconcelos a100763
Paulo Ferreira a96268
Rafael Fernandes a104271

30 de Março, 2025

CG

Resumo

A elevada utilização de dispositivos eletrónicos e computacionais na atualidade obriga cada vez mais à necessidade de implementação de software e hardware cada vez mais sofisticados. Uma das áreas onde podemos observar isso a acontecer é na área de computação gráfica, que já se demonstra presente em diversos aspetos do quotidiana, seja em aplicações CAD/CAM, filmes, jogos, interfaces e muito mais. Neste relatório será apresentado o processo de desenvolvimento de uma *engine* utilizando os conceitos mais básicos que existem por de trás destas aplicações. Aqui será descrito a segunda de quatro fases, sendo que esta terá maior incidência na implementação das transformações geométricas sobre as primitivas, assim como no inicio do desenvolvimento de uma cena 3D baseada no sistema solar e no inicio da implementação de uma GUI. Na fase final, já será apresentada uma demo, com iluminação, texturas e animações implementadas.

Área de Aplicação: Computação Gráfica

Palavras-Chave: OpenGL; C++; 3D Engine

Índice

1. Introdução

No âmbito da unidade curricular de Computação Gráfica, foi proposta a realização de um projeto prático que visa o desenvolvimento de um motor de mini cenas em três dimensões (3D), baseado em grafos. Este projeto, devido ao seu caráter progressivo, foi estruturado em quatro fases distintas, cada uma com objetivos específicos e complementares. A corrente fase (segunda) concentrou-se no desenvolvimento das **transformações geométricas**, sendo as duas seguintes fases dedicadas, respetivamente, à modelação de **curvas, superfícies cúbicas** e à utilização de *Vertex Buffer Objects* (**VBOs**); e, por fim, à implementação de normais e coordenadas de texturas, sendo estes elementos essenciais para a criação de cenas visualmente apelativas.

O presente relatório tem como objetivo descrever e analisar o processo de desenvolvimento da segunda fase do projeto, desde a sua conceção teórica até à sua concretização prática. Nesta etapa, dedicou-se especial atenção à criação hierárquica de cenas através de transformações geométricas.

Através deste trabalho, pretende-se não apenas consolidar os conceitos teóricos abordados no âmbito da unidade curricular, mas também demonstrar a sua aplicação prática no desenvolvimento de ferramentas gráficas robustas e eficientes. A execução deste projeto permitiu, ainda, uma maior familiarização com tecnologias e técnicas essenciais no domínio da computação gráfica, tais como o **OpenGL**, que desempenha um papel central na renderização de cenas 3D.

2. Transformações Geométricas

Quando se trabalha com modelos, a sequência das transformações aplicadas é crucial. Cada grupo mantém a sua própria matriz de transformações, que engloba tanto as mudanças do grupo em questão quanto as dos seus subgrupos. No entanto, os subgrupos não têm acesso direto às transformações aplicadas aos grupos pais hierarquicamente acima. Para calcular as transformações finais, a matriz de cada grupo é multiplicada pela matriz das transformações anteriores, acumulando todas as alterações feitas ao longo da hierarquia. Isso garante que o modelo final seja posicionado e orientado corretamente, conforme as transformações acumuladas.

A título de exemplo, segue o conteúdo de um ficheiro *.xml*, que será utilizado para explicar como ocorre o processo das transformações geométricas:

```
<group>
   <transform>
      <translate x="0" y="1" z="0" />
   </transform>
   <group>
       <transform>
           <rotate angle="180" x="1" y="0" z="0" />
           <scale x="0.2" y="0.5" z="0.2" />
       </transform>
       <models>
          <model file="sphere.3d" /> <!-- generator sphere 1 10 10 sphere.3d -->
       </models>
   </group>
</group>
```

`Translação (0,1,0):`

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

`Rotação 90º em torno do eixo X:`

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

`Escala (0.2, 0.5, 0.2):`

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 0.2 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.2 & 0 & 0 & 0 \\ 0 & 0 & -0.2 & 1 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2.1. Translação

Como se pode verificar no exemplo acima, ao aplicarmos uma translação a um grupo segundo um vetor arbitrário *(x, y, z)*, estamos a multiplicar a sua matriz correspondente pela seguinte matriz:

$$\begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Esta transformação desloca o objeto ao longo dos eixos *X, Y e Z*, adicionando os valores *x, y, z* às respetivas coordenadas dos vértices do objeto.
De modo a conseguirmos implementar esta transformação na prática, utilizamos a função `glTranslatef(x,y,z)` do *OpenGL*. Esta função multiplica a matriz de transformação atual pela matriz de translação, deslocando o modelo de forma dinâmica.

2.2. Rotação

Ao aplicar uma rotação segundo um eixo arbitrário e um dado ângulo a um grupo, estamos a multiplicar a sua matriz pela seguinte:

$$\begin{pmatrix} x^2 + (1 - x^2) * \cos(\theta) & x * y * (1 - \cos(\theta)) - z * \sin(\theta) & x * z * (1 - \cos(\theta)) + y * \sin(\theta) & 0 \\ y * x * (1 - \cos(\theta)) + z * \sin(\theta) & y^2 + (1 - y^2) * \cos(\theta) & y * z * (1 - \cos(\theta)) - x * \sin(\theta) & 0 \\ x * z * (1 - \cos(\theta)) - y * \sin(\theta) & y * z * (1 - \cos(\theta)) + x * \sin(\theta) & z^2 + (1 - z^2) * \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

No nosso projeto, utilizamos a função `glRotatef(ângulo,x,y,z)` do *OpenGL* para aplicar esta transformação aos objetos. Esta função multiplica a matriz de transformação atual pela matriz de rotação correspondente, permitindo a rotação em torno de qualquer eixo.

2.3. Escala

Ao aplicar uma escala segundo um vetor arbitrário *(x, y, z)* a um grupo, estamos a multiplicar a sua matriz pela seguinte:

$$\begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Esta transformação altera o tamanho do objeto ao longo dos eixos *X, Y e Z*.
Se `x = y = z`, a escala é **uniforme**, mantendo as proporções originais.
Caso contrário, a escala será **não uniforme**, podendo distorcer a forma do objeto.

Na prática, estamos a utilizar a função `glScalef(x,y,z)` do *OpenGL* para aplicar esta transformação aos objetos. Esta função multiplica a matriz de transformação atual pela matriz de escala, ajustando de forma dinâmica o tamanho dos modelos.

3. Sistema Solar

Um dos objetivos centrais desta fase foi a implementação de uma cena hierárquica que simula o sistema solar, aproveitando os conceitos de transformações geométricas e a estrutura em grafo desenvolvida. Para garantir precisão científica e riqueza de detalhes, utilizámos dados astronómicos disponibilizados pelo projeto Devstronomy.

A cena foi modelada como um grafo onde:

- O **Sol** atua como nó raiz, posicionado na origem do sistema de coordenadas.
- Os **planetas** são nós filhos do Sol, com transformações calculadas com base em distâncias médias ao Sol e raios equatoriais, mas pretende-se já num futuro próximo usar também as inclinações axiais e ajustar a posição aos períodos de translação.
- Os **Satélites naturais** são nós filhos dos respetivos planetas.

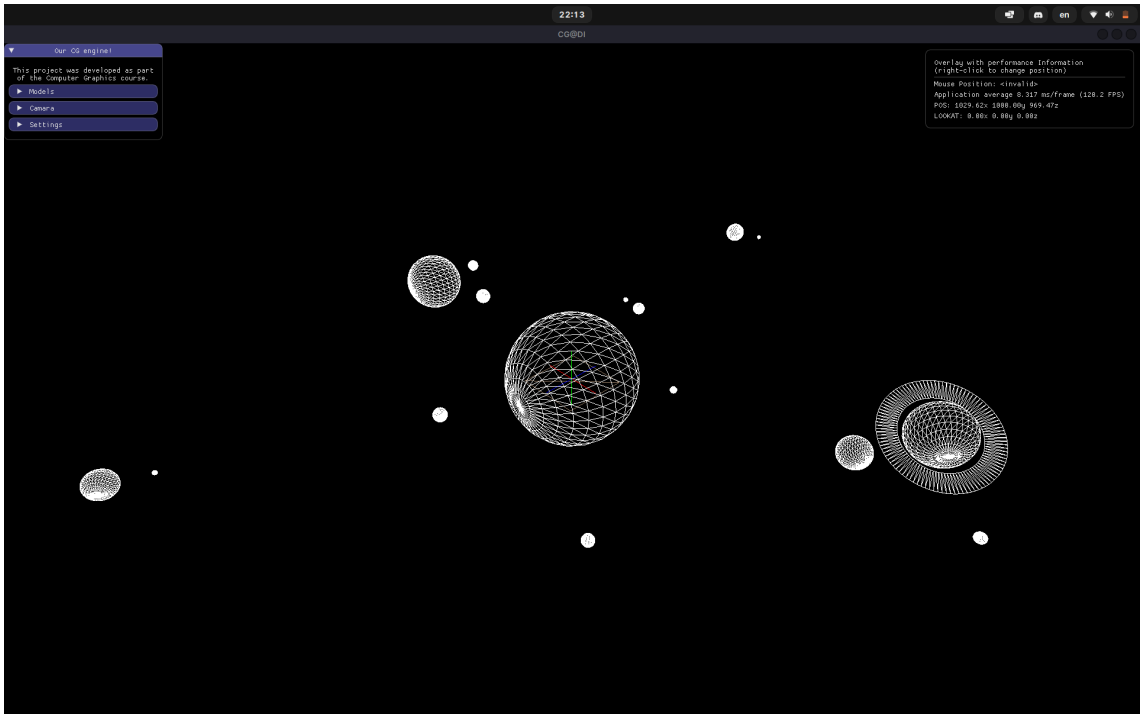

Figura 1: Scene do Sistema Solar

Esta abordagem não apenas demonstra a aplicação prática das transformações geométricas em hierarquias complexas, como também serve de base para futuras expansões (e.g., adição de asteroides, cometas, etc).

4. Testes

Para validar as diferentes funcionalidades desenvolvidas nesta fase, foi conduzida uma série de testes, cujos resultados são apresentados de seguida.

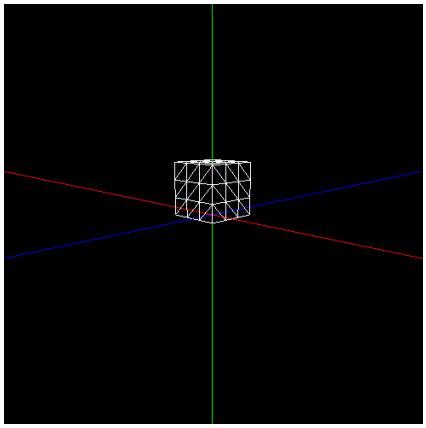

Figura 2: Teste nº 1: Cubo

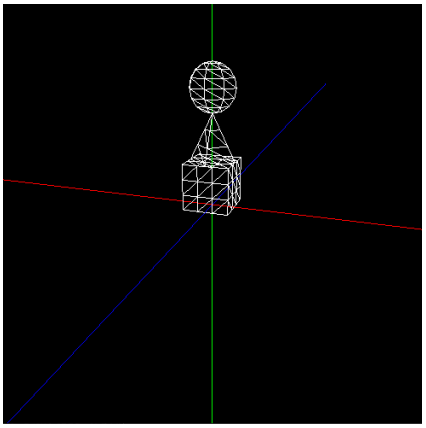

Figura 3: Teste nº 2: Cubo, Cone e Esfera

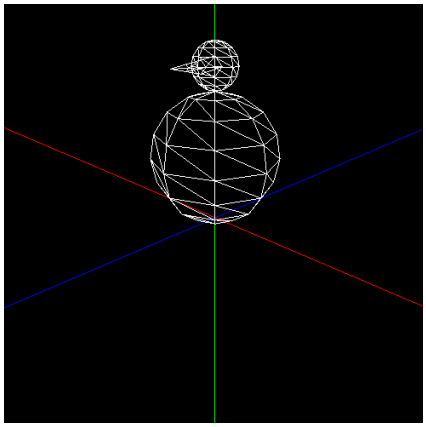

Figura 4: Teste nº 3: Boneco de neve

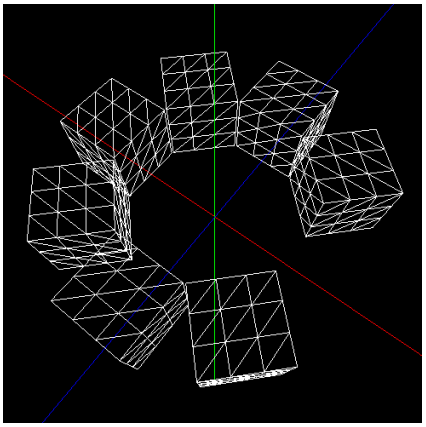

Figura 5: Teste nº 4: Cubos à volta do eixo

5. Interface Gráfica (GUI)

Um dos passos adicionais realizados nesta fase foi a implementação de uma **interface gráfica em modo Imediato(imGUI)** para facilitar a interação e depuração durante o desenvolvimento. Para agilizar o processo e evitar um esforço excessivo na programação manual de algo que não é o foco do desenvolvimento, optou-se pela integração da biblioteca **Dear ImGui** (disponível em https://github.com/ocornut/imgui). A biblioteca foi incorporada diretamente no projeto através da cópia dos seus ficheiros fonte, permitindo uma utilização eficiente e garantindo controlo da mesma.

Para fazer a integração com a *engine* foi desenvolvido um módulo dedicado, `EngineUI`, responsável por gerir todas as janelas e variáveis expostas na interface. Este módulo comunica diretamente com o estado interno da aplicação, possibilitando a modificação em tempo real de parâmetros como:

- **Configuração do GLUT**: Ativar/desativar ecrã inteiro, alternar entre modos de renderização(Wireframe, Filled e Points).
- **Controlo de câmara**: Ajustar posição, rotação e velocidade de movimento.
- **Métricas de depuração**: Monitorar a taxa de frames (FPS), coordenadas de objetos e outros dados em tempo de execução.

A estrutura modular da *GUI* foi pensada para permitir futuras expansões, como a adição de editores de hierarquia de cena ou propriedades de materiais.

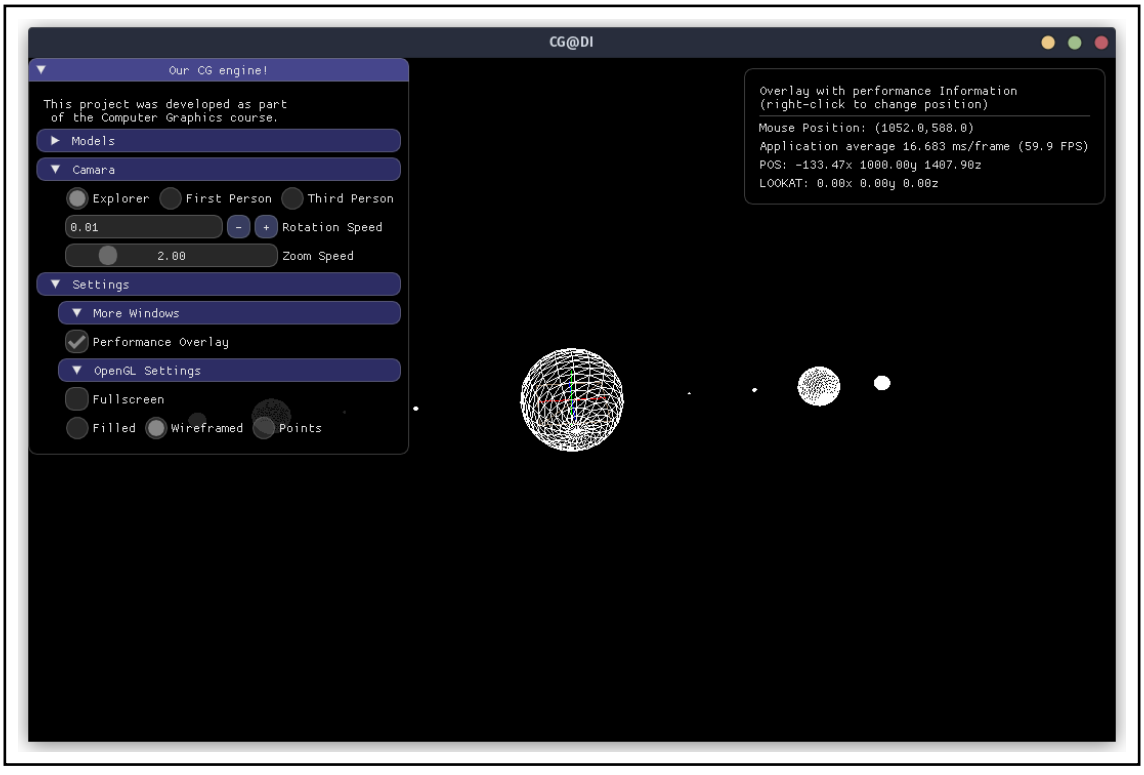

Figura 6: Primeira implementação da GUI.

6. Carregamento ficheiros .obj

Para além do conteúdo adicional mencionado no tópico anterior, o grupo de trabalho decidiu aprimorar a *engine* da aplicação, incorporando ao *parser* a funcionalidade de leitura de arquivos OBJ, isto é, arquivos padronizados que possuem armazenados dados de geometria 3D. Assim, a nossa aplicação passa a suportar novos modelos criados, por exemplo, em ferramentas como o *Blender*, levando à criação de cenas diversificadas, ampliando, desta forma, as possibilidades de uso e integração da *engine*.

Apesar do *parser* de ficheiros OBJ já incorporar o uso de normais e coordenadas de texturas, essas funcionalidades ainda não estão a ser utilizadas durante esta fase do projeto, já que a estrutura de dados que armazena a geometria de um modelo não possui os campos necessários para guardar essas informações. Desta forma, essa alteração será realizada nas próximas fases do projeto, uma vez que essas funcionalidades são requisitos para as próximas etapas.

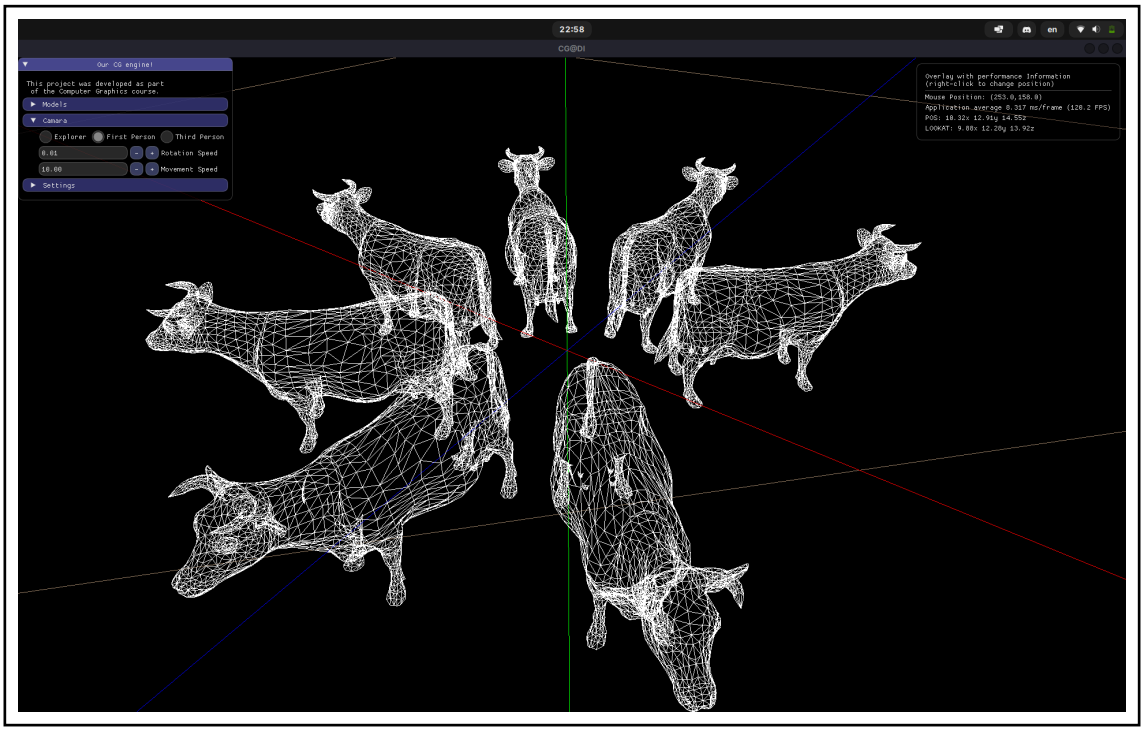

Figura 7: Carregamento de um ficheiro .obj utilizado numa cena de teste.

7. Conclusão e trabalho futuro

Esta segunda fase do projeto representou um avanço significativo no desenvolvimento da *engine*, consolidando os conceitos fundamentais de Computação Gráfica através da implementação prática de:

- Transformações geométricas hierárquicas (translação, rotação e escala)
- Modelação científica do sistema solar com dados astronómicos reais
- Integração de uma GUI funcional usando Dear ImGui
- Parser para ficheiros .obj com capacidade de expansão futura

Já para as próximas fases, planeia-se:

- A expansão do sistema solar com asteroides e cometas
- A otimização com VBOs/VAOs
- Integração completa de normais/UVs dos modelos .obj
- Desenvolvimento de editor de cenas na GUI

Este projeto continua a demonstrar o potencial da computação gráfica como ferramenta interdisciplinar, combinando conhecimentos de programação e matemática numa aplicação prática.