Computação Gráfica (3º ano de LCC)

4ª Fase

Relatório de Desenvolvimento

Grupo 18

Eduardo Pereira	Diogo Coelho	Pedro Oliveira	João Barbosa
(A70619)	(A100092)	(A97686)	(A100054)

18 de maio de 2025

Resumo

Este relatório descreve o desenvolvimento da quarta e última fase de um projeto, desenvolvido no âmbito da Unidade Curricular de Computação Gráfica, do 3º ano da Licenciatura em Ciências da Computação, na Universidade do Minho.

Este projeto consiste na implementação de duas grandes componentes: um **generator** e um **engine**, que em conjunto produzem figuras geométricas visíveis num espaço 3D.

O **generator** é responsável pela geração dos vértices relativos a cada figura e das coordenadas de textura, enquanto o **engine** trata da renderização dos objetos no espaço.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

No âmbito da Unidade Curricular de Computação Gráfica, foi-nos proposto, pelo docente, a implementação de um sistema capaz de desenhar figuras num espaço 3D.

Esta quarta fase do projeto tem como objetivo dar continuidade ao trabalho desenvolvido nas fases anteriores, introduzindo agora iluminação e a aplicação de texturas aos objetos. Estas alterações também têm a vantagem de trazer mais realismo ao Sistema Solar que já foi apresentado nas etapas anteriores.

Neste relatório será apresentada a nossa interpretação do problema, assim como as abordagens adotadas ao longo da implementação deste projeto.

1.1 Estrutura do Relatório

De forma a facilitar a leitura e compreensão deste documento, nesta secção será explicada a sua estrutura e o conteúdo de cada um dos capítulos, resumido e explicado.

- **Capítulo 1** - Definição do Sistema, que consiste em: Introduzir e descrever a contextualização do projeto desenvolvido, assim como os objetivos a atingir com o mesmo.
- **Capítulo 2** - Análise e Especificação: Descrever informalmente o problema, indicando o que se espera ser possível fazer.
- **Capítulo 3** - Descrição da implementação do `Generator`.
- **Capítulo 4** - Descrição da implementação do `Engine`.
- **Capítulo 5** - Apresentação e explicação da implementação do Sistema Solar Dinámico.
- **Capítulo 6** - Instruções de Utilização: Instruções para compilar e executar os programas do projeto
- **Capítulo 7** - Testes: Neste capítulo são apresentados diversos testes realizados sobre o nosso programa
- **Capítulo 8** - Conclusão: Contém as últimas impressões acerca do projeto desenvolvido e oportunidades de trabalho futuro.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

Nesta fase de implementação foram trabalhadas ambas as vertentes do projeto, `Generator` e `Engine`. O `Generator` evoluiu para possibilitar a geração de coordenadas de textura, que são essenciais para a correta aplicação da textura.
O `Engine` sofreu alterações para suportar a iluminação dos objetos, assim como a aplicação das texturas. Tal como nas fases anteriores, o `Engine` utiliza os ficheiros gerados pelo `Generator`.
As primitivas gráficas que temos até este ponto, são:

- **Plano**: Um quadrado no eixo **XZ**, centrado na origem, subdividido nas direções **X** e **Z**.
- **Caixa**: Requer **dimensão** e o número de **slices** por aresta, e está centrada na origem.
- **Esfera**: Requer um **raio**, **slices** e **stacks**, e está centrada na origem.
- **Cone**: Requer um **raio** para a base, **altura**, **slices** e **stacks**, e a base do cone está no eixo **XZ**.
- **Torus**: Definido por **dois raios** e o número de **rings** e **sides**.

O `Engine` foi ampliado para suportar iluminação e texturas, com novas funcionalidades no ficheiro `XML` e integração de um módulo auxiliar para os cálculos matemáticos necessários (`textures.cpp`).
Ambos os programas foram desenvolvidos em `C++`.

Capítulo 3

Generator

3.1 Implementação do Suporte a Texturas

Nesta fase do projeto, foi adicionada ao `Generator` a capacidade de gerar coordenadas de textura. Estas irão possibilitar a integração de texturas nos nossos modelos, de forma a torná-los mais realistas.
É importante que esta implementação esteja feita com precisão e rigor, pois se estas coordenadas não tiverem os valores esperados, as texturas inseridas irão ter um comportamento estranho, e perde-se o realismo esperado.

Cada uma das primitivas disponíveis tem uma forma de geração diferente, pelo que as fórmulas utilizadas para calcular cada uma destes grupos de coordenadas diferem entre si.

Na fase anterior, os modelos gerados incluíam apenas vértices, nesta fase 4, foram adicionadas coordenadas de textura e normais para todas as primitivas geométricas já disponíveis.
As coordenadas de textura foram calculadas com base na posição relativa dos vértices dentro da primitiva, garantindo um mapeamento preciso para a aplicação de texturas.

Foi introduzida a struct `VertexFull` para suportar a inserção de texturas através das variáveis **u** e **v**. Esta struct também guarda as normais através das variáveis **nx**,**ny** e **nz**. Também foi introduzida a função `writeVerticesFull()` para escrever os valores dos vértices, normais e coordenadas de textura nos ficheiros.

Estas coordenadas são fundamentais para mapear corretamente a posição da textura sobre a superfície do modelo.

Capítulo 4

Engine

O **Engine** foi mais uma vez a componente mais alterada nesta fase da implementação. Mais uma vez, esta processa os modelos tridimensionais a partir das especificações de um ficheiro `XML`, aplica as transformações esperadas e efetua o desenho da cena.
A renderização é realizada pela função **`renderScene()`**, que limpa o ecrã, configura a câmera, desenha os modelos em **wireframe** e troca os **buffers** para exibir uma cena nova.

4.1 Implementação de Iluminação

Nesta Fase 4, foi introduzido o suporte a propriedades de material para os modelos 3D, permitindo uma interação mais realista com as fontes de luz. Cada modelo agora pode ter propriedades como:

- **Difusa:** Luz refletida de forma uniforme em todas as direções
- **Ambiente:** Representa a luz ambiente presente na cena
- **Especular:** Reflexos brilhantes na superfície do objeto
- **Emissiva:** Luz própria emitida pelo objeto
- **Brilho (Shininess):** Intensidade da luz especular

Estas propriedades são configuradas no ficheiro XML e aplicadas no OpenGL usando a função `glMaterialfv()`.
O sistema de luzes suporta diferentes tipos de fontes de luz:

- **Luz Direcional:** Representa a luz proveniente de uma direção específica, por exemplo, a luz solar
- **Luz Pontual:** Emite luz em todas as direções a partir de um certo ponto
- **Spotlight:** Emite luz em forma de cone, com um ângulo de corte

Através do comando `glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ...)`, pode-se definir a intensidade da luz ambiente, que afeta todos os objetos na cena.

4.2 Implementação de Texturas

O carregamento das texturas é feito através da biblioteca DevIL, que permite ler diversos formatos de imagem. O código responsável por esta tarefa encontra-se na função `loadTexture()`. Esta função inicializa o **DevIL**, lê o ficheiro de imagem especificado, converte a imagem para formato **RGBA**, cria uma textura OpenGL e carrega os dados para a GPU através do comando `glTexImage2D()`.

No momento da renderização, para cada modelo que possui uma textura definida no ficheiro XML, o Engine faz o seguinte:

- Carrega a textura
- Ativa a textura correspondente através do `glBindTexture(GL_TEXTURE_2D, texID)`
- Ativa o mapeamento de texturas com o comando `glEnable(GLTEXTURE_2D)`

O OpenGL utiliza as coordenadas de textura fornecidas para mapear a imagem sobre a superfície do modelo em questão.
O Engine garante também que cada textura seja carregada apenas uma vez, reutilizando o identificador OpenGL (GLuint) sempre que necessário, otimizando assim o uso de memória e o desempenho.

4.3 Coordenadas de textura: Uso de VBO's

Cada modelo é carregado para a memória gráfica utilizando VBOs, que armazenam sequencialmente as posições dos vértices, as normais e as coordenadas de textura.
As coordenadas de textura são lidas dos ficheiros .3d, gerados pelo **Generator**, que agora incluem, para cada vértice, os valores de (u, v).
Os dados das coordenadas de textura são enviados para a GPU através de um buffer dedicado. O VBO das coordenadas de textura é ativado antes de desenhar o modelo.

A utilização de **VBOs** permite o envio eficiente de grandes quantidades de dados, aumentando assim o desempenho geral do programa.

4.4 Hierarquia de Transformações e da Cena

A estrutura hierárquica definida no `XML` é processada recursivamente pela função `renderGroup`. Cada grupo pode conter transformações geométricas (translação, rotação, escala) e modelos, com transformações aplicadas na ordem específica. Por exemplo, um planeta orbita o Sol (translação), gira sobre si mesmo (rotação) e pode ter luas como objetos filhos, que herdam as suas transformações.
Para translações temporizadas, o motor verifica a presença do atributo `time` e calcula a posição atual na curva Catmull-Rom. Se o parâmetro (align="True"), então a matriz de rotação é atualizada para alinhar o objeto à tangente da curva.
As transformações são acumuladas pelas funções `glPushMatrix` e `glPopMatrix`, funcionando como uma `stack`, garantindo assim que as operações em objetos filhos não afetem os pais.

Exemplo: Sistema Solar Dinâmico No ficheiro ‘dynamicsystem.xml‘, cada planeta é definido como um grupo com:

1. `Translação temporizada`: Uma curva `Catmull-Rom` com oito pontos, simulando órbitas.
2. `Rotação contínua`: Com recurso ao parâmetro `<rotate time="X»`, onde ‘X‘ é o período de rotação em segundos.
3. `Hierarquia de modelos`: Luas são grupos filhos com trajetórias e transformações próprias.

O cometa, por exemplo, utiliza uma curva aberta com quatro pontos e um modelo de `patch` de `Bézier` (*bezier*_10.3*d*), o que implica diversos requisitos pedidos nesta fase do projeto, demonstrando a flexibilidade do sistema para diferentes tipos de trajetórias e geometrias.

Capítulo 5

Sistema Solar Dinâmico

No enunciado deste projeto, também foi pedida uma demonstração de um Sistema Solar dinâmico, com os vários planetas, bem como as suas luas, expandindo do modelo estático.

Para este efeito, foi desenvolvido um ficheiro XML com as informações necessárias ao desenho:

- **Transformações geométricas:** Translações, rotações ou escalas
- **Modelos utilizados:**
  - **Sol:** sphere_10_20_20.3d
  - **Planetas:** sphere_10_20_20.3d
  - **Luas:** sphere_10_20_20.3d
  - **Anel de Saturno:** torus_15_14_20_20.3d

5.1 XML

No contexto deste projeto, foi desenvolvido um ficheiro `XML` que descreve um Sistema Solar dinâmico, onde cada planeta e as respetivas luas executam movimentos ao longo de curvas `Catmull-Rom`. A estrutura hierárquica do `XML` reflete a organização natural do sistema solar: o Sol ocupa o centro da cena, servindo de nó pai para todos os planetas (filhos) , e cada planeta pode ter as suas luas como nós-filho. As órbitas são parametrizadas através do atributo `time`, que define o período de translação de cada corpo , enquanto o parâmetro `(align="True")` garante que os objetos se mantêm orientados segundo a direção do movimento. Para maior clareza visual, é ainda possível ocultar a representação gráfica de certas órbitas com o atributo `(draw="false")`.

Nesta fase, o ficheiro `XML` também inclui a informação necessária para a correta inclusão da iluminação e da textura de cada corpo celeste.
O Sol é definido como uma fonte de luz, enquanto que os planetas e luas são afetados pela posição da fonte de luz.

5.2 Exemplo Sistema Solar

Em baixo está presente uma imagem da nossa interpretação do Sistema Solar pedido. Na pasta source do código está presente um pequeno vídeo que demonstra as etapas definidas até agora:

- translações e rotações dos planetas ao longo das órbitas definidas pelas curvas de `Catmull-Rom`
- iluminação a partir de uma fonte de luz (Sol)
- texturas de cada corpo celeste, definidas corretamente graças às coordenadas de textura

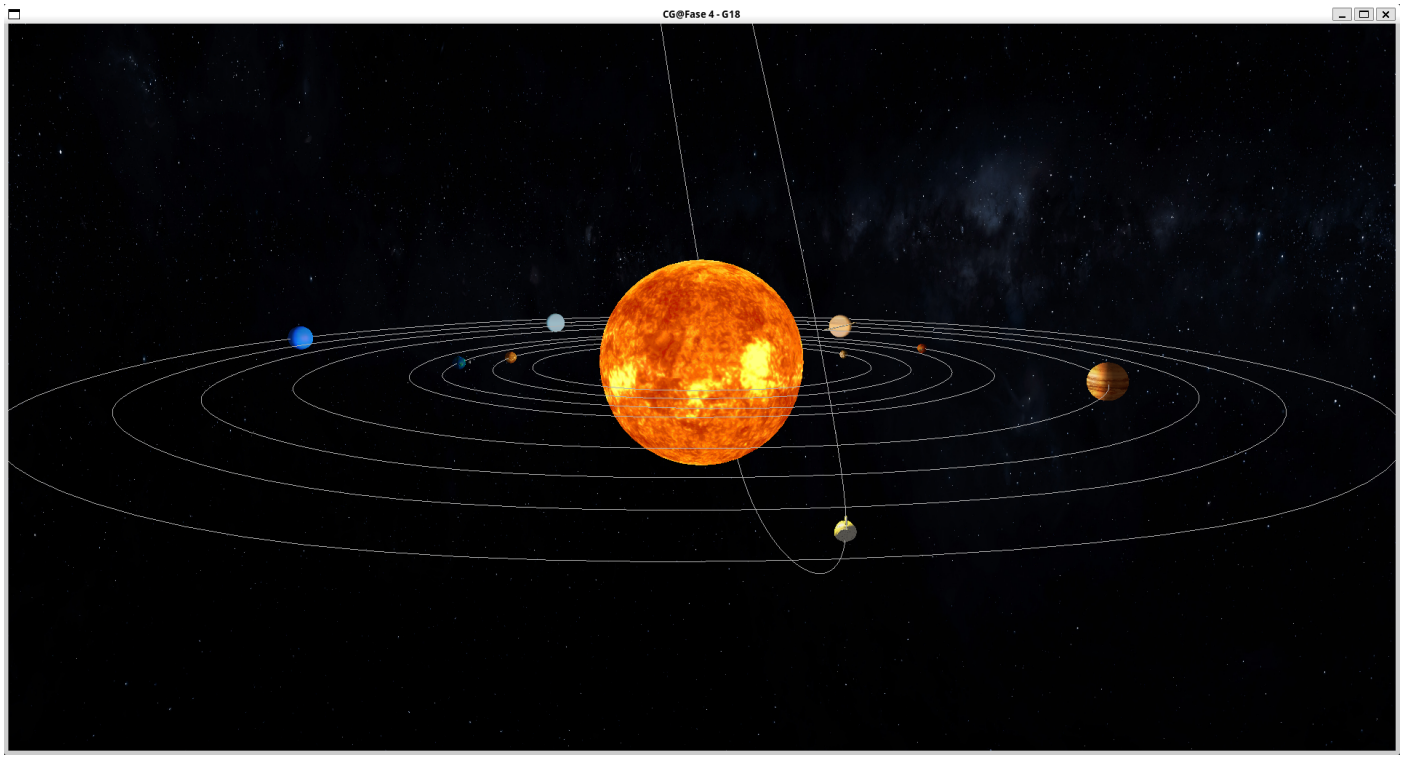

Figura 5.1: Sistema Solar

Capítulo 6

Instruções de Utilização

Para compilar e executar o nosso programa, existem duas formas:

6.1 Manualmente

1. Posicionar-se na pasta `build` com o comando:

   ```
   cd build
   ```

2. Executar os seguintes comandos:

   ```
   cmake ..
   ```

   seguido de

   ```
   make
   ```

3. Posicionar-se na pasta `generator` e executar o seguinte comando:

   ```
   ./generator nome_figura <parametros> nome_figura.3d
   ```

4. Sair da pasta `generator` com o comando:

   ```
   cd ..
   ```

5. Posicionar-se na pasta `build` e executar o seguinte comando:

   ```
   ./engine ../engine/configs/test_file_pretendido.xml
   ```

6.2 Script

Para simplificar o processo, criámos um **script** que executa automaticamente todas as instruções mencionadas. Para executar o **script**, siga os seguintes passos:

1. Tornar o **script** executável com o comando:

```
chmod +x run.sh
```

2. Executar o **script** com:

```
./run.sh
```

Após executar o **script**, será apresentado um menu que permite realizar diversas operações:

- Na primeira utilização, recomenda-se executar um `make clean` e recompilar o projeto. Para isso, seleciona-se a opção **1**.
- Criar objetos 3D utilizando a opção **2**.
- Visualizar os objetos já criados com a opção **3**.
- Executar os ficheiros de teste através da opção **4**.

Dentro da pasta do projeto, incluímos uma pequena demonstração para facilitar a compreensão do processo.

Capítulo 7

Testes

Nesta secção, serão apresentados alguns exemplos com as figuras renderizadas conforme as configurações dos arquivos de teste no formato **.XML**.

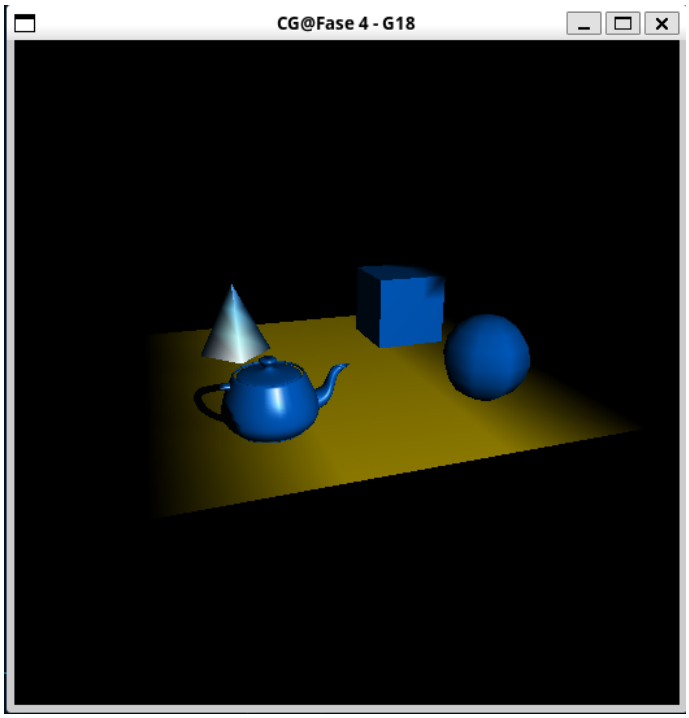

Figura 7.1: Teste 6

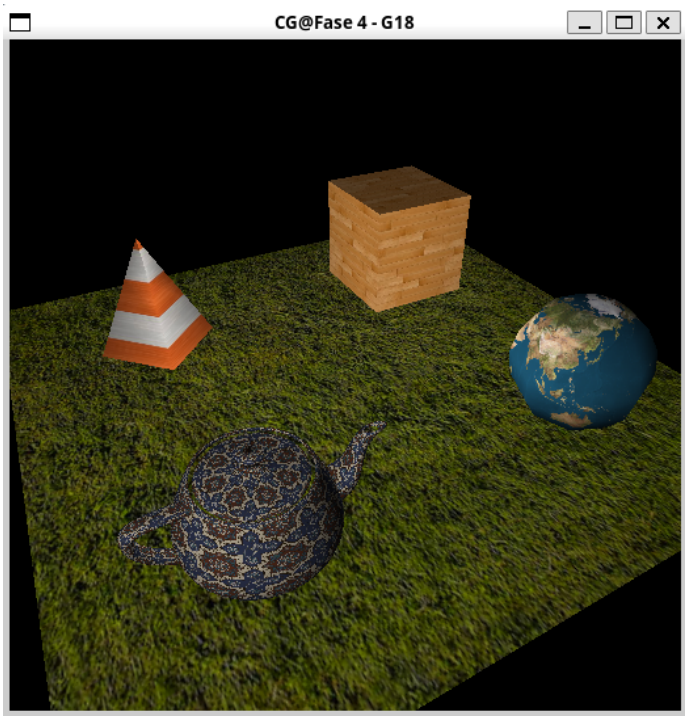

Figura 7.2: Teste 7

Capítulo 8

Conclusão

A quarta fase deste projeto proporcionou um avanço significativo em direção ao realismo desejado nas cenas desenhadas.
As novas funcionalidades constituem um marco importante neste projeto, com a implementação de iluminação e texturas, as geometrias previamente criadas ganharam um toque de realismo acrescido. Isto é mais evidente no caso do Sistema Solar, que realça a importância de ter uma implementação coerente, principalmente na geração das coordenadas de textura.
A utilização de VBOs em vez do modo imediato para a aplicação das texturas também trouxe grandes vantagens em termos de eficiência do programa.

Sentimos que atingimos com sucesso os principais objetivos propostos para esta fase, conseguindo integrar de forma funcional as novas `features`.
De um ponto de vista mais geral, pensamos que alcançamos os patamares esperados para este projeto, em parte até os completamos, com a inserção de um fundo estrelado para dar ainda mais realismo.

Contudo, reconhecemos que existe espaço para melhorias, nomeadamente na forma como se trabalha as texturas, atualmente, algumas texturas sofrem de um fenómeno conhecido como *Flickering*, que basicamente atribui um aspeto cintilante e trémulo à textura. Este problema tem a ver com o tamanho da textura face à superfície em que é aplicada e poderia ser corrigido com a aplicação de **MIP Mapping**.

Um ponto a melhorar também seria a organização do nosso código. Com o aumento da complexidade, em especial pela gestão de VBOs e pelas novas transformações temporizadas, tornou-se ainda mais evidente a necessidade de modularizar melhor a implementação do nosso código.

Em suma, pensamos que foi um projeto muito interessante, permitiu-nos utilizar vários conceitos dados ao longo do semestre, assim como aplicar outros conceitos e boas práticas aprendidos ao longo da licenciatura.
Além disso, é um projeto diferente, pois consegue-se ter uma componente gráfica associada a cada passo de desenvolvimento, que nem sempre se tem oportunidade de ver em desenvolvimento de programas.