Computação Gráfica (3º ano de LCC)

2ª Fase

Relatório de Desenvolvimento

Grupo 18

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

31 de março de 2025

Resumo

Este relatório descreve o desenvolvimento da segunda 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, 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 segunda fase do projeto tem como objetivo dar continuidade ao trabalho desenvolvido na primeira fase, expandindo a implementação de figuras geométricas e introduzindo novas funcionalidades. O foco desta etapa é a inclusão de uma nova figura, o `torus`, bem como a implementação de transformações geométricas, tais como translação, rotação e escala, que devem ser aplicadas antes da renderização das figuras, respeitando uma hierarquia definida.

Para validar a implementação, o projeto foi submetido a um conjunto de testes fornecidos, com o intuito de verificar se todas as funcionalidades estavam de acordo com os requisitos especificados. Por fim, foi solicitado o desenvolvimento de um ficheiro `XML` capaz de representar um sistema solar, utilizando as transformações geométricas para simular as relações espaciais entre os diferentes corpos celestes.

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.
- **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

Esta fase do projeto visa a implementação de cenas hierárquicas, utilizando transformações geométricas. Neste ponto, apenas o **Engine** será alterado significativamente, o **Generator** apenas é alterado para permitir a introdução de uma nova primitiva gráfica, o **Torus**, que terá utilidade para representar os anéis de Saturno.

Tal como na primeira fase, o **Generator** gera os ficheiros com os vértices a serem utilizados pelo **Engine**.
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**: Novo modelo geométrico adicionado, definido por **dois raios** e o número de **rings** e **sides**.

O **Engine**, interpreta um ficheiro **XML** que contém os parâmetros para a composição da cena a renderizar.

Esta cena define-se como uma árvore, em que cada `node` contém um grupo de transformações geométricas, podendo também incluir a indicação dos modelos a desenhar. Cada `node` desta árvore pode também ter mais `nodes` como filhos.
Para facilitar a leitura do ficheiro **XML**, utiliza-se a biblioteca **tinyXML2**. No entanto, nesta fase do projeto, o **Engine** sofreu modificações significativas, incluindo:

- Implementação das transformações geométricas (translação, rotação e escala) aplicadas antes da renderização, seguindo uma hierarquia predefinida.

- Aprimoramento da gestão da cena, garantindo que os objetos são desenhados corretamente e respeitam a ordem das transformações.
- Adição de um menu de debug, permitindo ao utilizador inspecionar os diferentes elementos da cena, visualizar as transformações aplicadas e testar o comportamento dos modelos geométricos de forma mais interativa.

Ambos os programas foram desenvolvidos em `C++`.

Capítulo 3

Generator

Este programa é responsável pela criação de representações tridimensionais de primitivas geométricas e guardá-las num ficheiro **.3d**. O gerador recebe como entrada um conjunto de parâmetros que especificam o tipo de figura geométrica a ser gerada, as suas dimensões e o nível de subdivisão da `mesh`. Atualmente, são suportadas cinco figuras primitivas: plano, caixa, cone, esfera e um torus. O gerador valida os parâmetros fornecidos, garantindo que são numéricos e semanticamente corretos.

A geração de figuras baseia-se na decomposição das superfícies em triângulos, utilizando um sistema de coordenadas cartesianas tridimensionais. Cada figura é descrita num conjunto de triângulos definidos por três vértices, guardados num ficheiro **.3d**. Este ficheiro contém as coordenadas dos vértices gerados, para, à posteriori, serem interpretados pelo motor gráfico.

Vamos apenas detalhar o algoritmo relativo ao **torus**, dado que foi a única alteração relativa ao generator.

3.1 Torus

O torus é definido por um raio exterior e um raio interior. A sua estrutura é composta por `rings` (anéis ao longo da circunferência principal) e `sides` (divisões ao longo da secção circular).

Para gerar a superfície do torus, calcula-se:

- O `ringRadius`, metade da diferença entre os raios exterior e interior.
- `O torusRadius`, soma do raio interior com o `ringRadius`.

A construção segue dois passos principais:

1. Para cada `ring`, calcula-se o ângulo e aplica-se uma rotação em torno do eixo Y.
2. Para cada `side`, geram-se pontos num círculo no plano XY e aplicam-se transformações para posicioná-los corretamente.

O modelo final é composto por quadriláteros formados por dois triângulos. A implementação utiliza transformações matriciais 4x4 para posicionar os vértices no espaço 3D.

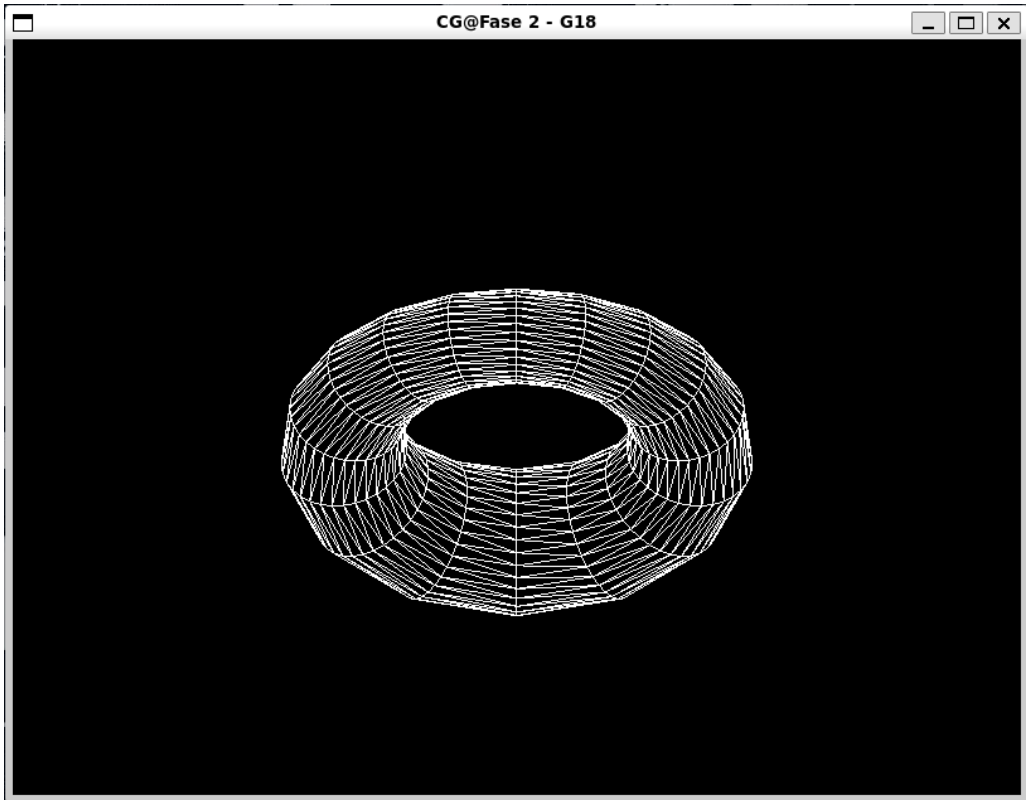

Figura 3.1: Torus gerado com parâmetros: **raio exterior** = 1, **raio interior** = 0.5, **rings** = 32 e **sides** = 16

Capítulo 4

Engine

O **Engine** foi 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 Parsing do ficheiro XML

Para implementar o suporte a transformações, com suporte a hierarquia de operações, desenvolvemos um sistema de **parsing** de `XML` que opera sobre a cena como se fosse um grafo de elementos. Durante o parsing do `XML`, mantemos a ordem de transformações armazenadas em cada ficheiro `XML`, armazenando-as num vetor `transformOrder`. Esta ordem das operações é fundamental para o resultado final, pois uma rotação seguida de uma translação produz um resultado diferente de uma translação seguida de uma rotação.

Para gerir a hierarquia de transformações recorremos a funções de pilha de matrizes do `OpenGL`, `glPushMatrix()` e `glPopMatrix()`.

Para renderizar a cena, utilizamos a função recursiva `loadModels()` que vai percorrer toda os grupos do ficheiro `XML` e carregar os vértices de cada modelo.

4.2 Transformações Matriciais

Na segunda fase do projeto, implementamos um sistema de transformações geométricas baseado em matrizes. Este sistema é fundamental para permitir a construção de cenas complexas com objetos inter-relacionados, como observado no sistema solar implementado nos testes.

As transformações geométricas são representadas através de matrizes 4x4 utilizando coordenadas homogéneas, o que permite combinar múltiplas operações de forma eficiente. Implementamos três tipos fundamentais de transformações:

- **Translação:** Permite mover objetos no espaço 3D usando um vetor de deslocamento (x, y, z).

- **Rotação:** Permite girar objetos em torno dos eixos coordenados. Implementamos rotações em torno dos eixos X, Y e Z.
- **Escala:** Permite redimensionar objetos ao longo dos eixos de coordenadas.

O nosso sistema suporta a combinação destas transformações, através da multiplicação de matrizes, permitindo aplicar sequências de operações aos objetos.
Isto é particularmente importante para criar hierarquias de transformações, onde um objeto-filho herda as transformações do seu pai.
Por exemplo, no sistema solar, as luas herdam as transformações do planeta que estão a orbitar, que por sua vez herda as transformações do Sol.

Para facilitar a execução destas transformações, implementamos estas funções para cada uma delas:

- `identityMatrix()`: Cria uma matriz identidade 4x4
- `translationMatrix(x,y,z)`: Gera uma matriz de translação
- `scaleMatrix(x,y,z)`: Gera uma matriz de escala
- `rotationMatrixX/Y/Z(angle)`: Gera matrizes de rotação em torno dos eixos
- `multiplyMatrices(a,b)`: Multiplica duas matrizes 4x4
- `transformVertex(v,m)`: Aplica uma transformação a um vértice

4.3 Movimentação da Câmara e Menu de *Debug*

4.3.1 Movimentos da Câmara

Nesta fase do projeto, foi implementado um sistema de controle de câmara que permite ao utilizador navegar livremente pelo ambiente 3D. A navegação baseia-se em num sistema de movimento linear e num sistema de movimento orbital. O sistema de movimento linear está implementado num esquema de teclas WASD, em que (W) move-se para a frente, (S) para trás, e lateralmente com (A,D), tudo com base na orientação da câmara. Implementamos também o movimento vertical com as teclas (Q) e (E).
Para a rotação da câmara, implementamos um sistema baseado em coordenadas esféricas, onde a posição da camara é definida por 3 parametros: raio (distancia ao ponto focal), ângulo (rotação horizontal) e angulo (elevação). A função `updateOrbit()` recalcula a posição da camara baseada nestes parametros esféricos.
Para controlar o modelo orbital, implementamos dois métodos: rato e teclado. As teclas IJKL permitem ajustar os angulos, proporcionando uma rotação em torno do ponto focal. As teclas + e - ajustam o zoom, ajustanto o raio da órbita.
Para assegurar uma experiência consistente, implementámos limitações nos ângulos e distâncias. O ângulo beta (elevação) é restrito ao intervalo de -89° a 89°, evitando inversões da câmara. Da mesma forma, o raio orbital tem limites mínimo e máximo para prevenir problemas de visualização associados a distâncias extremas.

4.3.2 Menu de *Debug*

Implementámos também uma interface de *Debug*, ativada através da tecla H. Esta interface oferece feedback imediato sobre o estado do sistema, apresentando informações sobre a cena 3D em questão.
A implementação da interface solicitou a criação de um sistema para renderizar texto na tela. A função `renderText()` manipula temporariamente as matrizes de projeção e modelação para estabelecer um contexto de desenho 2D, permitindo posicionar texto no ecrã.
A interface de *Debug* apresenta as seguintes informações:

- **Taxa de Frames por Segundo (FPS)**: Um indicador do desempenho da renderização, calculado a cada segundo pela contagem de `frames` renderizados no intervalo de tempo.
- **Posição da Câmara**: As coordenadas cartesianas exatas (x, y, z) da câmara no espaço 3D.
- **Ponto de Observação**: As coordenadas do ponto para o qual a câmara está a apontar (lookAt).
- **Parametros Orbitais**: Os valores atuais de raio, ângulo alfa e ângulo beta.
- **Contagem de Modelos**: O número total de modelos na cena, obtido recursivamente através da função `countModels()` que percorre a estrutura hierárquica de grupos.

Para complementar os controlos de navegação, implementámos funções adicionais. A tecla R funciona como um "`reset`", reposicionando a câmara para as coordenadas iniciais e reinicializando os parâmetros orbitais. A tecla F alterna entre o modo `wireframe` (esqueleto) e o modo preenchido. Finalmente, a tecla ESC proporciona uma forma conveniente de encerrar a aplicação.

Capítulo 5

Sistema Solar

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

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

O ficheiro XML foi desenvolvido com o objetivo de ser fácil de compreender e modificar, caso necessário.
O Sol funciona como o pai principal, pois, sendo este o centro do Sistema Solar, não depende de nenhum outro corpo.
Todos os planetas seguem este conceito, ou seja, dado que é suposto desenhar todos em torno do Sol, então considera-se que todos são objetos-filho do mesmo. Da mesma forma, as luas são objetos-filho dos seus respectivos planetas.

De acordo com este plano, o Sol é o primeiro a ser desenhado, no centro do plano. Para todos os planetas, é aplicada uma **rotação**, seguida de uma **translação** e ainda de uma **escala**.

- **Rotação:** Feita em torno do eixo y, de forma aos planetas ficarem distribuídos em volta do Sol
- **Translação:** Para respeitar a distância de cada planeta em relação ao Sol

- **Escala:** Transformação importante para garantir que os planetas ficam com o tamanho correto em relação ao Sol e aos outros planetas

No caso das luas, como estas seguem uma órbita do respectivo planeta em vez do Sol, é feita mais uma translação para colocar as luas na sua posição, seguida de uma escala para lhes atribuir o seu tamanho correto.

Em relação ao anel de Saturno, apenas é preciso garantir que o raio menor do Torus é superior ao raio do planeta. Por isso, este sofre apenas uma alteração em relação ao planeta, que é uma rotação em torno de x e de z, de forma a ”inclinar”este objeto de forma a parecer mais real.

```xml
<group>
    <group> <!-- SUN -->
        <transform>
            <scale x="5.5" y="5.5" z="5.5" />
        </transform>
        <models>
            <model file="sphere_10_20_20.3d" /> <!-- ./generator sphere 10 20 20 sphere_10_20_20.3d -->
        </models>
    </group>
    <group> <!-- MERCÚRIO -->
        <transform>
            <rotate angle="45" x="0" y="1" z="0" />
            <translate x="45" y="0" z="45" />
            <scale x="0.23" y="0.23" z="0.23" />
        </transform>
        <models>
            <model file="sphere_10_20_20.3d" /> <!-- ./generator sphere 10 20 20 sphere_10_20_20.3d -->
        </models>
    </group>
    <group> <!-- VÉNUS -->
        <transform>
            <translate x="55" y="0" z="55" />
            <scale x="0.32" y="0.32" z="0.32" />
        </transform>
        <models>
            <model file="sphere_10_20_20.3d" /> <!-- ./generator sphere 10 20 20 sphere_10_20_20.3d -->
        </models>
    </group>
    <group> <!-- TERRA -->
        <transform>
            <rotate angle="17" x="0" y="1" z="0" />
            <translate x="65" y="0" z="65" />
            <scale x="0.35" y="0.35" z="0.35" />
        </transform>
        <models>
            <model file="sphere_10_20_20.3d" /> <!-- ./generator sphere 10 20 20 sphere_10_20_20.3d -->
        </models>
        <group> <!-- LUA -->
            <transform>
                <translate x="20" y="20" z="20" />
                <scale x="0.23" y="0.23" z="0.23" />
            </transform>
                <models>
                    <model file="sphere_10_20_20.3d" /> <!-- ./generator sphere 10 20 20 sphere_10_20_20.3d -->
                </models>
        </group>
    </group>
```

Figura 5.1: Excerto do ficheiro XML para o Sistema Solar

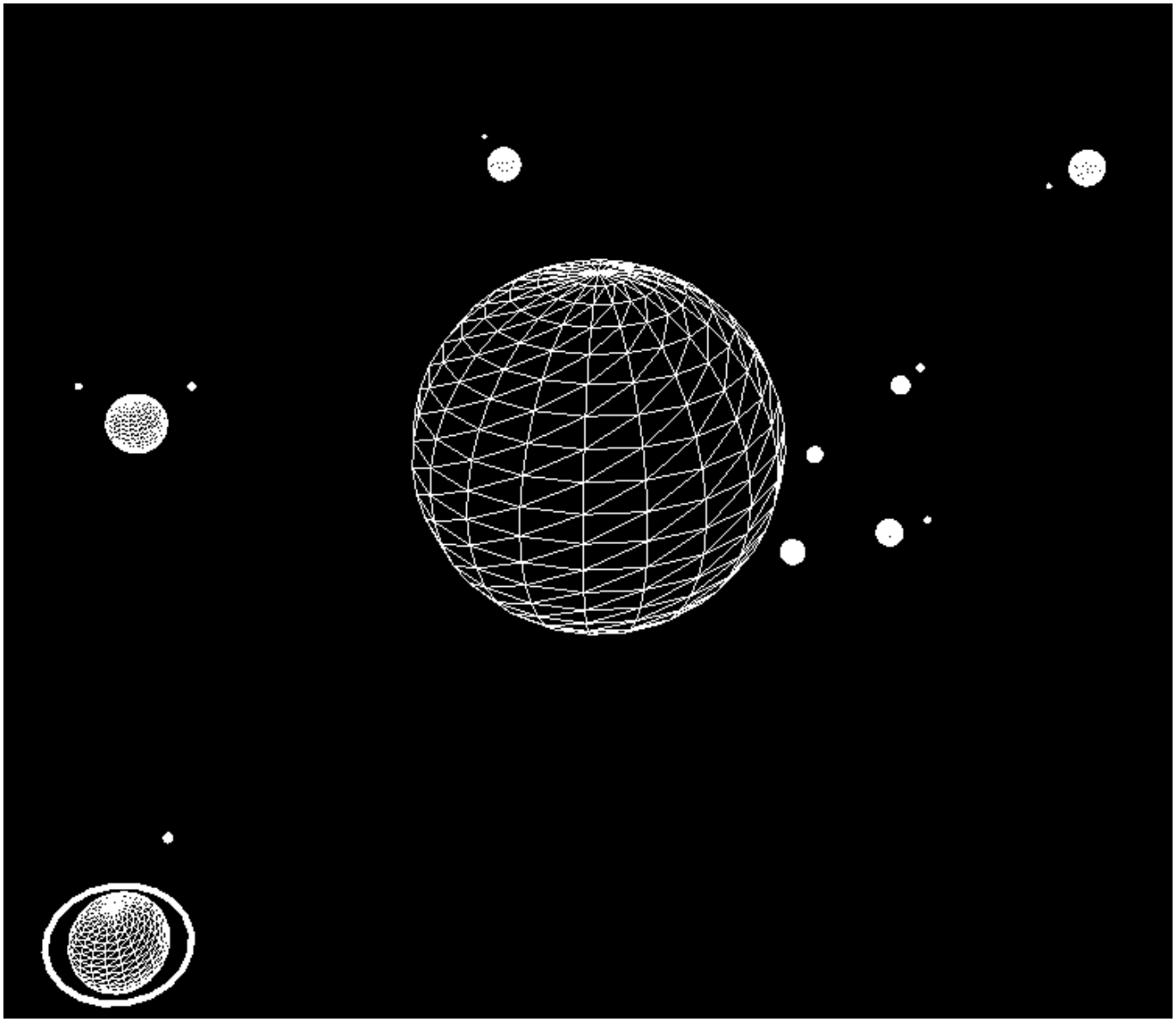

Figura 5.2: Desenho do Sistema Solar

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**.

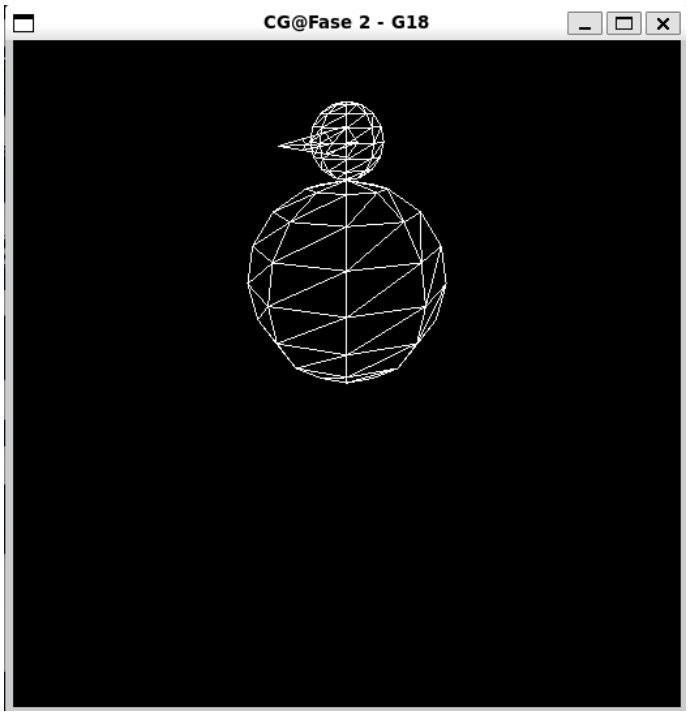

Figura 7.1: Teste 3

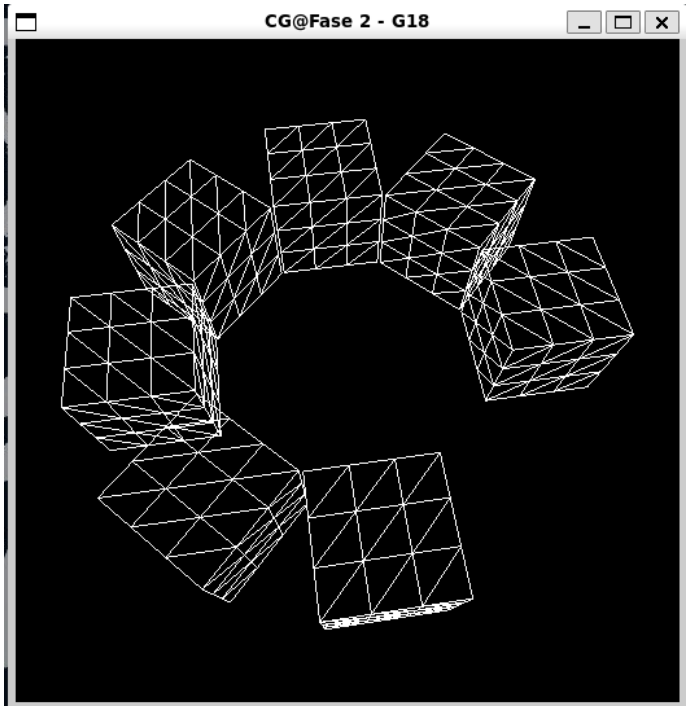

Figura 7.2: Teste 4

Capítulo 8

Conclusão

A segunda fase deste projeto representou um avanço significativo em relação à primeira, expandindo substancialmente as capacidades do sistema e proporcionando uma experiência muito mais rica e interativa para o utilizador.

A implementação do **Torus** como nova primitiva geométrica complementou perfeitamente o conjunto já existente de figuras, permitindo a criação de estruturas como o anel necessário para desenhar o planeta Saturno.

Os dois pontos mais interessantes nesta fase foi a utilização de matrizes para fazer transformações, que se revelou bastante vantajosa, e a capacidade de aninhar estas transformações de forma hierárquica, que permitiu-nos criar transformações mais complexas como, por exemplo, a criação do Sistema Solar.

A implementação dos movimentos de câmara através do teclado/rato e o menu de debug também foram ferramentas que ajudaram a perceber o que estava a ser bem desenhado ou não, pois permitiu-nos observar mais detalhadamente as figuras desenhadas. Além disso, são uma mais valia para um utilizador poder navegar por imagens mais complexas.

Pensamos que conseguimos atingir todos os objetivos desta fase de desenvolvimento, mas pode haver espaço para melhoria. Um dos aspetos que achamos que poderiamos melhorar, era a melhor organização e distribuição das funções para não ser tão confuso de navegar o nosso código de implementação.