

Computação Gráfica (3º ano de LCC)
2ª Fase
Relatório de Desenvolvimento
Grupo 18

Eduardo Pereira
(A70619)

Diogo Coelho
(A100092)

Pedro Oliveira
(A97686)

João Barbosa
(A100054)

26 de março de 2025

Resumo

Este relatório descreve o desenvolvimento da primeira 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 que o **engine** trata da renderização dos objetos no espaço.

Conteúdo

1	Introdução	3
1.1	Estrutura do Relatório	4
2	Análise e Especificação	5
2.1	Descrição informal do problema	5
2.2	Proposta de resolução	5
2.2.1	Generator	5
2.2.2	Exemplos Figuras Geradas	7
2.2.3	Engine	10
3	Instruções de Utilização	11
3.1	Manualmente	11
3.2	Script	12
4	Testes	13
5	Conclusão	18

Lista de Figuras

2.1	Excerto de um ficheiro .3d gerado	6
2.2	Plano gerado com parâmetros: unit = 1, slices = 3	7
2.3	Esfera gerada com parâmetros: radius = 1, slices = 10, stacks = 10	8
2.4	Caixa gerada com parâmetros: unit = 2, slices = 3	9
2.5	Cone gerado com parâmetros: radius = 1, height = 2, slices = 4, stacks = 3	10
3.1	Script - Menu	12
4.1	Teste 1 - XML	13
4.2	Teste 1 - Desenho	13
4.3	Teste 2 - XML	14
4.4	Teste 2 - Desenho	14
4.5	Teste 3 - XML	15
4.6	Teste 3 - Desenho	15
4.7	Teste 4 - XML	16
4.8	Teste 4 - Desenho	16
4.9	Teste 5 - XML	17
4.10	Teste 5 - Desenho	17

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.

O objetivo desta primeira fase é desenvolver um sistema para gerar figuras geométricas de forma que estes objetos sejam desenhados com base nas configurações de uma câmara definidas num ficheiro **XML**.

Este projeto é desenvolvido utilizando a linguagem **C++** e recorrendo à ferramenta **OpenGL** para o desenho das várias figuras geométricas.

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 seção 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** - Instruções de Utilização: Instruções para compilar e executar os programas do projeto
- **Capítulo 4** - Testes: Neste capítulo são apresentados diversos testes realizados sobre o nosso programa
- **Capítulo 5** - 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 trabalho requer a criação de dois programas para a renderização de modelos geométricos. O primeiro programa, **Generator**, tem como função gerar ficheiros que contêm os vértices das primitivas gráficas que vão ser utilizadas pelo outro programa **Engine**. Recebe como parâmetros os valores necessários para a criação dos modelos geométricos e armazena num ficheiro os vértices dos triângulos que definem as figuras. Com o gerador é possível obter os vértices dos triângulos para gerar as seguintes figuras:

- **Plano:** Um quadrado no eixo XZ, centrado na origem, subdividido nas direções X e Z.
- **Caixa:** Requer dimensão e o número de divisões 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.

O segundo programa, **Engine**, interpreta um ficheiro XML que contém os parâmetros para a composição da cena a renderizar, incluindo também a indicação dos modelos a desenhar. Deve ser utilizada a biblioteca `tinyXML2` para facilitar a leitura do ficheiro XML.

Ambos os programas devem ser feitos em C++.

2.2 Proposta de resolução

2.2.1 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 malha. Atualmente, são suportadas quatro figuras primitivas: plano, caixa, cone e esfera. 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.

```
1      -1 1 -1      You, 27 mir
2      -0.333333 1 -0.333333
3      -0.333333 1 -1
4      -1 1 -1
5      -1 1 -0.333333
6      -0.333333 1 -0.333333
7      -1 1 -0.333333
8      -0.333333 1 0.333333
9      -0.333333 1 -0.333333
10     -1 1 -0.333333
```

Figura 2.1: Excerto de um ficheiro **.3d** gerado

Cada primitiva geométrica segue um algoritmo específico para a sua criação:

- **Plano**: é subdividido em quadrados com base no número de divisões especificado (**slices**). Cada quadrado é representada por dois triângulos.
- **Caixa**: composta por seis faces, cada uma subdividida de forma semelhante ao plano.
- **Cone**: gerado a partir de um círculo base e de um vértice superior, subdividido em fatias (**slices**) ao longo da circunferência e em camadas (**stacks**) ao longo da altura.
- **Esfera**: construída utilizando coordenadas esféricas, onde os meridianos (**slices**) e paralelos (**stacks**) são utilizados para definir os vértices e os triângulos que compõem a superfície esférica.

A implementação do gerador está dividida em dois ficheiros principais:

- **generator.cpp**: Responsável pelo **parsing** dos argumentos, seleção da figura a gerar e invocação das funções adequadas para cada tipo de primitiva.
- **generatorAux.cpp**: Contém as funções específicas para a geração das diferentes figuras geométricas. Cada função calcula os vértices e escreve os resultados num ficheiro.

2.2.2 Exemplos Figuras Geradas

Plane

O plano é gerado no eixo XZ, i.e, $Y=0$. É dividido numa matriz de quadrados com base no número de divisões, i.e , o número de **slices**. Vamos supor que o número de **slices** é i . Cada quadrado é representado por 2 triângulos, formando um "grid" de triângulos. Como o plano é dividido em $i * i$ quadrados, temos $i * i * 2$ (2 triângulos por cada quadrado).

Para calcular as coordenadas dos vértices de cada triângulo, vamos usar a seguinte fórmula:

$$coordenada = i * comp - offset$$

Onde **comp** é o comprimento de cada divisão do plano, $comp = \frac{unit}{i}$, sendo **unit** o comprimento total da aresta do plano e ainda **offset** é a metade do comprimento total do plano, para centralizar o plano na origem, i.e, $offset = \frac{unit}{2}$.

Por exemplo se tivermos um plano 1x1 e 3 divisões, teríamos 9 quadrados, logo 18 triângulos.

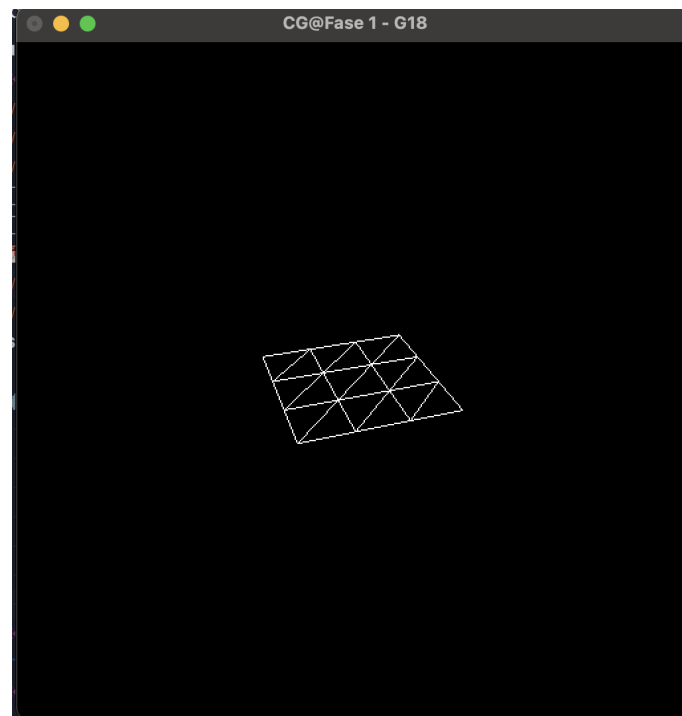


Figura 2.2: Plano gerado com parâmetros: **unit** = 1, **slices** = 3

Sphere

A esfera é gerada utilizando coordenadas esféricas. A ideia é dividir a esfera em **stacks** e **slices**. Cada vértice é calculado com coordenadas esféricas, usando as fórmulas:

- $x = r * \sin(\theta) * \cos(\theta)$

- $y = r * \cos(\theta)$
- $z = r * \sin(\theta) * \sin(\phi)$, onde θ é o ângulo de latitude (**stacks**) e ϕ é o ângulo de longitude (**slices**). Estas coordenadas geram 2 triângulos para cada quadrado formado pelas linhas de latitude e longitude.

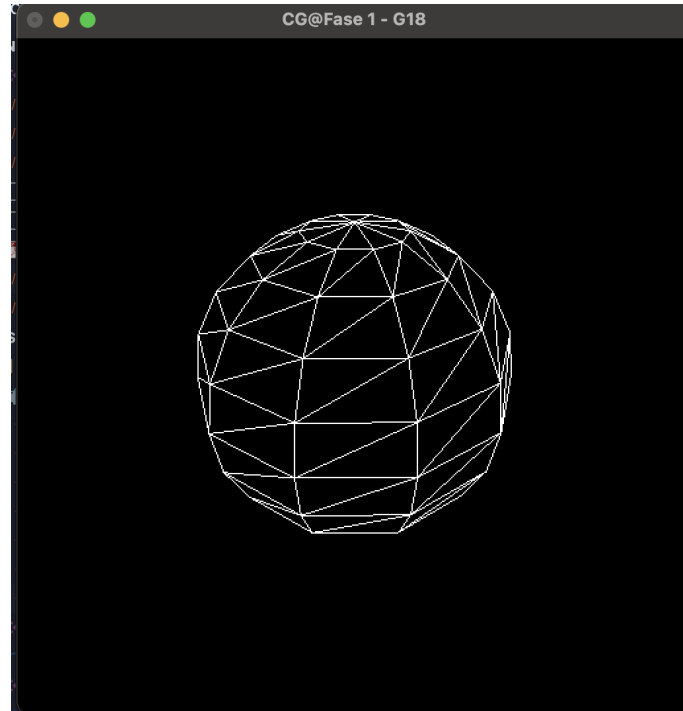


Figura 2.3: Esfera gerada com parâmetros: **radius** = 1, **slices** = 10, **stacks** = 10

Box

A caixa tem 6 faces, e cada face é gerada de forma semelhante ao plano, i.e, é subdividida em triângulos. As faces da caixa são divididas pelas coordenadas dos vértices, e cada face é subdividida em $i * i$ quadrados, com 2 triângulos cada.

Para cada face, as coordenadas x, y e z variam de acordo com a face em questão. Utilizamos os valores de comp e **offset**, apenas ajustando a direção da face (no eixo X, Y ou Z).

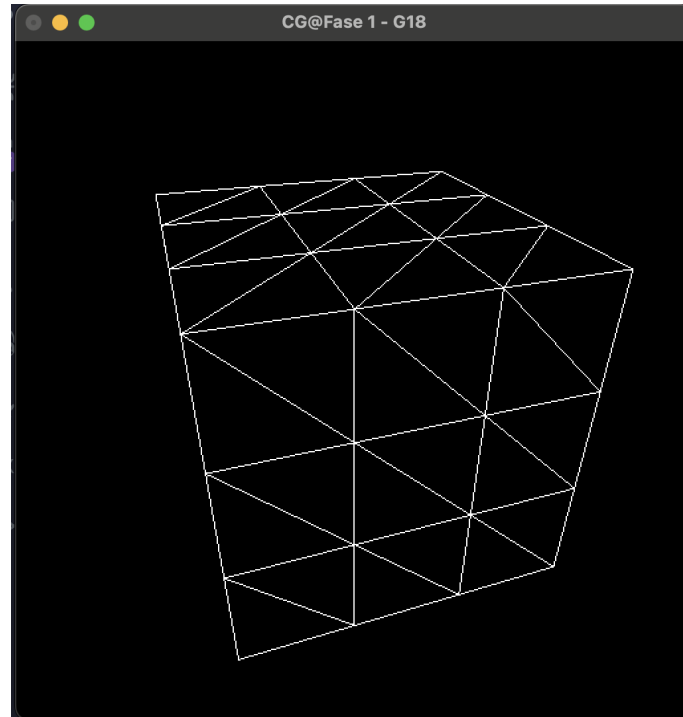


Figura 2.4: Caixa gerada com parâmetros: **unit** = 2, **slices** = 3

Cone

O cone tem um vértice superior e uma base circular. Para gerar o cone, a base é dividida em **slices** ao longo da circunferência, e a altura é dividida em camadas (**stacks**).

Para calcular as coordenadas da base, utilizam-se as coordenadas polares para calcular os pontos na base circular:

- $x = r * \sin(\theta)$
- $z = r * \cos(\theta)$

Cada fatia é representada por 2 triângulos, com o ponto central da base conectado aos pontos da circunferência.

Para as **stacks** do cone, utilizam-se as coordenadas radiais, mas ajustadas à altura, já que esta diminui à medida em que subimos.

O cálculo da posição dos vértices nas **stacks** é dado por $r = \frac{h}{r_b}$, onde r_b é o raio da base.

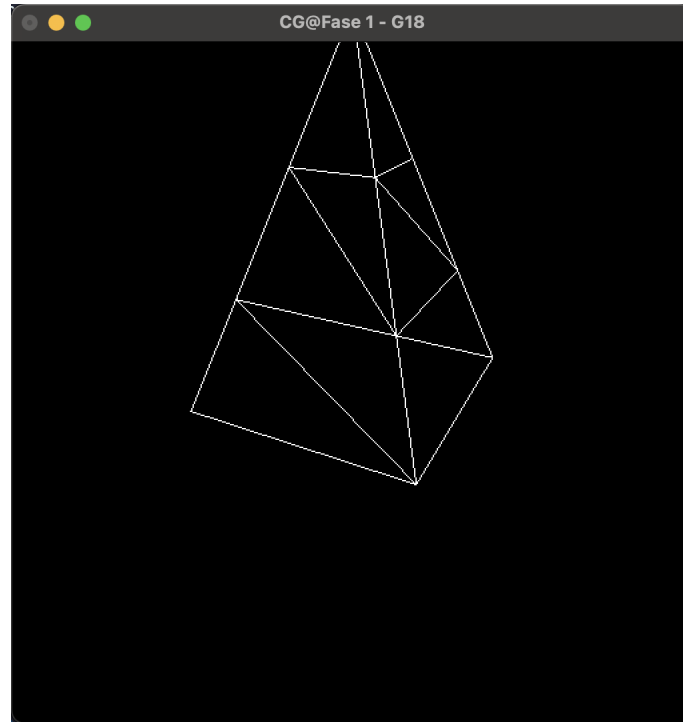


Figura 2.5: Cone gerado com parâmetros: **radius** = 1, **height** = 2, **slices** = 4, **stacks** = 3

2.2.3 Engine

O **Engine** é a componente responsável por processar os modelos tridimensionais e exibi-los utilizando o **OpenGL**. Este programa recebe como parâmetro de entrada um ficheiro **XML** contendo a configuração da cena, incluindo as definições da câmera, as dimensões da janela e os modelos a serem renderizados. É utilizada a técnica de **wireframe** para representar as coordenadas dos ficheiros **.3d**. Cada ficheiro começa com um número de triângulos a desenhar e segue com blocos de 4 linhas por triângulo, onde a primeira linha define a cor e as outras 3 as coordenadas dos vértices. Além de ler os modelos, o motor lê um ficheiro **XML** que configura a cena. O **XML** é lido utilizando a biblioteca **TinyXML2**, que extrai as informações necessárias para a renderização.

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.

Capítulo 3

Instruções de Utilização

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

3.1 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
```

3.2 Script

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

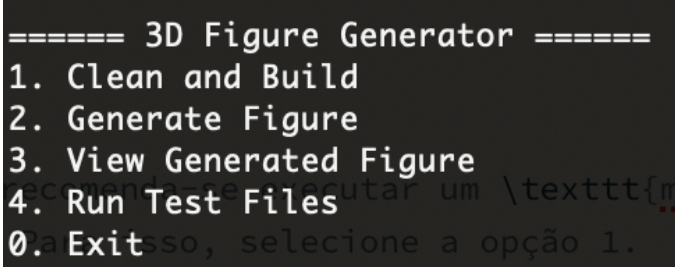
1. Tornar o **script** executável com o comando:

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

2. Executar o **script** com:

```
./run.sh
```

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



```
===== 3D Figure Generator =====
1. Clean and Build
2. Generate Figure
3. View Generated Figure
4. Run Test Files
0. Exit
```

Figura 3.1: **Script** - Menu

- 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 4

Testes

Nesta secção, serão apresentadas imagens com as figuras renderizadas conforme as configurações dos arquivos de teste no formato **.XML**.

1. **Teste 1** - Cone gerado com parametros: **radius** = 1, **height** = 2, **slices** = 4, **stacks** = 3

```
<world> | You, 5 days ago • added tests, update demo
<window width="512" height="512" />
<camera>
  <position x="5" y="-2" z="3" />
  <lookAt x="0" y="0" z="0" />
  <up x="0" y="1" z="0" />
  <projection fov="60" near="1" far="1000" />
</camera>
<group>
  <models>
    <model file="cone_1_2_4_3.3d" /> <!-- generator
    cone 1 2 4 3 cone_1_2_4_3.3d -->
  </models>
</group>
</world>
```

Figura 4.1: Configuração XML para o Teste 1, onde um cone é gerado com raio 1, altura 2, 4 **slices** e 3 **stacks**.

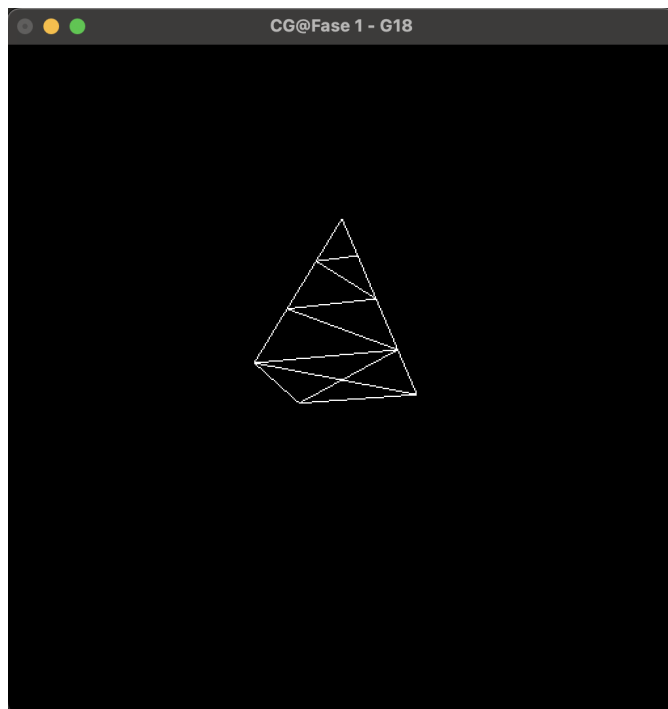


Figura 4.2: Cone renderizado com base na configuração do Teste 1. Observa-se a divisão em 4 **slices** e 3 **stacks**.

2. **Teste 2** - Cone gerado com parametros: **radius** = 1, **height** = 2, **slices** = 4, **stacks** = 3, mas com **fov** diferente.

```
<world>
  <window width="512" height="512" />
  <camera>
    <position x="5" y="-2" z="3" />
    <lookAt x="0" y="0" z="0" />
    <up x="0" y="1" z="0" />
    <projection fov="20" near="1" far="1000" />
  </camera>
  <group>
    <models>
      <model file="cone_1_2_4_3.3d" /> <!-- generator
        cone 1 2 4 3 cone_1_2_4_3.3d -->
    </models>
  </group>
</world> You, 5 days ago • added tests, update demo
```

Figura 4.3: Configuração **XML** para o Teste 2, onde um cone é gerado com raio 1, altura 2, 4 **slices** e 3 **stacks**, mas com um **fov** diferente

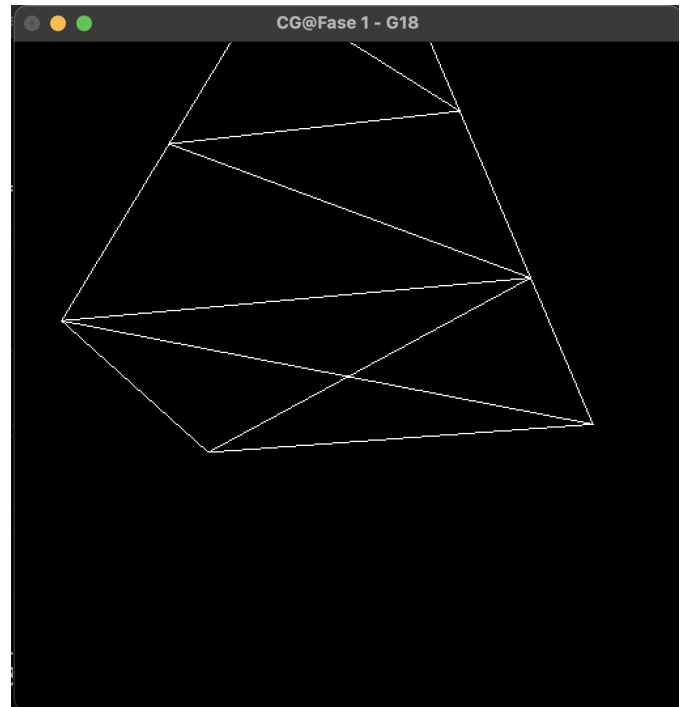


Figura 4.4: Cone renderizado com base na configuração do Teste 2. Observa-se a divisão em 4 **slices** e 3 **stacks**.

3. **Teste 3** - Esfera gerada com parametros: **radius** = 1, **slices** = 10, **stacks** = 10.

```
<world>
  <window width="512" height="512" />
  <camera>
    <position x="3" y="2" z="1" />
    <lookAt x="0" y="0" z="0" />
    <up x="0" y="1" z="0" />
    <projection fov="60" near="1" far="1000" />
  </camera>
  <group>
    <models>
      <model file="sphere_1_10_10.3d" /> <!-- generator
        sphere 1 10 10 sphere_1_10_10.3d -->
    </models>
  </group>
</world>
```

Figura 4.5: Configuração **XML** para o Teste 3, onde uma esfera é gerada com raio 1, 10 **slices** e 10 **stacks**.

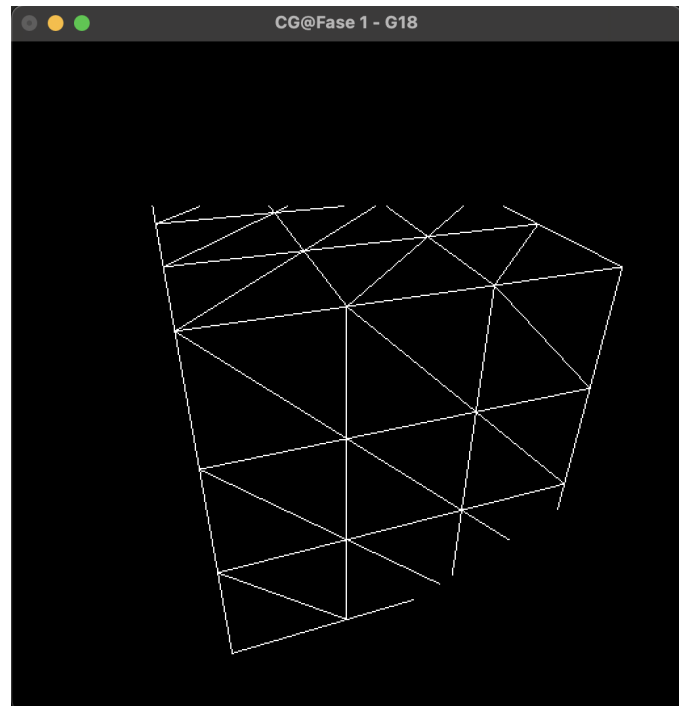


Figura 4.6: Esfera renderizada com base na configuração do Teste 3. Observa-se a divisão em 10 **slices** e 10 **stacks**.

4. **Teste 4** - Caixa gerada com parametros: **unit** = 2, **slices** = 3.

```
<world>
  <window width="512" height="512" />
  <camera>
    <position x="3" y="2" z="1" />
    <lookAt x="0" y="0" z="0" />
    <up x="0" y="1" z="0" />
    <projection fov="60" near="1" far="3.5" />
  </camera>
  <group>
    <models>
      <model file="box_2_3d" /> ←!— generator box 2 3
      box_2_3d →
    </models>
  </group>
</world>
```

You, 5 days ago • added tests, update demo

Figura 4.7: Configuração **XML** para o Teste 4, onde uma caixa é gerada com arestas de comprimento 2 e 3 **slices** por face.

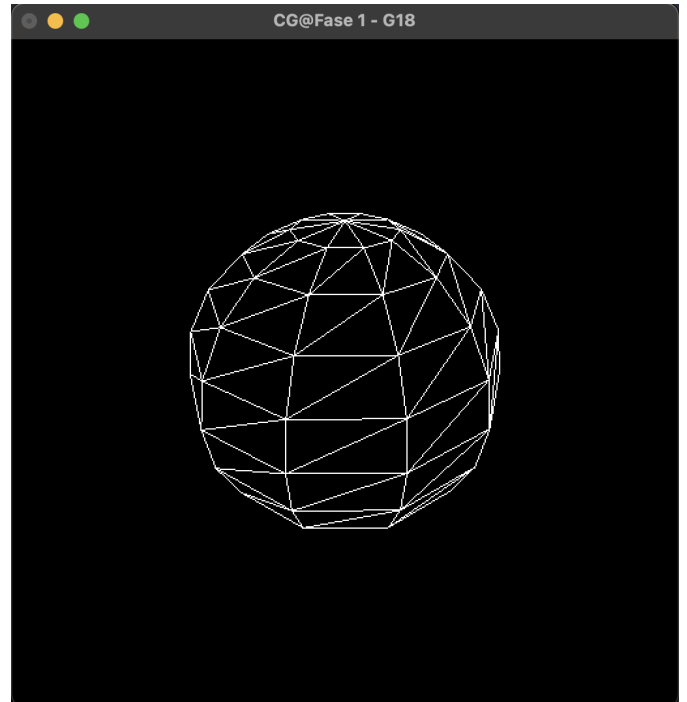


Figura 4.8: Caixa renderizada com base na configuração do Teste 4. Observa-se a divisão em 3 **slices** por face.

5. Teste 5 - Composição de figuras com um plano e uma esfera.

```
<world>
  <window width="512" height="512" />
  <camera>
    <position x="3" y="2" z="1" />
    <lookAt x="0" y="0" z="0" />
    <up x="0" y="1" z="0" />
    <projection fov="60" near="1" far="1000" />
  </camera>
  <group>
    <models>
      <model file="plane_2_3.3d" /> <!-- generator
plane 2 3 plane_2_3.3d -->
      <model file="sphere_1_10_10.3d" /> <!--
generator sphere 1 10 10 sphere_1_10_10.3d -->
    </models>
  </group>
</world>
```

Figura 4.9: Configuração **XML** para o Teste 5, onde um plano e uma esfera são renderizados na mesma cena.

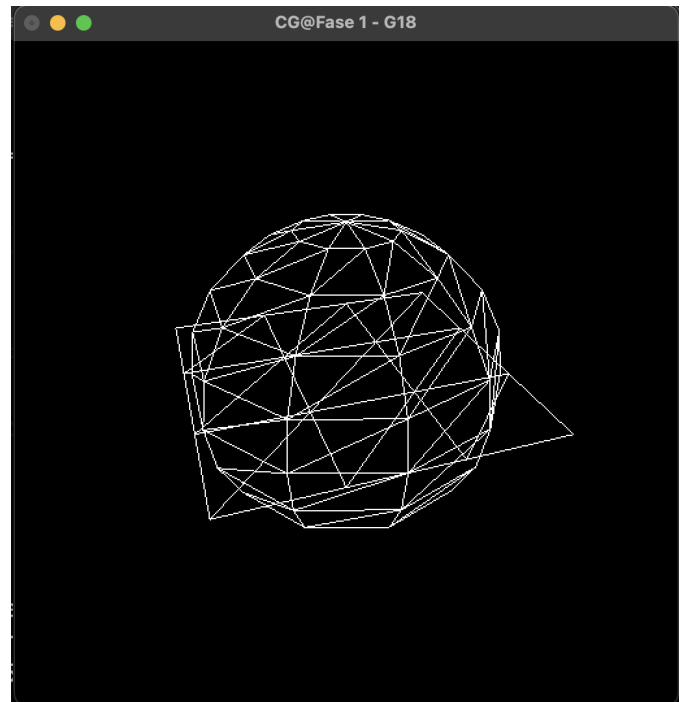


Figura 4.10: Cena renderizada com base na configuração do Teste 5. Observa-se o plano no eixo XZ e a esfera acima dele.

Capítulo 5

Conclusão

Nesta primeira fase deste projeto, tivemos oportunidade de aplicar os conceitos descritos ao longo das aulas. Observamos a importância da correção das expressões utilizadas para calcular os vértices corretamente antes do desenho das nossas formas geométricas. Chegamos à conclusão que qualquer erro nestes valores, assim como na ordem de geração dos pontos, podiam levar a grandes erros quando se chegava à fase de desenho dos modelos.

Este projeto tem sido uma experiência nova para nós, dado que até este ponto, ainda não tínhamos tido oportunidade de explorar esta área da computação.

Consideramos que o nosso projeto tem os pontos pedidos no enunciado do mesmo, no entanto, nas fases seguintes, esperamos conseguir implementar novas funções, por exemplo, mudar o ângulo de visão dos objetos através de input do teclado ou rato.