Universidade do Minho
Escola de Engenharia

Departamento de Informática

Licenciatura em Engenharia Informática

Computação Gráfica

Trabalho Prático (Fase 1)

Grupo n.º 28

Carlos Daniel Lopes Cunha, nº A106910

Pedro Nuno de Bastos Pinho Costa, nº A107375

Francisco Queirós Ribeiro da Costa Maia, nº A108962

Índice

1 Introdução

Nesta primeira fase, o trabalho no âmbito da unidade curricular de Computação Gráfica foca-se na criação de uma infraestrutura base necessária para a geração e visualização de primitivas gráficas fundamentais, estabelecendo a base para as etapas mais complexas de futuras fases. O desafio divide-se no desenvolvimento de duas aplicações complementares, um **gerador** e um **motor** de visualização. O gerador funciona como uma ferramenta de linha de comando dedicada ao cálculo geométrico de formas, neste caso, planos (um quadrado no plano XZ, centrado na origem, subdividido nas direções X e Z), cubos (dimensão e o número de divisões por aresta, centralizado na origem), esferas (raio, fatias e pilhas, centrado na origem) e cones (raio da base, altura, fatias e pilhas, a parte inferior do cone deve ser no plano XZ), permitindo controlar o nível de detalhe através da parametrização de divisões, fatias e camadas. Estes dados são exportados para ficheiros de formato próprio, que são posteriormente interpretados pelo motor. Por sua vez, o motor de visualização utiliza bibliotecas gráficas e um parser de XML para configurar o ambiente virtual, definindo parâmetros de câmara e projeção, e renderizar os modelos previamente gerados.

2 Generator

O **Generator** foi desenvolvido como uma ferramenta de linha de comando independente, cuja função principal é o cálculo geométrico e a exportação de modelos de polígonos para ficheiros de dados 3D. A sua implementação baseia-se numa estrutura organizada onde cada forma geométrica é criada por uma função específica que recebe parâmetros de dimensão e nível de detalhe. O funcionamento assenta no cálculo sistemático de coordenadas *(x, y, z)* para cada vértice, agrupando-os de três em três para definir triângulos, seguindo a regra de orientação da "mão direita" (sentido contrário aos ponteiros do relógio). O processo termina com a gravação destes dados num ficheiro com extensão .3d, que utiliza um formato simples: a primeira linha indica o número total de vértices, seguida pela lista completa das coordenadas de cada ponto. Esta abordagem permite que a geração da geometria seja feita de forma estática e prévia, otimizando o desempenho do motor de visualização que apenas terá de carregar os pontos resultantes sem necessidade de computação geométrica adicional em tempo de execução.

2.1 Modelos

2.1.1 Plano

A geração do plano baseia-se na criação de um quadrado bidimensional assente no plano *XZ*, centrado na origem do sistema de coordenadas. A superfície é dividida numa grelha regular de células quadradas, definida por um parâmetro de divisões. O processo começa por calcular o ponto de partida no canto inferior esquerdo da grelha e o tamanho de cada divisão, dividindo o comprimento total pelo número de segmentos pretendidos.

Para construir a geometria, o algoritmo percorre a grelha através de dois ciclos iterativos que representam os eixos X e Z. Em cada passo da iteração, são calculadas as coordenadas dos quatro cantos de uma célula: (x1, z1), (x1, z2), (x2, z2) e (x2, z1). Como o plano está fixo em y=0, cada célula quadrada é convertida em dois triângulos distintos para serem processados pela placa gráfica. O primeiro triângulo é formado pelos pontos (x1, 0, z1), (x1, 0, z2) e (x2, 0, z_2), enquanto o segundo triângulo utiliza os pontos (x1, 0, z1), (x2, 0, z2) e (x2, 0, z1). Esta ordem de vértices é fundamental para que a face do plano seja orientada corretamente para cima, seguindo a regra da mão direita na computação gráfica.

A variável *step* representa o tamanho de cada subdivisão ao longo de um eixo, e é calculado dividindo o comprimento total do objeto pelo número de divisões pretendidas:

$$\text{step} = \frac{\text{size}}{\text{divisions}}$$

Para garantir que o plano fica centrado na origem (0,0,0), o cálculo dos vértices começa na coordenada negativa *start* = -tamanho / 2. A partir deste ponto, o algoritmo avança em direção ao lado positivo, fazendo com que a geometria se distribua uniformemente pelos eixos e o centro do modelo coincida com o ponto central.

Para determinar a posição exata de cada quadrado da grelha, o algoritmo calcula as coordenadas dos seus quatro vértices através de fórmulas que utilizam os índices da iteração

(i, j) e o tamanho do passo (step):

- $x_1 = start + i * step$
- $z_1 = start + j * step$
- $x_2 = x_1 + step$
- $z_2 = z_1 + step$

2.1.2 Cubo (Box)

A geração do cubo baseia-se na construção de seis faces quadradas, posicionadas de forma a delimitar um volume tridimensional centrado na origem $(0, 0, 0)$. À semelhança do plano, cada face do cubo é subdividida numa grelha de células, permitindo controlar o nível de detalhe através do parâmetro de divisões por aresta. A implementação tira partido da lógica de subdivisão planar, aplicando-a de forma iterativa às seis orientações possíveis. O algoritmo calcula o tamanho do passo (*step*) dividindo a dimensão total pelo número de divisões, e define o valor fixo da face como metade da dimensão total (*fixed* = *size* / 2). Para cada face, duas coordenadas variam (percorrendo a grelha) enquanto a terceira permanece fixa, determinando se a face é frontal, traseira, superior, inferior ou lateral. Para cada uma das seis faces, o algoritmo executa os seguintes passos:

- **Faces de Topo e Base:** A coordenada Y é mantida fixa em $+$*fixed* e $-$*fixed*, respetivamente, enquanto X e Z variam para formar a grelha.
- **Faces Frontal e Traseira:** A coordenada Z é mantida fixa em $+$*fixed* e $-$*fixed*, enquanto X e Y variam.
- **Faces Laterais:** A coordenada X é mantida fixa em $+$*fixed* e $-$*fixed*, enquanto Y e Z variam.

Cada célula quadrada resultante desta subdivisão é dividida em dois triângulos. A ordem de especificação dos vértices em cada face é ajustada para respeitar a regra "da mão direita", garantindo que as normais de todas as faces apontam para o exterior do cubo, tornando o sólido corretamente visível a partir de qualquer ângulo externo. Tal como no plano, as coordenadas de cada vértice são determinadas somando o incremento do *step* multiplicado pelos índices da iteração (i, j) ao valor inicial negativo (*start* = -*size* / 2), assegurando que o sólido está perfeitamente centrado nos eixos coordenados.

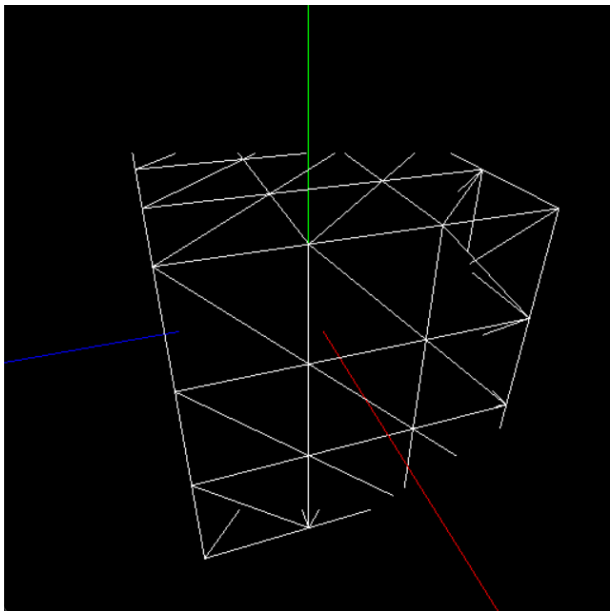


Imagem 1- Cubo gerado

2.1.3 Esfera (Sphere)

A geração da esfera é realizada através do uso de coordenadas esféricas, centrando o modelo na origem $(0, 0, 0)$. Ao contrário das primitivas planas, a superfície da esfera é definida por dois ângulos: o ângulo de longitude (α), que percorre a circunferência horizontal, e o ângulo de inclinação (β), que percorre a curvatura vertical entre os polos. O nível de detalhe e a suavidade da curvatura são controlados pelos parâmetros de fatias (*slices*) e camadas (*stacks*). O algoritmo calcula o incremento angular para cada um destes parâmetros:

- α_{step}**:** $2\pi/slices$, permitindo uma rotação completa de 360 graus.
- β_{step}**:** $\pi/stacks$, dividindo a curvatura de polo a polo.

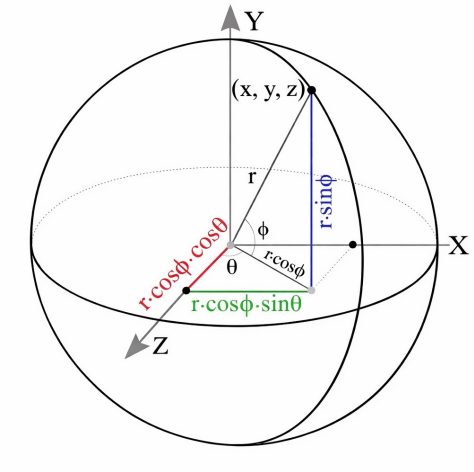


Imagem 2- Esfera

A construção da malha é feita através de dois ciclos iterativos que percorrem todos os intervalos angulares. Para cada "retângulo" esférico formado pela interseção de uma fatia com uma camada, calculam-se os quatro vértices correspondentes transformando as coordenadas esféricas em cartesianas (x, y, z) através das seguintes equações trigonométricas:

- $x = radius \cdot \cos(\beta) \cdot \sin(\alpha)$
- $y = radius \cdot \sin(\beta)$
- $z = radius \cdot \cos(\beta) \cdot \cos(\alpha)$

O algoritmo percorre o ângulo β de $-\pi/2$ até $\pi/2$, garantindo que a esfera é gerada de baixo para cima. Cada secção curva é então discretizada em dois triângulos, mantendo a ordem dos vértices necessária para que as faces fiquem orientadas para o exterior da esfera. Esta abordagem assegura uma distribuição uniforme dos triângulos e uma representação fiel do raio especificado em todas as direções.

2.1.4 Cone

A geração do cone é realizada de forma a que a sua base circular assente no plano XZ, com $y = 0$, crescendo verticalmente ao longo do eixo Y até atingir o seu vértice (topo) à altura especificada. A construção desta primitiva divide-se em duas partes fundamentais: a base e o corpo lateral, sendo ambas influenciadas pelos parâmetros de fatias (*slices*) e camadas (*stacks*).A base é gerada como um disco no plano XZ, centrado na origem. O algoritmo percorre a circunferência utilizando um incremento angular ($\alpha_{step} = 2\pi/slices$) e cria triângulos que ligam o centro $(0, 0, 0)$ aos pontos calculados no perímetro através de coordenadas polares:

- $x = radius \cdot \sin(\alpha)$
- $z = radius \cdot \cos(\alpha)$

O corpo lateral é construído por camadas horizontais (*stacks*). Em cada camada, o raio diminui linearmente à medida que a altura aumenta, partindo do raio total na base até chegar a zero no topo. Para implementar esta variação, o algoritmo utiliza incrementos fixos de altura (h_{step}) e de raio (r_{step}), calculados da seguinte forma:

- $h_{step} = \frac{height}{stacks}$
- $r_{step} = \frac{radius}{stacks}$

Para cada camada j (onde j varia de 0 até $stacks - 1$), definem-se os limites inferior e superior da secção atual:

- **Alturas:** $h_1 = j \cdot h_{step}$ e $h_2 = (j + 1) \cdot h_{step}$
- **Raios:** $r_1 = radius - (j \cdot r_{step})$ e $r_2 = radius - ((j + 1) \cdot r_{step})$

Estes valores de raio são combinados com os ângulos das fatias (α_1 e α_2) para obter as coordenadas cartesianas dos quatro pontos que definem a face lateral daquela camada:

- $P_1 = (r_1 \cdot \sin(\alpha_1), \quad h_1, \quad r_1 \cdot \cos(\alpha_1))$
- $P_2 = (r_1 \cdot \sin(\alpha_2), \quad h_1, \quad r_1 \cdot \cos(\alpha_2))$
- $P_3 = (r_2 \cdot \sin(\alpha_1), \quad h_2, \quad r_2 \cdot \cos(\alpha_1))$

- $P_4 = (r_2 \cdot \sin(\alpha_2), \quad h_2, \quad r_2 \cdot \cos(\alpha_2))$

Estes pontos formam um quadrilátero curvo que é discretizado em dois triângulos (exceto na camada final, onde $r_2 = 0$ e os pontos P_3 e P_4 coincidem no vértice, formando um único triângulo). Mantendo a consistência da orientação dos vértices, garante-se que a face exterior é a que recebe a renderização. Esta abordagem permite que, com um número elevado de fatias e camadas, a superfície lateral do cone apresente uma curvatura suave e precisa.

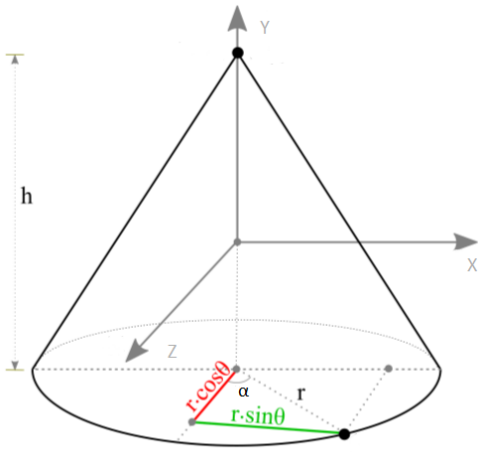


Imagem 3- Cone

2.1.5 Criação e Armazenamento em Ficheiros .3d

Após o cálculo dos pontos que compõem a primitiva geométrica, os dados são persistidos num ficheiro com extensão própria (*.3d*). A implementação utiliza um formato de texto simples, desenhado para facilitar a leitura (*parsing*). A estrutura do ficheiro segue o seguinte algoritmo de escrita:

1. **Cabeçalho:** A primeira linha do ficheiro contém um número inteiro que representa o total de vértices gerados. Esta informação permite ao motor de visualização alocar antecipadamente a memória necessária (usando estruturas como *std::vector::reserve*).
2. **Corpo de Dados:** Nas linhas seguintes, cada vértice é gravado individualmente. Cada linha contém as três coordenadas (x, y, z) separadas por espaços, representando um ponto no espaço.

A sequência de escrita dos vértices no ficheiro respeita estritamente a ordem de formação dos triângulos. Como o motor gráfico interpreta a lista de forma sequencial (agrupando cada três vértices num triângulo), a organização correta no ficheiro garante que a geometria seja reconstruída com a orientação e forma pretendidas.

3 Engine

A *engine* constitui o núcleo de visualização do projeto, assumindo a responsabilidade de interpretar as definições de cena e renderizar os modelos geométricos gerados previamente. Ao contrário do *generator*, esta componente foca-se na gestão do pipeline gráfico e na interação com o utilizador em tempo real. O funcionamento do motor baseia-se num fluxo de trabalho sequencial que começa com o processamento de um ficheiro de configuração XML, onde são extraídos os parâmetros para a construção do ambiente virtual, tais como as dimensões da janela de visualização e as propriedades da câmara. Uma vez lida a configuração, o motor carrega para a memória principal os dados dos ficheiros *.3d*, organizando os vértices em estruturas que permitem um acesso eficiente durante o ciclo de renderização. Através da utilização de bibliotecas gráficas padrão, o *engine* configura as matrizes de projeção e visualização, transformando as coordenadas matemáticas dos modelos em pixéis no ecrã. Esta separação de tarefas permite que o motor permaneça agnóstico em relação à forma como a geometria foi calculada, concentrando-se exclusivamente na fidelidade da representação visual e na correta aplicação das transformações espaciais definidas.

3.1 Leitura e Processamento de Ficheiros XML

O motor de visualização (*engine*) inicia a sua execução através da leitura de um ficheiro de configuração em formato XML. Este ficheiro serve como o "guia" da cena, definindo as propriedades da janela, o posicionamento da câmara virtual e a lista de modelos geométricos a carregar. A implementação utiliza a biblioteca *TinyXML-2* para percorrer a estrutura hierárquica do documento e extrair os atributos necessários.

O processo de *parsing* está organizado de forma a capturar três blocos fundamentais de informação:

- **Configuração da Janela (window):** Extraem-se os valores de largura (*width*) e altura (*height*), que determinam a resolução da janela criada via GLUT.
- **Configuração da Câmara (camera):** O algoritmo procura as *tags* de posição (*position*), ponto de foco (*lookAt*), vetor vertical (*up*) e parâmetros de projeção (*projection*). Estes dados são essenciais para configurar as matrizes de visualização e projeção do OpenGL, definindo o ponto de vista do observador no mundo 3D.
- **Carregamento de Modelos (group/models):** O motor percorre a lista de ficheiros *.3d* especificados. Para cada modelo, é invocada uma função de leitura que carrega a lista de vértices para a memória central, armazenando-os em estruturas de dados otimizadas para a renderização subsequente.

Esta abordagem cumpre integralmente os requisitos da primeira fase, garantindo que o motor é flexível e capaz de renderizar qualquer cena descrita no XML sem necessidade de recompilação do código fonte. A separação entre a definição da cena (XML) e a geometria (ficheiros .3d) estabelece uma base sólida para o desenvolvimento do *scene graph* nas fases seguintes do projeto.

Imagem 4- Ficheiro XML

3.2 Renderização e Visualização

A etapa final do motor consiste na transformação dos dados geométricos numa representação visual no ecrã. Este processo é gerido através do ciclo de eventos da biblioteca *GLUT* e das funções de desenho da API *OpenGL*. A renderização é executada de forma contínua na função de exibição (*display function*), que limpa os buffers de cor e profundidade antes de cada novo desenho para garantir a fluidez da imagem.

A visualização da cena é determinada pela configuração da câmara virtual, que utiliza os dados extraídos do XML. O posicionamento e orientação do observador são definidos pela função *gluLookAt*, que recebe as coordenadas da posição da câmara, o ponto para onde esta aponta (*lookAt*) e o vetor que define a direção vertical (*up*). Para a projeção, utiliza-se a função *gluPerspective*, que define o volume de visualização baseado no *Field of View* (FOV) e nos planos de corte *near* e *far*, garantindo uma perspetiva tridimensional.

O desenho dos modelos é realizado percorrendo a estrutura de dados onde os vértices estão armazenados. Aqui, utiliza-se o modo de desenho imediato do OpenGL, delimitado pelas funções *glBegin(GL_TRIANGLES)* e *glEnd()*. O motor percorre a lista de coordenadas sequencialmente, enviando cada conjunto de três vértices para formar um triângulo no espaço. Para efeitos de validação da geometria gerada, o motor foi configurado para renderizar os polígonos em modo *wireframe* (linhas), utilizando a função *glPolygonMode*.

4 Resultados Obtidos

A avaliação final desta primeira fase consistiu na validação das ferramentas desenvolvidas através da execução dos cenários de teste fornecidos pela equipa docente. O objetivo principal foi garantir que a geometria gerada pelo *generator* e a visualização produzida pelo *engine* são consistentes com os resultados esperados, tanto ao nível da estrutura da malha como da configuração da câmara:

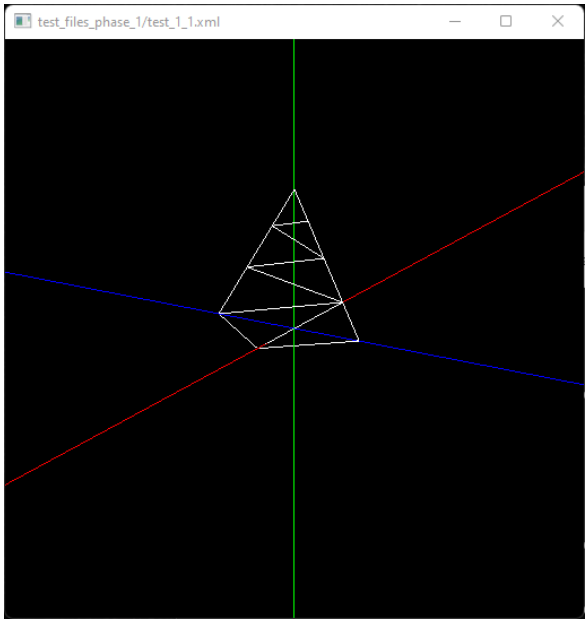
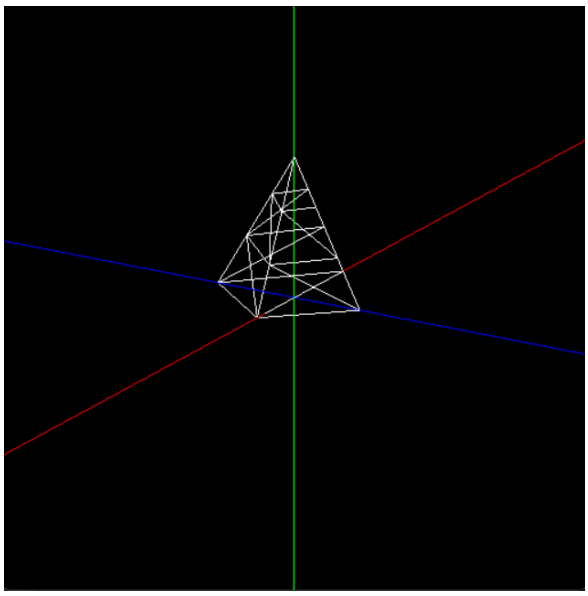


Imagem de referência (Professor)



Resultado obtido (Engine)

Imagem 5- Cone

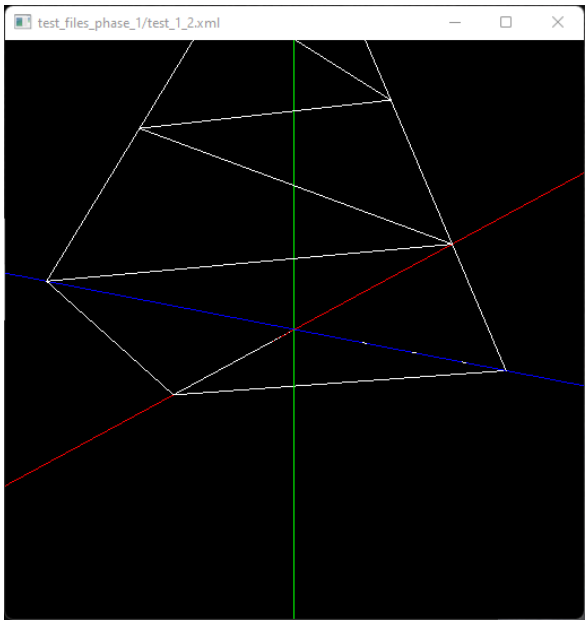
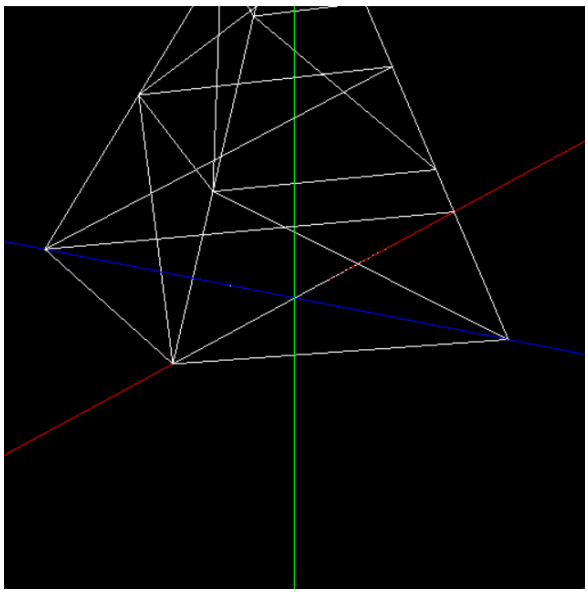


Imagem de referência (Professor)



Resultado obtido (Engine)

Imagem 6- Cone (zoom)

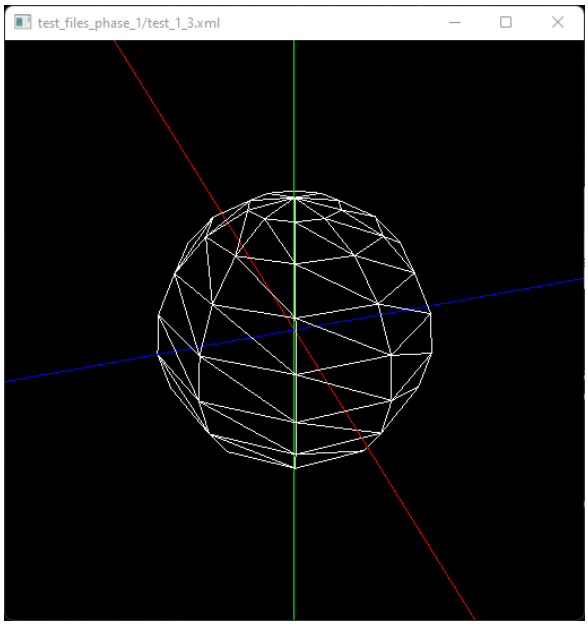
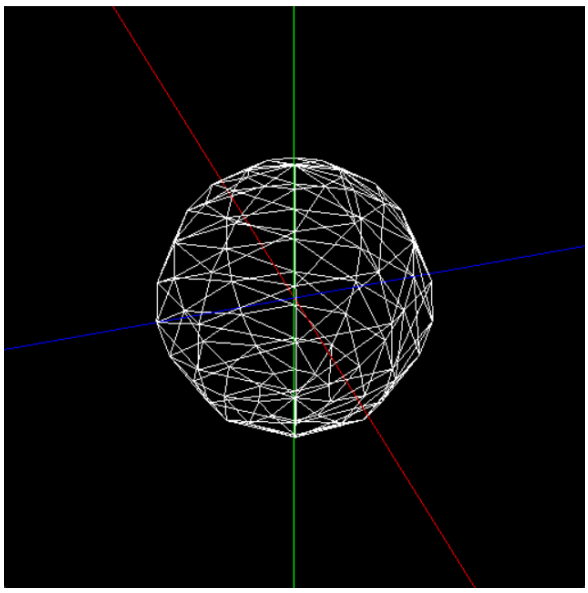


Imagem de referência (Professor)



Resultado obtido (Engine)

Imagem 7- Esfera

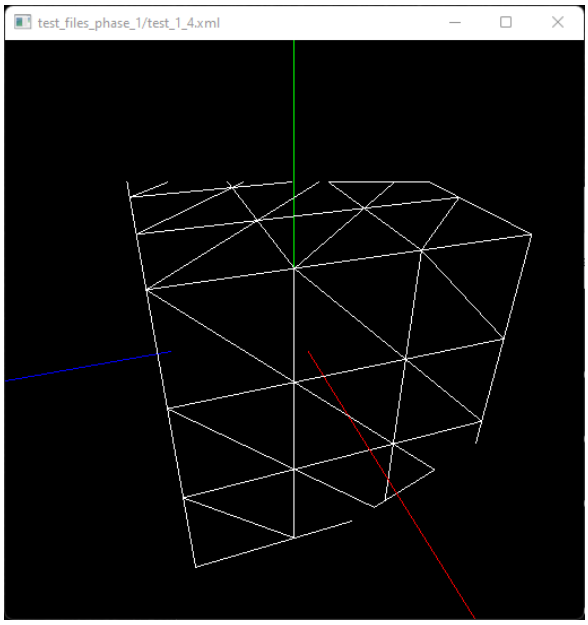
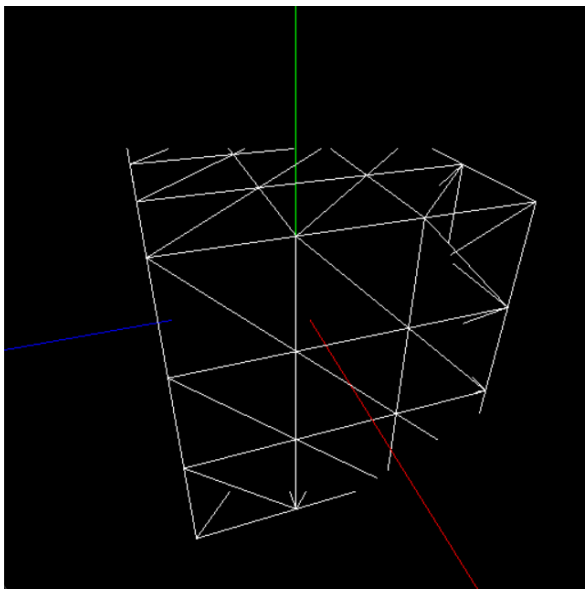


Imagem de referência (Professor)



Resultado obtido (Engine)

Imagem 8- Cubo

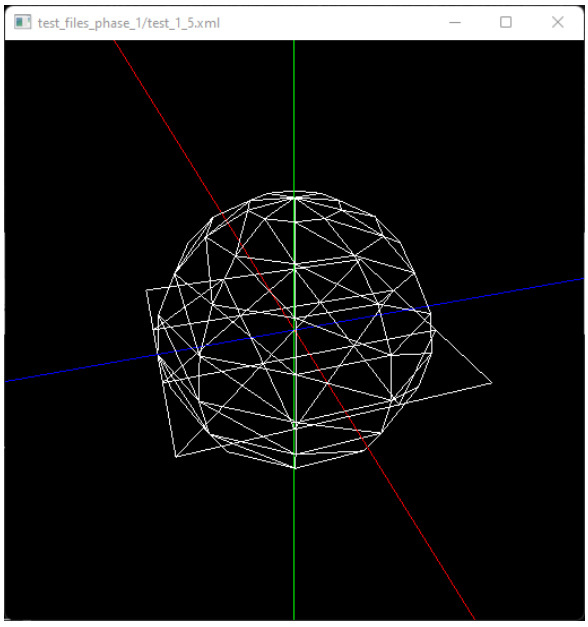
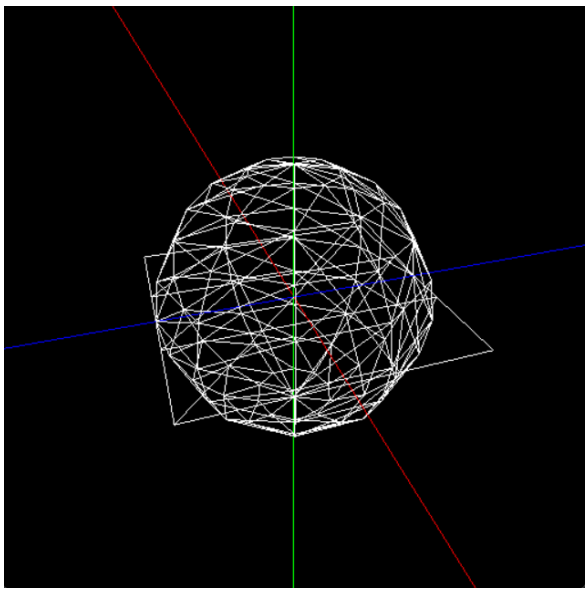


Imagem de referência (Professor)



Resultado obtido (Engine)

Imagem 9- Esfera com Plano

5 Conclusão

A conclusão desta primeira fase do projeto marca o estabelecimento com sucesso da infraestrutura base do motor de renderização. O desenvolvimento das duas aplicações independentes permitiu separar as responsabilidades da geração geométrica da lógica de visualização.

A implementação do *generator* permitiu consolidar conhecimentos práticos sobre a representação matemática de superfícies no espaço 3D. Através da transformação de formas como o plano, o cubo, a esfera e o cone em malhas de triângulos, foi possível compreender a importância de parâmetros como divisões, fatias e camadas no equilíbrio entre o detalhe visual e a eficiência computacional. Por outro lado, o desenvolvimento do *engine* forneceu uma introdução sólida ao uso da API OpenGL e ao processamento de ficheiros de configuração XML, garantindo que o sistema é capaz de montar cenas complexas de forma dinâmica.

A validação através dos testes propostos confirmou que tanto a geometria gerada como a configuração da câmara e perspetiva estão a funcionar conforme o esperado, produzindo resultados visuais idênticos aos modelos de referência..