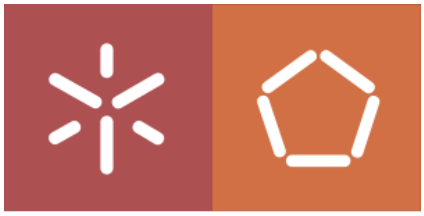


Universidade do Minho
Escola de Engenharia

Computação Gráfica

Trabalho prático: Fase 1 – Primitivas Gráficas

Grupo 38

Maria Inês da Rocha Pinto Gracias Fernandes, a104522

Pedro Manuel Macedo Rebelo, a104091

Vasco João Timóteo Gonçalves, a104527

a104522

a104091

a104527

2 de Março, 2025

Índice

Introdução

O presente relatório apresentará informações relativas à 1ª Fase do Trabalho Prático da Unidade Curricular Computação Gráfica, pertencente ao 3º Ano da Licenciatura em Engenharia Informática da Universidade do Minho, realizada no ano letivo 2024/2025.

O projeto, nesta fase, consiste em desenvolver um sistema que permita gerar e visualizar primitivas gráficas em 3D. Foram desenvolvidos um plano, uma caixa (paralelipípedo), uma esfera e um cone. Para isso, foram implementadas duas aplicações: um gerador de modelos que cria ficheiros com informações sobre os vértices das primitivas e um motor gráfico que processa esses ficheiros e exibe os modelos com base num ficheiro de configuração XML.

Estrutura do projeto

1. Aplicações

Esta fase, como referido anteriormente está dividida em duas aplicações distintas que se complementam:

- *Generator*: Tem a função de converter as primitivas geométricas para conjuntos de vértices armazenados em ficheiros estruturados por nós. Esta ferramenta auxilia o *Engine* na representação 3D.
- *Engine*: Aplicação que lê ficheiros XML que contêm referências a ficheiros criados pelo gerador. As transformações geométricas de cada objeto (informação acerca do tamanho da janela, da câmara e dos modelos a desenhar no ecrã) são representadas recorrendo à biblioteca OpenGL.

2. Classes

De modo a facilitar a implementação das funcionalidades anteriormente apresentadas decidimos criar algumas classes que servissem como suporte às mesmas.

2.1. Camera

A classe **Camera** é responsável por todas as funcionalidades que podem ser implementadas numa câmara. Trabalhamos sempre com coordenadas esféricas e convertemos para coordenadas cartesianas apenas quando necessário.

As coordenadas esféricas são o beta (ângulo vertical), o alpha (ângulo horizontal), e o raio (distância entre a câmara e o lookAt). Como temos de rodar a câmara em torno do lookAt (ponto de observação), é mais fácil trabalhar com coordenadas esféricas.

2.2. Parser

A leitura do ficheiro de configuração XML é realizada pela classe **SimpleParser**, que utiliza a biblioteca TinyXML2 para processar os elementos do ficheiro e armazenar as configurações necessárias para a renderização da cena.

O parser extrai as seguintes informações do XML:

- **Janela de renderização**: Obtém os atributos *width* e *height* do elemento *window*.
- **Configuração da câmara**: Lê as informações da posição (*position*), ponto de observação (*lookAt*), vetor up (*up*) e parâmetros de projeção (*projection*).
- **Modelos 3D**: Percorre os elementos *model* dentro de *group*, armazenando os ficheiros de modelo a serem carregados.

3. Ferramentas utilizadas

Com o intuito de demonstrar as funcionalidades do *engine*, utilizamos a ferramenta TinyXML2, que nos facilitou o processo de *parsing* do cenário utilizado. Para além desta, recorremos também à biblioteca em que assentam as funcionalidades gráficas do projeto, a OpenGL.

Todos os modelos são armazenados na diretoria '*files3d'*, criada automaticamente caso não exista.

Primitivas gráficas

1. Plano

Para obter um plano paralelo ao plano XZ e centrado na origem, subdividimos a superfície em pequenos quadrados. Cada quadrado é composto por dois triângulos que compartilham um lado, garantindo que se gera o plano corretamente.

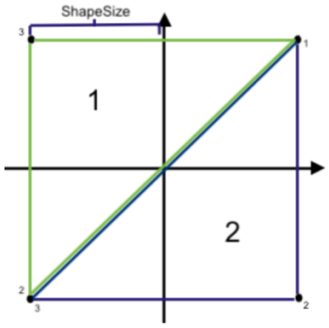


Figura 1 - Face superior do plano

O centro do plano é a origem do referencial (0,0,0), e os vértices dos triângulos são calculados com base no tamanho do plano e no número de divisões.

Os vértices de cada triângulo são definidos da seguinte forma:

Triângulo 1:
Ponto 1: (x1, 0, z1)
Ponto 2: (x3, 0, z3)
Ponto 3: (x2, 0, z2)

Triângulo 2:
Ponto 1: (x2, 0, z2)
Ponto 2: (x3, 0, z3)
Ponto 3: (x4, 0, z4)

2. Caixa

Para calcular os vértices da caixa, representamos cada face como uma matriz n por n, em que n é o número de divisões.

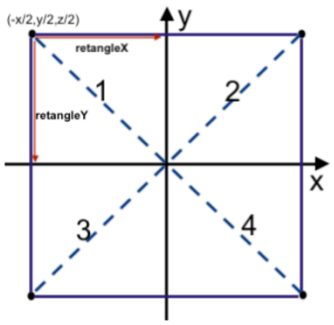


Figura 2 - Face frontal da caixa

Cada face é definida num dos seis planos principais (frontal, traseiro, superior, inferior, esquerdo e direito). A subdivisão das faces permite uma melhor resolução do modelo.

Para calcular os vértices de uma face com comprimento (size) e divisões (divisions), utilizamos as seguintes variáveis:

- **halfSize = size / 2.0f**, que define a metade do comprimento do cubo.
- **step = size / divisions**, que define o espaçamento entre os pontos.

Cada vértice da face é calculado iterativamente conforme a sua posição na matriz. Os vértices são posicionados de acordo com a face correspondente, e os triângulos são definidos seguindo a regra da mão direita.

Cada quadrado gerado por duas subdivisões da face é dividido em dois triângulos, conforme a lógica do plano, mas atribuindo valores diferentes a y:

Triângulo 1:
Ponto 1: (x1, y1, z1)
Ponto 2: (x3, y3, z3)
Ponto 3: (x2, y2, z2)

Triângulo 2:
Ponto 1: (x2, y2, z2)
Ponto 2: (x3, y3, z3)
Ponto 3: (x4, y4, z4)

3. Cone

A geração do cone foi dividida em duas partes principais: a base e a camada lateral.

A **base** do cone é um círculo centrado na origem no plano XZ. O círculo é dividido em "fatias" (*slices*), onde cada fatia forma um triângulo ligado ao centro. Os vértices da base são calculados variando o ângulo theta de 0 a 2π.

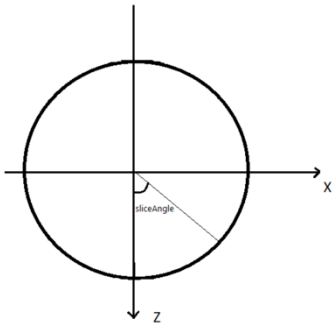


Figura 3 - Base do cone

Cada triângulo da base é definido pelos seguintes pontos:

- (0, 0, 0) (centro da base)
- (raio * cos(theta1), 0, raio * sin(theta1))
- (raio * cos(theta2), 0, raio * sin(theta2))

A **lateral** do cone é composta por fatias triangulares, criadas ao longo de camadas (*stacks*). Os vértices das fatias são calculados iterativamente com base na altura (H) e nos raios (R e R').

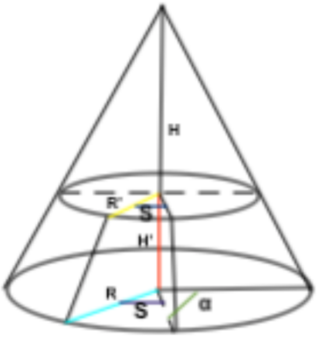


Figura 4 - Cone

Cada fatia da lateral do cone é formada por dois triângulos:

Triângulo 1:
(x1, y1, z1)
(x2, y1, z2)
(x3, y2, z3)

Triângulo 2:
(x2, y1, z2)
(x4, y2, z4)
(x3, y2, z3)

4. Esfera

A esfera foi gerada utilizando coordenadas esféricas, dividindo a superfície da esfera em *slices* (fatias verticais) e *stacks* (camadas horizontais). Cada quadrilátero gerado pelas subdivisões é decomposto em dois triângulos para a correta definição do modelo. Essa divisão permite uma transformação eficiente da superfície curva da esfera, aproximando-a através de pequenos triângulos.

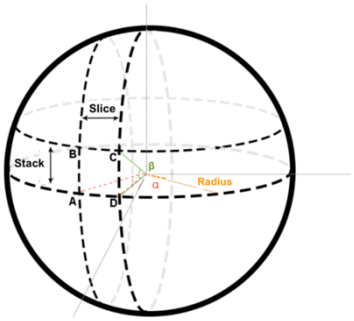


Figura 5 - Esfera

Os vértices da esfera são determinados por dois ângulos principais:

- *theta* (ângulo de inclinação), que controla a posição em relação ao eixo Y (latitude).
- *phi* (ângulo de rotação), que controla a posição ao redor do eixo Y (longitude).

Os ângulos são calculados iterativamente para cada *slice* e *stack*, dividindo a esfera uniformemente. A cada iteração, dois triângulos são gerados para formar um quadrilátero na superfície da esfera:

Triângulo 1:
(x1, y1, z1) (vértice superior esquerdo)
(x3, y3, z3) (vértice inferior esquerdo)
(x2, y2, z2) (vértice superior direito)

Triângulo 2:
(x2, y2, z2) (vértice superior direito)
(x3, y3, z3) (vértice inferior esquerdo)
(x4, y4, z4) (vértice inferior direito)

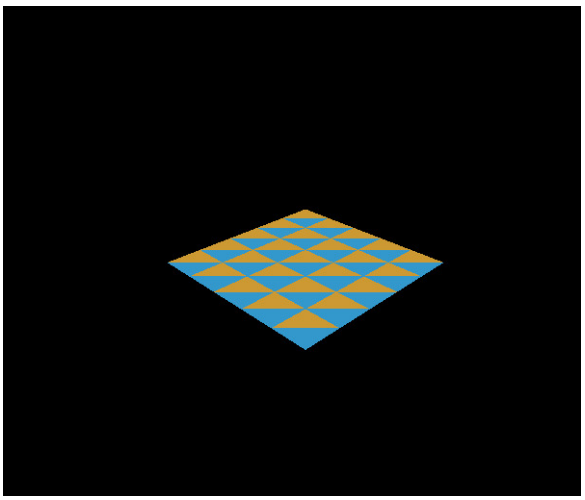
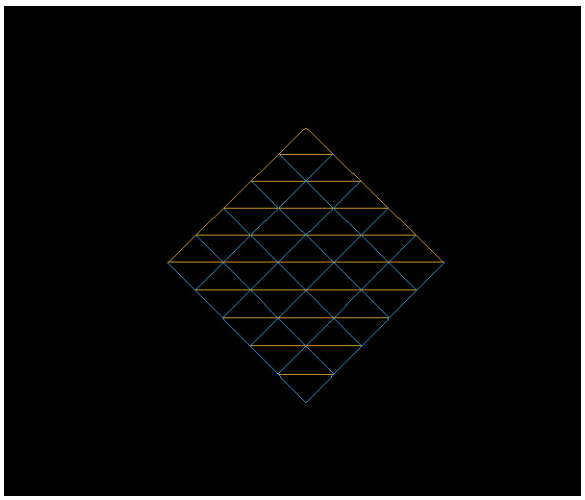
Este processo é repetido para cada *stack* e *slice*, garantindo que toda a superfície da esfera fica coberta por triângulos.

Modelos

1. Plano

O plano foi gerado com os seguintes argumentos:

- Length: 3
- Divisions: 5

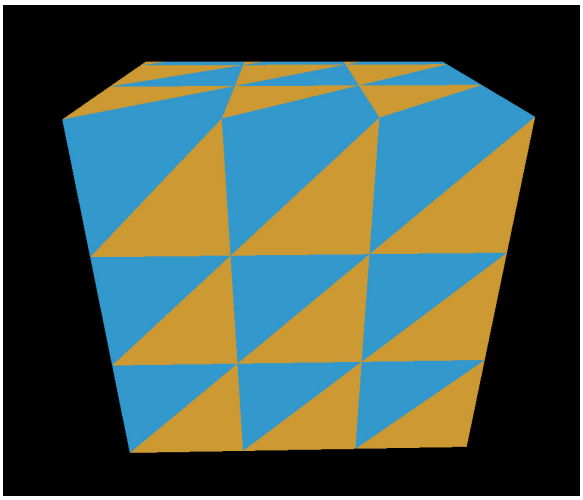
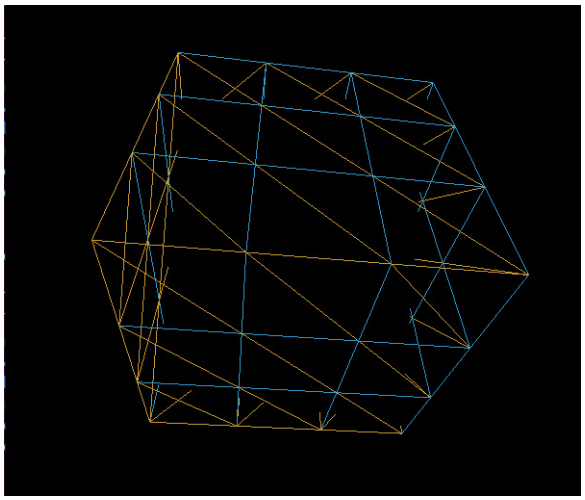



Figuras 6 e 7 – Modelos do plano

2. Caixa

A caixa representada nas figuras 8 e 9 foi gerada com os seguintes argumentos:

- Size: 2
- Divisions: 3

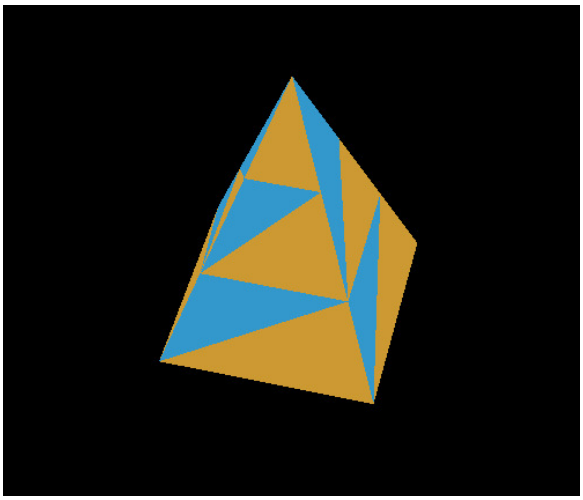
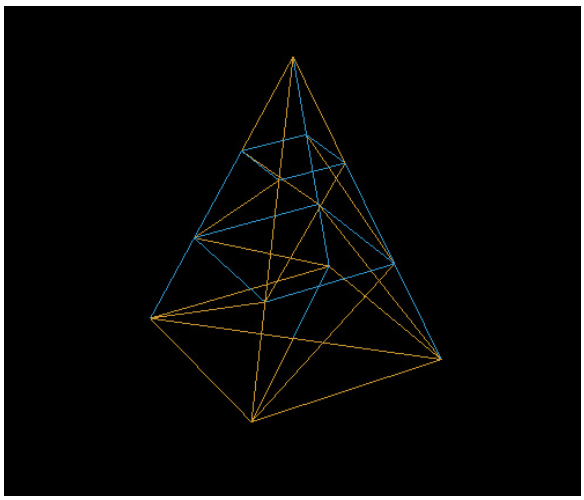



Figuras 8 e 9 – Modelos da caixa

3. Cone

O cone abaixo foi gerado com os argumentos:

- Radius: 1
- Height: 2
- Slices: 4
- Stacks: 3

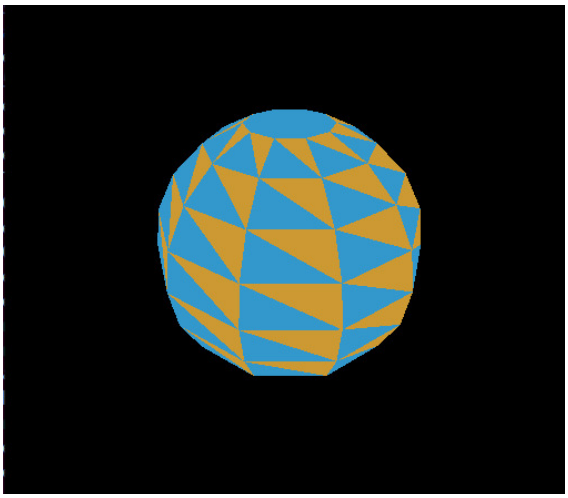
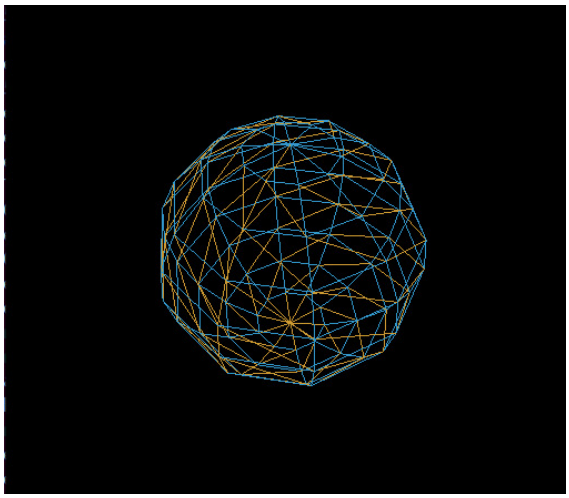



Figuras 10 e 11 – Modelos do cone

4. Esfera

A esfera gerada teve em consideração os seguintes argumentos:

- Radius: 1
- Slices: 10
- Stacks: 10

Figuras 12 e 13 – Modelos da esfera

Conclusão

O trabalho desenvolvido nesta primeira fase satisfez os requisitos pedidos. Esta implementação foi ponderada pensando já no objetivo final do projeto da implementação do sistema solar.

A realização desta primeira fase foi crucial para estabelecer uma base sólida para o restante desenvolvimento do projeto. A familiaridade adquirida com as ferramentas e a linguagem de programação durante a Fase 1 simplificará a execução da próxima tarefa. Deste modo, consideramos que estamos mais confortáveis com os procedimentos e métodos necessários para manipular geometrias e implementar as transformações desejadas.

Os modelos foram corretamente gerados e exibidos, garantindo uma base sólida para as próximas fases do trabalho.