Computação Gráfica (3º ano)

Trabalho Prático - Fase 1

Relatório de Desenvolvimento

Alexis Correia	João Fonseca	Moisés Ferreira	Rita Machado
a102495	a102512	a97020	A102508

2 de março de 2025

Conteúdo

Capítulo 1

Introdução

No âmbito da UC Computação Gráfica, foi-nos proposto o desenvolvimento de um mecanismo 3D baseado em minigráficos.

Serão usadas duas ferramentas, abordadas nas aulas, sendo elas OpenGL e C++.

Esta primeira fase consiste na criação de duas aplicações, uma que gera (**Generator**) ficheiros com as informações de modelos e outra que lê (**Engine**) as informações escritas num ficheiro XML e mostra os modelos de acordo com a informação lida.

O documento contêm as decisões tomadas pelo grupo bem como todas as etapas desenvolvidas.

Capítulo 2

Desenvolvimento

2.1 Generator

O *Generator* tem como função gerar a figura que pretendemos criar, para isso terá que receber parâmetros, tais como: o nome da figura (plane, box, sphere, cone), as suas dimensões (diferem consoante o tipo da figura) e ainda o nome do ficheiro destino onde se pretende guardar a figura criada.

Esta aplicação destina-se a gerar os vários vértices constituintes das primitivas gráficas requisitadas. Os vértices gerados irão formar triângulos, uma vez que estes são a unidade de constituição de todas as primitivas.

Nesta fase são necessárias as seguintes primitivas gráficas:

- Plano (um quadrado no plano XZ, centrado na origem, subdividido nas direções X e Z)
- Caixa (requer dimensão, e o número de divisões por aresta, centrada na origem)
- Esfera (requer raio, fatias e camadas, centrada na origem)
- Cone (requer raio inferior, altura, fatias e camadas, a parte inferior do cone deve ser no plano XZ)

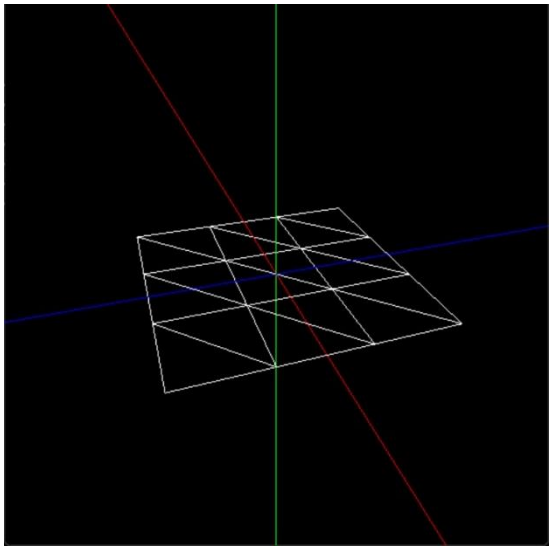
2.1.1 Plano

Para criar um plano a nossa função *drawPlane* recebe 2 argumentos:

- ***int length*** - esta variável determina o comprimento do plano;
- ***int divs*** - esta variável determina o número de divisões que formam o plano;

O plano tem n divisões e cada uma dessas n divisões está dividida em dois triângulos simétricos com 2 vértices coincidentes. Estes triângulos são paralelos com o plano *xOz*, ou seja, o valor da coordenada y é 0. Para cada divisão do plano são escritos 4 vértices (as coordenadas do x e z) que formam os 2 triangulo que compõem a superfície plana.

Exemplo de execução:



(a) .\generator plane 2 3 plane.3d

2.1.2 Box

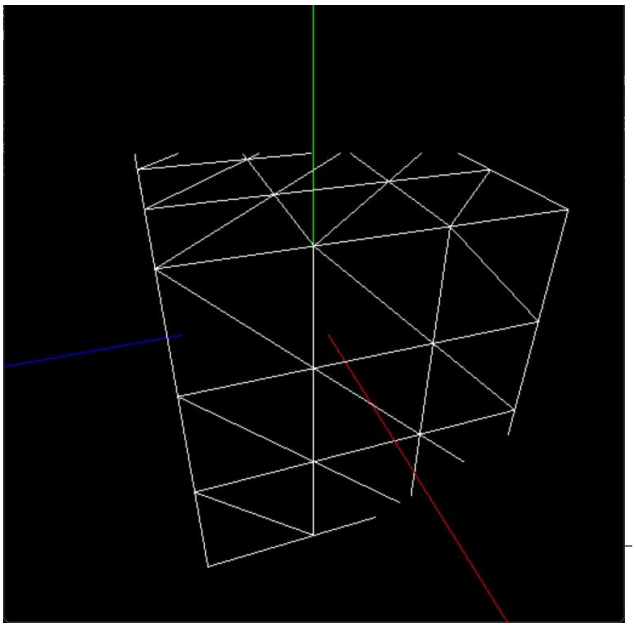
Para criar uma box a nossa função *drawBox* recebe 2 argumentos:

- ***int dimension*** - esta variável determina o comprimento de cada plano da box;
- ***int divisions*** - esta variável determina o número de divisões que formam cada plano da box;

Com a primitiva do plano feita, facilmente se conseguiu construir a box, uma vez que esta é obtida através de 6 planos paralelos dois a dois. Os valores das variáveis x, y e z são divididos por 2, representando a metade das dimensões do cubo.

Desta forma calculamos os vértices de uma face e seguidamente, por um loop *for* calculamos os vértices da face paralela. Com a ajuda de outro loop executamos a mesma operação para todas as faces.

Exemplo de execução:



(b) .*generator box 2 3 box.3d*

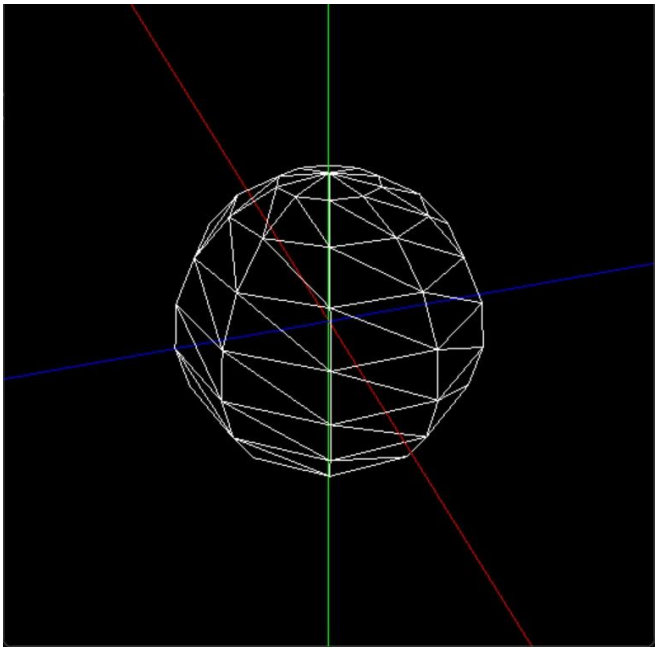
2.1.3 Esfera

Para criar uma esfera a nossa função *drawSphere* recebe 3 argumentos:

- ***float radius*** - esta variável indica o raio da esfera;
- ***int slices*** - esta variável determina o número de divisões, ou seja, o número de triângulos que formam a esfera ao longo do eixo X;
- ***int stacks*** - esta variável determina o número de divisões, ou seja, o número de triângulos que formam a esfera ao longo do eixo Y;

A esfera é construída por meio de um loop for que percorre todas as fatias e camadas, criando uma sequência de triângulos que vão formar a esfera completa. O loop principal itera sobre cada fatia da esfera, calculando as coordenadas dos vértices em cada fatia para a parte superior e inferior da esfera, bem como as coordenadas dos vértices para as fatias intermediárias.

Exemplo de execução:



(c) .*generator sphere 1 10 10 sphere.3d*

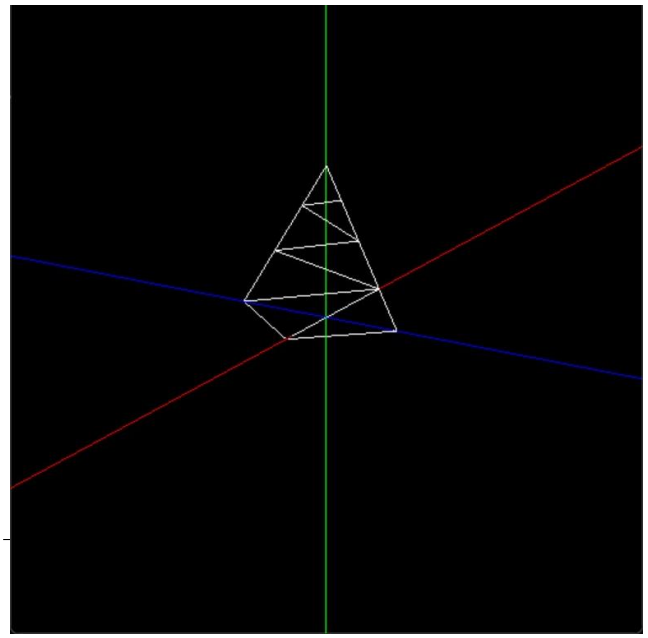
2.1.4 Cone

Para criar um Cone a nossa função *drawCone* recebe 4 argumentos:

- ***float radius*** - esta variável determina o raio da base do cone;
- ***float height*** - esta variável indica a altura do cone;
- ***float slices*** - esta variável determina o número de divisões, ou seja, o número de triângulos que formam o cone ao longo do eixo X;
- ***float stacks*** - esta variável determina o número de divisões, ou seja, o número de triângulos que formam o cone ao longo do eixo Y;

O cone será construído stack a stack, e em cada uma, slice a slice, estando este centrado na origem com base no eixo xOz. As coordenadas dos vértices do cone são obtidas através de cálculos trigonométricos e geométricos baseados nas dimensões do cone (raio e altura), bem como nos parâmetros de fatiamento (slices) e empilhamento (stacks).

Exemplo de execução:



(d) .*generator cone 1 2 4 3 cone.3d*

2.2 Engine

O Engine recebe um ficheiro XML com as configurações. Este ficheiro recebe informações da câmera e a indicação do ficheiro 3d que vamos ler. O ficheiro XML é lido unicamente quando o engine começa. Para ler este ficheiro, utilizamos a biblioteca RapidXML

2.2.1 Implementação

O programa utiliza as bibliotecas *rapidxml* e *glut* para desenhar figuras em 3D e manipular a câmera que as observa.

As variáveis iniciais definem as coordenadas e ângulos da câmera, a cor e tipo de desenho, bem como outras configurações.

A função *changeSize* é responsável por definir a perspectiva da cena a partir do tamanho da janela.

A função *renderScene* é responsável por renderizar a cena em si, incluindo a câmera, os eixos, a figura e outras configurações.

Capítulo 3

Conclusão

Foram sentidas algumas dificuldades ao longo desta Fase, nomeadamente as coordenadas polares exigidas pelo cone e pela esfera, e ainda a implementação de uma função para ler o ficheiro XML.

A realização desta Fase 1 permitiu uma melhor adaptação à cadeira e aos seus fundamentos, aplicando a matéria dada tanto nas aulas teóricas como nas práticas. Fez ainda com que praticássemos as várias funções do GLUT.

Por fim, queremos melhorar o nosso trabalho no decorrer de cada Fase, e consolidar ainda mais a matéria.