






# Fase 1

## Computação Gráfica

Março de 2022

Mestrado Integrado em Engenharia Informática  
Universidade do Minho

A89466	A89511	A89550
		
João Manuel Silva de Amorim	Rodrigo Caldas Meira	João Miguel Santos Sá

## 1 Introdução

Para a unidade curricular de Computação Gráfica foi nos proposto criar uma mini cena com desenhos 3D que se encontra dividido em 4 fases. Nesta primeira fase foi proposta a realização de um gerador e de um motor de gráficos primitivos, ou seja, ficamos encarregues de criar dois programas. Um programa que gera os vários pontos que caracterizam uma primitiva gráfica e guarda-os num ficheiro 3D. Outro programa capaz de ler um ficheiro XML, que contém o nome/es do ficheiro 3D que queremos desenhar, e desenhar os triângulos compostos pelos pontos lidos. Outra característica deste último programa é guardar em memória as informações do ficheiro XML que irá utilizar, para não obrigar a uma nova leitura do ficheiro cada vez que queremos obter um novo ponto, por exemplo.

## 2 Utilização do Projeto

Os ficheiros .3d gerados pelo generator, são colocados na pasta da engine e os ficheiros .xml de teste estão numa pasta testes dentro da engine, onde são utilizados. Por predefinição a engine tenta correr o ficheiro chamado "test\_1\_4.xml", mas o utilizador poderá passar outro ficheiro como argumento, se o desejar.

## 3 Desenho das Figuras

Os nossos modelos são construídos a partir de triângulos desenhados na cena. Antes de implementar a geração dos modelos a partir de um executável, o grupo focou-se em decidir a forma como as diferentes figuras serão obtidas a partir da biblioteca OpenGL, nomeadamente a ordem como os vértices são desenhados, pois isso tem naturalmente impacto na decisão do lado visível do triângulo.

### 3.1 Cone

Para desenhar o cone começamos por desenhar por slices, em cada slice fazemos o desenho do triângulo da base do cone e depois desenhamos os triângulos da face lateral por stacks. Com o uso de stacks, ao definir os vértices é agora necessário saber em que stack ou camada do cone é que estamos a inserir o novo vértice, pois as suas coordenadas serão condicionadas por esse atributo. Uma vez que as faces laterais do cone são lineares e não formam qualquer tipo de curva, para saber a coordenada Y (vertical) de cada ponto basta apenas dividir a altura do cone pelo número de stacks, todos os pontos de uma dada stack terão naturalmente o mesmo valor de Y.

Por outro lado, para saber as coordenadas no plano XZ, não basta saber o ângulo, tal como é feito no cilindro, uma vez que dependendo da stack, a distância a partir do centro (raio) altera-se. Novamente, como as faces do cone não apresentam curva basta dividirmos o raio do círculo base do cone pelo número de stacks, e sabemos o raio entre o centro e o ponto a inserir.

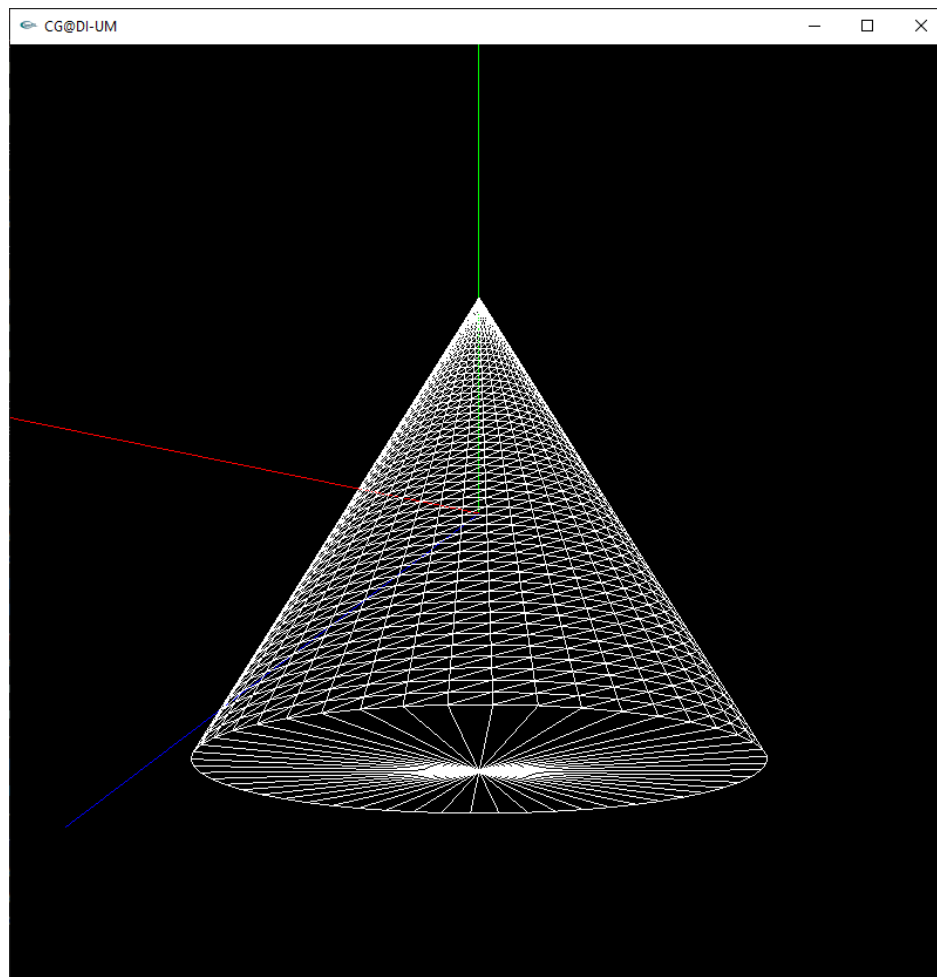


Figure 1: Esquema de formação de um cone com raio 1, 2 de altura, 50 slices e 50 stacks.

### 3.2 Esfera

Mas uma vez, a esfera também é desenhada por slice, tal como as figuras explicadas anteriormente. O que distingue o cálculo das coordenadas da esfera para com as do cone, é o facto de que as faces laterais / slices da esfera serem curvas. Desta forma não basta simplesmente dividir os valores de raio e altura pelo número de stacks. A solução é utilizar coordenadas esféricas: para cada ponto, é necessário saber um ângulo  $\alpha$  e um ângulo  $\beta$ , e utilizar as funções seno e cosseno para as transformações para cartesiano.

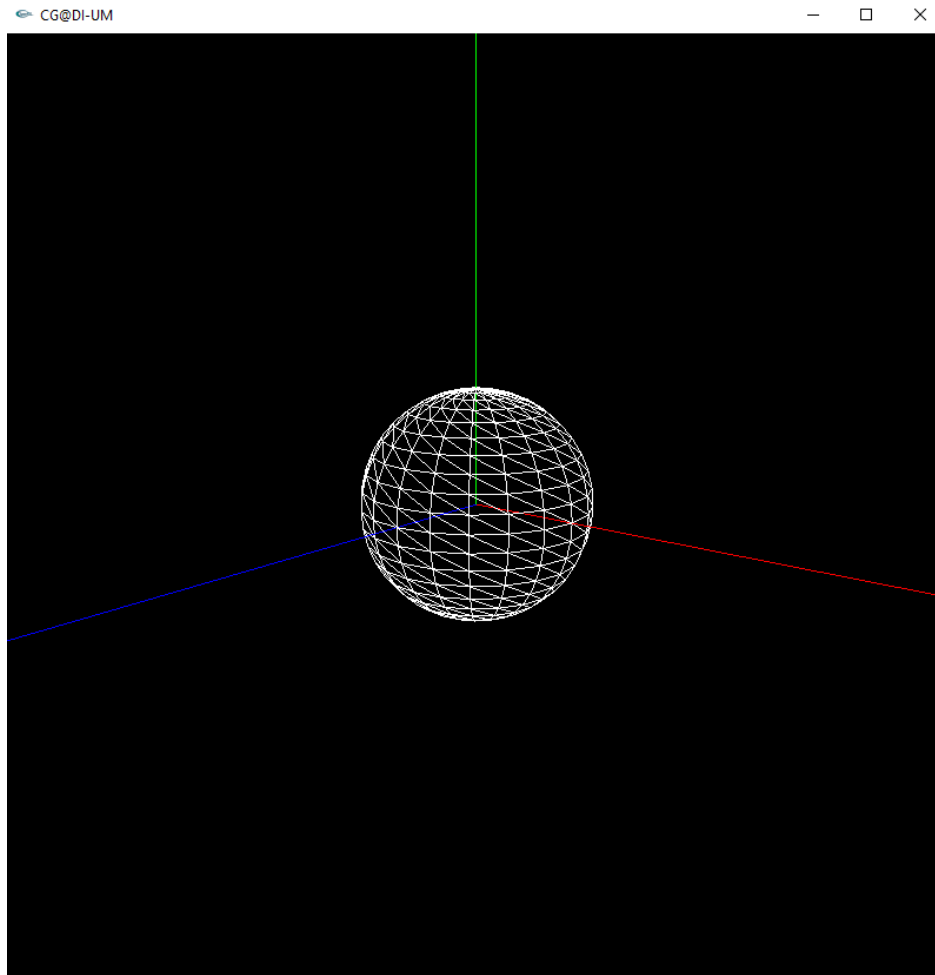


Figure 2: Esquema de formação de um esfera raio 1, 20 slices, 20 stacks.

### 3.3 Plano

A implementação do plano neste trabalho prático foi feita através de 2 ciclos *for*s. Basicamente, o plano foi elaborado com uma das faces da caixa, mais especificamente, o teto.

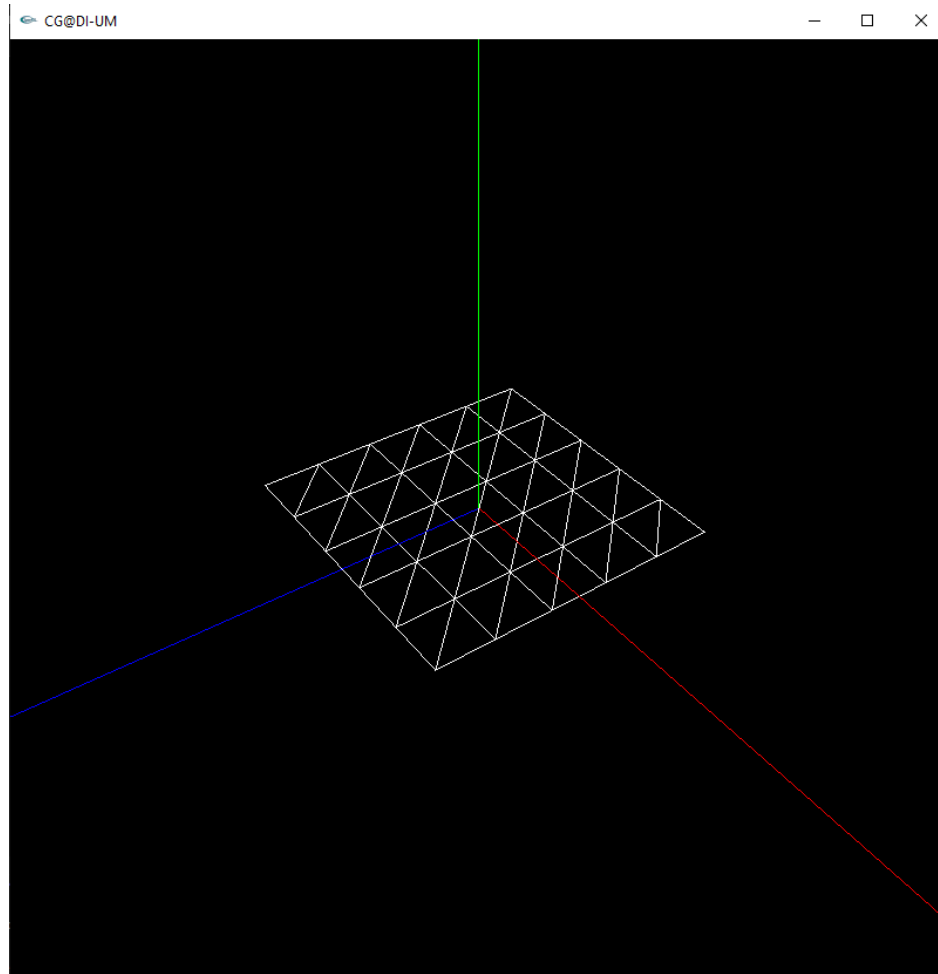


Figure 3: Esquema de formação de um plano com uma unidade de comprimento e 5 divisões por eixo.

### 3.4 Caixa

Uma caixa é um modelo de um cúbico com a particularidade de que pode ser dado o número de divisões que este deve apresentar por aresta.

Uma vez que as dimensões de uma caixa são variáveis, é necessário saber as coordenadas cartesianas, para cada divisão. Desta forma é utilizada 3 variáveis que representam as dimensões de X,Y,Z de cada subdivisão da caixa.

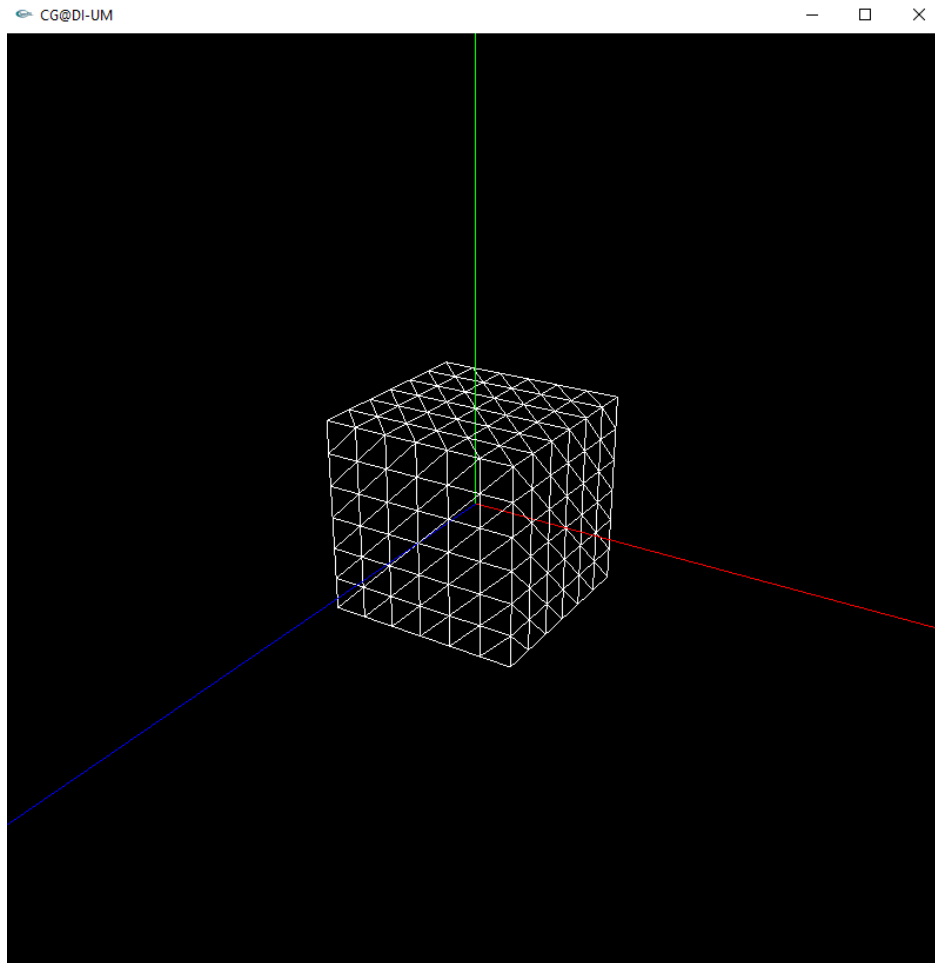


Figure 4: Esquema de formação de uma Caixa simétrica com 6 divisões por aresta.

## 4 Generator

Uma vez definidos e analisados os métodos de desenho para cada uma das figuras, o próximo passo foi a implementação do gerador. O generator é simplesmente um programa de linha de comandos, que recebe os argumentos definidos no enunciado e gera os ficheiros .3d. Caso os argumentos estejam mal formatados, uma mensagem de erro é escrita no ecrã.

### 4.1 Ficheiros .3d

É nos ficheiros .3d que são armazenados os vértices das figuras, entre outras informações relevantes para a engine. Primeiramente, os ficheiros .3d são escritos em **texto**, uma vez que isso nos facilita o parsing do ficheiro.

A estrutura de um ficheiro .3d é a seguinte:

- É escrito o número de vértices da figura.
- Para cada vértice, é escrita cada coordenada X,Y,Z, respetivamente, separados por espaços.

## 5 Engine

A engine é um programa que como argumento recebe um ficheiro xml com a estrutura descrita no enunciado. Relativamente, à estrutura de dados utilizada como está representada a seguir, temos o primeiro vetor que são os ficheiros 3d e o segundo vector desta estrutura é relativo aos vértices de cada um desses ficheiros 3d.

```
vector<vector<Vertex>> vertices;
```

Após a leitura do ficheiro xml e de todas as figuras 3d estarem em memória, o programa trata de as desenhar, fazendo uma simples tradução dos seus vértices. Basicamente, isso é tratado pela função `desenha()` como está representado a seguir. O primeiro ciclo `for` tem como objetivo iterar todas as figuras, mais especificamente, iterar cada um dos ficheiros 3d. Em relação ao segundo `for` este serve para iterar todos os pontos de um ficheiro 3d.

```
void desenha() {
    glBegin(GL_TRIANGLES);
    glColor3f(0.0f, 0.0f, 1.0f);
    for (int i = 0; i < vertices.size(); i++) {
        vector<Vertex> raquitico = vertices.at(i);
        for (int j = 0; j < raquitico.size(); j++) {
            glVertex3f(raquitico.at(j).getVertexX(),
                      raquitico.at(j).getVertexY(),
                      raquitico.at(j).getVertexZ());
        }
    }
    glEnd();
}
```

## 5.1 Parsing

O *parsing* do xml é feito através da biblioteca *tinysql2*, disponível no projeto enviado. O parser foi feito para extrair o conteúdo dos ficheiros xml, como a informação acerca da câmara que contém valores relativos à posição da mesma, de onde estamos olhar e também do up que é a inclinação da câmara. Além disso, também extraímos informação do nome dos ficheiros 3d. No mesmo ficheiro do parser também temos a função que lê os ficheiros 3d que são gerados pelo generator.

## 6 Conclusão

Concluindo, consideramos que conseguimos realizar o que foi pedido, nesta primeira fase do trabalho, tanto a parte de geração como a de desenho de gráficos, queremos ainda reforçar a solidificação dos conhecimentos até agora lecionados assim como a aquisição de conhecimentos relativos à linguagem C++ e às várias bibliotecas gráficas utilizadas.

## References

- [1] tinysql2 - <https://github.com/leethomason/tinysql2>.