

---

# Computação Gráfica 1ª Parte

---

## TRABALHO REALIZADO POR:

JOÃO FIGUEIREDO MARTINS PEIXE DOS SANTOS

FRANCISCO ALVES ANDRADE

LUÍS FILIPE CRUZ SOBRAL

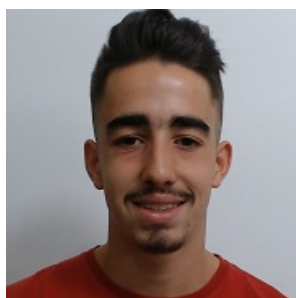
PAULO SILVA SOUSA



A89520 João Santos



A89474 Luís Sobral



A89465 Paulo Sousa



A89513 Francisco Andrade

GRUPO 14  
PROJETO SD  
2020/2021  
UNIVERSIDADE DO MINHO

# Índice

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Funcionalidades Implementadas</b>	<b>1</b>
<b>3</b>	<b>Estrutura do Projeto</b>	<b>1</b>
3.1	Aplicações . . . . .	1
3.2	Módulos . . . . .	1
3.3	Classes . . . . .	1
3.4	Ferramentas utilizadas . . . . .	2
<b>4</b>	<b>Primitivas gráficas</b>	<b>3</b>
4.1	Plano . . . . .	3
4.2	Caixa . . . . .	4
4.3	Cone . . . . .	5
4.3.1	Base . . . . .	5
4.3.2	Camada lateral . . . . .	6
4.4	Esfera . . . . .	7
<b>5</b>	<b>Generator</b>	<b>8</b>
5.1	Definição . . . . .	8
5.2	Funcionalidades . . . . .	8
<b>6</b>	<b>Engine</b>	<b>9</b>
6.1	Demonstração . . . . .	9
<b>7</b>	<b>Modelos</b>	<b>10</b>
7.1	Plano . . . . .	10
7.2	Caixa . . . . .	10
7.3	Cone . . . . .	10
7.4	Esfera . . . . .	11
<b>8</b>	<b>Conclusão</b>	<b>11</b>

---

# 1 Introdução

Nesta primeira fase do projeto, realizado no âmbito da disciplina de Computação Gráfica, foi-nos proposto o desenvolvimento de duas aplicações: o gerador dos arquivos com as informações dos modelos, e o engine que lê um arquivo de configuração, escrito em XML, e exibe os modelos. O projeto é desenvolvido na linguagem C++ e recorrendo à biblioteca OpenGL.

## 2 Funcionalidades Implementadas

- Plano
- Caixa
- Cone
- Esfera

## 3 Estrutura do Projeto

### 3.1 Aplicações

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

- 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 que representam cada objeto são representadas recorrendo à biblioteca OpenGL.

### 3.2 Módulos

- Generator: É responsável por gerar as figuras necessárias, utilizando para isso as classes Shape e Point. O respetivo ficheiro está encarregue de calcular todos os pontos para calcular os triângulos que vão representar o objeto pretendido e gravá-los no ficheiro desejado.
- Engine: É responsável por interpretar e representar o ficheiro XML que contém o nome dos ficheiros gerados pelo generator numa modelação 3D.

### 3.3 Classes

- Point: O Point.cpp é o ficheiro onde está definida a classe Point. Esta classe traduz a representação de um ponto num referencial para código, sendo este representado por 3 floats (X,Y,Z).
- Shape: Esta classe é responsável pela criação de figuras, os pontos que representam uma figura são armazenados num vetor.

---

### **3.4 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, recorreremos também à biblioteca em que assentam as funcionalidades gráficas do projeto, a OpenGL.

---

## 4 Primitivas gráficas

### 4.1 Plano

Para obter um plano paralelo ao plano XZ e centrado na origem, procedemos à criação de 2 triângulos sendo que estes terão um lado em comum criando assim a ilusão de um quadrado.

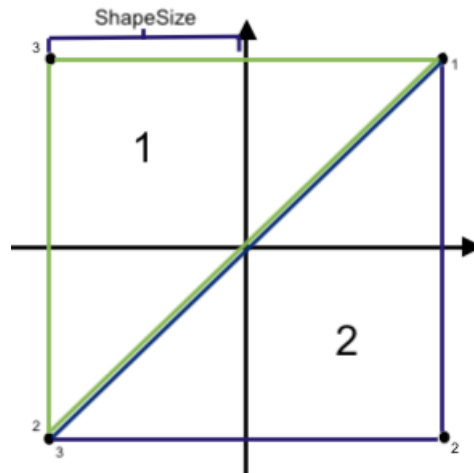


Figure 1: Face superior do plano

O centro do plano será a origem do referencial, ponto  $(0,0)$ , e os vértices do triângulo terão as seguintes coordenadas, onde `shapeSize` é metade do `size` fornecido, ou seja, metade do lado do quadrado:

- Triângulo 1:
  - Ponto 1:  $(\text{shapeSize}, 0, \text{shapeSize})$
  - Ponto 2:  $(-\text{shapeSize}, 0, -\text{shapeSize})$
  - Ponto 3:  $(-\text{shapeSize}, 0, \text{shapeSize})$
- Triângulo 2:
  - Ponto 1:  $(\text{shapeSize}, 0, \text{shapeSize})$
  - Ponto 2:  $(\text{shapeSize}, 0, -\text{shapeSize})$
  - Ponto 3:  $(-\text{shapeSize}, 0, -\text{shapeSize})$

---

## 4.2 Caixa

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

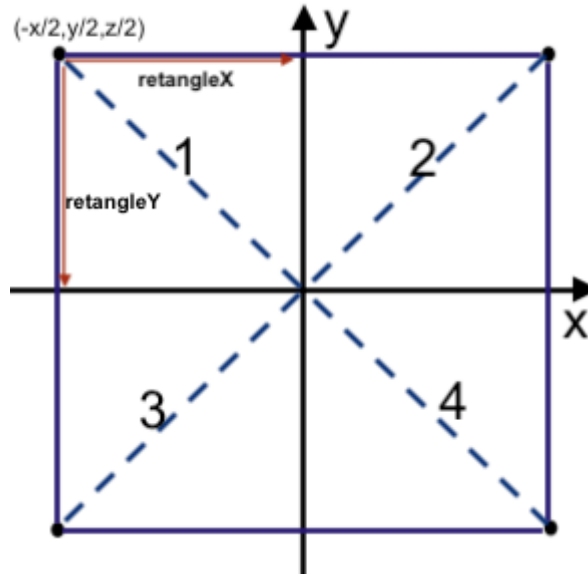


Figure 2: Face frontal da caixa

A figura 2, mostra a representação da face frontal de uma caixa com 2 divisões. Para calcular os vértices de uma face com  $X$  comprimento,  $Y$  largura e  $Z$  constante, usamos a seguinte algoritmo:

- $xLine\ inicial = -X/2$
- $yLine\ inicial = Y/2$   
Sendo o ponto inicial  $(-X/2, Y/2, Z/2)$ , os seguintes serão calculados através de somas de vetores.
- $rectangleX = X / 2$ , ou  $X / N$ (onde  $n$  é o número de divisões), este valor é incrementado ao valor de  $XLine$  inicial para calcular os restantes pontos na mesma linha.
- $rectangleY = Y / 2$ , ou  $Y / N$ (onde  $n$  é o número de divisões), este valor é incrementado ao valor de  $YLine$  inicial para calcular os restantes pontos na mesma coluna.

---

Cada vértice é dado por:

$$(xLine\_inicial + c * rectangleX, yLine\_inicial + l * rectangleY, Z)$$

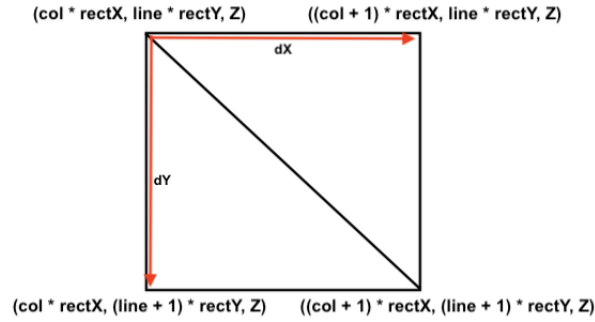


Figure 3: Pontos de um quadrado

Após calcular os vértices dos triângulos, estes são representados segundo a regra da mão direita. O exemplo da figura é para a face frontal da caixa, no entanto, para outras faces usamos o mesmo algoritmo e apenas variamos coordenadas diferentes.

### 4.3 Cone

O desenho do cone está dividido em duas etapas:

- Base
- Camada lateral

#### 4.3.1 Base

Ao desenhar a base é necessário variar o ângulo  $\alpha$  entre 0 e  $2\pi$ . O círculo que define a base será dividido em  $N$  fatias e para o recriar iteramos um inteiro  $i$  entre 0 e  $N$  e multiplicamos  $i$  pelo ângulo  $SliceAngle$ .

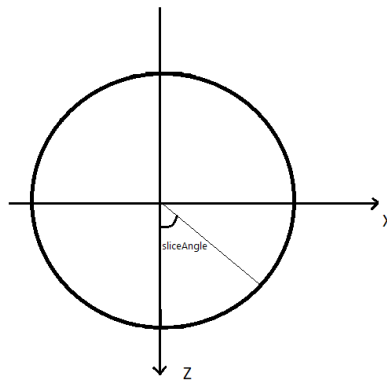


Figure 4: Base do Cone

$$SliceAngle = \frac{2\pi}{N}$$

$$\alpha = i * sliceAngle$$

---

Seguidamente, é possível desenhar o triângulo correspondente a cada fatia definido pelos seguintes pontos:

- $(\text{raio} * \cos(\alpha + \text{sliceAngle}), 0, \text{raio} * \sin(\alpha + \text{sliceAngle}))$
- $(0,0,0)$
- $(\text{raio} * \cos(\alpha), 0, \text{raio} * \sin(\alpha))$

#### 4.3.2 Camada lateral

De forma a desenhar a face lateral do cone é necessário construir vários triângulos. Calculamos os pontos de uma slice com base numa determinada camada(*height\_now*), na camada seguinte(*height\_next*) e nos seus respetivos raios (*radius\_now* e *radius\_next*). Seja *k* um número inteiro menor que o número de camadas e maior ou igual a zero que incrementa a cada iteração de um ciclo. Seja *height* a altura do cone e *radius* o raio da base do cone.

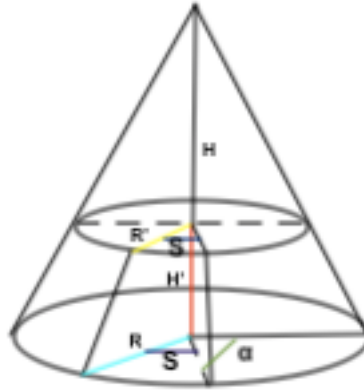


Figure 5: Cone

- Para  $k = 0$ :

$$\text{height\_now} = 0$$

$$\text{height\_next} = \frac{\text{height}}{\text{stacks}}$$

$$\text{radius\_now} = \text{radius}$$

$$\text{radius\_next} = \text{radius} - \frac{\text{radius}}{\text{stacks}}$$

Para todas as outras iterações os valores de *height\_now*, *height\_next*, *radius\_now* e *radius\_next* são atualizados para novos valores.

$$\text{height\_now} = \text{height\_next}$$

$$\text{height\_next} += \frac{\text{height}}{\text{stacks}}$$

$$\text{radius\_now} = \text{radius\_next}$$

$$\text{radius\_next} -= \frac{\text{radius}}{\text{stacks}}$$

Findada a ultima iteração onde *k* atinge o número de stacks passamos a desenhar a fatia seguinte, colocando novamente os valores iniciais.



De modo a criar uma esfera necessitamos de dividir a esfera em stacks e slices obtendo, assim, diversos quadriláteros/triângulos constituídos apenas por triângulos, como podemos observar em baixo.

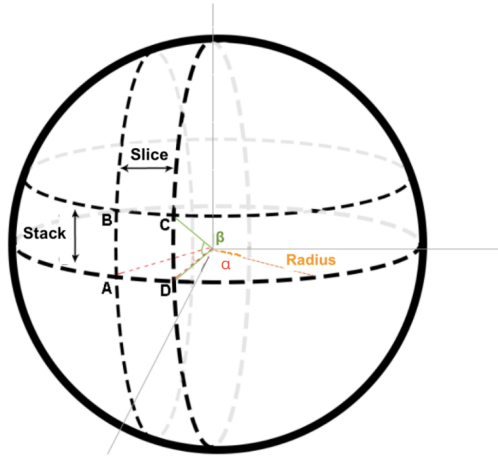


Figure 6: Esfera

Para representar os pontos de cada triângulo utilizamos coordenadas esféricas que obtivemos executando as seguintes equações:

$$radius = RaiodaEsfera$$

$$x = radius \times \cos(\alpha) \times \cos(\beta)$$

$$y = radius \times \sin(\beta)$$

$$z = radius \times \sin(\alpha) \times \cos(\beta)$$

Analisando as expressões acima verificamos que os pontos são obtidos através da variação dos ângulos  $\alpha$  e  $\beta$ . Alterando o valor de  $\alpha$  mudamos a slice em que nos encontramos, se alterarmos o valor de  $\beta$  mudamos a stack em que nos encontramos.

- Sejam  $i$  e  $j$  números inteiros maiores ou iguais a zero e menores que o número de slices(fatias) e stacks(camadas), respetivamente:

$$\alpha_i = i \times \frac{2 \times \pi}{slices}$$

$$\beta_j = j \times \frac{\pi}{stacks} - \frac{\pi}{2}$$

Assim, juntando os pontos gerados por  $\alpha_i$  e  $\beta_j$  consecutivos, é possível construir os triângulos necessários para a implementação da esfera.

---

## 5 Generator

### 5.1 Definição

Para o desenvolvimento da totalidade das figuras geométricas apresentadas no projeto, foram utilizados apenas triângulos. O *Generator* é responsável por efetuar todos os cálculos necessários para a concepção da forma pretendida, isto é, calcular os três vértices de todos os triângulos que integram a constituição do objeto. Isto deve-se ao facto de ser necessário o cálculo de todos os vértices dos respetivos triângulos para ser possível a visualização das figuras.

### 5.2 Funcionalidades

Foram desenvolvidas várias formas geométricas 3D diferentes. Estas podem ser conjuradas pelo utilizador através do *Generator*:

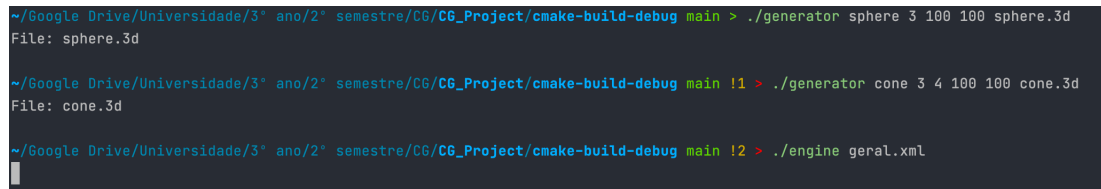
- Plano  
Cria um plano paralelo a xOz, recebendo como argumento o tamanho pretendido pelo utilizador;
- Caixa  
Cria um paralelepípedo com as dimensões fornecidos como argumento: Comprimento, Largura, Altura, *Divisions* (número de quadrados por face);
- Cone  
Cria um cone com as dimensões fornecidos como argumento: Raio, Altura, n<sup>o</sup> de slices (divisão vertical), n<sup>o</sup> de stacks (divisão horizontal) e o ângulo de cada slice;
- Esfera  
Cria uma esfera com as dimensões fornecidos como argumento: Raio, n<sup>o</sup> de slices e n<sup>o</sup> de stacks.

---

## 6 Engine

Após o cálculo dos vértices dos triângulos por parte do *generator*, o programa consegue iniciar a fase seguinte de representação 3D do objeto pretendido. Utiliza, para tal o *Engine* efetua a leitura do ficheiro XML que lhe é apresentado como argumento (fornecendo todo o *path* até este). Desta forma, é recolhida a listade ficheiros que deverão ser exectados e, utilizando o mesmo path, é realizada uma análise dos respetivos ficheiros de cada objeto, passando assim à representação da sua forma no ecrã.

### 6.1 Demonstração



```
~/Google Drive/Universidade/3º ano/2º semestre/CG/CG_Project/cmake-build-debug main > ./generator sphere 3 100 100 sphere.3d
File: sphere.3d

~/Google Drive/Universidade/3º ano/2º semestre/CG/CG_Project/cmake-build-debug main !1 > ./generator cone 3 4 100 100 cone.3d
File: cone.3d

~/Google Drive/Universidade/3º ano/2º semestre/CG/CG_Project/cmake-build-debug main !2 > ./engine geral.xml
```

Figure 7: Invocação do generator e do engine por terminal



```
<scene>
  <model file="../files/sphere.3d" />
  <model file="../files/cone.3d" />
</scene>
```

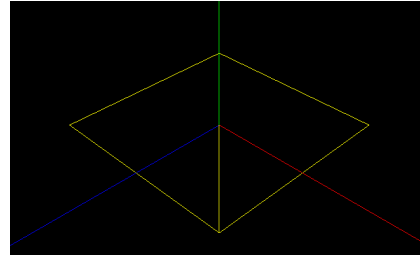
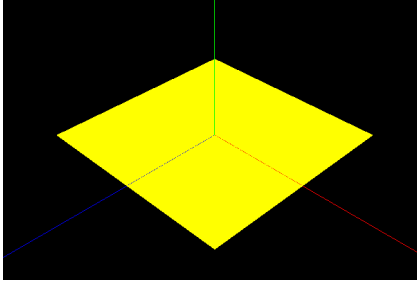
Figure 8: XML exemplo para representar as figuras calculadas pelo generator

---

## 7 Modelos

### 7.1 Plano

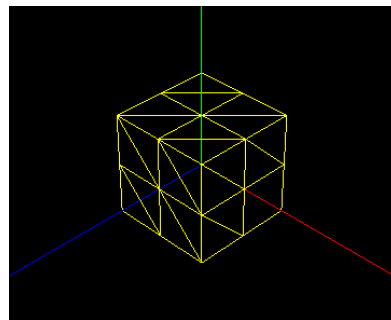
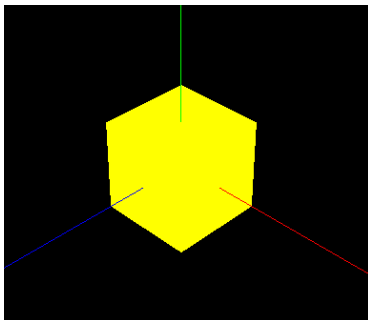
Plano standart: 2 triângulos paralelos a  $xOz$ .



### 7.2 Caixa

Caixa gerada com os seguintes argumentos:

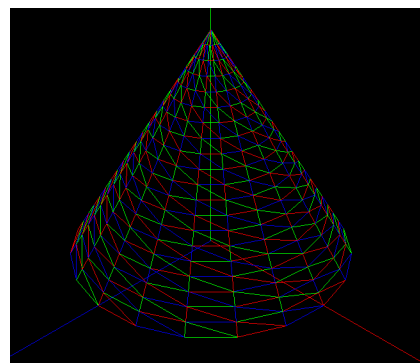
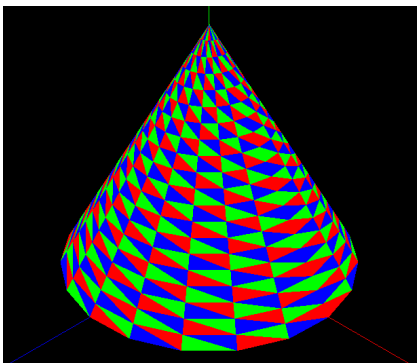
- X: 1
- Y: 1
- Z: 1
- Divisions: 2



### 7.3 Cone

Cone gerado com os seguintes argumentos:

- Radius: 2
- Height: 3
- Slices: 20
- Stacks: 20

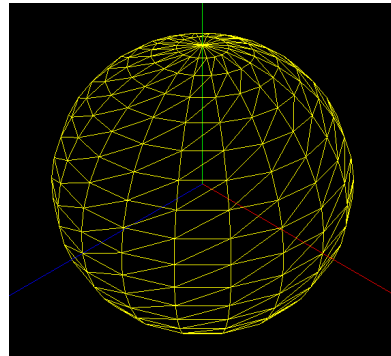
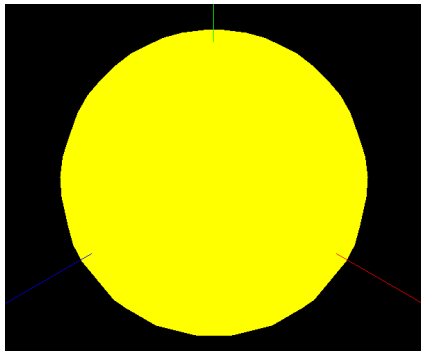


---

## 7.4 Esfera

Esfera gerada com os seguintes argumentos:

- Radius: 2
- Slices: 20
- Stacks: 20



## 8 Conclusão

Durante esta primeira fase entramos em contacto com situações difíceis que não encontramos durante as várias aulas. No entanto, olhamos para isto de forma positiva pois levou-nos a ter uma maior atenção àquilo que fazíamos e um maior empenho na realização deste trabalho.

A presente fase permitiu a consolidação da matéria lecionada nas aulas, nomeadamente sobre a utilização e domínio de ferramentas e conceitos associados à Computação Gráfica, utilizado para isso *OpenGL* e *Glut* e, ainda, permitiu a exploração de novos domínios como a leitura de ficheiros xml.