



Universidade do Minho  
Escola de Engenharia

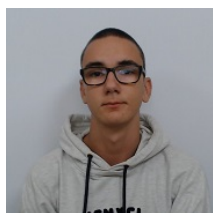
# Computação Gráfica

## 1ª Fase

João Pedro da Santa Guedes A89588  
Luís Pedro Oliveira de Castro Vieira A89601  
Carlos Miguel Luzia Carvalho A89605  
Bárbara Ferreira Teixeira A89610



A89588



A89601



A89605



A89610

14 de novembro de 2021

# Índice

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Contextualização . . . . .	1
1.2	Resumo . . . . .	1
<b>2</b>	<b>Arquitetura do Projeto</b>	<b>2</b>
2.1	Funcionalidades Implementadas . . . . .	2
2.2	Aplicações . . . . .	2
2.2.1	Gerador . . . . .	2
2.2.2	Motor . . . . .	3
2.3	Classes . . . . .	3
2.3.1	Point . . . . .	3
2.4	Ferramentas extra . . . . .	3
2.4.1	TinyXML2 . . . . .	3
<b>3</b>	<b>Primitivas Geométricas</b>	<b>4</b>
3.1	Plano . . . . .	4
3.1.1	Cálculo dos Vértices . . . . .	4
3.2	Caixa . . . . .	5
3.2.1	Cálculo dos Vértices . . . . .	5
3.3	Esfera . . . . .	8
3.3.1	Cálculo dos Vértices . . . . .	8
3.4	Cone . . . . .	11
3.4.1	Cálculo dos Vértices . . . . .	11
<b>4</b>	<b>Generator</b>	<b>15</b>
4.1	Descrição . . . . .	15
4.2	Utilização . . . . .	15
4.3	Demonstração . . . . .	15
<b>5</b>	<b>Engine</b>	<b>16</b>
5.1	Descrição . . . . .	16
5.2	Utilização . . . . .	16
5.3	Demonstração . . . . .	17

<b>6</b>	<b>Extras</b>	<b>18</b>
6.1	Câmara . . . . .	18
6.2	Representação da Primitiva . . . . .	18
<b>7</b>	<b>Apresentação dos Modelos</b>	<b>19</b>
7.1	Plano . . . . .	19
7.2	Caixa . . . . .	20
7.3	Esfera . . . . .	21
7.4	Cone . . . . .	22
<b>8</b>	<b>Conclusão e Trabalho Futuro</b>	<b>23</b>

# 1 Introdução

## 1.1 Contextualização

No âmbito da Unidade Curricular de Computação Gráfica foi nos pedido o desenvolvimento de um projeto dividido em 4 fases, com o objetivo de através de ferramentas como o C++ e o OpenGL criar um mini mecanismo 3D baseado num cenário gráfico. O relatório presente é referente à primeira fase, na qual o objetivo é a criação de algumas primitivas gráficas.

## 1.2 Resumo

Nesta primeira fase do desenvolvimento do projeto foi proposta a criação de duas aplicações essenciais ao funcionamento do mesmo, sendo estas:

- Gerador(Generator) - Gerar a informação essencial ao modelo como os vértices das figuras.
- Motor(Engine) - Ler um ficheiro XML e exibir os modelos pretendidos.

As primitivas gráficas a elaborar nesta fase do projeto serão o Plano, o Cubo, a Esfera e o Cone, sendo que é o objetivo da mesma gerar e exibir as mesmas primitivas.

Ao longo do presente relatório será explicado o procedimento para a construção de cada uma destas primitivas.

## 2 Arquitetura do Projeto

Na presente secção passaremos a demonstrar a utilidade

### 2.1 Funcionalidades Implementadas

- Plano: *plane side file.3d*

Gera um quadrado centrado na origem no plano XOZ.

- Caixa: *box X Y Z divisions file.3d*

Gera uma caixa centrada na origem com as dimensões X, Y e Z e o número de divisões pretendidas em cada face, dadas pelo utilizador.

- Esfera: *sphere radius slices stacks file.3d*

Gera uma esfera centrada na origem com raio, divisões e stacks dadas pelo utilizador.

- Cone: *cone radius height slices stacks file.3d*

Gera um cone centrado na origem com raio da base, altura, divisões e stacks dadas pelo utilizador.

### 2.2 Aplicações

Este projeto está dividido em duas aplicações que se complementam: o Gerador e o Motor, sendo este último auxiliado pelo Gerador. Este está incumbido de fazer a tradução de primitivas gráficas para conjuntos de vértices armazenados num ficheiro. Já o Motor vai ler o ficheiro XML contendo referências a ficheiros organizados pelo seu auxiliar representando assim as figuras recorrendo á biblioteca OpenGL.

#### 2.2.1 Gerador

**gerador.cpp** é onde estão definidas as estruturas das diferentes formas geométricas de forma a gerar os respetivos vértices. No fundo calcula todos os pontos necessários para criar os triângulos que dão vida às figuras desejadas. Este módulo surge como complementar ao motor.

### 2.2.2 Motor

**engine.cpp** é onde se encontram as funcionalidades principais do projeto, sendo responsável pela representação e interpretação do ficheiro XML. Permite a apresentação de uma janela exibindo os modelos pretendidos e ainda a interação com estes.

## 2.3 Classes

Como forma de simplificar e facilitar a implementação das funcionalidades acima mencionadas, decidimos criar uma classe que facilitasse o armazenamento da informação em memória. Surgiu assim a classe Ponto representativa de cada ponto necessário à construção das primitivas gráficas.

### 2.3.1 Point

O Point.cpp é a classe que traduz a representação de um ponto no referencial em código, um ponto (X,Y,Z) representado com 3 floats.

## 2.4 Ferramentas extra

### 2.4.1 TinyXML2

Para além do OpenGL onde assentam as funcionalidades gráficas do projeto recorreremos também à TinyXML2, uma ferramenta utilizada para fazer o parsing dos ficheiros XML de modo a explorar o seu conteúdo.

## 3 Primitivas Geométricas

### 3.1 Plano

Um plano é composto por dois triângulos que partilham dois vértices comuns, sendo que as suas dimensões são dadas pelos valores de  $X$  e  $Z$ . Como referido anteriormente, o plano formado será centrado na origem e contido no plano  $XOZ$ .

No entanto, como pretendemos que seja um plano visível de cima e de baixo, serão necessários quatro triângulos.

#### 3.1.1 Cálculo dos Vértices

As coordenadas dos vértices apenas variam em  $X$  e  $Z$ . Como o utilizador indica o tamanho que pretende para o seu plano, sabemos que estas coordenadas serão metade do tamanho pretendido, visto que o plano será centrado na origem.

De modo a podermos observar a face do plano quer de cima, quer de baixo, é necessário ter em conta a ordem da criação dos diferentes vértices dos triângulos. Para tal temos em conta a regra da mão direita, isto é, apontando o polegar no sentido em que pretendemos que a face seja visível, ao fecharmos a mão podemos observar a ordem pela qual os vértices devem ser criados.

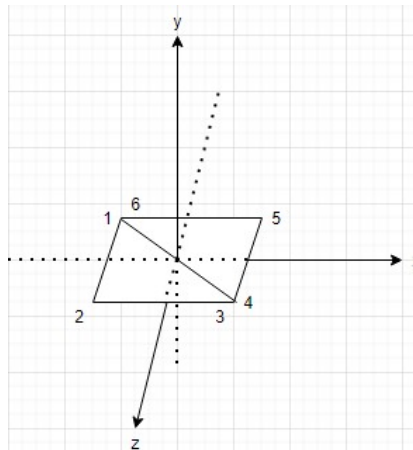


Figure 1: Diagrama do Plano

Assim ao seguirmos a ordem de desenho dos vértices 1-2-3 para o primeiro triângulo e 4-5-6 para o segundo triângulo, geramos um plano visível de cima. De forma a gerar um plano visível de baixo seguimos a mesma regra, logo a ordem a seguir será 2-1-3 para o primeiro triângulo e 5-4-6 para o segundo.

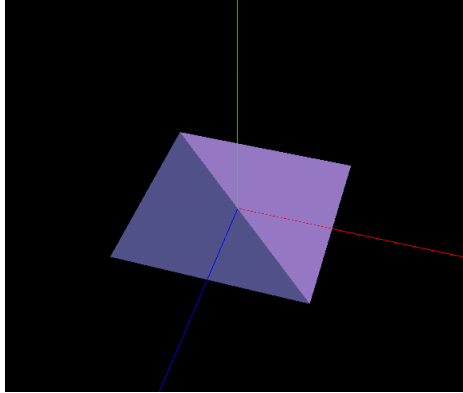


Figure 2: Resultado Final do Plano

## 3.2 Caixa

Uma caixa é composta por seis faces, sendo necessário quatro parâmetros para a sua construção:  $X$ ,  $Y$ ,  $Z$  e número de divisões de cada face. Estes parâmetros determinam respectivamente o comprimento, largura e altura da caixa, bem como o número de vezes que a face se vai subdividir.

### 3.2.1 Cálculo dos Vértices

De forma a centrar a figura na origem usamos metade dos valores de  $X, Y, Z$  fornecidos como coordenadas. Importante notar que a construção da caixa segue o mesmo raciocínio da construção do plano, isto é, cada divisão é composta por dois triângulos com dois vértices em comum. Apesar disso, o cálculo de todos os vértices necessários para construir uma caixa é algo mais complexo que a construção de um plano.

Algo que notámos no início da nossa resolução para a construção da caixa foi que poderíamos emparelhar as faces da mesma, isto é, a cada 2 faces as coordenadas vão ser iguais mudando apenas os sinais das mesmas. Assim passaremos a designar por "*frente*" e "*trás*" as faces contidas no plano  $YOX$ ,



passaremos a designar as faces contidas no plano XOZ por "*cima*" e "*baixo*" e, por fim, designaremos por "*direita*" e "*esquerda*" as faces contidas no plano ZOY.

Notámos também que a cada 2 faces existe sempre uma coordenada que não é modificada a não ser o seu sinal, não sendo necessários cálculos na respetiva coordenada quando estamos a tratar das divisões. Assim, nas faces da frente e de trás da caixa, a coordenada Z permanece igual para todos os pontos da respetiva face, nas faces de cima e de baixo permanece a coordenada Y igual e, por fim, nas faces da direita e da esquerda permanece a coordenada X igual.

Trataremos como exemplo para a demonstração do cálculo dos vértices da caixa, a face da frente da mesma.

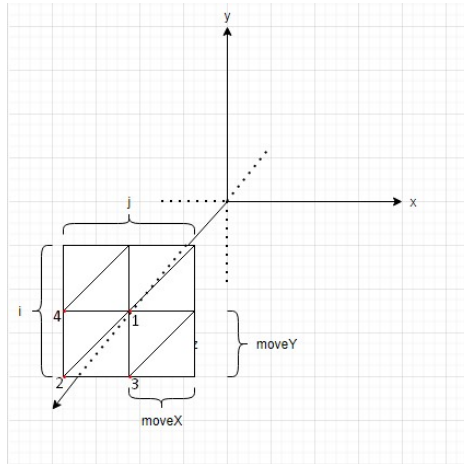


Figure 3: Diagrama da Face da frente da Caixa

De forma a podermos construir cada uma das faces com o número de divisões pretendidas pelo utilizador devemos percorrer cada uma delas em duas direções: **i** e **j**. Isto permite-nos dividir não só na horizontal de cada face, como também na vertical.

De forma a dividirmos cada face é necessário que os vértices sofram um desvio em cada uma das direções, desvio esse que é calculado de forma genérica como  $\text{moveX}||\text{Y}||\text{Z} = \text{X}||\text{Y}||\text{Z} / \text{divisões}$ . Assim dividimos cada lado da face de igual forma.

No caso da face da frente, os vértices são calculados da seguinte forma:

Para todo  $i$  e  $j$  menor que o número de divisões pretendidas:

- $v1 = ( -x + moveX + (j * moveX) , -y + moveY + (i * moveY) , z )$
- $v2 = ( -x + (j * moveX) , -y + (i * moveY) , z )$
- $v3 = ( -x + moveX + (j * moveX) , -y + (i * moveY) , z )$
- $v4 = ( -x + (j * moveX) , -y + moveY + (i * moveY) , z )$

Onde, neste caso,  $j$  percorre o eixo do X e  $i$  incrementa a altura após percorrer toda a linha.

Para que seja visível na orientação suposta desenhamos os vértices de acordo com a regra da mão direita, já explicada em cima. Obtemos então o primeiro triângulo visível se seguirmos a ordem 1-2-3 e o segundo seguindo a ordem 1-4-2.

O raciocínio para as restantes faces é igual, tendo sempre em consideração as propriedades acima mencionadas acerca das coordenadas que são modificadas em cada par de faces.

O resultado final da caixa gerada é o seguinte:

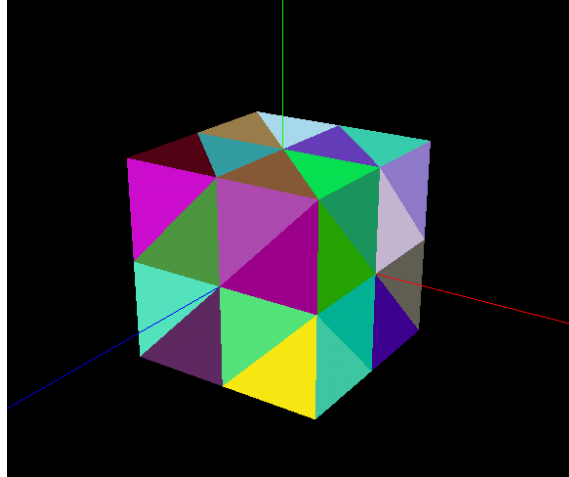


Figure 4: Resultado Final da Caixa

### 3.3 Esfera

A esfera é um sólido geométrico cujos pontos que formam uma superfície curva contínua se encontram equidistantes do centro. Para a definir é necessário ter como parâmetros o raio, o número de fatias (slices) e o número de camadas (stacks).

#### 3.3.1 Cálculo dos Vértices

Ao idealizarmos o cálculo dos vértices da esfera tentamos utilizar a mesma técnica relativamente ao plano e à caixa, isto é, tratamos a esfera como algo que possamos percorrer perante as suas slices (fatias) e as suas camadas (stacks) e com base em deslocações ao longo de ambas as linhas (horizontal e vertical) calcular os vértices necessários para a construção dos triângulos.

No entanto, ao contrário das primitivas gráficas previamente explicadas, não podemos trabalhar com estes valores como antes, ou seja, as coordenadas dos vértices terão de ser esféricas, isto é, vão depender da variação do ângulo com os eixos para podermos obter o resultado pretendido.

Assim, é necessário ter em consideração alguns valores que nos permitem alcançar este objetivo, tais como:

- $\text{moveL} = (2 * M\_PI) / \text{slices}$
- $\text{moveH} = M\_PI / \text{stacks}$

De forma geral os valores das coordenadas terão a seguinte forma:

- $x = \text{radius} * \cos(\alpha) * \sin(\beta)$
- $y = \text{radius} * \sin(\alpha)$
- $z = \text{radius} * \cos(\alpha) * \cos(\beta)$

Estes ângulos *alpha* e *beta* serão os formados pelos pontos que constituem cada triângulo.

De forma a gerarmos a esfera começamos por desenhar slice a slice a partir da "*base*" da esfera, isto é, partimos do centro e quer, no sentido de Y, quer no sentido de -Y, vamos desenhando as fatias todas até passarmos à stack seguinte.

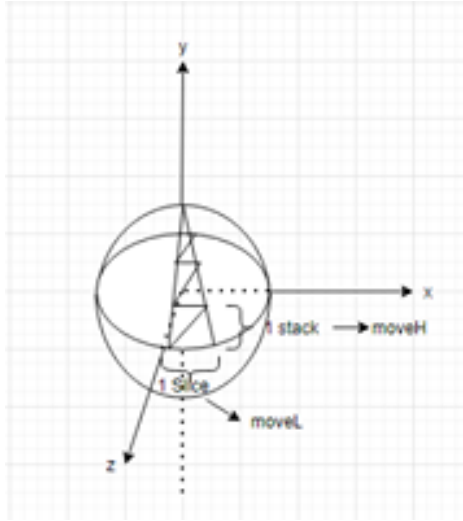


Figure 5: Diagrama representativo de meia slice de uma Esfera

De forma a podermos explicar o processo de cálculo vamos tomar como exemplo a parte superior da esfera, sendo que o processo para a parte inferior é o mesmo, sendo necessário apenas trocar o sinal da coordenada Y.

Assim, calculamos os seguintes pontos:

- $x1 = \text{radius} * \cos(\text{moveH} * i) * \sin(\text{moveL} * j)$
- $y1 = \text{radius} * \sin(\text{moveH} * i)$
- $z1 = \text{radius} * \cos(\text{moveH} * i) * \cos(\text{moveL} * j)$
- $x2 = \text{radius} * \cos(\text{moveH} * i) * \sin(\text{moveL} + \text{moveL} * j)$
- $y2 = \text{radius} * \sin(\text{moveH} * i)$
- $z2 = \text{radius} * \cos(\text{moveH} * i) * \cos(\text{moveL} + \text{moveL} * j)$
- $x3 = \text{radius} * \cos(\text{moveH} + \text{moveH} * i) * \sin(\text{moveL} * j)$
- $y3 = \text{radius} * \sin(\text{moveH} + \text{moveH} * i)$
- $z3 = \text{radius} * \cos(\text{moveH} + \text{moveH} * i) * \cos(\text{moveL} * j)$

- $\text{if}(i \neq (\text{stacks} / 2) - 1) \text{ radius} * \cos(\text{moveH} + \text{moveH} * i) * \sin(\text{moveL} + \text{moveL} * j)$
- $\text{if}(i \neq (\text{stacks} / 2) - 1) \text{ radius} * \sin(\text{moveH} + \text{moveH} * i)$
- $\text{if}(i \neq (\text{stacks} / 2) - 1) \text{ radius} * \cos(\text{moveH} + \text{moveH} * i) * \cos(\text{moveL} + \text{moveL} * j)$

Estes são os pontos necessários para a construção dos triângulos que compõe a nossa esfera. Como mencionado acima, utilizando a mesma linha de pensamento que na caixa e no plano, iterando a esfera em duas direções:  $i$  no sentido das stacks, ou seja, vertical, e  $j$  no sentido das slices, ou seja, horizontal.

No entanto não percorremos o número total de stacks, uma vez que construímos ao mesmo tempo quer a parte superior quer a parte inferior da esfera.

Outra coisa a ter em atenção, e que está acima mencionada, é a condição  $\text{if}(i \neq (\text{stacks} / 2) - 1)$ . Esta condição existe para distinguir as stacks dos topos da esfera, uma vez que nas mesmas não é necessário desenhar com 2 triângulos.

Assim sendo, a ordem correta a seguir, tendo como base a regra da mão direita de forma a que seja visível na direção pretendida, para gerar a slice da parte de cima de uma esfera será:  $(x1, y1, z1) - (x2, y2, z2) - (x3, y3, z3)$  para o primeiro triângulo e, caso não se trate do topo da esfera,  $(x3, y3, z3) - (x2, y2, z2) - (\text{radius} * \cos(\text{moveH} + \text{moveH} * i) * \sin(\text{moveL} + \text{moveL} * j), \text{radius} * \sin(\text{moveH} + \text{moveH} * i), \text{radius} * \cos(\text{moveH} + \text{moveH} * i) * \cos(\text{moveL} + \text{moveL} * j))$  para o segundo triângulo. Caso se trate do topo da esfera, o mesmo não terá um segundo triângulo, apenas o primeiro.

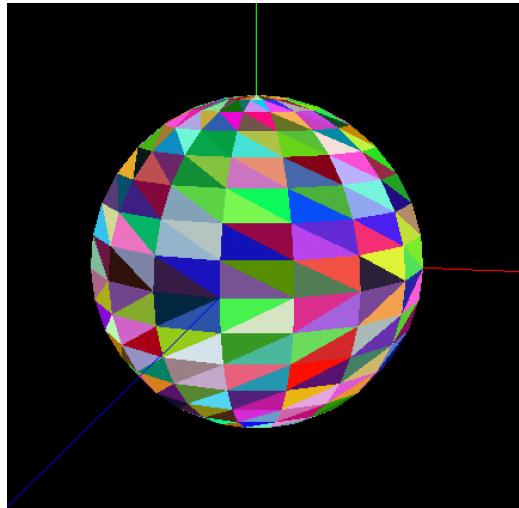


Figure 6: Resultado final de uma Esfera

### 3.4 Cone

O cone, um sólido geométrico é obtido a partir de uma pirâmide cuja base é um polígono regular, recebe como parâmetros o raio da base, a altura pretendida, o número de slices e o número de stacks. Tal como na esfera, quanto maior o número de slices e stacks melhor e mais notável se torna a curvatura do mesmo.

#### 3.4.1 Cálculo dos Vértices

Tal como fizemos com as outras primitivas gráficas, a nossa idealização para o cálculo dos vértices começou por seguir a mesma teoria, ou seja, iríamos iterar, tal como na esfera, pela linha correspondente às slices e pela linha correspondente às stacks.

No entanto, rapidamente nos apercebemo-nos que o output não era aquilo que esperávamos e começámos a procurar o porquê disso. Foi aí que notámos que o cone teria que *afunilar*, ou seja, ele teria que ir encolhendo à medida que vamos percorrendo o eixo dos Y. Ou seja, o cone iria fugir à superfície curva contínua da esfera e passaria a ser uma linha reta que unisse a base ao ponto no Y correspondente ao máximo da altura pretendida.

Concluimos então que à medida que íamos subindo na linha das stacks teríamos que de alguma forma "aproximar" o ponto do eixo dos Y de forma a formar esta linha reta. Para isso decidimos que deveríamos encontrar uma forma de "encurtar" o raio inicial para o cálculo dos vértices.

Tivemos então que alterar a nossa abordagem para o cálculo dos vértices do cone, tendo isto em conta.

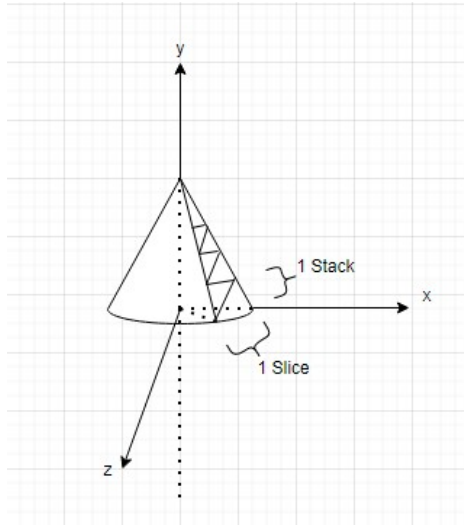


Figure 7: Diagrama representativo de uma slice do Cone

Para representarmos a base do cone fixámos um ponto, neste caso o  $(0,0,0)$  para servir de referência para o resto do cone. Precisamos de ter também em conta o número de slices indicadas no desenho da base. De forma a dividirmos de forma equalitária, dividimos  $360^\circ$  pelo número de fatias.

Assim, considerando que Y será igual para todos, os pontos que formam a base serão calculados da seguinte forma:

- $x1 = \text{radius} * \sin(\text{alfa} + \text{dimSide});$
- $z1 = \text{radius} * \cos(\text{alfa} + \text{dimSide});$
- $x2 = \text{radius} * \sin(\text{alfa});$
- $z2 = \text{radius} * \cos(\text{alfa});$

Onde  $\text{dimSide} = (2 * \text{MPI}) / \text{slices}$ .

De acordo com a regra da mão direita, e de forma a que a base do cone seja visível por baixo, os pontos deverão seguir a ordem:  $(0,0,0) - (x1,0,z1) - (x2,0,z2)$ . Todos os triângulos que compõem a base do cone possuem um ponto comum,  $(0,0,0)$ .

Uma coisa em comum com a esfera que o cone possui é a mesma característica relativamente à última stack, a do topo, ser composta somente por um triângulo e não dois, portanto os vértices são calculados de forma diferente:

- $x1 = (\text{radius} - (i * t)) * \sin(\text{alfa})$ ;
- $z1 = (\text{radius} - (i * t)) * \cos(\text{alfa})$ ;
- $x2 = (\text{radius} - (i * t)) * \sin(\text{alfa} + \text{dimSide})$ ;
- $z2 = (\text{radius} - (i * t)) * \cos(\text{alfa} + \text{dimSide})$ ;

A ordem dos pontos a ser armazenada para posterior interpretação é  $(x1,i*1,z) - (x2,i*1,z2) - (0,(i+1)*1,0)$ , onde  $l = \sqrt{(\text{pow}(h, 2) + \text{pow}(\text{radius}, 2))} / \text{stacks}$ . Este cálculo de  $l$  é o que nos permite subir de forma igual em  $Y$  para podermos definir as stacks.

Por fim, os pontos que formam os triângulos da face do cone são dados pelas seguintes expressões:

- $x1 = (\text{radius} - ((i + 1) * t)) * \sin(\text{alfa} + \text{dimSide})$ ;
- $z1 = (\text{radius} - ((i + 1) * t)) * \cos(\text{alfa} + \text{dimSide})$ ;
- $x2 = (\text{radius} - ((i + 1) * t)) * \sin(\text{alfa})$ ;
- $z2 = (\text{radius} - ((i + 1) * t)) * \cos(\text{alfa})$ ;
- $x3 = (\text{radius} - (i * t)) * \sin(\text{alfa})$ ;
- $z3 = (\text{radius} - (i * t)) * \cos(\text{alfa})$ ;
- $x4 = (\text{radius} - (i * t)) * \sin(\text{alfa} + \text{dimSide})$ ;
- $z4 = (\text{radius} - (i * t)) * \cos(\text{alfa} + \text{dimSide})$ ;



Tendo em conta a regra da mão direita mais uma vez, de forma a que os triângulos formados sejam visíveis na direção pretendida, a ordem pela qual deverão ser guardados será  $(x1,(i+1)*l,z1) - (x2,(i+1)*l,z2) - (x3,i*l,z3)$  para o primeiro triângulo, e  $(x3,i*l,z3) - (x4,i*l,z4) - (x1,(i+1)*l,z1)$  para o segundo.

Em todos estes pontos, o ângulo alfa corresponde ao ângulo do ponto atual e o ângulo  $\alpha + \dimSide$  ao ponto seguinte, possibilitando a iteração em torno de uma circunferência.

Apesar de termos modificado ligeiramente a linha de raciocínio em relação às outras primitivas, continuamos a percorrer  $i$  na linha das stacks, vertical, e  $j$  na linha das slices, horizontal.

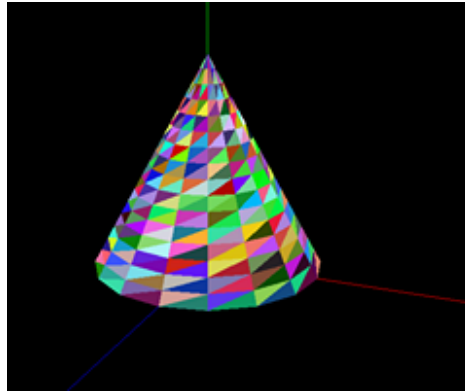


Figure 8: Representação do Cone

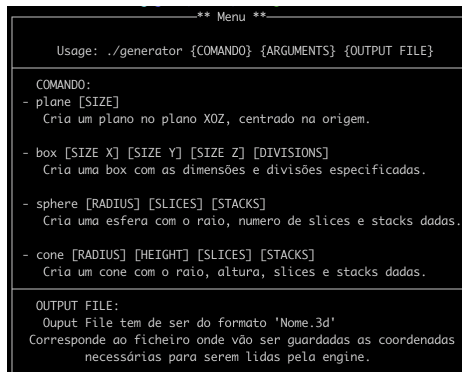
## 4 Generator

### 4.1 Descrição

O *generator* é, tal como o nome indica, responsável por gerar os ficheiros que contém a informação sobre as primitivas geométricas a desenhar, neste caso, sobre o conjunto de vértices que as compõem, conforme os parâmetros recebidos. Estes vértices são meros conjuntos de três pontos que correspondem à representação gráfica de triângulos.

### 4.2 Utilização

De seguida apresentamos um menu de ajuda para o utilizador poder ser capaz de usufruir do *generator* sem qualquer dúvida sobre a forma como o deve fazer e os parâmetros que deve passar ao mesmo. Este pode ser visualizado a partir do comando `./generator`, ou aparecerá quando o input não for o correto.



```

** Menu **

Usage: ./generator {COMANDO} {ARGUMENTS} {OUTPUT FILE}

COMANDO:
- plane [SIZE]
  Cria um plano no plano XOZ, centrado na origem.

- box [SIZE X] [SIZE Y] [SIZE Z] [DIVISIONS]
  Cria uma box com as dimensões e divisões especificadas.

- sphere [RADIUS] [SLICES] [STACKS]
  Cria uma esfera com o raio, numero de slices e stacks dadas.

- cone [RADIUS] [HEIGHT] [SLICES] [STACKS]
  Cria um cone com o raio, altura, slices e stacks dadas.

OUTPUT FILE:
Output File tem de ser do formato 'Nome.3d'
Corresponde ao ficheiro onde vão ser guardadas as coordenadas
necessárias para serem lidas pela engine.
```

Figure 9: Menu de Ajuda do Generator

### 4.3 Demonstração

A utilização do *generator* é deveras simples. Deve ser introduzida a primitiva que se pretende gerar e as dimensões necessárias de acordo com os inputs pedidos para cada uma delas, em conjunto com o nome do ficheiro final resultante onde será armazenada a informação sobre a mesma.

- **Plano**  
plane *side file*
- **Caixa**  
box *x y z divisions file*
- **Esfera**  
sphere *radius slices stacks file*
- **Cone**  
cone *radius height slices stacks file*




Figure 10: Exemplo de utilização do Generator

Em todos os ficheiros, os pontos de cada vértice são armazenados da seguinte forma: *coordX coordY coordZ*\n Para além disso, cada ficheiro começa com o número de pontos necessários à representação da primitiva, como forma de facilitar o parsing.

## 5 Engine

### 5.1 Descrição

O *engine* é responsável por receber ficheiros de configuração escritos em XML, dentro dos quais existe apenas informação sobre quais os ficheiros que foram criados pelo *generator* devem ser lidos e carregados. Assim, após leitura e parsing destes ficheiros, será interpretada e apresentada graficamente os modelos presentes nestes ficheiros.

### 5.2 Utilização

Para lermos e apresentarmos o conteúdo de um ficheiro, devemos passar o mesmo como argumento ao engine. Para tal invocamos o comando: *./engine XML\_file*

Apresentamos abaixo um menu de ajuda possível de invocar utilizando *./engine* ou quando o input for incorreto.

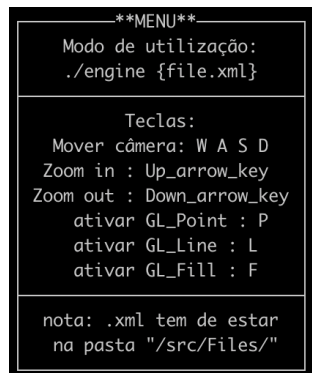


Figure 11: Menu de Ajuda do Engine

### 5.3 Demonstração

O engine deverá ler o ficheiro XML escrito corretamente de forma manual pelo utilizador, o qual a seguir será interpretado pelo mesmo. Após, será apresentado graficamente e será possível interagir com o mesmo de acordo com os comandos apresentados no menu de ajuda.

```
→ cmake-build-debug git:(master) x ./engine bolaDeDiscoteca.xml
```

Figure 12: Exemplo de utilização do Engine

O output esperado do engine será a demonstração gráfica da primitiva lida no ficheiro.

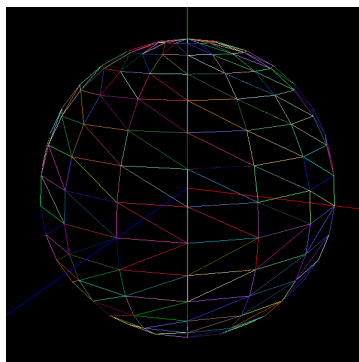


Figure 13: Exemplo de Output

## 6 Extras

Como forma de melhor podermos visualizar certas primitivas gráficas, bem como ir comprovando ao longo da construção desta primeira fase que nada se encontrava em falta ou a falhar, acabámos por implementar algumas funcionalidades extra com base no input do teclado do utilizador.

### 6.1 Câmara

Com o intuito de podermos observar as diferentes faces de várias primitivas gráficas a ser implementadas para nos certificarmos que cumpriram com a regra da mão direita, acabámos por implementar movimentação na câmara que nos possibilitasse visualizar de cima e de baixo cada primitiva e circular à volta da mesma.

Isto é feito com base no seguinte input do teclado do utilizador:

- 'w' -> Permite mover para cima
- 's' -> Permite mover para baixo
- 'a' -> Permite mover para a esquerda
- 'd' -> Permite mover para a direita

Permitimos também que o utilizador aproxime ou afaste a câmara da primitiva representada:

- `ARROW_KEY_UP` -> Aproxima a câmara da primitiva
- `ARROW_KEY_DOWN` -> Afasta a câmara da primitiva

### 6.2 Representação da Primitiva

Além das funcionalidades da câmara, oferecemos também a opção de alterar o modo de visualização da primitiva entre ser representada só por pontos, só por linhas ou completamente preenchida:

- 'p' -> Representação por pontos
- 'l' -> Representação por linhas
- 'f' -> Representação por preenchimento

## 7 Apresentação dos Modelos

### 7.1 Plano

Representação de um plano de dimensão 2.

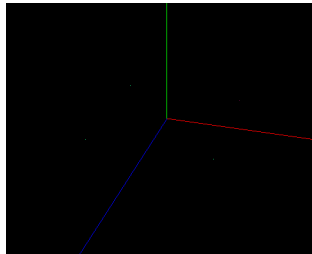


Figure 14: Representação dos Pontos de um Plano

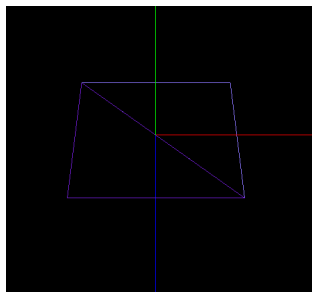


Figure 15: Representação das Linhas de um Plano

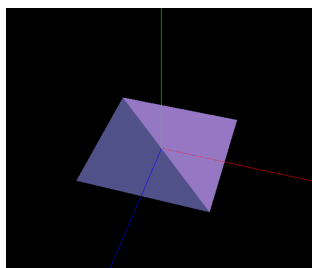


Figure 16: Representação de um Plano

## 7.2 Caixa

Representação de uma caixa de dimensões  $X=4$ ,  $Y=4$ ,  $Z=4$  e duas divisões.

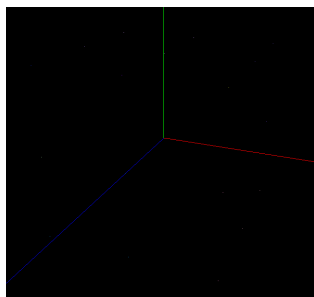


Figure 17: Representação dos Pontos de uma Caixa

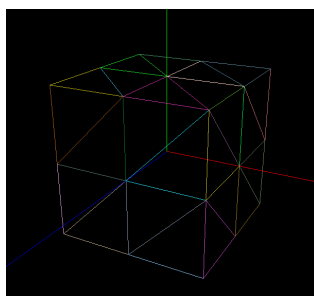


Figure 18: Representação das Linhas de uma Caixa

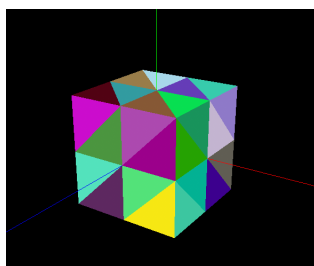


Figure 19: Representação de uma Caixa

### 7.3 Esfera

Representação de uma esfera de raio 2, 16 slices e 16 stacks.

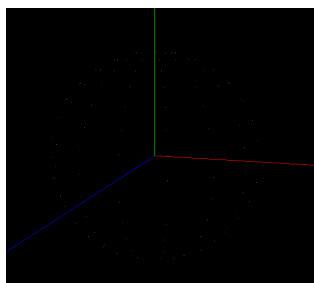


Figure 20: Representação dos Pontos de uma Esfera

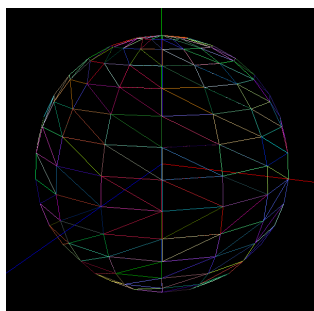


Figure 21: Representação das Linhas de uma Esfera

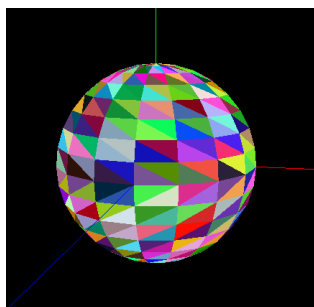


Figure 22: Representação de uma Esfera



## 7.4 Cone

Representação de um cona com raio da base 2, 3 de altura, 16 slices e 16 stacks.

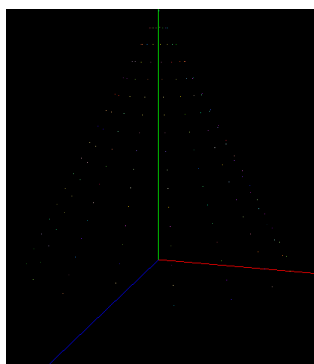


Figure 23: Representação dos Pontos de um Cone

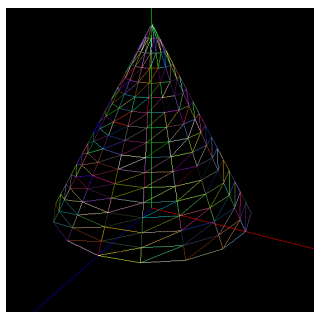


Figure 24: Representação das Linhas de um Cone

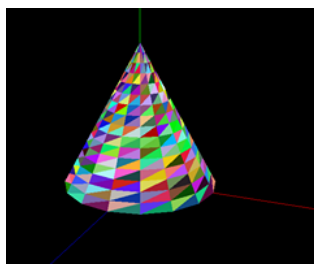


Figure 25: Representação de um Cone

## 8 Conclusão e Trabalho Futuro

A realização da primeira fase deste projeto permitiu-nos ambientar ainda melhor às ferramentas referentes à computação gráfica com as quais iremos trabalhar ao longo do mesmo, nomeadamente OpenGL e GLUT. Simultaneamente, permitiu-nos adquirir conhecimentos sobre uma nova linguagem, C++, útil e indispensável à realização do trabalho, linguagem essa que não tinha sido ainda abordada no curso.

De forma geral achamos que os requisitos da primeira fase foram cumpridos na íntegra e que o trabalho apresentado cumpre com as expectativas que o grupo tinha em relação à sua performance e conhecimento adquirido até ao momento na UC.

De forma resumida, o grupo acredita que esta primeira fase proporciona uma melhor ambientação às ferramentas que iremos utilizar durante o resto do projeto e que esta servirá como base para tudo aquilo que iremos construir a partir daqui.