

UNIVERSIDADE DO MINHO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

Computação Gráfica

Trabalho Prático - Fase I

GRUPO 21



Ana Pereira
A81712



Francisco Freitas
A81580



Maria Dias
A81611



Pedro Freitas
A80975

March 9, 2019

Índice

1	Introdução	2
2	Arquitetura de Código	3
2.1	Aplicações	3
2.1.1	Generator	3
2.1.2	Engine	3
2.2	Classes	3
2.2.1	Point.cpp	3
2.2.2	Shape.cpp	3
2.3	Primitivas	4
3	Primitivas Geométricas	5
3.1	Plano	5
3.1.1	Algoritmo de Criação	5
3.1.2	Modelo 3D	6
3.2	Cone	6
3.2.1	Algoritmo de Criação	6
3.2.2	Modelo 3D	8
3.3	Esfera	8
3.3.1	Algoritmo de Criação	8
3.3.2	Modelo 3D	9
3.4	Paralelepípedo	10
3.4.1	Algoritmo de Criação	10
3.4.2	Modelo 3D	12
3.5	Cilindro	12
3.5.1	Algoritmo de Criação	13
3.5.2	Modelo 3D	13
3.6	Torus	14
3.6.1	Algoritmo de Criação	14
3.6.2	Modelo 3D	16
4	Generator	17
4.1	Descrição	17
4.2	Usabilidade	17
4.3	Demonstração	17
5	Engine	19
5.1	Descrição	19
5.2	Usabilidade	19
5.3	Demonstração	19
6	Conclusão	22

1 Introdução

O trabalho que expomos neste relatório surge no âmbito da unidade curricular de Computação Gráfica, tendo como principal objetivo o desenvolvimento de um pequeno mecanismo 3D baseado num cenário gráfico. O projeto encontra-se dividido em quatro fases, sendo que neste relatório é exposto o trabalho realizado durante a primeira fase. Nesta, foram desenvolvidas algumas primitivas básicas: plano, cone, esfera, paralelepípedo, cilindro e torus, sendo que as duas últimas foram realizadas como forma de aproveitar e explorar as capacidades adquiridas no desenvolvimento das restantes.

2 Arquitetura de Código

Nesta secção é apresentada a estrutura do projeto, começando pelas aplicações implementadas, seguidas da exposição das classes que foram criadas para conseguir essas implementações e terminando com a abordagem das várias formas geométricas que foram conseguidas.

2.1 Aplicações

Esta fase do trabalho prático requer a criação de duas aplicações. A primeira aplicação gera ficheiros com a informação dos modelos a desenhar. Esta informação consiste nos vértices que compõem a figura. A segunda lê um ficheiro de configuração, em formato XML, e exhibe os modelos.

2.1.1 Generator

O *generator* é responsável por criar ficheiros com os pontos de uma determinada figura, para isso armazena os pontos da figura num vector e de seguida escreve-os num ficheiro.

2.1.2 Engine

O *Engine* é responsável por interpretar os ficheiros criados pela aplicação *generator* e apresentar os modelos pretendidos, sendo possível interagir com estes através de comandos que modificam a vista que temos dos objetos.

2.2 Classes

Para trazer à vida o projeto em questão, concordamos na criação de duas classes, *Point* e *Shape*, que são, respetivamente, a representação de cada ponto e de cada forma geométrica criada (conjunto de pontos). Esta decisão foi motivada com base no facto de estas duas classes fornecerem estrutura e organização ao programa.

2.2.1 Point.cpp

Classe que guarda as coordenadas x, y e z de um ponto, necessário para o desenho de cada vértice de um triângulo, uma vez que a base de todas as figuras desenhadas é o triângulo.

2.2.2 Shape.cpp

Classe que armazena num vector o conjunto de todos os pontos pertencentes a um determinado modelo.

2.3 Primitivas

Para modelar as formas geométricas propostas, foi necessário elaborar algoritmos que nos permitissem calcular as coordenadas de todos os vértices que constituem os diferentes modelos. Nesse sentido, foi criado um ficheiro .cpp para cada figura, contendo o seu algoritmo de criação dos vértices. Estes algoritmos passam a ser explicados no próximo capítulo.

3 Primitivas Geométricas

3.1 Plano

O plano está contido no plano XZ e é centrado na origem do referencial.

3.1.1 Algoritmo de Criação

O plano é composto por dois triângulos, que partilham dois pontos. Para fazer com que este plano fique voltado para cima, a 3 dimensões, é importante ter atenção à ordem da criação dos pontos. Temos de ter em atenção que a regra da mão direita foi aplicada ao plano a três dimensões e não a duas. Sabendo isto, para fazermos o primeiro triângulo, e aplicando a regra da mão direita, temos que a ordem de criação dos pontos é: BCO . Para o segundo triângulo a ordem dos pontos será: OAB . Tendo em conta que o plano é centrado na origem e apenas lhe é fornecido o tamanho do lado, temos fazer os cálculos para determinar a posição de cada ponto. Como este está definido no plano XZ concluímos facilmente que todos os pontos vão ter o valor 0 em relação ao eixo dos Y. Para estar centrado na origem, os pontos não poderão ter o valor do tamanho (*size*) mas sim apenas metade ($m = size/2$):

$$A = (-m, 0, m)$$

$$B = (m, 0, m)$$

$$C = (m, 0, -m)$$

$$O = (-m, 0, -m)$$

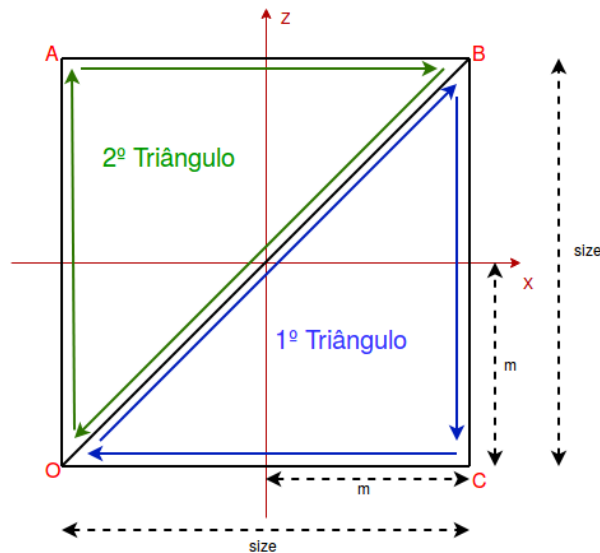


Figura 1: Plano a 2D

3.1.2 Modelo 3D

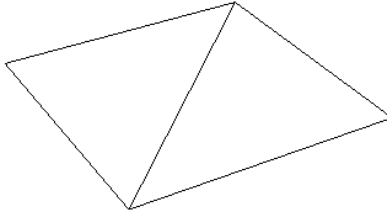


Figura 2: Modelo 3D de um plano

3.2 Cone

O cone é definido pelo seu raio, altura, número de divisões verticais e número de divisões horizontais. Esta figura é obtida a partir de uma pirâmide, em que a base é um polígono regular e o número de fatias da mesma tende para infinito. Portanto, quanto maior for o número de divisões verticais e horizontais, mais exata se torna a curvatura do cone.

3.2.1 Algoritmo de Criação

Para a criação do cone, consideram-se três processos: a criação da base, criação da superfície lateral do cone, e a criação do "bico" do cone (camada do topo).

Para o desenho da base, considerando que o cone está centrado na origem do referencial, obtém-se o valor do ângulo α , correspondente à amplitude para uma fatia (slice), fazendo a divisão de 360° pelo número de fatias que se pretende. Então, para o desenho de uma fatia da base, pegando, por exemplo, nos pontos C e D da figura 3, teríamos algo assim:

$$Dx = raio * \cos(\alpha + amplitude)$$

$$Dy = 0$$

$$Dz = raio * \sin(\alpha + amplitude)$$

$$x = 0$$

$$y = 0$$

$$z = 0$$

$$Cx = raio * \cos \alpha$$

$$Cy = 0$$

$$Cz = raio * \sin \alpha$$

Consideremos agora a construção da superfície lateral do cone, processo ilustrado na figura 3. Segundo a regra da mão direita, temos que a orientação dos vértices para os triângulos CDB e CBA são, respetivamente, $D \rightarrow B \rightarrow C$ e $B \rightarrow A \rightarrow C$. Como forma de exemplo, o ponto B tem as coordenadas:

$$r_i n \times \cos(\alpha + amplitude), topo, r_i n \times \sin(\alpha + amplitude))$$

e o ponto C tem as coordenadas

$$r * \cos \alpha, base, r * \sin \alpha)$$

As variáveis *base* e *topo* representam a altura inferior e altura superior de cada camada (*stack*) do cone, enquanto que as variáveis *r* e *r_in* representam o raio da circunferência do cone na base e no topo, respetivamente. Ou seja, no caso da primeira camada, que inclui os pontos A, B, C e D, o valor da base é $i \times height/stacks$, com $i = 0$, o valor do topo é $i \times height/stacks$, com $i = 1$, o valor de *r* é igual a *raio* — $(i \times raio/stacks)$, com $i = 0$, e o valor de *r_in* é *raio* — $(i \times raio/stacks)$, com $i = 1$, e com *raio* a representar o raio da base do cilindro. Naturalmente, o valor de *i* incrementa consoante subimos pelas camadas do cone.

Por fim, analisemos a construção do topo do cone. Basicamente, ligam-se os vértices ao centro para formar o bico do cone. Ou seja, o vértice central é sempre $(0, height, 0)$, e os exteriores apenas variam entre si em α .

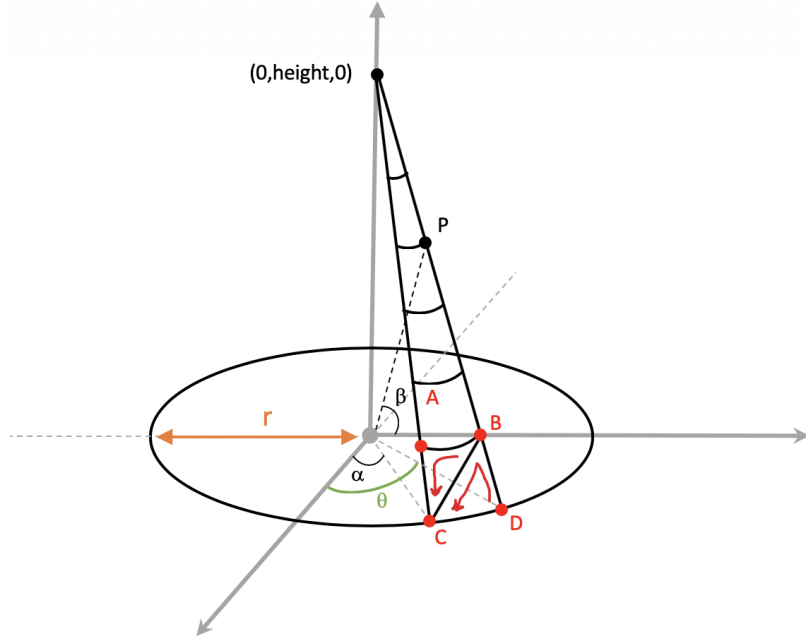


Figura 3: Esquema de decomposição de um cone em fatias e stacks

3.2.2 Modelo 3D

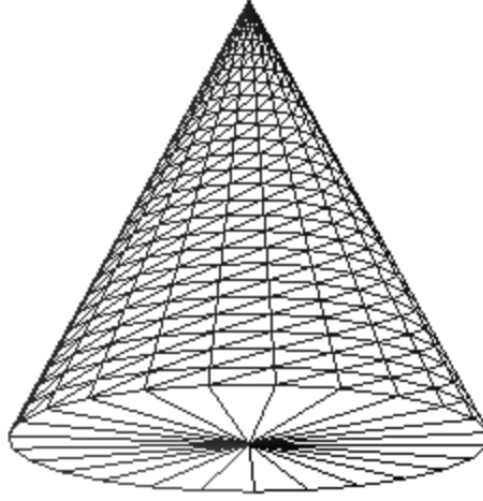


Figura 4: Modelo 3D de um cone com 25 divisões verticais e horizontais

3.3 Esfera

A esfera é definida pelo seu raio, número de divisões verticais e número de divisões horizontais.

3.3.1 Algoritmo de Criação

Na criação de uma esfera, são introduzidas as coordenadas esféricas, ou seja, para além de termos um ângulo α que cria uma circunferência no plano XZ, temos agora um ângulo β que representa o ângulo entre qualquer ponto da esfera em relação ao eixo XZ (este valor varia entre -90° e 90°). Segundo a regra da mão direita, temos que a orientação dos vértices para os triângulos CDB e CBA são, respetivamente, $D \rightarrow B \rightarrow C$ e $B \rightarrow A \rightarrow C$. Para definir as coordenadas dos pontos A, B e C usamos as expressões que se seguem. De notar que θ e φ representam a variação dos ângulos α e β à medida que se prossegue de fatia em fatia e de stack (nível horizontal) em stack, respetivamente. Considerando a figura 5,

$$\begin{cases} x = r \times \cos \beta \times \cos \alpha \\ y = r \times \sin \beta \\ z = r \times \cos \beta \times \sin \alpha \end{cases}$$

$$\theta = \alpha + \frac{2\pi}{slices}$$

$$\varphi = \beta + \frac{\pi}{stacks}$$

$$\begin{cases} x2 = r \times \cos \varphi \times \cos \theta \\ y2 = r \times \sin \varphi \\ z2 = r \times \cos \varphi \times \sin \theta \end{cases}$$

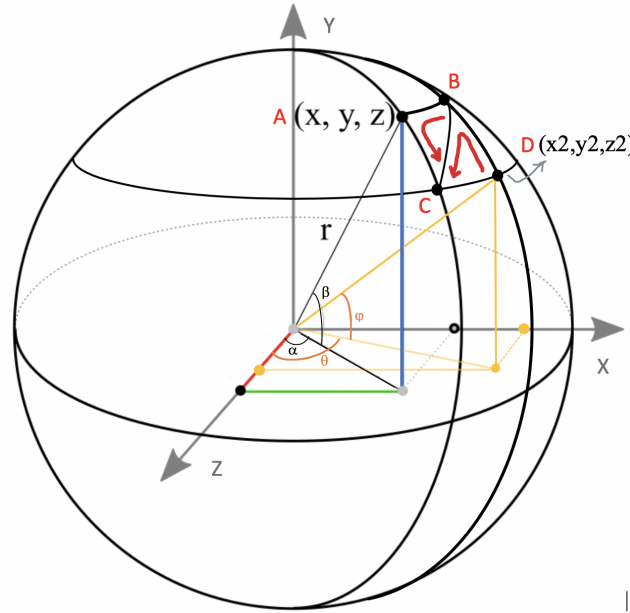


Figura 5: Esquema de criação de uma esfera

3.3.2 Modelo 3D

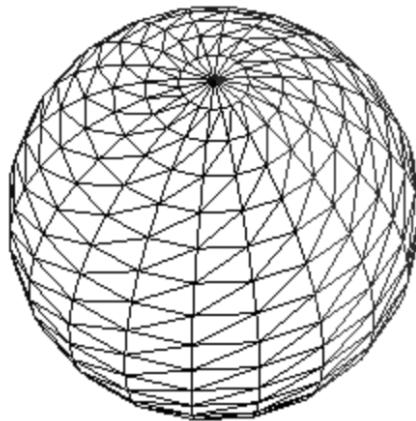


Figura 6: Modelo 3D de uma esfera com 20 divisões verticais e horizontais

3.4 Paralelepípedo

O paralelepípedo é definido pela sua largura, altura, comprimento e número de divisões em cada face.

3.4.1 Algoritmo de Criação

Para cada face do paralelepípedo é efetuado o mesmo raciocínio do plano. Sem divisões cada uma das faces do sólido será constituída por 2 triângulos, que partilham dois vértices entre si. No entanto, introduzir divisões no paralelepípedo implica ser necessário mais triângulos para desenhar uma face. Sendo div o número de divisões, temos 2^{div+1} triângulos por face.

Assim, é necessário calcular o desvio entre os vértices. Os desvios relativos aos eixos x, y e z, estão representados na figura 7 através das variáveis l, h e w, respetivamente.

$$\begin{cases} l = \frac{length}{div} \\ h = \frac{height}{div} \\ w = \frac{width}{div} \end{cases}$$

Considerando a face frontal do paralelepípedo, sabemos que o valor de z é constante para todos os vértices, tomando o valor de $\frac{width}{2}$. Começamos então por desenhar os triângulos da esquerda para a direita, incrementando os valores das coordenadas x e y dos pontos para obter os restantes triângulos. Obedecendo à regra da mão direita, a orientação dos vértices para os triângulos é $B \rightarrow C \rightarrow D$ e $A \rightarrow B \rightarrow D$.

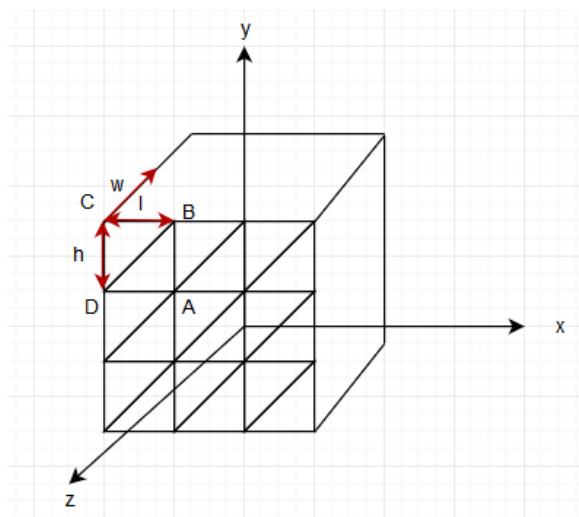


Figura 7: Esquema de construção de um paralelepípedo

Para i e $j \in [0, div[$:

Primeiro triângulo:

$$\left\{ \begin{array}{l} x = \frac{-length}{2} + (i+1) \times l \\ y = \frac{height}{2} - j \times h \\ z = \frac{width}{2} \end{array} \right.$$

$$\left\{ \begin{array}{l} x = \frac{-length}{2} + i \times l \\ y = \frac{height}{2} - j \times h \\ z = \frac{width}{2} \end{array} \right.$$

$$\left\{ \begin{array}{l} x = \frac{-length}{2} + i \times l \\ y = \frac{height}{2} - (j+1) \times h \\ z = \frac{width}{2} \end{array} \right.$$

Para $i=j=0$, desenhemos o triângulo BCD da figura.

Segundo triângulo:

$$\left\{ \begin{array}{l} x = \frac{-length}{2} + (i+1) \times l \\ y = \frac{height}{2} - (j+1) \times h \\ z = \frac{width}{2} \end{array} \right.$$

$$\left\{ \begin{array}{l} x = \frac{-length}{2} + (i+1) \times l \\ y = \frac{height}{2} - j \times h \\ z = \frac{width}{2} \end{array} \right.$$

$$\left\{ \begin{array}{l} x = \frac{-length}{2} + i \times l \\ y = \frac{height}{2} - (j+1) \times h \\ z = \frac{width}{2} \end{array} \right.$$

Para $i=j=0$, obtemos o triângulo ABD da figura.

3.4.2 Modelo 3D

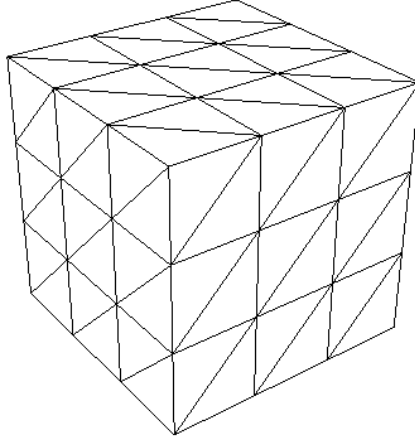


Figura 8: Modelo 3D de um cubo de lado 3

3.5 Cilindro

O cilindro é definido pelo raio da sua base, pela sua altura, número de divisões verticais e número de divisões horizontais.

A construção da superfície lateral do cilindro, representada na figura 9, começa com as seguintes definições, considerando as variáveis i e j , que variam de 0 ao número de divisões verticais (slices) e de 0 ao número de divisões horizontais (stacks), respetivamente:

$$\alpha = i \times 2\pi / \text{slices}$$

$$\text{salto} = 2\pi / \text{stacks}$$

$$\text{base} = j * \text{salto}$$

$$\text{topo} = (j + 1) * \text{salto}$$

Sabendo isto, podemos escrever, para o ponto C:

$$x = r * \cos(\alpha)$$

$$y = \text{base}$$

$$z = r * \sin(\alpha)$$

sendo r o raio da circunferência de base e topo do cilindro. Já para o ponto B teríamos:

$$x = r * \cos(\theta)$$

$$y = \text{topo}$$

$$z = r * \sin(\theta)$$

$$\theta = (i + 1) \times 2\pi / \text{slices}$$

3.5.1 Algoritmo de Criação

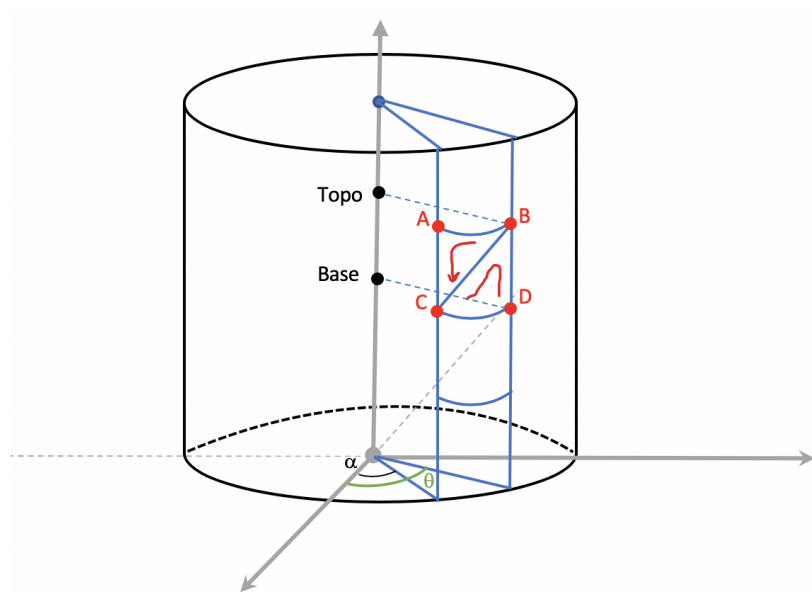


Figura 9: Esquema de uma fatia de um cilindro

3.5.2 Modelo 3D

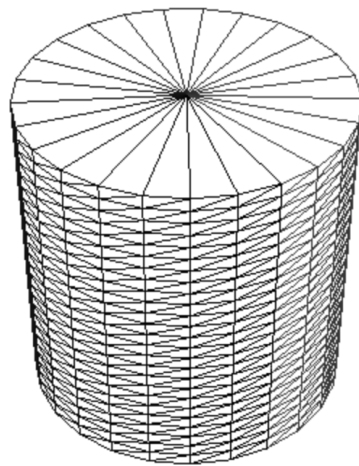


Figura 10: Modelo 3D de um cilindro com 25 divisões verticais e horizontais

3.6 Torus

O torus é uma figura que apresenta o formato aproximado de uma câmara de pneu. Em geometria, pode ser definido como o lugar geométrico tridimensional formado pela rotação de uma superfície circular plana de raio r , em torno de uma circunferência de raio R . Assim sendo, definimos um torus pelo seu raio interior, raio exterior, divisões verticais e divisões horizontais.

3.6.1 Algoritmo de Criação

Os eixos Z e X definem uma circunferência de raio exterior R , e os eixos X , Y e Z definem a circunferência de raio interior r .

Cada divisão vertical é feita com um desvio de $\alpha = i * 2\pi / slices$ e cada divisão horizontal é definida por um ângulo $\beta = j * 2\pi / stacks$, variável responsável por dar a volta à circunferência interna. As variáveis i e j variam entre 0 e o número de divisões verticais e divisões horizontais, respectivamente. As expressões gerais para a construção deste modelo são as seguintes:

$$\begin{cases} x = R + r * \cos(\beta) * \cos(\alpha) \\ y = r * \sin(\beta) \\ z = R + r * \cos(\beta) * \sin(\alpha) \end{cases}$$

em que α e β variam entre 0 e 2π , R é a distância do centro do tubo ao centro do toro, e r é o raio do tubo toroidal. Podemos visualizar essas variações nas figuras 11 e 12, em que estão representadas as variáveis mencionadas até agora.

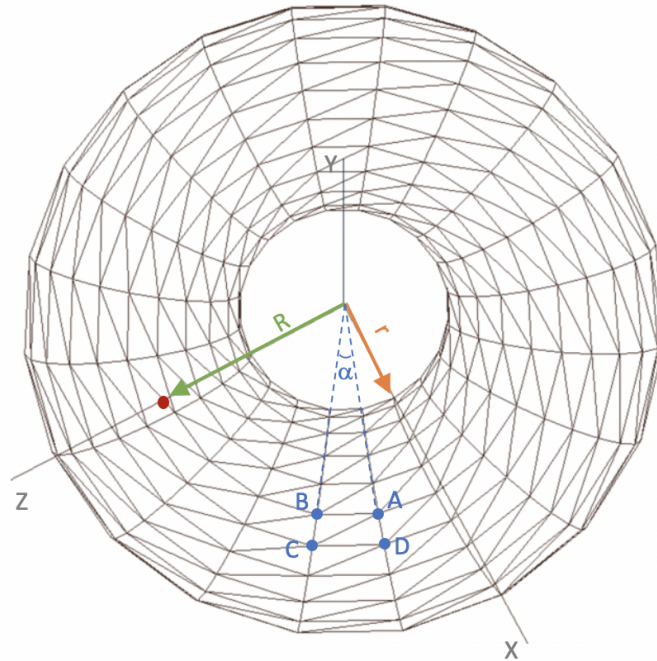


Figura 11: Vista de topo de um torus

A figura 12 representa um corte vertical no torus, possibilitando assim uma vista para o interior do tubo toroidal. Como podemos ver, entre B e C, o valor de α mantém-se, uma vez que os dois pontos estão no mesmo plano de corte vertical. No entanto, há um deslocamento relativamente à circunferência do tubo. Assim, teríamos as seguintes coordenadas, considerando que b_x , b_y e b_z são as coordenadas relativas ao ponto B e que c_x , c_y e c_z são relativas ao ponto C:

$$\begin{cases} b_x = R + r * \cos(\beta) * \cos(\alpha) \\ b_y = r * \sin(\beta) \\ b_z = R + r * \cos(\beta) * \sin(\alpha) \end{cases}$$

$$\begin{cases} c_x = R + r * \cos(\beta + \beta) * \cos(alfa) \\ c_y = r * \sin(\beta + \beta) \\ c_z = R + r * \cos(\beta + \beta) * \sin(alfa) \end{cases}$$

De forma análoga, entre os pontos A e C, por exemplo, alteram-se na expressão os valores de β e α . A construção dos vértices do torus segue a regra da mão direita, como temos visto para todas as outras figuras que já foram referidas.

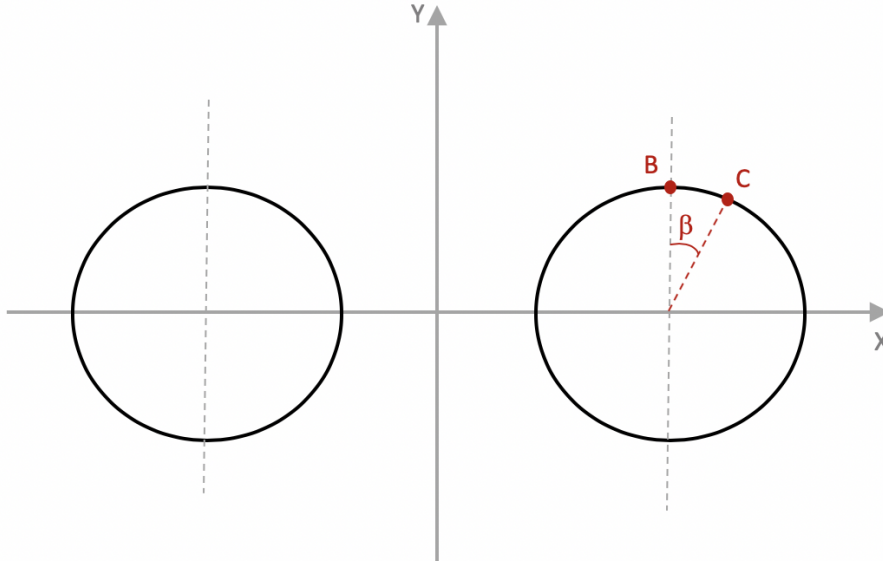


Figura 12: Corte vertical de um torus

3.6.2 Modelo 3D

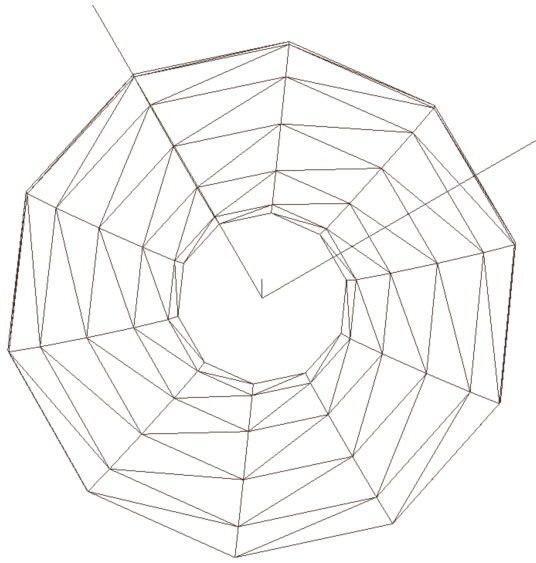


Figura 13: Modelo 3D de um torus com 10 divisões verticais e horizontais

4 Generator

4.1 Descrição

O *generator* recebe como argumento a forma que se pretende visualizar (*plane*, *box*, *sphere*, *cone* ou *cylinder*) e os argumentos necessários para se poder desenhar a forma e ainda o nome do ficheiro onde irão ser guardados os pontos. Desta forma, o *generator* aplica o método correto conforme os argumentos passados e escreve no ficheiro todos os pontos que são gerados para o desenho da figura.

4.2 Usabilidade

De seguida é apresentado o manual de ajuda do *generator*, onde é possível consultar todos os comandos que é possível usar e os parâmetros que devem seguir cada um desses comandos.

```
cyborg@cyborg-X555LJ ~/Desktop/Fase1/build $ ./generator -h
-----HELP-----

GUIDELINE: ./generator <SHAPE> ... <FILE>
              [-h]

SHAPE:
- plane <SIZE>
  Creates a square in the XZ plane with center in origin.

- box <HEIGHT> <WIDTH> <LENGTH> <DIVISIONS>
  Creates a box with these dimensions and these divisions.

- sphere <RADIUS> <SLICE> <STACK>
  Creates a sphere with this radius, this number of slices and this
  number of stacks.

- cone <RADIUS> <HEIGHT> <SLICE> <STACK>
  Creates a cone with this base radius, this height, this number of
  of slices and this number of stacks.

- cylinder <RADIUS> <HEIGHT> <SLICE> <STACK>
  Creates a cylinder with this base radius, this height, this number
  of slices and this number of stacks.

- torus <RADIUS> <RADIUS> <SLICE> <STACK>
  Creates a torus with this minor radius, this major radius, this
  number of slices and this number of stacks

-----END-----
```

Figura 14: Manual do Generator

4.3 Demonstração

O funcionamento do *generator* segue o manual de ajuda. Primeiramente deve-se indicar qual a figura e as respetivas dimensões a gerar, conforme os diferentes inputs que o programa aceita, em conjunto com o nome do ficheiro que irá ser gerado.

```
cyborg@cyborg-X555LJ ~/Desktop/Fase1/build $ ./generator plane 4 plane.3d
```

Figura 15: Exemplo de input do *generator*

Portanto, o *generator* irá criar uma diretoria, caso esta não exista, com o nome *files3d*, onde serão guardados todos os ficheiros gerados, esses podem ter qualquer formato. Os ficheiros apresentam todos a mesma estrutura:

$coordenada_x$ $coordenada_y$ $coordenada_z$

Em cada linha do ficheiro estão três números separados por um espaço, representando um ponto que corresponde a um vértice de um triângulo. O ficheiro resultante será algo idêntico à próxima figura.

```
cyborg@cyborg-X555LJ ~/Desktop/Fase1/build $ cat ../files3d/plane.3d
2.000000 0.000000 2.000000
2.000000 0.000000 -2.000000
-2.000000 0.000000 -2.000000
-2.000000 0.000000 2.000000
-2.000000 0.000000 2.000000
2.000000 0.000000 2.000000
```

Figura 16: Exemplo de output do *generator*

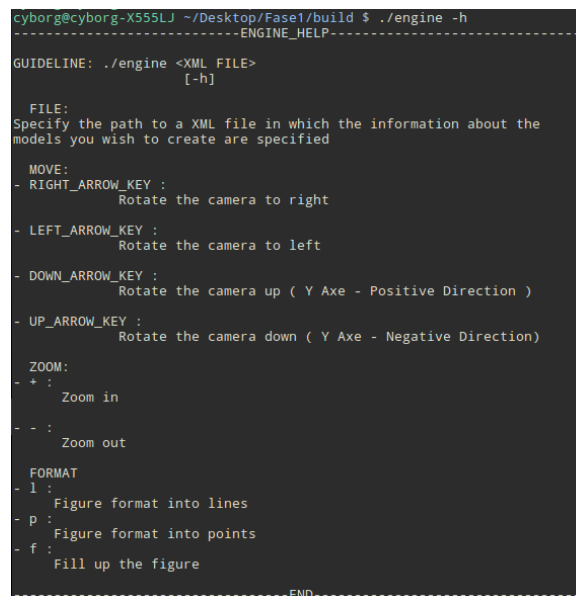
5 Engine

5.1 Descrição

O *engine* recebe ficheiros de configuração escritos em XML. Dentro destes ficheiros encontra-se informação relativa a ficheiros gerados pelo *generator*, nomeadamente os seus nomes. Desta forma, depois de o *engine* fazer *parsing* dos ficheiros, interpreta e apresenta graficamente os modelos presentes em cada ficheiro.

5.2 Usabilidade

Para que o utilizador não tenha dificuldades para usufruir do *engine*, foi criado um manual com todos os comandos que é possível usar, bem como os parâmetros que devem seguir cada um desses comandos.



```
cyborg@cyborg-X555LJ ~/Desktop/Fase1/build $ ./engine -h
-----ENGINE_HELP-----

GUIDELINE: ./engine <XML FILE>
           [-h]

FILE:
Specify the path to a XML file in which the information about the
models you wish to create are specified

MOVE:
- RIGHT_ARROW_KEY :
    Rotate the camera to right
- LEFT_ARROW_KEY :
    Rotate the camera to left
- DOWN_ARROW_KEY :
    Rotate the camera up ( Y Axe - Positive Direction )
- UP_ARROW_KEY :
    Rotate the camera down ( Y Axe - Negative Direction)

ZOOM:
- + :
    Zoom in
- - :
    Zoom out

FORMAT
- l :
    Figure format into lines
- p :
    Figure format into points
- f :
    Fill up the figure

-----END-----
```

Figura 17: Manual de ajuda do Engine

5.3 Demonstração

Os ficheiros de configuração escritos em XML que o *engine* interpreta são criados manualmente pelo utilizador e os ficheiros modelo presentes nestes são previamente gerados pelo *engine*. Depois da interpretação dos ficheiros e a apresentação dos modelos, pode-se interagir com estes consoante os comandos definidos no menu ajuda.

Exemplo de funcionamento do *engine*:

```
cyborg@cyborg-X555LJ ~/Desktop/Fase1/build $ cat ../files3d/exemplo.xml
<scene>
  <model file="../files3d/cone.3d" />
  <model file="../files3d/plane.3d" />
</scene>
```

Figura 18: Exemplo de um ficheiro de configuração em XML.

Executar o *engine*, passando o ficheiro anterior como input.

```
cyborg@cyborg-X555LJ ~/Desktop/Fase1/build $ ./engine ../files3d/exemplo.xml
```

Figura 19: Exemplo de input do *engine*

O output deverá ser a representação gráfica dos modelos pretendidos. Neste caso seria algo do género :

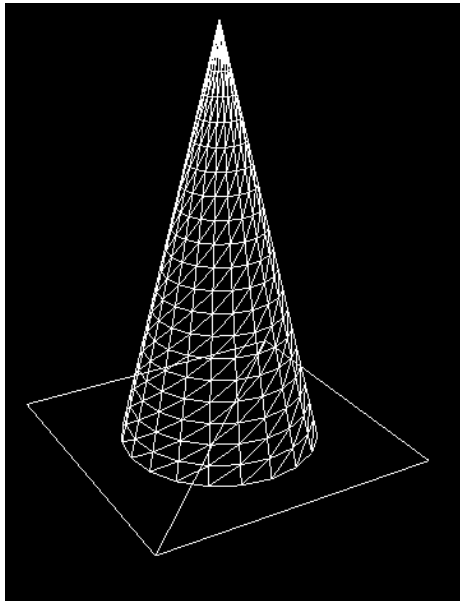
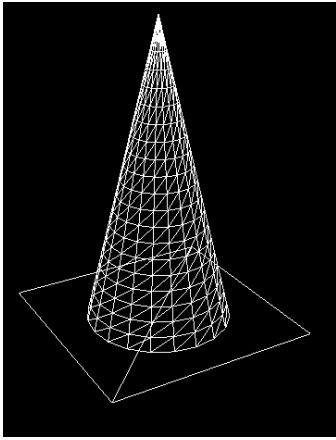
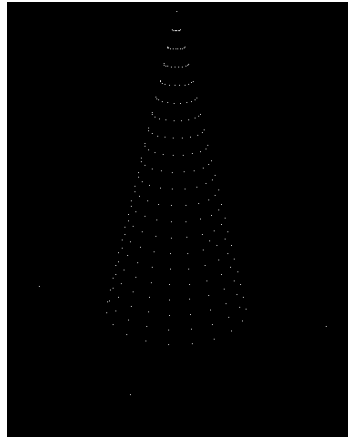


Figura 20: Exemplo do output do *engine*

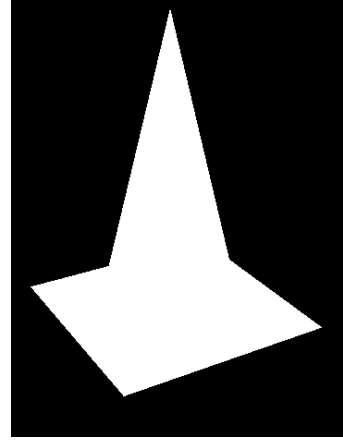
Depois da sua representação, também é possível ver os modelos de diferentes perspectivas e diferentes modos.



Comando L



Comando P



Comando F

6 Conclusão

A elaboração desta primeira fase do projeto consolidou os conhecimentos abordados na Unidade Curricular até à data presente. Deste trabalho, resultou também a familiarização com as ferramentas utilizadas para modelação 3D e com a linguagem C++.

Assim sendo, consideramos que o trabalho desenvolvido nesta fase representa, de certa forma, o pilar no qual o resto do projeto assentará, pilar esse que tomamos muito gosto em realizar e cuja construção julgamos bem sucedida.