



Universidade do Minho
Escola de Engenharia

Computação Gráfica

Fase 1 - Primitivas Gráficas

Relatório Trabalho Prático

Grupo 40

LEI - 3º Ano - 2º Semestre

A90969	Gonçalo Gonçalves
A98695	Lucas Oliveira
A89292	Mike Pinto
A96208	Rafael Gomes

Braga,
19 de fevereiro de 2025

Conteúdo

1	Introdução	3
2	Generator	4
2.1	Plano	4
2.2	Caixa	7
2.2.1	Planos Paralelos a YZ	7
2.2.2	Planos Paralelos a XY	9
2.2.3	Planos Paralelos a XZ	10
2.3	Esfera	13
2.4	Cone	17
2.5	Modo de Uso	19
2.5.1	Criação do executável	19
3	Engine	21
3.1	Modo de Uso	22
3.1.1	Execução da Engine	22
3.1.2	Comandos	22
4	Conclusão	23

Lista de Figuras

2.1	Pontos escritos num ficheiro .3D	4
2.2	Representação de um quadrilátero da subdivisão do plano.	5
2.3	Ordem de desenho dos quadriláteros de um plano com três subdivisões.	6
2.4	Exemplos de geração de diferentes Planos	6
2.5	Aplicação sobre o Plano rotação de -90^0 sobre o eixo do Z e translação sobre o eixo positivo do X.	8
2.6	Aplicação sobre o Plano rotação de 90^0 sobre o eixo do Z e translação sobre o eixo negativo do X.	8
2.7	Aplicação sobre o Plano rotação de 90^0 sobre o eixo do X e translação sobre o eixo positivo do Z.	9
2.8	Aplicação sobre o Plano rotação de -90^0 sobre o eixo do X e translação sobre o eixo negativo do Z.	10
2.9	Aplicação sobre o Plano translação sobre o eixo positivo do Y.	11
2.10	Aplicação sobre o Plano rotação de 180^0 sobre o eixo do X e translação sobre o eixo negativo do Y.	11
2.11	Exemplos de geração de diferentes Caixas	12
2.12	Exemplo da variação de α e β em uma esfera.	13
2.13	Distinção entre camadas de uma esfera.	14
2.14	Exemplo da ordem de criação das camadas de uma fatia de uma esfera com nove camadas.	15
2.15	Exemplo da ordem de calculo das coordenadas de uma camada de uma esfera.	15
2.16	Exemplos de geração de diferentes Esferas	16
2.17	Exemplo do calculo dos vértices da de uma fatia da base de um cone com seis <i>slices</i>	17
2.18	Exemplos da variância do raio e altura de um cone por cada camada.	18
2.19	Exemplos de geração de diferentes Cones	19

Capítulo 1

Introdução

Este relatório é elaborado no âmbito da primeira fase do trabalho prático da Unidade Curricular de Computação Gráfica do 2º semestre do 3º ano do curso de Engenharia Informática da Universidade do Minho.

Nesta fase inicial, foi proposto o desenvolvimento de duas aplicações: o **Generator**, que tem como objetivo principal servir como gerador de vértices para primitivas gráficas como **plano**, **caixa**, **esfera** e **cone**, além de armazenar a informação dos seus vértices em arquivos com a extensão *.3d*; e a aplicação **Engine**, que visa ler arquivos de configuração no formato *.xml* e desenhar as primitivas gráficas criadas a partir do **Generator**.

Ambas as aplicações foram implementadas utilizando a linguagem de programação C++ e a API OpenGL.

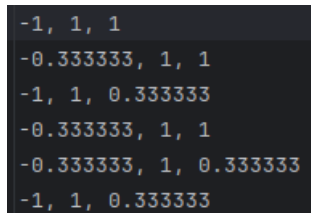
Este relatório detalha todas as etapas do desenvolvimento das aplicações, incluindo as decisões tomadas e o raciocínio por trás delas. Serão apresentados os desafios enfrentados durante o processo de implementação, bem como as soluções adotadas para superá-los.

Ao longo deste documento, serão explorados os métodos utilizados, as funcionalidades implementadas e os resultados obtidos, proporcionando uma visão abrangente do trabalho realizado nesta fase do projeto.

Capítulo 2

Generator

O *Generator* é a aplicação responsável por calcular todos os pontos dos triângulos que irão constituir as várias primitivas e será também responsável por escrever esses pontos num ficheiro .3D. À medida que os pontos vão sendo calculados, são adicionados a um *vector*. Optou-se por esta abordagem de forma a ter um programa o mais eficiente possível, já que a escrita frequente pode levar a *overhead* desnecessário, enquanto que o espaço que ocupa o seu armazenamento acaba por ser desprezável. A estrutura do ficheiro consiste em uma linha por cada ponto que é constituído por três coordenadas, sendo que cada coordenada é definida por um *float* separado por um espaço, representando as coordenadas X, Y e Z, respetivamente. Cada três pontos seguidos vai constituir um triângulo.



```
-1, 1, 1
-0.333333, 1, 1
-1, 1, 0.333333
-0.333333, 1, 1
-0.333333, 1, 0.333333
-1, 1, 0.333333
```

Figura 2.1: Pontos escritos num ficheiro .3D

Para o cálculo dos pontos para as diferentes primitivas é necessário fornecer ao *Generator* diferentes argumentos, tais como:

- Plano : *length, divisions*.
- Caixa : *length, divisions*.
- Esfera : *radius, slices, stacks*.
- Cone : *radius, height, slices, stacks*.

2.1 Plano

O Plano é uma primitiva gráfica centrada na origem do referencial (Ponto (0,0,0)) e contida no plano XZ. Ele é subdividido ao longo das direções X e

Z, formando quadriláteros de comprimento e largura $length/divisions$ em cada subdivisão. Cada quadrilátero é composto por dois triângulos (Figura 2.2). Para calcular os pontos que compõem o plano, concebe-se esta primitiva na forma de uma *Matriz* de dimensão $divisions \times divisions$, onde cada elemento representa um quadrilátero formado pelas subdivisões do plano.

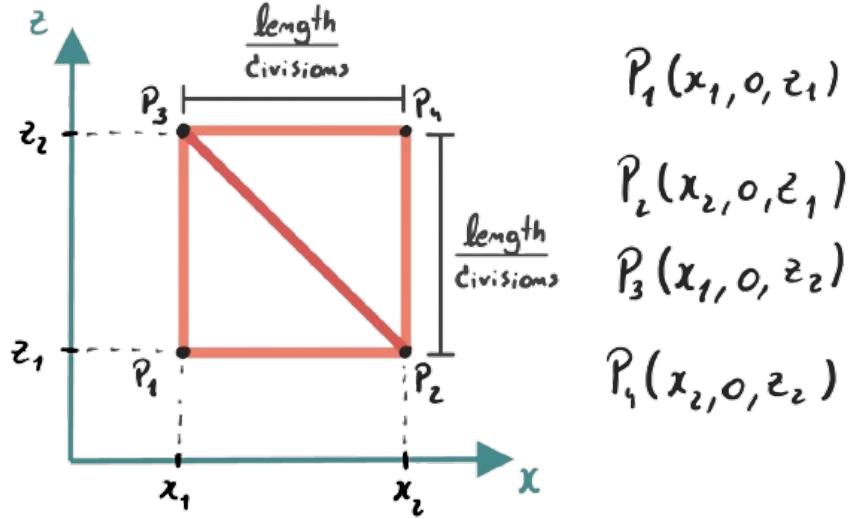


Figura 2.2: Representação de um quadrilátero da subdivisão do plano.

Para cada quadrilátero, são calculadas apenas duas coordenadas **X** e duas coordenadas **Z**, definidas como:

$$\begin{aligned} x_1 &= -length + (i \times square_size) \\ x_2 &= -length + ((i + 1) \times square_size) \\ z_1 &= -length + (j \times square_size) \\ z_2 &= -length + ((j + 1) \times square_size) \end{aligned}$$

onde, **length** é o comprimento do plano,

square_size = $length/divisions$,

i representa a linha da matriz,

e **j** representa a coluna da matriz, com $i, j \in [0, divisions]$.

Após o cálculo das coordenadas, os pontos P1, P2, P3 e P4 são criados para formar os triângulos que compõem o quadrilátero da subdivisão, conforme representado na Figura 2.2. O processo é repetido para todos os quadriláteros, conforme esquematizado na Figura 2.3, representando a ordem de criação dos quadriláteros de um plano com três subdivisões.

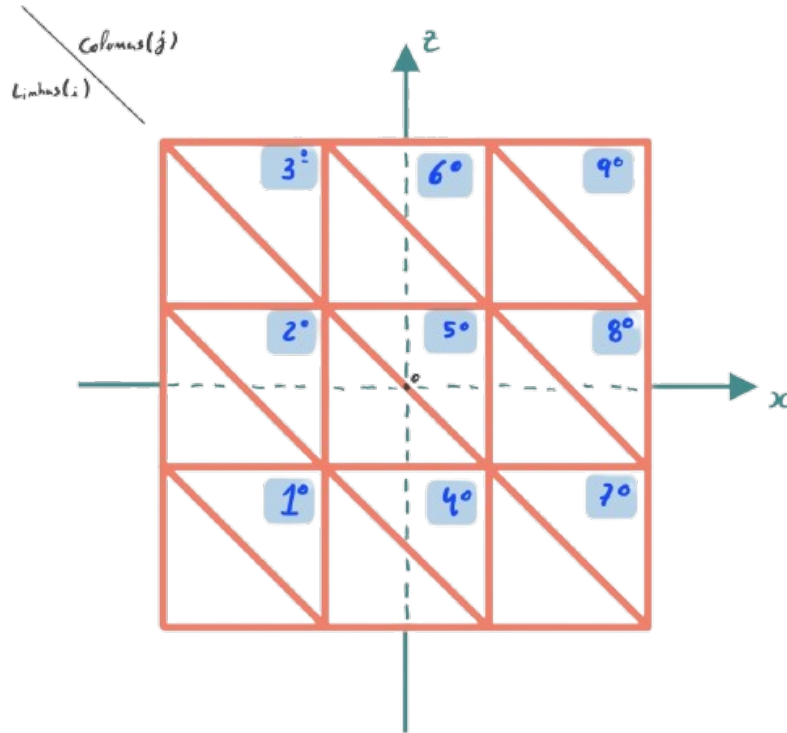
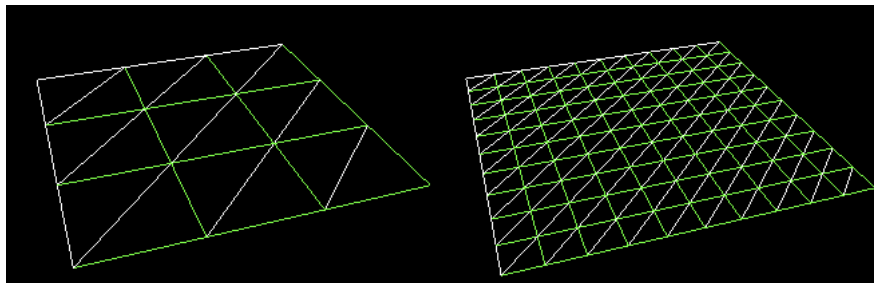


Figura 2.3: Ordem de desenho dos quadriláteros de um plano com três subdivisões.

No final do cálculo dos pontos de uma coluna, os valores de $z1$ e $z2$ retornam aos seus valores iniciais, enquanto os valores de $x1$ e $x2$ são incrementados em `square_size`, iniciando o cálculo de pontos dos quadriláteros de uma nova coluna.

Na Figura 2.4, apresenta-se o resultado de dois desenhos de um plano, um com comprimento 1 e 3 subdivisões e outro com comprimento 2 e 10 subdivisões.



(a) Plano com $length = 1$ e $divisions = 3$ (b) Plano com $length = 2$ e $divisions = 10$

Figura 2.4: Exemplos de geração de diferentes Planos

2.2 Caixa

Para o cálculo dos pontos que formam a Caixa utiliza-se o mesmo raciocínio que se aplica na construção do plano, mas construindo as seis faces da caixa simultaneamente. Sabe-se que a distância entre dois planos paralelos é de $length$, logo desloca-se cada plano em $length/2$, isto é, aplica-se uma translação.

Contudo, antes de efetuar esta translação, é necessário efetuar três rotações diferentes, de forma a obter planos paralelos a XZ, XY e YZ. Para isto, recorre-se ao cálculo matricial, respeitando a regra da mão direita, e efetuam-se rotações sobre o eixo do X e o eixo do Z, de qualquer ponto contido no plano XZ que foi onde foi construído o Plano do tópico anterior.

Tendo em conta o seguinte:

$$R_{x,a} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(a) & -\sin(a) & 0 \\ 0 & \sin(a) & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.1)$$

$$R_{y,a} = \begin{bmatrix} \cos(a) & 0 & \sin(a) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(a) & 0 & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.2)$$

$$R_{z,a} = \begin{bmatrix} \cos(a) & -\sin(a) & 0 & 0 \\ \sin(a) & \cos(a) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.3)$$

$$\cos(-a) = \cos(a) \quad (2.4)$$

$$\sin(-a) = -\sin(b) \quad (2.5)$$

Então, para um qualquer ponto do plano XZ em que $y=0$, $\begin{bmatrix} x \\ 0 \\ z \\ 1 \end{bmatrix}$, iremos aplicar as devidas rotações.

2.2.1 Planos Paralelos a YZ

Para o cálculo dos planos paralelos ao plano YZ efetuamos uma rotação sobre o eixo Z de 90° para a face lateral cujo x é positivo, e outra de -90° para a face cujo x é negativo, respeitando assim a regra da mão direita. Substituindo a , efetuando as alterações e aplicando o cálculo da rotação obtem-se:

$$R_{x,90} : \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ 0 \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ x \\ z \\ 1 \end{bmatrix} \quad (2.6)$$

$$R_{x,-90} : \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ 0 \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -x \\ z \\ 1 \end{bmatrix} \quad (2.7)$$

A partir das equações 2.6 e 2.7 podemos constatar que qualquer ponto do plano XZ cujo $y=0$ é conversível para um ponto do plano YZ cujo $x=0$ substituindo o x pelo y . Dependendo da rotação o x pode assumir valores positivos ou negativos. De seguida, fazendo uma translação de $\text{length}/2$ sobre eixo do X no plano que sofreu a rotação de -90° e fazendo uma translação de $-\text{length}/2$ no plano que sofreu a rotação de 90° , são obtidas duas das faces da caixa.

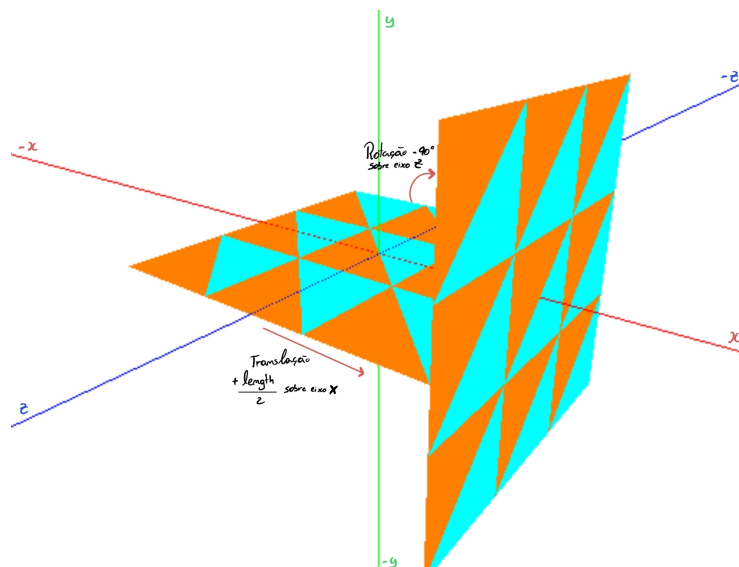


Figura 2.5: Aplicação sobre o Plano rotação de -90° sobre o eixo do Z e translação sobre o eixo positivo do X.

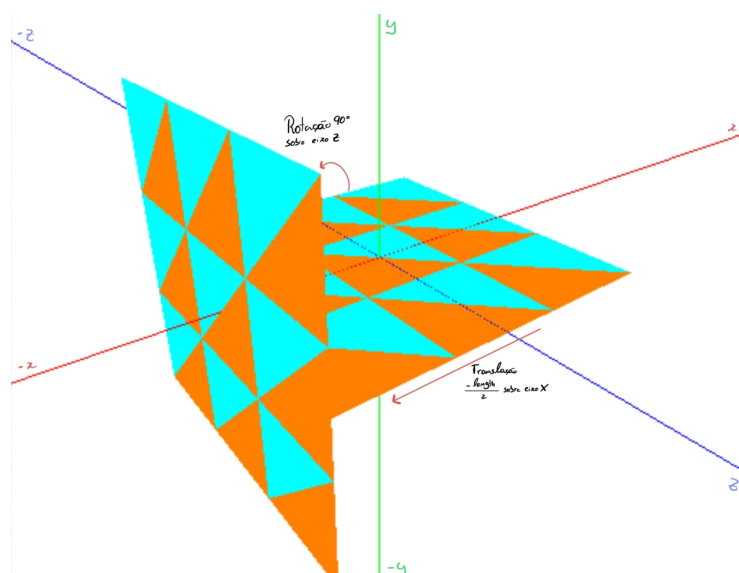


Figura 2.6: Aplicação sobre o Plano rotação de 90° sobre o eixo do Z e translação sobre o eixo negativo do X.

2.2.2 Planos Paralelos a XY

Já para o cálculo dos planos paralelos ao plano XY iremos efetuar rotações sobre o eixo X, uma rotação de 90° para a face cujo z é positivo e uma rotação de -90° para a face cujo z é negativo. Aplicando as substituições obtem-se:

$$R_{x,90} : \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ 0 \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ -z \\ 0 \\ 1 \end{bmatrix} \quad (2.8)$$

$$R_{x,-90} : \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ 0 \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ z \\ 0 \\ 1 \end{bmatrix} \quad (2.9)$$

A partir das equações 2.8 e 2.9 verifica-se que qualquer ponto do plano XZ cujo $y=0$ é conversível para um ponto do plano XY cujo $z=0$ substituindo o y pelo z . Conforme a rotação que foi feita, o z assume valores positivos ou negativos. De seguida, basta efetuar uma translação de $\text{length}/2$ sobre o eixo Y no plano que sofreu a rotação de 90° e uma translação de $-\text{length}/2$ no plano que sofreu a rotação de -90° , obtendo assim mais duas faces da caixa.

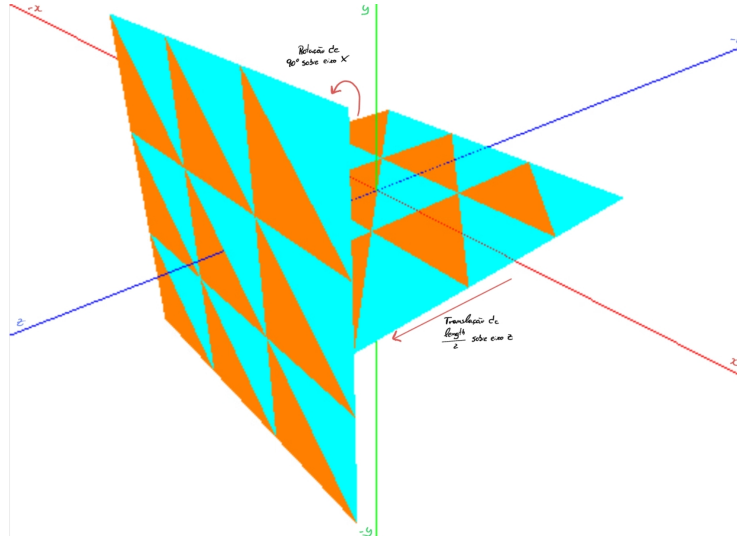


Figura 2.7: Aplicação sobre o Plano rotação de 90° sobre o eixo do X e translação sobre o eixo positivo do Z.

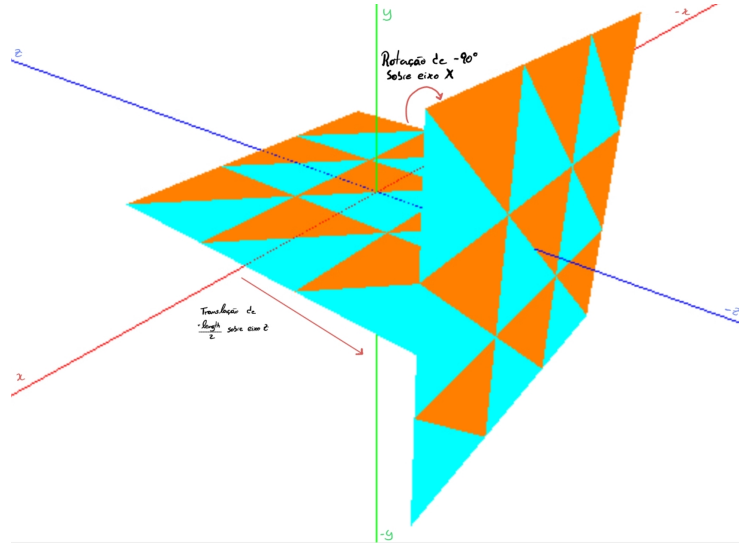


Figura 2.8: Aplicação sobre o Plano rotação de -90° sobre o eixo do **X** e translação sobre o eixo negativo do **Z**.

2.2.3 Planos Paralelos a XZ

Uma vez que o plano original do qual foram efetuadas as rotações é o plano paralelo a **XZ**, basta realizar uma translação de **length/2** sobre o eixo **Z** nesse plano original para obter a face de cima da caixa.

No entanto, para obter a face de baixo do quadro, primeiro é necessário efetuar uma rotação de 180° sobre o eixo **X**. Fazendo as substituições:

$$R_{x,180} : \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ 0 \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ 0 \\ -z \\ 1 \end{bmatrix} \quad (2.10)$$

Assim podemos verificar a partir da equação 2.10 que para obter o ponto pretendido basta colocar o valor do **z** negativo. A seguir, aplicando uma translação sobre o eixo **Z** de **-length/2** é obtida a face de inferior da caixa.

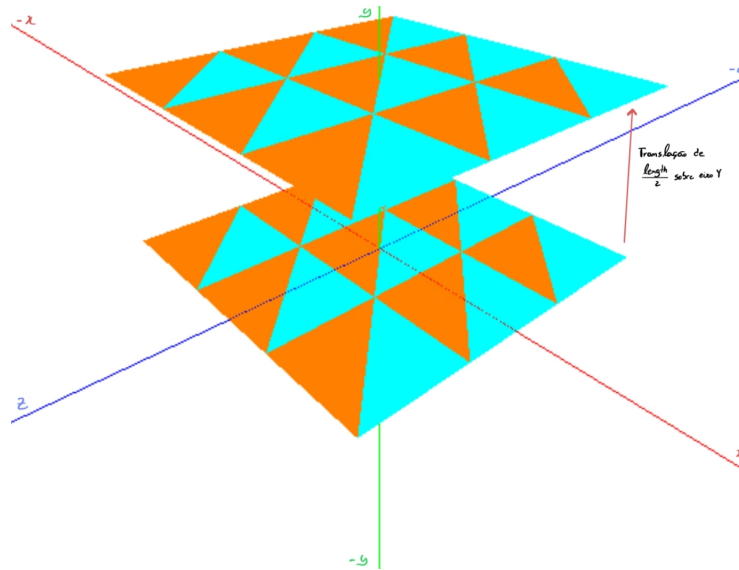


Figura 2.9: Aplicação sobre o Plano translação sobre o eixo positivo do Y.

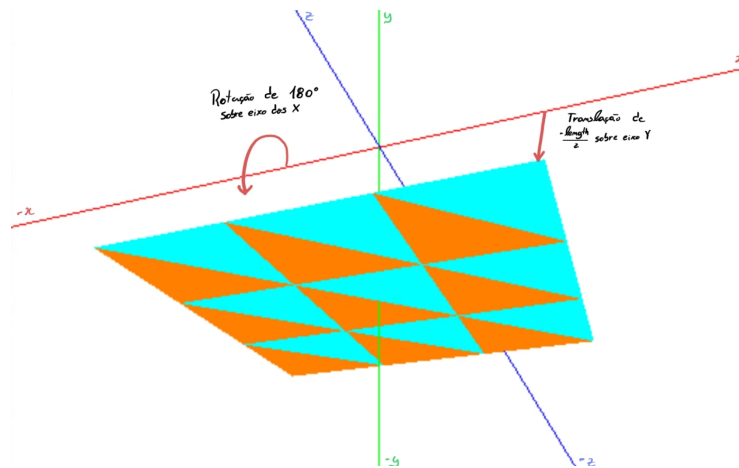
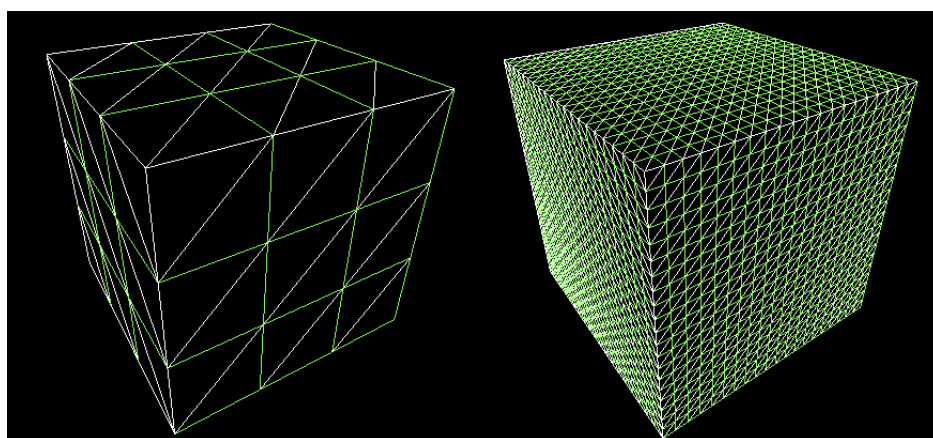


Figura 2.10: Aplicação sobre o Plano rotação de 180° sobre o eixo do X e translação sobre o eixo negativo do Y.

A partir destes simples cálculos matriciais que demonstramos nas últimas três secções, conseguimos calcular todos os pontos necessários para construir a caixa em apenas uma iteração.

Na Figura 2.11 apresenta-se o resultado de dois desenhos de uma caixa, ambas com $\text{length} = 2$, no entanto uma com $\text{grid } 3 \times 3$ e outra com $\text{grid } 20 \times 20$.



(a) Box com $length = 2$ e $grid = 3$

(b) Box com $length = 2$ e $grid = 20$

Figura 2.11: Exemplos de geração de diferentes Caixas

2.3 Esfera

Na construção da esfera primitiva, foram calculadas as coordenadas esféricas $(\alpha, \beta, \text{raio})$ para cada vértice. Em seguida, foram convertidas para o sistema cartesiano (x, y, z) usando as seguintes equações:

$$x = \text{raio} \times \cos \beta \times \sin \alpha \quad (2.11)$$

$$y = \text{raio} \times \sin \beta \quad (2.12)$$

$$z = \text{raio} \times \cos \beta \times \cos \alpha \quad (2.13)$$

A escolha de coordenadas polares foi feita devido à facilidade de calcular vértices usando um ângulo α e β , além de um raio constante a partir de um ponto central. A variação ocorre apenas nos ângulos α e β conforme o número de fatias (*slices*) e camadas (*stacks*), como ilustrado na Figura 2.12.

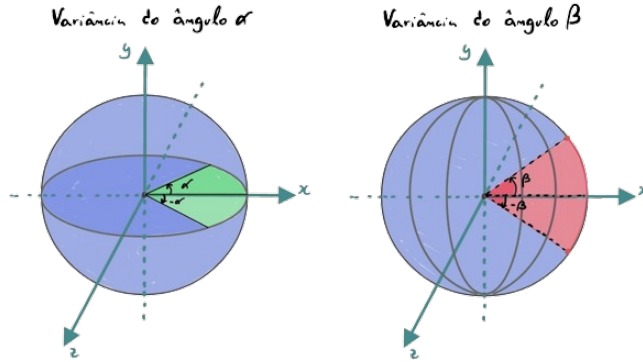


Figura 2.12: Exemplo da variação de α e β em uma esfera.

Para calcular a variação dos ângulos α e β e, posteriormente, os vértices da esfera, foram usadas as seguintes equações:

$$diff_slices(\Delta\alpha) = \frac{2\pi}{slices} \quad (2.14)$$

$$diff_stacks(\Delta\beta) = \frac{\pi}{stacks} \quad (2.15)$$

Para cada fatia, o valor de α varia entre $[i \times \Delta\alpha, (i + 1) \times \Delta\alpha]$, e para cada camada, o valor de β varia entre $[j \times \Delta\beta, (j + 1) \times \Delta\beta]$, onde i é o número da fatia a ser desenhada e j é o número da camada.

O cálculo dos pontos da esfera envolve calcular os vértices das camadas para cada fatia. Esse processo diferenciou o cálculo das camadas no topo e na base da esfera, das camadas que compõem o corpo. Cada fatia da esfera, as camadas que compõem o corpo são formadas por dois triângulos, enquanto as camadas que compõem o topo e a base da esfera são formadas apenas por um triângulo, como exemplificado no esquema da Figura 2.13.

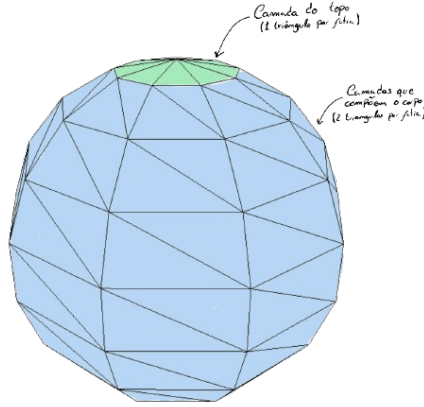


Figura 2.13: Distinção entre camadas de uma esfera.

Assim, para cada fatia, iniciando o cálculo do centro da esfera, os vértices das camadas superiores e inferiores são calculados simultaneamente, utilizando valores de β positivos e negativos. Isso otimiza o número total de iterações necessárias para calcular os vértices.

Isso resulta em dois casos possíveis, um onde o número de camadas da esfera é par e o cálculo do valor β coincide com o plano XZ , e outro onde o número de camadas é ímpar.

Para ambos os casos os valores de alfa iniciais são calculados usando:

$$\alpha = i \times \text{diff_slices} \quad (2.16)$$

$$\text{next_alpha} = (i + 1) \times \text{diff_slices} \quad (2.17)$$

Para o caso onde o número de camadas é ímpar, a camada central é calculada inicialmente usando os seguintes valores de β iniciais:

$$\beta_{Up} = \frac{\Delta\beta}{2} \quad (2.18)$$

$$\beta_{Down} = -\beta_{Up} \quad (2.19)$$

Com esses valores iniciais de β_{Up} e β_{Down} , quatro pontos são então calculados para formar o quadrilátero da camada central e, assim, formar os dois triângulos que a compõem.

Já para o caso onde o número de camadas é par os valores de β_{Up} e β_{Down} são zero.

Para calcular as camadas restantes que formam o corpo da esfera, são calculados oito pontos simultaneamente: quatro pontos para as camadas com $y > 0$ e os outros quatro para $y < 0$.

Para a camada superior, são utilizados dois valores de β :

$$\beta_{Sup1} = j \times \beta_{Up} \quad (2.20)$$

$$\beta_{Sup2} = (j + 1) \times \beta_{Up} \quad (2.21)$$

Enquanto para as camadas inferiores, os valores de β são:

$$betaInf1 = j \times betaDown \quad (2.22)$$

$$betaInf2 = (j + 1) \times betaDown \quad (2.23)$$

Para ambos os casos, os valores de α são determinados conforme as Equações 2.16 e 2.17.

Com base nessas equações, todos os vértices das camadas que compõem o corpo da esfera podem ser calculados, conforme a ordem exibida na Figura 2.14 e na Figura 2.15 que representa uma fatia da esfera com nove camadas.

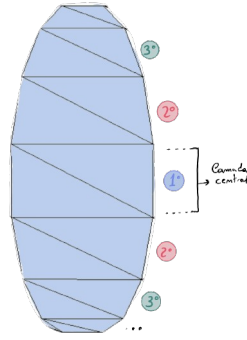


Figura 2.14: Exemplo da ordem de criação das camadas de uma fatia de uma esfera com nove camadas.

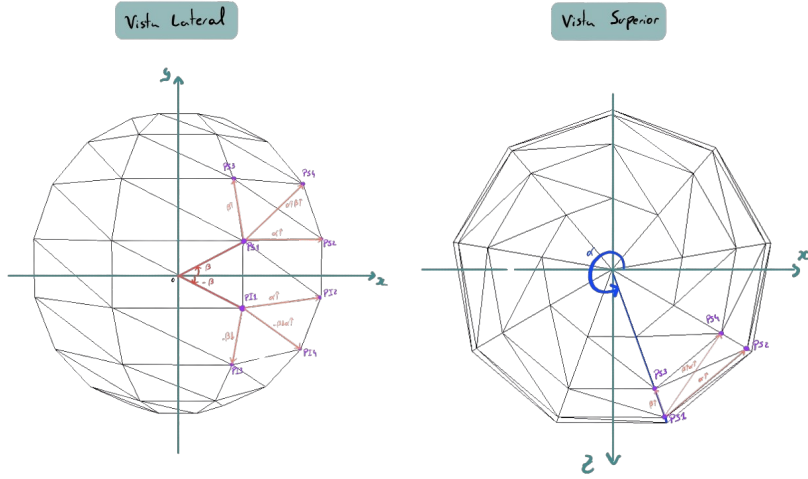
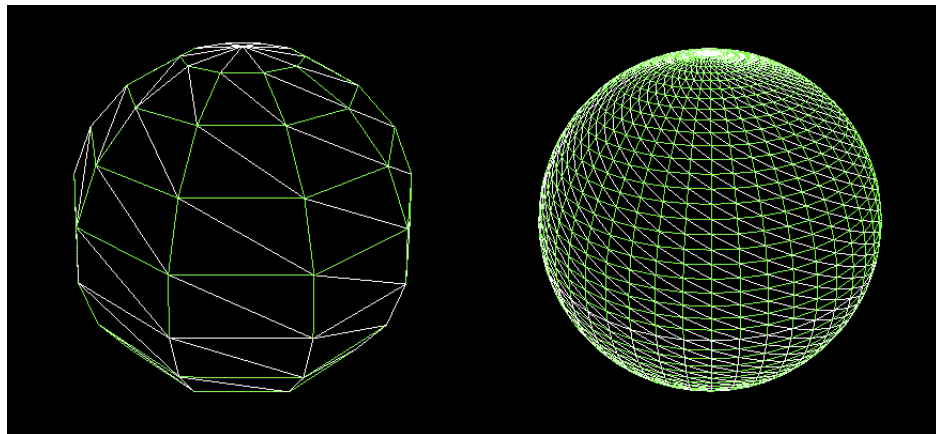


Figura 2.15: Exemplo da ordem de calculo das coordenadas de uma camada de uma esfera.

Por fim, são calculados os vértices dos triângulos da camada superior e inferior da esfera. Na camada superior, são utilizados os dois pontos da camada mais alta do corpo da esfera, juntamente com o ponto de coordenadas $(0, \text{radius}, 0)$. Já na camada inferior, são utilizados os pontos da camada mais baixa do corpo da esfera e o ponto de coordenadas $(0, -\text{radius}, 0)$.

Na Figura 2.18 apresenta-se o resultado de dois desenhos de uma esfera, ambas com $\text{raio} = 1$, no entanto uma tem $\text{slices} = 10$ e $\text{stacks} = 10$ e outra tem $\text{slices} = 50$ e $\text{stacks} = 50$.



(a) Esfera com $\text{raio} = 1$, $\text{slices} = 10$ e $\text{stacks} = 10$ (b) Esfera com $\text{raio} = 1$, $\text{slices} = 50$ e $\text{stacks} = 50$

Figura 2.16: Exemplos de geração de diferentes Esferas

2.4 Cone

O Cone é uma primitiva gráfica cuja sua base pertence ao plano **XZ**, centrado na origem. Para o cálculo dos seus vértices, recorreu-se ao uso de coordenadas polares, devido a todos os vértices do cone estarem circunscritos numa "circunferência", que posteriormente foram convertidas para a sua representação cartesiana através das seguintes equações:

$$x = radius \times \sin \alpha \quad (2.24)$$

$$z = radius \times \cos \alpha \quad (2.25)$$

A abordagem adotada passou por duas fases distintas: primeiramente, é calculado todos os vértices pertencentes à base desta primitiva e, em seguida, é calculado os vértices pertencentes às suas fases laterais.

No caso do cálculo dos vértices da base, é necessário possuir o raio do cone e o ângulo α a utilizar em cada fatia (*slice*).

Para a variação do ângulo α a utilizar, recorreu-se à equação:

$$alfa_diff(\Delta\alpha) = \frac{2\pi}{slices} \quad (2.26)$$

Obtendo assim o incremento a utilizar em cada fatia da base, tal que:

$$alfa = i \times alfa_diff \quad (2.27)$$

$$next_alfa = (i + 1) \times alfa_diff \quad (2.28)$$

onde i representa o número da fatia a desenhar. Com estes valores, foi possível calcular todos os pontos que formam a base do cone, como se pode observar pelo esquema da Figura

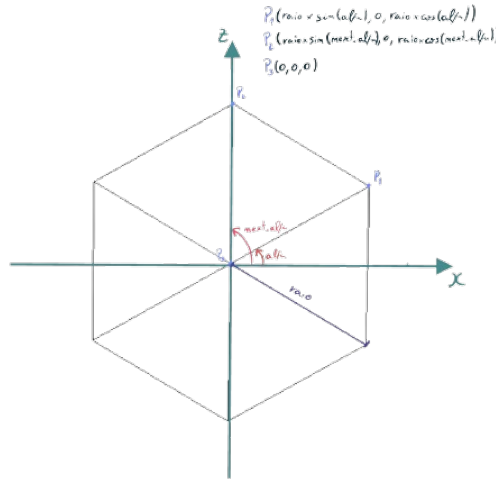
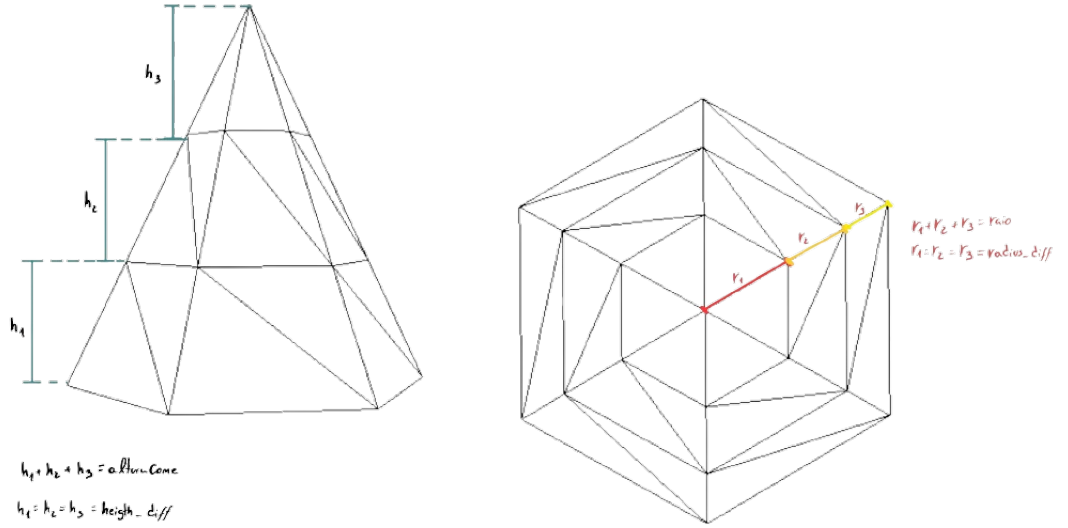


Figura 2.17: Exemplo do calculo dos vértices da de uma fatia da base de um cone com seis *slices*.

Após o cálculo dos pontos pertencentes à base do cone, deu-se o cálculo dos vértices pertencentes às suas faces laterais. A estratégia adotada foi semelhante ao cálculo da base, porém, neste caso será necessário descobrir qual a variação na altura das camadas e do seu raio.
Para tal usou-se as seguinte equações:

$$\text{height_diff} = \frac{\text{height}}{\text{stacks}} \quad (2.29)$$

$$\text{radius_diff} = \frac{\text{radius}}{\text{stacks}} \quad (2.30)$$



(a) Esquema ilustrativo do calculo da variância da altura de um cone. (b) Esquema ilustrativo do calculo da variância do raio das camadas de um cone.

Figura 2.18: Exemplos da variância do raio e altura de um cone por cada camada.

Com os valores das equações 2.26, 2.29 e 2.30 é então possível deduzir os valores do raio, altura e ângulo α a utilizar para o cálculo dos quatro pontos pertencentes às camadas do cone tal que:

$$\text{currentRadius} = \text{radius} - (i \times \text{radius_diff}) \quad (2.31)$$

$$\text{nextRadius} = \text{radius} - ((i + 1) \times \text{radius_diff}) \quad (2.32)$$

$$\text{currentHeight} = i \times \text{height_diff} \quad (2.33)$$

$$\text{nextHeight} = (i + 1) \times \text{height_diff} \quad (2.34)$$

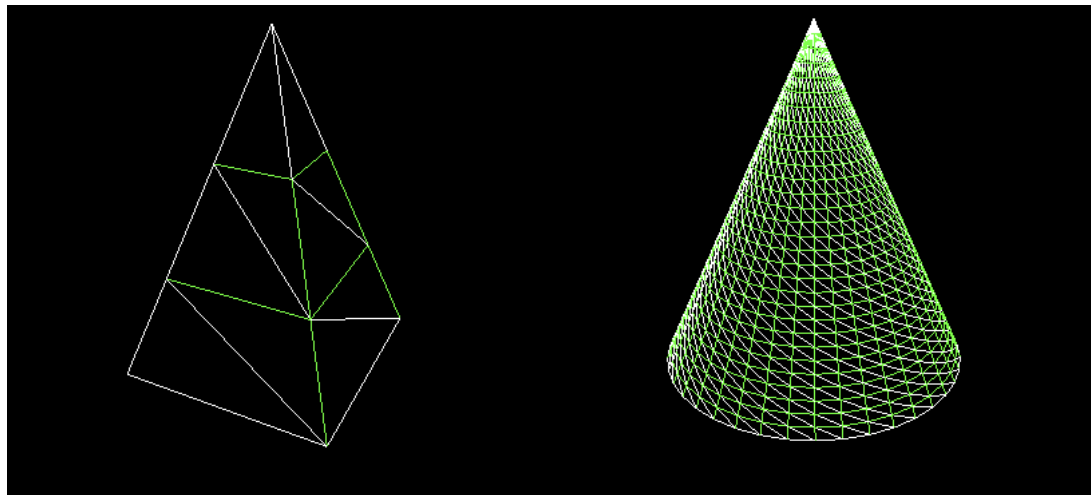
onde i representa o número da fatia a desenhar.

Assim o cálculo dos vértices que irão formar o quadrilátero pertencente à lateral do cone é dado por:

- P1 ($\text{currentRadius} \times \sin(\text{next_alfa})$, currentHeight , $\text{currentRadius} \times \cos(\text{next_alfa})$)
- P2 ($\text{nextRadius} \times \sin(\text{alfa})$, nextHeight , $\text{nextRadius} \times \cos(\text{alfa})$)
- P3 ($\text{currentRadius} \times \sin(\text{alfa})$, currentHeight , $\text{currentRadius} \times \cos(\text{alfa})$)
- P4 ($\text{nextRadius} \times \sin(\text{next_alfa})$, nextHeight , $\text{nextRadius} \times \cos(\text{next_alfa})$)

Utilizando os cálculos mostrados anteriormente, o processo é repetido em todas as camadas pertencentes a uma fatia do cone, onde com dois pontos pertencentes à base do cone são calculados os pontos da camada superior e repetido o processo até à última camada do cone. Este processo é repetido em todas as fatias.

Na Figura 2.19 apresenta-se o resultado de dois desenhos de um cone, ambas com `raio = 1` e `altura = 2`, no entanto um tem `slices = 4` e `stacks = 3` e o outro tem `slices = 40` e `stacks = 30`.



(a) Cone com `slices = 4` e `stacks = 3`

(b) Cone com `slices = 40` e `stacks = 30`

Figura 2.19: Exemplos de geração de diferentes Cones

2.5 Modo de Uso

2.5.1 Criação do executável

Durante o desenvolvimento do projeto foi criado um ficheiro designado por `CMakeLists` que com o auxílio da ferramenta `CMake` irá compilar ambas as aplicações *Generator* e *Engine*. Para tal é necessário seguir os seguintes passos na pasta raiz do projeto:

1. `$ mkdir build`
2. `$ cd build`
3. `cmake ...`
4. `make`

Após a execução destes passos irão ser criados ambos os executáveis.

Criação dos Modelos

Depois de gerar o executavel do ***Generator*** os modelos podem ser criados com os seguintes comandos:

- **Plano:** \$./generator plane length divisions "namefile".3d
- **Caixa:** \$./generator box length divisions "namefile".3d
- **Esfera:** \$./generator sphere radius slices stacks "namefile".3d
- **Cone:** \$./generator cone radius height slices stacks "namefile".3d

Capítulo 3

Engine

No *Engine* é onde fazemos desde o *parsing* de ficheiros XML, à leitura de modelos, desenho das primitivas, movimentação da câmara até à interação com o teclado.

Dado que a informação relativa à câmara, janela e modelos se encontra num documento XML, recorremos à biblioteca *tinixml2* para efetuar o *parsing* deste tipo de ficheiro. Assim, construímos o método *parseXML* que recebe como argumento o nome do ficheiro XML de onde vai recolher a informação contida no mesmo.

De seguida, vai procurar os vários elementos que o ficheiro pode conter e irá armazená-los em estruturas de dados criadas para o efeito. A informação da câmara é guardada em *struct Camara* onde armazena informação relativa aos parâmetros intrínsecos, nomeadamente informação sobre a projeção, tais como o *Field Of View*, *Near Plane*, *Far Plane* e também parâmetros intrínsecos, tais como a sua **Posição**, **Orientação** e **Direção**. Uma vez que a câmara se move na superfície de uma esfera, olhando sempre para o seu centro, resolvemos utilizar coordenadas esféricas para guardar a sua posição, guardando portanto o seu **Raio**, o ângulo **Alfa** e o ângulo **Beta**. De forma a obter esses valores chegamos às seguintes equações que calculam estes valores a partir dos três pontos relativos à posição da câmara fornecidos no ficheiro XML:

$$raio = \sqrt{px^2 + py^2 + pz^2} \quad (3.1)$$

$$alfa = \arccos(pz / \sqrt{px^2 + pz^2}) \quad (3.2)$$

$$beta = \arcsin(py / raio) \quad (3.3)$$

Para além disso, também é recolhida informação sobre a tela que é armazenada em *struct Window*, tal como a sua **Altura** e **Largura**.

Por último, informa também de quais os modelos que devem ser carregados. Assim, procede-se à leitura sequencial de cada um destes ficheiros .3D e são guardados todos os pontos lidos dentro de uma variável global *vectorjPointj* de forma a posteriormente proceder-se ao desenho dos mesmos.

O desenho dos triângulos é feito no método *renderScene* onde é desenhado um triângulo de cada vez alternando a cor interpoladamente. Também neste método são desenhados os eixos X, Y e Z para efeitos de auxílio de visualização e também se define os parâmetros da câmara tendo em conta a estrutura mencionada anteriormente. De forma a definir a posição da câmara é necessário agora

recorrermos às seguintes equações que definimos anteriormente para calcular as coordenadas cartesianas: 2.11, 2.12 e 2.13.

Por último, relativamente à interação com o teclado, recorreremos aos métodos *keyboardCallback* e *processSpecialKeys* para mover a câmara para a esquerda e para a direita através do decremento e incremento de alfa, respetivamente, mover a câmara para cima e para baixo incrementando e decrementando o beta, respetivamente, existindo o limite $\beta \leq |1.5|$ e também temos a possibilidade de alterar a *face* e o *mode* de *GLPolygonMode* e incrementar e diminuir o raio, de forma a aproximar e distanciar a câmara, tudo isto meramente para efeitos de visualização.

3.1 Modo de Uso

3.1.1 Execução da Engine

De modo a executar os ficheiros criados, é necessário fornecer ao *engine* um ficheiro *.xml* situado na pasta *XMLFiles*. No qual, dentro desse ficheiro, tem que existir a referência para um ficheiro *.3d*. Por fim, pode ser executado com o seguinte comando:

- `$./engine "namefile".xml`

3.1.2 Comandos

Como forma de interação com a aplicação **Engine** foram implementados os seguintes comandos:

- **Seta para Cima** - *Zoom IN*.
- **Seta para Baixo** - *Zoom OUT*.
- **A** - Rotação da câmara para a Esquerda.
- **D** - Rotação da câmara para a Direita.
- **W** - Rotação da câmara para a Cima.
- **S** - Rotação da câmara para a Baixo.
- **F** - Visualizar o preenchimento da primitiva
- **L** - Visualizar as arestas que definem a primitiva
- **P** - Visualizar os vértices que definem a primitiva
- **B** - Visualizar a parte de trás da primitiva.
- **N** - Visualizar a parte de frontal da primitiva.
- **M** - Visualizar simultaneamente a parte frontal e de trás.

Capítulo 4

Conclusão

De uma forma geral, consideramos que a realização desta fase foi bem sucedida, uma vez que acreditamos que cumprimos com todos os objetivos propostos para a mesma e ainda procuramos introduzir alguns extras, como é o caso a possibilidade de interação com o sistema, nomeadamente a movimentação da câmara e as opções de visualização das primitivas geradas.

Para além disso, esta primeira fase permitiu a consolidação através da prática de conhecimentos obtidos nas aulas teóricas assim como nas aulas práticas, expandindo o nosso conhecimento sobre a ferramenta *OpenGL*, *C++* e também a manipulação de ficheiros XML.

Em suma, é notável que a realização desta fase é essencial para o restante desenvolvimento do projeto dado que agora possuímos mais facilidade em trabalhar com as bibliotecas utilizadas.