



Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Computação Gráfica

Ano Letivo 2022/2023

Trabalho prático

Parte 1 - Primitivas gráficas

Grupo 14

Ana Rita Santos Poças, A97284
Miguel Silva Pinto, A96106
Orlando José da Cunha Palmeira, A97755
Pedro Miguel Castilho Martins, A97613

10 de março de 2023

Trabalho prático

Parte 1 - Primitivas gráficas

Grupo 14

Ana Rita Santos Poças, A97284

Miguel Silva Pinto, A96106

Orlando José da Cunha Palmeira, A97755

Pedro Miguel Castilho Martins, A97613

10 de março de 2023

Índice

1	Introdução	1
2	Estruturação do Projeto	2
3	Generator	3
3.1	Plano (Plane)	4
3.2	Caixa (Box)	6
3.3	Esfera (Sphere)	8
3.4	Cone	10
4	Engine	13
5	Demonstração	14
5.1	Manual de utilização	14
5.2	Execução do programa	14
6	Conclusão	15

Índice de figuras

3.1	Exemplo ilustrativo do formato dos ficheiros .3d gerados	3
3.2	Demonstração da ordem dos pontos que originam os triângulos que formam o plano .	4
3.3	Pseudocódigo da função geradora dos pontos do plano	5
3.4	Construção de um quadrado que compõe o plano	5
3.5	Construção do plano	6
3.6	Exemplo de um plano com $length = 1$, $divisions = 3$	6
3.7	Exemplo de uma caixa com $length = 1$ e $divisions = 3$	7
3.8	Exemplo ilustrativo de um valor $\Delta\alpha$ numa esfera.	8
3.9	Exemplo de esfera com 7 stacks	9
3.10	Exemplo ilustrativo de um valor $\Delta\beta$ numa esfera	9
3.11	Exemplo uma esfera com $raio = 1$, $slices = 10$ e $stacks = 10$	10
3.12	Ilustração do $\Delta\alpha$ obtido relativamente às slices do cone	10
3.13	Exemplo de obtenção dos deltas com um cone de 4 slices e 3 stacks	11
3.14	Exemplo de um Δ_y para um cone com 4 stacks	11
3.15	Exemplo de um cone com $raio = 1$, $altura = 2$, $slices = 10$ e $stacks = 10$	12

1 Introdução

Este relatório foi elaborado no âmbito da unidade curricular de Computação Gráfica, na qual nos foi proposto o desenvolvimento de duas aplicações: o **Generator**, um gerador de vértices para primitivas gráficas como planos, caixas, esferas e cones, e a **Engine**, uma aplicação capaz de ler ficheiros de configuração em XML para desenhar os vértices das primitivas gráficas geradas previamente.

Para a implementação destas aplicações, utilizámos a linguagem de programação C++ e recorreremos à ferramenta OpenGL.

Ao longo deste relatório, iremos descrever de forma detalhada as decisões e abordagens que foram adotadas e que permitiram a implementação das aplicações propostas.

2 Estruturação do Projeto

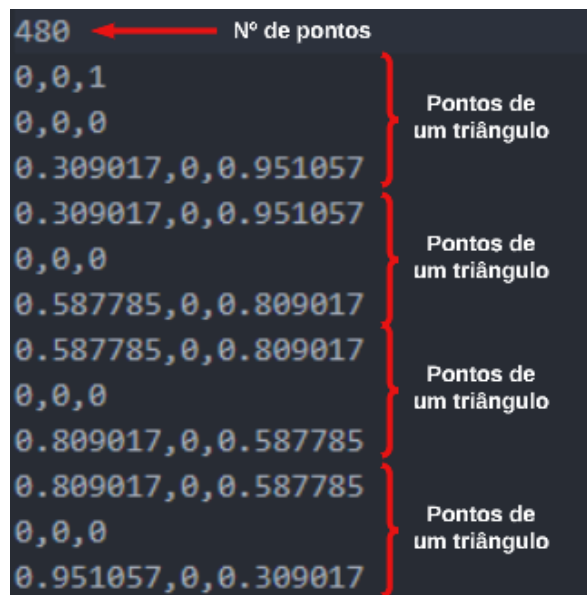
De forma a permitir uma melhor organização e modularização do código, o projeto foi estruturado de acordo com as seguintes diretorias:

- **engine**: contém o código necessário para a visualização 3d dos modelos.
- **generator**: contém o código necessário para o cálculo das coordenadas dos pontos para a representação do plano, caixa, esfera e cone bem como a criação dos ficheiros .3d de forma a depois serem utilizados pelo Engine.
- **tinyXML**: contém o package tinyXML, de forma a auxiliar a leitura de ficheiros XML, disponível em <http://www.grinninglizard.com/tinyxml>.
- **utils**: contém as estruturas e funções auxiliares que o *Engine* e o *Generator* utilizam para o seu devido funcionamento.

3 Generator

O generator é a aplicação responsável por calcular os pontos dos triângulos que permitem construir as diversas primitivas para um ficheiro .3d.

A estrutura do ficheiro resultante contém na primeira linha o número de pontos que constam no ficheiro, e nas restantes linhas apresentam-se os pontos. Os pontos encontram-se especificados em cada linha por três valores float separados por vírgulas, em que cada um desses valores representa a coordenada X , Y e Z , respetivamente.



The image shows a text-based representation of a .3d file. The first line is '480', with a red arrow pointing to it from the label 'Nº de pontos'. The following lines are grouped into four sets of three, each labeled 'Pontos de um triângulo' on the right. The first triangle's points are '0,0,1', '0,0,0', and '0.309017,0,0.951057'. The second triangle's points are '0.309017,0,0.951057', '0,0,0', and '0.587785,0,0.809017'. The third triangle's points are '0.587785,0,0.809017', '0,0,0', and '0.809017,0,0.587785'. The fourth triangle's points are '0.809017,0,0.587785', '0,0,0', and '0.951057,0,0.309017'. Red curly braces on the right side group the lines for each triangle.

```
480 ← Nº de pontos
0,0,1
0,0,0
0.309017,0,0.951057
0.309017,0,0.951057
0,0,0
0.587785,0,0.809017
0.587785,0,0.809017
0,0,0
0.809017,0,0.587785
0.809017,0,0.587785
0,0,0
0.951057,0,0.309017
```

Figura 3.1: Exemplo ilustrativo do formato dos ficheiros .3d gerados

Para construir os pontos para as diferentes primitivas, é necessário fornecer ao Generator os seguintes argumentos, consoante a primitiva em causa: Plano (*length*, *divisions*), Caixa (*length*, *divisions*), Esfera (*radius*, *slices*, *stacks*) e Cone (*radius*, *height*, *slices*, *stacks*).

Passamos agora a explicar em mais detalhe, os processos e estratégias para a construção de cada figura que o Generator é capaz de gerar.

3.1 Plano (Plane)

Os planos que se pretendem gerar no Generator devem estar contidos no plano XZ e o seu centro deve localizar-se no ponto $(0, 0, 0)$, a origem. Para além disso, a construção dos planos deve basear-se numa junção de triângulos.

Tendo em conta estes requisitos, seguiu-se o seguinte raciocínio para fazer a construção dos planos:

- O plano será visto como uma **matriz** de dimensão $divisions \times divisions$ em que cada elemento dessa matriz é um quadrado de lado $length/divisions$ dividido em dois triângulos cuja hipotenusa é a diagonal desse quadrado.
- A função de geração dos pontos do plano inicia com 4 pontos com as seguintes coordenadas:
 - $p1 = (-length/2, 0, -length/2)$
 - $p2 = (-length/2, 0, -length/2 + length/divisions)$
 - $p3 = (-length/2 + length/divisions, 0, -length/2)$
 - $p4 = (-length/2 + length/divisions, 0, -length/2 + length/divisions)$
- A matriz que representa o plano será construída linha a linha deslocando os pontos $p1$, $p2$, $p3$ e $p4$ pelo eixo do X (cada deslocação tem um comprimento de $length/divisions$). Cada elemento da linha (um quadrado) é composto por dois triângulos cujos vértices são $[p1, p2, p3]$ e $[p2, p4, p3]$ (a ordem dos vértices serve para o plano ser visto de cima, isto é, onde $y > 0$).

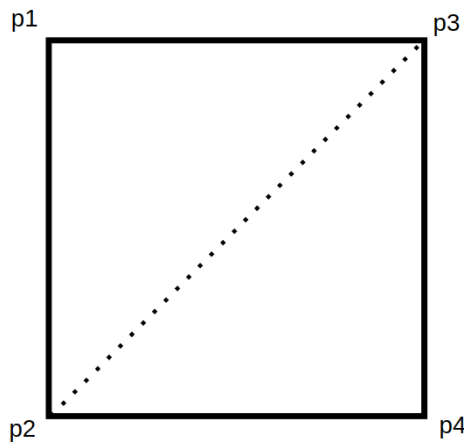


Figura 3.2: Demonstração da ordem dos pontos que originam os triângulos que formam o plano

- No fim da construção de cada linha, as coordenadas $p1_x$, $p2_x$, $p3_x$ e $p4_x$ voltam aos valores iniciais e as coordenadas $p1_z$, $p2_z$, $p3_z$ e $p4_z$ são incrementadas com o valor $length/divisions$. Posteriormente, repete-se o ponto anterior para construir uma nova linha.

O raciocínio explicitado acima pode ser resumido no seguinte excerto de pseudocódigo que representa a função que calcula os pontos do plano:


```

1  def generatePlane(length, divisions):
2      div_side = length / divisions
3      dimension2 = length / 2
4      p1 = (-dimension2, 0, -dimension2)
5      p2 = (-dimension2, 0, -dimension2 + div_side)
6      p3 = (-dimension2 + div_side, 0, -dimension2)
7      p4 = (-dimension2 + div_side, 0, -dimension2 + div_side)
8      plane = []
9      for linha in range(length):
10         for coluna in range(length):
11             # Primeiro triângulo do quadrado
12             plane.append((x(p1) + coluna * div_side, 0, z(p1)))
13             plane.append((x(p2) + coluna * div_side, 0, z(p2)))
14             plane.append((x(p3) + coluna * div_side, 0, z(p3)))
15             # Segundo triângulo do quadrado
16             plane.append((x(p2) + coluna * div_side, 0, z(p2)))
17             plane.append((x(p4) + coluna * div_side, 0, z(p4)))
18             plane.append((x(p3) + coluna * div_side, 0, z(p3)))
19         z(p1) += div_side
20         z(p2) += div_side
21         z(p3) += div_side
22         z(p4) += div_side
23     return plane

```

Figura 3.3: Pseudocódigo da função geradora dos pontos do plano

Para clarificar o processo de construção do plano, apresenta-se a seguir um exemplo em que se constrói um plano com $length = 2$ e $divisions = 3$.

O processo de construção de um dos quadrados que compõem o plano é feito de acordo com a seguinte figura:

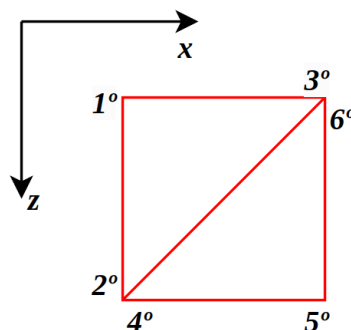


Figura 3.4: Construção de um quadrado que compõe o plano

Na figura acima, é apresentada a ordem que os vértices são criados para formar os dois triângulos.

Para formar o plano completo, os quadrados que formam o plano (figura 3.4) são construídos na seguinte ordem:

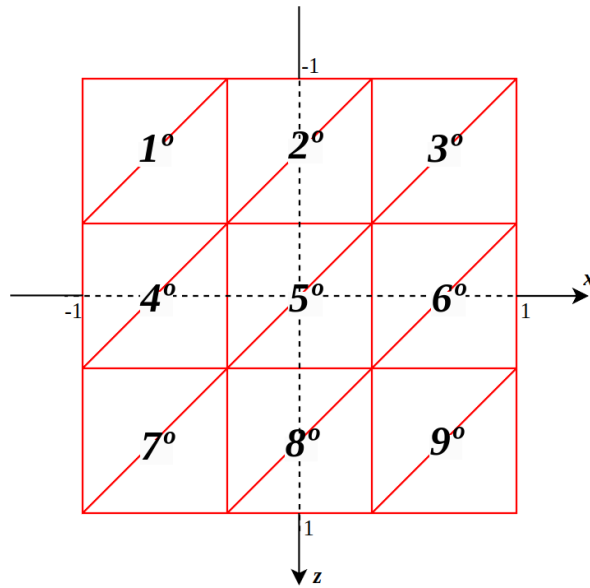


Figura 3.5: Construção do plano

Na figura seguinte, apresentamos um exemplo de um plano que é desenhado através do ficheiro .3d gerado:

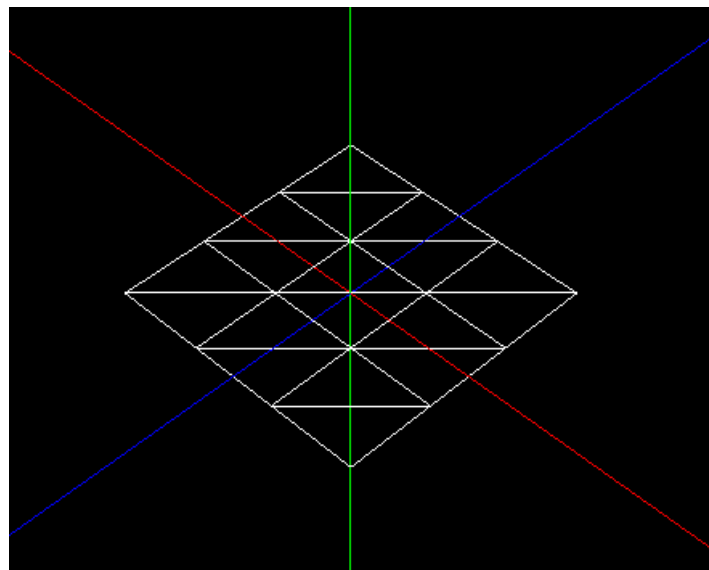


Figura 3.6: Exemplo de um plano com $length = 1$, $divisions = 3$

3.2 Caixa (Box)

Para o cálculo dos pontos, utilizou-se o mesmo raciocínio aplicado à construção do plano.

Na criação do plano, geramos um plano paralelo a XZ com $y = 0$. Para a caixa, geramos seis planos paralelos a XZ , XY e YZ com $y = \pm length/2$, $z = \pm length/2$ e $x = \pm length/2$, respectivamente.

Cada uma das funções que geram os planos paralelos a XZ , XY e YZ iniciam com os seguintes pontos:

- Função geradora de um plano paralelo a XZ :

- $p1 = (-length/2, \pm length/2, -length/2)$
- $p2 = (-length/2, \pm length/2, -length/2 + length/divisions)$
- $p3 = (-length/2 + length/divisions, \pm length/2, -length/2)$
- $p4 = (-length/2 + length/divisions, \pm length/2, -length/2 + length/divisions)$

- Função geradora de um plano paralelo a XY :

- $p1 = (-length/2, -length/2, \pm length/2)$
- $p2 = (-length/2, -length/2 + length/divisions, \pm length/2)$
- $p3 = (-length/2 + length/divisions, -length/2, \pm length/2)$
- $p4 = (-length/2 + length/divisions, -length/2 + length/divisions, \pm length/2)$

- Função geradora de um plano paralelo a YZ :

- $p1 = (\pm length/2, -length/2, -length/2)$
- $p2 = (\pm length/2, -length/2, -length/2 + length/divisions)$
- $p3 = (\pm length/2, -length/2 + length/divisions, -length/2)$
- $p4 = (\pm length/2, -length/2 + length/divisions, -length/2 + length/divisions)$

O cálculo dos pontos a partir de $p1, p2, p3$ e $p4$, é feito exatamente da mesma maneira como foi descrito no plano. A única diferença é que as funções que geram os planos paralelos a XZ , XY e YZ têm uma *flag* denominada “reverse” que, dependendo do seu valor, altera a ordem em que os pontos são inseridos na figura de modo a permitir inverter a orientação do polígono. Por exemplo, na função que gera o plano paralelo a XZ , se a *flag* “reverse” tiver o valor 0, o plano poderá ser visto de cima ($y > 0$). Caso “reverse” seja igual a 1, o plano poderá ser visto de baixo ($y < 0$).

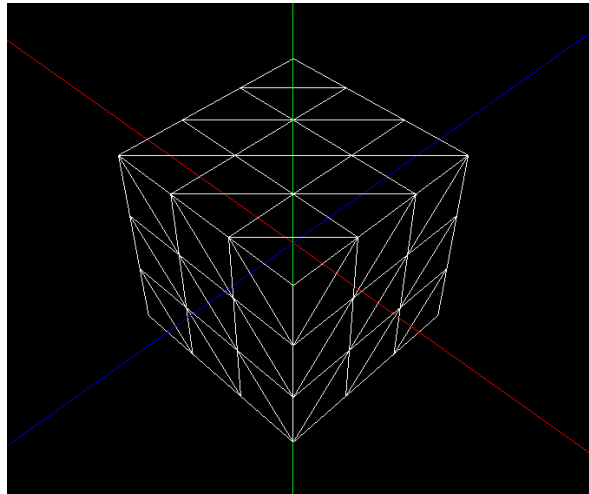


Figura 3.7: Exemplo de uma caixa com $length = 1$ e $divisions = 3$

3.3 Esfera (Sphere)

Para o cálculo dos pontos das esferas, recorreremos à utilização de coordenadas esféricas $(\alpha, \beta, raio)$, que são traduzidas para coordenadas cartesianas através das seguintes fórmulas:

- $x = raio \times \cos(\beta) \times \sin(\alpha)$
- $y = raio \times \sin(\beta)$
- $z = raio \times \cos(\beta) \times \cos(\alpha)$

Em que α representa o ângulo azimutal e β representa o ângulo polar.

De forma a conseguirmos calcular os pontos que constituem uma esfera, necessitamos dos seguintes argumentos: raio, slices (camadas verticais) e stacks (camadas horizontais). O número de pontos a calcular é diretamente proporcional ao valor dos dois últimos argumentos.

Inicialmente, são calculados os pontos base pelos quais se irão deduzir os restantes pontos. Para tal, é calculado um valor $\Delta\alpha$ que irá ser o resultado de $2\pi/slices$, que é o valor do ângulo pelo qual iremos deslocar o ponto base de origem $(0, 0, raio)$.

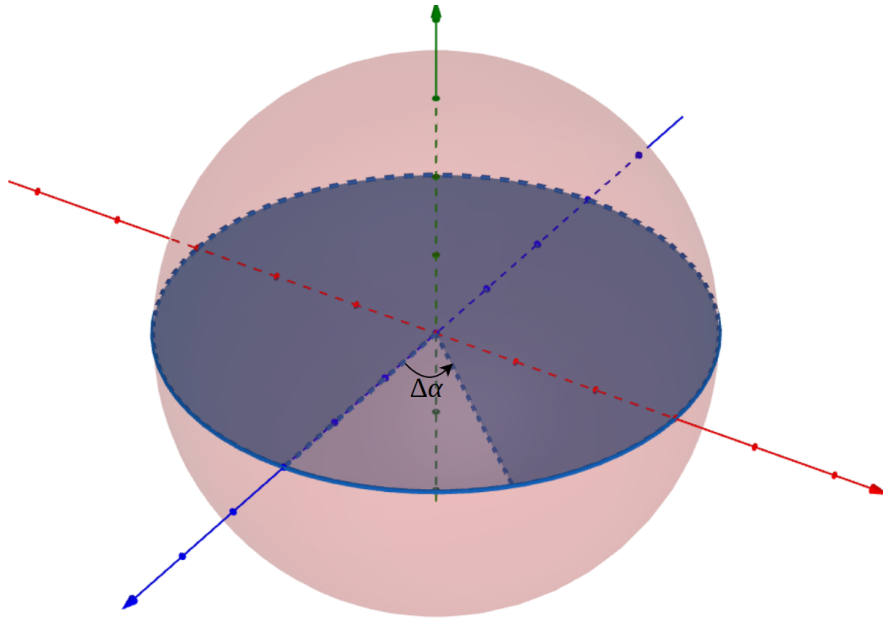


Figura 3.8: Exemplo ilustrativo de um valor $\Delta\alpha$ numa esfera.

No entanto, os pontos base da esfera podem não estar no plano XZ , no caso do número de camadas ser ímpar. Para ter isso em conta, é utilizado o valor de $\Delta\beta$ que é dado pela expressão $\pi/stacks$, em que são construídos dois conjuntos de pontos base, um conjunto com os valores de $\beta = \Delta\beta/2$ e o outro conjunto com os valores $\beta = -\Delta\beta/2$, tal como se encontra destacado na seguinte figura.

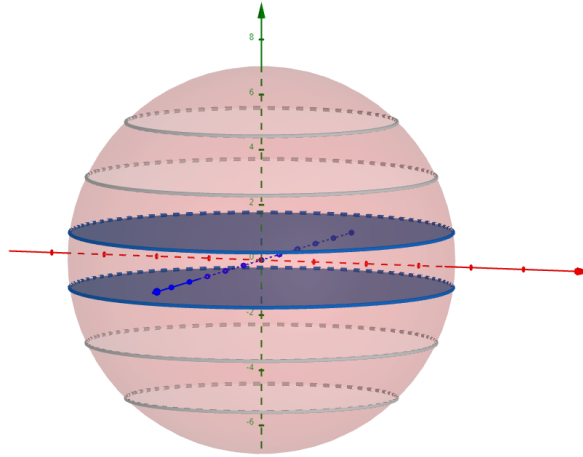


Figura 3.9: Exemplo de esfera com 7 stacks

Toda a restante lógica da determinação das coordenadas X e Z dos restantes pontos base através da variação de um valor alpha por $\Delta\alpha$ a cada ponto, aplica-se igualmente a este caso particular.

Para determinar as coordenadas do eixo Y , dos pontos relativos às stacks, apenas temos de alterar o valor de β nas coordenadas esféricas dos pontos base previamente calculados. Ou seja, para o desenho de uma slice da esfera, inicialmente pegamos em dois pontos base consecutivos e acrescentamos ao valor das suas coordenadas esféricas β , um valor $\Delta\beta$, que resulta de $\pi/stacks$. Para obtermos a parte da slice do hemisfério sul da esfera, o processo é semelhante, apenas tem de se somar o valor de β , por $-\Delta\beta$.

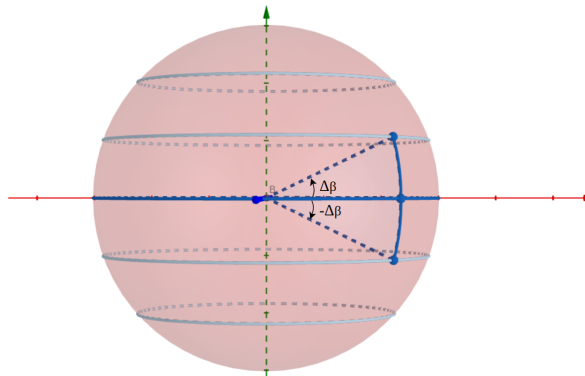


Figura 3.10: Exemplo ilustrativo de um valor $\Delta\beta$ numa esfera

Com a descoberta das coordenadas destes pontos, ordenam-se os pontos para desenhar os triângulos que formam a camada da slice, e repete-se o processo de incrementar/decrementar o valor de β desses pontos, para progredir na slice no sentido ascendente e no sentido descendente, simultaneamente. Pára-se quando se chega à última camada da slice, conectando os pontos base calculados das últimas stacks (última stack do hemisfério norte e do sul), com os pontos nos topos da esfera para ter uma slice completa. O processo repete-se para os restantes grupos de dois pontos base consecutivos, sendo a esfera construída slice a slice, em que uma slice é construída a partir do centro até aos polos superiores e inferiores.

Na figura seguinte, apresentamos um exemplo de uma esfera que é desenhada através do ficheiro .3d gerado:

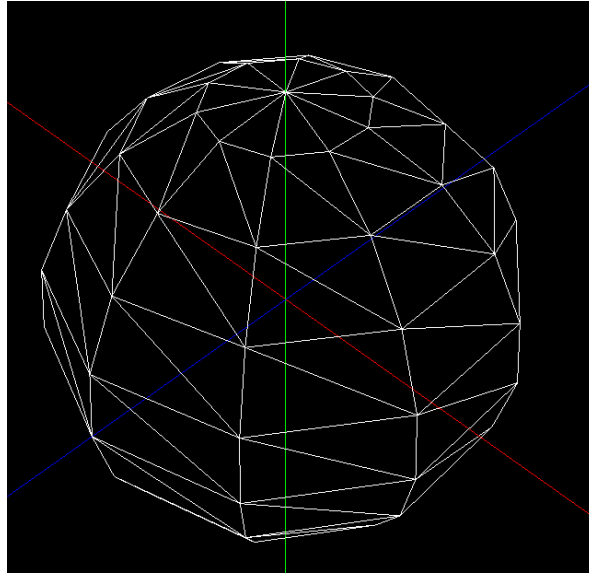


Figura 3.11: Exemplo uma esfera com $raio = 1$, $slices = 10$ e $stacks = 10$

3.4 Cone

A estratégia utilizada para calcular os pontos necessários para desenhar um cone, pode ser dividida em duas partes: o cálculo dos triângulos da base e os cálculos para as faces laterais do cone.

Relativamente à base do cone, foram utilizadas coordenadas esféricas, tal como na construção da esfera, sendo o processo de cálculo dos pontos da base do cone, similar ao da esfera, a única diferença é a necessidade de construir triângulos para a base.

Então para o cálculo da base, só precisamos de ter em conta o raio e número de slices (camadas verticais), para obtermos o valor de $\Delta\alpha$ através da fórmula $2\pi/slices$, obtendo assim o valor do ângulo pelo qual iremos deslocar o ponto de origem $(0, 0, raio)$.

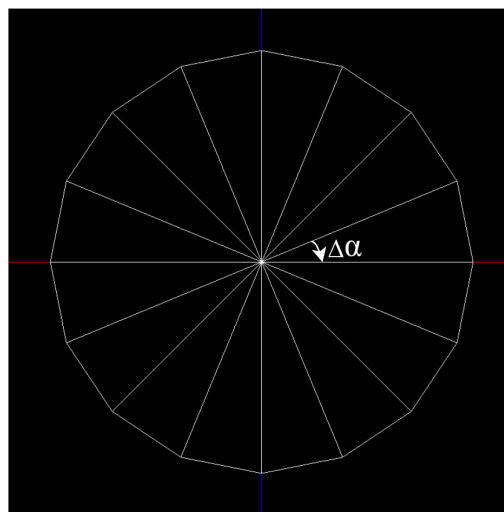


Figura 3.12: Ilustração do $\Delta\alpha$ obtido relativamente às slices do cone

Após termos calculado os pontos necessários para a base do cone, partimos para o cálculo das faces laterais. Para isso, precisamos de descobrir as coordenadas dos vértices das stacks do cone. Para as coordenadas X e Z , dividimos as coordenadas X e Z dos pontos base pelo número de stacks obtendo um valor ao qual chamaremos Δx e Δz , respectivamente.

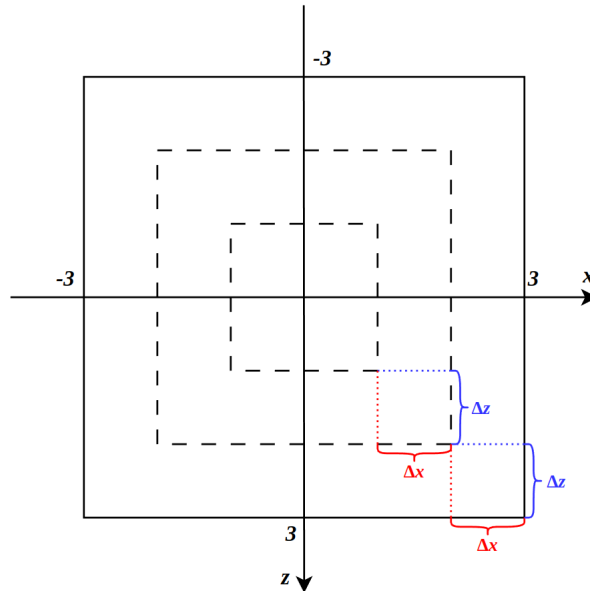


Figura 3.13: Exemplo de obtenção dos deltas com um cone de 4 slices e 3 stacks

Estes deltas são os valores pelos quais temos de decrementar os valores das coordenadas X e Z dos pontos da base, para obter as coordenadas dos pontos da base da próxima stack. Para descobrir o valor da coordenada Y , apenas temos de dividir a altura pelo número de stacks obtendo um valor Δy e para cada stack calculada, o valor de Y será obtido através deste valor.

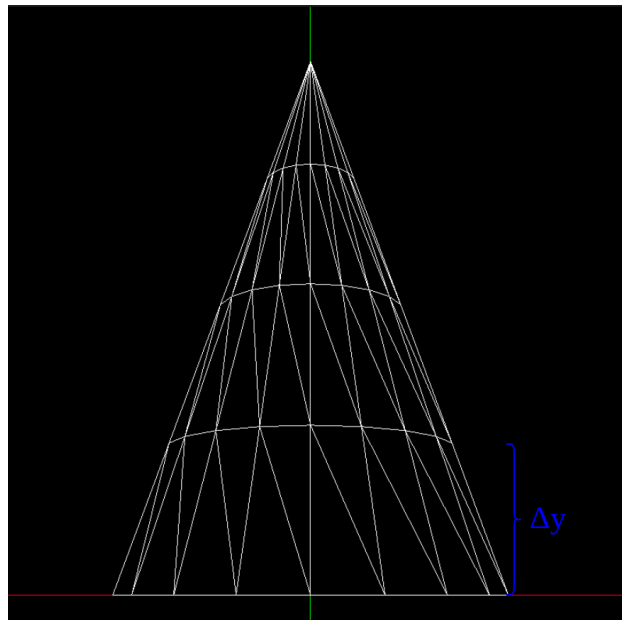


Figura 3.14: Exemplo de um Δy para um cone com 4 stacks

Utilizando os métodos explicados anteriormente, a construção do cone é feita slice a slice, em que pegamos em dois pontos consecutivos da base e aplicamos os cálculos para descobrir as coordenadas dos pontos da camada superior a esses dois pontos, e repete-se o processo até à última camada do cone. Após calcular todas as camadas de uma slice, repete-se este processo para as restantes slices do cone.

Na figura seguinte, apresentamos um exemplo de um cone que é desenhado através do ficheiro .3d gerado:

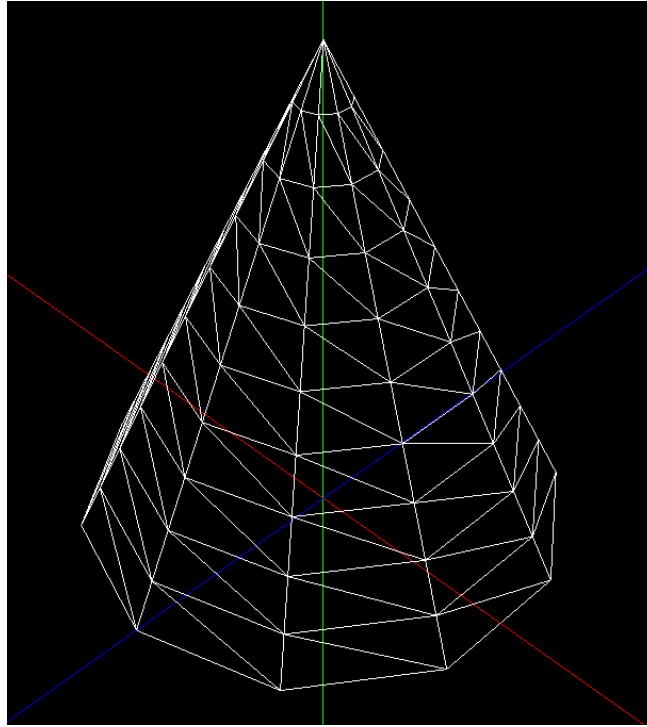


Figura 3.15: Exemplo de um cone com $raio = 1$, $altura = 2$, $slices = 10$ e $stacks = 10$

4 Engine

O Engine é uma aplicação que lê a informação no ficheiro de configuração e a partir dos ficheiros .3d indicados no ficheiro de configuração, desenha os triângulos que originam as primitivas que pretendemos.

A leitura dos ficheiros XML é feita no módulo config.cpp, em que convertemos os dados obtidos dos ficheiros XML para uma estrutura de dados, designada Config.

A estrutura Config contém as seguintes variáveis:

- **poscam:** um array de 3 elementos, do tipo float, em que os 3 elementos são respetivamente a coordenada x , y e z da posição da câmara.
- **lookAt:** um array de 3 elementos, do tipo float, em que os 3 elementos são respetivamente a coordenada x , y e z da posição lookAt da câmara.
- **up:** um array de 3 elementos do tipo float, em que os 3 elementos são respetivamente as coordenadas x , y e z do vetor “up” da câmara.
- **projection:** um array de 3 elementos do tipo float, em que os 3 elementos são respetivamente o parâmetro fov, near e far.
- **models:** uma variável do tipo List, que consiste numa lista com os nomes dos ficheiros dos modelos.

De forma a sermos capazes de manipular a vista da câmara, criámos as seguintes variáveis globais:

- **alpha** = $\cos^{-1}\left(\frac{z}{\sqrt{x^2+z^2}}\right)$
- **beta** = $\sin^{-1}\left(\frac{y}{\sqrt{x^2+y^2+z^2}}\right)$
- **radius** = $\sqrt{x^2 + y^2 + z^2}$

Nota: x , y e z são as coordenadas da posição da câmara.

A manipulação destas variáveis permite que se altere a vista da câmara e seja possível uma melhor interação com o sistema.

5 Demonstração

5.1 Manual de utilização

De forma a tornarmos o sistema mais interativo, incluímos as seguintes possibilidade de interação com o teclado:

- ↑ Zoom in
- ↓ 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** preenche a primitiva
- L** exhibe as linhas que definem a primitiva
- P** exhibe os pontos que constituem a primitiva

5.2 Execução do programa

Para compilarmos o projeto, usamos um ficheiro CMakeLists, que permite compilar o projeto através do cmake. Após a compilação, obtemos dois executáveis, o generator.exe e o engine.exe. Para executar o generator.exe, é necessário dar como argumento o nome de uma primitiva, com os seus respectivos argumentos necessários e no final, a diretoria para o qual vai ser gerado o ficheiro .3d resultante.

```
./generator.exe plane 1 3 ../Fase_1/outputs/plane.3d
```

Para o engine.exe, apenas é necessário fornecer o ficheiro XML com a informação relativa à câmara e os ficheiros gerados pelo *generator* que pretende carregar.

```
./engine.exe ../Fase_1/configs/test_1_4.xml
```

6 Conclusão

Esta primeira fase do trabalho permitiu que fôssemos capazes de consolidar os conhecimentos adquiridos nas aulas teóricas e práticas ao longo das últimas semanas, assim como nos possibilitou a aquisição de novos conhecimentos.

Acreditamos que fomos capazes de atingir todos os objetivos pretendidos para esta fase, isto é, a correta implementação do Generator e do Engine. Para além disso, introduziu-se alguns extras, mais concretamente, a possibilidade de interação com o sistema, permitindo a movimentação da câmara, bem como diferentes opções de visualização das figuras.

Por fim, consideramos que reunimos, ao longo desta fase, os elementos necessários para que consigamos progredir para a próxima fase do projeto.