Índice

[1. Introdução 4](#_Toc508650364)

[2. Figuras 5](#_Toc508650365)

[2.1. Plano 5](#_Toc508650366)

[2.2. Caixa 6](#_Toc508650367)

[2.3. Cone 8](#_Toc508650368)

[2.4. Esfera 10](#_Toc508650369)

[3. Estruturas 13](#_Toc508650370)

[3.1. Ponto 13](#_Toc508650371)

[3.2. Figura 13](#_Toc508650372)

[4. Aplicação 13](#_Toc508650373)

[4.1. Gerador 13](#_Toc508650374)

[4.2. Motor 13](#_Toc508650375)

[5. Conclusão 15](#_Toc508650376)

Índice de Figuras

[Figura 1 - Orientação do Plano 5](file:///C:\Users\jorge\Downloads\relatorio.docx#_Toc508646535)

[Figura 2 - Orientação do Cubo 6](file:///C:\Users\jorge\Downloads\relatorio.docx#_Toc508646536)

[Figura 3 – Face do Cone na Horizontal 8](file:///C:\Users\jorge\Downloads\relatorio.docx#_Toc508646537)

[Figura 4 - Corte Vertical Face Cone 9](file:///C:\Users\jorge\Downloads\relatorio.docx#_Toc508646538)

[Figura 5 - Ilustração do Processo de Construção do Cone 10](file:///C:\Users\jorge\Downloads\relatorio.docx#_Toc508646539)

[Figura 6 - Orientação da Esfera 11](file:///C:\Users\jorge\Downloads\relatorio.docx#_Toc508646540)

[Figura 7 - Pontos 12](file:///C:\Users\jorge\Downloads\relatorio.docx#_Toc508646541)

# Introdução

No presente relatório apresentamos a explicação da primeira fase de um projeto que tem como principal objetivo a conceção de um sistema solar, apresentado como um modelo 3D. Esta fase contempla a base do projeto, as respetivas formas geométricas. Na realização do projeto utilizamos a ferramenta GLUT para a apresentação dos diversos componentes. Nesta fase implementamos um programa que utiliza algoritmos para gerar cada uma das seguintes figuras: plano, cubo, cone e esfera.

Para além deste, implementamos também um programa que lê os ficheiros com os pontos relativos às figuras, previamente criados, e desenha essas figuras, através dos pontos lidos, com o auxílio da ferramenta mencionada acima.

# Figuras

## Plano

Para gerar o plano necessitamos de uma medida para o lado do quadrado (**N**). Por forma a centrar o plano na origem decidimos dividir a medida do lado por 2 e ficámos com os quatro pontos seguintes:

P1 = (-N/2, 0, -N/2)

P2 = (-N/2, 0, N/2)

P3 = (N/2, 0 , N/2)

P4 = (N/2, 0, -N/2)

Para gerar os 2 triângulos (que formam o plano) usamos a regra da mão direita para o orientar para cima (no eixo dos yy) por forma a que este plano seja visto logo que é gerado. Para isso, geramos os seguintes triângulos (a orientação pode ser vista na figura 1):

P4 -> P1 -> P2

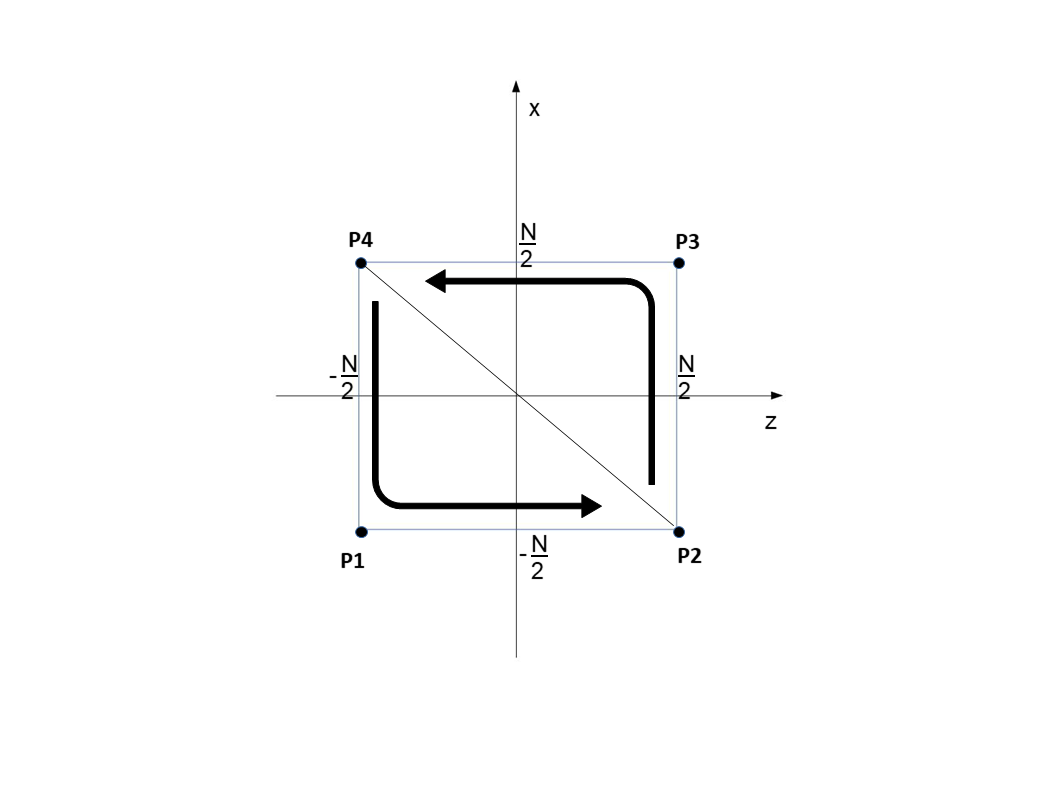
P2 -> P3 -> P4

Figura 1 - Orientação do Plano

## Caixa

Para gerar a caixa foram necessários uma medida para cada um dos comprimentos dos seus lados: X (*sideX*), Y (*sideY*) e Z (*sideZ*), e o número de divisões em que cada lado seria dividido (*divisions*). Para que esta ficasse centrado na origem, dividimos as medidas dos seus lados por 2. O valor das variáveis ficaria então entre um mínimo (min) e um máximo (max) tal que:

Variáveis em X:

*maxX* = *sideX*/2;

*minX* = -*maxX* = -*sideX*/2.

Variáveis em Y:

*maxY* = *sideY*/2;

*minY* = -*maxY* = -*sideY*/2.

Variáveis em Z:

*maxZ* = *sideZ*/2;

*minZ* = -*maxZ* = -*sideZ*/2.

Para calcular os pontos intermédios precisamos de mais um valor por cada eixo, isto é, a medida do lado de cada divisão:

*deltaX* **=** *sideX***/***divisions*;

*deltaY* **=** *sideY***/***divisions*;

*deltaZ* **=** *sideZ***/***divisions*.

É importante referir que cada face da caixa é um caso especial. Todos os pontos da mesma face apresentam uma coordenada com valor igual e duas com valor variável. Ao mudar de face, ou o valor constante dessa face muda (caso de mudar para uma face paralela à atual) ou a coordenada com valor igual em todos os pontos da nova face muda (caso de mudar para uma face perpendicular à atual). Por exemplo, mudar de uma face no plano superior y=3 para uma face no plano inferior (paralelo) y=-3 implica apenas inverter o valor constante da ordenada. Mudar de uma face no plano y=3 para uma face no plano x=2 implica passar a ter a abcissa dos pontos dessa face como constante e a ordenada (anteriormente constante) como variável. Por este motivo, cada face tem de ser desenhada de forma diferente.

Para exemplificar o processo de desenho de uma face escolhemos a face apresentada na figura, isto é, a face com o Z constante e máximo. Nesta face apenas as variáveis correspondentes às abcissas e ordenadas variam.

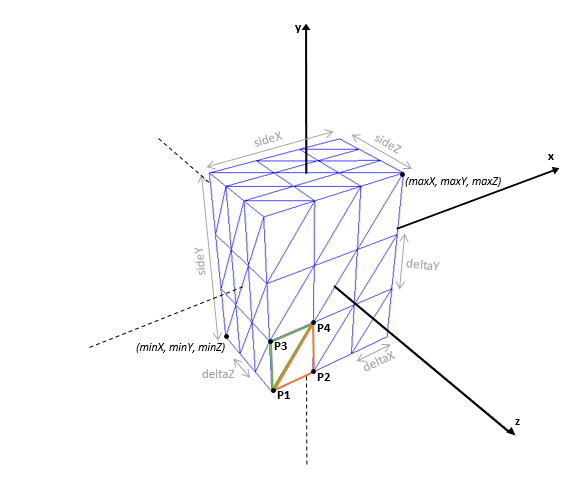


Figura 2 - Orientação do Cubo

Começamos por desenhar os triângulos da linha inferior, da esquerda para a direita, ou seja, ao longo do eixo dos X. Chegando ao fim da linha, passamos para a linha acima e voltamos a percorrer da esquerda para a direita.

É possível obter todos os valores necessários recorrendo a dois ciclos, onde i itera no ciclo externo e j itera no ciclo interno, e às seguintes fórmulas:

Para todo o i e j menor que *divisions*:

min + I \* delta / min + (i+1) \* delta

max + I \* delta / max + (i+1) \* delta

min + j \* delta / min + (j+1) \* delta

max + j \* delta / max + (j+1) \* delta

Para que todos os triângulos ficassem orientados para o exterior usamos a regra da mão direita. Observando a figura acima é possível ordenar os pontos indicados do seguinte modo:

Triângulo vermelho:

P1 = (minX, minY, maxZ)

P2 = (minX + deltaX, minY, maxZ)

P3 = (minX + deltaX, minY + deltaY, maxZ)

Triângulo verde:

P1 = (minX, minY, maxZ)

P3 = (minX + deltaX, minY + deltaY, maxZ)

P4 = (minX, minY + deltaY, maxZ)

Para esta face, a variável j é usada para obter os valores das abcissas ao logo do eixo dos X e a variável i é usada para obter os valores das ordenadas ao longo do eixo dos Y, com os valores das cotas constantes e máximos. Como caso geral desta face temos:

(minX + j\*deltaX, minY + i\*deltaY, maxZ)

(minX + (j+1)\*deltaX, minY + i\*deltaY, maxZ)

(minX + (j+1)\*deltaX, minY + (i+1)\*deltaY, maxZ)

(minX + j\*deltaX, minY + i\*deltaY, maxZ)

(minX + (j+1)\*deltaX, minY + (i+1)\*deltaY, maxZ)

(minX + j\*deltaX, minY + (i+1)\*deltaY, maxZ)

Para desenhar outras faces seria só inserir corretamente as fórmulas indicadas acima nas coordenadas correspondentes tendo em conta o valor mínimo dessas coordenadas, o delta que se pretende avançar a cada iteração em cada uma delas e como se pretende iterar a face em i e j.

## Cone

Para desenharmos a figura geométrica *cone* precisamos de quatro parâmetros iniciais: raio, altura, fatias e camadas. O raio define qual será a área da base do cone, a altura define o comprimento vertical, as fatias indicam o número de triângulos que compõem a base e as camadas definem em quantas seções cada face do cone será dividida.

Sucintamente o método utilizado passa pelo seguinte: desenhar em primeiro lugar o triangulo que compõe uma parte da base, desenhar os triângulos que representam a face lateral do cone (correspondente a esse mesmo triangulo) consoante o número de camadas pretendidas. Este processo é repetido n vezes, sendo que n é determinado pelo número de fatias passado como parâmetro. Um detalhe importante é que todos os triângulos foram desenhados utilizando a *regra da mão direita*, com a orientação sempre para fora da figura geométrica.

Os pontos da base eram calculados utilizando coordenadas polares:

xA = raio \* sin(i\*angulo);

zA = raio \* cos(i\*angulo);

xP = raio \* sin((i+1)\*angulo);

zP = raio \* cos((i+1)\*angulo);

As variáveis terminadas em A referem-se aos pontos antes da rotação e as variáveis terminadas em Z referem-se aos pontos depois da rotação (pontos que foram a base do triangulo).

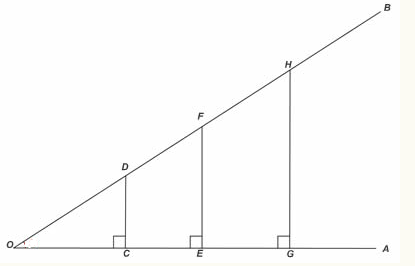
A parte mais morosa do processo passa pela construção das camadas em cada face. De maneira a que fosse possível calcular as coordenadas dos pontos necessárias para o desenho dos 2 triângulos que compõem cada camada foi necessário utilizar semelhança de triângulos.

Figura 3 – Face do Cone na Horizontal

Supondo que queríamos calcular o primeiro raio (GH) de uma divisão em 3 camadas temos de fazer os seguintes cálculos:

**GH = (AB \* OG) / OA**

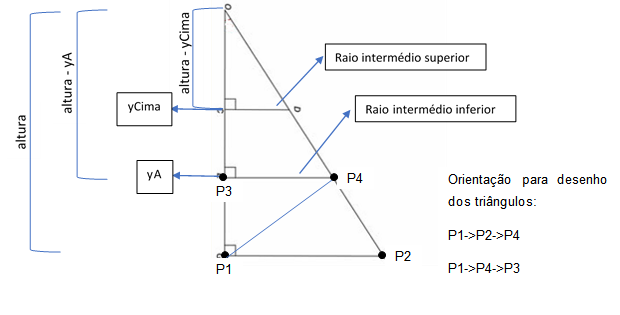
O processo é semelhante tanto para o calculo dos raios intermédios inferiores como para os raios intermédios superiores.

Figura 4 - Corte Vertical Face Cone

float raioIntermedioCima = raio \* ((altura-yCima) /altura);

float raioIntermedioBaixo = raio \* ((altura-yA) /altura);

Com a informação do raio intermédio e como utilizamos coordenadas polar, repetiremos o processo apresentado em cima para calcular os pontos que nos permitem desenhar as camadas.

xA = raioIntermedioBaixo \* sin(i\*angulo);

zA = raioIntermedioBaixo \* cos(i\*angulo);

xP = raioIntermedioBaixo \* sin((i+1)\*angulo);

zP = raioIntermedioBaixo \* cos((i+1)\*angulo);

float xACima = raioIntermedioCima \* sin(i\*angulo);

float zACima = raioIntermedioCima \* cos(i\*angulo);

float xPCima = raioIntermedioCima \* sin((i+1)\*angulo);

float zPCima = raioIntermedioCima \* cos((i+1)\*angulo);

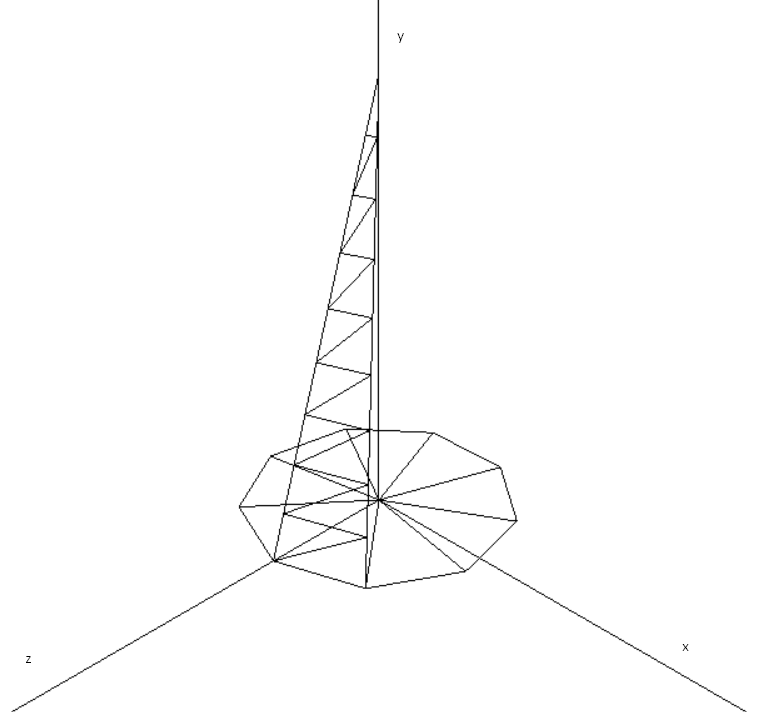
De seguida apresentamos uma imagem que representa o processo descrito anteriormente.

Figura 5 - Ilustração do Processo de Construção do Cone

## Esfera

Para desenhar a figura geométrica "esfera" precisamos de um raio, que poderia tomar qualquer valor real positivo, o número de fatias e o número de camadas. Para além disto, desenhar a esfera implicava que trabalhássemos com coordenadas esféricas, de modo que, foi necessário preceder à transformação destas coordenas para cartesianas. Aplicando o conhecimento adquirido de trigonometria convertemos através das seguintes formas:

y = raio \* cos(beta)

x = raio \* sin(beta) \* sin(alpha)

z = raio \* sin(beta) \* cos(alpha)

Sendo que beta está no intervalo 0 < beta < PI e alpha no intervalo 0 < alpha < 2\*PI. Isto implica que por cada fatia que é iterada é desenhada uma parte da esfera (em relação à fatia a esfera é totalmente desenhada no eixo dos yy), ao fim de iterarmos por todas as fatias temos então desenhada a esfera completa. Ou seja, ângulo alpha é referente ao deslocamento na vertical, enquanto que o ângulo beta é o deslocamento na horizontal.

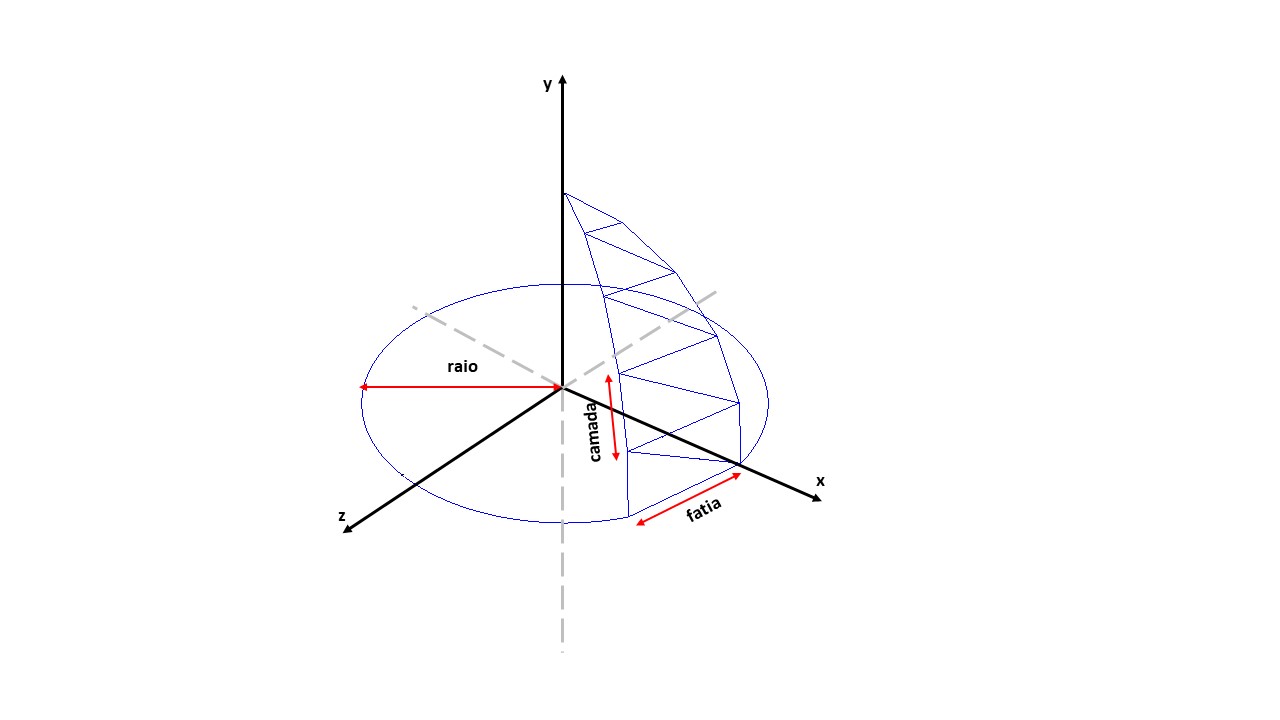


Figura 6 - Orientação da Esfera

A estratégia adotada pressupunha que para cada fatia eram desenhadas as N camadas, recebidas como parâmetro. Por cada camada eram desenhados 2 triângulos seguindo a regra da mão direita, para que ficasse orientado para o exterior. A iteração foi realizada atrvés de 2 ciclos for, um para a deslocação horizontal e outro para a vertical. O incremento dos ângulos (o que verdadeiramente nos permitia mover) era feito através da fórmula i\*alpha e j\*beta, sendo que tanto i como j estavam delimitados, respetivamente, ao número de fatias e de camadas. Ao aumentar o valor de alpha e beta seguindo uma operação de multiplicação, permitiu diminuir a percentagem de erro que poderia aparecer nas operações de virgula flutuante quando se usam somas.

Sendo assim os pontos seriam:

yCima = raio \* cos(j\*beta)

xAtualCima = raio \* sin(j\*beta)\* cos(i\*alpha)

zAtualCima = raio \* sin(j\*beta)\* sin(i\*alpha)

xProxCima = raio \* sin(j\*beta)\* cos((i+1)\* alpha)

zProxCima = raio \* sin(j\*beta)\* sin((i+1) \* alpha)

yBaixo = raio \* cos((j+1) \* beta)

xAtualCima = raio \* sin((j+1) \* beta)\* cos(i\*alpha)

zAtualCima = raio \* sin((j+1) \* beta)\* sin(i\*alpha)

xProxCima = raio \* sin((j+1) \* beta)\* cos((i+1)\* alpha)

zProxCima = raio \* sin((j+1) \* beta)\* sin((i+1) \* alpha)

P1 = (xAtualCima, yCima, zAtualCima)

P2 = (xProxCima, yCima, zProxCima)

P3 = (xAtualBaixo, yBaixo, zAtualBaixo)

P4 = (xProxBaixo, yBaixo, zProxBaixo)

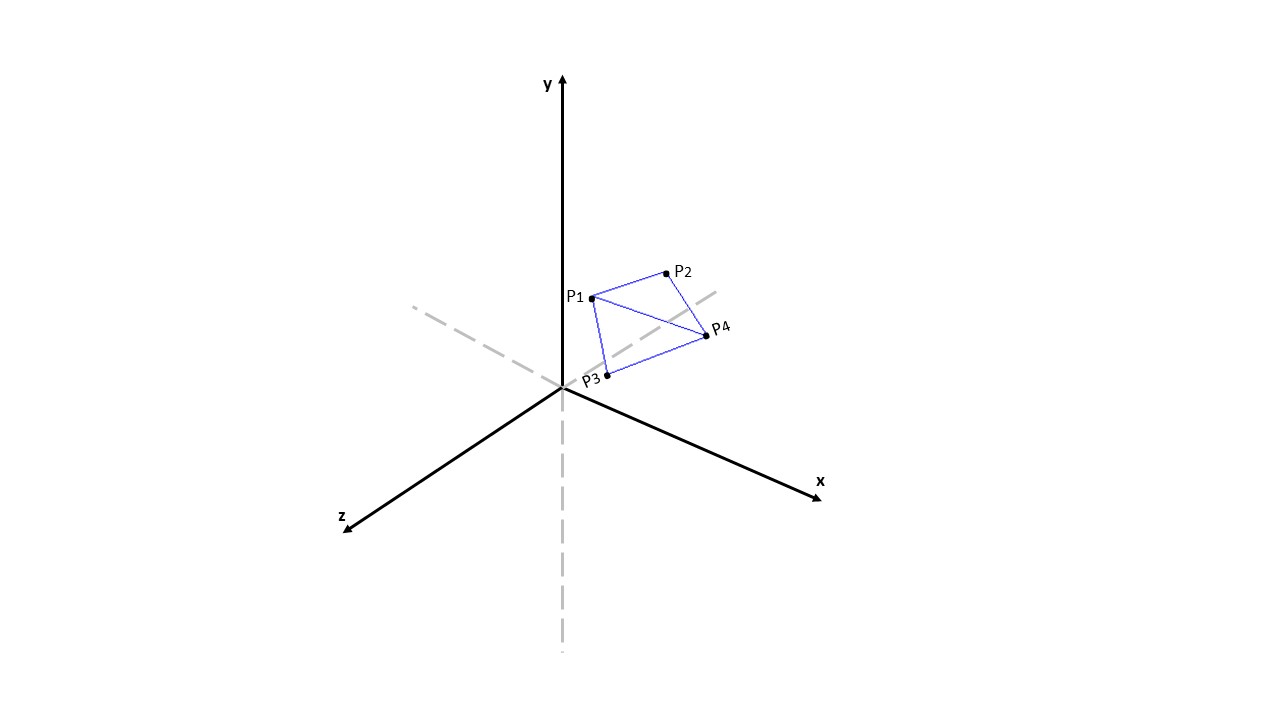


Figura 7 - Pontos

A orientação dos pontos, para formar o triângulo, seguiu então a seguinte ordem: P4 -> P2 -> P1 e P4 -> P1 -> P3.

# Estruturas

Para a realização desta fase do projeto decidimos criar 2 estruturas com o objetivo de organizar o código e tornar mais fácil a sua implementação.

## Ponto

A estrutura Ponto corresponde a um vértice, pelo que contém 3 *floats* que correspondem às coordenadas x,y e z da um ponto.

## Figura

A estrutura *Figura* corresponde a uma das figuras que geramos, ou seja, esta estrutura é constituída por uma lista de pontos, na qual a ordem é importante visto ser esta que permite desenhar os triângulos que irão gerar a forma geométrica.

# Aplicação

A realização desta fase do projeto consiste na implementação de duas aplicações: gerador e motor.

## Gerador

A aplicação *gerador* contém todas as primitivas de como será realizado cada um dos modelos referidos anteriormente. Esta aplicação gera uma lista de pontos e escreve-os para um ficheiro. Esse ficheiro contém na primeira linha o número de pontos, e nas restantes linhas contém as coordenadas x,y e z de cada ponto separadas por um espaço.

## Motor

A aplicação *motor* tem como funcionalidade ler um ficheiro *XML*, que contém os ficheiros com as figuras a serem representadas, e apresentar essas figuras com o auxílio da ferramenta *GLUT*.

Para auxiliar no *parsing* do ficheiro *XML* usamos a biblioteca *TinyXML2*, pelo que necessitamos de conter dois ficheiros: **tiny*xml*2.cpp** e **tinyxml2.h**.

Para ler os ficheiros que contêm as figuras (previamente criados com o auxílio do gerador) apenas é necessário ler a primeira linha para saber o número de pontos, e ir iterando linha a linha, até atingir o limite dos pontos existentes, guardando cada ponto numa lista (guardamos um *vector* de apontadores para objetos da classe Ponto). No final criamos um apontador para uma estrutura *Figura*, contendo o *vector* de apontadores para Ponto, e retornamos esse apontador.

Cada figura lida dos ficheiros vai sendo guardada numa lista de figuras (mais uma vez com o auxílio de *vector*). Posteriormente esta lista é iterada e para cada figura desenhamos os pontos, pela ordem que aparecem na lista de pontos da figura (por forma a desenhar corretamente o modelo), tendo no final apresentado todas as figuras referidas pelo ficheiro *XML*.

# Conclusão

O trabalho foi realizado com sucesso, já que conseguimos desenhar todas as figuras pretendidas, fica apenas a percepção de que poderíamos ter realizado algumas figuras para além do pedido, pois irão ser necessárias para as próximas fases. Essas figuras seriam: torus e elipse.

As partes onde sentimos mais dificuldades foram em relação ao *parsing* de *XML*, visto que não tínhamos grande conhecimento sobre as estruturas de *parsing* utilizadas para este tipo de linguagem.

Na próxima fase vamos necessitar de redefinir a forma de *parsing*, visto que termos de tratar de mais nodos nos ficheiros *XML*, sendo que também teremos de acrescentar algumas estruturas ao código, por forma a organizar os diferentes modelos que terão diferentes transformações. Nas fases seguintes vamos ainda acrescentar mais estruturas ao código, visto que vamos necessitar de acrescentar texturas e ter mais cuidado com o desempenho, mas falaremos disso mais em detalhe nas próximas fases.