Índice

[1. Introdução 4](#_Toc508541950)

[2. Figuras 5](#_Toc508541951)

[2.1. Plano 5](#_Toc508541952)

[2.2. Cubo 6](#_Toc508541953)

[2.3. Cone 8](#_Toc508541954)

[2.4. Esfera 9](#_Toc508541955)

[3. Estruturas 12](#_Toc508541956)

[3.1. Ponto 12](#_Toc508541957)

[3.2. Figura 12](#_Toc508541958)

[4. Aplicação 12](#_Toc508541959)

[4.1. Gerador 12](#_Toc508541960)

[4.2. Motor 12](#_Toc508541961)

Índice de Figuras

[Figura 1 - Orientação do Plano 5](file:///C:\Users\jorge\Desktop\UM\3_ANO\2-SEMESTRE\CG\relatorio.docx#_Toc508541968)

[Figura 2 - Orientação do Cubo 6](file:///C:\Users\jorge\Desktop\UM\3_ANO\2-SEMESTRE\CG\relatorio.docx#_Toc508541969)

[Figura 3 - Colocar Legenda 8](file:///C:\Users\jorge\Desktop\UM\3_ANO\2-SEMESTRE\CG\relatorio.docx#_Toc508541970)

[Figura 4 - Colocar Outra Legenda 9](file:///C:\Users\jorge\Desktop\UM\3_ANO\2-SEMESTRE\CG\relatorio.docx#_Toc508541971)

[Figura 5 - Orientação da Esfera 10](file:///C:\Users\jorge\Desktop\UM\3_ANO\2-SEMESTRE\CG\relatorio.docx#_Toc508541972)

[Figura 6 - Pontos 11](file:///C:\Users\jorge\Desktop\UM\3_ANO\2-SEMESTRE\CG\relatorio.docx#_Toc508541973)

# 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

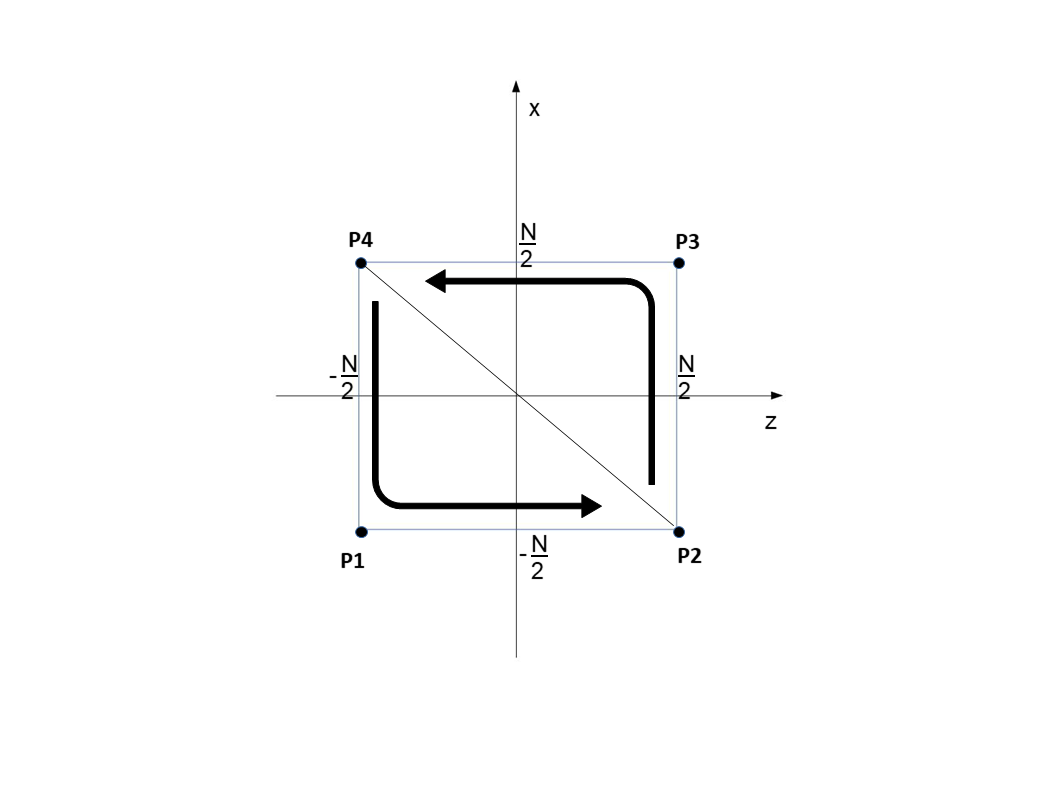
P2 -> P3 -> P4

Figura 1 - Orientação do Plano

## Cubo

Para gerar o cubo foram necessários uma medida para o comprimento dos seus lados (**side**) e o número de divisões em que cada lado seria dividido (**divisions**). Para que este ficasse centrado na origem, dividimos a medida do lado recebida por 2. O valor das variáveis ficaria então entre um mínimo (min) e um máximo (max) tal que:

**max** = **side**/2;

**min** = -**max** = -**side**/2.

Para calcular os pontos precisamos de mais um valor, a medida do lado de cada divisão:

**divisionSide = side/divisions**.

É importante referir que cada face do cubo é um caso especial. Todos os pontos da mesma face apresentam uma coordenada constante, contudo cada face apresenta um valor constante numa coordenada diferente da coordenada constante noutra face ou um valor constante diferente na mesma coordenada constante de outra face. Por exemplo uma face tem o x igual a min, outra face tem o x igual a max, outra tem o y igual ao max, etc. 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.

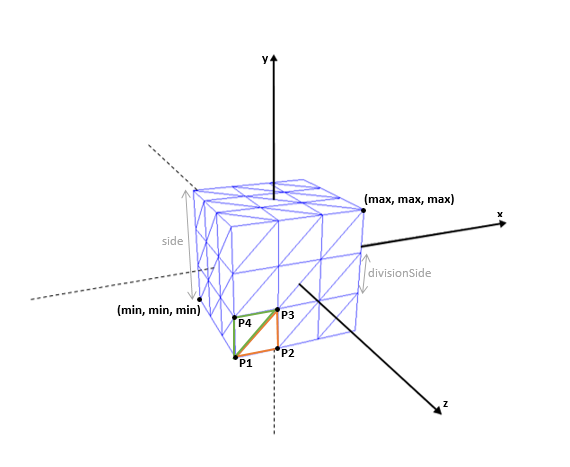
Começamos por desenhar os triângulos da linha inferior, da esquerda para a direita, ou seja, ao longo do eixo dos X. Chegando ao fim da linha, passamos para a linha acima e voltamos a percorrer da esquerda para a direita.

Figura 2 - Orientação do Cubo

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 ou igual a divisions:

**i1** = (min-1) + i\*divisionSide;

**i2** = min + i\*divisionSide;

**j1** = (min-1) + j\*divisionSide;

**j2** = min + j\*divisionSide;

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 do seguinte modo:

Triângulo vermelho:

**P1** = (j1, i1, max)

**P2** = (j2, i1, max)

**P3** = (j2, i2, max)

Triângulo verde:

**P1** = (j1, i1, max)

**P3** = (j2, i2, max)

**P4** = (j1, i2, max)

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. Para desenhar outras faces, basta trocar a ordem pela qual as variáveis são passadas aos vértices a desenhar e indicar se o valor constante nessa face é max ou min, tendo em conta a face que se pretende desenhar e ordenar os vértices tendo em cota a orientação que se pretende dar aos triângulos da mesma.

Por exemplo, para desenhar a face com a abcissa constante e mínima, iterando a variável Z no ciclo interior e a variável Y no ciclo exterior bastava fazer:

**P1** = (min, i1, j1)

**P2** = (min, i1, j2)

**P3** = (min, i2, j2)

**P1** = (min, i1, j1)

**P3** = (min, i2, j2)

**P4** = (min, i2, j1)

Nota: P1, P2, P3 e P4 são apenas pontos simbólicos, isto é, não correspondem a nenhum ponto em concreto. Servem apenas para mostrar a ordem pela qual os vértices são passados de modo a que os triângulos fiquem voltados para o exterior.

## 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).

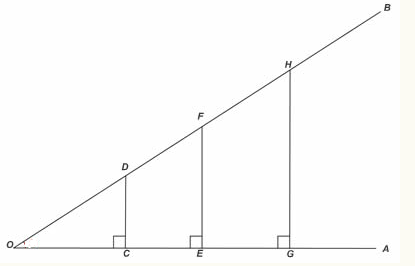
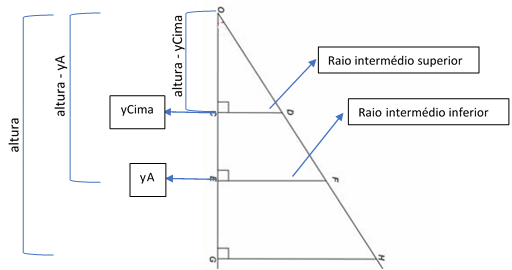
A parte mais morosa do processo passa pela construção das camadas em cada face. De maneira a que fosse possível calcular as coordenadas dos pontos necessárias para o desenho dos 2 triângulos que compõem cada camada foi necessário utilizar semelhança de triângulos.

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.



Orientação para desenho dos triângulos:

P1->P2->P4

P1->P4->P3

P4

P3

P1

P2

Figura 4 – Figura 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.

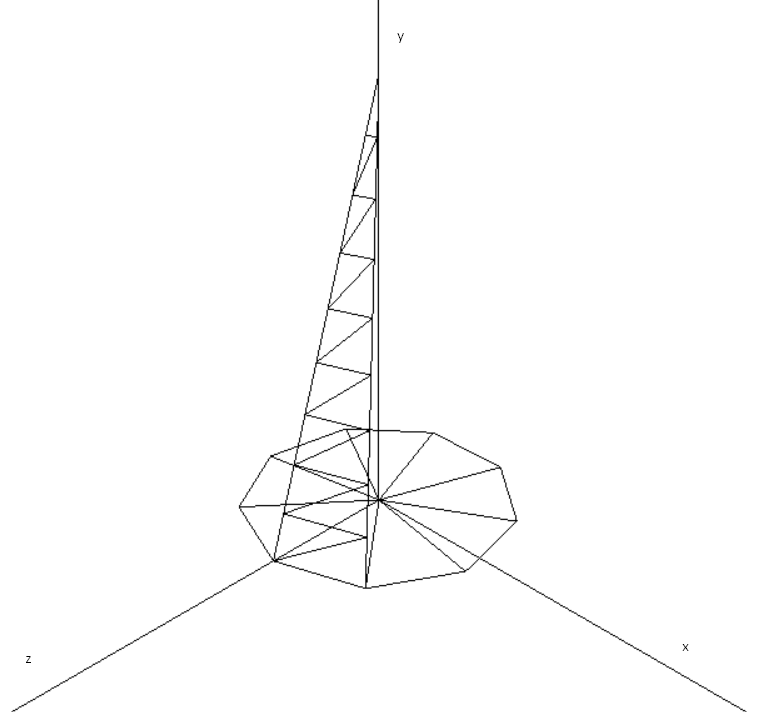


Figura 6 – Figura ilustrativa do processo de construção do cone

## 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)

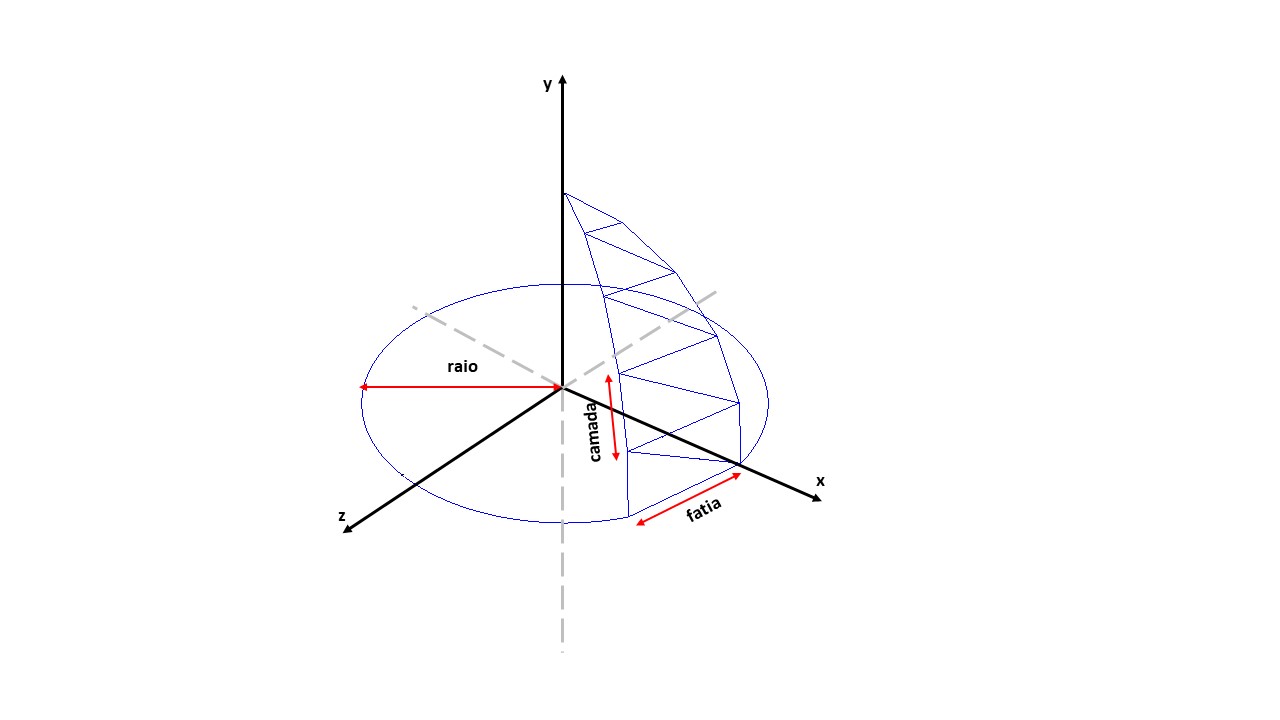
Sendo que beta está no intervalo 0 < beta < PI e alpha no intervalo 0 < alpha < 2\*PI. Isto implica que por cada fatia que é iterada é desenhada uma parte da esfera (em relação à fatia a esfera é totalmente desenhada no eixo dos yy), ao fim de iterarmos por todas as fatias temos então desenhada a esfera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

Figura 5 - 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)

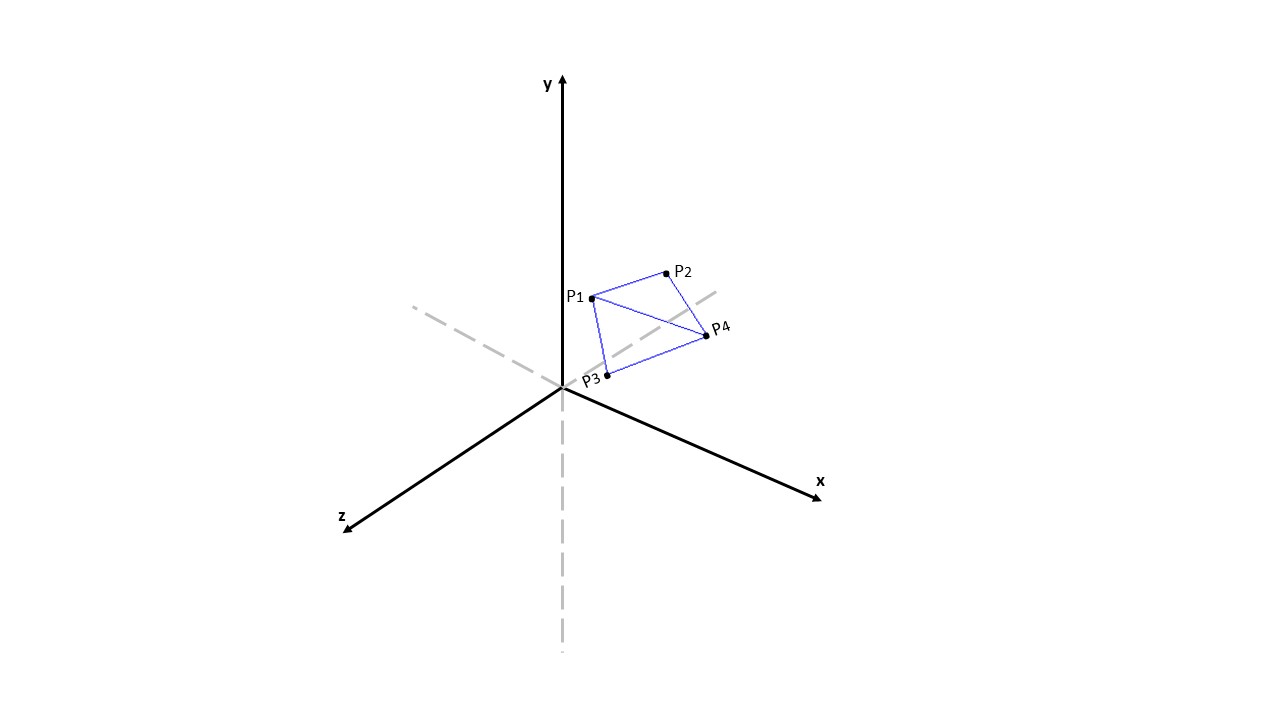


Figura 6 - Pontos

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, fica apenas a percepção de que poderíamos ter realizado algumas figuras a mais, pois irão ser necessárias para as próximas fases. Essas figuras passarão por: torus e elipse.

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

Para além disso, pensamos que o projeto foi bem desenvolvido e o resultado é bastante satisfatório. Na próxima fase vamos necessitar de redefinir a forma de *parsing*, visto 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 detalhado nas próximas fases.