



Universidade do Minho  
Escola de Engenharia

Mestrado Integrado em Engenharia Informática

**Computação Gráfica**

## **Relatório do Trabalho Prático 1**

A78890 Alexandre Costa  
A75248 Ana Sofia Gomes Marques  
A65277 Flávio Manuel Machado Martins  
A79799 Gonçalo Costeira

Grupo 50

6 Março 2020

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>i</b>
<b>2</b>	<b>Generator</b>	<b>ii</b>
2.1	Plane . . . . .	iii
2.2	Box . . . . .	iv
2.3	Sphere . . . . .	vi
2.4	Cone . . . . .	vii
<b>3</b>	<b>Engine</b>	<b>ix</b>
<b>4</b>	<b>Conclusão</b>	<b>x</b>
<b>5</b>	<b>Referências</b>	<b>xi</b>

# 1 Introdução

Este relatório visa apresentar as decisões tomadas na realização da primeira fase do trabalho prático da Unidade Curricular de Computação Gráfica. Procuramos assim justificar todas as considerações feitas na formulação do problema, na elaboração dos algoritmos de geração de vértices para os determinados modelos e o processo de construção e desenho das figuras através do input de um ficheiro previamente criado.

Toda a modulação do problema foi feita com recurso à linguagem C++, abordada nas aulas práticas da UC. O engine em particular faz uso de biblioteca TinyXML2 para que a leitura dos documentos que lhe são dados com input seja feita de forma simples e consistente. E por fim, utilizamos a API fornecida pelo OpenGL para visualização dos nossos modelos.

Numa primeira fase será descrito o processo de geração do ficheiro de vértices e posteriormente explicado o modo de desenvolvimento do dito engine que irá desenhar as figuras.

Para esta primeira fase, foram implementados os algoritmos para gerar os vértices de um:

- Plane

Quadrado em XZ centrado na origem

- Box

Modelo tridimensional com largura, comprimento, altura, número de fatias e stacks

- Sphere

Esfera centrada na origem, com raio, número de fatias e stacks

- Cone

Modelo de um cone, identificado pelo raio da base, altura, número de fatias e stacks.

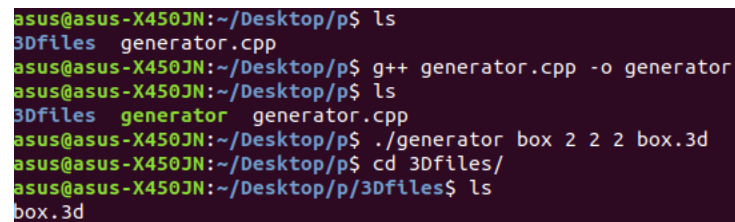
## 2 Generator

O Generator ou gerador, é um pequeno programa em C++, que depois de compilado, o correspondente executável que foi gerado escreve para um ficheiro os vértices da primitiva gráfica desejada, que serão guardados numa pasta chamada 3Dfiles.

Este programa é caracterizado por conter a função main, que recebe o input desejado para a criação dos modelos, assim como os respetivos algoritmos para a criação destes mesmos, que serão explicados nas seguintes subsecções do relatório.

Este programa aceita como input as seguintes opções:

- plane <length> <width> <filename>
- box <length> <width> <height> <filename>
- sphere <radius> <slices> <stacks> <filename>
- cone <bottom radius> <height> <slices> <stacks> <filename>



```
asus@asus-X450JN:~/Desktop/p$ ls
3Dfiles  generator.cpp
asus@asus-X450JN:~/Desktop/p$ g++ generator.cpp -o generator
asus@asus-X450JN:~/Desktop/p$ ls
3Dfiles  generator  generator.cpp
asus@asus-X450JN:~/Desktop/p$ ./generator box 2 2 2 box.3d
asus@asus-X450JN:~/Desktop/p$ cd 3Dfiles/
asus@asus-X450JN:~/Desktop/p/3Dfiles$ ls
box.3d
```

Figura 1: Compilação do generator e criação dum ficheiro .3d com os vértices de uma box com largura, comprimento e altura 2.

## 2.1 Plane

**Cabeçalho da função:** void plane(float length, float width, char\* filename);

**Parâmetros de entrada:** length: comprimento do plano; Width: largura do plano; char\* filename: nome do ficheiro que será criado com os vértices do plano.

A função que gera as informações dos vértices de um plano, designada por plane, limita-se a determinar os 4 vértices necessários para a criação do plano e posteriormente produzir dois triângulos que o irão formar. De referir que sendo um plano, centrado na origem do referencial XYZ e com a variável Y constante (Y=0), os vértices apresentam simetria face ao eixo dos X.

Deste modo os valores de X variam entre -length/2 e length/2, e os valores de Z entre -width/2 e width/2.

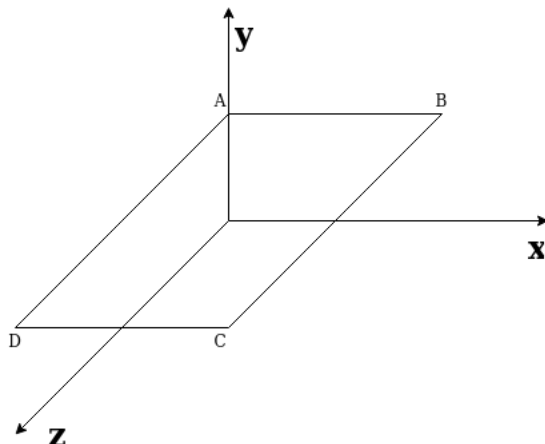


Figura 2: Plano ABCD

Partindo da Figura 2, para desenhar o plano apenas precisamos de observar as coordenadas de A,B,C e D, que são triviais sabendo que o plano tem centro na origem. A partir do descrito até agora, e assumindo:

$$x = \frac{length}{2}$$

$$y = 0$$

$$z = \frac{width}{2}$$

São facilmente descobertas as coordenadas dos pontos:

$$A(-x,y,-z);$$

$$B(x,y,-z);$$

$$C(x,y,z);$$

$$D(-x,y,z);$$

Com estes pontos podemos então desenhar ABC e ACD Gerando o plano a partir do ponto A, e segundo o OpenGL, para a superfície do plano ficar visível utilizamos a regra da mão direita no sentido inverso aos ponteiros do relógio, desenhamos os triângulos ABC e ACD. De modo a que o plano seja visível de ambas as direções foram desenhados também os triângulos ACB e ADC.

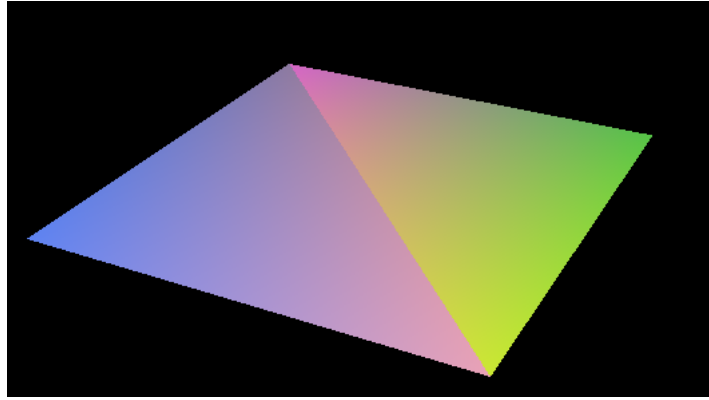


Figura 3: Exemplo de um plano

## 2.2 Box

**Cabeçalho da função:** void box(float length, float width, float heigth, char\* filename)

**Parâmetros de entrada:** float length, width e heigth que representam os valores de largura, comprimento e altura do cubo; char\* filename: nome do ficheiro que será criado com os vértices da caixa.

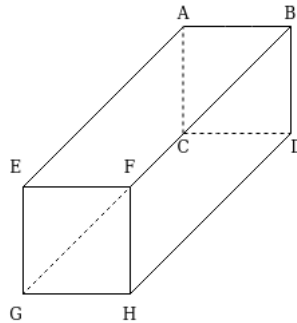


Figura 4: Representação duma box.

Com o auxilio da figura 3 é possível dividir a box em 6 faces, nomeadamente formadas pelos vertices ABDC, EGHF, AEFB, BFHD, ACGE, GCDH.

Da mesma forma que desenhamos o plano pelo utilização de dois triângulos aqui podemos fazer o mesmo, sendo que apenas muda o valor fixo, que em vez de ser  $y=0$  irá ser diferente nas varias faces.

Tal como no plano, os valores de X variam entre  $\frac{-length}{2}$  e  $\frac{length}{2}$ , e os valores de Z entre  $\frac{-width}{2}$  e  $\frac{width}{2}$ , agora, os valores de Y variam entre  $\frac{-height}{2}$  e  $\frac{height}{2}$ .

Com isto em mente podemos chegar a todas as coordenadas dos vertices da box, por exemplo para A teriamos as coordenadas  $(\frac{-length}{2}, \frac{height}{2}, \frac{-width}{2})$ .

Obtendo todos os vertices facilmente desenhamos a box, iniciando pela face ABDC, desenhando os triângulos ABD e DCA, desenhando assim aquela face virada para fora da Box.

Em seguida são desenhadas as restantes faces com dois triângulos apontados para o exterior pela norma da mão direita mencionada nas aulas, igual ao que já foi feito anteriormente, até se formar a box completa. Um zoom excessivo, ou a escolha de dimensões superiores à distância da câmara à origem pode levar ao aparente desaparecimento do modelo.

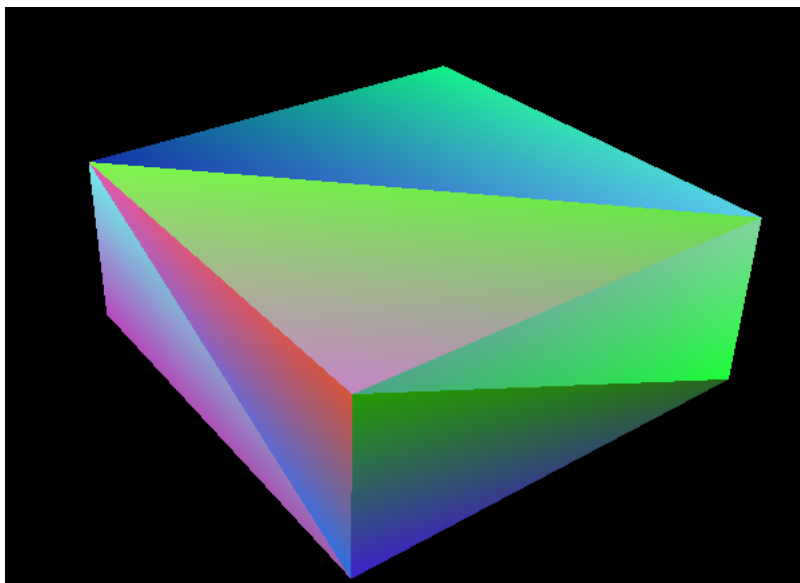


Figura 5: Exemplo de uma box

## 2.3 Sphere

**Cabeçalho da função:** void sphere(float radius, float slices, float stacks, char\* filename);

**Parâmetros de entrada:** float radius: valor do raio da esfera, float slices e stacks, que indica o número de fatias e stacks, sendo as slices o nº de subdivisões em volta do eixo X e as stacks o nº de subdivisões em volta do eixo Y, a aplicar na criação da esfera; char\* filename: nome do ficheiro que será criado com os vértices da esfera.

Para implementar um algoritmo de construção de uma esfera, foi usada a noção de coordenadas esféricas, onde qualquer ponto que pertence a uma determinada superfície esférica, pode ser identificado, considerando o referencial do glut, por três parâmetros: o raio  $r$  da esfera, o ângulo  $\alpha$ , em função ao eixo dos X e do ângulo  $\beta$  em função do eixo dos Y.

Assim, utilizando dois ciclos aninhados, um para iterar o eixo dos Y (stacks), começando com o valor de  $\beta$  em  $-\pi/2$  e terminando em  $\pi/2$ , utilizando  $b = \pi/\text{stacks}$  como intervalo e, dentro deste ciclo, o outro para iterar as fatias, começando com o ângulo  $\alpha$  em 0 e terminando em  $2\pi$ , utilizando  $a = 2\pi/\text{slices}$  como intervalo.

As coordenadas esféricas são dadas pelas seguintes expressões:

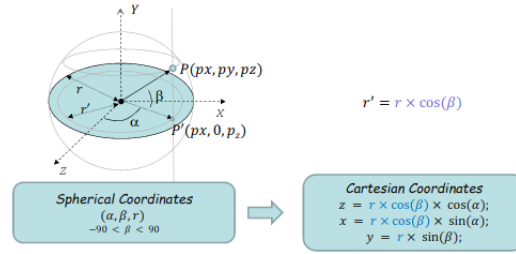


Figura 6: Coordenadas Esféricas

Em cada iteração do ciclo interior iremos ter um rectângulo formado por 4 pontos cujas coordenadas são facilmente descobertas recorrendo às coordenadas cartesianas referidas anteriormente. Assim, tendo em conta o  $r$  fixo apenas varia o valor de  $\beta$  e  $\alpha$  entre os 4 pontos, podemos calcular facilmente A utilizando  $\beta$  e  $\alpha$ , B utilizando  $\beta$  e  $\alpha+a$ , C utilizando  $\beta+b$  e  $\alpha+a$ , e D utilizando  $\beta+b$  e  $\alpha$ . De seguida desenhamos os triângulos ABC e CDA.

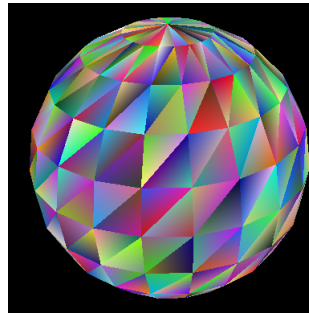


Figura 7: Exemplo de esfera com 20 slices e 10 stacks



## 2.4 Cone

**Cabeçalho da função:** void cone(float radius, float height, float slices, float stacks, char\* filename);

**Parâmetros da função:** float radius,height, que indicam os valores do raio da base e altura do cone; float slices e stacks, que indica o número de fatias e stacks a aplicar na criação do cone; char\* filename: nome do ficheiro que será criado com os vértices do cone.

A base do nosso cone é a primeira parte a ser desenhada e é representada por uma circunferencia formada por triangulos virados para baixo.

O ângulo de cada slice será calculado por  $a = \frac{2\pi}{slices}$

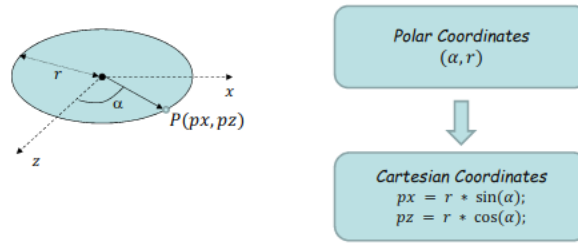


Figura 8: Coordenadas Polares

Para desenhar a base criamos um ciclo for, de 0 até  $2\pi$  e utilizando coordenadas polares, visíveis na figura 5, desenhamos triangulos entre o centro da origem(0,0,0), o ponto calculado pelas coordenadas polares para  $\alpha + a$  e r e por fim o ponto calculado pelas coordenadas polares  $\alpha$  e r. Desta forma o triangulo é desenhado de forma a que a base seja visível de fora do cone.

Para formar o restante cone, apenas foi preciso criar um algoritmo com dois ciclos “for” aninhados. O de fora itera as stacks e o ciclo de dentro serve para iterar o ângulo que varia entre 0 e  $2\pi$  em cada nível da stack. O desenho é semelhante ao feito para a esfera em cada subida de nível, sendo que neste caso os pontos são calculados com coordenadas polares, variando o raio e o  $\alpha$ .

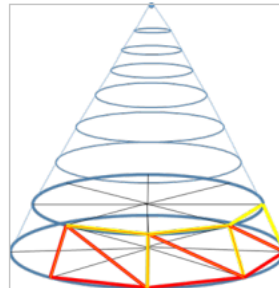


Figura 9: Esboço do algoritmo de criação de um cone. Serão criadas 8 stacks e 8 slices. Primeiro é iterado o anel da primeira stack, compondo os triângulos de cada fatia, e posteriormente iteram-se os seguintes níveis, repetindo o processo.

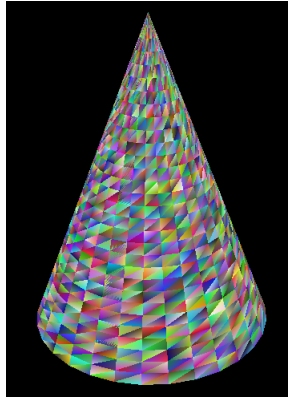


Figura 10: Exemplo de cone com 30 slices e 30 stacks

### 3 Engine

O engine ou motor representa a parte do nosso trabalho que faz a ligação entre todas as outras partes do projeto. O engine, lê um ficheiro xml que contém uma estrutura bastante simples, dividida em scenes.

Cada ficheiro que está referenciado nas tags `<model>` é um ficheiro de texto criado pelo generator e que contém todos os pontos das figuras que pretendemos representar. Como o engine lê os ficheiros de pontos a partir dum ficheiro XML utilizamos um parser xml feito em C++ chamado `tinyxml2`.

Com este trecho de código fomos capazes de criar um objeto do tipo `XMLDocument`. Em seguida, usando a função `LoadFile`, abrimos o ficheiro de configuração `formas.xml`, e começamos a manipular o seu conteúdo.

Sabendo então quais os ficheiros que devem ser lidos, este módulo vai simplesmente usá-los como input para funções específicas, responsáveis pelo parsing das linhas e criação dos vértices, posteriormente renderizados nas funções `glut`.

De refeito, que criamos ainda uma estrutura `Pontos` que tem como parâmetros 3 floats, que servem para registar as coordenadas deste mesmo. De seguida usamos um vector que utiliza a estrutura enunciada para os guardar. A função `guardaPontos` recebe como parâmetro o nome dos ficheiros e lê os pontos linha a linha, guardando cada coordenada x, y e z num `Ponto` e de seguida inserindo-as no vector.

## 4 Conclusão

Durante a realização deste trabalho fomos encontrando algumas dificuldades, mais concretamente na geração dos vértices da esfera e do cone. A transição para um sistema de coordenadas esféricas e cilíndricas trouxe ainda alguns problemas, que, a nosso ver, foram totalmente ultrapassados e compreendidos, simplificando a criação dos algoritmos destas figuras geométricas.

Deste modo cumprimos todas as exigências pretendidas para esta primeira fase do trabalho, construindo todas as figuras geométricas pedidas. Em suma, tentamos implementar uma solução geral o suficiente que fosse capaz de suportar os requisitos futuros para as próximas fases deste projeto.

## 5 Referências

<http://www.grinninglizard.com/tinyxml/>

<http://stackoverflow.com/questions/170686/what-is-the-best-open-xml-parser-for-c>

<http://maps.jpl.nasa.gov/>

<http://planetpixelemporium.com/planets.html>

<http://www.solarsystemscope.com/textures/>