

# Computação Gráfica

Etienne Costa(A76089)      Joana Cruz(A76270)  
Rafael Alves(A72629)      Maurício Salgado(por numero)

# **1 Introdução**

O relatório apresentado diz respeito à primeira fase do trabalho proposto no âmbito da unidade curricular de Computação Gráfica. O trabalho consiste no desenvolvimento de um gerador de vértices de algumas primitivas gráficas(plano, caixa, esfera e cone). Para além disto, também foi desenvolvida uma aplicação de leitura de ficheiros de configuração em XML que servirá para desenhar os vértices anteriormente gerados.

## 2 Generator

A função main do gerador vê qual a primitiva gráfica que se pretende gerar, como já referimos, pode ser um plano, uma caixa, uma esfera ou um cone. De seguida chama a função geradora correspondente, tendo em conta o número de argumentos pedidos, e escreve como resultado os vértices gerados.

- **Plano** - apenas recebe um argumento, por ser pedido um plano quadrado e corresponde ao lado do plano.
- **Caixa** - recebe como argumentos o comprimento, a altura, a largura e o número de divisões. Caso não se pretenda dividir esse argumento tomará o valor 1.
- **Esfera** - recebe como argumentos o raio, o número de slices(divisões verticais) e o número de stacks(divisões horizontais).
- **Cone** -

### 3 Plano

É pretendido um plano  $XZ$  quadrado, centrado na origem e feito com 2 triângulos. Para calcular os pontos que constituem o plano precisamos do tamanho de cada lado que nos dará informação sobre a dimensão do plano no eixo dos  $xx$ , e dimensão do plano no eixo dos  $zz$ . O plano contém 4 pontos e sendo centrado na origem temos que efetuar os seguintes cálculos:

$$\begin{aligned}x &= tamanho/2 \\y &= 0 \\z &= tamanho/2\end{aligned}$$

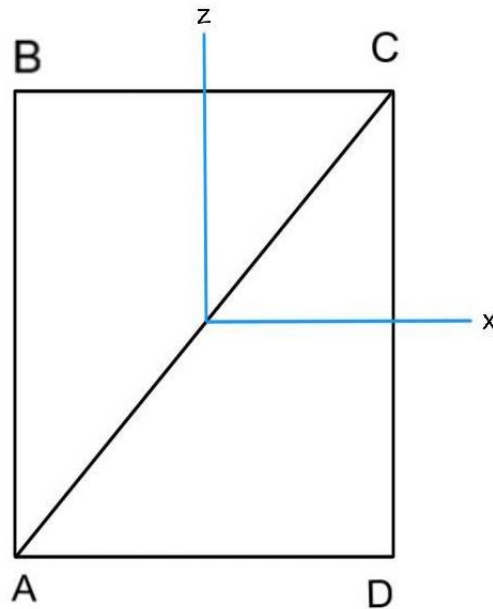


Figura 1: Figura ilustrativa de um plano XZ

Efetuando os cálculos, obtemos os seguintes pontos:

$$\begin{aligned}A &= (-x, 0, z) \\B &= (x, 0, z) \\C &= (-x, 0, -z) \\D &= (x, 0, -z)\end{aligned}$$

Gerando o plano a partir do ponto A, e segundo o OpenGL, para a superfície do plano ficar visível do lado de fora pela regra da mão direita, obtemos que os vértices dos triângulos  $ABC$  e  $ACD$ , terão como coordenadas:

$$ABC \rightarrow (-x, y, -z)(-x, y, z)(x, y, z)$$

$$ACD \rightarrow (-x, y, -z)(x, y, z)(x, y, -z)$$

Exemplo: *plane 5*

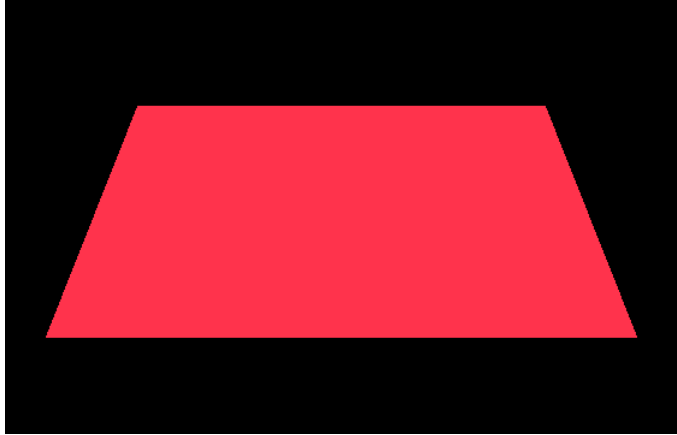


Figura 2: Exemplo de um plano

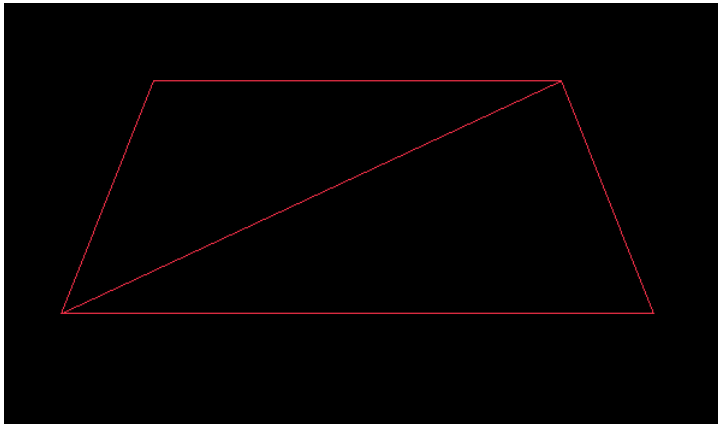


Figura 3: Exemplo de um plano com a representação dos triângulos

Caso fosse pedido qualquer plano  $XZ$  teríamos que receber dois parâmetros: as dimensões do plano no eixo do  $xx$  e no eixo  $zz$ .

## 4 Caixa

O cálculo dos pontos de uma caixa precisa dos seguintes parâmetros comprimento(dimensão no eixo dos  $xx$ ), altura(dimensão no eixo dos  $yy$ ), largura(dimensão no eixo dos  $zz$ ) e o número de divisões. Uma caixa pode ou não conter divisões pelo que precisamos de guardar informação sobre o número de divisões, sendo estas calculadas pelas seguintes equações:

$$divX = \frac{dimX}{div}$$

$$divY = \frac{dimY}{div}$$

$$divZ = \frac{dimZ}{div}$$

E de modo a que a caixa fique centrada na origem precisamos das coordenadas  $x$ ,  $y$ , e  $z$  do seu centro:

$$dimX = \frac{comprimento}{2}$$

$$dimY = \frac{altura}{2}$$

$$dimZ = \frac{largura}{2}$$

Precisamos de calcular as faces XY, as faces XZ e as faces YX

Exemplo: *box 4 4 4 2*

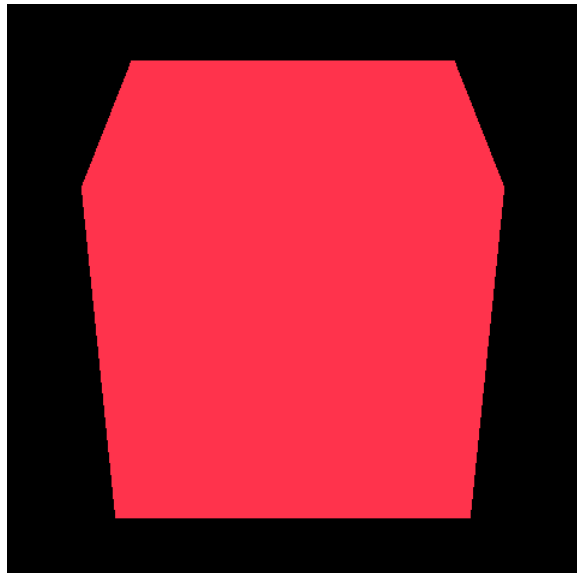


Figura 4: Exemplo de uma caixa com divisões

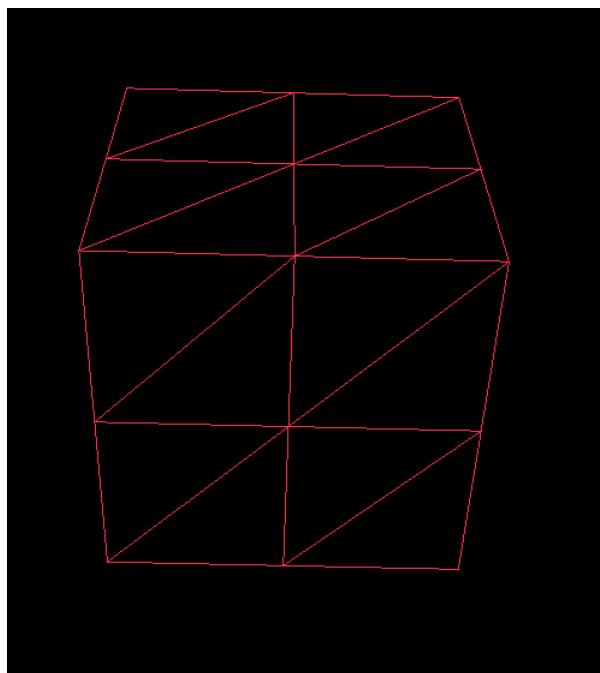


Figura 5: Exemplo de uma caixa com divisões com a representação dos triângulos

## 5 Esfera

O cálculo dos pontos de uma esfera necessita de 3 parâmetros: raio, slices que correspondem às divisões na vertical ao longo da esfera e stacks que correspondem às divisões na horizontal ao longo da esfera. Quanto maior o número de slices e stacks, maior será o número de pontos a determinar, ou seja, melhor será a precisão da esfera. Sabemos que:

- A intersecção entre uma slice e uma stack origina 4 pontos
- A distância entre cada um destes pontos ao centro é o raio
- Podemos ter um vetor para cada ponto, e esse vetor tem dois ângulos, um relativo ao eixo dos  $yy(\beta)$  e outro relativo ao eixo dos  $zz(\alpha)$ . Em vez de termos relativo ao eixo dos  $zz$ , poderíamos ter relativo ao eixo dos  $xx$
- O ângulo  $\alpha \in [0; 2\pi]$  e depende do número de slices
- O ângulo  $\beta \in [0; \pi]$ , e depende do número de stacks
- Temos um  $\Delta\alpha$  que será calculado através  $\frac{2 \times \pi}{slices}$
- Temos um  $\Delta\beta$  que será calculado através  $\frac{\pi}{stacks}$

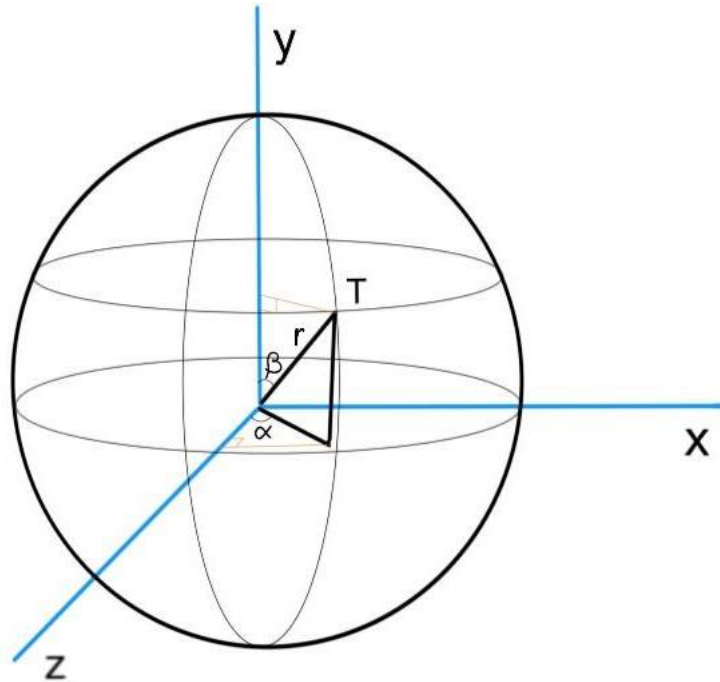


Figura 6: Representação de um ponto T na superfície de uma esfera e os respectivos ângulos

Após a nossa representação da esfera, facilmente conseguimos retirar as equações para obter as coordenadas do ponto T:

$$\begin{aligned}x &= r \times \sin(b) \times \sin(a) \\y &= r \times \cos(b) \\z &= r \times \sin(b) \times \cos(a)\end{aligned}$$



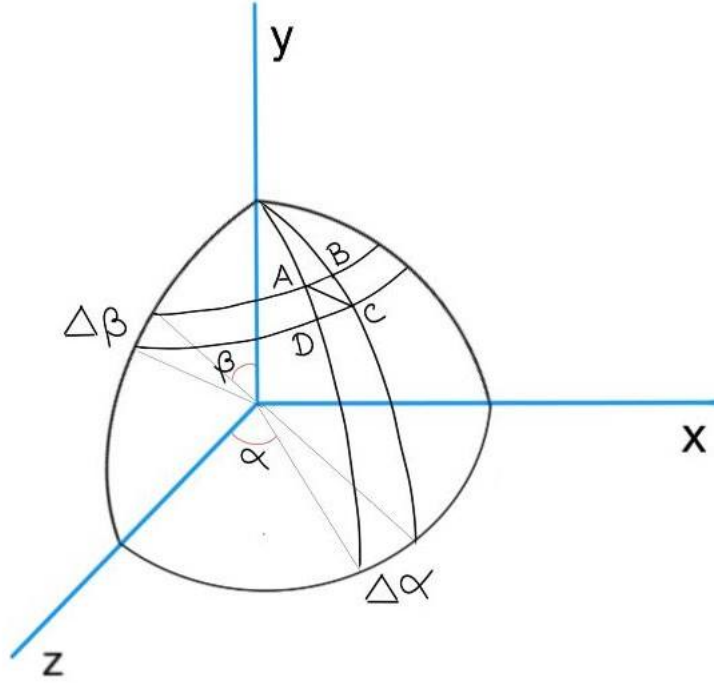


Figura 7: Representação dos 4 pontos de uma interseção

Na figura acima apresentada demonstramos um exemplo de uma interseção entre slices e stacks em que geramos os pontos  $A, B, C$  e  $D$ . Estes pontos servirão para calcular os vértices correspondentes aos triângulos  $ABC$  e  $ACD$ . Por último falta-nos determinar, os ângulos correspondentes aos pontos  $B, C$  e  $D$ .

Ponto	$\alpha$	$\beta$
B	$\alpha + \Delta\alpha$	$\beta$
C	$\alpha + \Delta\alpha$	$\beta + \Delta\beta$
D	$\alpha$	$\beta + \Delta\beta$

Para cada stack  $i$  {

$$\beta = i \times \Delta\beta$$

Para cada slice  $j$  {

$$\alpha = j \times \Delta\alpha$$

*Ponto A*

$$x = r \times \sin(\beta) \times \sin(\alpha)$$

$$y = r \times \cos(\beta)$$

$$z = r \times \sin(\beta) \times \cos(\alpha)$$

*Ponto B*

$$x = r \times \sin(\beta) \times \sin(\alpha + \Delta\alpha)$$

$$y = r \times \cos(\beta)$$

$$z = r \times \sin(\beta) \times \cos(\alpha + \Delta\alpha)$$

*Ponto C*

$$x = r \times \sin(\beta + \Delta\beta) \times \sin(\alpha + \Delta\alpha)$$

$$y = r \times \cos(\beta + \Delta\beta)$$

$$z = r \times \sin(\beta + \Delta\beta) \times \cos(\alpha + \Delta\alpha)$$

*Ponto D*

$$x = r \times \sin(\beta + \Delta\beta) \times \sin(\alpha)$$

$$y = r \times \cos(\beta + \Delta\beta)$$

$$z = r \times \sin(\beta + \Delta\beta) \times \cos(\alpha)$$

}  
}

Exemplo: *sphere 4 100 100*

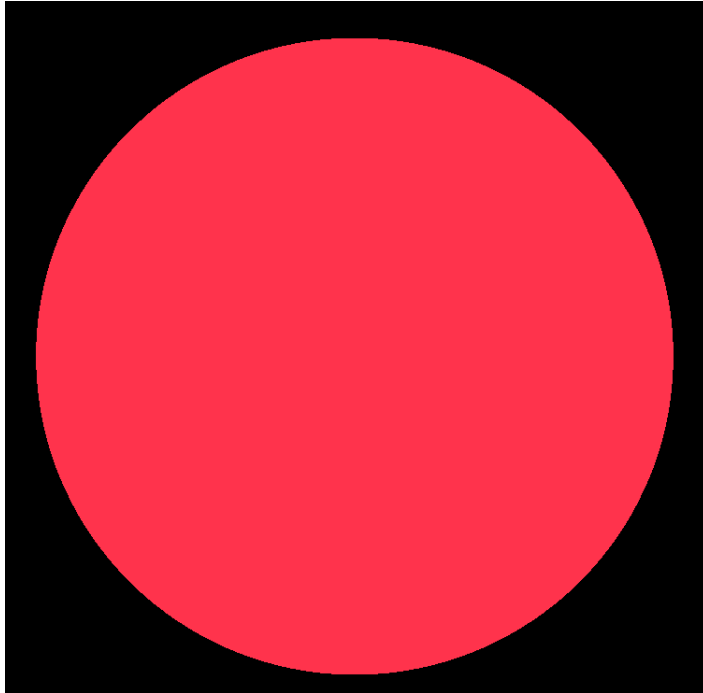


Figura 8: Exemplo de uma esfera

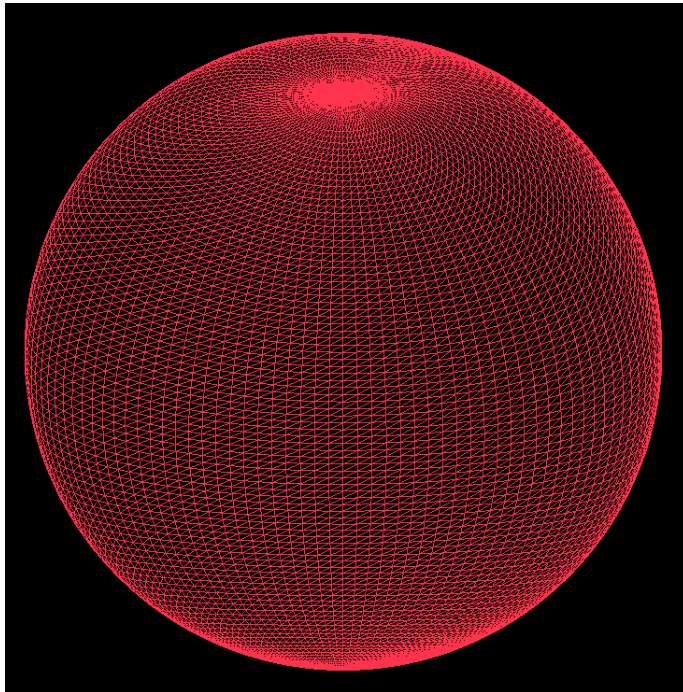


Figura 9: Exemplo de uma esfera com a representação dos triângulos

## 6 Cone

Para a geração dos pontos que irão representar o cone, foi necessário recorrer ao sistema de coordenadas polares. Tendo como base as seguintes fórmulas:

$$x = r \times \sin(\alpha)$$

$$y = i \times \frac{\text{altura}}{\text{stacks}}$$

$$z = r \times \cos(\alpha)$$

Para este algoritmo em concreto foi necessário optar por uma prática recorrente Divide and Conquer, resolvendo inicialmente a base do nosso cone. Para o efeito foi usado o seguinte ciclo:

```
// Base
for (int i = 0; i < slices; i++) {
    alfa = i * incAlfa;
    x = raio * sin(alfa);
    y = 0;
    z = raio * cos(alfa);

    file << 0 << " " << y << " " << 0 << endl;
    file << raio * sin(alfa + incAlfa) << " " << y << " " << raio * cos(alfa + incAlfa) << endl;
    file << x << " " << y << " " << z << endl;
}
```

Figura 10: Algoritmo para a geração da base.

Os ciclos que foram implementados posteriormente seguem a mesma base da geração de uma esfera, porém existe a necessidade de definir um novo raio e aumentar a respectiva altura do modelo a cada camada ao longo da iteração.

```
// Cone
for (int i = 0; i < stacks; i++) {
    y = i * incHeight;
    // New radius
    newRadius = aux * (altura - ((i+1) * incHeight));
    for (int j = 0; j < slices; j++) {
        alfa = j * incAlfa;

        x = oldRadius * sin(alfa);
        z = oldRadius * cos(alfa);

        // First Triangle
        file << x << " " << y << " " << z << endl;
        file << oldRadius * sin(alfa + incAlfa) << " " << y << " " << oldRadius * cos(alfa + incAlfa) << endl;
        file << newRadius * sin(alfa + incAlfa) << " " << y + incHeight << " " << newRadius * cos(alfa + incAlfa) << endl;

        // Second Triangle
        file << x << " " << y << " " << z << endl;
        file << newRadius * sin(alfa + incAlfa) << " " << y + incHeight << " " << newRadius * cos(alfa + incAlfa) << endl;
        file << newRadius * sin(alfa) << " " << y + incHeight << " " << newRadius * cos(alfa) << endl;
    }

    oldRadius = newRadius;
}

// Close File
file.close();
```

Figura 11: Algoritmo para a geração do cone.

Com base nos dois ciclos acima apresentados conseguimos assim gerar o seguinte cone:

Exemplo: *cone 1 2 50 50*

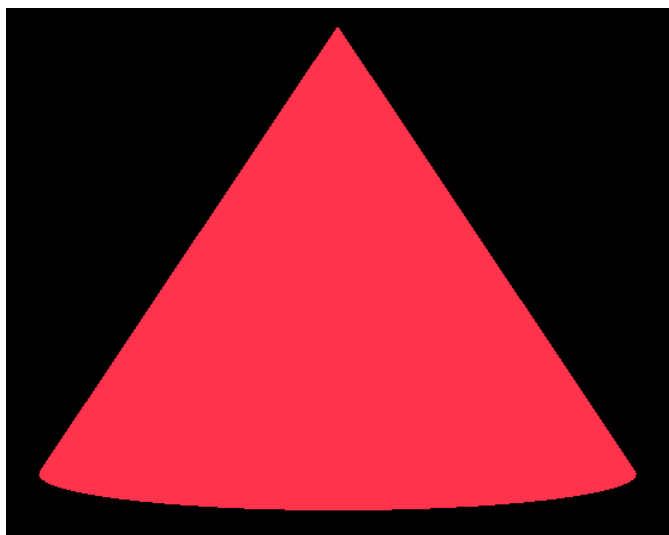


Figura 12: Exemplo de um cone

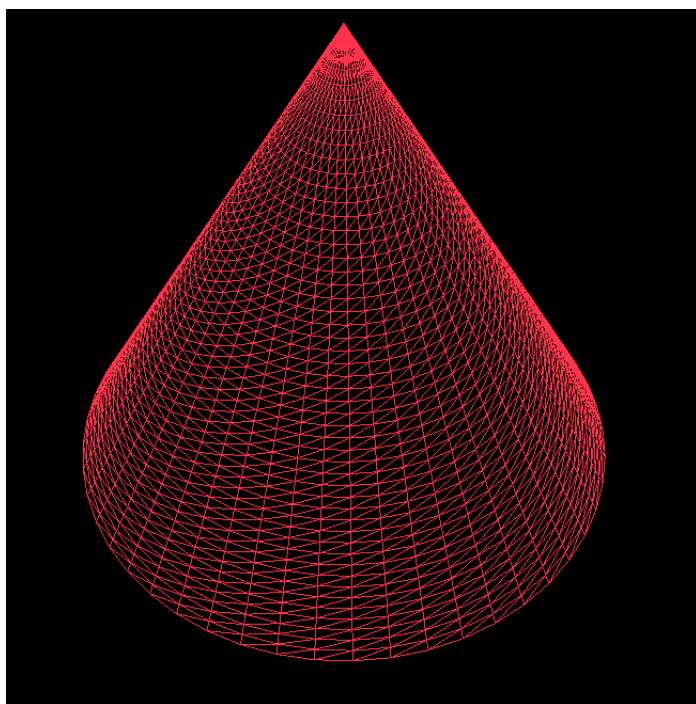


Figura 13: Exemplo de um cone com a representação dos triângulos

## **7 Conclusão**