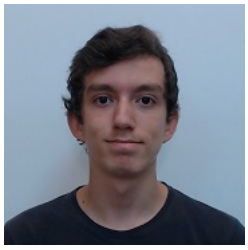
Trabalho Prático - fase 1
Primitivas gráficas

Catarina Cruz
A84011

Jorge Mota
A85272



Maria Silva
A83840

Mariana Marques
A85171

06 de Março de 2020

Universidade do Minho

Conteúdo

1 Introdução

Na Unidade Curricular de Computação Gráfica foi proposto um trabalho prático constituído por quatro fases, com o intuito de desenvolver um motor gráfico genérico para representar objetos a 3 dimensões. O objetivo final deste projeto é construir um sistema solar com as ferramentas desenvolvidas.

Nesta primeira fase é requerido duas aplicações, uma para gerar ficheiros com as informações dos modelos (gerador) e o próprio mecanismo capaz de ler um ficheiro de configuração escrito em XML e capaz de exibir os modelos (render engine). Criamos o gerador de vértices para quatro primitivas gráficas principais: plano, caixa, esfera e cone. Tivemos em conta parâmetros como raio (radius), altura (height), largura (width) e número de divisões(divisions/slices) e pilhas (stacks).

2 Coordenadas

Ao longo deste projeto, para facilitar a representação de diferentes primitivas gráficas foram usadas diferentes tipo de coordenadas. Para o plano e para a caixa, foram usadas as coordenadas cartesianas, no entanto para a esfera e para o cone foram usadas, respetivamente, coordenadas esféricas e cilíndricas.

Foi ainda usado o sistema de eixos típico da área de computação gráfica, representado na figura abaixo.

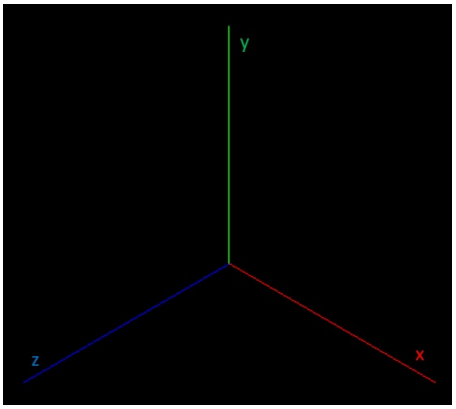


Figura 1: Orientação dos eixos cartesianos

2.1 Coordenadas Cartesianas

(x, y, z)

$x \in R,\ y \in R,\ z \in R$

2.2 Coordenadas Esféricas

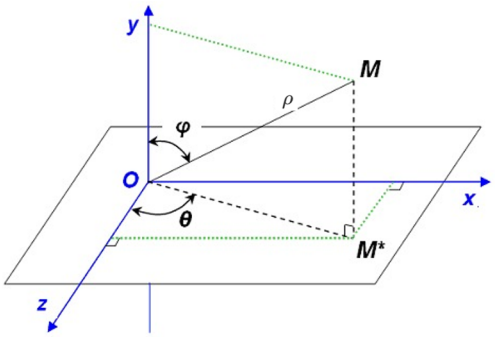


Figura 2: Esquema das coordenadas esféricas

Para o uso das coordenadas esféricas (ρ, θ, ϕ) tivemos em conta os seguintes parâmetros:

$$(x, y, z) \rightarrow (\rho, \theta, \phi)$$

$$\rho \in R^{+}$$
$$\theta \in [0, 2\pi[$$
$$\phi \in [0, \pi]$$

2.2.1 Conversão coordenadas cartesianas em esféricas

$$\rho = \sqrt{x^2 + y^2 + z^2}$$
$$\theta = arctan(x/z)$$
$$\phi = arccos(y/\sqrt{x^2 + y^2 + z^2})$$

2.2.2 Conversão coordenadas esféricas em cartesianas

$$x = \rho * sin(\theta) * sin(\phi)$$
$$\mathrm{y} = \rho * cos(\phi)$$
$$\mathrm{z} = \rho * cos(\theta) * cos(\phi)$$

2.3 Coordenadas Cilíndricas

Para o uso das coordenadas esféricas (ρ, y, θ) tivemos em conta os seguintes parâmetros:

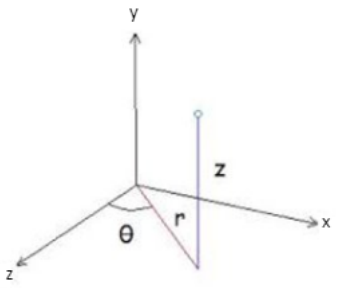


Figura 3: Esquema das coordenadas cilíndricas

$$(x, y, z) \rightarrow (\rho, y, \theta)$$
$$\mathrm{r} \in R^{+}$$
$$\mathrm{y} \in R$$
$$\theta \in [0, 2\pi]$$

2.4 Conversão coordenadas cartesianas em cilíndricas

$$r = \sqrt{x^2 + z^2}$$
$$\mathrm{y} = \mathrm{y}$$
$$\theta = arctang(x/z)$$

2.5 Conversão coordenadas cilíndricas em cartesianas

$$x = r * sin(\theta)$$
$$\mathrm{y} = \mathrm{y}$$
$$\mathrm{z} = \mathrm{r} * \cos(\theta)$$

3 Gerador

Com o objetivo de criar um gerador consistente, garantiu-se que uma primitiva é sempre gerada a partir de triângulos, de forma a ser possível modelar a forma pretendida. Por exemplo, para se gerar um quadrado, juntam-se 2 triângulos.

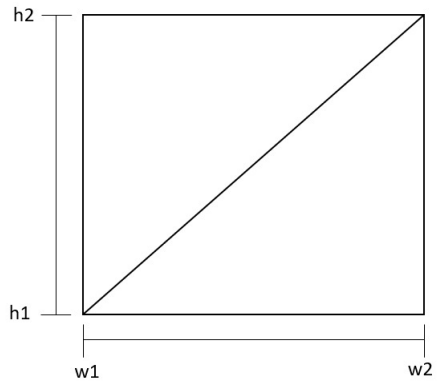


Figura 4: Quadrado criado a partir de 2 triângulos
(w1/w2 representam width e h1/h2 representam height)

Esta lógica é adotada para gerar todas as primitivas e os vértices obtidos como resultado serão guardados num ficheiro.

É também tido em conta que a forma de desenhar triângulos no OpenGL tem de ter uma certa orientação, orientação essa que é definida pela ordem de desenho dos vértices do triângulo. Assim, os vértices têm de ser construídos no sentido contra-relógio para que o triângulo desenhado seja visível.

Nota: Para simplificação, as primitivas geradas foram centradas na origem.

3.1 Plano

`plane [width]`

O plano é o mais simples de gerar , sendo apenas preciso 2 triângulos para construir um quadrado. O plano XZ é uma figura composta apenas por 6 pontos que formam 2 triângulos.

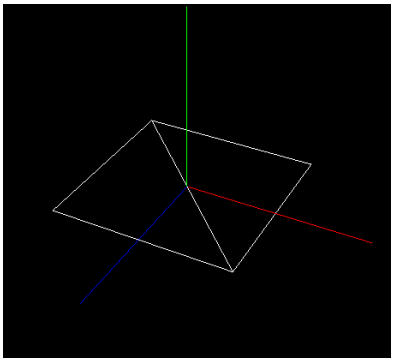


Figura 5: Plano gerado com width=1

3.1.1 Algoritmo

É passado como parâmetros o comprimento do quadrado (*width*) usado para calcular os diferentes vértices.

Para definir o triângulo superior:

```
(  width/2, 0 ,  width/2)
( -width/2, 0 , -width/2)
( -width/2, 0 ,  width/2)
```

Para definir o triângulo inferior:

```
( -width/2 , 0 , -width/2)
(  width/2 , 0 ,  width/2)
(  width/2 , 0 , -width/2)
```

3.2 Caixa

`box [width_x] [width_y] [width_z] [divisions]`

A caixa é como um paralelepípedo normal mas com divisões verticais e horizontais em cada face.

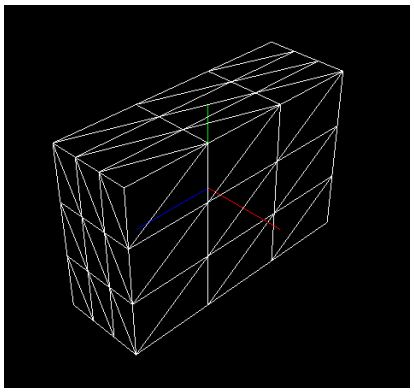


Figura 6: Caixa gerada com width_x=1 , width_y=2 , width_z=3 e divisions=2

3.2.1 Algoritmo

É passado como parâmetros o comprimento(*width_x*), largura(*width_z*), altura(*width_y*) e divisões(*divisions*) para a geração da caixa. Esta geração é feita da seguinte forma:

```
// Nota: operações feitas para centrar a caixa na origem

var half_width_x = width_x / 2;
var div_width_x = width_x / (divisions+1);

// Considerando que cada face é uma "matriz" de quadrados formados por 2
triângulos

Para 'i' de 0 até divisions+1
{
    Para 'j' de 0 até divisions+1
    {
        // Estas variáveis são valores das posições que delimitam cada
pequeno quadrado na face

        var wx1 = half_width_z - div_width_z*i
        var wx2 = half_width_z - div_width_z*(i+1)
```

```
        var hx1 = -half_width_y + div_width_y*j
        var hx2 = -half_width_y + div_width_y*(j+1)

    // Para gerar a face frontal no eixo dos X
        ( half_width_x , hx1 , wx1 )
        ( half_width_x , hx1 , wx2 )
        ( half_width_x , hx2 , wx2 )

        ( half_width_x , hx1 , wx1 )
        ( half_width_x , hx2 , wx2 )
        ( half_width_x , hx2 , wx1 )

        // Outras faces seguem a mesma ideia mas tendo em conta
cada um dos eixos
    }
}
```

3.3 Esfera

`sphere [radius] [slices] [stacks]`

Uma esfera apresenta um raio e é dividida em stacks e slices.

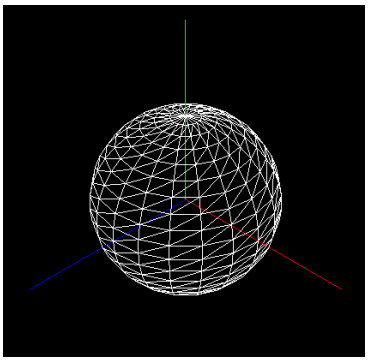


Figura 7: Esfera gerada com radius=1, slices=20 e stacks=20

3.3.1 Algoritmo

Para facilitar a criação dos pontos foram utilizadas coordenadas esféricas (ρ, θ, ϕ). Como a esfera consiste em "retângulos"no corpo geral e triângulos nos polos, dividimos a esfera em 3 partes Top , Body e Down, tal como esquematizado na figura abaixo.

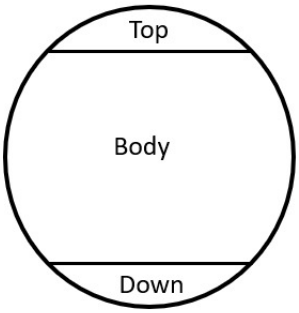


Figura 8: Divisão da esfera

Top e Down são os discos centrados nos polos da esfera formados apenas por triângulos e Body é o resto da esfera que é formada por "retângulos"(que na realidade são apenas 2 triângulos).

Todas estas partes são geradas a cada (2π / slices) radianos.

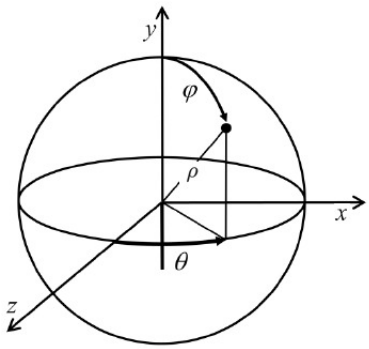


Figura 9: Geração da esfera a partir de coordenadas esféricas

```
var stack_angle = PI / stacks
var slices_angle = 2*PI / slices

Para 'i' de 0 a slices
{
    // Coordenadas esféricas (rho,theta,phi)

    // Top Triangle
    ( radius , 0       , 0       )
    ( radius , i*slices_angle     , stack_angle )
    ( radius , (i+1)*slices_angle , stack_angle )

    // Body
    Para 'j' de 0 a stacks
    {
        ( radius , (i+1)*slices_angle , j*stack_angle     )
        ( radius , i*slices_angle     , j*stack_angle     )
        ( radius , i*slices_angle     , (j+1)*stack_angle )

        ( radius , (i+1)*slices_angle , j*stack_angle     )
        ( radius , i*slices_angle     , (j+1)*stack_angle )
        ( radius , (i+1)*slices_angle , (j+1)*stack_angle )
    }

    // Down Triangle
    ( radius , 0     , PI         )
    ( radius , (i+1)*slices_angle   , PI - stack_angle )
    ( radius , i*slices_angle       , PI - stack_angle )
}
```

3.4 Cone

`cone [radius] [height] [slices] [stacks]`

Um cone apresenta um raio e altura e é dividida em stacks e slices.

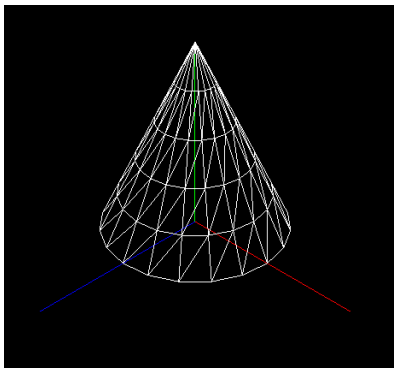


Figura 10: Cone gerado com radius=1, height=2, slices=20 e stacks=5

3.4.1 Algoritmo

Para gerar os pontos mais facilmente utilizou-se coordenadas cilíndricas (ρ, y, θ).

Tal como na esfera , dividimos o cone em 3 partes : Top , Body e Base.
O Top contém a parte do corpo do cone que possui apenas triângulos e formam um mini cone sem base , a Base contém a base do cone que forma um círculo a partir de triângulos e o Body é a parte do cone que contém apenas "rectângulos"(que na realidade são apenas 2 triângulos).

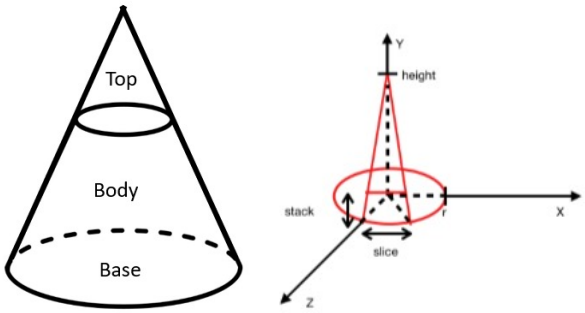


Figura 11: Divisão do cone e representação dos argumentos

Todas estas partes são geradas a cada (2π / slices) radianos.

```
var angle = 2*PI / slices
var stack_height = height / stacks
var stack_radius = radius / stacks

Para 'i' de 0 a slices
{
    var a1 = i*angle
    var a2 = (i+1)*angle

    // Coordenadas cilindricas (rho,y,theta)

    // Top Triangle
    ( 0              , height                , 0  )
    ( stack_radius   , height - stack_height , a1 )
    ( stack_radius   , height - stack_height , a2 )

    // Cone Body
    Para 'j' de 0 a stacks-1
    {
        var r1 = radius - j*stack_radius
        var r2 = radius - (j+1)*stack_radius
        var y1 = j*stack_height
        var y2 = (j+1)*stack_height

        ( r1 , y1 , a1 )
        ( r2 , y2 , a2 )
        ( r2 , y2 , a1 )

        ( r1 , y1 , a2 )
        ( r2 , y2 , a2 )
        ( r1 , y1 , a1 )
    }

    //Base Triangle
    ( radius , 0 , a2 )
    ( radius , 0 , a1 )
    ( 0      , 0 , 0  )
}
```

4 Render engine

Foi criado um render engine com funcionalidades básicas para poder desenhar os modelos gerados pelo gerador. Esta engine permite escolher um ficheiro no formato xml passado como argumento ao programa e renderizar os modelos lá invocados.

4.1 Câmara

É possível visualizar-se as diferentes primitivas, de duas formas diferentes, modo preenchido e em linhas.

Pode-se ainda alterar a posição da câmara, pressionando as setas, rodando esfericamente nas respetivas direções.

- $\rightarrow$: Desloca-se para a direita
- $\leftarrow$: Deslocase para a esquerda
- $\uparrow$: Desloca-se para cima
- $\downarrow$: Desloca-se para baixo
- `space`: Altera entre modo preenchido e linhas

4.2 XML Parser

Por opção, escolhemos utilizar a biblioteca de xml 'rapidxml'.

Cada mesh é criada a partir de um ficheiro nome.3d que contém todos os vértices.

O resultado da leitura do ficheiro xml passa para um objeto 'scene' que mantém uma estrutura com todas as 'mesh' que houver no ficheiro.

5 Conclusão

Nesta primeira fase do projeto foi possível compreender o modo de funcionamento de um motor gráfico e como é que o OpenGL desenha os diferentes modelos gerados por nós.

Podemos ainda refletir sobre como funciona a aritmética que está por detrás da construção de primitivas tridimensionais tais como as que foram propostas para esta fase do projeto.

Fazendo uma análise geral do trabalho desenvolvido concluímos que conseguimos fazer com que todas as primitivas fossem geradas e representadas pelo motor gráfico corretamente.