Universidade do Minho

Licenciatura em Ciências da Computação

Computação Gráfica

Phase 1 - Graphical Primitive

Grupo Nº 1

Bruna Carvalho	Carlos Beiramar	Daniel Ferreira	Ricardo Cruz
(a87982)	(a84628)	(a85670)	(a86789)

13 de março de 2021

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

O objetivo proposto nesta primeira fase do trabalho prático é o desenvolvimento de um programa de criação e desenho de algumas primitivas gráficas, sendo estas um plano, uma caixa, uma esfera e um cone.
Para isso, foram necessárias duas aplicações o ***generator*** - para gerar os ficheiros com a informação dos modelos a desenhar e o ***engine*** - para ler um ficheiro de configuração, escrito em *XML*, e desenhar os modelos gerados anteriormente.
Mais adiante, iremos apresentar, de forma pormenorizada, o processo de elaboração de cada primitiva, recorrendo a equações e esquemas.

Capítulo 2

Primitivas Gráficas

2.1 Plano

Para a criação de um plano é necessário, apenas, um parâmetro(**size**) que corresponde ao comprimento de cada lado. Dado que foram usados triângulos para formar o plano, foi necessário definir 4 pontos.
De modo a centrar o plano na origem do referencial **xOz**, calculou-se a metade do tamanho, passado como parâmetro, $\boldsymbol{half = size/2}$, as coordenadas dos pontos serão as seguintes:

- $P_1(half, 0, half)$
- $P_2(half, 0, -half)$
- $P_3(-half, 0, half)$
- $P_4(-half, 0, -half)$

Para que o plano fosse visível em todos os ângulos, foi implementada a "regra da mão direita"como se pode verificar pelo seguinte referencial:

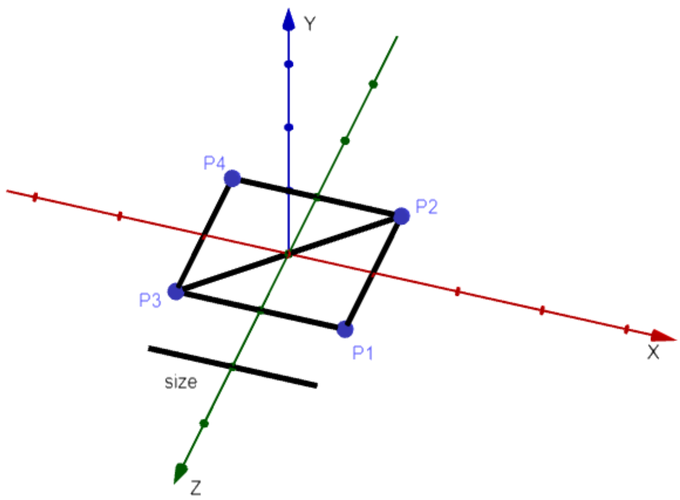


Figura 2.1: Plano

Com o intuito de ser possível visualizar o plano em diferentes perspetivas, foram desenhados 4 triângulos, 2 para cada perspetiva.
Para a direção contra os ponteiros do relógio:

- $T_1(P_1, P_2, P_3)$
- $T_2(P_2, P_4, P_3)$

Para a direção a favor dos ponteiros do relógio:

- $T_3(P_1, P_3, P_2)$
- $T_4(P_2, P_3, P_4)$

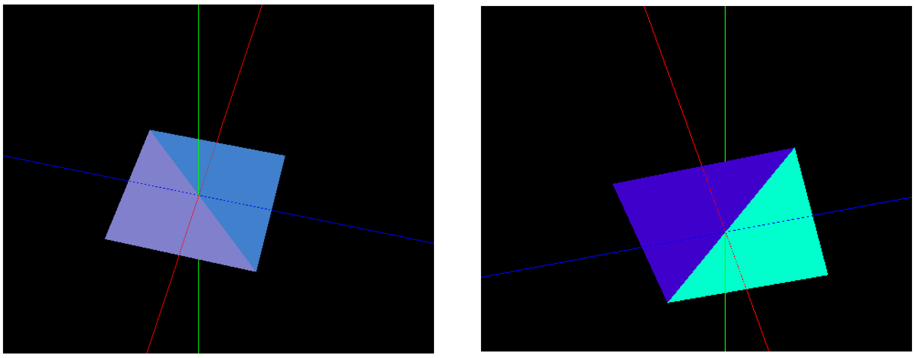
2.1.1 Resultado

Comando:

./generator plane size plane.3d

Em que:

(a) **size** = 4



(a) Plano de Cima (b) Plano de Baixo

Figura 2.2: Plano Visto de Cima e de Baixo

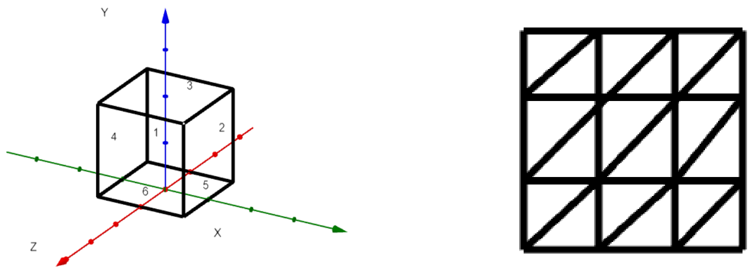
2.2 Caixa

Uma caixa é constituída por 6 faces e cada uma destas faces pode ser subdividida em N (passado como parâmetro) quadrados, sendo que estes quadrados são formados por dois triângulos. De maneira a perceber a forma como o código foi implementado, é importante salientar que, a construção da caixa foi feita face a face. Visto que um dos argumentos é o número de divisões por aresta, foi necessário calcular as novas dimensões **XYZ** da seguinte maneira:

$$\boldsymbol{newx = x/(N+1);}$$

$$\boldsymbol{newz = z/(N+1);}$$

$$\boldsymbol{newy = y/(N+1);}$$



(a) Caixa

(b) Face da Caixa

Figura 2.3: Faces da caixa numeradas e face da caixa

2.2.1 Face da Frente

Como a base está centrada na origem do referencial, os valores no eixo do **x** variam entre $[\frac{-x}{2}, \frac{x}{2}]$. Os valores no eixo do **y** variam entre $[0, height]$. Por fim, **z** toma sempre o valor $\frac{z}{2}$. Usou-se o **i** para iterar sobre o **x** e o **j** para iterar sobre o **y**.

- $P_1(i, j, \frac{z}{2})$
- $P_2(i + newx, j, \frac{z}{2})$
- $P_3(i + newx, j + newy, \frac{z}{2})$
- $P_4(i, j + newy, \frac{z}{2})$

Sendo que a orientação dos vértices é definida no sentido contrário ao dos ponteiros dos relógio, todos os triângulos desta face, são definidos da seguinte maneira:

- $T_1(P_1, P_2, P_3)$
- $T_2(P_1, P_3, P_4)$

2.2.2 Face da Trás

Como a base está centrada na origem do referencial, os valores no eixo do **x** variam entre $[\frac{-x}{2}, \frac{x}{2}]$. Os valores no eixo do **y** variam entre $[0, height]$. Por fim, **z** toma sempre o valor $-\frac{z}{2}$. Usou-se o **i** para iterar sobre o **x** e o **j** para iterar sobre o **y**.

- $P_1(i, j, -\frac{z}{2})$
- $P_2(i + newx, j + newy, -\frac{z}{2})$

- $P_3(i + newx, j, -\frac{z}{2})$
- $P_4(i, j + newy, -\frac{z}{2})$

Sendo que a orientação dos vértices é definida no sentido a favor dos ponteiros dos relógio, todos os triângulos desta face, são definidos da seguinte maneira:

- $T_1(P_1, P_2, P_3)$
- $T_2(P_1, P_4, P_2)$

2.2.3 Face da Cima

Como a base está centrada na origem do referencial, os valores no eixo do **x** variam entre $[-\frac{x}{2}, \frac{x}{2}]$. Os valores no eixo do **y** tomam sempre o valor de *height*. Por fim, **z** varia entre $[-\frac{z}{2}, \frac{x}{2}]$. Usou-se o **i** para iterar sobre o **x** e o **j** para iterar sobre o **z**.

- $P_1(i, y, j)$
- $P_2(i + newx, y, j + newz)$
- $P_3(i + newx, y, j)$
- $P_4(i, y, j + newz)$

Sendo que a orientação dos vértices é definida no sentido contrário ao dos ponteiros dos relógio, todos os triângulos desta face, são definidos da seguinte maneira:

- $T_1(P_1, P_2, P_3)$
- $T_2(P_1, P_4, P_2)$

2.2.4 Face da Esquerda

Como a base está centrada na origem do referencial, os valores no eixo do **x** toma sempre o valor de $-\frac{x}{2}$. Os valores no eixo do **y** variam entre $[0, height]$. Por fim, **z** varia entre $[-\frac{z}{2}, \frac{z}{2}]$. Usou-se o **i** para iterar sobre o **z** e o **j** para iterar sobre o **y**.

- $P_1(-\frac{x}{2}, j, i)$
- $P_2(-\frac{x}{2}, j, i + newz)$
- $P_3(-\frac{x}{2}, j + newy, i)$
- $P_4(-\frac{x}{2}, j + newy, i + newz)$

Sendo que a orientação dos vértices é definida no sentido a favor dos ponteiros dos relógio, todos os triângulos desta face, são definidos da seguinte maneira:

- $T_1(P_1, P_2, P_3)$
- $T_2(P_2, P_4, P_3)$

2.2.5 Face da Direita

Como a base está centrada na origem do referencial, os valores no eixo do **x** toma sempre o valor de $\frac{x}{2}$. Os valores no eixo do **y** variam entre $[0, height]$. Por fim, **z** varia entre $[-\frac{z}{2}, \frac{z}{2}]$. Usou-se o **i** para iterar sobre o **z** e o **j** para iterar sobre o **y**.

- $P_1(\frac{x}{2}, j, i)$
- $P_2(\frac{x}{2}, j + newy, i)$
- $P_3(\frac{x}{2}, j, i + newz)$
- $P_4(\frac{x}{2}, j + newy, i + newz)$

Sendo que a orientação dos vértices é definida no sentido contrário ao dos ponteiros dos relógio, todos os triângulos desta face, são definidos da seguinte maneira:

- $T_1(P_1, P_2, P_3)$
- $T_2(P_3, P_2, P_4)$

2.2.6 Face da Baixo

Como a base está centrada na origem do referencial, os valores no eixo do **x** variam entre $[-\frac{x}{2}, \frac{x}{2}]$. Os valores no eixo do **y** tomam sempre o valor de 0. Por fim, **z** varia entre $[-\frac{z}{2}, \frac{z}{2}]$. Usou-se o **i** para iterar sobre o **x** e o **j** para iterar sobre o **z**.

- $P_1(i, 0, j)$
- $P_2(i + newx, 0, j)$
- $P_3(i + newx, 0, j + newx)$
- $P_4(i, 0, j + newx)$

Sendo que a orientação dos vértices é definida no sentido a favor dos ponteiros dos relógio, todos os triângulos desta face, são definidos da seguinte maneira:

- $T_1(P_1, P_2, P_3)$
- $T_2(P_3, P_4, P_1)$

2.2.7 Resultados

Comando:

./generator box x y z slices box.3d

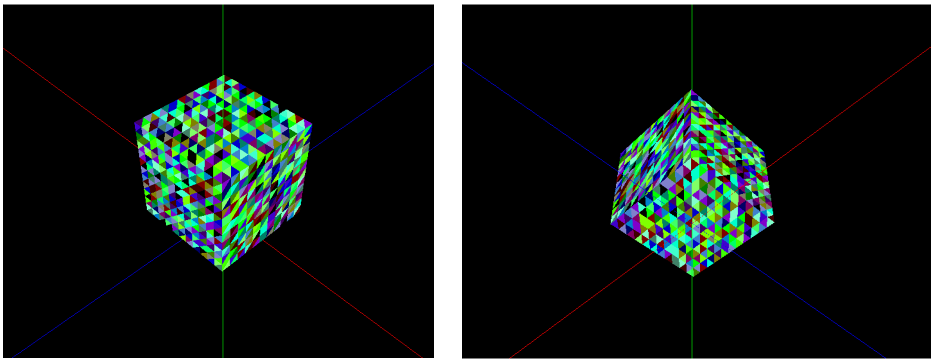
Em que:

a) **x** = 4

b) **y** = 4

c) **z** = 4

d) **slices** = 10



(a) Vista de Cima (b) Vista de baixo

Figura 2.4: Diferentes posições da Caixa

2.3 Esfera

Para a construção da esfera, a estratégia escolhida depende de dois ciclos for. O ciclo exterior percorre o número de *slices* (iterado por **i**) obtido por parâmetro e, o ciclo interior itera sobre o número de *stacks* (iterado por **j**) obtidas igualmente por parâmetro.
Antes de executar qualquer iteração, foi calculado a amplitude de cada um dos novos ângulos (α, β) a criar cada um dos triângulos necessários para formar a esfera. Ângulos esses que foram obtido através das seguintes operações:

$$slicesAlpha = 2 * \pi / slices;$$
$$stacksBeta = \pi / stacks;$$

Nota:

(i) $0 \leq \alpha \leq 360$ e $-90 \leq \beta \leq 90$

(ii) **slicesAlpha** representa a amplitude entre dois pontos adjacentes que partilham o mesmo **y**.

(iii) **stacksBeta** representa a amplitude entre dois pontos adjacentes, no qual diferem na coordenada **y**.

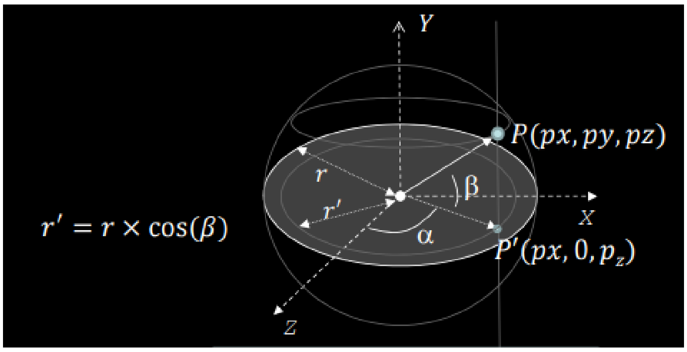


Figura 2.5: Esfera

As coordenadas de cada vértice foram calculadas através das seguintes expressões:

$$x = radius * cos(\beta) * sin(\alpha)$$
$$y = radius * sin(\beta)$$
$$z = radius * cos(\beta) * cos(\alpha)$$

$$\alpha = i * slicesAlpha$$
$$\beta = j * stacksBeta$$

Posto isto, em cada iteração do ciclo sobre as *stacks*, foram criados quatro vértices da seguinte forma:

- $P_1(radius * cos(\beta) * sin(\alpha), radius * sin(\beta), radius * cos(\beta) * cos(\alpha));$
- $P_2(radius * cos(\beta) * sin(\alpha + slicesAlpha), radius * sin(\beta), radius * cos(\beta) * cos(\alpha + slicesAlpha));$
- $P_3(radius * cos(\beta + stacksBeta) * sin(\alpha + slicesAlpha), radius * sin(\beta + stacksBeta), radius * cos(\beta + stacksBeta) * cos(\alpha + slicesAlpha));$
- $P_4(radius * cos(\beta + stacksBeta) * sin(\alpha), radius * sin(\beta + stacksBeta), radius * cos(\beta + stacksBeta) * cos(\alpha));$

Com estes quatro vértices iam sendo criados os seguintes triângulos:

- $T_1(p_1, p_2, p_3)$
- $T_2(p_1, p_3, p_4)$

2.3.1 Resultados

Comando:

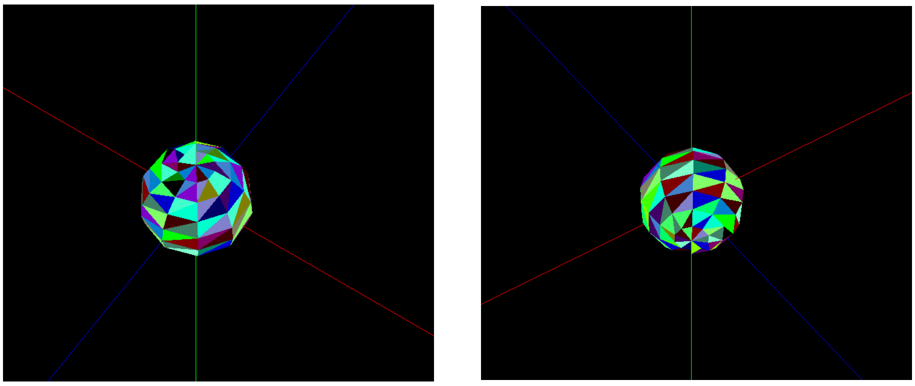
./generator sphere radius slices stacks sphere.3d

Em que:

a) **radius** = 1

b) **slices** = 10

c) **stacks** = 10



(a) Vista de Cima (b) Vista de baixo

Figura 2.6: Diferentes posições da Esfera

2.4 Cone

Antes da construção do cone, é preciso ter algumas considerações iniciais:

1. Base está no plano **xOz**, logo o $y = 0$;
2. $\alpha = 2 * \pi / slices$;
3. $stackHeight = height/stacks$;
4. **i** itera o número de slices;
5. **j** itera o número de stacks;
6. $upperRadius = radius - ((radius/stacks) * (j + 1))$;
7. $upper = (j + 1) * stackHeight$;
8. $bottomRadius = radius - ((radius/stacks) * j)$;
9. $bottom = j * stackHeight$;
10. $deltaAlpha = i * \alpha$.

A base foi elaborada da seguinte maneira:

- $P_1(0.0f, 0.0f, 0.0f)$
- $P_2(sin(deltaAlpha) * radius, 0.0f, cos(deltaAlpha) * radius)$
- $P_3(sin(deltaAlpha + \alpha) * radius, 0.0f, cos(deltaAlpha + \alpha) * radius)$

Com estes três vértices ia sendo criado o seguinte triângulo:

- $T_1(p3, p2, p1)$;

Para a criação do resto do cone foram percorridos dois ciclos, sendo que o ciclo exterior em relação às *stacks* (iterado pelo **j**) e o ciclo interior em relação às *slices* (iterado por **i**). A estratégia usada teve como base o desenho de triângulos entre duas alturas da *stack*, consecutivas, até ser atingida a altura dada como argumento. Nota, ainda para o facto, que quanto mais acima estava a *stack* que estava a ser iterada menor era o raio desta comparativamente à *stack* anterior, imediatamente abaixo.

- $P_1(bottomRadius*sin(deltaAlpha), bottom, bottom_radius*cos(deltaAlpha))$
- $P_2(upperRadius*sin(deltaAlpha+\alpha), upper, upperRadius*cos(deltaAlpha+\alpha))$
- $P_3(upperRadius*sin(deltaAlpha), upper, upperRadius*cos(deltaAlpha))$
- $P_4(bottomRadius*sin(deltaAlpha+\alpha), bottom, bottomRadius*cos(deltaAlpha+\alpha))$

Com estes pontos iam sendo criados os triângulos, da seguinte forma:

- $T_1(p1, p2, p3)$;
- $T2(p1, p4, p2)$;

2.4.1 Resultados

Comando:

./generator cone radius height slices stacks cone.3d

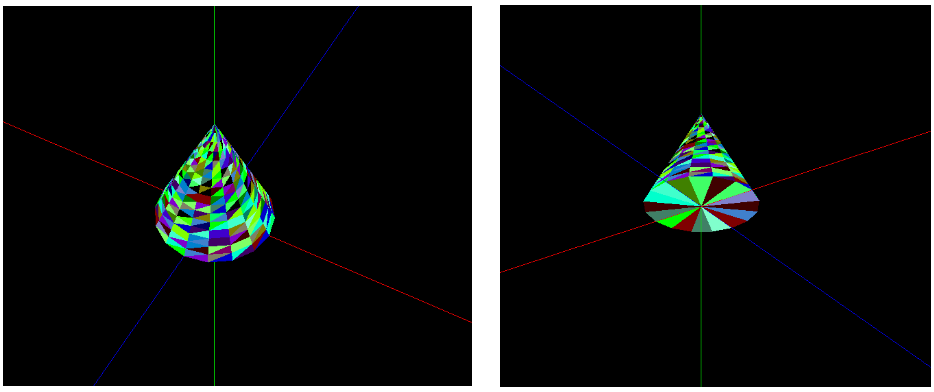
Em que:

(a) **radius** = 2

(b) **height** = 4

(c) **slices** = 16

(d) **stacks** = 16



(a) Vista de Cima

(b) Vista de baixo

Figura 2.7: Diferentes posições da Cone

Capítulo 3

XML

O ficheiro *engine.cpp* está responsável pela leitura dos ficheiros *XML* e guarda os pontos, em memória, para, posteriormente, desenhar os modelos.
Para guardar esta informação, foi desenvolvida uma classe *Scene* composta por dois vetores, um que guarda os nomes dos ficheiros ".3d"(*models*) e outro que armazena vetores de *floats*(*trianglesCoordinates*) correspondentes às coordenadas dos vértices de cada modelo. Dessa forma, para desenhar os modelos percorremos o vetor *trianglesCoordinates*.
De realçar, que nos ficheiros ".3d"cada linha corresponde a um triângulo com as respetivas coordenadas e à sua cor.
Para leitura e escrita de ficheiros *XML* recorreu-se à ferramenta *tinyxml2*. Sempre que um modelo é gerado, a função *updateXML* verifica se, no ficheiro *XML* a que se está a acrescentar, existe um outro modelo com o mesmo nome.
A leitura do ficheiro *XML* é realizada uma primeira vez quando o programa arranca, podendo voltar a ser realizada ao premir a tecla **R**.

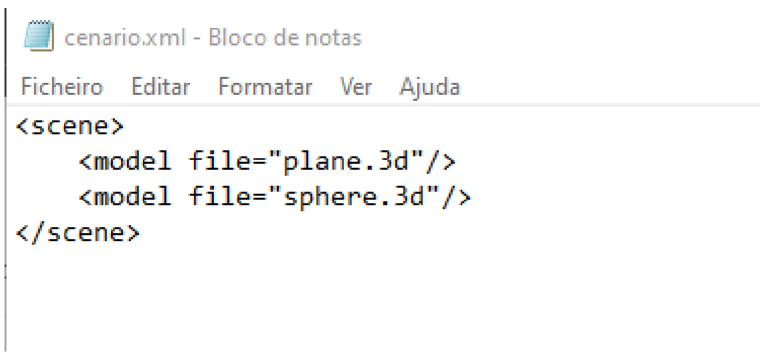


Figura 3.1: Ficheiro *XML*

Capítulo 4

Conclusão

Esta primeira fase permitiu-nos ganhar prática com as ferramentas que são úteis para UC e, também, com as primitivas gráficas.
Sentimos mais dificuldades na criação do cone e da esfera mas, no final, concluímos os resultados idealizados com sucesso.
Em suma, consideramos que cumprimos os objetivos desta etapa com a elaboração de dois executáveis pedidos e das funcionalidades sobre as primitivas que consideramos úteis ao utilizador.