Relatório Primeira Fase Trabalho Prático - Computação Gráfica

Grupo 34 :
António Luís de Macedo Fernandes (a93312)
José Diogo Martins Vieira (a93251)
João Silva Torres (a93231)
Ricardo Lopes Santos Silva (a93195)

13 de Março 2022

Contents

1 Introdução

1.1 Contextualização

No âmbito do desenvolvimento da primeira fase do projeto da unidade curricular de Computação Gráfica, foi-nos pedido duas aplicações. A primeira consiste num gerador de ficheiros xml com as especificações de diferentes figuras, nomeadamente, plano, caixa, esfera, cone e ainda fizemos como um extra para o cilindro. A segunda é um motor que produz as várias figuras num espaço 3D através do ficheiro xml.

As principais ferramentas para a concretização destas aplicações foram C++ e o OpenGl.

De seguida, iremos abordar com mais detalhe as principais estratégias utilizadas e ilustrar alguns exemplos das nossas aplicações.

1.2 Ferramentas e Bibliotecas Usadas

Para além do OpenGL e do GLUT, recorremos à biblioteca *TinyXML2* para nos auxiliar no parsing dos ficheiros *XML* de modo a explorar o seu conteúdo.

2 Generator

O "Generator" tem como objetivo gerar um ficheiro *XML* que contém uma referência ao nome de cada um dos ficheiros .3d criados. que guardam os vértices calculados para um futuro desenho da figura.

2.1 Comandos

Para gerar os ficheiros .3d é necessário introduzir os seguintes comandos:

- Plane - generator plane length divisions file.3d Gera um plano centrado em XOZ, com a *length* e número de *divisions* desejados
- Box - generator box X Y Z divisions file.3d Gera uma caixa centrada na origem com as dimensões X, Y e Z e número de *divisions* desejada
- Sphere - generator sphere radius slices stacks file.3d Gera uma esfera centrada na origem com raio, divisões e stacks fornecidas pelo utilizador
- Cone - generator cone radius height slices stacks file.3d Gera um cone com raio, altura, divisões e stacks dadas pelo utilizador
- (Extra) Cylinder - generator cylinder radius height slices stacks file.3d Gera um cilindro com raio, altura, divisões e stacks proporcionadas pelo utilizador

No entanto, com o comando *./generator help* a seguinte informação é apresentada:

```
Graphical primitives available:

Plane (a square in the XZ plane, centred in the origin, subdivided in both X and Z directions
Example ->./generator plane [length] [division] [filename]

Box (requires dimension, and the number of divisions per edge)
Example ->./generator box [length] [division] [filename]

Sphere (requires radius, slices and stacks)
Example ->./generator sphere [radius] [slices] [stacks] [filename]

Cone (requires bottom radius, height, slices and stacks)
Example ->./generator cone [radius] [height] [slices] [stacks] [filename]

Cylinder (requires radius, height, slices and stacks)
Example ->./generator cylinder [radius] [height] [slices] [stacks] [filename]
```

Figure 1: Help

2.2 Plane

Para gerar um plano recebemos como argumentos a length e as divisões do mesmo.
Para que este seja centrado na origem tivemos de aplicar a seguinte fórmula:

$$initialX = initialZ = \frac{-length}{2}, y = 0$$

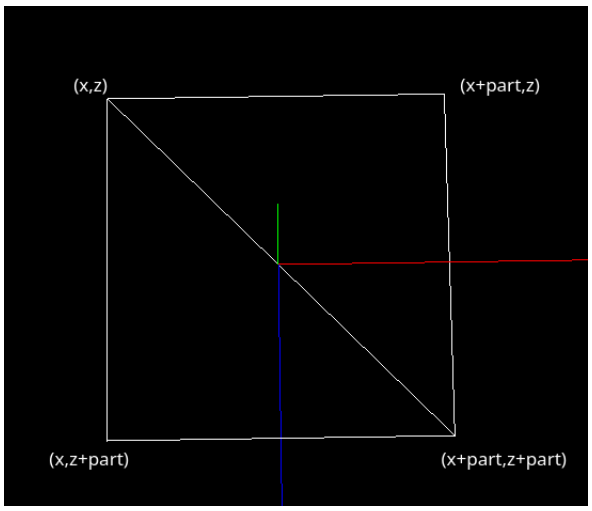

Figure 2: Plano.

De seguida para cada divisão pedida, é necessário desenhar dois triângulos. O cálculo dos vértices para dos triângulos foram as seguintes:

Sabemos que o y vai ser igual a uma constante, neste caso 0.

Precisamos assim de descobrir a posição do X e do Y, para cada divisão.

Calculamos primeiro o valor de cada parte:

$$part = \frac{length}{divisions}$$

Sabemos que vão existir (divisões*divisões). Usamos então um double loop, onde no exterior é incrementado o valor do x, e no interior o valor do z, sendo este o cáculo para calcular a posX e posZ:

$$posX = initialX + i * part;$$

$$posZ = initialZ + j * part;$$

onde i corresponde à variável incremental do ciclo exterior de intervalo [0-divisions[, e j corresponde à variável incremental do ciclo interior de intervalo [0-divisions[

Primeiro triângulo:

$$posX + part, 0, posZ + part$$

$$posX + part, 0, posZ$$

$$posX, 0, posZ$$

Segundo triângulo:

$$posX, 0, posZ$$

$$posX + part, 0, posZ$$

$$posX + part, 0, posZ + part$$

2.3 Box

Para a construção e o cálculo dos vértices da *Box* utilizamos a estratégia já delineada anteriormente para o plano, para as seis faces.

Então, começamos por optar por centrar a *Box* na origem relativamente aos três eixos.

Para desenhar as 6 faces dividiu-se o processo em 3, quando:

- x constante
- y constante
- z constante

Começamos então, para o caso de y constante, ou seja, pela face do topo e da base.

$$yTopo = \frac{length}{2} \qquad yBase = \frac{-length}{2}$$

Inicialmente, X e Z tomam o seguinte valor: $\frac{-length}{2}$

Estes dois valores vão, então, aumentar.

Usa-se assim a estratégia explicada anteriormente para o plano, tendo em atenção a ordem dos vértices, que tem de ser inversa, para ficar desenhado para fora da box e não para dentro.

Repete-se o processo para o X e o Z constantes.

Para os valores em que X é constante:

$$XEsquerda = \frac{-length}{2} \qquad XDireita = \frac{length}{2}$$

$$Yinicial = \frac{length}{2} \qquad ZInicial = \frac{length}{2}$$

Os valores de Y vão então diminuir e os de Z aumentar.

Para os valores de Z constante:

$$ZTrs = \frac{-length}{2} \qquad ZFrente = \frac{length}{2}$$

$$Yinicial = \frac{length}{2} \qquad XInicial = \frac{-length}{2}$$

Os valores de Y vão então diminuir e os de X aumentar.

2.4 Cone

Para a definição do cone são necessários os seguintes parâmentros: radius, height, slices e stacks. É necessário usar coordenadas polares.

Para saber qual o ângulo que vai ter cada slice:

$$angleSlice = \frac{2 * \pi}{slices}$$

Para saber a altura para cada stack:

$$heightStack = \frac{height}{stacks}$$

Já temos toda a informação que precisamos. Para desenhar o cone, temos de pensar que cada stack é uma circunferência distinta, pois o raio de cada circunferência decresce no sentido base-topo.

Assim precisamos de determinar o raio da circunferência para cada stack do nosso cone.Sabemos o raio da nossa base, altura do cone e de cada stack.

Assim conseguimos deconstruir o nosso cone, em cones mais pequenos, e utilizar uma vista espalmada de forma a vizualizá-lo como um triângulo.

Usando a semelhança de triângulos, conseguimos descobrir o raio do cone mais pequeno(ra):

$$ra = raio * \frac{stacks - currentStack}{stacks}$$

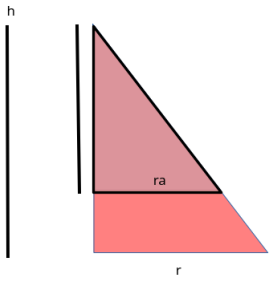

Figure 3: Semelhança de Triângulos

Sendo *stacks* o número de stacks do cone e currentStack a stack de qual queremos descobrir o raio, valor da base sendo 0 e o valor do topo -> stacks. Assim, já temos toda a informação necessária para calcular os vértices.

Com um double loop, onde o exterior percorre as slices e o interior as stacks temos:

Para as slices, sendo i a variável incremental de 0 a slices:

$$angle = i * anglePart \qquad nextAngle = (i+1) * anglePart$$

Para as stacks, sendo initialHeight =0;

$$rDown = r * \frac{stacks - j}{stacks} \qquad rUp = r * \frac{stacks - j - 1}{stacks}$$

$$heightDown = initialHeight + heightPart * j \qquad heightUp = initialHeight + heightPart * (j+1)$$

Calcula-se assim os vértices dos triângulos para cada slice e stack:

$$rUp * sin(nextAngle), heightUP, rUp * cos(nextAngle)$$

$$rUp * sin(angle), heightUP, rUp * cos(angle)$$

$$rDown * sin(angle), heightDown, rDown * cos(angle)$$

$$rDown * sin(angle), heightDown, rDown * cos(angle)$$

$$rDown * sin(nextAngle), heightDown, rDown * cos(nextAngle)$$

$$rUp * sin(nextAngle), heightUP, rUp * cos(nextAngle)$$

E ainda para a base do cone, desenhamos para cada slice:

$$r * sin(nextAngle), initialHeight, r * cos(nextAngle)$$

$$r * sin(angle), initialHeight, r * cos(angle)$$

$$0, initialHeight, 0$$

2.5 Sphere

Para gerar um esfera recebemos como argumentos radius, slices e stacks, é necessário utilizar coordenadas esféricas.

Para tal, precisamos de saber como vai ser o ângulo para cada divisão de slices(angelPart) e das stacks(heightPart), temos então:

$$anglePart = \frac{2\pi}{slices}$$

$$heightPart = \frac{pi}{stacks}$$

De seguida fazemos uso de um double loop, onde o exterior é para as slices e o interior para as stacks:

Para as slices: Precisamos de determinar o angulo atual e o próximo ângulo, (sendo i a variável incremental do loop [0,slices[):

$$angle = i * anglePart$$

$$nextAngle = (i + 1) * anglePart$$

Para as stacks: $initialAngle = \frac{-\pi}{2}$ Precisamos de determinar o angulo da stackAtuale e o da stack seguinte(sendo i a variável incremental do loop [0,stacks[)

$$angleUp = initialAngle + heightPart * (j + 1)$$

$$angleDown = initialAngle + heightPart * (j)$$

(em que $initialAngle = \frac{-\pi}{2}$)

Desta forma, temos assim toda a informação necessária para calcular os vértices. Recorrendo às coordenadas esféricas, temos:

Primeiro Triângulo:

$$radius * cos(angleUp) * sin(angle), radius * sin(angleUp), radius * cos(angle) * cos(angleUp)$$

$radius * cos(angleDown) * sin(angle), radius * sin(angleDown), radius * cos(angle) * cos(angleDown)$

$radius * cos(angleDown) * sin(nextAngle), radius * sin(angleDown), radius * cos(nextAngle) * cos(angleDown)$

Segundo Triângulo:

$radius * cos(angleUp) * sin(nextAngle), radius * sin(angleUp), radius * cos(nextAngle) * cos(angleUp)$

$radius * cos(angleUp) * sin(angle), radius * sin(angleUp), radius * cos(angle) * cos(angleUp)$

$radius * cos(angleDown) * sin(nextAngle), radius * sin(angleDown), radius * cos(nextAngle) * cos(angleDown)$

2.6 (Extra) Cylinder

Para gerar um cilindro recebemos como argumentos radius, height, slices e stacks.

A estratégia utilizada é semelhante à do cone, com a diferença que desta vez o raio não altera, ou seja para cada stack, o raio vai ser igual ao da base. E tendo neste caso uma base em baixo e em cima.

Assim, o cálculo dos vértices das partes das stacks é igual ao do cone exceto no raio, que variava no cone, mas aqui é sempre o mesmo.

Para as slices, é necessário desenhar uma base em baixo como no cone, mas também uma base em cima.

2.7 Ficheiros .3d

Após o cálculo de todos os vértices necessários para desenhar a primitiva gráfica, falta a escrita para o ficheiro.

Esta escrita é feita de um modo simples:

A primeira linha contém o número de vértices que o ficheiro contém.

Nas restantes linhas, encontra-se a informação das coordenadas relativamente a cada vértice.

Cada coordenada separada por um espaço, como se vê na figura em baixo.

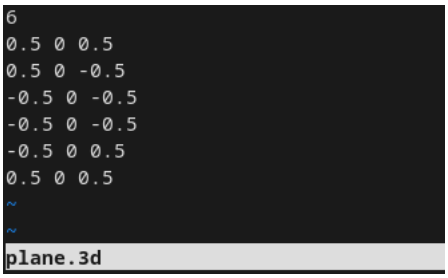

Figure 4: Exemplo ficheiro .3d

3 Engine

O engine será responsável para fazer o parsing dos ficheiros xml, e apresentar graficamente conforme a configuração. Para o parsing dos ficheiros xml, fazemos uso do tinyxml2.

É convertida a posição da camâra que se encontra nos ficheiros xml, de coordenadas cartesianas para coordenadas esféricas. Assim, é mantida a posição da configuração e permite posteriormente a utilização de input para mover a camâra no modo de explorador.

São lidos e guardados os pontos relativamente aos ficheiros da configuração xml, e são apresentados graficamente.

4 Apresentação dos modelos

Nesta secção são apresentados alguns exemplos da geração das diferentes figuras.

4.1 Plane

Gerado plano com 1 *length* e 10 *divisions*.

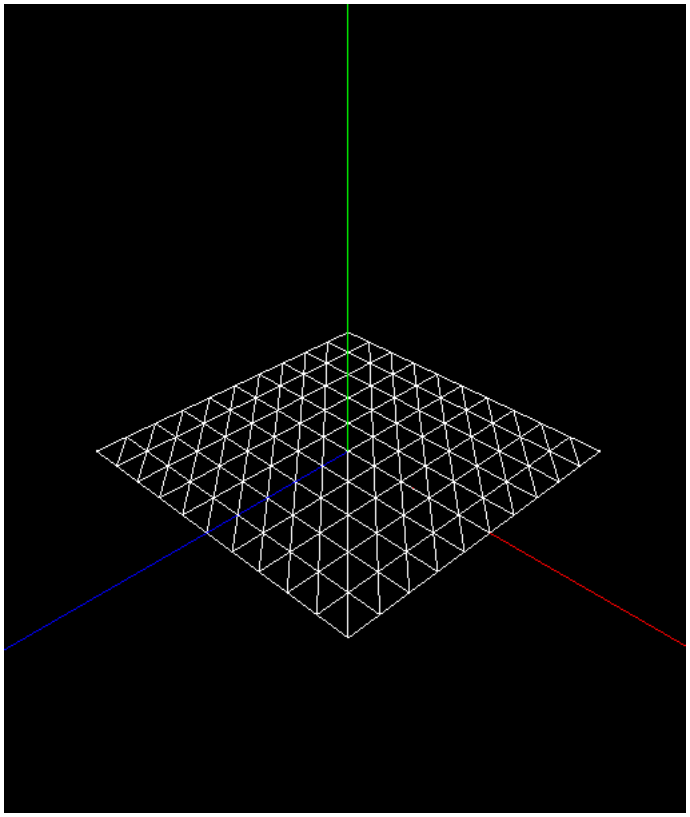

Figure 5: Plano

4.2 Box

Gerada uma box com 1 de length e cada lado dividido 3x3.

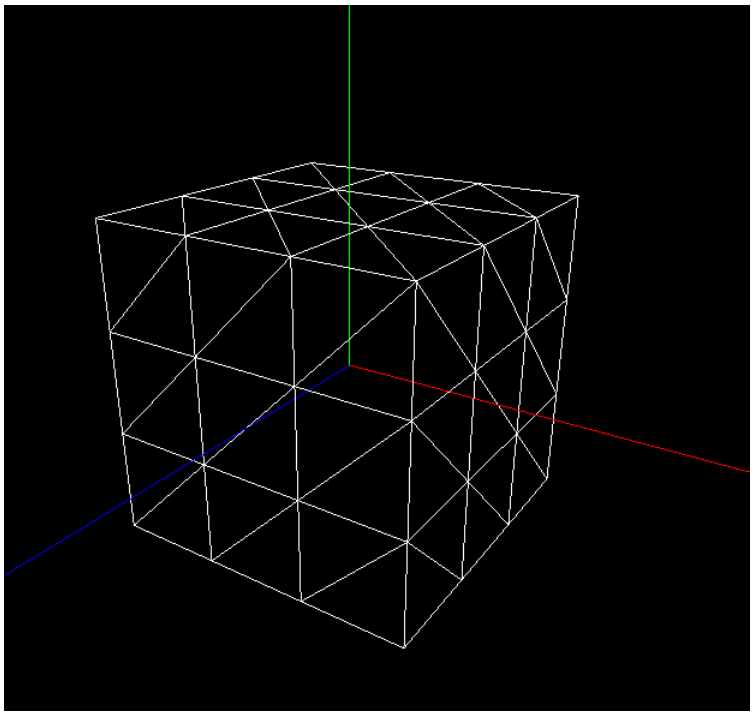

Figure 6: Cubo

4.3 Cone

Gerado um cone com raio 1, 2 de altura, 8 slices e 4 stacks.

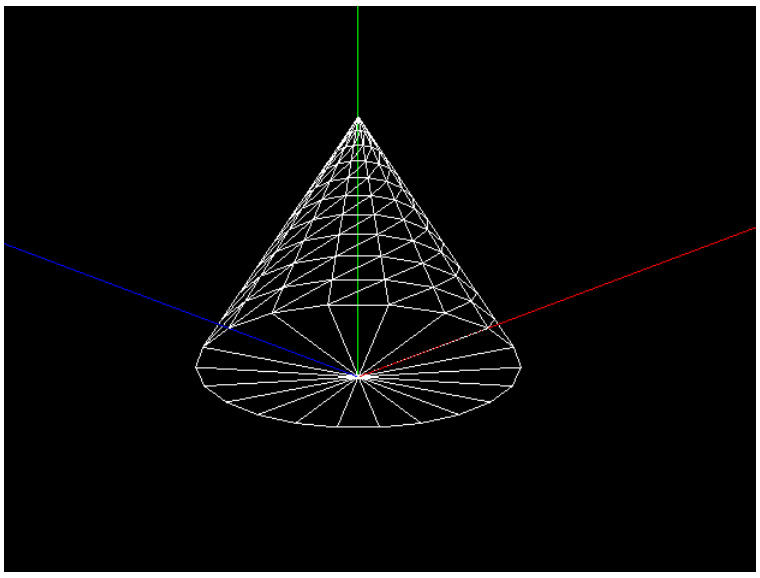

Figure 7: Cone

4.4 Sphere

Gerada uma sphere com raio 1, 10 slices e 10 stacks.

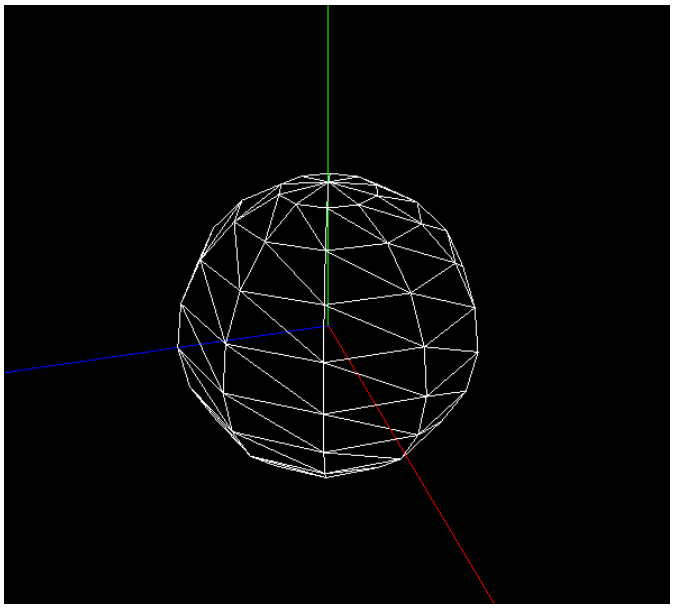

Figure 8: Esfera

4.5 Cylinder

Por fim geramos um cylinder com raio 1, 3 de altura, 20 slices e 4 stacks.

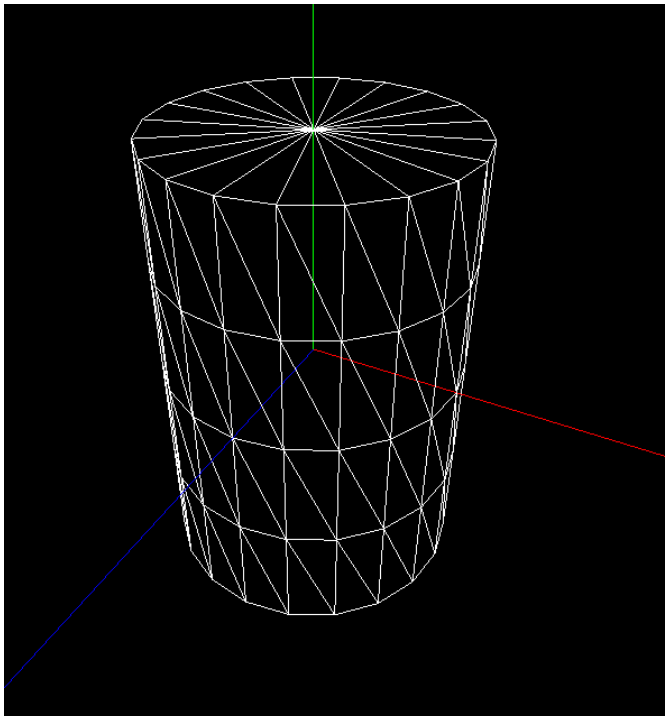

Figure 9: Cilindro

5 Conclusão e Trabalho Futuro

Nesta última parte da primeira fase constatamos que este projeto foi-nos útil para aprofundar o conhecimento adquirido em aula, nomeadamente na linguagem de programção C++ e no funcionamento do OpenGL. Além disso, desenvolvemos novas aptidões ao trabalhar com o XML.

Enquanto grupo, conseguimos distribuir bem o trabalho entre todos. Ajudamo-nos mutuamente e, de forma geral, o grupo teve um aproveitamento positivo, visto que o trabalho entregue cumpre todos os requisitos propostos pelos docentes da unidade curricular.