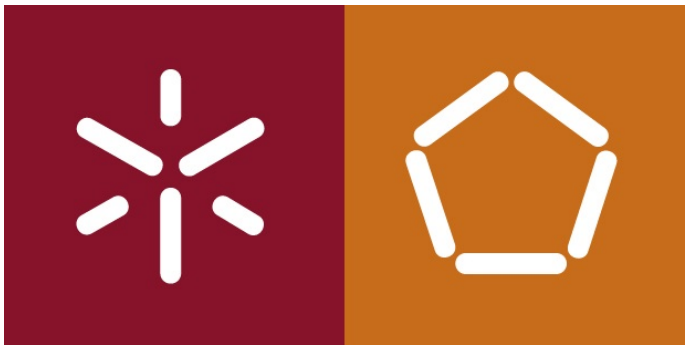

1º Trabalho Prático de Computação Gráfica

Universidade do Minho

3º Ano de MIEInf

Grupo 24

Pedro Medeiros A80580

Davide Matos A80970

Artur Ribeiro A82516

Helena Martins A82500

Conteúdo

1 Introdução

No âmbito da disciplina de Computação Gráfica foi-nos proposto a realização de um motor de representação de modelos gráficos 3D e exemplos que conseguissem demonstrar o seu potencial.

Este trabalho prático estará dividido em quatro fases, sendo que nesta primeira fase teriamos como objetivo realizar duas aplicações: uma para gerar ficheiros com a informação dos modelos a realizar e o motor, capaz de gerar e mostrar os modelos, a partir da leitura de ficheiros escritos em *XML*, com os parametros necessários para a criação de cada primitiva geométrica, que pode ter mais do que um modelo.

Assim, este projeto foi realizado na linguagem C++, usando a biblioteca GLUT (*OpenGL Utility Toolkit*) e TinyXML2, que nos ajuda no processo de *parsing*.

As funcionalidades implementadas nesta primeira fase do trabalho prático foram:

- **Obrigatórias**
  1. Plano
  2. Caixa com divisórias
  3. Esfera
  4. Cone
- **Extras**
  1. Cilindro
  2. Controlo de Câmara com teclado (*First Person*)
  3. Controlo do estilo do desenho com teclado (linhas, pontos ou preenchido)

2 Organização do Projeto

2.1 Aplicações

O projeto encontra-se separado em duas aplicações, sendo estas o gerador e o motor, tendo a capacidade de se complementar uma à outra. Enquanto o gerador consiste na tradução de primitivas geométricas par um conjunto de pontos armazenados em ficheiros, o motor é uma aplicação que lê ficheiros XML que contêm referencias aos ficheiros criados pelo gerador e os representa recorrendo à biblioteca OpenGL.

2.2 Módulos

2.2.1 Gerador

O módulo generator.cpp é responsável por gerar todas as figuras necessárias através do uso da classe parser.cpp que contêm as estruturas *Ponto* e *pointsStruct*. Este ficheiro, quando executado calcula todos os pontos necessários para formar os triângulos que dão origem à figura desejada e grava-os no ficheiro pretendido, com as dimensões passadas como argumento.

2.2.2 Motor

O módulo engine.cpp está responsável pela interpretação e a representação do ficheiro XML que contém o nome dos ficheiros gerados pelo generator numa modelação 3D. Este ficheiro, deve ser executado com nome do ficheiro XML que deve ser lido.

2.3 Classes

Nesta primeira fase do projeto, apenas definimos a classe Point, que se encontra no ficheiro parser.cpp. Nesta classe é traduzida a representação de um ponto no referencial para código.

2.4 Ferramentas Utilizadas

Neste projeto foram usadas as bibliotecas *OpenGL*, onde assentam as funcionalidades gráficas do projeto, e *TinyXML2*, que ajudou na simplificação do processo de parsing do cenário utilizado para demonstrar as funcionalidades do engine.

3 Primitivas Geométricas

3.1 Plano

Para realizar um plano centrado na origem, paralelo ao eixo xOz, foi necessário desenhar dois triângulos, fazendo um retângulo visível apenas pela parte superior (ambos os triângulos foram desenhados orientados para cima). A função que desenha os vértices dos triângulos do plano recebe como parametros o comprimento e a largura, sendo que para centrar na origem, dividimos por 2. Abaixo poderemos ver uma imagem explicativa da ordem em que os vértices foram desenhados, demonstrando a orientação que o plano terá. Assim, os vértices dos triângulos terão as seguintes coordenadas:

- **Primeiro Triângulo**
- Ponto 1: (comprimento/2, 0, largura/2)
- Ponto 2: (comprimento/2, 0, -largura/2)
- Ponto 3: (-comprimento/2, 0, largura/2)
- **Segundo Triângulo**
- Ponto 1: (comprimento/2, 0, largura/2)
- Ponto 2: (-comprimento/2, 0, -largura/2)
- Ponto 3: (-comprimento/2, 0, largura/2)

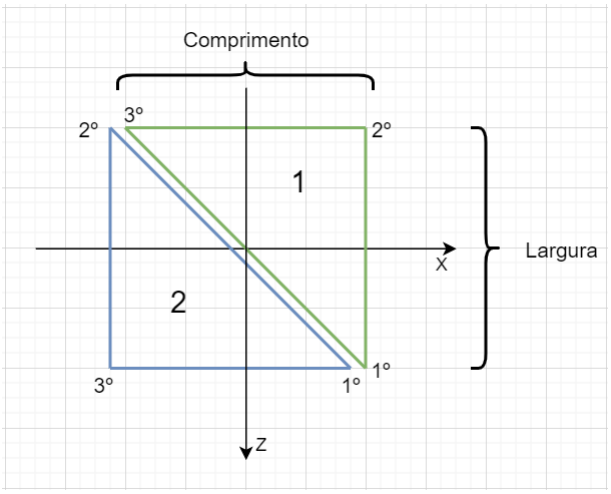

Figura 2: Plano

3.2 Caixa

Para a realização da caixa, centrada na origem, foi necessário desenhar cada uma das seis faces. De modo a exemplificar, vamos explicar o processo de desenhar a face da frente e as outras faces serão um processo análogo. Como se trata da face da frente, a coordenada Z será sempre largura/2 (porque a caixa está centrada na origem). Para as outras faces, será semelhante mas para o eixo ordenado correspondente. Resta então analisar as coordenadas X e Y. A fim de poder implementar divisões na caixa, foi necessário percorrer a face pela sua altura e comprimento, da seguinte forma:

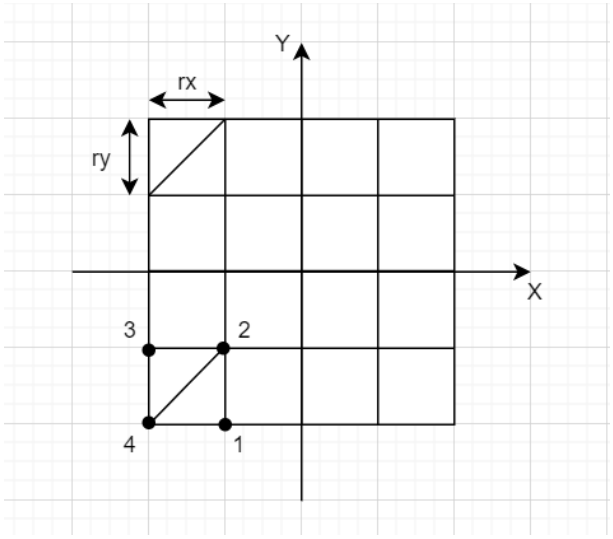

Figura 3: Caixa com 4 divisórias

$$Ponto1 : (-cx + j * rx + rx, -cy + i * ry) \tag{1}$$

$$Ponto2 : (-cx + j * rx + rx, -cy + i * ry + ry) \tag{2}$$

$$Ponto3 : (-cx + j * rx, -cy + i * ry + ry) \tag{3}$$

$$Ponto4 : (-cx + j * rx, -cy + i * ry,) \tag{4}$$

A fim de desenhar com a correta orientação, para cada quadrilátero desenhamos os dois triângulos na ordem 4->1->2 e 4->2->3, respetivamente.

Sendo:

$$cx = comprimento/2$$

$$cy = altura/2$$

$$cz = largura/2$$

$$rx = comprimento/divisoes$$

$$ry = altura/divisoes$$

$$rz = largura/divisoes$$

3.3 Esfera

Para criarmos a esfera, foi necessário dividirmos esta em camadas e fatias. Cada camada é constituida por um determinado número de fatias. Assim, se percorrermos uma camada, esta é constituída por diversos quadriláteros que são formados com a junção de dois triângulos. Para a formação desses mesmos triângulos, foram usadas coordenadas esféricas que foram obtidas pelas seguintes equações:

$$x = R * \sin(\beta) * \sin(\alpha) \tag{5}$$

$$y = R * \cos(\beta) \tag{6}$$

$$z = R * \sin(\beta) * \cos(\alpha), \tag{7}$$

onde R representa o raio da esfera.

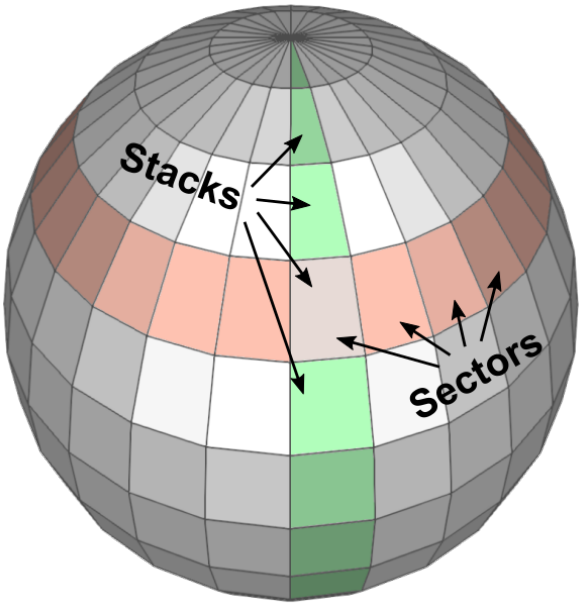

Figura 4: Esfera dividida em slices e stacks

Os pontos do triângulos constituintes da esfera são formados através da variação dos ângulos α e β. O ângulo α representa os ângulos de cada fatia, ou seja, quando pretendemos mudar de fatia, basta incrementarmos este valor. Quanto ao ângulo β, este representa os ângulos formados por cada camada, ou seja, quando pretendemos mudar de camada, basta incrementarmos o valor de β. Estes valores foram calculados da seguinte forma:

$$\alpha = i * \frac{2 * \pi}{n^o fatias} \tag{8}$$

$$\beta = j * \frac{\pi}{n^o camadas}, \tag{9}$$

sendo i a fatia e j a camada correspondentes.

3.4 Cone

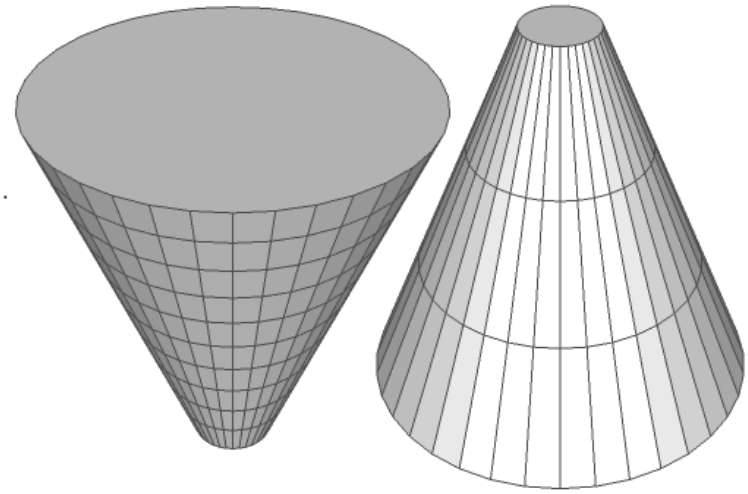

Figura 5: Cone dividido em stacks

Para desenharmos o cone, foi necessário desenhar a sua base e de seguida a superfície lateral. Para a base foi calculado um ângulo *tempangulo* que corresponde ao ângulo de cada fatia *i*.

$$tempangulo = i * \frac{2 * \pi}{n^o fatias} \tag{10}$$

Com um dos pontos de cada triângulo localizado na origem, foi-nos possível desenhar a base do cone bastando incrementar o valor do ângulo *tempangulo* nos restantes pontos:

- $$(R * \sin(tempangulo), 0, R * \cos(tempangulo)) \tag{11}$$
- $$(0, 0, 0) \tag{12}$$
- $$(R * \sin(tempangulo + angslices), 0, R * \cos(tempangulo + angslices)), \tag{13}$$

sendo R o valor do raio inicial e

$$angslices = \frac{2 * \pi}{n^o fatias} \tag{14}$$

Quanto ao desenho da lateral do cone, foi usado o valor *tempradius* para calcular o raio de cada camada, que vai diminuindo até ao topo.

$$tempradius = \frac{R}{n^o camadas}, \tag{15}$$

sendo R o valor do raio da base do cone.

Os pontos dos triângulos que constituem a lateral foram calculados da seguinte forma:

- $$((R - j * tempradius) * \sin(tempangulo), altstacks * j, (R - j * tempradius) * \cos(tempangulo)) \tag{16}$$

- $$((R - j * tempradius) * \sin(tempangulo + angslices), altstacks * j, (R - j * tempradius) * \cos(tempangulo + angslices)) \quad (17)$$
- $$((R - j * tempradius - tempradius) * \sin(tempangulo + angslices), altstacks * j + altstacks, (R - j * tempradius - tempradius) * \cos(tempangulo + angslices)) \quad (18)$$
- $$((R - j * tempradius - tempradius) * \sin(tempangulo), altstacks * j + altstacks, (R - j * tempradius - tempradius) * \cos(tempangulo)) \quad (19)$$

sendo R o raio inicial do cone e *altstacks* a altura de cada camada.

3.5 Cilindro

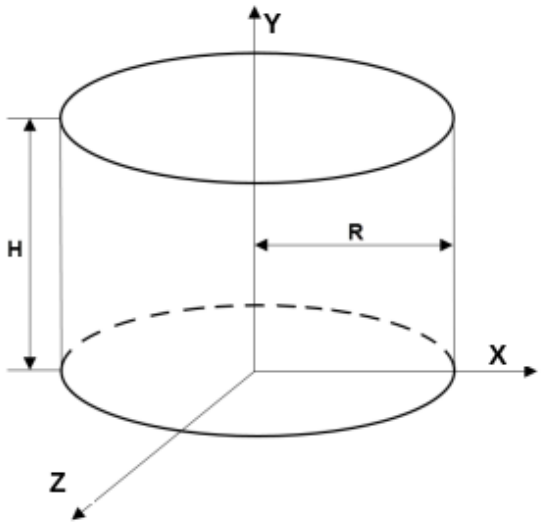

Figura 6: Cilindro

Para criarmos o cilindro tivemos que desenhar as bases e depois a lateral. Para as bases, o raciocínio foi idêntico: como estão divididas em fatias, primeiro calculamos o ângulo para cada fatia i:

$$nangle = \frac{2 * \pi}{n^o fatias} \tag{20}$$

$$angle = i * nangle \tag{21}$$

De seguida, calculamos os pontos do triângulos que formam a base de cima do seguinte modo:

- $$(R * \cos(angle), \frac{altura}{2}, R * \sin(angle)) \tag{22}$$
- $$(0, \frac{altura}{2}, 0) \tag{23}$$
- $$(R * \cos(angle + n_angle), \frac{altura}{2}, R * \sin(angle + n_angle)) \tag{24}$$

Os pontos dos triângulos que formam a base de baixo são calculados de forma idêntica, as únicas diferenças são a ordem pela qual são desenhados e o seu valor de y que é negativo devido ao cilindro estar centrado na origem:

- $$(0, -\frac{altura}{2}, 0) \tag{25}$$
- $$(R * \cos(angle), -\frac{altura}{2}, R * \sin(angle)) \tag{26}$$
- $$(R * \cos(angle + n_angle), -\frac{altura}{2}, R * \sin(angle + n_angle)) \tag{27}$$

Quanto à lateral do cilindro, os triângulos que a constituem são calculados da seguinte forma:

- $$(R * \cos(angle), \frac{altura}{2}, R * \sin(angle)) \tag{28}$$
- $$(R * \cos(angle + nangle), \frac{altura}{2}, R * \sin(angle + nangle)) \tag{29}$$
- $$(R * \cos(angle), -\frac{altura}{2}, R * \sin(angle)) \tag{30}$$
- $$(R * \cos(angle + nangle), -\frac{altura}{2}, R * \sin(angle + nangle)) \tag{31}$$

4 Gerador

4.1 Descrição

No trabalho, todas as figuras apresentadas são criadas à base de de triângulos. Para isso é necessário calcular previamente calcular as coordenadas de todos os pontos dos triângulos que representam a figura. Toda essa atividade era efetuada pelo generator, que está codificado para realizar todos os calculos necessários para a representação dos 3 vertices dos triângulos que constituem o objecto pretendido.

4.2 Funcionalidades

Como grupo, desenvolvemos várias formas geométricas 3D diferentes, podendo estas ser chamadas pelo utilizador através do generator. As opções possíveis são as seguintes:

```
preacher@Preacher-GL552VX:~/CG-Project/build$ ./generator -help
┌---------------------------------------------- PRIMITIVES ----------------------------------------------┐
|                                                                                                        |
|                                                                                                        |
|    ▸ PLANE                                                                                             |
|        $ ./generator plane <COMPRIMENTO> <LARGURA> <OUTPUT_FILE>                                       |
|                                                                                                        |
|    ▸ BOX                                                                                               |
|                                                                                                        |
|        $ ./generator box <COMPRIMENTO> <LARGURA> <ALTURA> [<DIVISOES>] <OUTPUT_FILE>                   |
|                                                                                                        |
|    ▸ SPHERE                                                                                            |
|                                                                                                        |
|        $ ./generator sphere <RAIO> <SLICES> <STACKS> <OUTPUT_FILE>                                     |
|                                                                                                        |
|    ▸ CONE                                                                                              |
|                                                                                                        |
|        $ ./generator cone <RAIO_BASE> <ALTURA> <SLICES> <STACKS> <OUTPUT_FILE>                         |
|                                                                                                        |
|    ▸ Cylinder                                                                                          |
|                                                                                                        |
|        $ ./generator cylinder <raio> <altura> <Slices>  <OUTPUT_FILE>                                  |
|                                                                                                        |
|                                                                                                        |
└--------------------------------------------------------------------------------------------------------┘
```

Figura 7: Menu *help* do generator

- **Plane:** É criado um plano paralelo ao plano xOz.
- **Box:** É criada uma caixa. As suas dimensões (comprimento, largura, altura e divisões) são dadas como argumento. As divisões dizem respeito ao número de quadrados por face.
- **Sphere:** Cria uma esfera onde as suas dimensões são dadas como argumento, o raio, o número de slices (divisão vertical) e o número de stacks (divisão horizontal).
- **Cone:** Cria um cone onde é dado como argumento o seu raio, altura, número de slices e número de stacks.
- **Cylinder**: É criado um cilindro. As suas dimensões (raio, altura, número de slices e número de stacks) são passadas como argumento.

5 Motor

5.1 Descrição

O motor funciona a base de ler um ficheiro XML que nesta fase contém apenas a informação de quais os modelos previamente gerados pelo generator que deve desenhar.
Também possui uma câmera *first person* com o objetivo de navegar no espaço e observar os modelos de diferentes prespectivas.
Para além disso, permite-nos alterar entre modos de desenho (linhas, pontos ou prenenchido). Tal como o generator , o engine possui também a funcionalidade *-help* que escreve no terminal os comandos da aplicação.

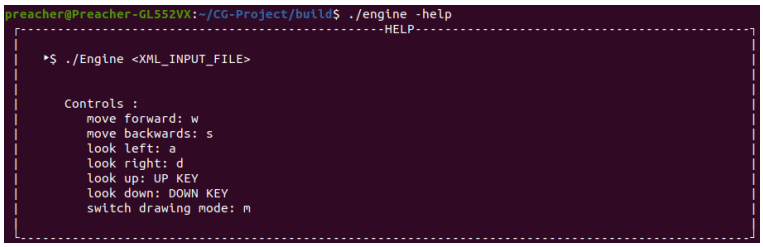

Figura 8: Engine -Help

5.2 Parser

o parser funciona lendo o ficheiro *XML* e guardando num vector de strings os nomes dos ficheiros correspondendo aos modelos pertendidos. Após esta fase, o parser irá ler cada ficheiro .3d (que contém os pontos) e guarda-os num único vector de pontos (descrito na secção seguinte) por ordem de leitura, todos os pontos, de todos os modelos lidos.

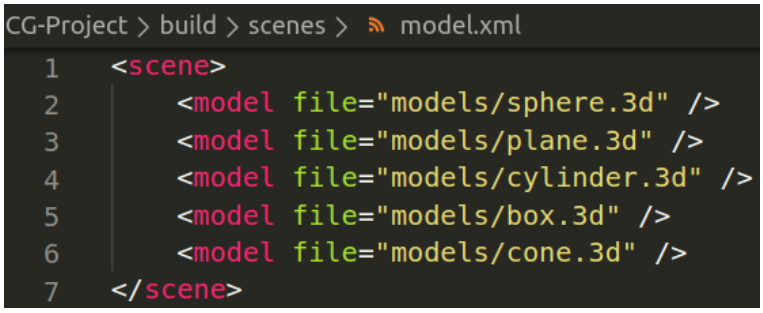

Figura 9: Exemplo do ficheiro XML

5.3 Estruturas

.
Foi criada uma estrutura Ponto que são simplesmente 3 floats correspondendo as coordenadas x , y e z do ponto em questão. Por fim utilizamos o *vector* do c++ para termos em memória um vector com os pontos pertendidos.

```
typedef struct {
  float x;
  float y;
  float z;
} Ponto;

typedef std::vector<Ponto> pointsStruct;
```

Figura 10: Estrutura de pontos

6 Modelos

Plano de 10 comprimento e 7 largura, paralelo ao eixo xOz gerado.

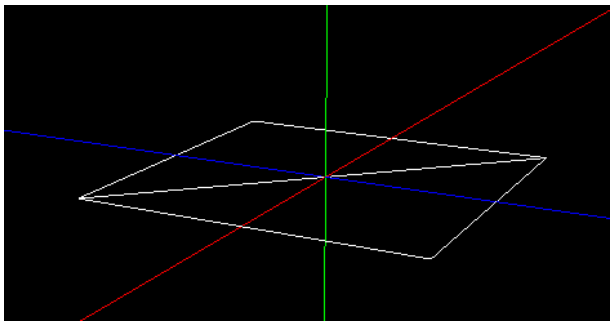

Figura 11: Plano

Caixa com 10 de comprimento, 10 de largura, 10 de altura e 5 divisões.

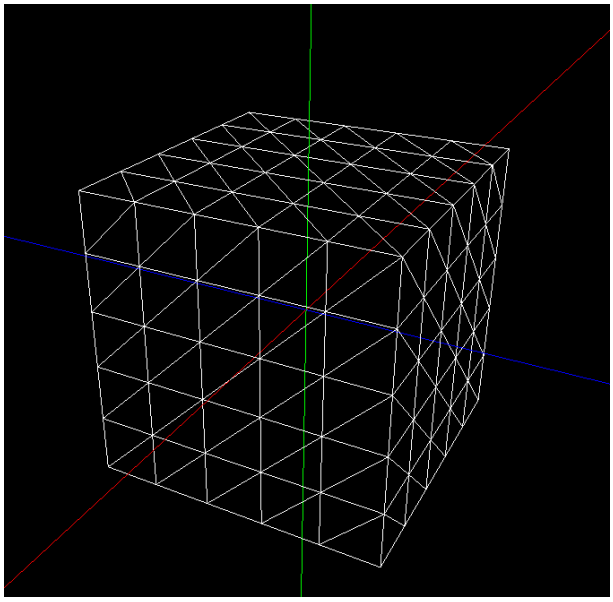

Figura 12: Caixa com 5 divisões

Esfera com 10 de raio, 50 slices e 50 stacks.

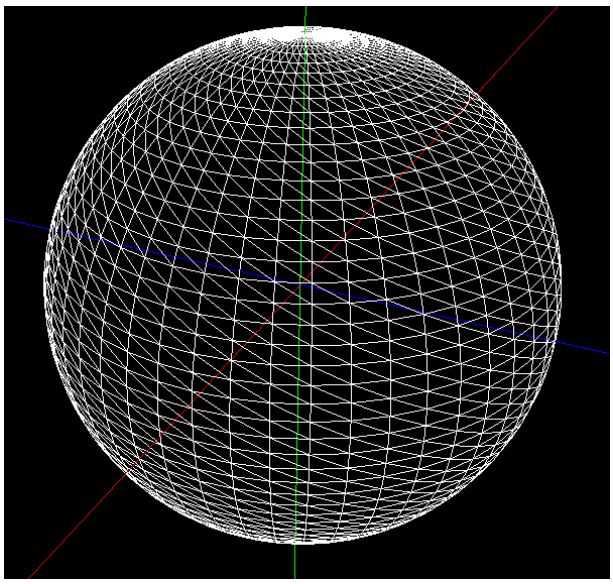

Figura 13: Esfera

Cone com 5 de raio, 10 de altura, 50 slices e 50 stacks.

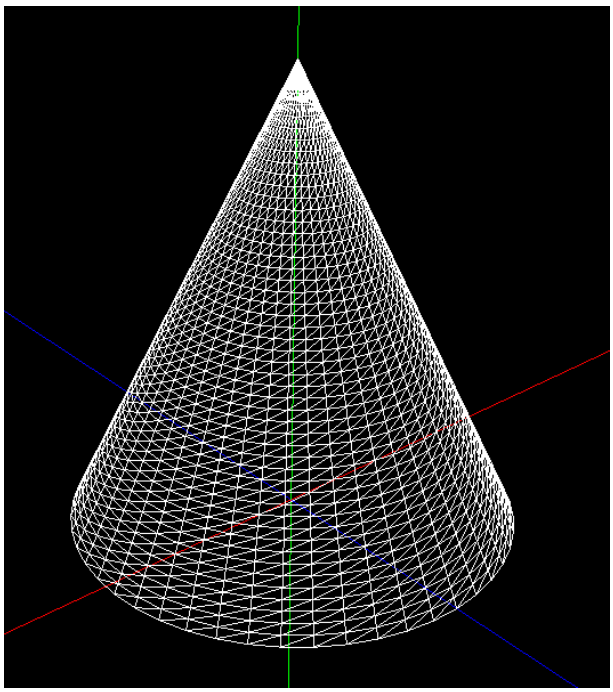

Figura 14: Cone

Cilindro de 5 de raio, 10 de altura e 50 slices.

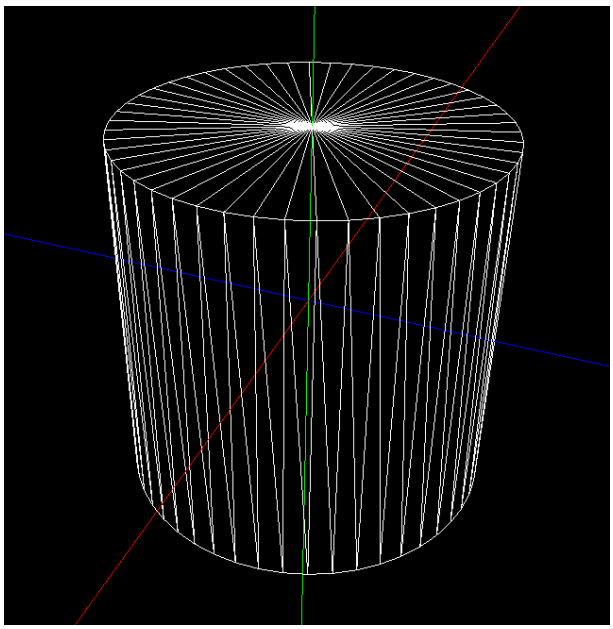

Figura 15: Cilindro

7 Conclusão

Com este projecto todos os elementos do grupo expandiram os seus conhecimentos de Computação gráfica, tanto no desenho dos modelos, como na manipulação de coordenadas esféricas assim como na programação em C++.