UNIVERSIDADE DO MINHO
MIEI

COMPUTAÇÃO GRÁFICA

Primeira Fase - Primitivas Gráficas

Grupo:
João Nunes - a82300
Shahzod Yusupov - a82617
Luís Braga - a82088
Luís Martins - a82298

Braga, Portugal
9 de Março de 2019

Conteúdo

1 Introdução

No âmbito da unidade curricular de Computação Gráfica foi proposta, numa primeira fase, que se realizassem dois sistemas. Um primeiro, que guardasse num ficheiro informações relativas a uma figura que se pretende desenhar futuramente, usando o **GLUT**. E um segundo mecanismo, que, lendo de um ficheiro, escrito em xml, configurações, desenhará o modelo segundo as indicações geradas anteriormente, obtendo-se assim uma figura.

Neste projecto as figuras abordadas serão um plano, caixa, uma esfera e por fim o cone.

2 Estrutura do projecto

O projecto centra-se à volta de duas classes principais sendo ela o *generator* e o *engine*, estas duas classes serão as classes responsáveis por fazer a maior parte do trabalho.

Para além dessas duas classes, foram criadas também outras classes auxiliares tais como o *Point*, que representa uma estrutura que guarda os valores de um ponto segundo o seu valor de x, y e z, e o *Struct* cuja finalidade será guardar os pontos segundo essa mesma estrutura de dados. Foi criado também o *parser*, para ler os ficheiros XML.

2.1 Generator

Neste projecto, o *generator* é responsável pelo cálculo dos pontos necessários para os desenhos dos triângulos que compõem a figura em questão. Como tal, foi necessário adoptar certos e determinados algoritmos para serem gerados esses pontos.

A classe *Point* juntamente com a *Struct* irão trabalhar lado a lado com o *generator*, na *Struct* pode-se encontrar os algoritmos necessários para gerar os pontos essenciais à construção das figuras pretendidas.

Após ter sido feito os cálculos dos pontos, o gerador irá escrever num ficheiro todos esses mesmos pontos relativos ao desenho da figura em questão. Cada linha irá representar um vértice da figura, onde estão presentes as coordenadas x, y e z separadas por espaços.

No generator, portanto, estão contemplados os cálculos fundamentais para gerar as seguintes figuras:

- Plane
- Box
- Sphere
- Cone

2.1.1 Plane

Antes de serem gerados os pontos para o desenho do plano, é necessário passar como argumento um **size**, ou seja, tamanho do lado. A ideia principal desta figura passa pelo desenho de dois triângulos com dois vértices coincidentes (**A** e **B**), centrados na origem, e como se trata de um plano em XZ, a coordenada y dos pontos será zero em todos.

O tamanho previamente referido no parágrafo anterior, como se pretende que este fique centrado na origem, é necessário dividir o valor passado como argumento por dois de modo a obter as coordenadas X e Z necessárias para formar ambos os triângulos de uma maneira uniformizada.

O raciocínio por detrás desta linha de pensamento poderá ser verificado no seguinte esquema:

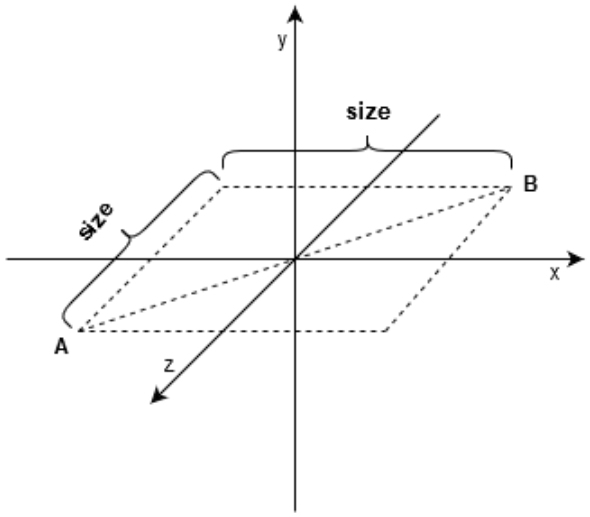

Figura 2.1: Plano desenhado no referencial cartesiano.

2.1.2 Box

A box consiste num paralelepípedo com 3 parâmetros : comprimento(cX), altura(cY), largura (cZ), existindo ainda outro parâmetro adicional designado por divisão. Uma box com 3 divisões refere se a um paralelepípedo em que cada face está dividida numa matriz (3x3).

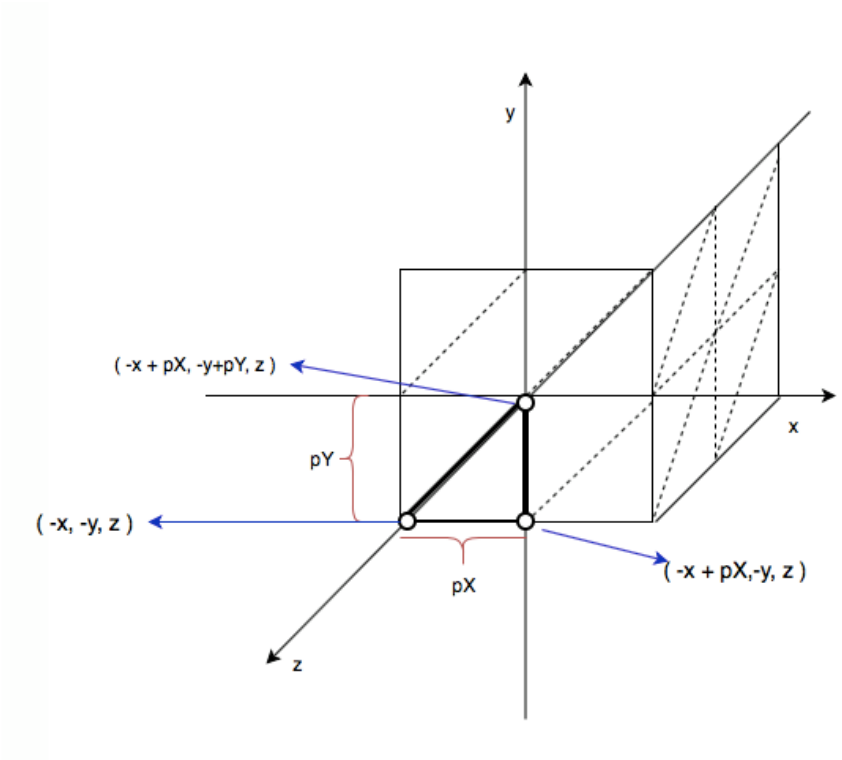

Figura 2.2: Face frontal e lateral da box desenhada no referencial cartesiano.

Como se pode observar na figura,o raciocínio por detrás da construção de cada face da box é o mesmo que da construção do plano (2 triângulos que têm 2 pontos em comum), diferindo apenas quando se adicionam divisões. Para o auxílio da construção da box foram implementadas 3 variáveis (pX,pY,pZ) que se obtêm dividindo o valor dos parâmetros cX,cY e cY pelo número de divisões. Neste exemplo em concreto o número de divisões é 2 e supondo que cX=cY=cZ=6, então pX,pY e pZ tomarão o valor de 3.

Escolhendo, por exemplo, a face voltada para a direita ,em que o valor de x é constante em todos os vértices a construção dos triângulos é feita da direita para a esquerda e de baixo para cima, ou seja , primeiramente constrói-se os triângulos da face do canto inferior direito, seguido do canto inferior esquerdo e só depois do canto superior direito e canto superior esquerdo. Isto obtém-se através da fórmula:

Para todo o i e j menor que o número de divisões
(x,-y + (i*pY),-z + (j*pZ))
(x,(-y+pY) + (i*pY),-z +(j*pZ))
(x,-y + (i*pY),(-z+pZ) + (j*pZ))
(x,(-y+pY) + (i*pY),-z + (j*pZ))
(x,(-y+pY) + (i*pY),(-z+pZ) + (j*pZ))
(x,-y + (i*pY),(-z+pZ) + (j*pZ))

Em que o j percorre em função do eixo dos z's e o i incrementa a altura no final de percorrer a linha, em que x=cX/2 ,y=cY/2 e z=cZ/2.

2.1.3 Sphere

No caso da esfera, é necessário passar como argumento o número de fatias (**slices**), o número de camadas (**stacks**), e por fim o raio da esfera (**radius**), tal como é possível observar no seguinte esquema:

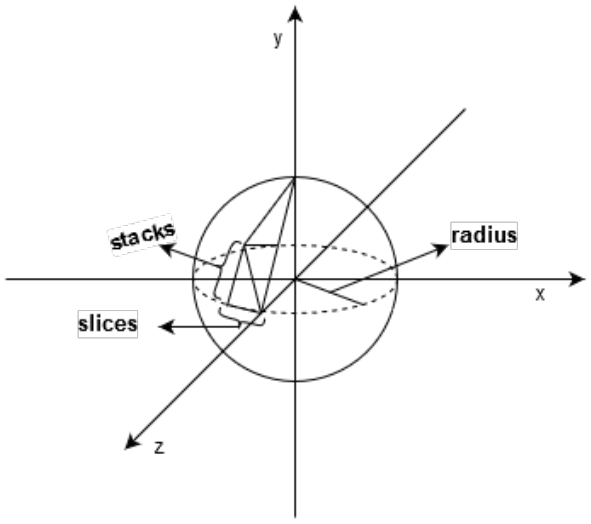

Figura 2.3: Esfera desenhada no referencial cartesiano.

São utilizadas como base, para gerar os pontos da esfera, as seguintes coordenadas cartesianas:

$$z = r * cos(\beta) * cos(\alpha)$$
$$x = r * cos(\beta) * sen(\alpha)$$
$$y = r * sen(\beta)$$

Onde temos que:

$$\beta = \pi / stacks$$
$$\alpha = 2\pi / slices$$
$$r = radius$$

O problema de desenhar a esfera foi simplificado partindo esta em três partes diferentes, o topo, a base e por fim o meio. O algoritmo implementado pelo grupo, na primeira iteração do ciclo interior em apenas dois ciclos constrói os triângulos na primeira camada. Os triângulos que correspondem à última camada, ou seja, da base, são construídos apenas na última iteração do ciclo.

No meio da esfera, são aplicadas operações, deslocando o ângulo correspondente ao β, calculando assim os próximos pontos que irão construir a esfera. O mesmo princípio também se aplica em relação ao α.

Para além disso, partindo do ponto de partida, em cada slice, a esfera é construída utilizando uma abordagem top down, slice a slice, de stack em stack, onde as seguintes variáveis são actualizadas:

$$x1 = radius * cos((j+1) * beta)$$
$$x2 = radius * cos(i * alfa) * sin((j+1) * beta)$$
$$x3 = radius * sin(i * alfa) * sin((j+1) * beta)$$
$$x4 = radius * cos((j+1) * beta)$$
$$x5 = radius * cos((i+1) * alfa) * sin((j+2) * beta)$$
$$x6 = radius * cos((j+2) * beta)$$
$$x7 = radius * sin((i+1) * alfa) * sin((j+2) * beta)$$

2.1.4 Cone

Para gerar os pontos necessários para desenhar o cone são precisos como argumentos: o raio da base (radius); a altura do cone (height); o número de fatias (slices) iguais em que se vai dividir o círculo da base e, por fim, o número de camadas (stacks) que vamos usar na construção ao longo do eixo y, por outras palavras, o número de "andares"em que vamos dividir a altura.

De forma a entender o processo de criação deste modelo solicita-se que se preste atenção à seguinte figura:

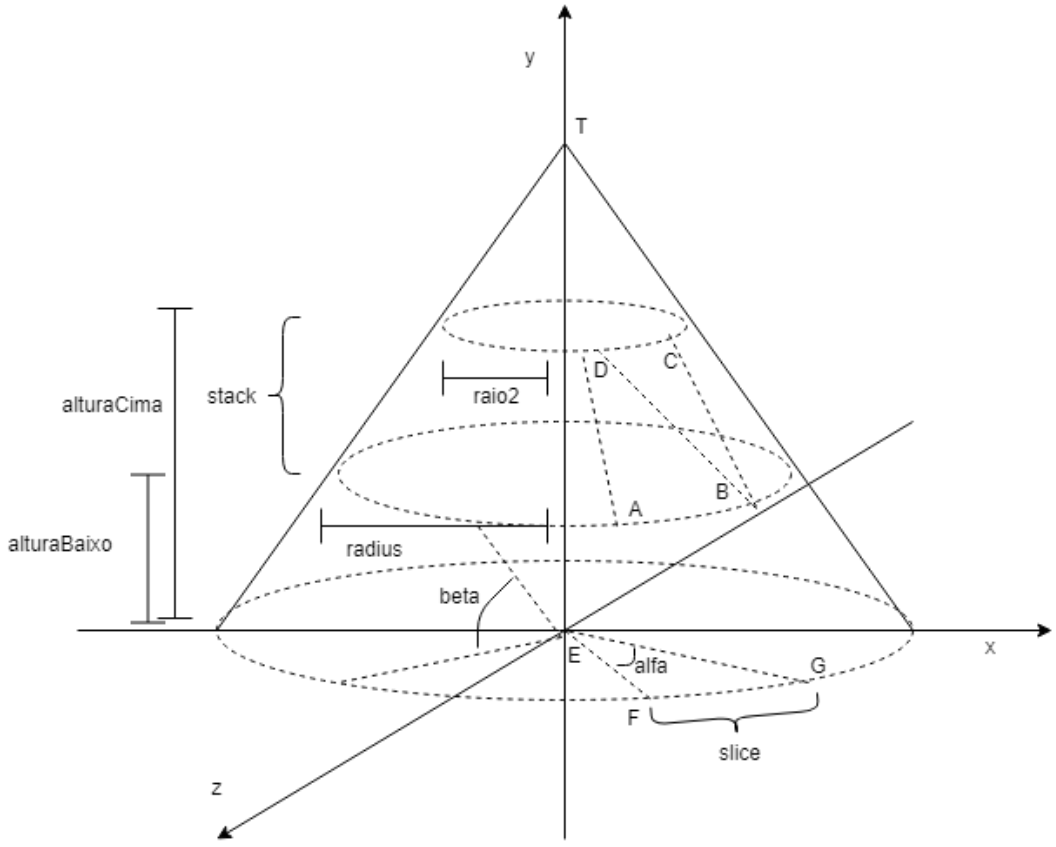

Figura 2.4: Cone desenhado no referencial cartesiano.

Pode-se dividir o processo de criação de pontos do cone em três fases distintas:

1. Base;
2. Lados;

3. Topo.

O cone é construído primeiramente stack a stack e em cada stack, slice a slice.

Começamos por calcular o ângulo α, a altura de cada stack e a tangente de β, como demonstrado abaixo.

$$\alpha = 2\pi/slices$$
$$alturaStack = height/stacks$$
$$\tan\beta = height/radius$$

Finda esta preparação, o algoritmo determina os pontos da primeira camada (base) e depois das fatias dessa camada e assim sucessivamente.

Por camada, calcula-se a "alturaCima"e o "raio2"(figura 2.3). De seguida, para essa stack, calculam-se os slices. Se estivermos na primeira camada calcula-se os pontos da base e lados da stack e então temos os seguintes pontos:

Para a base:
$$E \rightarrow (0, 0, 0)$$
$$G \rightarrow (x1, 0, z2)$$
$$F \rightarrow (x1, 0, z1)$$

Para os lados:
$$F \rightarrow (0, 0, 0)$$
$$G \rightarrow (x1, 0, z2)$$
$$A \rightarrow (x3, alturaCima, z3)$$

$$G \rightarrow (x1, 0, z2)$$
$$B \rightarrow (x4, alturaCimas, z4)$$
$$A \rightarrow (x3, alturaCima, z3)$$

Numa stack intermédia (entre base e topo), obtemos:

Os lados:
$$A \rightarrow (x1, alturaBaixo, z1)$$
$$B \rightarrow (x2, alturaBaixo, z2)$$
$$D \rightarrow (x3, alturaCima, z3)$$

$$B \rightarrow (x2, alturaBaixo, z2)$$
$$C \rightarrow (x4, alturaCima, z4)$$
$$D \rightarrow (x3, alturaCima, z3)$$

Para o topo:

O Topo:
$$D \rightarrow (x1, alturaBaixo, z1)$$
$$C \rightarrow (x2, alturaBaixo, z2)$$
$$T \rightarrow (0, height, 0)$$

É de notar que todos estes pontos estão coerentes com o código, incluindo a ordem de criação.

Resta-nos apenas referir como são as actualizadas em cada slice as variáveis x1,x2... utilizadas nos pontos descritos acima:

```
alfa = alfa * sliceAtual;
alfaIMais1 = alfa * (sliceAtual + 1);

x1 = radius * sin(alfa);
z1 = radius * cos(alfa);

x2 = radius * sin(alfaIMais1);
z2 = radius * cos(alfaIMais1);

x3 = raio2 * sin(alfa);
z3 = raio2 * cos(alfa);

x4 = raio2 * sin(alfaIMais1);
z4 = raio2 * cos(alfaIMais1);
```

2.2 Engine

O *engine* tem como principal objectivo a leitura dos ficheiros *XML* e desenhar as figuras contidas nesses mesmos ficheiros. De modo a auxiliar esta mesma leitura dos ficheiros, foi criada uma outra classe *parser* capaz de extrair a informação dos ficheiros *XML*. De modo a auxiliar a construção deste *parser*, foi utilizada uma livraria, o *tinyxml2*, com uma plenitude de funções cujo intuito é ajudar na construção de um *parser* simples, pequeno e eficaz.

Portanto, o *engine* abre também os ficheiros, e guarda a lista de pontos contidos em cada um deles, para serem usados no eventual desenho desses mesmos. É necessário percorrer todas as linhas do ficheiro, e guardar cada entrada como um vértice (x,y,z). Devido a este acontecimento, foi necessária a criação de uma estrutura cujo intuito é armazenar estes pontos, como tal foi criada a estrutura *Struct*. Cada entrada do ficheiro é transformada num objeto da classe *Point*, de seguida esse *Point* é adicionado à lista de pontos, ou seja, a *Struct*.

Tendo já os pontos necessários para o desenho dos triângulos que compõem as figuras, a próxima fase corresponde então ao desenho das figuras. Como tal, é necessário percorrer a estrutura supracitada e evocar a função do *OpenGL* usada para desenhar os vértices que compõem os triângulos da figura pretendida, o *glVertex3f*.

Agora sim, já se torna possível visualizar as figuras, consoante o input desejado, figuras essas que vão ter o seguinte aspecto.

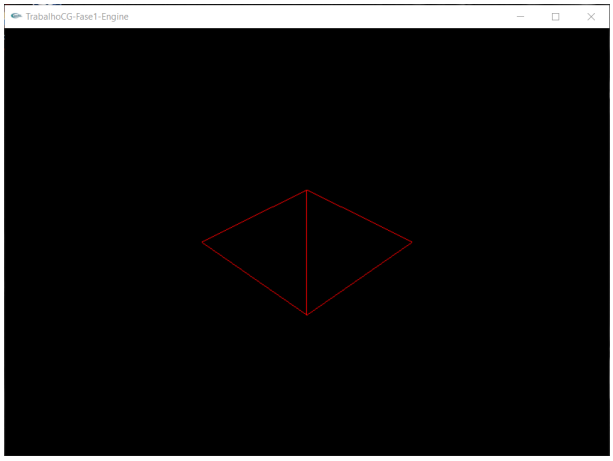

Figura 2.5: Plano 3D.

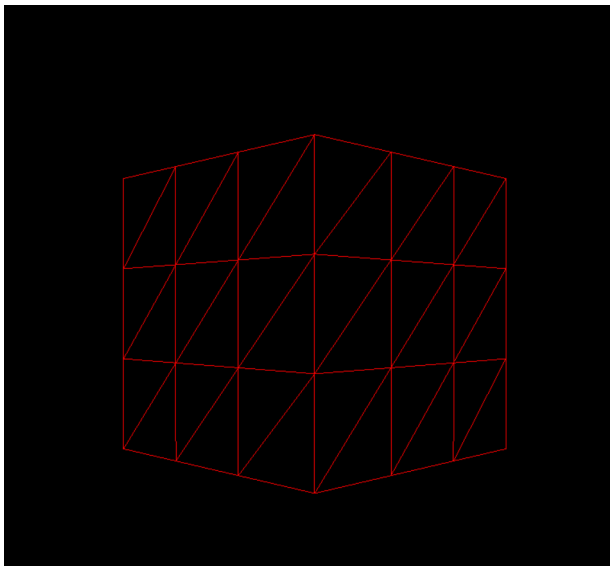

Figura 2.6: Box 3D.

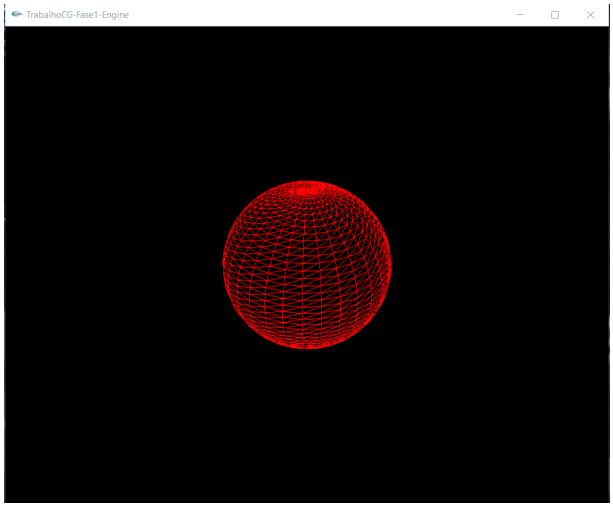

Figura 2.7: Esfera 3D.

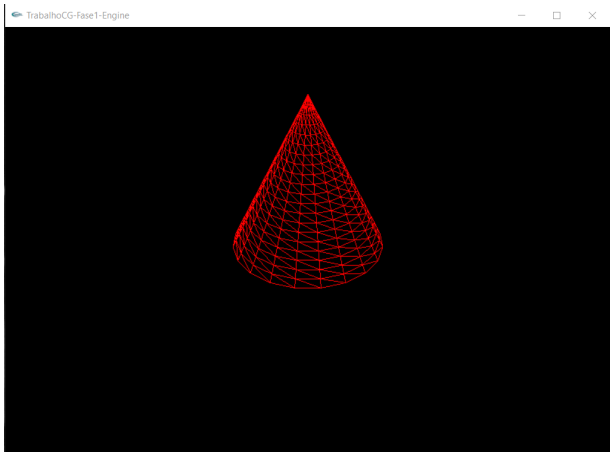

Figura 2.8: Cone 3D.

3 Comando help

Adicionou-se às funcionalidades o seguinte comando *help*, que poderá ser invocando usando a extensão "*-h*". O *help* tem como intuito apresentar um manual de ajuda para os utilizadores, onde é possível visualizar os comandos disponíveis e os seus argumentos.

Este comando está disponível tanto para o generator como para o engine, como se pode verificar nas seguintes figuras.

```
>generator.exe -h
```

Figura 3.1: Mecanismo help no generator.

```
*--------------------------HELP--------------------------*
|                                                        |
|            Modo de utlizacao:                          |
| $ generator.exe figura [dimensoes] ficheiro(.3d)       |
|                                                        |
|         Figuras:                                       |
|               -plane :                                 |
|                     Dimensoes:                         |
|                           -tamanho.                    |
|               -box :                                   |
|                     Dimensoes :                        |
|                           -X Y Z divisoes(opcional).   |
|               -sphere :                                |
|                     Dimensoes:                         |
|                           -Raio Fatias Pilhas          |
|               -cone :                                  |
|                     Dimensoes:                         |
|                           - Raio Altura Fatias Pilhas. |
|               -cylinder :                              |
|                     Dimensoes:                         |
|                           -Raio Altura Fatias.         |
|                                                        |
|         Exemplo de utilizacao:                         |
|           $ generator.exe sphere 1 10 10 sphere.3d     |
|                                                        |
*--------------------------HELP--------------------------*
```

Figura 3.2: Manual de ajuda do generator.

```
>engine.exe -h
```

Figura 3.3: Mecanismo help no engine.

```
*>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>HELP<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<*
|                                                                             |
|   Modo de utilizacao:                                                       |
|                       engine.exe (ficheiro xml)                             |
|   Nota: Os ficheiros 3d tem de estar gerados (pelo generator).              |
|                                                                             |
|   Exemplo de utilizacao:                                                    |
|                          engine.exe plane.xml                               |
|                                                                             |
|         Teclas:      ->       Respetivas acoes:                             |
|     (seta cima)      ->  Roda a posicao da camera para cima                 |
|     (seta baixo)     ->  Roda a posicao da camera para baixo                |
|     (seta esquerda)  ->  Roda a posicao da camera para a esquerda           |
|     (seta direita)   ->  Roda a posicao da camera para a direita            |
|            w         ->  Muda a cor da figura para branco                   |
|            r         ->  Muda a cor da figura para vermelho                 |
|            b         ->  Muda a cor da figura para azul                     |
|            g         ->  Muda a cor da figura para verde                    |
|            p         ->  Muda o formato da figura para ponto                |
|            l         ->  Muda o formato da figura para linha                |
|            f         ->  Preenche a figura                                  |
|            1         ->  Mostra o exterior                                  |
|            2         ->  Mostra o interior                                  |
|            +         ->  Aproxima a camara do ponto em foco                 |
|            -         ->  Afasta a camara do ponto em foco                   |
|                                                                             |
*>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>HELP<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<*
```

Figura 3.4: Manual de ajuda do engine.

4 Extras

Além do que era pretendido ser elaborado para esta primeira fase do trabalho prático, foram também elaborados alguns extras de modo a adicionar ainda mais funcionalidades à aplicação.

Para além das figuras desejadas, foi adicionada também uma primitiva gráfica para desenhar um cilindro, onde o utilizador, caso assim o escolha, terá que passar como input para o desenho deste mesmo, o **radius**, o número de **stacks** e por fim o número de **slices**. No final de todo o processo associado ao desenho da figura o cilindro terá o seguinte aspeto:

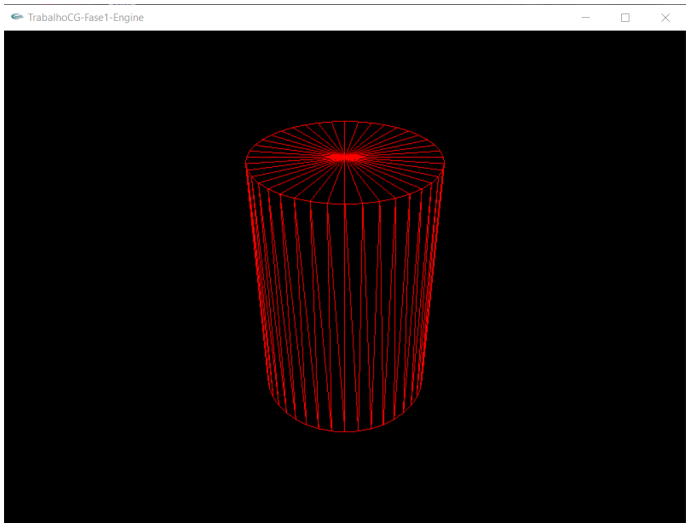

Figura 4.1: Cilindro 3D.

Com o propósito de melhorar a perspetiva sobre a própria figura, através da utilização de certas teclas do teclado, é possível mudar a posição da câmara, mudar o formato de desenho da figura para ponto, linha ou até mesmo sólido e ver o interior do modelo. Todos estes comandos mencionados encontram-se expostos na figura 3.4 do capítulo anterior.

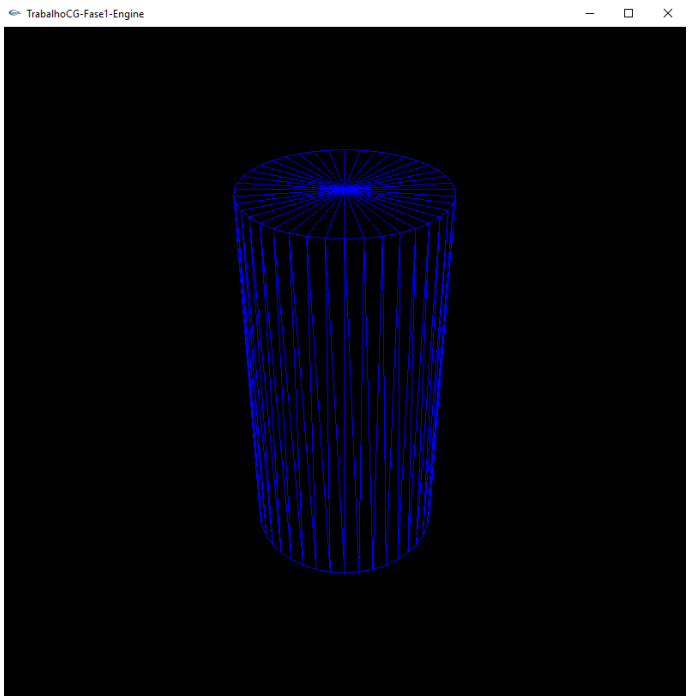

Figura 4.2: Cilindro 3D com cor mudada para azul após ter sido pressionada a tecla b.

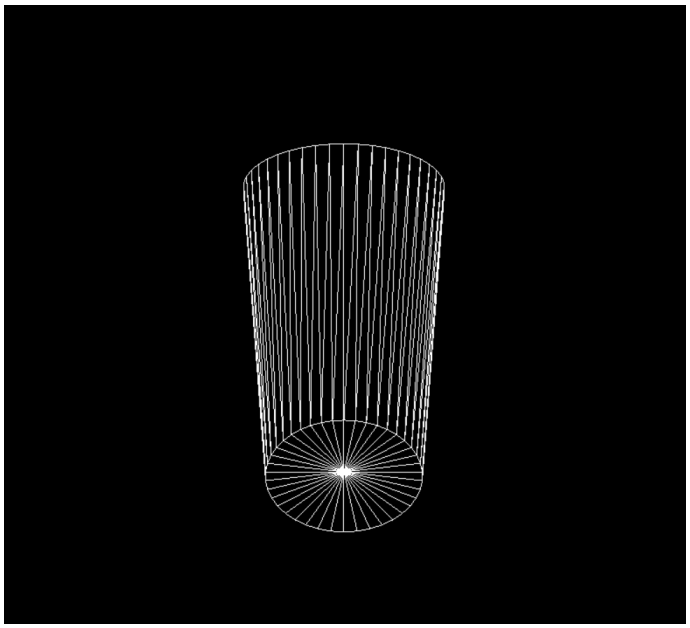

Figura 4.3: Interior do cilindro.

Portanto, com estas funcionalidades implementadas, é então possível verificar a correta representação das figuras desenhadas.

5 Conclusão

A elaboração desta primeira fase permitiu com que o grupo consolida-se conhecimentos relativos ao *OpenGL* e *GLUT* e, ao mesmo tempo, a linguagem de programação de C++, e até mesmo relembrar certos aspetos de geometria. Sendo todos esses aspetos importantíssimos na medida em que permite ao grupo obter experiência no ramo da computação gráfica.

Todos os requisitos propostos nesta fase foram atingidos, tendo todos sido implementadas com sucesso. O que não significa que não tenham ocorridos alguns *stumbling blocks* pelo caminho, tal como, a esfera e o cone devido à complexidade da implementação das coordenadas cartesianas e a divisão por camadas.

Contudo, o grupo espera que o sucesso obtido nesta fase do projeto se projete para as sucessivas fases dos trabalhos práticos seguintes.