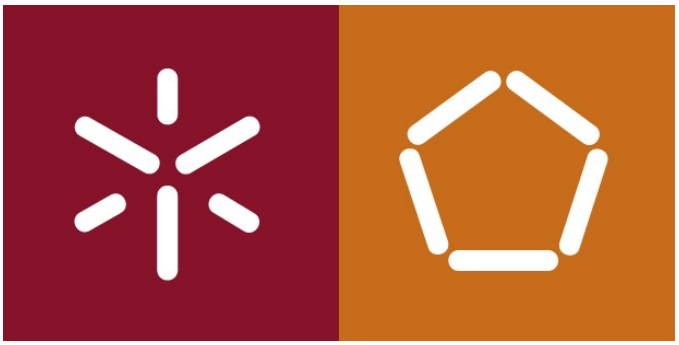
Universidade do Minho

Mestrado Integrado em Engenharia Informática



Computação Gráfica

Relatório do Trabalho Prático 1ªFase

Graphical Primitives

Grupo 36

Carlos Gomes
A77185

Nuno Silva
A78156

9 de Março de 2019

Conteúdo

Capítulo 1

Introdução

1.1 Contextualização

Foi nos proposto, através da unidade curricular Computação Gráfica, a criação de um mecanismo 3D baseado num cenário gráfico, mediante do qual teríamos de utilizar várias ferramentas, entre as quais C++ e OpenGl.

Nesta primeira de quatro fases, o objetivo enquadrava-se na criação de algumas primitivas gráficas.

1.2 Resumo

Para esta 1ª fase, foi necessária a criação de de duas aplicações que serão essenciais para o correto funcionamento desta fase:

- **Gerador** (*Generator*): O gerador irá receber argumentos e com esses argumentos vai criar um ficheiro .3D onde irá ter todos as coordenadas necessárias para criar o sólido que foi solicitado no argumento.

- **Motor** (*Engine*): O motor vai receber, como argumento, o caminho do ficheiro com formato XML. Nesse mesmo ficheiro, vai ter um elemento que se chama "file", esse file vai ser a figura criada pelo gerador.

As primitivas desenvolvidas nesta fase inicial são o Cone, Esfera, Plano e Paralelepípedo, sendo o objetivo desta etapa gerar estas mesmas primitivas gráficas em GLUT.

Também foi desenvolvido uma primitiva extra, o cilindro.

Capítulo 2

Arquitetura do Código

Após uma análise geral do problema e de várias opções de desenvolvimento, foi decidido implementar duas aplicações (**generator** e **engine**) em C++ que nos proporciona uma maior simplicidade na execução das tarefas necessárias para esta primeira fase do trabalho prático e as estruturas dos modelos que nos permitirá gerar os diversos vértices para a sua exibição.

2.1 Aplicações

Esta secção é destinada a elucidação sobre as aplicações necessárias para exibir e gerar os diversos modelos requiridos.

2.1.1 Gerador

generator.cpp : Aplicação onde estão definidos os métodos que posteriormente irão gerar os respetivos vértices das diferentes formas geométricas.

2.1.2 Motor

engine.cpp : Aplicação que possui as funcionalidades principais. Permite a apresentação dos modelos graficamente, através de uma janela.

2.2 Formas

Conjunto de ficheiros constituídos por algoritmos necessários para a criação dos vértices que posteriormente serão usados em diversas formas geométricas disponíveis.

2.2.1 Cone

cone.cpp : Algoritmo que nos permite obter os diversos vértices necessários para a criação de um cone.

2.2.2 Esfera

sphere.cpp : Algoritmo que nos permite obter os diversos vértices necessários para a criação de uma esfera.

2.2.3 Plano

plane.cpp : Algoritmo que nos permite obter os diversos vértices necessários para a criação de um plano.

2.2.4 Paralelepípedo

box.cpp : Algoritmo que nos permite obter os diversos vértices necessários para a criação de um paralelepípedo.

2.3 Parsing

De modo a explorar o conteúdo dos ficheiros XML, foi utilizado um *parser* como o tinyxml2. Foi escolhido este *parser* pois é de facil utilização com APIs semelhantes, menos alocação de memória e possui um rápida leitura de ficheiros.

Poderá ser encontrado no ficheiro **tinyxml2.cpp**, na diretoria *engine* .

Capítulo 3

Utilização e demonstração

Por forma a visualizar as representações gráficas de cada sólido geométrico, é necessário passar por algumas etapas:

A fim de correr a aplicação **generator.cpp**, é necessário passar alguns argumentos.

De forma a facilitar a geração dos pontos de um modelo desejado, é de seguida apresentado o manual de ajuda do *generator* que poderá ser acedido através do comando *./generator -help*.

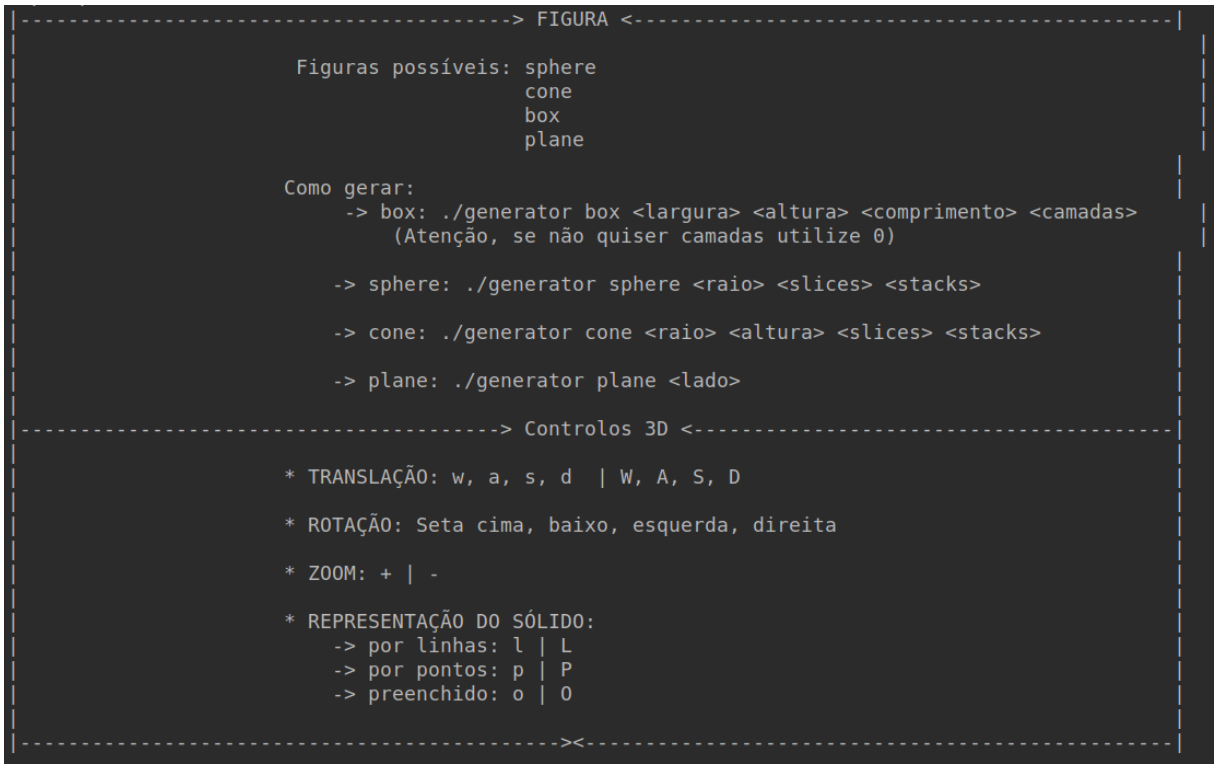


Figura 3.1: Manual de ajuda do generator

Utilizaremos o exemplo da *box*, para termos a box teremos de indicar largura, altura, comprimento e o nº de camadas pretendidas. Como poderemos ver no exemplo seguinte:

box 2 2 10 10

De seguida, foi gerado um ficheiro **box.3d**, neste ficheiro encontra-se as coordenadas

dos pontos necessários para poder gerar a figura pretendida. Antes de demonstrar o funcionamento desta aplicação é importante referir a criação de um ficheiro *xml*, que irá ser lido pelo **engine.cpp**. No entanto a criação é feita manualmente pelo utilizador e os ficheiros presentes no mesmo criados previamente pelo gerador. Após a criação dos ficheiros, o **engine.cpp** deverá ser corrido por forma a representar a figura pretendida, neste caso a **box**.

Demonstra-se prontamente um exemplo do funcionamento do **engine.cpp**, dado, por exemplo, o seguinte ficheiro *xml* como input:

```
<scene>
    <model file='box.3d'/>
</scene>
```

Figura 3.2: Exemplo de um ficheiro *XML*

O *output* deverá ser uma representação *GLUT* de uma box

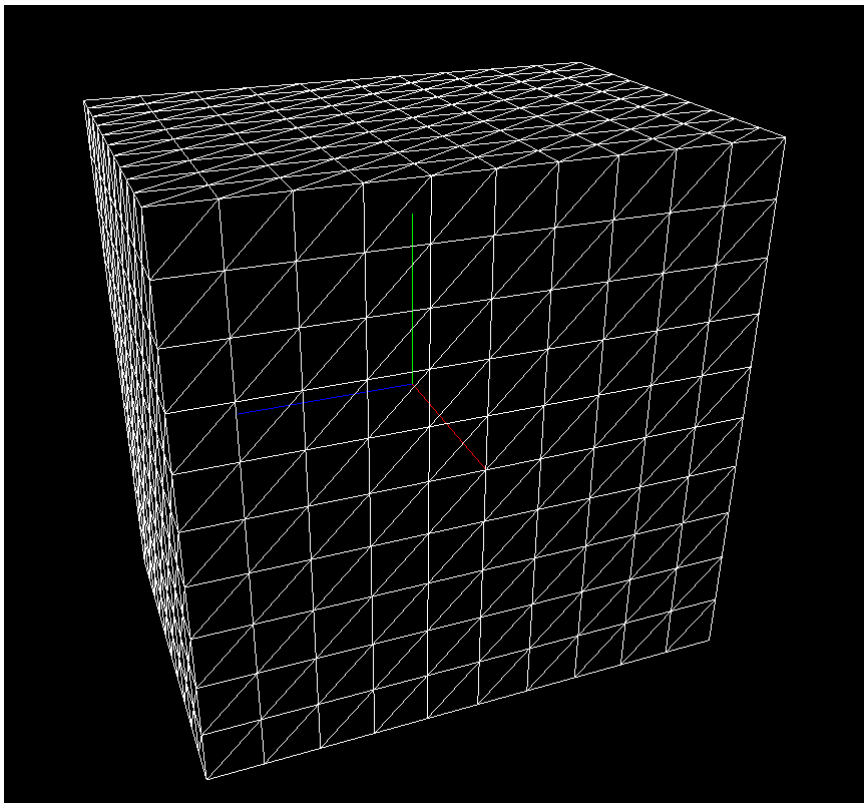


Figura 3.3: Exemplo de um *output* do *engine*

Após a visualização do modelo pedido, é possível interagir com os mesmos através do teclado. Usando as teclas A, W, S, D permitem efetuar translações do modelo apresentado, enquanto as setas permitem operar rotações do mesmo. E através das teclas '+' e '-' é possível executar um *zoom in* e *zoom out*, respetivamente.

Para além disso, é possível apresentar os modelos de diversas maneiras:

Comando L

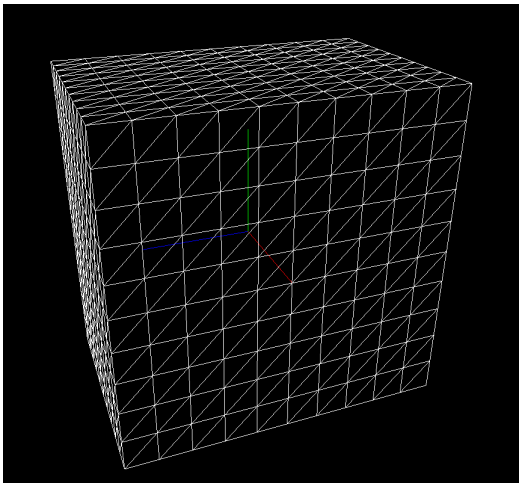


Figura 3.4: Exemplo de uma box usando linhas

Comando P

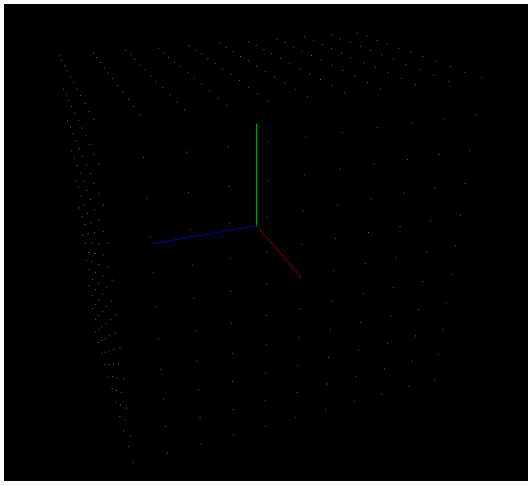


Figura 3.5: Exemplo de uma box usando pontos

Comando O

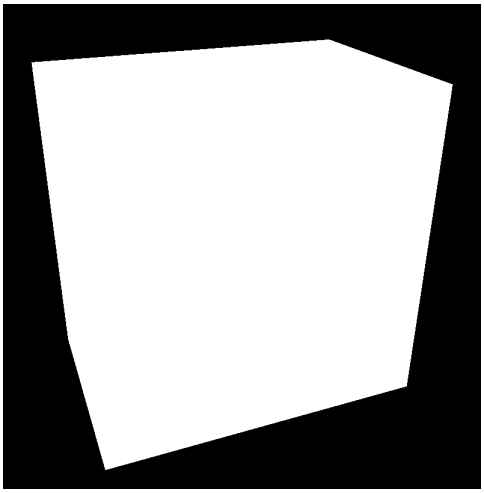


Figura 3.6: Exemplo de uma box preenchida

Capítulo 4

Primitivas Geométricas

4.1 Cone

Um cone é uma forma geométrica tridimensional que afunila suavemente a partir de uma base plana até um ponto chamado vértice ou vértice. Sendo a base um circunferência, conclui-mos que quanto maior for o número de camadas melhor será a curvatura do cone.

De forma a poder criar o cone, é necessário inserir 4 argumentos: **raio**, **altura**, **slices** (número de fatias) e **stacks** (número de camadas horizontais).

4.1.1 Algoritmo

Para construir um cone poderemos dividi-lo em duas partes: a ***Base*** e o ***Plano Lateral Curvo***.

De forma a desenvolver a base, fixamos uma altura e um ponto inicial, que neste caso será (0, (-altura)/2, 0), de modo a ficar centrada na origem, servindo assim como referência para o restante cone. Para desenharmos a circunferência que compõe a base do cone, temos de ter em conta o número de ***slices*** dadas, pois estas dividem a nossa base. Assim, e sabendo que a circunferência tem 2π radianos (que corresponde a 360o), podemos dividir essa totalidade angular pelo número de ***slices***, obtendo assim o deslocamento angular (desl1), necessário para o cálculo das coordenadas X e Z da nossa base.

Estas, podem ser calculadas da seguinte forma:

-Vértice atual:

*coordenadas1.x = raio*sin(a);*

coordenadas1.y = (-altura)/2;

*coordenadas1.z = raio*cos(a);*

-Próximo vértice:

*coordenadas3.x = raio*sin(a+desl1);*

coordenadas3.y = (-altura)/2;

*coordenadas3.z = raio*cos(a+desl1);*

O nosso α começa em 0 e percorre toda a circunferência que compõe a nossa base, acumulando o valor do deslocamento angular (desl1) a cada iteração (de *slice* em *slice*).

Após a base estar definida, falta conceber o Plano Lateral Curvo do nosso cone. Ora, sabemos que um cone, converge desde a base até à nossa altura máxima (altura/2), onde o raio será 0. Assim, teremos de decrementar o raio de *stack* em *stack*. Concluímos, portanto, que para além de um deslocamento angular, precisaremos de calcular um deslocamento radial, com a função de decrementar o raio.

Tal pode ser calculado, dividindo o nosso raio, pelo número de *stacks*, obtendo assim o nosso *desl_r*. Além disso, precisamos também de saber o deslocamento vertical necessário, para subir ao longo do nosso eixo Y, que marca a altura da nossa primitiva. Este deslocamento, chamemos de *desl2* é calculado dividindo altura pelo número de *stacks*.

Assim obtere-mos as seguintes expressões para obter os seguintes pontos do cone:

Portanto:

a += desl1;

novo_raio = raio - desl_r;

*coordenadas1.x = raio*sin(a);*

coordenadas1.y = (-altura)/2;

*coordenadas1.z = raio*cos(a);*

*coordenadas2.x = raio*sin(a+desl1);*

coordenadas2.y = (-altura)/2;

*coordenadas2.z = raio*cos(a+desl1);*

*coordenadas3.x = novo_raio*sin(a);*

coordenadas3.y = ((-altura)/2)+desl2;

*coordenadas3.z = novo_raio*cos(a);*

*coordenadas5.x = novo_raio*sin(a+desl1);*

coordenadas5.y = ((-altura)/2)+desl2;

*coordenadas5.z = novo_raio*cos(a+desl1);*

Posto isto, conseguiremos obter a seguinte representação gráfica:

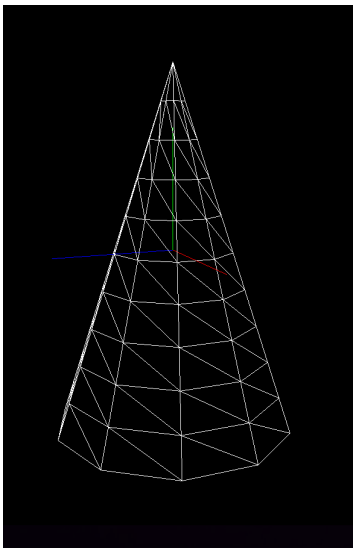


Figura 4.1: Cone gerado com as especificações atribuídas

4.2 Esfera

Uma esfera é um solido geométrico que é formado por uma superfície curva em que os pontos são equidistantes do centro. De forma a construir uma esfera, foi utilizado os seguintes parâmetros, **raio**, **slices** (número de fatias) e **stacks** (número de camadas).

De notar que quanto maior for o número de fatias e camadas, melhor será curvatura resultante.

4.2.1 Algoritmo

De forma a calcular os vértices que compõem a esfera, necessitamos de calcular os deslocamentos que esta irá usar. Existe então 2 deslocamentos possíveis, o vertical, de 0 a 180 graus (**desl2**) e o horizontal, de 0 a 360 graus (**desl1**), cujos os tais são obtidos por:

desl1 = 2π / slices;

desl2 = π / stacks;

A esfera começa por ser percorrida a partir de cima do eixo **Y**. Assim, e com estes deslocamentos podemos recorrer ao cálculo dos vértices *slice* a *slice*, em que seguindo a mesma ideologia do cone, em que o ângulo α é incrementado a cada iteração no valor de desl1, e quando este chegar a 2π, o ângulo β é incrementado em desl2, sendo o α reinicializado a 0. Sendo assim a esfera percorrida horizontalmente, até o ângulo β alcançar o valor π.

Para chegar-mos aos valores que cada coordenada desses vértices vão tomar, recorremos ao uso de coordenadas esféricas, onde a partir delas, conseguimos retirar as coordenadas cartesianas das mesmas, através das seguintes fórmulas:

*coordenadas1.x = raio * sin(a) * sin(b);*

*coordenadas1.y = raio *cos(b);*

*coordenadas1.z = raio * cos(a) * sin(b);*

A partir delas, podemos garantir os seguintes cálculos, que geram a esfera, respeitando toda a sua curvatura:

*coordenadas2.x = raio*sin(b+desl2)*sin(a+desl1);*

*coordenadas2.y = raio*cos(b+desl2);*

*coordenadas2.z = raio*sin(b+desl2)*cos(a+desl1);*

*coordenadas3.x = raio*sin(a+desl1)*sin(b);*

*coordenadas3.y = raio*cos(b);*

*coordenadas3.z = raio*sin(b)*cos(a+desl1);*

*coordenadas5.x = raio*sin(a)*sin(b+desl2);*

*coordenadas5.y = raio*cos(b+desl2);*

*coordenadas5.z = raio*sin(b+desl2)*cos(a);*

Posto isto, é obtido a seguinte representação:

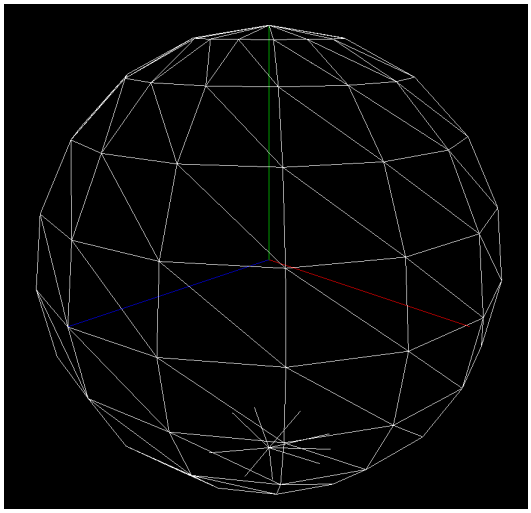


Figura 4.2: Esfera gerada com as especificações atribuídas

4.3 Plano

Um plano é constituído por dois triângulos que partilham entre si dois pontos. Todos os planos gerados estão contidos no plano **XZ**, **Y=0**, estando ele centrado na origem.

4.3.1 Algoritmo

Sabendo que este plano iria ser centrado na origem e encontrar-se no plano **XZ**, o valor de **Y** é uma constante e toma o valor de 0. Com isto prosseguimos para o calculo das coordenadas de **X** e **Z** dos vertices do plano, de forma a ficar centrado na origem, que, a partir das informações fornecidas pelo utilizador, podem ser descobertas da seguinte forma:

Sabendo que, 1 triângulo precisa de 3 vértices para ser definido, verificamos que desde já 2 vértices serão partilhados de forma a poder definir um plano.

Em seguida, sabendo que este plano iria ser centrado na origem e encontrar-se no plano **XZ**, o valor de **Y** é uma constante e toma o valor de 0. Com isto prosseguimos para o cálculo das coordenadas de **X** e **Z** dos vertices do plano, de forma a ficar centrado na origem, que, a partir das informações fornecidas pelo utilizador (**lado**), podem ser descobertas da seguinte forma:

float tamanho = lado / 2;

Criação do plano através de 4 pontos:

coordenadas.x = -tamanho;
coordenadas.y = 0;
coordenadas.z = -tamanho;

coordenadas.x = -tamanho;
coordenadas.y = 0;
coordenadas.z = tamanho;

coordenadas.x = tamanho;
coordenadas.y = 0;
coordenadas.z = tamanho;

coordenadas.x = tamanho;
coordenadas.y = 0;
coordenadas.z = -tamanho;

Assim obtemos de forma bem visível a representação gráfica de um plano através de triângulos.

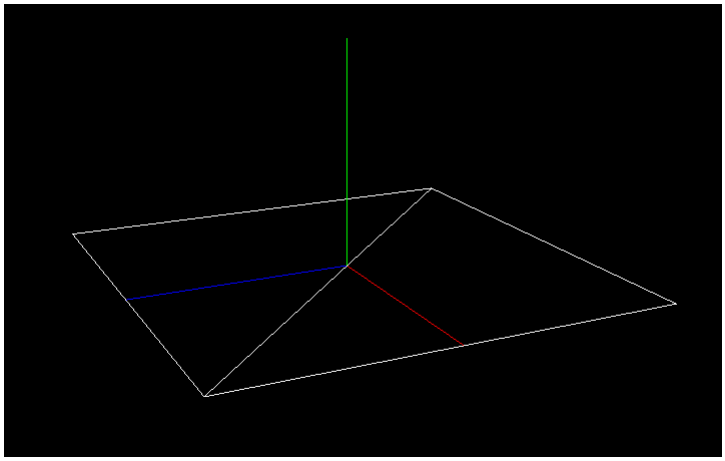


Figura 4.3: Plano gerado com as especificações atribuídas

4.4 Paralelepípedo

Considera-se um paralelepípedo, um prisma com 6 faces, em que as faces são paralelas entre si duas as duas. De forma a executar a sua construção é necessário indicar **comprimento**, **altura**, **largura** e **stacks** (número de camadas horizontais).

4.4.1 Algoritmo

Para a geração dos pontos dos triângulos, temos que considerar o número de camadas, então o espaçamento entre dois pontos pertencentes ao eixo $\boldsymbol{X}$ é dado pela *divisão do comprimento pelo número de camadas*. Para o espaçamento entre dois pontos do eixo do $\boldsymbol{Z}$ e $\boldsymbol{Y}$ substitui-se o comprimento pela largura e altura, respetivamente.

Considerando a face voltada para a frente, onde o valor z se mantém sempre constante em todos os pontos, podemos obter a mesma face que, anteriormente, era composta por dois triângulos, agora é composta por várias divisões. Assim sendo o desenho dos triângulos fez se da esquerda para a direita, sendo as coordenadas dos vértices calculadas pelo espaçamento e, chegando ao fim da linha incrementa-se a altura.

Assim, foi desenvolvida uma fórmula para a aplicação do algoritmo, sendo i (variável que permite a incrementação da altura) menor que j (variável para percorrer o eixo do x) e ambos menor que o número de camadas:

Os seguintes pontos formam um triângulo que faz parte da face da frente, sendo desl_x o espaçamento ente os pontos pertencentes ao eixo $\boldsymbol{X}$, desl_y o espaçamento entre os pontos do eixo $\boldsymbol{Y}$.

*coordenadas1.x = -x+(j*desl_x);*
*coordenadas1.y = -y+(i*desl_y);*
coordenadas1.z = comprimento;

*coordenadas2.x = (-x+desl_x)+(j*desl_x);*
*coordenadas2.y = -y+(i*desl_y);*
coordenadas2.z = comprimento;

*coordenadas3.x = (-x+desl_x)+(j*desl_x);*
*coordenadas3.y = (-y+desl_y)+(i*desl_y);*
coordenadas3.z = comprimento;

*coordenadas6.x = -x+(j*desl_x);*
*coordenadas6.y = (-y+desl_y)+(i*desl_y);*
coordenadas6.z = comprimento;

De forma a criar o outro triangulo da face frontal é utilizado os seguintes pontos

*coordenadas1.x = -x+(i*desl_x);*
coordenadas1.y = altura;
*coordenadas1.z = -z+(j*desl_z);*

*coordenadas2.x = -x+(i*desl_x);*
coordenadas2.y = altura;
*coordenadas2.z = (-z+desl_z)+(j*desl_z);*

*coordenadas3.x = (-x+desl_x)+(i*desl_x);*
coordenadas3.y = altura;
*coordenadas3.z = (-z+desl_z)+(j*desl_z);*

*coordenadas6.x = (-x+desl_x)+(i*desl_x);*
coordenadas6.y = altura;
*coordenadas6.z = -z+(j*desl_z);*

4.5 Extra

4.5.1 Cilindro

Dado os conhecimentos adquiridos no desenvolvimento das primitivas expostas anteriormente, o grupo decidiu criar uma primitiva extra, escolhendo assim um cilindro.

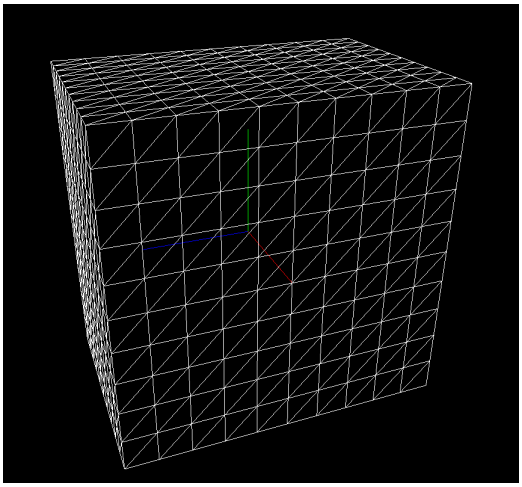


Figura 4.4: Paralelepípedo gerado com as especificações atribuídas

Deste modo, verificámos que um cilindro consiste me em duas bases circulares (circunferências) e um plano lateral curvo a interliga-las. Assim notou-se o requerimento de um **raio**, **altura** e o número de camadas verticais (**slices**).

4.5.2 Algoritmo

O algoritmo usado para a criação de um cilindro é muito semelhante ao usado para criar um cone, pois a única alteração que precisa de ser feita e do decremento do raio.

Num cilindro, o raio mantém-se desde a base ao topo, logo é desnecessária a criação de um deslocamento radial, pois o nosso raio será o mesmo ao longo da definição de vértices.

Assim com estas alterações conseguimos obter uma representação do cilindro como a seguinte:

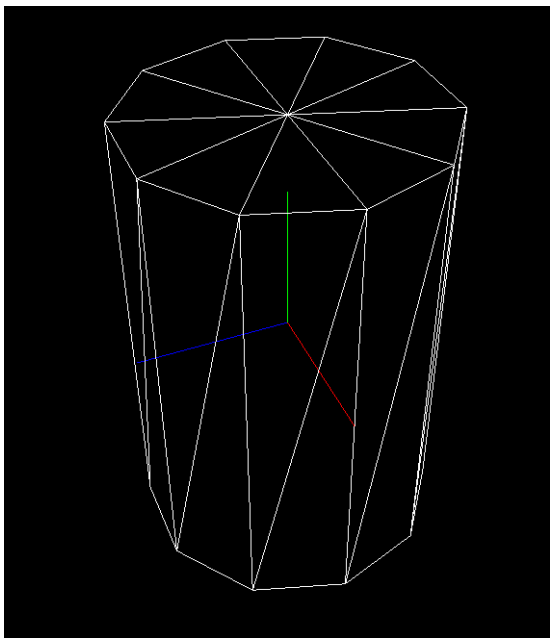


Figura 4.5: Cilindro gerado com as especificações atribuídas

Capítulo 5

Conclusão/Trabalho futuro

Nesta primeira fase é importante referir que na conceção das diversas primitivas gráficas, o grupo confrontou-se com dificuldades, que tiveram a ver com o desenvolvimento de um algoritmo que representasse de forma precisa a figura pedida, sem utilização de funções "pré-definidas"como acontece nas aulas práticas. Assim, as mesmas ocorreram na representação gráfica e tiveram origem em problemas com a aplicação ***engine.cpp***, devido a lapsos cometidos pelo grupo na construção do código.

No entanto, apesar dos constrangimentos surgidos, esta fase foi bastante enriquecedora pois exigiu do grupo empenho no desenvolvimento das aplicações que se utilizaram na representação gráfica; utilização de novas ferramentas como *OpenGL* e *GLUT* e aquisição de conhecimento de uma nova linguagem de programação, como *C++*.

Deste modo, estamos confiantes que com conhecimento adquirido, nesta primeira fase, nos ajude a conceber as restantes fases que irão surgir.