Universidade do Minho
Escola de Engenharia

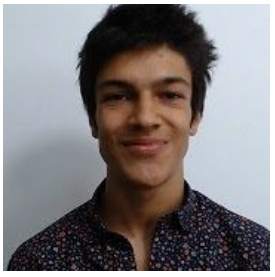
Relatório da 1ª Fase do Projeto de

Computação Gráfica

Engenharia Informática
Universidade do Minho 2020/2021


Alexandre Gomes (a89549)


Carolina Oliveira (a89523)


Manuel Barros (a89560)


Maria Ramos (a89541)

Índice

1. **Introdução** …….. 2
2. **Desenvolvimento** ……..... 3 a 13
   - **2.1. Generator** ……..….. 3 a 12
     - 2.1.1. Plane ……...….. 4 e 5
     - 2.1.2. Box ……..…..... 6 a 8
     - 2.1.3. Sphere ……...… 8 a 10
     - 2.1.4. Cone ……....… 11 e 12
   - **2.2. Engine** ……...……... 13
3. **Conclusão** ……...……. 14

1. Introdução

O trabalho a seguir apresentado foi realizado no âmbito da UC de Computação Gráfica do 2º semestre do 3º ano do curso de Engenharia Informática.

Na primeira fase do projeto, foi proposta a criação de dois programas: o *generator* e a *engine*.

O *generator* é responsável por gerar as informações necessárias à construção de quatro primitivas gráficas distintas: um plano, um cone, uma esfera e uma caixa. Estas informações serão guardadas em ficheiros *.3d,* que, posteriormente, serão interpretadas e utilizadas para a representação gráfica das primitivas pela *engine.* A *engine* deverá ainda ser capaz de, a partir de um ficheiro XML, saber quais serão os ficheiros .3d que deverá interpretar.

2. Desenvolvimento

2.1. Generator

O *generator* tem como função calcular os pontos que serão utilizados para representação gráfica das primitivas. Para tal, recebe parâmetros como o nome da figura que pretendemos desenhar, podendo esta ser *plane, box, sphere* ou *cone.* Dependendo da figura escolhida, recebe outros parâmetros que irão ser abordados mais à frente.

Os pontos calculados pelo *generator* são escritos num ficheiro *.3d*. O formato que escolhemos para estes ficheiros será apresentado na figura abaixo.

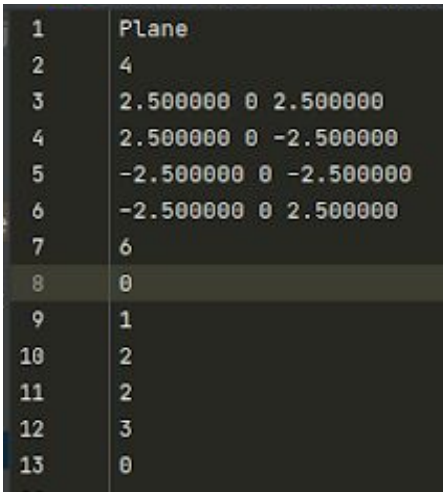


Figura 1 - plane.3d

A primeira e a segunda linha do ficheiro identificam a primitiva e o número de pontos distintos necessários para a representação gráfica da primitiva. As linhas seguintes (no exemplo da figura 1, da linha 3 a 6) representam as coordenadas dos pontos distintos representados no formato “x y z\n”. A cada um destes pontos associamos um índice, com a ordem em que aparecem no ficheiro, começando pelo índice 0 (no exemplo acima, o ponto com coordenadas (2.5, 0, 2,5) terá índice 0). Depois de todos os pontos distintos estarem representados no ficheiro, é escrito o número efetivo de pontos que será usado para a construção da primitiva (no exemplo o valor é 6, o que significa que serão usados 6 pontos para gerar um plano). Por fim, são escritos de forma ordenada os pontos, representados pelos índices a que estão associados, necessários para a representação gráfica da primitiva. Cada 3 pontos seguidos são um triângulo. No exemplo acima, temos:

- (2.5, 0, 2.5) tem índice 0

- (2.5, 0, 2.5) tem índice 1
- (2.5, 0, 2.5) tem índice 2
- (2.5, 0, 2.5) tem índice 3

Para a construção da primitiva em questão, um plano, serão utilizados 2 triângulos:

- 1º triângulo: ponto com índice 0, ponto com índice 1 e ponto com índice 2;
- 2º triângulo: ponto com índice 2, ponto com índice 3 e ponto com índice 0.

2.1.1. Plane

Para gerar os pontos necessários para desenhar um plano, para além de passarmos o argumento “Plane” ao *generator*, temos de escolher o comprimento (*length*) e a largura (*width*) do plano, como apresentado na figura abaixo.

```
CG/Fase1/Generator$ ./generator Plane 5 5 plane.3d
```

Para o desenho do plano, dividimo-lo em 2 triângulos. A função create_plane começa por calcular os 4 pontos distintos que definem o plano. De seguida, a função escreve os índices dos pontos na ordem contrária à dos ponteiros do relógio, para que sejam desenhados, 3 a 3, pela *engine*.

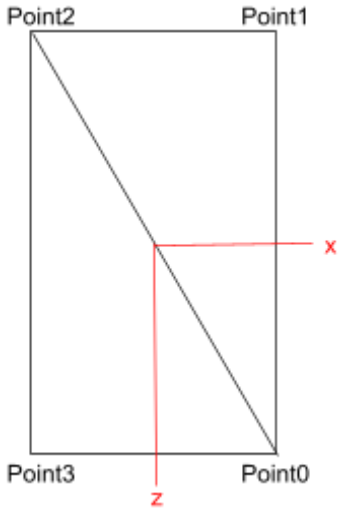


Figura 2 - Diagrama de um plano

A figura 3 exemplifica um diagrama de um plano, de comprimento *length* e largura *width,* com 4 pontos assinalados: Point0, Point1, Point2 e Point3. O plano está dividido nos triângulos (Point0 Points1 Point2) e (Point0 Point2 Point3). Estão também representados, a vermelho, os eixos positivos de x e y.

A forma de como calculamos as coordenadas de cada um dos pontos foi:

➨ **Point0**

$x = \frac{length}{2}$

$y = 0$

$z = \frac{width}{2}$

➨ **Point1**

$x = \frac{length}{2}$

$y = 0$

$z = -\frac{width}{2}$

➨ **Point2**

$x = -\frac{length}{2}$

$y = 0$

$z = -\frac{width}{2}$

➨ **Point3**

$x = -\frac{length}{2}$

$y = 0$

$z = \frac{width}{2}$

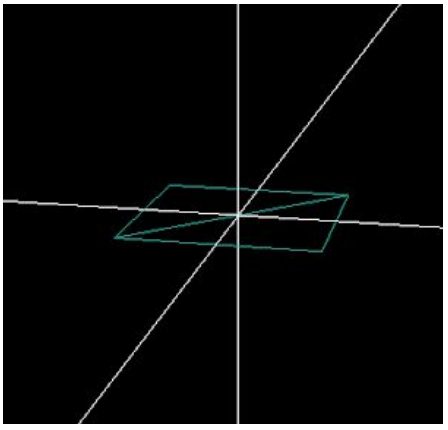


Figura 3 - Plano centrado na origem

Para as restantes primitivas todos os pontos e índices são guardados em vetores para facilitar a escrita no ficheiro destino e para ser fácil de manusear o código.

2.1.2. Box

Para determinar os pontos necessários para a construção de uma caixa, para além da indicação que queremos uma caixa, são necessários mais 4 parâmetros, comprimento (*length*), largura (*width*), altura (*height*) e opcionalmente o número de divisões (*divisions*)

```
CG/Fase1/Generator$ ./generator box 5 5 5 4 box.3d
```

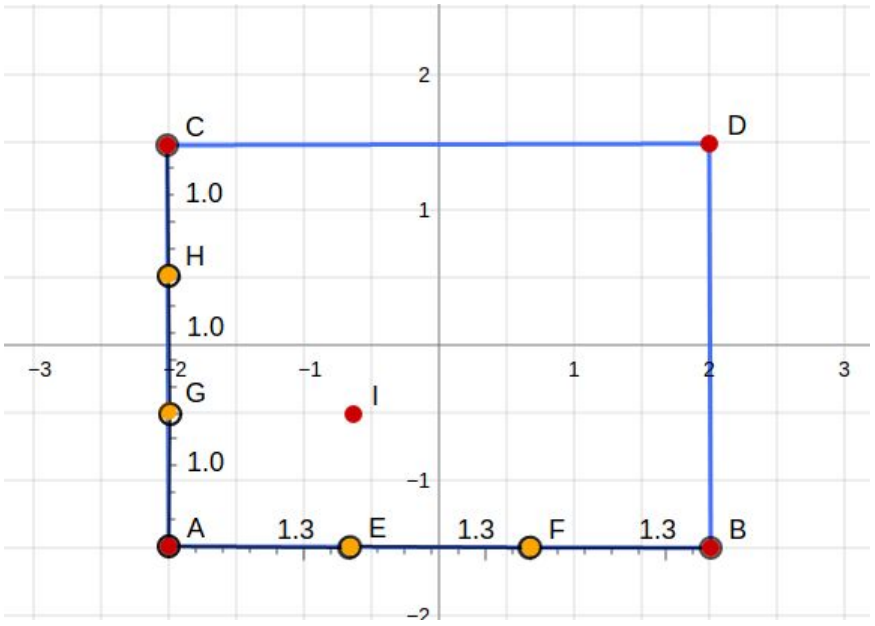
O centro da box criada é a origem do referencial por isso começamos por calcular o ponto inicial:

P0 = (-length/2,-height/2,-width/2)

Depois calculamos os pontos que pertencem aos planos que passam pelo ponto inicial, para isso precisamos das coordenadas iniciais desses planos (P0x,P0y,P0z).

Vamos assumir que length=4, width=3 e divisões=3, sabemos que no eixo do x todos os pontos estão afastados entre si 4/3(1.333) e no eixo dos z todos os pontos estão afastados entre si 3/3(1), a partir destes valores torna-se fácil de calcular todos os pontos para esta face, repetimos este processo 2 vezes para o planos zy yx, e por fim calculamos os 3 planos paralelos a estes.

Desta forma acabamos por calcular pontos a mais (faces têm arestas em comuns) mas o processo é intuitivo e a quantidade de pontos não é muito significativa.

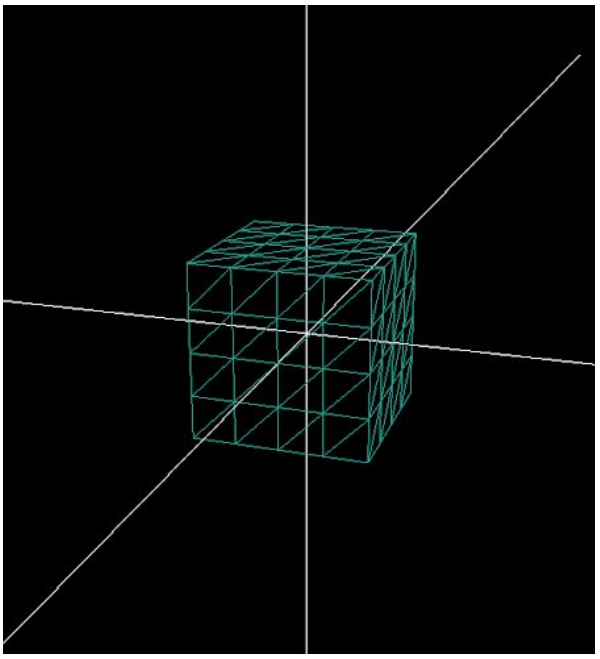


```
float x0=-(x/2),y0=-(y/2),z0=-(z/2);
float xinc=x/divs,yinc=y/divs,zinc=z/divs;

xzPlane(x0,y0,z0,xinc,zinc,divs+1,points);
zyPlane(x0,y0,z0,yinc,zinc,divs+1,points);
yxPlane(x0,y0,z0+z,xinc,yinc,divs+1,points);
xzPlane(x0,y0+y,z0,xinc,zinc,divs+1,points);
zyPlane(x0+x,y0,z0,yinc,zinc,divs+1,points);
yxPlane(x0,y0,z0,xinc,yinc,divs+1,points);
```

Finalmente temos de indicar os índices dos vértices a desenhar, continuando o exemplo acima e imaginando que a face da figura foi a primeira a ser desenhada então sabemos o índice do ponto inicial é o 0 e que são 16($(divs+1)^2$) pontos no total sendo que a cada 4 pontos estamos numa linha nova, para desenhar o triângulo AEI e AIG teríamos de escolher os índices 015 e 054 respetivamente.

Na função abaixo i0 é o índice inicial e steps é o número de índices por aresta(i0=0 steps=divisions+1 no caso acima).

```
void calcInds(int i0,int steps,vector<int> &indexs){
    for (int i=0;i<steps;i++){
        for (int j=0;j<steps;j++){
            indexs.push_back(i0+j+i*(steps+1));
            indexs.push_back(i0+(j+1)+i*(steps+1));
            indexs.push_back(i0+(j+1)+(i+1)*(steps+1));

            indexs.push_back(i0+j+i*(steps+1));
            indexs.push_back(i0+(j+1)+(i+1)*(steps+1));
            indexs.push_back(i0+j+(i+1)*(steps+1));
        }
    }
}
```

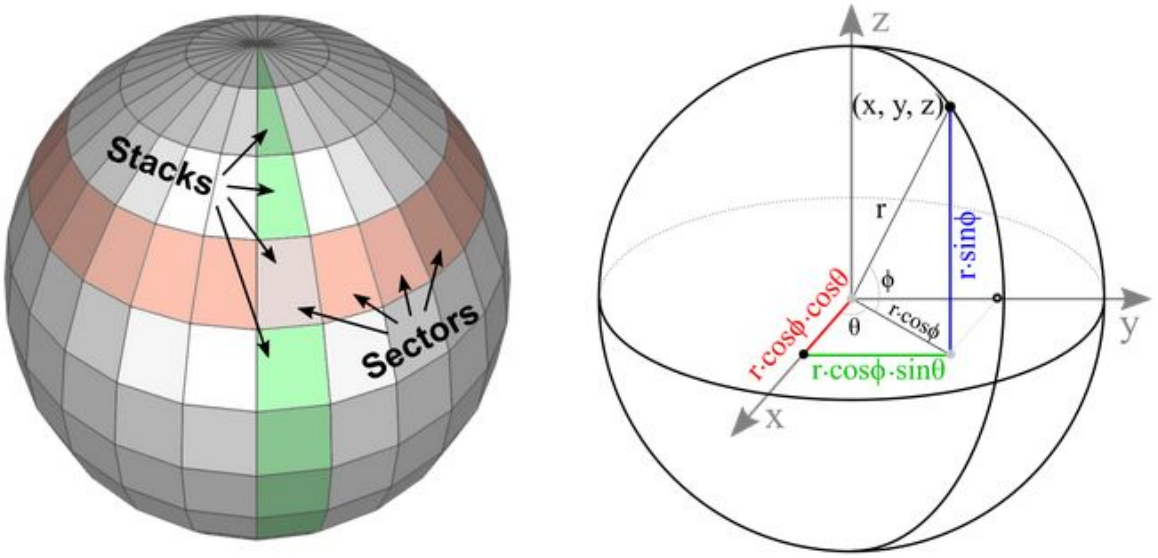


2.1.3. Sphere

Para calcular os pontos que irão dar origem a uma esfera, para além de indicarmos que se trata de uma esfera, são necessários mais 3 parâmetros: raio (*radius*), *slices* e *stacks*.

```
/CG/Fase1/Generator$ ./generator Sphere 10 20 15 sphere.3d
```

Apesar de poder parecer mais intimidante o cálculo dos pontos da esfera e cone são bastante fáceis de efetuar.

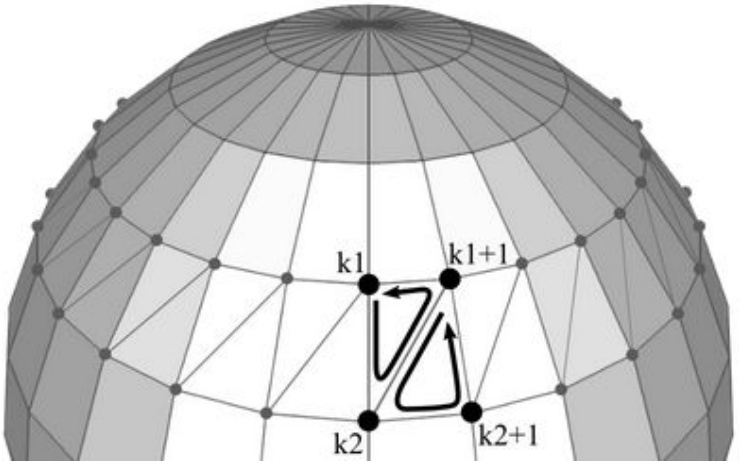


A figura à esquerda mostra uma esfera dividida entre stacks e setores (que são as slices no nosso caso), a figura à direita mostra as equações para o cálculo das coordenadas cartesianas de um ponto no espaço a partir das suas coordenadas polares, em que a origem é o ponto (0,0,0). O ângulo θ está compreendido entre 0º e 360º e o ângulo Φ está compreendido entre -90º e 90º.

Para começar sabemos que o ponto do topo da esfera tem como coordenadas (0,raio,0) e que o ponto da base da esfera tem como coordenadas (0,-raio,0), de seguida temos de ter em conta que a distância entre duas slices adjacentes é de 360/(número de setores) e que a distância entre duas stacks adjacentes é de 180/(número de stacks), com um simples ciclo que começa com θ=0º e Φ=-90+(180/stack)º e vai aumentando de acordo com os valores calculados em cima acabando em θ=360º e Φ=90-(180/stack)º (Φ nunca chegar a ser -90º e 90º visto que já sabemos as coordenadas desses pontos).

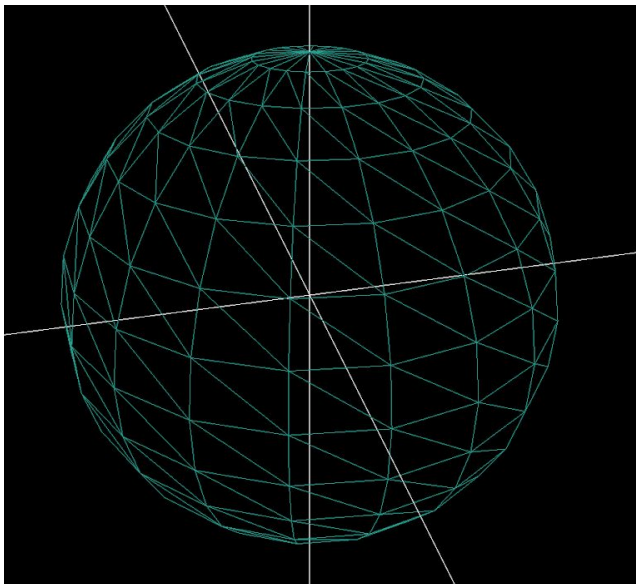
```
void midlePointsS(float radius,int slices,int stacks,vector<Point> &points){
    float stackang=0,sliceang=0,x1=0,y1=0,z1=0;
    for (int i=1; i<stacks; i++){
        stackang = (-M_PI/2)+(M_PI*((float)i/stacks));
        for (int j=0; j<slices; j++){
            sliceang = 2*M_PI*((float)j/slices);
            x1=(radius*cos(stackang)*cos(sliceang));
            z1=(radius*cos(stackang)*sin(sliceang));
            y1=radius*sin(stackang);
            points.push_back(Point(x1,y1,z1));
        }
    }
}
```

Para determinar a ordem dos índices temos de ter em conta que uma esfera tem (slices*(stacks-1)+2) pontos e que o ponto da base tem como índice 0, o ponto do topo tem como índice númeroTotalPontos-1, por fim temos de saber que pontos adjacentes que estejam na mesma slice têm (slices-1) índices entre si. Imaginemos que a esfera tem 10 slices e que o índice de k2 é 31 então sabemos que k2+1=32, k1=(32+slices)=42 e k1+1=43 (a nossa esfera é desenhada de baixo para cima).



Processo completo do cálculo dos pontos e índices de uma esfera.

```
points.push_back(Point (0,-radius,0));//bottom point
midlePointsS(radius,slices,stacks,points);
points.push_back(Point(0,radius,0));//top point

bottomIndexsS(slices,0,indexs);
middleIndexsS(slices,stacks,indexs);
topIndexsS(slices,stacks,indexs);
```



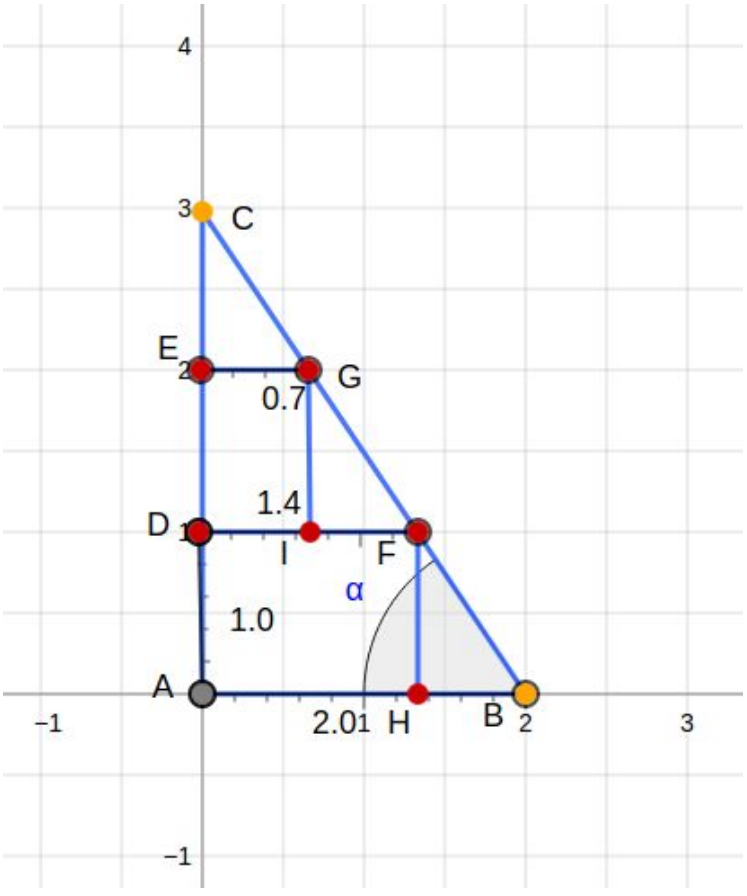
2.1.4. Cone

Para determinar os pontos necessários para a construção de um cone, para além da indicação que queremos um cone, são necessários mais 4 parâmetros: raio (*radius*), altura (*height*), *slices* e *stacks*.

```
/CG/Fase1/Generator$ ./generator Cone 5 8 20 15 cone.3d
```

Começamos também por calcular o ponto da base (0,0,0) e do topo (0,height,0)**,** para calcular os restantes pontos do cone usamos as mesma fórmulas que passam pontos com coordenadas polares para cartesianas(tendo em conta que a coordenada y do pontos já são fáceis de identificar ao contrário da esfera), para as slices o processo é igual ao da esfera, θ está compreendido entre 0º e 360º e pontos da mesma stack que são adjacentes tem um ângulo θ com (360/slices)º de diferença entre eles. A diferença está nas stacks visto que pontos de stacks diferentes têm raios diferentes.

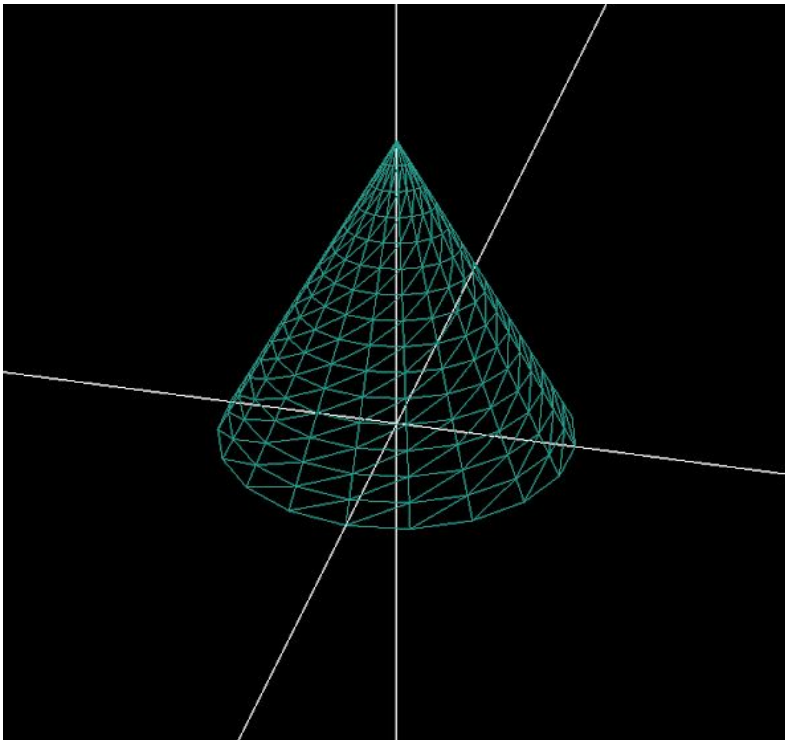
Analisando a figura de baix, começamos por calcular α=(arctan(height/radius)) e depois calculamos a diferença entre raios de pontos em stacks adjacentes(AB-DF), HB=IF=(height/stacks)*tan(90-α).



Com estes valores estamos prontos a calcular todos os pontos na superfície do cone.

Processo completo do cálculo dos pontos e índices de uma esfera.

```
points.push_back(Point(0,0,0));//bottom point
middlepointsC(radius,height,slices,stacks,points);
points.push_back(Point(0,height,0));//top point

bottomIndexsC(slices,indexs);
middleIndexsC(slices,stacks,indexs);
topIndexsC(slices,stacks,indexs);
```



2.2. Engine

A aplicação *engine* tem como função desenhar no ecrã as figuras 3D especificadas nos ficheiros criados com a aplicação *generator*. Esta aplicação deverá receber um ficheiro XML contendo a informação de quais os ficheiros que devem ser carregados.

A leitura e interpretação do ficheiro XML é feita com o auxílio da ferramenta *tinyXML* e resulta na construção de uma lista, em memória, do nome dos ficheiros a ler.

Uma vez que os ficheiros criados previamente com o *generator* contém já uma listagem completa e ordenada de todos os pontos que devem ser desenhados para produzir as figuras desejadas, o trabalho do *engine* é fácil: guardar esta lista de pontos em memória e, de seguida, iterar sobre essa lista, desenhando os pontos no ecrã.

Para cada ficheiro .3d lido, é criado um objeto da classe *Figura3D,* onde é guardado, numa variável de instância, o número total de pontos a desenhar e, numa matriz de *floats*, todos os pontos (pela ordem que devem ser desenhados).

Interpretando a sequência de pontos como uma lista de triângulos (glBegin(GL_TRIANGLES)) desenham-se os pontos com glVertex3f().

Na janela onde é desenhada a imagem, é possível explorar a cena usando o rato para mover a câmera (câmera do tipo *Explorer)* e a *scroll wheel* para aproximar ou afastar a câmera da origem do referencial.

3. Conclusão

A realização desta fase do trabalho possibilitou explorar melhor as funcionalidades do GLUT e, nomeadamente, aprender a desenvolver os modelos para a construção de figuras geométricas. Foi necessário um trabalho prévio para saber que tipo de operações deveriam ser utilizadas para o cálculo dos pontos dos triângulos que compõem as figuras.