Universidade do Minho
Escola de Engenharia
Mestrado Integrado em Engenharia Informática

Computação Gráfica

Ano Letivo de 2016/2017

Fase 1 – Primitivas Gráficas

A27748 - Gustavo José Afonso Andrez

A74634 - Rogério Gomes Lopes Moreira

A76507 - Samuel Gonçalves Ferreira

A71835 – Tiago Filipe Oliveira Sá

Docente: António José Borba Ramires Fernandes

6 de março de 2017

Índice

Introdução

O trabalho prático apresentado, do qual este relatório diz apenas respeito à Fase 1 tem como objetivo desenvolver competências na área dos gráficos 3D e no desenvolvimento de um motor para tal. O trabalho está dividido em 4 fases.
Nesta primeira fase foram criadas duas aplicações:

- **Generator** – gera um ficheiro com os vértices de uma da primitiva gráfica.
- **Drawer** – lê um ficheiro de configuração em XML e mostra os modelos.

As aplicações foram desenvolvidas recorrendo ao Visual Studio e à linguagem C++. Além disso, todo o desenvolvimento respeita a regra da mão direita e o referencial usado é o padrão.

Generator

A função main do gerador vê qual é a forma que se pretende gerar (1º argumento), chama a função geradora da forma pretendida (tendo em conta o número de argumentos fornecidos) e escreve o resultado num ficheiro com o nome fornecido como último argumento.

Observações:

- Plane - pode receber 1 ou 2 argumentos. No caso de 1 argumento é passada à função *plane* largura e comprimento com o mesmo valor (correspondendo a um quadrado);
- Box - é admissível a não introdução do parâmetro divisões. Neste caso o parâmetro divisões tomará o valor 1, correspondendo a um paralelipípedo com 2 triângulos em cada face;
- Os ficheiros são gerados pelo gerador e são lidos pelo Drawer numa pasta `xml_3d` que deve estar no mesmo diretório que as pastas dos projetos Visual Studio;
- No final do ficheiro que contém os pontos gerados é colocada a string “FIM” para que seja mais fácil na leitura identificar o fim da lista de pontos.

Plane

Sintaxe:

```
gerador plane <size> <fileName>
```

OU:

```
gerador plane <sizeX> <sizeZ> <fileName>
```

Características:

- São requeridos os parâmetros largura e comprimento (ou vice-versa);
- O retângulo gerado pertence ao plano XZ, ou seja, tem a coordenada y = 0;
- A figura gerada é centrada na origem;
- São gerados 2 conjuntos de 3 coordenadas correspondentes aos 2 triângulos que constituirão o retângulo.

Algoritmo:

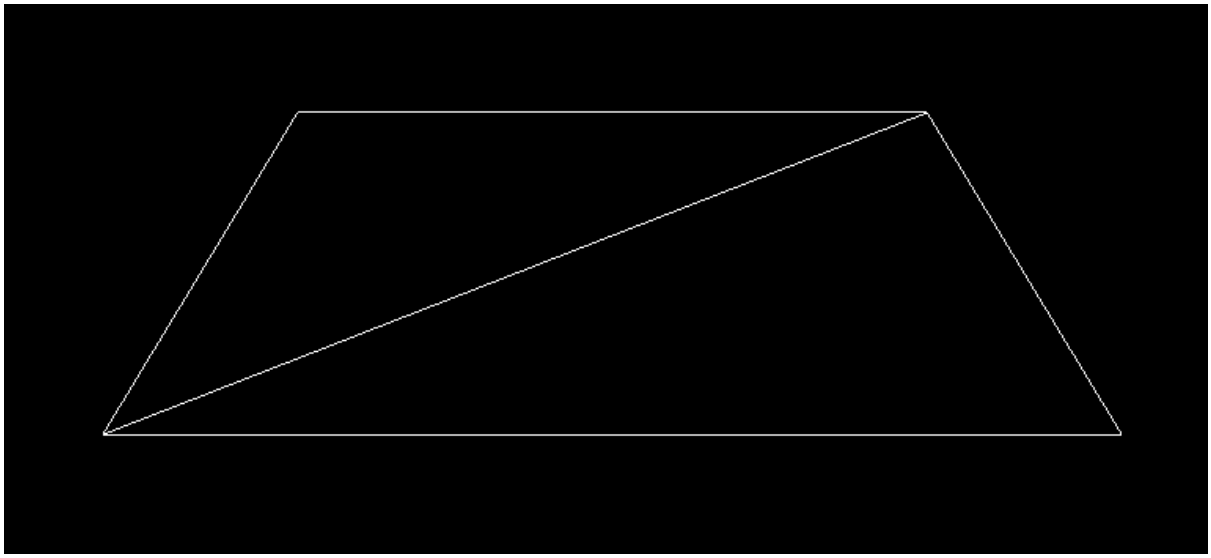
- Descrição dos 2 conjuntos de 3 pontos correspondentes aos triângulos que definem o retângulo atendendo a ordem de escrita dos pontos no sentido anti-horário de uma posição no semieixo positivo dos YY;

Pseudo código:

```
plane(largura, comprimento) {
   coordenada X = metade da largura;
   coordenada Z = metade do comprimento;
   escrever no ficheiro {
         1°triangulo:
           (-coordenada X, 0, coordenada Z)
           (coordenada X, 0, - coordenada Z)
           (-coordenada X, 0, - coordenada Z)
         2°triangulo:
           (-coordenada X,0, coordenada Z)
           (coordenada X,0, coordenada Z)
           (coordenada X, 0, -coordenada Z)
    escreve no ficheiro{"FIM"}
  }
}
```

Exemplo:

```
C:\>generator plane 5 5 plane.3d
```



Box

Sintaxe:

```
gerador box <xSize> <ySize> <zSize> <fileName>
```

OU:

```
gerador box <xSize> <ySize> <zSize> <divisions> <fileName>
```

Características:

- São requeridos os parâmetros largura, comprimento e altura;
- O paralelepípedo gerado é centrado na origem;

Algoritmo:

- posicionamento inicial na coordenada (-x/2, -y/2, -z/2);
- recolha dos conjuntos de 3 pontos dos triângulos da face pertencente ao plano X=-x/2, por incremento dos saltos no eixo dos ZZ e YY;
- recolha dos conjuntos de 3 pontos dos triângulos das faces pertencentes aos planos perpendiculares ao eixo dos XX, por incremento dos saltos no eixo dos ZZ, YY e XX;
- recolha dos conjuntos de 3 pontos dos triângulos da face pertencente ao plano X=x/2, por incremento dos saltos no eixo dos ZZ e YY;

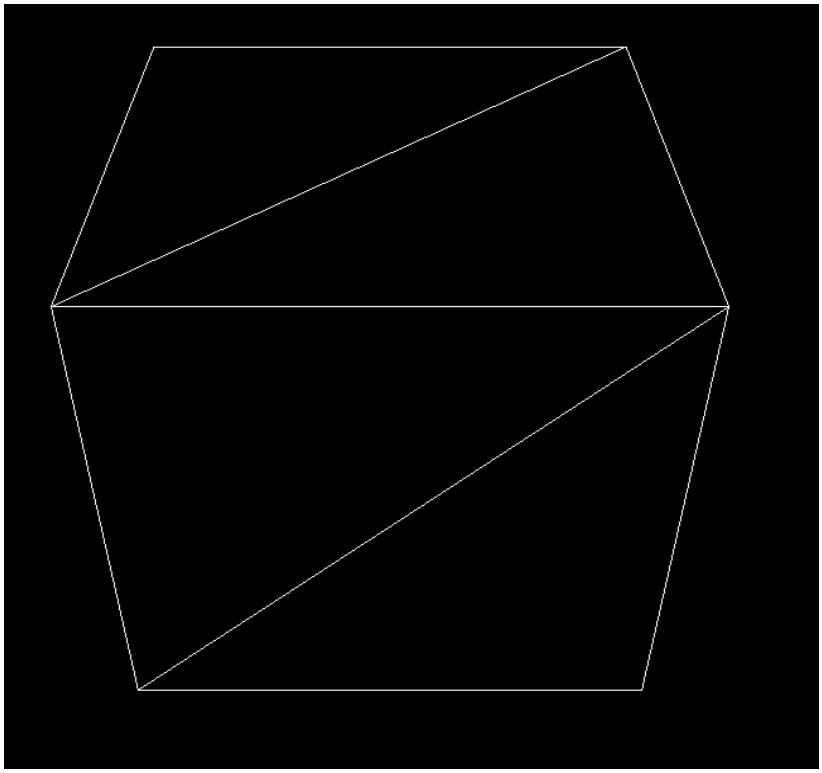
Pseudo código:

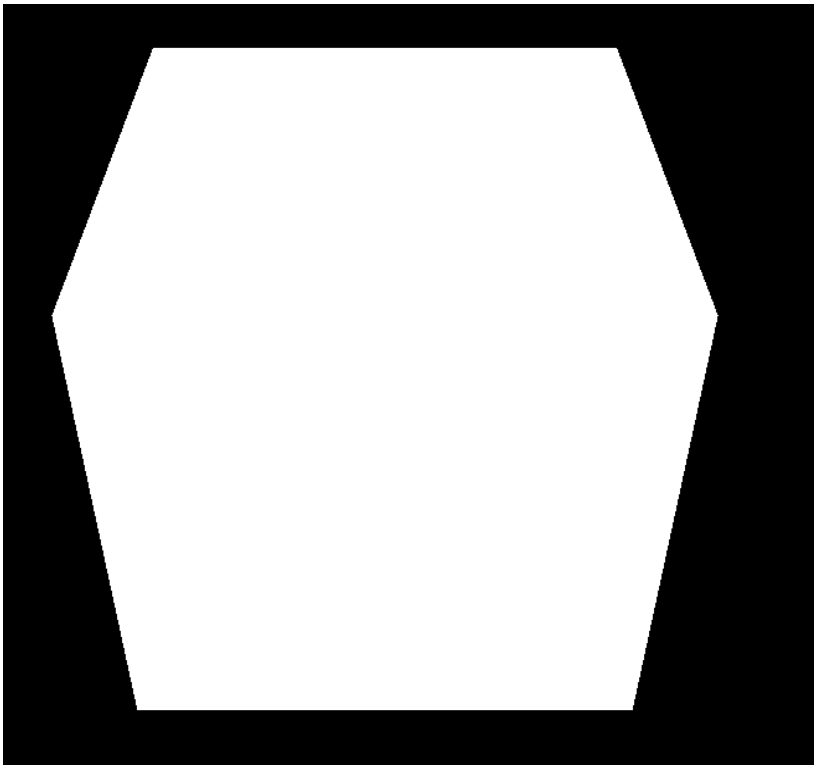
```
box(largura, altura, comprimento, nº de divisões) {
  coordenada X = largura/2;
  coordenada Y = altura/2;
  coordenada Z = comprimento/2;
  deltaX (passo) = largura / nº de divisões;
  deltaY (passo) = altura / nº de divisões;
  deltaZ (passo) = comprimento / nº de divisões;

  ciclo(i=0 até ao nº de divisões){
    // 1ª face (plano -x):
    se i=0 então
      ciclo(j=0 até ao nº de divisões -1){
      coordY = -y + j*deltaY;
      coordY1 = -y + (j+ 1)*deltaY;
```

```
      ciclo(k=0 até ao nº divisões){
        coordZ = -z + k*deltaZ;
        coordZ1 = -z + (k + 1)*deltaZ;
        escrever no ficheiro{
          coordenadas dos pontos dos 2 triângulos
          atendendo à ordem pela qual são desenhados
   }}}}
  (caso contrário){
    // última face(plano x)
      se(i = nº de divisões){
        ciclo(j=0 até ao nº de divisões -1){
          coordY = -y + j*deltaY;
          coordY1 = -y + (j+ 1)*deltaY;
          ciclo(k=0 até ao nº divisões){
            coordZ = -z + k*deltaZ;
            coordZ1 = -z + (k + 1)*deltaZ;
          escrever no ficheiro{
          coordenadas dos pontos dos 2 triângulos
          atendendo à ordem pela qual são desenhados
          }}}}
      ciclo(j=0 até ao nº de divisões -1){
        coordY = -y + j*deltaY;
        coordY1 = -y + (j + 1)*deltaY;
        coordZ = -z + j*deltaZ;
        coordZ1 = -z + (j + 1)*deltaZ;
        escrever no ficheiro{
          //Faces laterais (planos -z e z) e
          //Faces superior e inferior (-y e +y)
          coordenadas dos pontos dos 8 triângulos
          atendendo à ordem pela qual são desenhados
        }}}}
  escrever no ficheiro{"FIM"};
}
```

Exemplo (sem divisões):

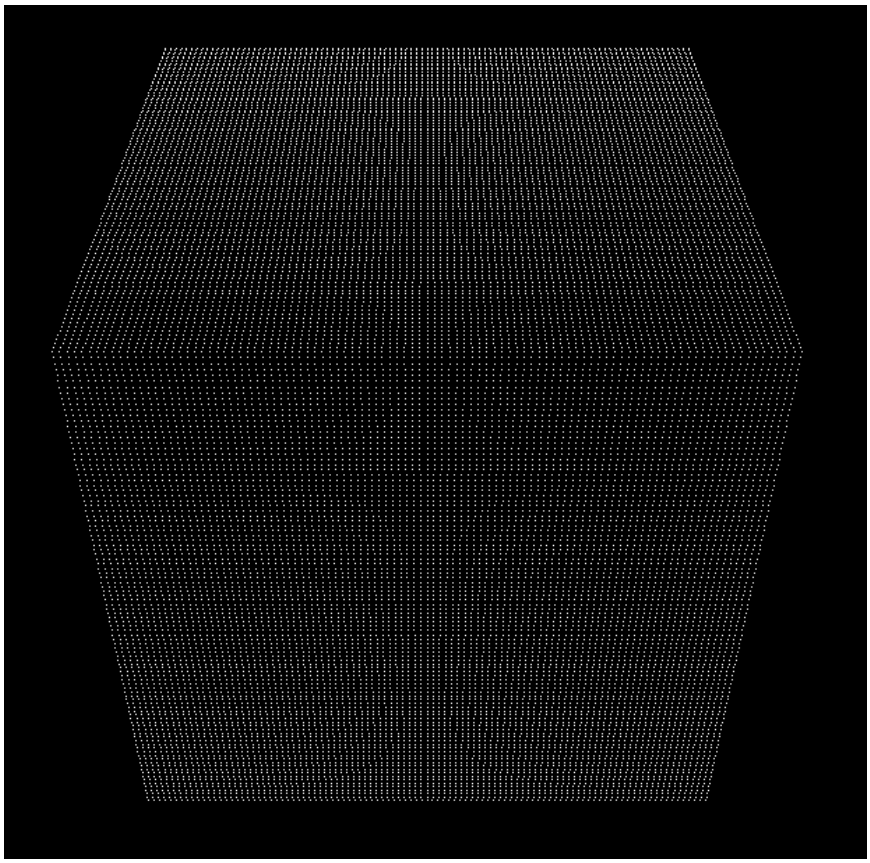
```
C:\>generator box 4 4 4 box.3d
```





Exemplo (com divisões):

```
C:\>generator box 4 4 4 100 box.3d
```



Sphere

Sintaxe:

```
gerador sphere <radius> <slices> <stacks> <fileName>
```

Características:

- são requeridos os parâmetros raio, divisões verticais e horizontais;
- centrada na origem;

Algoritmo:

- posicionamento inicial na coordenada (0, raio, 0);
- existe um ângulo α ∈ [0, π] e um Δ α igual a α /stacks;
- existe um ângulo β ∈ [0, 2π] e um Δ β igual a β /slices;
- os pontos dos triângulos são calculados fazendo o varrimento do domínio de β, a um passo de Δ β, e em cada valor, é varrido o domínio de α a um passo de Δ α, sendo definido um "anel" horizontal a cada passo de β.

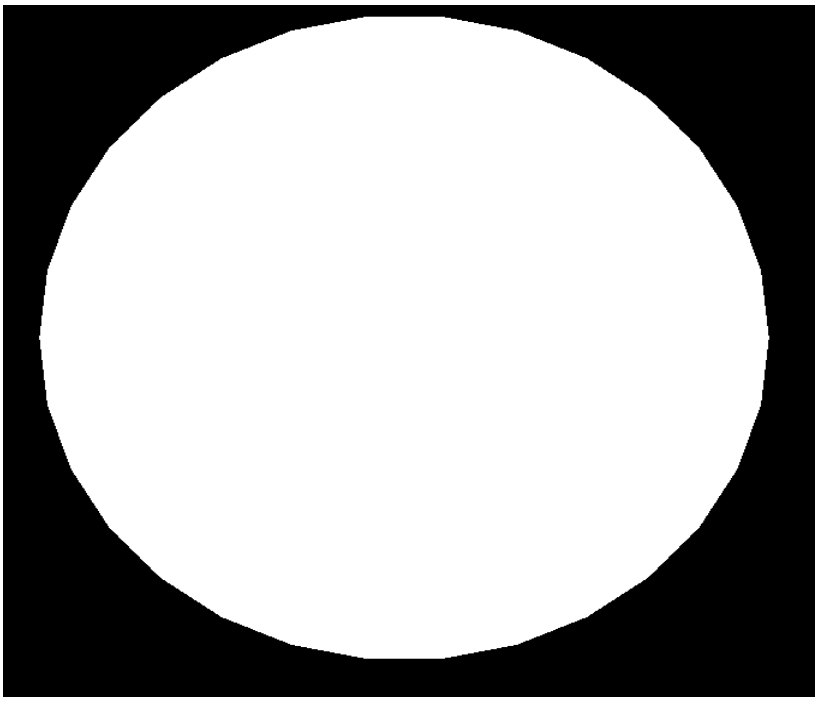
Pseudo código:

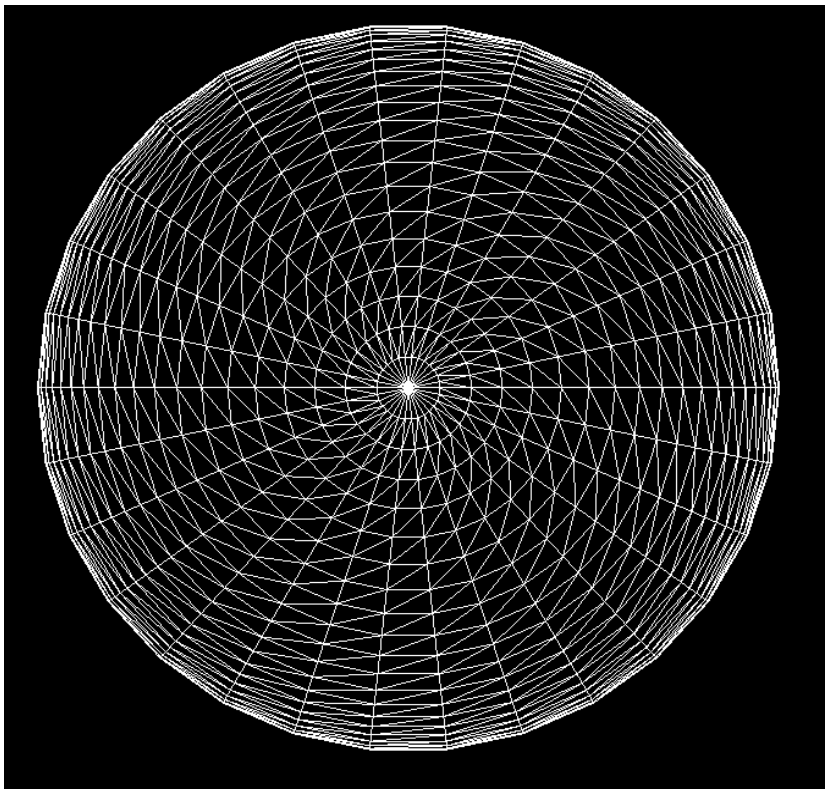
```
sphere(raio, slices, stacks) {
  deltaBeta = π/stacks;
  deltaAlpha = π /slices;
  ângulo beta, alpha;
  ciclo (i = 0 até ao número de stacks-1) {
  beta = i*deltaBeta;
    ciclo (j = 0 até ao número de slices-1) {
      alpha = j*deltaAlpha;
      p1x = radius*sin(beta)*cos(alpha);
      p1y = radius*sin(beta)*sin(alpha);
      p1z = radius*cos(beta);

      p2x=radius*sin((beta + deltaBeta))*cos(alpha);
      p2y =radius*sin((beta + deltaBeta))*sin(alpha);
      p2z = radius*cos((beta + deltaBeta));

      p3x = radius*sin(beta)*cos((alpha + deltaAlpha));
```

```
      p3y = radius*sin(beta)*sin((alpha + deltaAlpha));
      p3z = radius*cos(beta);

      p4x = radius*sin((beta + deltaBeta))*cos((alpha + deltaAlpha));
      p4y = radius*sin((beta +deltaBeta))*sin((alpha+deltaAlpha));
      p4z = radius*cos((beta + deltaBeta));
      escrever no ficheiro{
        (p2x, p2y, p2)
        (p3x, p3y, p3z)
        (p3x, p3y, p3z)
        (p2x, p2y, p2z)
        (p4x, p4y, p4z)
      }
    }
  }
escrever no ficheiro{"FIM"};}
```

Exemplo:

```
C:\>generator sphere 5 10 10 sphere.3d
```





Cone

Sintaxe:

```
gerador cone <bottomRadius> <height> <slices> <stacks> <fileName>
```

Características:

- são requeridos os parâmetros raio, altura, divisões verticais e horizontais;
- base do cone centrada na origem do plano XZ;

Algoritmo:

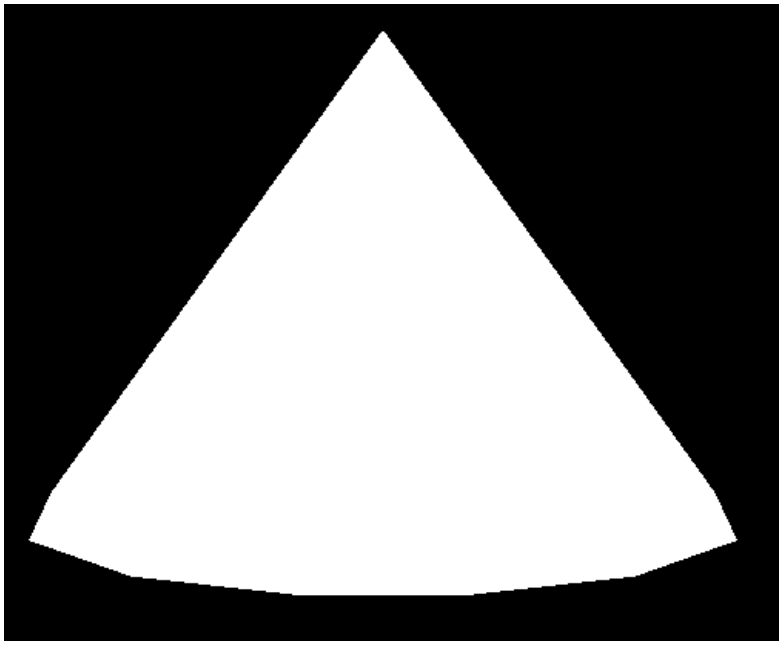
- posicionamento inicial na coordenada (0, 0, 0);
- existe um $h \in$ [0, altura] e um Δh igual a altura/stacks;
- existe um ângulo $\alpha \in [0, 2\pi]$ e um $\Delta \alpha$ igual a α /slices;
- os pontos dos triângulos da base são calculados fazendo o varrimento do domínio de α a um passo de $\Delta \alpha$.
- os pontos dos triângulos da lateral são calculados fazendo o varrimento do domínio de h a um passo de Δh e em cada valor é varrido o domínio de α a um passo de $\Delta \alpha$, sendo definido um “anel” horizontal a cada passo de h, tendo em consideração a atualização do raio.

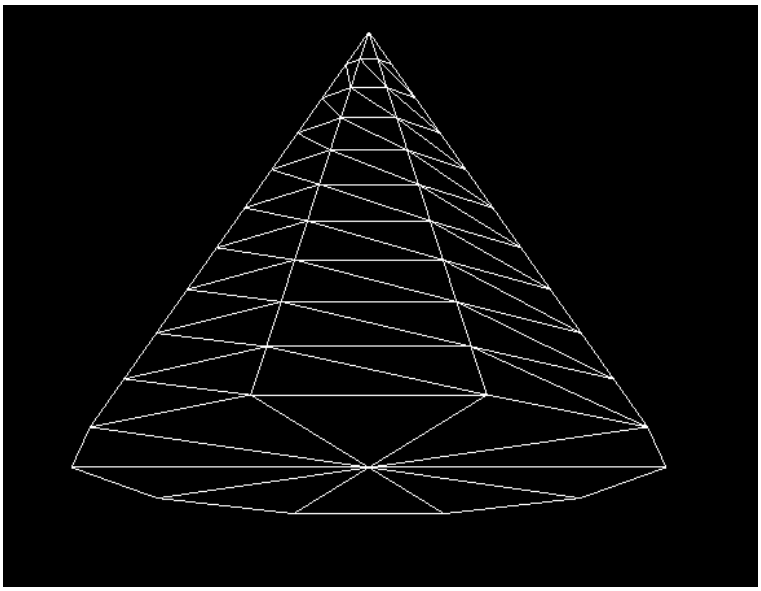
Pseudo código:

```
cone( radius, height, slices, stacks) {
  deltaHeight = height / stacks;
  deltaAlpha = 2□/slices;
  deltaRadius = radius / stacks;
  alpha, heightAux, radiusAux, radiusNext;
  ciclo (i = 0 até ao número de stacks-1){
    heightAux = i*deltaHeight;
    radiusAux = radius - i*deltaRadius;
    radiusNext = radius - (i + 1)*deltaRadius;
    ciclo (j = 0 até ao número de slices-1){
      alpha = j*deltaAlpha;
      cosAlpha= cos(alpha);
      cosNextAlpha = cos(alpha + deltaAlpha);
      sinAlpha = sin(alpha);
      sinNextAlpha = sin(alpha + deltaAlpha);
      escrever no ficheiro{
        (radiusAux*cosAlpha , heightAux , radiusAux*sinAlpha)
        (radiusNext*cosNextAlpha , heightAux+deltaHeight ,
radiusNext*sinNextAlpha)
        (radiusAux*cosNextAlpha , heightAux , radiusAux*sinNextAlpha)

        (radiusAux*cosAlpha , heightAux , radiusAux*sinAlpha)
        (radiusNext*cosAlpha , heightAux + deltaHeight ,
radiusNext*sinAlpha)
        (radiusNext*cosNextAlpha , heightAux + deltaHeight ,
radiusNext*sinNextAlpha)
      }
    }
  }
  ciclo (i = 0 até ao número de slices-1){
    alpha = i*deltaAlpha;
    escrever no ficheiro{
    (0,0,0)
    (radius*cos(alpha) , 0 , radius*sin(alpha))
    (radius*cos(alpha+deltaAlpha), 0 ,radius*sin(alpha+deltaAlpha))
    }
  }
  escrever no ficheiro{"FIM"};
}
```

Exemplo:

```
C:\>generator sphere 5 8 10 10 cone.3d
```





Drawer

A segunda parte do trabalho consiste na criação de um programa que, a partir da leitura de um ficheiro XML onde estão contidos os dados sobre as figuras a desenhar, gera essas mesmas figuras. Nesta 1ª fase do projeto o ficheiro XML apenas contém o nome do ficheiro com os pontos gerados na etapa anterior para os diversos tipos de figura.

O ficheiro XML lido pelo drawer tem, nesta etapa, este formato:

```
<scene>
     <model file="cone.3d" />
</scene>
```

A aplicação apenas desenha a(s) figura(s) a partir dos pontos dos modelos contidos no ficheiro xml. Estes pontos são lidos para memória, fazendo recurso à biblioteca TinyXml, e só depois é feito o rendering dos triângulos gerados.

Para além disto foram definidas funções que que apresentam as seguintes características:

- As setas do teclado servem para rodar a figura.
- As teclas W e S servem para zoom in e zoom out, respetivamente.
- O botão esquerdo do rato muda o tipo de preenchimento da figura (line, point ou solid).

Conclusão

Com a elaboração desta fase do projeto concluímos que a base do que vai ser o motor de geração de gráficos em 3D já está elaborada e vai suportar as restantes fases do trabalho.

Pelo referido, consideramos que os objetivos desta fase do trabalho foram cumpridos, incluindo o número de divisões na geração dos pontos de uma box.