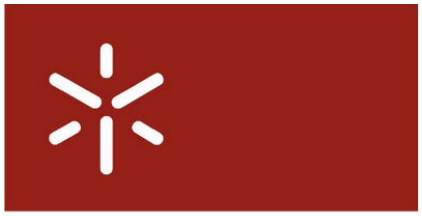


Universidade do Minho

Computação Gráfica

MIEI – 3º ano – 2º semestre

Universidade do Minho

Trabalho Prático - Parte 1

Nome	Nº Mecanográfico
António Jorge Monteiro Chaves	A75870
Carlos José Gomes Campos	A74745
Cesário Miguel Pereira Perneta	A73883
Luís Miguel Bravo Ferraz	A70824

Introdução

No âmbito da Unidade Curricular de Computação Gráfica, numa primeira fase, foi-nos proposto, o desenvolvimento de um gerador de vértices de algumas primitivas gráficas (plano, caixa, esfera e cone).

Para além disto, também foi desenvolvido um mecanismo de leitura de ficheiros de configuração em XML2, que tem como objetivo desenhar os vértices das primitivas gráficas anteriormente geradas, a partir de ficheiros escolhidos pelo utilizador.

Neste relatório incluímos uma descrição das várias etapas do trabalho, dando uma maior ênfase à descrição técnica do mesmo, com o recurso a equações, diagramas e figuras.

Fase 1: Primitivas Gráficas

1.1 Plano ZX

O desenho de um plano horizontal no eixo *zOx* é composto apenas por 2 triângulos. A chamada de geração de um plano requer a introdução de um número real, atribuído ao lado do plano. O valor calculado de $side = \frac{x}{2}$ permite-nos desenhar o plano com centro exato no meio do referencial, apenas variando os valores de x e z entre $[-side, side]$. De modo a que as figuras sejam visíveis pelo seu exterior, todos os triângulos devem ser desenhados tendo em conta a regra da mão direita.

```
 "plane":
try{
    x=(Double.parseDouble(args[1]))/2.0;
}catch(NumberFormatException e){
    System.out.println("Invalid Number");
}
sb.append(x+" 0 "+x+"\n");
sb.append(x+" 0 -"+x+"\n");
sb.append("-"+x+" 0 -"+x+"\n");
sb.append("-"+x+" 0 -"+x+"\n");
sb.append("-"+x+" 0 "+x+"\n");
sb.append(x+" 0 "+x+"\n");
```

Figura 1 Pontos do Plano

O desenho do plano na sua vista aérea (eixo *yy*) está representado no gráfico em baixo. Os pontos assinalados correspondem aos extremos do plano, com os quais serão formados os triângulos.

Para que o desenho seja visto desta perspetiva, a ordem pela qual os pontos dos triângulos devem ser escritos é:

- 1,2,3 Para o inferior;
- 2,4,3 Para o superior.

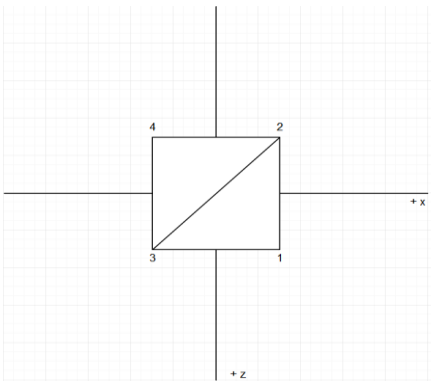


Figura 2 Gráfico do Plano

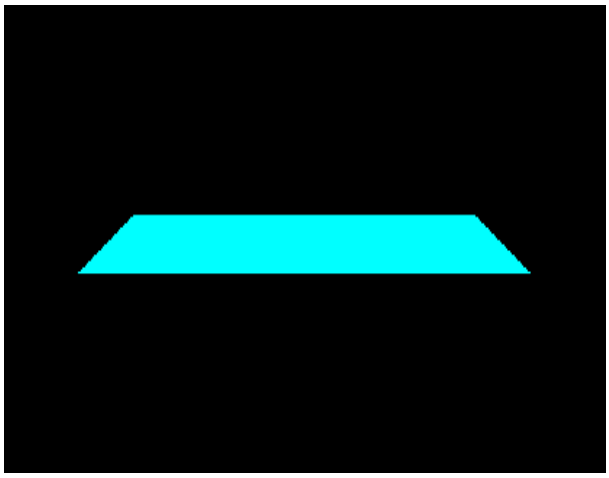


Figura 3 1ª Demo Scene Plano

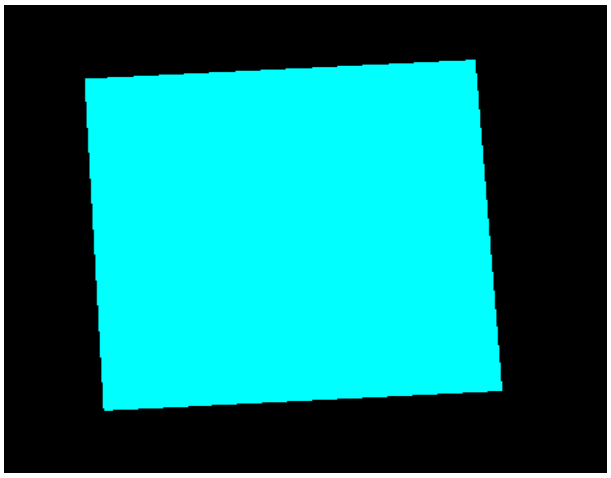


Figura 4 2ª Demo Scene Plano

1.2 Caixa

O desenho da caixa é dado por uma estratégia semelhante à do passo anterior. Uma caixa é composta por 6 planos, centrados na origem dos referenciais.

```
 "box":
try{
    x=(Double.parseDouble(args[1]))/2.0;
    y=(Double.parseDouble(args[2]))/2.0;
    z=(Double.parseDouble(args[3]))/2.0;
}catch(NumberFormatException e){
    System.out.println("Invalid Number");
}
// topo
sb.append(x+" "+y+" "+z+"\n");
sb.append(x+" "+y+" -"+z+"\n");
sb.append("-"+x+" "+y+" -"+z+"\n");
sb.append("-"+x+" "+y+" -"+z+"\n");
sb.append("-"+x+" "+y+" "+z+"\n");
sb.append(x+" "+y+" "+z+"\n");
// chao
sb.append(x+" -"+y+" "+z+"\n");
sb.append("-"+x+" -"+y+" "+z+"\n");
sb.append("-"+x+" -"+y+" -"+z+"\n");
sb.append("-"+x+" -"+y+" -"+z+"\n");
sb.append(x+" -"+y+" -"+z+"\n");
sb.append(x+" -"+y+" "+z+"\n");
// face esq
sb.append("-"+x+" -"+y+" "+z+"\n");
sb.append("-"+x+" "+y+" "+z+"\n");
sb.append("-"+x+" -"+y+" -"+z+"\n");
sb.append("-"+x+" -"+y+" -"+z+"\n");
sb.append("-"+x+" "+y+" "+z+"\n");
sb.append("-"+x+" "+y+" -"+z+"\n");
// face dir
sb.append(x+" -"+y+" -"+z+"\n");
sb.append(x+" "+y+" "+z+"\n");
sb.append(x+" -"+y+" "+z+"\n");
sb.append(x+" -"+y+" -"+z+"\n");
sb.append(x+" "+y+" -"+z+"\n");
sb.append(x+" "+y+" "+z+"\n");
// frente
sb.append(x+" -"+y+" "+z+"\n");
sb.append(" -"+x+" "+y+" "+z+"\n");
sb.append(" -"+x+" -"+y+" "+z+"\n");
sb.append(x+" -"+y+" "+z+"\n");
sb.append(x+" "+y+" "+z+"\n");
sb.append(" -"+x+" "+y+" "+z+"\n");
// tras
sb.append(x+" -"+y+" -"+z+"\n");
sb.append(" -"+x+" "+y+" -"+z+"\n");
sb.append(x+" "+y+" -"+z+"\n");
sb.append(x+" -"+y+" -"+z+"\n");
sb.append(" -"+x+" -"+y+" -"+z+"\n");
sb.append(" -"+x+" "+y+" -"+z+"\n");
```

Figura 5 Pontos Caixa

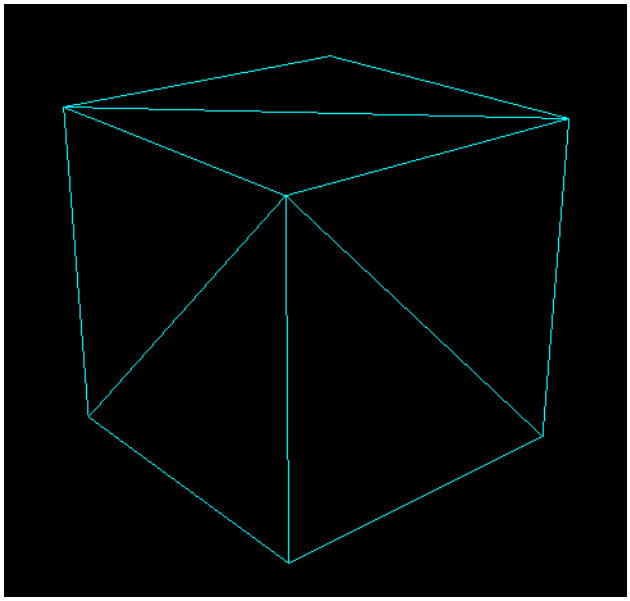


Figura 6 Demo Scene Caixa (GL_LINE)

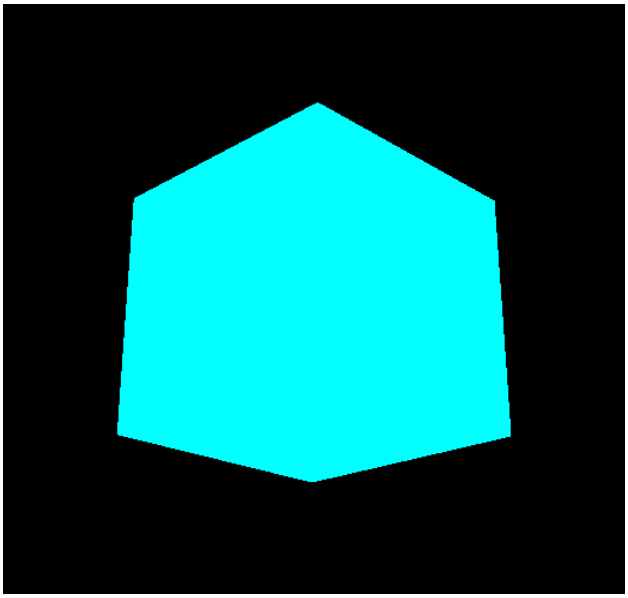


Figura 5 Demo Scene Caixa (GL_FILL)

Também é possivel definir a caixa como uma junção de várias caixas de tamanho inferior. Para esse efeito itera se pelas variáveis e para cada ponto são calculados 12 triângulos. A figura 6 exemplifica o código de três faces de cada Caixa contida dentro da Caixa principal.

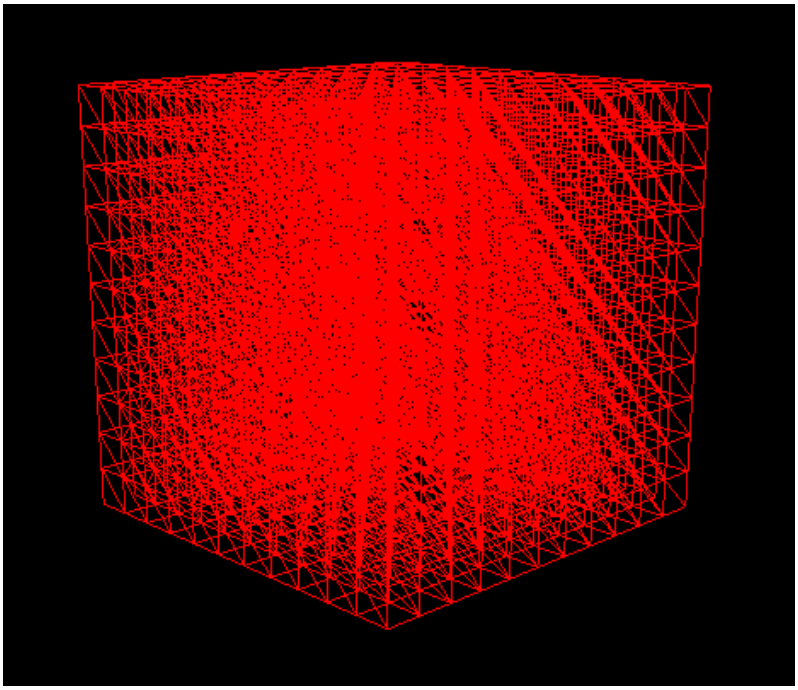


Figura 7 Demo Scene Caixa (iterativa)

```
Double varX=x/division, varY=y/division, varZ=z/division;
Double tmpx=0.0, tmpy=0.0, tmpz=0.0;
for(Double auxY=0.0;auxY<=y;auxY+=varY)
    for(Double auxX=0.0;auxX<=x;auxX+=varX)
        for(Double auxZ=0.0;auxZ<=z;auxZ+=varZ){
            tmpx=(auxX+varX);
            tmpy=(auxY+varY);
            tmpz=(auxZ+varZ);
            // topo
            sb.append(tmpx+" "+tmpy+" "+tmpz+"\n");
            sb.append(tmpx+" "+tmpy+" "+auxZ+"\n");
            sb.append(auxX+" "+tmpy+" "+auxZ+"\n");
            sb.append(auxX+" "+tmpy+" "+auxZ+"\n");
            sb.append(auxX+" "+tmpy+" "+tmpz+"\n");
            sb.append(tmpx+" "+tmpy+" "+tmpz+"\n");
            // chao
            sb.append(tmpx+" "+auxY+" "+tmpz+"\n");
            sb.append(auxX+" "+auxY+" "+tmpz+"\n");
            sb.append(auxX+" "+auxY+" "+auxZ+"\n");
            sb.append(auxX+" "+auxY+" "+auxZ+"\n");
            sb.append(tmpx+" "+auxY+" "+auxZ+"\n");
            sb.append(tmpx+" "+auxY+" "+tmpz+"\n");
            // face esq
            sb.append(auxX+" "+auxY+" "+tmpz+"\n");
            sb.append(auxX+" "+tmpy+" "+tmpz+"\n");
            sb.append(auxX+" "+auxY+" "+auxZ+"\n");
            sb.append(auxX+" "+auxY+" "+auxZ+"\n");
            sb.append(auxX+" "+tmpy+" "+tmpz+"\n");
            sb.append(auxX+" "+tmpy+" "+auxZ+"\n");
```

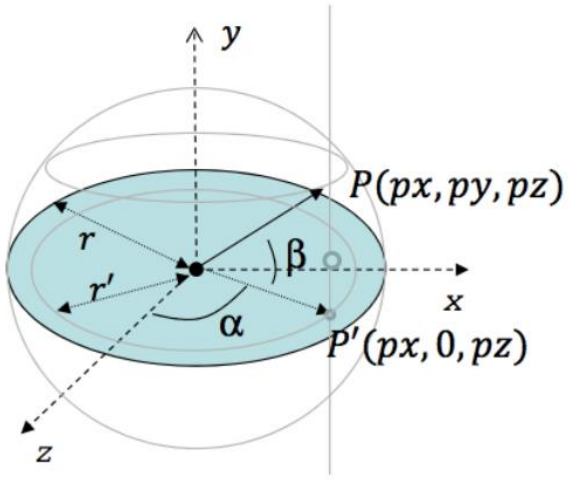
Figura 8 Pontos Caixa

1.3 Esfera

O desenho da esfera requer a discriminação prévia de 3 variáveis, representando o raio da esfera, o número de *slices* horizontais e o número de *stacks* verticais, que definem o número de pontos a calcular. Para utilizar coordenadas esféricas no cálculo dos pontos de cada triângulo, definem-se 2 ângulos – *alfa* e *beta*, em radianos - e 2 incrementos angulares para cada iteração:

```
"sphere":
try{
    radius=Double.parseDouble(args[1]);
    slices=Double.parseDouble(args[2]);
    stacks=Double.parseDouble(args[3]);
}catch(NumberFormatException e){
    System.out.println("Invalid Number");
}
alfa=0.0; beta=-Math.PI/2; varAlfa=2*Math.PI/slices; varBeta=Math.PI/stacks;
for(double st=0.0;st<=stacks;st++){
    beta=-Math.PI/2 + varBeta*st;
    for(double sl=0;sl<=slices;sl++){
        alfa=varAlfa*sl;
        sb.append(radius*Math.cos(beta)*Math.sin(alfa)+" "+radius*Math.sin(beta)+" "+radius*Math.cos(beta)*Math.cos(alfa)+"\n");
        sb.append(radius*Math.cos(beta+varBeta)*Math.sin(alfa+varAlfa)+" "+radius*Math.sin(beta+varBeta)+" "+radius*Math.cos(beta+varBeta)*Math.cos(alfa+varAlfa)+"\n");
        sb.append(radius*Math.cos(beta+varBeta)*Math.sin(alfa)+" "+radius*Math.sin(beta+varBeta)+" "+radius*Math.cos(beta+varBeta)*Math.cos(alfa)+"\n");
        sb.append(radius*Math.cos(beta)*Math.sin(alfa)+" "+radius*Math.sin(beta)+" "+radius*Math.cos(beta)*Math.cos(alfa)+"\n");
        sb.append(radius*Math.cos(beta)*Math.sin(alfa+varAlfa)+" "+radius*Math.sin(beta)+" "+radius*Math.cos(beta)*Math.cos(alfa+varAlfa)+"\n");
        sb.append(radius*Math.cos(beta+varBeta)*Math.sin(alfa+varAlfa)+" "+radius*Math.sin(beta+varBeta)+" "+radius*Math.cos(beta+varBeta)*Math.cos(alfa+varAlfa)+"\n");
    }
}
```

Figura 9 Pontos Esfera



1. $alfa = 0.0, alfa \in [0, 2\pi], varAlfa = \frac{2\pi}{slices}$
2. $beta = -\frac{\pi}{2}, beta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right], varBeta = \frac{2\pi}{stacks}$

Figura 10 Coordenadas Esféricas

A descrição de um ponto no espaço, partindo de coordenadas esféricas para cartesianas segue a seguinte transformação:

$$(alfa, beta, r) => \begin{cases} z = r * \cos(beta) * \sin(alfa) \\ x = r * \cos(beta) * \cos(alfa) \\ y = r * \sin(beta) \end{cases}$$

Sendo assim, dados os 2 ângulos e o raio da esfera, podemos calcular as coordenadas cartesianas de qualquer ponto, com a vantagem de se poder calcular pontos próximos, apenas incrementando o ângulo *alfa* ou *beta*. Para cada *Stack*, *alfa* é inicializado a 0 e percorre os pontos de acordo com o incremento *varAlfa*. Para cada ponto, é preciso calcular as coordenadas de 4 pontos.

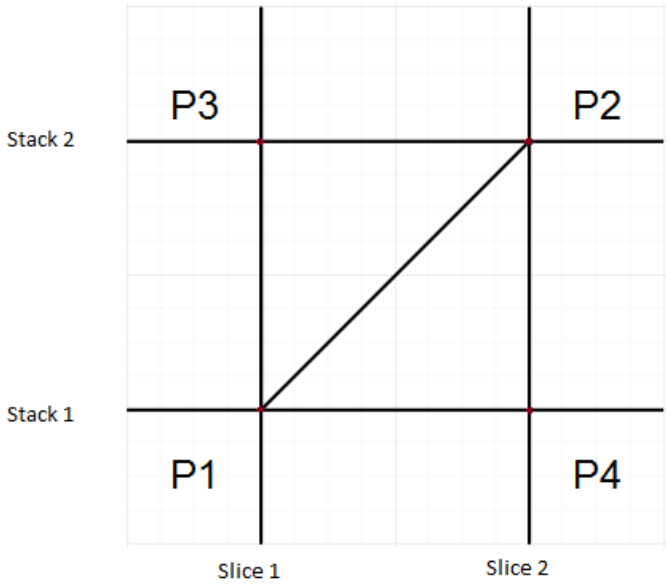


Figura 11 Pontos Triângulos

O primeiro ponto é o que coincide com a interseção da *slice* e *stack* iniciais de cada iteração, daí os valores das variáveis $alfa = alfa \ \& \ beta = beta$.

$$P1 => \begin{cases} z = r * \cos(beta) * \sin(alfa) \\ x = r * \cos(beta) * \cos(alfa) \\ y = r * \sin(beta) \end{cases}$$

O segundo ponto situa-se no vértice oposto ao primeiro, na interseção da *slice* e *stack* seguintes. O cálculo de (*x,y,x*) faz-se agora com $afla = alfa + varAlfa \ \& \ beta = beta + varBeta$.

$$P2 => \begin{cases} z = r * \cos(beta + varBeta) * \sin(alfa + varAlfa) \\ x = r * \cos(beta + varBeta) * \cos(alfa + varAlfa) \\ y = r * \sin(beta + varBeta) \end{cases}$$

Relativamente ao cálculo das coordenadas do terceiro ponto, pode observar-se que, em comparação com P1, se mantém o valor do ângulo horizontal *alfa*, enquanto que o valor de *beta* corresponde ao da próxima *stack*. O processo é análogo para P4, que no seu caso vê *alfa* a ser incrementado.

$$P3 => \begin{cases} z = r * \cos(beta + varBeta) * \sin(alfa) \\ x = r * \cos(beta + varBeta) * \cos(alfa) \\ y = r * \sin(beta + varBeta) \end{cases}$$

$$P4 => \begin{cases} z = r * \cos(beta) * \sin(alfa + varAlfa) \\ x = r * \cos(beta) * \cos(alfa + varAlfa) \\ y = r * \sin(beta) \end{cases}$$

Os 4 pontos calculados permitem desenhar 2 triângulos por ponto, por ordem P1, P2, P3 e P1, P4, P2. O programa gera para cada *stack* um "anel" de triângulos, ou seja, para cada iteração em *beta*, *alfa* percorre os pontos todos do anel.

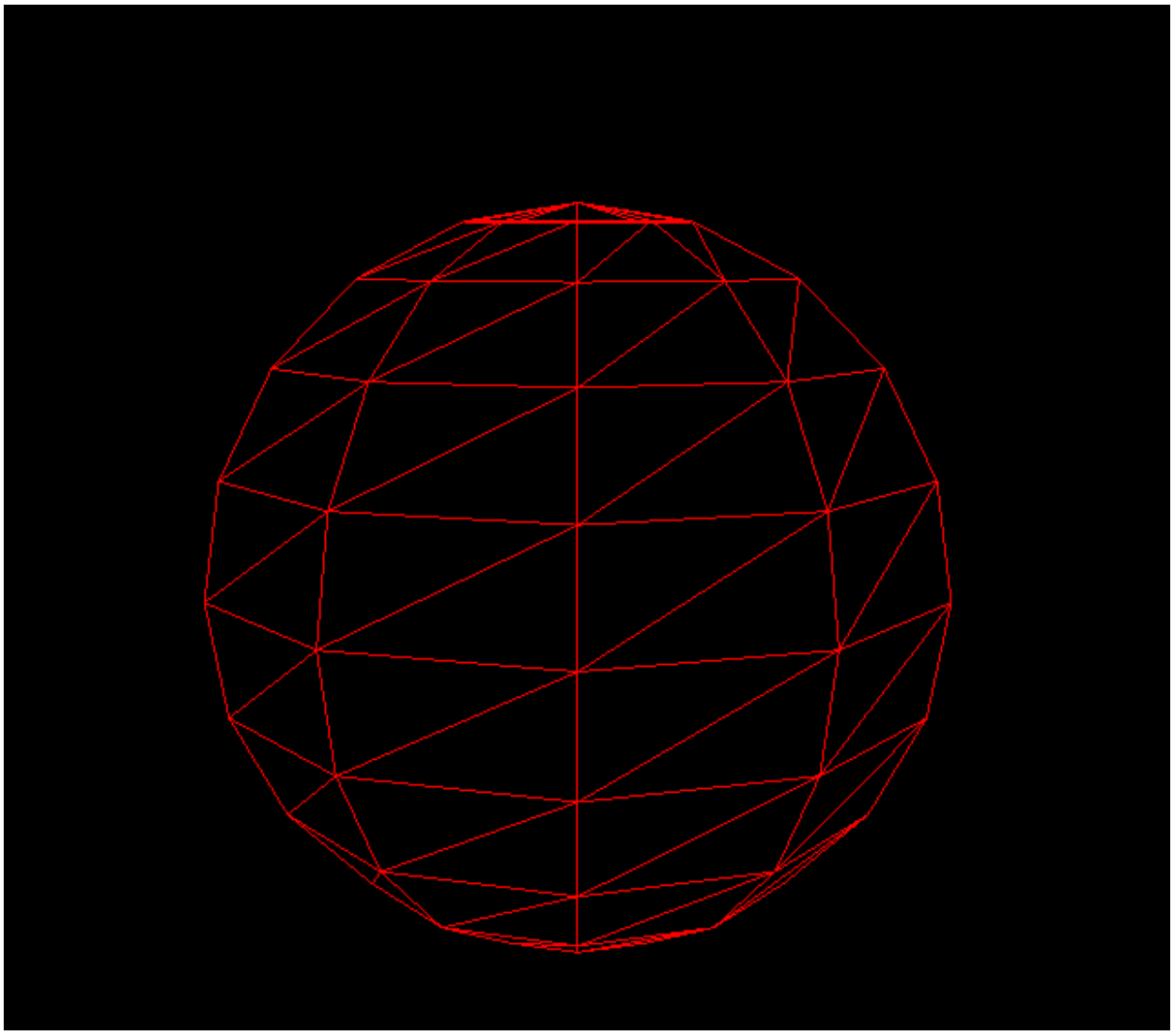


Figura 12 Demo Scene Esfera (GL_LINE)

1.4 Cone

Por último, o desenho do cone parte da transformação de variáveis polares em variáveis cartesianas, passando a altura do cone no parâmetro *y*. O conjunto de pontos calculados origina a face cónica do cone. Assim como no ponto anterior, a estrutura é percorrida gerando um anel de triângulos para cada *stack.* O valor da altura é fornecido aquando da compilação do ficheiro, junto com os valores do raio da base, número de *stacks* e número de *slices*.

```
 "cone":
try{
    radius=Double.parseDouble(args[1]);
    height=Double.parseDouble(args[2]);
    slices=Double.parseDouble(args[3]);
    stacks=Double.parseDouble(args[4]);
}catch(NumberFormatException e){
    System.out.println("Invalid Number");
}
alfa=0.0;
double cheight=0.0;
double raio=radius;
double varHeight=height/stacks;
double varRadius=radius/stacks;
double varAngle=2*Math.PI/slices;
//topo
for(double ch=0;(ch<height&&raio>0);ch+=varHeight){
    for(double ca=0;ca<=2*Math.PI;ca+=varAngle){
        alfa+=ca*varAngle;
        sb.append((raio-varRadius)*Math.sin(ca)+" "+(cheight+varHeight)+" "+(raio-varRadius)*Math.cos(ca)+"\n");
        sb.append(raio*Math.sin(ca)+" "+(cheight)+" "+raio*Math.cos(ca)+"\n");
        sb.append((raio-varRadius)*Math.sin(ca+varAngle)+" "+(cheight+varHeight)+" "+(raio-varRadius)*Math.cos(ca+varAngle)+"\n");
        sb.append(raio*Math.sin(ca+varAngle)+" "+(cheight)+" "+raio*Math.cos(ca+varAngle)+"\n");
        sb.append((raio-varRadius)*Math.sin(ca+varAngle)+" "+(cheight+varHeight)+" "+(raio-varRadius)*Math.cos(ca+varAngle)+"\n");
        sb.append(raio*Math.sin(ca)+" "+(cheight)+" "+raio*Math.cos(ca)+"\n");
    }
    cheight+=varHeight;
    raio-=varRadius;
    alfa=0.0;
}
//base
double baseRad=radius;
double baseheight=0.0;
for(double baseAng=0;baseAng<2*Math.PI;baseAng+=varAngle){
    sb.append("0.0 0.0 0.0\n");
    sb.append(baseRad*Math.sin(baseAng+varAngle)+" "+baseheight+" "+baseRad*Math.cos(baseAng+varAngle)+"\n");
    sb.append(baseRad*Math.sin(baseAng)+" "+baseheight+" "+baseRad*Math.cos(baseAng)+"\n");
}
```

Figura 13 Pontos Cone

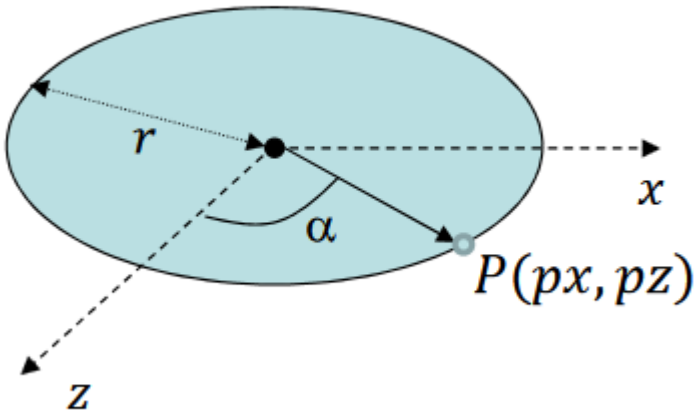


Figura 14 Coordenadas Polares

Os valores iniciais atribuídos às variáveis de altura e ângulo são os seguintes

$$height = 0.0, varHeight = height/stacks$$

$$alfa = 0.0, varAlfa = \frac{2\pi}{stacks}$$

A construção da primeira parte da figura calcula os pontos dos triângulos por uma ordem igual à das esferas, tendo em conta a transformação, neste caso, a partir de coordenadas polares.

$$(alfa, r) = \begin{cases} x = r * \cos(alfa) \\ y = height \\ z = r * \sin(alfa) \end{cases}$$

Os 4 pontos calculados para cada iteração de *alfa* são agora dados por:

$$P1 = \begin{cases} x = r * \cos(alfa) \\ y = height \\ z = r * \sin(alfa) \end{cases}$$

$$P2 = \begin{cases} x = (r - varRadius) * \cos(alfa + varAlfa) \\ y = height + varHeight \\ z = (r - varRadius) * \sin(alfa + varAlfa) \end{cases}$$

$$P3 = \begin{cases} x = (r - varRadius) * \cos(alfa) \\ y = height + varHeight \\ z = (r - varRadius) * \sin(alfa) \end{cases}$$

$$P4 = \begin{cases} x = r * \cos(alfa + varAlfa) \\ y = height \\ z = r * \sin(alfa + varAlfa) \end{cases}$$

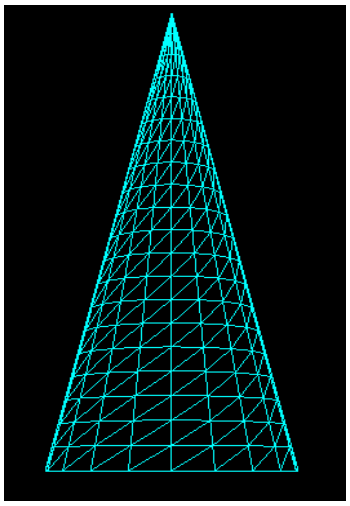


Figura 16 Demo Scene Cone

Observando o cone, pudemos afirmar que sempre que se pretende calcular um ponto que se encontra uma *stack* acima do original (pontos P2 e P3) é necessário incrementar a altura e decrementar o raio, ambos em proporção. O ângulo *alfa*, por sua vez, a cada ciclo da função oscila entre 0 e 2 π.

O desenho da base do cone faz-se partindo do centro da figura, juntando, para cada iteração, o ponto de ordem original, precedido do ponto de ordem seguinte.

$$P1 = \begin{cases} x = 0.0 \\ y = 0.0 \\ z = 0.0 \end{cases}$$

$$P2 = \begin{cases} x = r * \cos(alfa + varAlfa) \\ y = 0.0 \\ z = r * \sin(alfa + varAlfa) \end{cases}$$

$$P3 = \begin{cases} x = r * \cos(alfa) \\ y = 0.0 \\ z = r * \sin(alfa) \end{cases}$$

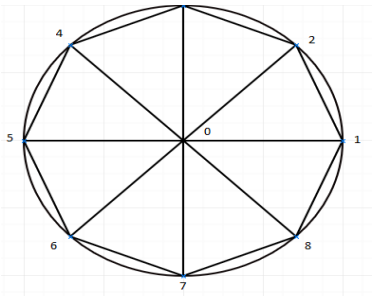


Figura 15 Exemplo de divisão do círculo

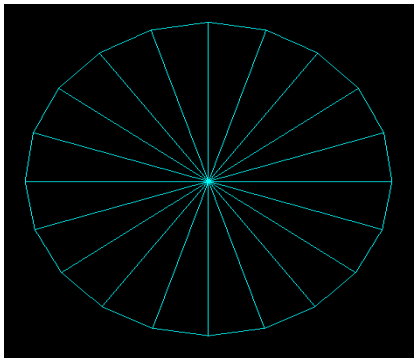


Figura 17 Exemplo de divisão do círculo

1.5 Motor

O ficheiro *Engine.h* tem como função a leitura dos pontos previamente gerados nos ficheiros ".3d". A função *xml parser* começa por ler o ficheiro *config.xml* e retira a informação das figuras geométricas a desenhar. Para cada ficheiro, são lidas linha a linha as coordenadas *x,y e z*, e guardados numa instância da *class Point*, que por sua vez é adicionada a uma instância da *class Model*. No final da função o modelo é guardado num vetor invocado na função *processModels* que, para uma lista de modelos já compilados desenha os respetivos triângulos, através da sua chamada na função *renderScene*.

Após o desenho das primitivas das figuras, foram inseridos Menus que permitem alterar a o preenchimento e cor das mesmas. Para aceder ao Menu basta apenas carregar na figura com o botão direito do rato.

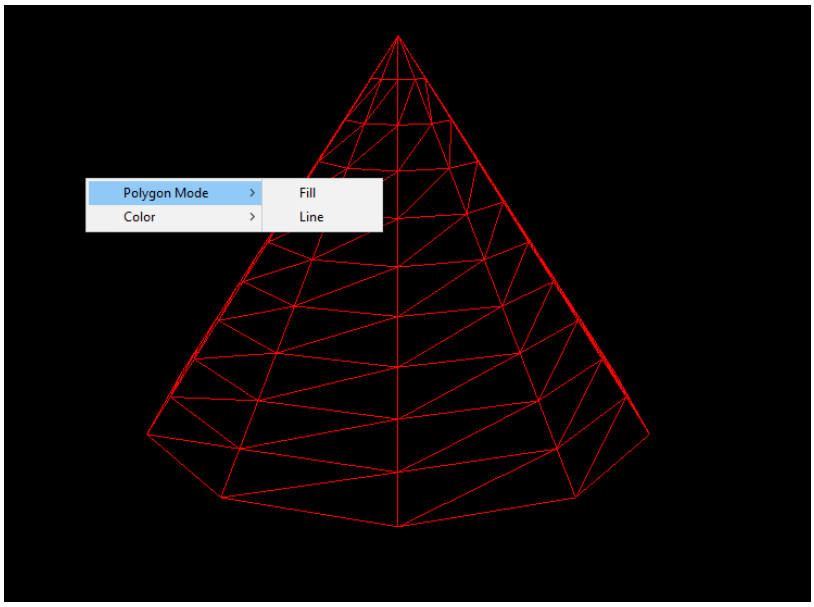


Figura 18 Exemplo Menu