Universidade do Minho

Mestrado Integrado em Engenharia Informática

Licenciatura em Ciências da Computação

Unidade Curricular de Computação Gráfica

Phase 1 – Graphical primitives

André Araújo - a87987

Carlos Ferreira - a87953

Daniel Ribeiro - a87994

Paulo Costa - a87986

Novembro, 2020

Data de Receção	
Responsável	
Avaliação	
Observações	

Phase 1 – Graphical primitives

André Araújo - A87987

Carlos Ferreira - A87953

Daniel Ribeiro - A87994

Paulo Costa - A87986

Março, 2021

<</opcional Dedicatória>>

Resumo

Neste documento, iremos abordar a primeira fase da avaliação prática da Unidade Curricular de Computação Gráfica. Nesta UC iremos ter 4 fases, nas quais iremos ser avaliados na componente prática.

Nesta primeira, iremos abordar 2 sistemas, denominados por Generator e Engine. Teremos de ter conta ainda, o desenho de 4 possíveis figuras geométricas, a utilização do GLUT, num dos sistemas acima referidos e ainda a leitura de um ficheiro, escrito em XML.

Palavras-Chave: Computação gráfica, Generator, Engine, figuras geométricas, GLUT, XML

Índice

Introdução

No âmbito da unidade curricular de Computação Gráfica foi proposta, numa primeira fase, que se realizassem dois sistemas. Primeiramente, um que guardasse num ficheiro informações relativas a uma figura que se pretenderia desenhar, usando o **GLUT.** E um segundo que, lendo um ficheiro, escrito em XML, desenharia o modelo segundo as indicações geradas anteriormente. Concluindo-se assim com a figura pretendida desenhada.

Em específico, as figuras pretendidas serão entre um plano, uma caixa (box), um cone e uma esfera.

Estrutura do Projeto:

O nosso projeto está dividido nos seguintes diretórios:

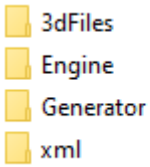

Figura 1- Diretórios

Nesta secção vamos explicar cada um deles e qual a sua função, sendo 2 deles para guardar ficheiros e outros 2 os directórios mais importantes onde temos as classes principais, o generator e o engine, que fazem todo o trabalho de geração e leitura dos pontos.

3dFiles

Na pasta "3dFiles", temos os ficheiros 3d gerados pelo generator, sempre que geramos algum ficheiro .3d este é automaticamente criado nessa mesma pasta, no xml que o engine lê, vamos direcionar-lho para ler o ficheiro .3d dessa pasta também.

1. Generator:

Na directoria Generator, temos lá a importante classe generator, a qual vai gerar o ficheiro.3d.

Para usar o Generator apenas temos de escrever na consola o seguinte:

$ ./Generator [objecto] [argumentos] [nome do ficheiro 3d]

Onde o objeto pode ser:

- Plane
- Box
- Sphere
- Cone

E os argumentos são os seguintes:

- Plane: [dimensão]
- Box: [x] [y] [z] [slices (opcional)]
- Sphere: [raio] [slices] [stacks]
- Cone: [raio] [altura] [slices] [stacks]

Assim um exemplo de execução do Generator seria:

$ ./Generator Sphere 5 10 20 Sphere.3d

Com esta execução seria gerado um ficheiro chamado Sphere.3d na pasta 3dFiles, este ficheiro iria conter os pontos de uma esfera com 5 de raio, 10 slices e 20 stacks pronto para ser lido pelo Engine.

O ficheiro .3d gerado na primeira linha vai ter o número de vértices gerados, a qual nos vai auxiliar na leitura do ficheiro na classe Engine, as outras linhas vão ser os pontos gerados escritos 1 ponto por linha com espaços a separar cada coordenada desta forma um ponto (x1, y1, z1) e outro ponto (x2, y2, z2) ficaria escrito no ficheiro da forma:

- x1 y1 z1
- x2 y2 z2
- (...,...,...)

Para gerar estes pontos o Generator faz uma série de cálculos para cada descobrir os pontos um dos objectos para depois escrevê-los no ficheiro.

1.1. Plano:

Para o desenho do plano quadrado em XZ é passado como argumento o tamanho do lado (size).

Este plano será constituído por 2 triângulos simétricos com 2 vértices coincidentes, e serão colocados de forma a que o plano esteja centrado na origem, como estamos a trabalhar no plano XZ todas as coordenadas y dos pontos serão 0.

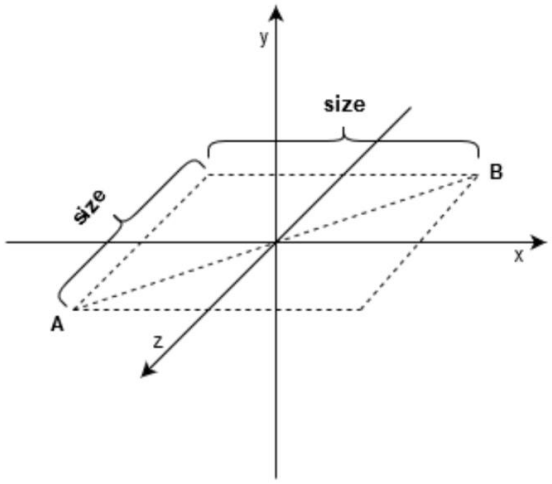

Figura 2- Esquema Plano I

Construímos os triângulos de forma a que sejam visíveis de cima, ou seja, os vértices foram dados por ordem contrária aos ponteiros do relógio:

Triângulo 1:

A = (size/2 , 0 , size/2)

B = (size/2 , 0 , -size/2)

C = (-size/2 , 0 , -size/2)

Triângulo 2:

A = (size/2 , 0 , size/2)

C = (-size/2 , 0 , -size/2)

D = (-size/2 , 0 , size/2)

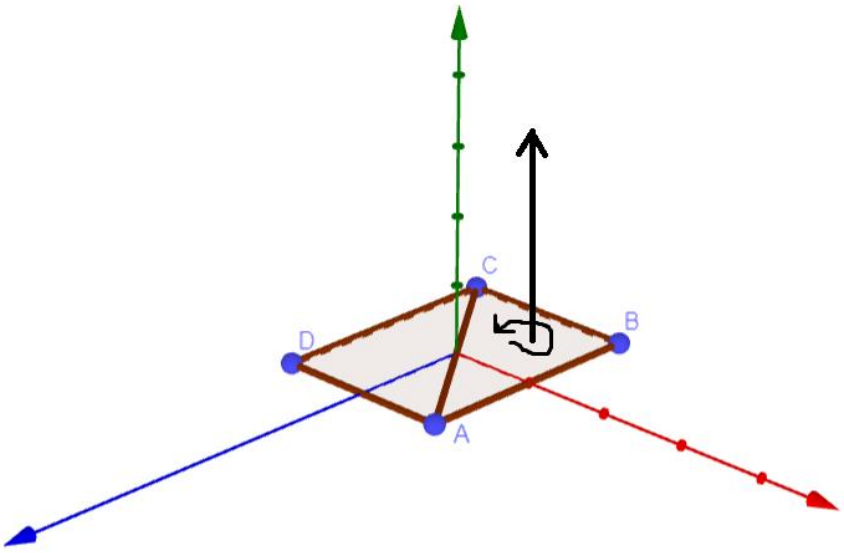

Figura 3- Esquema Plano II

1.2. Box:

A box consiste num cubo com 3 parâmetros: comprimento (x), altura(y), largura(z), mas neste exercício ainda com um parâmetro adicional designado por divisão (div), que tem como por objetivo delinear a divisão de cada face em uma matriz (div,div). Como, por exemplo, uma box com 3 divisões referir-se-á a um um cubo em que cada uma das faces, está dividida numa matriz (3x3).

Como se pode observar na figura, o raciocínio por detrás da construção de cada face da box, necessita de 3 variáveis que implementamos (pX, pY, pZ), que se obtêm a partir dos parâmetros de entrada x, y, z pelo número de divisões.

Vamos então abordar um caso, em que o número de divisões recebido é 2. Então, teremos a box, dividida em cada uma das suas faces como uma matriz (2x2), em que em cada uma destas divisões teremos 2 triângulos.

Inicialmente, colocamos as varáveis x e z com o valor da sua metade, isto para que fosse mais fácil fazer a sua divisão no referencial e fazer com que ele ficasse centrado e com a sua base em XZ. E utilizamos as 3 variáveis "extra", para nos facilitar realizar as divisões dos triângulos corretamente.

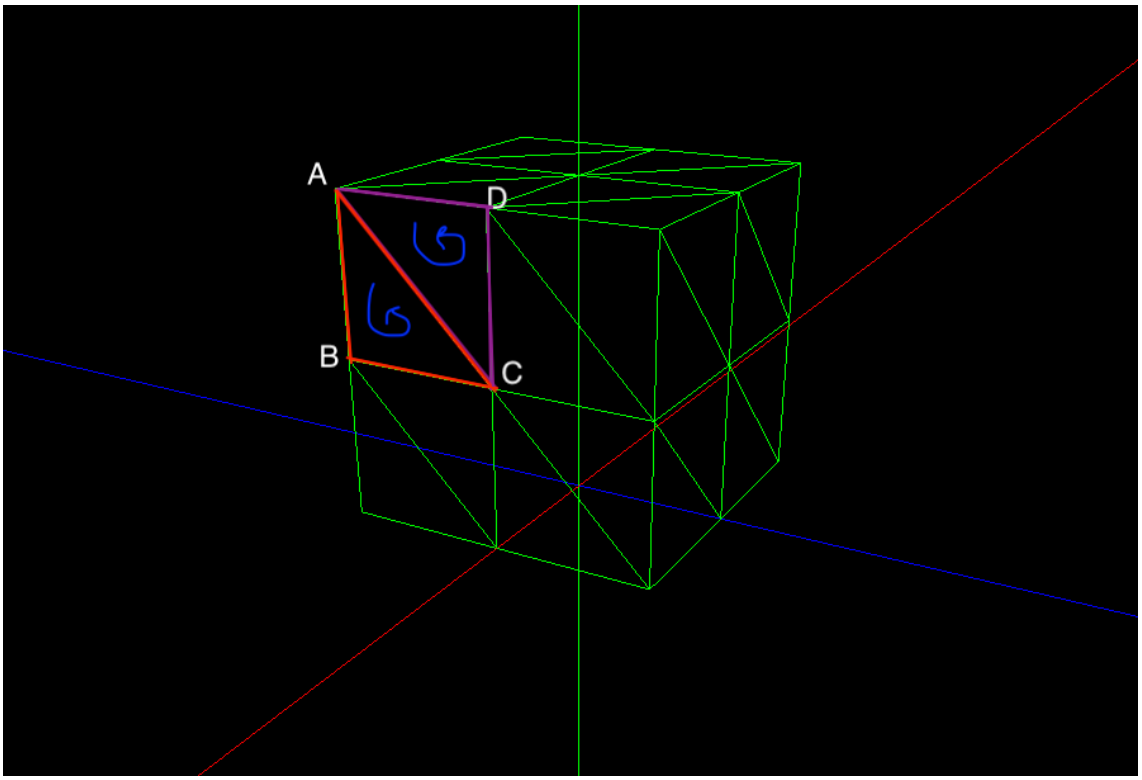

Figura 4- Esquema Box I

Na figura, temos o exemplo da face frontal desta box. Podemos ver que estes 2 triângulos são simétricos e têm 2 vértices coincidentes, e neste caso, terão sempre a coordenada do eixo dos Z como positiva, variando assim apenas o valor dos eixos restantes.

Construímos os triângulos de forma a que sejam visíveis de cima, ou seja, os vértices foram dados por ordem contrária aos ponteiros do relógio:

Triângulo 1:

A = (-x +(i*pX), y-(j*pY), z)

B = (-x +(i*pX), y-(j*pY) -pY, z)

C = (-x +(i*pX) +pX, y-(j*pY) -pY, z)

Triângulo 2:

A = (-x +(i*pX), y-(j*pY), z)

C = (-x +(i*pX) +pX, y-(j*pY) -pY, z)

D= (-x +(i*pX) +pX, y-(j*pY), z)

Obviamente, as variáveis i e j são criadas para fazer variar a zona da divisão que nos encontramos. Isto é, estamos a percorrer estas criações dentro de dois ciclos, um ciclo que incrementa a variável i até esta atingir o valor das divisões e outro ciclo com a variável j, que realiza o mesmo algoritmo. Com isto, temos as criações dos triângulos de cima para baixo e da esquerda para a direita.

Aqui podemos ver que as variáveis que tivemos de implementar, nos ajudam muito mesmo no que toca a “brincar” com as coordenadas de cada divisão, pois ao somar-mos estas ao valor atual da variável em questão, saltamos de divisão em divisão dessa coordenada.

Vamos ainda, apresentar mais um exemplo mas desta vez relativamente à face superior da box. Aqui teremos a variável y, relativa à altura da figura estática e sempre positiva. Enquanto, as restantes variam tal como vamos passar a mostrar.

Mais uma vez, faremos a construção dos diversos triângulos de cima para baixo e da esquerda para a direita.

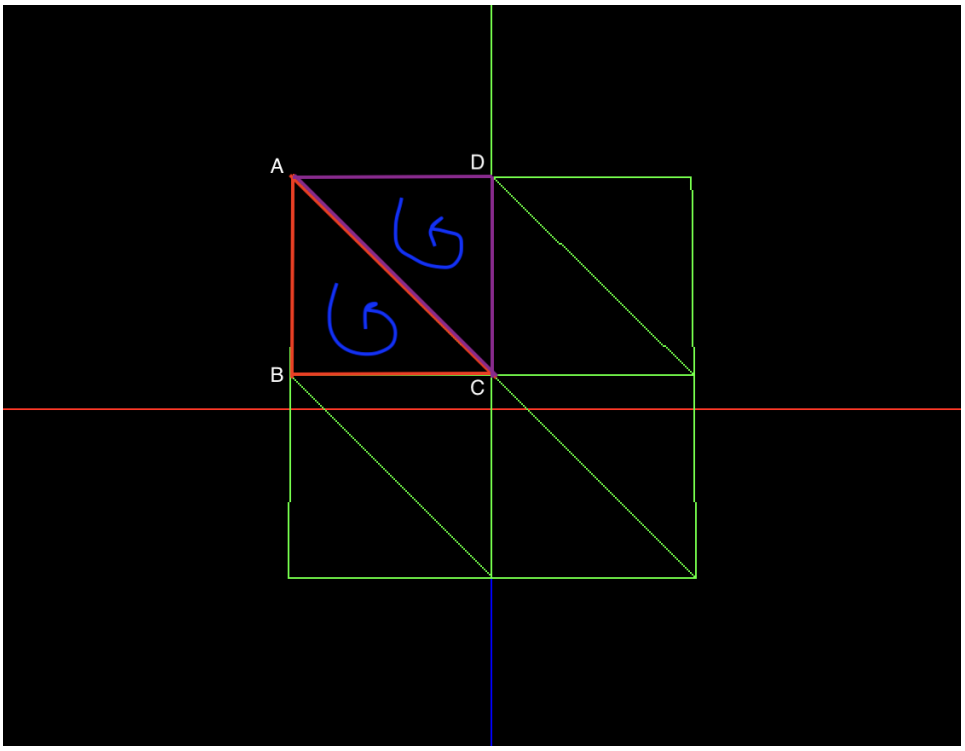

Figura 5- Esquema Box II

Triângulo 1:

A = (-x +(i*pX), y, -z +(j*pZ))

B = (-x +(i*pX), y, -z +(j*pZ) +pZ)

C = (-x +(i*pX) +pX, y, -z +(j*pZ) +pZ)

Triângulo 2:

A = (-x +(i*pX), y, -z +(j*pZ))

C = (-x +(i*pX) +pX, y, -z +(j*pZ) +pZ)

D= (-x +(i*pX) +pX, y, -z +(j*pZ))

Assim, podemos ver que o processo, acaba por ser semelhante em ambas as faces. Acaba por ter diferenças, mas depois de percebermos como temos de olhar para as coordenadas e como o método funciona, fica tudo mais fácil.

Concluindo, achamos que no início foi mais “chato” perceber como deveríamos abordar as divisões, mas um pensamento geométrico e uns desenhos à parte facilitaram a tarefa e achamos que, no fim do dia, foi completa com êxito.

1.3. Cone:

Para o desenho do cone são preciso como argumentos: o raio da base (radius), a altura do cone (height), número de fatias (slices) iguais em que se vai dividir a base, e por fim, número de camadas (stacks).

Construiremos o cone stack a stack, e em cada stack, slice a slice. Cada slice da base equivale a 1 triângulo, cada slice das stacks equivale a dois triângulos, e cada slice da última stack do cone (topo) equivale a 1 triângulo.

Podemos dividir o processo em 3 passos:

1. Base
2. Meio do cone
3. Topo

1.Base:

É essencial calcular o ângulo de cada slice ($\alpha = 2\pi$ /slices) e a altura de cada stack (height/stacks).

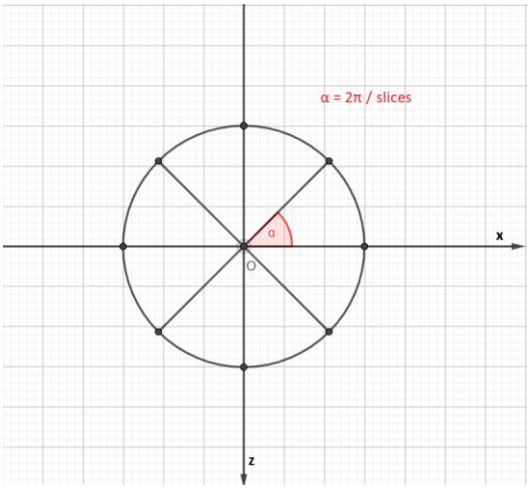

Figura 7- Esquema Cone I

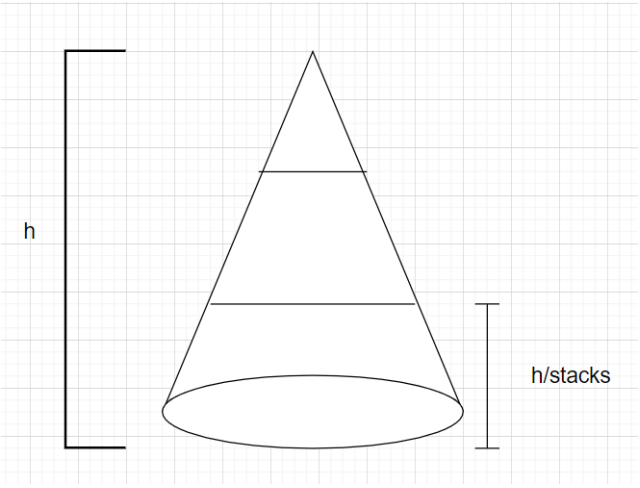

Figura 6- Esquema Cone II

Para todos os triângulos da base é fácil perceber que um dos vértices será a origem (0,0,0), e que os dois outros vértices serão obtidos a partir das coordenadas polares, onde:

px = radius * sin (alpha)

pz = radius * cos (alpha)

Sabemos que os 2 vértices devem sempre estar a uma amplitude **alpha** um do outro, por isso teremos duas variáveis para guardar os ângulos.

alpha1

alpha2 = alpha1 + **alfa**

Como queremos apenas ver a base de baixo, criámos os vértices do triângulo pela orientação dos ponteiros do relógio.

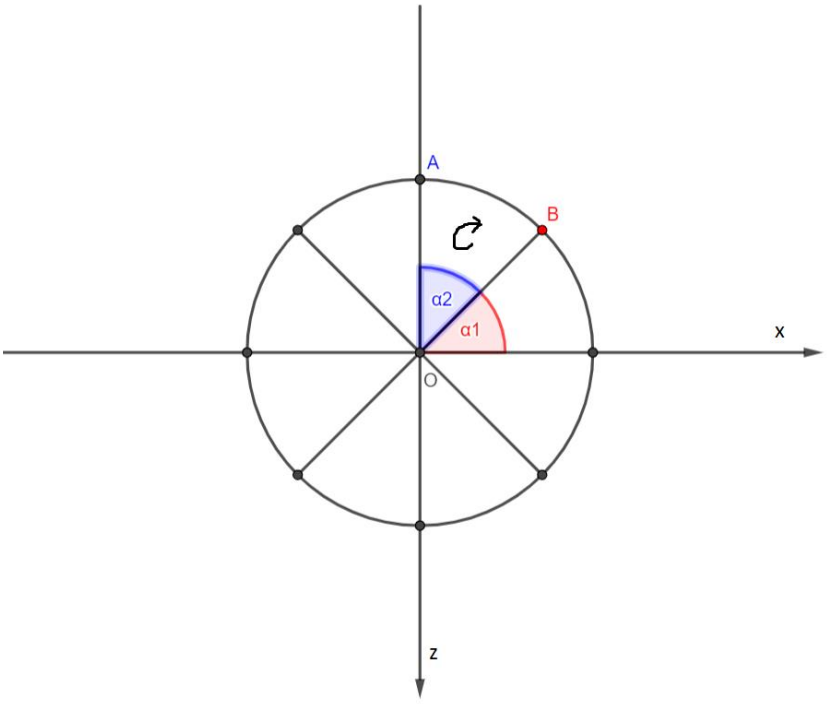

Figura 8- Esquema Cone III

Tal como o triângulo OAB da imagem acima, todos os triângulos serão então da forma:

V1 = (0,0,0)

V2 = (radius * sin(alpha2) ,0, radius * cos(alpha2))

V3 = (radius * sin(alpha1) ,0, radius * cos(alpha1))

Nesta etapa aproveitamos também para construir a primeira stack, pois já temos quase toda a informação que necessitamos.

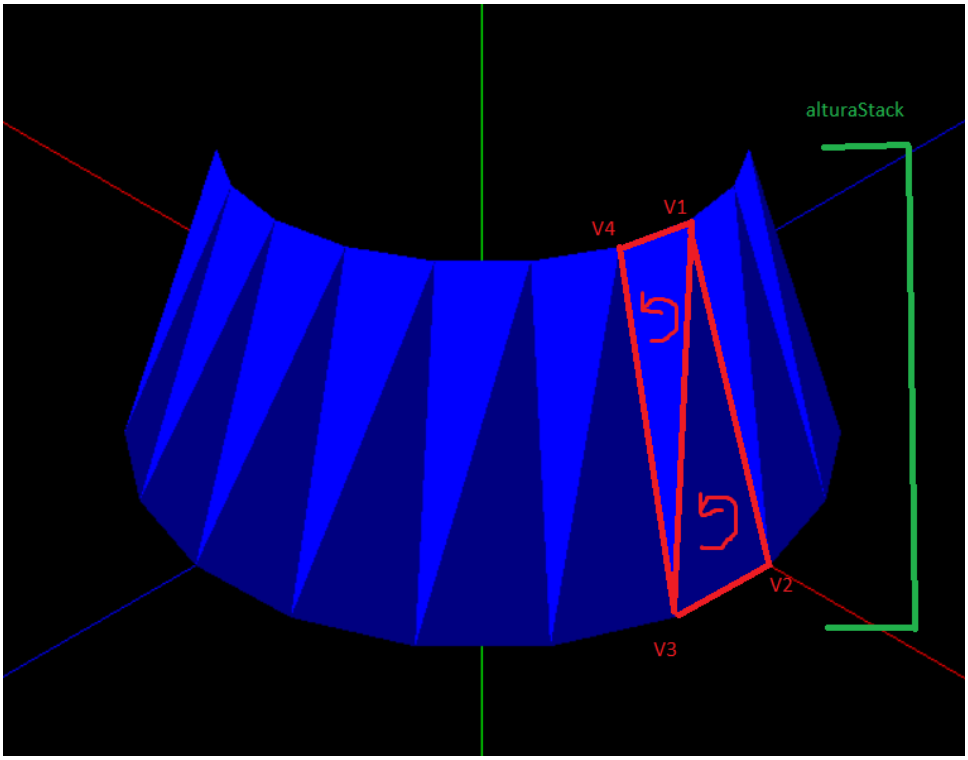

Figura 9- Esquema Cone IV

Sabemos já os vértices V2 e V3 da construção da base, e é fácil descobrir V1 e V4 , logos os dois triângulos são criados da forma:

Triângulo 1:

V2 = (radius * sin(alpha2), 0, radius * cos(alpha2))

V1 = (raio2 * sin(alpha2), alturaCima , raio2 * cos(alpha2))

V3 = (radius * sin(alpha1), 0, radius * cos(alpha1))

Triângulo 2:

V3 = (radius * sin(alpha1), 0, radius * cos(alpha1))

V1 = (raio2 * sin(alpha2), alturaCima, raio2 * cos(alpha2))

V4 = (raio2 * sin(alpha1), alturaCima, raio2 * cos(alpha1))

2. Meio do cone

Nesta etapa criámos todas as stacks menos a última, pois a última por este método apesar de funcionar cria sobreposição de pontos, por isso é melhor separar das do meio.

Como os triângulos das stacks envolvem informação de duas stacks, ou seja, o topo de uma stack será a base da stack superior e a base de uma stack será o topo da stack inferior. (isto no meio do cone, que é onde estámos a trabalhar).

Variáveis usadas:

raio2 -> guarda o raio da base da stack superior

raio2ant -> guarda o raio do teto da stack inferior

alturaCima -> guarda a altura da stack base da stack superior

alturaBaixo -> guarda a altura do teto da stack inferior

A cada stack o raio2 é calculado de forma proporcional ao número de stacks, ou seja,

raio2 = raio2 - (radius/stacks).

Os triângulos são criados da seguinte forma:

Triângulo 1:

V2 = (raio2ant * sin(alpha2), alturaBaixo, raio2ant * cos(alpha2))

V1 = (raio2 * sin(alpha2), alturaCima, raio2 * cos(alpha2))

V3 = (raio2ant * sin(alpha1), alturaBaixo, raio2ant * cos(alpha1))

Triângulo 2:

V3 = (raio2ant * sin(alpha1), alturaBaixo, raio2ant * cos(alpha1))

V1 = (raio2 * sin(alpha2), alturaCima, raio2 * cos(alpha2))

V4 = (raio2 * sin(alpha1), alturaCima, raio2 * cos(alpha1))

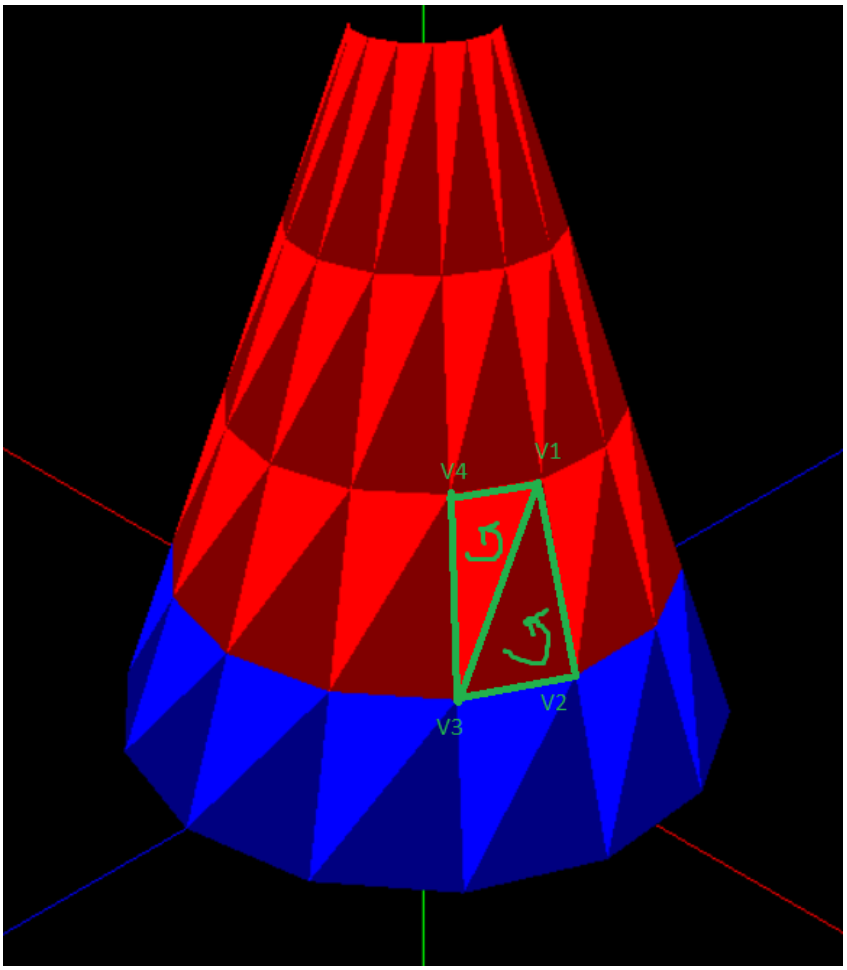

Figura 10- Esquema Cone V

3. Topo

Para os triângulos da última stack, é fácil perceber que, um dos vértices de todos os triângulos será (0,height,0). Os outros 2 vértices utilizam informação da stack inferior.

Sendo assim criados:

Triângulo:

V3 = (raio2ant * sin(alpha2), alturaBaixo, raio2ant * cos(alpha2));

V1 = (0, height, 0);

V2= (raio2ant * sin(alpha1), alturaBaixo, raio2ant cos(alpha1));

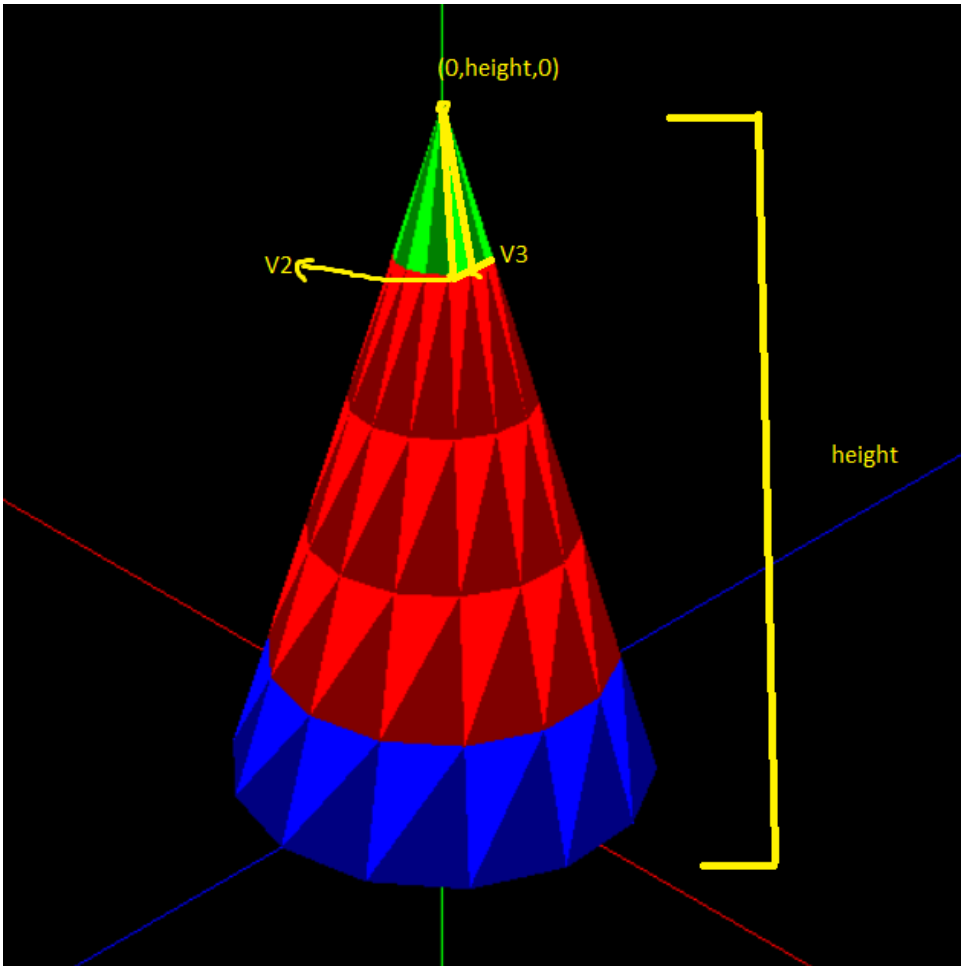

Figura 11- Esquema Cone Vl

1.4. Esfera:

Para o desenho da esfera são preciso como argumentos: o raio da base (radius), o número de fatias (slices) iguais em que se vai dividir a base, e por fim, número de camadas (stacks).

Para descobrir as coordenadas dos pontos utilizamos coordenadas esféricas (Spherical Coordinates), onde como todos as fatias e camadas têm distanciamento igual:

- alpha = 2π/slices
- beta = π/stacks

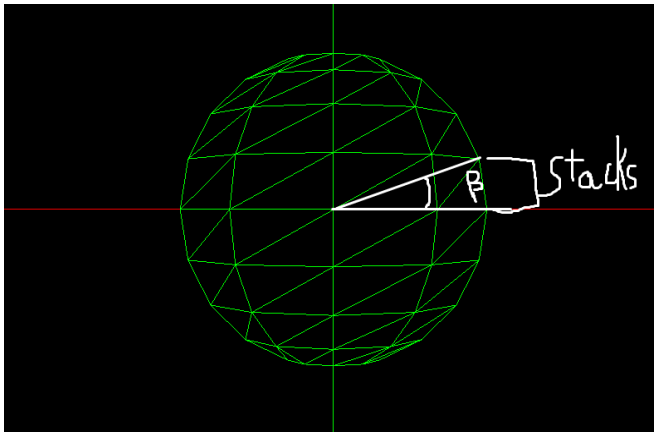

Figure 12- Esquema Sphere I

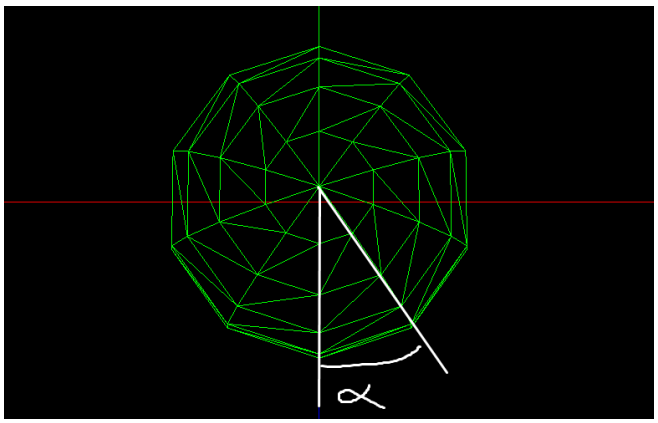

Figure 13- Esquema Sphere II

E transformando em coordenadas cartesianas temos:

x = radius * cos (beta) * sin (alpha)

y = radius * cos (beta) * cos (alpha)

z = radius * sin (beta)

Desta forma já conseguimos descobrir as coordenadas cartesianas dos pontos.

Para desenhar os triângulos percorremos cada fatia e começamos por desenhar dois triângulos um para a camada superior e outro para a camada inferior. Caso o número de camadas (stacks) seja superior a 2 percorremos as camadas intermedias e desenhamos dois triângulos de forma a formar um quadrado.

Camada superior:

V1 = (radius * cos(π/2) * sin(alpha1), radius * sin(π /2), radius * cos(π /2) *cos(alpha1))

V2 = (radius * cos(π /2-beta) * sin(alpha1),radius * sin(π/2-beta),radius * cos(π/2-beta) * cos(alpha1))

V3 = (radius * cos(π /2 - beta) * sin(alpha1+alpha),radius * sin(π /2-beta), radius * cos(π /2 - beta) * cos(alpha1+alpha))

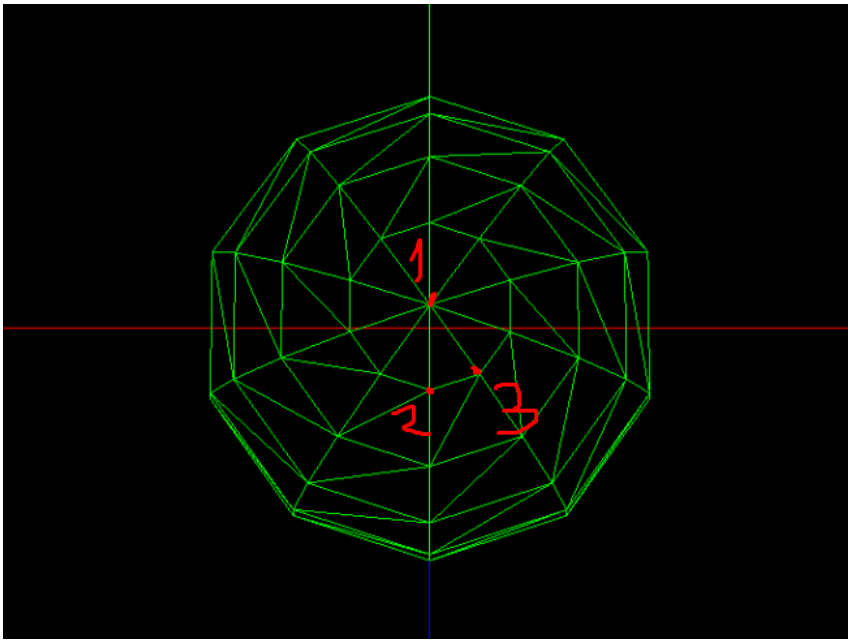

Figure 14- Esquema Sphere III

Camada inferior:

V4 = (radius * cos(-π /2) * sin(alpha1), sin(-π /2) * radius, radius * cos(-π /2) * cos(alpha1))

V5 = (radius * cos((-π /2) + beta) * sin(alpha1 + alpha), radius * sin((-π / 2) + beta), radius * cos((-π / 2) + beta) * cos(alpha1 + alpha))

V6 = (radius * cos((-π /2) + beta) * sin(alpha1), radius * sin((-π / 2) + beta), radius * cos((-π / 2) + beta) * cos(alpha1))

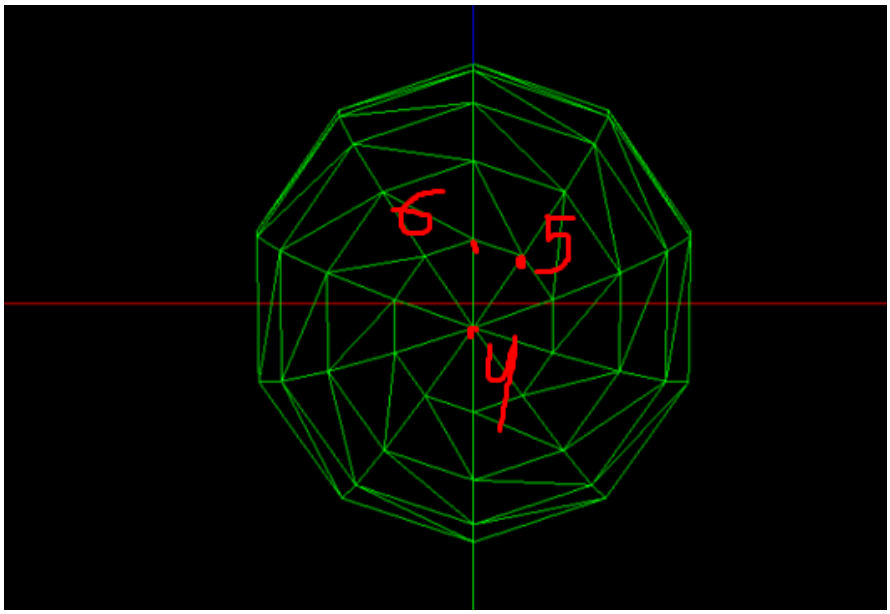

Figure 15- Esquema Sphere lV

Camadas intermedias:

V7 = (radius * cos(beta1) * sin(alpha1), radius * sin(beta1), radius * cos(beta1) * cos(alpha1))

V8 = (radius * cos(beta1 - beta) * sin(alpha1), radius * sin(beta1 - beta), radius * cos(beta1 - beta) * cos(alpha1))

V9 = (radius * cos(beta1) * sin(alpha1 + alpha), radius * sin(beta1), radius * cos(beta1) * cos(alpha1 + alpha))

V10 = (radius * cos(beta1 - beta) * sin(alpha1+ alpha), radius * sin(beta1 - beta), radius * cos(beta1 - beta) * cos(alpha1+ alpha))

Onde:

$0 \leq alpha1 < 2\pi - alpha$

$-\pi/2 + beta \leq beta1 < \pi/2 - beta$

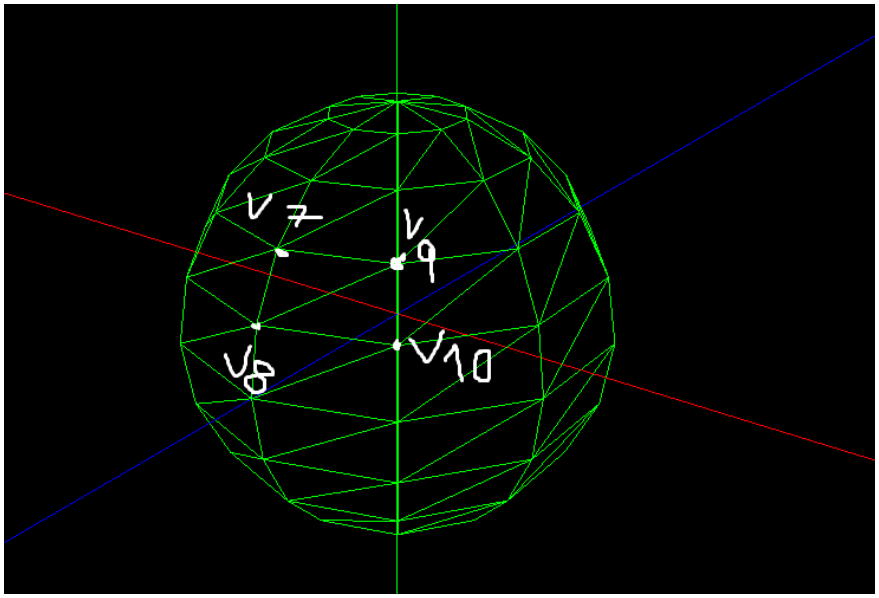

Figure 16- Esquema Sphere V

2. Engine:

XML

Ao iniciar o Engine o ficheiro chamado “ficheiro.xml” dessa mesma pasta será lido, esse ficheiro tem lá escrito os ficheiros .3d que serão lidos e processados pelo Engine.

Para ler o ficheiro 3d gerado anteriormente o ficheiro.xml teria de ter escrito o seguinte:

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

Outros ficheiros .3d podem ser adicionados ao mesmo xml e todos eles serão lidos e desenhados pelo Engine.

O objetivo principal do Engine é ler os ficheiros XML, nos quais contêm referência aos ficheiros 3d que vão ser lidos e desenhados.

Para nos auxiliar na leitura dos ficheiros XML usamos a livraria tinyxml2 que nos foi indicada no enunciado do trabalho, sendo esta uma livraria simples e eficaz para o *parsing* de ficheiros XML.

A partir dessa livraria criamos uma função “readXML” que vai ler o “ficheiro.xml” usando essa livraria, sendo esta função chamada quando o Engine é Iniciado, a função tem um ciclo onde vai a cada elemento <model>, lendo o atributo file, onde tem o caminho para o ficheiro .3d, passando-o para a função “readFile”.

A função “readFile”, vai ler o ficheiro .3d, lendo a primeira linha que são o número de vértices e fazendo um clico que vai iterar no número dos vértices, que são as linhas do ficheiro pois temos um vértice por linha, e a cada iteração guardamos o vértice (x,y,z) naquela linha em memória numa lista de “Pontos” que é uma simples classe com variáveis (x,y,z).

Finalmente a cada renderScene chamamos a função Draw que vai á lista de pontos, iterando sobre ela, desenhando 3 pontos de cada vez fazendo um triângulo.

Exemplos de output do Engine:

- **Plane com dimensão = 5**

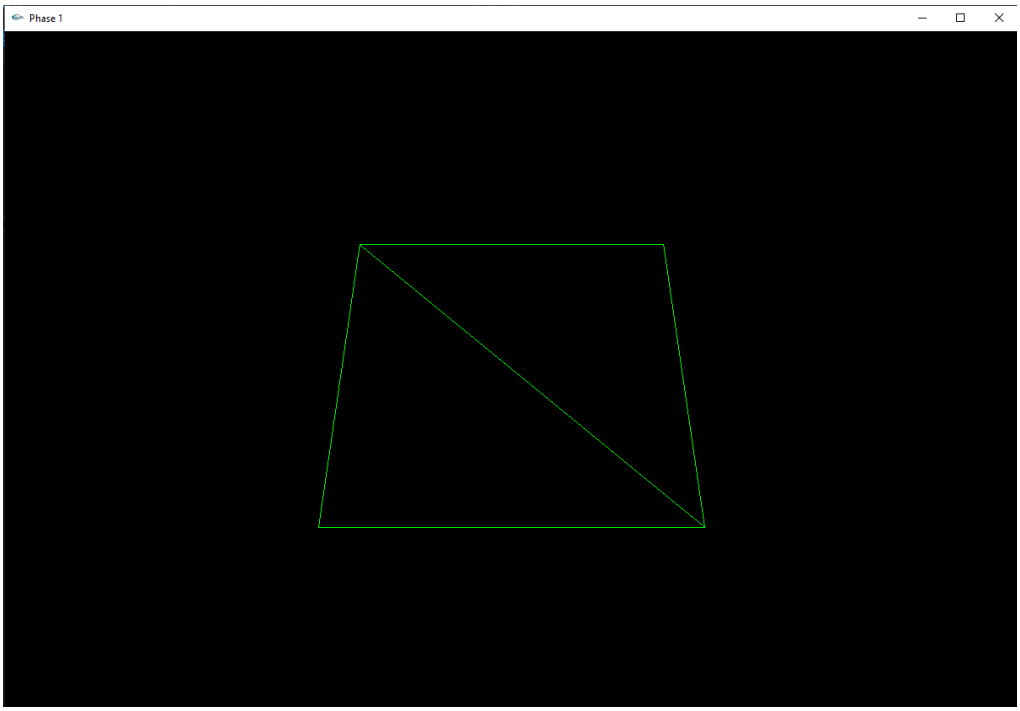

Figura 17- Output Engine I

- **Box com x = 2, y = 2, z = 2 e slices = 2**

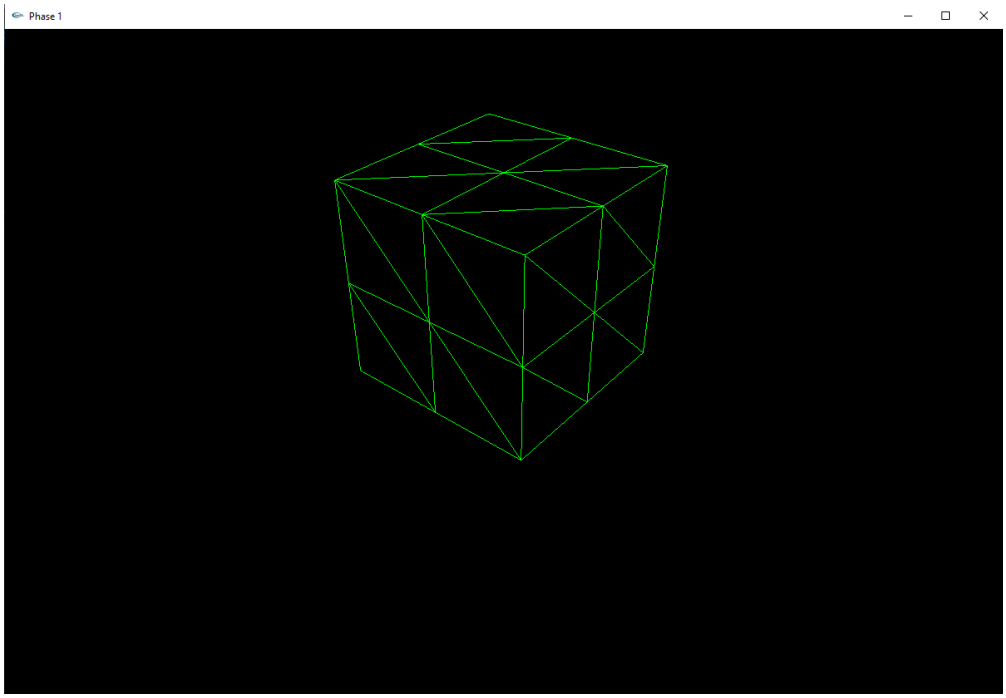

Figura 18- Output Engine II

- **Sphere raio = 5, slices = 100, stacks = 100**

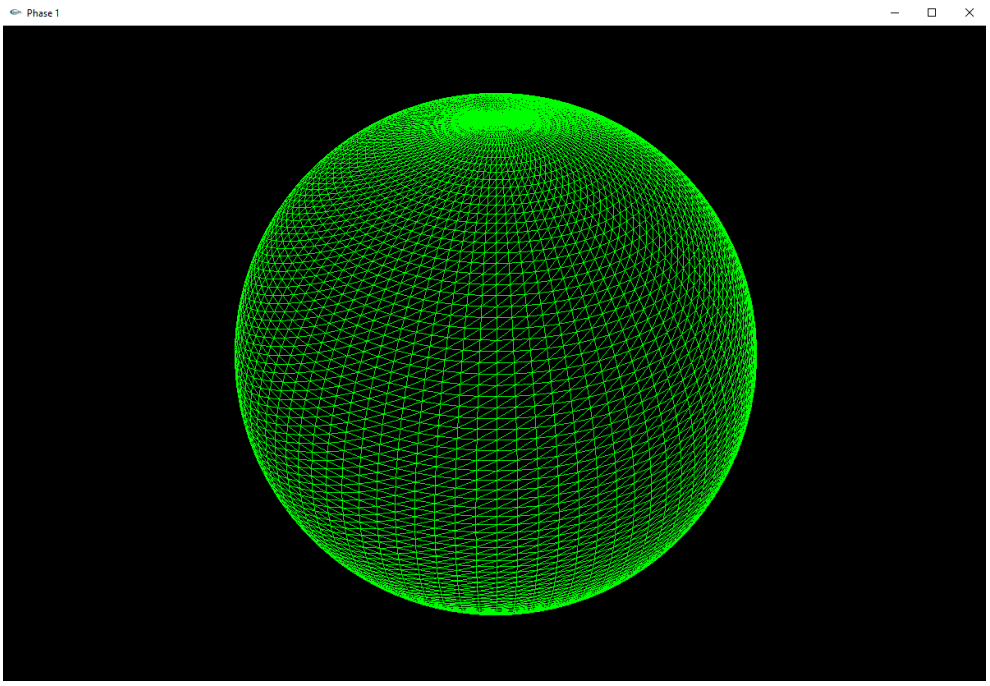

Figura 19- Output Engine III

- **Cone com raio = 4, altura = 4, slices = 20, stacks = 20**

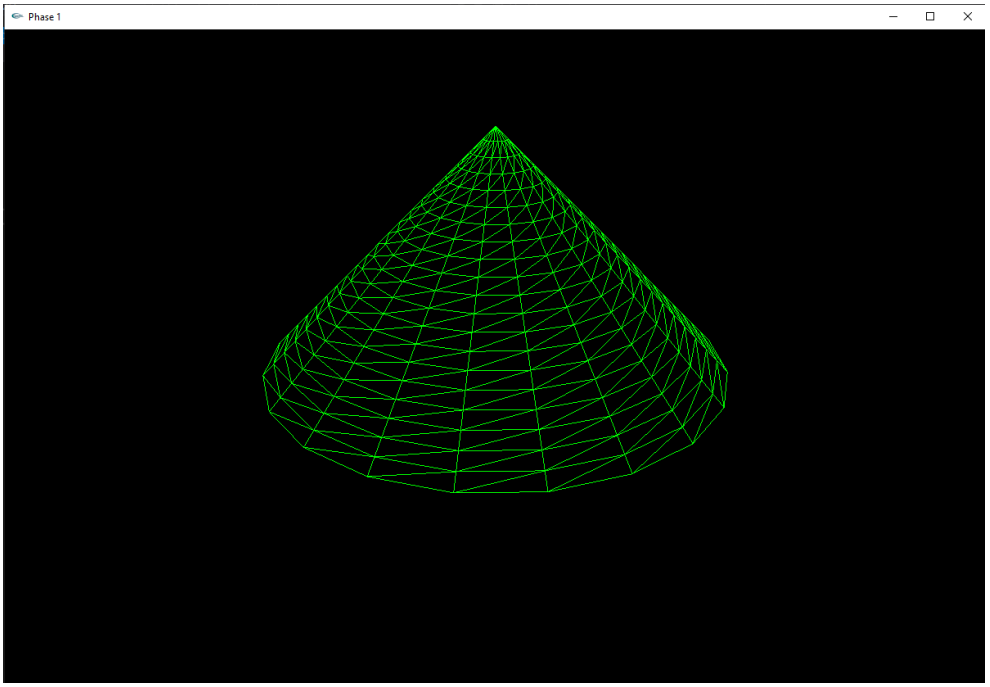

Figura 20- Output Engine IV

Extras:

Função keyboard :

Permite-nos interagir com as figuras a serem apresentadas na janela

Premindo a tecla:

- → **e :** Remove os eixos XYZ caso eles estejam desenhados, ou desenha-os caso contrário
- → **f :** Muda o polígono desenhado para modo GL_FILL
- → **l:** Muda o polígono desenhado para modo GL_LINE
- → **p :** Muda o polígono desenhado para modo GL_POINT
- → **r :** Muda a cor do polígono para vermelho
- → **g :** Muda a cor do polígono para verde
- → **b :** Muda a cor do polígono para azul

Após premir uma das teclas acima basta apenas utilizar **glutPostRedisplay(),** para o desenho na janela ser atualizado.

```
283  //funcao que d@ fun@@o @s teclas premidas
284  void keyboard(unsigned char key, int x, int y){
285      if (key == 'e') {
286          eixos = !eixos;    }
287
288      if (key == 'f') {
289          tipo = GL_FILL;    }
290
291      if (key == 'l') {
292          tipo = GL_LINE;    }
293
294      if (key == 'p') {
295          tipo = GL_POINT;   }
296
297      if (key == 'r') {
298          v = 1.0f;
299          g = 0.0f;
300          b = 0.0f;          }
301
302      if (key == 'g') {
303          v = 0.0f;
304          g = 1.0f;
305          b = 0.0f;          }
306
307      if (key == 'b') {
308          v = 0.0f;
309          g = 0.0f;
310          b = 1.0f;          }
311
312      glutPostRedisplay();
313  }
      f keyboard
```

Figura 21- Função keyboard

Câmara utilizada:

Para facilitar a visualização das figuras optamos por implementar a camara em modo explorador (Explorer Mode Camera). Onde a camara move-se na superfície da esfera a olhar para o seu centro ou seja a origem.

Para isso implementamos duas funções **processMouseButtons (int button, int state, int xx, int yy)** e **processMouseMotion (int xx, int yy)** que calculam a posição da camara.

Ao pressionar o botão esquerdo alteramos o alpha (movendo o rato para a esquerda ou para a direita onde o alpha aumenta e diminui respetivamente) e o beta (movendo o rato para baixo ou para cima onde o beta aumenta e diminui respetivamente) estando limitado o intervalo entre -85 e 85 graus.

Ao pressionar o botão direito alteramos o raio (movendo o rato para baixo ou para cima onde o raio diminui e aumenta respetivamente) sendo este maior ou igual a 3. Quanto menor o raio maior é a proximidade a origem.

O alpha, o beta e raio são utilizados para calcular as coordenadas cartesianas da posição através de coordenadas esféricas (Spherical Coordinates) que utilizam o alpha, o beta e raio para especificar um ponto na superfície de uma esfera.

Conclusão:

Esta primeira fase permitiu ao grupo, a consolidação de vários conhecimentos relativos ao OpenGL e ao GLUT, mas ainda, adquirir mais prática na linguagem de programação C++, algo que nos obrigou a uma pesquisa sobre alguns aspetos da mesma. Podemos ainda dizer que este trabalho, deu para colocarmos em prática aspetos relativos a uma UC de primeiro ano, denominada por Geometria.

Todos os requisitos propostos nesta fase foram atingidos, tendo todos sido implementados com sucesso. Obviamente, uns com mais "atrito" pelo caminho do que outros. Podemos dizer que a esfera, o cone e ainda certos aspetos no Engine, foram aqueles pontos em que mais dificuldades tivemos, os dois primeiros pela complexidade da implementação das coordenadas e pela divisão por camadas, o último pela forma de como iriamos abordar a escrita, leitura e manipulação do ficheiro XML, para a posterior tarefa, que era o desenho das figuras geométricas pretendidas.

Contudo, esperemos que o sucesso obtido nesta primeira fase do projeto continue nas próximas etapas do mesmo.

Andre Araújo
Henrique Ferreira
Daniel Ribeiro
Paulo Costa