



Universidade do Minho  
Departamento de Informática

## Computação Gráfica

Ano Letivo 2023/2024

# Trabalho Prático Fase 1 - Primitivas Gráficas

### **Grupo 13**

Ema Martins - A97678

Gonçalo Costa - A100824

Henrique Malheiro - A97455

Marta Rodrigues - A100743

## Índice

Introdução .....	3
Arquitetura .....	3
Generator .....	3
Plane .....	4
Box .....	5
Cone .....	7
Sphere .....	9
Estruturas Auxiliares .....	12
Engine .....	13
Conclusão .....	13

## Introdução

Este relatório foi elaborado no âmbito da primeira fase do trabalho prático da unidade curricular de Computação Gráfica, na qual nos foi proposto o desenvolvimento de duas aplicações: o Generator, um gerador de vértices para primitivas gráficas como planos, caixas, esferas e cones, e a Engine, uma aplicação capaz de ler ficheiros de configuração em XML para desenhar os vértices das primitivas gráficas geradas previamente.

Para a implementação de tal, recorreremos à utilização da linguagem de programação C++ e à ferramenta OpenGL, ambas lecionadas nas aulas práticas. As diferentes metodologias e abordagens implementadas serão detalhas mais à frente neste relatório.

## Arquitetura

O projeto foi estruturado da seguinte forma:

- **engine:** traduz os pontos gerados pelo **generator** para figuras 3d;
- **generator:** responsável pelo cálculo das coordenadas dos pontos para a representação do plano, caixa, esfera e cone, bem como a criação dos ficheiros .3d de forma a depois serem utilizados pelo **engine**;
- **pugiXML:** contém o package pugiXML, de forma a auxiliar o parser para leitura de ficheiros XML;
- **utils:** possui as estruturas auxiliares que o Engine e o Generator utilizam para o seu devido funcionamento.

## Generator

Contém os diversos ficheiros responsáveis por calcular os pontos dos triângulos que permitem construir as diversas primitivas para gerar o ficheiro .3d de uma figura específica.

A estrutura do ficheiro resultante possui assim:

- Na primeira linha com o número total de pontos no ficheiro gerado;
- Os sucessivos pontos que constroem a figura pedida. Cada ponto é constituído por 3 valores float, separados por vírgula, sendo esses X, Y e Z respectivamente.

```
54
-0.5,0,-0.5
-0.5,0,-0.166667
-0.166667,0,-0.166667
-0.166667,0,-0.166667
-0.166667,0,-0.5
-0.5,0,-0.5
-0.166667,0,-0.5
-0.166667,0,-0.166667
0.166667,0,-0.166667
0.166667,0,-0.166667
0.166667,0,-0.5
-0.166667,0,-0.5
```

Figure 1: Exemplo de um ficheiro .3d

Para construir os pontos para as diferentes primitivas, é necessário fornecer ao Generator os seguintes argumentos:

- O nome da figura que se deseja desenhar (plane, box, sphere ou cone);

- O sucessivos valores necessários para construir a primitiva pretendida (Plane - length, divisions; Box - length, divisions; Sphere - radius, slices, stacks; Cone - radius, height, slices, stacks);
- Ficheiro destino .3d onde os pontos serão armazenados, para posteriormente serão lidos pelo engine.

Explicaremos agora os diferentes ficheiros em mais detalhe.

## Plane

Os planos precisam estar contidos no plano XZ e o seu centro deve estar na origem (0,0,0).

O plano será visto como uma matriz de dimensão *divisões* × *divisões* em que cada elemento dessa matriz é um quadrado de lado (*comprimento/divisões*) dividido em dois triângulos cuja hipotenusa é a diagonal desse quadrado. Para a construção do plano:

1. São inicializados 4 pontos de seguintes coordenadas:
  - $p0 = (-\text{comprimento}/2, 0, -\text{comprimento}/2)$
  - $p1 = (-\text{comprimento}/2, 0, -\text{comprimento}/2 + \text{divisões})$
  - $p2 = (-\text{comprimento}/2 + \text{divisões}, 0, -\text{comprimento}/2 + \text{divisões})$
  - $P3 = (-\text{comprimento}/2 + \text{divisões}, 0, -\text{comprimento}/2)$
2. Após esses pontos serem guardados o plano continuará a ser construído linha a linha deslocando os pontos inicialmente mencionados ao longo do eixo do X, sendo cada deslocação do tamanho da divisão da comprimento pelo número de divisões (*comprimento/divisões*). Cada quadrado é composto por dois triângulos de vértices [p0, p1, p2] e [p0, p2, p3].
3. Quando a linha acaba, ou seja, atinge o valor de divisões pedidas, os valores dos pontos voltam aos originais e incrementa-se o valor do Z com o mesmo valor (resultado da divisão da comprimento pelo número de divisões - *comprimento/divisões*) e volta se a repetir o processo anterior.

Seguem-se duas figuras ilustrativas para auxiliar na explicação do raciocínio acima.

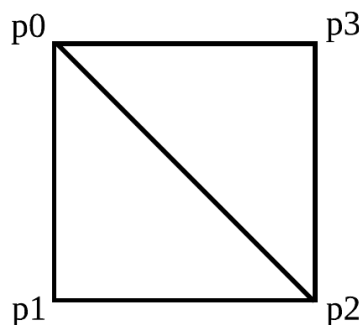


Figure 2: Processo de construção de um dos quadrados

Na figura acima, é apresentada a ordem pelo que os vértices gerados inicialmente estão dispostos, para poder formar os dois triângulos necessários. Para formar o plano completo, os quadrados que formam o plano são construídos na seguinte ordem:

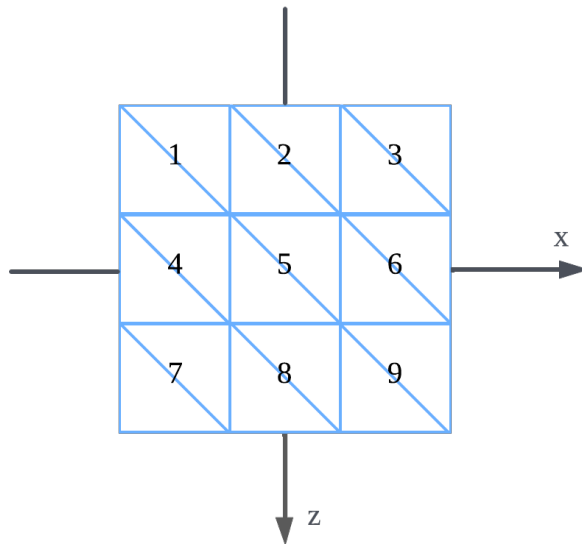


Figure 3: Processo de construção do plano

Com o comprimento = 1 e divisões = 3, a figura a seguir representa o plano desenhado através do ficheiro .3d gerado:

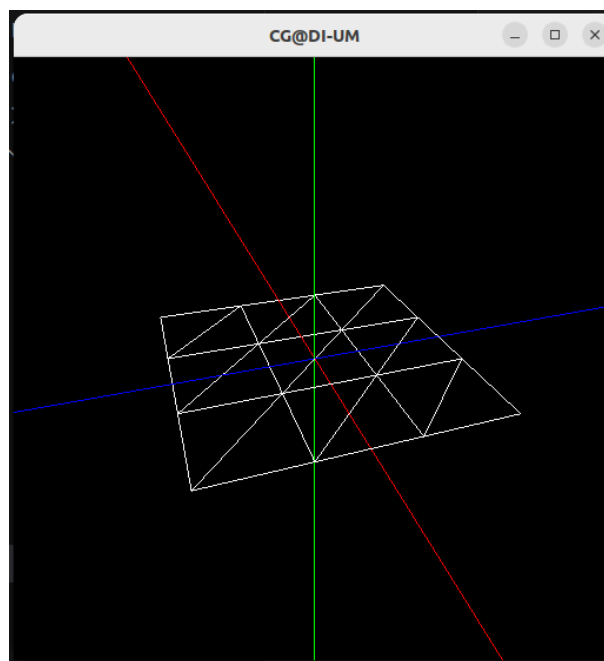


Figure 4: Exemplo de um plano com comprimento 1 e 3 divisões

## Box

Para obter os pontos que levariam à construção da caixa foi aplicado o mesmo raciocínio que o do plano, extendendo esse mesmo.

Deste modo, foram criados 6 planos diferente, isto é, foram gerados dois planos paralelos a cada plano que contém os eixos XZ, XY e YZ, sendo os valores de  $y = \pm \text{comprimento}/2$ ,  $z = \pm \text{comprimento}/2$  e  $x = \pm \text{comprimento}/2$ , respectivamente.

Os seguintes pontos, são aqueles que iniciam as funções que geram os planos paralelos aos planos especificados em cima:

### 1. Função geradora de um plano paralelo a XZ:

- $p0 = (-\text{comprimento}/2, \pm\text{comprimento}/2, -\text{comprimento}/2)$
- $p1 = (-\text{comprimento}/2, \pm\text{comprimento}/2, -\text{comprimento}/2 + \text{comprimento}/\text{divisões})$
- $p2 = (-\text{comprimento}/2 + \text{comprimento}/\text{divisões}, \pm\text{comprimento}/2, -\text{comprimento}/2)$
- $p3 = (-\text{comprimento}/2 + \text{comprimento}/\text{divisões}, \pm\text{comprimento}/2, -\text{comprimento}/2 + \text{comprimento}/\text{divisões})$

### 2. Função geradora de um plano paralelo a XY:

- $p0 = (-\text{comprimento}/2, -\text{comprimento}/2, \pm\text{comprimento}/2)$
- $p1 = (-\text{comprimento}/2, -\text{comprimento}/2 + \text{comprimento}/\text{divisões}, \pm\text{comprimento}/2)$
- $p2 = (-\text{comprimento}/2 + \text{comprimento}/\text{divisões}, -\text{comprimento}/2, \pm\text{comprimento}/2)$
- $p3 = (-\text{comprimento}/2 + \text{comprimento}/\text{divisões}, -\text{comprimento}/2 + \text{comprimento}/\text{divisões}, \pm\text{comprimento}/2)$

### 3. Função geradora de um plano paralelo a YZ:

- $p0 = (\pm\text{comprimento}/2, -\text{comprimento}/2, -\text{comprimento}/2)$
- $p1 = (\pm\text{comprimento}/2, -\text{comprimento}/2, -\text{comprimento}/2 + \text{comprimento}/\text{divisões})$
- $p2 = (\pm\text{comprimento}/2, -\text{comprimento}/2 + \text{comprimento}/\text{divisões}, -\text{comprimento}/2)$
- $p3 = (\pm\text{comprimento}/2, -\text{comprimento}/2 + \text{comprimento}/\text{divisões}, -\text{comprimento}/2 + \text{comprimento}/\text{divisões})$

O cálculo dos pontos a partir destes inicializados é feito como foi especificado anteriormente no plano.

Porém, ao contrário do plano as funções que geram os planos paralelos a XZ, XY e YZ têm uma flag “reverse” que permite alterar a ordem em que os pontos são inseridos na figura de modo a inverter a orientação do polígono, isto é, trocando a ordem entre os vértices da diagonal dos triângulos. Por exemplo, na função que gera o plano paralelo a XZ, se a flag tiver o valor 0, o plano poderá ser visto de cima, com  $y > 0$ . Caso a flag tome valor 1, o plano poderá ser visto de baixo, com  $y < 0$ .

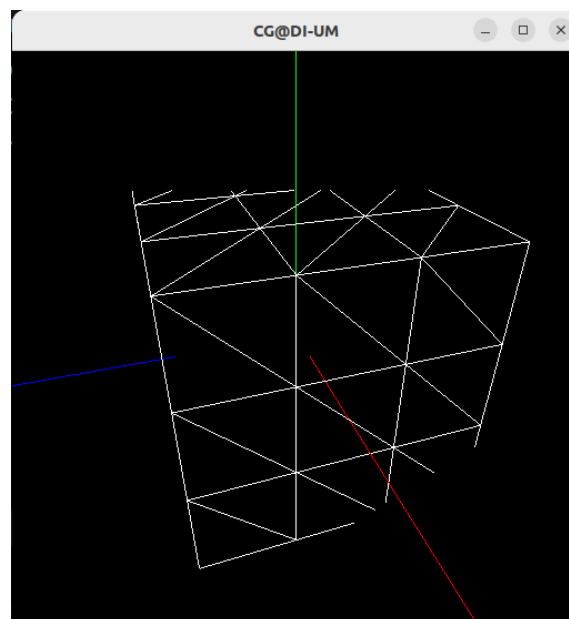


Figure 5: Exemplo de uma box com comprimento 2 e 3 divisões

## Cone

Para criação do cone tivemos em consideração 4 parâmetros muito importantes, sendo esses a altura do cone, o raio da base, o número de stacks e de slices.

Começamos por calcular a base do cone, dividindo  $2\pi$  por  $n^{\circ}\text{slices}$  obtendo assim o ângulo ( $\alpha$ ) entre os vértices e o raio da circunferência, permitindo assim, com o auxílio de coordenadas polares, calcular as coordenadas de x e z de cada ponto partir das seguintes fórmulas:

- $x = \text{raio} \times \cos(\alpha)$
- $z = \text{raio} \times \sin(\alpha)$

Sendo  $y = 0$  um valor constante pois a base do cone será definida no plano xz.

Deslocando o ponto da origem através do ângulo obtido  $(0, 0, \text{raio})$ , foi nos possível calcular os diversos pontos da base.

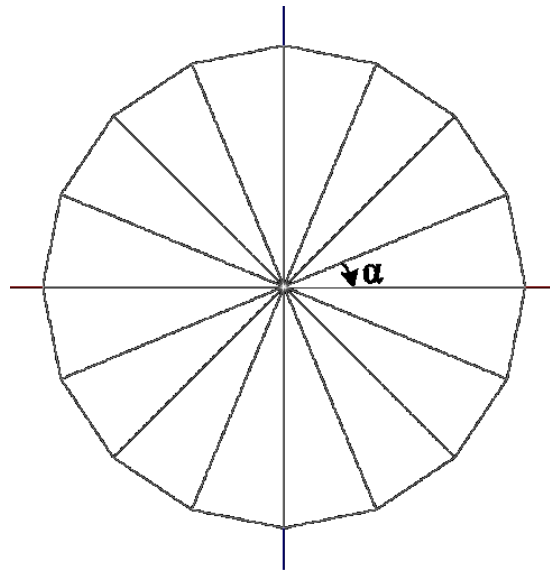


Figure 6: Ilustração da base de um cone e o seu  $\alpha$

Após calculados os pontos necessários para a base do cone, falta o cálculo das faces laterais do cone.

Para a criação das laterais recorreremos à criação de triângulos sucessivos até o número dos mesmos atingir o número de *slices*.

Como o cone é dividido em diversas *stacks* que são representadas por cortes horizontais, foi preciso calcular o tamanho de cada *stack*, esse valor foi obtido calculando o resultado da divisão da *altura do cone* pelo número de *stacks*.

Para além disso, à medida que vamos avançando na construção do cone e “subindo” no cone, o raio irá também variar, vai ser necessário calcular o raio da stack seguinte para encontrar as coordenadas, através da seguinte expressão:

- Próximo raio =  $\text{raio} * \left(1 - \left(\frac{n}{\text{stacks}}\right)\right)$ , sendo *raio* o raio inicial fornecido, *n* a *stack* atual e *stacks* o número de *stacks total*

Conseguimos chegar a esta fórmula analisando a vista de cima de um cone, como representado na figura a seguir. Através desta podemos visualizar que as *stacks* partem o raio de forma igual, por exemplo com 3 *stacks*, o  $\Delta\text{raio}$  representa  $\frac{1}{3}$  do raio fornecido, sendo assim o raio da primeira divisão  $\frac{2}{3}$  do raio fornecido.

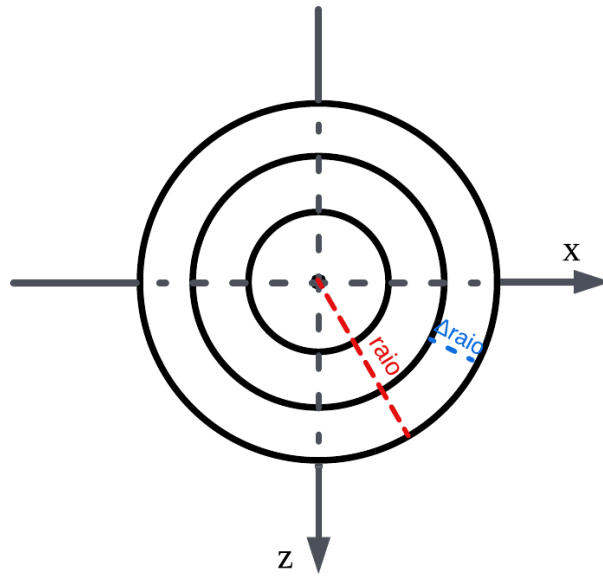


Figure 7: Ilustração auxiliar da explicação (vista cima)

Com todas essas informações, acrescentando que a base do cone estará contida no plano XZ, podemos formar as 4 equações para o cálculo sucessivo dos triângulos que constituirão cada stack:

- $p0 = (\text{raio atual} \times \sin(\text{slice atual} \times \text{alfa}), \text{altura da stack} \times \text{stack atual}, \text{raio atual} \times \cos(\text{slice atual} \times \text{alfa}))$
- $p1 = (\text{raio atual} \times \sin((\text{slice atual} + 1) \times \text{alfa}), \text{altura da stack} \times \text{stack atual}, \text{raio atual} \times \cos((\text{slice atual} + 1) \times \text{alfa}))$
- $p2 = (\text{proximo raio} \times \sin(\text{slice atual} \times \text{alfa}), \text{altura da stack} \times \text{próxima stack}, \text{proximo raio} \times \cos(\text{slice atual} \times \text{alfa}))$
- $p3 = (\text{proximo raio} \times \sin((\text{slice atual} + 1) \times \text{alfa}), \text{altura da stack} \times \text{próxima stack}, \text{proximo raio} \times \cos((\text{slice atual} + 1) \times \text{alfa}))$

A construção do cone é feita slice a slice, progredindo dentro de cada slice pelas suas stacks, até à penúltima camada do cone. Finalmente, para o topo do cone, há exceção do resto do anterior, são reaproveitados os últimos pontos calculados e desenha-se um triângulo final por *stack*, e tem um dos vértices com  $y = (0, \text{altura do cone}, 0)$ . Obtendo assim exemplos de diferentes cones desenhados a partir dos ficheiros .3d gerados:



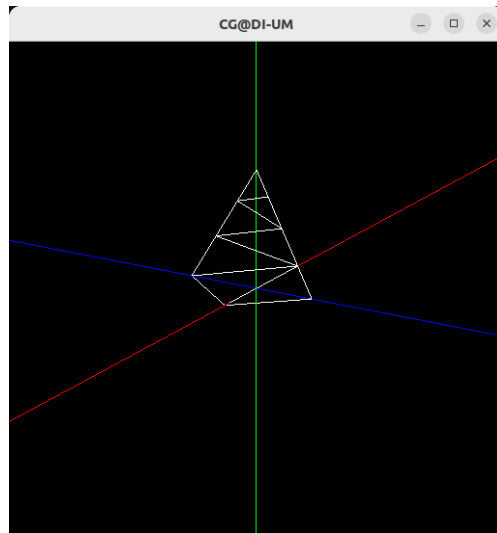


Figure 8: Exemplo de um cone com raio 1, 2 de altura, 4 *slices* e 3 *stacks*

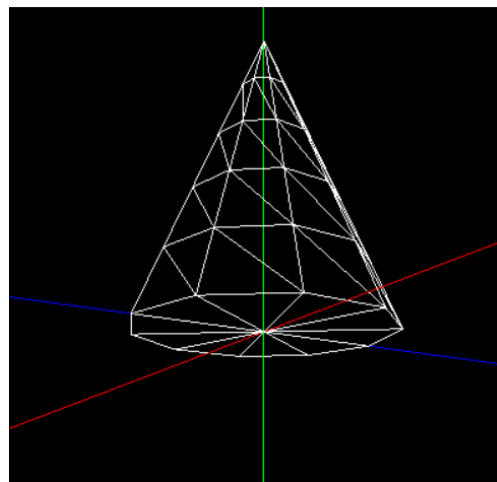


Figure 9: Exemplo de um cone com raio 2, 5 de altura, 10 *slices* e 5 *stacks*

## Sphere

Para calcular os pontos das esferas, recorreremos às coordenadas esféricas  $(\alpha, \beta, \text{raio})$  que serão transformadas em coordenadas cartesianas através das fórmulas seguintes:

- $x = \text{raio} \times \cos(\beta) \times \sin(\alpha)$
- $y = \text{raio} \times \sin(\beta)$
- $z = \text{raio} \times \cos(\beta) \times \cos(\alpha)$

Assim, para calcular os pontos de uma esfera precisamos do raio, do número de slices (camadas verticais) e do número de stacks (camadas horizontais). O ângulo entre as diferentes slices  $\alpha$  será o resultado de  $2\pi / \text{slices}$ , e, então, representa o valor do ângulo pelo qual iremos deslocar o ponto base de origem  $(0, 0, \text{raio})$ .

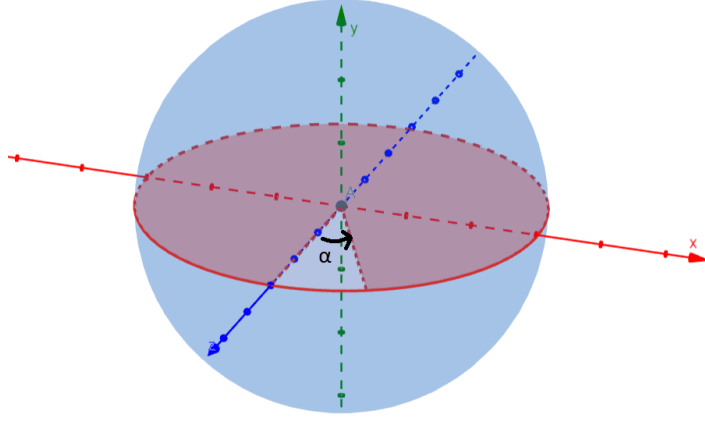


Figure 10: Exemplo de um valor  $\alpha$  numa esfera

Para determinar o deslocamento necessário entre as camadas das stacks, calculamos o ângulo  $\beta$  como  $\pi / \text{stacks}$ , sendo que este valor, durante a construção da esfera, irá variar entre  $[-\frac{\pi}{2}, \frac{\pi}{2}]$  de modo que a coordenada y possa ser negativa e positiva.

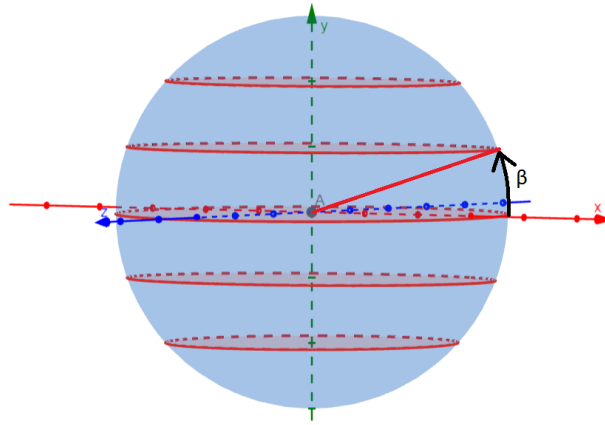


Figure 11: Exemplo de um valor  $\beta$  numa esfera com 6 stacks

Com todas as informações necessárias decifradas, podemos então inferir as primeiras coordenadas para a construção da esfera:

- $p1 = \text{raio} \times \cos(\text{beta atual}) \times \sin(\text{alfa atual}), \text{raio} \times \sin(\text{beta atual}), \text{raio} \times \cos(\text{beta atual}) \times \cos(\text{alfa atual})$
- $p2 = \text{raio} \times \cos(\text{próximo beta}) \times \sin(\text{alfa atual}), \text{raio} \times \sin(\text{próximo beta}), \text{raio} \times \cos(\text{próximo beta}) \times \cos(\text{alfa atual})$
- $p3 = \text{raio} \times \cos(\text{próximo beta}) \times \sin(\text{próximo alfa}), \text{raio} \times \sin(\text{próximo beta}), \text{raio} \times \cos(\text{próximo beta}) \times \cos(\text{próximo alfa})$
- $p4 = \text{raio} \times \cos(\text{beta atual}) \times \sin(\text{próximo alfa}), \text{raio} \times \sin(\text{beta atual}), \text{raio} \times \cos(\text{beta atual}) \times \cos(\text{próximo alfa})$

Os triângulos da esfera serão desenhados stack a stack, “descendo” pela esfera, decrementando o valor de  $\beta$ , e enquanto nos encontramos na stack a ser gerada, todas as slices são percorridas (através do incremento de  $\alpha$  pelo valor do ângulo determinado inicialmente), ou seja, só se avança para a próxima stack quando todos os triângulos das slices da stack atual são calculadas.

Entre duas stacks e duas slices vai então existir um “quadrado” constituído por dois triângulos de estrutura semelhante à Figure 2. No entanto, visto que as slices da stack superior e a inferior da esfera são apenas formados por um único triângulo com orientações diferentes, quando a stack atual é a primeira a ser desenhada, ou seja a do topo da esfera, é apenas construído o triângulo da esquerda, por outro lado caso seja a última, a que se encontra no fundo da esfera, só se desenha o triângulo da direita. Assim, na imagem a seguir, que procura representar melhor este processo, o triângulo vermelho é gerado em todas as slices das stacks que não sejam a superior e o triângulo azul é gerado em qualquer slices das stacks que não sejam a inferior da esfera.

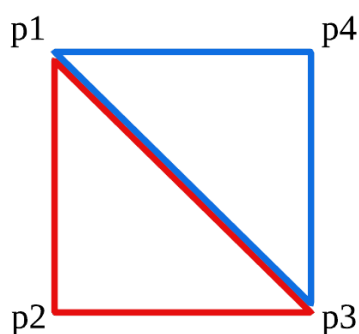


Figure 12: Processo seleção dos triângulos a desenharm nas stacks

Na figura seguinte, apresentamos assim um exemplo de uma esfera desenhada através do ficheiro .3d gerado com o ficheiro teste .xml :

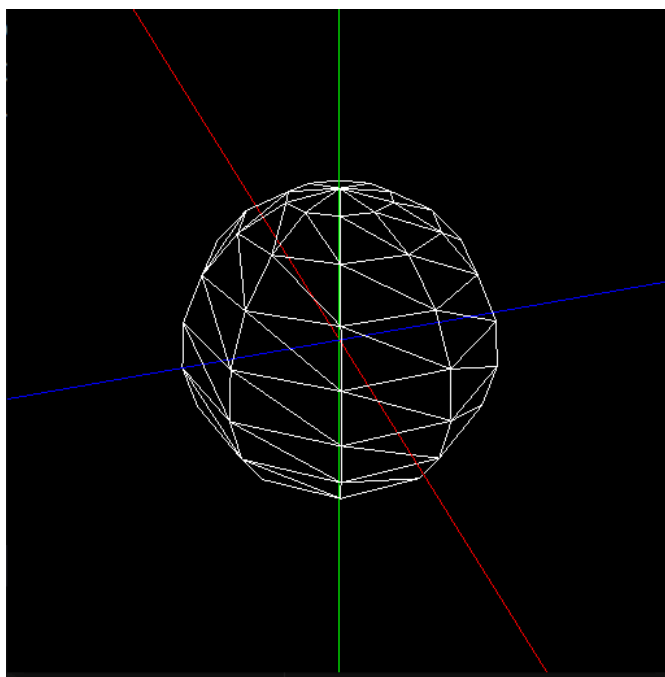


Figure 13: Exemplo de uma esfera raio 1, 10 slices e 10 stacks

Adicionalmente, a próxima imagem representa o resultado de outro teste fornecido, composta por uma esfera e um plano, sendo tal possível pela lógica do engine exposto mais à frente neste relatório:

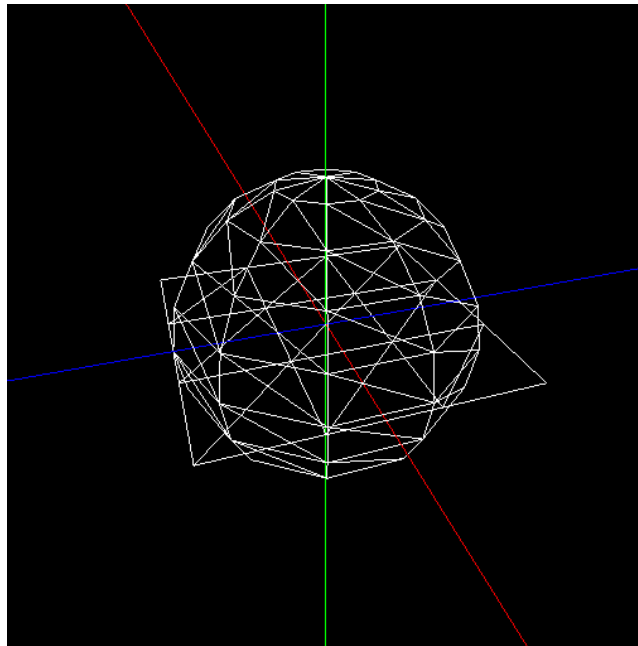


Figure 14: Exemplo de uma esfera raio 1, 10 slices e 10 stacks e um plano de comprimento 2 e 3 divisões

## Estruturas Auxiliares

Vale realçar que foram criadas as estruturas auxiliares **figura** e **ponto**.

Uma figura representa o total de pontos de uma figura e é constituída por um *vector* de pontos. Os pontos por si são compostos por três valores float que correspondem aos valores de x, y e z. Também foram criadas funções auxiliares para as estruturas, para possibilitar a manipulação das mesmas como inserção de pontos de figuras e a inicialização, e, no caso das figuras, poder fazer a escrita para o ficheiro pretendido.

```
struct ponto {
    float x, y, z;
};

Ponto newPonto(float x, float y, float z) {
    Ponto n = (Ponto)malloc(sizeof(struct ponto));
    if (n != NULL) {
        n->x = x; n->y = y; n->z = z;
    }
    return n;
}
```

Figure 15: Estrutura de um ponto

```
typedef struct figura {
    std::vector<Ponto> pontos;

    // Construtor padrão
    figura() {}
}Figura;

void addPonto(Figura& f, Ponto p);
void writeToFile(Figura f, const char* file_path);
```

Figure 16: Estrutura de uma Figura

## Engine

O engine, utilizando a biblioteca *pugixml*, lê e faz o parse do ficheiro de configuração (em *XML*), guardando a sua informação em formato de árvore, associando cada nível de indentação a um subnível dessa árvore.

Percorrendo os diversos filhos da árvore, configura a página, desde dimensões da janela (altura e largura), posição da câmara com vetores (posição, lookAt e up) bem como os parâmetros da matriz projeção (field of view, near e far).

Por fim, percorre-se os diversos *models* indicados no ficheiro, carregando os seus vértices, indicados nos ficheiros .3d gerados no generator, num vector por figura. Posteriormente, junta-se os diversos vetores num buffer comum, para que o render scene desenhe as figuras, segundo a metodologia de VBOs. Essa metodologia permite uma melhor performance, uma vez que os pontos são carregados unicamente na *main*, sendo apenas deslocados quando a posição da câmara ou da figura se alterem.

## Conclusão

Nesta primeira fase do trabalho fomos capazes de consolidar os diversos conhecimentos adquiridos nas primeiras aulas teóricas e práticas desta unidade curricular.

Apesar de diversas dificuldades enfrentadas acreditamos que fomos capazes de atingir todos os objetivos propostos para esta primeira fase, ou seja, a implementação do Generator e do Engine. Além disso, introduziu-se como extras a capacidade de movimentação da câmara e a capacidade de mudar o modo de visualização da figura.

Graças à consolidação destas bases obtidas ao longo do desenvolvimento desta fase, o grupo sente-se preparado para continuar o desenvolvimento para as futuras etapas.