

Universidade do Minho
Departamento de Informática

Computação Gráfica

Grupo 32

1ªFase



Afonso Magalhães

A95250



Artur Leite

A97027



Diana Teixeira

A97516

1. Introdução e Objetivos

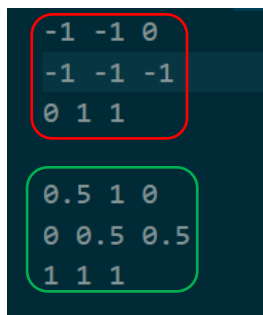
O nosso programa pode ser dividido em duas partes igualmente importantes, o *generator* e o *engine*. O *Generator* é responsável por gerar os pontos que representam as primitivas incluídas. Por sua vez, o *Engine* irá interpretar este conjunto de pontos e, posteriormente, gerar a representação 3D correspondente. As primitivas que desenvolvemos nesta fase são:

- Plano;
- Caixa;
- Esfera;
- Cone;

É também importante referir que nem todas as figuras incluídas no relatório seguem o sistema de coordenadas de mão direita, sendo figuras obtidas pela web.

2. Generator

Todas as primitivas desenvolvidas são desenhadas a partir de triângulos, ou seja, múltiplos conjuntos de três pontos. Desta forma, escreve num ficheiro de output os conjuntos de pontos que irão ser posteriormente lidos pelo *Engine*. Nestes ficheiros .3d, uma linha corresponde a um ponto, na medida em que três linhas seguidas correspondem a um triângulo.



```
-1 -1 0
-1 -1 -1
0 1 1

0.5 1 0
0 0.5 0.5
1 1 1
```

Triângulo formado pelos pontos: (-1,-1,0); (-1,-1,-1); (0,1,1)

Triângulo formado pelos pontos: (0.5,1,0); (0,0.5,0.5); (1,1,1)

Figura 1- Formato para ficheiros .3d

2.1. Plano

Para gerarmos um plano, necessitaríamos de definir um comprimento de lado, juntamente com um número de divisões, sendo estas efetuadas no eixo do x e do z. O método que optamos por utilizar para conseguirmos desenhar este plano envolve dois ciclos for. Em cada iteração, serão estabelecidos os pontos para dois triângulos retângulos que juntos perfazem uma divisão do plano. Ao ser incrementado um certo valor, a seguinte iteração irá desenhar outros dois triângulos que perfazem uma nova divisão. Isto irá repetir-se até todas as divisões no eixo do x ficarem estabelecidas, em seguida, outro ciclo irá iterar e o processo irá repetir-se, estabelecendo uma nova linha de divisões, todas com um valor z diferente das divisões estabelecidas anteriormente. Adicionalmente, uma translação poderá ser necessária, para que o plano se encontre centrado na origem.

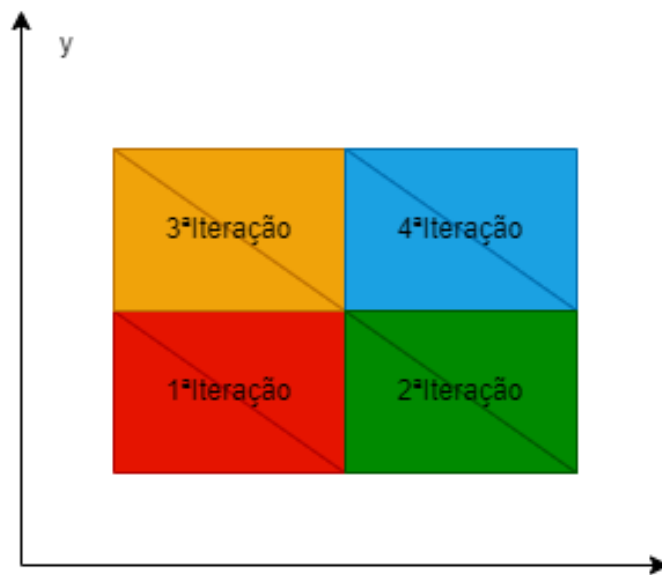


Figura 2- Exemplo de desenvolvimento das iterações

A seguinte imagem representa a implementação que utilizamos, num formato 3D.

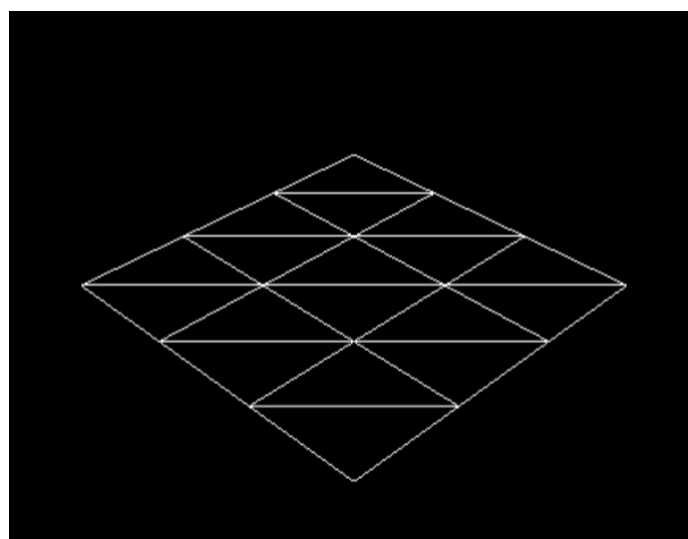


Figura 3- Plano gerado. 3 divisões, 1 de comprimento de lado

2.2. Caixa

De imediato, estabelecemos uma ligação entre as faces da caixa e os planos definidos anteriormente. Para o estabelecimento desta primitiva, optamos pela abordagem de definir 6 planos idênticos e consequentemente aplicar translações e rotações pertinentes no contexto da face que queremos representar.

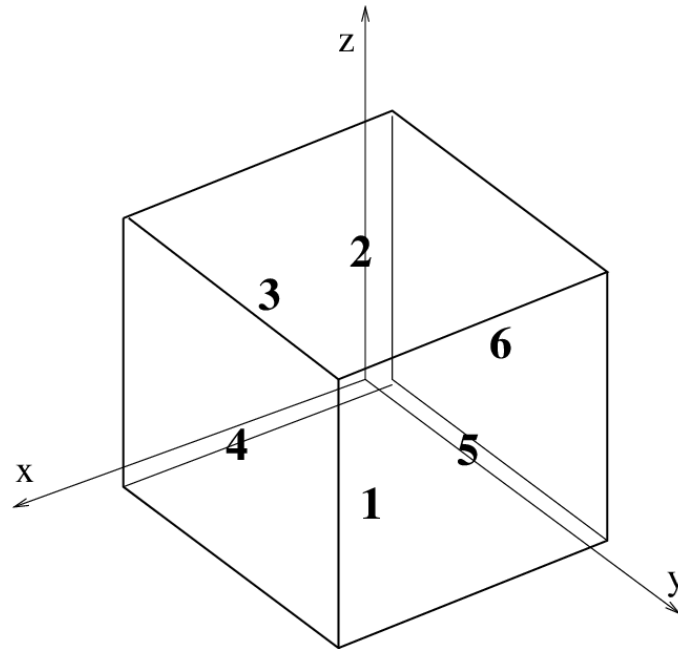


Figura 4- Exemplo de um cubo num plano 3D

Tomando a face 1 como o plano inserido inicialmente, ou seja, o plano base, podemos aferir o seguinte:

- A face 2 é obtida através do uso de uma translação no eixo z.
- As faces 3 a 6, podem ser obtidas através do uso de uma rotação seguida de uma translação aplicadas à posição inicial da face 1.
- Adicionalmente, uma translação poderá ser aplicada a todas as faces da caixa, incluindo a base, de forma a colocar a estrutura centrada na origem.

Tendo em conta o exemplo acima, com o intuito de realizar as transformações descritas, utilizamos a função `computeOperation`, aplicando-a ao plano base definido anteriormente o número de vezes necessário para completar a caixa. Eis o resultado obtido:

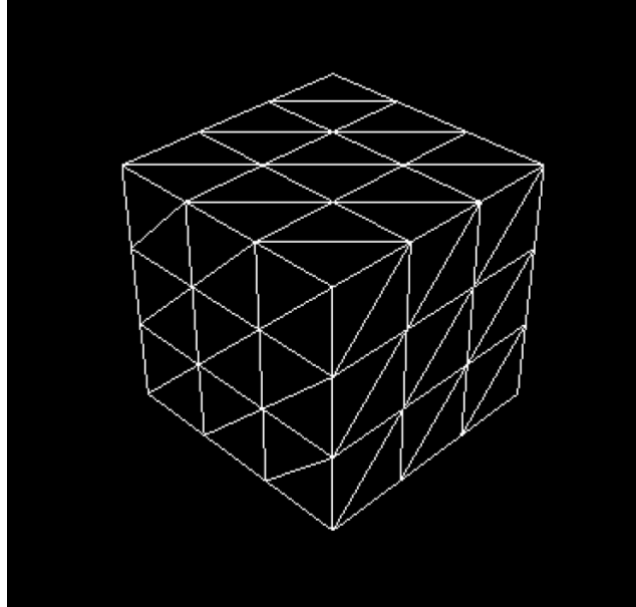


Figura 5- Caixa gerada. Faces equivalentes ao plano da figura 3

2.3. Cone

Para obtermos a representação de um cone, podemos considerar este como uma pilha de círculos cujo o raio vai diminuindo progressivamente ao longo da sua altura, convergindo finalmente num ponto único que representa o topo do cone em questão.

Para tal, precisamos de obter a sua altura, o número de *stacks* que o vão compor, o número de fatias que dividem cada círculo e o raio da sua base. Através destes valores, somos capazes de deduzir algumas equações uteis para a realização desta figura.

$$\text{altura de uma stack} = \frac{\text{altura do cone}}{\text{número de stacks}}$$

$$\text{ângulo } \propto \text{ de uma fatia} = \frac{2\pi}{\text{número de fatias de cada círculo}}$$

Na nossa implementação, optamos por realizar o processo ao contrário, ou seja, partimos do topo do cone, fazendo com que as *stacks* ficassem progressivamente maiores, ao invés de diminuírem como tinha descrito inicialmente. Eis o cálculo que utilizamos para saber o raio da *stack* atual.

$$\text{raio da stack atual} = \text{stack atual} * \frac{\text{raio da base do cone}}{\text{número de stacks}}$$

Finalmente, como podemos construir cada círculo considerando que estes se encontram num plano 2D com uma altura fixa, torna-se bastante útil para nós saber a altura onde cada *stack* irá ser introduzida, deduzindo assim o seu parâmetro y das suas coordenadas:

$$altura\ da\ stack = altura\ de\ uma\ stack * stack\ atual$$

Tendo a coordenada y definida com sucesso, podemos então utilizar coordenadas polares com o intuito de obter as coordenadas x e z :

$$x = raio\ da\ stack * \sin(\alpha * n^{\circ} da\ fatia)$$

$$z = raio\ da\ stack * \cos(\alpha * n^{\circ} da\ fatia)$$

Com recurso a estes cálculos, podemos então aplicar um ciclo `for`, onde iremos desenharmos o conjunto dos triângulos que irão perfazer cada círculo, tendo sempre em conta que, na próxima iteração, os valores deveram alterar em função da *stack* em que nos encontramos, fazendo com que, desta forma, os círculos fiquem progressivamente maiores. De notar a necessidade do cone se ter de encontrar no plano xz . Eis o resultado desta implementação em questão:

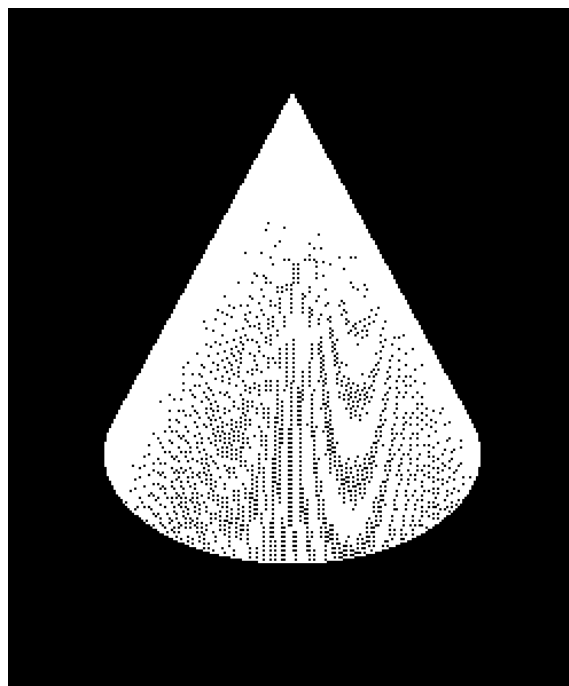


Figura 6- Cone gerado

2.4. Esfera

Para realizar a construção da esfera, iremos precisar de receber o seu raio, número de fatias e número de *stacks*.

Para além disso, começamos por identificar a necessidade de utilizar as coordenadas esféricas, sendo importante obter um α e um β para realizar os cálculos necessários. Eis como deduzimos os seus valores através dos inputs recebidos:

$$\alpha = \frac{2\pi}{\text{número de slices}}$$

$$\beta = \frac{2\pi}{\text{número de stacks}}$$

Adicionalmente, para cada *stack*, as coordenadas que utilizamos são obtidas através dos seguintes cálculos:

$$y = \cos(\beta * n^{\circ} \text{ da stack})$$

$$x = \text{raio da esfera} * \sin(\alpha * n^{\circ} \text{ da fatia}) * \sin(\beta * n^{\circ} \text{ da stack})$$

$$z = \text{raio da esfera} * \cos(\alpha * n^{\circ} \text{ da fatia}) * \sin(\beta * n^{\circ} \text{ da stack})$$

Inicialmente, encontramos algumas dificuldades em descobrir quais os parâmetros que seriam necessários para cada coordenada, nomeadamente se utilizaríamos o seno ou o cosseno. Para tal, analisamos a seguinte imagem como guia:

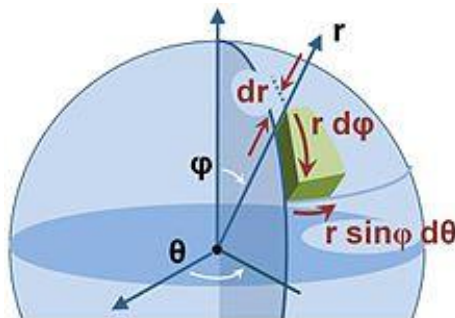


Figura 7- Guia de coordenadas esféricas

Tendo tudo isto corretamente definido, apenas realizamos dois ciclos *for*, que iteram para construir cada *stack* com o número correto de fatias. Teremos ainda de garantir que a esfera tem de se encontrar na origem. Eis o resultado:

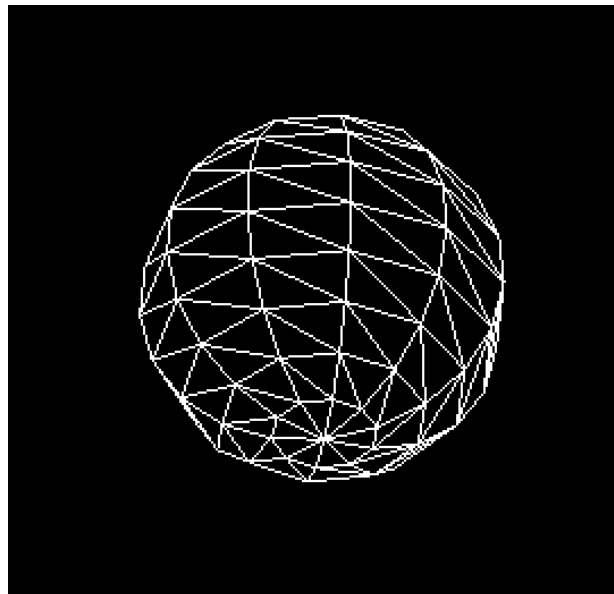


Figura 8- Esfera gerada

3. Engine

Após obtermos os ficheiros feitos pelo nosso generator, necessitamos de implementar um Engine capaz de interpretar e desenhar as primitivas que pretendemos, de forma a ser possível visualizar estes sólidos de forma eficiente.

3.1. Leitura XML

Com o intuito de ler os ficheiros XML, optamos por utilizar a biblioteca tinypng. Isto deve-se principalmente à sua simplicidade e ao conjunto de funcionalidades que oferece que vão de acordo com aquilo que necessitamos, garantindo ainda que não é muito pesada.

Tendo em conta o formato .3d mencionado anteriormente, o engine vai ler os outputs em grupos de 3 linhas, utilizando as funções do OpenGL para desenhar os triângulos correspondentes.

É também através da leitura deste ficheiro XML onde será feita a configuração de todo o ambiente tais como a width e height da janela bem como todos os valores necessários para a disposição da Câmera.

4. Conclusão

Com a finalização desta primeira fase, acreditamos que fomos capazes de nos familiarizar com programação em C++ e OpenGL e todas as ferramentas disponibilizadas por estes. Acreditamos ainda que fomos capazes de definir uma boa base para uma futura complexidade acrescida no projeto.

Quanto a aspetos a melhorar, queremos adicionar um movimento de câmara mais dinâmico, tornando o processo de visualização das figuras mais apelativo e informativo.