



UNIVERSIDADE DO MINHO

LICENCIATURA EM ENGENHARIA INFORMÁTICA

Computação Gráfica
Fase 1 - Graphical Primitives
Grupo 21

Duarte Parente (A95844) Gonçalo Pereira (A96849)
José Moreira (A95522) Santiago Domingues (A96886)

Ano Letivo 2022/2023

Índice

1	Introdução	3
2	Generator	4
2.1	Plano	4
2.2	Caixa	5
2.3	Cone	5
2.4	Esfera	6
3	Engine	9
3.1	Ficheiro de Configurações	9
4	Análise de Resultados	11
5	Conclusão	14

Capítulo 1

Introdução

O presente relatório visa apresentar a primeira fase do projeto a desenvolver no âmbito da Unidade Curricular de Computação Gráfica. Ao longo do documento irá ser explicada a metodologia adotada para a resolução dos problemas, assim como as diversas opções tomadas ao longo desta fase inicial.

O objetivo desta etapa passou pelo desenvolvimento de duas aplicações distintas, o **Generator** e o **Engine**. A primeira é responsável por gerar o conjunto de pontos que desenhem as primitivas geométricas propostas, designando-se estes por modelos. Em relação à segunda aplicação o seu principal foco passa por interpretar e apresentar as primitivas geradas previamente.

De forma a corresponder com os requisitos propostos no enunciado foram desenvolvidas as seguintes primitivas:

- **Plano:** um quadrado situado no plano XZ e centrado na origem;
- **Caixa:** centrada na origem, e que requer parâmetros como a sua dimensão e o número de divisões;
- **Esfera:** centrada na origem, e que requer parâmetros como o raio e o número de *slices* e *stacks*;
- **Cone:** centrado na origem, e que requer parâmetros como o raio da base, a altura e o número de *slices* e *stacks*;

Para facilitar a compilação do código e estruturação do projeto foram criadas duas **Makefiles**, uma para cada aplicação desenvolvida.

Capítulo 2

Generator

O módulo **Generator** é o responsável por gerar os ficheiros com as informações dos **modelos**, sendo que nesta primeira fase, o único elemento gerado para os modelos são os **vértices**. Para a criação destes ficheiros, o **Generator** irá receber como parâmetros o tipo de primitiva que se pretende criar, juntamente com as informações necessárias para a criação das mesmas, recebendo também, por último, o nome do ficheiro no qual serão armazenados os vértices. Vértices esse que serão posteriormente utilizados pelo **Engine**.

Nesta fase, é necessária a criação de pelo menos quatro tipos de primitivas gráficas: o **plano**, a **caixa**, o *cone* e a *esfera*, sendo importante referir que todas estas figuras são construídas com triângulos.

2.1 Plano

O plano representa um quadrado no plano XZ, centrado na origem, e subdividido em ambas as direções (X e Z). Para a criação do plano, o generator recebe como argumentos o comprimento do mesmo, e o número de divisões ao longo de cada eixo.

O método adotado pelo grupo para a determinação dos vértices foi o seguinte: começando no vértice de um dos "cantos" do plano, e calculando, através dos argumentos, o comprimento dos lados dos triângulos (comprimento do lado do plano dividido pelo número de divisões de cada eixo) são determinados os restantes vértices e, por conseguinte, os triângulos (equiláteros) constituintes do plano.

De forma a garantir que o plano é visível de qualquer ponto de vista, são calculados novamente todos os triângulos, mas desta vez, em cada triângulo, os vértices são gerados em ordem contrária à primeira. Assim é possível observar cada triângulo dos dois lados.

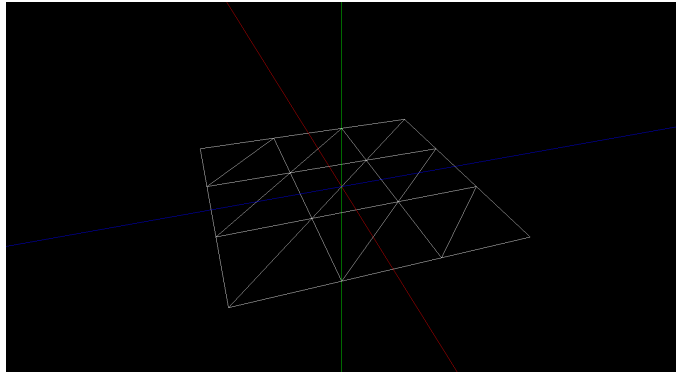


Figura 1: Exemplo de plano ("`./generator plane 2 3 plane.3d`")

2.2 Caixa

Em relação à caixa, esta é centrada na origem e é semelhante a um cubo. Tem como argumentos a dimensão(o comprimento) e o número de divisões por aresta.

Na determinação dos vértices da caixa, o grupo utilizou a implementação do plano que já tinha concebido. De uma forma muito intuitiva é possível perceber que uma caixa (um cubo) é composta por 6 planos. Com base nisto, o grupo gerou a caixa utilizando então os 6 planos ou, mais propriamente, 3 pares de planos, em que cada par é composto por dois planos, porém com a criação dos respectivos vértices na direção oposta, e com afastamento entre eles.

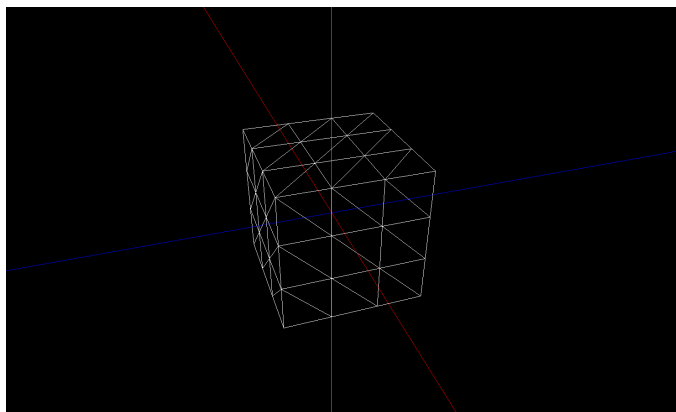


Figura 2: Exemplo de caixa ("`./generator box 1 3 box.3d`")

2.3 Cone

O cone tem a sua base no plano XZ e recebe como valores o raio, a altura, o número de arestas da base (e consequentemente número de faces laterais) e o número de níveis ou divisões horizontais (as *stacks*).

A abordagem feita pelo grupo para obter os vértices que constroem o cone foi "olhar" para o mesmo como sendo um conjunto de "bases" que vai diminuindo gradualmente até ficar apenas um ponto (o vértice do topo do cone). Esta diminuição de tamanho gradual a cada nível é calculada de acordo com o número de níveis do cone e a altura do mesmo, pois sabemos que os níveis possuem todos a mesma altura. De seguida, é feita a conexão dos vértices através dos triângulos, como mostra a figura seguinte.

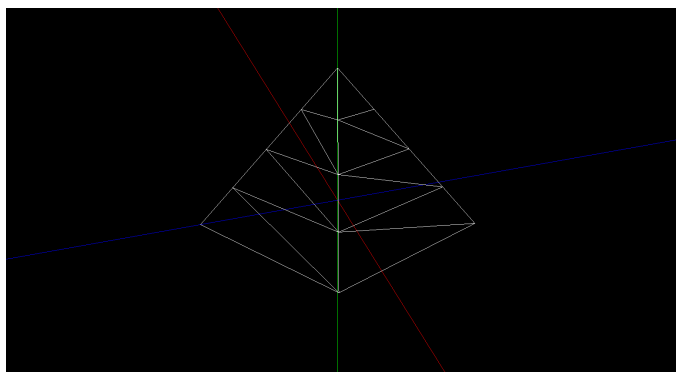


Figura 3: Exemplo de cone ("./generator cone 1 1 5 4 cone.3d")

2.4 Esfera

Relativamente à esfera, tal como no caso da caixa, esta é centrada na origem (ponto $(0,0,0)$) e recebe como argumentos o raio, o número de *slices*, isto é o número de linhas verticais que intersejam a superfície da esfera e o número de *stacks*, isto é o número de linhas horizontais que intersejam a superfície da esfera. Adicionalmente foi necessário estabelecer dois ângulos, o ângulo de *Azimuth* e o ângulo *Polar*, representados no código por *alpha* e *beta* respetivamente, representados na figura seguinte.

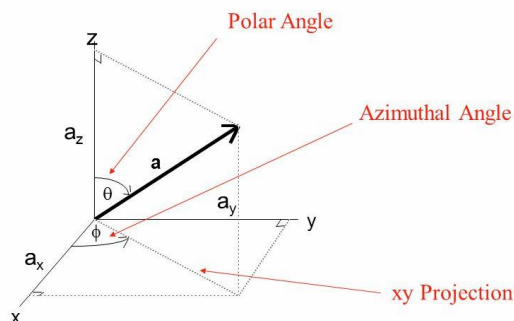
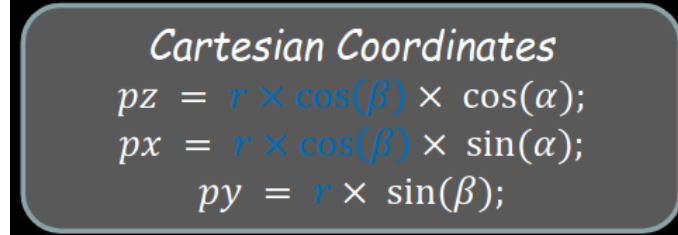


Figura 4: Sistema de Coordenadas Esféricas

Todas as coordenadas cartesianas que definem a totalidade de pontos da esfera foram geradas de acordo com as seguintes equações:



$$\begin{aligned} pz &= r \times \cos(\beta) \times \cos(\alpha); \\ px &= r \times \cos(\beta) \times \sin(\alpha); \\ py &= r \times \sin(\beta); \end{aligned}$$

Figura 5: Equações para determinação das Coordenadas Cartesianas

Note-se que estas equações foram utilizadas tendo em conta o eixo Y como sendo o eixo vertical, no entanto, no nosso projeto o eixo vertical é o eixo Z, pelo que foi necessário fazer um ajuste no cálculo do ângulo Polar, isto é, o ângulo *beta*. Sendo assim o ângulo toma o valor de $\frac{\pi}{2} - \text{beta}$. Importante salientar que o valor da variação do ângulo *Polar* é de $\frac{\pi}{stacks}$, tendo em conta que o ângulo *beta* pode variar de $-\frac{\pi}{2}$ a $\frac{\pi}{2}$ e terá tantos valores quanto o número de *stacks* e o valor da variação do ângulo de *Azimuth* é de $\frac{2\pi}{slices}$, tendo em conta que o ângulo *alpha* pode variar de 0 a 2π e terá tantos valores quanto o número de *slices*.

A partir deste conjunto de valores foi possível obter todos os pontos necessários para a criação da esfera. Para tal foi necessário iterar sobre as *stacks*, de modo a definir os planos horizontais, e para cada *stack* iterar sobre o número de *slices*, para definir os planos verticais. Em cada iteração sobre as *slices* são definidos conjuntos de 3 pontos que formam os triângulos necessários para a construção da esfera, partindo de um ponto de coordenadas (px,py,pz). De notar que os pontos px,py,pz foram obtidos a partir das equações previamente estabelecidas. Como os pontos são obtidos pela ordem inversa dos ponteiros do relógio, a partir de um ponto é simples obter os demais, sequencialmente pela soma dos respetivos valores da variação dos ângulos *alpha* e *beta* com os próprios ângulos.

Partindo de um ponto arbitrário, o próximo ponto a ser obtido será o ponto resultante da substituição do ângulo *beta*, nas equações cartesianas, pela soma do mesmo com a sua variação, ou seja, $\beta = \beta + \frac{\pi}{stacks}$. De seguida obtém-se o último ponto pela substituição do ângulo *alpha* e do ângulo *beta* pela soma da variação de cada um, ou seja $\alpha = \alpha + \frac{2\pi}{slices}$ e $\beta = \beta + \frac{\pi}{stacks}$, respetivamente. Estando assim formado o primeiro triângulo. O segundo triângulo será formado de forma análoga, partindo do primeiro e último pontos do primeiro triângulo e introduzindo um novo ponto no fim, resultante da soma do valor da variação de *alpha* com o mesmo. Completada a iteração sobre todas as *stacks* conseguimos obter todos os pontos necessários para a construção da esfera.

Finalizando assim a sua génese, os resultados obtidos podem ser observados nas seguintes figuras:

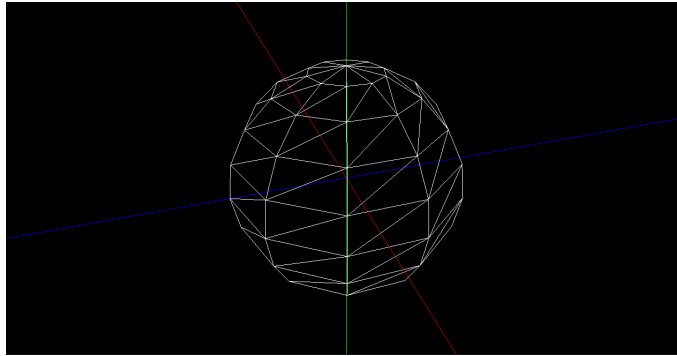


Figura 6: Exemplo de esfera ("`./generator sphere 1 10 10 sphere.3d`")

Capítulo 3

Engine

No módulo relativo ao **Engine** foi desenvolvida uma primeira versão do motor 3D. Nesta fase do projeto o objetivo desta aplicação é a leitura de um **ficheiro de configurações** e os respetivos **modelos**, gerados previamente, e que se encontram indicados neste ficheiro. A partir daí é responsável pela sua leitura e interpretação, assim como a respetiva construção dos elementos visados.

3.1 Ficheiro de Configurações

Nesta fase do projeto o ficheiro de configurações do **Engine**, escrito em XML, é passado como argumento ao programa, contendo informações acerca da definição da câmara e dos modelos que terá de carregar. Segue-se um exemplo de um ficheiro de configuração da primeira fase:

```
<world>
  <window width="512" height="512" />
  <camera>
    <position x="5" y="-2" z="3" />
    <lookAt x="0" y="0" z="0" />
    <up x="0" y="1" z="0" />
    <projection fov="20" near="1" far="1000" />
  </camera>
  <group>
    <models>
      <model file="plane.3d" />
      <model file="box.3d" />
      <model file="sphere.3d" />
    </models>
  </group>
</world>
```

Para a leitura do ficheiro optamos por usar o parser `tinxml2`, dada a sua simplicidade em relação a outras alternativas experimentadas como o `tinysql`. De forma a respeitar o requisito

definido que dita que este ficheiro só será lido uma única vez foi necessário pensar numa estrutura apropriada para um fácil acesso e armazenamento dos dados.

```
class Settings{

    /* Janela */
    int width;
    int height;

    /* Câmara */
    Point position;
    Point lookAt;
    Point up;
    Point projection;

    /* Modelos */
    vector<string> models;
}
```

A solução encontrada passou pela definição desta classe que é criada e preenchida com os dados apropriados logo no arranque do programa.

Para a definição das propriedades da janela recorreremos à função *glutInitWindowSize()*, enquanto que para o ajuste da projeção da câmara foi usada a *gluPerspective()*, ficando as restantes propriedades da câmara definidas através da *gluLookAt()*.

Em relação à leitura e interpretação dos ficheiros gerados pelo **Generator** foi acordada a sua interpretação como um conjunto de pontos que seriam guardados num **vector**. A partir daí poderiam ser acedidos 3 a 3 no sentido de gerar o triângulo pretendido, através da função *glVertex3f()*.

Capítulo 4

Análise de Resultados

Neste capítulo serão apresentados os resultados do programa quando confrontado com os testes e respectivos ficheiros de configuração disponibilizados pela equipa docente. Foram disponibilizados 5 testes, cada um com diferentes configurações de câmara e modelos para interpretar.

Seguem-se representados os resultados obtidos com o programa desenvolvido (metade esquerda da figura), e respetiva comparação com o resultado esperado (metade direita da figura).

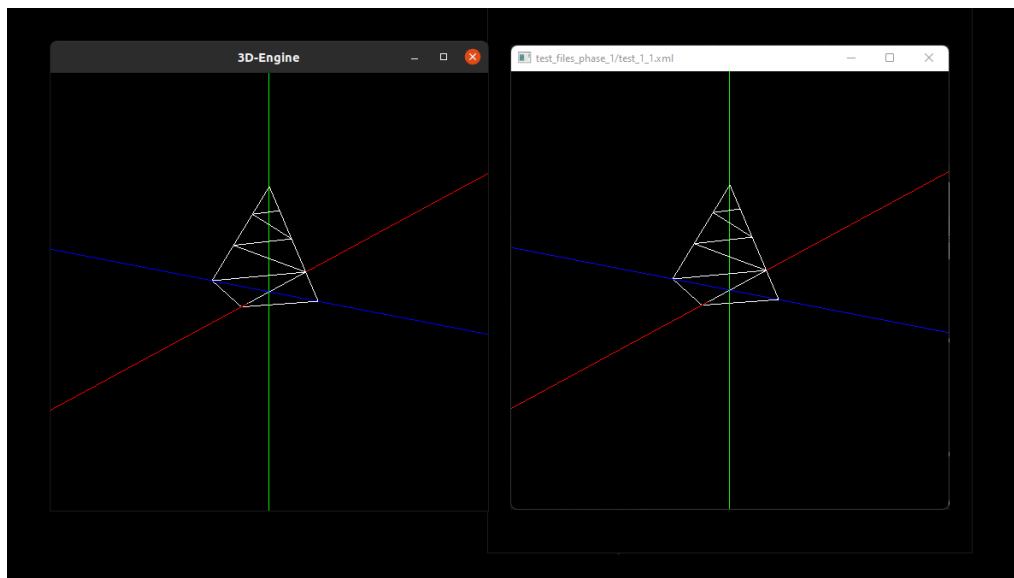


Figura 7: Teste 1

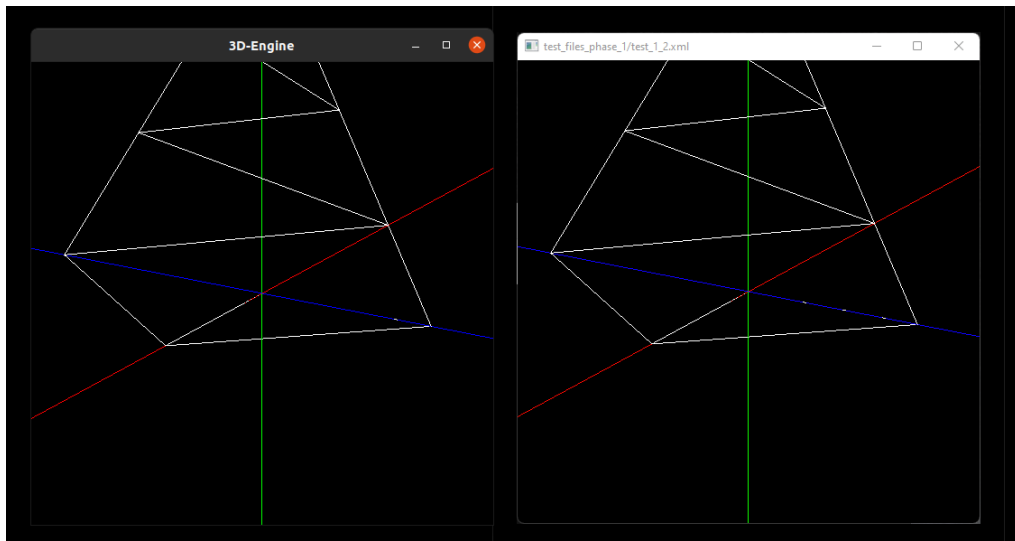


Figura 8: Teste 2

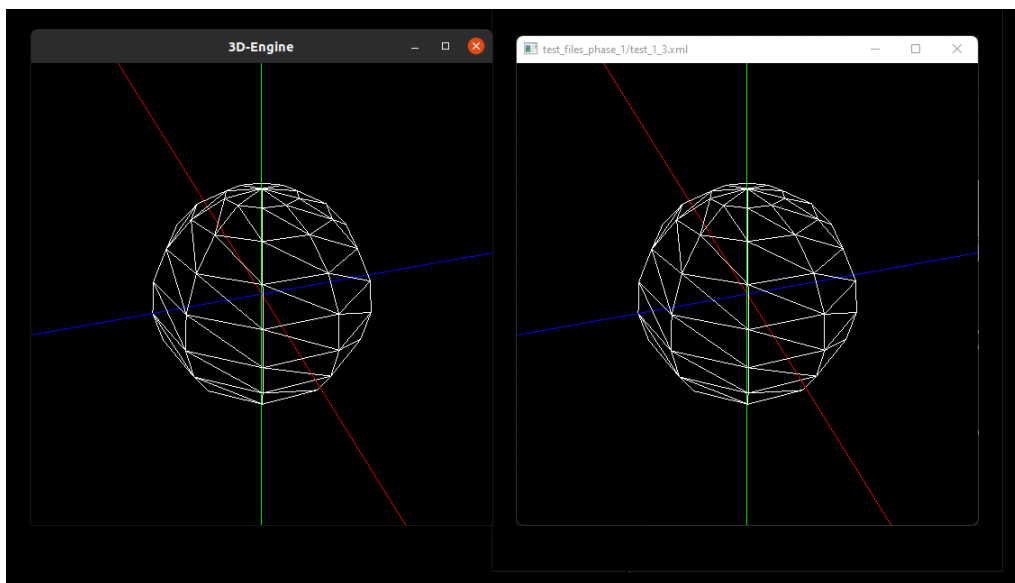


Figura 9: Teste 3

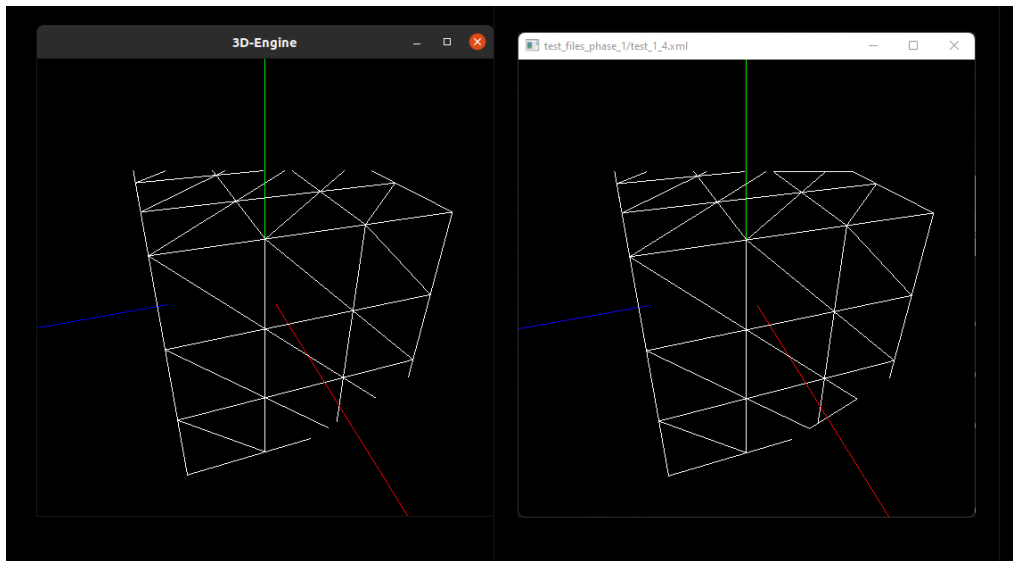


Figura 10: Teste 4

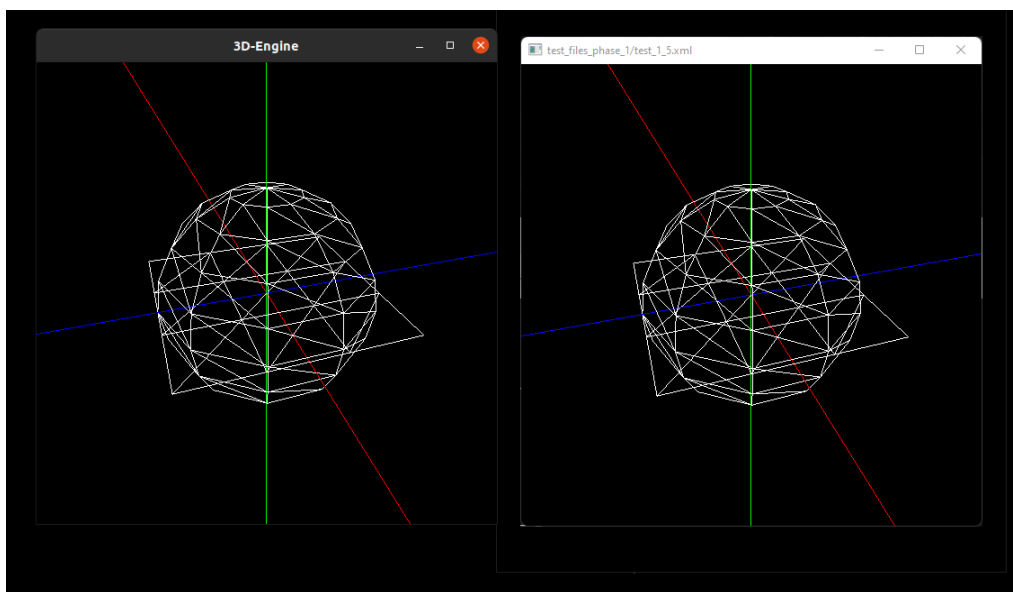


Figura 11: Teste 5

A partir da análise das figura conseguimos constatar que todos os testes foram passados com sucesso. Este resultado indica que para além das primitivas estarem a ser corretamente geradas, interpretadas e produzidas, também a câmara está a ser definida com as configurações pretendidas.

Capítulo 5

Conclusão

Esta primeira fase do projeto foi desafiadora na medida em que foi a primeira vez que o grupo trabalhou tanto com C++ como com o GLUT, para além dos exercícios praticados nas aulas práticas até à data. Num primeiro momento, o grupo definiu como iriam ser as estruturas do *Engine* e do *Generator*, passando de seguida à implementação do código.

Em relação às primitivas gráficas, a esfera foi a que gerou um grau de dificuldade ligeiramente acrescido, uma vez que tiveram de ser tomadas mais decisões na sua implementação do que nas restantes, e sendo uma figura que consideramos de uma complexidade superior às restantes, ocupou ao grupo mais tempo.

Em suma, todos os objetivos propostos pelos docentes foram cumpridos nesta primeira fase do projeto.