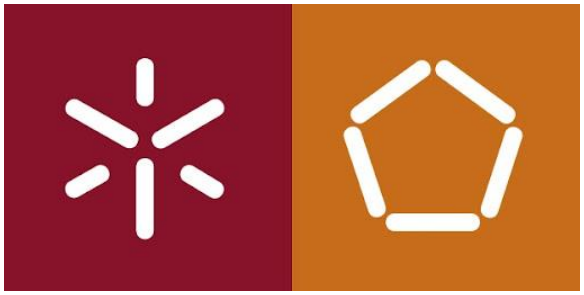

Universidade do Minho

Relatório do Projeto Fase 1

Ano letivo 2021/2022

Março 2022

Licenciatura em Engenharia Informática

Unidade Curricular de Computação Gráfica

Ana Gonçalves a93259

Bruno Pereira a93298

Francisco Toldy a93226

João Delgado a93240

Conteúdo

Introdução

Esta fase do trabalho tinha como objetivo desenvolver duas aplicações, uma para gerar os ficheiros com a informação dos modelos, neste caso, vértices dos mesmos, e depois o engine, capaz de ler os ficheiros e mostrar os modelos.

Assim, o trabalho a realizar dividiu-se em 2 subfases em que os membros do grupo se reuniram no início para dividir tarefas, e no fim para juntar o trabalho feito autonomamente no entretanto.

Na primeira subfase, foi feita a distribuição das primitivas a desenvolver, sendo definido o formato em que estas seriam guardadas em ficheiro.

Na segunda subfase foi necessário desenvolver um parser do ficheiro gerado na primeira subfase e dos ficheiros XML.

Assim sendo, no seguinte ficheiro está apresentada cada uma das primitivas, descrevendo a correspondente função de escrita, as equações para se obter o cada um dos vértices, como também o pseudocódigo em anexo.

Seguidamente, está presente a descrição de cada um dos parsers necessários para a leitura de ficheiros xml e 3d, como também a função de desenho de cada um dos triângulos.

Por último serão apresentados os resultados finais do projeto juntamente com a conclusão do desenvolvimento deste projeto.

Gerador

O gerador consiste numa aplicação com o único objetivo de obter os múltiplos vértices da primitiva escolhida, seguido da sua escrita no ficheiro 3d. Mais especificamente, a função *main* vai guardar em variáveis globais os argumentos inseridos e o número correspondente (utilizadas na determinação dos vértices objetivo), seguido da chamada da função *choosePrimitive*.

Esta última tem com única função de chamar a função adequada dependendo da primitiva fornecida como parâmetro. Isto é, no caso de a primitiva escolhida ser um plano (Ex: plano 3 4 plano.3d) os argumentos vão ser utilizados pela função *writePlane*. No caso de a primitiva escolhida por o cubo, será chamada a função *writeCube* (Ex: box 4 5 box.3d). Se for o caso da primitiva do cone (Ex: cone 1 2 4 3 cone.3d) vai ser chamada *writeCone*. E finalmente, se a primitiva escolhida for a esfera (Ex: sphere 4 5 2 sphere.3d) será chamada a função *writeSphere*.

1. Plano

O desenvolvimento do plano baseou-se em criar um ciclo que cria os triângulos que formam o mesmo. Os triângulos são criados aos pares para formar quadrados, assim, a forma como os vértices são calculados passa por, partindo de uma coordenada X e Z iniciais, aumentar/diminuir quando apropriado, o valor das mesmas, consoante o tamanho do lado do triangulo a desenhar. O tamanho do lado do triangulo é calculado dividindo o comprimento da aresta selecionada, pelo número de divisões. O X e Z iniciais a cada iteração são calculados de forma diferente:

- o X é calculado com a soma da coordenada x mais negativa possível com a linha atual em que nos encontramos, obtida através da multiplicação do comprimento de um triangulo com a divisão em que nos encontramos

$$x_{inicial} = -\left({comprimento}/{2}\right) \qquad x_{atual} = x_{inicial} + (divisão_{atual} * size)$$

- o Z é calculado com a soma da coordenada z mais positiva possível com a linha atual em que nos encontramos, obtida através da multiplicação do comprimento de um triangulo com a divisão em que nos encontramos

$$z_{inicial} = {comprimento}/{2} \qquad z_{atual} = z_{inicial} - (divisão_{atual} * size)$$

A coordenada X mais negativa é obtida através do simétrico da metade do comprimento de um lado do plano, e a coordenada Y obtida apenas pela metade do comprimento do lado do plano. Sendo que a metade do comprimento é utilizada de modo a obter um plano centralizado na origem do referencial.

A seguir apresentamos um esquema de como é construído o plano, sempre com base numa "linha", tanto no eixo dos x como dos z. E também podemos verificar a que triângulos são referidos no pseudocódigo (Figura 25) em anexo.

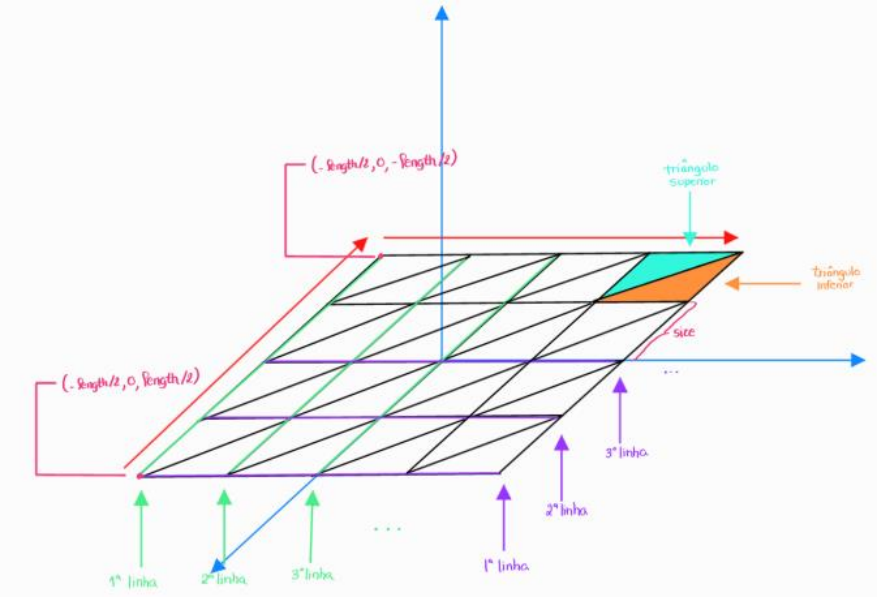

Figura 1 - Formato de Desenho do Plano

2. Cubo

O desenvolvimento desta primitiva baseou-se muito na forma como a primitiva anterior foi elaborada. Assim, ao invés de se ter recorrido a um ciclo para desenvolver um plano, a contagem de ciclos subiu para 3, sendo cada um deles dedicado a 3 pares de planos paralelos que constituem o cubo.

O primeiro ciclo forma as bases horizontais da caixa, sendo que cria simultaneamente os vértices das duas bases. Cada uma destas bases possui uma coordenada Y constante. Cada uma destas bases possui uma coordenada Y constante (valor na base inferior é o valor negativo da divisão do tamanho da aresta por 2 e na base superior o valor é o simétrico da inferior).

$$x_{inicial} = -\left({comprimento}/{2}\right) \qquad x_{atual} = x_{inicial} + (divisão_{atual} * size)$$

$$z_{inicial} = {comprimento}/{2} \qquad z_{atual} = z_{inicial} - (divisão_{atual} * size)$$

O segundo ciclo forma 2 das faces verticais da caixa, nomeadamente a frente-esquerda e face que a esta é paralela. Analogamente ao primeiro ciclo, cada uma destas faces possui uma coordenada Z constante (calculada da mesma forma que no primeiro ciclo)

$$x_{inicial} = -\left({comprimento}/{2}\right) \qquad x_{atual} = x_{inicial} + (divisão_{atual} * size)$$

$$y_{inicial} = -\left({comprimento}/{2}\right) \qquad y_{atual} = y_{inicial} + (divisão_{atual} * size)$$

Por último, o terceiro ciclo é responsável por criar os vértices das restantes faces verticais (frente direita e trás-esquerda). Analogamente aos ciclos anteriores, cada uma destas faces possui uma coordenada X constante.

$$z_{inicial} = -\left({comprimento}/{2}\right) \qquad z_{atual} = z_{inicial} + (divisão_{atual} * size)$$

$$y_{inicial} = -\left({comprimento}/{2}\right) \qquad y_{atual} = y_{inicial} + (divisão_{atual} * size)$$

À semelhança do que foi apresentado na secção relativa à primitiva do plano, segue-se o gráfico que ilustra a construção de ambas as bases do cubo.

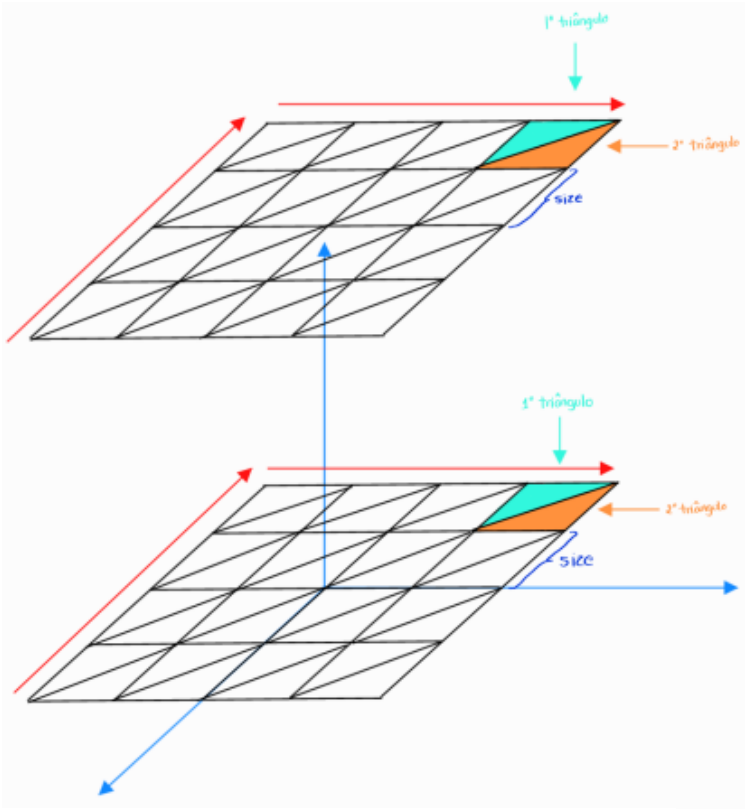

Figura 2 - Formato de Desenho das Bases do Cubo

Segue-se o gráfico que ilustra a construção das fases “frente esquerda” e “trás direita”.

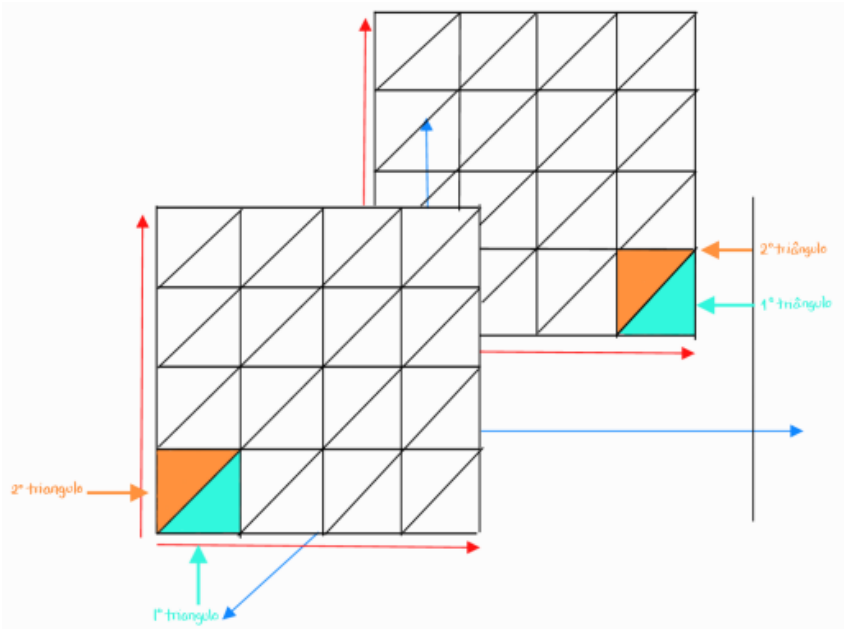

Figura 3 - Formato de Desenho das Faces "Frente esquerda" e "Trás Direita"

E finalmente, segue-se o gráfico que ilustra a construção das fases “frente direita” e “trás esquerda”.

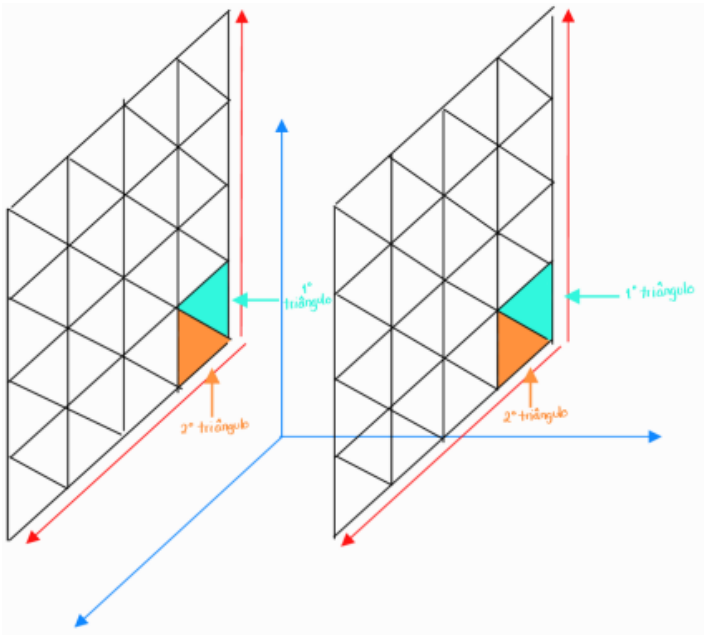

Figura 4 - Formato de Desenho das Faces "Frente Direita" e "Trás Esquerda"

3. Cone

Para desenhar um cone são necessários 4 parâmetros:

1. Altura
2. Raio
3. Número de slices
4. Número de stacks

As slices correspondem a divisões verticais do cone, e as stacks correspondem a divisões horizontais do cone.

a. Stacks

Para atingir o desenho final do cone, dividiu-se o problema em problemas mais pequenos, focando primeiro a atenção no desenho de apenas uma slice, sendo esta subdividida nas suas stacks.

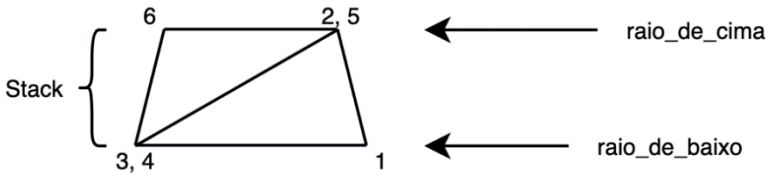

Figura 5 - Exemplo de Stack do Cone

Uma stack de uma slice é na verdade um trapézio que pode ser desenhado sob a forma de dois triângulos como se pode ver na Figura 5. Sendo assim começa-se por se desenhar o triângulo cuja base coincide com a base inferior do trapézio.

Para o desenho do triangulo é necessário calcular as coordenadas dos seus vértices, para isso recorreu-se ao uso de coordenadas polares.

Para a utilização de coordenadas polares na resolução desta tarefa foi necessário calcular o ângulo necessário mover as coordenadas horizontalmente de um vértice do triangulo para outro, e isso foi feito dividindo 2π pelo número de slices necessárias para desenhar o cone.

Assim para calcular o vértice número 1 da *Figura 5* utilizou-se as seguintes formulas:

$$coordenada_x = raiodebaixo \times \sin(angulo\ de\ baixo)$$

$$coordenada_z = raiodebaixo \times \cos(angulo\ de\ baixo)$$

$$coordenada_y = stackatual \times alturadestack$$

Onde o ângulo de baixo corresponde ao ângulo utilizado na face inferior e o raio de baixo corresponde ao raio da face inferior da stack que é sempre maior que o raio da face superior da stack.

A ordem de desenho dos vértices corresponde à apresentada na *Figura 5*.

Quando a stack estiver desenhada, o raio da face inferior passa a ter o valor da antiga face superior , e é calculado o raio da nova face superior através da expressão:

$$raiodecima = raiodecima - stepdoraio$$

E o processo repete-se novamente, até toda a slice estar desenhada.

b. Slice

Uma slice, como foi mencionado anteriormente, é constituída por várias stacks.

Portanto, depois de se conseguir desenhar uma slice, basta repetir o processo para todas as slices do cone.

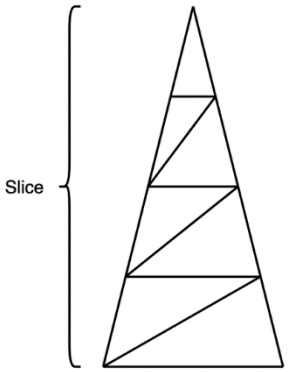

Figura 6 - Exemplo de uma Slice do Cone

c. Base

Para o desenho de base, são novamente utilizadas coordenadas polares. A base é dividida em triângulos sendo que para calcular os vértices que constituem cada um desses triângulos é necessário utilizar as seguintes expressões:

$$coordenada_x = raio \times \sin(angulo)$$

$$coordenada_z = raio \times \cos(angulo)$$

$$coordenada_y = 0.0$$

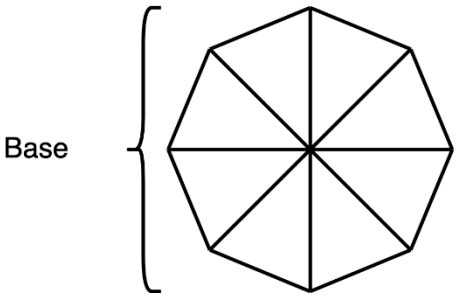

Figura 7 - Base do Cone

4. Esfera

Os pontos da esfera são gerados com base nas coordenadas esféricas lecionadas nas aulas práticas.

A sua estrutura principal baseia-se em dois ciclos. O ciclo interior, que vai gerar os vértices para todas as slices para uma dada stack. Este ciclo, está envolvido por um ciclo exterior, cujo objetivo é repetir este processo para todas as stacks da esfera. Assim, estarão gerados e escritos os vértices no ficheiro "sphere.3d".

a. Estruturas Criadas

Uma struct, vertex, que contém a informação relativa a um ponto. Desta forma, é possível manter a informação das 3 coordenadas do ponto (x,y,z) numa estrutura juntas.

b. Função *AssignCoords*

É uma função cujo propósito é diminuir a quantidade de código e aumentar a legibilidade do mesmo na função *sphereCoords*, mantendo as equações das coordenadas esféricas dentro desta

Esta função, com base nos parâmetros:

- raio da esfera
- stack que está a ser gerada
- slice que está a ser gerada
- valor do ângulo beta inicial
- quanto terá de andar o ângulo β em cada stack que será gerada (deslocamento β)
- quanto terá de andar o ângulo α a cada slice que será gerada (deslocamento α)

Irá criar um vértice com as coordenadas px, py, pz, com base nas equações das coordenadas esféricas:

$$p_x = r * \cos(\beta) * \sin(\alpha)$$

$$p_y = r * \sin(\beta)$$

$$p_z = r * \cos(\beta) * \cos(\alpha)$$

Em que:

$$\begin{cases} \beta = \frac{\pi}{4} - (deslocamento\ \beta) * \text{stack}_{\text{atual}} \\ \alpha = \text{slice}_{\text{atual}} * deslocamento\ \alpha \\ deslocamento\ \beta = \frac{\pi}{\text{stacks}} \\ deslocamento\ \alpha = \frac{2\pi}{\text{slices}} \end{cases}$$

c. Função sphereCoords

Esta função tem como objetivo gerar os pontos da esfera e, depois de gerados, escrevê-los no ficheiro pela ordem que é desejado que sejam desenhados.

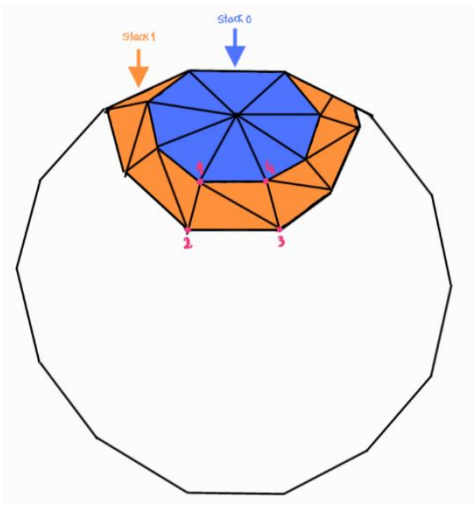

Figura 8 - Ordem pela qual os vértices são escritos: (1,2,3) (1,3,4)

Os vértices vão ter as seguintes coordenadas:

$$Vértice\ 1: \begin{cases} \beta = \frac{\pi}{4} - (deslocamento_\beta * 1) \\ \alpha = 0 * \text{deslocamento}_\alpha = 0 \end{cases}$$

$$Vértice\ 2: \begin{cases} \beta = \frac{\pi}{4} - \left(deslocamento_\beta * (1+1)\right) = \frac{\pi}{4} - 2 * deslocamento_\beta \\ \alpha = 0 * \text{deslocamento}_\alpha = 0 \end{cases}$$

$$Vértice\ 3: \begin{cases} \beta = \frac{\pi}{4} - 2deslocamento_\beta \\ \alpha = (0+1) * \text{deslocamento}_\alpha = \text{deslocamento}_\alpha \end{cases}$$

$$Vértice\ 4: \begin{cases} \beta = \frac{\pi}{4} - deslocamento_\beta \\ \alpha = \text{deslocamento}_\alpha \end{cases}$$

Engine

1. Parser de Ficheiros XML

O parser de ficheiro XML, *readXML,* tem como único objetivo aceder ao ficheiro xml fornecido e retirar as informações relativamente as características da câmera como também os nomes dos ficheiros que contêm os vêrtices dos modelos a ser desenhados.

2. Parser de Ficheiros 3d

Esta função tem como objetivo ler as coordenadas dos pontos do ficheiro .3d, colocá-las num vetor intermédio para, posteriormente, com o auxílio da função draw, enviá-las para a placa gráfica para poderem ser desenhados sob a forma de triângulos.

Resultados

1. Teste 1

```xml
<world>
    <camera>
        <position x="5" y="-2" z="3" />
        <lookAt x="0" y="0" z="0" />
        <up x="0" y="1" z="0" />
        <projection fov="60" near="1" far="1000" />
    </camera>
    <group>
        <models>
            <model file="cone.3d" /> <!-- generator cone 1 2 4 3 cone.3d -->
        </models>
    </group>
</world>
```

Figura 9: Ficheiro Test_1_1.xml

```
0;0;1;0;0.666667;0.666667;-1;0;-4.37114e-08
-1;0;-4.37114e-08;0;0.666667;0.666667;-0.666667;0.666667;-2.91409e-08
0;0.666667;0.666667;0;1.33333;0.333333;-0.666667;0.666667;-2.91409e-08
-0.666667;0.666667;-2.91409e-08;0;1.33333;0.333333;-0.333333;1.33333;-1.45705e-08
0;1.33333;0.333333;-0;2;-5.96046e-08;-0.333333;1.33333;-1.45705e-08
-0.333333;1.33333;-1.45705e-08;-0;2;-5.96046e-08;5.96046e-08;2;2.6054e-15
-1;0;-4.37114e-08;-0.666667;0.666667;-2.91409e-08;8.74228e-08;0;-1
8.74228e-08;0;-1;-0.666667;0.666667;-2.91409e-08;5.82818e-08;0.666667;-0.666667
-0.666667;0.666667;-2.91409e-08;-0.333333;1.33333;-1.45705e-08;5.82818e-08;0.666667;-0.666667
5.82818e-08;0.666667;-0.666667;-0.333333;1.33333;-1.45705e-08;2.91409e-08;1.33333;-0.333333
-0.333333;1.33333;-1.45705e-08;5.96046e-08;2;2.6054e-15;2.91409e-08;1.33333;-0.333333
2.91409e-08;1.33333;-0.333333;5.96046e-08;2;2.6054e-15;-5.2108e-15;2;5.96046e-08
8.74228e-08;0;-1;5.82818e-08;0.666667;-0.666667;1;0;1.19249e-08
```

Figura 10: Ficheiro cone.3d

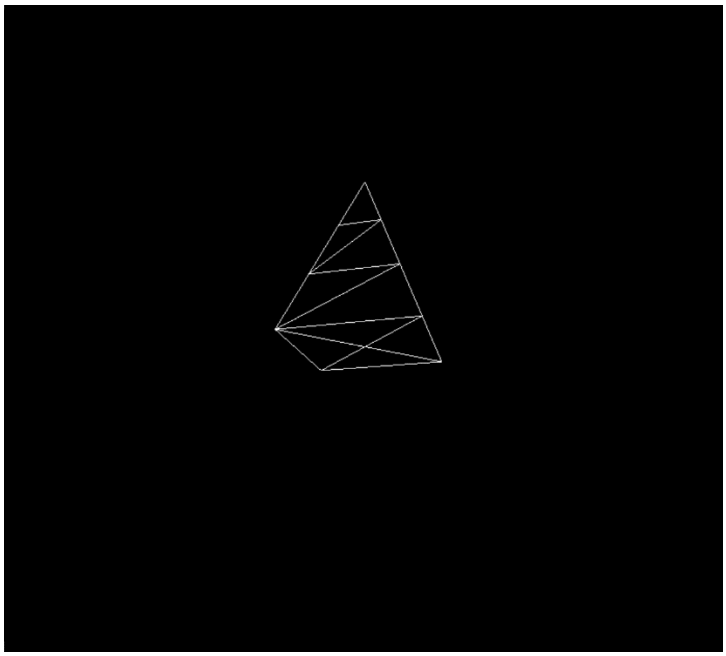

Figura 11: Resultado final test_1_1

2. Teste 2

```xml
<world>
    <camera>
        <position x="5" y="-2" z="3" />
        <lookAt x="0" y="0" z="0" />
        <up x="0" y="1" z="0" />
            <projection fov="40" near="1" far="1000" />
    </camera>
    <group>
        <models>
            <model file="cone.3d" /> <!-- generator cone 1 2 4 3 cone.3d -->
        </models>
    </group>
</world>
```

Figura 12: Ficheiro Teste_1_2.xml

```
0;0;1;0;0.666667;0.666667;-1;0;-4.37114e-08
-1;0;-4.37114e-08;0;0.666667;0.666667;-0.666667;0.666667;-2.91409e-08
0;0.666667;0.666667;0;1.33333;0.333333;-0.666667;0.666667;-2.91409e-08
-0.666667;0.666667;-2.91409e-08;0;1.33333;0.333333;-0.333333;1.33333;-1.45705e-08
0;1.33333;0.333333;-0;2;-5.96046e-08;-0.333333;1.33333;-1.45705e-08
-0.333333;1.33333;-1.45705e-08;-0;2;-5.96046e-08;5.96046e-08;2;2.6054e-15
-1;0;-4.37114e-08;-0.666667;0.666667;-2.91409e-08;8.74228e-08;0;-1
8.74228e-08;0;-1;-0.666667;0.666667;-2.91409e-08;5.82818e-08;0.666667;-0.666667
-0.666667;0.666667;-2.91409e-08;-0.333333;1.33333;-1.45705e-08;5.82818e-08;0.666667;-0.666667
5.82818e-08;0.666667;-0.666667;-0.333333;1.33333;-1.45705e-08;2.91409e-08;1.33333;-0.333333
-0.333333;1.33333;-1.45705e-08;5.96046e-08;2;2.6054e-15;2.91409e-08;1.33333;-0.333333
2.91409e-08;1.33333;-0.333333;5.96046e-08;2;2.6054e-15;-5.2108e-15;2;5.96046e-08
```

Figura 13: Ficheiro cone.3d

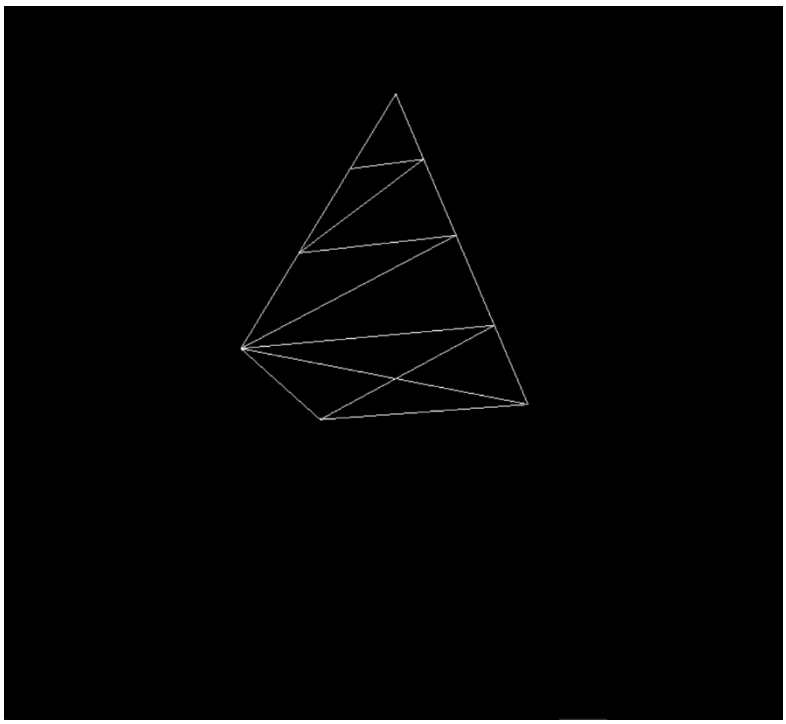

Figura 14: Resultado final de test_1_2

O resultado neste teste não foi o pretendido devido às configurações do teste fornecido, mas seria o correto com um valor de fov menor do que o que está presente no ficheiro.

3. Teste 3

```xml
<world>
    <camera>
        <position x="3" y="2" z="1" />
        <lookAt x="0" y="0" z="0" />
        <up x="0" y="1" z="0" />
            <projection fov="60" near="1" far="1000" />
    </camera>
    <group>
        <models>
            <model file="sphere.3d" /> <!-- generator sphere 1 10 10 sphere.3d -->
        </models>
    </group>
</world>
```

Figura 15: Ficheiro test_1_3.xml

```
-0;1;-4.37114e-08;0;0.951057;0.309017;0.181636;0.951057;0.25
-0;1;-4.37114e-08;0.181636;0.951057;0.25;-2.56929e-08;1;-3.53633e-08
-2.56929e-08;1;-3.53633e-08;0.181636;0.951057;0.25;0.293893;0.951057;0.0954915
-2.56929e-08;1;-3.53633e-08;0.293893;0.951057;0.0954915;-4.1572e-08;1;-1.35076e-08
-4.1572e-08;1;-1.35076e-08;0.293893;0.951057;0.0954915;0.293893;0.951057;-0.0954915
-4.1572e-08;1;-1.35076e-08;0.293893;0.951057;-0.0954915;-4.1572e-08;1;1.35076e-08
-4.1572e-08;1;1.35076e-08;0.293893;0.951057;-0.0954915;0.181636;0.951057;-0.25
-4.1572e-08;1;1.35076e-08;0.181636;0.951057;-0.25;-2.56929e-08;1;3.53633e-08
-2.56929e-08;1;3.53633e-08;0.181636;0.951057;-0.25;-2.70151e-08;0.951057;-0.309017
-2.56929e-08;1;3.53633e-08;-2.70151e-08;0.951057;-0.309017;3.82137e-15;1;4.37114e-08
3.82137e-15;1;4.37114e-08;-2.70151e-08;0.951057;-0.309017;-0.181636;0.951057;-0.25
3.82137e-15;1;4.37114e-08;-0.181636;0.951057;-0.25;2.56929e-08;1;3.53633e-08
2.56929e-08;1;3.53633e-08;-0.181636;0.951057;-0.25;-0.293893;0.951057;-0.0954915
2.56929e-08;1;3.53633e-08;-0.293893;0.951057;-0.0954915;4.1572e-08;1;1.35076e-08
4.1572e-08;1;1.35076e-08;-0.293893;0.951057;-0.0954915;-0.293893;0.951057;0.0954915
```

Figura 16: Ficheiro sphere.3d

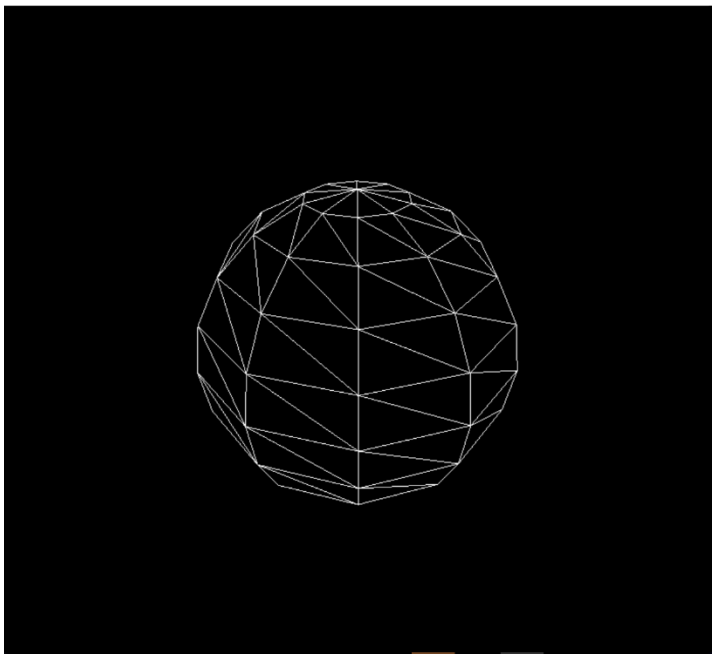

Figura 17: Resultado final do test_1_3

4. Teste 4

```xml
<world>
    <camera>
        <position x="3" y="2" z="1" />
        <lookAt x="0" y="0" z="0" />
        <up x="0" y="1" z="0" />
            <projection fov="60" near="1" far="3.5" />
    </camera>
    <group>
        <models>
            <model file="box.3d" /> <!-- generator box 2 3 box.3d -->

        </models>
    </group>
</world>
```

Figura 18: Ficheiro test_1_4.xml

```
-1;-1;1;-0.333333;-1;0.333333;-0.333333;-1;1
-1;-1;1;-1;-1;0.333333;-0.333333;-1;0.333333
-1;1;1;-0.333333;1;1;-0.333333;1;0.333333
-1;1;1;-0.333333;1;0.333333;-1;1;0.333333
-0.333333;-1;1;0.333333;-1;0.333333;0.333333;-1;1
-0.333333;-1;1;-0.333333;-1;0.333333;0.333333;-1;0.333333
-0.333333;1;1;0.333333;1;1;0.333333;1;0.333333
-0.333333;1;1;0.333333;1;0.333333;-0.333333;1;0.333333
0.333333;-1;1;1;-1;0.333333;1;-1;1
0.333333;-1;1;0.333333;-1;0.333333;1;-1;0.333333
0.333333;1;1;1;1;1;1;1;0.333333
0.333333;1;1;1;1;0.333333;0.333333;1;0.333333
-1;-1;0.333333;-0.333333;-1;-0.333333;-0.333333;-1;0.333333
-1;-1;0.333333;-1;-1;-0.333333;-0.333333;-1;-0.333333
-1;1;0.333333;-0.333333;1;0.333333;-0.333333;1;-0.333333
```

Figura 19: Ficheiro box.3d

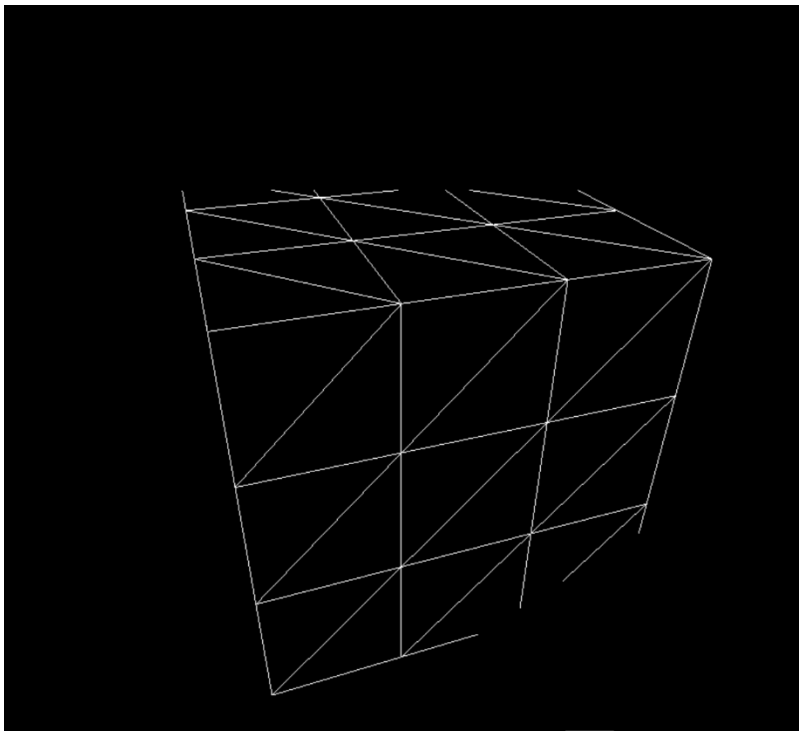

Figura 20: Resultado final do test_1_4

5. Teste 5

```xml
<world>
    <camera>
        <position x="3" y="2" z="1" />
        <lookAt x="0" y="0" z="0" />
        <up x="0" y="1" z="0" />
            <projection fov="60" near="1" far="1000" />
    </camera>
    <group>
        <models>
            <model file="plane.3d" /> <!-- generator plane 3 plane.3d -->
            <model file="sphere.3d" /> <!-- generator sphere 1 10 10 sphere.3d -->

        </models>
    </group>
</world>
```

Figura 21: Ficheiro test_1_5.xml

```
-1.5;0;1.5;-0.5;0;1.5;-0.5;0;0.5
-1.5;0;1.5;-0.5;0;0.5;-1.5;0;0.5
-0.5;0;1.5;0.5;0;1.5;0.5;0;0.5
-0.5;0;1.5;0.5;0;0.5;-0.5;0;0.5
0.5;0;1.5;1.5;0;1.5;1.5;0;0.5
0.5;0;1.5;1.5;0;0.5;0.5;0;0.5
-1.5;0;0.5;-0.5;0;0.5;-0.5;0;-0.5
-1.5;0;0.5;-0.5;0;-0.5;-1.5;0;-0.5
-0.5;0;0.5;0.5;0;0.5;0.5;0;-0.5
-0.5;0;0.5;0.5;0;-0.5;-0.5;0;-0.5
0.5;0;0.5;1.5;0;0.5;1.5;0;-0.5
0.5;0;0.5;1.5;0;-0.5;0.5;0;-0.5
```

Figura 22: Ficheiro plane.3d

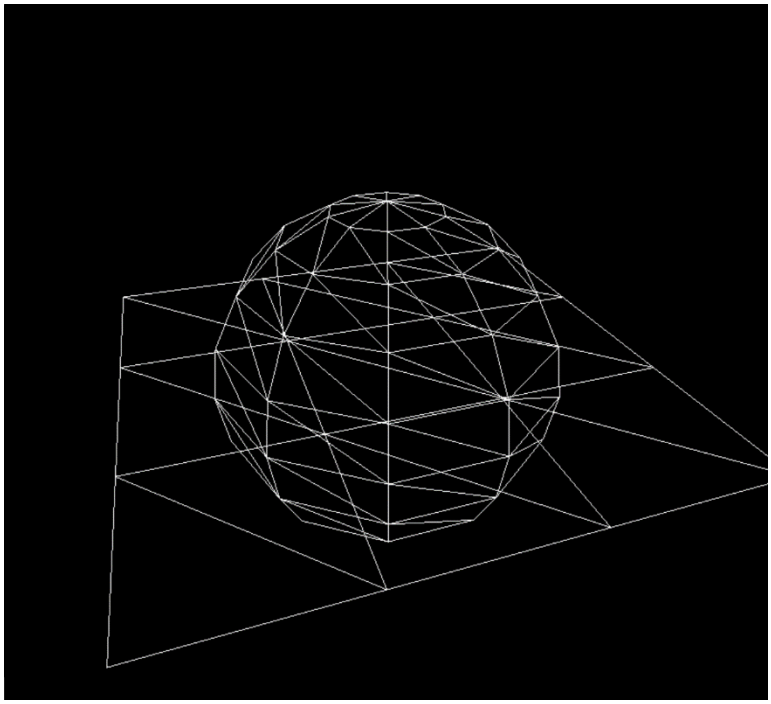

Figura 23: Resultado final do test_1_5

Conclusão

Terminada a primeira fase deste trabalho, foram atingidos todos os objetivos propostos pelo docente. Os resultados obtidos estão maioritariamente em concordância com os resultados indicados no ficheiro de testes fornecido aos alunos e o código apresentado é legível e compreensível com o auxílio do pseudocódigo anexado neste relatório. O grupo considera que aspetos a melhorar seriam, por exemplo, adicionar mais primitivas opcionais para o utilizador. Apesar disso, consideramos que com o que neste documento se encontra apresentado, e a melhor compreensão das ferramentas a utilizar que foi conseguida, estão estabelecidas corretamente as bases para as fases seguintes deste trabalho.

Anexos

```
void drawPlane(comprimento, número_de_Divisões, nome_do_ficheiro){

    Abrir ficheiro de escrita

    size corresponde ao tamanho de um lado do triangulo
    startx corresponde a posicao mais negativa no eixo dos x que o plano pode tomar
    startxFixo corresponde a posicao mais negativa no eixo dos x que o plano pode tomar, constante ao longo da função
    starty corresponde a posicao mais positiva no eixo dos y que o plano pode tomar
    startyFixo corresponde a posicao mais positiva no eixo dos y que o plano pode tomar, constante ao longo da função

    Para cada número de divisões que o plano tem{

        O inicio de cada linha no eixo dos z corresponde à posição mais positiva possível no eixo dos z mais o actual número da divisão vezes o tamanho de cada divisão

        Para cada número de divisões que o plano tem{

            O inicio de cada linha no eixo dos x corresponde à posição mais negativa possível no eixo dos x mais o actual número da divisão vezes o tamanho de cada divisão

            Escrever no ficheiro o triangulo inferior:
                Escrever ponto superior esquerdo que corresponde à interseção da linha inicial do eixo dos x e a linha inicial do eixo dos z
                Escrever ponto inferior esquerdo que corresponde à interseção da linha inicial do eixo dos x mais o tamanho de uma divisão e a linha inicial do eixo dos z
                Escrever ponto inferior direito que corresponde à interseção da linha inicial do eixo dos x mais o tamanho de uma divisão e a linha inicial do eixo dos z mais o tamanho de uma divisão

            Escrever no ficheiro o triangulo superior:
                Escrever ponto superior esquerdo que corresponde à interseção da linha inicial do eixo dos x e a linha inicial do eixo dos z
                Escrever ponto inferior direito que corresponde à interseção da linha inicial do eixo dos x mais o tamanho de uma divisão e a linha inicial do eixo dos z mais o tamanho de uma divisão
                Escrever ponto supeior direito que corresponde à interseção da linha inicial do eixo dos x e a linha inicial do eixo dos z mais o tamanho de uma divisão
        }
    }

    Fechar ficheiro de escrita

}
```

Figura 25: Pseudocódigo da função writePlane

```
void drawCube(int length, int divisions, char*fileName){

    inicializa o ficheiro correspondente a fileName

    // ------------- BASES CUBO

    size => comprimento de cada lado dos triangulos ao dividir length por divions
    startx => define a posição inicial do x como o simétrico da divisão da length por 2
    startxFixo => define o x como o ponto de inicio fixo para cada face a desenhar
    startz => definido calculando a posição inicial do z dividindo a length por 2 (startz)
    startzFixo => define o z  como o ponto de inicio fixo para cada face a desenhar
    bottomY => define o y da base inferior como o simetrico da metade da length
    topY=> define o y da base inferior como metade da length


    inicializar uma variavel j e enquanto essa variavel nao chegar ao valor de divisions {

        definir startz como a subtração de j multiplicado por size e o z fixo inicial

        inicializar uma variavel i e enquanto essa variavel nao chegar ao valor de divisions fazer {

            definir startx como a soma do x fixo e a multiplicação de i por size

            // ----------- BASE INFERIOR

            definir os 3 vertices de um triangulo:
            definir vertice um com as coordenadas startx, bottomY, startz
            definir vertice dois com as coordenadas: startx + size, bottomY, startz - size
            definir vertice tres com coordenadas : startx + size, bottomY, startz

            Guardar os pontos em ficheiro no formato "coordenadas1 ; coordenadas2 ; coordenadas3"

            definir os 3 vertices do segundo triangulo:
            definir vertice um com as coordenadas startx, bottomY, startz
            definir vertice dois com as coordenadas: startx, bottomY, startz - size
            definir vertice tres com coordenadas : startx + size, bottomY, startz - size

            Guardar os pontos em ficheiro no formato "coordenadas1 ; coordenadas2 ; coordenadas3"
          // ---------- BASE SUPERIOR


          definir os 3 vértices de um triangulo
          definir vertice um com as coordenadas startx, topY, startz
          definir vertice dois com as coordenadas: startx + size, topY, startz
          definir vertice tres com coordenadas : startx + size, topY, startz - size

          Guardar os pontos em ficheiro no formato "coordenadas1; coordenadas2; coordenadas3"

          definir os 3 vertices do segundo triangulo
          definir vertice um com as coordenadas : startx, topY, startz
          definir vertice dois com as coordenadas : startx + size, topY, startz - size
          definir vertice tres com as coordenadas : startx, topY, startz - size

          Guardar os pontos em ficheiro no formato "coordenadas1 ; coordenadas2 ; coordenadas3"

        }

    }
```

Figura 24: Pseudocódigo da função writeCube

```
// ---------------- frente esquerda e trás-direita -------------------

size => comprimento de cada lado dos triangulos ao dividir length por divisions
startx => define a posição inicial do x como o simétrico da divisão da length por 2
startzFixo => define o z como o ponto de inicio fixo para cada face a desenhar

startz => definido calculando a posição inicial do z como o simetrico da divisao  de length por 2
endz => simetrico do startz
starty => definido calculando a posição inicial do y como o simetrico da divisao  de length por 2
startYFixo => define o y como o ponto de inicio fixo para cada face a desenhar

para cada divisao {

    definir starty como a soma da multiplicação do indice do ciclo por size com o z fixo inicial

    para cada divisao que existir {

        definir startx como a soma do startYfixo com a multiplicação do indice do ciclo por size


        //------- LADO 1 ------ Trás Direita

        definir os 3 vértices de um triangulo
        definir vertice um com as coordenadas startx, starty, startz
        definir vertice dois com as coordenadas: startx + size, starty, startz
        definir vertice tres com coordenadas : startx, starty+size, startz

        Guardar os pontos em ficheiro no formato "coordenadas1; coordenadas2; coordenadas3"

        definir os 3 vertices do segundo triangulo
        definir vertice um com as coordenadas : startx, starty + size, startz
        definir vertice dois com as coordenadas : startx + size, starty, startz
        definir vertice tres com as coordenadas : startx + size, starty + size, startz

        Guardar os pontos em ficheiro no formato "coordenadas1 ; coordenadas2 ; coordenadas3"

        //------- LADO 2 ------ Frente Esquerda

        definir os 3 vértices de um triangulo
        definir vertice um com as coordenadas startx, starty, endz
        definir vertice dois com as coordenadas: startx, starty + size, endz
        definir vertice tres com coordenadas : startx + size, starty, endz

        Guardar os pontos em ficheiro no formato "coordenadas1; coordenadas2; coordenadas3"

        definir os 3 vertices do segundo triangulo
        definir vertice um com as coordenadas : startx, starty + size, endz
        definir vertice dois com as coordenadas : startx + size, starty + size, endz
        definir vertice tres com as coordenadas : startx + size, starty, endz

        Guardar os pontos em ficheiro no formato "coordenadas1 ; coordenadas2 ; coordenadas3"
    }
}

    // ---------------- frente direita e trás-esquerda -------------------

size => comprimento de cada lado dos triangulos ao dividir length por divisions
startx => define a posição inicial do x como o simétrico da divisão da length por 2
startz => definido calculando a posição inicial do z como o simetrico da divisao  de length por 2
startzFixo => define o z como o ponto de inicio fixo para cada face a desenhar
starty => definido calculando a posição inicial do y como o simetrico da divisao  de length por 2
startYFixo => define o y como o ponto de inicio fixo para cada face a desenhar

para cada divisao {

    definir startz como a soma da multiplicação do indice do ciclo por size com o z fixo inicial

    para cada divisao que existir {

        definir starty como a soma do startYfixo com a multiplicação do indice do ciclo por size


        //------- LADO 1 ------ Frente Direita

        definir os 3 vértices de um triangulo
        definir vertice um com as coordenadas startx, starty, startz+size
        definir vertice dois com as coordenadas: startx, starty + size, startz
        definir vertice tres com coordenadas : startx, starty, startz

        Guardar os pontos em ficheiro no formato "coordenadas1; coordenadas2; coordenadas3"

        definir os 3 vertices do segundo triangulo
        definir vertice um com as coordenadas : startx, starty + size, startz
        definir vertice dois com as coordenadas : startx, starty, startz+size
        definir vertice tres com as coordenadas : startx, starty + size, startz+size

        Guardar os pontos em ficheiro no formato "coordenadas1 ; coordenadas2 ; coordenadas3"
```

Figura 26: Pseudocódigo da função writeCube

```
            //------- LADO 2 ------ Tras Esquerda

            definir os 3 vértices de um triangulo
            definir vertice um com as coordenadas endx, starty, startz
            definir vertice dois com as coordenadas: endx, starty + size, startz
            definir vertice tres com coordenadas : endx, starty, startz + size

            Guardar os pontos em ficheiro no formato "coordenadas1; coordenadas2; coordenadas3"

            definir os 3 vertices do segundo triangulo
            definir vertice um com as coordenadas : endx, starty + size, startz + size
            definir vertice dois com as coordenadas : endx, starty, startz + size
            definir vertice tres com as coordenadas : endx, starty + size, startz

            Guardar os pontos em ficheiro no formato "coordenadas1 ; coordenadas2 ; coordenadas3"
        }
    }
```

Figura 27: Pseudocódigo da função writeCube

```
desenhar cone(raio, altura, slices, stacks)

    step_do_raio = raio / numero_de_stacks;

    raio_de_baixo = raio;

    raio_de_cima = raio - step_do_raio;

    altura_da_stack = altura / numero_de_stacks;

    step_do_angulo = 360 graus / numero_de_slices;


    //desenhar slices
    slice_atual = 0;
    enquanto slice_atual < numero_slices{


        stack_atual = 0
        enquanto stak_atual < numero_stacks{

            //primeiro triangulo
            //calcular primeiro ponto
            coordenada_x_de_baixo = raio_de_baixo * sin ( angulo_de_baixo );
            coordenada_z_de_baixo = raio_de_baixo * cos ( angulo_de_baixo );

            escrever ponto (coordenada_x_de_baixo, stack_atual * altura_de_stack , coordenda_z_de_baixo) no ficheiro;

            //calcular segundo ponto
            coordenada_x_de_cima = raio_de_cima * sin (angulo_de_cima);
            coordenada_z_de_cima = raio_de_cima * cos ( angulo_de_cima);

            escrever ponto (coodenadas_x_de_cima, (stack_atual + 1) * altura_de_stack, coordenada_z_de_cima);

            avançar o angulo de baixo um step;

            //calcular o terceiro ponto
            coodenada_x_de_baixo = raio_de_baixo * sin ( angulo_de_baixo );
            coodenada_z_de_baixo = raio_de_baixo * cos ( angulo_de_baixo );

            escrever ponto (coordenada_x_de_baixo, stack_atual * altura_de_stack, coordenada_z_de_baixo);

            //segundo triangulo

            //primeiro ponto
            escrever ponto (coordenada_x_de_baixo, stack_atual * altura_de_stack, coordenada_z_de_baixo);
            //segundo ponto
            coordenada_x_de_cima = raio_de_cima * sin (angulo_de_cima);
            coordenada_z_de_cima = raio_de_cima * cos (angulo_de_cima);
            escrever o ponto (coordenada_x_de_cima, (stack_atual + 1 ) * altura_de_stack, coordenad_z_de_cima);

            avançar o angulo de cima um step;

            //terceiro ponto

            coordenada_x_de_cima = raio_de_cima * sin (angulo_de_cima);
            coordenada_z_de_cima = raio_de_cima * cos (angulo_de_cima);
            escrever o ponto (coordenada_x_de_cima, (stack_atual + 1 ) * altura_de_stack, coordenad_z_de_cima);

            raio_de_baixo = raio_de_cima;
            raio_de_cima = raio_de_cima - step_do_raio;
            angulo_de_cima = angulo_de_cima + step_do_angulo;
            angulo_de_baixo = angulo_de_baixo + step_do_angulo;
            stack_atual++;
        }

        angulo_de_cima = angulo_de_cima - step_do_angulo;
        angulo_de_baixo = angulo_de_baixo - step_do_angulo;
        raio_de_baixo = raio;
        raio_de_cima = raio - step_do_raio;
        slice_atual++;

    }
```

Figura 28: Pseudocódigo da função writeCone

```
    //desenhar a base do cone:
    slice_atual = 0;
    enquanto slice_atual < numero_de_slices{

        coordenada_x_de_baixo = raio * sin ( angulo_de_baixo );
        coordenada_z_de_baixo = raio * cos ( angulo_de_baixo );
        escrever ponto (coordenada_x_de_baixo, 0.0, coordenada_z_de_baixo) no ficheiro;

        anvançar um step no angulo de baixo;

        coordenada_x_de_baixo = raio * sin ( angulo_de_baixo );
        coordenada_z_de_baixo = raio * cos ( angulo_de_baixo );
        escrever ponto (coordenada_x_de_baixo, 0.0, coordenada_z_de_baixo) no ficheiro;
        escrever ponto ( 0.0, 0.0, 0.0) no ficheiro;
    }
```

Figura 29:Pseudocódigo da função writeCone

```
assignCoords(radius, currStack, currSlice, i_beta, d_beta, d_alpha){
        vertex v = {
        radius * cos(i_beta - (currStack * d_beta)) * sin(currSlice * d_alpha),
        radius * sin(i_beta - (currStack * d_beta)),
        radius * cos(i_beta - (currStack * d_beta)) * cos(currSlice * d_alpha),
        };
    return v;
}
```

Figura 30: Pseudocódigo da função assignCoords

```
sphereCoords(radius, number_of_slices, number_of_stacks, filename){
    displacement_alpha = (2*PI) / number_of_slices
    displacement_beta = (PI) / number_of_stacks
    initial_alpha = 0
    initial_beta = PI/2
    for current_stack = 0 to current_stack = number_of_stacks{
        for current_slice = 0 to current_slice = number_of_slices{

            //gerar as coordenadas para os 3 primeiros vértices do 1º triangulo
            vertex1 = assignCoords(radius, current_stack, current_slice, inicial_beta, displacement_beta, displacement_alpha)
            vertex2 = assignCoords(radius, current_stack+1, current_slice, inicial_beta, displacement_beta, displacement_alpha)
            vertex3 = assignCoords(radius, current_stack+1, current_slice+1, inicial_beta, displacement_beta, displacement_alpha)

            //gerar o últimoo vértice do segundo triângulo
            vertex4 = assignCoords(radius, current_stack, current_slice+1, inicial_beta, displacement_beta, displacement_alpha)

            //escrever cada coordenada de cada vértice de cada triângulo num ficheiro
            //separando cada coordenada com um ponto e vírgula (";")
            escrever vertex1 para o 1º triangulo
            escrever vertex1 para o 1º triangulo
            escrever vertex1 para o 1º triangulo
            escrever vertex1 para o 2º triangulo
            escrever vertex3 para o 2º triangulo
            escrever vertex4 para o 2º triangulo
            current_slice++
        }
        current_stack++
    }
}
```

Figura 31Pseudocódigo da função sphereCoords

```
void readXML(){

    Abrir ficheiro XML
    Carregar o ficheiro

    if( ficheiro carregado corretamente ){

        l_RootElement corresponde ao primeiro elemento do ficheiro com a tag "world"

        if( l_RootElement foi obtido com sucesso ){

            l_camera corresponde ao primeiro elemento obtido apartir de l_RootElement com a tag  "camera"

            if( l_camera foi obtida com sucesso ){

                l_position corresponde ao primeiro elemento obtido apartir de l_camera com a tag "position"

                colocar em x1 a coordenada x da posição da camera
                colocar em z1 a coordenada z da posição da camera
                colocar em t1 a coordenada y da posição da camera

                l_lookAt corresponde ao primeiro elemento obtido apartir de l_camera com a tag "lookAt"

                colocar em x1 a coordenada x da posição para onde a camera está a "olhar"
                colocar em y1 a coordenada y da posição para onde a camera está a "olhar"
                colocar em z1 a coordenada z da posição para onde a camera está a "olhar"

                l_up corresponde ao primeiro elemento obtido apartir de l_camera com a tag "up"

                colocar em x1 a coordenada x da do vetor up da camera
                colocar em y1 a coordenada y da do vetor up da camera
                colocar em z1 a coordenada z da do vetor up da camera

                l_proj corresponde ao primeiro elemento obtido apartir de l_camera com a tag "projection"

                colocar em fov o valor de fov no elemento l_proj
                colocar em near o valor de near no elemento l_proj
                colocar em far o valor de far no elemento l_proj

                l_group corresponde ao primeiro elemento obtido apartir de l_RootElement com a tag "group"

                if(l_group foi obtido com sucesso){

                    l_models corresponde ao primeiro elemento obtido apartir de l_group com a tag "models"

                    if( l_models foi obtido com sucesso ){

                        l_models corresponde ao primeiro elemento obtido apartir de l_models com a tag "model"

                        while( existem modelos ){

                            colocar em vertices o conjunto de vertices lidos do ficheiro 3d lido
                            avançar para o próximo modelo

                        }
                    }
                }
            }
        }
    }else{
        Informar de erro no seu carregamento
    }
}
```

Figura 32: Pseudocódigo da função readXML

```
parser (filename){
    abrir o ficheiro
    vector<char*> coordinates

    para cada linha no ficheiro{
        obter as coordenadas separadas por um ponto e vírgula (";")
        inserir cada coordenada no vetor coordinates
    }
    i = 0

    para cada 3 coordenadas no vetor{
        criar uma estrutura do tipo vertex
        coordenada x na estrutura vertex = vector[i]
        coordenada y na estrutura vertex = vector[i+1]
        coordenada z na estrutura vertex = vector[i+2]
        i = i + 3
        inserir a estrutura vertex no vetor vertices
    }
}
```

Figura 33: Pseudocódigo da função parser de ficheiros 3d