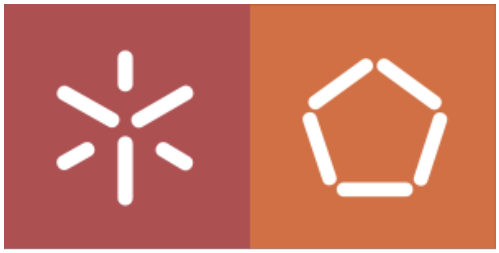

Universidade do Minho
Escola de Engenharia

Computação Gráfica

Fase 1

Daniel José Silva Furtado A97327
Ricardo Lopes Lucena A97746
Ricardo Silva Machado Araújo A96394
Nuno Miguel Leite da Costa A96897

A97327 A97746 A96394 A96897

Conteúdo

1 Introdução

No âmbito da unidade curricular de Computação Gráfica, foi-nos proposto desenvolver duas aplicações: um gerador capaz de gerar arquivos com informação dos modelos e um engine que lê arquivos de configuração escritos em XML e capaz de exibir modelos. Este projetos encontra-se divido em 4 fases, sendo nesta primeira fase pedido a construção de primitivas básicas: plano, box, esfera e cone, tal como a sua representação. Para a realização deste projeto recorremos à linguagem C++ e à bibloteca OpenGL.

2 Estrutura do Projeto

2.1 Aplicações

Nesta secção, serão discutidas as principais aplicações que permitem a criação e representação de diferentes modelos. Como referido anteriormente está dividida em duas aplicações distintas.

2.1.1 Generator

O módulo **generator.cpp** é responsável por gerar e converter todas as primitivas geométricas para conjuntos de vértices armazenados em ficheiros. Desta forma calcula todos os pontos necessários para formar diferentes triângulos que formarão a figura pretendida e grava-os no ficheiro desejado.

2.1.2 Engine

O módulo **engine.cpp** é responsável por interpretar e representar ficheiros *XML* que contêm referências a ficheiros criados pelo **Generator**. O ficheiro *XML* deverá conter todos os ficheiros com extensão *.3D* necessários para a construção das primitivas geométricas.

2.2 Classes

Com o objetivo de simplificar o processo de desenvolvimento das aplicações mencionadas anteriormente, optamos por criar diversas classes que funcionam como suporte para essa finalidade.

2.2.1 Ponto

A classe **Ponto** definida em **ponto.cpp** tem como objetivo traduzir a representação de um ponto num referencial para código. Desta forma, cada ponto é representado pelas coordenadas x, y e z que são três floats.

2.3 Ferramentas Utilizadas

Com o intuito de tornar a execução do trabalho prático mais fácil, optamos por utilizar algumas ferramentas auxiliares.

2.3.1 TinyXML2

Além da bibliotca OpenGL, que foi utilizada para implementar as funcionalidades gŕaficas do projeto, também empregamos a biblioteca TinyXML2, que facilita o processo de *parsing* do cenário utilizado.

3 Primitivas Gráficas

3.1 Plano

O Plano que era pretendido representar apresentava as especificações de que era contido no plano XZ, centrado na origem e subdividico nas direções X e Z. Para a construção deste plano consideramos inicialmente, que este seria visto de cima e com isso, utilizamos a regra da mão direita (em que o polegar fica direcionado para cima, e os pontos são desenhados no sentido em que a mão fechada ou no sentido contrário ao ponteiro dos relógios). Como é um plano contido no plano XZ, todos os vértices criados terão a sua coordenada Y com o valor de 0.

Para a elaboração desta primitiva, recorremos a dois *for's* em que o mais externo percorria na vertical(representado pelo i) enquanto que o *for* interior percorria na horizontal(representado por j). Utilizamos duas variáveis temporárias que são **xtemp** e **ztemp**, que representam o valor que o **X** e o **Z** terão na iteração seguinte.

Deste modo, cada triângulo que se encontra em cada divisão do plano terá as seguintes coordenadas:

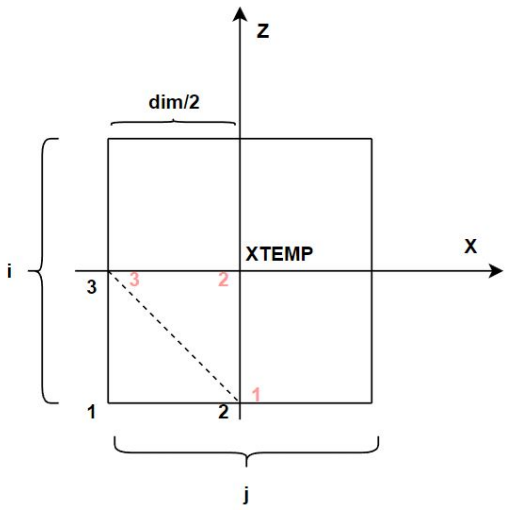

Figura 1: Esboço da construção do Plano

3.2 Box

Para formar a box, o nosso grupo percebeu que poderia utilizar o mesmo raciocínio utilizado no plano. Deste modo, a box é constituída por 6 fases, todas igualmente com o mesmo número de divisões passado como argumento, assim para a construção recorremos a **3 etapas**, cada uma com 2 *for's* e que funcionam da mesma maneira, sendo que cada etapa diz respeito às 2 faces paralelas aos plano XZ,XY,YZ.

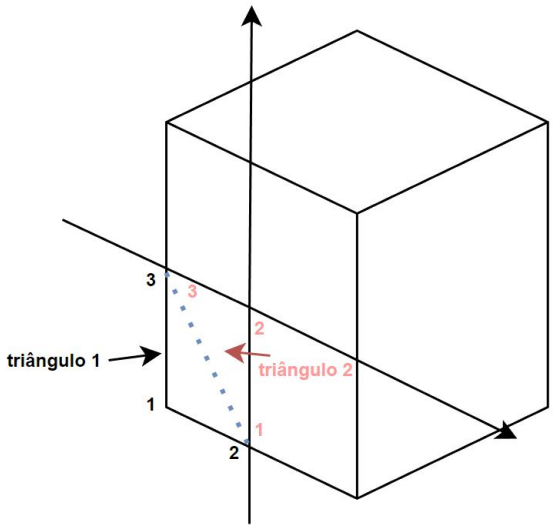

Figura 2: Esboço da construção da Box

Na figura 2, é possível observar como formamos uma face com divisões da box. Em cada etapa, começamos pela divisão que se encontra no canto inferior esquerdo, em que desenhamos o triângulo 1 e o triângulo 2 por esta ordem (a ordem de construção dos pontos de cada triângulo encontra-se na figura). Após desenhar uma divisão, passamos para a divisão que encontra-se à direita. Quando acabarmos de formar uma horizontal, iniciamos a construção das divisões que encontram-se mais em cima, repetindo o processo descrito em cima.

3.3 Esfera

Para criar os triângulos que juntos formam uma esfera, é preciso dividir a esfera em *slices* e em *stacks*. Além disso, é necessário definir um raio para a esfera. Para representar cada um dos vértices do triângulo, é necessário definir as suas coordenadas esféricas, que são determinadas pelas seguintes equações:

- $x = raio \cdot \cos(\beta) \cdot \sin(\alpha)$
- $y = raio \cdot \sin(\beta)$
- $z = raio \cdot \cos(\beta) \cdot \cos(\alpha)$

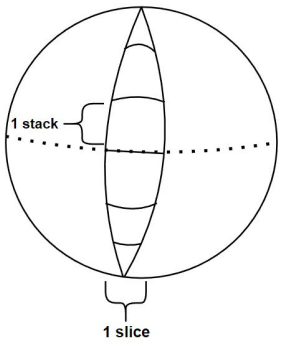

Figura 3: Esboço da construção da Sphere

Cada ponto dos triângulos é determinado por diferentes combinações dos ângulos α e β. Para mudar a camada em que estamos a construir os triângulos, ajustamos o valor de β. Já para variar de *slice*, modificamos o valor de α.

A estratégia uilizada para formar a esfera partiu da utilização de *for's*. Recorremos a dois *for's*, sendo o *for* mais exterior responsável por percorrer as *slices* e o *for* mais interior por percorrer as *stacks*. Ao percorrer as *stacks*, verificávamos qual *stack* estávamos a formar, caso fosse a primeira *stack* estaríamos a formar o triângulo que corresponde à *stack* mais inferior, caso fosse à ultima *stack* estaríamos a formar o triângulo que corresponde à mais superior, nos outros casos estavamos a formar as intermédias.

3.4 Cone

Para construir um cone, precisamos das seguintes informações: o raio da sua base, a sua altura, o número de *slices* e o número de *stacks*. A forma como é construído o cone é semelhante à forma de como é construído o cilindro (construção que abordamos nas aulas práticas), deste modo a abordagem ao nível das *stacks* envolve trabalhar mais com raios e ângulos, enquanto que ao nível de *slices* permance idêntico. Tal como na esfera, para definir cada um dos vértices dos triângulos recorremos às seguintes coordenadas esféricas:

- $x = raio \cdot \cos(\beta) \cdot \sin(\alpha)$
- $y = raio \cdot \sin(\beta)$
- $z = raio \cdot \cos(\beta) \cdot \cos(\alpha)$

Algumas informações importantes são:

- O ângulo default de cada *slice* é (2 M P I)/número de *slices*;
- Cada *stack* possui uma altura de: altura do cone/número de *stack*;
- A modificação do raio da *stack* seguinte efetua-se com base no raio/número de *stacks*;

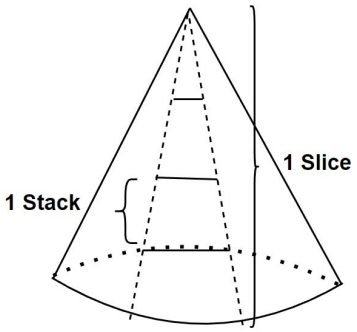

Figura 4: Esboço da construção do Cone

A estratégia utilizada para formar o cone consiste na utilização de dois *for's*. Ambos os *for's* são utilizados para percorrer as *stacks* e as *slices*, sendo o *for* mais externo utilizado para percorrer as *stacks* e o *for* mais interno utilizado para percorrer as *slices*. Deste modo, à medida que percorremos as *slices* vamos construíndo as *stacks* e a base do cone.

4 Generator

O papel do Generator é criar todas as primitivas solicitadas no trabalho prático. Ele realiza isso ao calcular todos os vértices necessários para formar os diferentes triângulos que são a base da construção das primitivas. De seguida, o Generator armazena os pontos criados em um arquivo específico solicitado pelo utilizador.

4.1 Demonstração

Para criar uma forma geométrica específica ao usar o Generator, é necessário fornecer como entrada o nome da forma acompanhado das suas dimensões e divisões (nas que necessitam), bem como o nome do arquivo no qual se deseja guardar o resultado. Isso permite que o **Generator** calcule os vértices necessários e armazene-os no arquivo especificado para que a forma geométrica possa ser exibida posteriormente. É importante fornecer as informações corretas de entrada para garantir que a forma geométrica criada corresponda às especificações desejadas. A maneira como cada primitiva é construída no generator, é a seguinte:

- Para o Plano

  `plane [DIMENSION] [DIVISIONS ][OUTPUT FILE]`

- Para a Box

  `box [DIMENSION] [DIVISIONS] [OUTPUT FILE]`

- Para a Sphere

  `sphere [RADIUS] [SLICES] [STACKS] [OUTPUT FILE]`

- Para o Cone

  `cone [RADIUS] [HEIGHT] [SLICES] [STACKS] [OUTPUT FILE]`

```
/CG-Project$ ./generator plane 3 4 plane.3d
```

Figura 5: Generator de um Plano

```
96
-1.500000, 0.000000, -1.500000
-0.750000, 0.000000, -1.500000
-1.500000, 0.000000, -0.750000
-0.750000, 0.000000, -1.500000
-0.750000, 0.000000, -0.750000
-1.500000, 0.000000, -0.750000
-0.750000, 0.000000, -1.500000
```

Figura 6: Alguns dosvértices do Plano

5 Engine

Como descrito no enunciado do trabalho, o Engine ou Motor, é responsável por interpretar e representar o ficheiro XML que lhe é passado como argumento, tanto como, representar graficamente as diferentes primitivas geométricas incluidas no ficheiro XML, que provém do Generator.

Exemplo de um ficheiro XML:

```xml
<world>
    <window width="512" height="512" />
    <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" />
            <model file="cone.3d" />
        </models>
    </group>
</world>
```

Exemplo de um possível input:

```
/CG-Project$ ./engine example.xml
```

6 Modelos 3d

6.1 Plano

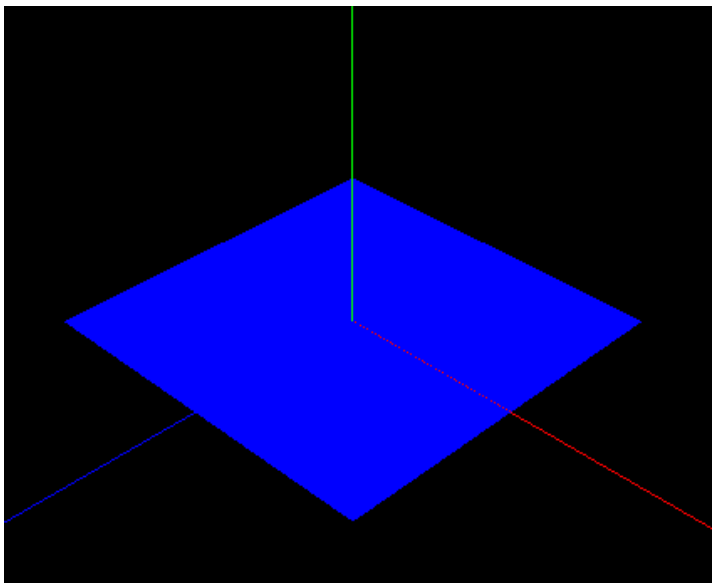

Figura 7: Construção do Plano Preenchido

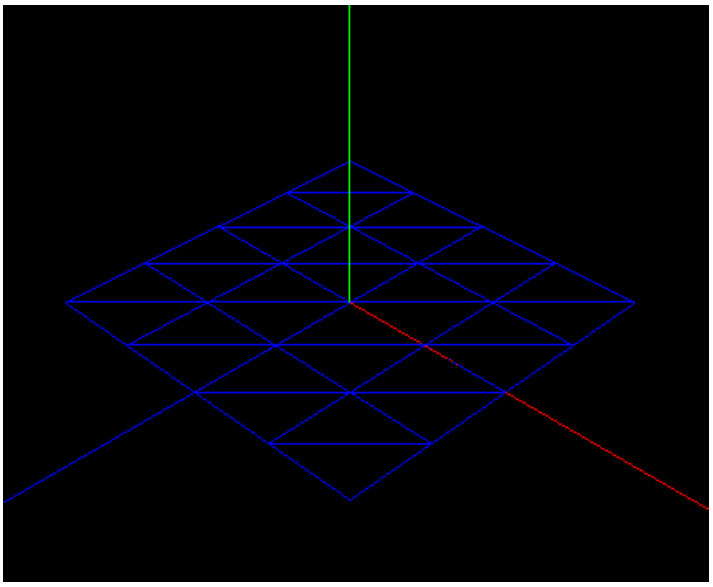

Figura 8: Construção do Plano Arestas

6.2 Box

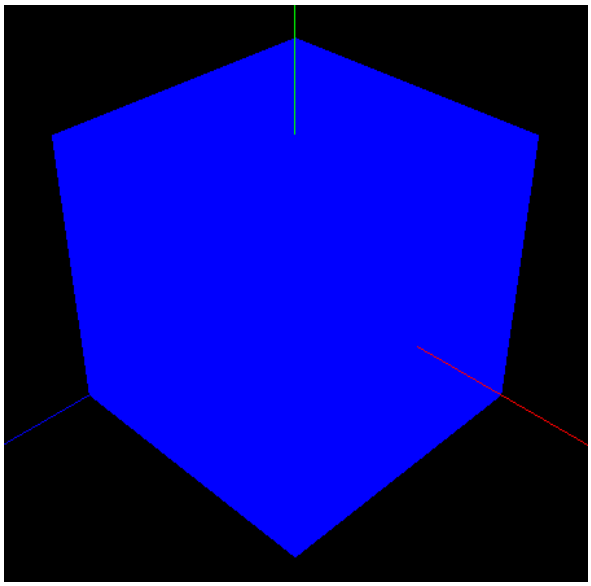

Figura 9: Construção da Box Preenchida

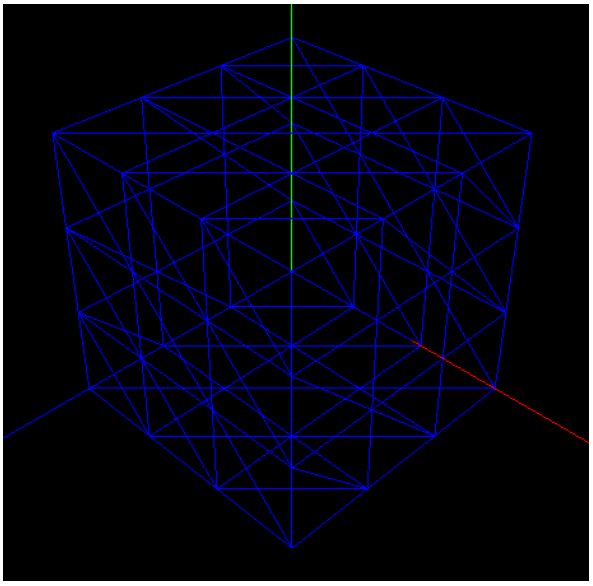

Figura 10: Construção da Box Arestas

6.3 Sphere

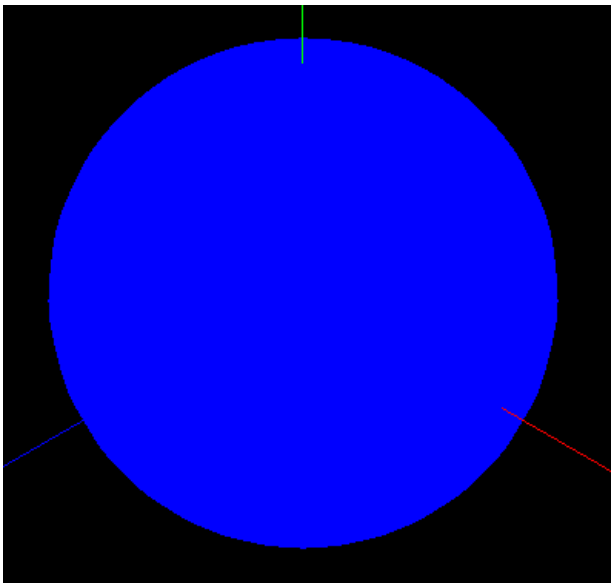

Figura 11: Construção da Sphere Preenchida

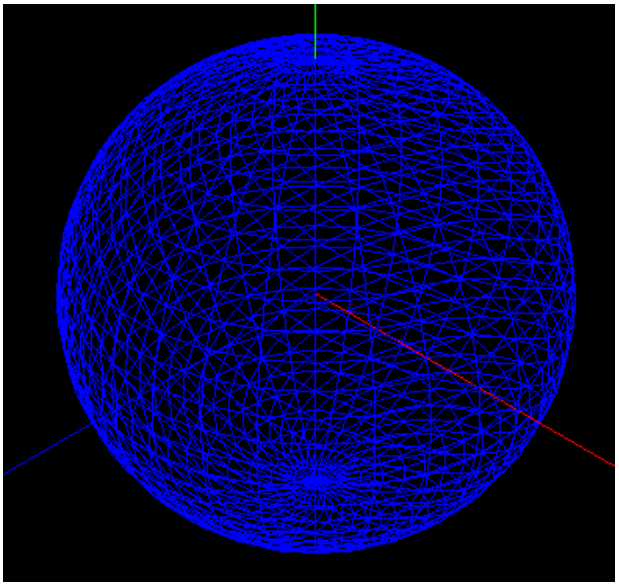

Figura 12: Construção da Sphere Arestas

6.4 Cone

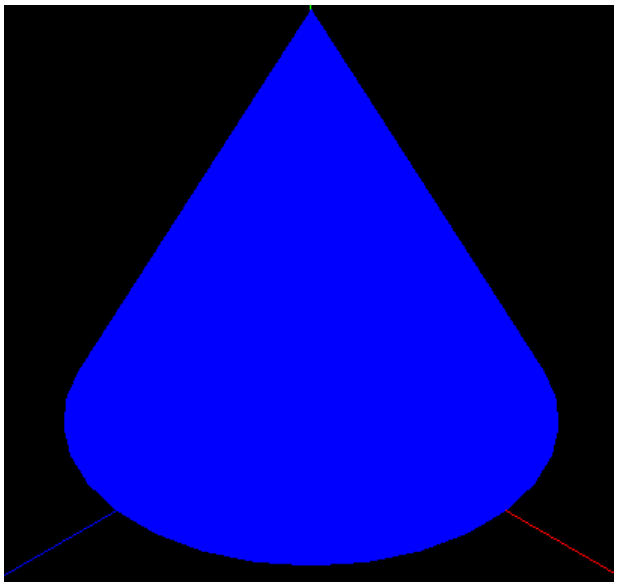

Figura 13: Construção do Cone Preenchido

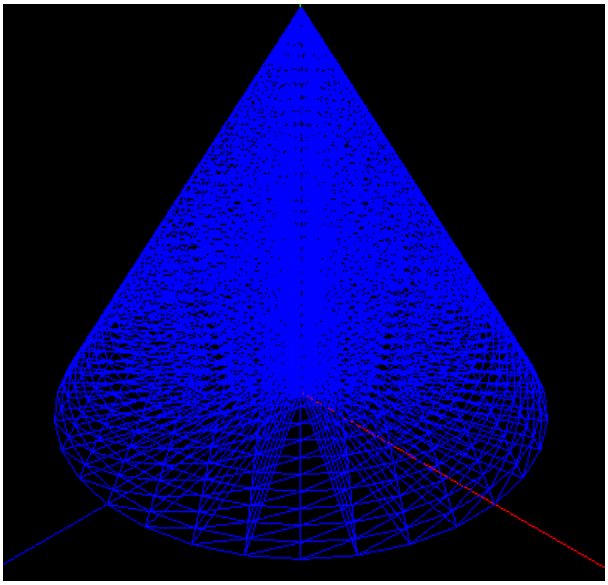

Figura 14: Construção do Cone Arestas

7 Extras

Além do que foi solicitado, acrescentamos a capacidade de interagir através do teclado (tal como foi abordado nas aulas práticas e através da função *myKeyboardFunc*), bem como a adição de uma camêra 3D.

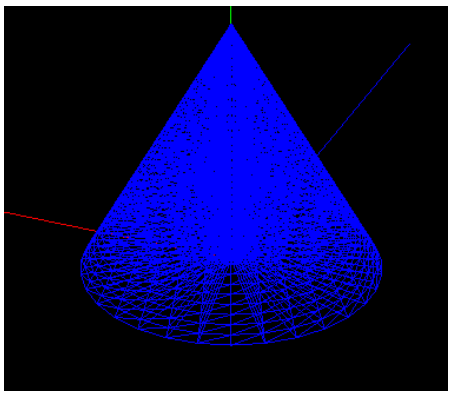

Figura 15: Exemplo da camêra 3d

8 Conclusão

A primeira fase do trabalho prático teve uma grande importância, uma vez que nos proporcionou a oportunidade de trabalhar com ferramentas e conceitos relacionados à disciplina de Computação Gráfica e permitiu que adquiríssimos experiência em áreas como o uso do GLUT e OpenGL.

Além disso, como todo o trabalho foi desenvolvido utilizando a linguagem C++, a qual nenhum dos membros do grupo havia tido contato anteriormente, tivemos a oportunidade de aumentar o nosso conhecimento em linguagens de programação bem como a sua utilização.

Esperamos que esta primeira fase sirva como uma base sólida para as próximas etapas do trabalho prático. O trabalho realizado até o momento foi de extrema importância para familiarizarmo-nos com a implementação dos conceitos aprendidos nas aulas e com as ferramentas utilizadas para este fim.

Em resumo, a primeira fase permitiu-nos acumular uma valiosa experiência em relação à utilização de ferramentas e conceitos da Computação Gráfica, empregando OpenGL e GLUT, bem como aprender uma nova linguagem de programação, C++, que foi muito útil e amigável durante todo o processo.