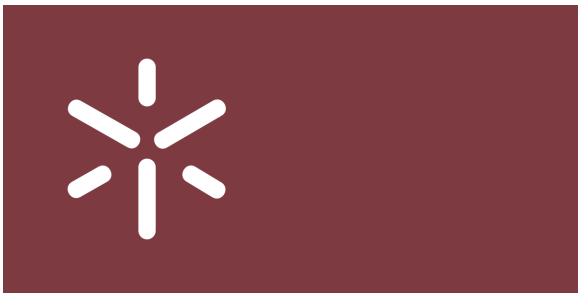

Universidade do Minho

Fase 1 - Primitivas Gráficas

Trabalho Prático de Computação Gráfica

Mestrado Integrado em Engenharia Informática

João Amorim A74806

10 de Março de 2021

1 Introdução

No âmbito da Unidade Curricular de Computação Gráfica, foi proposto que se desenvolvesse um cenário baseado em gráficos 3D. Este projeto consiste em 4 fases, sendo o objetivo desta primeira a criação de um mecanismo de geração de primitivas gráficas, nomeadamente o plano, a caixa, a esfera, o cone e adicionalmente o cilindro.

Nesta primeira fase era também necessária a criação de um motor que permitisse efetuar a leitura de ficheiros de configuração em XML assim como a leitura dos ficheiros 3d, que contêm os vértices das primitivas gráficas e que foram obtidos a partir dos ficheiros XML, para que este mesmo motor consiga desenhar as já mencionadas primitivas gráficas. Adicionalmente, foram definidas todas as ferramentas necessárias para que exista uma interação, por parte do utilizador, com as primitivas geradas, nomeadamente com o teclado e o rato.

Este relatório tem como objetivo descrever toda a aplicação, ajudando a executar a mesma, contendo obviamente uma descrição pormenorizada das primitivas gráficas que foram geradas ao longo do trabalho.

2 Arquitectura da Solução

Tal como era pedido, este projeto é dividido em duas aplicações principais: o gerador e o motor.

2.1 Gerador

É aqui no Gerador que estão presentes os algoritmos para a geração das diferentes primitivas gráficas. Para cada primitiva gráfica, é implementado o algoritmo que a descreve e consoante os pontos que são obtidos, estes vão sendo gravados no seu respetivo ficheiro 3d.

Na main são recebidos os parâmetros usados na invocação das diversas funções, funções estas que darão origem à geração das primitivas. O gerador deve ser executado com os seguintes comandos:

```
\CG\CG-TP1\Generator>g++ generator.cpp -o gen

\CG\CG-TP1\Generator>gen plane 5 plane.3d
```

Neste caso, o primeiro cria o executável da aplicação para o gerador e o segundo cria um ficheiro chamado *plane.3d* que contém no seu interior as coordenadas x, y e z de todos os pontos que constituem a primitiva gráfica gerada, neste caso um plano de lados 5. É também possível no gerador a execução do comando "gen info" que imprimirá no ecrã instruções sobre os dados necessários para a criação de todas as primitivas, entre outros.

```
Geracao das Primitivas:

    $gen plane lado [file_name.3d]
    $gen box comp largura alt n_camadas[file_name.3d]
    $gen cone raio alt n_fatias n_camadas [file_name.3d]
    $gen sphere raio n_fatias n_camadas [file_name.3d]
    $gen cylinder raio alt n_fatias n_camadas [file_name.3d]
```

Figure 1: Comando info com comandos para a geração de primitivas

2.2 Engine

O Engine tem como objetivo a leitura dos ficheiros *XML* que contêm o nome dos ficheiros *3d* a serem lidos , fazendo o parsing dos mesmos com a utilização do parser *tinyxml2*. Depois de ser efetuado o parsing, o Engine tem a função de de ler o/os ficheiros *3d* que contêm os vértices necessários ao desenho das primitvas gráficas, armazenando-os num vetor de *Vertice* chamado *vertices*. Este *Vertice* corresponde a um ponto de coordenadas x, y, z, que tal como o nome indica, corresponde a um vértice da primitiva gráfica. Para a implementação deste Engine, foi utilizado o esqueleto utilizado nas aulas práticas contendo diretrizes obrigatórias do *GLUT*, tendo sido adicionadas todas as variáveis e funcionalidades que achei necessárias para a resolução dos objetivos.

É importante mencionar que aqui tive problemas com o include do glut.h sempre que tentava criar o executável do Engine, algo que terá de ser revisto com os Professores para a próxima fase. A execução do Engine foi efetuada a partir do *Visual Studio*, onde são passados na *main*, como parâmetros da função *parser* os nomes dos ficheiros XML previamente gerados manualmente, dando inicio assim ao parsing dos ficheiros XML e posterior desenho das primitivas gráficas. Tal como já mencionado, os ficheiros XML já se encontram definidos, assim como um ficheiro 3d para cada uma das primitivas gráficas, com parâmetros escolhidos aleatoriamente, para efeitos de demonstração. Foi também criado um ficheiro XML para a leitura de um ficheiro AllXML.3d que desenha todas as primitivas criadas.

```
<scene>
    <model file="plane.3d" />
    <model file="box.3d" />
    <model file="sphere.3d" />
    <model file="cone.3d" />
    <model file="cylinder.3d" />
</scene>
```

Figure 2: Ficheiro AllXML.xml

Quando executado o Engine, se não forem passados quaisquer argumentos, será feito o parsing do ficheiro AllXML, sendo apresentadas todas as primitivas no ecrâ. Para uma primitiva em específico, basta passar como primeiro argumento o nome do ficheiro XML correspondente à primitiva que se pretenda observar.

De seguida apresentam-se os comandos que podem ser utilizados para interagir com as primitivas no motor:

```
Rotacao da camara -> Arrow Keys (UP, DOWN, LEFT, RIGHT)
Zoom In/Out -> + / -  or PAGE UP / PAGE DOWN
Representacao do solido:
     GL_LINE -> l | L
     GL_FILL -> f | F
     GL_POINT -> p | P
     Menu -> Botao Direito do Rato
Inicio/Reset -> i / I
```

Figure 3: Comandos para Interação com as Primitivas

3 Primitivas Gráficas

Para ir de encontro ao que é pedido no enunciado, as primitivas criadas recorrem às posições relativas dos vértices dos vários modelos. É importante mencionar que para o desenho à custa de vértices em OpenGL, é necessário fazer as divisões em triângulos. Nas imagens de cada primitiva, será apresentado o eixo XZY, em que o eixo X corresponde ao vermelho, o Z ao azul e o Y ao verde.As imagens de cada triângulo, de cada primitiva, são geradas aleatoriamente.

3.1 Plano

Um plano é constituído por dois triângulos que partilham dois pontos entre si. Tal como pedido, o plano está contido no plano XZ e centrado na origem.

3.1.1 Algoritmo

Os triângulos que compõem o plano, para poderem ser observados, têm de ser desenhados por uma certa ordem. Neste caso, como queremos observar a face voltada para cima, o desenho dos pontos que constituem os triângulos deve-se dar no sentido oposto ao dos ponteiros do relógio. Ao mesmo tempo, é necessário ter em atenção que foi requisitado que o plano se encontrasse centrado na origem. Desta maneira, as coordenadas de cada ponto correspondem a metade do valor do lado do plano. Por exemplo, se quisermos gerar um plano de dimensões 2, os pontos gerados devem ser (1,0,-1), (-1,0,-1), (-1,0,1) para o triângulo da esquerda e (1,0,1), (1,0,-1), (-1,0,1) para o triângulo da direita.

3.1.2 Modelo 3D

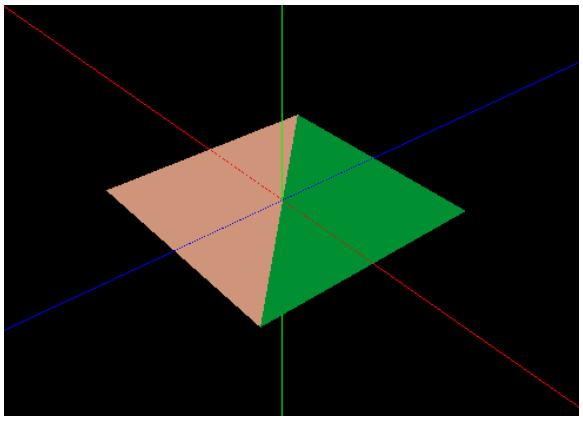

Figure 4: Modelo 3D do Plano obtido com 5 de lado

3.2 Caixa

Uma caixa considera-se um prisma de seis faces, onde temos que ter em consideração a existência de um o comprimento x, a largura z, a altura y, assim como o número de divisões.

3.2.1 Algoritmo

O desenvolvimento das faces da caixa segue a mesma teoria do plano, no entanto, como são implementadas camadas, é preciso um algoritmo mais complexo. Assim, desta vez, para a geração dos pontos dos triângulos, temos que considerar o número de camadas. Posto isto, o espaçamento entre dois pontos pertencentes ao eixo X é dado pela divisão do comprimento pelo número de camadas. Para o espaçamento entre dois pontos do eixo do Z e Y foi substituído o comprimento pela largura e altura, respetivamente. Considerando a face voltada para a frente, onde o valor de z se mantém sempre constante em todos os pontos, é possível obter a mesma face que, anteriormente era composta por dois triângulos, agora é composta por várias divisões. Assim, o desenho dos triângulos faz-se da esquerda para a direita, sendo as coordenadas dos vértices calculadas pelo espaçamento e, chegando ao fim da linha incrementa-se a altura. Assim, foi desenvolvida uma fórmula para a aplicação do algoritmo, sendo ambos i (variável que permite a incrementação da altura) e j (variável para percorrer eixo do x) menores que o

número de camadas:

1. (xx + (espC * j), yy + (espA * i), z)
2. (xx + (espC * j) + espC, yy + (espA * i), z)
3. (xx + (espC * j) + espC, yy + (espA * i) + espA, z)
4. (xx + (espC * j) + espC, yy + (espA * i) + espA, z)
5. (xx + (espC * j), yy + (espA * i) + espA, z)
6. (xx + (espC * j), yy + (espA * i), z)

Os pontos 1,2,3 formam um triângulo e 4,5,6 formam outro, ambos da face da frente, sendo espC o espaçamento entre os pontos pertencentes ao eixo X e espA o espaçamento entre os pontos do eixo Y. Para as restantes faces, o raciocínio é o mesmo, tendo apenas em atenção qual a variável que terá sempre o mesmo valor e quais as que têm que ser alteradas.

3.2.2 Modelo 3D

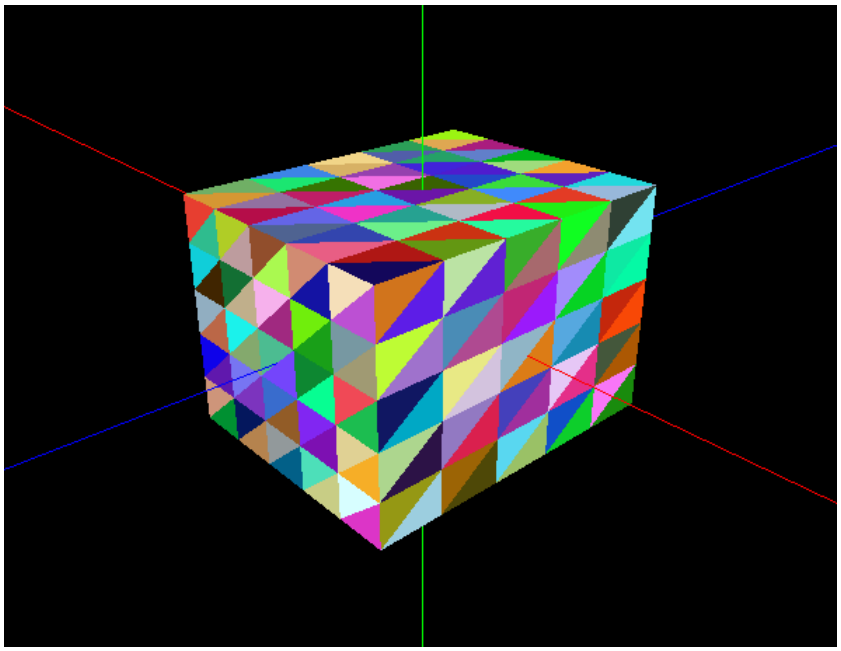

Figure 5: Modelo 3D da Caixa obtida com 8x10x7 e 5 divisões.

3.3 Esfera

Uma esfera é um sólido geométrico que é formado por uma superfície curva em que os pontos são equidistantes do centro. Para se construir uma esfera os seguintes parâmetros têm que ser inseridos, raio (r), número de fatias cv e número de camadas ch. Quanto maior forem os valores de fatias e número de camadas, melhor será a curvatura resultante.

3.3.1 Algoritmo

Para fazer uma esfera é importante estabelecer algumas variáveis importantes.

- Espaçamento entre fatias (espV) = 2* / cv.
- Espaçamento entre camadas (espH) = / ch.

Após o cálculo destas variáveis podemos começar a desenhar a figura. Para isso utilizamos fórmulas para descobrir o valor das variáveis X,Y e Z nos pontos pertencentes à superfície da esfera. Essas fórmulas utilizam os valores dos ângulos alpha e beta que são respectivamente, o ângulo no plano horizontal XZ e o ângulo no plano vertical YZ. Aplicando as fórmulas para descobrir quais os quatro pontos pertencentes a um trapézio criado na superfície da esfera conseguimos desenhar os dois triângulos que fazem esse trapézio:

```
x1 = r * sin(angV) * sin(angH);
y1 = r * cos(angH);
z1 = r * sin(angH) * cos(angV);

x2 = r * sin(angH) * sin(angV + espV);
y2 = r * cos(angH);
z2 = r * sin(angH) * cos(angV + espV);

x3 = r * sin(angH + espH) * sin(angV + espV);
y3 = r * cos(angH + espH);
z3 = r * sin(angH + espH) * cos(angV + espV);

x4 = r * sin(angH + espH) * sin(angV);
y4 = r * cos(angH + espH);
z4 = r * sin(angH + espH) * cos(angV);
```

Para desenhar toda a esfera, existem dois ciclos, em que o primeiro percorre o número de camadas e o segundo o número de fatias, utilizando o algoritmo descrito.

3.3.2 Modelo 3D

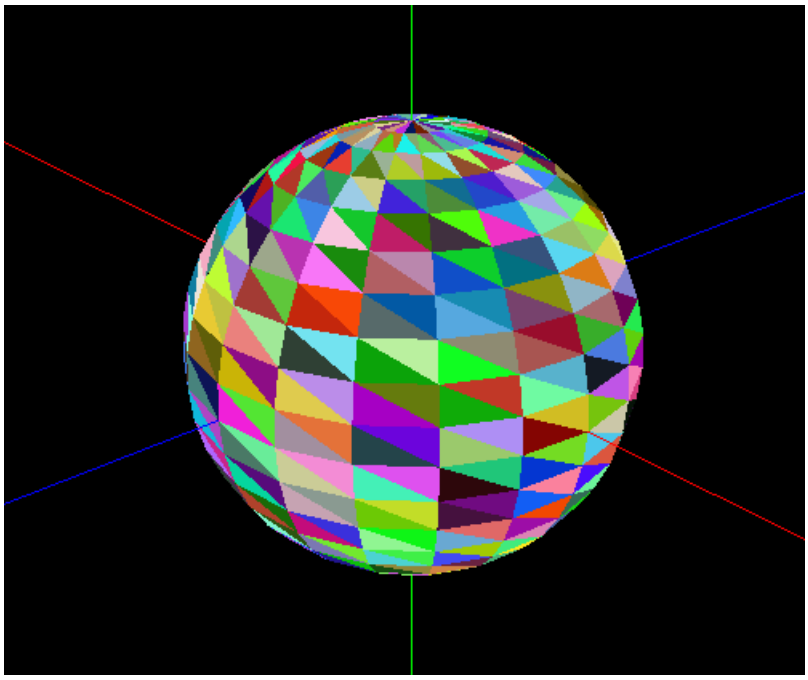

Figure 6: Modelo 3D da Esfera com raio 5, 20 slices e 20 camadas

3.4 Cone

O cone é um sólido geométrico que é obtido através de uma pirâmide com uma circunferência como base, logo, quanto maior for o número de fatias melhor será a curvatura do cone. E preciso inserir os seguintes parâmetros quando se quer fazer um cone, sendo eles, um raio (r), uma altura (a), um número de fatias (slices) e um número de camadas horizontais (cH).

3.4.1 Algoritmo

Tal como na esfera, no cone também é preciso definir algumas variáveis importantes primeiro:

- Espaçamento entre fatias (espS) : 2*pi / slices;
- Espaçamento entre camadas (espH) : a/cH;
- Altura da base do cone (alt) : -(a/2);

A construção do cone tem duas fases. Numa primeira fase é feita a circunferência, ou seja, a base do triângulo. Para tal, utilizamos as seguintes fórmulas que utilizam um ângulo (alpha) no plano horizontal XZ:

```
x1 = 0;
y1 = alt;
z1 = 0;

x2 = r * sin(ang + espS);
y2 = alt;
z2 = r * cos(ang + espS);

x3 = r * sin(ang);
y3 = alt;
z3 = r * cos(ang);
```

Utilizando as fórmulas acima, conseguimos descobrir os pontos para construir os triângulos correspondentes à base. Um dos pontos em todos os triângulos da circunferência é o ponto (0,-(a/2),0) que corresponde ao centro da mesma.

Com recurso a um ciclo que corre o mesmo número de vezes quantas slices existem, fica assim desenhada a base.

Posteriomente, temos a fase de construção da parte de cima do cone em cH camadas horizontais. Para tal é feito um ciclo que calcula algumas variáveis em cada iteração que permitem obter informaçõoes sobre a camada em que se está a trabalhar. Esse ciclo é inicializado com um inteiro i que começa em zero e em cada iteração soma um.

```
- Camada de baixo (camadaB) : alt + (i*espH);
```

```
- Camada de acima (camadaA) : alt + ((i+1) * espH);
- Raio da camada de baixo (raioB) : ((r/cH)*i);
- Raio da camada de cima (raioA) : r - ((r/cH) * (i+1));
```

Depois de saber estes valores, conseguimos aplicar um ciclo dentro do ciclo em que estamos, que utiliza as fórmulas que são utilizadas na fase de criação da base para descobrir os pontos do trapézio na superfície do cone. Com isto, é possível criar os dois triângulos que compõe o trapézio.

3.4.2 Modelo 3D

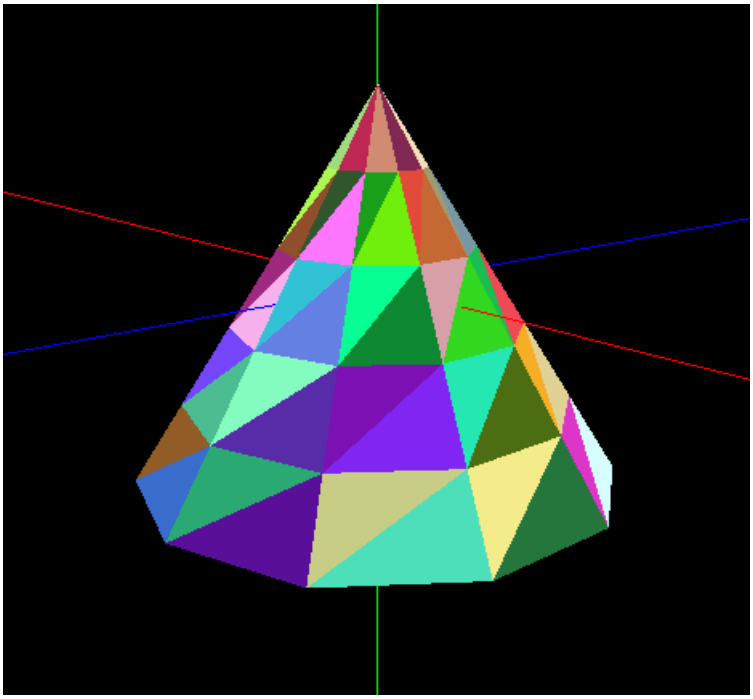

Figure 7: Modelo 3D do Cone com 5 de raio, 8 de altura, 10 slices e 5 camadas

3.5 Cilindro

O cilindro é um sólido geométrico que é constituído por duas bases, sendo elas circunferências. Para construir um cilindro é necessário inserir como parâmetro um (raio), uma (altura), o número de camadas verticais

(slices) e o número de camadas horizontais (slicesH). É de salientar, que quanto maior for o valor de slices melhor será a curvatura do cilindro.

3.5.1 Algoritmo

Tal como no cone, é necessário calcular algumas variáveis importantes.

- Espaçamento entre fatias (espS) : 2*pi / slices;
- Espaçamento entre camadas (espH) : altura / slicesH;
- Altura da base de baixo do cilindro (alt) : -(altura/2);

O cilindro é bastante semelhante ao cone, na medida em que para a construção das faces de cima e de baixo, são utilizadas as fórmulas que se utilizam na construção da base do cone, incluindo um ângulo (alpha). A segunda fase da construção do cilindro diz respeito à construção da superfície lateral. Para isso, é utilizado um ciclo que permite a construção da superfície entre camadas horizontais. Os pontos que fazem essa superfície são descobertos noutro ciclo que utiliza as seguintes fórmulas:

```
x1 = raio * sin(ang);
y1 = altCamada + espSH;
z1 = raio * cos(ang);

x2 = raio * sin(ang);
y2 = altCamada;
z2 = raio * cos(ang);

x3 = raio * sin(ang + espS);
y3 = altCamada;
z3 = raio * cos(ang + espS);

x4 = raio * sin(ang + espS);
y4 = altCamada + espSH;
z4 = raio * cos(ang + espS);
```

Utilizando as fórmulas conseguimos descobrir os pontos que formam um quadrado na superfície lateral. Posteriormente, construindo dois triângulos conseguimos representar esse quadrado em cada iteração do ciclo.

3.5.2 Modelo 3D

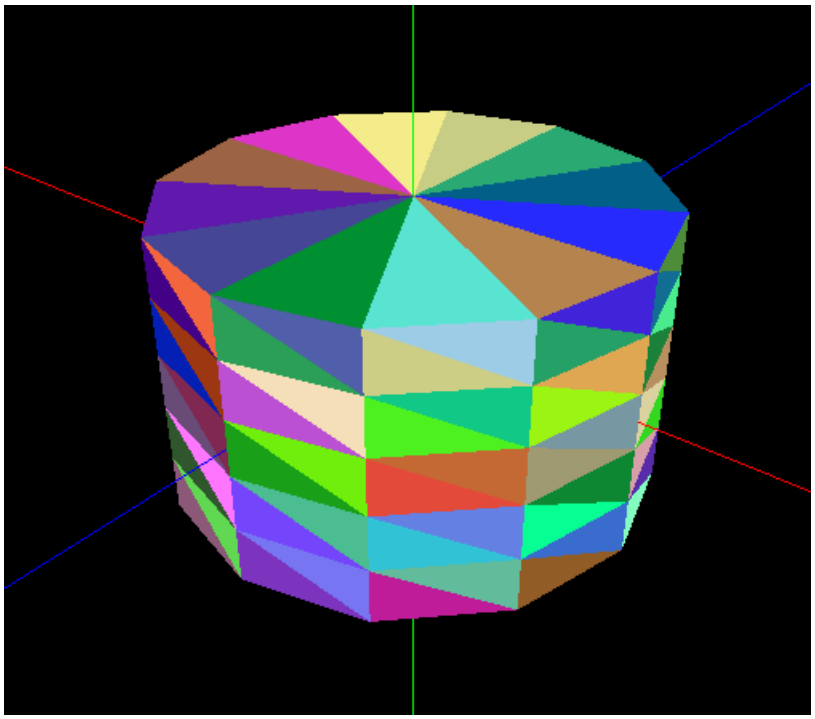

Figure 8: Modelo 3D do Cilindro com 6 de raio, 6 de altura, 12 slices e 5 camadas.

Com a apresentação do cilindro, fica assim concluída a apresentação das primitivas gráficas criadas nesta fase do trabalho prático.

4 Conclusão

Em suma, posso dizer que estou satisfeito com o trabalho realizado. Esta primeira parte permitiu-me consolidar o conhecimento na utilização de ferramentas associadas à Computação Gráfica como *OpenGL* e *GLUT*, da utilização da linguagem C++ assim como adquirir uma melhor noção dos algoritmos associados à construção de primitivas gráficas, tais como as que estão presentes neste trabalho, tendo até sido adicionadas mais primitivas além das que eram requisitadas.

Sendo estes, na minha opinião, os principais objectivos desta fase do trabalho prático, diria que o balanço final do conhecimento adquirido é mais do que positivo. Um aspeto a melhorar nesta 1ª fase seria a implementação de uma câmara FPS, que tal como os Professores já mencionaram, será bastante útil mais para a frente.