Universidade do Minho

Escola de Engenharia

Computação Gráfica

Licenciatura em Engenharia Informática

Fase 1 - Graphical Primitives

Grupo 12

Alexandre Fernandes - [A94154]
Ana Pires - [A96060]
Henrique Vaz - [A95533]
Mariana Marques - [A93198]

Março, 2023

Conteúdo

1. Introdução

No âmbito da UC de Computação Gráfica, é proposto o desenvolvimento de um projeto que seja capaz de gerar um cenário gráfico baseado num motor 3D, que é, posteriormente, dividido em fases para facilitar todo o processo.

Como o próprio título desta fase indica, (*Graphical Primitives*), é proposto o desenvolvimento de um projeto que permita gerar primitivas gráficas, tais como, **Plano**, **Caixa**, **Esfera** e **Cone**, através do **Generator** e o **Engine**.

Em relação à implementação, foi escolhida como linguagem de programação C++ e como ferramenta auxiliar gráfica o OpenGL.

2. Estrutura

2.1 Generator

O primeiro passo do desenvolvimento do modelo, foi *generator*. Este tem como função gerar os ficheiros *.3d*, ficheiros estes que contêm os vértices da primitiva pretendida, de acordo com os argumentos dados.

2.2 Engine

O engine tem como função apresentar a visualização das primitivas. Para tal ser possível, este recebe como argumento o ficheiro XML em questão, que contém a configuração da câmara e os ficheiros gerados pelo *Generator*, que contêm os vértices das figuras.

3. Primitivas Gráficas

3.1 Plano

De forma a ser possível gerar o plano, são então necessários 2 parâmetros:

- **Lado** - O comprimento dos lados do plano que se pretende gerar;
- **Subdivisões** - O número de subdivisões feitos ao longo de cada eixo.

Para calcular os pontos do plano considera-se que cada sub-divisão do plano corresponde a dois triângulos, e através de dois ciclos aninhados, calcula-se os pontos dos respetivos triângulos.

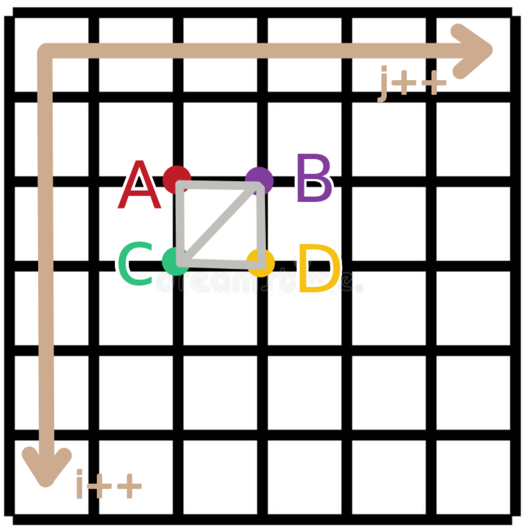

Figura 3.1: Representação gráfica dos triângulos numa sub-divisão

Desta forma, é necessário calcular as coordenadas dos pontos, onde i e j delimitam a região do plano em questão, *size* representa o comprimento do lado de cada sub-divisão e *h_size* representa o *offset* necessário para o plano ficar centrado em relação aos eixos:

- **Ponto A**
  x = -h_size + (size * j);
  y = 0;
  z = h_size - (size * i);
- **Ponto B**
  x = -h_size + (size * j);
  y = 0;
  z = h_size - (size * (i+1));

- **Ponto C**
  x = -h_size + (size * (j+1));
  y = 0;
  z = h_size - (size * i);

- **Ponto D**
  x = -h_size + (size * (j+1));
  y = 0;
  z = h_size - (size * (i+1));

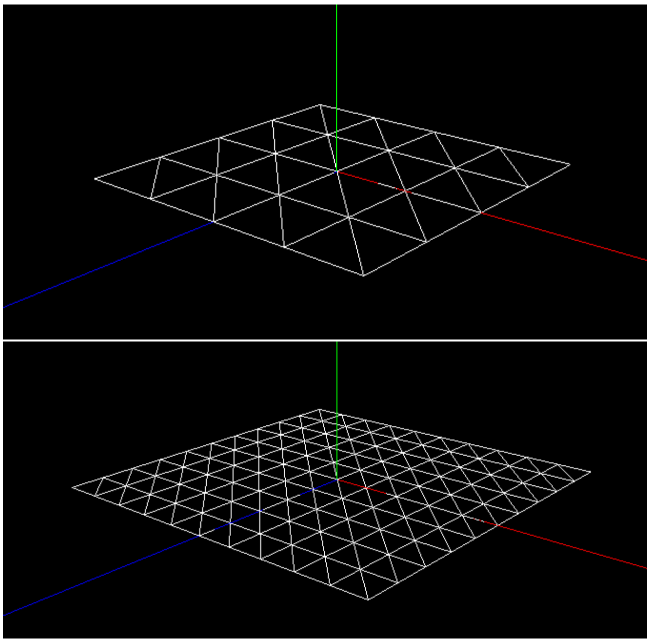

Figura 3.2: Representação de um Plano - Comprimento 1 e 4 divisões por eixo vs. Comprimento 2 e 10 divisões por eixo

Nota: Devido à falta de especificação, o grupo decidiu que o plano seria visível tanto por cima como por baixo.

3.2 Caixa

Para gerar esta primitiva, são requeridos 2 parâmetros:

- **Lado** - O comprimento dos lados da caixa que se pretende gerar;
- **Subdivisões** - O número de subdivisões feitos para cada lado.

À semelhança do que acontece no plano, os pontos da caixa são calculados tendo em conta a sub-divisão de cada uma das suas faces num número de quadrados especificado pelo utilizador, sendo estes, por sua vez, compostos por 2 triângulos.

De modo a tornar a geração do sólido mais eficiente, e tendo em conta que este partilha a mesma estratégia de formulação do plano, decidiu-se proceder ao desenho de um quadrado em todas as faces da caixa por iteração do ciclo interior da função. Deste modo, o processo de cálculo das coordenadas dos pontos resultará num número significativamente menor de instruções por ciclo, aumentando o desempenho do programa e contribuindo para uma melhor escalabilidade do tamanho e detalhe do sólido.

Um exemplo da geração de um quadrado na face superior da caixa seria o seguinte:

- **Ponto A**
  x = -h_size + (size * j);
  y = h_size;
  z = h_size - (size * i);
- **Ponto B**
  x = -h_size + (size * j);
  y = h_size;
  z = h_size - (size * (i+1));
- **Ponto C**
  x = -h_size + (size * (j+1));
  y = h_size;
  z = h_size - (size * i);
- **Ponto D**
  x = -h_size + (size * (j+1));
  y = h_size;
  z = h_size - (size * (i+1));

Nota: *para as restantes faces basta alternar a ordem das coordenadas x, y e z entre si.

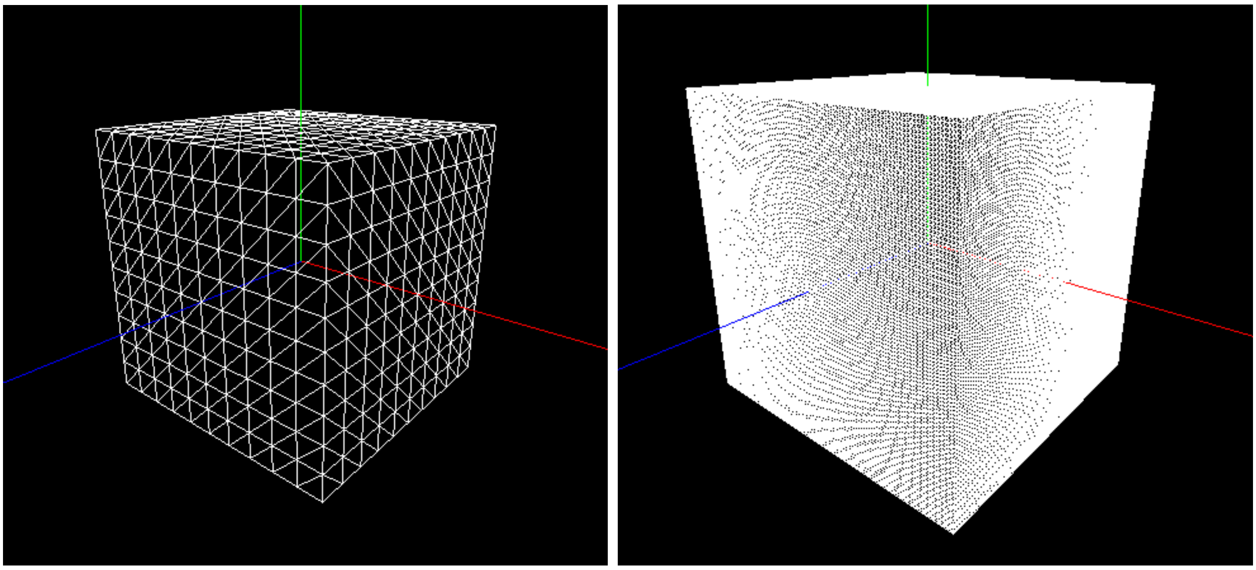

Figura 3.3: *Grid* 10 vs. *Grid* 100

3.3 Esfera

De forma a ser possível gerar a esfera, são necessários 3 parâmetros:

- **Raio** - O raio da esfera que se pretende gerar;
- **Slice** - Como a esfera é dividida verticalmente, em fatias, é necessário decidir o número de *slices* a ser utilizado;
- **Stack** - Como a esfera é dividida horizontalmente, em pilha, é necessário decidir o número de *stacks* a ser utilizado.

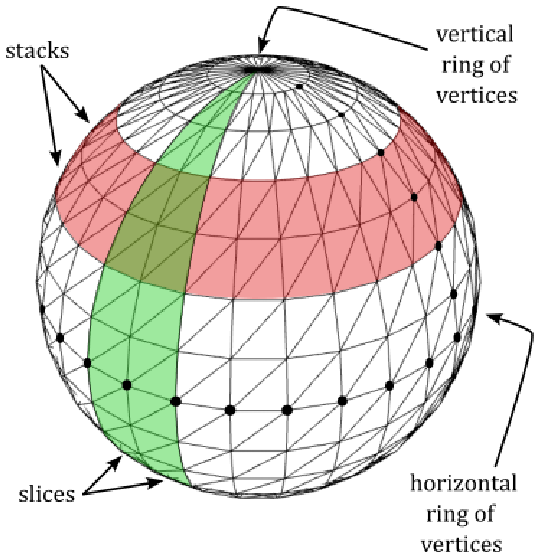

Figura 3.4: Representação gráfica - *Slices* e *Stacks*

Assim, tendo em conta a ideia de *slices* e *stacks* considera-se os seguintes ângulos alpha e beta:

- $\alpha = 2\pi/\text{slices}$ - Responsável por gerar as *slices* em relação ao plano xOz.
- $\beta = \pi/\text{stacks}$ - Responsável por gerar as *stacks* em relação ao eixo Y.

E, também, é necessário realizar, para todos os vértices, a conversão de coordenadas esféricas para coordenadas cartesianas, através das seguintes fórmulas no vértice(x, y, z):

- x = radius * sin(β) * cos(α);
- y = radius * cos(β);
- z = radius * sin(β) * sin(α);

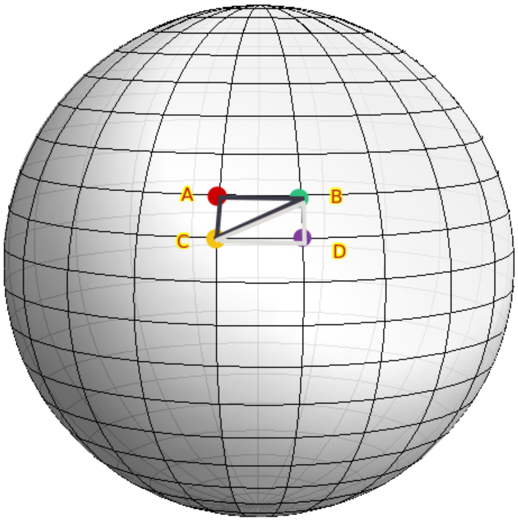

Figura 3.5: Representação gráfica da região delimitada

Desta forma, para ser possível gerar a esfera, implementa-se um ciclo aninhado, onde o i representa as *stacks* e o j representa as *slices*, onde a cada iteração são declarados os 4 pontos que representam a região limitada pela *stack* e *slice* em questão e, posteriormente, são considerados dois triângulos.

Tendo em conta a seguinte imagem, o triângulo superior é formado pelos pontos ABC e o trinângulo inferior é formado pelos pontos BCD.

- **Ponto A**
  x = radius * sin(β * (i+1)) * cos(α * (j+1));
  y = radius * cos(β * (i+1));
  z = radius * sin(β * (i+1)) * sin(α * (j+1));
- **Ponto B**
  x = radius * sin(β * i) * cos(α * (j+1));
  y = radius * cos(β * i);
  z = radius * sin(β * i) * sin(α * (j+1));
- **Ponto C**
  x = radius * sin(β * (i+1)) * cos(α * j);
  y = radius * cos(β * (i+1));
  z = radius * sin(β * (i+1)) * sin(α * j);
- **Ponto D**
  x = radius * sin(β * i) * cos(α * j);
  y = radius * cos(β * i);
  z = radius * sin(β * i) * sin(α * j);

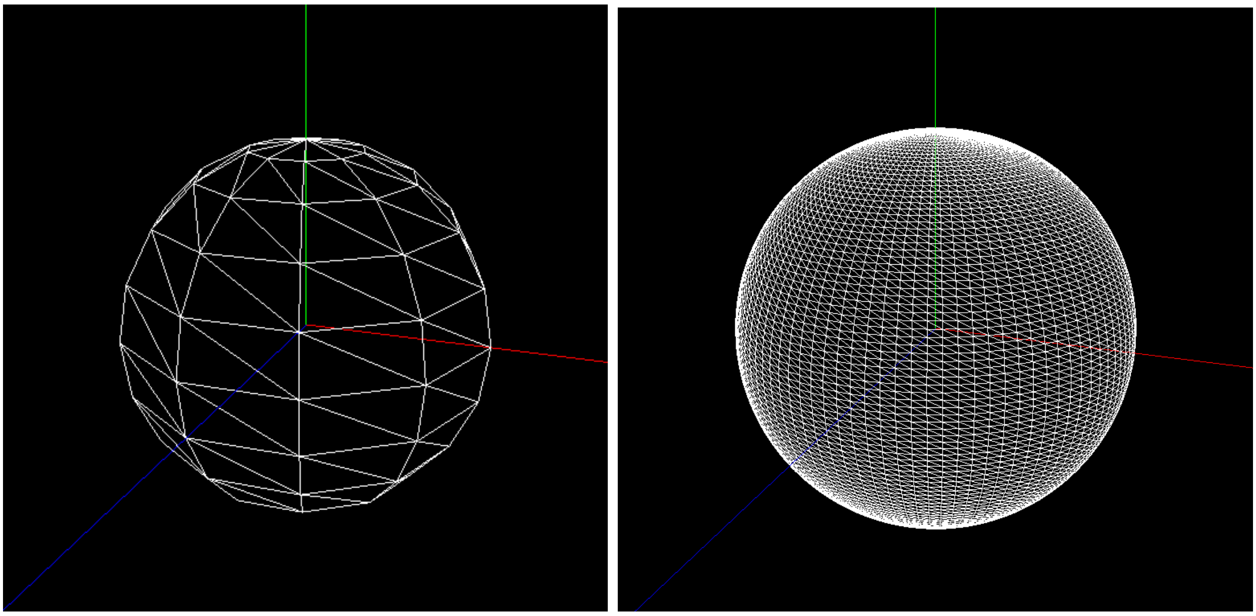

Figura 3.6: 10 *Slices* e *Stacks* vs. 100 *Slices* e *Stacks*

3.4 Cone

De forma a ser possível gerar o cone, são necessários 4 parâmetros:

- **Raio** - O raio da base do cone que se pretende gerar;
- **Altura** - A altura do cone;
- **Slices** - Como o cone é dividido verticalmente, em fatias;
- **Stacks** - Como o cone é dividido horizontalmente, em pilhas.

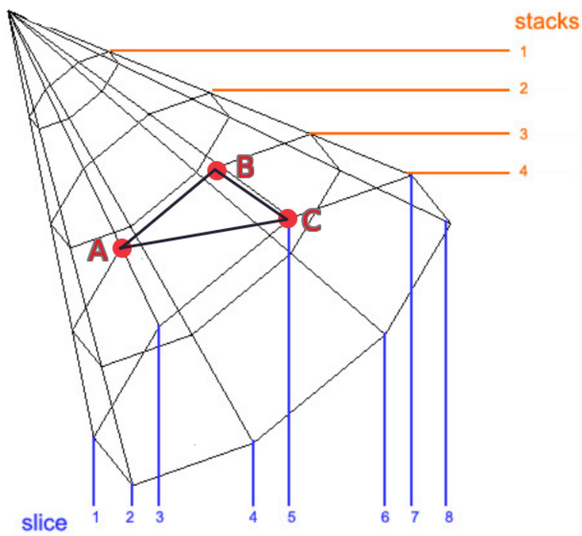

Figura 3.7: Representação de um Cone - *Slices* e *Stacks*

Sendo a base de um cone uma circunferência, é através do número de *slices* que é possível saber quantos triângulos irão compor a base, pelo que se determina o valor de α a partir do seguinte cálculo:

- $\alpha = 2\pi/slices$

Posteriormente, em cada *slice* são utilizados 3 pontos, como a altura do cone, $(0, height, 0)$ e dois pontos que fazem parte da base. Como a base se trata de uma circunferência, torna-se estritamente necessário recorrer à passagem de coordenadas cartesianas para polares, através da seguinte fórmula:

- $x = r * cos\theta$
- $z = r * sen\theta$

Após gerar as *slices*, são construídas as *stacks* baseadas na circunferência atual e na circunferência de cima.

Por fim, é necessário calcular os pontos A, B e C para todas as *stacks* em todas as *slices*, onde i representa o número da *stack* que está a ser gerada e a variável j a *slice*:

- **Ponto A**
  x = (radius - (radius/stacks) * i) * sin(α * (j+1));
  y = stack_height*i;
  p4z = (radius - (radius/stacks) * i) * cos(α * (j+1));

- **Ponto B**
  x = (radius - (radius/stacks) * i) * sin(α * j);
  y = stack_height*i;
  z = (radius - (radius/stacks) * i) * cos(α * j);

- **Ponto C**
  x = (radius - (radius/stacks) * (i-1)) * sin(α * j);
  y = stack_height*(i-1);
  z = (radius - (radius/stacks) * (i-1)) * cos(α * j);

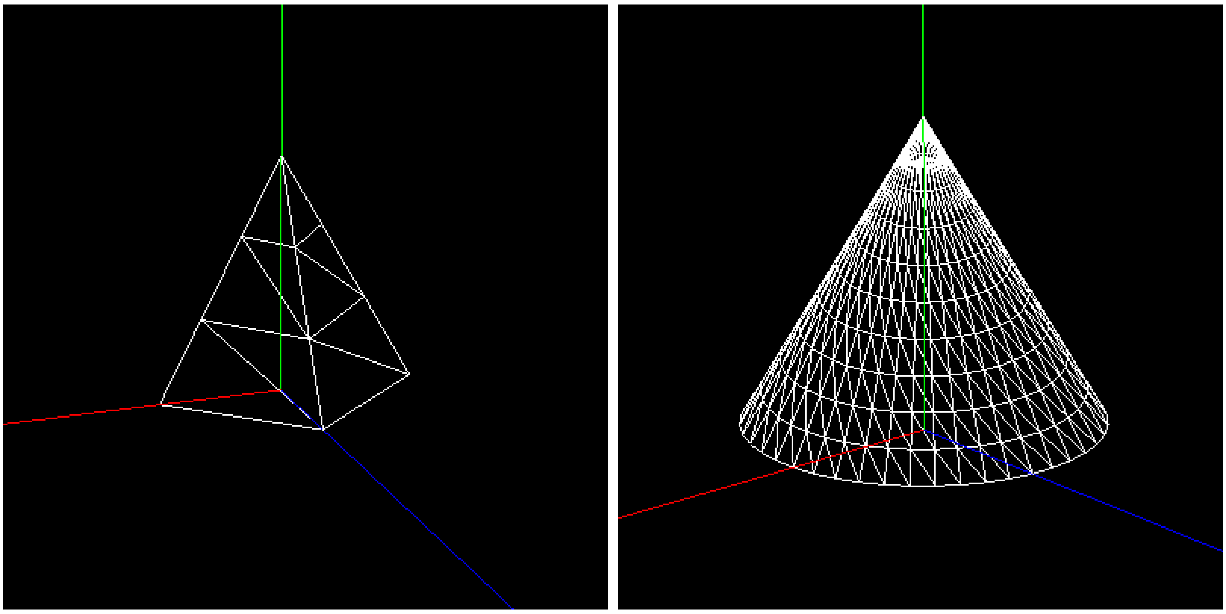

Figura 3.8: 4 Slices e 3 Stacks vs. 40 Slices e 10 Stacks

4. Câmara

Para facilitar a visualização das figuras, foi implementada a possibilidade de mover tanto a câmara como as figuras. As teclas associadas a cada movimento são:

- **W,S,A,D,J,K** : Alterar o ponto para o qual a câmara "olha"individualmente em cada eixo.
- **+,-** : Zoom in e xoom out.
- **F,L,P** : Modo *Fill*, *Line* e *Point*.
- **B,N,M** : GL_BACK, GL_FRONT e GL_BACK_AND_FRONT.
- **KEY_UP,KEY_DOWN,KEY_LEFT,KEY_RIGHT** : Rotação da câmara em relação a cada eixo.

5. Conclusão

Com a realização desta fase, foi possível consolidar conhecimentos relacionados à renderização de gráficos 3D e à aplicação de conceitos aprendidos nas aulas lecionadas, no contexto da utilização da ferramenta *OpenGL* e, também, expandir o conhecimento em C++ e manipulação de ficheiros XML.