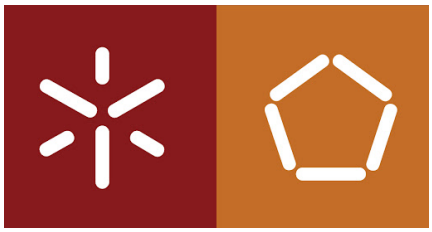

Universidade do Minho

Licenciatura em Engenharia Informática

Computação Gráfica - Fase 1
Grupo 38

Bernardo Emanuel Magalhães Saraiva - A93189
José João Cardoso Gonçalves - A93204
Mariana Rocha Marques - A93198 Rui Filipe Coelho Moreira - A93232

Ano Lectivo 2021/2022

Conteúdo

1 Introdução

Para a primeira fase do Trabalho Prático da UC de Computação Gráfica, foi proposto aos alunos o desenvolvimento de uma aplicação que permitisse gerar diferentes tipos de primitivas gráficas de acordo com os inputs recebidos pelo utilizador.

Para alcançar o objetivo proposto, a presente fase consta com duas aplicações distintas. A primeira tem como função gerar os vértices dos sólidos, gerando um ficheiro *.3d* contendo os vértices, sendo esta dada pelo nome de **generator**. A segunda aplicação complementa a anterior, no sentido de que representa graficamente os pontos obtidos através da leitura de um ficheiro XML, sendo esta chamada de **engine**.

Para a implementação destas duas aplicações foi utilizada a linguagem de programação C++, bem como se recorreu ao OpenGL para possibilitar o desenvolvimento gráfico do trabalho proposto.

2 Estrutura do código

2.1 Generator

O generator é o primeiro passo da aplicação desenvolvida, tendo como função gerar os ficheiros que contêm os vértices do sólido pretendido, de acordo com os argumentos dados. Deste modo, o generator efetua o parsing dos argumentos recebidos e chama a função do respetivo sólido com esses argumentos, através da função *write3D(T primitive, string fileName)*.

Cada vértice é composto por três coordenadas, x, y e z, onde cada sequência de três vértices corresponde a um triângulo. Aqui, é tido em conta a regra da mão direita, de forma a garantir a construção correta dos triângulos.

2.2 Engine

O *Engine* corresponde ao módulo cuja função é a leitura dos ficheiros *XML* e a visualização da representação gráfica proveniente dos ficheiros *.3d* resultantes do *generator*. Deste modo, a aplicação conta com algumas funções importantes no que diz respeito ao desenho das primitivas. É de referir que para efetuar o parsing do XML recorreu-se à biblioteca TinyXML2, uma biblioteca de análise sintática de XML para a linguagem C++. É também nesta classe que são calculados os movimentos da câmara, pelo que se descreve com maior detalhe na Secção 7.

3 Plano

Para a criação de um plano são utilizados como parâmetros:

- **lado:** comprimento do lado do plano
- **divisions:** número de divisões a serem feitas a cada eixo

Como se trata de um plano em XZ, todos os pontos terão o seu parâmetro y a 0. Considerando *lado* como o comprimento introduzido de um dos lados, px/pz como o comprimento de cada sub-divisão ($px = lado/divisions$), e que o plano está centrado na origem, os primeiros dois triângulos a serem desenhados utilizariam estes pontos:

- Ponto A: $(px * i - lado/2, 0, pz * q - lado/2)$
- Ponto B: $(px * i - lado/2, 0, pz * q - lado/2 + pz)$
- Ponto C: $(px * i - lado/2 + px, 0, pz * q - lado/2)$
- Ponto D: $(px * i - lado/2 + px, 0, pz * q - lado/2 + pz)$

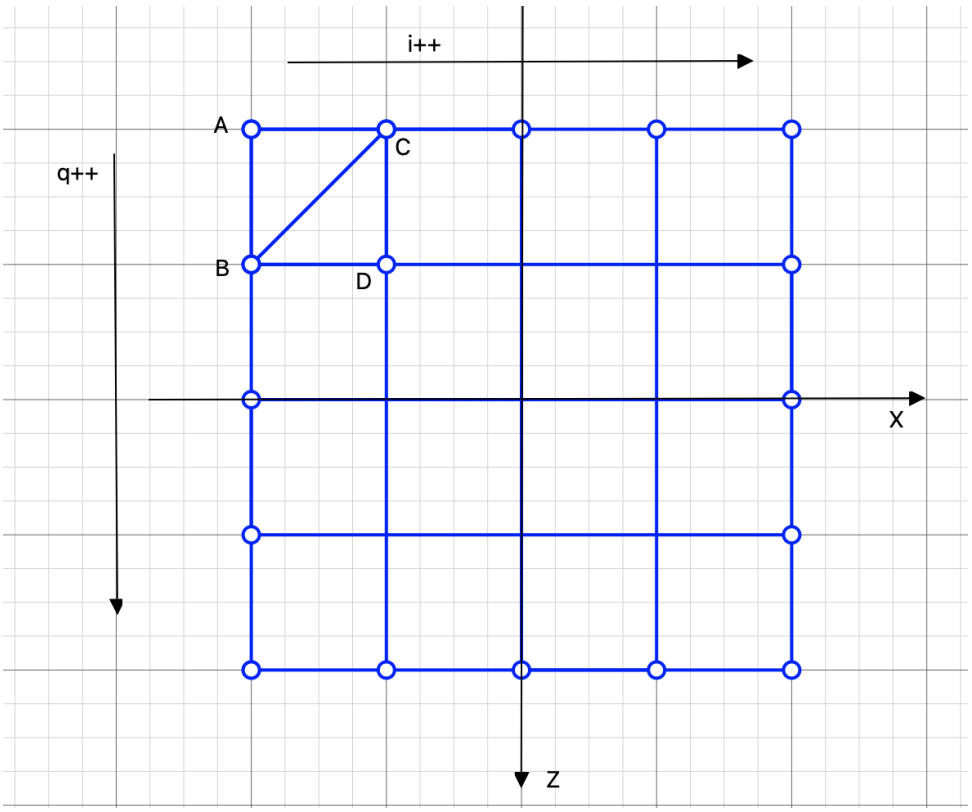

Figura 1: Plano

Para este ser construído, utilizamos dois ciclos para desenharmos os triângulos, coluna a coluna, iterando i e o q gerando os pontos com as fórmulas acima.

4 Box

Para além do plano descrito anteriormente, também foi desenhada a Box. Para o desenho desta primitiva, é necessário passar dois argumentos:

- **Edge:** Este argumento indica o comprimento correspondente ao lado do sólido.
- **Grid:** Este argumento indica o número de divisões de face que o cubo possui, sendo que cada face irá conter $grid^2$ quadrados por face. A titulo de exemplo, se for indicado o valor 3 para o grid (por exemplo), o cubo será dividido em $3^2 = 9$ quadrados, como se demonstra na imagem seguinte.

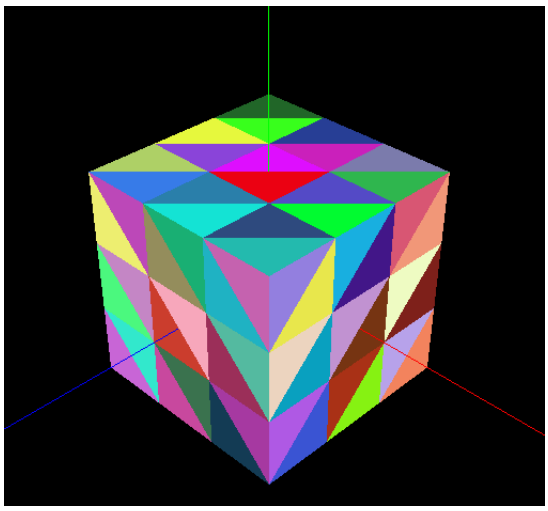

Figura 2: Imagem exemplificativa de Box com edge = 2 e grid = 3

Relativamente ao cálculo dos pontos, os mesmos são gerados a partir de um *ciclo for* aninhado dentro de outro - fazendo assim referência ao $grid^2$ quadrados por face - que desenha os pontos.

Tal como descrito anteriormente no plano, cada quadrado é composto por 2 triângulos. Na construção deste sólido, após calcular cada conjunto de quatro pontos numa dada iteração, os pontos são agrupados em grupos de 3, com o objetivo de formar triângulos obedecendo à regra da mão direita.

Nota: O centro do cubo corresponde à origem do referencial onde o modelo é construído.

5 Esfera

De modo a desenhar a esfera, são necessários 3 parâmetros:

- **Raio:** O raio da esfera que se pretende representar;
- **Slice:** Uma vez que a esfera é dividida em fatias, tal como demonstrado na Figura 4, o utilizador necessita de especificar o número de slices que pretende, através do terceiro argumento;
- **Stack:** Tal como nas slices, o utilizador necessita de especificar o nº de stacks que pretende utilizar para representar a esfera, tal como representado na Figura 4

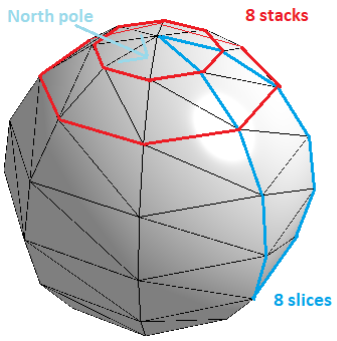

Figura 3: Representação gráfica dos Slices e Stacks

Para proceder ao cálculo dos pontos necessários para o desenho dos triângulos recorremos a coordenadas esféricas, realizando esta abstração com as seguintes variáveis:

- *Radius:* Distância do ponto à origem (raio da esfera)
- $\alpha = 2\pi/\text{slices}$
- $\beta = \pi/\text{stacks}$

Com o α conseguimos representar a rotação em radianos entre o eixo do x e o eixo do z podendo tomar valores entre 0 e 2π, já com o β representamos a rotação entre o plano $x\text{O}z$ e o eixo do y, de forma a gerar valores entre 0 e π.

Apesar desta abstração possibilitar a representação da esfera o OpenGL apenas reconhece coordenas cartesianas, de modo que, é necessário realizar a conversão através das seguintes fórmulas:

$$x = radius * cos(\beta) * sin(\alpha)$$

$$y = radius * sin(\beta)$$

$$z = radius * cos(\beta) * sin(\alpha)$$

De forma a desenhar a esfera, é necessário implementar um ciclo aninhado onde a cada iteração é desenhado um slice de cada stack da esfera, cada slice é subdividida em dois triângulos que são representados da seguinte forma:

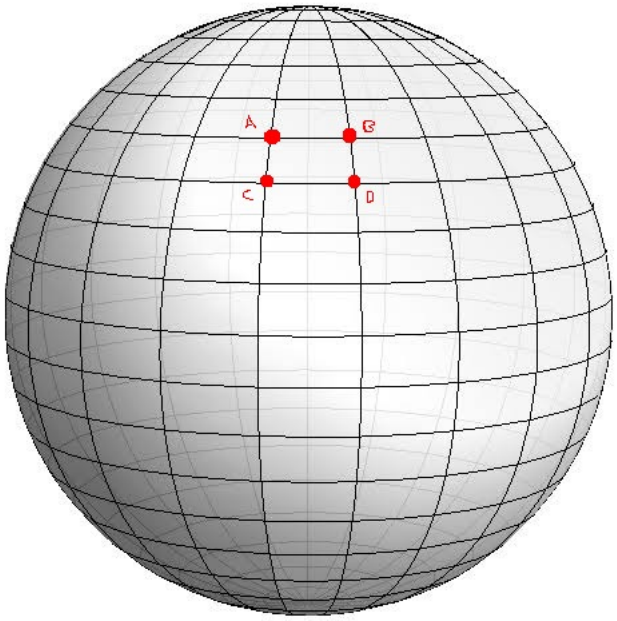

Figura 4: Representação gráfica de um esfera subdividida em Slices

O triângulo superior é formado pelos pontos ABC, já o inferior é formado pelos pontos BCD.

- **Ponto A:**

$$\begin{aligned} \mathrm{X} &= r * cos((j+1) * \beta) * sin(i * \alpha) \\ \mathrm{Y} &= r * sin((j+1) * \beta) \\ \mathrm{Z} &= r * cos((j+1) * \beta) * cos(i * \alpha)) \end{aligned}$$

- **Ponto B:**

$$\begin{aligned} \mathrm{X} &= r * cos((j+1) * \beta) * sin((i+1) * \alpha) \\ \mathrm{Y} &= r * sin((j+1) * \beta) \\ \mathrm{Z} &= r * cos((j+1) * \beta) * cos((i+1) * \alpha)) \end{aligned}$$

- **Ponto C:**

$$\begin{aligned} \text{X} &= r * cos(j * \beta) * sin(i * \alpha)) \\ \text{Y} &= r * sin(j * \beta) \\ \text{Z} &= r * cos(j * \beta) * cos(i * \alpha)) \end{aligned}$$

- **Ponto D:**

$$\begin{aligned} \text{X} &= r * cos(j * \beta) * sin((i+1) * \alpha) \\ \text{Y} &= r * sin(j * \beta) \\ \text{Z} &= r * cos(j * \beta) * cos((i+1) * \alpha)) \end{aligned}$$

Nota1: i e j representam respetivamente slice e stack

Nota2: É importante referir que quanto maior o número de slices e stacks passado (tanto na esfera como no cone), maior será a perfeição do sólido representado, uma vez que leva a um maior detalhe do sólido, mas também mais recursos do PC. Na figura abaixo é possível verificar uma demonstração dessa diferença.

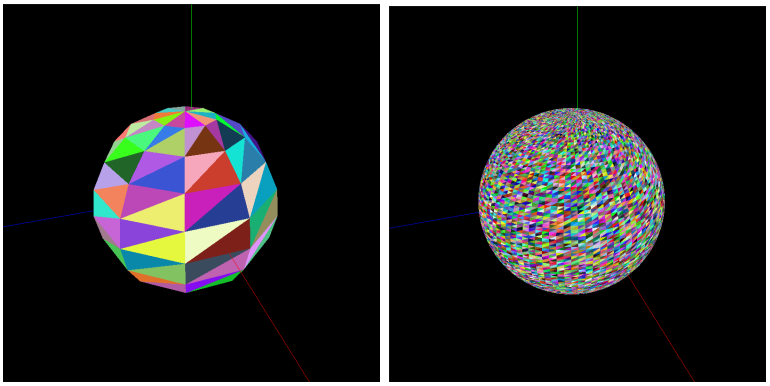

Figura 5: Comparação entre uma esfera com 10 stacks e slices e outra com 100 stacks e slices

6 Cone

Para desenhar esta primitiva, é necessário que o utilizador introduza 4 parâmetros:

- **Radius:** O raio da base;
- **Height:** A altura do cone;
- **Slices:** O número de fatias em que se divide o sólido. Estas são divididas desde a circunferência que compõe a base até ao topo;

- **Stacks:** O número de camadas em que o sólido é dividido, contadas horizontalmente;

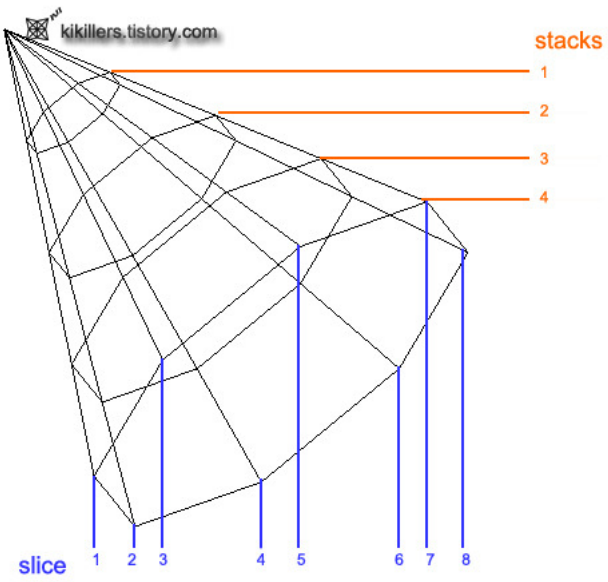

Figura 6: Imagem ilustrativa das slices e stacks de um cone

Sendo a base de um cone uma circunferência, a mesma irá conter a divisão das *slices*, pelo que se determina o valor de α a partir do seguinte cálculo:

$$\alpha = 2\ \pi / slices$$

Posteriormente, para cada slice são desenhados os 3 pontos que a compõem, sendo estes o topo do cone e os dois pontos da circunferência que a delimitam. Deste modo, para o desenho destes pontos da slice na circunferência, torna-se estritamente necessário recorrer à passagem das coordenadas cartesianas para polares, sendo a base desenhada no eixo $x\mathrm{O}z$.

Para a conversão das coordenadas polares para coordenadas cartesianas, recorre-se à seguinte fórmula:

$$x = r\ *\ cos\ \theta$$

$$z = r\ *\ sen\ \theta$$

Posteriormente, a cada slice são construídas as suas stacks, pelo que se calcula os seus pontos baseando-se na circunferência atual e na circunferência de cima. Para tal, passa-se a explicar cada uma das variávies utilizadas e o objetivo de cada uma:

- **StackHeight:** Esta variável corresponde à altura da stack que se está a desenhar, que é calculada através da fórmula que se apresenta em seguida

$$stackHeight = (height / stacks) * j$$

- **NextRadius:** Por forma a determinar o raio da circunferência seguinte (acima da atual), utilizamos a seguinte fórmula:

$$nextRadius = (height - stackHeight) * radius / height;$$

- **stackHeight2 e nextRadius2:** Uma vez que temos que formar um quadrado, precisamos de identificar a circunferencia de cima para poder unir os pontos, usando para isso esta variável.

$$stackHeight2 = (height / stacks) * (j + 1)$$

$$nextRadius2 = (height - stackHeight2) * radius / height$$

Por fim, após o cálculo dos pontos, é utilizada a regra da mão direita para ordenar os pontos que se pretende representar.

7 Movimentação da Câmara

Com o objetivo de analisar a primitiva desenvolvida, foram desenvolvidas duas funções que permitem movimentar a câmara e o sólido. Assim, descreve-se em seguida as funções desenvolvidas e as teclas associadas a cada movimento.

- **W — A — S — D:** translação do cubo me relação aos eixos x e z (em relação ao referencial);
- **H — L:** girar a câmara em volta da figura;
- **J — K:** alterar a escala da figura;
- **Q:** voltar à apresentação inicial da figura;
- **1 — 2 — 3:** alterar o viewmode da figura, ou seja, mostrar só as linhas, só os pontos ou colorido (GL_FILL, GL_LINE ou GL_POINT);
- **KEY_UP — KEY_DOWN — KEY_LEFT — KEY_RIGHT:** Movimento de rotação da câmara em torno de uma superfície esférica, de forma a visualizar a primitiva previamente desenhada de vários ângulos.

8 Conclusão

Em suma, a presente fase do trabalho prático permitiu a representação de primitivas gráficas de sólidos geométricos através de gerar pontos correspondentes a um dado sólido, com determinadas dimensões e características, provenientes do input do utilizador.

Com o desenvolvimento desta primeira fase foi possível consolidar conhecimentos relacionados com a produção de gráficos 3D, através da renderização de primitivas gráficas, bem como aplicar os conhecimentos obtidos nas aulas num contexto prático e compreender melhor os conceitos de OpenGL.

Para além destes conhecimentos, a presente fase do trabalho permitiu expandir o conhecimento na linguagem C++ e na manipulação de ficheiros XML.

No decorrer do desenvolvimento desta fase, as principais dificuldades do grupo centraram-se na idealização das funções que levam à representação dos sólidos, uma vez que as mesmas requerem a marcação dos pontos dependendo dos valores recebidos como argumento.