Relatório da fase 1

Nome: Luís Gustavo Aires Guimarães Maia
Nº Aluno: A95656
Nº Grupo: 50

1. Introdução

No âmbito da Unidade Curricular de Computação Gráfica foi-nos proposto desenvolver um pequeno motor gráfico e demonstrar o seu potencial. Para tal, foram criados 2 programas: *Generator* e *Engine*. O programa *Generator* é responsável pela criação das faces e vértices de 4 primitivas: *Plane, Box, Sphere* e *Cone*. Após a sua criação, são escritas as coordenadas dos seus vértices em ficheiros *.3d*. O programa *Engine* é responsável pela leitura de um ficheiro de configuração *XML* e de representar graficamente as primitivas criadas.

2. Decisões e abordagens

Generator

Para o desenvolvimento deste programa foram criadas 4 funções que geram as coordenadas dos vértices de cada primitiva.

- ***Plane***

  A função *generatePlane (float size, int divisions, std::vector<float>& vertices)* gera um plano 3D subdividido em pequenas células quadradas, usando triângulos como elementos básicos. A abordagem usada consiste em dividir um quadrado de tamanho *size* numa grelha com *divisions* * *divisions* células, resultando em células quadradas menores, cada uma representada por dois triângulos.

  Para calcular as coordenadas, foi necessário calcular a metade do tamanho do plano para o centrar na origem (*metade = size / 2.0f*) e também calcular o tamanho de cada célula (*tamanho_celula = size / divisions*).

  O plano é gerado percorrendo cada célula da grelha com dois *loops* aninhados, usando a variável *i* que controla as coordenadas do eixo Z e a variável que controla as coordenadas do eixo X. As coordenadas de cada célula serão (x1, 0, z1) para o canto inferior esquerdo e (x2, 0, z2) para o canto superior direito, sendo:

  x1= *-size/2 + j*tamanho_celula*
  z1= *-size/2 + i*tamanho_celula*
  x2= x1 + *tamanho_celula*
  z2= z1 + *tamanho_célula*

  Cada célula é dividida em dois triângulos. Os vértices são inseridos no vetor *vertices* na ordem correta para garantir a formação dos triângulos com as coordenadas dos vértices dos dois triângulos da célula sendo:

  Primeiro triângulo: Vértice A: (x2, 0, z2), Vértice B: (x2, 0, z1) Vértice C: (x1, 0, z1)

Segundo triângulo: Vértice D: (x1, 0, z1), Vértice E: (x1, 0, z2) e Vértice F: (x2, 0, z2)

- ***Box***

  A função *generateBox (float size, int divisions, std::vector<float>& vertices)* gera uma caixa 3D subdividida em pequenas células quadradas, cada uma formada por dois triângulos. A abordagem usada consiste em dividir cada face da caixa numa grelha de *divisions * divisions* e gerar os vértices de cada triângulo correspondente. Para calcular as coordenadas, foi necessário calcular a metade do tamanho da caixa para a centrar na origem (*metade = size / 2.0f*) e também calcular o tamanho de cada célula (*tamanho_celula = size / divisions*). Tratando-se de um cubo, a caixa tem as seguintes faces:
  - face = 0: Frente (z = +*size* / 2)
  - face = 1: Trás (z = −*size* / 2)
  - face = 2: Direita (x = +*size* / 2)
  - face = 3: Esquerda (x = −*size* / 2)
  - face = 4: Topo (y = +*size* / 2)
  - face = 5: Fundo (y = −*size* / 2)

  Cada face da caixa é dividida numa grelha de *divisions * divisions*. Os *loops i* e *j* percorrem essa grelha para calcular os cantos das células: (x1, y1) para o canto inferior esquerdo da célula e (x2, y2) para o canto superior direito da célula, com:

  x1= *-size/2 + j*tamanho_celula*
  y1= *-size/2 + i*tamanho_celula*
  x2= x1 + *tamanho_celula*
  y2= y1 + *tamanho_célula*

  Para cada célula, são gerados **dois triângulos**, que cobrem a área do quadrado. Cada face tem um eixo fixo (x, y ou z constante), e os vértices são calculados de acordo com a face correspondente.

  Exemplo - Face Frontal (z = +metade)
  - **Primeiro triângulo:**
    - (x2, y2, metade)
    - (x2, y1, metade)
    - (x1, y1, metade)
  - **Segundo triângulo:**
    - (x1, y1, metade)
    - (x1, y2, metade)
    - (x2, y2, metade)

  Este padrão é repetido para cada uma das seis faces da caixa.

- ***Sphere***

  A função *generateSphere (float radius, int slices, int stacks, std::vector<float>& vertices)* gera uma esfera 3D subdividida em fatias (*slices*) e camadas (*stacks*). Cada segmento da esfera é representado por pequenos triângulos que compõem sua superfície. A esfera é construída

iterativamente usando coordenadas esféricas, onde: *theta* representa a latitude (ângulo do polo norte ao polo sul) e *phi* representa a longitude (ângulo ao redor do eixo vertical).
A função divide a esfera em faixas de latitude (*stacks)* e fatias de longitude (*slices*).
Os ângulos são definidos da seguinte forma:

$$\theta 1 = \pi * (i / stacks)$$
$$\theta 2 = \pi * (i + 1 / stacks)$$
$$\phi 1 = 2\pi * (j + 0.5 / slices)$$
$$\phi 2 = 2\pi * (j + 1.5 / slices)$$

Cada ponto da esfera é convertido de coordenadas esféricas para cartesianas usando:

$$x = r * \sin(\theta) * \cos(\phi),\ y = r * \cos(\theta) \text{ e } z = r * \sin(\theta)*\sin(\phi)$$

Usando estas equações, são calculados quatro pontos para cada célula da esfera:

(x1, y1, z1) → Ponto na latitude *theta1*, longitude *phi1*
(x2, y2, z2) → Ponto na latitude *theta1*, longitude *phi2*
(x3, y3, z3) → Ponto na latitude *theta2*, longitude *phi1*
(x4, y4, z4) → Ponto na latitude *theta2*, longitude *phi2*

Cada célula da esfera é dividida em dois triângulos:
Primeiro triângulo: (x1, y1, z1), (x2, y2, z2) e (x3, y3, z3)
Segundo triângulo: (x3, y3, z3), (x2, y2, z2) e (x4, y4, z4)
Estes triângulos cobrem toda a superfície da esfera.

- ***Cone***

A função *generateCone (float radius, float height, int slices, int stacks, std::vector<float>& vertices)* gera um cone 3D dividido em fatias (*slices*) ao redor do eixo e camadas (*stacks*) ao longo da altura. O cone é composto por uma base "circular" e uma superfície lateral formada por pequenos triângulos.
Para calcular as coordenadas dos vértices do cone, foram necessários os seguintes valores:

stackHeight = *height / stacks* → Altura de cada camada ao longo do eixo Y.
angleStep = 2 * π / *slices* → Incremento angular para gerar os pontos ao longo da circunferência da base.
radiusStep = *radius* / *stacks* → Variação do raio ao longo das camadas para afunilar o cone.

Para gerar o cone foi preciso dividir em duas partes: a base e a superfície lateral. A base do cone é um "círculo" no plano XY, com centro em (0, 0, 0) e raio *radius*. Ela é dividida em fatias triangulares que compartilham o centro:
Cada fatia é composta por um triângulo com vértices:

Centro da base → (0,0,0)
Ponto no círculo atual → (r * cos(θ), 0, r * sin(θ))
Próximo ponto no círculo → (r * cos(θnext), 0, r * sin(θnext))
Os cálculos de cada ponto seguem:

$$x = r * \cos(\theta)$$
$$z = r * \sin(\theta)$$

Estes triângulos são adicionados ao vetor *vertices*.

A superfície lateral do cone é formada por pequenos quadriláteros, cada um subdividido em dois triângulos. Esses quadriláteros são definidos entre duas camadas consecutivas (*stacks*).
Cada camada tem um raio e altura específicos:

currentHeight = j * *stackHeight* → Altura da camada atual
nextHeight = (j+1) * *stackHeight* → Altura da próxima camada
currentRadius = *radius* - j * *radiusStep* → Raio da camada atual
nextRadius = *radius* - (j+1) * *radiusStep* → Raio da próxima camada

Cada quadrilátero é definido pelos pontos:
Base do quadrilátero (camada atual): (x1, *currentHeight*, z1) e (x2, *currentHeight*, z2)
Topo do quadrilátero (próxima camada) (x3, nextHeight, z3) e (x4, nextHeight, z4)
Estes pontos seguem:

$$x = raio * \cos(\theta)$$
$$z = raio * \sin(\theta)$$

Cada quadrilátero é dividido em dois triângulos:
Triângulo 1 (inferior): (x1, *currentHeight*, z1), (x3, *nextHeight*, z3) e (x4, *nextHeight*, z4)
Triângulo 2 (superior): (x2, *currentHeight*, z2), (x1, *currentHeight*, z1) e (x4, *nextHeight*, z4)
Esses triângulos são armazenados em *vertices*.

Engine

Para o programa *Engine* foi seguida a seguinte lógica:

- Primeiro, o programa lê o ficheiro de configuração *XML* que lhe é passado como parâmetro, fazendo o seu *parsing* usando a biblioteca *TinyXML*. Nesta fase, são criadas estruturas de dados que guardam todas as informações do ficheiro de configuração (definições de câmara, modelos a ser usados, etc.) para que seja feita apenas uma leitura deste ficheiro.
- De seguida, os ficheiros *.3D* previamente criados pelo *Generator* são carregados para um vetor, colocando nele todas as coordenadas dos vértices dos modelos presentes no ficheiro de configuração.
- Após este último passo, são carregadas todas as informações do ficheiro de configuração (que estão nas estruturas de dados) para as funções da biblioteca *GLUT* de modo a que as cenas pretendidas sejam montadas.
- Por último, são desenhados os triângulos das primitivas.