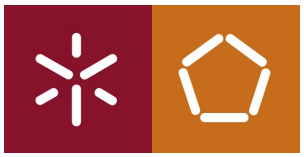

Universidade do Minho

Computação Gráfia

2018/2019

Phase 1 – Graphical primitives

Trabalho realizado por:	*Número de Aluno:*
Nelson Gonçalves	A78713
João Aloísio	A77953
César Henriques	A64321
Ricardo Ponte	A79097

8 de Março de 2019

Conteúdo

1 Introdução

A vertente prática da unidade curricular de Computação Gráfica tem como base a utilização do OpenGL, recorrendo à biblioteca GLUT, para a construção de modelos 3D.

A produção dos modelos referidos envolve diversos temas, entre os quais, transformações geométricas, curvas e superfícies, iluminação e texturas. Com o objetivo de aplicar e demonstrar a aprendizagem destes tópicos, foi proposta a realização de um projeto prático, que se encontra dividido em quatro fases, cada uma com uma data específica de entrega.

A existência destas quatro fases tem como finalidade a organização e simplificação do desenvolvimento do projeto, sendo que cada uma deverá respeitar os requisitos básicos, inicialmente estipulados no enunciado do trabalho.

Este relatório diz respeito à realização da primeira fase do projeto mencionado. Para uma melhor compreensão do objetivo desta fase, na secção seguinte será devidamente descrito o problema a resolver.

2 Descrição do Problema

Esta primeira fase do projecto requer a criação de duas aplicações: uma que gera ficheiros com a informação dos modelos a desenhar (nesta fase apenas guarda os pontos dos vértices do modelo), o Gerador, e um Motor que lê as configurações num ficheiro, escrito em XML, e representa gráficamente os modelos.

- Gerador (recebe como parâmetro o tipo de primitiva gráfica, parâmetros relativos ao modelo e o nome do ficheiro onde vão ser guardados os vértices da figura a representar:
  - **plano** - quadrado no eixo XZ, centrado na origem, feito com 2 triângulos;
  - **caixa** - requer coordenadas x,y,z e , opcionalmente, número de divisões;
  - **esfera** - requer raio , fatias , camadas;
  - **cone** - requer raio inferior, altura,fatias e camadas;
  - guardas os pontos gerados correspondentes à figura geométrica pedida em ficheiros no formato **nomefigura.3d**.
- Motor (lê o ficheiro XML que contém os modelos das primitivas):
  - lê os modelos do ficheiro XML.
  - **armazena os vértices** dos modelos em meória.
  - **desenha**, utilizando triângulos, as primitivas relativas aos modelos.

3 Resolução do Problema

3.1 Gerador

Abaixo estão enunciados os métodos de geração de pontos das figuras geométricas e a respetiva fundamentação dos mesmos.

3.1.1 Plane

Para a construção do plano é necessário o desenho de dois triângulos, sendo este centrado na origem e sobre os eixos x e z. Sendo assim, todos os vértices possuem a coordenada y a 0, e as coordenadas x e z têm valores iguais ao módulo.

De forma a ser possível gerar os pontos necessários para a sua construção, foi passado como argumento o comprimento do lado do plano. Posto isto, tanto como no x e no z, as suas coordenadas irão ser iguais ao $\frac{comprimento}{2}$, diferenciando ou não consoante o quadrante em que o vértice se localiza.

3.1.2 Box

Uma caixa é composta por 6 planos e cada plano é composto por 2 triângulos, ou seja, no total serão desenhados 12 triângulos, 2 por cada face da caixa. A caixa terá o centro base nas coordenadas (0,0,0). O gerador irá receber 3 argumentos x, y, z, comprimento, largura e altura, respetivamente. Com isto a caixa terá $\frac{x}{2}$ para o sentido positivo e negativo do eixo x, $\frac{z}{2}$ para o sentido positivo e negativo do eixo z, e y de altura.

3.1.3 Sphere

Uma esfera neste programa receberá como argumentos o raio (*radius*) , número de fatias (*slices*) e número de camadas (*stacks*), para além do seu ficheiro de *output*. Para gerar uma esfera é necessário utilizar coordenadas esféricas. Com recurso a estas será possível através de dois ângulos α e β , e da distância do ponto à origem (*radius*), calcular as coordenadas de um ponto. É sabido que o ângulo α varia entre $[0, 2\pi]$ e que o ângulo β varia entre $[\frac{-\pi}{2}, \frac{\pi}{2}]$. Sabe-se também que α será dividido em *slices* e que β será dividido em *stacks* , ou seja, será necessário percorrer *slices* divisões com o ângulo α e *stacks* divisões com o ângulo β. Por consequência, para o ângulo α cada divisão valerá $\frac{2\pi}{slices}$, em que o ciclo irá desde 0 até à divisão *slices*. Relativamente ao ângulo β como inicia em $\frac{-\pi}{2}$ o seu valor estará contido em $[\frac{-stacks}{2}, \frac{stacks}{2}]$. Cada divisão valerá $\frac{\pi}{stacks}$. Uma vez que estão a ser utilizadas coordenadas esféricas é possível calcular qualquer pontos tendo por base apenas a variação dos ângulos. A fórmula de cálculo das coordenadas x, y, z será muito semelhante ao cálculo das coordenadas da câmara, sendo definido pelas expressões:

$$x = radius \times \sin(\alpha) \times \cos(\beta) \tag{1}$$

$$y = radius \times \sin(\beta) \tag{2}$$

$$z = radius \times \cos(\beta) \times \cos(\alpha) \tag{3}$$

A estratégia adoptada foi calcular 2 ângulos de cada vez recorrendo às variáveis *alfa1* , *alfa2* , *beta1* e *beta2* de modo a calcular os vértices necessários para o desenho correto de uma *slice*,isto será feito em todas as *slices*. As variáveis são respectivas aos ângulos α e β relativos à iteração atual e à próxima, respectivamente.

3.1.4 Cone

Um cone será composto pelo raio (*radius*) , altura (*height*) , fatias(*slices*) e camadas (*stacks*) bem como o ficheiro de *output*.O ângulo α varia entre $[0, 2\pi]$ e será dividido em *slices* divisões. Para cada divisão este ângulo irá ser incrementado $\frac{2\pi}{slices}$. A distância de um ponto à origem é calculada em relação à altura. A altura total do cone vai ser dividida em *stacks* divisões e por cada uma destas a altura será incrementada $\frac{height}{stacks}$. Recorrendo ao raio (*radius*, que é alterado aquando da altura) é possível, sabendo o ângulo α , calcular as coordenadas pretendidas. Neste caso, a coordenada y não necessitará de ser calculada sendo que esta corresponde ao valor da altura considerada. Sendo assim, as fórmulas de cálculo para as coordenadas x e z são:

$$x = radius \times \sin(\alpha) \tag{4}$$

$$z = radius \times \cos(\alpha) \tag{5}$$

Contudo no programa, optou-se por utilizar funções que calculam os eixos e/ou altura relativos à iteração em que um ponto se encontra. Sendo assim o raio passará a ser representado pela expressão $radius = radius - \frac{(radius \times i)}{stacks}$ e altura pela expressão $\frac{height \times i}{stacks}$, em que i corresponde ao número da iteração do ciclo. Substituindo nas fórmulas acima e adicionando a fórmula da altura temos que:

$$x = (radius - \frac{(radius \times i)}{stacks}) \times \sin(\alpha) \tag{6}$$

$$z = (radius - \frac{(radius \times i)}{stacks}) \times \cos(\alpha) \tag{7}$$

$$height = \frac{height \times i}{stacks} \tag{8}$$

Pode-se utilizar estas fórmulas para calcular os pontos da base do cone, bastando para isso igualar *height* a 0. Desde a divisão inicial até à divisão *slices* os valores do ângulo α serão incrementados de modo a percorrer todos os valores de α. A estratégia adotada foi calcular 2 ângulos de cada vez recorrendo às variáveis *alfa1* e *alfa2* à semelhança do procedimento seguido na esfera, sendo que desta vez não é necessário o ângulo β.

3.2 Motor

3.2.1 Leitura do ficheiro XML

Uma vez que a informação relativa aos ficheiros que devem ser carregados (ficheiros .3d) se encontra num ficheiro XML, utilizou-se a biblioteca *TinyXML-2* para realizar o *parsing* do mesmo. Este ficheiro XML será passado como argumento ao Motor. Para ler o ficheiro XML foi implementada a função *readXML* que , com o auxílio da biblioteca mencionada acima percorrerá o ficheiro e sempre que encontra um modelo armazena o seu caminho no *array* de **string** *paths*.

3.2.2 Leitura dos modelos

Após saber-se quais os modelos que devem ser carregados, procede-se à leitura dos mesmos. Para este efeito percorre-se o *paths* anteriormente preenchido e posteriormente a função *loadXML* será invocada. Esta função percorrerá todos os caminhos contidos em *paths*. Cada modelo será percorrido de inicio a fim, sendo que serão lidas 3 linhas de cada vez formando assim um vértice. Esse vértice será inserido no triângulo e a cada 3 vértices adicionados o triângulo será adicionado ao modelo. Após um modelo ser percorrido este irá ser adicionado ao **vector** $< Model >$ *modelz*.

3.2.3 Desenho das Primitivas

Tendo como objetivo o desenho das primitivas dos vértices previamente carregados para memória foi criada a função *drawFiles*. Esta função percorrerá os modelos derivados dos vértices e irá desenhar os triângulos requeridos para a primitiva pedida , recorrendo à função *glVertex3d*, para desenhar os vértices constituintes dos triângulos. Para melhorar a percepção da constituição das primitivas os triângulos irão , alternadamente, mudar de cor, sendo a cor azul escuro ou vermelho. Isto é conseguido na função *drawFiles* , que a cada 3 pontos desenhados alterna entre as cores mencionadas acima.

3.2.4 Câmara

A câmara que fornece a visão das primitivas pode ser movimentada para cima,baixo,direita,esquerda sendo ainda possível aproximar ou afastar do ponto de referência. Neste caso o ponto de foco da câmara será a origem (0,0,0), sendo este o ponto de referência.

Posto isto, foram definidas as variáveis globais *alpha*, *beta* e *radius*, à semelhança das apresentadas na figura. Como será explicado mais à frente, a posição da câmara poderá ser alterada tendo em conta os valores dessas variáveis.

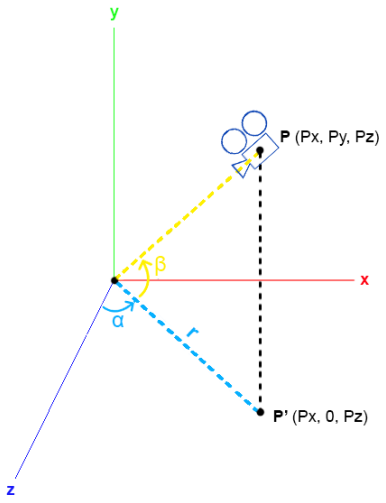

Figura 1: *Representação da posição relativa da câmara*

A função *gluLookAt* permite definir vários parâmetros, dentro destes , dado o contexto, destacam-se:

- Posição da câmara
- Posição do ponto de referência do ponto focado pela câmara
- Direção do vector up

Tal como foi mencionado anteriormente, o ponto de referência será a origem sendo assim , a direção do vetor up e posição do ponto de referência são nulos. Para calcular a posição da câmara mediante os valores de α e β, as coordenadas da câmara serão armazenadas nas variáveis *xaxis*, *yaxis* e *zaxis*, respetivamente , dadas por:

$$xaxis = radius \times sin(\alpha) \times cos(\beta) \tag{9}$$

$$yaxis = radius \times sin(\beta) \tag{10}$$

$$zaxis = radius \times cos(\beta) \times cos(\alpha) \tag{11}$$

3.2.5 Teclado

Para uma utilização mais confortável das funcionalidades de interatividade do teclado com o utilizador foram definidos os seguintes comandos:

- **w - Movimentar para cima**
- **s - Movimentar para baixo**
- **a - Movimentar para a esquerda**
- **d - Movimentar para a direita**
- **q - Aproximar da figura geométrica**
- **e - Distanciar da figura geométrica**

4 Demo das primitivas

4.1 Plano

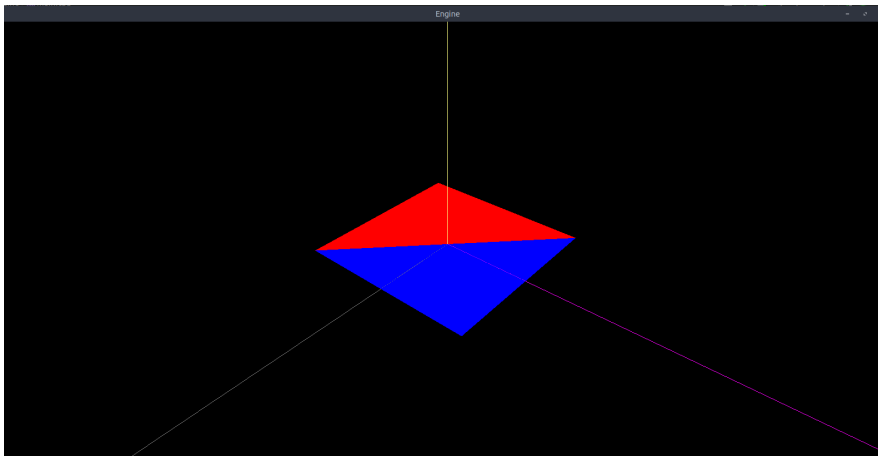

Figura 2: *Plano*

4.2 Caixa

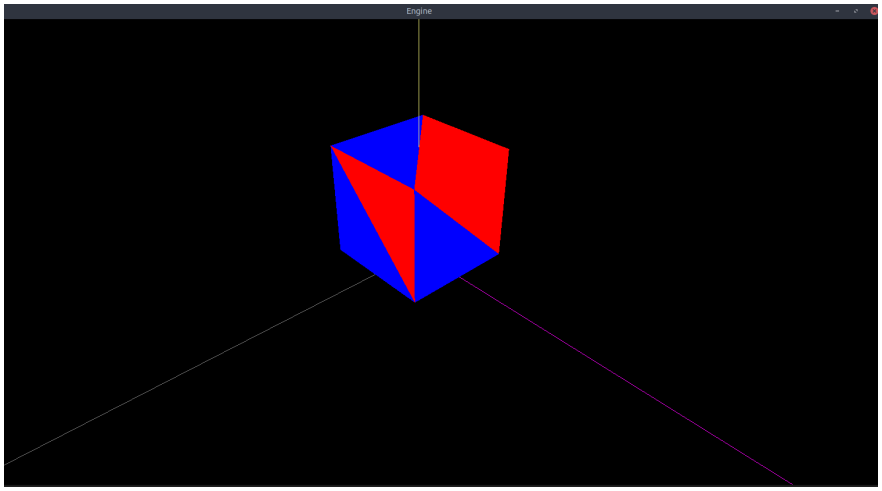

Figura 3: *Caixa*

4.3 Esfera

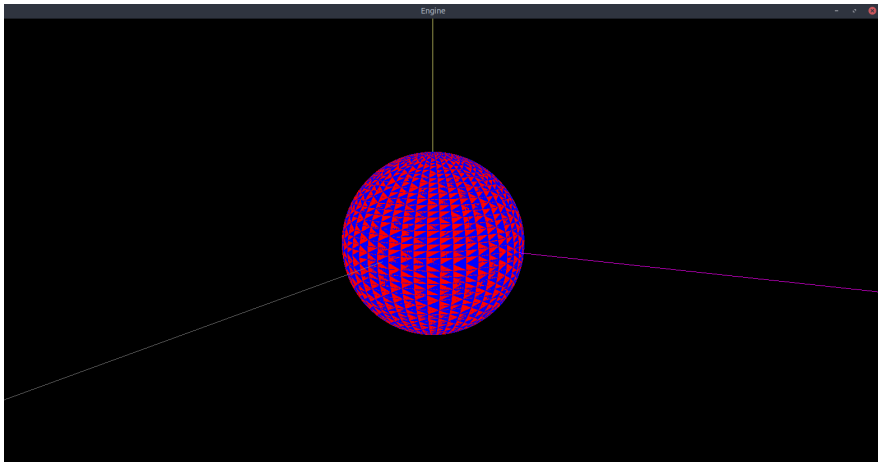

Figura 4: *Esfera*

4.4 Cone

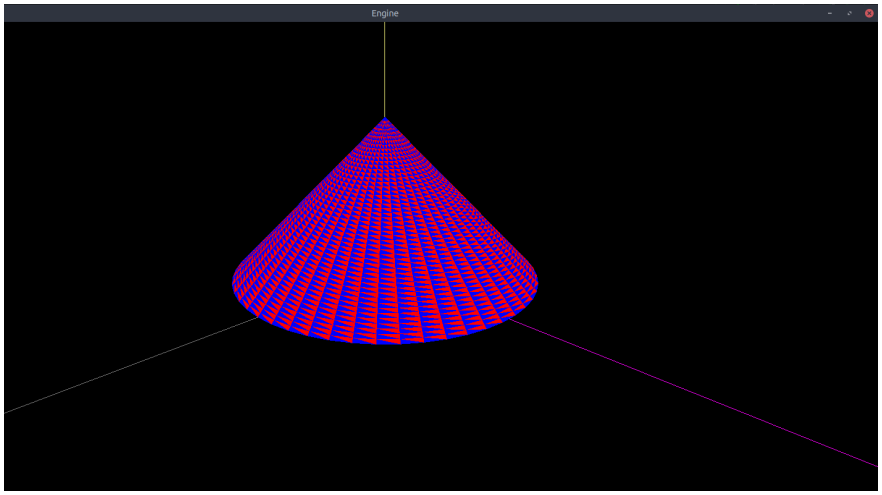

Figura 5: *Cone*

5 Conclusões

No geral, o grupo considerou que a realização desta fase do projecto foi satisfatória visto que tanto o Motor, para desenhar os modelos gerados, como o Gerador, que gera os modelos a serem carregados, funcionam para todas as figuras pedidas. O motor lê e interpreta um ficheiro XML, guarda a informação necessária e, após isso, desenha o modelo. Foi implementada a movimentação da câmara com um sistema de interação simples, fácil e intuitivo de utilizar. Contudo um dos entraves na realização do projecto foi a necessidade de utilizar caminhos absolutos para leitura do ficheiro *Config.xml*, tendo assim de fazer alteração do caminho do ficheiro em alguns parâmetros dependendo da máquina em questão, a geração de pontos e escrita no ficheiro XML referido, com caminhos absolutos também.

Visto que agora foram adquiridas mais competências a nível prático em trabalhar com as ferramentas e linguagem utilizadas, a realização da próxima tarefa, que envolve transformações geométricas, poderá ser relativamente mais simples.