Universidade do Minho

Computação Gráfica

Trabalho Prático I - Primitivas Gráficas Simples

Alfredo Gomes	Cecília Marciel	Cláudia Marques	João Vieira
A71655	A71926	A71509	A71489

06 de Março de 2017

Conteúdo

1 Introdução

O presente trabalho tem como objetivo a criação de um programa de desenho 3D. Este trabalho será dividido em quatro fases, sendo este relatório correspondente à primeira fase.

A primeira fase do projeto consiste na criação de duas aplicações. A primeira com o objetivo de criar ficheiros com a informação dos modelos (vértices), através de um *generator* e a segunda com o objetivo de ler a informação cujas primitivas se encontram num ficheiro XML, mostrando posteriormente os modelos gerados através do *engine*. As primitivas gráficas requeridas foram:

- Plano: um quadrado no eixo XZ, centrado na origem composto por 2 triângulos;
- Paralelepípedo: sólido composto nas coordenadas X,Y,Z;
- Esfera: sólido composto por raio, slices e stacks;
- Cone: sólido composto por raio da base, altura, slices e stacks.

2 Fase I - Primitivas Gráficas Simples

2.1 Generator

A primeira etapa de desenvolvimento deste projeto foi a implementação da aplicação "generator" que recebe o nome do sólido que se pretende desenhar e os argumentos que o caraterizam e gera um ficheiro que contém os pontos dos triângulos necessários ao desenho gráfico da primitiva pedida.

As primitivas gráficas disponíveis e os argumentos que as caraterizam são:

- *plane* - requer como argumento o comprimento do lado;
- *box* - requer como argumento o comprimento, largura e altura;
- *sphere* - requer como argumento o raio, o número de slices e stacks;
- *cone* - requer como argumento o raio da base, a altura e o número de slices.

O ficheiro gerado pelo *generator* tem formato .3d. A primeira linha possuí o número de pontos necessários para caraterizar a primitiva, a segunda linha o número de triângulos necessários e as restantes linhas representam os triângulos, onde cada triângulo formado tem os seus vértices XYZ separados por um ";", já cada triângulo encontra-se também, separado por um *newLine*.

O formato do comando do *generator* é:

```
> generator primitiva_grafica argumentos ficheiro.3d
```

Para auxiliar a construção das primitivas gráficas, criou-se a classe *Ponto*, de modo a que a informação de um vértice ficasse agregada. Esta classe possuí como variáveis as coordenadas de um ponto (x,y,z) e o método **printFile** que imprime as coordenadas de um ponto para o ficheiro requerido.

De modo a imprimir os vértices de um triângulo para um ficheiro, definiu-se a função **printTriangulos**, que recebe como parâmetro os três pontos do triângulo e o nome do ficheiro destino onde os vértices serão armazenados.

2.2 Primitivas gráficas

2.2.1 Plano

A primitiva plano é suportada pela função **plano** possui como parâmetros o comprimento do lado a desenhar e o nome do ficheiro no qual devem ser guardados os vértices. Como só recebe uma medida de comprimento, o plano a construir terá todos os lados com igual comprimento.

O plano é centrado na origem e para a sua construção são necessários dois triângulos. Para o efeito, são gerados um conjunto de três vértices para cada triângulos. A ordem dos pontos de cada vértice é definida pela regra da mão direita, de forma a ditar o lado para o qual o triângulo estará virado. O plano é só visível apenas de um lado, pois só foram construídos dois triângulos.

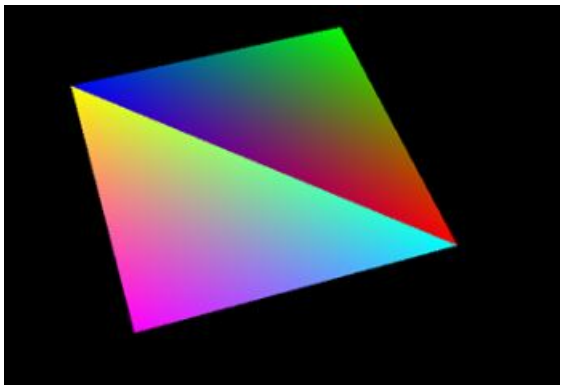

Figura 1: Representação do plano

2.2.2 Caixa

Para a criação da caixa, foi definida a função **box** que recebe como parâmetros o comprimento, a largura e altura da caixa, bem como o nome do ficheiro onde guardar os vértices dos triângulos que compõem a figura.

Uma caixa é constituída por seis faces planas, em que cada plano é constituído por dois triângulos, sendo que a função **box** retorna os vértices dos 12 triângulos, ou seja, temos um total de 36 vértices.

A caixa é centrada na origem, portanto os parâmetros que caraterizam a primitiva (comprimento, altura, largura e altura) são divididos por 2 de forma a se calcularem as coordenadas dos vértices. Tal como na função anterior, os pontos de cada vértice são impressos pela ordem da regra da mão direita.

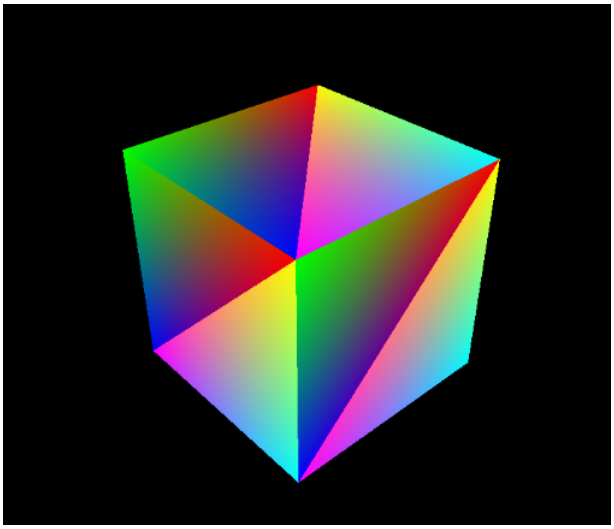

Figura 2: Representação da caixa

2.2.3 Esfera

A primitiva esfera foi definida recorrendo à função **sphere** que possui como argumentos o raio, o número de *slices*, o número de *stacks* e o nome do ficheiro destino onde os vértices serão armazenados.

Para o desenho desta primitiva recorremos às coordenadas esféricas. O sistema esférico de coordenadas é um sistema que permite a localização de um ponto qualquer em um espaço de formato esférico através de um conjunto de três valores. Para transformar as coordenadas esféricas em coordenadas cartesianas usamos a seguinte fórmula:

```
x = raio * cos(beta) * sin(alpha);
y = raio * sin(beta);
z = raio * cos(beta) * cos(alpha);
```

O domínio do ângulo *alpha* é de [0, 2π] e do ângulo *beta* é de [0, π].
Foram criadas as seguintes variáveis:

```
double anguloAlfa = (2 * M_PI) / slices;
double anguloBeta = (M_PI) / stacks;
double anguloB = -(M_PI) / 2 + anguloBeta;
double anguloBCima = (M_PI) / 2 - anguloBeta;
double anguloA = 0, anguloAPolos = 0;
double size = stacks + stacks - 2;
int nrTriangulos = slices*(stacks-1)*2;
int nrVertices = 2 + (slices*(stacks - 1));
```

A esfera é centrada na origem. Para o desenho desta primitiva começa-se pela última stack, ou seja, define-se o ponto com coordenadas (0,-radius,0) e a partir deste ponto

desenham-se todos os slices pertencentes a esta stack. Em cada iteração da construção dos slices, são definidos quatro pontos, pontos esses onde o ângulo alpha varia segundo o salto. A construção da primeira stack segue o mesmo raciocínio, sendo apenas o ponto de inicio diferente. Este ponto tem coordenadas (0,radius,0).

Na construção das stacks intermédias são utilizados dois ciclos que percorrem todos os slices e stacks. No primeiro ciclo, é controlado o ângulo *alpha*e no segundo o ângulo *beta*. Em cada iteração do segundo ciclo, são definidos quatro pontos que irão formar um quadrado, ou seja, dois triângulos. Este salto é definido pelas variáveis *anguloAlpha*.

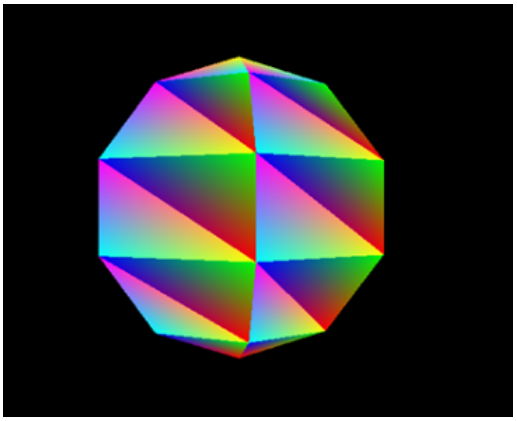

Figura 3: Representação da esfera

2.2.4 Cone

De forma a gerar o cone, foi definida a função **cone** que recebe como parâmetro o raio da base, a altura, o número de *slices*, o número de *stacks* e o nome do ficheiro destino onde os vértices serão armazenados. Apesar de termos como parâmetro o número de stacks, não conseguimos implementar o cone dividido em stacks. O cone terá apenas uma stack, independentemente do valor associado.

Para o desenho desta primitiva recorremos às coordenadas polares, que nos permitem saber que cada ponto de um plano é determinado pela sua distância em relação a um ponto fixo e do ângulo em relação a uma direção fixa. Para transformar as coordenadas polares em coordenadas cartesianas usamos a seguinte fórmula:

```
x = raio * sin(alpha);
z = raio * cos(alpha);
```

Para desenhar a base, foi definida a função **circunferenciaBaixo**, que itera sobre o número de slices, utilizando a razão entre 2π e o número de slices para incrementar o ângulo *alpha*, desde que este toma o valor 0 até complementar o circulo.

Para desenhar a lateral do cone, foi seguido o mesmo raciocínio do desenho da base (circunferência) apenas acrescentado o ponto da coordenada y, que tem como valor a altura.

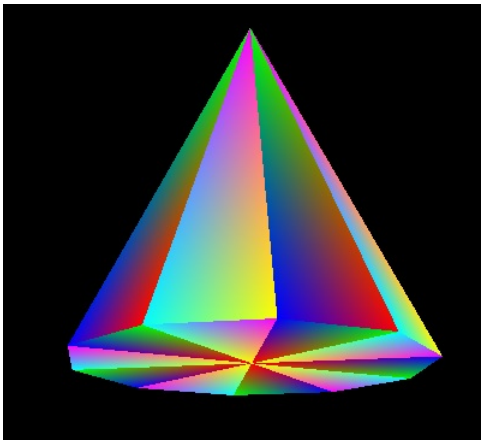

Figura 4: Representação do cone

2.3 Engine

O *engine* é responsável pelo desenho das formas geométricas, sendo que a aplicação lê a configuração do ficheiro em XML com a informação dos modelos, usando o *tinyxml* e desenha o sólido correspondente.

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

Cada tag *scene* pode ter vários modelos desenhados, *model*, que possuem como atributo o ficheiro .3d.

A função **get_3d** é responsável por descobrir o nome dos ficheiros .3d de onde serão retiradas as coordenadas dos pontos.

A função **parser** é responsável por retirar as coordenadas dos pontos de um ficheiro .3d.

Quanto às movimentações da câmara, esta pode ser conseguida através do uso das teclas:

- 'w' e 's' movem para cima e para baixo, no eixo dos x;
- 'a' e 'd' movem para os lados, ou seja, no eixo dos z;
- 'q' e 'e' movem no eixo dos y;
- 'PgDn' e 'PgUp' fazem zoom in e out;
- ArrowKeys fazem a rotação da câmara, mantendo o lookAtPoint;

3 Conclusão

Depois de realizada esta primeira fase do trabalho pudemos compreender o uso da ferramenta Visual Studio para a construção de formas geométricas 3D.A realização desta primeira fase do projeto foi muito útil para a consolidação dos conhecimentos previamente lecionados nas aulas teóricas e práticas da Unidade Curricular de Computação Gráfica.

Pensamos ter atingido todos os objetivos propostos nesta primeira fase, nomeadamente a construção das aplicações *engine* e *generator*. Todas as primitivas gráficas estão a funcionar corretamente.

Sentimos mais dificuldade na construção dos algoritmos de desenho da esfera e do cone. Passadas as dificuldades, concluímos que o trabalho foi bem conseguido.