Universidade do Minho

Computação gráfica

Departamento de Informática

Fase 1

Grupo 40:
João Abreu — A84802
Tiago Magalhães — A84485
Hugo Matias — A85370
Sérgio Gomes — A67645

6 de Março de 2020

Conteúdo

1. Fase 1 - Primitivas gráficas

1.1 Generator

O gerador foi feito para criar ficheiros que armazenem a informação necessária para criar as primitivas gráficas requeridas. Assim, o generator recebe o tipo da primitiva gráfica, os parâmetros requeridos para a criação do modelo e o ficheiro de destino onde os vértices necessários para criado o modelo vão ser armazenados.

O formato usado para armazenar os vértices do modelo é relativamente simples como se poderá ver nos subcapítulos adiante.

1.1.1 Plano

Para construir um plano são necessários quatro pontos. Assim, é possível construir dois triângulos de modo a que estes formem um plano. Na função de construção de um plano são fornecidos como argumentos a largura e o comprimento, que são usados para calcular os vértices dos triangulos que formam o modelo.

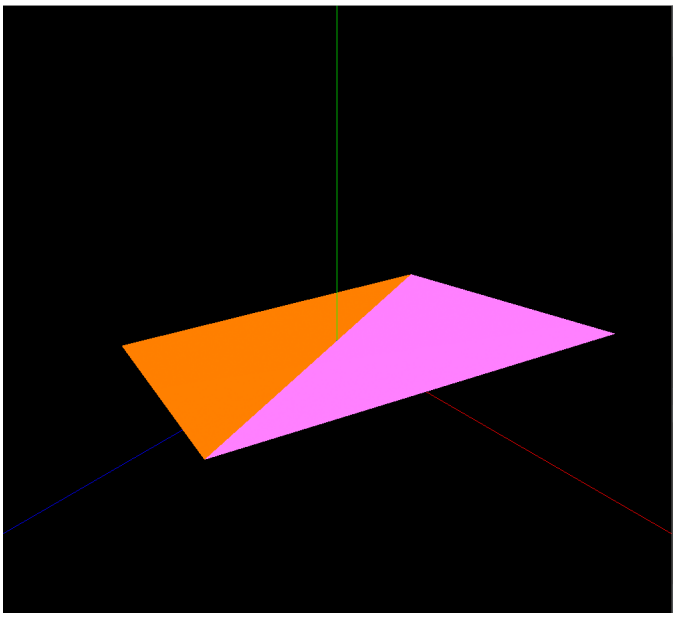

Figura 1.1: Plano.

1.1.2 Caixa

Na construção de uma caixa são necessários o comprimento, a largura e a altura. Estes parâmetros são usados para criar pontos de modo a que por sua vez seja possivel criar triângulos que formem as seis faces da

caixa. A função de construção de uma caixa recebe os parâmetros referidos acima, cacula quais os vertices que irão definir o modelo da caixa e define as faces de acordo com esses vértices. A figura abaixo é um exemplo de uma caixa. Note-se que não é possível ver os triangulos que constituiem as faces de trás, da esquerda e da base, isto deve-se a regra da mão direita o que permite a câmara capte apenas o que está na frente desta.

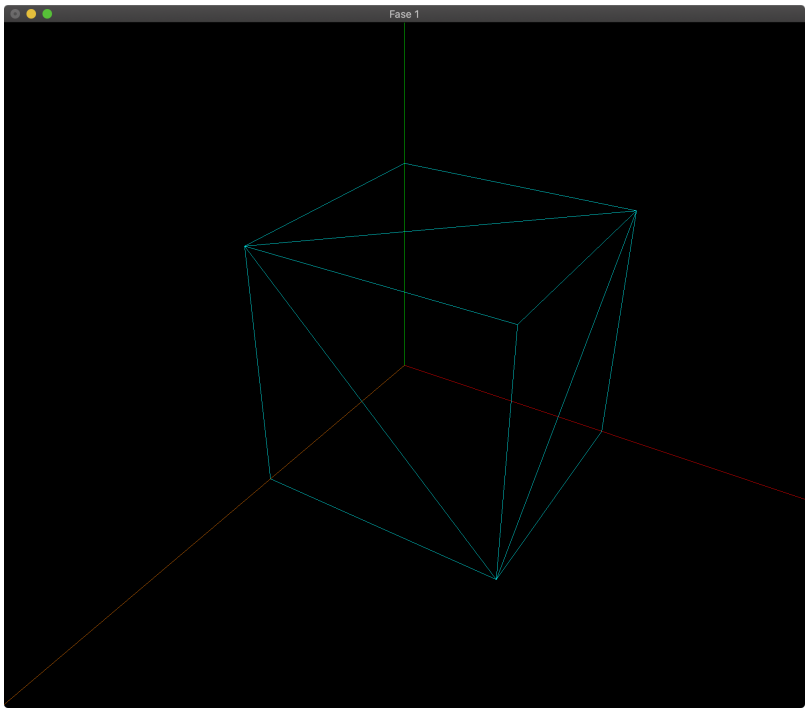

Figura 1.2: Cubo.

1.1.3 Esfera

Para construir uma esfera são necessários o raio,e os números de slices e stacks. Com estes parâmetros é possível derivar a partir de quatro pontos inicais, os restantes pontos constintuintes da esfera. Porém para isto ser possível é necessário que o número de stacks e slices seja relativamente alto,pois caso contrário não é possivel calcular a quantidade de pontos necessários de modo a construir uma esfera. A função que dita a costrução da esfera recebe os parâmetros necessários referidos acima, são criados quatro pontos usando angulos alpha e beta predefinidos(dependem do número de slices e stacks) que vão sendo atualizados de acordo com o número de slices e stacks, dando assim origem a outros pontos constituintes da esfera.

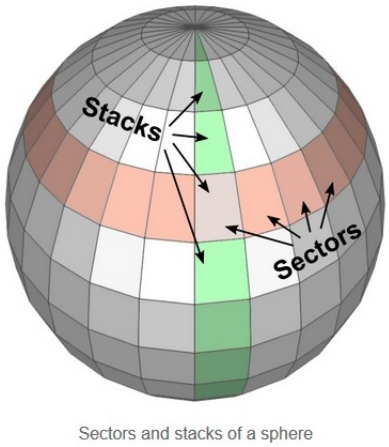

Figura 1.3: Setores e camadas de uma esfera.

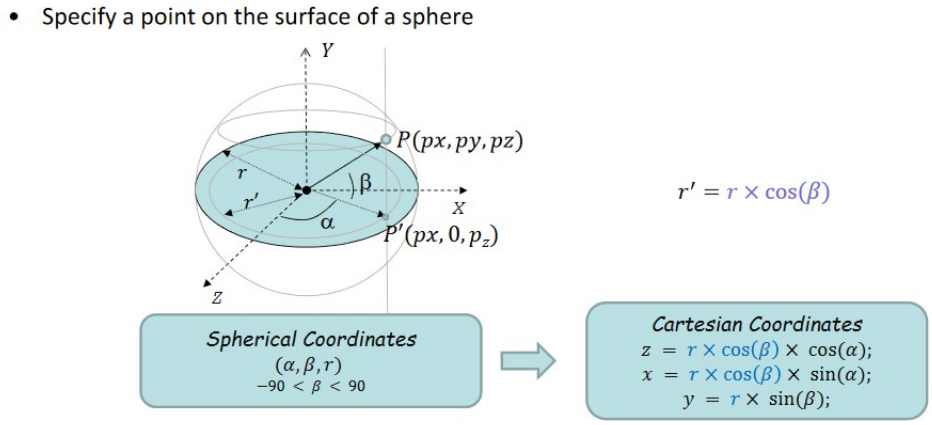

Figura 1.4: Transformação coordenadas polares.

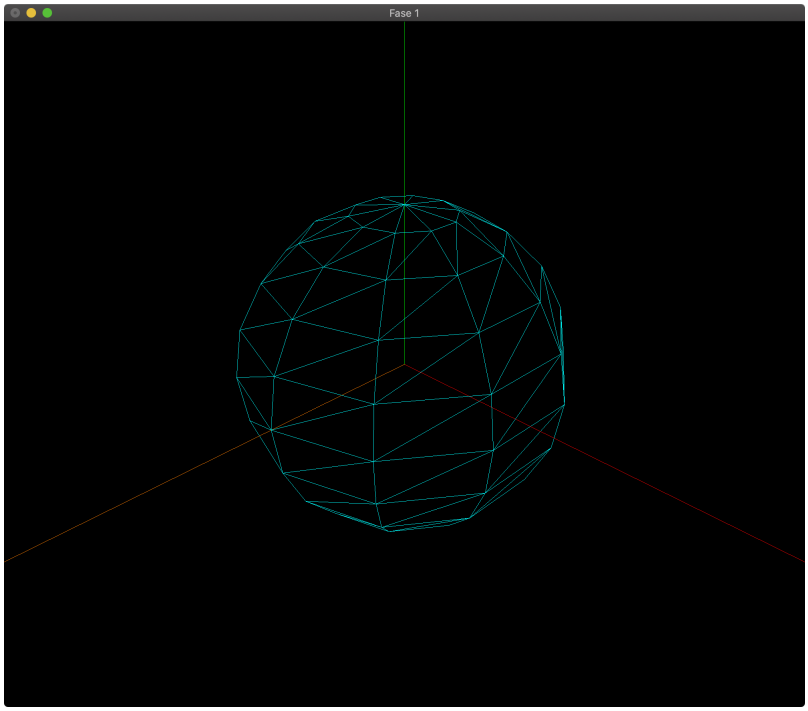

Figura 1.5: Esfera.

1.1.4 Cone

Para fazermos o cone começamos por definir quatro pontos (2 triângulos/interseção slices e stacks), como na esfera, os y's dos pontos foram obtidos através da divisão da altura pelas stacks dando altura de cada camada, para sabermos as cordenadas do eixo do x e z precisamos do raio,e pela semelhança de triângulos sabemos que o raio vai decrescer proporcionalmente, ou seja, o raio dos pontos superiores será menor do que o raio dos pontos inferiores assim obtemos a diferença de raios através da divisão do raio pelas stacks. Para sabermos o ângulo fizemos o seguinte, como ângulo alfa que gira de forma horizontal assim, a deslocação do alpha em cada slice será 2*pi(dá um volta completa) /slices.

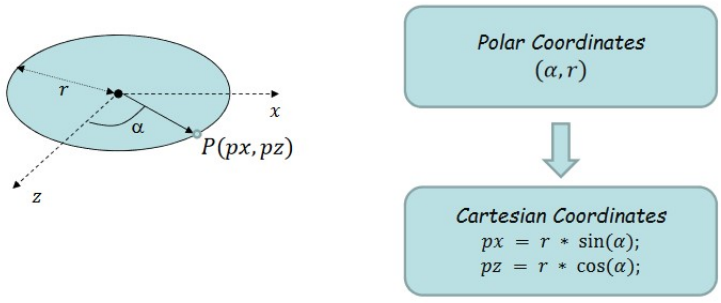

Figura 1.6: Transformação coordenadas polares.

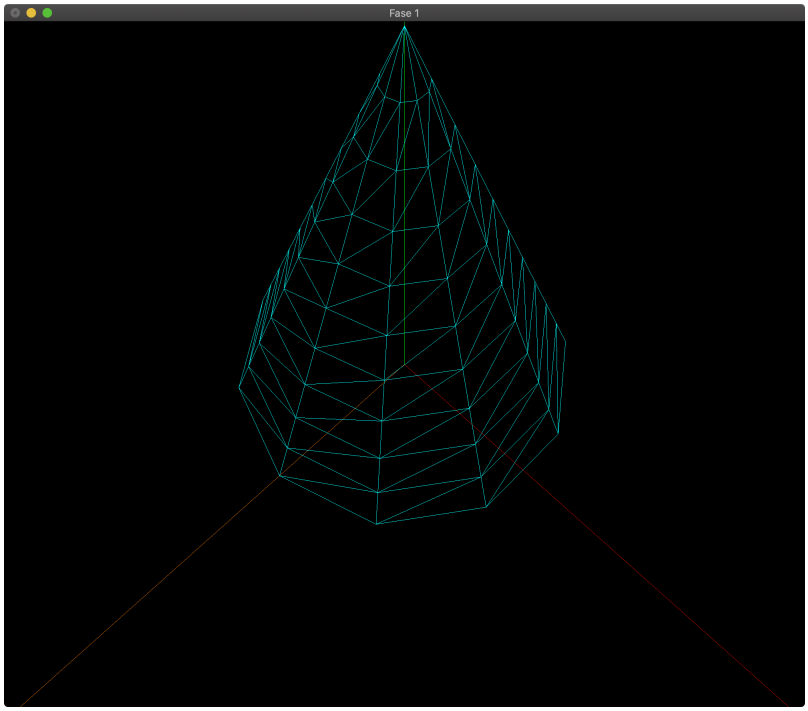

Figura 1.7: Cone.

1.2 Engine

É no Engine é onde a aplicação sabe o que tem a fazer. Este é capaz de abrir e ler o ficheiro XML que contém as primitivas a ser desenhadas. Neste ficheiro estarão presentes o full path para os ficheiros .3d, contendo cada um destes os vértices da respetiva figura. Deste modo, apenas precisamos de guardar estes vértices e depois desenhar recorrendo às ferramentas OpenGL.

Figura 1.8: Exemplo do ficheiro XML.

2. Extras

2.1 Divisões Cubo

No desafio de divisão da caixa, começamos por definir três variáveis que determinam a distância ,em cada eixo, entre cada ponto à próxima divisão.

```
float xShift = x/d;
float yShift = y/d;
float zShift = z/d;
```

De seguida são feitos três ciclos que a cada iteração vão atualizando os pontos de acordo com as variáveis referidas acima. Cada um destes ciclos vai ter três contadores, cujos contadores vão ser multiplicados por estas variáveis de modo a calcular os pontos da divisão corretos.

2.2 Movimento da câmera

Para conseguirmos fazer o movimento da câmera seguimos as regras de transformações de coordenadas polares para coordenadas cartesianas presentes nos slides da disciplina. A porção do código, e colocamos cada ponto devidamente transformado na função gluLookAt tal como está representado abaixo.

```
float px = r * cos(beta) * sin(alpha);
float py = r * sin(beta);
float pz = r * cos(beta) * cos(alpha);

// set the camera
glLoadIdentity();
gluLookAt(px,py,pz,
        0.0, 0.0, 0.0,
        0.0f, 1.0f, 0.0f);
```

E inicializámos os valores da seguinte forma:

```
float r = 20;
float alpha = M_PI/4;
float beta = M_PI/4;
```

Deste modo, estaremos a uma distância r = 10 do ponto de referência (0,0,0). Estes ângulos variam com as setas do teclado. Escolhemos PI / 70 para a variação do ângulo não por nenhuma razão em específico mas sim para notarmos a diferença entre movimentos.

```
void processSpecialKeys(int key, int xx, int yy) {

    switch (key) {
        case GLUT_KEY_DOWN:
            if (beta > -M_PI / 2)
                beta -= (float) M_PI / 70;
            break;
        case GLUT_KEY_UP:
            if (beta < M_PI / 2)
                beta += (float) M_PI / 70;
            break;
        case GLUT_KEY_RIGHT:
            alpha += (float) M_PI / 70;
            break;
        case GLUT_KEY_LEFT:
            alpha -= (float) M_PI / 70;
            break;
        default:
            break;
    }
    glutPostRedisplay();
}
```