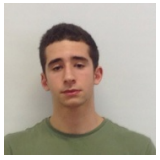
Computação Gráfica
MIEI
Grupo 27

Gonçalo Pereira

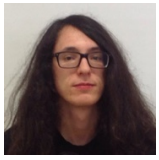
A74413

António Silva



A73827

André Diogo



A75505

6 de Março de 2017

Conteúdo

Capítulo 1

Introdução

Aquando desta primeira fase, coube-nos a responsabilidade de implementar um gerador de primitivas gráficas (e, consequentemente, um interpretador para indicar quais os parâmetros para que tal gerador funcionasse), do qual se retiraria um ficheiro em formato `.3d`.

De seguida, e através da importação de ficheiros em formato `.xml`, foi-nos incumbida a tarefa de criar um motor de *rendering* que estaria encarregue de carregar apenas os ficheiros indicados através de um ficheiro que configuração, previamente fornecido. Por fim, tal motor de renderização invocaria essas primitivas gráficas e desenhá-las-ia, em conjunto.

Como tal, e com este trabalho, demonstraremos como implementamos a nossa geração de primitivas gráficas, assim como qual o funcionamento do nosso motor de renderização.

Capítulo 2

Primitivas Gráficas

2.1 Plano

A função `generatePlane` gera um mero plano, centrado na origem, composto por dois triângulos. Desenhará consoante os dados fornecidos, desde `-length/2 até length/2` e `-width/2 até width/2`.

2.2 Caixa

A função `generateBox`, com a face de baixo, isto é, a face inferior, de `y == 0`, centrada na origem, à semelhança do cone e cilindro, apresenta duas interpretações:

- Caso não receba o parâmetro de divisões (ou caso receba mas na eventualidade de `divisões == 0`), a função **desenha o paralelepípedo normalmente**, optando por desenhar uma face de cada vez. Em cada uma delas tratará de desenhar um plano com os seus dois triângulos orientados para fora.
- Caso receba o parâmetro de **divisões maior do que 1**, optará por desenhar **consoante as subdivisões que lhe cabem**, isto é, desenhará as faces de igual modo ao anterior mas dividindo os eixos usados que lhe correspondem pelo número de divisões fornecido. Em cada uma das faces terá dois `for loops` que desenharão consoante as duas coordenadas que lhe correspondem (pois, ao desenhar uma face, uma das coordenadas é sempre igual se enquadrarmos a caixa nos três eixos). Tal resulta num número de triângulos quadrático em relação ao número de divisões, visto que uma divisão da caixa resulta em quatro planos por face, duas divisões resultam em nove planos por face, e por aí em diante. (Figura 2.1 e 2.2).

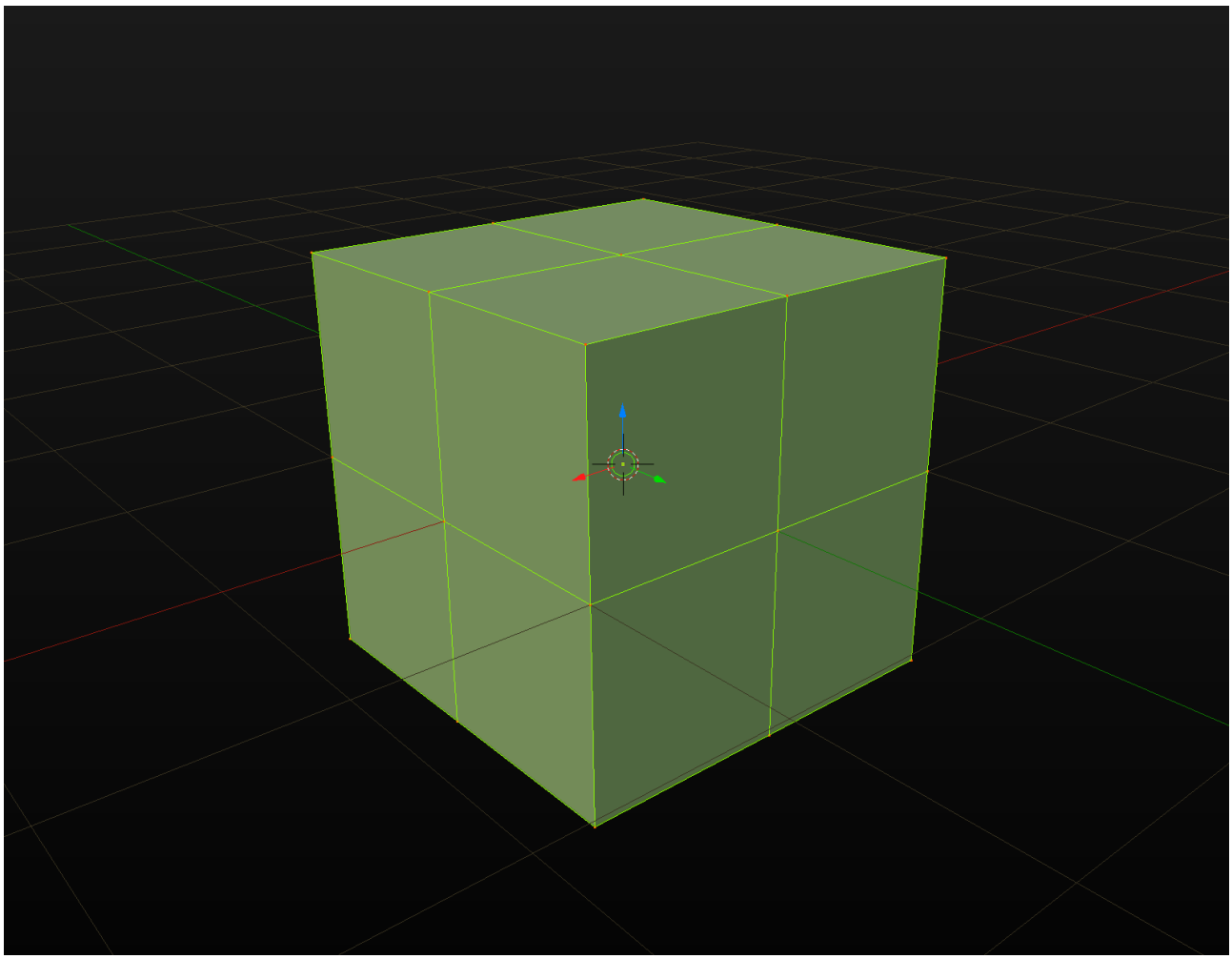


Figura 2.1: Cubo com uma sub-divisão.

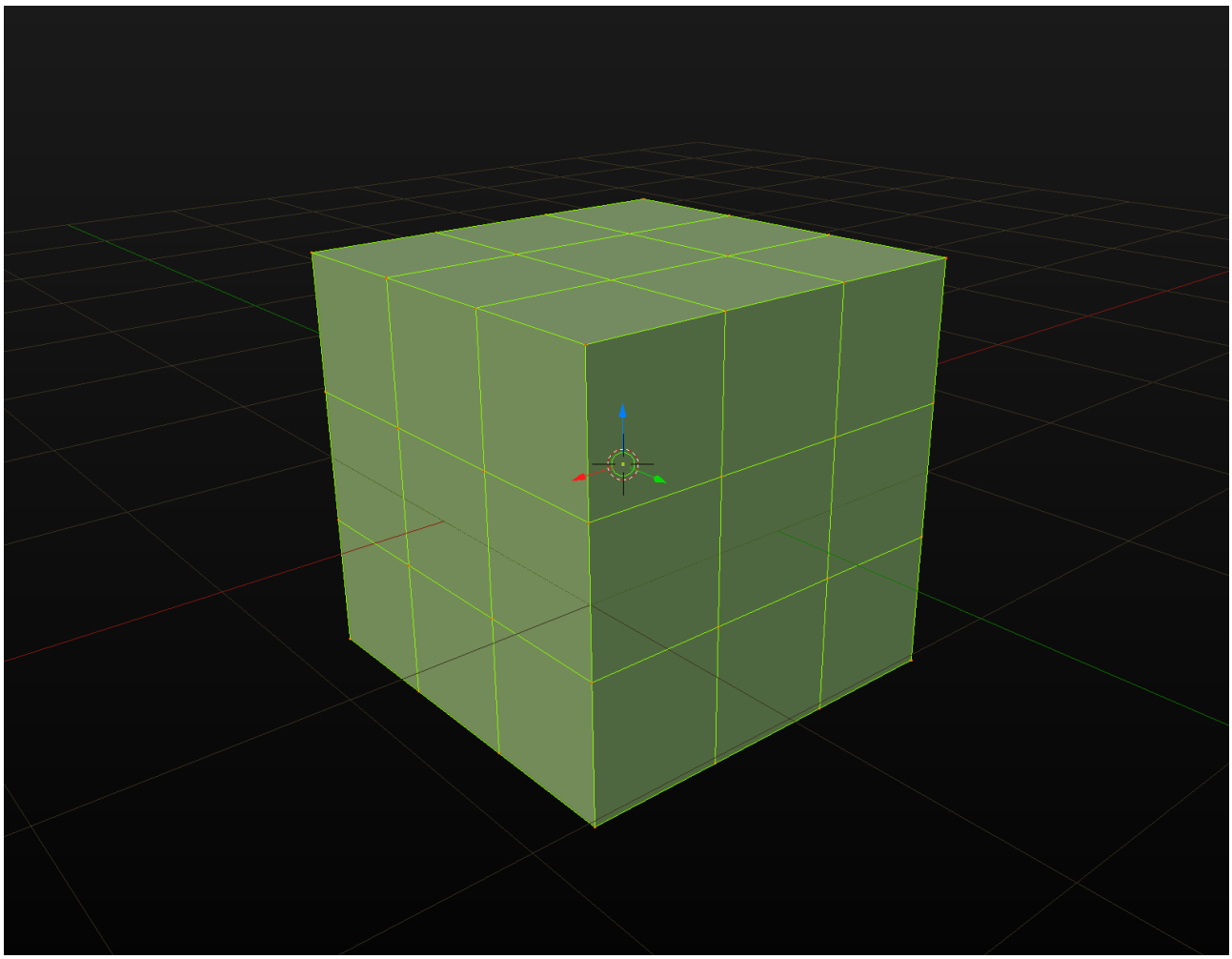


Figura 2.2: Cubo com duas sub-divisões.

2.3 Esfera

A função `generateSphere` possui um ciclo externo que percorre *stacks* de uma ponta à outra da esfera e um ciclo interno que percorre os vértices correspondentes a cada corte vertical na *stack* do ciclo externo.

Em cada iteração do ciclo externo é calculada a distancia dos dois *loops* de vértices que formam a *stack* usando o **raio** e o **cosseno** do ângulo que o *loop* faz com o eixo dos **yy**.

Em cada iteração do ciclo interno são calculados os vértices necessários para formar os dois triângulos que compõem um quadrado da *stack*, a partir da distância anteriormente referida, **senos** e **cossenos** (Figura 2.3).

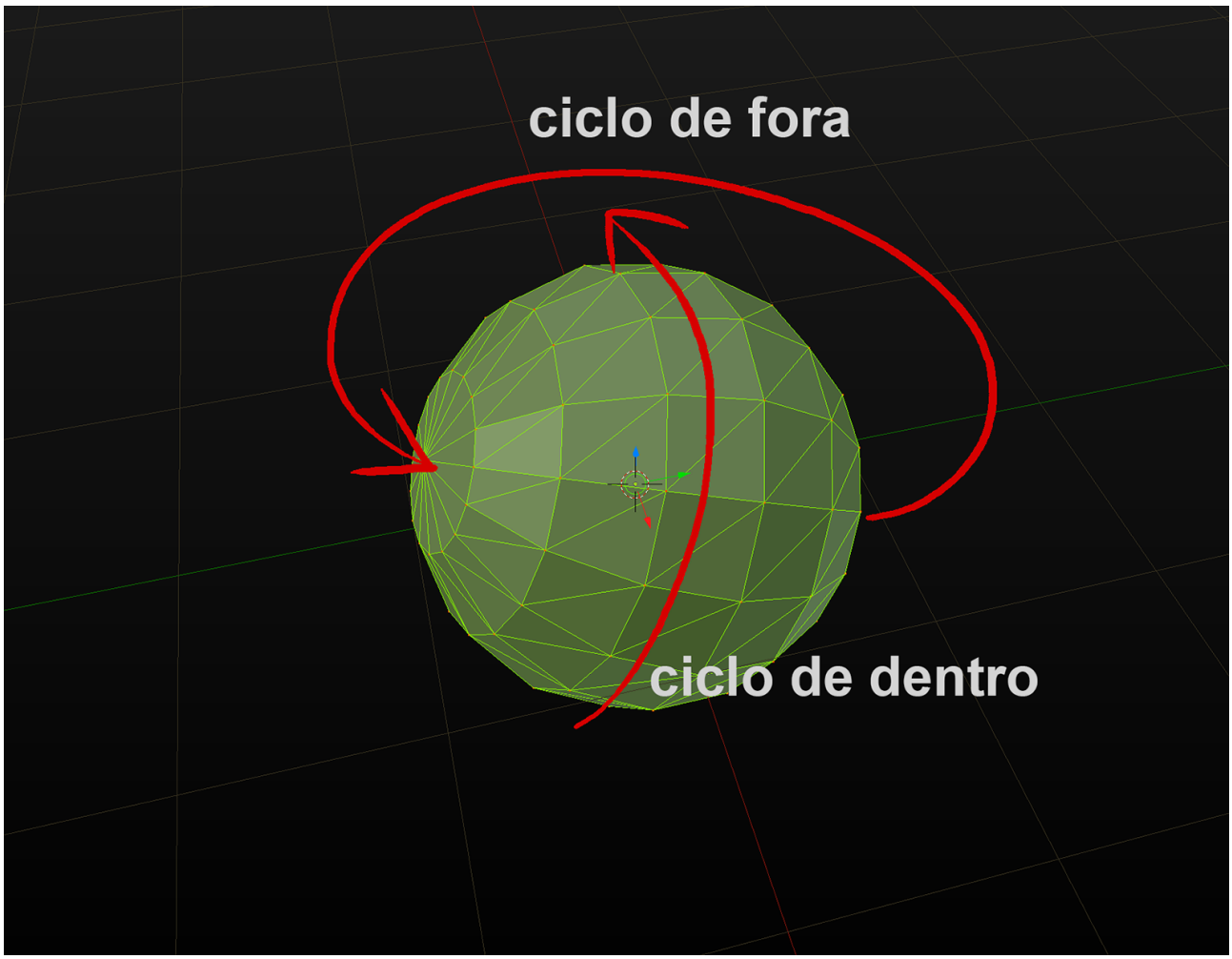


Figura 2.3: Plano de desenho da Esfera.

2.4 Cone

A função `generateCone` é semelhante à `generateSphere` estruturalmente, exceto que necessita de um caso especial para a base e as distancias dos *loops* de vértices ao eixo dos **yy** são lineares, não dependendo de funções trigonométricas.

O ciclo da base é semelhante ao das *stacks*, mas apenas necessita de gerar um triângulo em cada iteração, sendo um dos vértices do mesmo o centro da base.

Não foi necessário um caso especial para o topo visto que, como a distancia do último *loop* de vértices ao eixo **yy** é zero, os vértices gerados convergem formando um só vértice.

Capítulo 3

Motor de Renderização

O motor é o **executável gerado que desenha uma cena com base num ficheiro** `XML` que lhe é fornecido como parâmetro de entrada.

É este motor que necessita o formato com que é exportado para ficheiro cada modelo gerado pelo gerador. Decidimos que o formato seria **uma linha com o número de vértices seguida de outra linha com todos as coordenadas de todos os vértices** seguidos, separados por espaços.

Isto permite ao motor, aquando do *parsing* do ficheiro `XML` , assistido pela livraria `tinyxml2`, ao encontrar uma *tag* de modelo, criar uma instância de componente de modelo, essencialmente um *array* com todas as coordenadas seguidas lidas do ficheiro, para adicionar a um vetor na instância global da classe principal do motor, a `SceneTree`.

O método de `renderTree()` desta instância é chamado durante a função de desenho da cena e essencialmente atravessa todos estes componentes no vetor e pede-lhes para se desenharem.

Tal ocorre atravessando os *arrays* de coordenadas e desenhando um triângulo a cada nove vértices.

O motor permite rodar a câmara utilizando as teclas de seta e mudar entre modos de desenho com preenchimento, apenas pontos ou apenas linhas com o primeiro botão do rato.

Capítulo 4

Gerador

O gerador aceita parametros tais como especificados no enunciado, requer um parametro que indica a primitiva a gerar seguida das suas dimensões e divisões desejadas e finalmente um caminho para um ficheiro onde escrever o modelo.

Tal é implementado através dum muito simples interpretador que procura estes parâmetros das primitivas.