Primitivas Gráficas
Unidade Curricular de Computação Gráfica

Licenciatura em Ciências da Computação
Universidade do Minho

Manuel Marques (A85328) | Pedro Oliveira (A86328) | João Rodrigues (A84505) | Eduardo Pereira (A84098)

5 de Março de 2020

Índice

1 Contextualização **2**
1.1 Enunciado . . . 2
1.2 Resumo . . . 3

2 Apresentação das soluções **4**
2.1 Generator . . . 4
2.2 Definição dos pontos . . . 5
2.3 Ficheiro xml . . . 7
2.4 Engine . . . 8

3 Conclusão **10**

Índices de Figuras

2.1 Definição dos pontos do lado visivel . . . 5
2.2 Definição dos pontos do lado não visivel . . . 5
2.3 Definição dos pontos para a caixa . . . 6
2.4 Definição dos pontos para a esfera . . . 6
2.5 Definição dos pontos para o cone . . . 7
2.6 Informação da cena no ficheiro xml . . . 7
2.7 Resultado do box.3d . . . 8
2.8 Resultado do plane.3d . . . 8
2.9 Resultado do cone.3d . . . 8
2.10 Resultado do sphere.3d . . . 9
2.11 Resultado com todos os modelos desenhados . . . 9

Capítulo 1

Contextualização

1.1 Enunciado

Phase 1 – Graphical primitives

This phase requires two applications: one to generate files with the models information (in this phase only generate the vertices for the model) and the engine itself which will read a configuration file, written in XML, and display the models.

To create the model files an application (independent from the engine) will receive as parameters the graphical primitive's type, other parameters required for the model creation, and the destination file where the vertices will be stored.

In this phase the following graphical primitives are required:

- Plane (a square in the XZ plane, centred in the origin, made with 2 triangles)
- Box (requires X, Y and Z dimensions, and optionally the number of divisions)
- Sphere (requires radius, slices and stacks)
- Cone (requires bottom radius, height, slices and stacks)

For instance, if we wanted to create a sphere with radius 1, 10 slices, 10 stacks, and store the resulting vertices in file sphere.3d, and assuming our application is called *generator*, we could write on a terminal:

```
C:\>generator sphere 1 10 10 sphere.3d
```

The file format should be defined by the students and can support additional information to assist the reading process, for example, the file may contain a line at the beginning stating the number of vertices it contains.

Afterwards, the engine will receive a configuration file, written in XML. In this phase the XML file will contain only the indication of which previously generated files to load.

Example of a XML configuration file for phase one:

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

1.2 Resumo

Tendo em conta o problema apresentado, criámos dois executáveis, *Generator* e *Engine*. O *Generator* recebe os argumentos apresentados no enunciado e cria os ficheiros com os pontos que servirão de referência para o GLUT desenhar as figuras pedidas.
O *Engine* começa por ler o ficheiro xml passado como argumento e faz o parsing dos dados necessários para a criação da *cena*. Esta leitura é feita apenas uma vez, pois, guardamos a informação dos modelos numa estrutura de dados, o que aumenta aumenta significativamente o desempenho geral.

Capítulo 2

Apresentação das soluções

2.1 Generator

Este programa ao ser executado(passando os argumentos necessários), vai gerar ficheiros que contêm os pontos necessários para desenhar as figuras passadas como parâmetro.
Exemplo:
`./Generator plane 4 plane.3d` vai gerar um ficheiro *plane.3d* com os pontos necessários para formar um plano de tamanho 4×4.

Em seguida apresentam-se os comandos para gerar os ficheiros paras as restantes figuras.

```
./Generator box 4 4 4 4 box.3d
./Generator cone 1 4 10 10 cone.3d
./Generator sphere 1 10 10 sphere.3d
```

2.2 Definição dos pontos

Para a sequência correcta de pontos, é necessário ter em atenção o lado para os quais queremos que a figura seja desenhada, *"CW - Clock Wise* ou *"CCW - Counter Clock Wise*, pois, se toda a sequência de pontos fosse apenas num sentido, ao rodar um objecto, por exemplo, o plano de trás não seria visivel.
Apresenta-se, abaixo, a forma como elaboramos a definição dos pontos para as diferentes figuras. Na figura 2.1 os pontos são definidos em

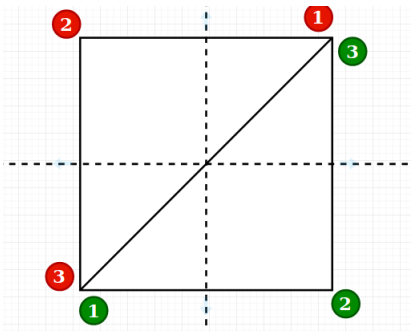

Figura 2.1: Definição dos pontos do lado visivel

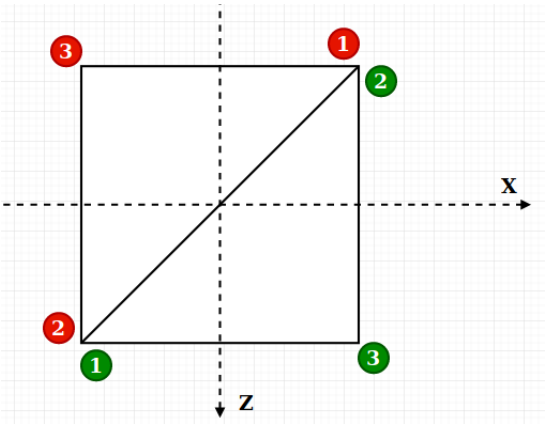

Figura 2.2: Definição dos pontos do lado não visivel

"CCW" ou Regra da mão direita.
Na figura 2.2, pelo contrário, são definidos em *"CW"* ou Regra da mão esquerda.

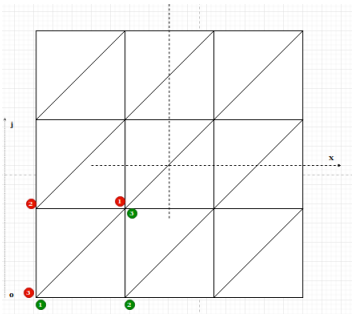

Figura 2.3: Definição dos pontos para a caixa

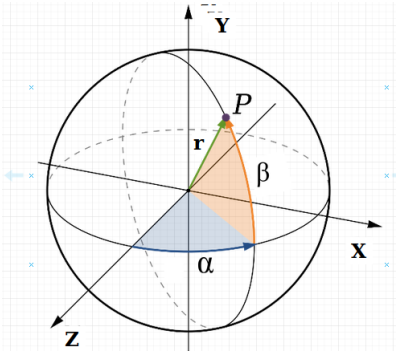

Figura 2.4: Definição dos pontos para a esfera

Na figura 2.3 podemos analisar a forma como é desenhada a caixa. Para cada plano da caixa, e tendo em conta o número de divisões da caixa, vai-se desenhando quadrados mais pequenos de dimensão d em que $d = tamanho \div divisoes$.
Já na figura 2.4, consideramos dois ângulos, α e β em que α é o ângulo horizontal e β o ângulo vertical que vão sendo aumentados gradualmente conforme o número de stacks e slices, compondo a esfera.

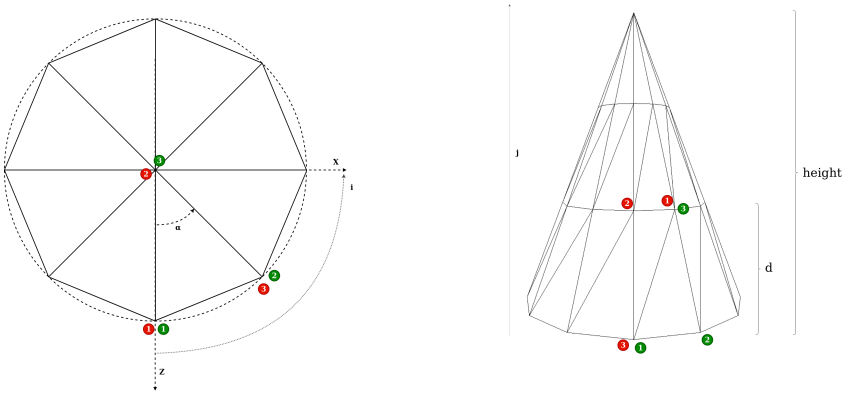

Figura 2.5: Definição dos pontos para o cone

Na figura 2.5 começamos por desenhar a base recorrendo a um ângulo α definido conforme o número de slices para desenhar a circunferência da base. Para desenhar o cone propriamente dito, diminui-se o raio para que a largura do cone diminua gradualmente à medida que vai aumentando a altura.

2.3 Ficheiro xml

Este ficheiro é passado como argumento aquando da invocação do *Engine* e contém a informação da cena que será posteriormente desenhada. Para efeito de apresentação, criou-se um ficheiro *demo.xml* cuja informação contida é a seguinte:

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

Figura 2.6: Informação da cena no ficheiro xml

2.4 Engine

Este programa é o motor que irá processar e desenhar toda a cena. Se o ficheiro *demo.xml* contivesse apenas cada uma das quatro linhas sempre que fosse lido, com os ficheiros que foram gerados previamente através do *Generator*, o resultado seria o seguinte:

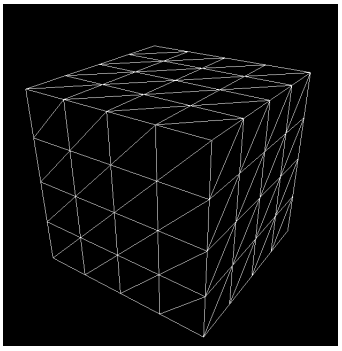

Figura 2.7: Resultado do box.3d

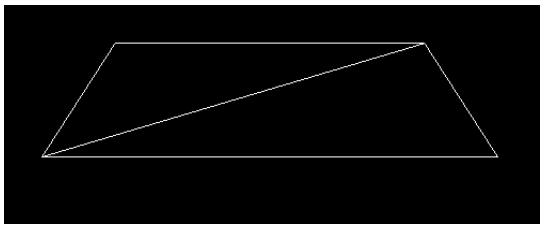

Figura 2.8: Resultado do plane.3d

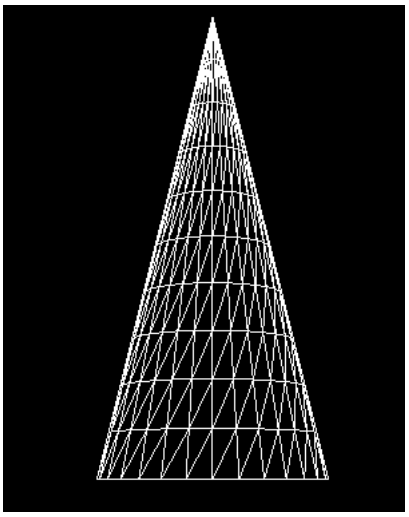

Figura 2.9: Resultado do cone.3d

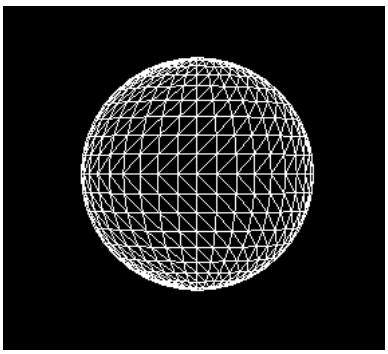

Figura 2.10: Resultado do sphere.3d

No entanto, como a cena é desenhada com todos os modelos pedidos, o resultado final é o que se mostra de seguida.

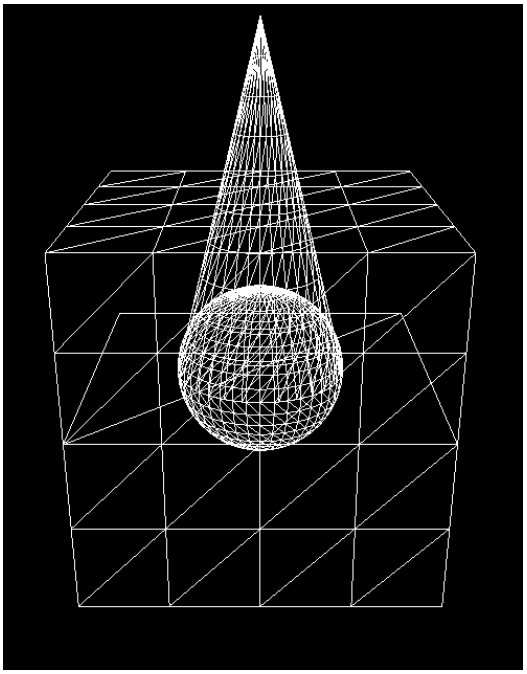

Figura 2.11: Resultado com todos os modelos desenhados

Capítulo 3

Conclusão

A realização desta primeira fase do projecto, permitiu uma introdução ao mecanismo da computação gráfica, nomeadamente a forma como são desenhados os objectos e cuidados a ter para a correcta implementação desse desenho.
Um dos aspectos que gostariamos de ver melhorado é a duplicação de pontos. Acreditamos que com uma estrutura de dados que guardasse a sequência de conexão dos pontos três a três, conseguiriamos ter apenas cada um dos pontos definido e seria simplesmente usar as sequências para desenhar os triângulos. Tentaremos melhorar esse aspecto numa próxima fase.