Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Computação Gráfica

Ano Letivo de 2022/2023

Grupo 24

André Castro Alves (a95033) Cláudio Alexandre Freitas Bessa (a97063)
José Fernando Monteiro Martins (a97903)

8 de março de 2023

Indice

1 Introdução

O documento apresentado visa apresentar as metodologias praticadas e processos utilizados durante a primeira fase do projeto que tem vindo a ser realizado no âmbito da UC de Computação Gráfica.

Nesta primeira fase, é solicitado a criação de duas aplicações, uma *engine* e um *generator*. O *generator* possui como objetivo, gerar as primitivas geométricas, que iremos mencionar posteriormente, e guardar as informações das mesmas através dos seus pontos e triâgulos em ficheiros com a extensão **.3d**. Após a geração deste ficheiro através de um **Makefile**, a *engine* tem como objetivo, interpretar esses ficheiros por meio do um ficheiro de configuração **XML**.

De forma a cumprir os objetivos desta primeira fase, as primitivas criadas foram:

- Plane (a square in the XZ plane, centred in the origin, subdivided in both X and Z directions
- Box (requires dimension, and the number of divisions per edge, centred in the origin)
- Sphere (requires radius, slices and stacks, , centred in the origin)
- Cone (requires bottom radius, height, slices and stacks, the bottom of the cone should be on the XZ plane)

2 Descrição das Aplicações

2.1 Generator

Tal como mencionado previamente, o **generator** tem como objetivo gerar, i.e. criar, cada primitiva geométrica. Para tal são criadas classes partilhadas ou classes abstratas para uma classe denominada de **Model**. As classes partilhadas são a classe **Point3D** que não é só partilhada entre as classes do **generator**, mas sim de igual forma com a **engine**. Seguidamente temos a classe partilhada **Triangle** que tal como o nome indica faz a parte de gerar os triângulos utilizando os pontos provenientes da classe partilhada **Point3D**. Esta class **Triangle** é utilizada em todas as classes responsáveis por criar as diferentes primitivas, uma vez que, tal como um dos fundamentos de Computação Gráfica, tudo é construído seguindo um conjunto de triângulos. Por fim, as outras classes responsáveis pelas primitivas, e.g. **Plane**, **Box**, **Sphere** e **Cone**, funcionam como objetos abstratos para serem reutilizados na classe **Model** que irá gerar os ficheiros de *output* com a ajuda do **Makefile** providenciado na diretoria do **generator** e executando na *bash* os comandos definidos no enunciado do trabalho prático.

Estes ficheiros **.3d** irão ser gerados na mesma diretoria do **Makefile** e sendo opcionalmente necessário movê-los para a diretoria *models* presente na aplicação do **engine**. Nestes ficheiros é visível ao longo de todo o mesmo um conjunto de *strings* subdivididas sobre espaços. Na primeira linha está presente, respetivamente, o número de *slices*, número de *stacks* e o raio, que por defeito é 0, que irá ser utilizado para questão relacionadas com texturas/sombras futuramente. Seguidamente é visível as coordenadas de todos os pontos existentes e essenciais para a criação dessa primitiva e por fim o conjunto de índices para a criação dos triângulos. Esses índices definem onde está determinado ponto do triângulo no ficheiro.

2.2 Engine

Já a aplicação **engine** é responsável pela leitura dos ficheiros gerados *a priori* pelo **generator** utilizando como auxílio um ficheiro de configuração cirado em **XML**.

O **engine** foi construído tendo como base aceitar documentos **XML** cuja possua, respetivamente a seguinte notação: **world**, **window**, **camera position**, **lookAt**, **up**, **projection**, **group**, **models** e **model**. É de relembrar que nem todos os elementos são obrigatórios, tais como os relativos à **window** e definições da **camera** uma vez que possuem valores **default**. Outros elementos que possam estar dentro deste ficheiro irão ou ser ignorados ou levantaram erros.

```
<world>
        <window width="512" height="512"/>
        <camera>
                <position x="10" y="10" z="10"/>
                <lookAt x="0" y="0" z="0"/>
                <up x="0" y="1" z="0"/>
                <projection fov="60" near="1" far="1000"/>
        </camera>
        <group>
                <models>
                        <model file="../models/box.3d"></model>
                </models>
        </group>
</world>
```

Figura 2.1: XML Template

Devido a ser necessário um conjunto de detalhes é fornecido na diretoria do **engine** a pasta **config** com diversas configurações prontas a ser utilizadas.

3 Formulação das Primitivas

3.1 Plane

A primitiva do **plano** foi a que aparentava e realmente demonstrou ser a menos complexa uma vez apenas necessitar de 2 **triângulos**, resumindo assim em 4 **pontos**. Para a **gerar** estes **pontos** é utilizado a medida de comprimento dada, explorando assim metade desse valor fornecido como *input* em cada direção tendo como base o plano XZ. Assim, o plano fica centrado na origem.

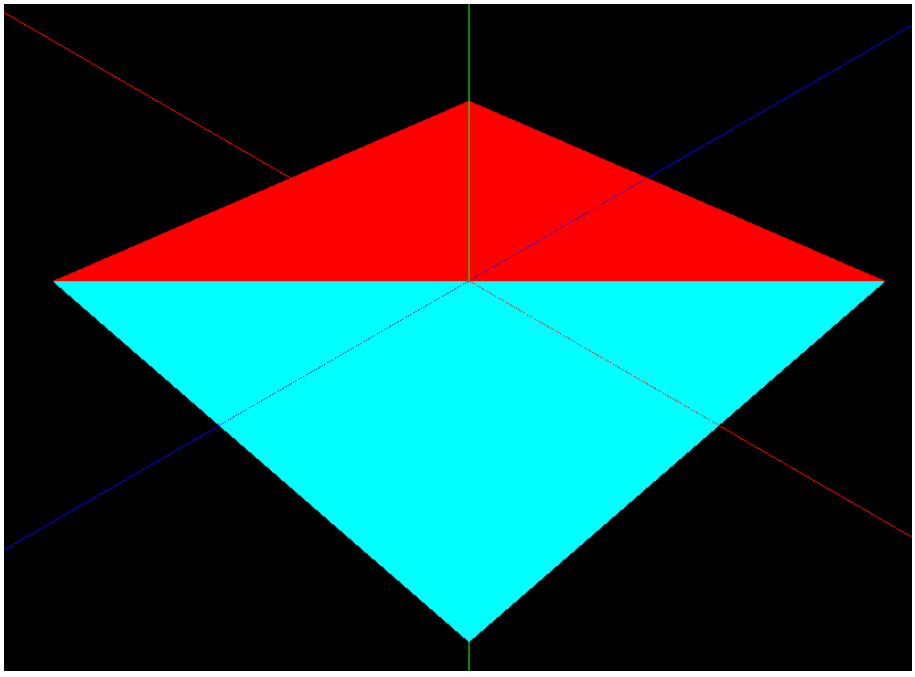

Figura 3.1: Primitiva do *Plane*

3.2 Box

Seguidamente, a primitiva da **caixa** é gerada a partir do valor de *input* do comprimento, limitando assim a possibilidade da caixa ser um **paralelipípedo**. Da mesma forma, centrada na origem e possuíndo agora 6 planos finitos de forma a definir cada face da primitiva. Necessitando assim de 12 **triângulos** e portanto 8 **pontos**, no mínimo, dependedo do número de *slices* e/ou *stacks*.

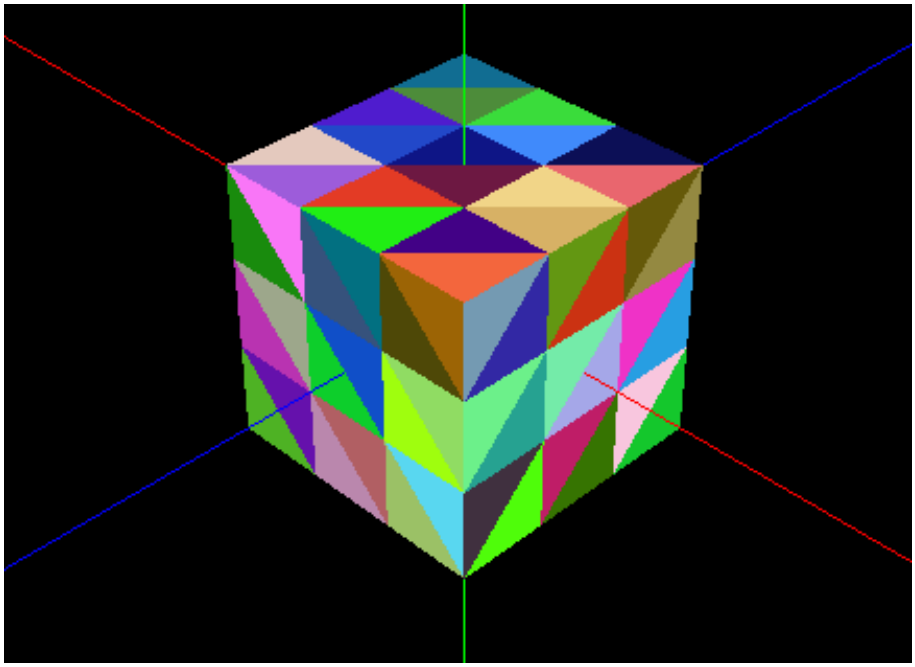

Figura 3.2: Primitiva da *Box*

3.3 Sphere

De forma a definir a **esfera** utilizamos, para além dos valores inseridos como *input*, coordenadas polares como intermédio para obter as finais coordenadas cartesianas para definir os pontos. Utilizamos prinicipalmente 3 fórmulas de forma a conseguir definir cada coordenada segundo os diferentes eixos.

- $y = radius * cos(\theta)$
- $x = cos(\alpha) * sin(\theta) * radius$
- $z = -sin(\alpha) * sin(\theta) * radius$

As fórmulas acima descritas utilizam θ e α onde, respetivamente, definem os ângulos formados com o eixo Y e o plano horizontal, plano XZ. O ângulo vertical seguindo os ponteiros do relógio **CW** e o outro ângulo seguindo o inverso **CCW**.

Posteriormente é necessário, tal como no **cone** definir os **triângulos** aos pares de forma a obter a visualização de **quadrados** para reutilizar os **pontos**. Apenas possúi as excessões dos seus polos que é necessário efetuar a construção de uma **GL-TRIANGLES-FAN** para **fixar** o **ponto central** de cada "base".

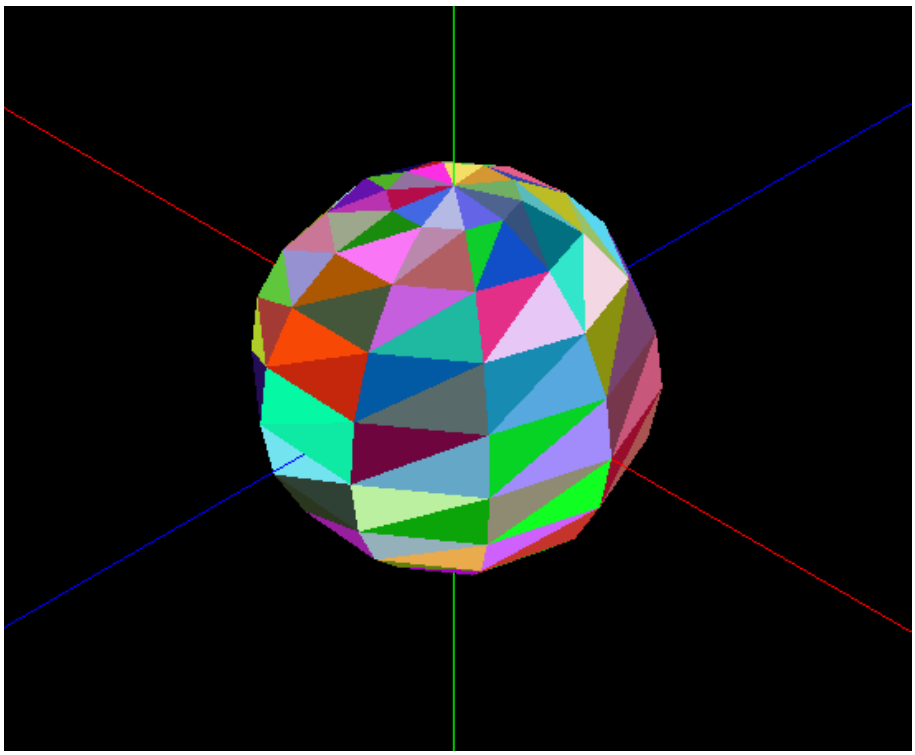

Figura 3.3: Primitiva da *Sphere*

3.4 Cone

O **cone** foi a primitiva que pdemonstrou um grau de complexidade superior uma vez que é a única figura que não é simétrica em todos os eixos, tal como a **esfera**, **cubo** ou **plano**.

De modo a facilitar o processo de construção do **cone** recorremos a uma tratégia **top-down** onde iniciaríamos pelo **vértice** do mesmo até à sua base, percorrendo cada **stack**. Devido a isso foram utilizadas as seguintes fórmulas com a ajuda do **raio** fornecida pelo utilizador no *input*.

- $height - incr = height/stacks$
- $y = height - incr * stack$
- $stack - radius = ((height - y) * radius)/height$
- $x = cos(\alpha) * stack - radius$
- $z = -sin(\alpha) * stack - radius$

Para criar os **triângulos** apenas utilizamos a mesma estratégia da **esfera**, criando-os aos pares de forma a criar os **quadrados**.

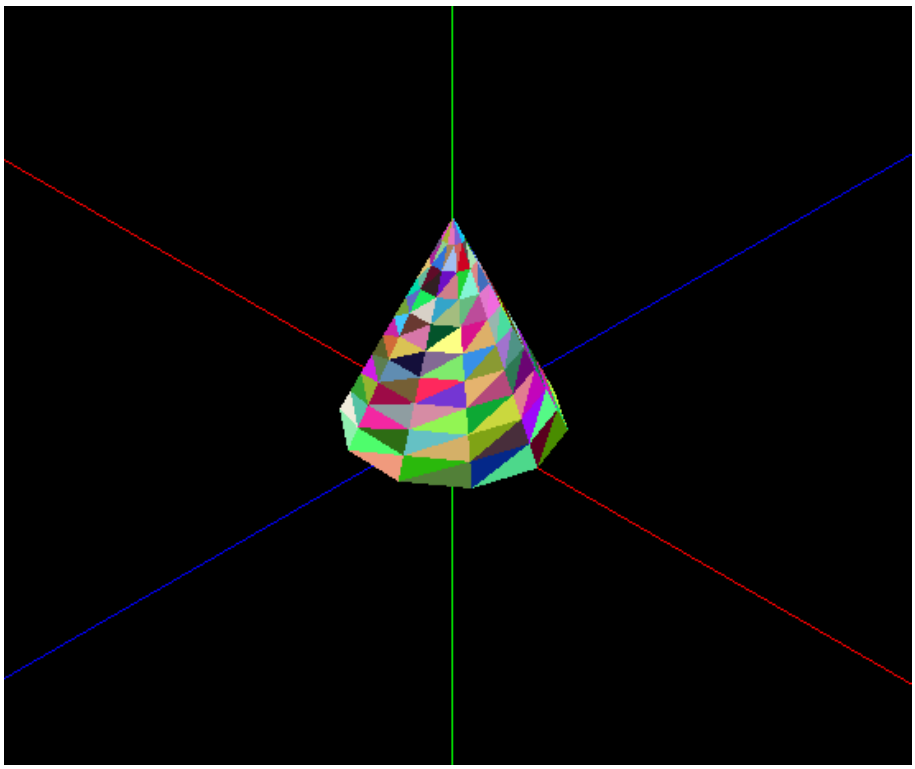

Figura 3.4: Primitiva do *Cone*

4 Transformações

De modo a dinamizar a interação com os modelos gerados, criamos um conjunto de transformações bastante simples que são executadas quando é efetuado um *click* em determinada *key*. Seguidamente identificamos essas mesmas *keys* e o evento que tais ativam, sendo que todos os eventos utilizam o mesmo valor de 0.1 unidades quer para somar ou multiplicar ao atributo em questão ou simplemente trocar o nome da variável.

4.1 Modelo

4.1.1 Modo de Desenho

- *Key* '**P**' -> Define o seu estilo de desenho para **point**
- *Key* '**L**' -> Define o seu estilo de desenho para **line**
- *Key* '**F**' -> Define o seu estilo de desenho para **fill**

5 Conclusão

Em suma, chegando ao periodo de entrega desta primeira etapa do trabalho prático, comprovamos e aplicamos os conhecimentos lecionados e praticados durante as aulas teóricas e teórico-práticas. Conseguimos aperfeiçoar, e para os que não tinham utilizado obter, o conhecimento sobre C++.

É assim verificável, os diferentes parâmetros e processos que tivemos em atenção para o processo de desenvolvimento desta fase introdutória. A criação do **generator** facultou-nos um maior conhecimento sobre algumas funções já existentes no **glut** como no caso dos métodos **glWire**. Após isso apenas necessitamos de recorrer ao *converter* de **XML** em uma estrutura de dados para a sua utilização tal e qual ao lecionado durante as aulas práticas. Optamos por uma forma adaptável para uma futura total abstração entre modelos sendo apenas visível inicialmente essa abstração na parte do **generator**.