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)

11 de abril de 2023

Indice

1 Introdução

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

Nesta segunda fase, é solicitado a implementação de *scenes* de forma hierarquica, utilizando transformações geométricas. Através de um conjunto de nodos com um conjunto de transformações definidas nos ficheiros de configuração, e.g. **XML**. Cada nodo apenas pode conter uma transformação de cada tipo, i.e. uma translação, uma rotação e uma operação de redimensão. Contudo, esses nodos referem-se não necessariamente a todos os modelos existentes nesse ambiente, daí a uma estrutura hierarquica para existência de transformações diferentes para cada grupo da árvore.

De forma a cumprir os objetivos desta fase e adiantando alguns requisitos da próxima fase foi implementado:

- Apenas podem existir transformação geométricas dentro de um grupo
- Transformações são aplicados a todos os modelos e submodelos
- Criação de um sistema solar
- Torus
- VBOs

2 Descrição das Aplicações

2.1 Generator

Apesar de não ser obrigatório, foi acrescentado a primitiva da **Torus** de modo a obter um sistema solar mais parecido à realidade. Tirando esse facto, nenhuma das outras primitivas ou qualquer outro componente do **generator** foi alterado de modo a manter-se a integridade do código fornecido na fase anterior.

2.2 Engine

Para o melhor desenvolvimento e abstração de dados foram criadas classes. De modo a seguir a normalização prévia do **generator**, onde conseguimos gerar modelos utilizando a abstração do **Model**, no **engine** foi implementada a class **Figure**.

Da mesma forma foram implementadas classes para ajudar no processo de automatização de leitura de ficheiros de configuração ou para uma melhoria na eficiência dos mesmos. Esses ficheiros são respetivamente, **Camera**, **Group** e **VBO** que serão mencionados posteriormente.

Na continuação do esclarecimento sobre o **engine**, é visível, agora, uma nova diretoria titulado de **transformations** onde nela estão visíveis todas as possíveis transformações que são as seguintes:

- **Translação** -> recebe em 3 *floats* as coordenadas, 0 em caso de omissão
- **Rotação** -> recebe em 3 *floats* as coordenadas, 0 em caso de omissão e o ângulo a efetuar
- **Escalanamento** -> recebe em 3 *floats* as coordenadas, 0 em caso de omissão

De maneira a tornar abstrata essas transformações é criado um *header* com o nome de **Transform** onde é aplicada todas as transformações da mesma forma, fazendo assim com que todas as transformações sejam herdadas desta classe principal.

Adicionalmente, estão disponiveis os seguintes ficheiros **XML** na pasta de *config* no **engine**:

- **box.xml** -> uma *box* sem transformações
- **box-circle.xml** -> um circulo feito com *box's*
- **composed.xml** -> pilha de modelos
- **cone.xml** -> um *cone* sem transformações
- **config.xml** -> um *plane* e uma *sphere* sem transformações
- **plane.xml** -> um *plane* sem transformações
- **plane-sphere.xml** -> um *plane* e uma *sphere* sem transformações
- **single-transformation.xml** -> um modelo de uma *box*
- **snowman.xml** -> um boneco de neve
- **sphere.xml** -> uma *sphere* sem transformações
- **solar-system.xml** -> um sistema solar (requisito)
- **torus.xml** -> uma *torus* sem transformações

3 Formulação das Primitivas

3.1 Torus

De forma a definir o **Torus**, tivemos de pensar de duas formas diferentes. Sendo que sabiamos que poderia ser um **torus** convencional, ou seja com volume, ou um **torus** sem volume, como por exemplo no anel de saturno, sendo que este caso teria de ser tratado, já que teriamos de fazer pelo menos um para integrar o sistema solar. No caso do **torus** ter volume utilizamos 3 fórmulas que nos permitiram calcular a coordenada do proximo ponto nos 3 eixos.

- distHorizontal = $thickness * cos(sideAngle)$;
- x = $(innerRadius + distHorizontal) * cos(stackAngle)$;
- y = $thickness * sin(sideAngle)$;
- z = $(innerRadius + distHorizontal) * sin(stackAngle)$;

Dentro destes, a *thickness* refere-se à distância entre o raio interior e exterior do **torus**, o *sideAngle* e o *stackAngle* variam a cada iteração por cálculos interinos à função, de forma a definir os próximos pontos corretamente. Por fim o *innerRadius* é, como o próprio nome indica o raio do circulo interior do **torus**. Após efetuar estes cálculos, colocamos o vérice correspondente no vetor de vertices e usamos uma função auxiliar *addSquare* para colocar os quadrados correspondentes no vetor.

No caso do **torus** não ter volume, apenas usamos a *stack* e a *stackAngles* para gerar o **torus**. À semelhança do anterior temos de fazer alguns calculos de forma a determinar as coordenadas.

- xNearCenter = $innerRadius * cos(stacksAngle)$;
- xFarCenter = $(innerRadius + thickness) * cos(stacksAngle)$;
- zNearCenter = $innerRadius * sin(stacksAngle)$;
- zFarCenter = $(innerRadius + thickness) * sin(stacksAngle)$;

Realizamos por isso um **torus** que não tem dimensão no eixo dos y, tornando-se assim bidemensional.

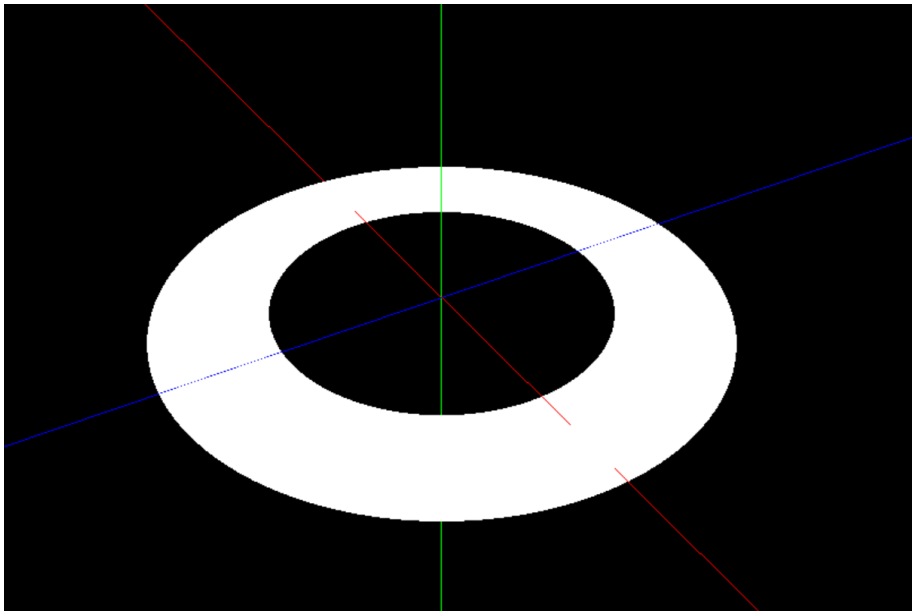

Figura 3.1: Primitiva da *Torus*

4 Câmara em 3ª Pessoa

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 da variável *boost* que pode ser aumentado consoante o pretendido quer para somar ou multiplicar ao atributo em questão ou simplemente trocar o nome da variável.

- Key '**O**' -> Visualização dos eixos
- Key '**F**' -> Visualização do modelo com **FILL** ou **LINE**
- Key '**A**' -> Rodar para a esquerda
- Key '**D**' -> Rodar para a direita
- Key '+' -> Zoom in
- Key '**-**' -> Zoom out
- Key '**MOUSE1**' -> Movimentação na horizontal
- Key '**MOUSE2**' -> Movimentação na vertical
- Key '**ARROW-UP**' -> Movimentação positiva no eixo dos X
- Key '**ARROW-DOWN**' -> Movimentação negativa no eixo dos X
- Key '**ARROW-LEFT**' -> Movimentação negativa no eixo dos Z
- Key '**ARROW-RIGHT**' -> Movimentação positiva no eixo dos Z

5 VBOs

Visto que esta fase não apresentava grandes precalços optamos por implementar já os **VBOs**. Desta forma, seria possível melhorarmos a *performance* para cenários mais complexos, i.e cenários mais complexos dos utilizados até ao momento.

Esta implementação segue quase na totalidade a fornecidade e lecionada nas aulas práticas, i.e. existem 2 *buffers*, um que possui os valores dos pontos e outro dos seus índices. Assim também é guardado a sua dimensão em 2 outras variáveis. É também guardado o nome dos ficheiros em um *map*.

6 Sistema Solar

Sendo um dos requisitos a implementação de um ficheiro de configuração que representasse um sistema solar, sem utilizar texturas ainda, efetuamos essa implementação utilizando esferas para representar os planetas, obviamente, e utilizamos a *Torus* que foi implementada precisamente para este cenário. Tivemos algumas dificuldades na visualização completa do cenário uma vez que há um limite de renderização mas consideramos que vai de encontro ao solicitado.

Para a definição de cada planeta quer a nível da sua posição ou dimensão utilizamos respetivamente as escalas de **1:1MM** e **1:1.5MM** .

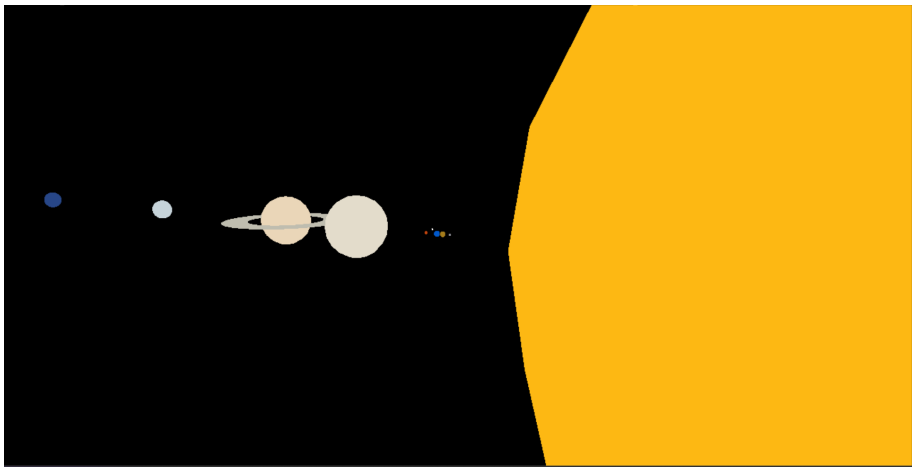

Figura 6.1: Sistema Solar

7 Apontamentos Extra

De modo a complementar algo que não foi referido na entrega anterior do trabalho decidimos acrescentar esta secção de modo a explicar melhor algumas decisões.

7.1 Pointers

Uma das maiores vantagens entre C++ e C é a gestão de memória. Ao contrário de C, não é necessário haver um cuidado manual por nossa parte sobre a memória de determinados objetos, e.g. *malloc*, *free*, em C++ já existem construtores e destrutores que atuam consoante objetos são criados ou "destruídos". Daí não utilizarmos *pointers* como em C mas sim **shared-pointers** onde conseguimos tratar dessa parte logística de memória de forma dinâmica.

8 Conclusão

Em suma, chegando ao periodo de entrega desta segunda etapa do trabalho prático, comprovamos e aplicamos os conhecimentos lecionados e praticados durante as aulas teóricas e teórico-práticas.

É assim verificável, os diferentes parâmetros e processos que tivemos em atenção para o processo de desenvolvimento desta fase mais intermédia. De um modo geral, esta fase não se demonstrou demasiado exigente, permitindo um avanço no desenvolvimento do projeto para além da fase 2.