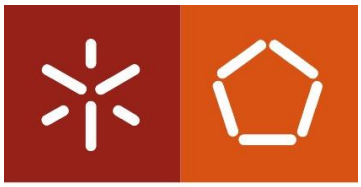

Universidade do Minho

Departamento de Informática

Programação Orientada a Objetos

Grupo Nº110

José Duarte Alves (A89563)

José Mário Peixoto (A89985)

28 de junho de 2021

Duarte

José

Capítulo 1

Introdução

O presente relatório visa apresentar a segunda fase do trabalho proposto no âmbito da Unidade Curricular de Computação Gráfica.

Esta fase é composta pelo aprimoramento das aplicações realizadas na fase anterior. Nesta fase, a Engine foi modificada de forma a que o conteúdo do xml fosse mais complexo, conseguindo assim construir primitivas associadas a transformações e adicionalmente cores para cada modelo. Ao Generator foi adicionada a criação de uma nova primitiva, o Torus.

As transformações a aplicar nesta fase, bem como a informação dada para a sua aplicação, estão a seguir descritas:

- Translação (recebe o vetor da translação a efetuar, na forma de 3 floats, as coordenadas não presentes assumirão o valor 0);
- Rotação (recebe o ângulo da rotação a efetuar, além disso recebe os eixos em que vai ser aplicado);
- Escalamento (recebe, na forma de 3 floats, a escala para cada coordenada, sendo assumido o valor 1 em caso de omissão).

Além disso, decidimos nesta fase criar uma câmara na terceira pessoa que facilitará a exploração do cenário criado pelo xml, o que levou a uma modificação do sistema de input. Implementámos ainda todos os desenhos através de VBOs com índices de modo a melhorar a performance de cenários computacionalmente mais pesados. Como forma de melhorar a nossa representação do sistema solar decidimos ainda adicionar a criação de um Torus ao Generator. Finalmente, decidimos modificar os raw pointers utilizados por smart pointers de modo a tirar partido dessa funcionalidade de C++.

Capítulo 2

Leitura do ficheiro em XML

2.1 Parsing

Foram criadas algumas classes de modo a facilitar a leitura e posterior desenho do cenário proveniente do xml. Essas classes são:

- Transform (classe simples que servirá como interface de modo a guardar um vetor de todas as transformações sem diferenciação).
- Translation (classe que implementa Transform e guarda a informação sobre uma translação).
- Rotation (classe que implementa Transform e guarda a informação sobre uma rotação).
- Scale (classe que implementa Transform e guarda a informação sobre uma escala).
- Figure (classe que contém os valores RGB de cada primitiva a desenhar e o nome do ficheiro a ler).
- Group (classe que se aproveita das classes anteriores e contém a informação sobre um grupo, contém um vetor de transformações a aplicar, um map onde strings identificam instancias da classe Figure a desenhar e um vetor da própria classe Group que contém os grupos filhos deste).

O parsing é feito através do uso de tinyxml2, recorrendo ao uso das funções getFirstChildElement() e NextSiblingElement(). Com estas chamadas conseguimos obter toda a informação necessária do XML sendo ela guardada nas classes acima faladas.

2.2 Desenho

Após ser enviada a informação de todos os grupos irmãos dentro de scene para um vetor da classe Group, temos todos os dados para poder desenhar o cenário.

Para o fazer basta iterar por cada group fazendo um pushMatrix no iníco e um popMatrix no fim, de modo a guardar e carregar estados anteriores, entre estas instruções basta aplicar as transformações e desenhar os modelos, chamando a função recursivamente para cada Group filho.

Adicionalmente, estão disponíveis os seguintes ficheiros XML já previamente gerados:

- "sphere.xml": contém uma esfera.
- "box.xml": contém uma caixa.
- "cone.xml": contém um cone.
- "plane.xml": contém um plano.
- "torus.xml": contém um torus.
- "staticSolarSystem.xml": contém um sistema solar estático.
- "snowman.xml": contém um boneco de neve.
- "angel.xml": contém um anjo.

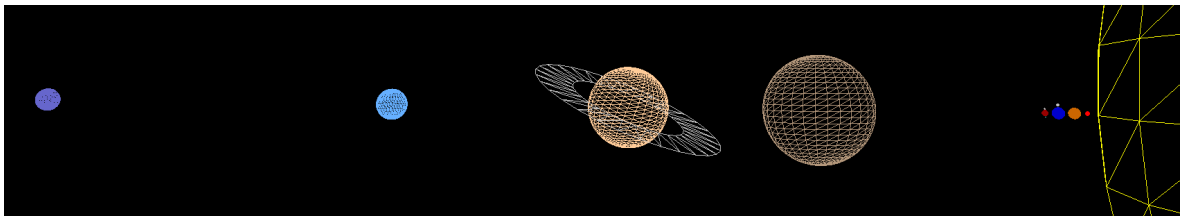

Figura 2.1: Sistema solar estático

Capítulo 3

Torus

O Torus criado no Generator para esta fase tem como argumentos: o raio do torus a criar (widenessRadius), o raio que define a grossura do torus (thicknessRadius), o número de anéis a criar, e por fim o número de lados de cada um deles. Sendo θ o ângulo do anel atual e α o ângulo do lado, conseguimos obter os pontos do torus através das seguintes fórmulas:

$$y = thicknessRadius * cos(\alpha)$$

$$distHorizontal = thicknessRadius * sin(\alpha)$$

$$x = (widenessRadius - distHorizontal) * sin(\theta)$$

$$z = (widenessRadius - distHorizontal) * cos(\theta)$$

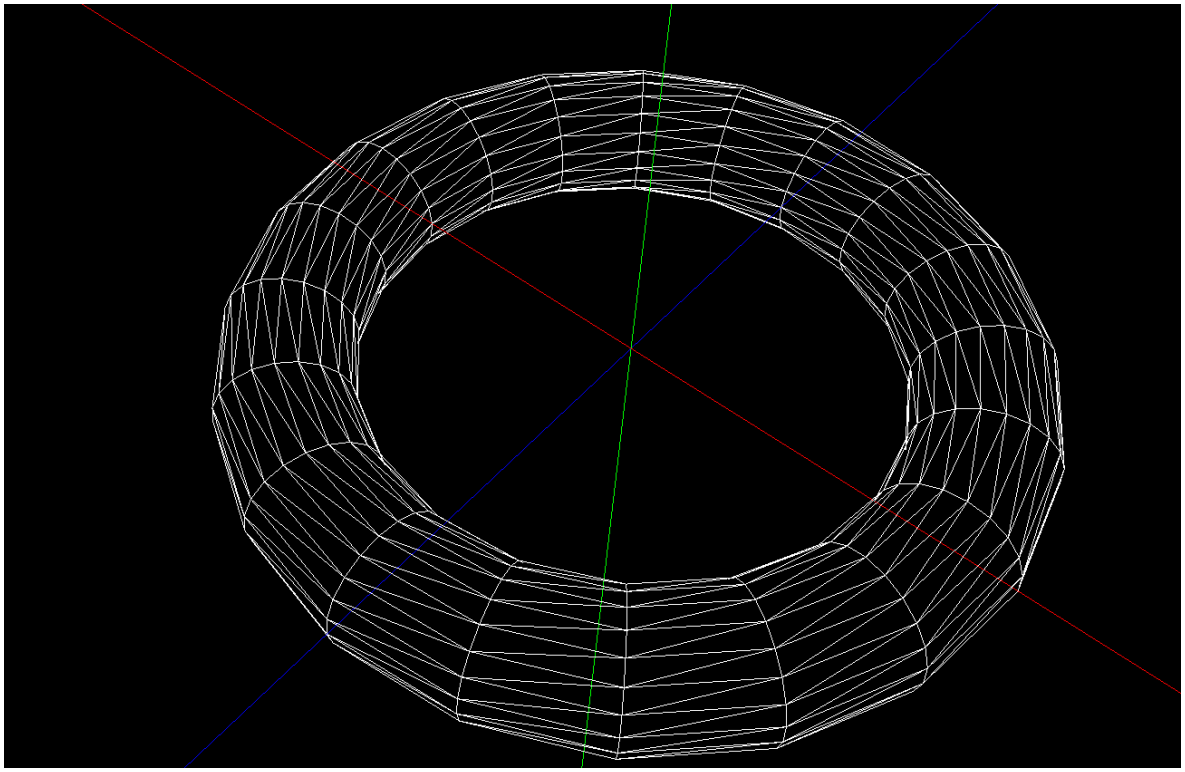

Figura 3.1: Torus

Capítulo 4

Câmara 3ª pessoa

Como a duração da Fase 2 era extensa, decidimos aproveitar o tempo restante a implementar funcionalidades extra. Uma delas foi a adição de uma câmara manipulável com as teclas WASD e com a qual podemos usar o rato para olhar em volta.

A implementação desta funcionalidade tira partido da localização do rato no ecrã, Sabendo a posição anterior e calculando o seu offset. A partir deste e da sensibilidade escolhida, é calculado o ângulo a virar a câmara. A movimentação foi feita tendo em conta o ponto para o qual o utilizador está a olhar, normalizando o vetor (teclas 'W' e 'S') e fazendo um cross product com o eixo y (teclas 'A' ou 'D').

Além disso o Utilizador tem a possibilidade de definir a velocidade de movimentação para se deslocar mais depressa ou mais devagar com as teclas 'G' e 'F', respetivamente.

Capítulo 5

Modificação do sistema de input

Após um período de teste da nossa câmara, verificámos que o nosso sistema de input poderia estar melhor implementado se fosse feito com um array de estados de cada tecla. Os problemas que nos levaram a esta modificação foram:

- Não poder ter duas teclas pressionadas ao mesmo tempo;
- Movimentação pouco fluída;
- Pequeno delay quando uma tecla era segurada ("held") para a câmara andar nessa direção.

Com esse pensamento decidimos modificar as funções de input, de modo a que alterarem esse array de estados para true quando uma tecla fosse pressionada, sendo ele apenas passado a false assim que a tecla fosse largada. Adicionalmente é feita em cada chamada à função renderScene, no inicio da mesma, uma chamada à função que processa o input do utilizador. Esta função está responsável por verificar os key_states relevantes ao jogo e caso estejam a true execuar a ação dos mesmos.

Desta forma, ações contínuas, tais como movimentação, serão sempre tratadas na função de processamento de input. As ações instantâneas, como a ativação do modo wired são tratadas no momento em que se deteta que a tecla foi pressionada, por modo a evitar repetições da ativação, e que seja uma experiência mais intuitiva para o utilizador.

Capítulo 6

VBOs

Após a implementação da câmara, considerámos que o próximo passo seria aumentar a performance para cenários muito preenchidos, e para isso implementamos arrays de índices para melhorar a velocidade de desenho de cada frame.

Esta implementação foi feita de acordo com o lecionado nas aulas práticas, não sendo por isso feita aqui uma descrição muito profunda. Existe um mapa de strings que serão os nomes dos ficheiros para uma nova classe VBO, a qual possui 2 buffers. Um é preenchido com os valores dos pontos e o outro com os valores dos índices. Tem, além disto, o número de índices e de pontos.

Esta adição foi relativamente intuitiva, visto que na fase anterior tínhamos já guardado os pontos através de índices no ficheiro.3d.

Capítulo 7

Utilização de smart pointers

Na primeira fase do trabalho, este foi entregue sem ter em consideração memory leaks. Não queríamos incorrer no mesmo erro nesta fase, e para tal foram modificados todos os raw pointers, isto é, apontadores feitos como em C, para apontadores inteligentes.

Os apontadores inteligentes são apontadores que quando ficam "out of scope"no programa não apagam apenas o apontador, mas também a memória para a qual apontavam. Existem dois tipos principais de apontadores inteligentes: unique_ptr e shared_ptr.

No nosso trabalho recorremos maioritariamente ao uso de shared_ptr, pois é aquele que mais fielmente representa um apontador de C, podendo ser partilhado e copiado para outras variáveis ao contrário do unique_ptr.

Capítulo 8

Conclusão

Ao longo do presente relatório, deu-se a conhecer todas as dificuldades sentidas ao longo da elaboração desta segunda fase do projeto, assim como todas as funcionalidades extra implementadas.

De um modo geral, realização da segunda fase do trabalho não se mostrou demasiado exigente, sendo feita sem grandes precalços. Deste modo, foi possível a introdução da câmara na terceira pessoa, o uso de VBOs, assim como melhorias significativas na qualidade do código entregue.

O grupo considera que a entrega desta segunda fase cumpre com todos os requisitos presentes no enunciado, e que é seguramente uma base sólida para iniciar o trabalho na terceira fase.