Universidade do Minho

Escola de Engenharia

Computação Gráfica

Licenciatura em Engenharia Informática

Fase 2 - Geometric Transforms

Grupo 12

Ana Pires - [A96060]
Mariana Marques - [A93198]

Abril, 2023

Conteúdo

1. Introdução

Como o próprio nome indica, a segunda fase do trabalho prático tem como objetivo a implementação de transformações geométricas, tais como, *translação*, *rotação* e *escala*.

Para além das transformações geométricas, é agora admitida hierarquia entre nodos, onde os nodos "filhos"herdam as transformações geométricas do nodo "pai". Desta forma, é necessária a implementação de uma estrutura de dados que seja capaz de cumprir os requisitos.

Para isto ser possível, foi necessária uma reestruturação no código, nomeadamente, no **Engine** no que toca ao *parser* do ficheiro XML.

2. Primitivas Gráficas

Para a realização desta segunda fase, foram utilizadas as primitivas gráficas descritas na primeira fase, tais como:

1. *Plano*
2. *Caixa*
3. *Cone*
4. *Esfera*

De modo a aproximar-se a representação do sistema solar à realidade, acrescentamos uma nova figura ao *generator*: o **Torus**. Esta primitiva é utilizada para a representação do planeta Saturno, mais especificamente para o seu anel.

2.1 Torus

Para esta figura são necessários quatro parâmetros:

- ***Radius*** (R da figura 2.1 (b))- Distância entre o centro do Torus, até ao centro do tubo
- ***ringRadius*** (r da figura 2.1 (b)) - Raio da espessura do Torus
- ***Slices*** - Corte do Torus na vertical
- ***Stacks*** - Corte do Torus na horizontal

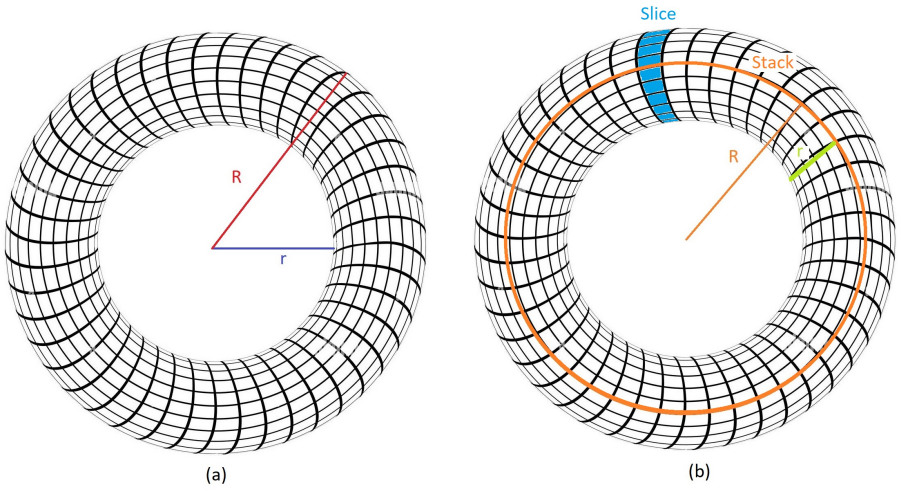

Figura 2.1: (a) Raio interior e raio exterior do torus (b) *Radius* e *ringRadius* do Torus

Para calcular o *Radius* e o *ringRadius*, foi necessário utilizar duas variáveis: o **raio_interior** (r da figura 2.1 (a)) e o **raio_exterior** (R da figura 2.1 (a)) do Torus.

Os cálculos efetuados foram, portanto:

- *Radius* = (raio_interior + raio_exterior) / 2.0f
- *ringRadius* = (raio_interior - raio_exterior) / 2.0f

Após estes valores calculados, itera-se as *slices* e as *stacks* através de dois ciclos.

3. Reestruturação do Projeto

Para ser possível cumprir os requisitos exigidos pela segunda fase, foi necessário realizar mudanças no projeto.

3.1 Estruturas de Dados

3.1.1 Window

Classe responsável por guardar a informação referente à janela, como a largura (*width*) e altura ((*height*)).

Para tal ser possível, foi utilizada a função **readWindow**.

3.1.2 Camera

Classe responsável por guardar informação referente à câmara, tal como:

- *Position* - Ponto responsável pela localização da câmara;
- *Up* - Ponto responsável pela direção da perspetiva da câmara;
- *LookAt* - Ponto responsável pela orientação da câmara;
- *Projection* - Ponto responsável pela escolha do tipo de perspetiva que será utilizada;

Para tal ser possível, foi utilizada a função **readCamera**.

3.1.3 Transform

Classe abstrata responsável apenas por uma transformação, que pode ser uma translação (*Classe Translation*), rotação (*Classe Rotation*) ou escala (*Classe Scale*).

Em relação à translação, após a herança dos atributos e métodos da classe mãe, é aplicada a função *glTranslatef* no método abstrato *doAction()*.

Em relação à rotação, para além da herança dos atributos e métodos da classe mãe, é adicionado um atributo extraordinário responsável pela rotação da primitiva (*angle*), tal como, é aplicada a função *glRotatef* no método abstrato *doAction()*.

Em relação à escala, após a herança dos atributos e métodos da classe mãe, é aplicada a função *glScalef* no método abstrato *doAction()*.

3.1.4 Transforms

Classe responsável por guardar a lista de transformações em questão.

Para tal ser possível, foi utilizada a função **readTransforms**.

3.1.5 Model

Classe responsável por guardar a informação referente às primitivas em questão, sendo cada modelo responsável por um modelo, como o nome do ficheiro (*file*) e os pontos que a compõem (*points*).

3.1.6 Models

Classe responsável por guardar a lista de modelos em questão.

Para tal ser possível, foi utilizada a função **readModels**.

3.1.7 Color

Classe responsável por guardar informação responsável pela coloração das primitivas através do modelo RGB, através da função *glColorf*.

- *R* Compreende um valor entre 0 e 255 e é responsável pela intensidade da cor vermelha.
- *G* Compreende um valor entre 0 e 255 e é responsável pela intensidade da cor verde.
- *B* Compreende um valor entre 0 e 255 e é responsável pela intensidade da cor azul.

3.1.8 Group

Classe responsável por guardar toda a informação referente a um grupo, tal como:

- *Transforms* - Lista de Transformações;
- *Models* - Lista de Modelos;
- *Groups* - Lista de sub-grupos;
- *Color* - Cor;

Desta forma, é com a variável que corresponde à lista de sub-grupos que é possível implementar hierarquia entre grupos e uma a aplicação correta das transformações entre os mesmos.

3.1.9 Tags

A classe **Tags** está responsável pela criação de uma estrutura de dados que captura toda a informação contida no XML em causa, tais como:

- *Window* - Informação referente à janela;
- *Camera* - Informação referente à câmara;
- *Groups* - A variável groups é apenas um grupo, que se trata do grupo principal, uma vez que este contêm todos os restantes grupos apresentados no ficheiro XML;

Para tal ser possível, foi utilizada a função **readGroups** que devolve o grupo principal, como foi referido anteriormente.

3.2 Engine

Como já foi referido anteriormente, é de salientar que o *Engine* sofreu alterações em relação à fase anterior, no que toca, principalmente, à leitura dos ficheiros XML e à utilização de diferentes estruturas de dados para permitir hierarquia entre grupos, através da função *readXML* que devolve uma estrutura de dados com toda a informação necessária.

Em relação à função **renderScene**, foram adicionadas novas funcionalidades, tais como:

- *moveCamera* - Atualiza a posição da câmara mediante o processamento dos inputs, recorrendo ao alpha, beta e radius;
- *drawAxis* - Permite a visualização dos eixos mediante o processamento dos inputs;
- *updateDrawMode* - Seleciona o método de renderização mediante o valor da variável *drawMode*;
- *drawGroup* - Função responsável pela renderização de todos os grupos com os respetivos modelos, transformações e cor;

Em relação à função **changeSize**, o método *gluLookAt* passa agora a utilizar variáveis contidas no ficheiro XML.

3.3 Generator

Em relação ao **Generator**, foi necessária a adição da primitiva *torus* que irá, posteriormente, ser usada no ficheiro XML do Sistema Solar.

3.4 Câmara

Para facilitar a visualização das figuras, foram implementados três modos de renderização, juntamente com a possibilidade de mover a câmara. As teclas associadas a cada opção são, então:

Modo renderização:

- **0** : Modo FILL - Apresentação preenchida.
- **1** : Modo LINE - Apresentação com apenas linhas.
- **2** : Modo POINT - Apresentação com apenas pontos.

Opções de movimento da câmara:

- **+,-** : Zoom In e Zoom Out.
- **.** (ponto): Apresentar/Retirar os eixos *x, y, z*.
- **KEY_UP,KEY_DOWN,KEY_LEFT,KEY_RIGHT** : Rotação da câmara em relação a cada eixo.

4. Sistema Solar

Numa primeira implementação do Sistema Solar, implementa-se o sol, os planetas e as luas de uma maneira simples.

4.1 Sol, Planetas e Luas

Por opção do grupo, define-se o sol, 8 planetas, dos quais, alguns com as principais luas (a lua da Terra, três luas de Júpiter (lua IO, lua Europa e lua Ganymede), duas de Saturno (lua Mimas e lua Titã), uma lua de Urano e uma lua de Neptuno), recorrendo à primitiva *esfera* cada uma com a sua respetiva cor.

É de salientar, Para a representação dos anéis de Saturno, recorreu-se à nova primitiva **Torus**.

De modo a facilitar a observação de todos os planetas, opta-se por não utilizar as dimensões do sol e as distâncias dos planetas, com a sua escala verdadeira.

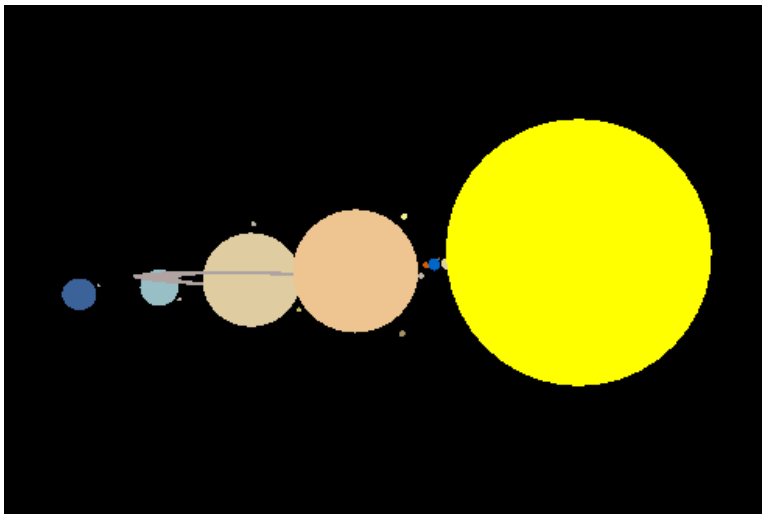

Figura 4.1: Planetas alinhados (sem rotações)

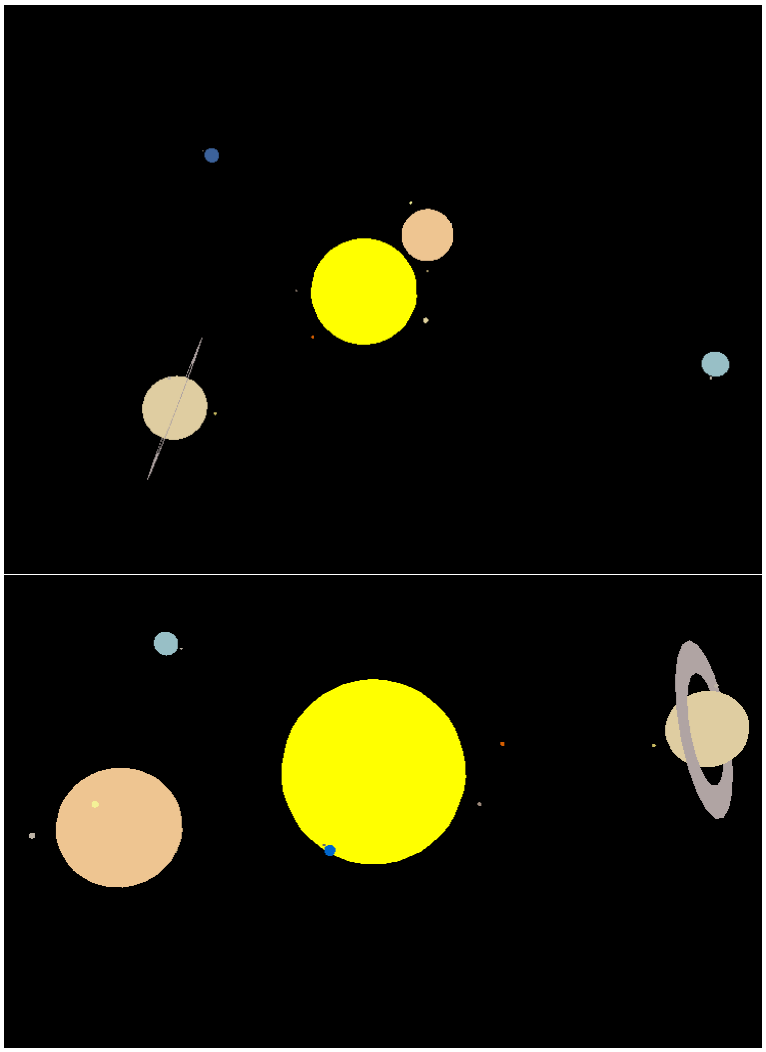

Figura 4.2: Sistema Solar Final

5. Testes

Ao utilizar os quatro testes fornecidos pelo docente, verificou-se que se realizaram todos com sucesso, como se confirma pela apresentação dos resultados obtidos, a seguir.

5.1 Teste 1 - Cubo

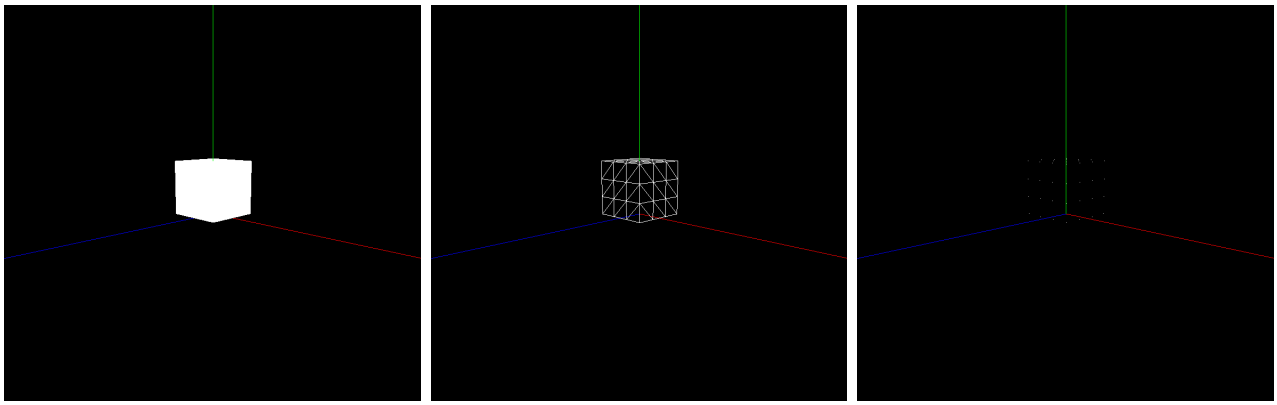

Figura 5.1: Cubo em modo Fill vs. Cubo em modo Line vs. Cubo em modo Point

5.2 Teste 2 - Figuras

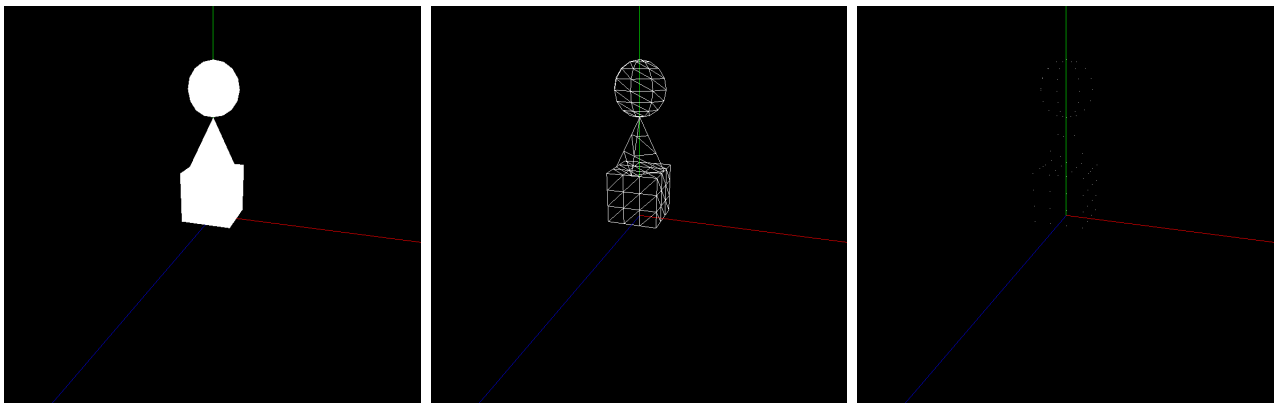

Figura 5.2: Figuras em modo Fill vs. Figuras em modo Line vs. Figuras em modo Point

5.3 Teste 3 - Boneco de neve

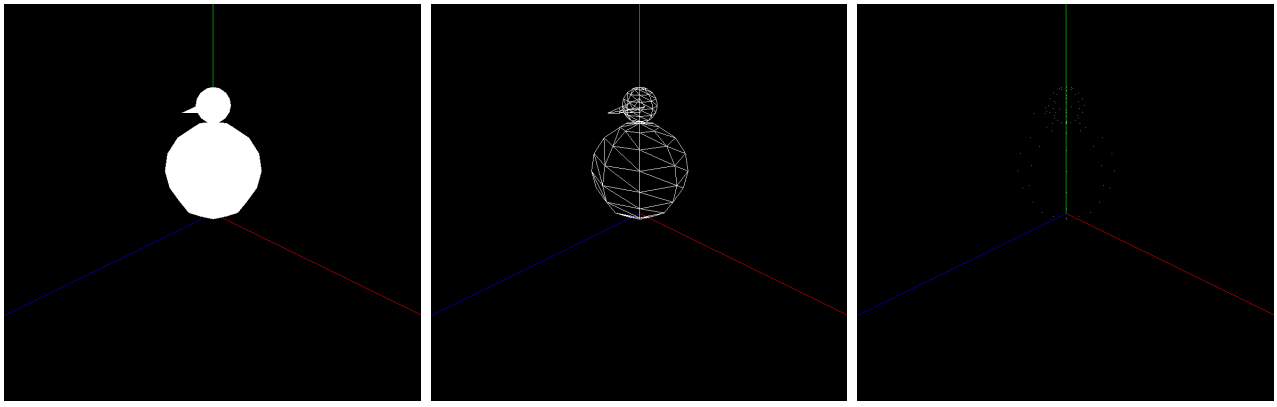

Figura 5.3: Boneco de Neve em modo Fill vs. Boneco de Neve em modo Line vs. Boneco de Neve em modo Point

5.4 Teste 4 - Fila de cubos

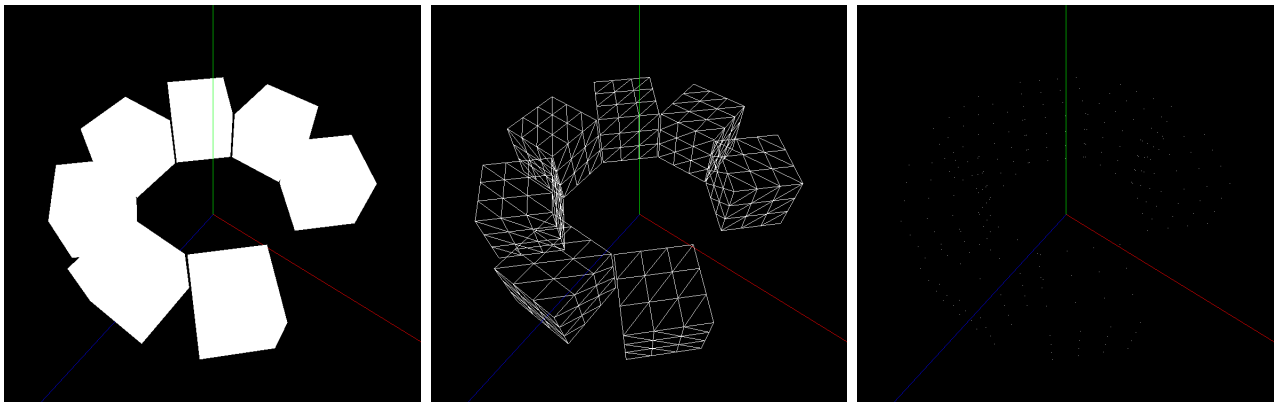

Figura 5.4: Cubos em modo Fill vs. Cubos em modo Line vs. Cubos em modo Point

6. Conclusão

Com a realização desta fase, é notória a extensão dos conhecimentos adquiridos em aula, como por exemplo da linguagem de programação utilizada e manipulação de ficheiros XML. Além disto, consolidou-se uma melhor compreensão específica ao manuseio de rotações, translações e escalas, que era precisamente o objetivo desta fase.

No final de uma revisão geral do projeto até ao momento, verificou-se, então, que todos os requisitos estipulados para esta etapa do trabalho prático foram cumpridos com sucesso.