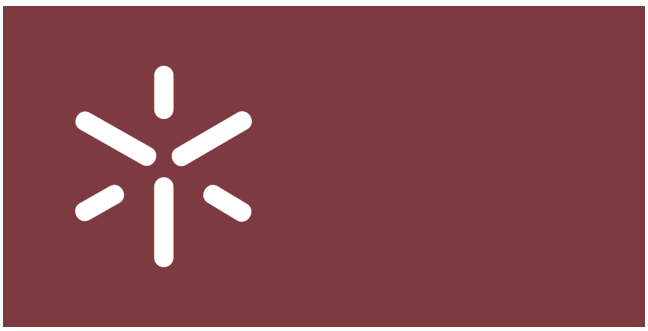

Universidade do Minho

Escola de Engenharia

Licenciatura em Engenharia Informática

Projeto da U.C. Computação Gráfica

2ª Fase

Ano letivo 2023/2024

Realizado por:

A91672 - Luís Ferreira
A93258 - Bernardo Lima
A100543 - João Pastore
A100554 - David Teixeira

Conteúdo

Lista de Figuras

1 Introdução

No âmbito da segunda fase do projeto da Unidade Curricular de Computação Gráfica, o objetivo foi efetuar alterações ao *engine* criado na fase anterior, de modo a ser capaz de receber uma variedade de primitivas geométricas, podendo também aplicar a estas uma variedade de transformações geométricas, seguindo uma hierarquia, através de um ficheiro *XML*.

Com a finalidade de testar as alterações pretendidas desta fase, utilizamos os ficheiros de teste fornecidos pelos docentes, acompanhados pela criação de um modelo estático do sistema solar, de forma a testar o progresso alcançado.

2 Parser

Grande parte das modificações requeridas na segunda fase do projeto (como referido anteriormente) incidirão sobre o *Engine*, com o intuito de permitir a criação de cenas que empregam transformações geométricas organizadas de maneira hierárquica. Para alcançar este objetivo, foi essencial conceber novas estruturas de dados para armazenar todas as informações pertinentes do ficheiro *XML* e, por conseguinte, gerar as cenas conforme especificado.

2.1 Estruturas de Dados Introduzidas

Para armazenar o novo tipo de formato do ficheiro *XML*, foi necessário alterar o nosso `parser.cpp`. Eis a constituição do novo *parser*.

```
struct Parser {
    Window window;
    Camera camera;
    Group rootNode;
};
```

A `struct Parser` contém três campos, cada uma sendo uma outra `struct: Window, Camera e Group`. Vamos discutir cada uma delas:

- `struct Window`: Tem dois campos, `int width` e `int height`, que armazenam as dimensões da janela, ou seja, a largura e a altura, respectivamente.
- `struct Camera`: Possui quatro campos, `Position position, LookAt lookAt, Up up e Projection projection`, que armazenam todas as configurações da câmera. Aqui está uma breve descrição de cada componente:
  - `struct Position`: Armazena a posição de um objeto num espaço 3D, com coordenadas (x, y, z) nos campos `float x, float y e float z`.
  - `struct LookAt`: Armazena as coordenadas para onde a câmera está a olhar no espaço 3D, utilizando os campos `float x, float y e float z`.
  - `struct Up`: Armazena as coordenadas que definem a direção para cima da câmera no espaço 3D, nos campos `float x, float y e float z`.
  - `struct Projection`: Armazena as configurações de projeção da câmera, incluindo o campo de visão (*fov*), e os planos de corte *near* e *far*. Utiliza os campos `float fov, float near e float far`.
- `struct Group`: Esta estrutura é utilizada para armazenar um grupo de transformações, ficheiros de modelo e outros grupos, permitindo a criação de uma hierarquia de objetos. Possui os seguintes campos :
  - `std::vector<Transform> transforms`: Vetor que armazena todas as transformações que devem ser aplicadas ao grupo. Cada transformação é representada por uma `struct Transform`, que contém informações sobre o tipo de transformação (rotação ou translação), as coordenadas da transformação e o ângulo de rotação (através das variáveis `char type, float x, float y, float z e float angle`).
  - `std::vector<ModelFile> modelFiles`: Vetor que armazena todos os ficheiros modelo que fazem parte do grupo. Cada ficheiro de modelo é representado por uma `struct ModelFile`, que contém o nome do ficheiro do modelo (através da variável `std::string filename`).

– `std::vector<Group> children`: Vetor que armazena todos os grupos-filho deste grupo. Isto permite a criação de uma hierarquia de grupos, onde cada grupo pode conter outros grupos, bem como transformações e ficheiros de modelo.

2.2 *Parsing*

O *parsing* de um ficheiro *XML* é realizado através da função:

`Parser* ParserSettingsConstructor(const std::string& filePath)`.

Esta função recebe como argumento o caminho para o ficheiro *XML* a ser analisado e retorna um *pointer* para uma `struct Parser`, contendo as informações extraídas do ficheiro.

O processo de *parsing* é dividido em várias funções auxiliares que são chamadas a partir da função `ParserSettingsConstructor`. Aqui está uma visão geral do processo de *parsing*:

– A função `loadXML()` é responsável por carregar um ficheiro *XML* a partir do *path* fornecido como argumento. Utiliza-se a biblioteca *tinyxml2* para isso. Se o carregamento for bem-sucedido, o *pointer* para o documento *XML* é retornado.

– A função `parseWindowSettings()` é chamada para extrair as configurações da janela do *XML*. Verifica se o elemento `<window>` existe no *XML* e, se existir, extrai os atributos `width e height`, representando a largura e a altura da janela, respectivamente.

– A função `parseCameraSettings()` é responsável por extrair as configurações da câmera do *XML*. Procura pelo elemento `<camera>` no *XML* e extrai as suas subsecções `<position>, <lookAt>, <up>, e <projection>`, preenchendo os campos correspondentes na `struct Camera`.

– A função `parseGroupNode()` é utilizada para extrair as informações do nó raiz do grupo (e recursivamente os seus filhos, se existirem) do *XML*. Procura pelo elemento `<group>` no *XML* e extrai as suas subseções `<transform> e <models>`, preenchendo os campos correspondentes na `struct Group`. Para além disso, para cada subelemento `<group>`, a função é chamada recursivamente para construir a hierarquia de grupos.

Após o processo de *parsing* ser concluído com sucesso, um *pointer* para a `struct Parser` é retornado, contendo todas as informações extraídas do ficheiro *XML*. Essas informações podem, então, ser utilizadas conforme necessário pelo programa que chama a função de *parsing*. Se ocorrer algum erro durante o processo, o programa imprime uma mensagem de erro e termina a execução.

3 Engine

Com as alterações feitas para suportar a hierarquia de cenas, foi posteriormente necessário efectuar algumas alterações ao *engine*, possibilitando o desenho de **primitivas**.

3.1 Desenho

Para respeitar a hierarquia, incorporámos o método já existente, `drawFiguras()`, num novo método `drawGroups()`, com a finalidade de podermos aplicar as transformações apenas nos subgrupos. Isto é possível através das propriedades do `glPushMatrix()` e `glPopMatrix()`.

Com estes dois métodos, as estruturas de dados do *parser*, e invocando o `drawGroups()` de maneira recursiva, é possível aplicar todas as transformações aos subgrupos correspondentes, uma vez que o `glPushMatrix()` **guarda as transformações aplicadas**. Chegando ao último elemento desse *branch* (equiparando a navegação pelo *XML* a uma árvore), o `glPopMatrix()` **remove apenas as transformações aplicadas desse *branch***, podendo navegar pela árvore representada no ficheiro *XML*, sem nunca aplicar transformações não desejadas a outras primitivas geométricas, podendo aplicar apenas as pretendidas.

Para guardarmos as primitivas geométricas, utilizamos uma lista de figuras, de forma semelhante à fase anterior, mas esta é limpa durante a chamada recursiva, tentando minimizar a memória alocada pelas figuras ao longo da renderização com o novo método `cleanList()` da `list.cpp`.

Relativamente às **transformações**, estas vão sendo aplicadas utilizando os métodos `glTranslatef()`, `glRotatef()` e `glScalef()`, sendo este selecionado com base no *type* da estrutura *transform*.

4 Sistema Solar

4.1 Geometry

Para a criação do sistema solar, foi-nos pedida uma representação dos seus planetas. Neste ponto, pensámos que seria necessário a criação de mais uma primitiva geométrica, uma vez que o planeta Saturno posssui um anel vísivel, e que achámos necessária a inclusão do mesmo no modelo.

4.2 Anel de Saturno

Para a criação do anel, utilizamos uma abordagem semelhante à criação de um círculo, recorrendo a coordenadas polares, ligando cada fatia ($deltaAngle = 2\pi/slices$) do anel a um *raio interno*, terminando este no *raio externo*, formando efetivamente um anel.
Para a visualização deste anel ser possível de uma perspetiva *top/down* ou *bottom/up*, a criação de cada fatia, composta normalmente por **dois triângulos**, foi também acompanhada pela criação de mais dois, com as orientações **opostas**. Criando assim **4 triangulos por fatia**.

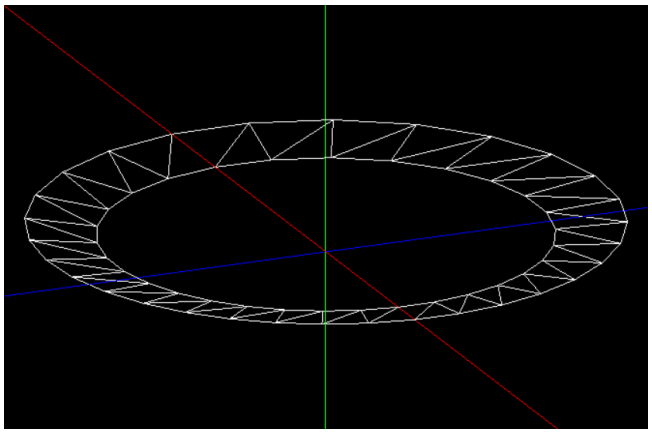

Figura 1: Anel de Saturno

4.3 Formato do ficheiro XML

Para o desenvolvimento do ficheiro *XML* da demo pretendida pelos docentes, decidimos utilizar como modelos o **Sol**, os **oito planetas do sistema solar** e a **Lua**.

De forma a termos uma representação realista, decidimos colocar os planetas e a Lua à escala em comparação com o tamanho da primitiva da Terra, pondo de parte o Sol, pois verificamos que denegreria a visualização do modelo. Decidimos, também, dispor as primitivas ao longo do eixo x do referencial, estando no início deste, o Sol, terminando no último planeta, Neptuno.

Referenciando exclusivamente o *XML*, partindo do *group* inicial, temos o primeiro subgrupo, o **Sol**, onde aplicamos uma escala de 10.9, voltando ao grupo inicial.

```
<group>
  <group>
    <transform>
      <scale x="19" y="19" z="19"/>
      <!--  Tamanho do Sol  -->
    </transform>
    <models>
      <model file="..\output\sphere.3d"/>
      <!--  Sol  -->
    </models>
  </group>
```

Figura 2: Excerto *XML* do Sol

4.4 Mércurio, Vénus, Marte, Júpiter, Urano e Néptuno

Para estas primitivas, a abordagem no *XML* é idêntica. Começando por aplicar uma translação de um valor variável ao grupo principal, de modo a esta transformação se aplicar às restantes primitivas. De seguida, é aplicada uma escala variável antes de apresentar o *model file* da primitiva, voltando posteriormente ao grupo anterior.

```
<group>
  <transform>
    <translate x="24" y="0" z="0"/>
    <!--  Distância  -->
  </transform>
  <group>
    <transform>
      <scale x="0.38" y="0.38" z="0.38"/>
      <!--  Tamanho de Mercúrio  -->
    </transform>
    <models>
      <model file="..\output\sphere.3d"/>
      <!--  Mercúrio  -->
    </models>
  </group>
```

Figura 3: Exemplo no *XML* (Mércurio)

4.5 Terra

Para incorporar a Lua juntamente com a Terra, decidimos implementar subgrupos adicionais.

Isto permite que a Lua mantenha a mesma translação em relação ao início do referencial, enquanto possibilita uma translação e escala exclusivamente aplicada apenas a si. Desta forma, utilizamos a hierarquia de forma mais eficiente, criando dois subgrupos: um para a Terra e outro para a Lua.

```
<group>
  <transform>
    <translate x="5" y="0" z="0"/>
    <!-- Distância -->
  </transform>
  <group>
    <models>
      <model file="..\output\sphere.3d"/>
      <!-- Terra -->
    </models>
  </group>
  <group>
    <transform>
      <translate x="1" y="0" z="2"/>
      <!-- Distância da Lua à Terra -->
      <scale x="0.273" y="0.273" z="0.273"/>
      <!-- Tamanho da Lua -->
    </transform>
    <models>
      <model file="..\output\sphere.3d"/>
      <!-- Lua -->
    </models>
  </group>
```

Figura 4: Excerto do *XML* da Terra e Lua

4.6 Saturno e o Seu Anel

Do mesmo modo que o caso anterior, este grupo inclui o subgrupo do *model file* de Saturno. Além disso, possui outro subgrupo para representar o anel de Saturno. Ambos os subgrupos mantêm a mesma translação do grupo pai e permitem uma rotação de 26.7 graus no sentido do eixo y, assim como uma escala diferente para este último.

```
<group>
  <transform>
    <translate x="30" y="0" z="0"/>
    <!-- Distância -->
  </transform>
  <group>
    <transform>
      <scale x="9" y="9" z="9"/>
      <!-- Tamanho de Saturno -->
    </transform>
    <models>
      <model file="..\output\sphere.3d"/>
      <!-- Saturno -->
    </models>
  </group>
  <group>
    <transform>
      <rotate angle="26.7" x="1" y="0" z="1"/>
      <!-- Rotação do Anel -->
      <scale x="1.2" y="1.2" z="1.2"/>
      <!-- Tamanho do Anel -->
    </transform>
    <models>
      <model file="..\output\ring.3d"/>
      <!-- Anel de Saturno -->
    </models>
  </group>
```

Figura 5: Excerto do *XML* de Saturno e o seu Anel

4.7 Resultado Final

Ao utilizar o *engine* juntamente com o *XML*, obtivemos o modelo conforme ilustrado na Figura 6.

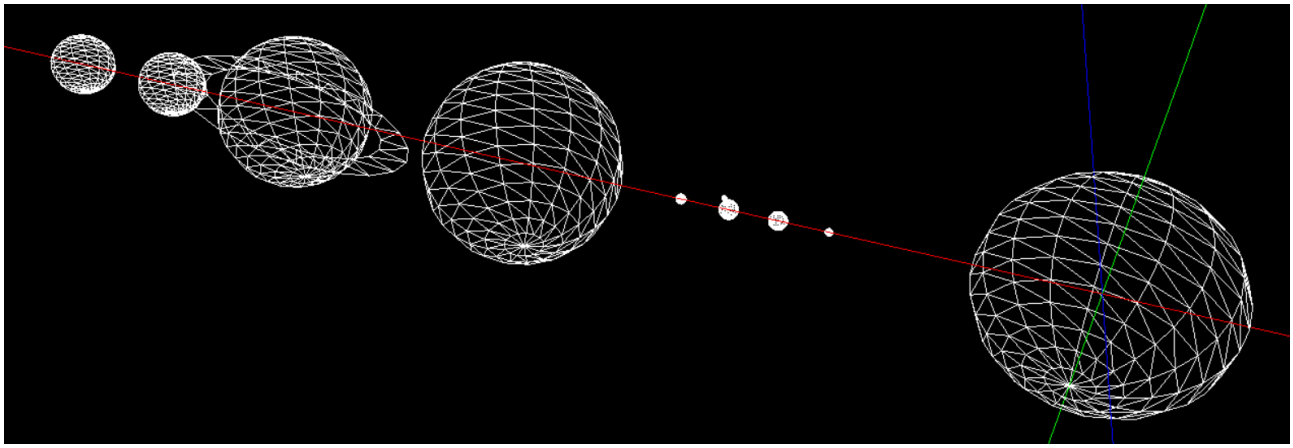

Figura 6: Modelo do sistema solar

5 Conclusão

A presente etapa deste trabalho viabilizou a consolidação dos conhecimentos adquiridos sobre transformações geométricas, assim como conhecimento da gestão das matrizes de transformações. Acredita-se que todos os objetivos delineados para esta fase foram alcançados de forma eficaz, tendo finalizado esta fase com sucesso.