Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Computação Gráfica

Ano Letivo 2022/2023

Trabalho prático

Parte 2 - Transformações geométricas

Grupo 14

Ana Rita Santos Poças, A97284
Miguel Silva Pinto, A96106
Orlando José da Cunha Palmeira, A97755
Pedro Miguel Castilho Martins, A97613

5 de abril de 2023

Trabalho prático

Parte 2 - Transformações geométricas

Grupo 14

Ana Rita Santos Poças, A97284
Miguel Silva Pinto, A96106
Orlando José da Cunha Palmeira, A97755
Pedro Miguel Castilho Martins, A97613

5 de abril de 2023

Índice

Índice de figuras

1 Introdução

A segunda fase do projeto da cadeira de **Computação Gráfica**, consistiu em continuar o trabalho já feito na primeira fase e acrescentar novas funcionalidades, essencialmente para a *Engine*.

As mudanças contemplaram a capacidade da ***Engine*** ser capaz de aplicar **transformações geométricas** aos modelos de forma **hierárquica**, através de instruções introduzidas por um **ficheiro XML** num determinado formato.

Como validação final do trabalho desenvolvido, foi escrito um ficheiro XML que descreve as instruções a realizar pela *Engine* para desenhar um **Sistema Solar**.

Ao longo deste relatório, iremos descrever de forma detalhada as decisões e abordagens que foram adotadas e que permitiram a implementação das aplicações propostas.

2 Motor

Uma boa parte das alterações solicitadas na fase 2, serão feitas no **Engine**, para que seja capaz de criar cenas que usem **transformações geométricas** organizadas de forma **hierárquica**. De maneira a alcançar este objetivo foi necessário arquitetar novas estruturas de dados para se armazenar toda a informação necessária do ficheiro XML e consequentemente gerar as cenas especificadas.

2.1 Estruturas de dados

Para armazenar todas as informações do ficheiro de configuração, recorreu-se à implementação de diversas estruturas de dados. Entre todas elas, as principais são a *Config*, *Group*, *Transform* e *Tree*.

A estrutura *Config* serve para armazenar todas as informações do ficheiro de configuração do `engine`. Para isso, esta estrutura foi implementada do seguinte modo:

```
struct config{
    float window[2];
    float poscam[3];
    float lookAt[3];
    float up[3];
    float projection[3];
    Tree groups;
};
```

Figura 2.1: Implementação da estrutura *Config*

Nesta estrutura, o campo `window` contém as dimensões da janela. Os campos `poscam`, `lookAt`, `up` contêm, respectivamente, a posição da câmara, a posição para onde a câmara olha e o vector *up* da câmara. O campo `projection` contém os parâmetros *fov*, *near* e *far* da câmara.

Finalmente, o campo `groups` armazena todos os grupos (transformações geométricas e *models*) numa árvore conforme a hierarquia que consta no ficheiro de configuração. Os grupos na árvore estão implementados na seguinte estrutura:

```
struct group{
    List transforms;
    List models;
};
```

Figura 2.2: Implementação da estrutura *Group*

Na `struct group` a lista *transforms* contém todas as transformações geométricas (translações, rotações e escalas) que serão aplicadas aos modelos na lista *models*. Cada modelo é uma lista de pontos gerada a partir do seu ficheiro .3d.

As transformações geométricas estão implementadas na seguinte estrutura:

```
struct transform{
    char type;
    float x,y,z;
    float angle;
};
```

Figura 2.3: Implementação da estrutura *Transform*

Nesta estrutura, o campo `type` pode tomar os valores `'t'`, `'r'` ou `'s'`, indicando se a transformação em questão é uma translação, rotação ou escala, respectivamente. Os campos `'x'`, `'y'` e `'z'` são os parâmetros da transformação e o campo `'angle'` é o ângulo de rotação caso a transformação geométrica seja uma rotação. O campo `'angle'` é ignorado se a transformação for uma escala ou translação.

A árvore que armazena os grupos na estrutura `Config` está implementada do seguinte modo:

```
struct tree{
    void* valor;
    List filhos;
};
```

Figura 2.4: Implementação da estrutura *Tree*

Decidimos utilizar uma árvore, mais concretamente uma *Rose Tree*, uma vez que precisamos de uma estrutura com recursividade para armazenar a informação dos nodos `Group` do ficheiro XML, bem como os seus respetivos nodos filho.

2.2 *Parsing* do ficheiro XML

Para realizar o reconhecimento do ficheiro XML, implementámos a função `xmlToConfig`. O processo de reconhecimento inicia com a obtenção dos dados relativos à janela (*window*) e dos dados relativos à câmara (posição, *lookAt*, vetor *up*, fov, near e far). Seguidamente, é invocada a função `parseGroups` para fazer *parse* aos nodos *group*.

A função `parseGroups` começa por recolher os dados relativos às transformações geométricas e modelos do grupo. Para recolher as transformações do grupo, começamos por verificar o nome da sua *tag* (*rotate*, *translate* ou *scale*) para atribuir o valor ao campo `type` e recolhemos seus valores `x`, `y` e `z` (e o `angle`, no caso da rotação). Relativamente aos modelos, primeiramente é feita uma análise dos ficheiros .3d especificados no XML, sendo cada ficheiro transformado numa estrutura de dados `Figura` (uma lista de pontos). Posteriormente, cada figura gerada a partir do seu ficheiro .3d é adicionada à lista de modelos (campo `models` da estrutura `Group`, ver figura 2.2) do grupo que está a ser criado.

No fim, a função é invocada recursivamente para tratar todos os *groups* filhos do *group* que acabou de processar.

No fim de recolher todos os grupos, a árvore de grupos é criada e é inserida na estrutura de configuração do *engine* (ver figura 2.1).

2.3 Desenho

Após o *parse* de toda a informação do ficheiro, precisamos de percorrer a árvore `groups` resultante da análise do ficheiro e tratar de desenhar os modelos introduzidos, mas também gerir as transformações presentes no ficheiro. Para tal, foi criada a função *drawGroups*, que é invocada na função *renderScene*, recebendo como argumento o nodo raiz da árvore `groups` que contém toda a informação para o desenho completo da cena especificada no ficheiro XML.

A função *drawGroups*, faz uma travessia *pre-order* da árvore e em cada nodo que percorre, irá armazenar a matriz de desenho através do método *glPushMatrix*, uma vez que pretendemos guardar o estado da matriz para que possa ser recuperado, caso ainda haja grupos hierarquicamente superiores ao nodo em questão.

Após ser feito o *glPushMatrix*, vão ser executadas as transformações indicadas no nodo pela ordem que aparecem, através das funções *glTranslatef*, *glScalef* e *glRotatef*, e de seguida vão ser desenhadas as figuras especificadas na árvore, através da função *drawFiguras* que já era utilizada na 1ª fase.

Por fim, aplica-se recursivamente a função *drawGroups* a todos os nodos filhos desse nodo, e após a aplicação recursiva aos nodos filho, será reposto o estado prévio da matriz através do método *glPopMatrix*, uma vez que significa que a travessia está a subir na árvore, sendo necessário obter a matriz de desenho no estado anterior às alterações realizadas pelo nodo.

3 Sistema Solar

Com o intuito de demonstrar as funcionalidades desenvolvidas nesta segunda fase do projeto, foi requerida uma demonstração de um desenho do Sistema Solar. Para tal, foi escrito um ficheiro XML que especifica todas as transformações e modelos necessários para o desenho de todos os astros do Sistema Solar. No ficheiro está contemplado o Sol, e todos os seus planetas desde Mercúrio a Neptuno, estando também desenhada a Lua relativamente à Terra e no caso específico de Saturno está desenhado o anel característico de Saturno.

3.1 Anel de Saturno

Para podermos complementar Saturno com o seu anel, decidimos acrescentar ao *generator* a primitiva *ring* que permite desenhar o anel pretendido.

Os argumentos necessários para criar esta primitiva são r_i, r_e e *slices* que representam, respectivamente, o raio interno, o raio externo e o número de divisões que anel terá. O significado destes argumentos pode ser clarificado na figura seguinte em que temos um anel com oito *slices*:

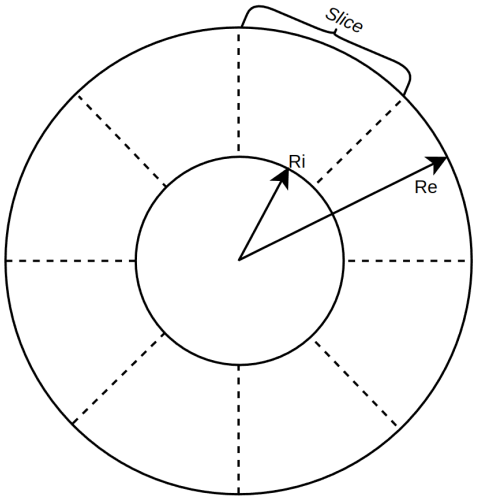

Figura 3.1: Significado dos argumentos de criação do *ring*

Cada *slice* do anel será constituída por quatro triângulos em que dois servem para a *slice* ser visualizada por cima e os outros dois servem para a *slice* ser visualizada por baixo.

O anel é construído *slice* a *slice* e os seus vértices são deslocados segundo um ângulo $\Delta = 2\pi/slices$ (os vértices utilizados para criar as *slices* estão em coordenadas esféricas $(\alpha, \beta, raio)$).

O processo de formação de uma *slice* pode ser demonstrado na figura seguinte:

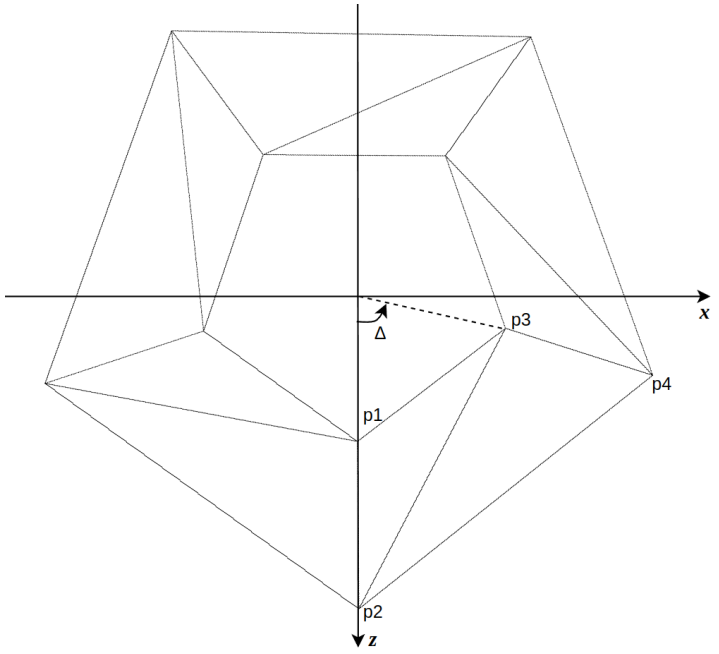

Figura 3.2: Criação de uma *slice* para um *ring* com 5 *slices*

Para a criação dos triângulos da primeira *slice*, são inseridos na lista de pontos do anel os pontos $p1, p2, p3$ e $p4$ nas seguintes ordens: $[p_1, p_2, p_3], [p_3, p_2, p_4], [p_3, p_2, p_1]$ e $[p_2, p_3, p_4]$. Na iteração seguinte, o valor de α dos pontos $p1, p2, p3$ e $p4$ é incrementado pelo valor de Δ e o processo repete tantas vezes quanto o número de *slices* fornecido.

3.2 Modelos utilizados

Os modelos utilizados para o desenho do Sistema Solar, foram gerados a partir do *generator*, tendo sido apenas usados dois ficheiros .3d, um com a primitiva *sphere* e o outro com a primitiva *ring*.

Para a esfera foram utilizados os seguintes parâmetros (raio = 1, *stacks* = 20, *slices* = 20) sendo gerado o ficheiro *sphere.3d* e para o anel foram utilizados (raio interior = 7, raio exterior = 9, *slices* = 30) sendo gerado o ficheiro *ring.3d*.

A esfera foi utilizada para desenhar o Sol, os planetas, e a Lua, e o anel foi utilizado para desenhar o anel de Saturno.

3.3 Formato do ficheiro XML

O ficheiro XML final contém vários grupos que representam o Sol ou um dos planetas, que nos casos específicos irão conter subgrupos para representar objetos relativos a um determinado astro, mais concretamente a Lua relativa à Terra e o Anel de Saturno relativo a Saturno.

O Sol foi desenhado no centro do referencial apenas sofrendo uma escala para o tornar maior.

```
<group>
    <transform>
        <scale x="20" y="20" z="20"/> <!-- Tamanho do Sol -->
    </transform>
    <models>
        <model file="../Fase_2/outputs/sphere.3d" /> <!-- Sol -->
    </models>
</grou
```

Figura 3.3: Excerto do ficheiro XML relativo ao Sol

Para os planetas não se apresentarem alinhados ao longo de um eixo, é aplicada uma rotação aleatória relativa ao eixo Y e, de maneira a distanciar os planetas relativamente ao Sol, são aplicadas translações aos modelos. Consoante o tamanho do planeta, foram também aplicadas escalas para que o seu tamanho seja minimamente realista.

```
<group>
    <transform>
        <rotate angle ="45" x="0" y="1" z="0"/> <!-- Rotação relativa ao Sol de Mercúrio-->
        <translate x="24" y="0" z="0"/>
        <scale x="0.3" y="0.3" z="0.3"/> <!-- Tamanho de Mercúrio -->
    </transform>
    <models>
        <model file="../Fase_2/outputs/sphere.3d" /> <!-- Mercúrio -->
    </models>
</group>
```

Figura 3.4: Excerto do ficheiro XML relativo a Mercúrio

O desenho da Lua relativamente à Terra implica a translação para se posicionar no sítio correto relativamente aos outros planetas, e de seguida são usados dois subgrupos, um para desenhar a Terra e outro para desenhar a Lua. O desenho da Terra corresponde ao padrão de desenho dos outros planetas, enquanto que a Lua sofre uma pequena translação relativamente à posição central da Terra.

```
<group>
    <transform>
        <rotate angle ="185" x="0" y="1" z="0"/> <!-- Rotação relativa ao Sol da Terra-->
        <translate x="34" y="0" z="0"/>
    </transform>
    <group>
        <transform>
            <scale x="0.5" y="0.5" z="0.5"/> <!-- Tamanho da Terra -->
        </transform>
        <models>
            <model file="../Fase_2/outputs/sphere.3d" /> <!-- Terra -->
        </models>
    </group>
    <group>
        <transform>
            <translate x="0" y="0" z="1"/> <!-- Distância da Lua à Terra -->
            <scale x="0.1" y="0.1" z="0.1"/> <!-- Tamanho da Lua -->
        </transform>
        <models>
            <model file="../Fase_2/outputs/sphere.3d" /> <!-- Lua -->
        </models>
    </group>
</group>
```

Figura 3.5: Excerto do ficheiro XML relativo à Terra e Lua

Para o desenho do anel de Saturno foi aplicada a mesma estratégia de criar dois subgrupos, um para o desenho de Saturno, e outro para o desenho do anel de Saturno. Ao anel, foi apenas aplicado uma rotação para simular a inclinação do anel de Saturno.

```
<group>
    <transform>
        <rotate angle ="30" x="0" y="1" z="0"/> <!-- Rotação relativa ao Sol de Saturno-->
        <translate x="66" y="0" z="0"/>
    </transform>
    <group>
        <transform>
            <scale x="5" y="5" z="5"/> <!-- Tamanho de Saturno -->
        </transform>
        <models>
            <model file="../Fase_2/outputs/sphere.3d" /> <!-- Saturno -->
        </models>
    </group>
    <group>
        <transform>
            <rotate angle ="20" x="1" y="0" z="1"/> <!-- Ângulo de rotação do anel de Saturno -->
            <scale x="1" y="1" z="1"/> <!-- Tamanho do Anel -->
        </transform>
        <models>
            <model file="../Fase_2/outputs/ring.3d" /> <!-- Anel de Saturno -->
        </models>
    </group>
</group>
```

Figura 3.6: Excerto do ficheiro XML relativo ao anel de Saturno

3.4 Resultado final

Ao correr a *engine* com o ficheiro `solar.xml` como argumento, obtivemos os seguintes resultados.

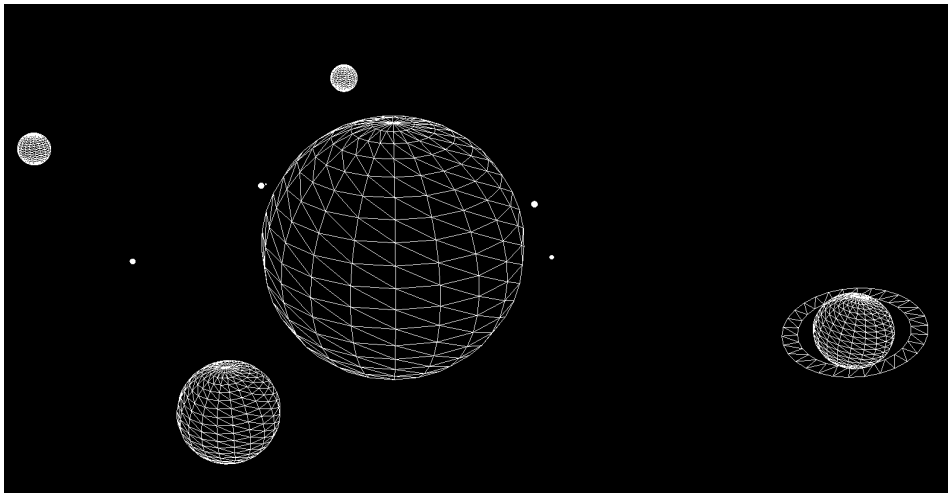

Figura 3.7: Imagem do Sistema Solar

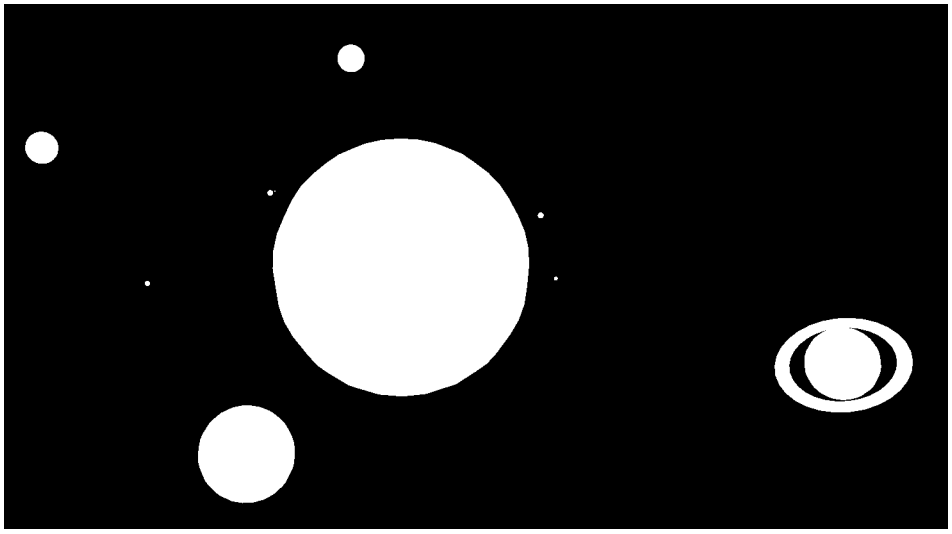

Figura 3.8: Imagem do Sistema Solar no modo GL_FILL

A seguinte imagem permite ver em melhor detalhe a Lua, uma vez que não foi possível visualizá-la nas imagens acima devido à sua reduzida dimensão relativamente aos outros planetas.

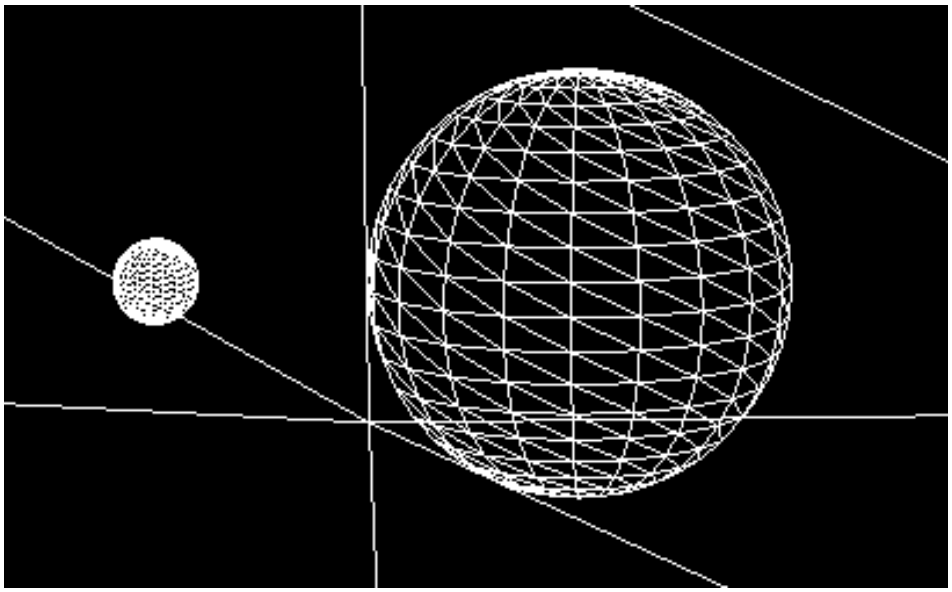

Figura 3.9: Imagem ampliada da Terra e da Lua

Foi também elaborado um ficheiro que mostra os planetas do Sistema Solar em linha reta, para permitir uma visualização mais comparativa entre os planetas e as suas distâncias relativas.

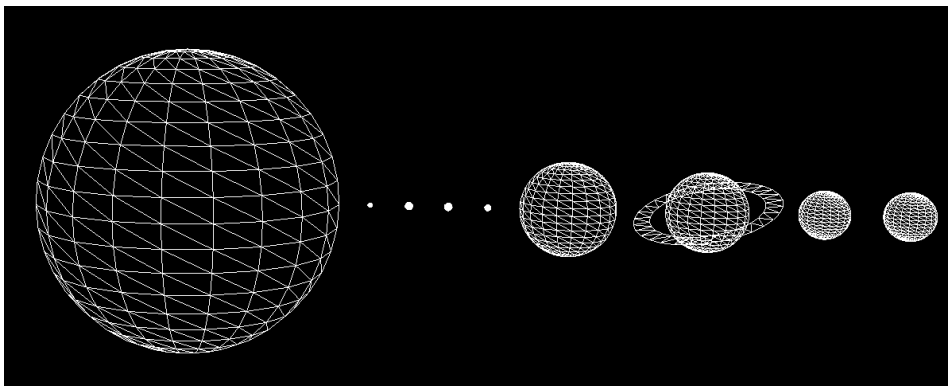

Figura 3.10: Imagem do Sistema Solar alinhado

4 Conclusão

Ao longo da elaboração desta segunda fase do projeto, foi possível consolidar os conceitos relativos a transformações geométricas e a gestão das matrizes de transformação.

Acerca da concretização do trabalho realizado, encontramo-nos satisfeitos, uma vez que conseguimos implementar todas as funcionalidades pedidas para esta fase bem como alguns extras como a geração de um *Ring*, uma primitiva adicionada para tornar Saturno mais parecido com a realidade no desenho do Sistema Solar. Também implementámos algumas pequenas funcionalidades extra como movimentação da câmara em rotação, alteração da maneira como os polígonos são renderizados e a capacidade de dar *toggle* de um eixo de coordenadas.

Por fim, consideramos que reunimos, ao longo desta fase, os elementos necessários para que consigamos progredir para a próxima fase do projeto.