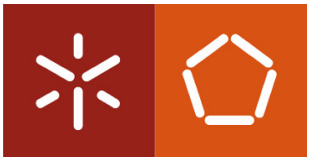
Universidade do Minho

Escola de Engenharia



Computação Gráfica

Licenciatura em Engenharia Informática

Geometric Transforms

Trabalho Prático - Fase 2 - Grupo 7

João Silva - [A89293]
Francisco Novo - [A89567]
João Vieira - [A93170]
Tiago Ribeiro - [A93203]

Abril, 2022

Conteúdo

Introdução **2**

Estrutura do ficheiro XML **3**

Alterações no motor **4**

Estruturas de dados 4

Leitura do ficheiro XML 5

Desenho dos modelos 5

Alterações no gerador **6**

Toro 6

Sistema Solar **7**

Conclusão **8**

Introdução

No âmbito da Unidade Curricular de Computação Gráfica, foi desenvolvida a segunda fase do trabalho prático proposto pelos docentes. O principal objetivo desta fase era a criação de um cenário gráfico 3D do sistema solar, a partir da leitura de um ficheiro em formato XML.

Para isto, foi necessário proceder à alteração do trabalho feito na primeira fase, mais concretamente no motor, pois a estrutura do ficheiro XML sofreu alterações, tendo sido para o efeito criadas classes que permitiram armazenar o conteúdo do ficheiro, para depois serem desenhados os modelos e as transformações respetivas em 3D.

Para uma representação mais fiel à realidade do sistema solar, foram efetuadas alterações no gerador. Foi criada uma nova primitiva gráfica, o toro, para permitir reproduzir os anéis de Saturno.

Estrutura do ficheiro XML

A estrutura do ficheiro XML é diferente da estrutura vista na 1ª fase do projeto. Este novo formato contém transformações, que podem ser aplicadas a um grupo inteiro de modelos. Deste modo, o formato do XML poderia ser representado da seguinte maneira:

1. Começa com um world, que pode conter uma câmara e um grupo.
2. As especificações da câmara no ficheiro XML permanecem iguais às da fase anterior.
3. Um group, que pode conter um "models", um "transform"e vários "group".

Assim, podemos aplicar várias transformações aos objetos que estamos a desenhar, sendo estas do tipo "translate", "rotate"e "scale".

Em termos práticos, um exemplo do formato que se poderá encontrar seria:

```
<!-- Cada campo contem um comentario a indicar a sua funcao -->
<world> <!-- Corresponde ao world referido em cima.-->
    <camera> <!-- corresponde as configuracoes pretendidas na camara. -->
        <position x="10" y="10" z="10" />
        <lookAt x="0" y="0" z="0" />
        <up x="0" y="1" z="0" />
         <projection fov="60" near="1" far="1000" />
     </camera>
    <group> <!-- Unico group permitido (E possivel ter mais groups desde que seja
        dentro deste group) -->
        <transform> <!-- Campo onde sao indicadas as transformacoes -->
            <!-- Tipos de transformacoes -->
            <translate x="4" y="0" z="0" />
            <rotate angle="30" x="0" y="1" z="0" />
            <scale x="2" y="0.3" z="1" />
            <color r="0" g="1" b="0" />
        </transform>
        <models> <!-- Modelos que se pretende desenhar. -->
            <model file="cone.3d" />
            <model file="plane.3d" />
        </models>
    </group>
</world>
```

Após entender como vai ser apresentada a informação que é necessária para interpretar o ficheiro, é possível passar à fase de desenvolvimento.

Alterações no motor

Nesta fase foram necessárias algumas alterações no motor que foi desenvolvido na primeira fase, mais especificamente às estruturas de dados, à forma como é feita a leitura do ficheiro XML, e à forma de como são desenhados os modelos.

Estruturas de dados

No package dos structs foram adicionadas algumas classes de modo a ajudar a implementar as transformações:

- **transform**: Guarda os valores de uma transformação pretendida no ficheiro, assim como o tipo da transformação.

```
class transform{
    float x;
    float y;
    float z;
    float ang;
    transformation trans;
```

Figura 1: Classe transform

- **group**: Guarda o conteúdo de cada group que possa ser encontrado.

```
class group {
    std::vector<figure> models;
    std::vector<transform> transformations;
    std::vector<group> groups;
```

Figura 2: Classe group

O tipo da transformação é guardado com recurso a um *enum*, que irá indicar o tipo de transformação a ser aplicada a um group.

```
enum class transformation {
    rotate,
    translate,
    scale,
    color};
```

Figura 3: Classe transformation

Adicionalmente, foram adicionados os métodos que permitem manipular estas estruturas de dados, como *getters* e *setters*.

Leitura do ficheiro XML

Foi necessário também fazer alterações no que toca à leitura do ficheiro XML. Primeiramente, foi dividida a função que realizava a leitura em duas partes: *lerCamera()* e *lerGroup()*. Isto ajuda na modularidade do código e evita que a função de leitura se torne demasiado comprida.

A função *lerCamera()* trata de ler e guardar as informações contidas no campo "camera"no ficheiro XML, guardando as configurações da câmara, assim como as coordenadas polares da posição inicial da mesma.

A função *lerGrupo()* recebe o apontador para o campo "group"do ficheiro, assim como a estrutura de dados onde deverá armazenar os dados. Começa-se por criar um ciclo, onde a função percorre cada filho do apontador fornecido no input, sendo analisado o *value* de cada um deles. No caso de termos o value equivalente a "transform"sabemos que estamos a lidar com uma transformação. Neste caso, analisamos os filhos deste campo e guardamos os valores fornecidos, colocando-os na estrutura de dados apropriada a partir do método *setTransform()*. No caso em que o value do elemento a analisar é igual a "models", os dados são tratados de igual modo, sendo a informação armazenada através da invocação do método *lerFich3D()*. No caso do value ser igual a "group", inicializa-se um novo objeto group. De seguida, a função é chamada recursivamente, onde é fornecido este novo objeto como argumento, repetindo novamente todo o processo descrito, sendo que no final o group devolvido irá ser acrescentado aos vetor groups do group principal.

Desenho dos modelos

Terminada a leitura do ficheiro, é necessário desenhar os models fornecidos com as devidas transformações.

Deste modo, no *renderScene* é invocada a função *drawFigures()*. Nesta função começa-se por fazer um pushMatrix, assegurando que as alterações feitas são aplicadas à cena em questão. De seguida, são percorridas todas as transformações presentes no grupo, determinando o seu tipo e aplicando-as.

Depois de terem sido aplicadas as transformações, é feita uma iteração sobre os models armazenados no objeto "grupo", desenhando-os.

Por último, são percorridos os grupos que foram lidos no ficheiro, sendo chamada a função *drawFigures* recursivamente, de forma a aplicar as transformações referentes a esse grupo em específico.

Alterações no gerador

Para representar o anel de um planeta o grupo criou uma nova primitiva gráfica, o toro. Esta será usada no cenário gráfico do sistema solar e irá representar os anéis de Saturno.

Toro

A função que gera o toro recebe o raio interior, o raio exterior, o número de *slices* e o número de *stacks*. O toro desenhado irá estar centrado na origem do referencial e estará "de pé". O toro é obtido através da rotação de uma circunferência num eixo.

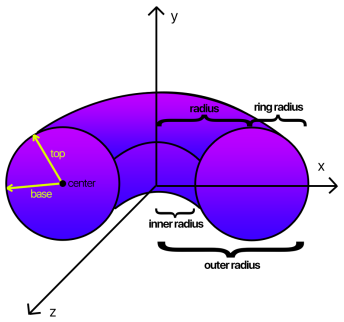


Figura 4: Toro e as suas medidas

Para desenhar o toro é obtido o raio que define a espessura do mesmo (*ring radius* na Figura 4) e a média do raio inferior com o raio superior (*center* na Figura 4). Após isto, são efetuadas iterações sobre as *slices*, onde são obtidas duas circunferências (a atual e a próxima). Dentro destas iterações são percorridas as *stacks*, onde são formados quadriláteros entre as circunferências, sendo estes divididos em dois para formar dois triângulos. Ao fim de percorrer as *stacks* irá ser obtido uma espécie de um "tubo"e após todas as iterações sobre as *slices* é obtido o toro completo.

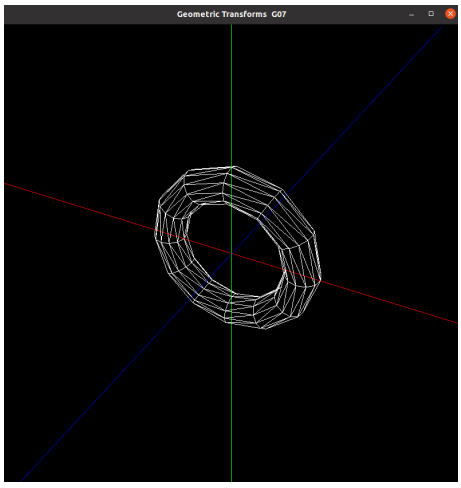


Figura 5: *Torus* com raio interior 2.5, raio exterior 4, 12 *slices* e 12 *stacks*

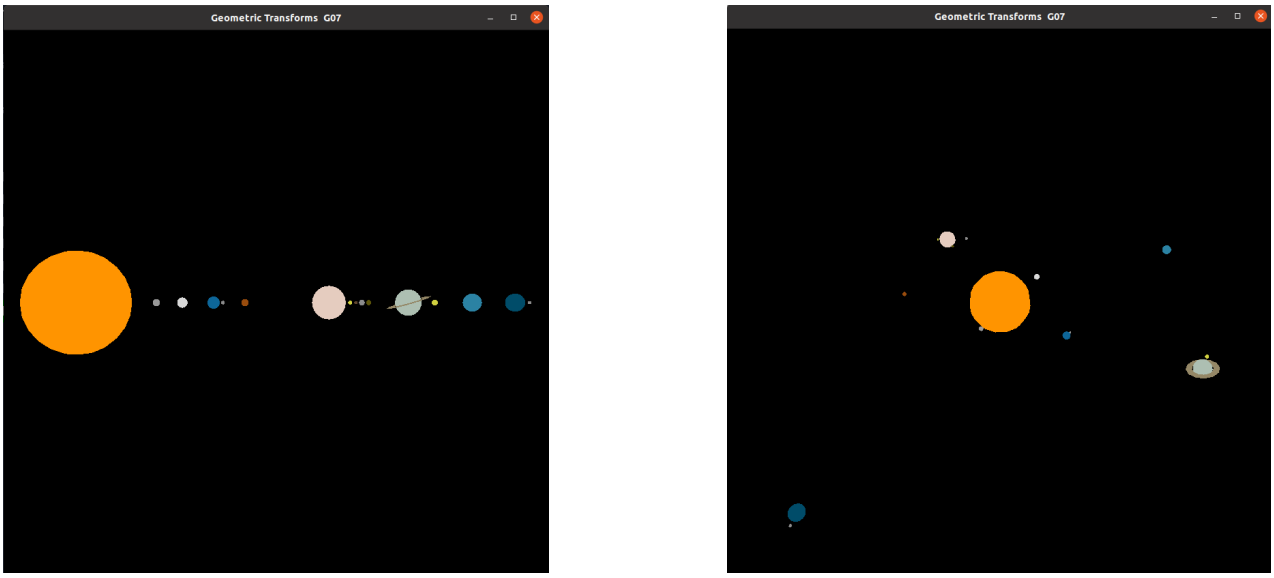
Sistema Solar

Para cumprir o objetivo deste trabalho prático, foi criado um cenário gráfico 3D representativo do sistema Solar através de um ficheiro em formato XML. Este modelo pretende captar uma representação apelativa e clara do sistema solar, sendo que o grupo pretendeu implementar um modelo onde haja um equilíbrio entre uma representação realista e perceptível, dado que ao seguir uma escala real implicaria uma grande desproporcionalidade entre o tamanho dos vários planetas.

Para este efeito, foi usada a primitiva gráfica da esfera para representar o Sol e todos os planetas do sistema solar, sendo que para aumentar o realismo do cenário, foram também desenhados alguns satélites dos seguintes planetas: Terra (Lua), Júpiter (Io, Europa, Ganimedes e Calisto), Saturno (Titã) e Neptuno (Tritão). Para desenhar os anéis de Saturno foi usada a primitiva gráfica do toro, que permitiu uma representação mais fiel à realidade do mesmo.

Relativamente ao ficheiro XML, este começa com as definições da câmara, seguido do grupo que irá conter todos os modelos e transformações geométricas. Neste grupo principal está contido o modelo do Sol, sendo que este tem um grupo filho onde se localizam todos os grupos dos planetas, organizados por distância ao Sol. Se um planeta tiver satélites, o seu grupo irá ter um grupo filho para cada um deles, sendo que para Saturno também irá ter um grupo para os seus anéis.

Neste modelo foram utilizadas todas as transformações geométricas: o *translate* para ditar a distância dos planetas ao Sol, o *rotate* para simular a órbita dos planetas em torno do Sol, o *scale* para alterar o tamanho dos planetas e a *color* para alterar a cor dos planetas.



(a) Planetas alinhados

(b) Sistema solar

Figura 6: Cenários gráficos 3D criados

Conclusão

O trabalho desenvolvido permitiu aprofundar o conhecimento sobre transformações geométricas, bem como aumentar a nossa experiência com o GLUT.

Verificamos que a aplicação é capaz de ler ficheiros XML e construir os cenários 3D nelas descritos. Também foi construído o cenário gráfico do sistema solar, sendo importante salientar que o grupo pretende continuar a desenvolver e melhorar este cenário nas seguintes fases do trabalho prático.

Por fim, consideramos que o trabalho desenvolvido foi positivo, apesar de algumas dificuldades iniciais na leitura de ficheiros, e este cumpre com os requisitos propostos no enunciado.