Universidade do Minho
Escola de Engenharia

Licenciatura em Engenharia Informática 2021/2022

Computação Gráfica

Trabalho prático - Fase 2

- Geometric Transforms -

4 de abril

Benjamim Miranda Costa (A87985)
Maria Sofia Rocha Gomes (A93314)
Marisa Ferreira Soares (A92926)
Miguel Rodrigues Santa Cruz (A93194)

Índice

1. Introdução

No presente relatório será explicado a estratégia da conceção desta segunda fase, as principais alterações em relação à primeira fase, como a arquitetura do programa e os respetivos resultados

O principal objetivo desta segunda fase do trabalho é modificar o *engine* da primeira fase de modo a que passasse a incluir informação acerca das transformações aplicadas aos objetos, contidas nos ficheiros XML que o engine recebe como argumento.

2. Mudanças relativamente à primeira fase

Nesta segunda fase do trabalho prático foram implementadas diversas melhorias no código comparativamente à primeira fase devido aos avanços na proficiência com a linguagem C + +.

Para facilitar a visualização das cenas *3D* foram melhoradas as configurações existentes no *engine*.

➔ Com a tecla *F1* é possível alternar entre visualizar os eixos XYZ.

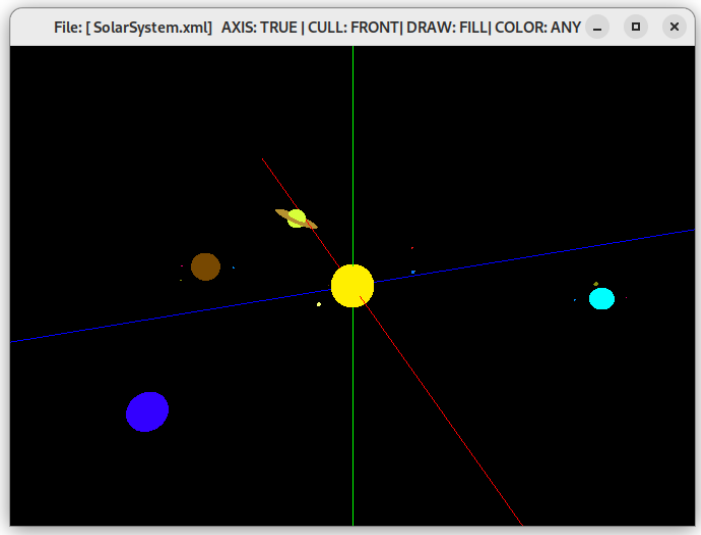

Figura 1. Eixos XYZ visíveis

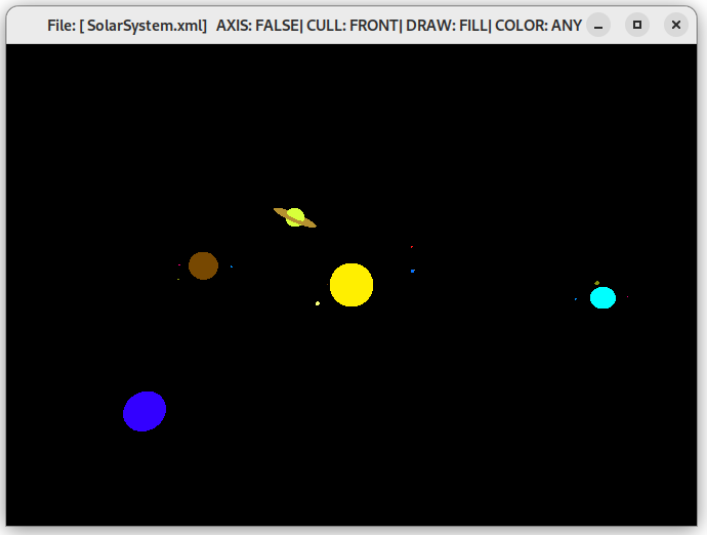

Figura 2. Eixos XYZ não visíveis

➔ Com a tecla *F2* é possível alternar o *Face Culling*(*Front Face Culling* ou *Back Face Culling*).

➔ Com a tecla F3 é possível alternar entre a vista de modo cheio ou wireframe.

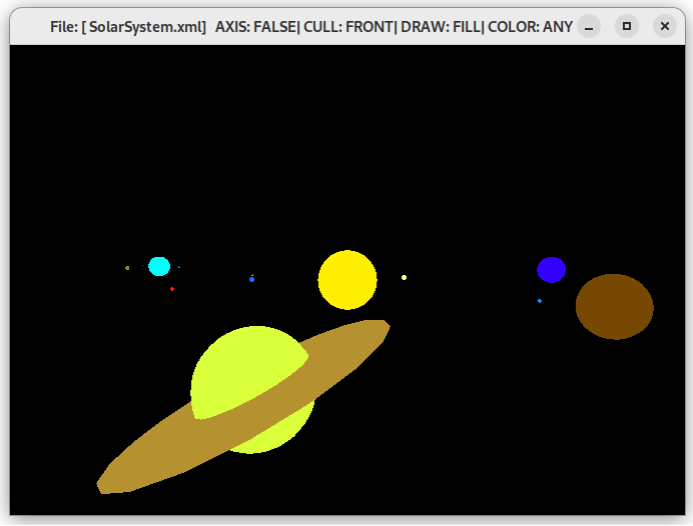

Figura 3. Desenhado com *GL_FILL*

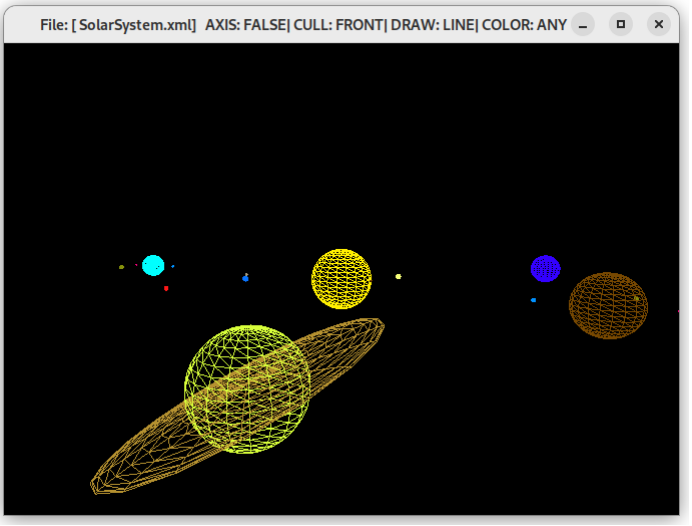

Figura 4. Desenhado com *GL_LINE*

➔ Com a tecla *F4* podemos alternar entre visualizar a cor do modelo contida no ficheiro XML (quando não estiver definida é utilizado branco por padrão) ou visualizar alternadamente cada face com uma cor diferente, como ilustrado na figura 6.

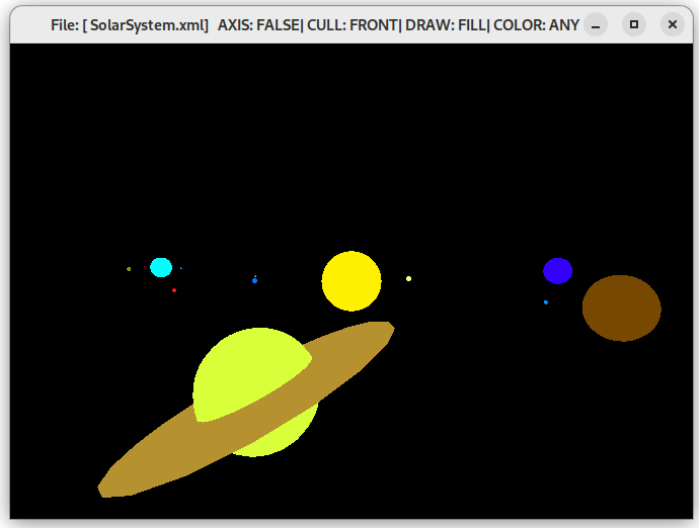

Figura 5. Desenhado com a cor contida no ficheiro XML

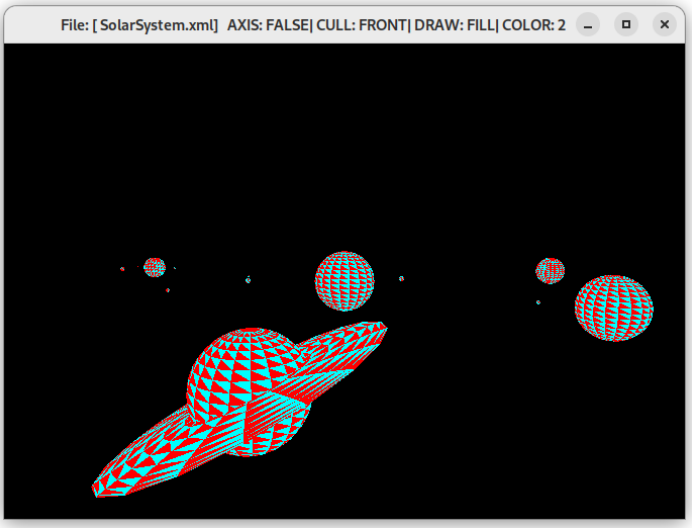

Figura 6. Desenhado com duas cores alternadas

Para facilitar a visualização das opções ativas no momento bem como o nome do ficheiro *XML* aberto no momento é mostrado ao utilizador essa informação na barra de título da janela ativa.

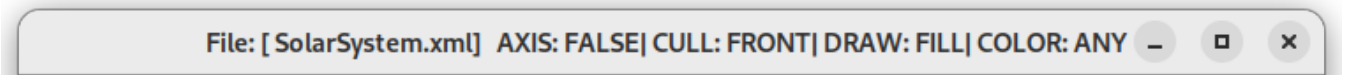

Figura 7. Título da janela ativa

3. Classes Atualizadas

3.1. Classe *Transformation*

Para implementar a funcionalidade desta segunda fase foi definida uma nova classe denominada *transformation*.

```
class Transformation{
public:
    /* 0 - Transform
     * 1 - Rotate
     * 2 - Scale
     * 3 - Color
     * */
    float color_r = 1;
    float color_g = 1;
    float color_b = 1;
    int type;
    float x;
    float y;
    float z;
    float angle;
public:
    void applyTransformation();
    std::string toString();
};
```

Figura 8. Classe *Transformation*

Esta classe contém um *int type* que identifica o tipo de transformação a aplicar, tomando o valor 0 caso se trate de uma transformação, 1 caso se trate de uma rotação, 2 no caso de uma escala e 3 caso a *tag* lida do ficheiro XML seja informação acerca da cor do objeto. A informação sobre a cor do objeto foi adicionada pelo grupo para permitir colorir os diferentes modelos, é definida no ficheiro XML dentro do elemento "*transformations*" e contém a seguinte sintaxe:

```
<color r="255" g="239" b="0" />
```

Os valores de *RGB* são depois convertidos para *floats* entre 0 e 1 e utilizados na função *glColor*

3.2. Classe Model

A classe que é responsável por guardar em memória os modelos lidos do ficheiro *XML* é a classe Model. Esta classe contém o nome do ficheiro "*.3d*" correspondente ao modelo atual, uma lista de objetos do tipo *Point* (contém as coordenadas *xyz* de um vértice) e uma lista de objetos da classe *transformation* (descrita acima) que foi adicionada nesta segunda fase.

```
class Model {
public:
    std::string filename = "NaN";
    std::list<Point> points;
    std::list<Transformation> transformations{};
    void drawModel() const;
    void printOut();
};
```

Figura 9. Classe *Model*

Foram também definidos dois métodos para esta classe, o método *drawModel* que aplica as transformações da lista *transformations* e desenha os vértices definidos na lista *points* e o método *printOut* que escreve no *stdout* o nome do ficheiro e a lista de transformações aplicadas (útil para *debug*).

3.3. Classe *CameraConfig*

A classe *CameraConfig* foi atualizada, sendo adicionado um novo método denominado *printOut* que escreve para o *stdout* os diversos campos desta classe, com o objetivo de facilitar o *debug*.

```
class CameraConfig {
public:
    float cameraX;
    float cameraY;
    float cameraZ;
    float lookAtX;
    float lookAtY;
    float lookAtZ;
    float upX;
    float upY;
    float upZ;
    float fov;
    float near;
    float far;
public:
    void printOut();
};
```

Figura 10. Classe *CameraConfig*

4. Demo Sistema Solar

Como ficheiro de teste desta segunda fase foi elaborado um ficheiro XML denominado "*SolarSystem.xml*" que contém a informação para o desenho dos planetas do sistema solar bem como as transformações a aplicar a cada planeta e a sua respectiva cor. A título de exemplo apresenta-se de seguida o conteúdo do ficheiro *XML* responsável por desenhar o planeta Terra e a Lua.

```
<!-- Earth -->
<group>
    <transform>
        <color r="9" g="116" b="255" />
        <rotate angle="276" x="0" y="1" z="0" />
        <translate x="75" y="0" z="0" />
        <scale x="2" y="2" z="2" />
    </transform>
    <models>
        <model file="sphere.3d" />
    </models>
</group>
```

Figura 11. *XML* Planeta Terra

```
<!-- Earth's Moon -->
<group>
    <transform>
        <color r="164" g="153" b="124" />
        <rotate angle="276" x="0" y="1" z="0" />
        <translate x="75" y="3" z="3" />
        <scale x="0.8" y="0.8" z="0.8" />
    </transform>
    <models>
        <model file="sphere.3d" />
    </models>
</group>
```

Figura 12. *XML* Lua

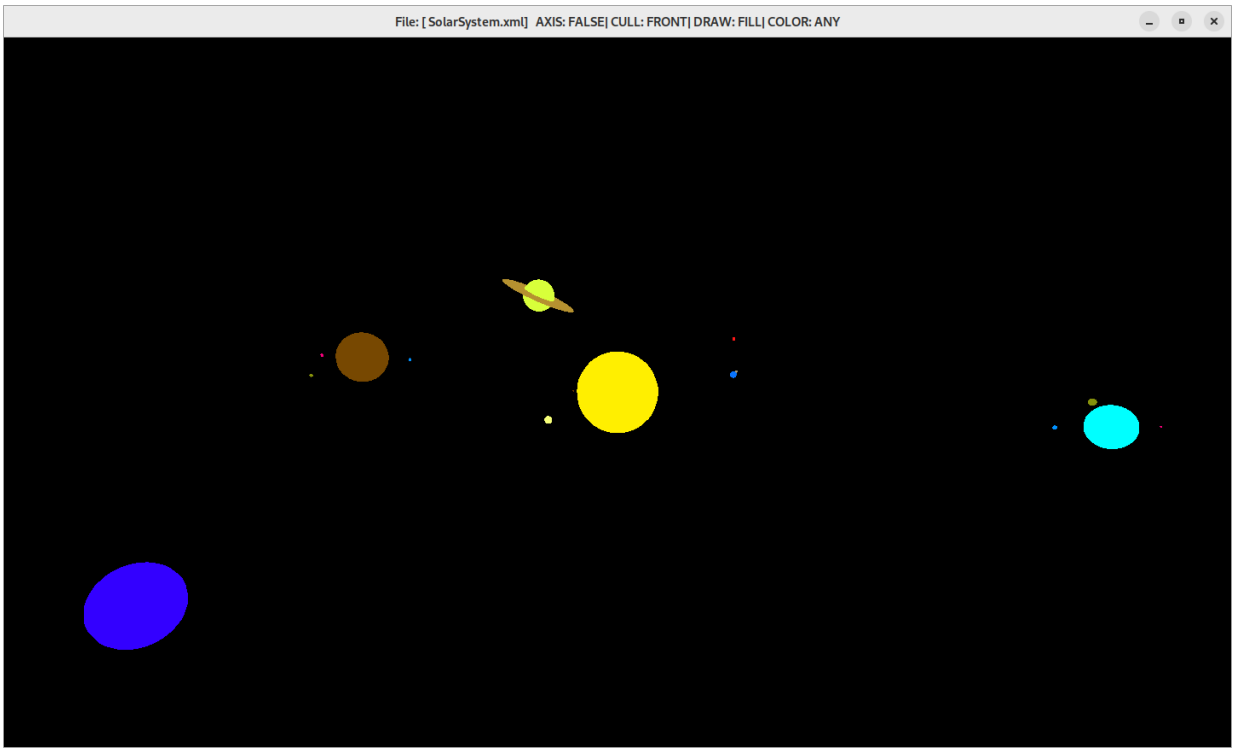

Figura 13. Sistema Solar

No *stdout* é possível visualizar as seguintes informações (resultado de invocar os métodos *printOut* das classes *Model* e *CameraConfig*). A figura abaixo representa parte do *output* obtido ao correr o engine com o ficheiro "*SolarSystem.xml*" como argumento.

```
miguel@fedora build $ ./engine SolarSystem.xml
XML File: SolarSystem.xml
---------------------------------
Camera Configurations:
Position x: -300 y: 100 z :-300
LookAt x: 0 y: 0 z :0
Up x: 0 y: 1 z :0
FOV: 60
Near: 1
Far: 1000
---------------------------------
Model Filename: sphere.3d
Transformations List:
Color x: 255.000000 y: 239.000000 z: 0.000000
Translate x: 0.000000 y: 0.000000 z: 0.000000
Scale x: 23.000000 y: 23.000000 z: 23.000000
---------------------------------
Model Filename: sphere.3d
Transformations List:
Color x: 114.000000 y: 61.000000 z: 0.000000
Rotate angle: 70.000000 x: 0.000000 y: 1.000000 z: 0.000000
Translate x: 25.000000 y: 0.000000 z: 0.000000
Scale x: 0.500000 y: 0.500000 z: 0.500000
---------------------------------
Model Filename: sphere.3d
Transformations List:
Color x: 247.000000 y: 255.000000 z: 121.000000
Rotate angle: 120.000000 x: 0.000000 y: 1.000000 z: 0.000000
Translate x: 50.000000 y: 0.000000 z: 0.000000
Scale x: 2.000000 y: 2.000000 z: 2.000000
---------------------------------
Model Filename: sphere.3d
Transformations List:
Color x: 9.000000 y: 116.000000 z: 255.000000
Rotate angle: 276.000000 x: 0.000000 y: 1.000000 z: 0.000000
Translate x: 75.000000 y: 0.000000 z: 0.000000
Scale x: 2.000000 y: 2.000000 z: 2.000000
---------------------------------
```

Figura 14. *Output* de `./engine SolarSystem.xml`

5. Conclusão

Depois de realizada a segunda fase do trabalho prático, podemos afirmar que concluímos todos os objetivos que eram esperados e ainda implementamos alguns dos objetivos extra, como a informação acerca da cor dos modelos contida no ficheiro XML.

Consideramos que houve um avanço significativo na utilização da linguagem C++ e que por esse motivo a qualidade do código desenvolvido aumentou significativamente relativamente à primeira fase deste trabalho, quer em termos de utilização de funções mais avançadas do C++ quer em termos da eficiência do código e da sua documentação.