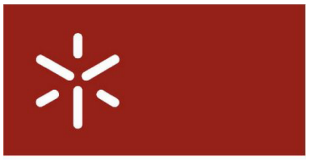
Universidade do Minho

Departamento de Informática

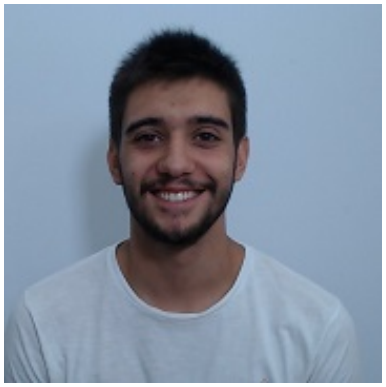


Universidade do Minho

Computação Gráfica

Fase 2 - Transformações Geométricas

Grupo 4



André Gonçalves Vieira
A90166

abril de 2022

Conteúdo

1 Introdução

1.1 Contextualização

No âmbito da Unidade curricular de Computação Gráfica, foi proposto a o desenvolvimento de modelos 3D, através da utilização de *OpenGL* com base na biblioteca *GLUT* e desenvolvido na linguagem C++.

Nesta segunda fase, de forma a dar continuidade à primeira do trabalho prático, foi necessário evoluir o projeto, criando um cenário, através de transformações geométricas. O objetivo principal é apresentar um modelo estático do Sistema Solar.

1.2 Resumo

Para esta fase do projeto, houve a necessidade de reestruturação do *paser* do ficheiro XML, de modo a permitir a utilização de transformações geométricas, como translações, rotações e escalonamentos, no desenho das primitivas gráficas efetuadas na primeira fase deste trabalho prático.

Concluindo, todas as modificações foram desenvolvidas com o objetivo de gerar as primitivas gráficas para desenhar um modelo do Sistema Solar.

2 Estrutura do Projeto

2.1 Generator

O **generator.cpp**, tal como foi referido na fase anterior, é onde estão determinadas as estruturas para cada uma das formas geométricas a representar, com o objetivo de gerar os vértices das figuras.

2.2 Engine

O **engine.cpp** contém as principais funções deste projeto, como por exemplo a interpretação e leitura dos ficheiros XML. Devido á utilização de uma nova estrutura XML houve necessidade de alterar o método utilizado para realizar o *parsing*.

Para além disso, com a introdução das transformações geométricas no desenho, foi necessário alterar a forma como a informação é armazenada. Este armazenamento organiza a informação de forma hierárquica, e o *GLUT* processa-a de maneira diferente.

2.3 Classes

Nesta fase do projeto, é necessário de interpretar uma estrutura XML um pouco mais complexa. Consequentemente, houve necessidade da criação de novas classes que serão úteis nesse sentido, tais como as classes **Scale**, **Rotate**, **Translate**, **Transformation**, **Shape** e **Group**.

2.3.1 Shape

A classe **Shape** guarda um vector de pontos proveniente de um ficheiro *.3d* . A definição da classe encontra-se no em apêndice

2.3.2 Translate

A classe **Translate** contém toda a informação que é necessária para efetuar uma translação, ou seja, variáveis de instância x, y e z, que irão descrever o vetor usado para a sua execução. A definição da classe encontra-se no em apêndice

2.3.3 Rotate

A classe **Rotate** contém o necessário para efetuar uma rotação, isto é, tal como na translação apresenta as variáveis de instância x, y, e z, que ajudam a representar o vetor usado para a efetuar a rotação, e ainda uma variável *angle* que consiste no ângulo a aplicar na rotação. A definição da classe encontra-se no em apêndice

2.3.4 Scale

A classe **Scale** contém o necessário para uma alteração de dimensões a uma figura, ou seja, é preciso uma relação entre as 3 variáveis de instância x, y e z originais e as mesmas após o redimensionamento. A definição da classe encontra-se no apêndice.

2.3.5 Transformation

A classe **Transformation** contém toda a informação para se tornar possível uma transformação geométrica ou um conjunto delas. Como variáveis de instância possui uma **Rotate** *rt*, uma **Translate** *tr* e uma **Scale** *sc*. A definição da classe encontra-se no apêndice.

2.3.6 Group

A classe **Group** guarda as informações relativas a cada grupo retirado do ficheiro XML. Assim, é possível saber as transformações efetuadas (*transformations*), que inclui translação, rotação e escala, umq **Shape** (*pontos*) e ainda os grupos inseridos nesse grupo (*groups*), sendo possível haver vários grupos dentro de um grupo. A definição da classe encontra-se no em apêndice

3 *Parsing* do Ficheiro XML e Ciclo de *Rendering*

Para o *parsing* do ficheiro XML, e tal como na primeira fase, foi utilizada a biblioteca ***TinyXML2***. Para começar o *parsing*, tenta-se abrir o ficheiro. Em caso de fracasso, a operação é abortada. Depois, tenta-se obter o elemento *world*. Mais uma vez, em caso de fracasso, aborta-se a operação.

Caso o ficheiro seja aberto com sucesso e o elemento *world* seja obtido, faz-se o *parse* do elemento *camera* percorrem-se os vários elementos *group* nele contidos, guardando-os, depois de tratados, num *vector* global.

3.1 *Parsing* de um elemento *group*

- **Elemento *transform***

  Este elemento contém as transformações geométricas dentro de cada elemento *group*. Este elemento pode conter os elementos:

  ***translate* :** Neste elemento, pretende-se obter os seus três atributos. Caso um atributo não seja encontrado, assume o valor predefinido 0. No final, a translação é adicionada ao grupo em questão.

  Caso este elemento não esteja presente, a translação é ignorada para este grupo.

  ***rotate* :** Neste elemento, pretende-se obter os seus quatro atributos. Caso um atributo não seja encontrado, assume o valor predefinido 0. No final, a rotação é adicionada ao grupo em questão.

  Caso este elemento não esteja presente, a rotação é ignorada para este grupo.

  ***scale* :** Neste elemento, pretende-se obter os seus três atributos. Caso um atributo não seja encontrado, assume o valor predefinido 1. No final, o escalamento é adicionado ao grupo em questão.

  Caso este elemento não esteja presente, o escalamento é ignorado para este grupo.

- **Elemento *models***

  Neste elemento, pretende-se obter um ou mais elementos *model*. Para cada elemento, tentamos obter o seu atributo *file* que, caso não seja encontrado, não é carregado um ficheiro.

  Caso este elemento não esteja presente, não são carregados modelos para este grupo.

- **Elemento *group***

  Em relação a este elemento, é invocada a própria função para processar todas as suas repetições, usando recursividade.

3.2 Ciclo de *Rendering*

Este ciclo foi criado com o objetivo de desenhar as figuras e sabe-se que, a cada iteração do ciclo, é feito um **getTransformations()** de modo a obter a transformação efetuada. Posteriormente, após obtermos esses dados, utilizam-se as funções **glRotatef()**, **glTranslatef()**, **glScalef()** de modo a podermos desenhar todas as transformações efetuadas. Finalmente, apenas falta percorrer o dado grupo e, para cada figura inserida neste, desenhá-la sequencialmente, de modo a formar o pretendido.

Este processo ocorre para cada grupo-filho de forma recursiva, com o objetivo de renderizar toda a informação que vem no vector de **Group** *groups*.

4 Resultados

Abaixo estão apresentadas duas figuras com o resultado dos ficheiros XML *test_2_4.xml* (fornecido pela equipa docente e gerado pelo comando "*./gen 3 3 box.3d*") e *"SistemaSolar.xml"* (gerado com o comando "*./gen sphere 1 32 32 sphere.3d*").

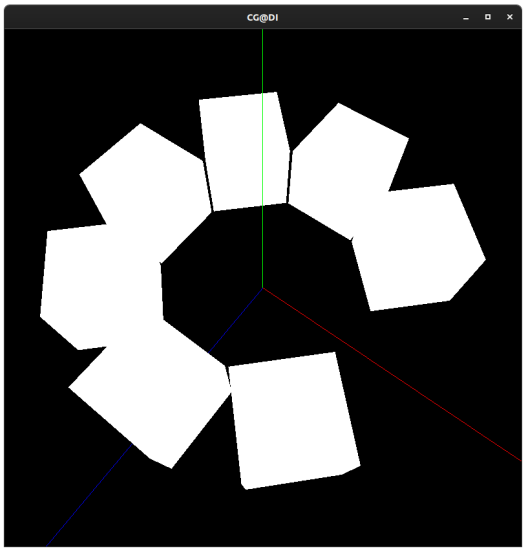


Figura 1: Ficheiro de teste formado a partir de *test_2_4.xml*

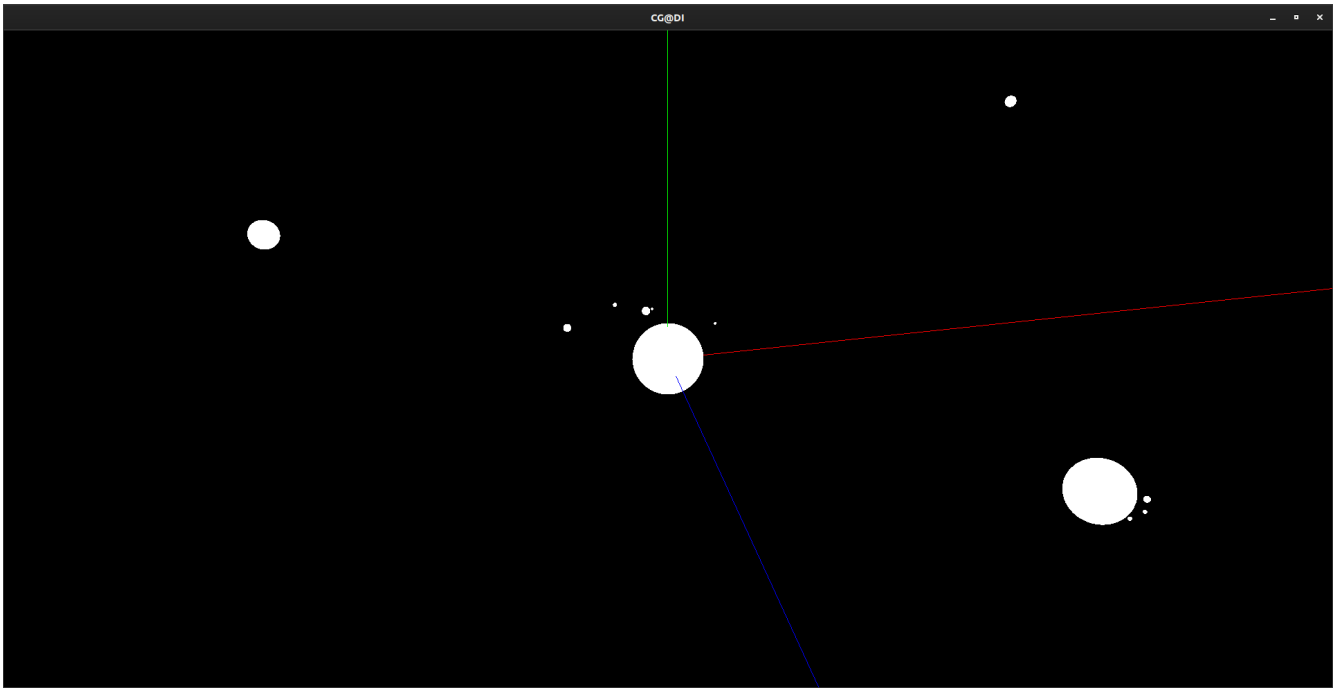


Figura 2: Ficheiro de teste formado a partir de *SistemaSolar.xml*

5 Conclusão e Trabalho Futuro

Concluída a segunda fase do trabalho prático, pode-se afirmar que foram consolidados os conceitos sobre transformações geométricas que foram abordados nas aulas. Também se ganhou mais destreza para trabalhar com a linguagem *C++* e com o *GLUT*, devido às dificuldades que foram ultrapassadas ao longo desta fase de trabalho.

A Classes

A.1 Shape

```
class Shape {
    private:
        vector<Ponto> points;
    public:
        Shape( vector<Ponto> points) {
            this->points = points;
        };
    vector<Ponto> getPoints() {return this->points;};
};
```

A.2 Translate

```
class Translate{
    private:
        float x;
        float y;
        float z;

    public:
        Translate();
        Translate (float x, float y, float z);
        float getX();
        float getY();
        float getZ();
        void setX(float a);
        void setY(float a);
        void setZ(float a);
        void add(Translate t);
};
```

A.3 Rotate

```
class Rotate{
    private:
        float angle;
```

```
        float x;
        float y;
        float z;

    public:
        Rotate();
        Rotate (float angle, float x, float y, float z);
        float getAngle();
        float getX();
        float getY();
        float getZ();
        void setAngle(float a);
        void setX(float a);
        void setY(float a);
        void setZ(float a);
        void add(Rotate r);
};
```

A.4 Scale

```
class Scale{
    private:
        float x;
        float y;
        float z;

    public:
        Scale();
        Scale (float x, float y, float z);
        float getX();
        float getY();
        float getZ();
        void setX(float a);
        void setY(float b);
        void setZ(float c);
        void add(Scale s);
};
```

A.5 Transformation

```
class Transformation{
    private:
        Rotate rt;
        Scale sc;
        Translate tr;

    public:
        Transformation();
        Rotate getRotate();
        Scale getScale();
        Translate getTranslate();
        void setTranslate(Translate tr);
        void setRotate(Rotate rt);
        void setScale(Scale sc);
};
```

A.6 Transformation

```
class Group {
    private:
        Transformation transformations;
        vector<Shape> pontos;
        vector<Group> groups;

    public:
        Group();
        void addTranslate(float x, float y, float z);
        void addRotate(float angle, float axisX, float axisY, float axisZ);
        void addScale(float x, float y, float z);
        Transformation getTransformations();
        void addShape(Shape shape);
        vector<Shape> getShape();
        void addGroup(Group group);
        vector<Group> getGroups();
};
```