



**Universidade do Minho**  
Escola de Engenharia  
Licenciatura em Engenharia Informática

## **Unidade Curricular de Computação Gráfica**

Ano Letivo de 2022/2023

# **Parte 2 - Transformações Geométricas**

Filipa Gomes (A96556) Miguel Gomes (A93294) Pedro Pacheco (A61042)  
Rita Lino (A93196)

14 de abril de 2023

# CG

# Resumo

Neste relatório explicitaremos as nossas decisões na realização da segunda fase do trabalho da Unidade Curricular de Computação Gráfica. Visando, deste modo, explicar como foram implementadas as Transformações Geométricas no *engine*, que lê a configuração dos ficheiros e renderiza os modelos.

**Área de Aplicação:** Computação Gráfica.

**Palavras-Chave:** OpenGL, C++, XML, Transformações geométricas, Primitivas gráficas.

# Índice

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Engine</b>	<b>2</b>
2.1	Transformation . . . . .	2
2.2	Scene . . . . .	2
2.3	<i>XML Parsing</i> . . . . .	3
<b>3</b>	<b>Sistema Solar</b>	<b>5</b>
3.1	Script . . . . .	5
<b>4</b>	<b>Testes</b>	<b>7</b>
<b>5</b>	<b>Conclusões e trabalhos futuros</b>	<b>9</b>
5.1	Otimizações . . . . .	9
5.2	Extras . . . . .	9
<b>6</b>	<b>Referências</b>	<b>10</b>

# Lista de Figuras

3.1	Sistema Solar - Wireframe . . . . .	6
3.2	Sistema Solar - Solid . . . . .	6
4.1	Figura de teste_2_1 . . . . .	7
4.2	Figura de teste_2_2 . . . . .	7
4.3	Figura de teste_2_3 . . . . .	8
4.4	Figura de teste_2_4 . . . . .	8

# 1 Introdução

O objetivo desta tarefa prática é desenvolver um motor 3D baseado em grafos de cena e fornecer exemplos de uso que demonstrem o seu potencial. A tarefa está dividida em quatro fases, sendo esta a segunda. Para cada fase, serão fornecidos um conjunto de ficheiros XML de configuração para fins de teste e avaliação. Cada configuração é acompanhada pela respetiva saída visual.

Neste relatório fazemos uma descrição das funcionalidades pedidas para a segunda fase do projeto. Assim sendo, esta fase envolve a implementação de transformações geométricas base no motor que por sua vez lê um ficheiro de configuração XML e exibe os modelos num ambiente de apresentação gráfica.

Esta fase requer a implementação de três Transformações Geométricas: a translação (*translate*), a rotação sobre um eixo (*rotate*) e a escala (*scale*). Foi também desenvolvido um grafo de cena para representação do Sistema Solar, sendo esta desenvolvida recorrendo a um *script* em *Python*.

Em resumo, a segunda fase deste projeto expande as bases criadas para o motor 3D baseado em grafos de cena, recorrendo a Transformações Geométricas com o objetivo final de exibir os modelos de forma visualmente atraente.

## 2 Engine

### 2.1 Transformation

Uma vez que a ordem das operações é relevante para o resultado final, estas são armazenadas num vetor pela ordem que aparecem no XML. Para tal esta classe **Transformation** baseia-se simplesmente na aplicação das transformações geométricas solicitadas no cenário.

A definição da classe **Transformation** é feita da seguinte forma.

```
1
2 enum class TransformationType {
3     None,
4     Translate,
5     Rotate,
6     Scale
7 };
8
9 class Transformation {
10     public:
11         TransformationType type;
12         int angle;
13         float x;
14         float y;
15         float z;
16
17         Transformation(TransformationType type, float x, float y, float z, int
angle);
18         Transformation();
19         auto show() -> string;
20         void apply();
21 };
```

Após alguma reflexão chegamos a conclusão que esta implementação não seria a mais correta, tendo como objetivo a correção da mesma para a próxima fase do projeto.

### 2.2 Scene

Para facilitar o armazenamento da cena e dos seus componentes foram desenvolvidas a classe **Scene** e suas classes descendentes **Group**.

```
1 class Scene {
2
3     public:
4         Camera camera;
5         Group group;
6 }
```

```

7   Scene(Camera camera, Group group);
8   Scene();
9
10  auto render(bool picker = false) -> void;
11  auto load_models() -> void;
12 };
13
14 class Group {
15
16     public:
17         vector<Group> subgroups;
18         vector<Model> models;
19         vector<Transformation> transformations;
20
21         Group(vector<Group> subgroups, vector<Model> models, vector<
Transformation> transformations);
22         Group();
23
24         //(...)
25
26         auto applyTransformations();
27         auto render(bool picker = false) -> void;
28         auto load_models() -> void;
29 };

```

## 2.3 XML Parsing

Uma vez que o **Parser** já estava desenvolvido para a realização da primeira fase deste projeto foi apenas necessário a *atualização* de como os campos eram interpretados. Com esse objetivo o **Parser** original foi redesenhado com o intuito de o tornar mais modular e para suportar *parsing* de subgrupos e das respectivas transformações.

O método responsável pelo *parsing* de um Grupo foi definido da seguinte forma:

```

1  auto Parser::parse_group(XMLElement* element_group) -> Group {
2      Group group;
3      XMLElement* child_element = element_group->FirstChildElement();
4      while (child_element != nullptr){
5          if (strcmp(child_element->Name(), "group") == 0){
6              group.addSubgroup(parse_group(child_element));
7          }
8          else if (strcmp(child_element->Name(), "models") == 0){
9              group.models = parse_models(child_element);
10         }
11         else if (strcmp(child_element->Name(), "transform") == 0){
12             group.transformations = parse_transformations(child_element);
13         }
14         child_element = child_element->NextSiblingElement();
15     }
16     return group;
17 }

```

Este é dependente de outros 2 métodos: *parse\_models* e *parse\_transformations*, mas também de si mesma para suportar subgrupos.

```
1 auto Parser::parse_transformations(XMLElement* element_transform) -> vector<
  Transformation>{
2   vector<Transformation> transformations;
3   TransformationType type;
4   float x, y, z;
5   int angle;
6   XMLElement* child_element = element_transform->FirstChildElement();
7   while (child_element != nullptr){
8       if (strcmp(child_element->Name(), "translate") == 0){
9           type = TransformationType::Translate;
10          x = atof(child_element->Attribute("x"));
11          y = atof(child_element->Attribute("y"));
12          z = atof(child_element->Attribute("z"));
13      }
14      else if (strcmp(child_element->Name(), "rotate") == 0){
15          type = TransformationType::Rotate;
16          angle = atoi(child_element->Attribute("angle"));
17          x = atof(child_element->Attribute("x"));
18          y = atof(child_element->Attribute("y"));
19          z = atof(child_element->Attribute("z"));
20      }
21      else if (strcmp(child_element->Name(), "scale") == 0){
22          type = TransformationType::Scale;
23          x = atof(child_element->Attribute("x"));
24          y = atof(child_element->Attribute("y"));
25          z = atof(child_element->Attribute("z"));
26      }
27      transformations.push_back(Transformation(type, x, y, z, angle));
28      child_element = child_element->NextSiblingElement();
29  }
30  return transformations;
31 }
32
33 auto Parser::parse_models(XMLElement* models_element) -> vector<Model>{
34   vector<Model> models;
35   XMLElement* model_element = models_element->FirstChildElement();
36   while (model_element != nullptr){
37       string file = model_element->Attribute("file");
38       string name = model_element->Attribute("name") ? model_element->
  Attribute("name") : file;
39       models.push_back(Model(file, name));
40       model_element = model_element->NextSiblingElement();
41   }
42   return models;
43 }
```



## 3 Sistema Solar

### 3.1 Script

Com o auxílio de algumas fontes incluídas nas **Referências** deste relatório relativas às dimensões do Sol, Planetas e os seus Satélites naturais correspondentes, e das distâncias entre si, foi produzido um *script* que permite gerar um ficheiro XML cuja interpretação feita pelo *engine* resulta no Sistema Solar.

De maneira que as distâncias e dimensões dos astros e planetas fossem mais realísticas, foi necessário realizar uma escala possibilitando a sua visualização.

A indentação típica de um ficheiro XML foi possível devido ao uso de várias *libraries* da linguagem de Python, a leitura dos dados dos astros presentes em ficheiros **.csv** e a aleatoriedade da colocação dos asteroides presentes na cintura de asteroides.

Um ficheiro XML foi gerado com o intuito de armazenar todos os planetas e respectivos satélites naturais, bem como todos os (500) asteroides que constituem a cintura de asteroides e o anel de Saturno (*toro*).

É utilizado um ficheiro **.3d**, para a otimização da geração do Sistema Solar, para elementos de menor dimensão, com uma esfera de com baixo número de stacks e slices. Visando criar um elemento Sol realista, utilizou-se um ficheiro **.3d** de uma esfera com um enorme número de stacks e slices de modo que seja o mais parecido com uma esfera redonda.

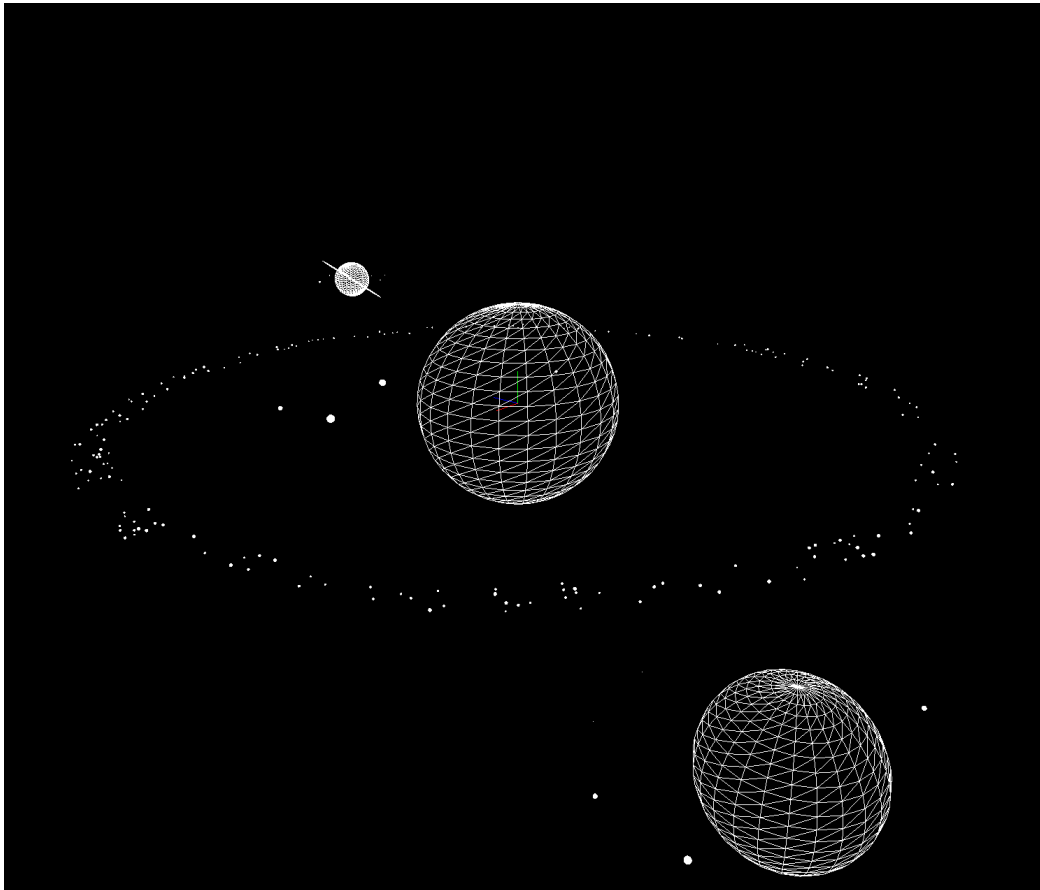


Figura 3.1: Sistema Solar - Wireframe

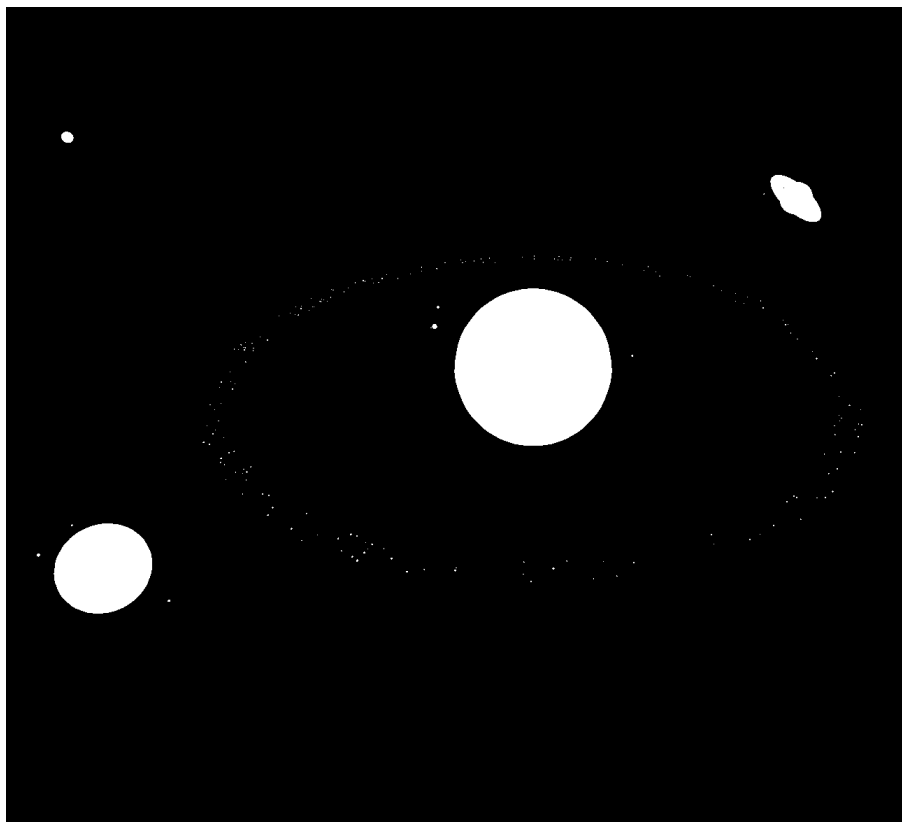


Figura 3.2: Sistema Solar - Solid

## 4 Testes

Por meio de ficheiros disponibilizados é possível fazer a validação da implementação das **Transformações Geométricas** usando os cenários de teste. Seguem os cenários resultantes dos mesmos:

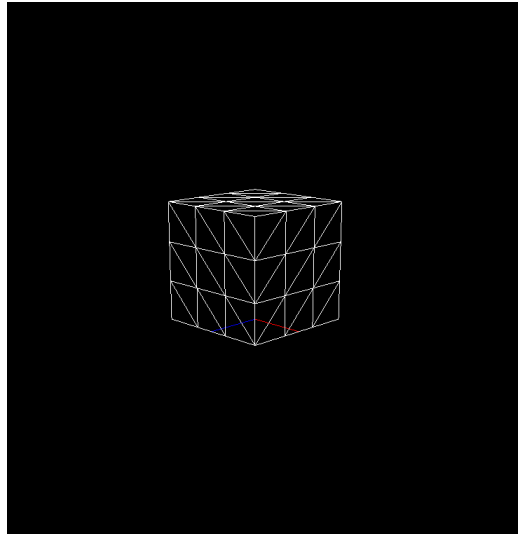


Figura 4.1: Figura de teste\_2\_1

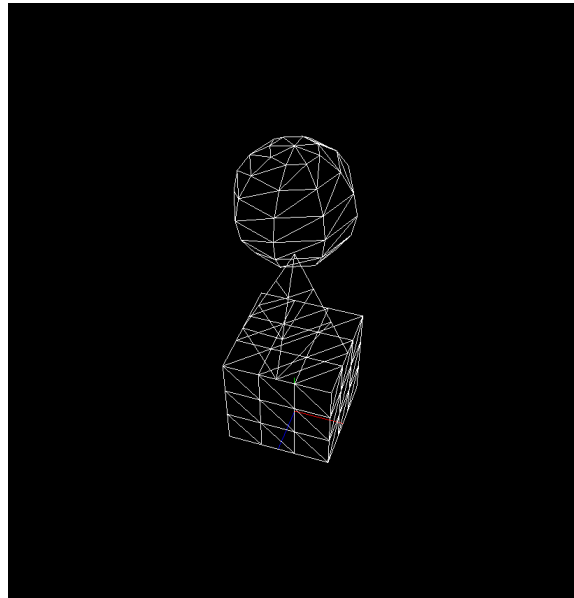


Figura 4.2: Figura de teste\_2\_2

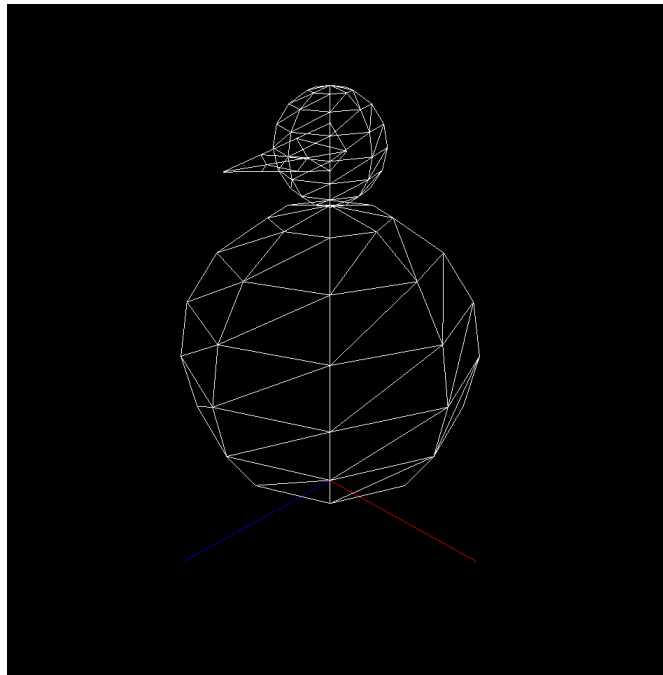


Figura 4.3: Figura de teste\_2\_3

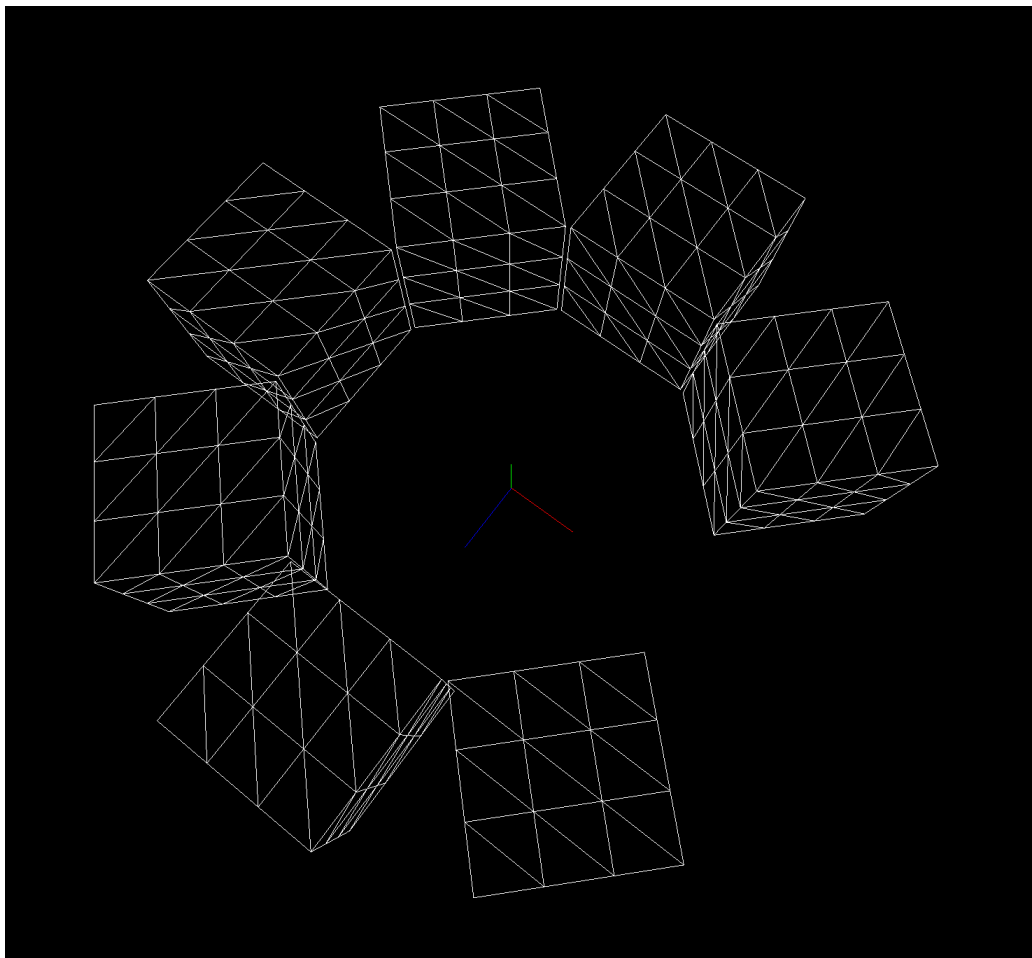


Figura 4.4: Figura de teste\_2\_4

## 5 Conclusões e trabalhos futuros

A realização desta fase do projeto permitiu-nos consolidar os conhecimentos adquiridos durante as aulas acerca de transformações de modelos 3D, matrizes e câmaras. Com isto, fomos capazes de construir um modelo inicial do sistema solar, que vai ser necessário nas próximas fases deste trabalho prático.

Considera-se que foi efetuado um bom trabalho, e que o programa implementa uma boa variedade de funcionalidades. No entanto, no futuro, será melhorado relativamente ao seu aspeto estético, utilizando texturas e iluminação e no aspeto funcional com a implementação de otimizações e funcionalidades extra.

### 5.1 Otimizações

O cenário do *Sistema Solar* contém centenas de astros, o que leva ao aparecimento de *bottlenecks* que não estiveram em consideração durante a implementação do *engine* na primeira fase e na expansão do mesmo nesta fase. O maior *bottleneck* encontrado foi a leitura dos ficheiros *model* uma vez que o mesmo ficheiro era lido centenas de vezes. Esse problema produz também uma elevada utilização de memória. Como implementação para a próxima fase pretendemos corrigir esses *bottlenecks* e quais queres outros que apareçam durante o desenvolvimento. Além disso, pretendemos trocar o vetor de pontos dos modelos por buffers armazenados na placa gráfica (*VBOs*).

### 5.2 Extras

As funcionalidades extra ainda em desenvolvimento são suporte de cores por modelo, *Pick and Click* para identificação de vários modelos com o nome desse grupo/modelo com utilização do rato e a criação de uma câmara *FPV*. Pretendem-se que estas funcionalidades estejam implementadas completamente até a próxima fase deste projeto.

## 6 Referências

1. **Dados sobre planetas e luas (NASA).** (04/04/2022).
2. **Distancia relativa entre planetas e luas (National Geographic).** (04/04/2022).
3. **Referência para o *'script'* gerador do *Sistema Solar* (Github).** (12/04/2023).