Universidade do Minho

Departamento de Informática

Computação Gráfica

Phase 2 – Geometric Transforms

Grupo 39

Gonçalo Araújo Brandão A100663
Maya Gomes A100822
Luís de Castro Rodrigues Caetano A100893

5 de abril de 2024

Conteúdos

1 Introdução

A produção dos modelos e primitivas desta fase engloba diversas transformações geométricas, tal como rotações, translações e escalas. Com o intuito de solidificar a aprendizagem dos mesmos, foi proposta a realização da segunda fase do projeto prático.

Este relatório aborda a execução da segunda fase do mencionado projeto.

É crucial que os objetivos mencionados sejam alcançados com êxito. Por essa razão, o formato do relatório foi estruturado de maneira a abordar inicialmente a descrição do problema, seguida por algumas melhorias das primitivas (fase 1), isto é, da criação de uma nova primitiva, culminando nos aspetos essenciais relativos ao Engine. Além disso, optamos por incluir uma secção dedicada à descrição da criação do xml do Sistema Solar.

2 Descrição do problema

A segunda fase deste projeto visa o aprimoramento do *Engine* desenvolvido na fase inicial. O objetivo é introduzir novas funcionalidades para processar transformações geométricas, como translações, escalas e rotações. Para além disso, nesta etapa, o arquivo de configuração deve conter um modelo estático do sistema solar.

Assim como na primeira fase, a estrutura do projeto é mantida, tendo se então as seguintes diretorias: *Generator, Engine e Models*.

- **Generator**: Foi criada uma nova primitiva denominada de arco (*ring*).
- **Engine**: compreende um conjunto de funções e estruturas dedicadas à leitura dos ficheiros XML e à representação gráfica de cada modelo.
- **Models**: Esta diretoria abriga os arquivos .3d e XML, onde os modelos são armazenados.

3 Melhorias da primeira fase

Considerando que nesta fase será necessário a implementação do sistema solar procuramos criar mais uma primitiva de forma a conseguirmos representar um pouco melhor o planeta Saturno. O nosso objetivo foi criar um genero de anel de forma a depois, junto com uma esfera mais pequena, representar o planeta Saturno.

3.1 *Ring* - arco para Saturno.

De forma a desenvolvermos uma coroa circular necessitou-se de atualizar o *Generator* desenvolvido na primeira fase. Desta forma, foi criada a função *createRing* e adaptou-se a main de forma a possibilitar a criação de um ficheiro .3d com os vertices de um anel.

Com um raciocinio semelhante aos das primitivas da primeira fase temos que o anel recebe as medidas de um raio (*radius*), um raio externo (*outradius*) e um número de *slices* pelo qual vai ser dividido.

Anel: coroa circular no plano xz, centrado na origem: **ring radius outradius slices file.3d**

Sendo *inradius* o raio interno do anel, temos que:

$$\alpha = 2\pi \div slicesa, a = [0, slices[$$
$$\text{inradius} = \text{outradius} - \text{radius}$$

Para cada segmento, calcula-se as coordenadas (x e z) dos quatro pontos (pt1, pt2, pt3 e pt4) que formam dois triângulos adjacentes. Os pontos pt1 e pt4 pertencem ao círculo interno, enquanto pt2 e pt3 pertencem ao círculo externo. Sendo que para esses pontos temos que:

$$\alpha 1 = 2\pi \div slicesa$$
$$\alpha 2 = 2\pi \div slices(a+1)$$

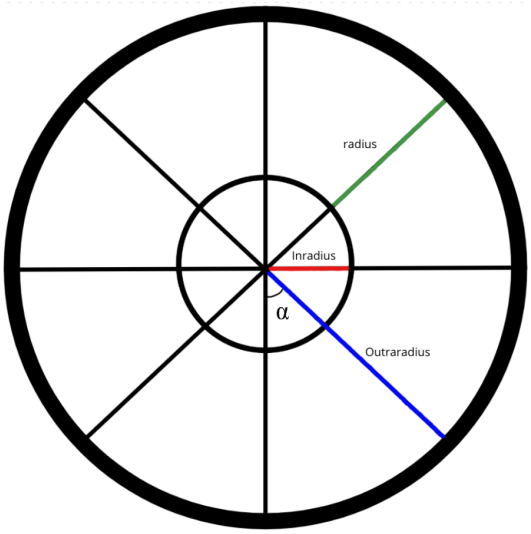

Figura 1: Esquema da divisão do anel em *slices*.

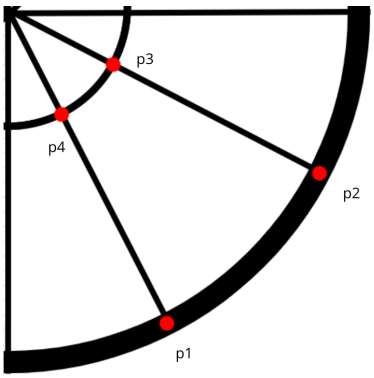

Figura 2: Esquema da divisão do anel em *slices*.

Desta forma, para os pontos da Figura 2, os valores das coordenadas x e z dos pontos 1, 2, 3 e 4, vão ser obtidos pelas equações seguintes:

$$\text{x1} = \text{outradius} * \text{sinf}(\alpha 1);$$
$$\text{x2} = \text{outradius} * \text{sinf}(\alpha 2);$$
$$\text{x3} = \text{inradius} * \text{sinf}(\alpha 2);$$
$$\text{x4} = \text{inradius} * \text{sinf}(\alpha 1);$$
$$\text{z1} = \text{outradius} * \text{cosf}(\alpha 1);$$
$$\text{z2} = \text{outradius} * \text{cosf}(\alpha 2);$$
$$\text{z3} = \text{inradius} * \text{cosf}(\alpha 2);$$
$$\text{z4} = \text{inradius} * \text{cosf}(\alpha 1);$$

3.1.1 Resultados

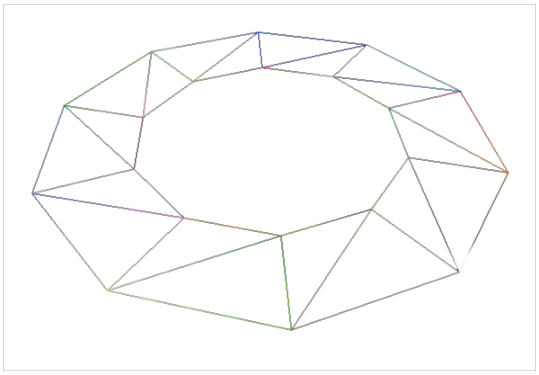

Figura 3: Ring sem preenchimento.

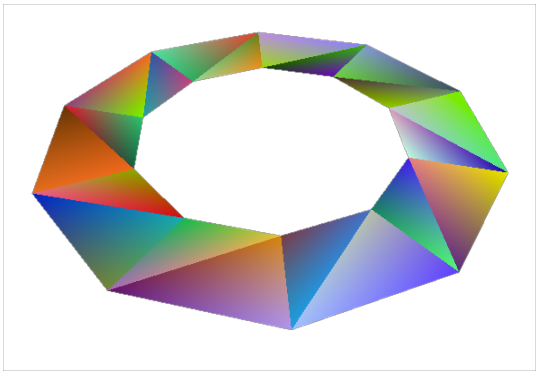

Figura 4: Ring com preeenchimento.

4 Estrutura da aplicação - Segunda Fase

4.1 Resolução do problema

4.1.1 Novas estruturas

Nesta fase do trabalho, sendo que os ficheiros xml fornecidos eram diferentes, foi necessário adaptarmos as funções de *parsing*. Desta forma, também surgiu a necessidade de armazenarmos mais informações, pois esta fase já inclui transformações geométricas.
Numa primeira fase, criou-se a estrutura *Transform* que representa uma transformação geométrica:

```
struct Transform {
    float angle, x, y, z;
    Transform() : angle(0), x(0), y(0), z(0) {}
};
```

Figura 5: Estrutura *Transform*.

Criou-se também a estrutura *Transforms* que representa as transformações geometricas aplicadas a um *group*:

```
struct Transforms {
    Transform scale;
    Transform translate;
    Transform rotate;
```

Figura 6: Estrutura *Transforms*.

De seguida, criou se uma estrutura *Model* e uma estrutura *Group* para os respetivos parâmetros existentes nos ficheiros XML:

```
struct Model {
    string file;
    Transform translate;
};
```

Figura 7: Estrutura *Model*.

```
struct Group {
    Transforms transform;
    vector<Model> models;
    vector<Group> groups;
};
```

Figura 8: Estrutura *Group*.

As estruturas da primeira fase, como a World ou a Point por exemplo, mantêm-se.

4.1.2 Pasing dos ficheiros XML

Para relizar o parsing dos ficheiros XML criamos a função *parseXML* na qual, numa primeira fase, lemos e guardamos os elementos da "world", "window", "camera", "position", "lookAt", "up"e "projection". De seguida é chamada a função auxiliar *parseGroups* de forma a lermos e guardarmos os elementos dos grupos do XML.

A função *parseGroups* itera sobre os diversos *groups* do xml aplicando a função *parseGroup*.

A função *parseGroup* é responsável por analisar um *group*. Assim, primeiramente lê-se as transformações do grupo guardando as mesmas no parametro *Transforms transform* da estrutura *Group*. Estas são guardadas com base no seu tipo, podendo ser: *translate, scale ou rotate*.

De seguida, são analisados os *models* do *group*. Aí é guardado o "file", neste caso o ficheiro .3d, que contem os pontos da nossa figura. Tal como na primeira fase, para a analisar os ficheiros .3d utilizamos a função *readFile*.

De seguida é efetuada a recursiva de forma a averiguar se há mais *groups* dentro do primeiro *group* aplicando a recursiva de parseGroup aos mesmos.

De forma a guardar os grupos pela ordem certa (para serem desenhados na mesma ordem) utilizamos um *boolean* "primeiroGrupo"de forma a irmos guardando o primeiro grupo, ou seja, o grupo pai, primeiro e os grupos filhos depois.

4.1.3 Desenho das primitivas

De forma a serem desenhadas as primitivas criamos a função *drawGroups* que percorre os diversos *groups* dos diversos *models* de forma a serem desenhados os pontos.

A função recebe um parâmetro do tipo *Group*, que contém as informações sobre as transformações e os modelos dentro desse mesmo grupo. Numa primeira fase é feito *glPushMatrix()* de forma a que as transformações aplicadas dentro do grupo não afetem os outros grupos. De seguida são efetuados as funções *glRotatef(), glTranslatef() e glScalef()*, de forma a aplicar respetivamente uma rotação, uma translação e uma escala ao grupo de acordo com os parâmetros armazenados no objeto referido anteriormente.

De seguida, é criado um loop que itera sobre os modelos do grupo. Dentro do loop é chamada a função auxiliar drawPoints(). Esta função permite-nos desenhar os pontos, assim é utilizado o *glBegin(GL_TRIANGLES)* definindo os triangulos. É criado um loop que itera sobre todos os pontos do modelo e usando a *glColor3f()* e a *glVertex3f()* é atrbuída uma cor aleatória e é definido um vértice para o *OpenGL* com as respetivas coordenadas.

No fim da *drawGroups*, usa-se a função *glPopMatrix()*.

4.1.4 Resultados

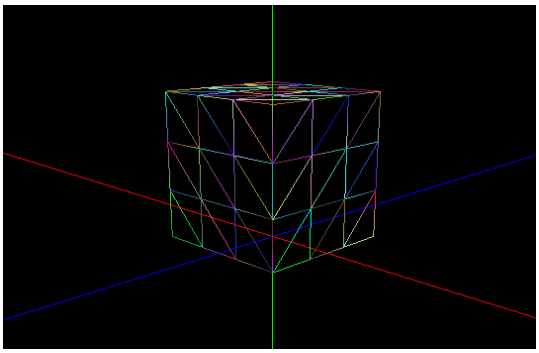

Figura 9: Caixa sem preenchimento.

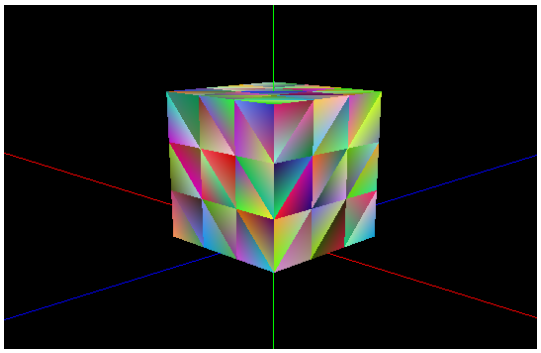

Figura 10: Caixa com preenchimento.

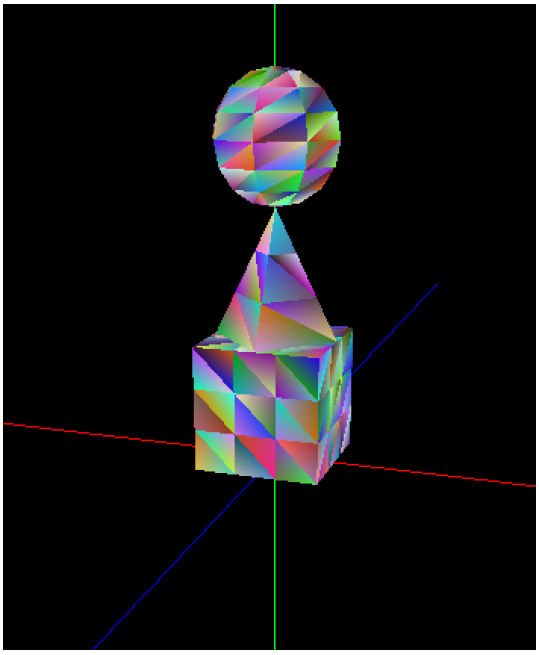

Figura 11: Caixa, cone e esfera sem preenchimento.

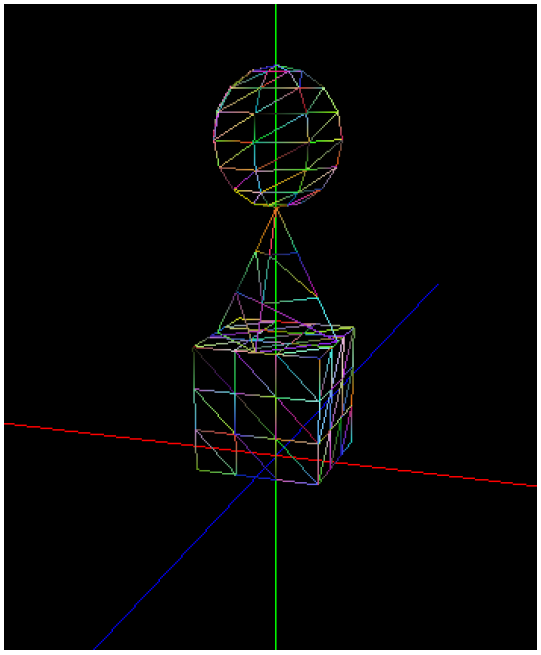

Figura 12: Caixa, cone e esfera com preeenchimento.

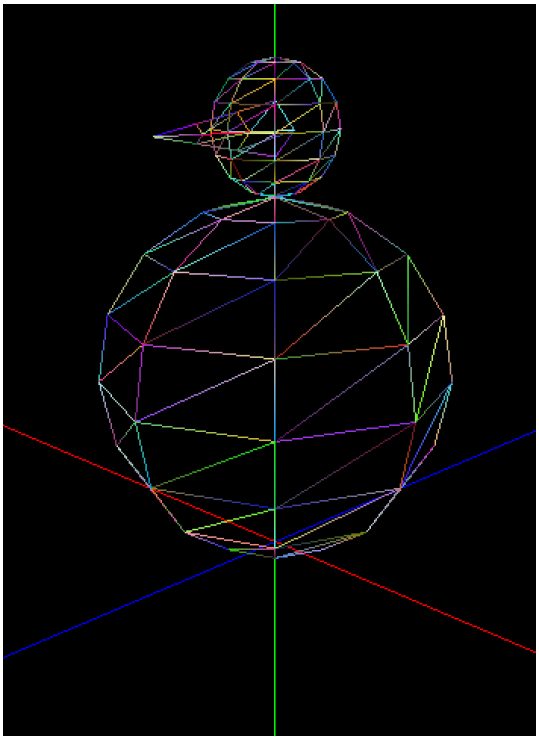

Figura 13: Snow Man sem preenchimento.

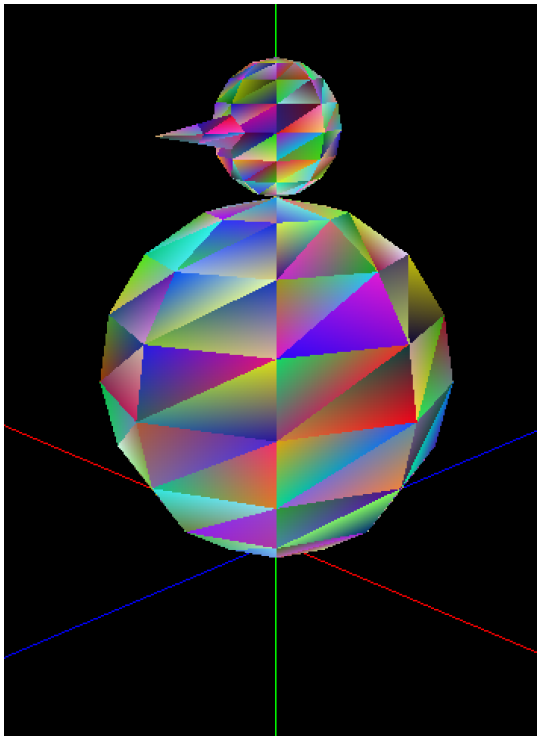

Figura 14: Snow Man com preeenchimento.

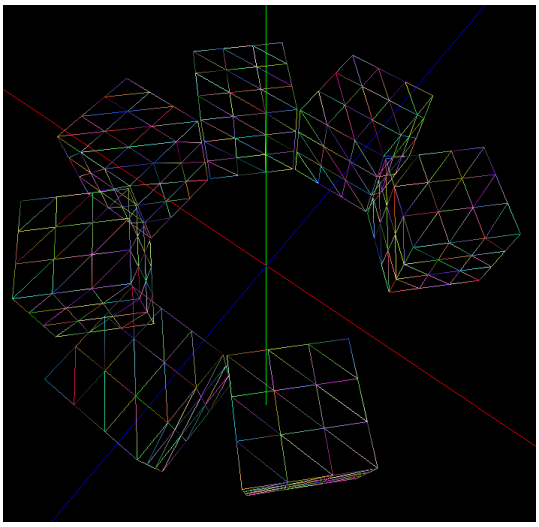

Figura 15: Caixas sem preenchimento.

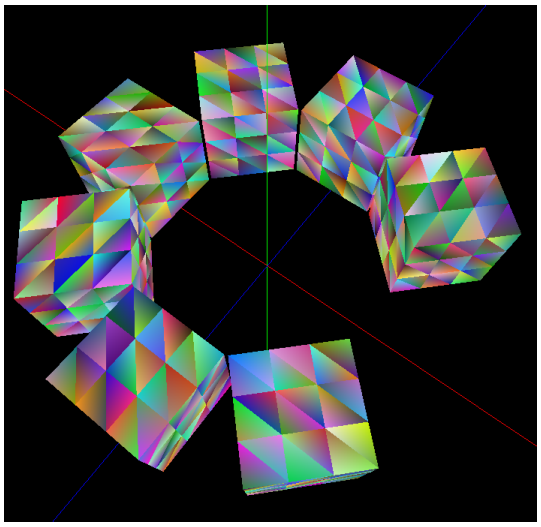

Figura 16: Caixas com preeenchimento.

5 Sistema Solar

Antes de desenvolver o modelo do sistema solar é preciso ter em consideração algumas propriedades:

- o translate de um grupo é somado ao do subgrupo;
- o valor do scale de um grupo é multiplicado ao valor do subgrupo;
- o valor do rotate do grupo é somado ao do subgrupo.

As figuras utilizadas serão a esfera e o arco. Como sabemos, estas primitivas possuem dimensões e posições fixas, pelo que, será necessário recorrer a transformações geometricas para alterar a orientação, a dimensão e a localização das mesmas relativamente ao referencial. Desta forma, iremos recorrer a escalas, rotações e translações de modo a representarmos as figuras o mais próximo da realidade possível.
Para compor o ficheiro xml, foi necessário percebermos as distâncias dos planetas em relação ao Sol e a da Lua em relação ao planeta Terra e o diâmetro de cada elemento do modelo.

5.1 Ficheiro xml

Neste ficheiro, definimos as dimensões de cada planeta, procurando respeitar as dimensões reais aplicando uma escala proporcional. Utilizamos o valor 200 para representar a maior distância entre um planeta e o Sol, ou seja, entre Neptuno e o Sol. Contudo, como as dimensões eram demasiado pequenas, no resultado final não era possível distinguir certos planetas. Foi, então, necessário aumentar estes valores. Procuramos, no entanto, manter os planetas que são menores/maiores.
Para além disso, também percebemos que a distância entre os planetas teria de ser calculada pela distância entre os centros dos mesmos.

De seguida, procuramos elaborar o planeta Saturno, pois sendo composto por uma esfera e um arco poderia ser mais complexo. Assim, o *group* de Saturno será composto por dois subgrupos, um para a esfera e outro para o anel.
Para o anel, determinamos o valor da rotação do mesmo, considerando o eixo de rotação e a inclinação orbital. Para isso, aplicamos uma rotação no eixo do z com um ângulo de inclinação para o anel. A ideia era ficar inclinado de forma a termos uma melhor visualização. Utilizamos uma inclinação de 20^{o}C.

Voltando as escalas, com base nos raios reais de cada planeta procuramos valores que tivessem uma certa proporção. No entanto, como referido anteriormente essa proporção não esta adequada a vida real, pois para ser mais percetível necessitámos de adaptar os valores.

Observando os valores dos raios conseguimos imediatamente perceber que poderíamos dividir os planetas em vários escalas. Os primeiros quatro planetas (Mercúrio,Vênus, Terra e Marte) estão com raios dentro de um certo intervalo enquanto que os restantes estão dentro de outro intervalo. Assim, a nossa ideia foi atribuir uma escala igual ou semelhante nos planetas com raio homogêneo. O intervalo do primeiro grupo seria menor e o do segundo maior, fazendo com que os valores do segundo grupo relativamente ao primeiro grupo fossem também maiores.

Numa primeira fase, escolhemos como valor de referencia 0,8 para o Sol (por ser perto de 1, sendo que o sol é bastante grande). De seguida, pegamos nos valores dos raios dos planetas e com uma regra de três simples fomos calculando o respetivo valor de cada um. Desta forma respeitávamos uma certa proporção entre os planetas e era utilizada a mesma escala para todos. No entanto, como podemos reparar na tabela abaixo, ao chegar a planetas como Júpiter, isto é, planetas maiores, atingíamos novamente o valor 0,8. Este valor não podia representar Júpiter, pois já estava a ser utilizado para o Sol. Júpiter é ainda bastante menor que o Sol, por isso escolhemos uma nova escala para estes planetas maiores. Assim, para Júpiter atribuímos o valor de 0,2 calculando o valor correspondente para Saturno. Em relação a Urano e Neptuno, os raios encontram se num intervalo de valores bastantes semelhantes, pelo que usamos a mesma escala para ambos.

Tipo	**Nome**	**Raio** (km)	**Scale**	**Scale** $\times 10$	**Scale** $\times 10 \times 0.25$	**Scale** $\times 10 \times 0.4$
Estrela	Sol	695 000	0.8			
Planeta	Mercúrio	2 440	0.0028	0.028		
Planeta	Vênus	6 052	0.0069	0.069		
Planeta	Terra	6 371	0.0073	0.073		
Satélite	Lua	1 737	0.002	0.02		
Planeta	Marte	3 390	0.0039	0.039		
Planeta	Júpiter	69 911	0.08	0.8	0.2	
Planeta	Saturno	58 232	0.067	0.67	0.1675	
Planeta	Urano	25 362	0.0291	0.291	0.07	0.12
Planeta	Netuno	24 622	0.0283	0.283	0.07	0.11

Tabela 1: Scale das figuras.

Relativamente aos *translate*, mais uma vez, a nossa ideia foi escolher um valor de referência. Assim, para a primeira distância, ou seja, a distância entre o Sol e Mercúrio escolhemos o valor 1. De seguida, com a regra de 3 simples fomos escolhendo os valores para todas as distâncias entre os planetas. Mais uma vez, alguns valores não nos convinham, pelo que, fomos ajustando como podemos ver na tabela abaixo.

É de notar que também as translações aplicadas a um planeta são aplicadas ao seu respetivo satélite. Ou seja, no caso da Lua avaliamos a distância com a Terra e não o sol. Desta forma decidimos aplicar um translate de 0.1 somente.

Tipo	**Nome**	**Distância com o Sol**	**Translate**	**Translate 1 - 155 000 000**	**Translate 1 - 360 000 000**	**Translate 1 - 500 000 000**
Estrela	Sol	0	0			
Planeta	Mercúrio	56 847 190	1.0			
Planeta	Vênus	107 710 466	1.895			
Planeta	Terra	149 597 870	2.632			
Planeta	Marte	227 388 763	4.0			
Planeta	Júpiter	777 908 927	13.68	5.018		
Planeta	Saturno	1 421 179 771	25.0	9.168		
Planeta	Urano	2 872 279 117	50.52	18.53	7.979	
Planeta	Netuno	4 487 936 121	78.94	28.954	12.466	8.976

Tipo	**Nome**	**Distância com a Terra**	**Translate**
Satélite	Lua	394 400	0.1

Tabela 2: Translate das figuras.

No caso do anel de Saturno, como este está no grupo do planeta vai herdar a translação do mesmo e vai estar centrado. Sendo assim, basta aplicar uma *scale* de modo a que este permaneça ligeiramente afastado de Saturno. Para efeitos estéticos acrescentamos um *rotate* de forma a obter uma ligeira inclinação do arco, como referido anteriormente.

5.1.1 Resultado

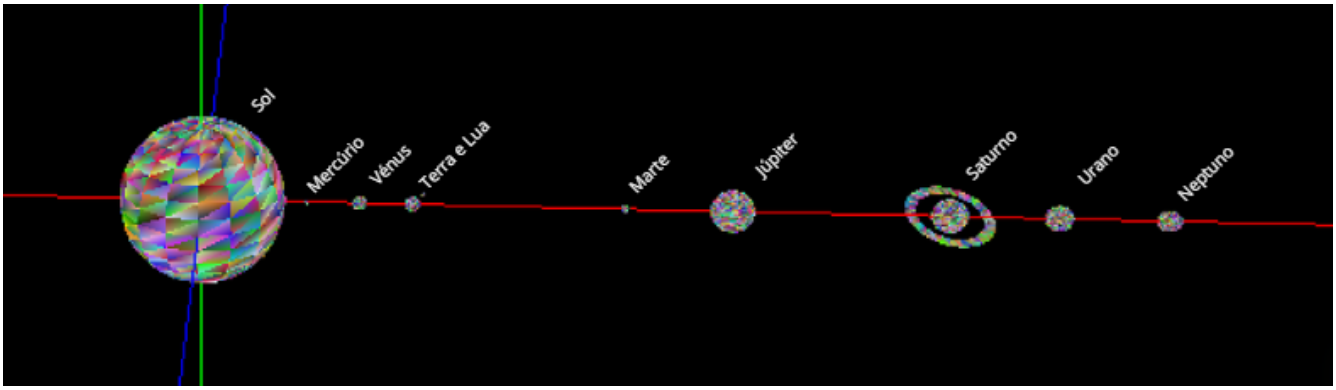

Figura 17: Resultado do xml do Sistema solar.

Finalmente procuramos aplicar rotações, pois na verdade os planetas não se encontram perfeitamente alinhados. Para isso, a nossa ideia foi aplicar um *rotate* sobre o eixo do z. Desta forma, definimos o valor da rotação de cada planeta de acordo com o eixo de rotação e a inclinação orbital. Como sabemos, quando o ângulo é positivo para o lado esquerdo, é necessário aplicar o valor negativo ao ângulo para a inclinação se verificar para o lado direito.

Nome	**Inclinação em Graus**	**Rotate**
Mercúrio	0.1	-0.1
Vênus	177	-177
Terra	23.5	-23.5
Marte	25	-25
Júpiter	3	-3
Saturno	27	-27
Urano	98	-98
Netuno	30	-30

5.1.2 Resultado Final

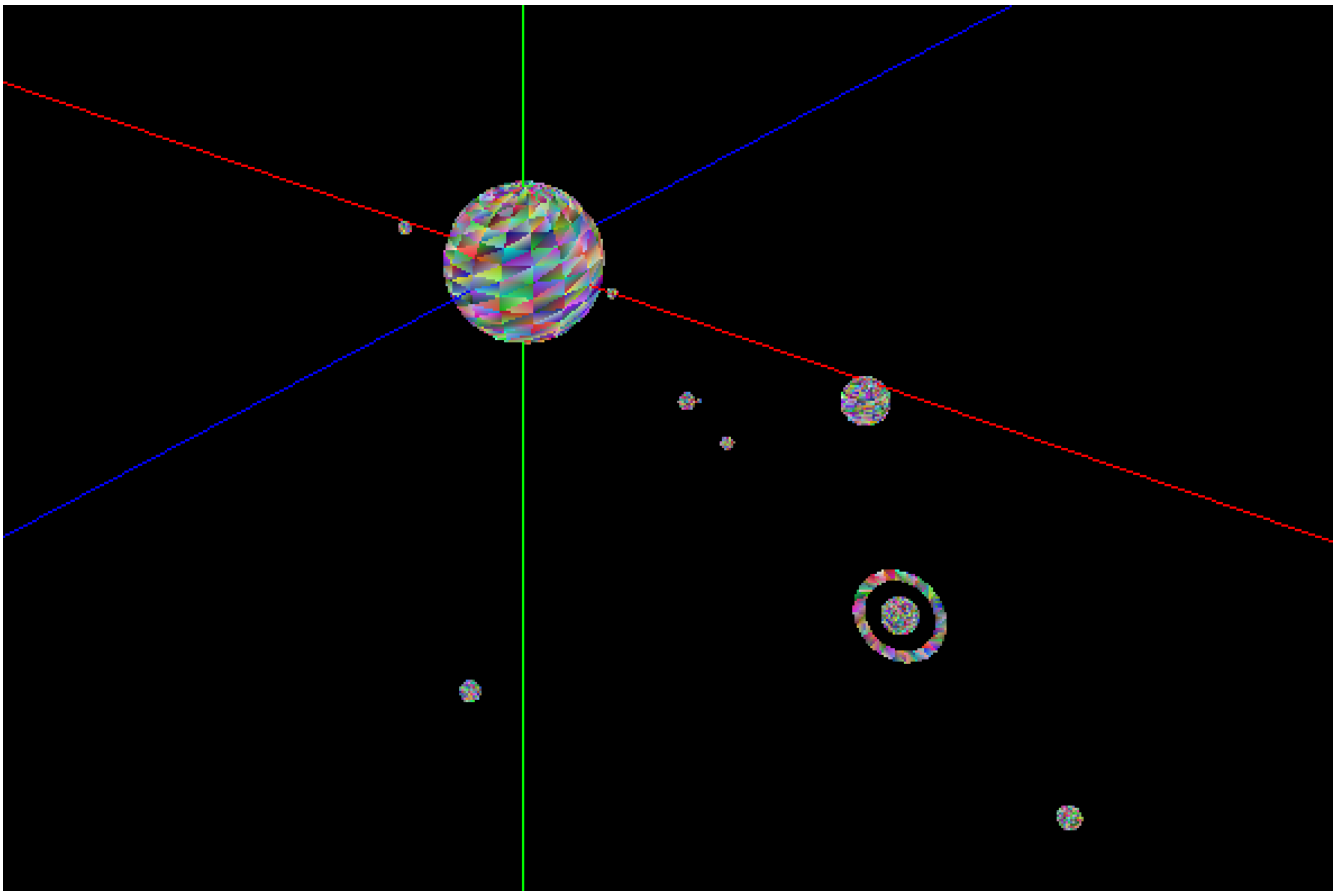

Figura 18: Resultado final do xml do Sistema solar.

6 Conclusão

Olhando para trás, podemos dizer que conseguimos fazer um bom trabalho nesta fase, pois conseguimos cumprir com todas as tarefas porpostas.

Terminar esta fase foi importante para continuarmos o projeto, dado que agora possuímos mais conhecimento relativamente ao tema em si, nomeadamente sobre lidar com transformações geométricas.

Surgiram algumas dificuldades no uso de funções recursivas e na representação do sistema solar. Conseguirmos uma solução limpa e fácil de visualização requeriu diversas tentativas especialmente para encontrar escalas corretas e as movimentações adequadas para cada planeta. No entanto, conseguimos um resultado positivo e superar as dificuldades encontradas.

Para concluir, consideramos que houve um balanço positivo do trabalho realizado e esperamos continuar neste rumo para o resto do projeto.