Universidade do Minho

Mestrado Integrado em Engenharia Informática

Computação Gráfica

Phase 2 – Geometric Transforms

Autores:

João Cabo A75064

José Sousa A74678

8 de Abril de 2018

Conteúdo

1 Introdução

A segunda fase do projeto, tem como objetivo permitir que o nosso engine, criado na fase anterior, leia e processe informação sobre transformações geométricas e hierarquia contida no ficheiro XML que vai receber.

A estrutura destes ficheiro sofreu então uma grande mudança, em vez de conter apenas informação sobre as primitivas geométricas que pretende desenhar, contém também a formação de uma hierarquia entre esses modelos. Esta hierarquia tem associada a si diversas funcionalidades,por exemplo, a **Cor** e **transformações geométricas**, tais como: **Translate, Rotate e Scale**. Estas serão responsáveis pelo modo como cada um dos modelos são desenhados.

Será assim necessário alterar a forma como é feita a leitura e o processamento desta informação. Foram criadas novas classes de forma a satisfazer estes novos requisitos.

Todas estas mudanças foram feitas com o intuito de gerar primitivas que permitissem representar um modelo estático do Sistema Solar.

2 Arquitetura

2.1 Classes Novas ou Alteradas

2.1.1 Generator

Tal como explicado na fase anterior esta é a aplicação responsável por gerar os vértices de cada modelo e criar um ficheiro para cada um deles. Para esta fase foi acrescentada a primitiva Torus e como tal o gerador foi alterado de forma a permitir gerar esta primitiva.

2.1.2 Engine

Tal como na fase anterior a classe engine tem diversas funcionalidades. Permite apresentar as várias primitivas e interagir com estas, através de alguns comandos. Devido à alteração da estrutura do ficheiro XML foi necessário alterar o modo como se realizava o parsing. A maneira como as formas são renderizadas também foi alterada de forma a permitir o uso de uma nova estrutura que serve para armazenar vários tipos de informação.

2.1.3 Torus

Classe que, com o devido algoritmo, nos permite obter os vértices necessários para a criação de um torus.

2.1.4 Action

Classe abstrata contém toda a informação relativa às ações (classes) que vão ser aplicadas às primitivas. Estas ações são:

- Translation
- Rotation
- Scale
- Color

É também a classe que quando lhe é passado um XMLElement, dependendo do tipo deste elemento são retirados todos os atributos relativos à ação. Também possui um método para aplicar a transformação (também é dependente do tipo do elemento)

2.1.5 Parser

A Classe Parser contém todos os métodos que são necessários para realizar a correta leitura e interpretação do ficheiro XML. Esta classe trata de armazenar, na estrutura indicada para isso, toda a informação relevante do ficheiro XML que é passado como input. A ferramenta utilizada é a mesma da primeira fase, tinyxml2.

2.1.6 Group

Classe que tem como objetivo armazenar toda a informação relativa a um grupo. Um grupo contém a seguinte informação:

- Modelos do grupo(shapes)
- Transformações geométricas e Cor(actions)
- Filhos (para se formar uma hierarquia um grupo pode ter outro grupo dentro de si, estes grupos são denominados childs)

2.2 Classes Inalteradas

2.2.1 Vertex

Esta classe manteve-se inalterada. Representa um vértice de um triângulo, composto pelas coordenadas (x,y,z).

2.2.2 Shape

Tal como a classe Vertex também se mantém inalterada. Representa um modelo 3D, composta por um conjunto dde vértices, sendo esses os necessários para a criação do mesmo.

2.2.3 Plane

Classe criada na primeira fase, que se manteve inalterada. A Classe Plane, com o devido algoritmo,permite-nos obter os vértices necessários para a criação de um plano.

2.2.4 Box

Classe criada na primeira fase, que se manteve inalterada. A Classe Box com o devido algoritmo, permite-nos obter os vértices necessários para a criação de uma caixa.

2.2.5 Sphere

Esta classe é exatamente igual à que foi criada na primeira fase. Permite-nos obter os vértices necessários para a criação de uma esfera.

2.2.6 Cone

Classe criada na primeira fase, que se manteve inalterada. A Classe Cone, com o devido algoritmo, permite-nos obter os vértices necessários para a criação de um cone.

2.3 TinyXML2

Ferramenta utilizada para fazer o parsing dos ficheiros XML.

3 Algoritmos

3.1 Torus

Um torus é um sólido geométrico que apresenta a forma de um donut. Pode ser definido pela rotação de uma superfície circular plana de raio interior, em torno de uma circunferência de raio exterior. Os parâmetros para desenhar um Torus são os seguintes:

- **r** - raio interior
- **R** - raio exterior
- **sides** - número de lados por cada secção radial
- **rings** - número de divisões radiais

3.1.1 Algoritmo

Para conseguirmos desenhar o Torus é preciso ter em consideração a sua estrutura, o raio interior e exterior. Inicialmente temos de decidir quais serão os exos responsáveis por definir as circunferências que vamos usar para percorrer e desenhar o Torus. Os eixos Y e X definem a circunferência com raio exterior **R** e os eixos X,Y e Z definem a circunferência com raio interior **r**. Na figura que apresentamos a seguir é possível observar o que cada um representa.

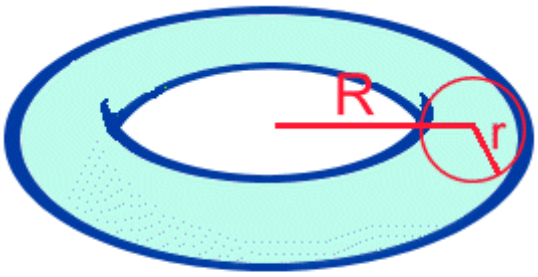

Figura 1: Representação do Torus

Para ser possível iterar através das circunferências definidas, é necessário recorrer aos parâmetros sides e rings (dividem as circunferências em várias partes, semelhantes às stacks e slices das outras primitivas).

A figura que apresentamos a seguir ilustra o método de construção do Torus:

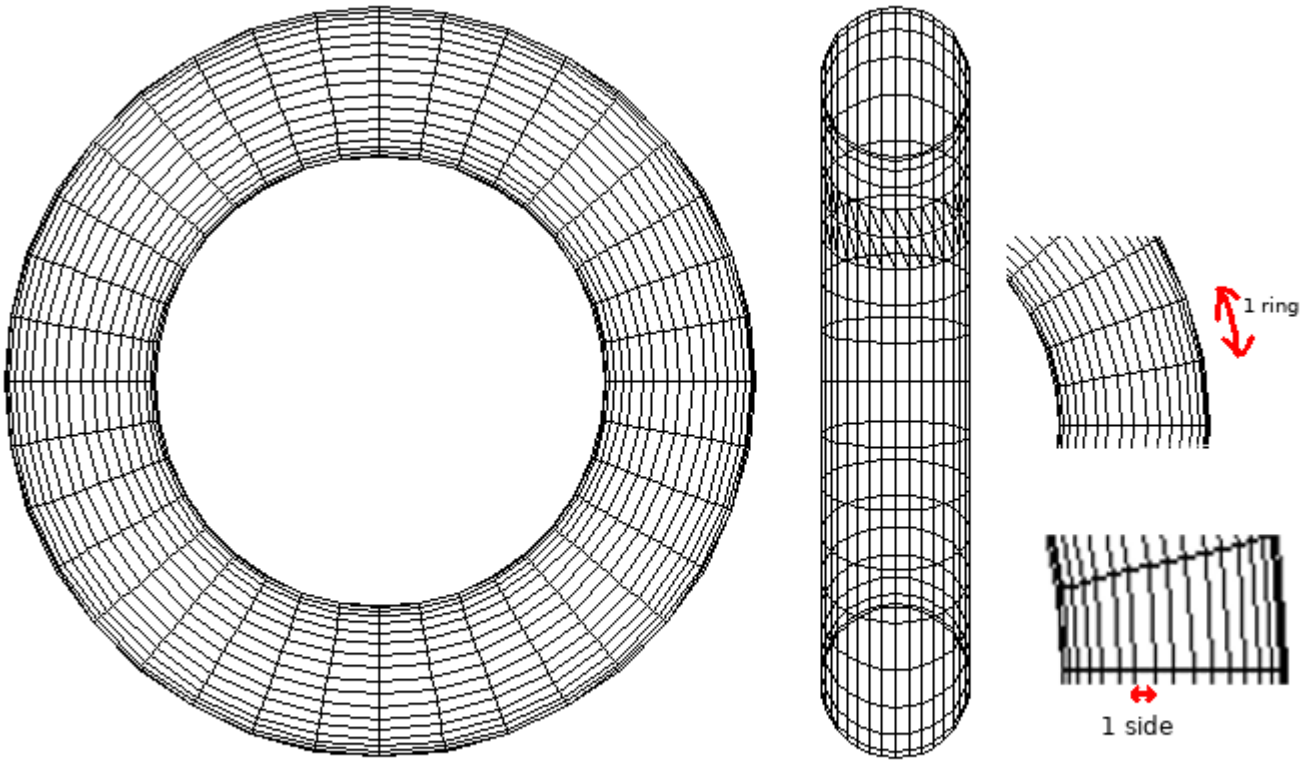

Figura 2: Representação da construção do Torus

Cada parte é definida por uma amplitude dada por:

- sideSize = (2*M_PI) / sides
- ringSize = (2*M_PI) / rings

Assim, é possível obter facilmente os pontos que formam as circunferências usando as funções trignométricas cos e sin. Inicialmente (i=0) são definidos os seguintes pontos:

- alpha = i* ringSize;
- nextalpha = alpha + ringSize;
- x0 = cos(alpha);
- y0 = sin(alpha);

- x1 = cos(nextalpha);
- y1 = sin(nextalpha);

Os pontos x0 e y0 representam os pontos atuais de amplitude, x1 e y1 representam os próximos pontos (próximo anel após ser incrementada a amplitude), relativamente à circunferência de raio externo. Passamos assim a desenhar entre dois "limitadores"de um anel. Do mesmo modo que percorremos a circunferência externa, percorremos a circunferência interna, apenas sendo necessário adicionar sideSize em cada iteração, de forma a dar a volta à circunferência interna. Para definir os pontos basta obter os seguintes valores:

- r = radiusSmall * cos(j*sideSize + radiusBig;
- z = radiusSmall * sin(j*sideSize);

O valor de r é o factor que afasta os pontos em relação ao centro, adicionado para esse efeito o raio externo, de forma a que todos os pontos se encontrem para lá desse valor e também multiplicando o raio interno por cos(j*sideSize), para definir o ponto da circunferência externa.

Para o valor de z, apenas é necessário multiplicar o valor do raio interno por sin(j*sideSize), isto porque o valor do desvio do raio externo apenas se aplica aos eixos X e Y.

Por fim basta multiplicarmos o valor de r pelos pontos x0, y0, x1 e y1 para obtermos as coordenadas dos pontos do Torus e assim, formar os triângulos.

4 Parser

O processo de leitura efetuado pelo engine encontra-se na classe Parser. Este processo inicia-se quando é fornecido um ficheiro XML como input à aplicação do engine. Começamos assim a percorrer o ficheiro recursivamente através da função findElement(XMLElement*,Group*) que a cada chamada recebe o elemento do XML que está a percorrer e o grupo onde a informação está a ser armazenada.

Na primeira chamada desta função é passado como parâmetro o primeiro filho do elemento **scene** do ficheiro XML, que será o primeiro grupo. A seguir, é testado se o elemento é uma transformação com o intuito de armazenar a informação na estrutura do grupo que estamos a percorrer:

- **Translation**-foundTranslation(element,group)
- **Rotation**-foundRotation(element,group)
- **Scale**-foundScale(element,group)

- **Color**-foundColor(element,group)

Caso o elemento não seja uma transformação só poderá ser ou uma lista de modelos ou um filho.

Caso seja um modelo é chamada a função findModels que irá processar todos os ficheiros dos modelos criando para cada modelo (através da função readFile) uma forma Shape onde o conteúdo será o conjunto de pontos que formam o ficheiro do modelo lido. Esta Shape será adicionada à lista de formas do grupo, de forma a manter a informação do grupo atualizada.

Caso seja um filho é necessário criar um novo objeto Group e adicioná-lo à lista de filhos do grupo atual, com o objetivo de criar a hierarquia pretendida, chamando recursivamente a função findElement e fornecendo como input o novo filho.

Se não corresponda a nenhum destes casos, passamos para o elemento irmão.

5 Estrutura

Usando o novo parser e analisando a estrutura do ficheiro XML é possível concluir qual será a melhor estrutura para armazenar toda a informação recolhida durante o processamento.

Cada grupo será identificada através de um **ID**, para isto, é necessário a aplicação engine possuir uma variável global na aplicação de forma a ser possível contar o total dos grupos que foram criados até ao momento.

Foi também necessário criar um vector que contém os modelos do grupo (**shapes**), sendo que cada um dos elementos deste vector irá conter os pontos para construir os modelos em questão.

Devido à necessidade da hierarquia ser preservada, foi necessário criar um vector de filhos, dai existir o vector **groups**.

Por fim, foi necessário criar um terceiro vector capaz de armazenar todas as transformações que foram aplicadas ao grupo em questão de forma ordenada (**actions**).

Na figura seguinte apresentamos a estrutura que explicamos acima:

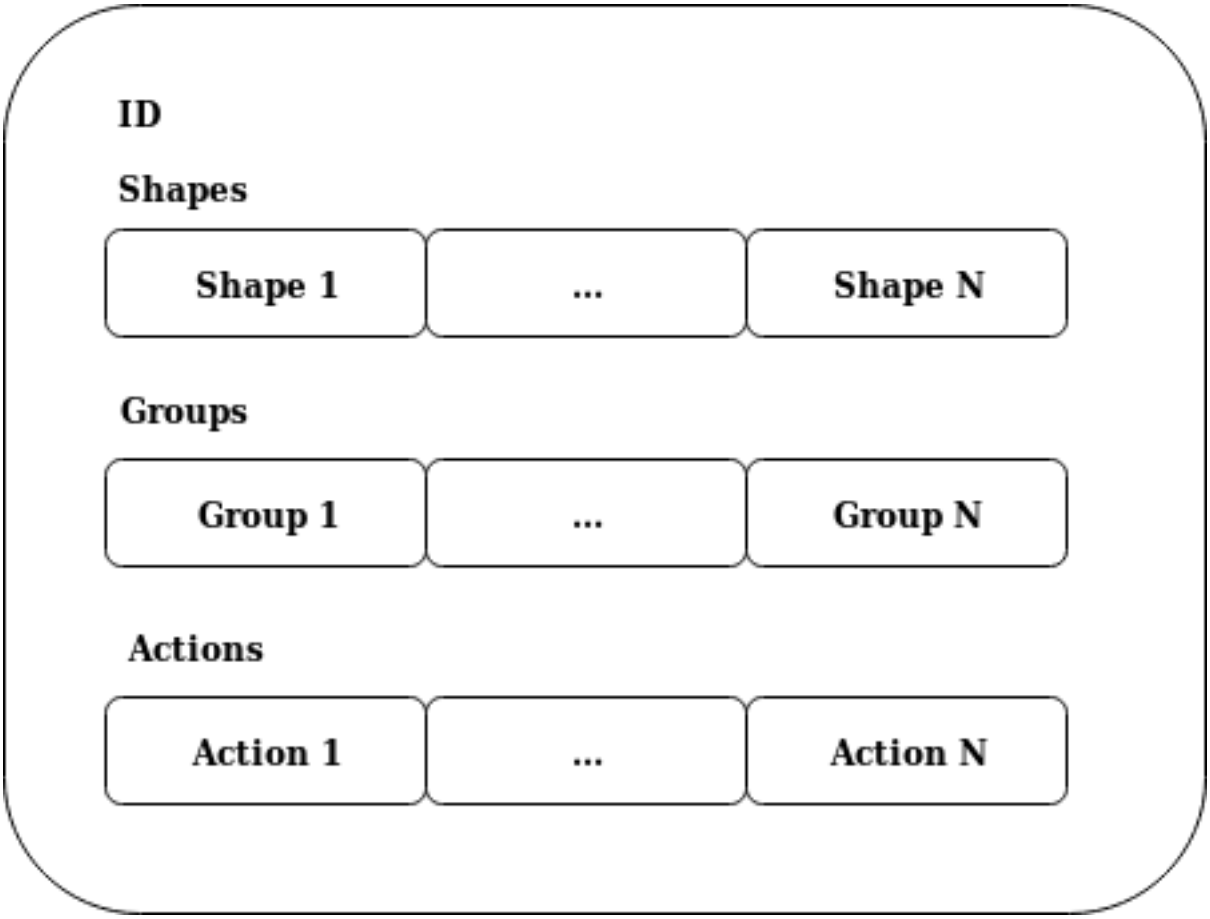

Figura 3: Representação da estrutura de cada grupo

6 Rendering

Na aplicação engine é criada uma variável global, chamada de **scene**, que guarda toda a informação dos restantes grupos. Esta informação é renderizada na função **renderScene** que é semelhante à apresentada na primeira fase mas contém uma alteração. Dentro desta função é chamada recursivamente a função **render** que recebe como argumento, na sua primeira chamada, a variável global **scene**. Como consequência do uso de transformações geométricas, ou seja, como a matriz de transformação será alterada é necessário guardar o estado inicial dela e, após feitas as alterações pretendidas, este estado deverá ser reposto. Para isto são usados os métodos **glPushMatrix()** e **glPopMatrix()**.

Antes de desenhar é necessário aplicar todas as informações anteriormente armazenadas para o grupo atual, por isso o vetor de actions é percorrido e

utilizando o método **apply** são realizadas as transformações.

Após serem efetuadas todas as transformações é necessário percorrer os modelos que se encontram armazenados no grupo atual e, para cada um, desenhar os pontos de modo a ser possível criar os triângulos que constituem a figura pretendida.

Por fim a função é chamada recursivamente para cada um dos filhos do grupo atual sendo assim possível renderizar toda a informação contida no ficheiro XML.

7 Resultado Final- Sistema Solar

O resultado final é apresentado nas figuras que se seguem:

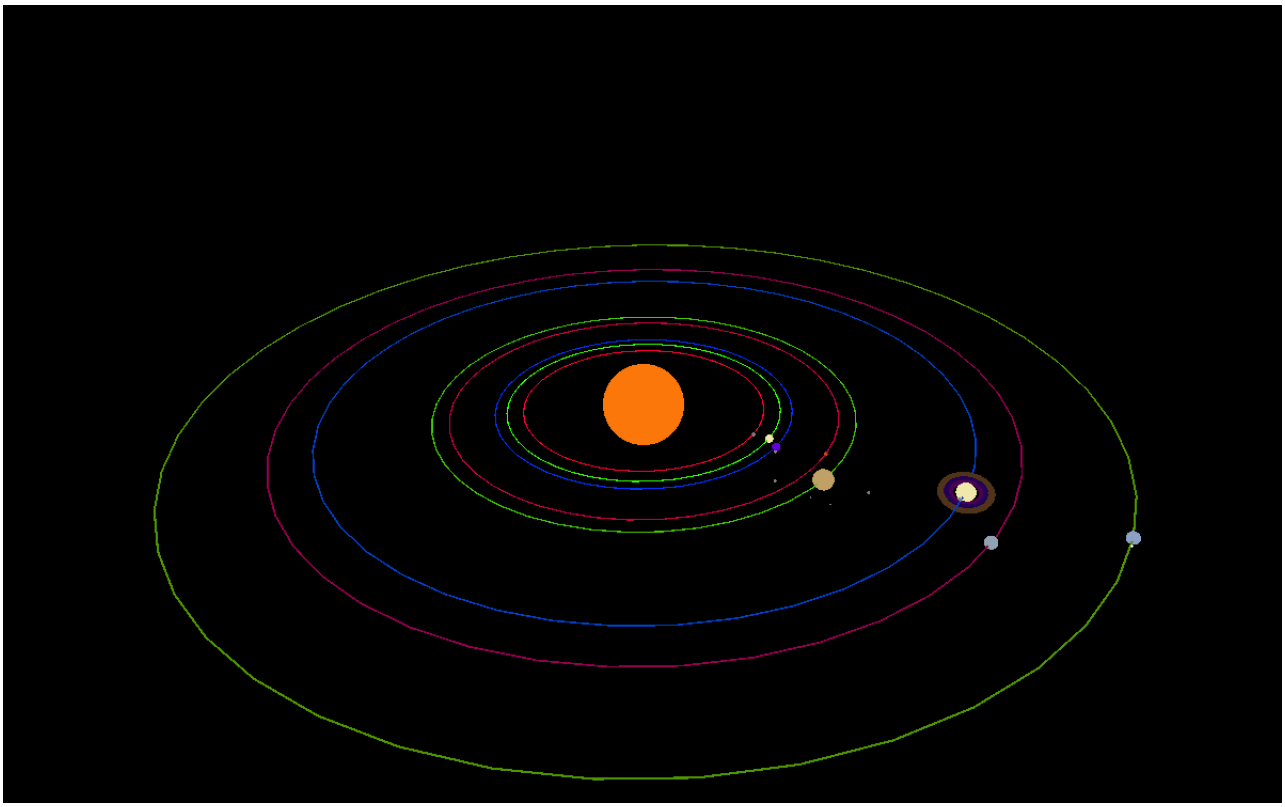

Figura 4: Sistema Solar completo

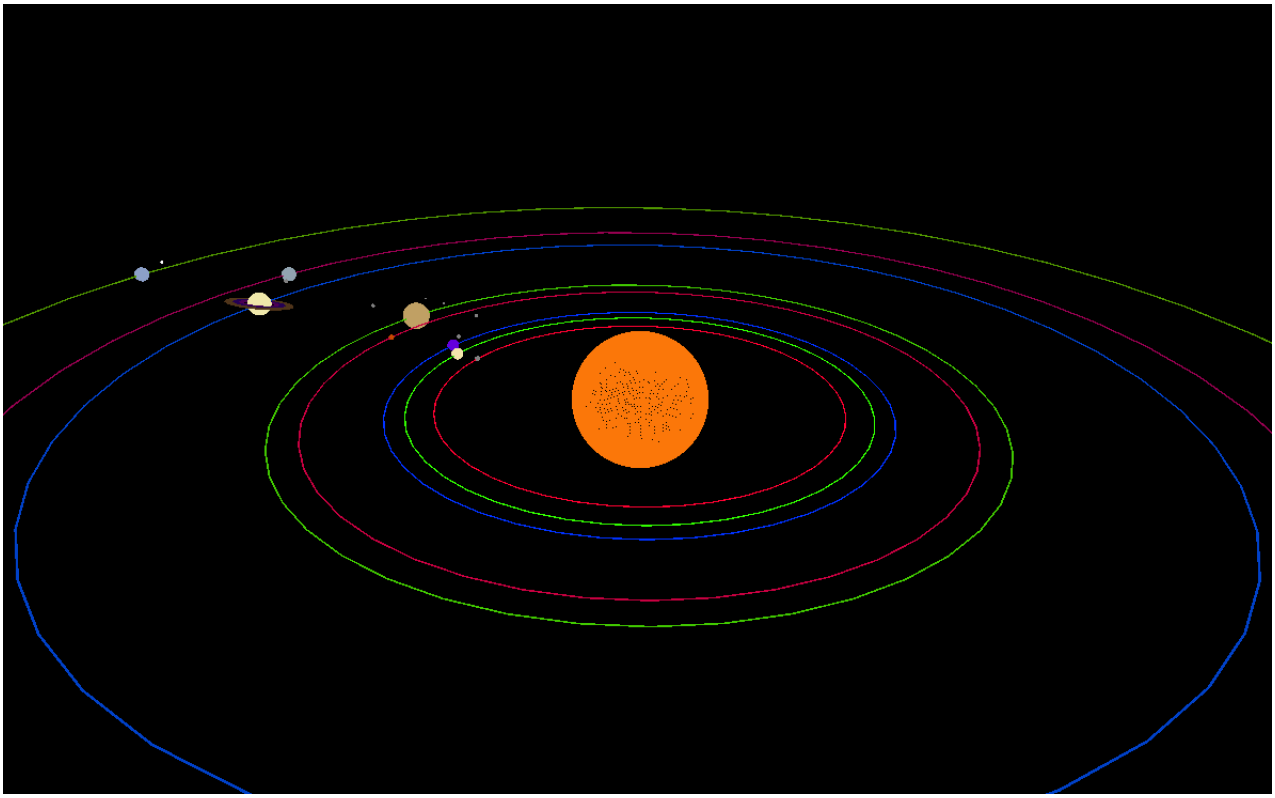

Figura 5: Sistema solar completo visto de outra prespetiva

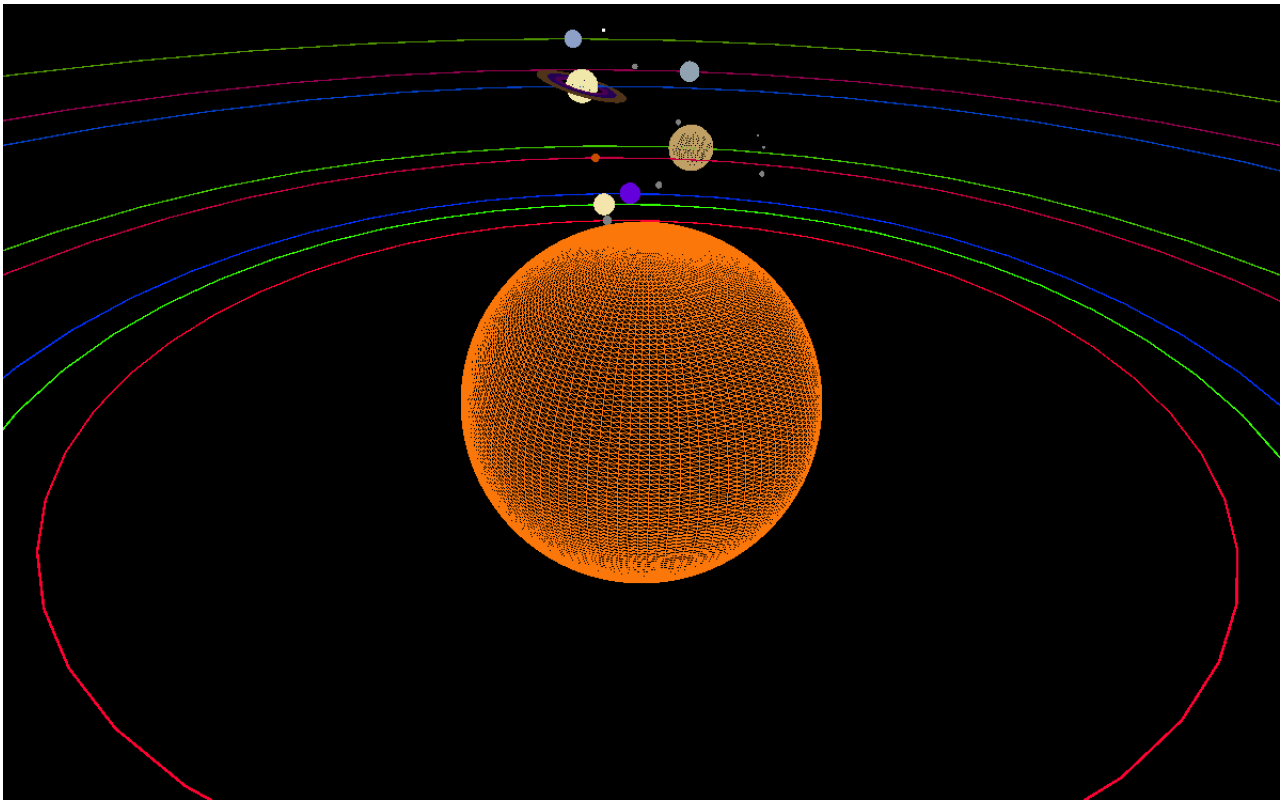

Figura 6: Sistema solar visto a partir do Sol

8 Conclusão

Concluindo assim a segunda fase deste projeto, é possível afirmar que esta fase nos ajudou a compreender melhor como se realiza a construção de uma scene mais completa, tendo agora sido possível compreender como funciona a hierarquia e como aplicar transformações geométricas à scene. Após esta segunda fase, encontramos-nos ainda mais motivados para a realização deste projeto e esperamos conseguir melhorar cada vez mais.