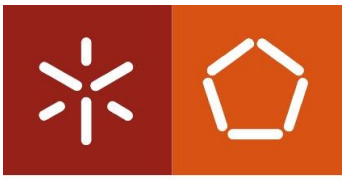

Universidade do Minho

Departamento de Informática

Trabalho Prático
Computação Gráfica

Catarina Machado (a81047) Gonçalo Faria (a86264)
João Vilaça (a82339)

25 de Março de 2019

Conteúdo

1 Fase 2 - Transformações Geométricas

O objetivo da segunda fase do trabalho prático foi o de melhorar o motor gráfico iniciado na primeira fase. Esta melhoria foi feita através da especificação e consequente interpretação de cenas hierárquicas em xml.

1.1 Coordinate Frame

O módulo *Coordinate Frame*, criado na primara fase deste projecto para simplificar e optimizar o processo de geração dos modelos correspondentes às primitivas gráficas, foi aproveitado nesta fase do projecto.

Este módulo representa um sistema de coordenadas e pode ser interpretado como um nó de uma árvore. Esta árvore é navegável apenas no sentido dos nós filhos para nós pais e contém na sua raiz uma lista onde se encontram guardados todos os pontos que foram desenhados.

```
typedef struct frame {
        double t[4][4];
        vector<Point> points;
        struct frame * ref;
        GLuint buffer;
} *CoordinateFrame;
```

Cada um dos nós mantém como estado, numa matriz, a informação referente à base e à origem do sistema de coordenadas local que representa assim como o número do buffer da placa gráfica em uso. O número do buffer será usado para identificar o buffer em que serão carregados os pontos na placa gráfica, para que estes sejam apresentados no ecrã.

Para que seja crida uma nova raiz é chamado o construtor sem argumentos e para originar uma bifurcação é chamado o mesmo construtor mas agora especificando um sistemas de coordenadas inicial.

```
CoordinateFrame mkCoordinateFrame();
CoordinateFrame mkCoordinateFrame(CoordinateFrame referencia);
```

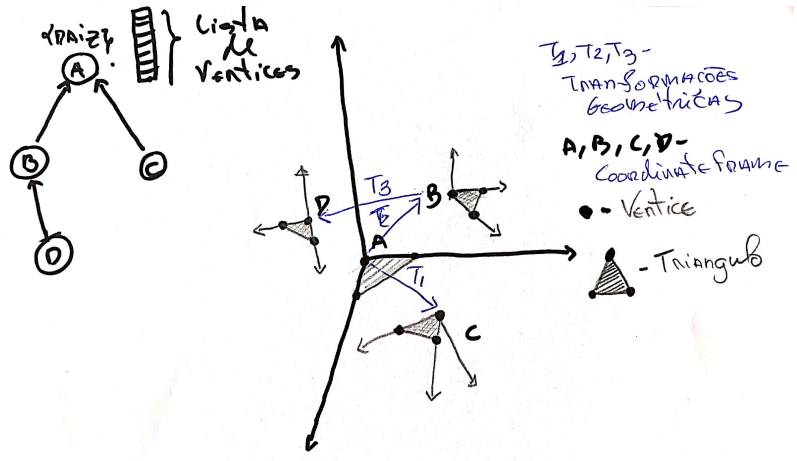

Figura 1: Representação da árvore de Sistema de Coordenadas.

1.1.1 Rotação

A rotação do sistema de coordenadas é feita através das matrizes de rotação referentes a cada um dos eixos. Essas matrizes encontram-se apresentadas em baixo.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

Tendo o eixo de rotação e o ângulo, são obtidos os ângulos de rotação segundo cada um dos eixos e, através do produto das matrizes apresentadas em cima, é calculada a rotação correspondente aos parâmetros especificados.

$$R = R_z(\alpha)\, R_y(\beta)\, R_x(\gamma) \tag{4}$$

Com a matriz de rotação e após o produto com a matriz com a base e origem do sistema de coordenadas, constante na instância de *Coordinate Frame*, esta torna-se a nova matriz representante do sistema de coordenadas local.

A função que realiza esta funcionalidade encontra-se em baixo.

```
void frameRotate(CoordinateFrame m,
                 double angle,
                 double vx,
                 double vy,
                 double vz);
```

1.1.2 Translação

A translação do sistema de coordenadas é feita através da matriz de translação definida em baixo.

$$T_{\mathbf{v}} = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Com a matriz de translação e após o produto com a matriz com a base e origem do sistema de coordenadas, constante na instância de *Coordinate Frame*, esta torna-se a nova matriz representante do sistema de coordenadas local.

A função que realiza esta funcionalidade encontra-se em baixo.

```
void frameTranslate(CoordinateFrame m,
                    double vx,
                    double vy,
                    double vz);
```

1.1.3 Escala

A escala do sistema de coordenadas é feita através da matriz de escala definida em baixo.

$$S_v = \begin{bmatrix} v_x & 0 & 0 & 0 \\ 0 & v_y & 0 & 0 \\ 0 & 0 & v_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

Com a matriz de escala e após o produto com a matriz com a base e origem do sistema de coordenadas, constante na instância de *Coordinate Frame*, esta torna-se a nova matriz representante do sistema de coordenadas local.

A função que realiza esta funcionalidade encontra-se em baixo.

```
void frameScale(CoordinateFrame m,
                double vx,
                double vy,
                double vz);
```

1.1.4 Desenhar Ponto

O desenho de um ponto num sistema de coordenadas é feito, essencialmente, partindo de um sistema de coordenadas e três coordenadas relativas a esse.

O ponto indicado é multiplicado pela matriz do sistema de coordenadas em questão e é propagada na árvore de sistemas de coordenadas até à raiz para que seja registado na lista contida nesta.

A função que realiza esta funcionalidade encontra-se em baixo.

```
void framePoint(CoordinateFrame m,
                double vx,
                double vy,
                double vz);
```

1.2 Interpretação da especificação de cena

A análise e interpretação da especificação da cena foi feita recursivamente devido à forma como a especificação em xml foi definida pelos docentes da unidade curricular. A árvore dos sistemas de coordenadas é criada e tem como raiz a *CoordinateFrame* indicada na primeira invocação do **Algoritmo 1**.

A função *parseModels*, invocada quando um elemento do tipo "models" é encontrado, é essencialmente a função de análise de xml da primeira fase do projeto. Esta função percorre os vários elementos com nome "model" e desenha todos os pontos constantes no ficheiro especificado no campo atributo no sistema de coordenadas indicado como argumento.

É de salientar que na chamada recursiva de *parseGroups* é criada uma nova instância de *CoordinateFrame*. Esta instância é dada como parâmetro da invocação recursiva mas, no entanto, após o termino desse procedimento é apagada. Isto acontece pois não se pretende que as possíveis transformações geométricas constantes no sub grupo a analisar afetem os elementos fora deste. Ou seja, o objetivo desta instanciação do novo sistemas de coordenadas local, é o de registar os pontos desenhados na raiz da árvore de sistemas de coordenadas.

Algorithm 1 Parse

```
1: procedure parseGroups(XMLNode group, CoordinateFrame state)
2:     e ← group.FirstChild()
3:     while e ≠ NULL do
4:         if e.Value() is "models" then
5:             parseModels(e, state)
6:         if e.Value() is "translate" then
7:             x ← e.DoubleAttribute("X")
8:             y ← e.DoubleAttribute("Y")
9:             z ← e.DoubleAttribute("Z")
10:            frameTranslate(state,x,y,z)
11:        if e.Value() is "rotate" then
12:            angle ← e.DoubleAttribute("angle")
13:            x ← e.DoubleAttribute("axisX")
14:            y ← e.DoubleAttribute("axisY")
15:            z ← e.DoubleAttribute("axisZ")
16:            frameRotate(state,angle,x,y,z)
17:        if e.Value() is "scale" then
18:            x ← e.DoubleAttribute("stretchX", 1.0)
19:            y ← e.DoubleAttribute("stretchY", 1.0)
20:            z ← e.DoubleAttribute("stretchZ", 1.0)
21:            frameScale(state,x,y,z)
22:        if e.Value() is "group" then
23:            newState ← mkCoordinateFrame(state)
24:            parseGroups(e, newState)
25:            unmkCoordinateFrame(newState)
           e ← e.NextSibling()
```

1.3 Motor Gráfico

Para que os pontos contidos na árvore de sistemas de coordenadas sejam apresentados no ecrã é usada a biblioteca *OpenGL*. Inicialmente com *framBufferData* é criado e inicializado um objeto buffer de *OpenGL* capaz de guardar a lista de pontos e, posteriomente, no momento de desenho de cena, é usada a função *frameDraw* para apresentar os pontos no ecrã.

```
void frameBufferData(CoordinateFrame reference);
void frameDraw(CoordinateFrame reference);
```

1.4 Cena de demonstração

Para a segunda fase do trabalho prático foi nos proposta a elaboração de uma *demo scene* estática do modelo do sistema solar, que contém o sol, os planetas e as respetivas luas.

De modo a reaproveitarmos o trabalho realizado na primeira fase, o ficheiro com o modelo da esfera foi utilizado para a representação de todos os elementos do sistema solar. Assim, a esfera utilizada tem raio 1, 40 slices e 40 stacks, ficando com o resultado final apresentado na figura seguinte.

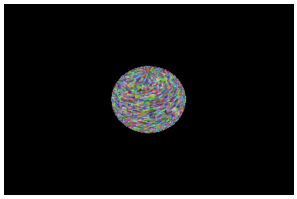

Figura 2: Esfera com raio 1, 40 slices e 40 stacks.

O nosso próximo objetivo prendeu-se na decisão das escalas a adotar, de modo a que o sistema solar tenha um resultado globalmente próximo da realidade. Para isso, tomamos como exemplo as duas seguintes figuras, que representam o Sistema Solar em escala e o Sistema Solar com todos os planetas e luas, Figura 3 e Figura 4, respetivamente.

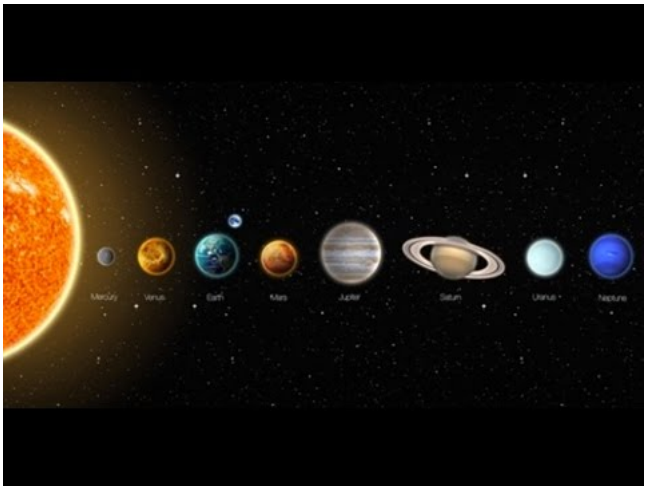

Figura 3: Sistema Solar em escala.

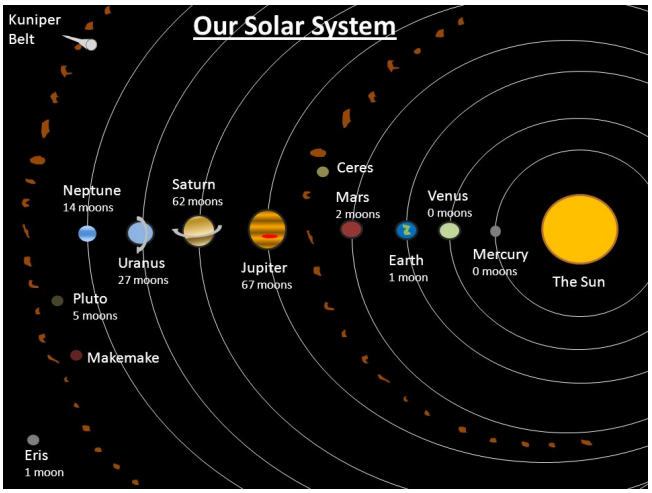

Figura 4: Sistema solar com todos os planetas e luas.

Em seguida, tendo em consideração que já possuíamos uma ideia mais concreta do sistema solar a representar, começamos por desenhar o sol, no centro do nosso referencial, dentro de um *group*, que, tal como já foi referido, contém o ficheiro *sphere.xml* como *model*.

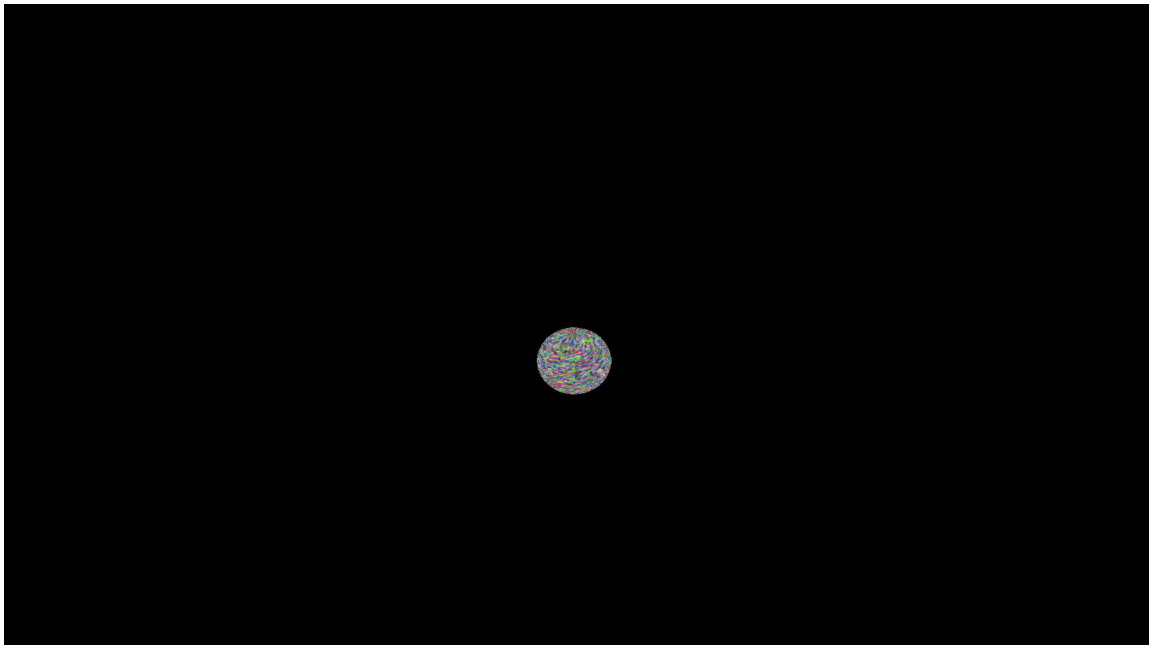

Figura 5: Modelo sistema solar com o sol e planetas.

Consequentemente, nesta fase decidimos começar por desenhar os planetas em linha uma vez que os planetas são ainda todos da mesma cor e fica mais claro perceber o esquema geral que pretendemos aperfeiçoar nas próximas fases do projeto. No final desta fase, depois de já termos as posições relativas definidas, deixamos de representar os planetas em linha.

Posteriormente, com o objetivo de obter uma proporção geral do tamanho dos vários planetas o mais representativa possível (tendo sempre em consideração que uma abordagem 100% realista não seria benéfica devido à enorme disparidade de tamanhos dos planetas), fomos testando a representação dos planetas variando os valores da *scale* de cada um deles. Assim, após termos

chegado às proporções que consideramos serem as mais apropriadas, aplicamos *translates* a cada um dos planetas de modo a ficarem praticamente igualmente espaços entre si. Todos os 8 planetas são *subgroups* do sol.

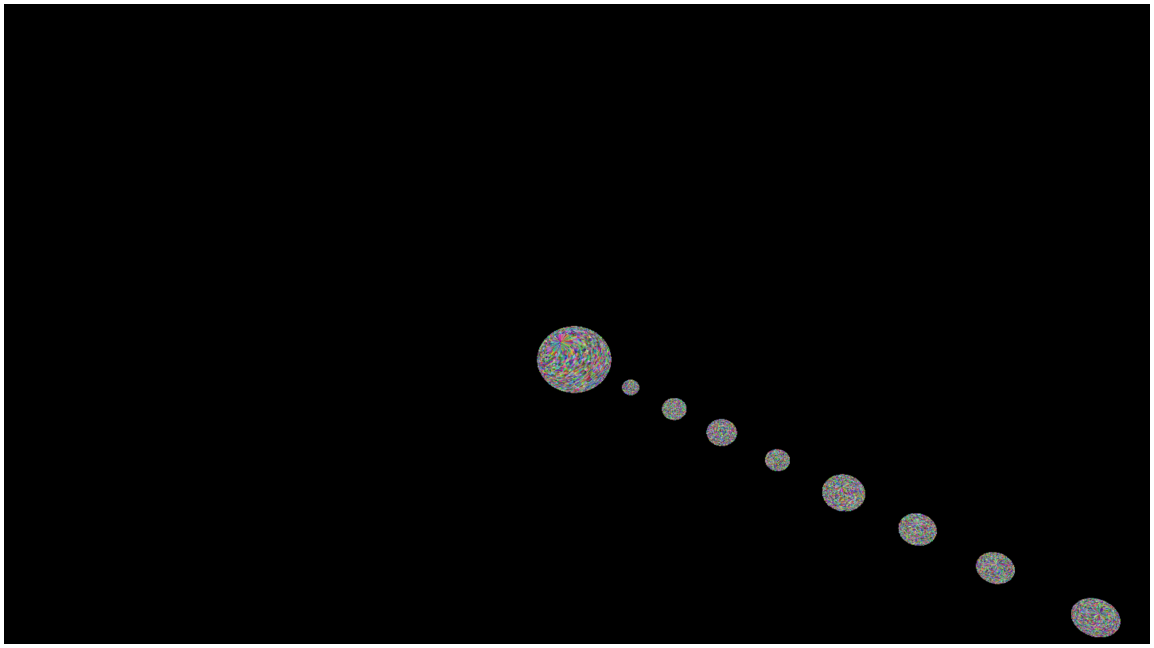

Figura 6: Modelo sistema solar com o sol e planetas.

Em sequida, procedemos à representação das luas de cada um dos planetas, caso se aplique. As luas de cada planeta foram representadas como um *subgroup* do mesmo.

Uma vez que nem Mercúrio nem Vénus possuem luas, começamos por desenhar a lua do planeta Terra e as 3 luas de Marte, através de *translates* e *scales*.

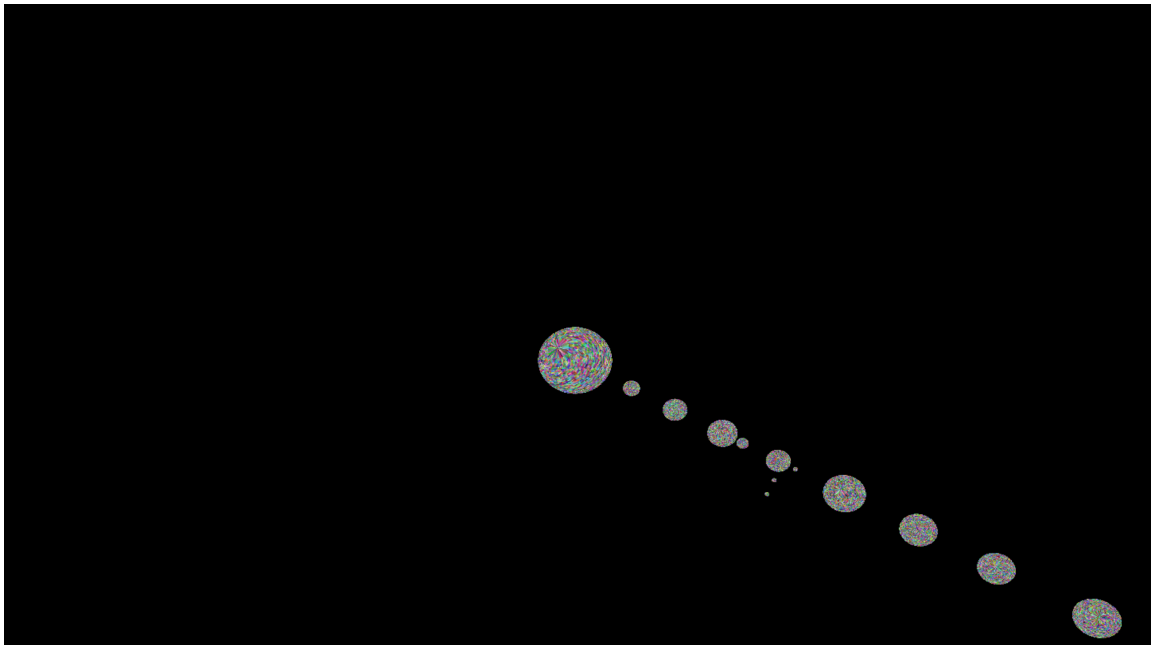

Figura 7: Modelo sistema solar com sol, planetas, lua da Terra e luas de Marte

Tendo em consideração que alguns dos planetas possuem bastantes luas, decidimos, pelo menos numa fase inicial, não as representar a todas. Deste modo, Júpiter, ao invés das 67 luas tem somente 13, Saturno tem 10 ao invés de 62, Urano tem 5 ao invés de 27 e Nepturno tem somente 4 ao invés de 14.

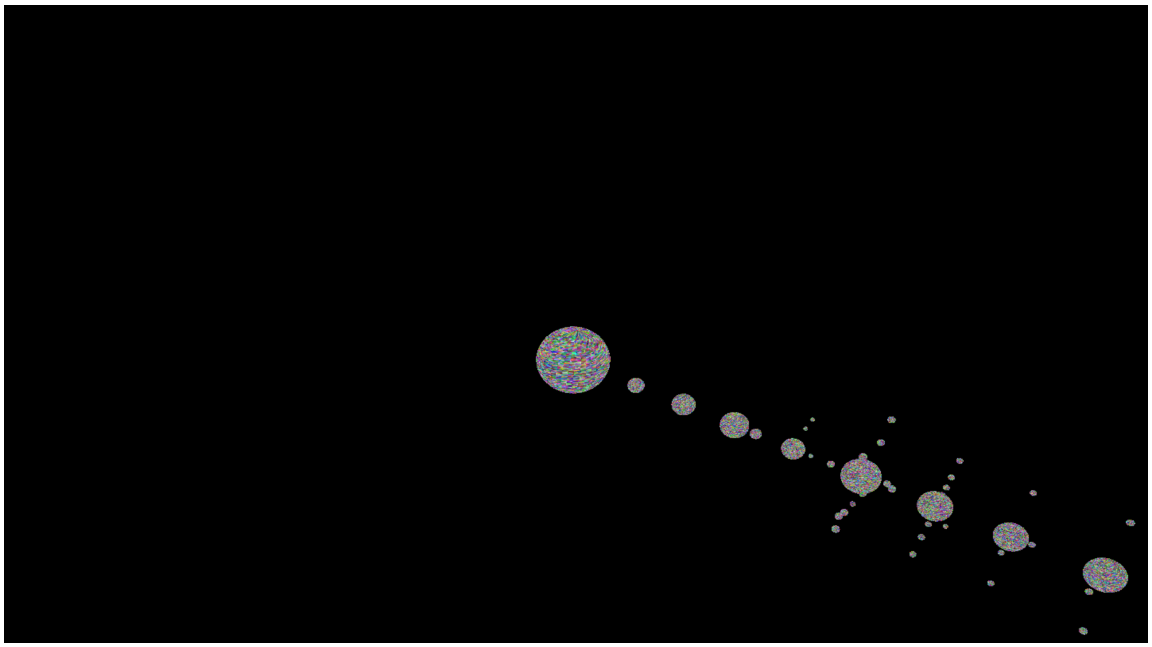

Figura 8: Modelo sistema solar em linha final.

Por fim, alteramos os valores nos *translates* do Y e do Z de modo a que os planetas fiquem espalhados pelo sistema solar, mas mantendo-se na sua órbita.

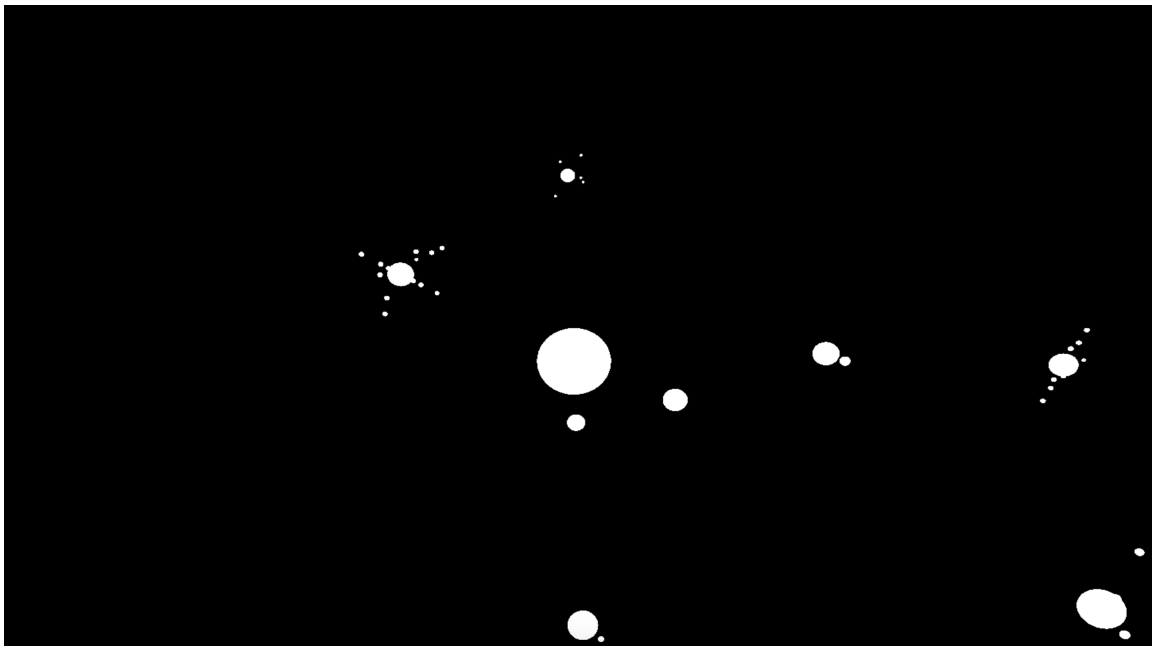

Figura 9: Modelo sistema solar final.

Nas próximas fases temos como objetivos principais desenhar as órbitas dos planetas (elipses) em torno do sol e adicionar cor e textura a cada um dos planetas, luas e ao sol.