Universidade do Minho

Mestrado Integrado de Engenharia Informática

Normais e Coordenadas de Texturas

Fase IV

Autores:

Sara Pereira	A73700
Filipe Fortunato	A75008
Frederico Pinto	A73639

21 de Maio de 2018

Conteúdo

1 Introdução

Neste trabalho, o quarto da Unidade Curricular de Computação Gráfica, foi nos pedido que fizéssemos um mini mecanismo 3D com ambiente gráfico sendo que para isso utilizamos diversas ferramentas apresentadas nas aulas práticas, entre as quais o **OpenGl** e **C++** Nesta última fase do trabalho foi incluído texturas e iluminação ao trabalho anteriormente desenvolvido, tendo como objetivo final a criação de um Sistema solar animado.

2 Resumo

Nesta última fase do trabalho, é normal que a maioria das funcionalidades do trabalho se mantenham inalteradas mas também são melhoradas e adicionadas mais algumas.

No gerador, são feitas algumas alterações de maneira a que este seja capaz, de através de uma imagem, ter a capacidade de calcular as normais e a textura para os vários vértices das diferentes primitivas geográficas.

No motor, são incluídas novas funcionalidades e algumas modificações. Como o ficheiro XML tem também agora informações em relação à iluminação do cenário, o parser responsável por ler esses ficheiros é alterado e também é alterada a forma de processamento da informação para construir o cenário. Para além disso, sofre mais algumas alterações nos **VBO's** e com a preparação das Texturas durante o processamento do ficheiro de configuração.

3 Arquitetura do código

3.1 Aplicações

Neste tópico serão apresentadas as alterações feitas nas duas aplicações principais do nosso projeto.

3.1.1 Gerador

Para esta fase do trabalho era pretendido que esta aplicação fosse capaz de gerar, para além dos pontos referentes às formas geométricas requisitadas, as respetivas normais e coordenadas de texturas dos mesmos pontos.

3.1.2 Motor

O motor, nesta fase, foi alterado de maneira a ser possível o desenho e representação de funcionalidades de textura e normais de aplicação de luminosidade. Ao mesmo tempo, também tiveram que ser efetuadas alterações na escrita dos pontos nos respetivos ficheiros **.3d** assim como no ficheiro XML. Estas mudanças serão explicadas numa fase posterior deste relatório.

4 Classes

Neste tópico serão apresentadas as classes que receberam as alterações mais significativas.

4.1 Transforms

Sendo esta a classe que guarda toda a informação relativa ao conjunto de transformações aplicadas a um grupo (filhos incluídos) assim como todo o processo de criação de VBOs, apenas faria sentido serem aqui implementados todos os processos de associação de texturas aos modelos. Logo, foram criados dois VBOs extra para guardar informação quanto às normais dos pontos e quanto às coordenadas de textura. A informação sobre a textura, para além das coordenadas, inclui o seu ID e dimensões da imagem a ser texturizada.

```
1  #include <vector>
2  #include "Transformacao.h"
3  #include "Ponto.h"
4  #include <fstream>
5  #include <iostream>
6  #include <string>
7
8  #ifdef __APPLE__
9  #include <IL/il.h>
10 #include <GLUT/glut.h>
11 #else
12 #include <IL/il.h>
13 #include <GL/glew.h>
14 #include <GL/glut.h>
15 #endif
16
17
18
19 using namespace std;
20 #ifndef PROJECT_TRANSFORMS_H
21 #define PROJECT_TRANSFORMS_H
22
23
24 class Transforms{
25     string text; //nome da imagem que se vai buscar ao XML
26     string tipo;
27     Transformacao t;
28     vector<Ponto> pontos;
29
30     int pos;
31     vector<Transforms> subgrupo;
32     vector<Ponto> normal;
33     vector<Ponto> textura;
34
35     // VBO
36     GLuint buffer[3];
37     float *v, *n, *textu;
38     float p_tam, n_tam, tex_tam;
39
40     //Textura:
```

```
41      unsigned int tt, width, height;
42      unsigned int texID;
43      unsigned char *data;
44
45  public:
46      Transforms();
47      Transforms(string tipo, Transformacao t, vector<Transforms> sub
        , vector<Ponto> pontos);
48      string getText(){ return text; }
49      string getTipo(){ return tipo; }
50      Transformacao getTransformacao(){ return t; }
51      vector<Transforms> getSubgrupo(){ return subgrupo; }
52      vector<Ponto> getPontos(){ return pontos; }
53      unsigned int getTexID(){ return texID;}
54      void setText(string t){ text = t;}
55      void setTipo(string t){ tipo = t; }
56      void setTrans(Transformacao trans){ t = trans;}
57      void setSubgrupo(vector<Transforms> sub){ subgrupo = sub; }
58      void setPontos(vector<Ponto> p){ pontos = p;}
59      void setNormal(vector<Ponto> n){ normal = n;}
60      void setTextura(vector<Ponto> tex){ textura = tex;}
61      void toVertex();
62      void setVBO();
63      void draw();
64      void push_child(Transforms t){subgrupo.push_back(t);}
65      void newText();
66  };
67
68
69  #endif //PROJECT_TRANSFORMS_H
```

plano

5 Gerador

5.1 Aplicações das Normais e planos de Textura

Para a obtenção das normais de uma figura geométrica é necessário encontrar o vetor perpendicular a cada ponto que constitui a mesma. Quanto à textura é necessário fazer uma mapeamento de uma imagem 2D para uma figura 3D, revestindo a figura geométrica com o resultado deste mapeamento.

5.1.1 Plano

Para obter o conjunto dos vetores das normais, basta apenas verificar em que plano cartesiano é desenhado o mesmo. Como apenas são construídos planos no eixo **xOz**, qualquer vetor perpendicular a este plano pode ser normal ao mesmo. No entanto definiu-se como normal ao mesmo o vetor (0,1,0).

Neste caso, o processo de obtenção das coordenadas de textura é obtido fazendo a correspondência direta a cada vértice.

5.1.2 Cubo

Para obter o conjunto dos vetores normais, basta fazer o mesmo que fizemos no Plano, isto é, verificar em que plano cartesiano é desenhado o mesmo. Desta forma, de seguida apresentamos os diferentes vetores normais para cada face do Cubo:

- **Face Frontal** $\rightarrow (0, 0, 1)$
- **Face Esquerda** $\rightarrow (-1, 0, 0)$
- **Face Direita** $\rightarrow (1, 0, 0)$
- **Face Traseira** $\rightarrow (0, 0, -1)$
- **Topo** $\rightarrow (0, 0, 1)$
- **Face Frontal** $\rightarrow (0, 0, -1)$

Apresentamos de seguida uma imagem que serve como modelo de como são obtidos os pontos de textura.

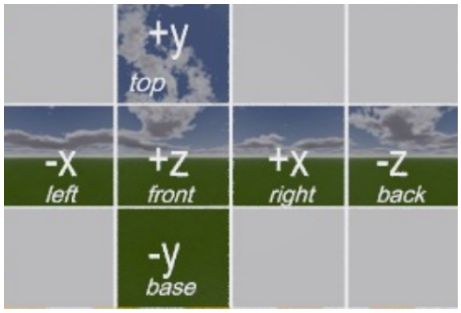

Figura 1: Exemplo de imagem 2D para um Cubo

Esses pontos são obtidos descobrindo a posição da imagem à qual corresponde a face do cubo em questão, iterando no mesmo sentido em que o cubo é desenhado.

5.1.3 Esfera

Para obtermos o vetor das normais da esfera precisamos de ter em conta a orientação da origem do referencial até ao ponto em questão. Como ao longo do processo de desenho essa informação nos é disponibilizada, quando são determinadas as coordenadas de um vértice, estes podem ser usados para o vetor normal. Posto isto, resta-nos encontrar a direção da origem até ao ponto e não essa distância. Os pontos são calculados da seguinte forma:

```
(x,y,z)=(x/raio,y/raio,z/raio)
```

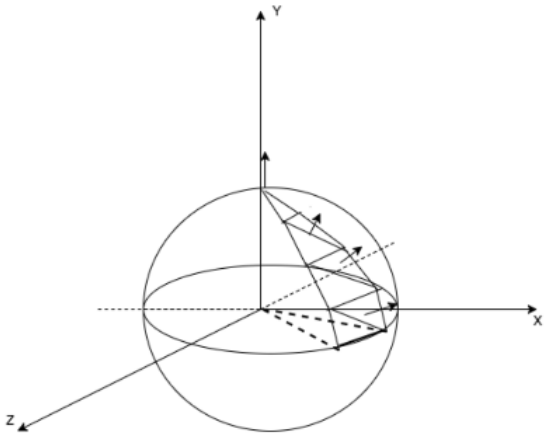

Figura 2: Vetores normais para uma slice da esfera e respetivo referencial cartesiano

Para calcular os pontos relativos às texturas, enquanto fazemos as iterações que descobrem os pontos para construir os triângulos que constituem a esfera, calculamos os pontos das texturas atribuindo a cada triângulo criado, um triângulo criado na imagem fornecida como textura.

5.1.4 Cone

Para calcularmos as normais de cada vértice, dividimos o cone em 2 partes: a base e o corpo. Sendo assim, as normais de cada vértice para a base e corpo são as seguintes:

- **Base** $\rightarrow (0, 0, -1)$
- **Corpo** $\rightarrow (sin(alpha), a/cH, cos(alpha))$ sendo que **a** referes-se ao comprimento do corpo e **cH** refere-se às camadas Horizontais. Com estes valores determinamos o vetor da normal fazendo **a/cH** que corresponde a Y.O X e o Z são determinados pelo **sin e cos** pois estes movem-se no sentido da circunferência. O **alpha** é a amplitude na qual o vértice se encontra.

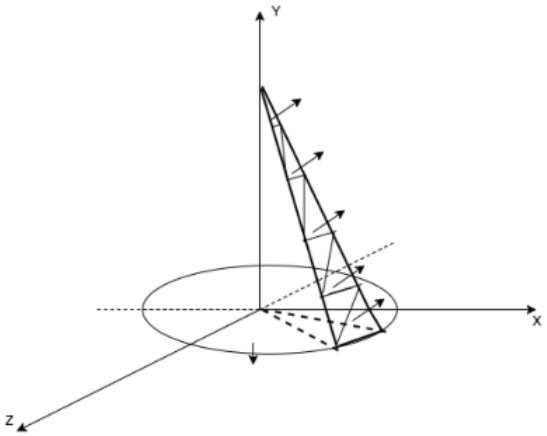

Figura 3: Vetores normais para uma slice de um cone e respetivo referencial cartesiano

5.1.5 Cilindro

A forma de obtenção dos vetores normais seguem a mesma ideia anterior, sendo que neste caso dividimos o Cilindro em 3 partes: Topo, Base e Corpo e obtemos os respetivos valores das normais da seguinte maneira:

- **Base** $\rightarrow (0, 0, -1)$
- **Topo** $\rightarrow (0, 0, 1)$
- **Corpo** $\rightarrow (sin(alpah), 0, cos(alpha))$ sendo que alpha indica a amplitude na qual o vértice se encontra.

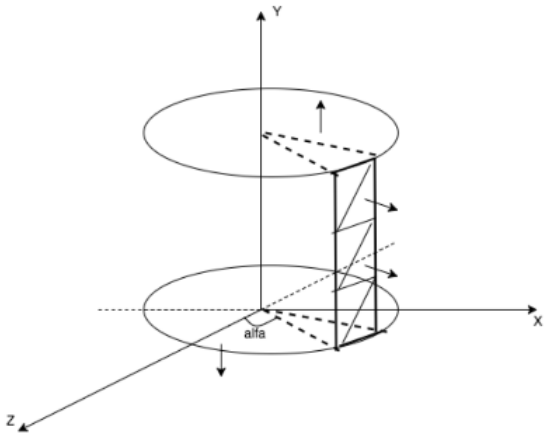

Figura 4: Vetores normais para uma slice de um cilindro e respetivo referencial cartesiano

5.1.6 Torus

Neste caso, a obtenção dos vetores das normais seguem o raciocínio da esfera, isto é, no momento de desenho conseguimos determinar a orientação de cada vetor normal, sendo que, devido ao facto de apenas precisarmos da orientação da origem até aos diferentes vértices, basta apenas determinar essa orientação através da seguinte fórmula:

cos(theta + shiftT)*cos(phi + shiftP), sin(theta + shiftT) * cos(phi + shiftP), sin(phi + shiftP)

Sendo que **phi + shiftP** representa o desvio de cada lado do anel e **theta + shiftT** representa o desvio do anel.

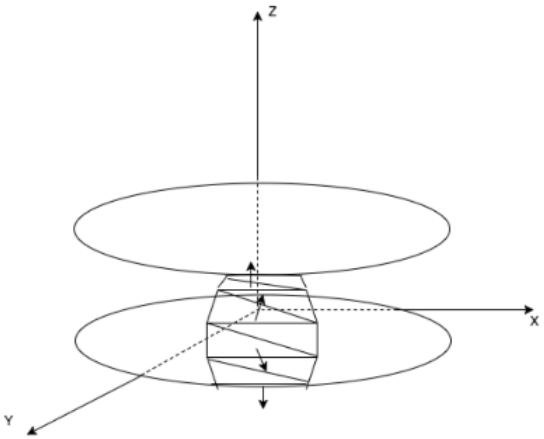

Figura 5: Vetores normais para uma slice de um torus e respetivo referencial cartesiano

Tal como na esfera, no Torus, também aproveitamos as iterações utilizadas na descoberta dos pontos dos triângulos para calcular o respetivo triângulo na imagem da textura.

6 Motor

6.1 Preparação dos VBOs e texturas

Nesta fase foi necessária a implementação de dois VBOs extraordinários ao dos pontos constituintes de um grupo. Desta maneira foi criado um para guardar as normais dos pontos e um para as coordenadas das texturas. Ao mesmo tempo, foi alterada a maneira de interpretação dos mesmo, i.e, a inicialização dos mesmos e carregamento das texturas durante o processo de leitura do ficheiro de configuração. Isto foi conseguido através do desenvolvimento dos métodos **setVBO()** e **newText()**, unicamente se o conjunto tiver uma textura associada.

6.1.1 Funcionalidades de iluminação

Para as propriedades de iluminação se manterem é necessário o cálculo correto das normais dos diversos modelos. Através da seguinte imagem podemos verificar o funcionamento das normais a um figura, no entanto o grupo optou por, em vez de calcular as normais para cada triângulo, calcular as mesmas mas para cada ponto constituinte da figura desejada, de maneira a conseguir uma aproximação mais credível da realidade.

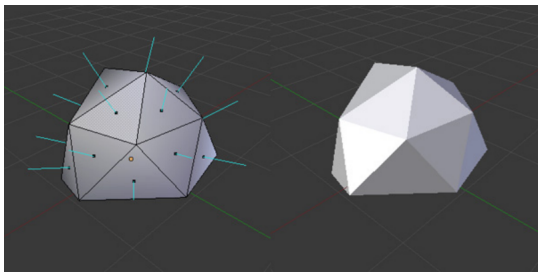

Figura 6: Normais de uma figura geométrica

Ao mesmo tempo, para conseguir esta aproximação realista, é necessário abordar diferentes parâmetros para a cor. Esta é constituída por 4 componentes:

- Diffuse colour - corresponde a cor que um objeto apresenta quando expoxto a luz branca, ou seja, define a cor do próprio objeto e não a reflexão da luz no mesmo.
- Ambient colour - cor que um objeto apresenta quando não está expoxto a nenhum feixe de luz.
- Emissive colour - cor que um objeto emite após ser expoxto a luz
- Specular light - brilho que aparece nos objetos quando iluminados

Segue-se um exemplo da combinação de alguns dos parâmetros anteriormente mencionados, de modo a criar um modelo realista.

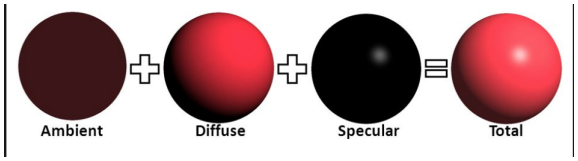

Figura 7: Combinação dos diferentes parâmetros da cor

6.1.2 Posição e direção da luminosidade

Ao construir um sistema de iluminação é importante ter em conta a origem da luz, pois é esta que vai proporcionar os ângulos de iluminação dos diferentes constituintes do nosso sistema solar. Ao mesmo tempo, também, é preciso ter em conta a direção da mesma. Assim, existem duas opções para a implementação deste sistema, sendo estas:

- Luz proveniente de um ponto que emite feixes de luz em todas as direções
- Luz proveniente de um ponto que emite um único feixe de luz direcionado

Desta maneira, a distinção entre estas duas situações é dada por um array **pos** constituído pelas coordenadas . No nosso caso, o pretendido é que a luz seja proveniente do Sol. Para isto, a posição será (0,0,0) e, como queremos que este provenha unicamente da estrela, a sua direção será para todos os lados, por isso, 1. Logo, a representação do array seria **pos[4] = {0,0,0,1}**. Em relação aos outros componentes:

```
GLfloat amb[3] = { 0.0, 0.0, 0.0 };
GLfloat diff[4] = { 1.0f, 1.0f, 1.0f, 1.0f };
GLfloat matt[3] = { 5, 5, 5 };
```

6.1.3 Processo de Renderização

O processo usado nesta fase foi maioritariamente semelhante ao da fase anterior, sendo acrescentadas funcionalidades em relação às texturas. Assim, antes de serem desenhadas as transformações em relação a um grupo é feita a verificação se este possui textura ou não, desenhando apenas se tiver, desta maneira é-nos permitido ter satélites com textura e satélites apenas com cor. Ao mesmo tempo, é na mesma função que se encontram implementadas as funções que permitem a existência de uma câmara FPS.

7 Desenvolvimento da Câmara

A câmara funciona à base de 4 teclas e do cursor. Sendo estas A,W,S,D, são responsáveis pelo movimento para a frente, para trás, para a esquerda e para a direita. Já o cursor é responsável pela mudança de direção da câmara. Estas duas funcionalidades agregam-se com o objetivo de cobrir todos os pontos possíveis que a posição da câmara pode tomar. Nesta fase, foi implementada a funcionalidade de poderem ser pressionadas várias teclas ao mesmo tempo, sendo executada a junção dos dois movimentos.

8 Resultados Obtidos

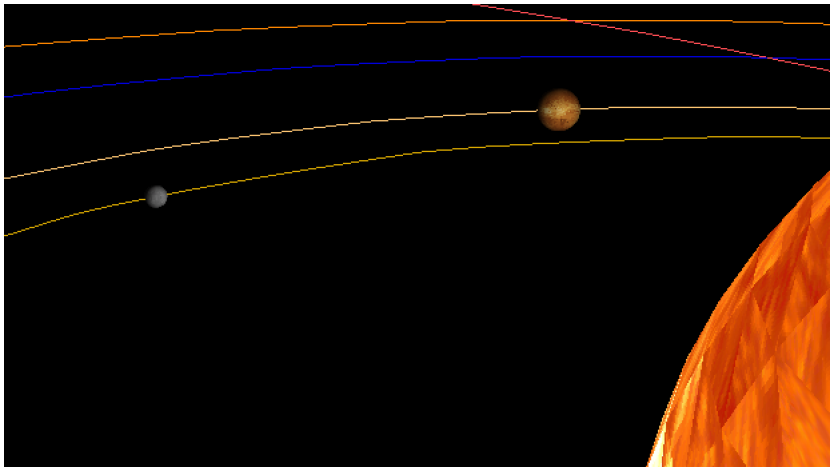

Figura 8: Visão parcial dos primeiros planetas do sistema solar

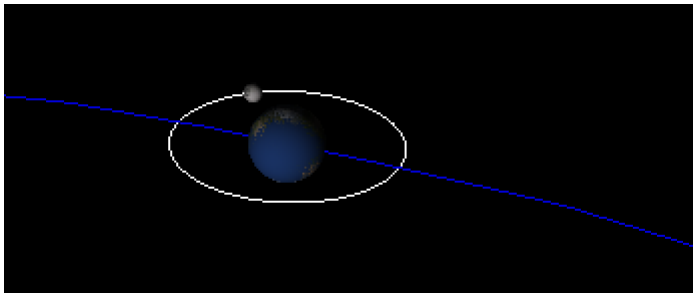

Figura 9: Visualização do planeta Terra

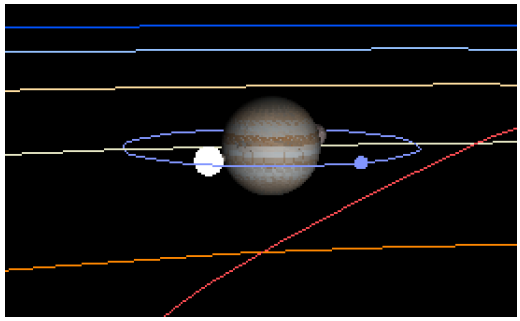

Figura 10: Visualização do planeta Júpiter

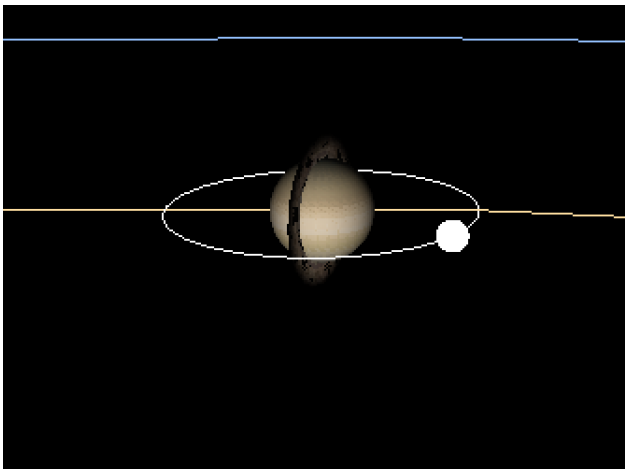

Figura 11: Visualização do planeta Saturno

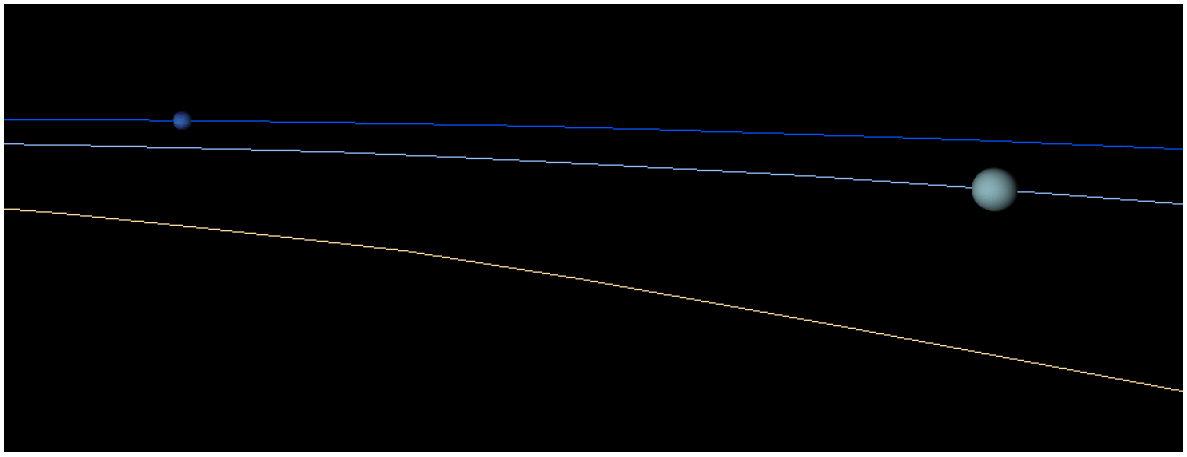

Figura 12: Visualização dos planetas Urano e Neptuno

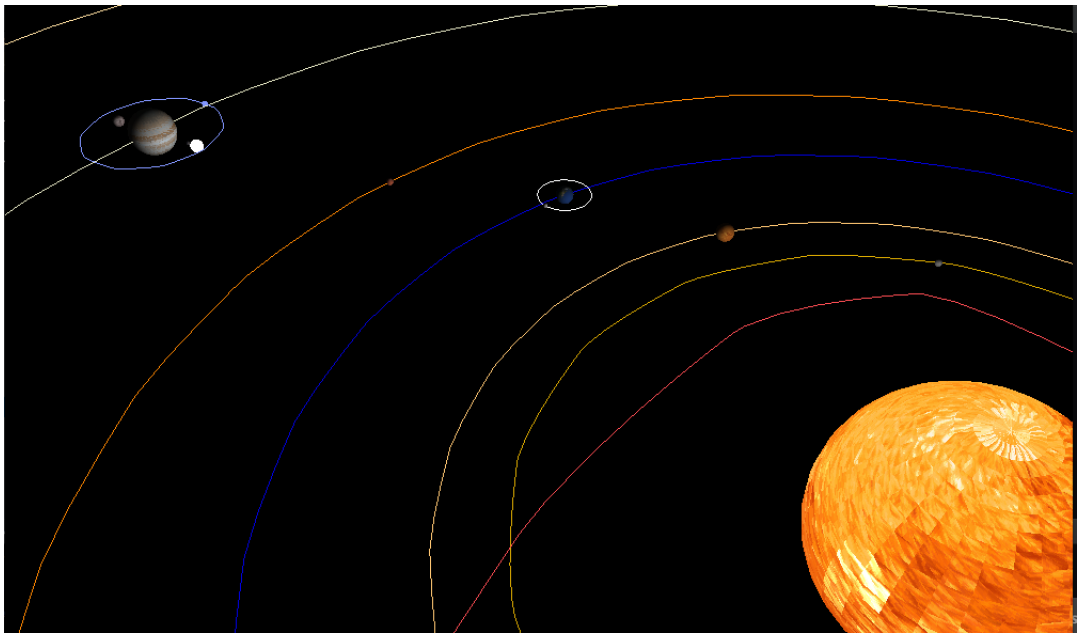

Figura 13: Visualização parcial do Sistema Solar

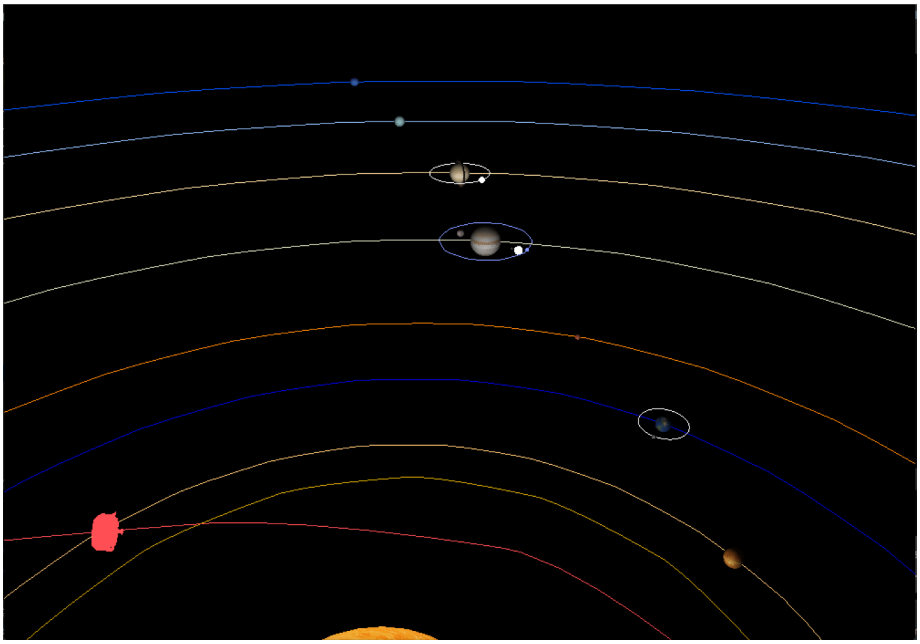

Figura 14: Visão do sistema visto do Sol

9 Conclusão

Durante a elaboração desta fase tivemos algumas dificuldades, que foram ultrapassadas devido às bases que fomos desenvolvendo ao longo das fase anteriores.

No fim desta quarta e última parte achamos que os objetivos globais foram alcançados com sucesso pois desenvolvemos um Sistema Solar animado minimamente realista, como pedido inicialmente.

Desta forma os consideramos que com o desenvolvimentos das diferentes partes deste projeto, que o nosso conhecimento em relação às ferramentas e utilidades do OpenGl e do GLUT cresceu bastante com o desenvolvimento destas.