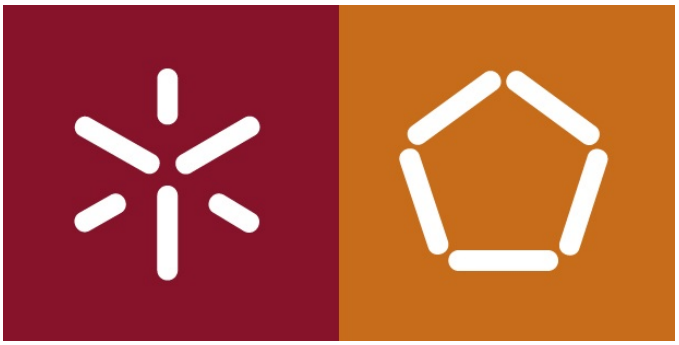
Universidade do Minho

Mestrado Integrado em Engenharia Informática



Computação Gráfica

Relatório do Trabalho Prático 4ªFase

Normals and Texture Coordiantes

Grupo 36

Carlos Gomes
A77185

Nuno Silva
A78156

15 de Maio de 2019

Conteúdo

Capítulo 1

Introdução e Resumo

1.1 Introdução

Foi-nos introduzido, no âmbito da Unidade Curricular de Computação Gráfica, a criação de um mecanismo 3D de representação gráfica de um cenário, sendo que para tal teríamos de utilizar várias ferramentas apresentadas nas aulas práticas entre as quais C++ e OpenGL.

Este trabalho foi dividido em quatro partes, sendo esta a quarta e última fase com o objectivo a inclusão de texturas e iluminação ao trabalho anteriormente desenvolvido, tendo como a criação do modelo animado do Sistema Solar.

1.2 Resumo

Como se trata da última parte do projeto prático, é natural que se mantenham uma grande parte das funcionalidades criadas nas fases anteriormente a esta e, por outro lado, algumas delas terem sofrido alterações de modo a melhorar o desempenho e cumprir com mais eficiência os requisitos necessários.

Assim, esta fase traz consigo duas novidades que irão levar a várias alterações, tanto ao nível do *engine* como do *generator*.

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

Passando para *engine*, este sofrerá algumas modificações e, ao mesmo tempo, receberá também novas funcionalidades. 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.

A última modificação, e não menos importante, que o engine sofrerá, será relativamente com a preparação dos **VBOs** e das texturas durante o processo de leitura de informação proveniente do ficheiro de configuração.

Com isto temos a finalidade de gerar eficazmente um modelo do Sistema Solar ainda mais realista do que o elaborado na fase anterior, pois este passará a dispor texturas e iluminação.

Capítulo 2

Arquitetura do Código

Tendo em mente a continuação do trabalho desenvolvido na fase anterior, e de várias opções de desenvolvimento, foi decidido manter as duas aplicações (**generator** e **engine**). Sendo esta última alvo de algumas modificações mais acentuadas tendo em vista o cumprimento dos requisitos necessários. Ficou tambem decidido a alteração da estrutura do trabalho, de forma a torna-lo mais organizado. Também é de notar que foi criada uma nova directoria onde estarão contida as imagens das texturas necessárias para a realização do projecto.

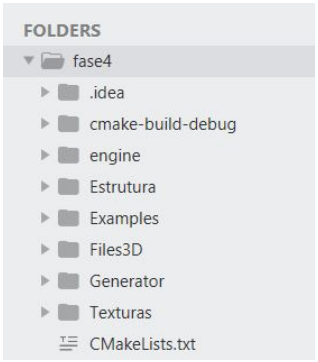


Figura 2.1: Organização do trabalho

2.1 Aplicações

Nesta secção, como se esperava, várias classes/funções tiveram que sofrer algumas alterações. Sendo assim iremos apresentar as classes que sofreram alterações mais significativas durante a realização desta fase ao nosso ver.

2.1.1 Gerador

generator.cpp : Para esta fase final do projeto, é pretendido que o gerador seja capaz de criar, para além dos pontos dos vértices já requisitados desde a primeira fase, as normais e coordenadas de textura a cada um destes. Portanto, para atingirmos esse meio, alteramos a "maneira"de como algumas funções funcionavam, como por exemplo a função sphere ou função plano e ao mesmo tempo tentamos organizar melhor o alinhamento do código para facilitar a leitura.

2.1.2 Motor

engine.cpp : Com as devidas alterações já implementadas no *Gerador*, é facil de identificar a necessidade de aplicação de funcionalidade de iluminação e de representação de texturas, através das respetivas normais e coordenadas de textura, respetivamente. Mas, no entanto também sofreu algumas alterações que iremos explorar ao longo do relatório.

2.2 Classes

Com a nova fase, iremos apenas explorar as classes que mais sofreram alterações e que sentimos a necessidade de explicitar essas alterações e o porquê de termos feito assim.

2.2.1 Group

Comecemos pela classe "mãe", a classe Group. Nesta fase, como foi pedido para calcularmos as coordenadas das normais e das texturas, sentimos a necessidade de acrescentarmos mais dois vectores de tipo Vertices que vão ter o nome de normal e textura para depois ser possível desenhar a figura completa e com textura inserida. Também houve a necessidade de termos que adicionar algumas variáveis extra para guardarmos informação sobre VBOs.

```
class Group {
    std::string nome;
    Transformacao* tranformacoes;
    std::vector<Vertices*> vertics;
    std::vector<Vertices*> normal;
    std::vector<Vertices*> textura;
    std::vector<Group*> filhos;
    //VBOS
    GLuint buffer[3];
    int nvertices;
    int nnormais;
    int ntexturas;
    float* v;
    float* n;
    float* t;
    unsigned int tt, width, height;
    unsigned int texID;
    unsigned char *data;
    std::string nomeTextura;

    public:
        Group();
        void setFilho(std::vector<Group*> g);
        void insereTransformacoes(Transformacao* t);
        void insereVerts(std::vector<Vertices*> vert);
        void insereNorms(std::vector<Vertices*> nms);
        void insereTexts(std::vector<Vertices*> texts);
        void insereFilho(Group* f);
        void insereNome(std::string name);
        Transformacao* getTransformacoes();
        std::vector<Group*> getFilhos();
        std::string getNome();
        void desenha();
        void VBO();
        void insereTextura(std::string text);
        std::string getTextura();
        void novaTextura();
        unsigned int getTexID();
```

2.2.2 Vertices

Na classe Vertices, manteve-se quase tudo só houve uma pequena ligeira alteração, inserimos a função normalCalc, que como o nome indica, serve para calcular as normais de cada vértice.

```
class Vertices {
    float vertice_x;
    float vertice_y;
    float vertice_z;


public:
    Vertices();
    Vertices(float x, float y, float z);
    float getX1();
    float getY1();
    float getZ1();
    void insereX1(float xx);
    void insereY1(float yy);
    void insereZ1(float zz);
    static Vertices* normalCalc(Vertices* v);
};
```

Capítulo 3

Gerador

3.1 Aplicação das Normais e planos de Texturas

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 um mapeamento de uma imagem 2D para uma figura 3D, revestindo a figura com o resultado deste mapeamento.

3.1.1 Plano

Para obter o conjunto dos vetores normais, de cada vértice, é unicamente necessário verificar o plano cartesiano em que este plano geométrico será desenhado. Como o plano foi desenhado no plano **XOZ**, todos os vértices possuirão a mesma normal, como tal todos os vértices partilham como normal o vetor (0,0,1).

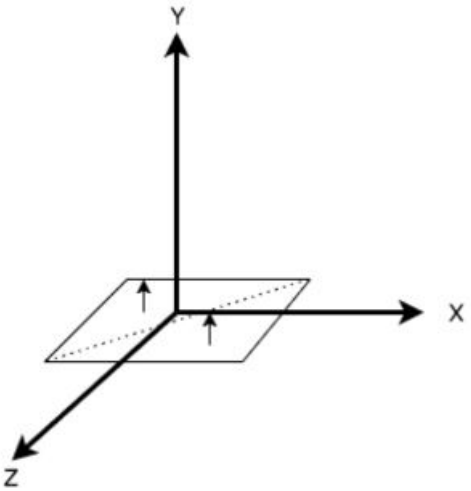


Figura 3.1: Referencial cartesiano, com os vetores normais do plano

3.1.2 Esfera

No sentido de 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)

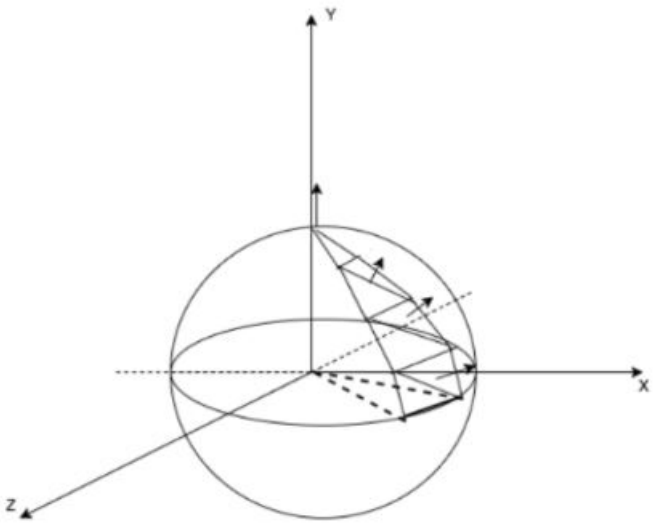


Figura 3.2: Vetores normais para uma slice da esfera e respetivo referencial cartesiano

Para calcular a normal de cada vértice, o grupo desenvolveu a seguinte função.

```
Vertices* Vertices::normalCalc(Vertices* v) {
    float l, coordX, coordY, coordZ;

    l = sqrt(v->vertice_x * v->vertice_x + v->vertice_y * v->vertice_y + v->vertice_z * v->vertice_z);
    coordX = v->vertice_x / l;
    coordY = v->vertice_y / l;
    coordZ = v->vertice_z / l;

    return new Vertices(coordX, coordY, coordZ);
}
```

Figura 3.3: Algoritmo, extraido do ficheiro *Vertices.cpp*, usado para o calculo das normas de um vértice

De seguida os valores calculados serão guardados num vetor, para posteriormente serem escritos num ficheiro, para ai serem interpretados, mais tarde no parser.

Também é possível reparar que a função sphere sofreu alterações comparando com as outras fases. Ora, na primeira versão desta função era possível retornar um vector de vertices que criava a esfera e de seguida pegavamos nesse respetivo resultado e passavamos como argumento na função escreverFicheiro para escrever o ficheiro sphere.3d. Mas, como

agora é necessário também escrever as normais e as texturas, decidimos fazer isso tudo só na função sphere, sendo assim necessário fazer algumas mudanças em relação a função.

Figura 3.4: Comparando o algoritmo anterior com o algoritmo atual

Capítulo 4

Motor

4.1 Preparação dos VBOs e texturas

Para esta 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.

4.1.1 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 numa figura.Sendo assim, optamos 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 credivel da realidade.

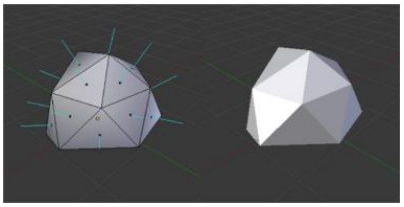


Figura 4.1: Normais de uma figura geométrica

De forma a 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 exposto a luz branca, ou seja, define a cor própria do objeto e não a reflexão da luz no mesmo.
- **Ambient colour** - cor que um objeto apresenta quando não está exposto a nenhum feixe de luz.

- **Emissive colour** - cor que um objeto emite após ser exposto a luz
- **Specular colour** - A cor da luz da relfexão especular, sendo esta o tipo de reflexão caracteristica de uma superficie brilhante.

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

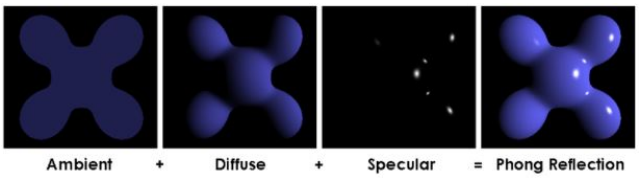


Figura 4.2: Exemplo de combinação dos diferentes componentes de cor

4.1.2 Posicionamento e Direcionamento da iluminação

4.1.3 Processos de renderização

O processo utilizado nesta fase foi algo semelhante ao da fase anterior, sendo acrescentadas funcionalidades relativamente às texturas. Assim, é-nos permitido desenhar planetas ou satélites com texturas ou com cor, conforme a vontade do utilizador.

Capítulo 5

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. Sendo o cursor, reponsável 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. No entanto, também fomos "obrigados"a ter que remover algumas funcionalidades que existiam nas fases anteriores, como por exemplo as opções de desenho das figuras.

Capítulo 6

Resultados Obtidos

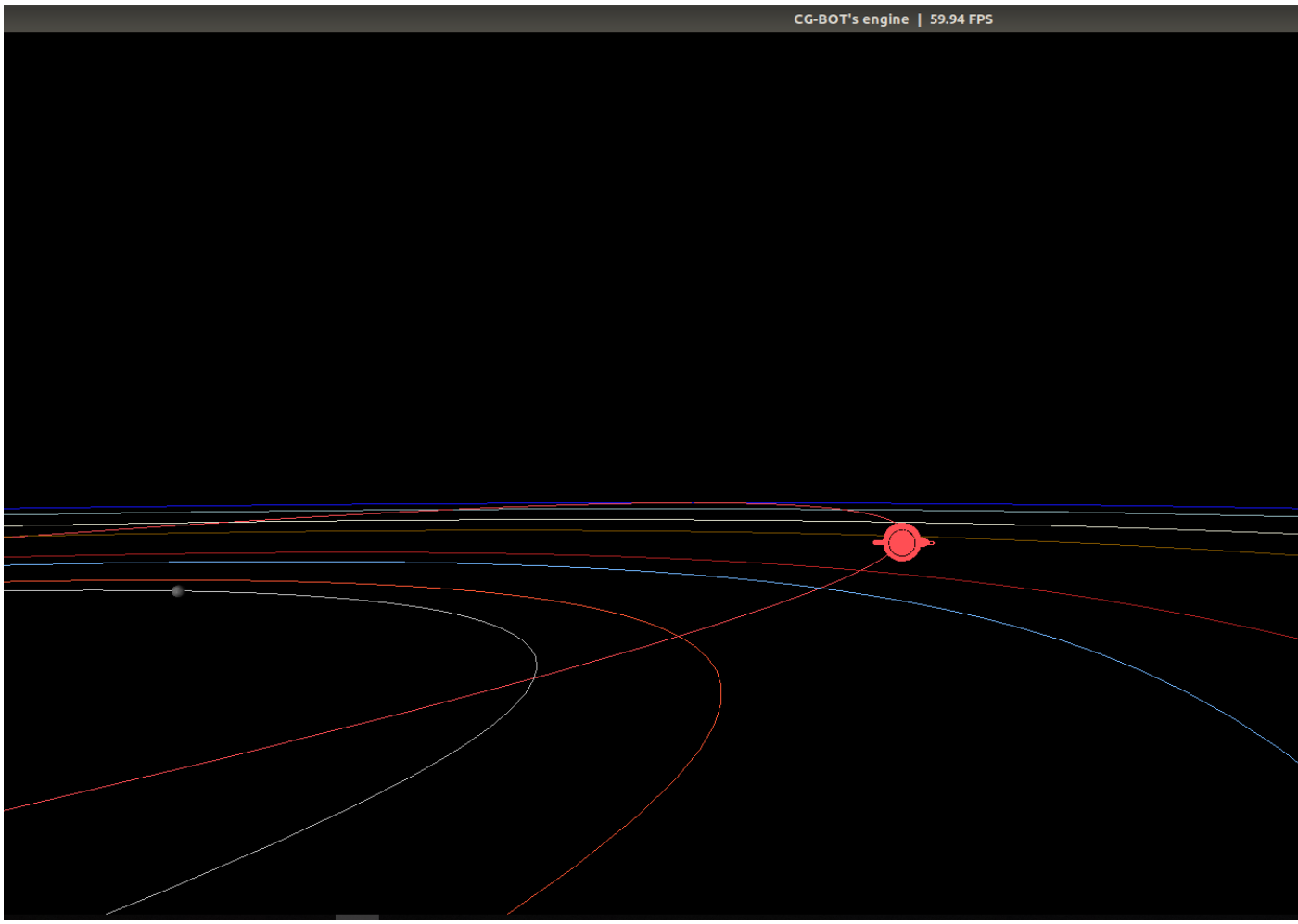


Figura 6.1: Tea cup

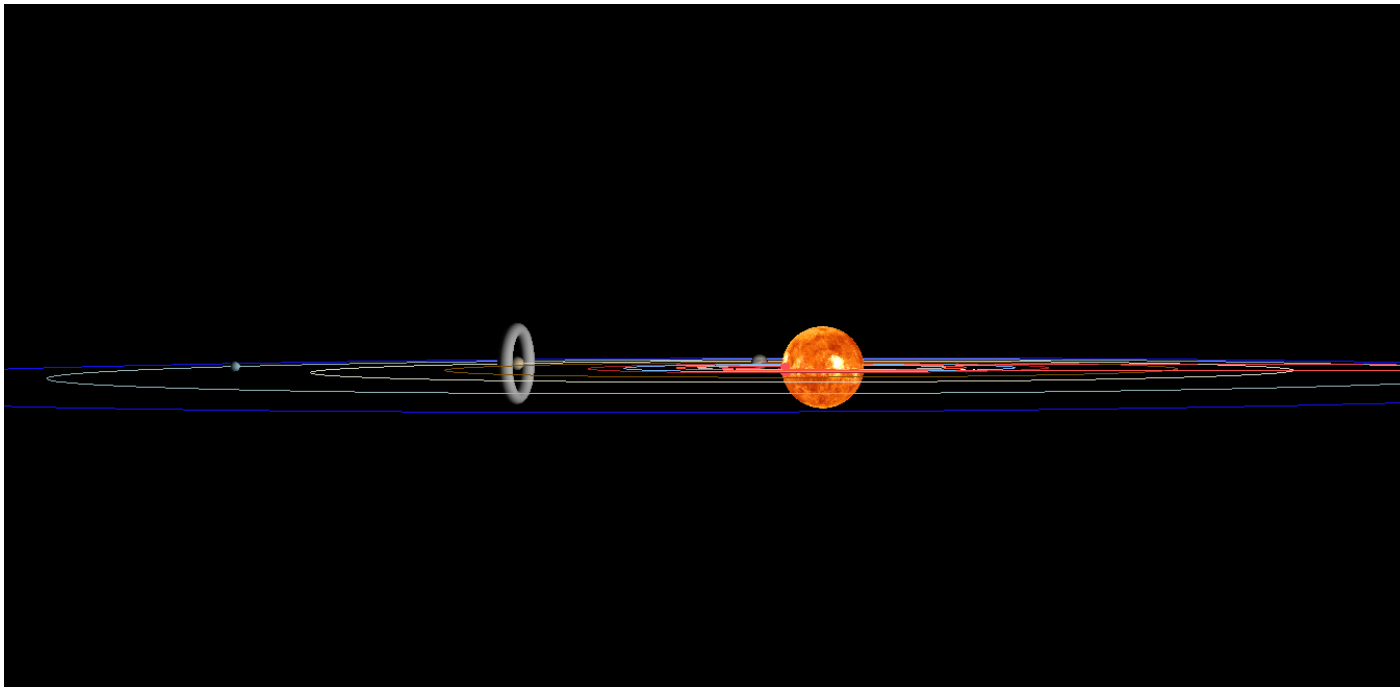


Figura 6.2: Sistema Solar

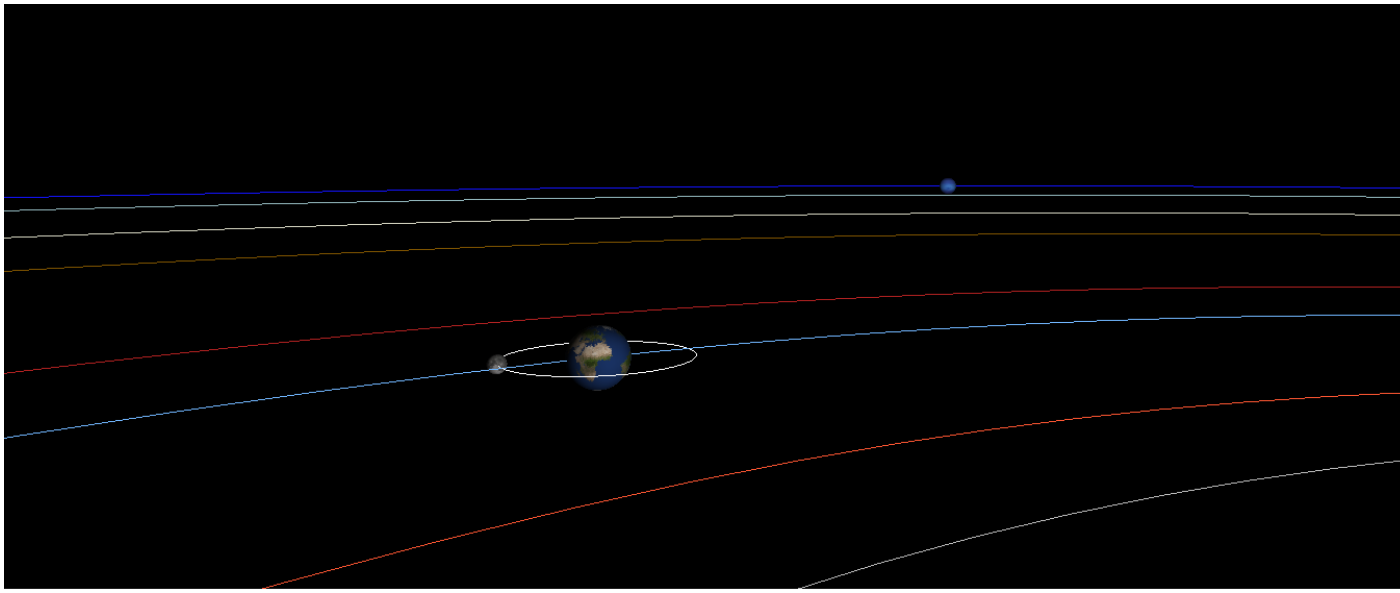


Figura 6.3: Terra e Lua

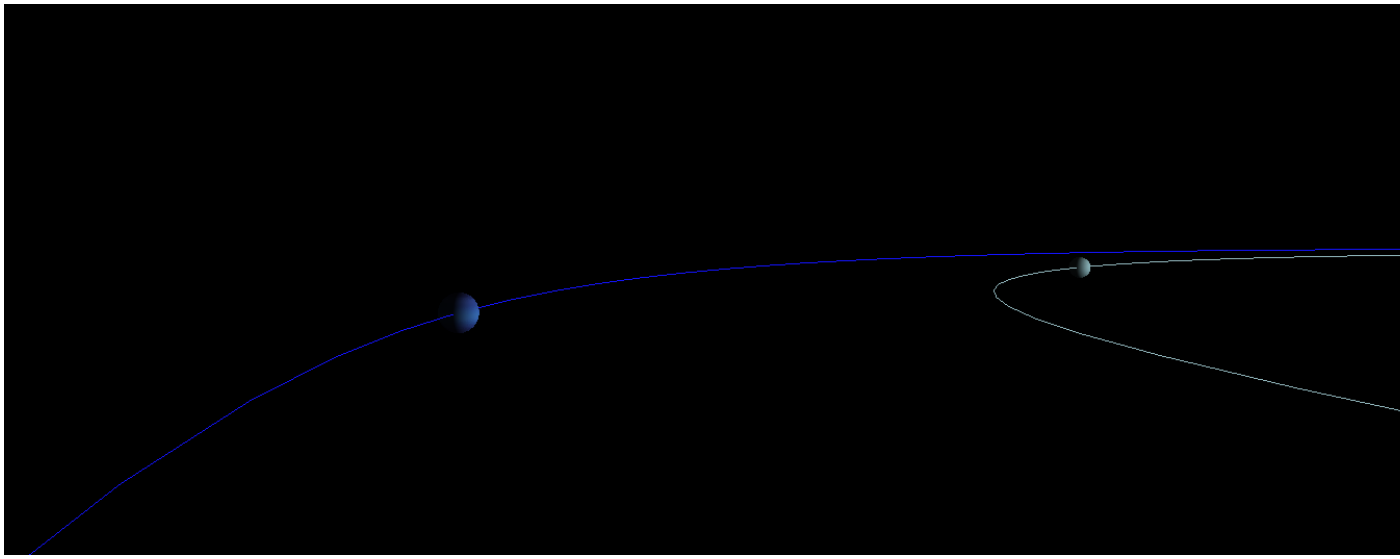


Figura 6.4: Outros planetas..

Capítulo 7

Conclusão/Trabalho futuro

Dado por terminada esta ultima fase, após uma pequena introspeção o grupo conclui que o resultado final é totalmente satisfatorio.

Contundo nesta durante o desenvolvimento desta fase consideramos que obtivemos uma maior dificuldade na aplicação da iluminação, bem como nas texturas em certas ocasiões.

Assim sendo, e após a superação de todas adversidades encontradas, consideramos que o nosso conhecimento no que diz respeito à utilização das funcionalidades do OpenGl e do GLUT cresceu exponencialmente, na medida que consideramos bastante aptos e com boas bases para desenvovler projetos futuros.