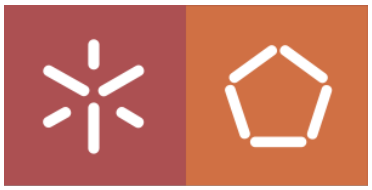

Universidade do Minho
Escola de Engenharia

Universidade do Minho
Mestrado Integrado em Engenharia Informática

**Computação Gráfica**

# Projeto Computação Gráfica
# 4ª Fase

31 Maio 2020


Ana Margarida Campos
(A85166)


Ana Catarina Gil
(A85266)


Tânia Rocha
(A85176)

# Contents

**1 Introdução** **4**
1.1 Contextualização . . . . . 4
1.2 Resumo do trabalho desenvolvido . . . . . 4

**2 Trabalho desenvolvido nesta fase** **5**
2.1 Generator . . . . . 5
2.1.1 Gerar Normais . . . . . 5
2.1.2 Gerar Texturas . . . . . 7
2.2 Texturas . . . . . 9
2.2.1 Extra . . . . . 10
2.3 Luzes . . . . . 11
2.4 Ficheiro XML . . . . . 12
2.5 Resultado Final . . . . . 13
2.6 Conclusão . . . . . 17

# List of Figures


2.1 cálculo da normal de um ponto numa esfera . . . . . 5
2.2 Comparação do Ponto e da Normal na esfera . . . . . 6
2.3 Comparação do Ponto e da Normal no disco . . . . . 6
2.4 Método de cálculo das normais . . . . . 6
2.5 Comparação dos Pontos com as Normais no teapot . . . . . 7
2.6 Comparação dos Pontos com as Normais na cintura . . . . . 7
2.7 cálculo devido à escala das texturas . . . . . 8
2.8 Pontos de Textura da esfera . . . . . 8
2.9 Exemplo do cálculo dos pontos de textura . . . . . 8
2.10 Pontos para a textura do anel . . . . . 8
2.11 Pontos para a textura do teapot . . . . . 9
2.12 cálculo para a escala . . . . . 9
2.13 cálculo dos pontos de textura . . . . . 9
2.14 Estruturas de dados referentes à Luz . . . . . 11
2.15 Atribtuto *texture* no caso do planeta Terra . . . . . 12
2.16 Cor emissiva no Sol . . . . . 12
2.17 Implementação da Luz no ficheiro xml . . . . . 12
2.18 Foto do Sistema Solar . . . . . 13
2.19 Foto do Sistema Solar . . . . . 13
2.20 Foto do Sistema Solar . . . . . 14
2.21 Foto do Sistema Solar . . . . . 14
2.22 Foto do Sistema Solar . . . . . 15
2.23 Foto do Sistema Solar . . . . . 15
2.24 Foto do Sistema Solar . . . . . 16
2.25 Cometa Teapot . . . . . 16

# Introdução

## 1.1 Contextualização

O presente relatório foi elaborado no âmbito da unidade curricular de Computação Gráfica de maneira a explicar o trabalho desenvolvido na quarta e última fase do projeto prático.

Os principais objetivos desta fase do trabalho foram a criação de coordenadas de textura e normais para cada vértice bem como a implementação das funcionalidades de iluminação e textura no Sistema Solar.

## 1.2 Resumo do trabalho desenvolvido

De maneira e incorpar todos os objetivos desta fase no projeto foi necessário acrescentar novas componentes ao trabalho desenvolvido nas outras fases. Uma vez que todos os modelos utilizados no Sistema Solar deverão passar a ter texturas e normais foi adicionado ao *generator* o cálculo destas primitivas e a impressão das mesmas nos ficheiros *.3d* para depois serem renderizadas corretamente. Relativamente ao *engine*, foi necessário alterar a maneira de ler os ficheiros *.3d* uma vez que agora é necessária a leitura das normais e texturas e foi também necessária a implementação de carregar imagens provenientes do ficheiro XML bem como interpretar as luzes.

# Trabalho desenvolvido nesta fase

## 2.1 Generator

Até à fase anterior, o programa *Generator* gerava cooordenadas dos pontos constituíntes dos modelos. Para podermos processeguir à aplicação das texturas, este programa teve de ser alterado de modo a passar a calcular também as coordenadas normais e de textura de cada ponto.

### 2.1.1 Gerar Normais

As normais dos pontos são gerados para que a partir de uma fonte de luz, seja possivel gerar uma sombra de uma primitiva respetiva. A normal de um ponto é calculada por face.

Foram então alteradas a primitivas gráficas de maneira a terem um vetor normais que possui o conjunto das normais dos seus vertices. Deste modo, a geração destas normais foi efetudada nos métodos de geração dos vertices de cada modelo e posteriormente armazenadas no ficheiro .3d respetivo.

#### Esfera

No case da esfera, as normais são implementadas de uma maneira simples, visto que o próprio ponto é o vetor sem a multiplicação pelo raio, ou seja, o vetor normal inicia-se no centro da esfera até ao respetivo ponto ao qual queremos calcular.

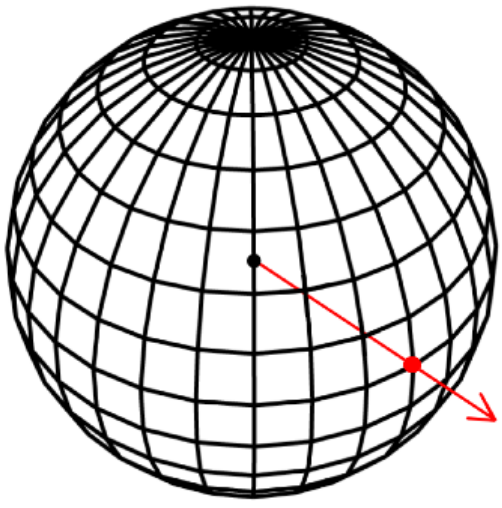

Figure 2.1: cálculo da normal de um ponto numa esfera

Comparando a estrutura para guardar os pontos e as normais reparamos que, de facto, a única diferença entre os dois é a multiplicação das coordenadas com o raio, no caso do Ponto:

```
pontos.push_back(Ponto( a: raio * cosf( x: stack*beta) * sinf( x: slice*alpha),  b: raio * sinf( x: stack*beta),  c: raio * cosf( x: stack*beta) * cosf( x: slice*alpha)));
pontos.push_back(Ponto( a: raio * cosf( x: (stack - 1)*beta) * sinf( x: slice*alpha),  b: raio * sinf( x: (stack - 1)*beta),  c: raio * cosf( x: (stack - 1)*beta) * cosf( x: slice*alpha)));

normais.push_back(Ponto( a: cosf( x: stack*beta) * sinf( x: slice*alpha),  b: sinf( x: stack*beta),  c: cosf( x: stack*beta) * cosf( x: slice*alpha)));
normais.push_back(Ponto( a: cosf( x: (stack - 1)*beta) * sinf( x: slice*alpha),  b: sinf( x: (stack - 1)*beta),  c: cosf( x: (stack - 1)*beta) * cosf( x: slice*alpha)));
```

Figure 2.2: Comparação do Ponto e da Normal na esfera

### Disco

No caso do disco, obtenção dos vetores seguem o o mesmo raciocionio do cálculo dos pontos, visto que é uma figura 2D mas com a contrução de ambos os lados:

```
pontos.push_back(Ponto( a: inner * cos(angle),  b: inner * sin(angle), c: 0));
pontos.push_back(Ponto( a: outer * cos(angle),  b: outer * sin(angle), c: 0));

normais.push_back(Ponto( a: inner * cos(angle),  b: inner * sin(angle), c: 0));
normais.push_back(Ponto( a: outer * cos(angle),  b: outer * sin(angle), c: 0));
```

Figure 2.3: Comparação do Ponto e da Normal no disco

### Teapot

Tal como nos casos anteriores, as normais do teapot são calculadas no mesmo metodo onde são calculados os pontos do modelo. Neste caso, foi necessário criar ainda um outro método para permitir gerar as normais, visto que o teapot foi contruído através dos paches de Bezier fornecidos pelos docentes da UC. O método desenvolvido é o seguinte:

```
Ponto pontoBezierNormal(int p, float u, float v) {
    Ponto ponto = Ponto( a: 0.0,  b: 0.0,  c: 0.0);

    for (int j=0; j<4; j++){
        for (int i=0; i<4; i++) {

            int indexPatch = j*4+i;
            int indexCP = patches[p*16 + indexPatch];
            ponto = Ponto( a: ponto.getX() + controlPoints[indexCP * 3 + 0],
                           b: ponto.getY() + controlPoints[indexCP * 3 + 1],
                           c: ponto.getZ() + controlPoints[indexCP * 3 + 2]);
        }
    }
    return ponto;
}
```

Figure 2.4: Método de cálculo das normais

Como podemos observar este método calcula as normais de um ponto e devolve esse mesmo ponto para ser posteriormente utilizado para guardar as normais para futuramente serem utilizadas da seguinte maneira:

```
pontos.push_back(pontoBezier(p, u: (u+inc), v: (v+inc)));
pontos.push_back(pontoBezier(p, u, v));
pontos.push_back(pontoBezier(p, u, v: (v+inc)));

normais.push_back(pontoBezierNormal(p, u: (u+inc), v: (v+inc)));
normais.push_back(pontoBezierNormal(p, u, v));
normais.push_back(pontoBezierNormal(p, u: (u+inc), v));
```

Figure 2.5: Comparação dos Pontos com as Normais no teapot

Assim, através do cálculo da normal naquele ponto, o programa guarda no vetor *normais* os pontos correspondentes para o vetor normal.

**Cintura**

No caso da cintura, como, tal como explicado nas fases anteriores, é constituída por esferas, o método de cálculo dos vetores normais é o mesmo que na esfera. Em cada esfera gerada constituínte da cintura é calculado o vetor normal de maneira semelhante à esfera normal e ponteriormente armazenado no vetor *normais*.

```
pontos.push_back(Ponto( a: ondex + r * cos( _X: (q + 1) * beta) * sin( _X: (i + 1) * alpha), b: r * sin( _X: (q + 1) * beta),
          c: ondez + r * cos( _X: (q + 1) * beta) * cos( _X: (i + 1) * alpha)));
pontos.push_back(Ponto( a: ondex + r * cos( _X: q * beta) * sin( _X: i * alpha), b: r * sin( _X: q * beta),
          c: ondez + r * cos( _X: q * beta) * cos( _X: i * alpha)));
pontos.push_back(Ponto( a: ondex + r * cos( _X: q * beta) * sin( _X: (i + 1) * alpha), b: r * sin( _X: q * beta),
          c: ondez + r * cos( _X: q * beta) * cos( _X: (i + 1) * alpha)));

normais.push_back(Ponto( a: ondex + r * cos( _X: (q + 1) * beta) * sin( _X: (i + 1) * alpha), b: r * sin( _X: (q + 1) * beta),
                         c: ondez + r * cos( _X: (q + 1) * beta) * cos( _X: (i + 1) * alpha)));
normais.push_back(Ponto( a: ondex + r * cos( _X: q * beta) * sin( _X: i * alpha), b: r * sin( _X: q * beta),
                         c: ondez + r * cos( _X: q * beta) * cos( _X: i * alpha)));
normais.push_back(Ponto( a: ondex + r * cos( _X: q * beta) * sin( _X: (i + 1) * alpha), b: r * sin( _X: q * beta),
                         c: ondez + r * cos( _X: q * beta) * cos( _X: (i + 1) * alpha)));
```

Figure 2.6: Comparação dos Pontos com as Normais na cintura

Este método de geração de vetores normal, neste caso, permite que a cintura não seja vista como um só, mas sim como partes individuais, neste caso as esferas, que refletem a luz do sol individualmente.

### 2.1.2 Gerar Texturas

Para conseguirmos inserir texturas nos objetos existentes no nosso sistema solar é necessário definir os pontos textura para cada uma dessas figuras e posteriormente inserir no ficheiro .3d correspondente.

No caso das coordenadas de textura servem para que se possa atribuír uma texutra a um determinado modelo. Cada ponto tera um par de cordenada $x$ e $\mathbf{y}$, que varia, entre 0 e 1. Estas coordenadas são utilizadas pelo *OpenGL* para saber que pontos da imagem a ser utilizada como textura corresponte ao ponto do modelo em questão.

**Esfera** No caso da esfera, estes pontos de textura foram divididos pelo numero de slices e stacks, visto que no caso das texturas o comprimento e a largura há sujeição a uma escala de 0 a 1. O cálculo desta medida foi efetuado assim:

```
float itStacks = 1.0f/((float) stacks);
float itSlices = 1.0f/((float) slices);
float cSlices = 0;
float cStacks = 0;
```

Figure 2.7: cálculo devido à escala das texturas

Posteriormente, tal como feito no cálculo das normais, a textura em cada ponto também vai ser efetuada no mesmo método de geração de pontos de cada modelo e de seguida armazenada para ser escrita no ficheiro .3d.

```
texturas.push_back(PontoText(cSlices,  b: cStacks - itStacks));
texturas.push_back(PontoText(cSlices, cStacks));
```

Figure 2.8: Pontos de Textura da esfera

Por fim é efetuado o armazenamento em VBOs e tal como podemos observar, este é feito a partir apenas de duas coordenadas como afirmado anteriormente:

```
for (unsigned int textura = 0; textura < texturas.size(); textura++){
    fprintf(f,  _Format: "%f %f\n", texturas[textura].getX(), texturas[textura].getY());
}
```

Figure 2.9: Exemplo do cálculo dos pontos de textura

**Anel**

No caso do disco, nome dado à nossa função, mais concretamente o anel dos planetas, a textura teve de ser implementada de uma maneira ligeiramente diferente, visto que o anel tem pontos apenas no seu limite exterior e interior. Neste caso, torna-se então desnecessário o cálculo dos pontos de textura, visto que apenas é necessário que seja definido como ponto de textura cada coordenada do limite inferior e exterior, ou seja:

```
for (unsigned int i = 0; i < pontos.size(); i++) {
    fprintf(f,  _Format: "%f %f\n", 0.1f, 0.5f);
    fprintf(f,  _Format: "%f %f\n", 0.9f, 0.5f);
}
```

Figure 2.10: Pontos para a textura do anel

**Teapot**

Para o teapot, tal como na geração de normais, foi apenas necessário criar os pontos para a textura no mesmo método que se geraram os pontos do modelo:

```
texturas.push_back(PontoText( a: (u+inc),  b: (v+inc)));
texturas.push_back(PontoText(u, v));
texturas.push_back(PontoText(u,  b: (v+inc)));
```

Figure 2.11: Pontos para a textura do teapot

**Cintura**

Por fim , no caso da cintura, tal como realizado para as normais, visto que a cintura é constituída por pequenas esferas, os pontos de textura foram calculados para cada esfera. E como observado no caso da esfera foi necessário fazer o cálculo a partir das stacks e das slices para ajustar à escala das texturas:

```
float itStacks = 1.0f/((float) stacks);
float itSlices = 1.0f/((float) slices);
float cSlices = 0;
float cStacks = 0;
```

Figure 2.12: cálculo para a escala

É possivel então calcular os pontos da textura:

```
texturas.push_back(PontoText( a: (u+inc),  b: (v+inc)));
texturas.push_back(PontoText(u, v));
texturas.push_back(PontoText(u,  b: (v+inc)));
```

Figure 2.13: cálculo dos pontos de textura

## 2.2 Texturas

Por predefinição, é o *Engine* que carrega todos os modelos a serem executados no sistema solar e respetivas texturas. Para aplicação das texturas, no ficheiro XML deve estar presente junto a cada modelo o atributo *texture*, juntamente com o ficheiro .3d respetivo.

Quando é chamado o atributo *texture* é então carregado, recorrendo à função *loadTexture*.

Um nota importante é ainda que dentro deste frupo é ainda possivel definir variáveis de luz para permitir que estes modelos estejam "iluminados".

### 2.2.1 Extra

Para além dos modelos já aqui apresentados foi ainda desenvolvido um "background" para o nosso sistema solar. Para tal, construímos uma esfera cujo lado visivel é o seu interior e geramos a mesma com grandes dimenções de maneira a que o sistema solar esteja no seu interior. Posteriormente, adicionamos uma textura à mesma, tal como nas outras esferas apresentadas, permitindo assim criar um ambiente "espacial" no nosso sistema solar.

## 2.3 Luzes

Para acrescentar a funcionalidade de iluminação no nosso sistema tivemos de implementar a leitura da nova tag *<Luzes>*.

Na função de leitura dos dados contidos no *scene.xml* sempre que é lida uma nova luz (contida na tag anteriormente referida) começamos por fazer glEnable da luz e posteriormente iterar cada um dos seus elementos. Apesar de no nosso trabalho apenas usarmos uma fonte de luz (partindo do sol), devido ao facto do OpenGl suportar apenas oito luzes em simultâneo, decidiu-se limitar a leitura e coerentemente o armazenamento de diferentes luzes para este mesmo valor.

Dentro desta tag decidiu-se associar o tipo de luz, apesar do nosso trabalho suportar apenas a luz do tipo pontual, as coordenadas de posição da mesma, e as quatro compontes da luz para a definir, nomeadamente as componentes ambient, specular, diffuse e emission. Estes dados vão ser aramzenados na nova estrutura de dados, denominada por *Luz*. Esta irá conter também um ID que identifica cada luz.

```
class Material {
public:
    float *diffuse=NULL, *specular=NULL, *emissive=NULL, *ambient=NULL;
};

class Luz {
public:
    string type;
    float pos[4];
    Material color;
    float cons=NULL, quad=NULL, linear=NULL;
    unsigned int id;
};
```

Figure 2.14: Estruturas de dados referentes à Luz

Posteriormente, quando no código for chamada a função *drawGroup* esta chama a função luzes, que é responsável por gerar e tratar as fontes de luz de acordo com os dados armazendos e com o auxílio da funções do OpenGL.

Ainda ligado a esta componente da luz, foi necessário dar "luz própria" ao Sol, ou a qualquer outro elemento que achassemos necessário a possuir. Para cobrir esta funcionalidade foi necessário adicionar mais uma componente à estrutura *Figura*, nomeandamente uma variavél do tipo *Material*, sendo esta uma das structs anteriormente ilustradas. Para esta parte apenas se irá usar as variáveis ligadas à luz emissiva.

## 2.4 Ficheiro XML

De maneira a ser possível implementar as novas funcionalidades ao sistema foi necessária a adição de novas *tags* e atributos ao ficheiro XML da fase anterior.

Uma vez que todos os elementos do Sistema Solar passaram a possuir textura, foi adicionado dentro de cada *model*, junto ao atributo *file* que indica o ficheiro *.3d* utilizado, o atributo *texture* que faz referência à imagem representativa da textura.

```
<models>
    <model file="sphere.3d" texture="terra.jpg" />
</models>
```

Figure 2.15: Atribtuto *texture* no caso do planeta Terra

Para além disso, em alguns elementos do Sistema Solar, nomeadamente no sol e no cosmos, foi adicionada a componente emissiva uma vez que esta permite emitir a cor do objeto em todas as direções. Para tal são definidas três componentes correspondentes às 3 cores primárias: *emiR*, *emiG* e *emiB*.

```
<models>
    <model file="sphere.3d" texture="sol.jpg" emiR="1" emiG="1" emiB="1"/>
</models>
```

Figure 2.16: Cor emissiva no Sol

De maneira a implementar a iluminação, neste caso proveniente do sol, foi adicionado no ínicio do ficheiro um ponto de luz, posicionado nas coordenadas (0,0,0) (coordenadas do sol) que possui as componentes difusa e ambiente. Para tal foram adicionadas a tag *Luzes* e as diversas componentes da luz.

```
<Luzes>
    <luz type="POINT" X="0" Y="0" Z="0" diffR="1" diffG="1" diffB="1" ambR="1" ambG="1" ambB="1"/>
</Luzes>
```

Figure 2.17: Implementação da Luz no ficheiro xml

Uma vez que anteriormente os elementos do Sistema Solar estavam desenhados à escala e de maneira ser possível uma melhor compreensão e visão da fase final do projeto, decidimos aumentar a escala a todos os componentes do sistema. Para tal foi também alterado no ficheiro XML as escalas e as coordenadas correspondentes às órbitas dos planetas e luas.

## 2.5 Resultado Final

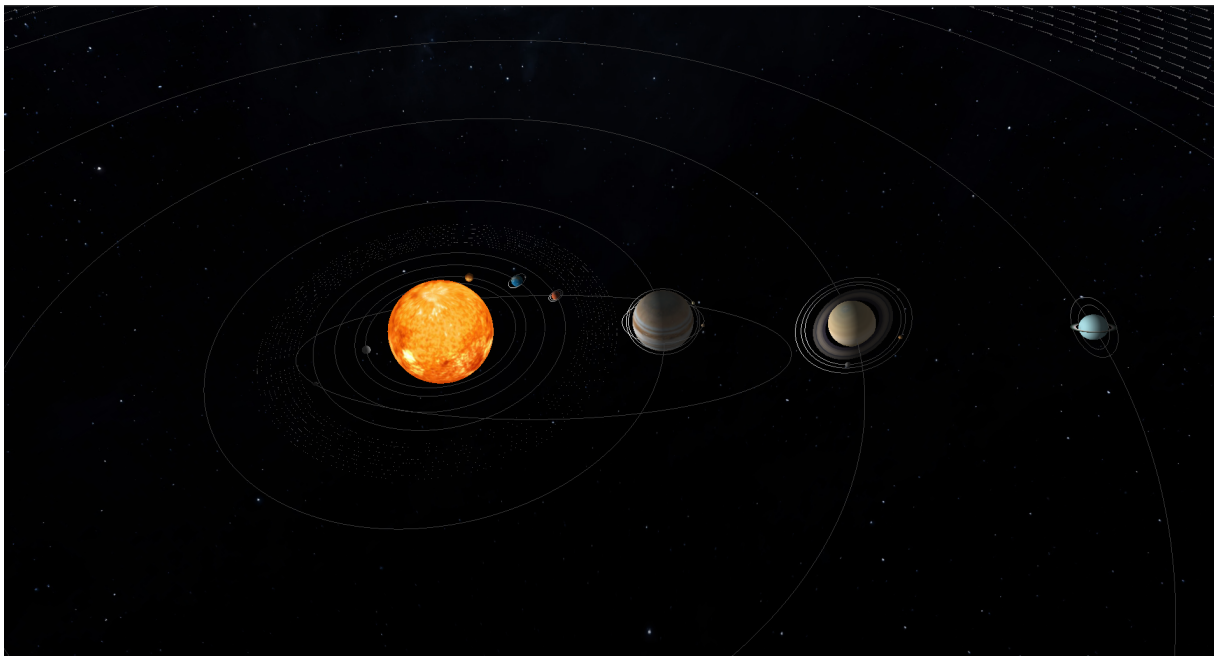

Figure 2.18: Foto do Sistema Solar

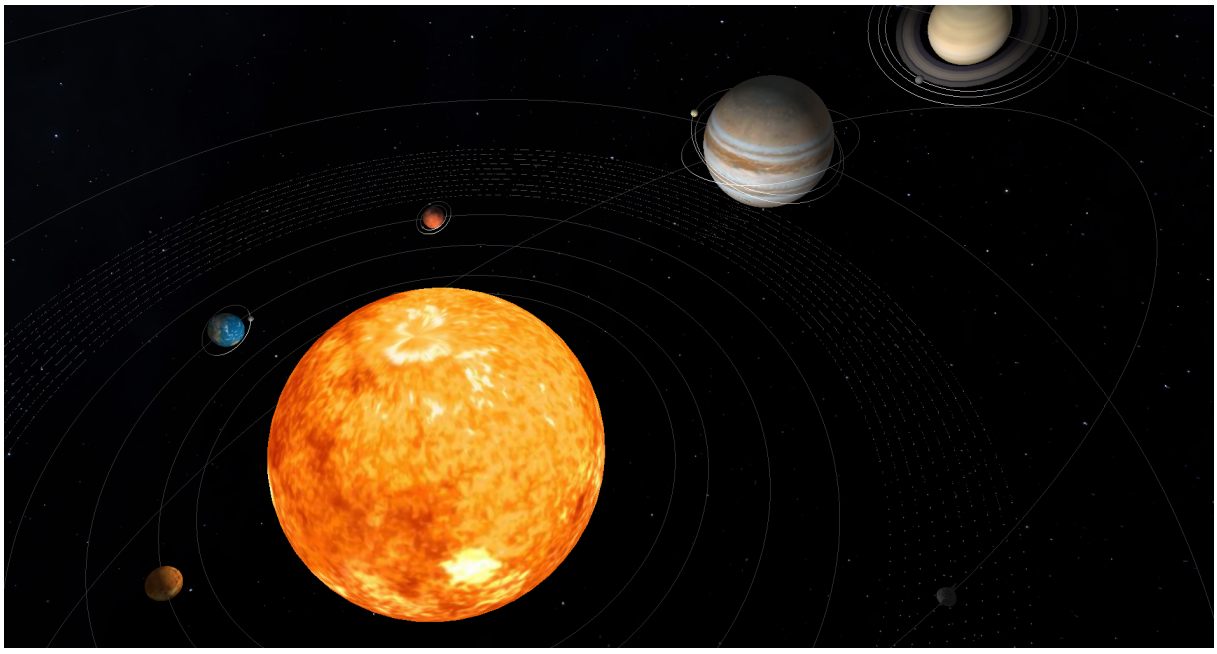

Figure 2.19: Foto do Sistema Solar

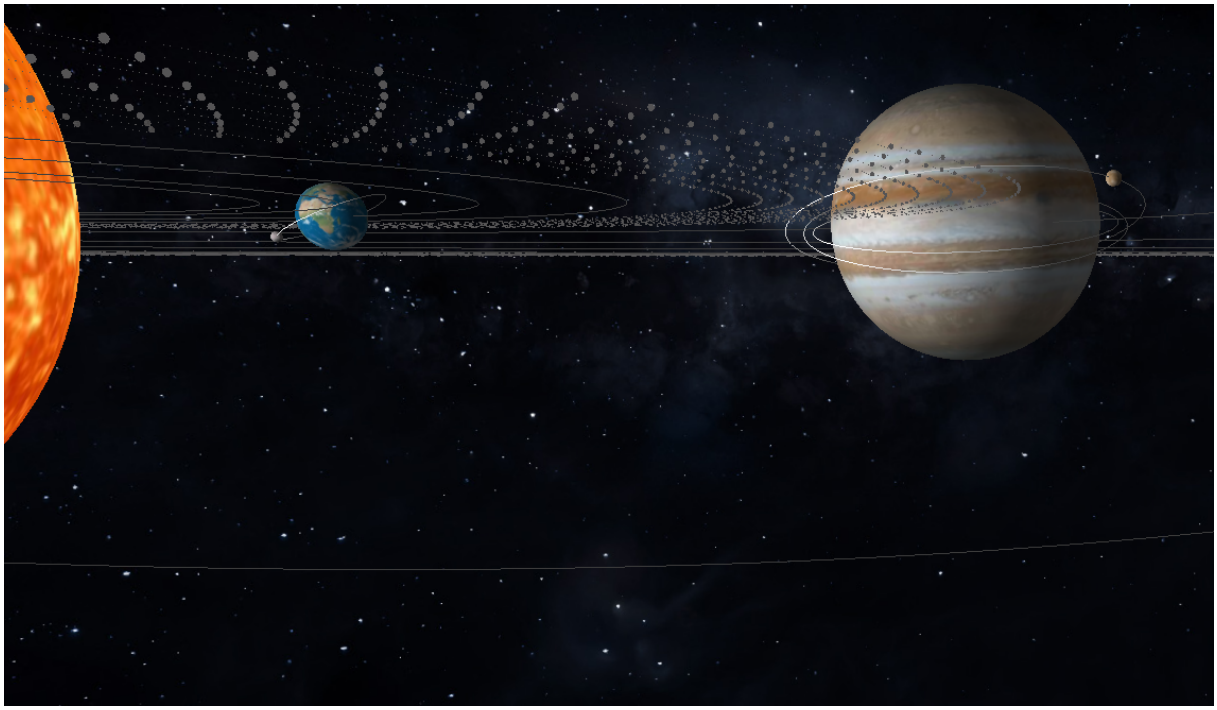

Figure 2.20: Foto do Sistema Solar

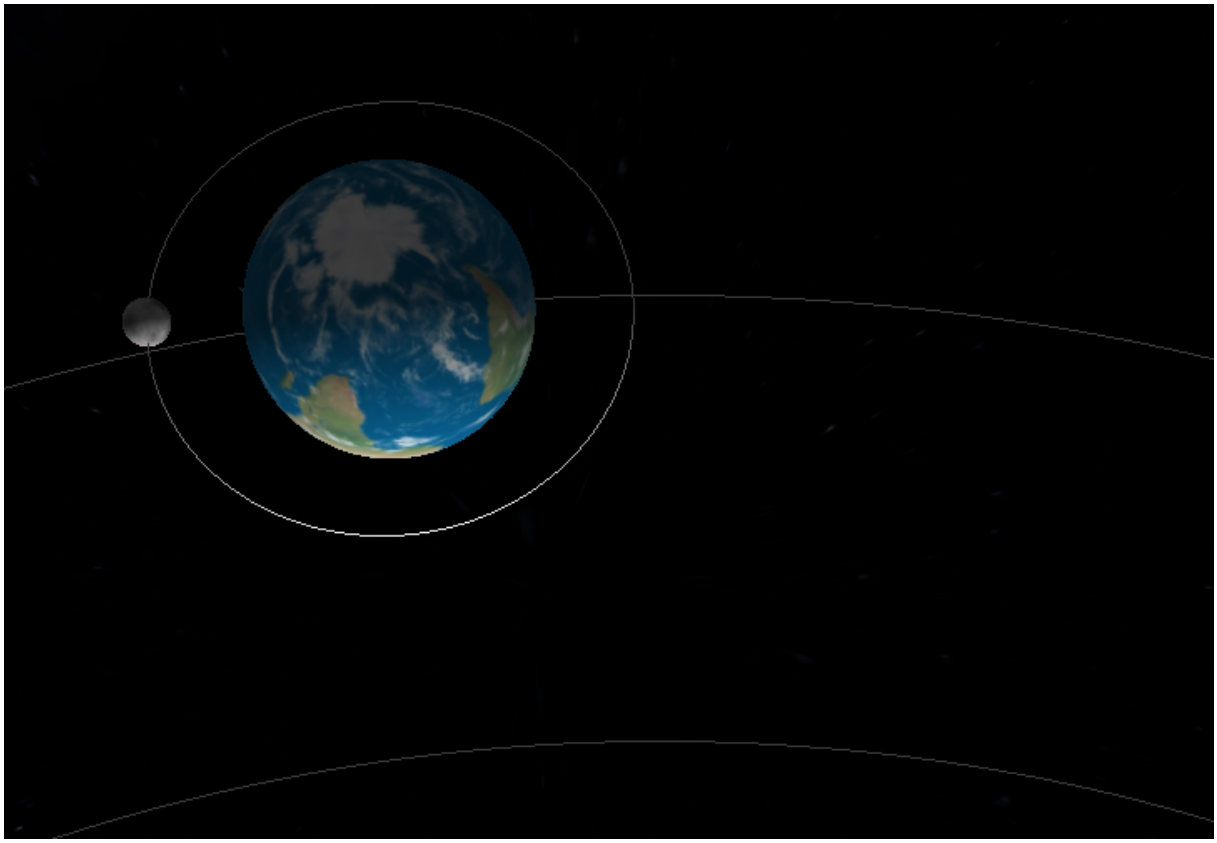

Figure 2.21: Foto do Sistema Solar

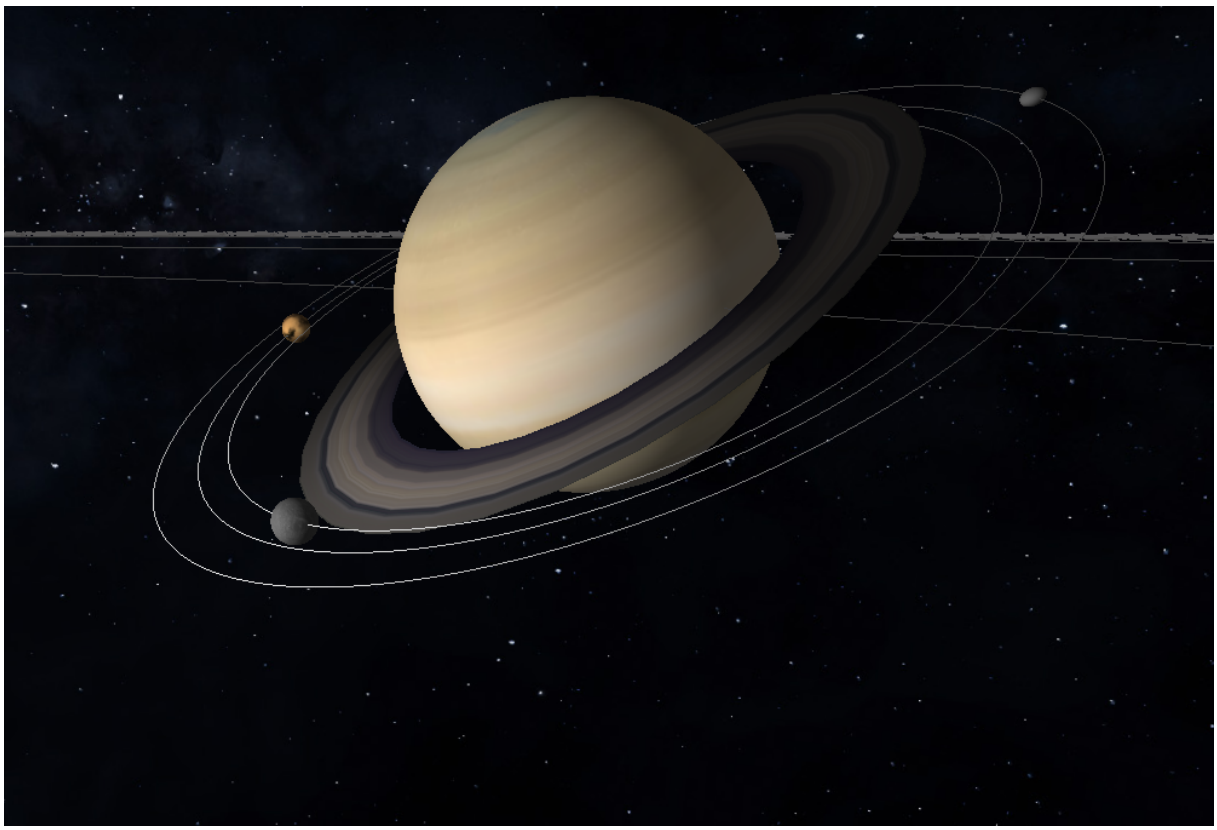

Figure 2.22: Foto do Sistema Solar

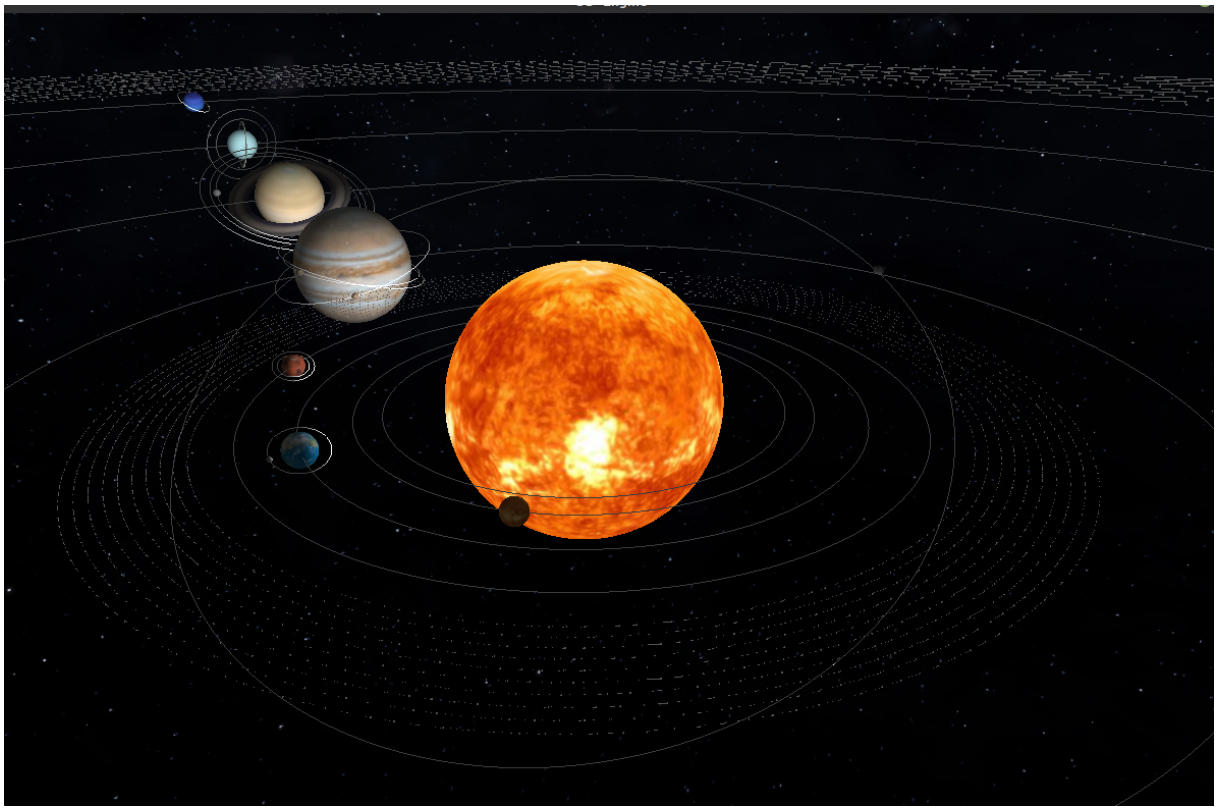

Figure 2.23: Foto do Sistema Solar

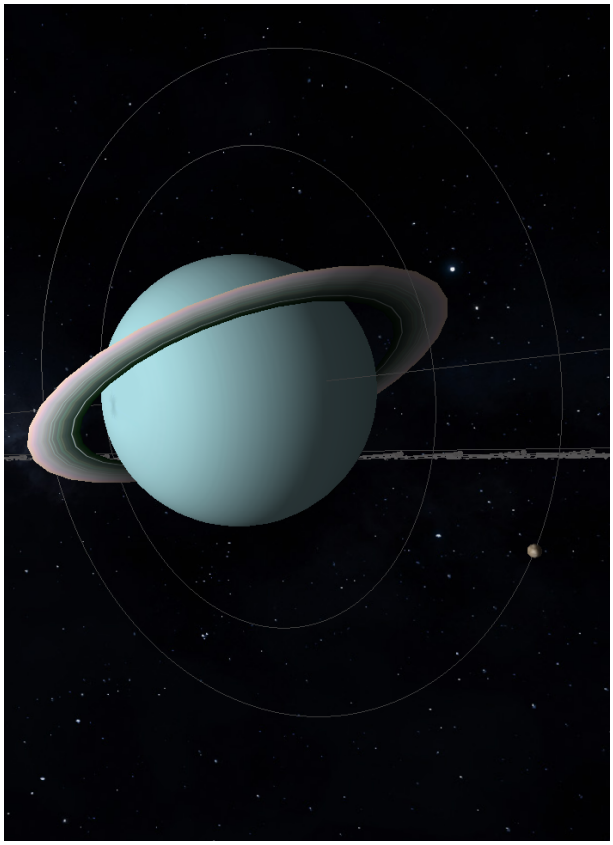

Figure 2.24: Foto do Sistema Solar

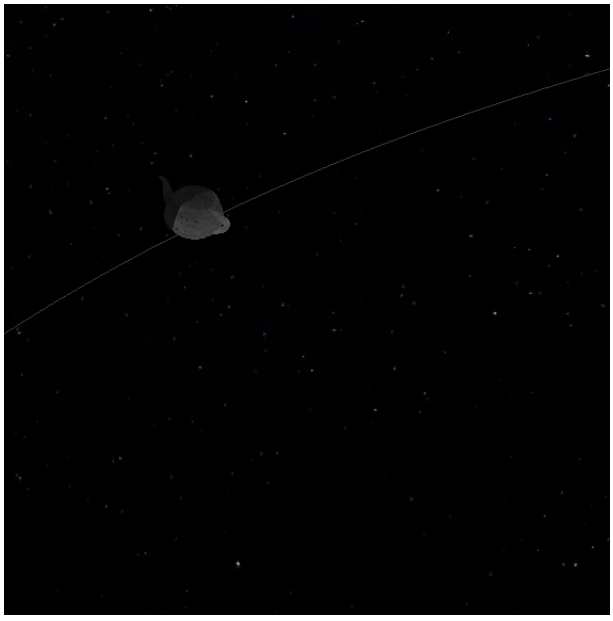

Figure 2.25: Cometa Teapot

## 2.6 Conclusão

Terminada esta quarta e última fase do projeto desenvolvido todos os membros do grupo sentem-se satisfeitos com o resultado obtido. Apesar das diversas dificuldades ao longo de todas as fases do projeto, achamos que de forma geral todos os objetivos foram cumpridos com sucesso, obtendo assim um Sistema Solar minimamente realista, tal como fora proposto.

Posto isto, consideramos que com o trabalho desenvolvido nas diferentes partes deste projeto, o nosso conhecimento face às ferramentas e utilidades do OpenGL e do GLUT cresceu significativamente, conseguindo por em prática os conhecimentos teóricos adquiridos nas aulas com o auxílio dos guiões desenvolvidos ao longo de todo o semestre.