Computação Gráfica (3º ano de LCC)

# Trabalho Prático (Fase 4) — Grupo 3

Relatório de Desenvolvimento

André Lucena Ribas Ferreira (A94956) Carlos Eduardo da Silva Machado (A96936)
Gonçalo Manuel Maia de Sousa (A97485)

4 de junho de 2023

**Resumo**

Este relatório aborda a solução proposta para o enunciado da 4ª fase do Trabalho Prático da Unidade Curricular "Computação Gráfica". Isto inclui a introdução de iluminação, texturas e cor aos modelos criados.

# Conteúdo

# Capítulo 1

# Introdução

O presente relatório tem como objetivo apresentar a solução concebida pelo Grupo 3 para a 4ª fase do Trabalho Prático da Unidade Curricular "Computação Gráfica".

Esta fase consistiu em implementar normais e coordenadas de textura para cada um dos modelos existentes e integrados nas fases anteriores. Também se atualizou a estrutura aceitável do ficheiro *xml* para poder corresponder tanto a iluminação e a texturas, podendo então ser possível usufruir das mesmas.

À demo desta parte, foi acrescentadas luzes e texturas aos planetas e cores às luas.

## 1.1 Estrutura do Relatório

Para além deste, o relatório compreende diferentes Capítulos. Em 2 apresenta-se extensão à implementação da aplicação *generator*. Em 3 apresenta-se as extensões à implementação da aplicação *engine*. Em 4 expõe-se imagens tiradas aos modelos gerados a partir dos *xmls* dos *test files*. Em 5 apresenta-se a conclusão do relatório.

# Capítulo 2

# *Generator*

Neste capítulo, abordamos as mudanças feitas no código relacionado com o *generator*, adicionando funcionalidades pedidas pelo enunciado e alterando o modo como fazíamos algumas operações como a escrita dos pontos em ficheiro. Podemos, então, dividir-los em três partes:

- IBO's
- Normais
- Coordenadas de Textura

## 2.1 IBO's

A implementação mais exigente de **IBO** junto de normais e de coordenadas de textura implica uma mais preocupada consideração pelos pontos escritos em ficheiro por parte do **generator**. Nomeadamente, tomamos em consideração todas as faces e vertentes possíveis de cada objeto, tal como a possibilidade de lhe aplicar texturas de modo a dar mais riqueza às cenas.

Para tal fim, cada um dos pontos foi considerado num espaço de 8 dimensões, com as coordenadas espaciais, as coordenadas do vetor normal e as coordenadas das texturas a ele aplicadas. Qualquer variação de um destes elementos implica a consideração de um novo ponto e, como tal, de um novo indíce adicionado a o **IBO**.

No ficheiro de modelo *.3d*, foram escritas as coordenadas de posição seguidas das coordenadas normais e das coordenadas de texturas. O número de vértices precede todos estes valores e informa-nos de todos eles.

## 2.2 Normais

As normais de cada modelo são definidas tendo em consideração cada uma das suas faces, entendendo se os seus vértices devem ser ou não considerados pertencentes a duas faces distintas, de modo a distinguir os elementos da 8ª dimensão, como descrito anteriormente.

#### Plane

Para o **plane** as normais são dadas facilmente pela orientação do plano, segundo a direção positiva do eixo *yy*, particularmente.

#### Box

As normais da **box** podem ser feitas de modo análogo ao plano, definindo o vetor normal ao plano de cada uma das faces do cubo.

#### Cone

As normais da base do **Cone** são trivialmente feitas, as restantes normais são dadas pela rotação do vetor (h, r), após ser normalizado pelo ângulo correto, em que h é a altura do cone e r o raio da base.

#### Sphere

Semelhante à maneira como foram calculados os pontos da esfera os vetors normais são calculados pela rotação de um *array* de vetores à volta do eixo y. Os vetores do *array* são por sua vez calculados por rotação do vetor (0,1,0).

#### Cylinder

As normais das bases do cilindro são feitas trivialmente, além disso, as normais do corpo do cilindro são feitas por rotação de um vetor paralelo ao eixo z.

#### Torus

As normais do cone são feitas de modo análogo à esfera. São gerados os vetores de um círculo e este é rodado à volta do eixo y.

## 2.3 Coordenadas de Textura

#### Plane

As coordenadas de textura do plano são diretamente retiradas das subdivisões dos eixos.

#### Box

As coordenadas de textura do cubo são feitas mapeando a textura em todas as faces do cubo, tratando cada uma como um plano próprio.

#### Cone

No caso do cone a textura é mapeada sobre a superfície. De modo que quanto mais próximo do topo do cone mais comprimida é a textura. É importante notar que os pontos do topo do cone bem como os pontos da "borda"são repetidos.

#### Sphere

As coordenadas de textura da esfera são feitas da mesma maneira do cone. Neste caso os pontos do topo e a base da esfera são repetidos.

# Capítulo 3

# *Engine*

Neste capítulo, vamos abordar as mudanças que fizemos ao código relativo ao *engine* de modo a suprir as novas necessidades enunciadas na fase 4 e alguns extras.

Deste modo, vamos dividir este capítulo em:

- Normais e Texturas
- Luzes e Cores
- Mipmapping

## 3.1 Normais e Texturas

A implementação do **IBO** do lado da *engine* baseia-se em conseguir ler corretamente do ficheiro as coordenadas corretas e de as associar corretamente ao seu lugar dentro de cada um dos **VBO's** definidos, tal como na fase passada se fez apenas com as coordenadas dos pontos. Desse modo, geraram-se dois novos **VBO's** e associaram-se-lhe as semânticas próprias de cada um. A leitura continua a ocorrer a partir do ficheiro de modelos, que verifica se o modelo já foi lido ou não, colocando apenas um de cada instância do modelo no **VBO**, repetindo apenas os índices.

```
unsigned int before = points->size();
points->resize(before + n);
normals->resize(before + n);
texCoords->resize(2*before/3 + 2*n/3);
filestream.read((char*)(points->data() + before), sizeof(float) * n);
filestream.read((char*)(normals->data() + before), sizeof(float) * n);
filestream.read((char*)(texCoords->data() + 2*before/3), sizeof(float) * 2*n/3);

unsigned int n_indices;
filestream.read((char*)&n_indices, sizeof(unsigned int));
//printf("%d\n", n_indices);
unsigned int* indices_buf = (unsigned int*)malloc(sizeof(unsigned int) * n_indices);
filestream.read((char*)(indices_buf), sizeof(unsigned int) * n_indices);
```

Uma característica importante das texturas está na expansão da classe **Model**, que necessita que se lhe associe o *char* correspondente à textura que foi carregada, quando encontrada no *xml*. Cada modelo saberá qual a sua textura e, no momento de desenho, esta é carregada e utilizada.

```
glBindTexture(GL_TEXTURE_2D, groupModel->texID);
glBindBuffer(GL_ARRAY_BUFFER, buffer[0]);
glVertexPointer(3, GL_FLOAT, 0, 0);
```

## 3.2 Luzes e Cores

Para guardar os três tipos de luzes criamos 3 tipos de classes diferentes que herdam da mesma classe *Lights*, essas classes são: *Point, Directional e Spotlight*. A classe *Point* e *Directional* têm o mesmo principio, o que apenas muda é o facto de o quarto argumento no primeiro ser 1 e no segundo ser 0, pois a coordenada w num ponto é 1 e num vetor é 0. A *Spotlight* vai conter um *array* que representa o ponto, um vetor que representa a direção e um *cutoff* que especifica o ângulo de propagação máximo da luz. Além disso, existe uma instância *GLuint number* que indica qual a luz que é, ou seja, se for a primeira terá o valor de *GL_LIGHT0*, isso permite que ao chamar a função *getNumber* nos dê qual a luz que o objeto indica. Além disso, temos o método *drawLight* que quando executado utiliza as funções do *opengl* indicadas para aquele tipo de luz.

```
void Spotlight::drawLight() {
    glLightfv(number, GL_POSITION, point);
    glLightfv(number, GL_SPOT_DIRECTION, dir);
    glLightfv(number, GL_SPOT_CUTOFF, &cutoff);
}

void Point::drawLight() {
    glLightfv(GL_LIGHT0, GL_POSITION, point);
}

void Directional::drawLight() {
    glLightfv(number, GL_POSITION, point);
}

GLuint Point::getNumber(){
    return Point::number;
}

GLuint Directional::getNumber(){
    return Directional::number;
}

GLuint Spotlight::getNumber(){
    return Spotlight::number;
}
```

Como podemos ver no código acima demonstrado, aplicamos *glLightfv* com *GL_POSITION* tanto para a classe *Point* quanto para a classe *Directional*, já a *Spotlight* temos de fazer *glLightfv* de acordo com as instâncias acima mencionas.

Deste modo, além do *glEnable(GL_LIGHTING* para ativar as luzes, e aplicar *GL_LIGHT_MODEL_AMBIENT* para a cor ambiente, aplicamos o seguinte ciclo uma vez antes do main loop, de modo a inicializar as luzes:

```
if (!lights->empty()) glEnable(GL_LIGHTING);
  for(Light* light: *lights){
    glEnable(light->getNumber());
    glLightfv(light->getNumber(), GL_AMBIENT, light->ambient);
    glLightfv(light->getNumber(), GL_DIFFUSE, light->diffuse);
    glLightfv(light->getNumber(), GL_SPECULAR, light->specular);
  }
```

Depois, na função *renderScene* fazemos:

```
for(Light* light: *lights){
    light->drawLight();
}
```

Para ajustar a luz, pois a mesma poderá ser afetada pelas transformações geométricas e precisamos de a ajustar.

O *parsing* do ficheiro *xml* para o caso das luzes ficará numa nova função:

Temos a opção no *xml* de adicionar com às componentes de cada luz e de colocar, ou não luz ambiente global.

```
void parse_lights(xml_node<> *lights_node, vector<Light*>* lights, float* amb, int*
    amb_active){
    xml_node<> *temp;
    xml_attribute<> *attr;
    GLuint numbers[] = {GL_LIGHT0, GL_LIGHT1, GL_LIGHT2, GL_LIGHT3, GL_LIGHT4, GL_LIGHT5,
    GL_LIGHT6, GL_LIGHT7};
    int number = 0;
    if((temp = lights_node->first_node("amb"))){
        if((attr = temp->first_attribute("active"))){
            if(!strcmp(attr->value(), "false")){
                amb_active = 0;
            }

            if((attr = temp->first_attribute("R"))){
                amb[0] = atof(attr->value())/255;
            }

            if((attr = temp->first_attribute("G"))){
                amb[1] = atof(attr->value())/255;
            }

            if((attr = temp->first_attribute("B"))){
                amb[2] = atof(attr->value())/255;
            }
        }
    }
    for(temp = lights_node->first_node("light"); number < 8 && temp; temp = temp->
    next_sibling("light"), number++){
        if((attr = temp->first_attribute("type")) && !strcmp(attr->value(), "point")){
            Point* point = new Point();
            if((attr = temp->first_attribute("posx")))
                point->point[0] = atof(attr->value());

            if((attr = temp->first_attribute("posy")))
                point->point[1] = atof(attr->value());

            if((attr = temp->first_attribute("posz")))
                point->point[2] = atof(attr->value());

            point->point[3] = 1.0;

            point->number = numbers[number];
            lights->push_back(point);
        } else if((attr = temp->first_attribute("type")) && !strcmp(attr->value(), "
    directional")){
            Directional* directional = new Directional();
            if((attr = temp->first_attribute("dirx")))
                directional->point[0] = atof(attr->value());

            if((attr = temp->first_attribute("diry")))
                directional->point[1] = atof(attr->value());

            if((attr = temp->first_attribute("dirz")))
                directional->point[2] = atof(attr->value());

            directional->point[3] = 0.0;
```

```
53
54          //printf("%f %f %f\n", directional->point[0], directional->point[1], directional
    ->point[2]);
55
56          directional->number = numbers[number];
57          lights->push_back(directional);
58      } else if((attr = temp->first_attribute("type")) && !strcmp(attr->value(), "
    spotlight")){
59          Spotlight* spotlight = new Spotlight();
60          if((attr = temp->first_attribute("posx")))
61              spotlight->point[0] = atof(attr->value());
62
63          if((attr = temp->first_attribute("posy")))
64              spotlight->point[1] = atof(attr->value());
65
66          if((attr = temp->first_attribute("posz")))
67              spotlight->point[2] = atof(attr->value());
68
69          spotlight->point[3] = 1.0;
70
71          if((attr = temp->first_attribute("dirx")))
72              spotlight->dir[0] = atof(attr->value());
73
74          if((attr = temp->first_attribute("diry")))
75              spotlight->dir[1] = atof(attr->value());
76
77          if((attr = temp->first_attribute("dirz")))
78              spotlight->dir[2] = atof(attr->value());
79
80          if((attr = temp->first_attribute("cutoff")))
81              spotlight->cutoff = atof(attr->value());
82          spotlight->number = numbers[number];
83          lights->push_back(spotlight);
84      }
85
86      if(temp->first_node("color")){
87          xml_node<> *tempColor;
88          xml_attribute<> *tempAttr;
89          if((tempColor = temp->first_node("diffuse"))){
90              if((tempAttr = tempColor->first_attribute("R"))){
91                  lights->at(number)->diffuse[0] = atof(tempAttr->value())/255;
92              }
93
94              if((tempAttr = tempColor->first_attribute("G"))){
95                  lights->at(number)->diffuse[1] = atof(tempAttr->value())/255;
96              }
97
98              if((tempAttr = tempColor->first_attribute("B"))){
99                  lights->at(number)->diffuse[2] = atof(tempAttr->value())/255;
100             }
101         }
102
103         if((tempColor = temp->first_node("ambient"))){
104             if((tempAttr = tempColor->first_attribute("R"))){
105                 lights->at(number)->ambient[0] = atof(tempAttr->value())/255;
106             }
107
108             if((tempAttr = tempColor->first_attribute("G"))){
109                 lights->at(number)->ambient[1] = atof(tempAttr->value())/255;
```

```
                }

                if((tempAttr = tempColor->first_attribute("B"))){
                    lights->at(number)->ambient[2] = atof(tempAttr->value())/255;
                }
            }

            if((tempColor = temp->first_node("specular"))){
                if((tempAttr = tempColor->first_attribute("R"))){
                    lights->at(number)->specular[0] = atof(tempAttr->value())/255;
                }

                if((tempAttr = tempColor->first_attribute("G"))){
                    lights->at(number)->specular[1] = atof(tempAttr->value())/255;
                }

                if((tempAttr = tempColor->first_attribute("B"))){
                    lights->at(number)->specular[2] = atof(tempAttr->value())/255;
                }
            }
        }
    }
}
```

É feito da mesma maneira que outros nodos do *xml*, com o detalhe que temos um *array* com os *GLuints* das luzes e são atribuídas às luzes pela ordem do *xml*.

Em relação aos modelos, fazemos o *parsing* da seguinte maneira:

```
if((temp = node_models->first_node("color"))){
            xml_node<> *tempColor;
            xml_attribute<> *tempAttr;
            if((tempColor = temp->first_node("diffuse"))){
                if((tempAttr = tempColor->first_attribute("R"))){
                    model->diffuse[0] = atof(tempAttr->value())/255;
                }

                if((tempAttr = tempColor->first_attribute("G"))){
                    model->diffuse[1] = atof(tempAttr->value())/255;
                }

                if((tempAttr = tempColor->first_attribute("B"))){
                    model->diffuse[2] = atof(tempAttr->value())/255;
                }
            }

            if((tempColor = temp->first_node("ambient"))){
                if((tempAttr = tempColor->first_attribute("R"))){
                    model->ambient[0] = atof(tempAttr->value())/255;
                }

                if((tempAttr = tempColor->first_attribute("G"))){
                    model->ambient[1] = atof(tempAttr->value())/255;
                }

                if((tempAttr = tempColor->first_attribute("B"))){
                    model->ambient[2] = atof(tempAttr->value())/255;
                }
            }

```

```
            if((tempColor = temp->first_node("specular"))){
                if((tempAttr = tempColor->first_attribute("R"))){
                    model->specular[0] = atof(tempAttr->value())/255;
                }

                if((tempAttr = tempColor->first_attribute("G"))){
                    model->specular[1] = atof(tempAttr->value())/255;
                }

                if((tempAttr = tempColor->first_attribute("B"))){
                    model->specular[2] = atof(tempAttr->value())/255;
                }
            }

            if((tempColor = temp->first_node("emissive"))){
                if((tempAttr = tempColor->first_attribute("R"))){
                    model->emissive[0] = atof(tempAttr->value())/255;
                }

                if((tempAttr = tempColor->first_attribute("G"))){
                    model->emissive[1] = atof(tempAttr->value())/255;
                }

                if((tempAttr = tempColor->first_attribute("B"))){
                    model->emissive[2] = atof(tempAttr->value())/255;
                }
            }

            if((tempColor = temp->first_node("shininess"))){
                if((tempAttr = tempColor->first_attribute("value"))){
                    model->shininess = atof(tempAttr->value());
                }
            }
        }
```

Dividimos cada componente por 255 de modo a garantir que os valores fiquem entre 0 e 1. Caso não haja o nodo *Color* no *xml*, ou alguma das componentes não for indicada tomamos com padrão os valores dados pelo unciado. A classe *Model* terá no seu interior as seguintes instâncias:

```
    float diffuse[4] = {200, 200, 200, 1.0};
    float ambient[4] = {50, 50, 50, 1.0};
    float specular[4] = {0, 0, 0, 1.0};
    float emissive[4] = {0, 0, 0, 1.0};
    GLfloat shininess = 0;
```

Para cada modelo, antes de efetivamente desenha-lo, indicamos as luzes do objeto com as funções *glMaterialfv* e *glMaterialf*:

```
    glMaterialfv(GL_FRONT, GL_SPECULAR, groupModel->specular);
  glMaterialfv(GL_FRONT, GL_AMBIENT, groupModel->ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, groupModel->diffuse);
    glMaterialfv(GL_FRONT, GL_EMISSION, groupModel->emissive);
    glMaterialf(GL_FRONT, GL_SHININESS, groupModel->shininess);
```

## 3.3 Mipmapping

Adicionamos *Mipmapping* com a *flag* $GL\_LINEAR\_MIPMAP\_LINEAR$ o que faz com que algumas imagens do exemplo fiquem ligeiramente diferentes por causa da interpolação.

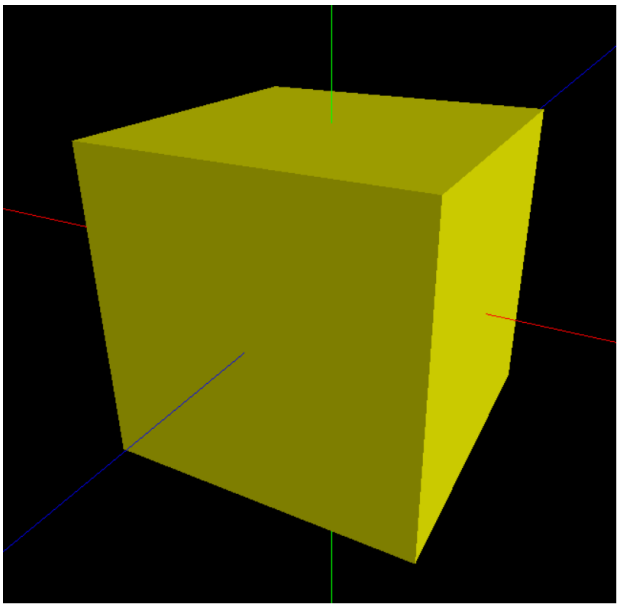
# Capítulo 4

# Resultados

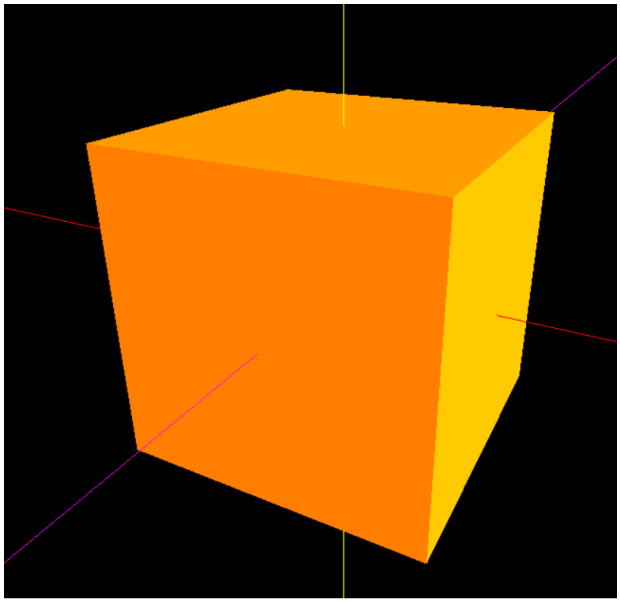
Neste capítulo apresentamos os resultados obtidos da execução de ambas as aplicações utilizando os ficheiros de teste fornecidos.

## 4.1 Demo

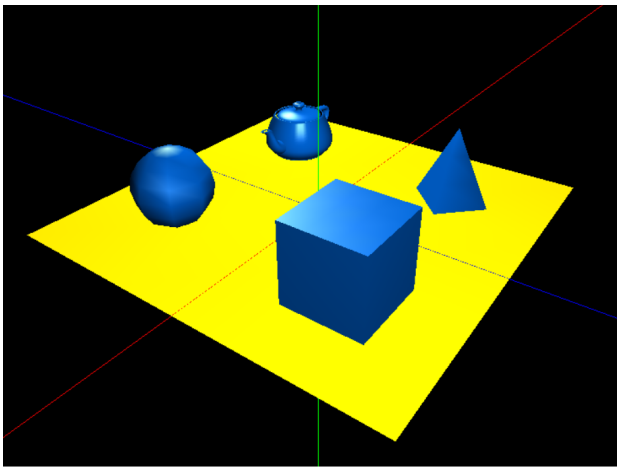
A Demo foi estendida acrescentando texturas a cada um dos planetas principais e à lua da Terra, tal como foram acrescentadas luzes e cor emissiva ao Sol para emular a iluminação e as sombras emitidas. Também foram dadas cores a cada um dos restantes corpos celestes, incluindo do cometa. Foi também corrigida a trajetória da elipse do cometa para ser mais circular.



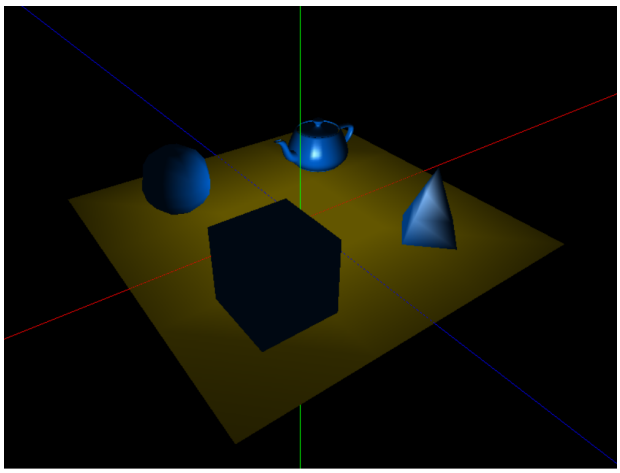
(a) Teste 1



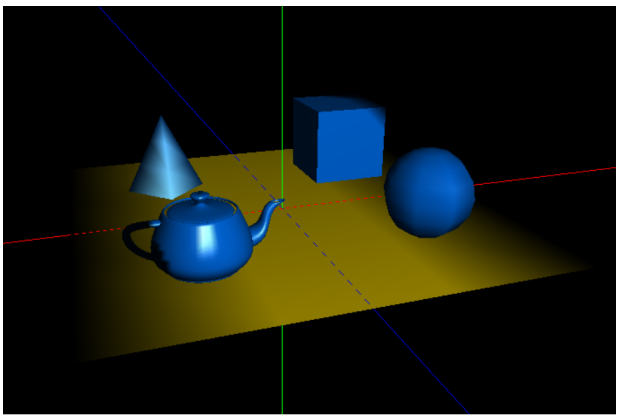
(b) Teste 2



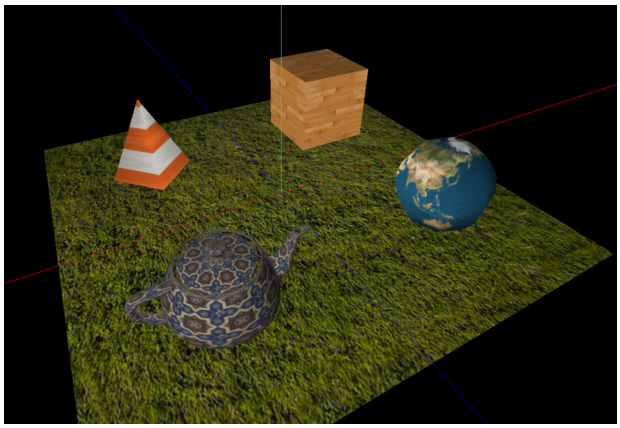
(c) Teste 3



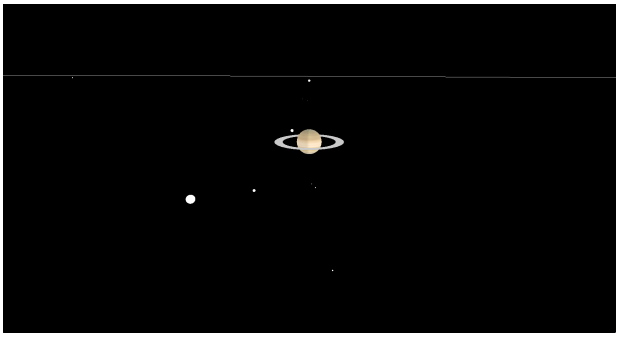
(d) Teste 4



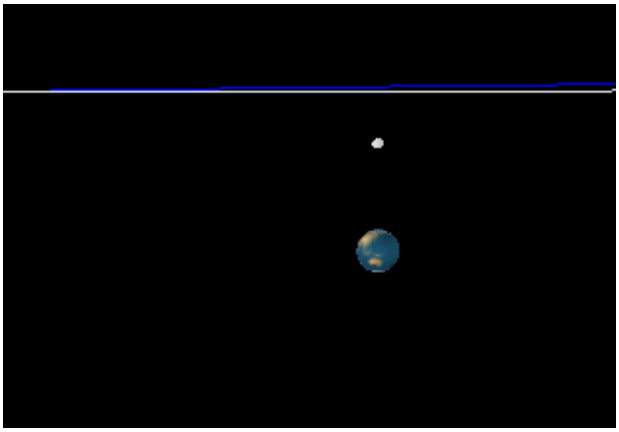
(a) Teste 5



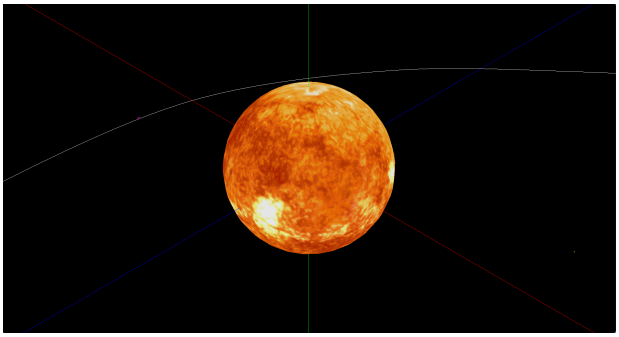
(b) Teste 6



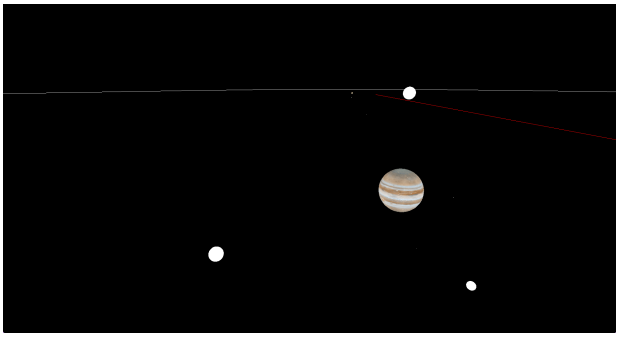
(a) Saturno



(b) Terra



(a) Sol



(b) Júpiter

# Capítulo 5

# Conclusão

Em suma, ao longo deste relatório foi implementada . Também foi implementada . O desenho das primitivas foi potenciado pela utilização de . Por fim, a demo foi aumentada para incluir luzes e texturas.