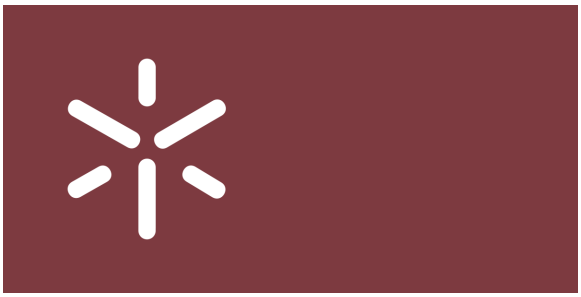

Universidade do Minho

Fase 4 - Normals and Textures Coordinates

Trabalho Prático de Computação Gráfica

Mestrado Integrado em Engenharia Informática

João Amorim A74806

02 de Maio de 2021

1 Introdução

No âmbito da Unidade Curricular de Computação Gráfica, foi proposto que se desenvolvesse um cenário baseado em gráficos 3D. Este projeto consiste em 4 fases, sendo esta a última fase.

Nesta fase, será necessário efetuar mudanças no Generator e no Engine, de maneira a que os objetivos sejam cumpridos. No Generator, deverão ser geradas as normais e coordenadas das texturas, enquanto que no Engine, deverão ser criadas as funcionalidades que activem a iluminação e as texturas das cenas apresentadas.

Este relatório tem como objetivo descrever toda a aplicação, ajudando a executar a mesma, contendo obviamente uma descrição de todo o trabalho desenvolvido. No final, serão feitas conclusões sobre todo o trabalho.

2 Arquitetura da Aplicação

2.1 Generator

Nesta última fase, voltam a ser necessárias algumas modificações ao Generator. Desta vez, é necessário guardar, nos ficheiros .3d criados, as normais e os pontos das texturas para cada primitiva gerada, de maneira a que a Iluminação e as Texturas possam depois ser implementadasno Engine.

2.2 Engine

No Engine serão então implementadas as funcionalidades da Iluminação e da atribuição das Texturas. Para isto, serão criadas novas classes, novas funções e a leitura dos ficheiros XML será feita de forma diferente.

3 Generator

3.1 Normais e Texturas

Tal como já foi mencionado, o objetivo desta fase é implementar as funcionalidades para geração das coordenadas para as texturas, assim como as normais. De seguida, serão demonstrados os passos utilizados para gerar os mesmos.

3.1.1 Plane

Para obter o conjunto dos vetores das normais, bata apenas verificar em que plano cartesiano é desenhado o mesmo. Uma vez que apenas são construídos planos no eixo xOz, definiu-se como normal o vetor (0,1,0).

3.1.2 Box

Para obter as normais da Box, basta fazer o mesmo que foi feito no plano, mas agora para cada face:

- Face Frontal: (0,0,1)
- Face Esquerda: (-1,0,0)
- Face Direita: (1,0,0)
- Face de Trás: (0,0,-1)

- Face de Cima: (0,0,1)
- Face de Baixo: (0,0,-1)

Para a definição das texturas da Box, utilizemos a seguinte imagem como exemplo:

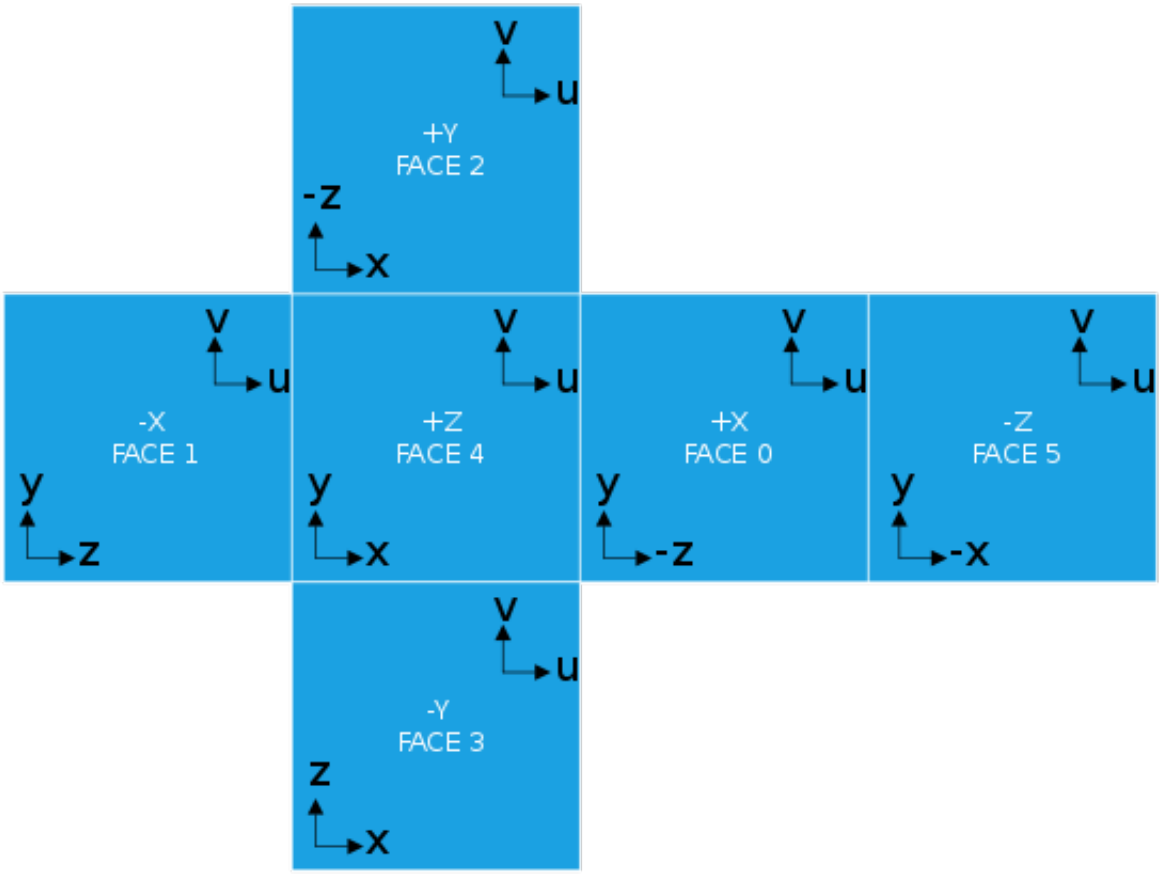

Figure 1: Exemplo de mapeamento 2D para a Box

Sendo o valor máximo de coordenadas (1,1), dividimos o eixo do Y em três partes, o eixo X em quatro partes e é efetuado o mapeamento dos pontos das texturas de acordo com a face que estamos a desenhar.

3.1.3 Sphere

Antes de serem explicadas as normais e texturas da sphere, é importante notar que o desenho dos vários triângulos que constituem a sphere teve de ser mudado devido a alguns erros, erros dos quais me apercebi após efetuar o primeiro mapeamento das texturas.

Para o cálculo das normais, era necessário determinar o ângulo entre a normal e um determinado ponto da superfície. Com o cáluclo dos diversos vértices da esfera, a obtenção das normais foi bastante trivial, uma vez que bastou extrair o já mencionado ãngulo do cálculo de cada vértice.

```
p1x_S = radius * sin(beta) * sin(alpha);
p1y_S = radius * cos(beta);
p1z_S = radius * sin(beta) * cos(alpha);
```

Figure 2: Pontos das normais da Sphere

Em relação às texturas, o processo é feito com o cálculo de quanto em quanto é que uma textura tem de ser aplicada. Este intervalo é obtido ao dividir 1 pelo número de slices da esfera, para X, e 1 pelo número de stacks da esfera, para Y, permitindo assim obter as coordenadas para a aplicação.

```
// Textures
float s;
float deltaS = 1.0f / slices;
float t;
float deltaT = 1.0f / stacks;
```

Figure 3: Intervalos para aplicação das texturas

3.1.4 Cone

Para a geração das normais do Cone, vamos começar pela base. Seguindo a linha de pensamento das outras primitivas e estando esta base no plano xOz, a normal será (0,-1,0). No caso da geração das normais das laterais, à medida que cada ponto é obtido, é inserida a respetiva normal. Esta normal é obtida de acordo com a fórmula: (sin(alpha), a/cH, cos(alpha)) "a" refere-se ao comprimento do corpo e cH refere-se às camadas horizontais.O valor de Y é dado por a/cH, enquanto que X e Z são determinados pelo sin e

cos, uma vez que estes se movem no sentido da circunferência. O alpha é a amplitude na qual o vértice se encontra.

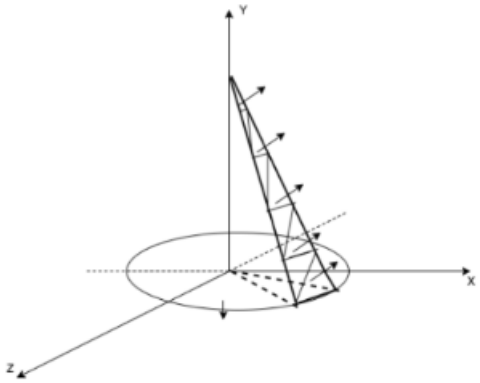

Figure 4: Vetores normais para uma slice de um cone

3.1.5 Cylinder

3.1.6 Torus

4 Engine

4.1 Leitura do XML

Nesta fase, o ficheiro XML volta a ser alterado. Antes de serem explicadas as alterações feitas, é necessário ter em conta que nesta fase passou a ser utilizado o tinyxml em vez do tinyxml2. Havia problemas na leitura de alguns atributos que com o tinyxml eram resolvidos.

Em primeiro lugar, é necessário criar um mecanismo de parsing para a Iluminação. Para isto, são criadas duas novas funções, nomeadamente parseLights() e xmlReadLight(). A primeira lê cada elemento "light" dentro de "lights", enquanto que a segunda percorre todos os atributos da luz. Neste caso, estes atributos correspondem aos valores de x, y e z que constituem aquilo que será o ponto passado como GL_POSITION, na altura da implementação da Iluminação no Engine.

```
void parseLights(TiXmlElement* lights){

    for (TiXmlElement* light = lights->FirstChildElement(); light; light = light->NextSiblingElement())
    {
        std::string lightName(toLower(light->Value()));

        if (lightName == "light") {

            xmlParseLight(light);
        }
        else {
            cout << "problema no xml com as luzes" << endl;
        }
    }
}
```

Figure 5: Função parseLights

```
void xmlParseLight(TiXmlElement* light){

    for (TiXmlAttribute* a = light->FirstAttribute(); a; a = a->Next()) {

        if (!strcmp(a->Name(), "posX")) {
            pos.setX(readFloat(a->Value()));
        }
        else if (!strcmp(a->Name(), "posY")) {
            pos.setY(readFloat(a->Value()));
        }
        else if (!strcmp(a->Name(), "posZ")) {
            pos.setZ(readFloat(a->Value()));
        }

        else if (!strcmp(a->Name(), "type")) {
            if (!strcmp(a->Value(), "POINT")) {
                type = "POINT";
            }
            else if (!strcmp(a->Value(), "DIRECTIONAL")) {
                type = "DIRECTIONAL";
            }
            else if (!strcmp(a->Value(), "SPOT")) {
                type = "SPOT";
            }
        }
    }
}
```

Figure 6: Função xmlParseLights

Em segundo lugar, é também necessário fazer a leitura do nome do ficheiro que contém as texturas para determinado objeto. Este nome estará dentro do elemento model, onde se encontra também o nome do ficheiro .3d, tal como nas últimas fases. Ao ser lido o elemento model, o valor do atributo file continua a ser passado para como argumento para a função que faz a leitura do ficheiro .3d e o nome do ficheiro com as textura é guardado num vetor chamado **texFiles**, que será depois utilizado pelo engine.

```
<models>
    <model file="sphere.3d" textura="texture_mercury.jpg"/>
</models>
```

Figure 7: XML que contém o nome do ficheiro com as texturas

A leitura do nome do ficheiro com as texturas, é feito dentro na função usada nas últimas fases

4.2 VBOs

Em termos de VBOs, foram criados três novos tipos. O primeiro, buffersNorms, que irá conter as coordenadas dos pontos para as normais, e os outros buffersTextsCoords buffersTexID que são utilizados para o desenho das texturas. Toda a inicialização destes VBOs continua a ser feita dentro da função **prepareData()**, sendo o VBO que guarda as texturas geradas inicializado na função **loadTextures**

```
void loadTextures() {

    buffersTexID = (GLuint*)malloc((sizeof(GLuint)) * nVBOTex);

    for (int i = 0; i < nVBOTex; ++i) {

        buffersTexID[i] = loadTex(texFiles[i]);
    }
}
```

Figure 8: Função loadTextures

```
void prepareData() {
    //vertices
    buffersVerts = (GLuint*)malloc((sizeof(GLuint)) * nVBOVert);
    //normais
    buffersNorms = (GLuint*)malloc((sizeof(GLuint)) * nVBONorm);
    //textures
    buffersTextsCoords = (GLuint*)malloc((sizeof(GLuint)) * nVBOTex);

    //vbos dos vertices
    glGenBuffers(nVBOVert, buffersVerts);
    for (int i = 0; i < nVBOVert; ++i) {
        glBindBuffer(GL_ARRAY_BUFFER, buffersVerts[i]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(float) *
            vertVBOs[i].size(), vertVBOs[i].data(), GL_STATIC_DRAW);
    }
    //vbos das normais
    glGenBuffers(nVBONorm, buffersNorms);
    for (int i = 0; i < nVBONorm; ++i) {
        glBindBuffer(GL_ARRAY_BUFFER, buffersNorms[i]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(float) * normVBOs[i].size(),
            normVBOs[i].data(), GL_STATIC_DRAW);


    }
    //vbos das texturas
    glGenBuffers(nVBOTex, buffersTextsCoords);
    for (int i = 0; i < nVBOTex; ++i) {

        glBindBuffer(GL_ARRAY_BUFFER, buffersTextsCoords[i]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(float) * texVBOs[i].size(),
            texVBOs[i].data(), GL_STATIC_DRAW);
    }
}
```

Figure 9: Função prepareData()

4.3 Iluminação

Para a iluminação, era pedido que fosse criada iluminação para o Sistema Solar, sendo esta iluminação de três tipos, Point, Directional e Spotlight.

Seguindo os princípios que foram seguidos até agora, foi criada uma nova classe Light que é responsável pela criação das operações de iluminação.

De acordo com os tipos de iluminação que era necessário definir, a classe Light foi construída da seguinte maneira:

POINT:

```
if(enable == true && sun == true) {
    glEnable(GL_LIGHT0);
    glEnable(GL_LIGHTING);
    float pos[4] = { x,y,z,1};
    glLightfv(GL_LIGHT0, GL_POSITION, pos);

    GLfloat amb[4] = { 0.0, 0.0, 0.0, 1.0 };
    GLfloat diff[4] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat spec[4] = { 0, 0, 0, 1 };

    glLightfv(GL_LIGHT0, GL_AMBIENT, amb);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diff);
    glLightfv(GL_LIGHT0, GL_SPECULAR, spec);

    GLfloat matt[3] = { 5, 5, 5 };
    glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, matt);
}
else if (enable == true && sun == false) {
    glEnable(GL_LIGHT0);
    glEnable(GL_LIGHTING);

    GLfloat amb[4] = { 1, 0.8, 0.8, 1.0 };
    glLightfv(GL_LIGHT0, GL_AMBIENT, amb);

    GLfloat matt[3] = { 0.1, 0.1, 0.1 };
    glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, matt);
}
```

DIRECTIONAL:

```
else if (type == "DIRECTIONAL") {
```

```
        if (enable == true) {
            glEnable(GL_LIGHT0);
            glEnable(GL_LIGHTING);

            float pos[4] = { x,y,z,0 };
            glLightfv(GL_LIGHT0, GL_POSITION, pos);

            GLfloat amb[4] = { 0.0, 0.0, 0.0, 1.0 };
            GLfloat diff[4] = { 1.0, 1.0, 1.0, 1.0 };
            GLfloat spec[4] = { 0, 0, 0, 1 };

            glLightfv(GL_LIGHT0, GL_AMBIENT, amb);
            glLightfv(GL_LIGHT0, GL_DIFFUSE, diff);
            glLightfv(GL_LIGHT0, GL_SPECULAR, spec);
        }
        else {
            glDisable(GL_LIGHTING);
        }
}
```

SPOT:

```
    if (enable == true){
            glEnable(GL_LIGHT0);
            glEnable(GL_LIGHTING);

            float pos[4] = { x,y,z,1 };
            glLightfv(GL_LIGHT0, GL_POSITION, pos);

            GLfloat amb[4] = { 0.0, 0.0, 0.0, 1.0 };
            GLfloat diff[4] = { 1.0, 1.0, 1.0, 1.0 };
            GLfloat spec[4] = { 0, 0, 0, 1 };

            glLightfv(GL_LIGHT0, GL_AMBIENT, amb);
            glLightfv(GL_LIGHT0, GL_DIFFUSE, diff);
            glLightfv(GL_LIGHT0, GL_SPECULAR, spec);

            GLfloat spotDir[3] = { 0.0, 0.0, -1.0 };

            glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spotDir);
            glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);
```

```
        glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 0.0);

    }else{
        glDisable(GL_LIGHTING);
    }
```

No array pos, o valor de pos[3] indica se o ponto (x,y,z) corresponde a um ponto ou a um vetor. Para as luzes do tipo Point e Spotlight, este valor será 1, enquanto que para a luz Directional será 0. O ponto que define a GL_POSITION, corresponde então ao (x,y,z) que são lidos do ficheiro xml.

Em relação à implementação da luz como operação no vetor de Operações, esta é introduzida imediatamente antes e depois de uma operação de desenho ser introduzida, com a particularidade de que se a operaçao de desenho for para o Sol, é atribuído o GL_EMISSION para lhe dar luz própria no caso da luz POINT. A Operação que se encontra antes contém todas as caracteristicas da luz enquanto que a segunda faz apenas o glDisable(GL_LIGHTING). Infelizmente, a luz DIRECTIONAL e SPOTLIGHT não estão funcionais. Existe algum erro que penso ser relacionado com o Enable e Disable da iluminação, mas devido a falta de tempo, não o vou conseguir corrigir a tempo.

4.4 Cor e Texturas

Em primeiro lugar, antes de se explicar a implementação das texturas, é necessário falar da Cor, tanto da que foi implementada nas primeiras fases, assim como aquela que é pedida nesta fase.

Em relação à cor que foi implementada nas primeiras fases, esta foi removida por completo dos ficheiros xml e do engine, uma vez que tinha implicações diretas com as texturas.

Em relação à Cor que era pedida para implementar nesta fase, nomeadamente no que diz respeito aos vários componentes da cor, Difusa, Especular, Ambiente e Emissiva, esta foi uma funcionalidade que não foi implementada. Cada componente da cor foi definida, também na classe Light, mas devido a erros na leitura do ficheiro xml e dos ficheiros .3d, aliado à falta de tempo para os corrigir, acabei por ter que, mais uma vez, aceitar derrota. A ideia inicial era aceitar um outro tipo de luz, chamada "COLOR", que em vez de atribuir iluminação aos objetos desenhados, atribuía a respetiva componente de cor. Isto era apenas uma fase de experimentação, que seria posteriormente dividida em cada componente da cor e lida corretamente do ficheiro xml, a partir do modelo, mas uma vez que não estava a funcionar, acabou por ficar apenas definida na classe Light.

Mesmo assim, deixo aqui a definição para cada componente da Cor:

```
else if (type == "COLOR") {
        GLfloat diff[4] = { 1.0, 1.0, 1.0, 1.0 };
        GLfloat spec[4] = { 0.0, 0.0, 0.0, 1.0 };
        GLfloat emis[4] = { 0.0, 0.0, 0.0, 1.0 };
        GLfloat amb[4] = { 1.0, 1.0, 1.0, 1.0 };

        glMaterialfv(GL_FRONT, GL_DIFFUSE, diff);
        glMaterialfv(GL_FRONT, GL_SPECULAR, spec);
        glMaterialfv(GL_FRONT, GL_EMISSION, emis);
        glMaterialfv(GL_FRONT, GL_AMBIENT, amb);
}
```

Em relação às texturas, é feito uso dos dois VBOs já explicados, buffersTextsCoords e buffersTexID. As funções de inicialização preenchem os VBOs com a informação necessária, que serão depois utilizados nas Operações de desenho na classe Draw.

4.5 Classes

Em relação às Classes, todas se mantiveram como na última fase, com a excepção das classes Light e Point2d, que foram criadas especificamente para esta fase:

```
class Point2d {
    float u;
    float v;

public:
    Point2d();
    Point2d(float a, float b);
    float getU() { return this->u; }
    float getV() { return this->v; }
    void setU( float b) { this->u = b; }
    void setV( float b) { this->v = b; }
};
```

Figure 10: Point2d.h

```
class Light : public Operation {

    float x;
    float y;
    float z;
    bool enable;
    bool sun;
    string type;


public:
    Light(float x, float y, float z, bool enable, bool sun, string type) {
        this->x = x;
        this->y = y;
        this->z = z;
        this->enable = enable;
        this->sun = sun;
        this->type = type;
    };

    void run();
};
```

Figure 11: Light.h

5 Resultado e Balanço Final

Para demonstração de resultados, foram criadas duas Scenes principais e várias Scenes secundárias. As principais contêm a versão final do Sistema Solar, cujo nome é "solarsystem.xml" e uma outra versão do Sistema Solar, mas com as outras primitivas gráficas que não a Sphere, a representarem os diferentes corpos. Isto permite ver que de facto todas as normais e texturas foram calculadas para cada primitiva.

As várias Scenes secundárias são apenas as primitivas gráficas que forma sendo criadas ao longo do projeto e com uma textura aplicada.

Em relação a alguns aspetos importantes que devem ser mencionados, em primeiro lugar e tal como foi explicado na primeira fase, não existe um executável para o Engine, devido a um erro com includes que não consegui resolver. Desta maneira, o nome do ficheiro XML a ser lido é passado na função main() do Engine, que quando executado, abre a janela com o trabalho desenvolvido. Penso também que apesar de nenhuma fase estar "um espetáculo", os vários resultados que foram obtidos ao longo do semestre, em cada fase, foram positivos principalmente estando a fazer o trabalho sozinho.

Seguem-se de seguida as imagens dos dois Sistemas Solares obtidos, do Cylinder e da Box com texturas:

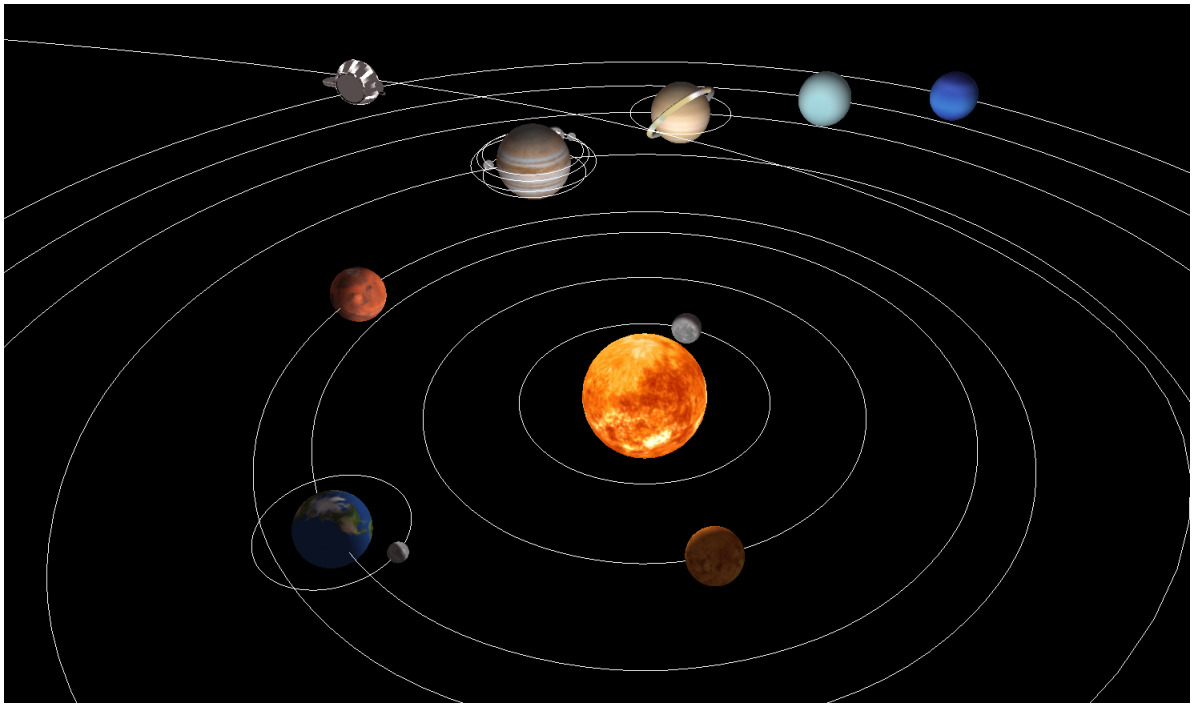

Figure 12: Sistema Solar Final.h

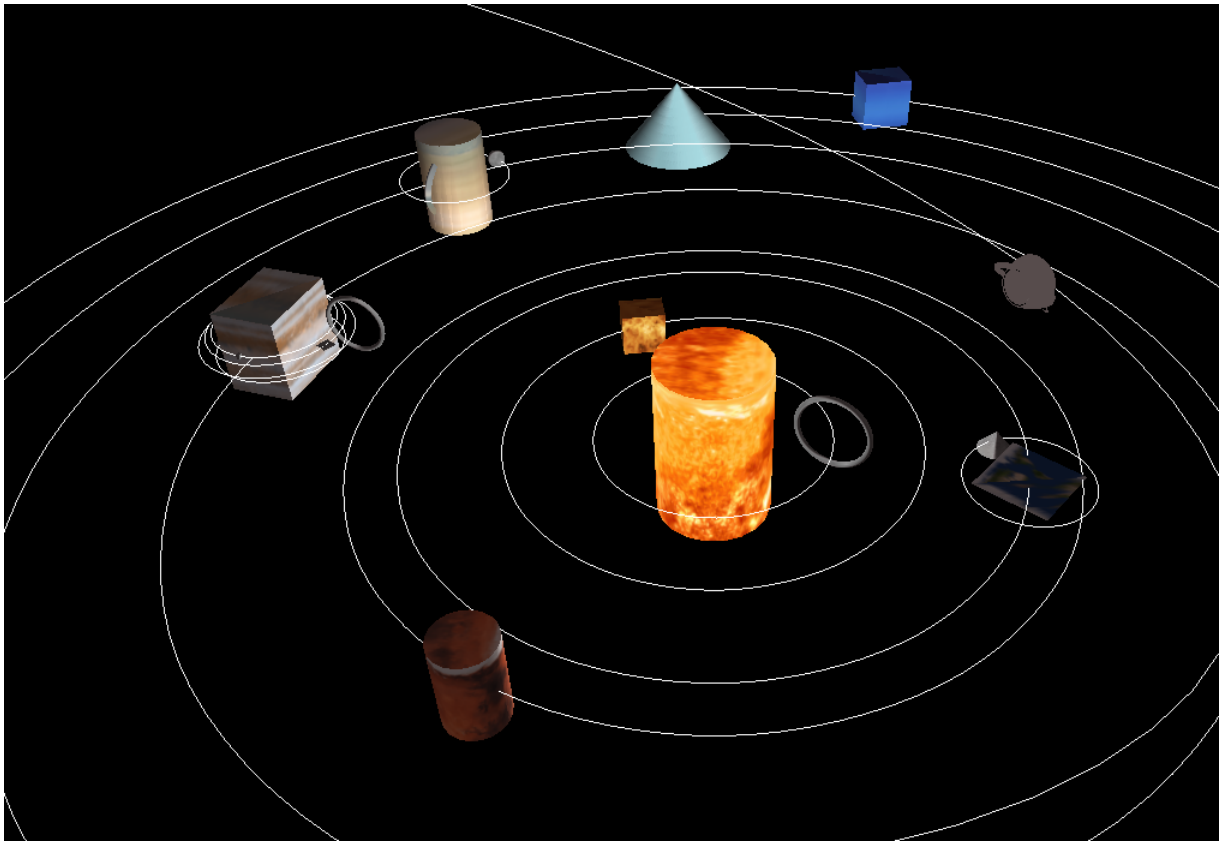

Figure 13: Sistema Solar com Primitivas como corpos

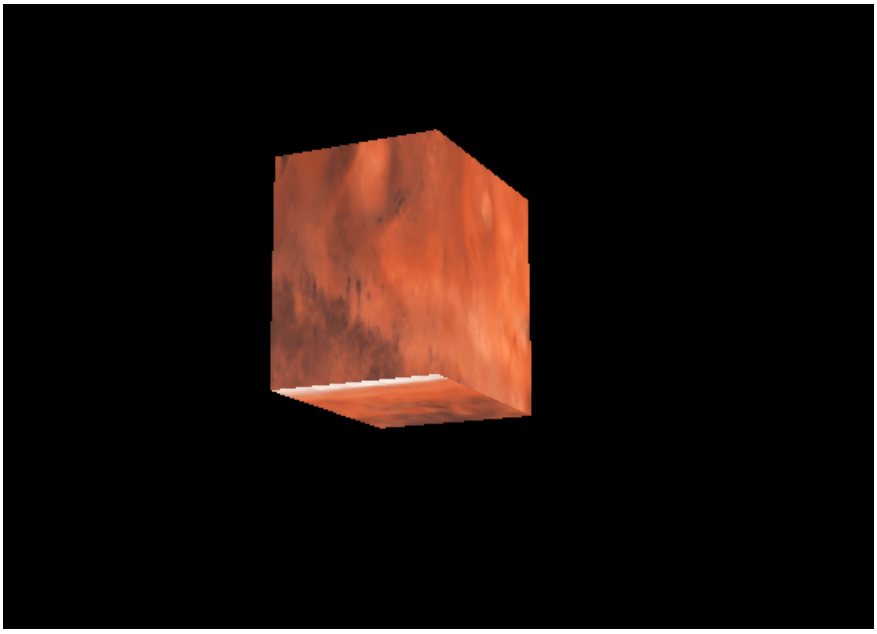

Figure 14: Box final com texturas

6 Conclusão

Sendo o trabalho concluído nesta fase, posso dizer que o balanço final é bastante positivo. Nesta fase, algumas das funcionalidades não foram implementadas da melhor maneira, apesar do facto de que a teoria estava presente, faltando apenas alguma ginástica programacional.

Foi sem dúvida um trabalho interessante, por vezes frustrante, mas que me proporcionou certamente um alargamento de conhecimento e de raciocinio.

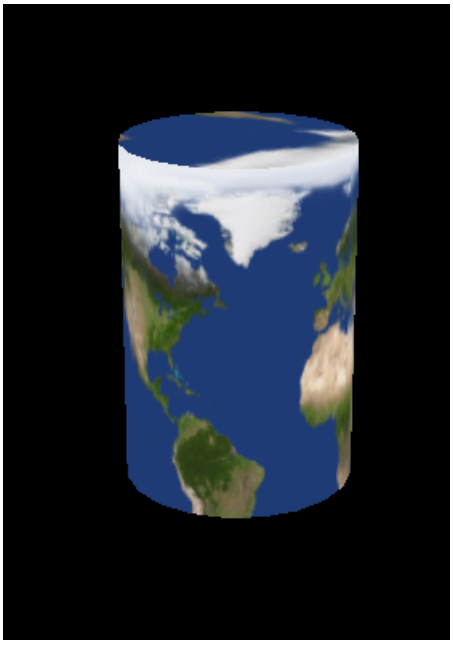

Figure 15: Cylinder final com texturas