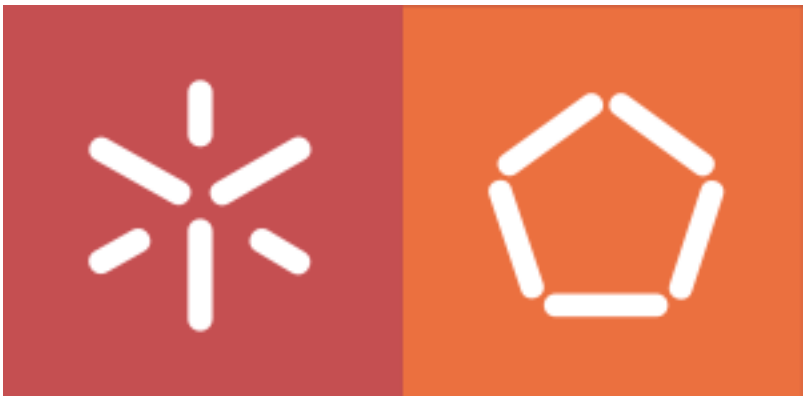

Universidade do Minho

Departamento de Informática

Computação Gráfica

Grupo nº 5

Pedro Veloso (A89557)

Carlos Preto (A89587)

Simão Monteiro (A85489)

Mafalda Costa (A83919)

Índice

Introdução

Nesta última fase do trabalho prático de Computação Gráfica, foi necessário fazer alterações quer ao Gerador, quer ao Engine.

Relativamente ao *Generator*, foi necessário acrescentar as coordenadas das normais e as coordenadas de textura para cada figura.

O *Engine* passou a suportar iluminação e a aplicar texturas a cada uma das figuras. Deste modo, foi necessário rearranjar a estrutura de dados já desenvolvida nas fases anteriores de modo a permitir registar a nova informação passada pelo ficheiro XML.

Para além das modificações enunciadas acima, o grupo optou por implementar dois mecanismos extra. O primeiro mecanismo extra implementado foi o *Picking*, de maneira a poder selecionar-se um planeta e visualizar o seu nome no canto superior esquerdo. Posteriormente, de maneira a aumentar o realismo do nosso modelo do Sistema Solar, desenvolveu-se uma box invertida, isto é, uma box com todas as faces e normais a apontar para dentro do cubo, colocando-se posteriormente esta no ficheiro XML com a textura da galáxia, e fazendo *scale* nesta de modo a todos os elementos do Sistema Solar se encontrarem dentro dessa box, dando assim o efeito de *background* de céu estrelado ao nosso modelo.

Generator

No Generator foi necessário realizar algumas alterações. Previamente, apenas era necessário calcular as coordenadas nos eixos X, Y e Z para cada figura e guardá-los num ficheiro .3D, que posteriormente seria lido pelo Engine. Nesta última fase, será necessário calcular as coordenadas das normais e as coordenadas das texturas de cada ponto.

Desta forma, cada vértice de uma figura terá 8 coordenadas, as três primeiras referentes às posições nos eixos X, Y e Z, respetivamente, as três seguintes serão correspondentes às coordenadas das normais nos eixos X, Y e Z, respetivamente, e as últimas duas coordenadas serão referentes às coordenadas de texturas desse mesmo vértice, sendo estas relativas aos eixos X e Y, visto que se trabalhará com texturas 2D.

Normais

Plano

As normais do plano são bastante simples, visto que este está contido no plano XZ. Como se pode ver na Figura 1, para a face de cima do plano, as coordenadas das normais dos pontos nessa face serão dados pelas coordenadas (0, 1, 0). Por sua vez, as coordenadas das normais dos pontos na face de baixo serão sempre (0, -1, 0).

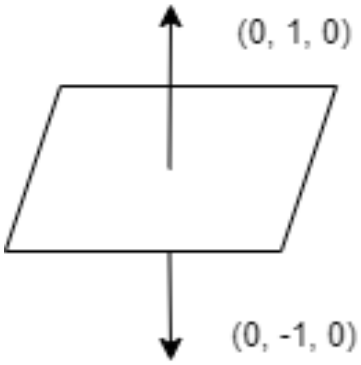

Figura 1: Normais Plano

Cubo

Para o cubo, a definição das normais de cada ponto dependerá da face na qual cada ponto está inserido. Assim, para as faces de cima e de baixo, as coordenadas das normais de cada ponto serão (0, 1, 0) e (0, -1, 0), respetivamente. Os pontos contidos nas faces de frente e de trás terão de coordenadas das normais (0, 0, 1) e (0, 0, -1), respetivamente, e por fim, os pontos contidos nas faces da direita e da esquerda terão de coordenadas (1, 0, 0) e (-1, 0, 0), respetivamente.

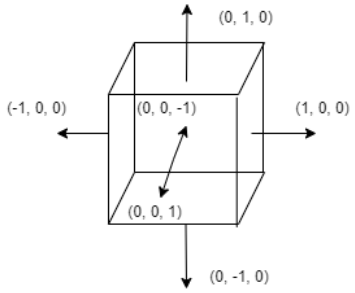

Figura 2: Normais Cubo

Esfera

Para calcular as normais de cada vértice da esfera, é preciso ter em conta a direção para o qual o ponto aponta. Essa direção é dada pelas mesmas coordenadas polares de um dado ponto, só que removendo-lhe o raio, tal como se pode observar na Figura 3.

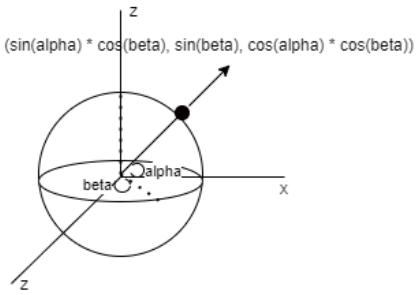

Figura 3: Normais Esfera

Cone

Para se calcular as coordenadas normais de cada ponto do cone, é preciso ter em atenção em que face um dado vértice se encontra, visto que um cone é constituído por uma base representada por um círculo e uma superfície lateral.

Assim, as coordenadas das normais dos pontos na base são dadas por (0, -1, 0), uma vez que essa face estará orientada para baixo.

Relativamente à superfície lateral do cone, a normal de cada ponto, será dada por (sin(alpha), height/stacks, cos(alpha)), onde alpha representa o ângulo associado a um determinado ponto.

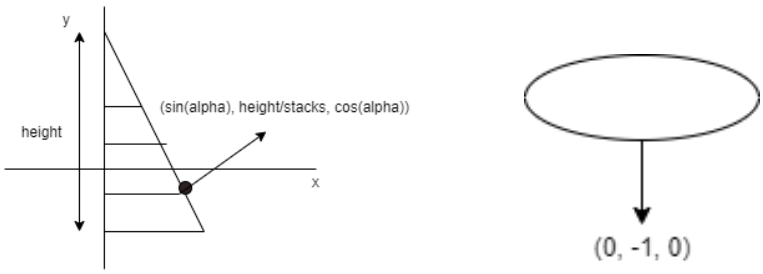

Figura 4: Normais Cone

Patches de *Bezier*

De maneira a saber as coordenadas das normais de cada ponto, será necessário calcular o produto externo entre as derivadas em u e v. Assim, através dos vetorU, vetorV e as suas derivadas, é possível calcular a normal num determinado ponto.

$$vetorU = [u^3 u^2 u^1 1] \mid vetorV = [v^3\ v^2\ v^1\ 1]$$

$$vetorU' = [3u^2\ 2u^1\ 1\ 0] \mid vetorV' = [3v^2\ 2v^1\ 1\ 0]$$

$$dU = vetorU' \times Matriz_{Bezier} \times Matriz_P \times {Matriz_{Bezier}}^T \times vetorV$$

$$dV = vetorU \times Matriz_{Bezier} \times Matriz_P \times {Matriz_{Bezier}}^T \times vetorV'$$

$$normal = dV \times dU$$

Texturas

Plano

Uma vez que o grupo definiu que o utilizador poderia indicar o número de divisões por aresta pretendidas, as coordenadas de textura de um dado ponto serão dadas pela partição na qual o vértice se encontra. Assim, as coordenadas em X e Y serão (i/partition, j/partition), como se pode ver na Figura 4, onde se considerou o número de divisões por aresta como 2.

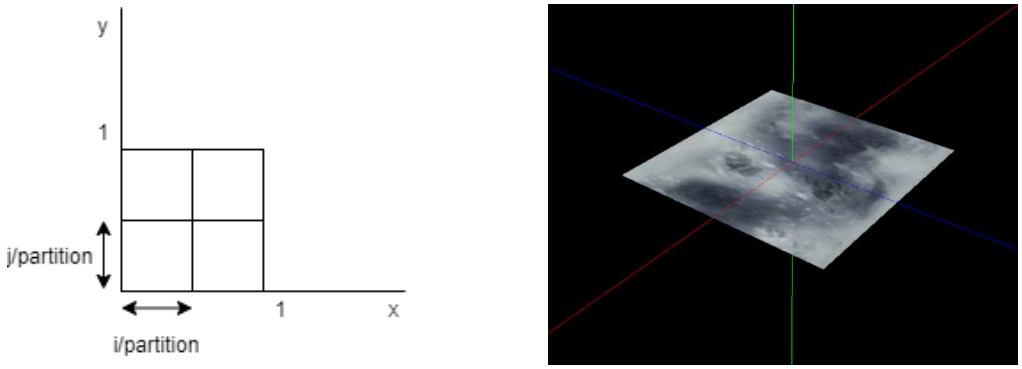

Figura 5: Texturas Plano

Cubo

Para o cubo, foi necessário definir o *template* da sua textura. Como se pode observar na Figura 5, a texturaHorizontal será 1/4 e a texturaVertical será 1/3, sendo estes valores obtidos pelo número de faces em cada eixo. Por sua vez, também é necessário considerar o número de divisões por aresta, sendo por isso cada coordenada de textura definida à custa da divisão na qual se encontra. Assim, para cada uma das esferas, e sabendo qual a diferença entre cada face do cubo, é possível aplicar uma textura ao cubo inteiro, ao invés de se aplicar a mesma textura a cada face.

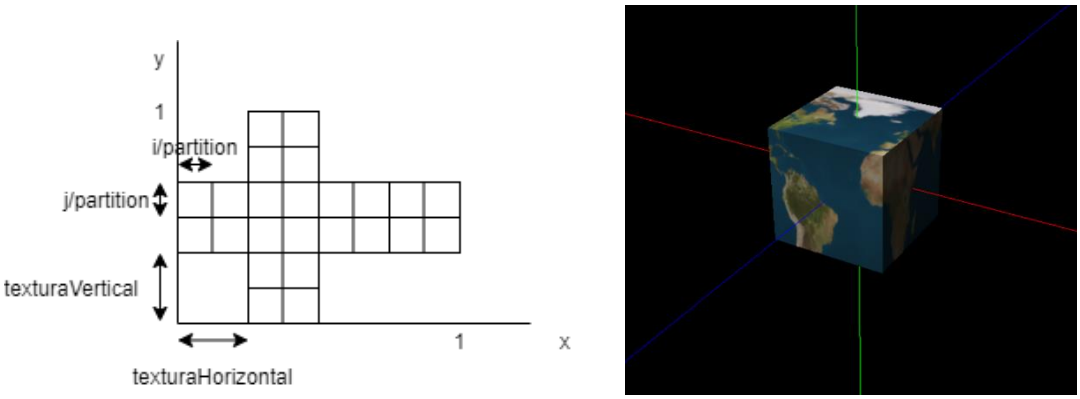

Figura 6: Texturas Cubo

Esfera

De maneira a calcular as coordenadas de textura dos vértices da esfera, considerou-se que o *template* da imagem da textura seria idêntico ao do plano. Com base na slice e stack na qual um vértice se encontra, que é dada pelo índice i e j, respetivamente, é possível determinar quais as coordenadas de textura desse ponto.

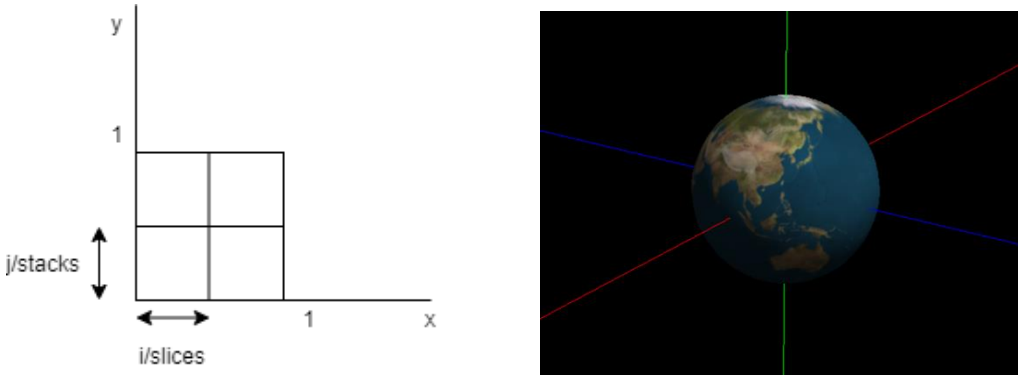

Figura 7: Texturas Esfera

Cone

Para o cone, considerou-se o *template* fornecido para os cilindros, só que removendo a circunferência da parte de cima, visto que o cone apenas tem uma circunferência na base.

O grupo apenas considerou que imagens de dimensão 512px por 512px seriam capazes de ser aplicáveis como textura do cone. Com base nas dimensões fornecidas, calcula-se as coordenadas de textura, quer dos vértices da base, quer dos vértices das faces laterais.

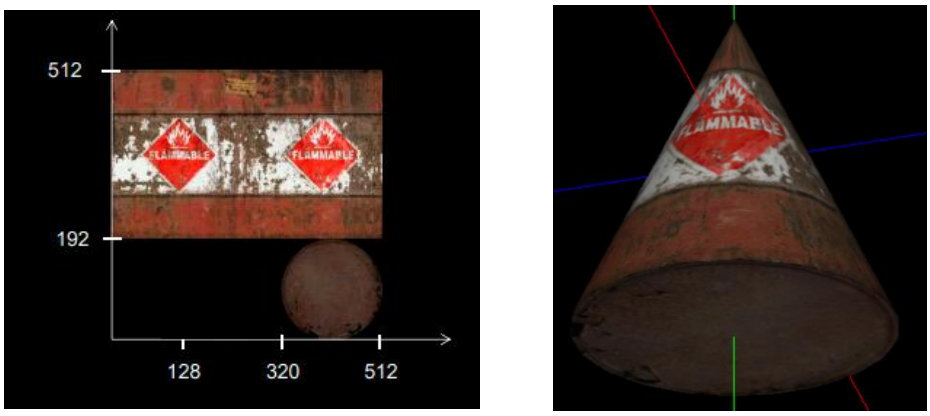

Figura 8: Texturas Cone

Patches* de *Bezier

Com base no nível de tesselagem e os índices atuais do ponto, é possível determinar quais as coordenadas de textura de cada vértice. Assim, as coordenadas de textura dos vértices segundo os eixos X e Y serão dadas por (newV, newU), de maneira à textura ficar orientada horizontalmente, e onde:

$$u = 1/level \,\&\, v = 1/level$$

$$\text{newU} = \text{j} * \text{u}$$

$$newV = w * v$$

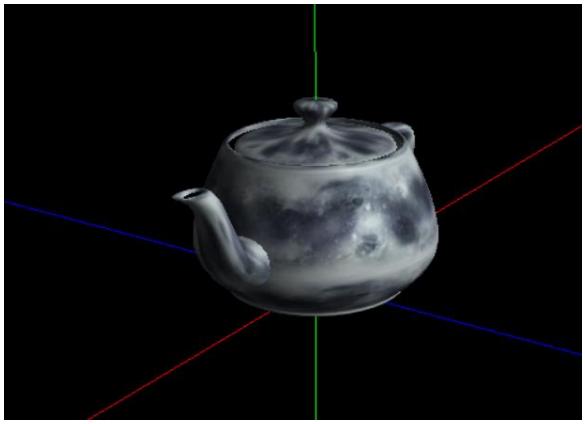

Figura 9: Texturas Patches de Bezier

Engine

Para a realização desta fase foi necessário realizar algumas modificações no modo como a informação é registada em memória, bem como no modo em que o *parsing* é realizado uma vez que agora são passados mais parâmetros pelo ficheiro XML. Para além destas modificações foi necessário preparar a *engine* para a iluminação e para as texturas aplicadas aos objetos.

Estrutura de dados

A principais alterações na estrutura de dados é referente à classe Modelo, onde passou-se a registar todos os triângulos que serão desenhados no ecrã, numa única lista que em conjunto das listas que contém as normais e as coordenadas de texturas do modelo, permite apresentar ao utilizador o modelo texturizado e iluminado. Assim a classe Modelo é apresentada da seguinte forma:

```
class Modelo{
    public:

    list<Triangulo> triangulos;
    list<Ponto> normais;
    list<Ponto> texturas;
    string nomeTextura;

    RGB amb;
    RGB diff;
    RGB spec;
    RGB emi;
```

Figura 10: Classe Modelo

Como realizou-se alterações na classe Modelo, foi necessário realizar também alterações à classe *ModeloVBO*, que regista o mesmo tipo de informação. Contudo, a principal diferença é que em vez de se registar o nome da textura, regista-se o id que a textura recebe após ser carregada.

```
class ModeloVBO {
public:

    GLuint triangulos;
    GLuint normais;
    GLuint texturas;
    int tam;
    int texturaId;

    RGB amb;
    RGB diff;
    RGB spec;
    RGB emi;

    ModeloVBO(GLuint p, GLuint n, GLuint t, int tam, int tID, RGB a, RGB d, RGB s, RGB e) {
        this->triangulos = p;
        this->normais = n;
        this->texturas = t;

        this->tam = tam;

        this->texturaId = tID;

        this->amb = a;
        this->diff = d;
        this->spec = s;
        this->emi = e;
    }
};
```

Figura 11: Classe ModelosVBO

Contudo para realizar a iluminação é necessário desenhar na cena as luzes do modelo, que podem ter diferentes características (podem ser um ponto de luz, ou uma luz direcional ou até um foco de luz), assim a classe *Scene* contém nesta quarta fase do trabalho, uma lista com todas as luzes a representar.

```
class Scene {
    public:
        Group g;
        list<Luz> luz;
```

Figura 12: Classe Scene

Para armazenar todas as informações relativas ao tipo de luz, criou-se a classe *Luz*.

Iluminação

Uma vez que uma cena passou a representar iluminação, é necessário ativar as novas funcionalidades, sendo acrescentado nesse sentido o seguinte excerto de código:

```
int numLuz = 0;
for (list<Luz>::iterator it = luz.begin(); (it != luz.end()) && numLuz<8; it++) {
    glEnable(GL_LIGHT0 + numLuz);

    glLightfv(GL_LIGHT0 + numLuz, GL_AMBIENT, light_ambient);
    glLightfv(GL_LIGHT0 + numLuz, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0 + numLuz, GL_SPECULAR, light_specular);

    numLuz++;
}
```

Figura 13: Ativação das funcionalidades desta fase

Posteriormente, deve-se “ligar” todas as luzes (presentes na lista de luzes já referida) sendo para tal realizado, na função *renderScene* o seguinte excerto de código:

```
for (list<Luz>::iterator it = luz.begin(); (it != luz.end()) && numLuz < 8; it++) {

    switch (it->tipoLuz)
    {
        case 'P': {
            float pos[4];
            pos[0] = it->posLuzX;
            pos[1] = it->posLuzY;
            pos[2] = it->posLuzZ;
            pos[3] = 0.0f;
            glLightfv(GL_LIGHT0 + numLuz, GL_POSITION, pos);
            break;
        }
        case 'D': {
            float pos[4];
            pos[0] = it->posLuzX;
            pos[1] = it->posLuzY;
            pos[2] = it->posLuzZ;
            pos[3] = 1.0f;
            glLightfv(GL_LIGHT0 + numLuz, GL_POSITION, pos);
            break;
        }
        case 'S': {
            GLfloat spot_direction[] = { it->posLuzX, it->posLuzY, it->posLuzZ };
            glLightf(GL_LIGHT0 + numLuz, GL_SPOT_CUTOFF, it->angleLuz);
            glLightfv(GL_LIGHT0 + numLuz, GL_SPOT_DIRECTION, spot_direction);
            break;
        }
    }

    numLuz++;
}
```

Figura 14: Luzes

Texturas

Para desenhar as texturas dos objetos deve-se realizar a leitura das coordenadas de textura e armazená-las na variável *texturas* presente na classe *ModeloVBO*, acima apresentada. Posteriormente, deve-se de desenhar as texturas juntamente com os triângulos, como se observa no excerto de código seguinte:

```
glBindBuffer(GL_ARRAY_BUFFER, it->triangulos);
glVertexPointer(3, GL_FLOAT, 0, 0);

glBindBuffer(GL_ARRAY_BUFFER, it->normais);
glNormalPointer(GL_FLOAT, 0, 0);

glBindBuffer(GL_ARRAY_BUFFER, it->texturas);
glTexCoordPointer(2, GL_FLOAT, 0, 0);

glDrawArrays(GL_TRIANGLES, 0, it->tam);
```

Figura 15: Desenhar as texturas

Contudo, antes de mandar desenhar as texturas é necessário ativar esta funcionalidades, recorrendo à função:

"glEnableClientState (GL_TEXTURE_COORD_ARRAY)"

Picking

De maneira a complementar o trabalho, o grupo optou por implementar *Picking*. Através do *Picking*, é possível selecionar, com o botão do meio do rato, cada elemento do Sistema Solar, fazendo com que no canto superior esquerdo do ecrã apareça o nome desse mesmo elemento selecionado, tal como se observa na figura seguinte.

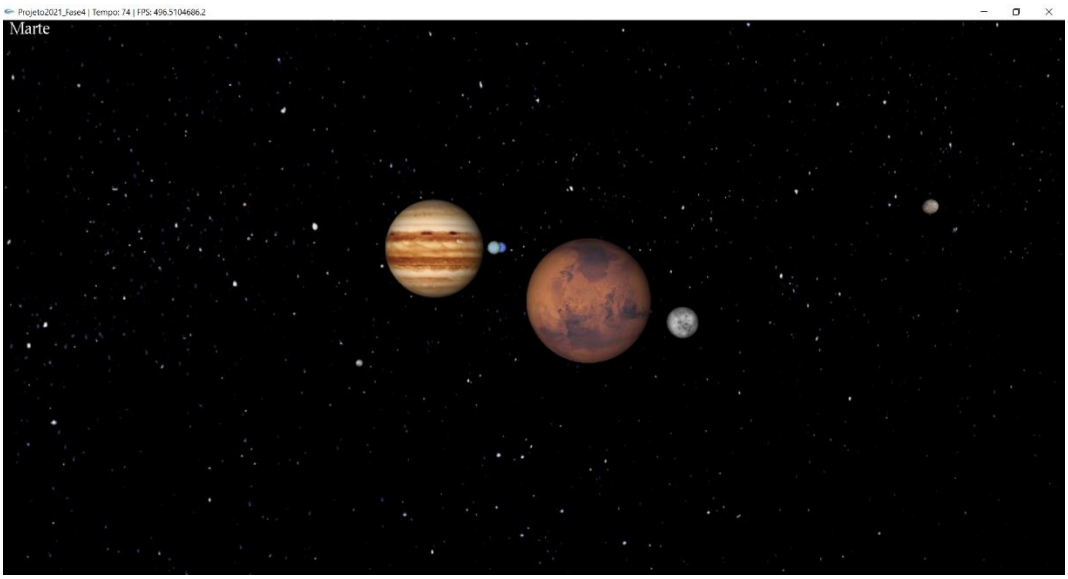

Figura 16: Picking

Sistema Solar

Após aplicar as texturas aos planetas e a iluminação à cena obteve-se o seguinte modelo do sistema solar.

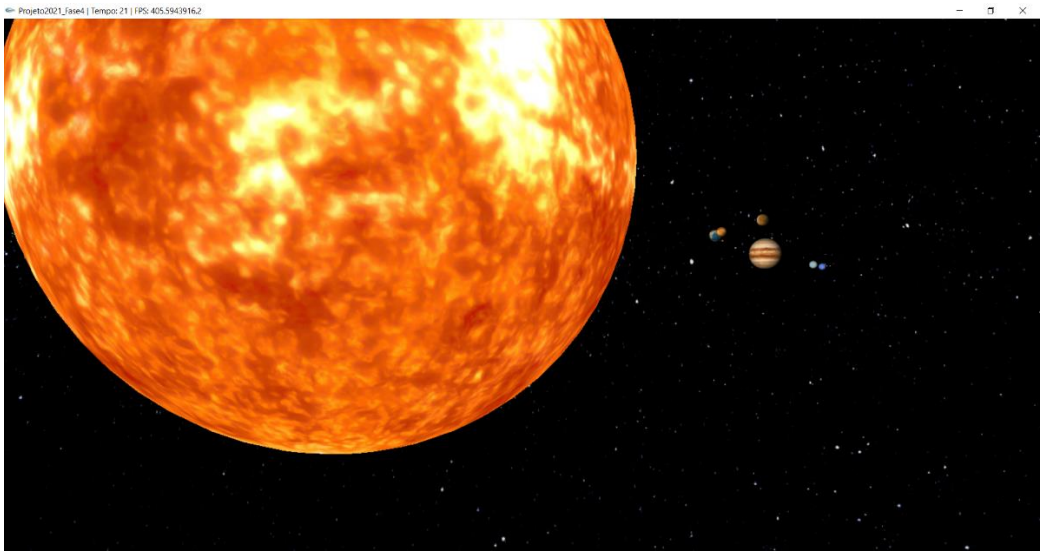

Figura 17: Sistema Solar

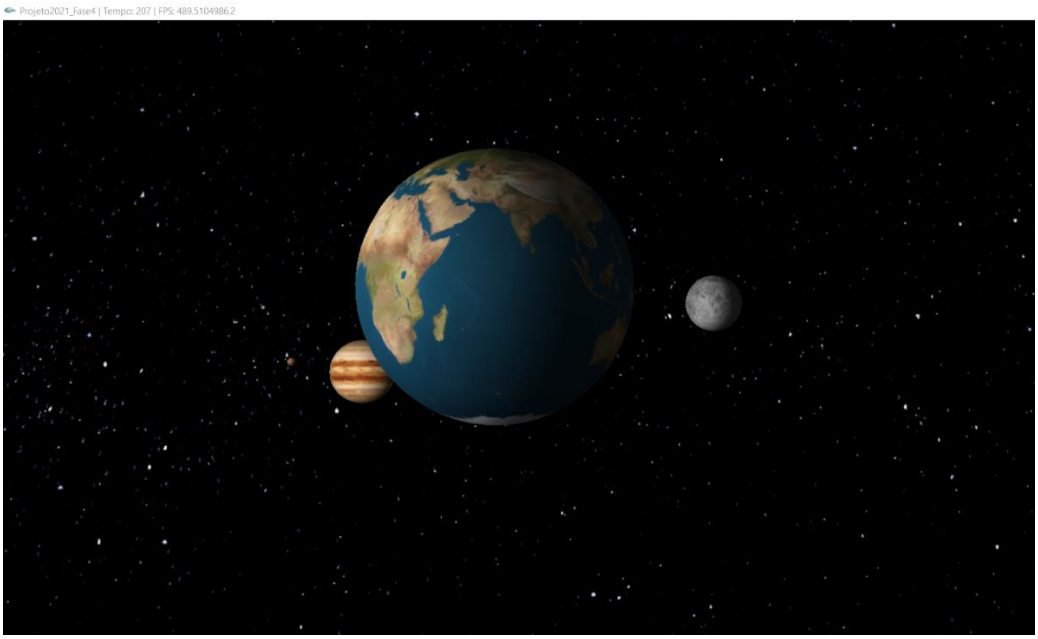

Figura 18: Terra e Lua

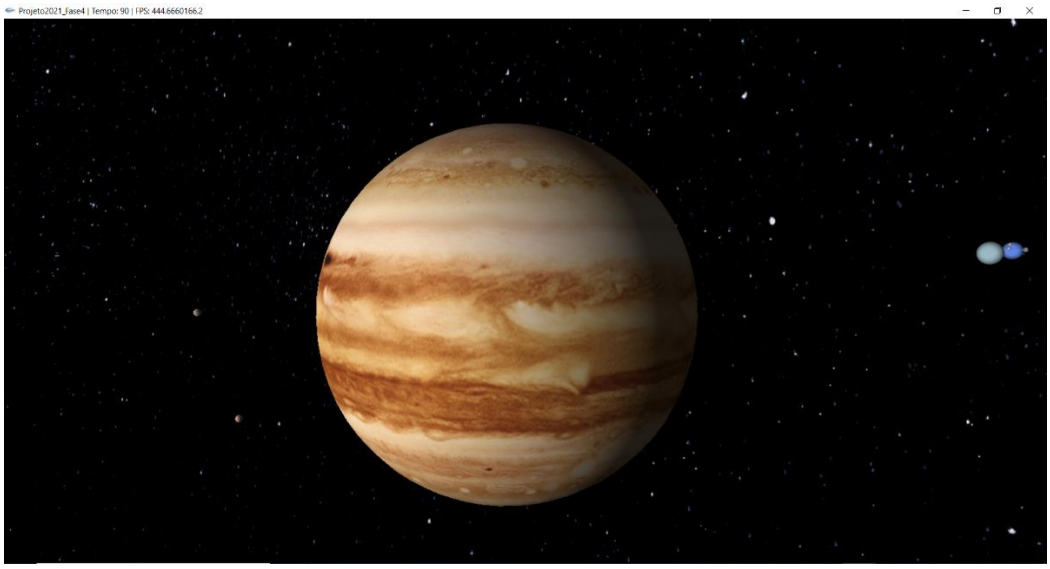

Figura 19: Júpiter

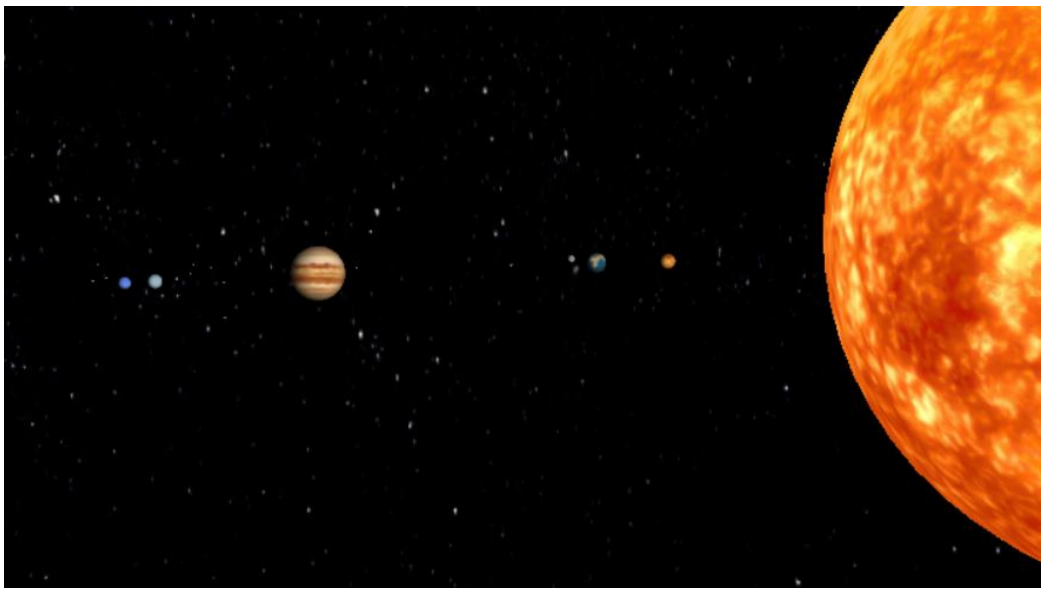

Figura 20: Sistema Solar Perspetiva Esquerda

Conclusão

Após a realização de todas as fases do trabalho foi possível uma representação do sistema solar, que apesar de simplificado permite visualizar a existência dos vários planetas existentes, bem como algumas das suas luas.

Apesar de nas fases anteriores já se ter uma representação 3D do sistema solar, com a realização desta fase, que acrescentou a iluminação e as texturas dos planetas, é possível verificar de uma forma mais agradável o trabalho desenvolvido.

Além do ficheiro XML do Sistema Solar, foram desenvolvidos modelos extra para testar a aplicação das normais e texturas às diferentes figuras possíveis.

O grupo encontra-se satisfeito com o trabalho desenvolvido, e julga ter cumprido todos os requisitos pretendidos para cada Fase de desenvolvimento do trabalho prático. Contudo, como trabalho futuro, e de maneira a acrescentar mais realismo à demo criada deveria ser acrescentado mais Luas a planetas como por exemplo Júpiter e Saturno, para além de ajustar com mais precisão a escala e posição dos planetas.