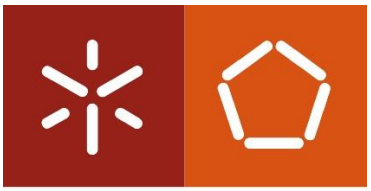


Universidade do Minho

Mestrado Integrado de Engenharia Informática

Computação Gráfica (2ºSemestre - 2019/20)

Relatório CG - TP4

Grupo Nº 33

A85813 António Alexandre Carvalho Lindo

A85400 Nuno Azevedo Alves da Cunha

A85919 Pedro Dias Parente

Braga

31 de Maio de 2020

Conteúdo

Lista de Figuras

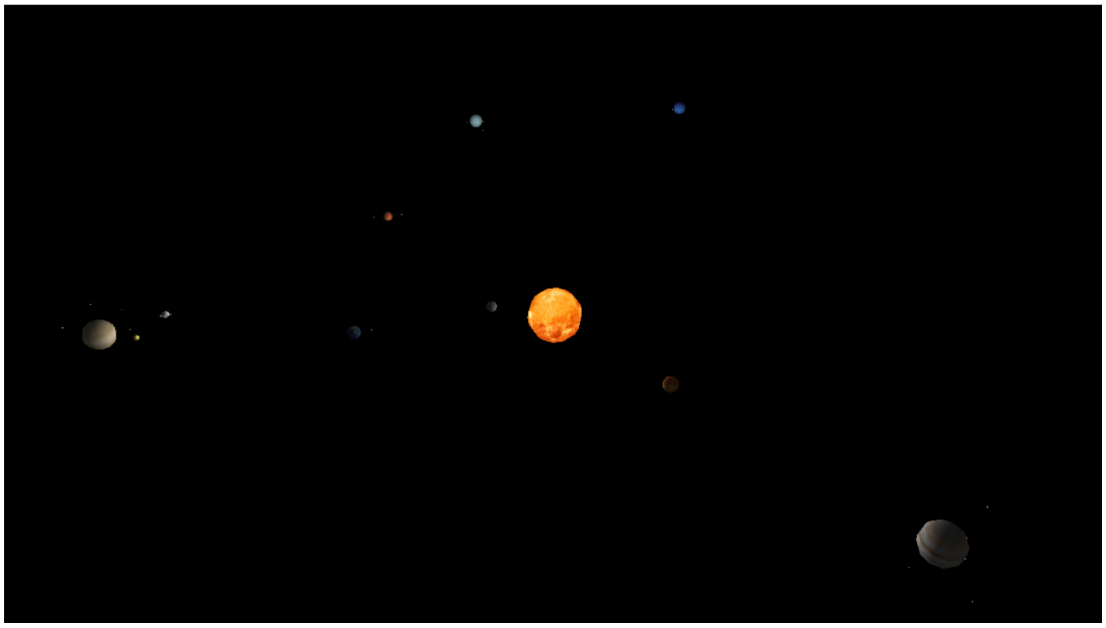
Capítulo 1

Introdução

Este relatório tem como objetivo descrever a implementação da 4ª fase do trabalho prático de CG.

A meta desta fase do Trabalho consistiu em implementar iluminação, texturas e normais para cada vértice. Foi necessária a extenção do formato XML usado de modo a poder definir as componentes difusa, especular, emissiva e ambiente das cores e definir as fontes de luz.

Figura 1.1: Sistema solar



Capítulo 2

Generator

Houve algumas alterações no Generator pois foi preciso fazer umas alterações no cálculo das curvas de bezier e também foi preciso adicionar as funcionalidades do cálculos das normais e das texturas ao Generator. É de algum interesse realçar que agora são gerados 3 ficheiros por geração de figura: os ficheiros .3d, os ficheiros .3dn e os ficheiros .3dt, sendo que estes correspondem ao cálculo dos pontos, das normais, e das coordenadas das texturas, respetivamente.

2.1 Cálculo das curvas de Bezier

Entre a fase anterior e esta fase apercebemos-nos que a forma como estávamos a calcular os pontos das curvas não era a mais eficiente, pois através de cálculos matriciais chegamos ao valor muito mais facilmente, para além do mais que para calcular as normais necessárias para a luz também foi preciso cálculos matriciais, por isso mudar a forma como estavamos a calcular anteriormente pareceu-nos a decisão mais acertada.

Figura 2.1: Cálculo matricial dos pontos e respetivas normais

$$p(u,v) = [u^3 \quad u^2 \quad u \quad 1]M \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} M^T \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

$$\frac{\partial p(u,v)}{\partial u} = [3u^2 \quad 2u \quad 1 \quad 0]M \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} M^T V^T$$

$$\frac{\partial p(u,v)}{\partial v} = UM \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} M^T \begin{bmatrix} 3v^2 \\ 2v \\ 1 \\ 0 \end{bmatrix}$$

2.2 Normais

Como referido, para o cálculo da reflexão da luz, é preciso calcularmos as normais de cada vértice.

2.2.1 Plano

No plano o vetor normal é obviamente apenas o vetor (0,1,0).

2.2.2 Cubo

Para o Cubo o cálculo das normais é simples: para cada face calcula-se o vetor que "sai"da face, com um comprimento de 1 (pois tem que estar normalizado).

2.2.3 Esfera

Para a Esfera também foi bastante simples calcular: para cada vértice normaliza-se o vetor que vai da origem ao vértice.

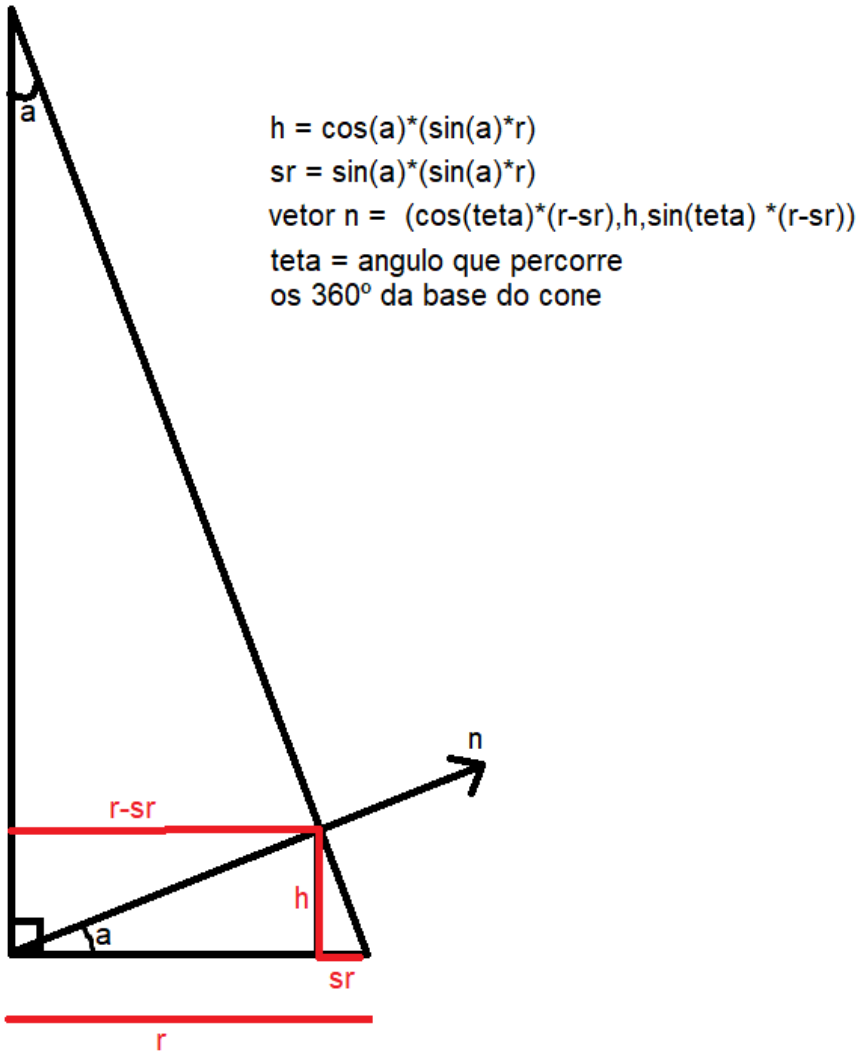
2.2.4 Curvas de Bezier

Como dito na secção anterior, o cálculo das normais para cada vértice é feito com um cálculo matricial.

2.2.5 Cone

O cálculo das normais do cone foram um pouco mais complexas. A seguinte imagem pode ajudar a perceber a estratégia utilizada:

Figura 2.2: Cálculo das normais do Cone



Chegamos à conclusão que para qualquer vértice do cone, a sua normal é o vetor que parte da origem até ao ponto de altura h, apenas tendo de ser orientado com as componentes x e z corretas (sendo que a altura h **NÃO é dependente da altura do vértice**, é calculado logo no ínicio da geração do cone e mantém-se constante durante a geração).

2.3 Texturas

2.3.1 Cubo e Plano

O cálculo das coordenadas das texturas para o Cubo e Plano é um processo muito simples, é apenas preciso ter cuidado com a sincronização dos vértices desenhados com as coordenadas das texturas (pois o cálculo das coordenadas das texturas em si é um processo trivial para estas figuras).

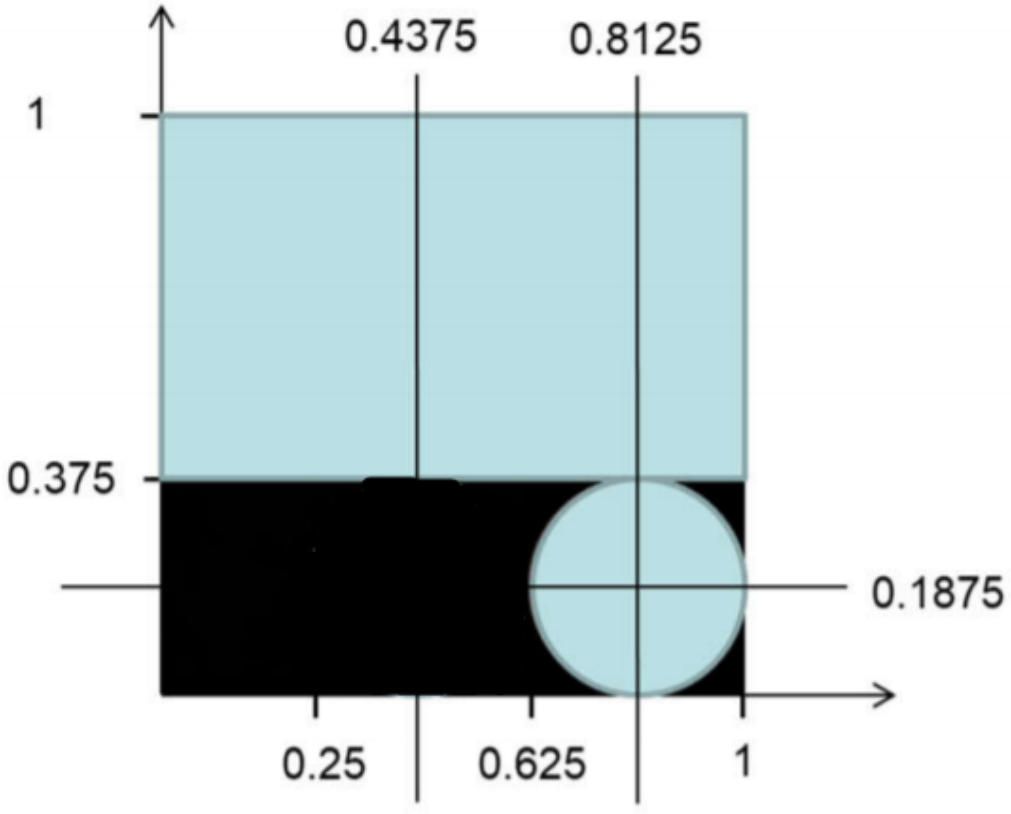
2.3.2 Esfera

No caso da esfera a imagem da textura foi dividida pelo número de stacks e slices vertical e horizontalmente, respetivamente, e sincronizadas aos vértices dos triângulos correspondentes, sendo importante salientar que no caso do topo e base da esfera o cálculo das coordenadas da textura é ligeiramente diferente devido à natureza de como é desenhada a esfera.

2.3.3 Cone

Semelhante ao que foi feito no guião 11 no cilindro, o cálculo das coordenadas das texturas no cone foi feita assumindo que o parte lateral do cone estava definida acima de y = 0.375 e que a base do cone estava definida abaixo desse y e depois de x = 0.625.

Figura 2.3: Coordenadas de textura do Cone



Capítulo 3

XML

Nesta fase foram adicionados várias extenções ao XML:

Figura 3.1: Câmera

```
<camera dirX='-1' dirY='-1' dirZ='-1' posX='200' posY='200' posZ='200' type='1'>
```

A extensão câmera contem a direção para qual a câmera está a apontar (dirX, dirY, dirZ), a posição onde a mesma se encontra (posX, posY, posZ) e ainda se ela é fixa ou móvel (type).

A extensão light contem o tipo de luz que pode ser:

Figura 3.2: Light Spotlight

```
<light type='SPOTLIGHT' posX='0' posY='-0.5' posZ='-4' dirX='0' dirY='0' dirZ='-1' ang='10' att='0.005' exponent='60'/>
```

- Spotlight: Neste caso contem a posição (posX, posY, posZ), a direção para onde está a apontar (dirX, dirY, dirZ), o ângulo (ang), a atenuação (att) e o exponent (exponent).

Figura 3.3: Light Point

```
<light type='POINT' posX='0' posY='0' posZ='0' att='0.0000001'/>
```

- Point: Neste caso contem a posição (posX, posY, posZ) e a atenuação (att).

Figura 3.4: Light Directional

```
<light type='DIRECTIONAL' dirX='1' dirY='-1' dirZ='1'/>
```

- Directional: Contem a direção (dirX, dirY, dirZ).

Figura 3.5: Model

```
<model file='sun.3d' material='sun.mat' texture='sun.jpg'/>
```

Model foi extendido podendo agora adicionar-se um textura (texture) e/ou um material (material).

Capítulo 4

Engine

4.1 Iluminação

4.1.1 Luzes

Relativamente à iluminação, foram implementados 3 tipos de iluminação. (Point, Directional, Spotlight) sendo possivel colocar até 8 luzes por Scene. A definição destas luzes é feita antes de ser desenhado qualquer triangulo, são assim gerais, não havendo luzes a iluminar grupos individuais por exemplo.

Figura 4.1: Light Setup

```
void setupLight(Light* l, int i) {
    float pos[4], dir[4];
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0 + i);
    if (l->getType() == L_POINT) {
        pos[0] = l->getPosX(); pos[1] = l->getPosY(); pos[2] = l->getPosZ(); pos[3] = 1;
        glLightf(GL_LIGHT0 + i, GL_QUADRATIC_ATTENUATION, l->getAtt());
        glLightfv(GL_LIGHT0 + i, GL_POSITION, pos);
    }
    if (l->getType() == L_DIRECTIONAL) {
        dir[0] = l->getDirX(); dir[1] = l->getDirY(); dir[2] = l->getDirZ(); dir[3] = 0;
        glLightfv(GL_LIGHT0 + i, GL_POSITION, dir);
    }
    if (l->getType() == L_SPOTLIGHT) {
        pos[0] = l->getPosX(); pos[1] = l->getPosY(); pos[2] = l->getPosZ(); pos[3] = 1;
        dir[0] = l->getDirX(); dir[1] = l->getDirY(); dir[2] = l->getDirZ(); dir[3] = 1;
        glLightfv(GL_LIGHT0 + i, GL_POSITION, pos);
        glLightfv(GL_LIGHT0 + i, GL_SPOT_DIRECTION, dir);
        glLightf(GL_LIGHT0 + i, GL_SPOT_CUTOFF, l->getAngCuttof());
        glLightf(GL_LIGHT0 + i, GL_SPOT_EXPONENT, l->getExponent());
        glLightf(GL_LIGHT0 + i, GL_QUADRATIC_ATTENUATION, l->getAtt());
    }
    GLfloat qaAmbientLight[] = { 0.6, 0.6, 0.6, 1.0 };
    GLfloat qaDiffuseLight[] = { 0.8, 0.8, 0.8, 1.0 };
    GLfloat qaSpecularLight[] = { 1.0, 1.0, 1.0, 1.0 };
    glLightfv(GL_LIGHT0 + i, GL_AMBIENT, qaAmbientLight);
    glLightfv(GL_LIGHT0 + i, GL_DIFFUSE, qaDiffuseLight);
    glLightfv(GL_LIGHT0 + i, GL_SPECULAR, qaSpecularLight);
}
```

4.1.2 Materiais

Os materiais são definido num ficheiro de texto em que podem ser definidos ou omitidos(em favor dos valores default) os diversos parametros dos materiais.

Figura 4.2: Materials

```
if (material) {
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, material->getAmb());
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, material->getDiff());
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, material->getSpec());
    glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, material->getEmissive());
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, material->getShininess());
}
```

Existe ainda a função **defaultMaterials()** que e chamada no fim de desenhar cada modelo e dá reset ao material.

4.1.3 Normais

É ainda necessário para que a iluminação funcione corretamente o uso de NORMAL_ARRAYS usando os dados dos ficheiros carregados no setup e gerados pelo generator.

Figura 4.3: Normals

```
glEnableClientState(GL_NORMAL_ARRAY);
glBindBuffer(GL_ARRAY_BUFFER, normalsBuf[currentModel]);
glBufferData(GL_ARRAY_BUFFER, normals.size() * sizeof(float), &normals[0], GL_STATIC_DRAW);
glNormalPointer(GL_FLOAT, 0, 0);
```

4.2 Texturização

Na parte da texturização a engine é responsável por carregar as texturas para o software e associar as texturas corretas a cada TextureArray do modelo.

O carregamento das texturas é feito usando a livraria devIL na seguinte função:

Figura 4.4: Load Texture

```
void loadTexture(std::string s, Model * model) {

    unsigned int t, tw, th;
    unsigned char* texData;
    unsigned int texID;
    ilInit();
    ilEnable(IL_ORIGIN_SET);
    ilOriginFunc(IL_ORIGIN_LOWER_LEFT);
    ilGenImages(1, &t);
    ilBindImage(t);
    ilLoadImage((ILstring)s.c_str());
    tw = ilGetInteger(IL_IMAGE_WIDTH);
    th = ilGetInteger(IL_IMAGE_HEIGHT);
    ilConvertImage(IL_RGBA, IL_UNSIGNED_BYTE);
    texData = ilGetData();

    glGenTextures(1, &texID);

    glBindTexture(GL_TEXTURE_2D, texID);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);

    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, tw, th, 0, GL_RGBA, GL_UNSIGNED_BYTE, texData);
    glGenerateMipmap(GL_TEXTURE_2D);

    glBindTexture(GL_TEXTURE_2D, 0);

    model -> setTextureID(texID);

}
```

A texturização em si é feita com os TEXTURE_CORD_ARRAYS carregando os dados do ficheiro gerado pelo gerador.

Figura 4.5: Texture

```
if (textures.size() != 0) {
    glEnableClientState(GL_TEXTURE_COORD_ARRAY);
    glBindTexture(GL_TEXTURE_2D, model->getTextureID());
    glBindBuffer(GL_ARRAY_BUFFER, texturesBuf[currentModel]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(float) * textures.size(), &textures[0], GL_STATIC_DRAW);
    glBindBuffer(GL_ARRAY_BUFFER, texturesBuf[currentModel]);
    glTexCoordPointer(2, GL_FLOAT, 0, 0);
}
```

4.3 Câmera

Foi adicionada uma extensão ao XML permitindo assim não só definir a câmara na scene como também desenhar até um grupo e uma luz no espaço câmera.

4.3.1 Tipos de Camera

A câmera pode ser de 2 tipo diferentes, é possivel criar uma camera fixa que apenas permite zoom, ou uma camera móvel que permite moviento livre.

4.3.2 Setup camera

A camêra é inicializada com os parametros de localização e vetor de direção do olhar. Este vetor de direção do olhar será depois convertido para dois ângulos de modo a utilizar coordenadas esféricas que facilitam o posicionamento do ponto para que a camera olha e também para facilitar o movimento da câmera no caso de esta ser desse tipo.

4.3.3 Movimento

O movimento da câmera no caso desta ser móvel é feito usando os ângulos das coordenadas esféricas para calcular o vector que levaria a posição seguinte da câmera e somando-o à posição atual. É possivel alterar estes angulos usando as arrow keys.

4.3.4 Desenho no espaço câmera

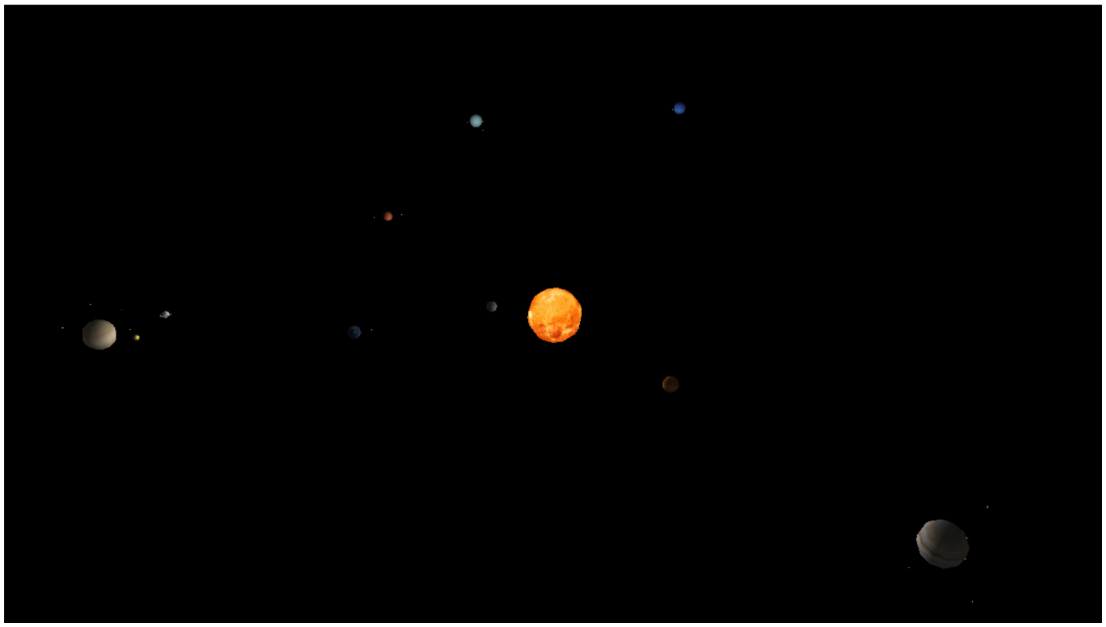
O desenho no espaço câmera é feito antes da função **gluLookAt** assim tudo o que será desenhado(incluindo possivelmente uma fonte luminosa) ficará fixo em relação à mesma.

4.4 Testes e resultados

Com o intuito de demonstrar todas as funcionalidades implementadas geramos vários testes que podem ser explorados pelo docente. Os testes que geramos são: SolarSystem.xml (projeto principal), caixasEmpilhadas.xml e montanha.xml.

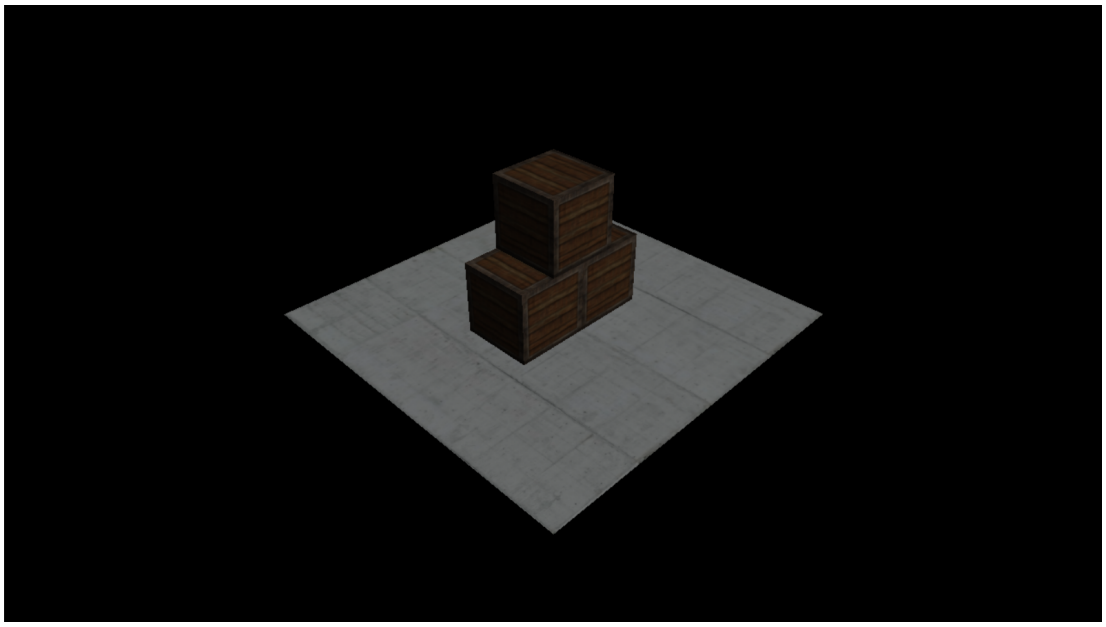
4.4.1 SolarSystem.xml

Figura 4.6: Solar System



4.4.2 caixasEmpilhadas.xml

Figura 4.7: Caixas



4.4.3 montanha.xml

Figura 4.8: Montanha