Ciências ULisboa

Faculdade de Ciências da Universidade de Lisboa

RELATÓRIO 2º TRABALHO DE COMPUTAÇÃO GRÁFICA 2019/2020

WebGl

Computação Gráfica Grupo 19:

Alexandre Monteiro nº51023
António Fróis nº51050
Pedro Mena nº51024

Indíce

Introdução

Neste trabalho foi nos dado uma cena 3D com um modelo de um edificio para efetuar alterações, gerar representações e animações usando a biblioteca WebGL. Para realizar os exercicios que nos foram dados partimos dos ficheiros originais para fazer cada exercicio.

Exercicio 1

Alinea a:

Foram criados em dois ficheiros Blender diferentes um cubo e uma pirâmide, com 0,12 unidades de tamanho e no Blender também foram mudadas as coordenadas de forma a aparecerem os objetos nos dois lados do edifício. Para a produção do cubo foi simplesmente adicionado a mesh do cubo, para fazer a pirâmide partimos de um cubo, selecionamos os 4 vértices superiores do cubo e foi feito merge no centro. Depois estes documentos foram exportados com as flags dadas, convertidos em ficheiros json e adicionados à pasta models. Nos dois ficheiros json destes sólidos foi colocada a linha "diffuse" com os valores corretos para pintar o cubo e a pirâmide com as cores que foram pedidas no enunciado do trabalho.


```
{
  "alias" : "Cubo",
  "vertices" : [-3.108275,0.0,-0.06,-3.108275,0.0,0.06,-3.228275,0.0,0.06,-3.228275,0.0,-0.06,-3.108275,0.12,-0.06,-3.108275,0.12,0.06,-3.228275,0.12,0.06,-3.228275,0.12,-0.06],
  "indices" : [1,3,0,7,5,4,4,1,0,5,2,1,2,7,3,0,7,4,1,2,3,7,6,5,4,5,1,5,6,2,2,6,7,0,3,7],
  "ambient"  : [0.1,0.1,0.1,1.0],
  "diffuse"  : [1,0.65,0,1.0],
  "specular" : [0.2,0.2,0.2,1.0]
}
```


Figura 1 - Ficheiro json do Cubo


```
{
  "alias" : "Piramide",
  "vertices" : [3.22828,0.0,-0.06,3.22828,0.0,0.06,3.10828,0.0,0.06,3.10828,0.0,-0.06,3.16828,0.12,0.0],
  "indices" : [1,3,0,0,4,1,1,4,2,2,4,3,4,0,3,1,2,3],
  "ambient"  : [0.1,0.1,0.1,1.0],
  "diffuse"  : [0,1,1,1.0],
  "specular" : [0.2,0.2,0.2,1.0]
}
```


Figura 2 - Ficheiro json da Piramide

Por fim adicionaram-se duas funções Scene.loadObject() á função load de modo a que estes fossem carregados na cena.

```
function load(){
    //Load the office building
    Scene.loadObjectByParts('models/geometry/Building/part','Office',758);
    //Load the ground
    Scene.loadObject('models/geometry/Building/plane.json','Plane');
    Scene.loadObject('models/geometry/Building/obj.json','Object');
    Scene.loadObject('models/geometry/Cubo/Cubo.json','Cubo');
    Scene.loadObject('models/geometry/Piramide/Piramide.json','Piramide');
}
```

Figura 3 - Alterações da função load() para introduzir o cubo e a pirâmide

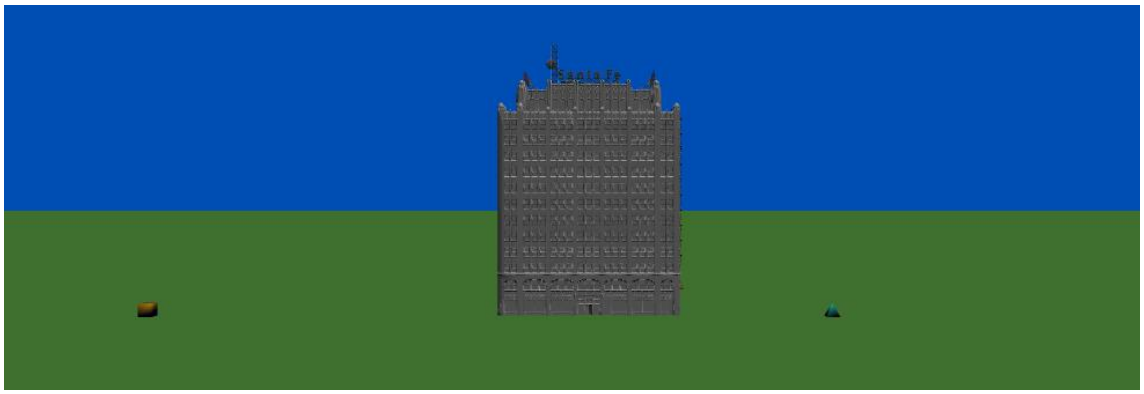


Figura 4 - Cubo e Pirâmide no cenário 3D

Alinea b:

Para esta alínea foram alterados os valores do prg.uLightPosition na função configure() de modo a que a luz só incidisse na zona do edifico que está de frente para a câmara, então alterou-se o valor da luz no x e no y para 0 e o z para 1 de modo a que a luz só incidisse na frente do edifício.

```
//Update lights for this example
gl.uniform4fv(prg.uLightAmbient,      [0.05,0.05,0.05,1.0]);
gl.uniform3fv(prg.uLightPosition,     [0,0,1]);
gl.uniform4fv(prg.uLightDiffuse,      [0.7,0.7,0.7,1.0]);
```

Figura 5 - Alterações na posição da luz

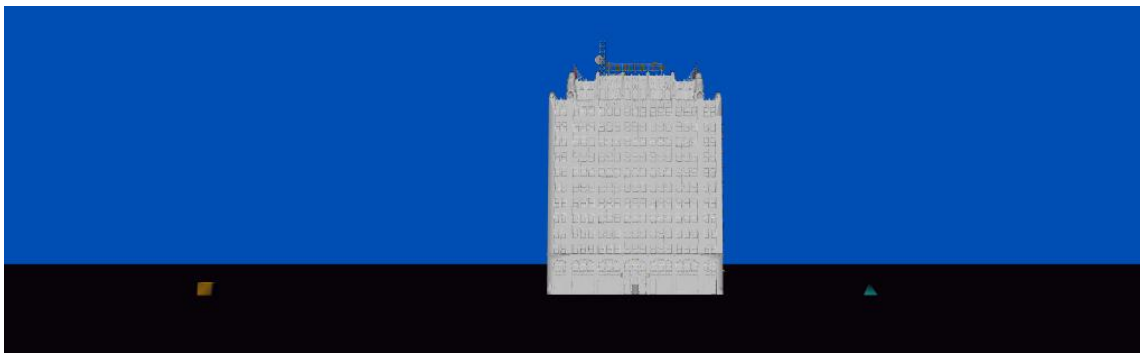


Figura 6 - Luz a incidir apenas a frente do edificio

Alinea c:

Alterou-se a posição inicial da câmara alterando-se apenas o valor da coordenada x de modo a que a câmara fosse o mais para a direita possível de modo a que ao depois se mexer a câmara com o movimento do rato se visse centrado a fachada do lado direito do edifício.

```
//Creates and sets up the camera location
camera = new Camera(CAMERA_ORBIT_TYPE);
camera.goHome([3,1,8]);
camera.hookRenderer = draw;
```

Figura 7- Posição inicial da câmara alterada

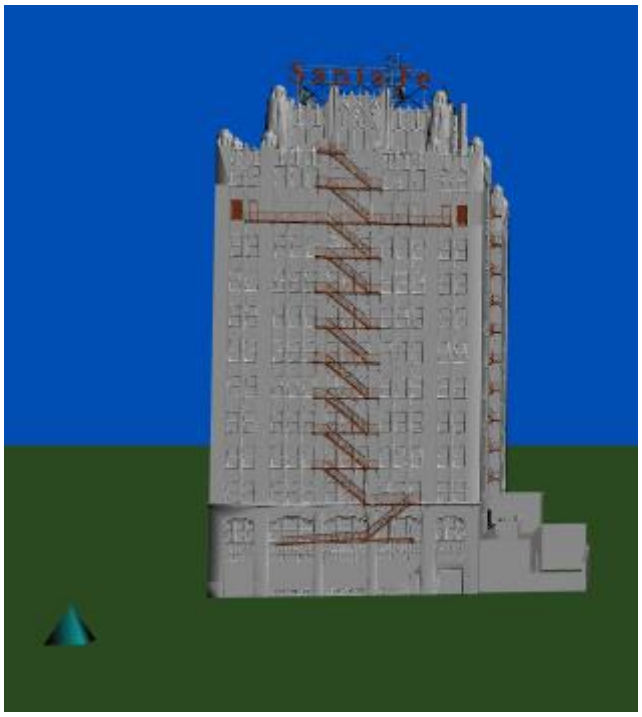


Figura 8 - Visão lateral do edifício

Alinea d:

Fez-se o mesmo que na alínea c mas alterou-se a coordenada y de modo a que a câmara se chegasse mais perto do edifício para se obter a imagem acima ao arrastar o rato. (Igual ao c mas alterando apenas o valor de y)

```
//Creates and sets up the camera location
camera = new Camera(CAMERA_ORBIT_TYPE);
camera.goHome([1,0,8]);
camera.hookRenderer = draw;
```

Figura 9 - Posição inicial da câmara alterada

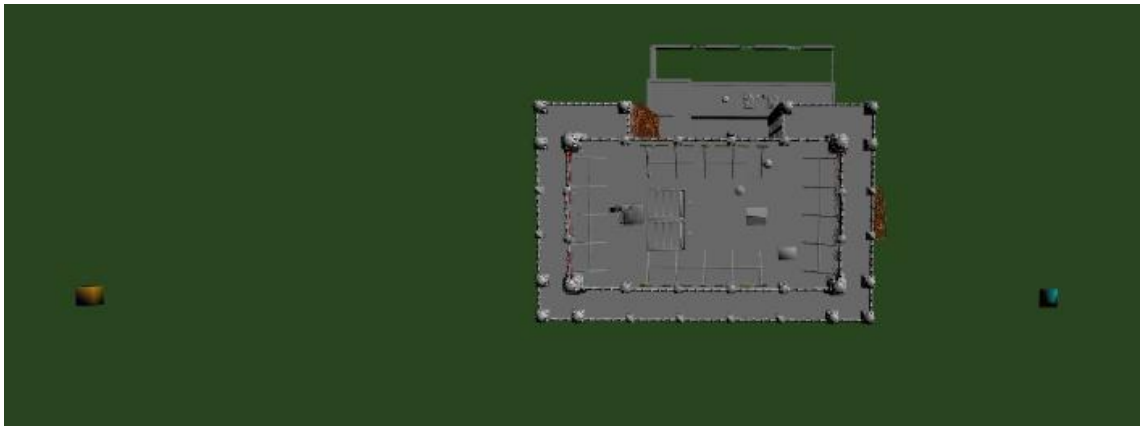


Figura 10 - Visão de topo do edifíco

Alinea e:

De modo a se obter uma visão do que se encontra no interior do edifício foi alterado o valor zNear que inicialmente estava a 1 e foi alterado para 8.4, ou seja, alterou-se a posição do front clipping plane para ser possivel visualizar o interior do edifício. Após estas alterações podemos observar que no interior se encontra a cabeça de um macaco pintada de amarelo.

```
var fovy = 30;
function updateTransforms(){
    if (projectionMode == PROJ_PERSPECTIVE){
        mat4.perspective(fovy, c_width / c_height, 8.4, 400.0, pMatrix);
    }
    else{
        cw = c_width/250;
        ch = c_height/250;
        mat4.ortho(-cw, cw, -ch, ch, -10.0, 10.0, pMatrix);
    }
}
```

Figura 11 - Valor do zNear alterado

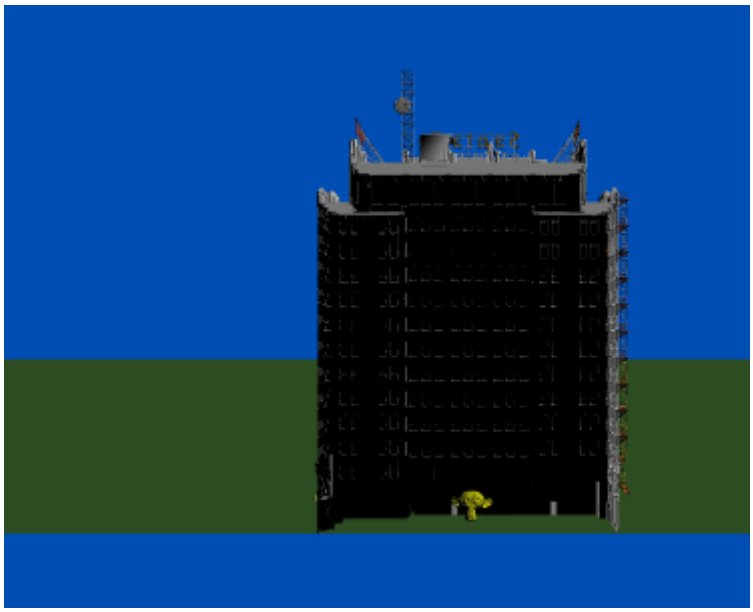
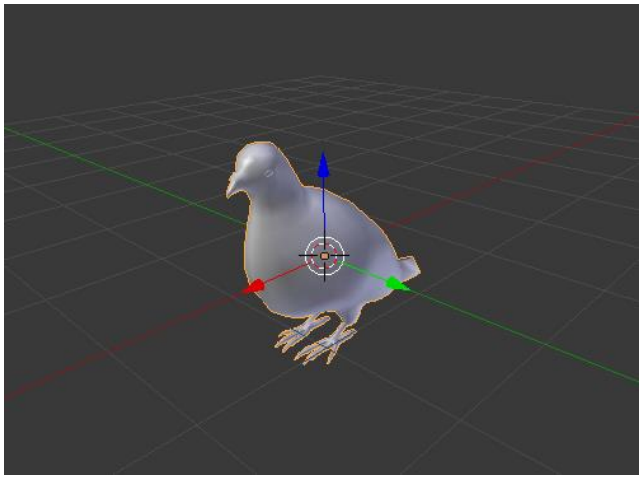


Figura 12 - Resultado das alterações da posição do front clipping plane

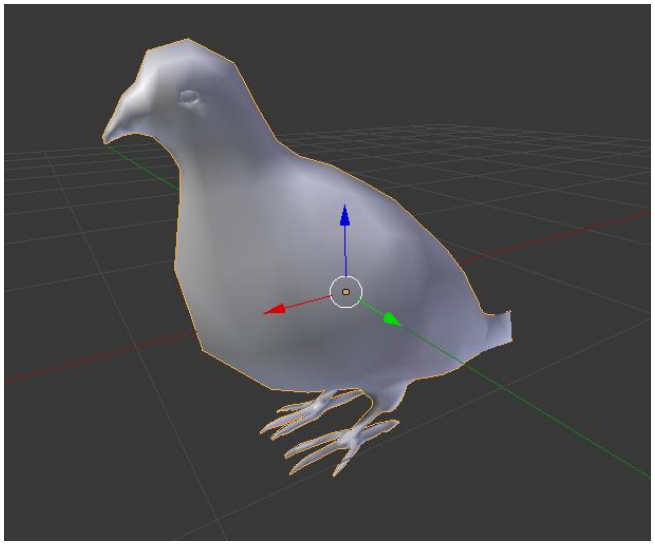
Exercicio 2

Alinea a:



Antes das modificações serem iniciadas fomos buscar o modelo do pombo que foi utilizado no nosso projeto do Blender.

Figura 13 - Modelo inicial do pombo



A primeira modificação que foi feita foi a redução do numero de faces, para tal ser feito foi aplicado o modificador Decimate com a opção de Un-subdivide. Escolhemos aplicar este modificador com 2 iterações, porque a partir da terceira iteração o aspeto do modelo começava a ficar deformado.

Figura 14 - Decimate aplicado

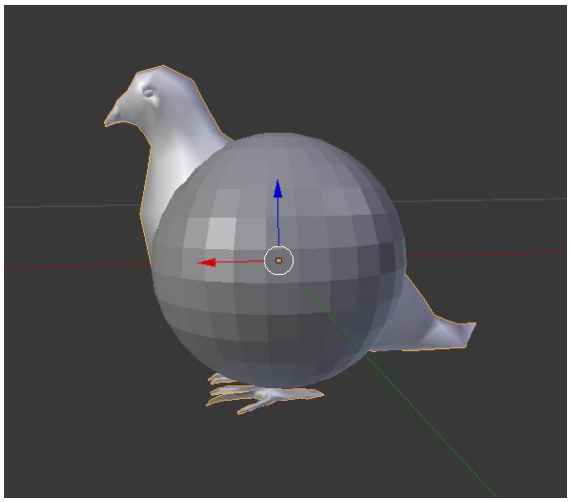


Figura 15 - Esfera usada como guia na redução de tamanho do pombo

Depois procedemos á redução das dimensões do nosso modelo. No enunciado estava a ser pedido que o pombo não ocupasse um espaço superior a uma esfera de 0.01 unidades de diâmetro. Então decidimos criar uma esfera com 0.01 unidades de diâmetro para nos guiarmos na redução do tamanho do pombo (a esfera foi apagada no final). No Blender usamos a opção de Scale para a redução até o pombo não sobressair a esfera.

No final o modelo foi convertido para o formato .Json, usando as opções corretas no Blender para gerar os ficheiros .obj e .mtl e o script obj_parser.py para gerar o .json pretendido, e depois o modelo foi colocado nos ficheiros do trabalho, também decidimos pintar o pombo de amarelo para ser possivel estar destacado do edificio.

Alinea b:

Para carregar o modelo na cena original foi adicionada uma linha de código á função load() para adicionar o pombo. Para colocar o pombo no topo do torreão no ponto [0.1; 2; 0.1] do cenário as coordenadas foram já definidas no Blender no ponto [0.09; 0.1; 2] para dar o aspeto que o pombo está pousado no torreão. Os sistemas de coordenadas do Blender e do WebGL são diferentes, no Blender a altura é representada pelo eixo z enquanto no WebGL é representada pelo eixo y, se o sistema de coordenadas do Blender fosse igual ao do WebGL teriamos definido a localização do pombo no ponto [0.09, 2, 0.1]. Inicialmente tinhamos definido a posição do pombo no eixo do x no ponto 0.1, mas na visualização da cena ele encontrava-se a flutuar ao lado do torreão, então corrigimos a coordenada x para 0.09.

```
function load(){
    //Load the office building
    Scene.loadObjectByParts('models/geometry/Building/part','Office',758);
    //Load the ground
    Scene.loadObject('models/geometry/Building/plane.json','Plane');
    Scene.loadObject('models/geometry/Building/obj.json','Object');
    //Load the pigeon
    Scene.loadObject('models/geometry/pombo_CG19.json','pombo_CG19');
}
```

Figura 16 - Função load alterada para adicionar o pombo.

Alinea c:

```
camera = new Camera(CAMERA_ORBIT_TYPE);
camera.goHome([0,2,2]);
camera.hookRenderer = draw;
```

```
var fovy = 7;
function updateTransforms(){
    if (projectionMode == PROJ_PERSPECTIVE){
        mat4.perspective(fovy, c_width / c_height, 1, 400.0, pMatrix);
    }
    else{
        cw = c_width/250;
        ch = c_height/250;
        mat4.ortho(-cw, cw, -ch, ch, -10.0, 10.0, pMatrix);
    }
}
```

Figura 17 - Definições da câmara e Field of view alterados

Para apresentar o pombo primeiro alteramos o tipo de câmara para orbitar o objeto quando a movemos com o rato, alteramos a posição inicial da câmara e diminuimos o field of view (a variável fovy) para ampliar a imagem no pombo.

No final movemos a câmara com o rato para tirarmos alguns screenshots para apresentar os detalhes do pombo pousado após estas alterações.

Figura 18 - Pombo pousado 1

Figura 19 - Pombo pousado 2

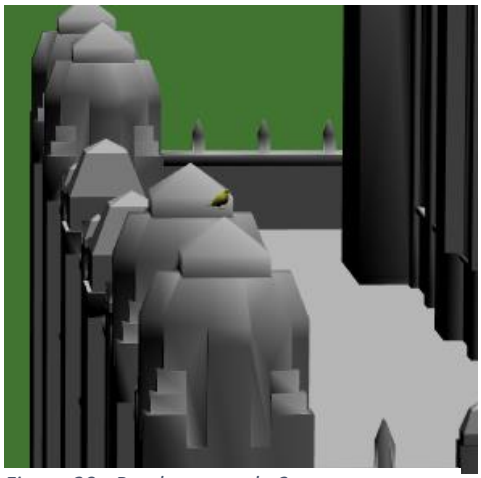


Figura 20 - Pombo pousado 3

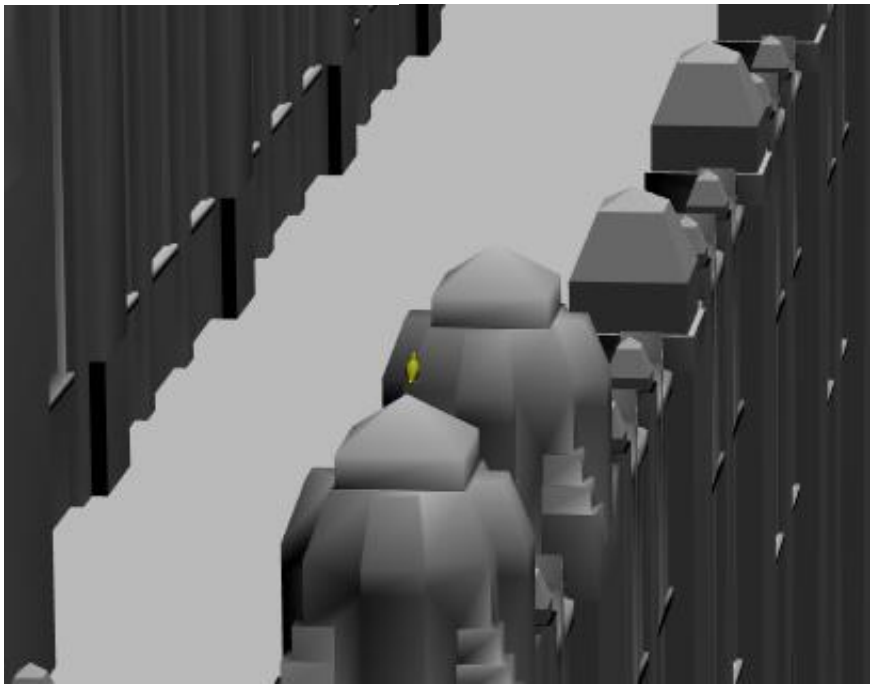
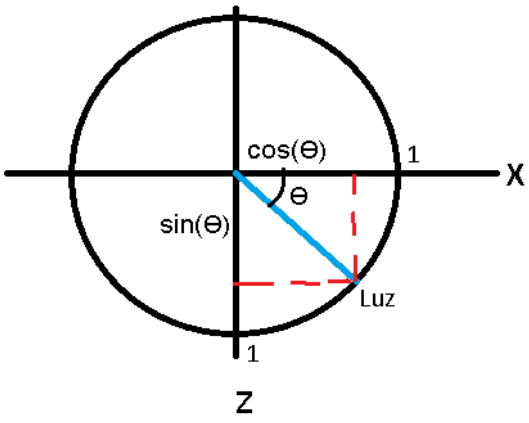


Figura 21 - Pombo pousado 4

Exercicio 3

Alinea a:

Nesta alinea fizemos uma função á parte, que vai ser chamada na função draw(), para movimentar a fonte de luz em torno do edificio.
A ideia da função é atualizar o ângulo da luz e a sua posição dependendo deste ângulo. Foi implementada a função renderLoop(), que será chamada na função de entrada da aplicação WebGL, para criar um ciclo para a função draw() e consequentemente para a função que foi criada para a luz, neste ciclo o valor do angulo vai sendo incrementado para fazer a fonte de luz movimentar-se em torno do edifício.



Para um ângulo Θ, a cada iteração a coordenada do X da luz vai para cos(Θ) e a coordenada do Z vai para sin(Θ), para este movimento não é necessário alterar o valor do Y, visto que a luz vai girar em torno do eixo Y, então a coordenada do Y vai se manter sempre constante, a uma altura de 2. Como as funções de seno e cosseno têm um dominio [-1 ;1], com os valores que atribuimos, a fonte de luz vai andar á volta do eixo do Y a uma distância de 1, e isso faria com a luz atravessasse o edificio devido ás suas dimensões, então para cada iteração multiplicamos o X e o Z por 3 para se encontrar a uma distância de 3 em relação ao eixo Y.

LINK PARA A ANIMAÇÃO: https://youtu.be/00aYvcEICUU

```
/**
 * Função que movimenta a fonte de luz em torno do edificio
 */
    var lightAngle = 0;
function animateLight(){
    gl.uniform3fv(prg.uLightPosition,[Math.cos(lightAngle)*3, 2, Math.sin(lightAngle)*3]);
    lightAngle+=0.1;
}
```

Figura 22 - Função de animação da fonte de luz

Alinea b:

Para esta alinea também foi criada uma função á parte que também é chamada na função draw(), e que irá entrar em ciclo pela função renderLoop(). Para a rotação da câmara foram feitas operações de translação, rotação e translação novamente á matriz da câmara para criar a rotação em torno do edifício. Para atualizar este ângulo foi implementada a função animate() para tal. Esta função vai ser chamada no renderLoop().

$$T(-X,-Y,-Z) \times Ry(\Theta) \times T(X,Y,Z)$$

Para fazer a câmara subir e descer, foi inicializada uma variável que vai sendo incrementada e o seu seno vai sendo calculado a cada iteração. Se o seno desta variável for positiva é feita uma translação para subir, e se for negativa é feita uma translação para descer.

LINK PARA A ANIMAÇÃO: https://youtu.be/peoyMhDUibQ

```
/**
 * Função que dá movimento á camera
 */
    var roTangle = 0;
    var transValue = 0;
function animateCamera(){
    mat4.translate(camera.matrix,[0,-1,-8]);
    mat4.rotate(camera.matrix, roTangle * Math.PI / 180, [0, 1, 0]);
    mat4.translate(camera.matrix,[0,1,8]);

    if(Math.sin(transValue)> 0){
        mat4.translate(camera.matrix,[0,  0.02, 0]);
     }else if(Math.sin(transValue)< 0){
        mat4.translate(camera.matrix,[0, -0.02, 0]);
     }
    transValue+=0.05;
    mat4.set(camera.matrix, nMatrix);
    mat4.inverse(nMatrix);
    mat4.transpose(nMatrix);

}
```

Figura 23 - Função de animação da câmara

Alinea c:

Nesta alinea apenas foram chamadas as duas funções anteriores que foram criadas na função draw() em simultâneo para criar a animação final.

LINK PARA A ANIMAÇÃO: https://youtu.be/8ONNBOsKIBg

```
function draw() {
    gl.viewport(0, 0, c_width, c_height);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    try{

        updateTransforms();
        animateCamera(); //----Alinea b----
        setMatrixUniforms();
        animateLight(); //----Alinea a----

        gl.uniform1i(prg.uUpdateLight,updateLightPosition);
```

Figura 24 - Função draw editada para animação simultânea da luz e da câmara

```
/**
* Updates the angle of rotation by a little bit each time
*/
function animate() {
    var timeNow = new Date().getTime();
    if (lastTime != 0) {
        var elapsed = timeNow - lastTime;
        roTangle = (90 * elapsed) / 1000.0;
    }
    lastTime = timeNow;
}

/**
* Render Loop
*/
function renderLoop() {
    requestAnimFrame(renderLoop);
    draw();
    animate();
}

/**
* Entry point. This function is invoked when the page is loaded
*/
var app = null;
function runWebGLApp() {
    app = new WebGLApp("canvas-element-id")
    app.configureGLHook = configure;
    app.loadSceneHook   = load;
    app.drawSceneHook   = draw;
    app.run();
    renderLoop();
}
```

Figura 25 - animate() e renderLoop() implementado e runWebGLApp() editado