COMPUTAÇÃO GRÁFICA

Relatório Projeto 2

Engenharia Informática – 2018/2019

Grupo 88

Diogo Nogueira, Nº 49435

Filipe Capela, Nº 50296

Pedro Almeida, Nº 46401

Índice

Diogo Nogueira 49435
Filipe Capela 50296
Pedro Almeida 46401

1- Modificações na cena original

O ficheiro Castle.html correspondente à primeira alínea, quando aberto apresenta a câmera vista a partir da frente do castelo, a um nível a partir do chão mais elevado, com uma elevação negativa de 45º, e com a fonte de luz a ser colocada em frente às portas do castelo. Isto para ser possível demonstrar algumas das alterações necessárias para a realização desta alínea (nomeadamente mudanças de posição da câmera e fonte de luz), e ser possível observar o cubo e a pirâmide que inserimos no interior do castelo.

A – Criação de Novos Objetos

Para adicionar os dois novos objetos (um cubo e uma pirâmide) na nossa cena utilizamos o Blender para os criar. De seguida, exportámos cada um dos objetos para formato .obj com alguns parâmetros ativados, tal como aprendemos nas aulas práticas (Apply Modifiers, Write Materials, Triangulate Faces, Objects as OBJ Objects e Material Groups). Para converter os ficheiros gerados pelo Blender em JSON, utilizámos o ficheiro obj_parser.py fornecido correndo no terminal *"python obj_parser.py piramide.obj piramide.mtl"* para a pirâmide e *"python obj_parser.py cubo.obj cubo.mtl*" para o cubo. Após as conversões, inserimos os objetos na cena através da função load() (Fig. 5) e alterámos um pouco os ficheiros JSON de cada um deles, mudando o posicionamento dos vértices para que estes não ficassem sobrepostos e a cor dos objetos para que sejam mais fáceis de distinguir. Foi alterada a posição inicial da câmera e a sua elevação de acordo com os parâmetros da Fig.5, de modo a conseguirmos ver os objetos no interior do castelo.

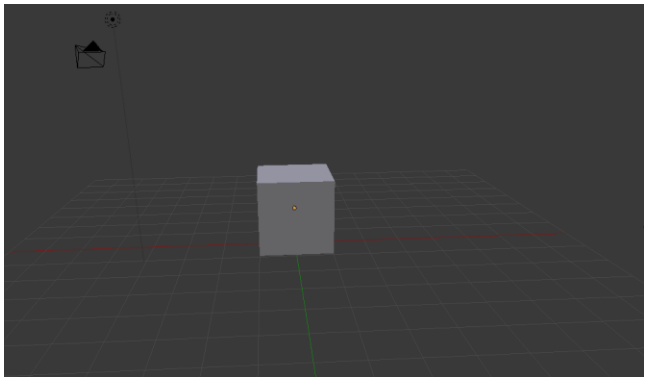

Fig. 1 - Cubo Blender

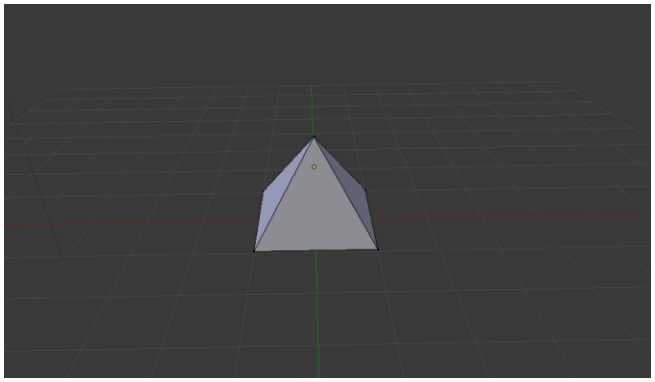

Fig.2 – Pirâmide Blender

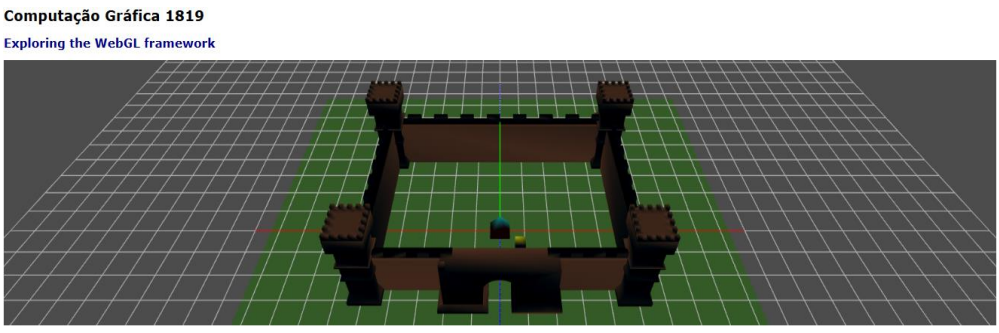

Fig. 3 - Cena WebGL com os objetos inseridos

Grupo 88

Filipe Capela 50296
Pedro Almeida 46401

```
/**
*  Configures the gl context
*/
var camera = null;
var interactor = null;

function configure(){

    gl.clearColor(0.3,0.3,0.3, 1.0);
    gl.clearDepth(100.0);
    gl.enable(gl.DEPTH_TEST);
    gl.depthFunc(gl.LEQUAL);

    //Creates and sets up the camera location
    camera = new Camera(CAMERA_TRACKING_TYPE);
    camera.goHome([0,70,60]);
    camera.setElevation(-45.0);
    camera.hookRenderer = draw;

    //Creates and sets up the mouse and keyboard interactor
    var canvas = document.getElementById('canvas-element-id');
    interactor = new CameraInteractor(camera, canvas);

    //Update lights for this example
    gl.uniform4fv(prg.uLightAmbient,      [0.05,0.05,0.05,1.0]);
    gl.uniform3fv(prg.uLightPosition,     [0, 20, 0]);
    gl.uniform4fv(prg.uLightDiffuse,      [0.7,0.7,0.7,1.0]);

    //init gui with camera settings
    //initGUIWithCameraSettings();

    //init transforms
    initTransforms();
}
```

Fig. 4 - Função configure() que configura a cena WebGL

```
function load(){
    Floor.build(100,5);
    Axis.build(50);
    Scene.addObject(Floor);
    Scene.addObject(Axis);
    Scene.loadObject('models/geometry/castle.json','Castle');
    Scene.loadObject('models/geometry/plane.json','Plane');
    Scene.loadObject('models/geometry/cubo.json','Cubo');
    Scene.loadObject('models/geometry/piramide.json','Piramide');
}
```

Fig. 5 - Função load() que carrega os objetos em formato JSON para a cena WebGL

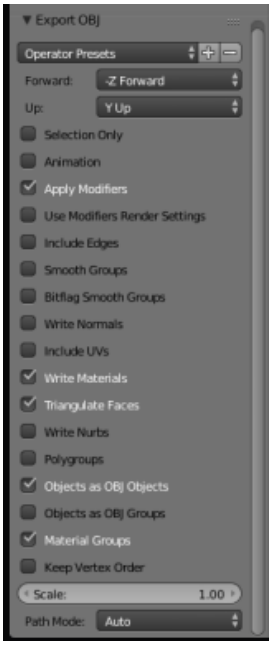

Fig. 6 Opções Export OBJ

B – Modificar a posição de fonte de luz

O posicionamento da fonte de luz foi alterado de forma a iluminar a zona do castelo de frente para a câmara, para isso, alterámos a função configure() do ficheiro castle.html modificando a linha de código em que chamamos a função uniform3fv(). Foram passados à função os parâmetros prg.uLightPosition e os valores das coordenadas a partir de onde a luz é irradiada, para que esta se situe na mesma posição da câmera (Fig. 1). Mudámos o valor de todos os eixos de forma a estes ficarem iguais aos passados como parâmetro na função goHome() sendo que agora a primeira coordenada passa do valor 60 para o valor 0, assim estamos a fazer variar o eixo dos X.

```
/**
*  Configures the gl context
*/
var camera = null;
var interactor = null;

function configure(){

    gl.clearColor(0.3,0.3,0.3, 1.0);
    gl.clearDepth(100.0);
    gl.enable(gl.DEPTH_TEST);
    gl.depthFunc(gl.LEQUAL);

    //Creates and sets up the camera location
    camera = new Camera(CAMERA_TRACKING_TYPE);
    camera.goHome([60,10,60]);
    camera.hookRenderer = draw;

    //Creates and sets up the mouse and keyboard interactor
    var canvas = document.getElementById('canvas-element-id');
    interactor = new CameraInteractor(camera, canvas);

    //Update lights for this example
    gl.uniform4fv(prg.uLightAmbient,      [0.05,0.05,0.05,1.0]);
    gl.uniform3fv(prg.uLightPosition,     [60, 10, 60]);
    gl.uniform4fv(prg.uLightDiffuse,      [0.7,0.7,0.7,1.0]);

    //init gui with camera settings
    //initGUIWithCameraSettings();

    //init transforms
    initTransforms();
}
```

Fig.1 - Função configure() com a posição da fonte de luz alterada

Fig. 2 - Posição fonte de luz antes

Fig.3 - Posição fonte de luz depois

C – Modificação da posição da câmara para vista frontal

Para mudar a posição da câmera para vista frontal alteramos de novo a função configure() que configura a cena WebGL. Alterámos a linha de código que define a posição inicial da câmera tal como mostra a Fig. 1. Modificámos o parâmetro da função goHome() sendo que agora a primeira coordenada passa do valor 60 para o valor 0, assim estamos a fazer variar o eixo dos X.

```
/**
*  Configures the gl context
*/
var camera = null;
var interactor = null;

function configure(){

    gl.clearColor(0.3,0.3,0.3, 1.0);
    gl.clearDepth(100.0);
    gl.enable(gl.DEPTH_TEST);
    gl.depthFunc(gl.LEQUAL);

    //Creates and sets up the camera location
    camera = new Camera(CAMERA_TRACKING_TYPE);
    camera.goHome([0,10,60]);
    camera.hookRenderer = draw;

    //Creates and sets up the mouse and keyboard interactor
    var canvas = document.getElementById('canvas-element-id');
    interactor = new CameraInteractor(camera, canvas);

    //Update lights for this example
    gl.uniform4fv(prg.uLightAmbient,      [0.05,0.05,0.05,1.0]);
    gl.uniform3fv(prg.uLightPosition,     [60, 10, 60]);
    gl.uniform4fv(prg.uLightDiffuse,      [0.7,0.7,0.7,1.0]);

    //init gui with camera settings
    //initGUIWithCameraSettings();

    //init transforms
    initTransforms();
}
```

Fig.1 - Função configure() com a posição inicial da câmera alterada

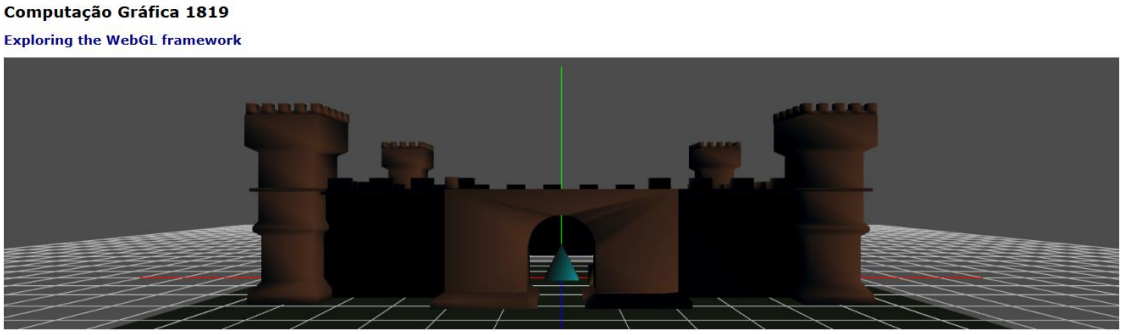

Fig.2 - Posição da câmera alterada para vista frontal

D – Modificação da posição da câmara para vista do topo

Para que a câmera ficasse em vista de topo mudámos novamente a função configure(). Alterámos outra vez a linha de código que define as coordenadas da posição inicial da câmera, sendo que neste caso, o valor das coordenadas X e Z da função goHome() foi alterado para 0 e o da coordenada Y para 100, com o intuito de obter uma visão mais afastada em altura, para que todo o castelo coubesse na janela de visualização. Adicionámos uma linha de código onde chamamos a função setElevation() de forma a alterar a elevação de câmera, de modo a conseguirmos ver o interior do castelo olhando diretamente de cima (daí termos colocado o parâmetro da função a -90).

```
/**
*  Configures the gl context
*/
var camera = null;
var interactor = null;

function configure(){

    gl.clearColor(0.3,0.3,0.3, 1.0);
    gl.clearDepth(100.0);
    gl.enable(gl.DEPTH_TEST);
    gl.depthFunc(gl.LEQUAL);

    //Creates and sets up the camera location
    camera = new Camera(CAMERA_TRACKING_TYPE);
    camera.goHome([0,100,0]);
    camera.setElevation(-90.0);
    camera.hookRenderer = draw;

    //Creates and sets up the mouse and keyboard interactor
    var canvas = document.getElementById('canvas-element-id');
    interactor = new CameraInteractor(camera, canvas);

    //Update lights for this example
    gl.uniform4fv(prg.uLightAmbient,      [0.05,0.05,0.05,1.0]);
    gl.uniform3fv(prg.uLightPosition,     [60, 10, 60]);
    gl.uniform4fv(prg.uLightDiffuse,      [0.7,0.7,0.7,1.0]);

    //init gui with camera settings
    //initGUIWithCameraSettings();

    //init transforms
    initTransforms();
}
```

Fig. 1 - Função configure() com posição inicial e elevação da câmera alteradas

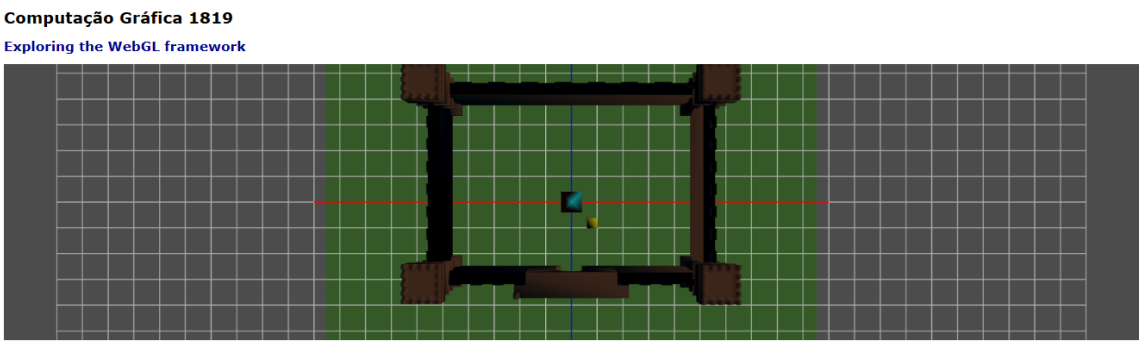

Fig. 2 - Vista de topo da cena

E – Imagens dos objetos com omissão do castelo

Nesta alínea removemos o modelo do castelo e deixámos só os dois objetos criados por nós na cena. Para isso, apenas removemos a linha de código da função load() que carregava o ficheiro JSON com o modelo do castelo.

```
/**
* Loads the scene
*/
function load(){
    Floor.build(100,5);
    Axis.build(50);
    Scene.addObject(Floor);
    Scene.addObject(Axis);
    //Scene.loadObject('models/geometry/castle.json','Castle');
    Scene.loadObject('models/geometry/plane.json','Plane');
    Scene.loadObject('models/geometry/cubo.json','Cubo');
    Scene.loadObject('models/geometry/piramide.json','Piramide');
}
```

Fig.1 - Função load() com a linha que importa o ficheiro castle.json comentada

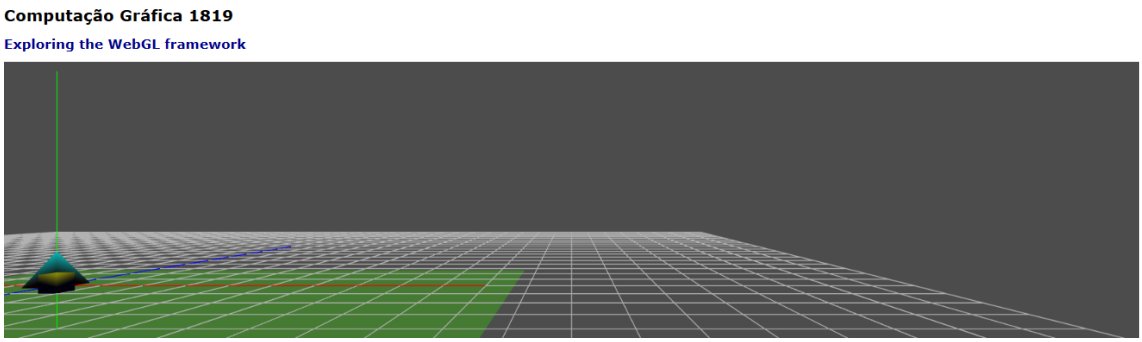

Fig.2 - Cena apenas com os objetos criados visíveis

2 – Novos modelos de objetos

A – Modificação de tenda e personagem

Começámos por inserir os objetos a exportar, neste caso a tenda e o personagem, num ficheiro à parte um de cada vez. Após isso retirámos as modificações feitas ao objeto de origem da seguinte forma:

– Para ambos os objetos retiramos a smoothness que estes tinham aplicada;

– Para a tenda isso foi suficiente para que a mesma estivesse pronta para fazer o export;

– Para a personagem foi necessário retirar também a animação que estava ligada ao objeto e também um modificador que tinha sido aplicado ao objeto. Depois do processo de preparação estar acabado fizemos export, a ambos os objetos, ao selecionar as opções necessárias:

– Apply Modifiers;

– Write Materials;

– Triangulate Faces;

– Objects as OBJ Objects;

– Material Groups.

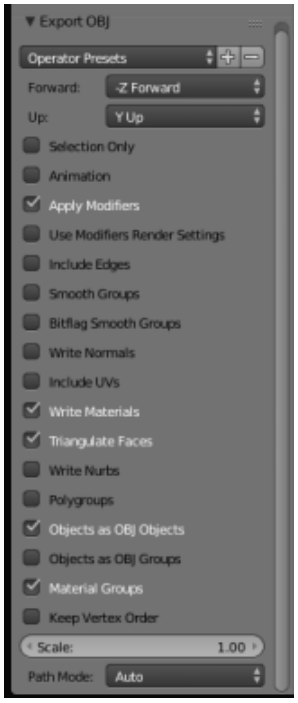

Fig. 1 - Opções de Export OBJ

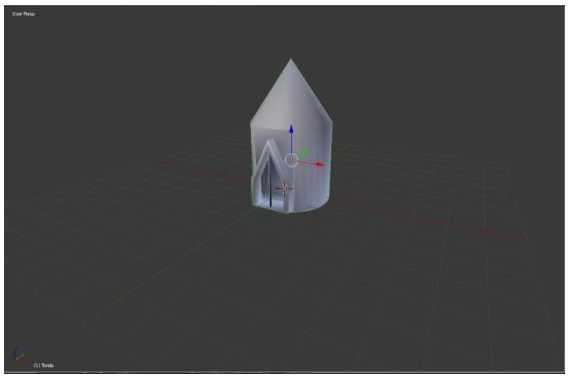

Fig. 2 - Tenda Antes

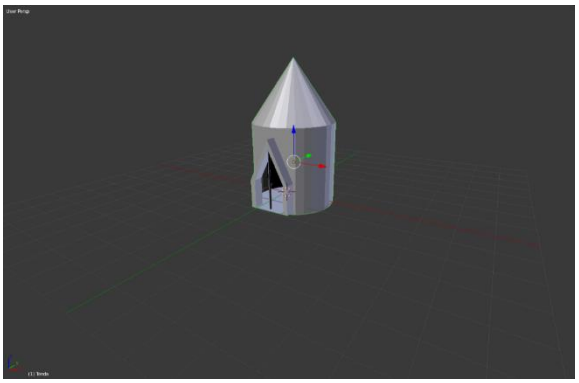

Fig.3 - Tenda Depois

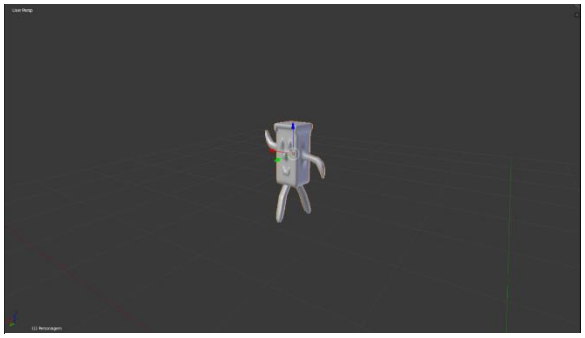

Fig. 4 - Personagem Antes

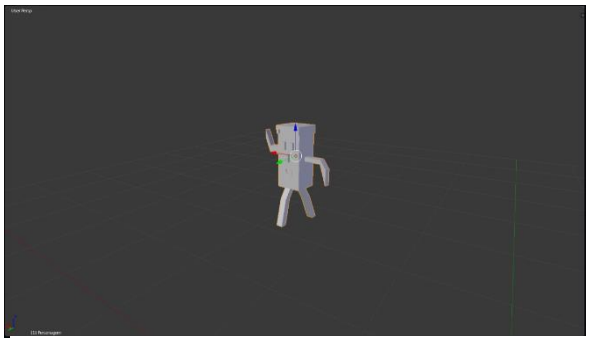

Fig. 5 - Personagem Depois

B – Substituição dos objetos Cubo e Pirâmide

Para a substituição dos objetos originais na cena, o cubo e a pirâmide, pelos objetos do grupo, a tenda e a personagem desenhados previamente no Blender, foi primeiro utilizado um processo de tentativa-erro para que as dimensões ficassem com valores aceitáveis. Para tal foi aplicado uma mudança de escala para x2 na tenda e uma mudança de escala para x3 na personagem de forma a que estes ficassem com as medidas desejadas por parte do grupo e então ficarem visíveis quando se colocasse a câmera de uma altura considerável.

Os objetos foram reposicionados tendo em conta os eixos ordenados, de modo a, dentro do castelo, se apresentarem lado a lado, no centro do castelo.

Após as alterações estarem completas e os exports estarem feitos, as mudanças no código para alteração dos objetos foram feitas:

```
/**
* Loads the scene
*/
function load(){
    Floor.build(100,5);
    Axis.build(50);
    Scene.addObject(Floor);
    Scene.addObject(Axis);
    Scene.loadObject('models/geometry/castle.json','Castle');
    Scene.loadObject('models/geometry/plane.json','Plane');
    Scene.loadObject('models/geometry/personagem.json','personagem');
    Scene.loadObject('models/geometry/tenda.json','tenda');
}
```

Fig.1 – Colocação das linhas de código na função load() onde se faz load dos objetos personagem e tenda

Verificámos o resultado para nos certificarmos de que estava tudo na sua devida posição e o resultado foi o esperado (De notar que foi alterada a posição da luz, de forma a termos uma melhor visão dos objetos e das suas cores).

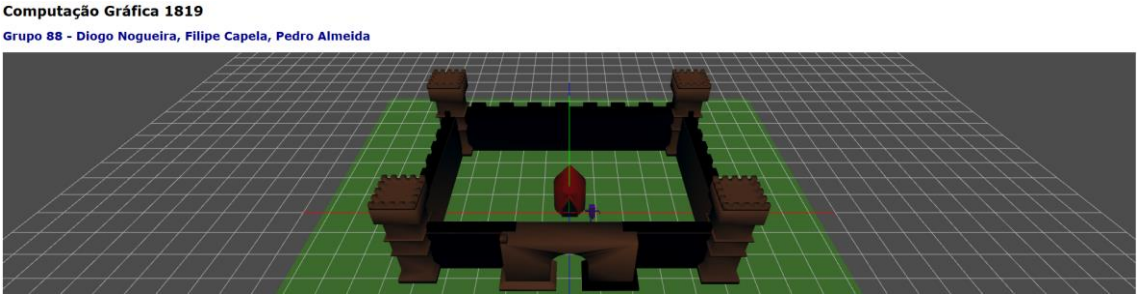

Fig.2 - Visão Final com os objetos no interior do castelo

3 – Movimentação da câmara e iluminação

A – Animação movimentando a fonte de luz em torno do castelo

Para realizarmos a animação da fonte de luz, tivemos de inserir no nosso trabalho duas funções, a animate() e a função renderLoop(). Estas funções, o que fazem, é, no caso da animate(), fazer variar a rotação em função do tempo dando-nos a nossa noção de frames por segundo. Neste caso da animação da luz decidimos optar por utilizar o valor 2500.0 de forma a termos uma animação mais acelerada. Na função renderLoop, ao utilizar a função requestAnimFrame(), estamos a fazer um loop usando a função draw, que delineia todas as matrizes de vértices dos nossos objetos. De modo a conferir a rotação efetivamente, a função draw() teve de ser alterada sendo adicionadas as porções de código da figura 2 e 3.

A porção de código na figura 2 é comum tanto para a rotação da câmera como da luz, sendo que aqui faz-se a rotação de forma circular usando a função rotate e se coloca a câmera a uma distância de 100 unidades de forma a conseguirmos visualizar o castelo na sua totalidade e os objetos dentro dele.

Na porção de código da figura 3 é onde efetivamente fazemos a variação da matriz responsável pelos focos de luz.

```
var lastTime = 0;
var angle = 0;
/**
* 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;
        angle += (90 * elapsed) / 2500.0;
    }
    lastTime = timeNow;
}

function renderLoop() {
    requestAnimFrame(renderLoop);
    draw();
animate();
}
```

Fig. 1- Inserção das funções animate() e renderLoop(), fazendo variar o valor de angle

```
        mat4.identity(mvMatrix);
        mat4.translate(mvMatrix, [0.0, 0.0, -100]); //Sets the camera to a reasonable distance to view the part
        mat4.rotate(mvMatrix, 30*Math.PI/180, [1,0,0]);
        mat4.rotate(mvMatrix, angle*Math.PI/180, [0,1,0]);
```

Fig. 2 – Linhas de código que conferem rotação às matrizes

```
//Código responsável pela rotação da luz
mat4.set(mvMatrix, nMatrix);
mat4.inverse(nMatrix);
mat4.transpose(nMatrix);
```

Fig. 3 - Linhas de código responsáveis pela rotação exclusiva da câmera

B – Animação movimentando a câmara em torno do castelo

Para realizar a animação apenas da câmera, fez-se as mesmas alterações da alínea a), sendo estas a adição das funções animate() e renderLoop(), presentes na figura 1, e a inserção das linhas de código na função draw() de modo a criar o efeito de rotação para cada frame, presentes na figura 2.

Neste caso da animação da luz decidimos optar por utilizar o valor 5000.0 de forma a termos uma animação mais lenta, de forma a conseguirmos analisar bem a rotação da câmera e o facto de a luz estar fixa.

A diferença entre as duas animações está na matriz à qual se passa o efeito de rotação, sendo este processo feito nas linhas de código presentes na figura 3. Deste modo temos a rotação da câmera apenas, e a luz fica numa posição fixa.

```
var lastTime = 0;
var angle = 0;
/**
* 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;
        angle += (90 * elapsed) / 5000.0;
    }
    lastTime = timeNow;
}

function renderLoop() {
    requestAnimFrame(renderLoop);
    draw();
animate();
}
```

Fig. 1 - Variação do valor de velocidade de rotação dos objetos

```
mat4.identity(mvMatrix);
mat4.translate(mvMatrix, [0.0, 0.0, -100]); //Sets the camera to a reasonable distance to view the part
mat4.rotate(mvMatrix, 30*Math.PI/180, [1,0,0]);
mat4.rotate(mvMatrix, angle*Math.PI/180, [0,1,0]);
```

Fig. 2 – Linhas de código que conferem rotação às matrizes

```
//Código responsável pela rotação da camera

gl.uniformMatrix4fv(prg.uMVMatrix, false, mvMatrix);
gl.uniformMatrix4fv(prg.uPMatrix, false, pMatrix);
```

Fig. 3 - Linhas de código responsáveis pela rotação exclusiva da câmera

C – Incorporar os dois movimentos em uma animação

Para incorporar os dois movimentos, quer o da câmera quer o da luz, o que fizemos foi juntar as linhas de código responsáveis pela rotação da câmera, e as linhas de código responsáveis pela rotação da luz (Figura 1). A velocidade de rotação foi alterada na função animate() para o valor de 5000.0 (Figura 2), e assim conseguirmos observar que conforme a câmera se vai movimentando, a luz como que segue este movimento, à mesma velocidade. Portanto, o foco de luz encontra-se na posição da nossa câmera ao longo da animação, conseguindo assim ter os dois movimentos numa animação. Desta forma é-nos possível ver o castelo, a tenda e a personagem através de diferentes ângulos e diferentes ângulos de incidência de luz.

```
mat4.identity(mvMatrix);
mat4.translate(mvMatrix, [0.0, 0.0, -100]); //Sets the camera to a reasonable distance to view the part
mat4.rotate(mvMatrix, 30*Math.PI/180, [1,0,0]);
mat4.rotate(mvMatrix, angle*Math.PI/180, [0,1,0]);

//Código responsável pela rotação da camera
gl.uniformMatrix4fv(prg.uMVMatrix, false, mvMatrix);
gl.uniformMatrix4fv(prg.uPMatrix, false, pMatrix);

//Código responsável pela rotação da luz
mat4.set(mvMatrix, nMatrix);
mat4.inverse(nMatrix);
mat4.transpose(nMatrix);
```

Fig.1 - Linhas de código responsáveis pelas rotações da luz e câmera

```
var lastTime = 0;
var angle = 0;
/**
* 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;
        angle += (90 * elapsed) / 5000.0;
    }
    lastTime = timeNow;
}

function renderLoop() {
    requestAnimFrame(renderLoop);
    draw();
animate();
}
```

Fig.2 - Variação do valor de velocidade de rotação dos objetos

Links das Animações

De modo a ser possível visualizar cada uma das animações, tanto das alíneas 3a), 3b) e 3c), seguem os links de uma pasta onde estão colocados os 3 ficheiros.

Pasta do Google Drive com as Animações:
https://drive.google.com/drive/folders/1_DXC4iTDTbhkmBjz9Zokhw1V_CwbUqVn?usp=sharing