Sistemas Gráficos Interativos
Aulas Práticas

M.EIC – 2023/2024

Trabalho Prático 1
Desenv. de uma aplicação introdutória em Three.js

1. Introdução

Atualmente é possível gerar gráficos interativos 3D em browsers web utilizando para o efeito algumas tecnologias de grande divulgação. Uma destas tecnologias é a WebGL; assenta diretamente sobre a OpenGL (OpenGL ES 2.0/3.0), apresentando-se como uma tecnologia de baixo nível. Uma outra tecnologia, de nível superior, é a Three.js (https://Threejs.org/); tal como a WebGL é programada em JavaScript e é sobre ela que incidirá grande parte do trabalho relativo à unidade curricular.

Neste contexto, com recurso a Three.js e a JavaScript, serão desenvolvidas aplicações gráficas que permitirão o estudo, análise e implementação de conceitos de Computação Gráfica, alguns avançados, e que podem ser corridas nos web browsers recentes.

Para o arquivo e controlo de versões dos trabalhos práticos, será utilizado o sistema de Gitlab da FEUP (git.fe.up.pt). A página do Moodle contém informações detalhadas sobre a configuração inicial deste sistema, que devem ser tidas em atenção antes de prosseguir.

Objetivos do Trabalho Prático TP1

Este trabalho foca-se no desenho de primitivas, na sua organização e na implementação de uma interface que permite controlar algumas das suas propriedades. A declaração e uso de transformações geométricas é um aspecto importante, nomeadamente para a produção de geometria complexa.

Os principais objetivos do presente trabalho são, assim, instalar, explorar e iniciar a utilização da tecnologia Three.js, assim como estudar e expandir os exemplos de base fornecidos. Os temas principais a abordar são:

- Estudo da arquitetura de uma aplicação em Three.js
- Criar objetos simples (meshes, a partir de primitivas);
- Criar grupos (agregações de outros grupos ou meshes)
- Afetar transformações geométricas a objetos/grupos existentes;
- Afetar propriedades de visualização aos objetos existentes;
- Criar e utilizar uma interface gráfica (GUI) para controlar aspectos da cena e os seus objetos.

2. Código de base do exercício

Tome o código base fornecido no projeto git correspondente ao seu grupo de trabalho. Como pode observar na Figura 1, a cena 3D resultante compõe-se de:

- Três cones coloridos representando um sistema de eixos ***xyz***;
- Um cubo centrado e localizado na parte positiva do eixo dos ***yy***;
- Um plano centrado na origem dos eixos;
- Uma fonte de luz pontual (não visível).

O cubo e o plano constituem-se de materiais distintos.

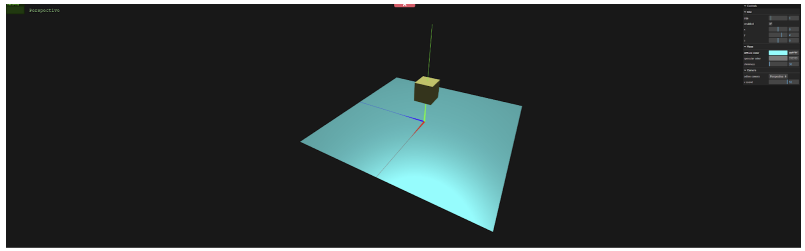

Figura 1: Ilustração do resultado visual correspondente ao código base fornecido.

No canto superior direito existe ainda uma área onde se encontra-se implementada uma interface com a qual é possível ativar ou parametrizar alguns aspectos da cena. Com o rato, é ainda possível:

Botão esquerdo - rodar a cena em torno do ponto origem das coordenadas;
Roda de scroll - aproximar/afastar (em alternativa: CTRL + Botão esquerdo);
Botão direito - "deslizar" a câmara lateralmente (pan).

O código de base fornecido para o exercício está agrupado em vários ficheiros que descrevem classes e instâncias (objetos) interligadas. Os ficheiros são, designadamente:

- **index.html**: ponto de arranque da aplicação; este ficheiro contém a estrutura HTML, CSS e importa as bibliotecas JS necessárias à execução da aplicação.
- **main.js**: ponto de arranque de Javascript, invocado como módulo em index.html; constrói e inicializa os principais objetos da aplicação designadamente:
  - **MyApp.js**: Instância da aplicação;
  - **MyContents.js**: Instância dos conteúdos que compõem a cena 3D;
  - **MyAxis.js**: efetua a representação 3D do sistema de eixos, suportado num *helper* e em 3 instâncias da primitiva cone. Uma instância da classe MyAxis é criada com a construção da cena em MyContents.js.
  - **MyGuiInterface.js**: Instância da interface gráfica 2D (MyGuiInterface.js)

  NOTA: alguns destes objetos são interligados, uma vez que o objeto de conteúdos e a interface gráfica 2D necessitam de conhecer o objeto da aplicação e vice-versa.

Depois da construção da aplicação, o ciclo principal de display é corrido repetidamente através da chamada ao método *render()*, previsto em **MyApp**.

A classe **MyGuiInterface** implementa a interface gráfica 2D onde se apresentam inputs de texto, checkboxes, sliders, menus drop-down, entre outros elementos que podem ser utilizados para interagir com a cena criada; mais informação sobre a utilização dos elementos da

biblioteca **dat.GUI** está disponível em: https://github.com/dataarts/dat.gui/blob/master/API.md

Leia com atenção os comentários disponíveis no código pois fornecem informação importante sobre o seu funcionamento e utilização.

3. Notas ao repositório Git para os projetos

O código correspondente a este trabalho, desenvolvido ao longo das aulas práticas, deve ser mantido num repositório online, alojado no Gitlab da FEUP. Os estudantes são organizados em grupos de dois elementos. Na primeira aula, a cada grupo será atribuído um repositório próprio no Gitlab que já contém o código de base deste trabalho, descrito anteriormente.
Mais informações sobre a utilização do Git e do Gitlab da FEUP, assim como sobre a preparação do ambiente de trabalho estão disponíveis na página Moodle.

4. Componentes do trabalho

As componentes do trabalho, a estender ao longo de quatro semanas, são descritas de seguida.

4.1. Introdução ao Three.js

Nas secções seguintes enunciam-se algumas ações a realizar sobre o código exemplo disponibilizado na página moodle que ajudarão a perceber alguns dos princípios básicos da tecnologia Three.js.

4.1.1. Contexto

O código fornecido prevê a construção de um WebGLRenderer e respetiva associação a um elemento HTML com área (por exemplo uma DIV) onde o resultado visual gerado por Three.js é apresentado.

4.1.2. Câmaras

O código fornecido prevê a utilização, por defeito, de uma câmara do tipo "Perspective", embora seja possível, através da interface gráfica, alterar para "Orthographic" com três pontos de vista possíveis.

Identifique nos ficheiros relacionado com a câmara e:

- Experimente o efeito de alterar os parâmetros de
  - `PerspectiveCamera( 75, aspect, 0.1, 1000 )`
- Idem com `position.set(10,10,3)`
- Analise com detalhe o código de criação de cada uma das 4 câmaras ortogonais, nomeadamente a definição dos limites *left*, *right*, *top*, ... e *far*, com base em `frustumSize` e a sua utilização em `OrthographicCamera()`.
- Idem para `*.up, *.position, *.lookAt()`
- Na interface, identifique as funcionalidades de interação utilizadas e o seu funcionamento que permite a alteração do tipo de câmara e de algum parâmetro.
- ❖ Crie uma nova câmara "`Perspective`", com um diferente ponto de vista inicial.
- ❖ Crie duas novas câmaras ortogonais `right, back`.
- ❖ Adicione, na interface, no menu de seleção de câmara, as novas câmaras criadas nos pontos anteriores.

4.1.3. Transformações Geométricas

A tecnologia Three.js utiliza matrizes para representar transformações geométricas 3D: translação (posição), rotação e escalamento. Todo o objeto 3D em Three.js (que herda da classe Object3D) contém um atributo *matrix* que representa a posição (`position`), a rotação e o escalamento do objeto.
Este atributo - *matrix* - armazena as transformações do objeto relativamente ao seu ascendente na cena. Para obter as transformações absolutas de um objeto com respeito ao sistema de coordenadas mundo é necessário observar o atributo `matrixWorld`.

A cena fornecida contém dois objetos, um plano horizontal e um cubo (*box)*. Este último aparece centrado no eixo dos ***yy*** e deslocado para cima.
Identifique no construtor da *box* a instrução que define `position` e a instrução que define `rotate`.

- Experimente outras deslocações, nomeadamente em outras direções
- Qual é o efeito visual da rotação efetuada? Para perceber melhor o seu efeito, altere o ângulo de rotação para -20º e depois para 20º
- Altere a ordem de execução:
  - Rotação em ***x*** seguida de translação em ***z***
  - Translação em ***z*** seguida de rotação em ***x*** (ordem inversa) Encontra alguma diferença?
- Implemente um escalamento (1,1,2) e reponha rotação=0, translação=(0,2,0)
- Altere a ordem de execução:
  - Escalamento seguido de translação
  - Translação seguida de escalamento (ordem inversa) Encontra alguma diferença?

4.1.4. Primitivas Geométricas (*Geometries*)

A tecnologia Three.js disponibiliza uma grande variedade de geometria pré-definida, a maior parte da qual geometria 3D, que designamos por primitivas geométricas..

Nesta parte do trabalho, o que lhe é solicitado é que explore diferentes primitivas geométricas, incluindo-as no seu código e testando-as nas suas várias vertentes (p.e. dimensões, ângulos, número de stacks, número de slices, etc. Sem prejuízo de outros objetos, explore os seguintes:

- Plane
- Circle
- Box
- Sphere
  - Completa
  - Parcial (parte de esfera)
- Cylinder (completo e parcial)
- Cone (completo e parcial)
- Polyhedron (sem subdivisões)

4.1.5. Modelação de uma cena 3D

Modele uma cena contendo o seguinte conjunto de objetos:

- Chão (pode aproveitar o plano existente)
- Quatro paredes (planos)
- Uma mesa centrada na cena, constituída por:
  - Tampo (paralelepípedo)
  - Pernas (cilindros)
- Um prato sobre a mesa (cilindro)
- Um bolo circular sem uma fatia, sobre o prato (parte de cilindro)

- Uma vela ao centro do bolo (cilindro)
- Uma chama na vela (triângulo… cone…)

Usando de alguma criatividade, mas limitando-se às técnicas aqui apreendidas, acrescente outros objetos a seu gosto.

0.1. Iluminação

A tecnologia Three.js é bastante completa no que respeita a tipos de fontes de luz, variedade de materiais e modelos de iluminação que serão abordados seguidamente.
Para os próximos exercícios defina, no Git FEUP, um segundo *branch* do seu código; nesse novo *branch* criado, regresse ao estado correspondente ao código original que lhe foi fornecido (ou seja, ao 1º *commit*). No ficheiro `MyContents.js`, identifique as instruções relacionadas com a fonte de luz e com os materiais associados ao cubo e ao plano fornecidos.

0.1.1. Materiais e modelo de iluminação

Verifique, no código, que a fonte de luz é do tipo *PointLight*, caraterizada por emitir luz igualmente em todas as direções e que se encontra no ponto (0, 20, 0). Verifique também a declaração de um *handler* representativo (a fonte de luz é uma entidade invisível). Aconselha-se a consulta do manual no que respeita à definição do respetivo construtor.

Altere, no código (identifique, no manual da Three.js o que representa cada item):

```
this.diffusePlaneColor =  "rgb(128,128,128)"
this.specularPlaneColor = "rgb(128,128,128)"
this.planeShininess = 30
...
this.planeMaterial = new
THREE.MeshPhongMaterial({
        color: this.diffusePlaneColor,
        specular: this.specularPlaneColor,
        emissive: "#000000", shininess:
this.planeShininess})
...
const pointLight = new THREE.PointLight(
0xffffff, 500, 0);
pointLight.position.set( 0, 20, 0 );
...
const ambientLight = new THREE.AmbientLight(
0x6f6f6f );
```

NOTA: no código base poderá ser necessário corrigir o termo marcado a bold.

Para as questões seguintes use como referência o modelo de iluminação local de Phong, caraterizado pelas três componentes, ambiente, difusa e especular.

- Altere a posição da fonte de luz de para (0, 20, 0) para (0, -20, 0). Comente as alterações de iluminação encontradas, nomeadamente:
  - no plano
  - na face superior do cubo
  - nas faces laterais do cubo
- Altere a intensidade da fonte de luz para 5 e a sua posição para (0, 2, 0); repita os pontos anteriores.
- Faça
  - `this.diffusePlaneColor = "rgb(0,0,0)"`
  - Que componente(s) de iluminação encontra no plano?
- Faça também:
  - `this.planeShininess = 400`
  - Que diferenças encontra? (experimente outros valores…)
- Faça agora (note que há mais um parâmetro em `PointLight`):
  - `this.diffusePlaneColor = "rgb(128,128,128)"`
  - `this.specularPlaneColor = "rgb(0,0,0)"`
  - `this.planeShininess = 0`
  - `...`
  - `pointLight = new THREE.PointLight(0xffffff, 5, 0, 0);`
  - `pointLight.position.set( 0, 20, 0 );`
  - Veja e memorize o resultado
- Faça agora:
  - `pointLight = new THREE.PointLight(0xffffff, 5, 15, 0)`
  - Comente sobre as alterações observadas.
  - Reponha o penúltimo parâmetro em 0 (*intensity*) e faça variar o último parâmetro entre 0, 1 e 2 (*decay*); comente os resultados.

0.1.2. Fontes de Luz

Até ao momento temos vindo a usar uma fonte de iluminação ambiente e uma fonte de luz pontual, omnidirecional (`pointLight`). No entanto, a tecnologia Three.js disponibiliza outros tipos de fontes de luz que serão exploradas nas etapas seguintes.

Retome o código original que lhe foi fornecido (1º commit). Comente as linhas de código relativas à fonte de luz pontual que assim deixa de existir; aumente a intensidade da luz ambiente para o valor 4:

```
const ambientLight = new THREE.AmbientLight(
0x555555, 4 );
```

Como esperado, os dois objetos estão minimamente iluminados e sem influência da direção incidente da luz.

Fonte de Luz Direcional
De seguida vai ser declarada uma fonte de luz direcional; verifique a definição do respetivo construtor (consulte a documentação do Three.js).

- Comente o código correspondente à atual fonte de luz pontual (será substituída pelas seguintes).
- Crie uma instância de fonte de luz direccional, cor branca, intensidade 1 cd, localização (0,10,0), atribuída a uma variável local designada `light2`. Adicione um handler para a fonte de luz direccional recém-criada. Acrescente a fonte de luz e o respectivo handler à cena.
  - Verifique o efeito da alteração do segundo parâmetro do construtor de `DirectionalLight(...)`, relativo à intensidade e medido em candelas (cd).
  - Altere a posição da fonte de luz para `(5, 10, 2)`. Comente sobre a nova iluminação obtida.
  - Altere a posição da fonte de luz para `(-5, 10, -2)`. Comente sobre a nova iluminação obtida.
  - Consulte a documentação sobre o parâmetro `.target` desta classe e acrescente um target diferente do de *default*.

Fonte de Luz "Foco"
A fonte de luz do tipo "Foco" (`SpotLight`) emite segundo um cone de luz, a partir do vértice do cone. Para mais detalhes, consulte a documentação do Three.js e analise o respetivo *constructor*.

No código fornecido, junto ao código de pointLight que comentou anteriormente, acrescente uma fonte de luz do tipo `spotLight` (e respetivo *handler*):

a. Cor: branca
b. Intensidade: 15
c. Distância limite: 8
d. Ângulo de *spot*: 40º
e. Penumbra: 0
f. Decaimento: 0
g. Posição: (2,5,1)
h. *Target*: (1,0.1)

- Interprete com o detalhe possível o *handler* e o seu efeito na iluminação do plano.
- Compare, nas 3 faces iluminadas do cubo, as respectivas iluminações.

Acrescente agora, na interface 2D, uma secção para controlo desta fonte de luz, permitindo alterar as alíneas a) a f) da lista anterior, assim como a coordenada **y** da posição.
Altere o ângulo de spot para 35º

- Compare, nas 3 faces iluminadas do cubo, as respectivas iluminações.
- Verifique o efeito do cone de luz, também na face superior do cubo.

Altere o valor de penumbra para 0.2 (significado de 20%)

- Verifique o efeito desta alteração na iluminação do plano e do cubo
- Repita, agora com penumbra = 0.7
- Idem com penumbra = 1.0

Efetue a alteração de outros valores, na interface, e verifique o seu efeito.

Fonte de Luz "Area Rectangular"
Estude autonomamente a classe de luz planar: `RectAreaLight`

Interface com Fontes de Luz
Acrescente, na interface, um novo "folder" que permita controlar parâmetros/propriedades da fonte de luz `SpotLight`:

- Color
- Intensity
- Distance
- Spot Angle (em graus)
- Penumbra ratio
- Decay with distance
- Position X
- Position Y
- Target X
- Target Y

Exemplo:

- `const lightFolder = this.createFolder('Spot light')`
- `lightFolder.add(this.contents.spotLight, 'intensity', 0, 40).name("intensity (cd)");`
- `lightFolder.add(this.contents.spotLight, 'distance', 0, 20).name("distance");`

Nota: no caso do ângulo, o valor recolhido da interface é expresso em graus; no entanto, o sistema espera ângulos expressos em radianos… é por isso necessária uma função que efetue a conversão.

0.1.3. Texturas

As texturas constituem um componente muito importante em *software* de computação gráfica 3D pois permitem adicionar realismo visual aos objetos, sem aumentar a complexidade geométrica dos mesmos.

Uma textura de mapeamento é uma imagem 2D; os seus pixels tomam o nome de texels mas, independentemente da sua resolução, as texturas enquadram-se num sistema de eixos U e V (ou S e T…), sendo que o comprimento U e a largura V da imagem possuem valor 1.0.

Em Three.js, a textura é mapeada de forma que o seu canto inferior esquerdo (coordenadas UV = (0,0)) fica colocado no primeiro vértice do polígono em que se efetua o mapeamento; também, o seu bordo horizontal inferior segue a direção da primeira aresta do polígono. Estas regras podem no entanto ser alteradas através de transformações geométricas de texturas.

Na página moodle pode encontrar duas imagens para serem usadas como texturas. Coloque-as numa pasta "textures" juntamente com os ficheiros do código.

No código da classe MyContents, identifique a linha com o comentário:

```
        // plane related attributes
```

O bloco de linhas que se segue a essa linha deve ser substituído pelo código seguinte:

```
        // plane related attributes
            //texture
    this.planeTexture =
       new
THREE.TextureLoader().load('textures/feup_b.jpg');
    this.planeTexture.wrapS = THREE.RepeatWrapping;
    this.planeTexture.wrapT = THREE.RepeatWrapping;

            // material
    this.diffusePlaneColor =  "rgb(128,128,128)"
    this.specularPlaneColor = "rgb(0,0,0)"
    this.planeShininess = 0

            // relating texture and material:
                // two alternatives with different
results

                    // alternative 1
    this.planeMaterial = new THREE.MeshPhongMaterial({
            color: this.diffusePlaneColor,
            specular: this.specularPlaneColor,
            emissive: "#000000", shininess:
this.planeShininess,
            map: this.planeTexture })
                    // end of alternative 1

                    // alternative 2
        // this.planeMaterial = new
THREE.MeshLambertMaterial({
        //        map : this.planeTexture });
                    // end of alternative 2

let plane = new THREE.PlaneGeometry( 10, 10 );
```

Genericamente, esta sequência de instruções começa por carregar um ficheiro com uma imagem que será usada como textura em modo "repeat" em ambas as direções U e V (número de repetições ainda não definido…).
Depois define o material de base do objeto (*diffuse, specular, shininess*).
Finalmente define a forma de relacionar o material de base e a textura; o ThreeJS disponibiliza duas alternativas que estão presentes no código acima (só uma delas deve estar ativa, a segunda está comentada).

No código fornecido, na função `init()`, encontre a linha seguinte:

```
        // Create a Plane Mesh with basic material
```

O bloco de linhas que se segue a essa linha deve ser substituído pelo código seguinte:

```
        // Create a Plane Mesh with basic material
        let planeSizeU = 10;
        let planeSizeV = 7;
        let planeUVRate = planeSizeV / planeSizeU;

        let planeTextureUVRate = 3354 / 2385; // image
dimensions
        let planeTextureRepeatU = 1;
        let planeTextureRepeatV =
            planeTextureRepeatU * planeUVRate *
planeTextureUVRate;
        this.planeTexture.repeat.set(
            planeTextureRepeatU, planeTextureRepeatV
);
        this.planeTexture.rotation = 0;
        this.planeTexture.offset = new
THREE.Vector2(0,0);

        var plane = new THREE.PlaneGeometry(
planeSizeU, planeSizeV );
        this.planeMesh = new THREE.Mesh( plane,
this.planeMaterial );
        this.planeMesh.rotation.x = -Math.PI / 2;
        this.planeMesh.position.y = 0;
        this.app.scene.add( this.planeMesh );
```

Aqui, declaram-se as dimensões do objeto "plano" (10*7); fixa-se em "1" o número de repetições na direção ***U***; determina-se o número de repetições em ***V*** de forma a manter o aspeto comprimento/largura da textura no ecrã (atendendo à relação comprimento/largura em *texels* da textura e à relação comprimento/largura do objeto "plano"; definem-se valores para as transformações geométricas rotação e translação (*offset*) da textura. O restante do código já é conhecido: definição do objeto "plano", sua conversão em *mesh* e adição na cena.

Relação Materiais & Texturas

- Corra o programa (se necessário ajuste a fonte de luz) e verifique:
  - As cores da imagem do plano.
  - Altere a cor difusa do material de cinzento para vermelho:
    - `this.diffusePlaneColor = "rgb(128,0,0)"`
  - Comente sobre a mudança de cores da imagem no plano… parece ser evidente a "mistura" da cor do material com as cores da textura!
- Comente agora o código assinalado com "`Alternativa 1`" e descomente o código assinalado com "`Alternativa 2`".
  - Que consequência tem para a imagem? (cores…)

***Wrapping* de Texturas - Escalamento**

Existem três formas de efetuar "wrapping". Por este meio é possível controlar o tamanho da textura; de certa forma, permite-se assim efetuar a transformação geométrica *Scale* em texturas.

- Analise o resultado obtido, agora em termos de dimensões do polígono e da textura.
  - A imagem da textura aparece inteira no plano?
  - A proporção comprimento/largura da textura na imagem aparenta ser a mesma?
  - Altere as dimensões do plano para (10*3) e repita os pontos anteriores.
  - Interprete o resultado obtido com base na linha de código:
  - `let planeTextureRepeatV =`
  - `planeTextureRepeatU * planeUVRate * planeTextureUVRate;`
  - Altere agora:
  - `let planeSizeU = 10;`
  - `let planeSizeV = 3;`
  - O polígono ficou mais reduzido numa dimensão; repita os pontos anteriores.
- Reponha `planeSizeV = 7` e altere
  - `planeTextureRepeatU = 2.5`
  - Verifique a repetição da textura em ambas as direções
  - Verifique que, em cada réplica da imagem de textura, se mantém a proporção comprimento/largura. Justifique…
- Altere:

```
        this.planeTexture.wrapS =
THREE.ClampToEdgeWrapping;
        this.planeTexture.wrapT =
THREE.ClampToEdgeWrapping;
```

  - Interprete o resultado comparando com a situação anterior.
- Altere:

```
        this.planeTexture.wrapS =
THREE.MirroredRepeatWrapping. ;
        this.planeTexture.wrapT =
THREE.MirroredRepeatWrapping. ;
```

  - Interprete o resultado comparando com as situações anteriores.

***Offset* de Texturas - Translação**

Como se viu, o primeiro vértice do polígono fica usualmente (default) mapeado na coordenada UV=(0,0) da textura. No entanto, a utilização de valores de *offset* diferentes de zero podem alterar este facto.

- Reponha:

```
        let planeTextureRepeatU = 1;
        this.planeTexture.wrapS =
THREE.RepeatWrapping. ;
        this.planeTexture.wrapT =
THREE.RepeatWrapping. ;
```

- Altere o vetor na instrução:
  - `this.planeTexture.offset = new THREE.Vector2(0.2,0.1);`
  - O polígono deslocou-se 0.2 unidades para a direita; fica a sensação de que a imagem da textura se deslocou para a esquerda...
  - O polígono deslocou-se 0.1 unidades para cima; fica a sensação de que a imagem da textura se deslocou para baixo...
  - Em ambas as direções, não restam espaços vazios, pois são preenchidos em acordo com o modo `RepeatWrapping.`

Rotação de Texturas

A textura pode ser rodada em torno da origem das coordenadas ***UV***.

- Reponha:
  - `this.planeTexture.offset = new THREE.Vector2(0,0);`
- Altere:
  - `this.planeTexture.rotation = 30 * Math.PI / 180;`
  - Comente o resultado obtido; experimente com outros valores de rotação.
- Consulte a documentação e verifique que é possível considerar, como ponto de rotação, um outro ponto que não a origem das coordenadas ***UV***.

Cubo com textura

Efetue as alterações necessárias no seu código, de forma a que seja aplicada às faces do cubo uma nova textura, baseada na imagem feup_entry.jpg.

Interface com textura

Sugere-se que se acrescente, na interface, um novo "folder" que permita controlar parâmetros/propriedades da textura do plano:

- Wrapping mode U: (3 opções)
- Wrapping mode V: (3 opções)
- Repeat U: slider/number
- Repeat V: slider/number
- Offset U: slider/number
- Offset V: slider/number
- Rotation: slider/number

0.1.4. Melhoramentos na cena 3D

Introduza melhoramentos na sua cena 3D, fazendo uso de materiais, fontes de luz e texturas.

- Materiais difusos nas paredes da sala
- Material especular nas pernas da mesa
- Tampo da mesa com aspeto de madeira
- Dois quadros pendurados numa parede com as fotografias dos dois estudantes
- Uma paisagem numa janela em outra parede
- Um foco de luz a salientar o bolo sobre a mesa

Usando de criatividade, mas limitando-se às técnicas aqui apreendidas, acrescente outros objetos e facilidades a seu gosto.

==========

As restantes componentes do trabalho serão adicionadas oportunamente.

==========

5. Notas sobre a avaliação do trabalho

Composição dos Grupos: Os trabalhos devem ser realizados em trabalho de grupo de dois estudantes. Em caso de impossibilidade (por exemplo por falta de paridade numa turma), deve ser discutida com o docente a melhor alternativa.

Avaliação do Trabalho de Grupo: O código resultante do trabalho de grupo será apresentado e defendido em sala de aula, perante o docente respetivo. Ambos os estudantes do grupo devem estar presentes na sessão de avaliação (em caso de impossibilidade, devem acordar previamente uma alternativa com o docente das aulas práticas). A classificação atribuída aos dois estudantes pode ser diversa.

Tendo em atenção as funcionalidades enunciadas, serão considerados os seguintes critérios para efeitos de Avaliação do Trabalho de Grupo:

Estruturação e Documentação do código	TBD
— TBD —	TBD
— TBD —	TBD
— TBD —	TBD
— TBD —	TBD
— TBD —	TBD

De acordo com a formulação constante na ficha de unidade curricular, a avaliação deste trabalho conta para a classificação final com um peso:

80% * 25% = 20%

Datas Principais:

- Início: 18/09/2023
- Entrega: 16/10/2023
- Avaliação nas aulas: 17 - 20/10/2023

Plano de trabalhos sugerido:

- **Semana de 18/09:**
  - Introdução à tecnologia Three.js; Transf. Geométricas; Primitivas geométricas.
- **Semana de 25/09:**
  - Iluminação e Materiais; Texturas.
- **Semana de 02/10:** TBD
- **Semana de 09/10:** TBD