

# Computação Gráfica (MIEIC)

2019/2020

## ***Projeto Final***

v2.2 (2020/05/06)

### ***Objetivos***

- Aplicar os conhecimentos e técnicas adquiridas até à data
- Criação de uma cena complexa, com diferentes geometrias, materiais, texturas e luzes
- Exploração de shaders, interação, controlo por teclado e animação

### ***Descrição***

Pretende-se com este projeto a criação de uma cena que combine os diferentes elementos explorados nas aulas anteriores, acrescentando alguns novos elementos. Para este trabalho deve usar como base o código que é fornecido no Moodle, que corresponde a uma cena vazia. Terá posteriormente de adicionar alguns dos objetos criados anteriormente.

A cena será no final genericamente constituída por:

- Uma paisagem envolvente (*cube map*)
- Um veículo controlado pelo utilizador
- Um terreno
- Outros elementos

Os pontos seguintes descrevem as principais características dos diferentes elementos pretendidos. É dada alguma liberdade quanto à composição dos mesmos na cena, para que cada grupo possa criar a sua própria cena.

O enunciado está dividido em duas partes, com uma proposta de um plano de trabalhos no final de cada uma delas.

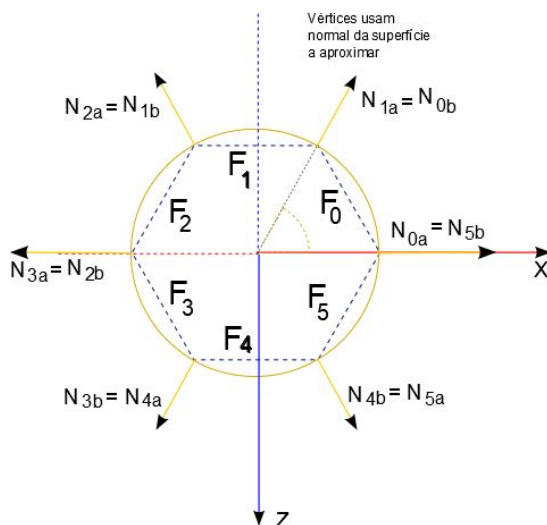
# Parte A

## 1. Criação de objetos de base

### 1.1. MyCylinder - Cilindro (sem topos)

Crie uma nova classe **MyCylinder** que aproxime um cilindro de raio de uma unidade, com um número variável de "lados" (**slices**), e altura de uma unidade em Y. A base do cilindro deve ser coincidente com o plano XZ e centrada na origem.

As normais de cada vértice devem ser definidas de forma a aproximar uma superfície curva, de acordo com o método de Smooth Shading de Gouraud, como demonstrado na **Figura 1**.



**Figura 1:** Ilustração das normais a atribuir a cada vértice no caso de um cilindro aproximado por seis lados.

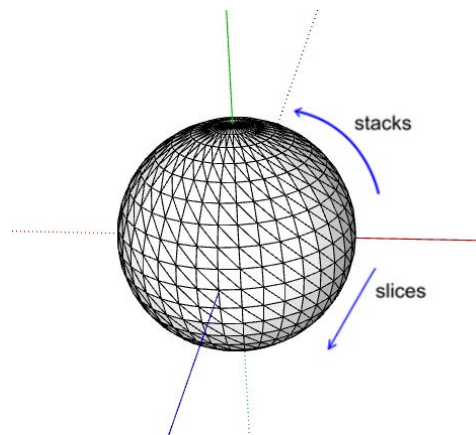
Para cada vértice, defina as coordenadas de textura de forma a mapear uma textura à volta do cilindro. Note que como a textura dá a volta ao cilindro poderá necessitar de repetir a primeira linha vertical de vértices pois estes são, simultaneamente, o início e o fim da textura.

### 1.2. MySphere - Esfera

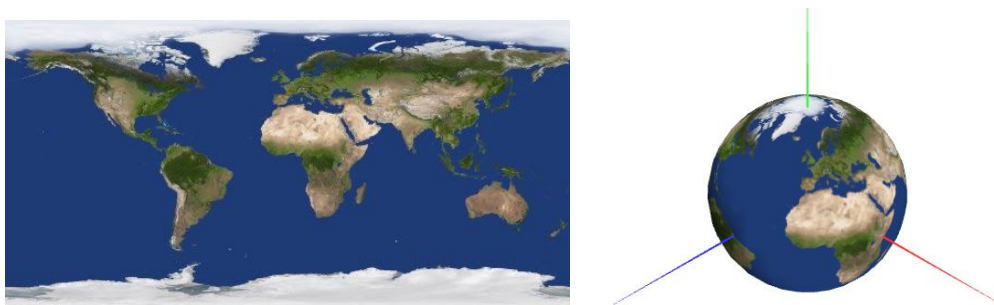
No código fornecido encontra uma classe MySphere correspondente a uma esfera com o centro na origem, com eixo central coincidente com o eixo Y e raio unitário.

A esfera tem um número variável de "lados" à volta do eixo Y (slices), e de "pilhas" (stacks), que corresponde ao número de divisões ao longo do eixo Y, desde o centro até aos "pólos" (ou seja, número de "fatias" de cada semi-esfera). A **Figura 2** tem uma representação visual da esfera.

Pretende-se que analise o código deste objeto, e acrescente o código necessário para gerar as coordenadas de textura para aplicar texturas à superfície da esfera, como demonstrado na **Figura 3**.



**Figura 2:** Imagem exemplo de uma esfera centrada na origem.



**Figura 3:** Exemplo de textura (disponível no código base) e sua aplicação numa esfera

### 1.3. Cube Map - Criação de um cubemap para o ambiente

Com este exercício pretende-se que crie um *cube map* que sirva como ambiente de fundo da cena. Um *cube map* pode ser definido como:

- um cubo de grandes dimensões (bastante superiores à da cena visível),
- com componente especular e difusa nulas, e componente ambiente forte,
- apenas com as faces interiores visíveis,
- e às quais são aplicadas texturas que representam a envolvente da cena (uma paisagem, por exemplo; ver **Figura 4**).

Este objeto pode ser obtido de duas formas diferentes (devem escolher uma):

- A. usando por base o **MyUnitCube** e uma única textura com a disposição da Figura 4, e mapeando diferentes partes da mesma a cada uma das faces ou

- B. usando por base o **MyUnitCubeQuad** e seis texturas diferentes, uma para cada face do cubo

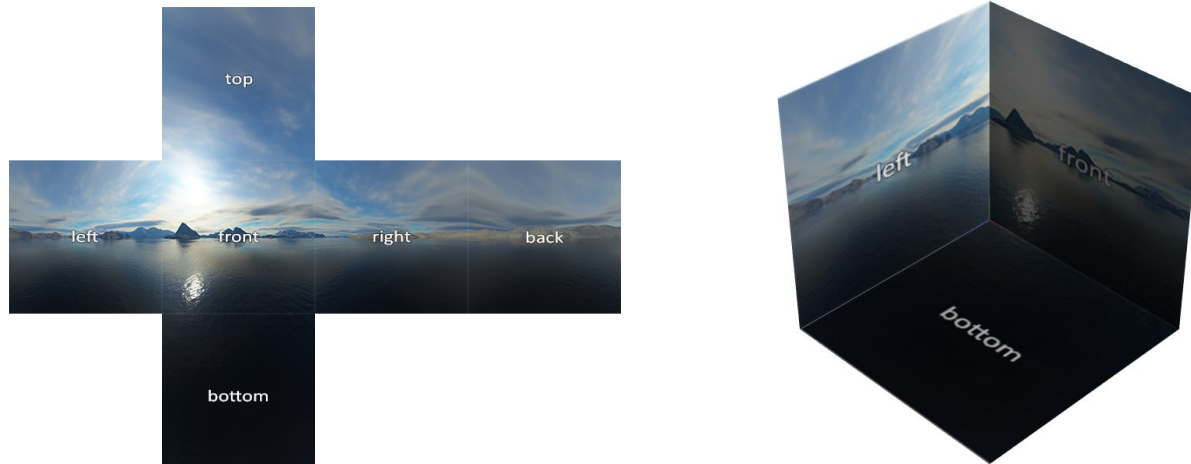
Escolha uma dessas duas opções e inclua na pasta do projeto uma cópia do código da classe respectiva - **MyUnitCube** ou **MyUnitCubeQuad**. Modifique essa cópia para criar uma classe **MyCubeMap**, de forma a ser visível por dentro, e com coordenadas de textura de acordo com essa opção.

O *cube map* deve ser unitário e ao ser usado na cena, deve ser escalado de forma a medir **50 unidades de lado**. Se necessário, poderá ter de alterar a posição da câmara de forma a que fique no interior e centrada dentro do *cube map*.

O código de base inclui um exemplo de imagens para cada um dos dois tipos de *cube map* (uma semelhante à da Figura 4, e um pack de 6 imagens para a opção de seis *quads*).

Devem procurar/criar pelo menos mais uma imagem envolvente à vossa escolha, para permitir depois ao utilizador escolher entre essa e a de exemplo através da interface gráfica (ver ponto 2.2). Como ponto de partida, podem consultar o endereço:

<https://www.cleanpng.com/free/skybox.html>



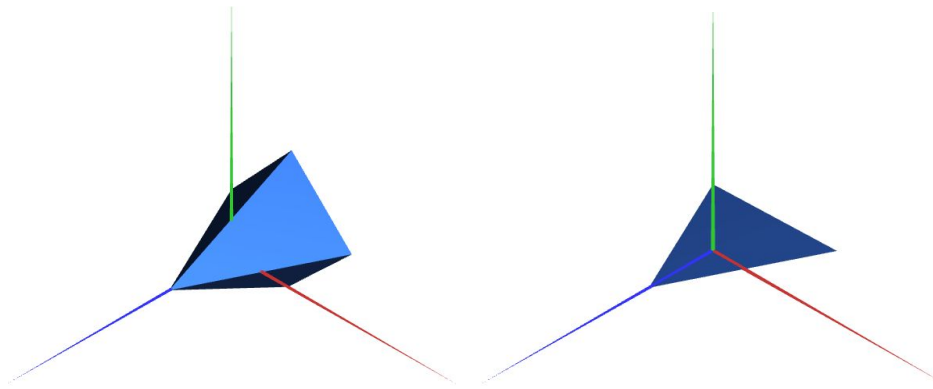
**Figura 4:** Imagem do tipo skybox, e a sua aplicação num cubemap visualizado de fora

### 1.4. **MyVehicle** - Versão preliminar

Crie a classe **MyVehicle**, que irá representar o veículo na cena. A constituição final do veículo será detalhada na Parte B do enunciado. Nesta fase deve criar-se apenas uma representação básica que permita identificar a sua posição e orientação. Para esse efeito, sugerimos que use uma forma simples, como um triângulo não-equilátero como os **MyTriangle** criados para o Tangram, ou a **MyPyramid** fornecida no TP3 (ver **Figura 5**).

Independentemente da geometria escolhida, deve ser garantido o seguinte:

- Ser centrada na origem,
- Ter tamanho unitário
- Ter o eixo central alinhado com o eixo positivo dos ZZ, ou seja: ter a frente a apontar na direção de +Z, como demonstrado na **Figura 5**.



**Figura 5:** Exemplificação da versão preliminar de MyVehicle

a) Pirâmide: b) Triângulo no plano XZ

## 2. Interface

### 2.1. Controlo de veículo por teclado

Para poder controlar o movimento do veículo na cena utilizando teclas, será necessário alterar:

- A interface, para detetar as teclas pressionadas
- A cena, para invocar funções de controlo no veículo em função das teclas detetadas
- O veículo, para contemplar funções de controlo como “rodar” e “avançar” que alteram a posição e orientação do mesmo

Apresentam-se em seguida de forma mais detalhada os passos necessários para o efeito.

**NOTA IMPORTANTE:** Se usar *copy-paste* com o código seguinte, alguns símbolos podem não ser bem reconhecidos (apesar de parecerem iguais) e causar erros em Javascript. Evite fazê-lo, ou verifique bem o código.

1. Altere a classe **MyInterface**, adicionando os seguintes métodos para processar várias teclas ao mesmo tempo:

```
initKeys() {
    // create reference from the scene to the GUI
    this.scene.gui=this;

    // disable the processKeyboard function
    this.processKeyboard=function(){};

    // create a named array to store which keys are being pressed
    this.activeKeys={};
}
```

```

processKeyDown(event) {
    // called when a key is pressed down
    // mark it as active in the array
    this.activeKeys[event.code]=true;
};

processKeyUp(event) {
    // called when a key is released, mark it as inactive in the array
    this.activeKeys[event.code]=false;
};

isKeyPressed(keyCode) {
    // returns true if a key is marked as pressed, false otherwise
    return this.activeKeys[keyCode] || false;
}

```

**NOTA:** No final da função *init()* da **MyInterface**, chame a função *initKeys()*.

2. Na classe **MyScene** acrescente o seguinte método *checkKeys()* e acrescente uma chamada ao mesmo no método *update()*.

```

checkKeys() {
    var text="Keys pressed: ";
    var keysPressed=false;

    // Check for key codes e.g. in https://keycode.info/
    if (this.gui.isKeyPressed("KeyW")) {
        text+=" W ";
        keysPressed=true;
    }

    if (this.gui.isKeyPressed("KeyS")) {
        text+=" S ";
        keysPressed=true;
    }
    if (keysPressed)
        console.log(text);
}

```

Execute o código e verifique as mensagens na consola quando “W” e “S” são pressionadas em simultâneo.

3. Prepare a classe **MyVehicle** para se deslocar:
  - Acrescente no construtor variáveis que definam:
    - a orientação do veículo no plano horizontal (ângulo em torno do eixo YY)
    - a sua velocidade (inicialmente a zero)
    - a sua posição (x, y, z)
  - Crie a função **MyVehicle.update()** para atualizar a variável de posição em função dos valores de orientação e velocidade.
  - Use as variáveis de posição e orientação na função **display()** para orientar e posicionar o veículo.
  - Crie os métodos **turn(val)** e **accelerate(val)** para alterar o ângulo de orientação, e para aumentar/diminuir o valor da velocidade (em que **val** podem ser valores positivos ou negativos).
  - Crie a função **reset()** que deverá recolocar o veículo na posição inicial, com rotação e velocidade nula.
4. Altere o método **checkKeys()** da **MyScene** de forma a invocar os métodos **turn()**, **accelerate()** ou **reset()** do veículo para implementar o seguinte comportamento em função das teclas referidas:
  - **Aumentar ou diminuir a velocidade** conforme se pressionar “W” ou “S”, respectivamente.
  - **Rodar o veículo** sobre si mesmo para a esquerda ou direita se pressionar as teclas “A” ou “D” respetivamente.
  - Pressionar a tecla “R” deverá invocar a função **reset()** do veículo.

## 2.2. Controlos adicionais na interface

Acrescente os seguintes controlos adicionais na interface gráfica (GUI), para poder parametrizar o movimento do veículo e a aparência do *cube map*:

1. Crie uma *list box* onde apareçam as **diferentes texturas de paisagens envolventes** disponíveis, para o utilizador poder alterar a textura do *cube map* em tempo de execução (sugestão: siga o exemplo das texturas no TP4).
2. Crie um *slider* na GUI chamado **speedFactor** (entre 0.1 e 3) que multiplique o valor da velocidade de deslocamento do veículo, a cada vez que se pressiona uma tecla “W” ou “S”.
3. Crie outro *slider* na GUI chamado **scaleFactor** (entre 0.5 e 3) que permita escalar o veículo de forma a que seja mais fácil observar as suas animações.

## Proposta de plano de trabalho - Parte A

- Semana de 30 de março: Início do ponto 1
- Semana de 6 de abril: Continuação do ponto 1; Ponto 2.1
- Semana de 13 de abril: Ponto 2.2 (+ início da Parte B, a ser publicada até lá)

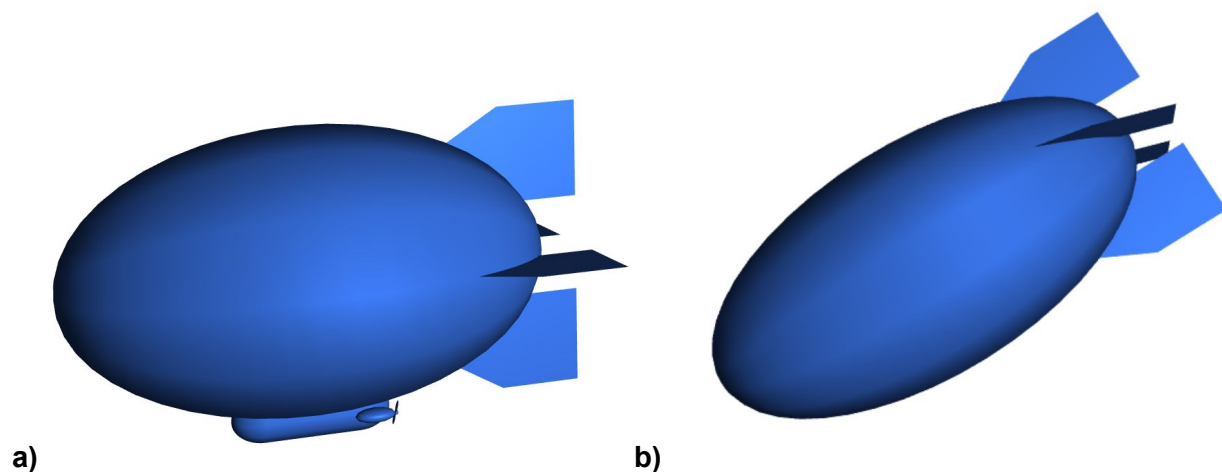
# Parte B

## 3. *MyVehicle - Dirigível*

### 3.1. *Modelação do Dirigível*

Altere a classe **MyVehicle** para representar um dirigível. Para modelar o dirigível, poderá utilizar uma combinação dos diferentes objetos criados anteriormente, de forma a que o dirigível seja constituído por corpo, gôndola (compartimento dos passageiros) com dois motores de hélice, e lemes. A **Figura 5** ilustra o tipo de estrutura pretendida, com geometrias simples.

O dirigível a ser desenvolvido pode usar geometrias diferentes do exemplo, não devendo no entanto ter um número excessivo de polígonos (por questões de desempenho). Sugerimos o uso de cilindros e esferas, mas poderão criar outros objetos que considerem aplicáveis.



**Figura 5:** a) Modelação-exemplo do dirigível, criado com esferas, cilindros, triângulos e quadrados. b) vista de cima com lemes inclinados (ver ponto 3.2)

O dirigível deverá ser visível quando a cena é iniciada, de forma a facilitar a avaliação da mesma, assim como os outros elementos da cena. Coloque-o de forma que o centro do elipsóide fique na posição (0, 10, 0). O corpo principal deverá ter, aproximadamente, comprimento de 2 unidades, e largura e altura de 1 unidade. Os diferentes objetos utilizados para criar o dirigível deverão ter materiais e texturas aplicados aos mesmos, adequados às partes do dirigível que representam.

Submeta uma imagem da aparência do dirigível que permita ver os detalhes e as texturas.





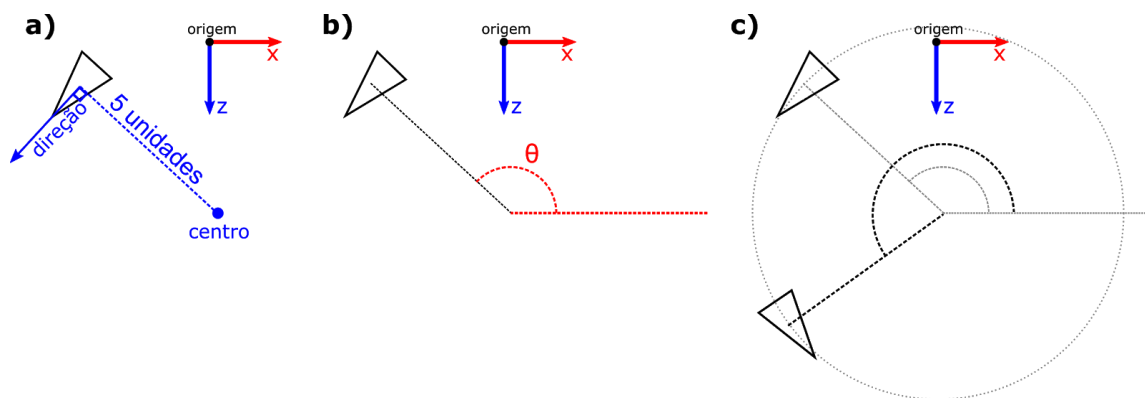
### 3.2. Animações adicionais do Dirigível

Pretende-se criar mecanismos de animação e controlo à classe **MyVehicle**, em complemento às funcionalidades criadas na Parte A do enunciado.

1. As **hélices** do dirigível deverão ter uma animação de rotação, idealmente proporcional à velocidade atual do dirigível.
2. Os **lemes verticais** deverão inclinar para a esquerda ou direita quando o dirigível está a alterar a sua direção. Enquanto a tecla “A” ou “D” for pressionada, os lemes deverão inclinar na direção oposta da rotação.
3. **Piloto automático**: Crie um novo método de animação do dirigível, que é ativado/desativado quando a tecla “P” for pressionada. Enquanto estiver ativo, as outras teclas serão ignoradas, com exceção da tecla de *reset* (“R”).

Este método representa um modo de piloto automático, durante o qual o dirigível se move em círculos consecutivos com **5 unidades de raio, a partir do ponto em que se encontra no momento**. Para implementar esta animação, representada na **Figura 6**, é necessário:

- Determinar o centro do círculo de animação a partir da posição e orientação atuais;
- Determinar ângulo inicial em relação à direção paralela ao eixo XX;
- Definir a posição inicial utilizando o centro e ângulo inicial calculados;
- Atualizar a posição e orientação a cada frame de forma a que o dirigível se mova em círculo, com orientação igual à tangente do círculo na posição atual;
- **Uma volta completa deve demorar 5 segundos, independentemente do desempenho do computador usado.**



**Figura 6:** Exemplificação dos passos da animação “piloto automático”

## 4. Terreno

Pretende-se adicionar um terreno à cena, criado com uma versão simplificada dos shaders da água criados no TP 5.

O terreno será construído recorrendo a duas texturas:

- Uma que funciona como mapa de alturas (como o mapa de ondulação usado na água)
- Outra como informação da cor a ser mapeada sobre o terreno

A Figura 7 contém um exemplo possível de MyTerrain, com as respetivas texturas.

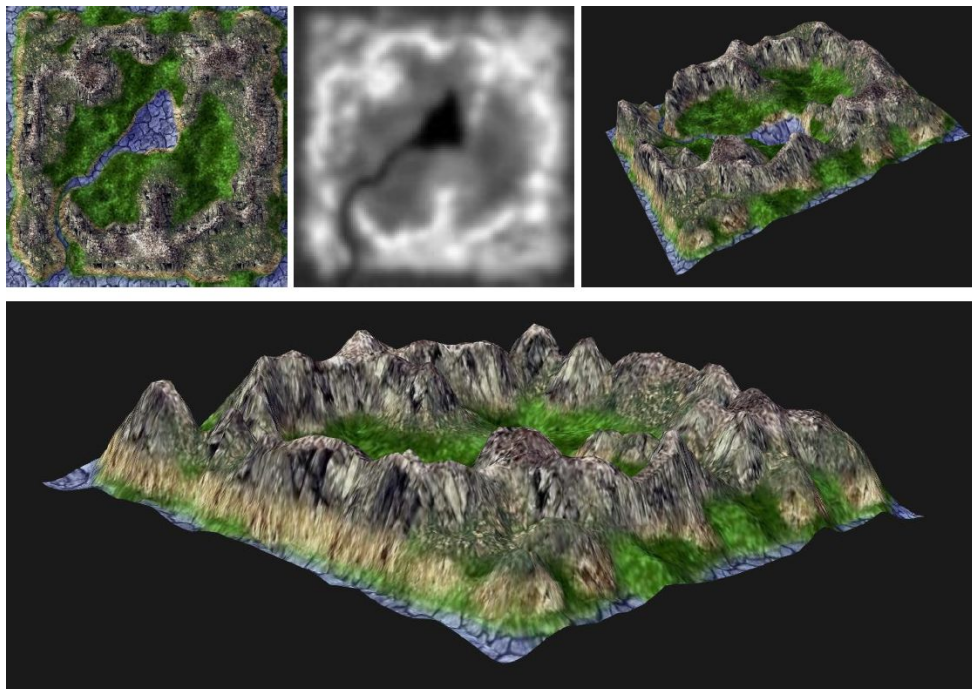


Figura 7: Textura de cor, mapa de alturas e geometria gerada.  
(origem: Outside of Society [http://oos.moxiecode.com/js\\_webgl/terrain/index.html](http://oos.moxiecode.com/js_webgl/terrain/index.html))


Para obter este efeito, sugere-se que:

- Crie cópias dos ficheiros dos shaders “water.frag” e “water.vert”, com os novos nomes “terrain.frag” e “terrain.vert”;
- Retire dos novos shaders as partes referentes à animação;
- Crie uma nova classe MyTerrain, que será constituído por um MyPlane com 20 divisões em cada direção (ver aula prática de shaders), e que contenha, inicialize e use os shaders terrain.frag e terrain.vert preparados nos pontos anteriores;
- Crie uma instância de MyTerrain na cena com dimensões de 50x50 unidades, e com altura máxima de 8 unidades.
- Teste uma versão inicial com as texturas de cor e altura fornecidas.
- Modifique a textura de alturas (usando um editor de imagem) de forma a que haja uma zona central plana de dimensões aproximadas de 20x20 unidades (altura a zero, Y=0).

Em alternativa, poderá substituir as texturas fornecidas por outro par de texturas de cor e altura para o efeito - mas que terá de ter a zona central plana como descrito.

Na classe **MyScene**, altere o código de forma a que:

- Se necessário, para alinhar as imagens do cubemap com o terreno, aplicar translação em Y no terreno ou no cubo;
- A posição e orientação inicial da câmara permita ver o dirigível e a maior parte da zona plana do terreno.

Capture uma perspetiva da cena contendo o ambiente de fundo criado com o cubemap, o terreno, e o veículo.  (2)

## 5. Lançamento de mantimentos

Pretende-se adicionar uma nova funcionalidade ao MyVehicle, de forma a que o dirigível possa largar mantimentos (nova classe MySupply).

### 5.1. Criação de MySupply

Crie uma nova classe MySupply, que será baseada no código da classe MyUnitCubeQuad (ou seja, terá o mesmo código inicialmente, que deverá ser alterado como indicado a seguir), com materiais e texturas à escolha de cada grupo (p.ex. de uma caixa de madeira). Esta classe deverá contemplar os seguintes estados:

- INACTIVE: Quando ainda não foi lançada - não é desenhada;
- FALLING: Quando é lançada e enquanto está em queda - é desenhada com o material e texturas referidos acima, e animada como descrito no ponto 5.3;
- LANDED: Quando cai no terreno, é desenhada de forma diferente para representar o resultado da queda (p.ex com as suas seis faces “espalhadas” no chão / caixa aberta).

Sugere-se que implemente uma máquina de estados simples dentro da classe:

- Defina uma enumeração dos estados possíveis, e no construtor inicialize uma variável que representará o estado.

```
const SupplyStates = {  
  INACTIVE: 0,  
  FALLING: 1,  
  LANDED: 2  
};  
constructor ()  
{  
  this.state=SupplyStates.INACTIVE;  
}
```

- Nas funções `MySupply.update()` e `MySupply.display()`, utilize o valor de `this.state` para escolher o tipo de atualização/desenho a levar a cabo (por exemplo, a função `update` só precisará de atualizar a posição da caixa quando ela está a cair, nos outros estados provavelmente não terá de fazer nada).
- Implemente os métodos necessários para despoletar as mudanças de estado:
  - `drop(dropPosition)` - para transitar de `INACTIVE` para `FALLING` e começar a animação de queda a partir da `dropPosition` (posição atual do dirigível)
  - `land()` - a ser chamada quando a animação de queda termina, para determinar se atingiu o plano XZ ( $Y=0$ ) ou não. Se isto se verificar, deve transitar de `FALLING` para `LANDED`.
  - Outros métodos que considerem relevantes (por exemplo, e como sugestão apenas, podem ter funções de desenho auxiliares - `displayFalling`, `displayOnLanded` - que sejam chamadas pela função `display`, dependendo do estado)

## 5.2. Controlo do Lançamento

Adicione a `MyScene` a funcionalidade de lançar 5 mantimentos do dirigível, que deverá funcionar da seguinte forma:

- Ao pressionar a tecla “L”, um objeto `MySupply` deverá surgir na parte inferior do dirigível, e iniciar a sua queda;
- O mantimento deverá demorar 3 segundos a atingir o plano XZ ( $Y$  nulo);
- Quando o mantimento atingir o plano XZ, deve ser desenhado de forma diferente (ver estado `LANDED` no ponto 5.1)

**Nota:** a função de reset (tecla “R”) deve também fazer reset aos mantimentos, removendo-os da cena e disponibilizando-os para novos lançamentos.

Sugere-se para esse efeito que altere o seguinte na classe `MyScene` (sugestões não exaustivas, poderão ter de acrescentar/alterar outras funcionalidades):

- Acrescente uma lista de 5 objetos do tipo `MySupply`, todos inativos inicialmente;
- Nas funções `update()` e `display()` da cena, acrescente a invocação dos respetivos métodos `update()` e `display()` para cada um dos 5 objetos;
- Acrescente o código necessário na função `checkKeys()` para detetar a tecla “L” e:
  - invocar o método `drop()` de um objeto inativo, fornecendo a posição atual do dirigível, iniciando assim a sua queda;
  - Atualizar um contador interno `nSuppliesDelivered` que indica o número de mantimentos que já foram lançados (a ser usado no ponto 6);

Capture uma imagem da cena onde se veja um mantimento em queda (com o estado *FALLING*), e outra na qual se observe um mantimento já no terreno (com o estado *LANDED*). Utilize o zoom da câmara para se poder observar bem os detalhes dos objetos em questão.



(3)



(4)

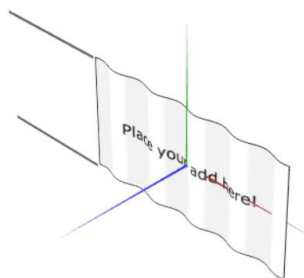
## 6.Shaders

### 6.1. Bandeira no dirigível

Adicione um objeto da classe **MyPlane** à classe **MyVehicle**, e coloque-o atrás do corpo do dirigível, semelhante ao exemplo na **Figura 8**. Altere a classe **MyPlane** de forma a que o plano tenha dupla face (*double-sided*).

Crie e inicialize shaders em **MyVehicle** que serão aplicados ao novo objeto bandeira, que deverão:

- Alterar a altura dos vértices do plano de forma a recriar um efeito de “ondulação” no objeto, seguindo uma função sinusoidal (Sugestão: use as coordenadas de textura como parâmetro);
- O efeito de ondulação deverá ser animado, e a velocidade da animação deverá depender da velocidade do dirigível;
- Aplique uma textura à escolha com o *fragment shader*.



**Figura 8:** Exemplo de bandeira com ondulação e uma simples textura aplicada

Capture uma imagem que demonstre o dirigível com a bandeira criada, em que seja possível observar a ondulação na mesma.



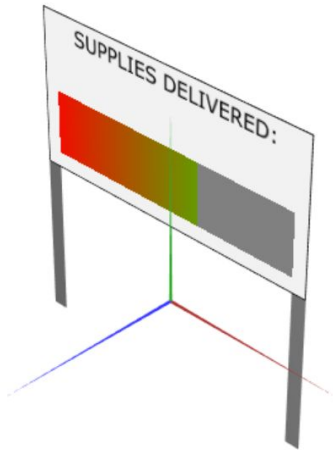
(5)

### 6.2. Contador de mantimentos entregues

Crie uma nova classe **MyBillboard**, para representar um mostrador com o número de mantimentos (**MySupply**) lançados, tal como contabilizado pela variável *nSuppliesDelivered*.

Este mostrador deverá ser constituído por (ver **figura 9**, dimensões indicativas):

- um plano de base com uma textura decorativa que inclua o texto “Supplies Delivered”
- Uma barra de progresso (detalhes abaixo)
- duas traves que suportam a geometria acima.



**Figura 9:** Exemplo de billboard

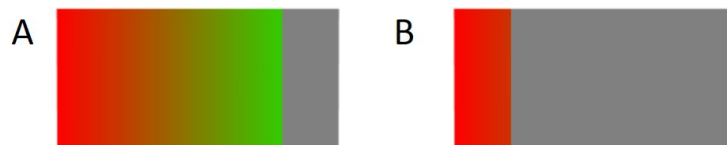
O plano de base deve ter dimensões 2 x 1 unidades, as traves devem ter altura de uma unidade, e o mostrador deve ser colocado no terreno numa posição bem visível no ponto de vista inicial da câmara.

A barra de progresso deve ser constituída por um único plano de dimensões 1.5 x 0.2 unidades (largura x altura) associado a um par de shaders desenvolvido especificamente para o efeito e ativado na classe **MyBillboard**, obedecendo ao seguinte (ver **figura 10**):

- Quando a barra está totalmente preenchida (ou seja, todos os mantimentos entregues) deve ter o aspeto de um gradiente horizontal que varie de vermelho (RGB = 1,0,0) para verde (RGB = 0,1,0). Este gradiente deve ser calculado no *fragment shader*;
- Quando apenas uma parte dos mantimentos tiver sido entregue, só deve ser visível a percentagem correspondente do gradiente. Por exemplo, quatro mantimentos entregues num total de cinco corresponderá a 80% do gradiente total;
- A restante área deve ser preenchida com uma cor constante à escolha, desde que contraste com os tons do gradiente (no exemplo, cinzento RGB = 0.5,0.5,0.5).

Capture uma imagem do mostrador quando três mantimentos tiverem sido lançados. 📷 (6)

Submeta o código final. 📄 (1)



**Figura 10:** Exemplos de barra de progresso com gradiente de vermelho a verde, desenhado até 80% (imagem A) ou 20% do plano (imagem B)

# Notas sobre a avaliação do trabalho

A classificação máxima a atribuir a cada alínea corresponde a um desenvolvimento ótimo da mesma, no absoluto cumprimento com todas as funcionalidades enunciadas. Sem perda da criatividade desejada num trabalho deste tipo, não serão contabilizados, para efeitos de avaliação, quaisquer desenvolvimentos além dos que são pedidos, como forma de compensar outros componentes em falta.

## 1. Criação de objetos de base (3 valores)

- 1.1. MyCylinder (1.5 valores)
- 1.2. MySphere (0.5 valores)
- 1.3. Cube Map (1 valor)
- 1.4. MyVehicle (0 valores, contabilizado no ponto 3)

## 2. Interface (3 valores)

- 2.1. Controlo de veículo por teclado (2 valores)
- 2.2. Controlos adicionais na interface (1 valor)

## 3. Dirigível (4 valores)

- 3.1. Modelação do Dirigível (2 valores)
- 3.2. Animações adicionais (2 valores)

## 4. Terreno (1 valor)

## 5. Lançamento de mantimentos (3.5 valores)

- 5.1. Criação de MySupply (2 valores)
- 5.2. Controlo do Lançamento (1.5 valores)



## 6. Shaders (3.5 valores)

- 6.1. Bandeira (2 valores)
- 6.2. Contador de mantimentos entregues (1.5 valores)

## 7. Estrutura e qualidade de software (2 valores)

# Checklist

Até ao final do trabalho deverá submeter as seguintes imagens e versões do código via Moodle, **respeitando estritamente a regra dos nomes**:

-  Imagens (6): (nomes do tipo "proj-t<turma>g<grupo>-n.png")
-  Código em arquivo zip (1): (nomes do tipo "proj-t<turma>g<grupo>-n.zip")