

# **Computação Gráfica (L.EIC)**

## ***Projeto 2022/2023***

### **Notas preliminares**

Em relação à versão inicial, o presente documento apresenta as seguintes alterações:

2023/04/05:

- Nota adicional na classe ***MyPanorama*** que consta no último parágrafo da secção 2.
- Adição de novas secções 5, 6 e 7.
- Adição das cotações inerentes à avaliação do trabalho.
- Adição de uma recomendação de Plano de Trabalho.

# **Computação Gráfica (L.EIC)**

## **Projeto 2022/2023**

Versão v1.0 - 2023/04/05

## **Objetivos**

- Aplicar os conhecimentos e técnicas adquiridas até à data
- Utilizar elementos de interação com a cena, através do teclado e de elementos da interface gráfica
- Utilizar *Shaders* na modelação/visualização de objetos
- Utilizar animação de componentes

## **Descrição**

Pretende-se com este projeto a criação de uma cena que combine os diferentes elementos explorados nas aulas anteriores. Para este trabalho deve usar como base o código que é fornecido no Moodle, que corresponde a uma cena com um plano de 400x400 unidades. Terá posteriormente de adicionar alguns dos objetos criados em trabalhos anteriores.

A cena, no final do projeto, deve ser genericamente constituída (pelo menos) por:

- Um terreno com elevações, criadas através de um shader;
- Uma floresta, composta por árvores usando geração procedural e shaders;
- Uma ave, animada e controlável pelo utilizador, assim como o seu ninho.
- Um conjunto de ovos espalhados pelo terreno

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 crie a sua própria cena.

## **Preparação do Ambiente**

Devem descarregar o código disponibilizado para este trabalho do Moodle e colocar a pasta **project** contida no ficheiro .zip, ao mesmo nível dos trabalhos anteriores.

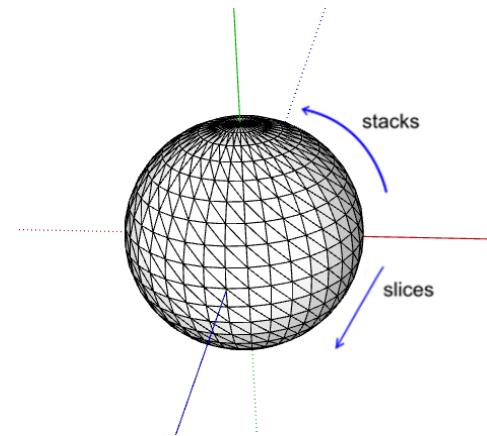
O código fornecido para o projeto tem uma cena constituída apenas pelos eixos do sistema de coordenadas, um objeto **Plane**, e uma fonte de luz. Deverá incluir posteriormente neste projeto alguns elementos presentes nos trabalhos anteriores, nomeadamente os cilindros, cones, cubos, etc., conforme forem sendo necessários.

## 1. Criação de Esfera

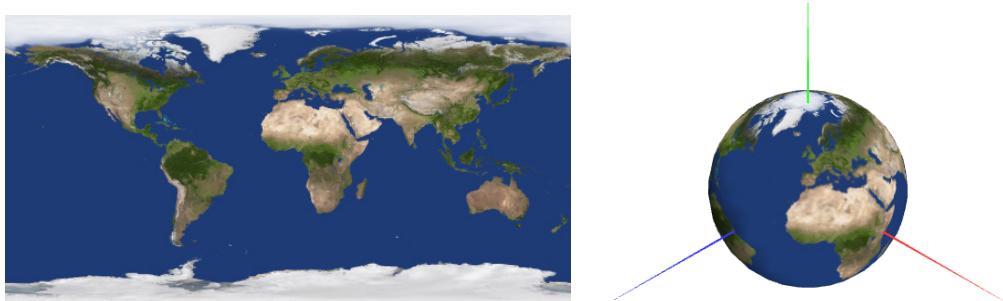
Crie uma nova classe **MySphere** que crie uma esfera com o centro na origem, com eixo central coincidente com o eixo Y e raio unitário.

A esfera deve ter um número variável de "lados" à volta do eixo Y (*slices*), e de "pilhas" (*stacks*), este último correspondendo ao número de divisões ao longo do eixo Y, desde o centro até aos "pólos" (ou seja, número de "pilhas" de cada semi-esfera). A **Figura 1** tem uma representação visual da esfera.

O algoritmo utilizado deve ser eficiente e deve gerar as coordenadas de textura para a respetiva aplicação à superfície da esfera, como demonstrado na **Figura 2**.



**Figura 1:** Imagem exemplo de uma esfera centrada na origem.



**Figura 2:** Exemplo de textura (disponível no Moodle) e sua aplicação numa esfera

Coloque uma instância de teste da esfera na cena e crie uma tag no repositório neste ponto.



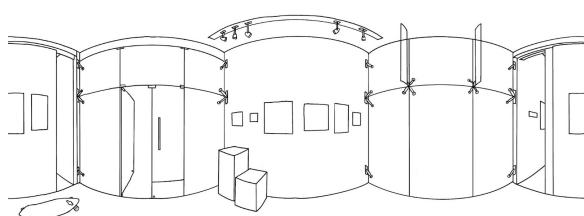
## 2. Criação de Panoramas

Parameterize a classe MySphere para poder inverter as suas faces, de forma a que seja visível por dentro e não por fora (atenção à ordem dos vértices e às normais). A ideia é poder usar uma esfera de grandes dimensões à volta da cena com uma imagem panorâmica, para criar uma paisagem, e

o observador deslocar-se dentro dessa esfera. Podem encontrar imagens panorâmicas / 360º no formato equirretangular, com as da Figura 3 p.ex. em

<https://www.flickr.com/search/?text=equirectangular%20landscape>

(nos links da Fig. 3 podem visualizar a sua aplicação de forma interativa)



**Figura 3:** Dois exemplos de imagens equirectangulares

Crie a classe **MyPanorama** que:

- no seu construtor receba uma **CGFtexture**, e seja responsável por criar uma **MySphere** invertida, e um material apenas com componente emissiva e a textura associada,
- no seu método **display** ative a textura e desenhe a esfera com um raio de 200 unidades

Inclua na cena um destes panoramas. Sugere-se que experimente diferentes valores para o parâmetro FoV (field of view) da câmara da cena, de forma a que a distorção de perspetiva seja satisfatória.

Neste ponto, faça capturas de ecrã que permitam ver uma ou duas perspectivas da cena e do panorama. Crie uma tag no repositório neste ponto.



**Nota adicional:**

Altere a classe **MyPanorama** para que fique centrada na posição da câmara, passando a mover-se com a mesma e dando a ilusão de a superfície esférica estar sempre posicionada no infinito. **Nota:** pode utilizar para o efeito o membro *position* da câmara guardada na **cena** (*this.camera.position*). Trata-se de um *vec3*, pelo que as várias coordenadas podem ser obtidas acedendo às posições [0], [1] e [2].

### 3. Inclusão de uma Ave

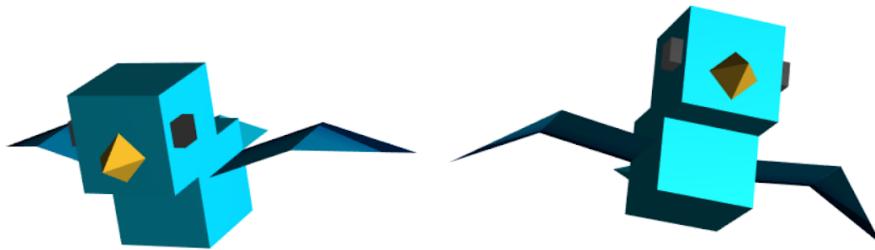
Pretende-se que seja incluída uma ave na cena. Deve ser animada e dotada de movimento controlável a partir do teclado de acordo com as subsecções seguintes.

#### 3.1. Modelação da ave

Crie uma nova classe **MyBird**, para representar uma ave. Para modelar a ave, poderá utilizar uma combinação dos diferentes objetos utilizados anteriormente (também em outros trabalhos), de forma a que a ave seja constituída por corpo, cabeça (com olhos e bico), e asas. Cada asa deverá

ser constituída por duas partes. A figura 4 ilustra o tipo de estrutura pretendida, mas com o corpo e a cabeça simplificados.

A ave deve usar geometria um pouco mais complexa do que a apresentada na imagem para o corpo e a cabeça (não devendo no entanto ter um número excessivo de polígonos). Sugerimos o uso de cones ou cilindros com um número reduzido de lados (3 a 5), esferas ou outros objetos que considerem relevantes.



**Figura 4:** Exemplificação do modelo da ave, simplificada com cubos para o corpo e cabeça.

A ave deverá ser bem visível quando a cena é iniciada (recorrendo a uma boa definição dos parâmetros iniciais da câmara) de forma a facilitar a avaliação da mesma. Poderá ter comprimento até 2 unidades, sendo que com asas abertas poderá ter largura até 3 unidades.

Os diferentes objetos utilizados para criar a ave deverão ter materiais e texturas aplicados aos mesmos, adequados às partes da ave que representam. Mostre uma imagem da aparência da ave (suficientemente perto para ver a mesma em detalhe) .



### **3.2. Animação da Ave**

Neste ponto devem criar-se os mecanismos de animação e controlo para um objeto da classe *MyBird*.

Coloque a Ave na cena a cerca de 3 unidades acima do chão. A ave deverá ter duas animações aplicadas de forma constante ao longo do tempo, simulando o movimento do vôo:

1. Uma animação para oscilar levemente para cima e para baixo, sendo que cada oscilação completa (cima e baixo) deverá demorar 1 segundo.
2. Uma segunda animação para o bater das asas; a velocidade do bater de asas deverá depender da velocidade atual da ave (ver próxima secção).

### **3.3. Controlo da Ave**

Para poder controlar o movimento da ave na cena, será necessário prever o pressionar de uma ou mais teclas em simultâneo.

1. Altere a classe **MyInterface**, adicionando os seguintes métodos para processar várias teclas ao mesmo tempo (**nota:** não fazer copy-paste deste documento, pois alguns caracteres podem ser convertidos de forma incorreta):

```
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;
}
```

No final da função *init()* de **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. Altere a classe **MyBird** de acordo com o seguinte:

- Acrescente no construtor variáveis que definam:
  - a orientação da ave (sugestão: ângulo em torno do eixo YY)
  - a sua velocidade (inicialmente a zero)
  - A sua posição (x, y, z)
- Altere a função **update** para atualizar a variável de posição em função dos valores de orientação e velocidade
- Use as variáveis de posição e de orientação na função **display()** para orientar e posicionar a ave.
- Crie os métodos **turn(v)** e **accelerate(v)** para rodar e para aumentar/diminuir a velocidade da ave.

4. Preveja que as teclas possam invocar os métodos **turn** e **accelerate** da ave, de forma a implementar o seguinte comportamento:

- Acelerar ou travar conforme se pressionar "W" ou "S", respectivamente.
- Rodar a ave para a esquerda ou direita se pressionar as teclas "A" ou "D";
- Fazer "reset" à posição e velocidade da ave através da tecla "R" ; isto é, a ave deverá ser colocada na posição inicial, com rotação e velocidade nula.

4. Crie um *slider* na GUI chamado *speedFactor* (entre 0.1 e 3) que acelere e desacelere a velocidade de deslocamento, rotação e bater de asas da ave.

5. Crie outro *slider* na GUI chamado *scaleFactor* (entre 0.5 e 3) que permita escalar a ave de forma a que seja mais fácil observar as suas animações.

Crie uma tag no repositório neste ponto.

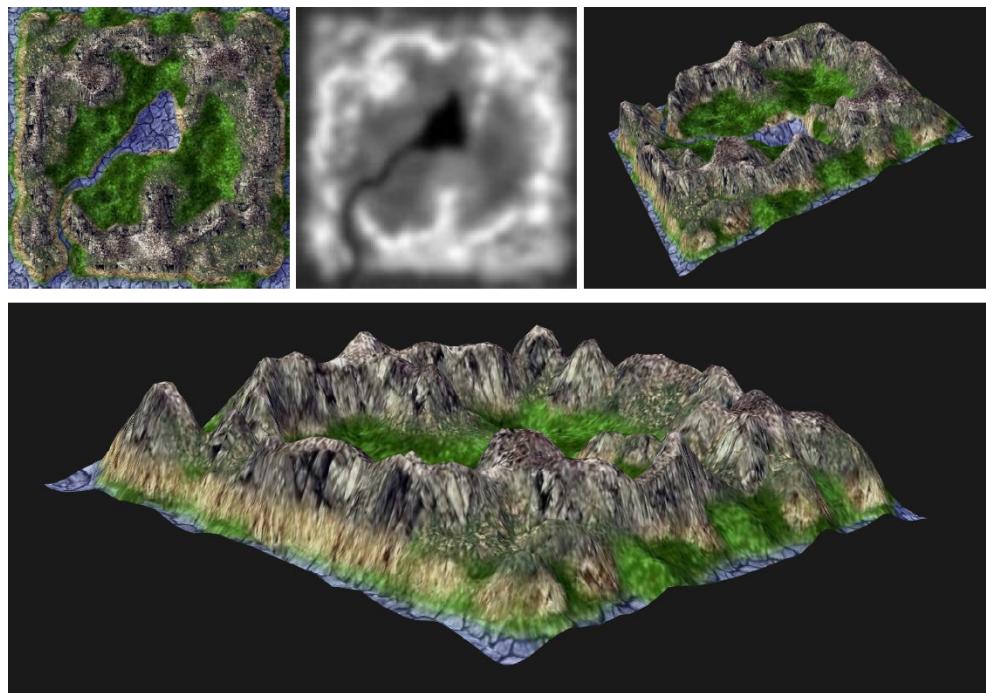


(3)

## 4. Terreno

Pretende-se melhorar o terreno, nomeadamente introduzindo altimetria, utilizando *shaders*. Crie uma nova classe **MyTerrain**, que será constituída por um **Plane** (aula prática de *shaders*), para representar o terreno com elevações. Estas deverão ser obtidas com base em uma textura de níveis de cinzento que assim funciona como mapa de alturas. A criação dos shaders, das texturas associadas e a sua aplicação devem ser encapsuladas dentro da classe **MyTerrain**.

A figura 5 contém um exemplo possível de **MyTerrain**. Deve substituir o **Plane** inicial de 400x400 unidades por um **MyTerrain** com a mesma dimensão.



**Figura 5:** Textura de cor, mapa de alturas e geometria gerada.  
 (origem: Outside of Society [https://oosmoxiecode.com/archive/js\\_webgl/terrain/index.html](https://oosmoxiecode.com/archive/js_webgl/terrain/index.html) )

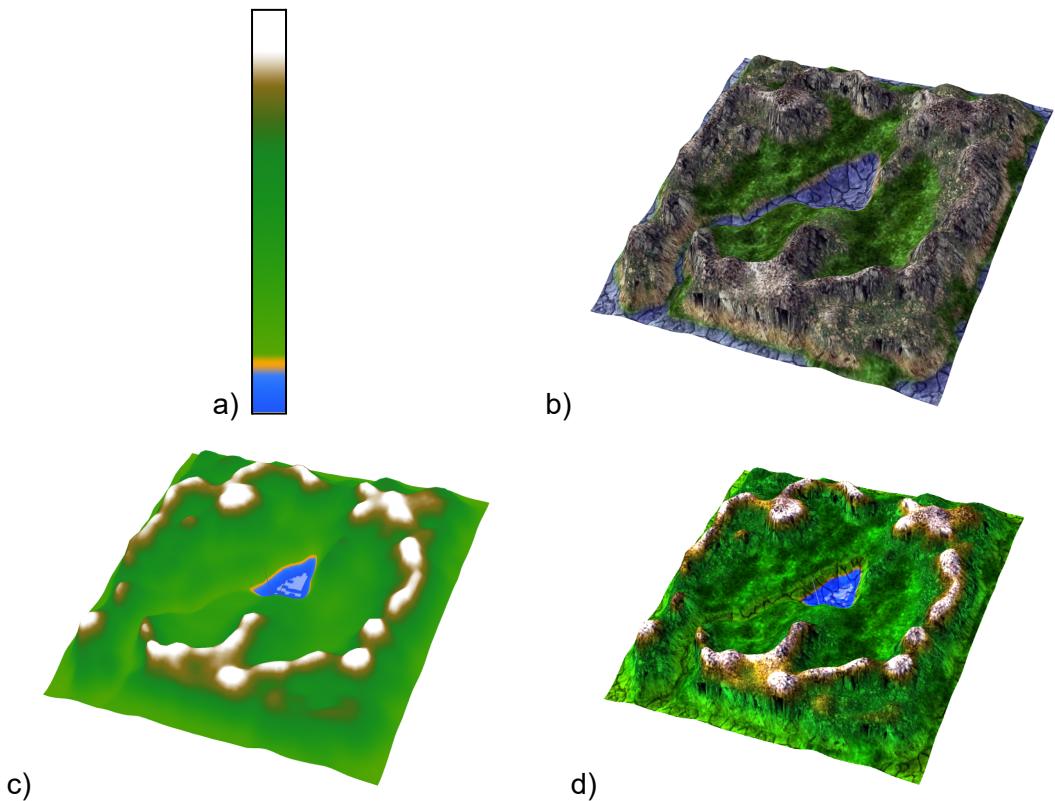
1. Teste uma versão inicial com as duas texturas, de cor e de altura, fornecidas.
2. Modifique a textura de alturas (usando um editor de imagem) de forma a que haja uma zona **plana** de dimensões aproximadas de 20x20 unidades ( $\frac{1}{3}$  da largura/comprimento do terreno).

NOTA: poderá **substituir este par de texturas por outro**, desde que respeite as regras enunciadas.

3. Altere o **shader** de forma a que:
  - a. Receba uma terceira textura representando um gradiente de cor - **altimetria** (ver figura 6a, imagem fornecida com o código).
  - b. Em cada fragmento, em vez de aplicar apenas a cor da textura original do terreno, combine essa cor com o valor da cor do gradiente correspondente à altimetria: nas altitudes mais elevadas a cor original deverá ser mais “branca” e nas inferiores será mais “azul” (ver exemplo na figura 6c e 6d). Essa combinação deve ser de 70% da cor original e 30% da cor da zona correspondente da textura de altimetria (Nota: poderá ser feito algum ajuste a escalas ou offsets para adaptar o gradiente à gama de alturas; pode pôr o peso das componentes como um parâmetro).

Neste ponto, faça capturas de ecrã que permitam ver uma ou duas perspectivas do terreno e que incluam a ave e o fundo. Crie uma tag no repositório neste ponto.

(3) (4)



**Figura 6:** Utilização de uma textura para colorir zonas do terreno em função da altura

- a) gradiente;
- b) terreno só com textura de cor original
- c) terreno só com cor do gradiente;
- d) combinação gradiente e cor original

## 5. Ovos e Ninho

Pretende-se adicionar uma nova funcionalidade a **MyBird**, de forma a que a ave apanhe, um a um, os ovos (**MyBirdEgg**) espalhados pelo terreno e os largue no ninho (**MyNest**).

1. Criação dos objetos
  - a. Crie uma nova classe **MyBirdEgg**, com base em **MySphere** que aplique diferentes fatores de escala em YY aos dois hemisférios, permitindo que um deles fique mais “alongado”. Deve ser aplicado um material e uma textura adequados (devem ter “pintas”).
  - b. Coloque vários objetos desta classe em cena (quatro, pelo menos), com posições e rotações à escolha/aleatórias, pousados na zona plana do terreno. Utilize um vetor de ovos, para poder ser genérico/configurável.
  - c. Crie uma nova classe **MyNest**, que representará um ninho de aves, constituído por objetos à sua escolha. Aplique materiais e texturas adequados para se assemelhar a um ninho.
  - d. Coloque um objeto de **MyNest** na zona plana da cena, garantindo que tanto este como os objetos de **MyBirdEgg** são visíveis no início da cena, para avaliação.
2. Adicione a **MyBird** a funcionalidade de apanhar e largar ovos da seguinte forma:
  - a. Ao pressionar a tecla “P” a ave deve descer até ao nível do chão (zona plana) e voltar a subir à altura inicial num período de 2 segundos (mantendo a sua velocidade em XZ).

- b. Se nesse processo, quando a ave toca no chão, estiver um ovo nesse ponto (deve ser implementada uma margem de tolerância), esse ovo deve ser “apanhado” pela ave.  
**Sugestão:** adicione a referência desse ovo a ***MyBird***, de forma a que passe a ser desenhado no ***display*** da ave, e retire a referência do ovo da cena, para deixar de ser desenhado no terreno.
- c. Se a ave estiver a transportar um ovo e for pressionada a tecla “O” ao passar nas imediações do ninho, deve largar o ovo de forma a que este caia e se deposite no ninho (ou seja, a sua referência deve ser adicionada ao ninho e retirada da ave). Devem ser previstas, dentro do ninho, algumas posições possíveis para os ovos serem depositados individualmente.

Registe este ponto do desenvolvimento:  (4)  (5)

## 6. Integração de Árvores

Pretende-se criar árvores com base em *billboards* para enriquecer o ambiente.

### 6.1. Criação de *billboard*

Crie uma nova classe ***MyBillboard***, que deverá ter uma instância de ***MyQuad***. Pode parametrizar o construtor do ***MyBillboard*** como achar conveniente. Posteriormente:

1. Altere a função *display* do ***MyBillboard*** para receber como parâmetros as coordenadas x, y, z onde o quad deverá ser desenhado.
2. Na função *display* do ***MyBillboard***, rode o quad de forma a que esteja sempre orientado para a câmara. Isto significa que a normal do quad deverá possuir a mesma orientação que o vector que vai da sua origem até à câmara (ver secção 1). **Nota:** a informação necessária para rodar um vector de forma a que tenha a orientação de outro, ambos **normalizados**, pode ser obtida através de:
  - a. Ângulo:  $\cos^{-1}$  (função ***Math.acos***) do valor obtido do produto interno entre os dois vectores (função ***vec3.dot***).
  - b. Eixo de rotação: produto externo (função ***vec3.cross***).
3. Modifique a transformação anterior de forma a que o quad seja desenhado sempre na vertical. Para isso deverá considerar apenas as componentes horizontais (XX e ZZ) dos vectores referidos.
4. Aplique um material com a textura *billboardtree.png* fornecida (ou outra que considere adequada) antes do *display* do billboard na cena. Para obter o efeito de transparência desejado (fig. 5), deverá evitar que os fragmentos do quad cuja componente alpha do texel correspondente não seja completamente opaca sejam desenhados. **Nota:** pode alterar o *default fragment shader* da WebCGF (*lib\CGF\shaders\Gouraud\textured\fragment.glsL*) de forma a descartar os fragmentos que não cumpram o requisito (i.e., fragmentos com alpha menor que 1) usando a palavra reservada ***discard*** (ex.: *if(...)* ***discard***;).



**Figura 7:** Billboard da árvore virado para a câmara e com transparência.

## 6.2. Arvoredo

Crie duas classes que representem dois tipos de grupos de árvores, para depois poderem ser espalhados pela cena. A classe ***MyTreeGroupPatch*** define um conjunto de 9 árvores distribuídas numa grelha de 3x3, mas não perfeitamente alinhadas, e com alguma variedade de dimensões (fig. 8) e de texturas. A classe ***MyTreeRowPatch*** define um grupo de 6 árvores em linha, ligeiramente desalinhadas de forma arbitrária (fig. 9). Em ambos os casos, a textura aplicada a cada árvore deverá ser aleatória de entre um conjunto de 3 elementos (podem utilizar texturas à escolha, desde que devidamente creditadas no código/README).



**Figura 8:** Grupo de 9 árvores distribuídas em 3x3 (***MyTreeGroupPatch***).



**Figura 9:** Grupo de 6 árvores distribuídas em linha (***MyTreeRowPatch***).

Mostre uma imagem aérea da floresta e do terreno. (5) (6)

## 7. Desenvolvimentos adicionais

Das seguintes três alternativas, escolha uma (e só uma) para implementação.

- A. Trajetória em parábola do ovo quando é largado pela ave para ser depositado no ninho.
- B. Deformação de árvores com o vento, com base no deslocamento de vértices dos quads respetivos.
- C. Colocação de árvores nas zonas não planas. Para garantir que as árvores ficam visualmente alinhadas com o chão (nem enterradas nem no ar), esta opção deve recorrer ao mapa de alturas do terreno.

Submeta o código final.  (6)  (7)

---

## 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, de acordo com os valores abaixo. 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 esfera ( 1.5 valores )**
- 2. Criação de Panoramas ( 1 valor )**
- 3. Modelação de uma Ave ( 2 valores )**
- 4. Animação da Ave ( 1.5 valores )**
- 5. Controlo da Ave ( 2 valores )**
- 6. Terreno ( 1.5 valores )**
- 7. Ovos e Ninho**

- 7.1. Criação dos objetos ovo e ninho ( **1 valor** )
- 7.2. Implementação do método de apanhar/largar ovos ( **2 valores** )

### 8. Árvores

- 8.1. Modelação de uma árvore ( **1.5 valor** )
- 8.2. Arvoredo ou grupos de árvores ( **1.5 valores** )

### 9. Valorização ( **1.5 valores** )

- 10. Aspetto geral da cena e criatividade ( 1.5 valores )**
- 11. Estrutura e qualidade de software ( 1.5 valores)**

# Proposta de plano de trabalho

Recomenda-se o seguinte plano de trabalho ao longo do tempo restante do semestre:

Finalização:	Pontos do enunciado
14 de abril	Pontos 1, 2 e 3 (Esfera, Panorama, Inclusão de Ave)
21 de abril	Ponto 3 - conclusão; Ponto 4 (Terreno)
28 de abril	Ponto 5 (Ovos e Ninho)
12 de maio	Pontos 6 e 7 (Integração de Árvores, Desenvolvimento Adicional)
19 de maio	Refinamentos

Durante as aulas da semana de 22 de maio, far-se-á a entrega dos trabalhos e as respectivas avaliações.

## 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 "project-t<turma>g<grupo>-n.png")



**GIT Commits/Tags (7):** (formato “proj-1”, “proj-2”, ...)

## Vídeo

Deverá também preparar um **vídeo de 1 minuto em mp4**, a ser submetido em área própria a ser indicada no Moodle. O vídeo deve ser capturado do ecrã demonstrando todas as funcionalidades implementadas (nomes do tipo "project-t<turma>g<grupo>.mp4").