

# Computação Gráfica (L.EIC)

## Tópico 1

### *Introdução e primeiros passos*

## Objetivos

- Instalar, explorar e aprender a utilizar as bibliotecas e exemplos de base para os trabalhos, bem como os procedimentos para a submissão de resultados
- Aprender a criar e utilizar uma interface gráfica (GUI) para controlar aspectos da cena e os seus objetos.
- Aprender a criar objetos simples

## Introdução

Atualmente é possível gerar gráficos interativos 3D em browsers web, recorrendo à tecnologia **WebGL** e à linguagem **JavaScript**.

Esta forma de desenvolvimento 3D tem as vantagens de não necessitar da instalação de bibliotecas ou a compilação de aplicações, e de poder facilmente disponibilizar as aplicações em diferentes sistemas operativos e dispositivos (incluindo dispositivos móveis). No entanto, para aplicações mais exigentes, é recomendável a utilização de linguagens e bibliotecas mais eficientes, como o **C++** e o **OpenGL**. Porém, dado que a API **WebGL** é baseada em **OpenGL** (mais especificamente **OpenGL ES 2.0/3.0**), há uma série de conceitos comuns, pelo que o **WebGL** pode ser visto como uma boa plataforma de entrada nas versões atuais da tecnologia **OpenGL**.

No contexto de **CG**, iremos então recorrer ao **WebGL** e a **JavaScript** para criar pequenas aplicações gráficas que ilustram os conceitos básicos de Computação Gráfica, e permitam a experimentação prática dos mesmos, e que podem ser corridas em *web browsers* recentes (**que suportem WebGL**).

Para apoiar o desenvolvimento, a biblioteca **WebCGF (Web Computer Graphics @ FEUP)** foi desenvolvida pelos docentes da unidade curricular e por alguns estudantes, especificamente para as aulas de Computação Gráfica, com o objetivo de abstrair alguma da complexidade de inicialização e criação de funcionalidades de suporte, permitindo a focalização nas componentes relevantes aos conceitos de Computação Gráfica a explorar.

Para o desenvolvimento e **controlo de versões** dos trabalhos práticos, será utilizado o sistema de Gitlab da UP ([gitlab.up.pt](https://gitlab.up.pt)). O [documento de informações gerais de funcionamento de CG - Parte 2](#) tem informações mais detalhadas sobre a configuração inicial deste sistema, e deve ser lido com atenção antes de prosseguir.

# Recursos disponibilizados

## A biblioteca 'WebCGF'

### Estrutura

A biblioteca [WebCGF](#) (Web Computer Graphics @ FEUP) - tem como classes principais as seguintes:

- **CGFApplication (+)** - Gere as questões genéricas de inicialização da aplicação e bibliotecas de apoio, e interliga os outros componentes
- **CGFscene (\*)** - É responsável pela inicialização, gestão e desenho da cena
- **CGFinterface (\*)** - É usada para construir e gerir a interface com o utilizador; pode aceder ao estado interno da cena para, por exemplo, ativar ou desativar funcionalidades (p.ex. luzes, animações). Na sua base está a biblioteca **dat.GUI**:

<https://github.com/dataarts/dat.gui/blob/master/API.md>

A biblioteca contempla também as seguintes classes que representam entidades que podem integrar uma cena (lista não exaustiva):

- **CGFObject (\*)** - Representa um objeto genérico, que deve implementar o método **display()**; os objetos a serem criados devem ser sub-classes de **CGFObject**
- **CGFlight (+)** - Armazena alguma informação associada a uma fonte de luz (poderá ser estendida por sub-classes para implementar características adicionais)
- **CGFcamera (+)** - Armazena a informação associada a uma câmara

Para a correta execução dos trabalhos, espera-se que **venham a estender as classes assinaladas com (\*)**, de forma a implementar as cenas, interface e objetos requeridos em cada um dos trabalhos, tal como exemplificado na secção seguinte.

As classes assinaladas com **(+)** são **classes utilitárias de exemplo, não exaustivas**, a instanciar para facilitar a gestão e armazenamento das entidades associadas (podendo no entanto ser estendidas por sub-classes, se desejarem acrescentar funcionalidades). A biblioteca inclui ainda alguns objetos pré-definidos, como é o caso dos eixos (**CGFaxis**), e mais algumas classes auxiliares, mas que não deverão ser necessárias para os primeiros trabalhos práticos de CG.

### Interação de base

Em termos de interação, por omissão é possível manipular a vista utilizando o rato da seguinte forma:

**Botão esquerdo** - rodar a cena em torno da origem

**Roda de scroll** - aproximar/afastar; em alternativa, pode ser utilizado **CTRL + Botão esquerdo**

**Botão direito** - "deslizar" a câmara lateralmente

## Código de base do exercício

O código de base fornecido para o exercício estende as classes referidas na secção anterior de forma a implementar o desenho de uma cena muito simples.

A classe **MyScene** estende **CGFscene**, e implementa os métodos **init()** e **display()**:

- **init()** : Contém o código que é executado uma única vez no início, depois da inicialização da aplicação. É aqui que tipicamente se inicializam variáveis, criam objetos ou são feitos cálculos intensivos cujos resultados podem ser armazenados para posterior consulta.
- **display()** : Contém o código que efetivamente desenha a cena repetidamente. Este método será o foco deste primeiro trabalho.

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

Em particular, o código de exemplo contido no método **display()** está dividido em três secções:

- Inicialização do fundo, câmara e eixos
- Transformações geométricas
- Desenho de primitivas


Este trabalho focar-se-á no desenho de primitivas e na sua organização. A declaração e uso de transformações geométricas, e na combinação de ambas para produzir uma geometria composta será abordada no próximo trabalho prático.

A classe **MyInterface** estende **CGFinterface**, e implementa o método **init()**:

- **init()** : Nesta função, a interface gráfica é inicializada. É possível adicionar *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>

## Trabalho prático

Os próximos pontos descrevem os tópicos abordados durante esta aula prática, assim como as tarefas a realizar.

Algumas das tarefas estão anotadas com o ícone  (captura de imagem). Nestes pontos deverão capturar uma imagem da aplicação para disco (p.ex. usando Alt-PrtScr ou Win+Shift+S em Windows ou Cmd-Shift-3 em Mac OS X para capturar para a clipboard e depois gravar para ficheiro num utilitário de gestão de imagens à escolha). As imagens devem ser gravadas na pasta “screenshots” do trabalho, com o nome no formato **"CG-t<turma>g<grupo>-tp1-n.png"**, em que **turma** e **grupo** corresponde ao número de turma e grupo definido no ficheiro de grupos TP, e **n** corresponde ao número fornecido no exercício (p.ex. **"CG-t01g01-tp1-1.png"**). Devem ser incluídas também no ficheiro README.md, indicando o ponto a que dizem respeito (ver exemplo no ficheiro).



Nas tarefas assinaladas com o ícone (código), devem:

- Fazer um **commit no Git** e associar uma **tag** no formato “**tp1-n**” (ex: “tp1-1”) (ver [documento de informações gerais de funcionamento de CG - Parte 2](#) ).

## 1. Utilização da Interface Gráfica (GUI)

A interface gráfica (GUI) permite interagir com a cena criada, fornecendo diferentes tipos de elementos para controlar determinados aspetos da dita cena.

A GUI surge como um menu colapsável na página da cena, no canto superior direito. No código de base fornecido com o trabalho prático, existe um controlador do tipo **checkbox** que controla a visibilidade do eixo de coordenadas.

Para esse efeito, a cena foi preparada para mostrar condicionalmente os eixos, na classe **MyScene**, onde se:

- Inicializa uma variável na cena - *displayAxis* - na função **init()** com um valor booleano
- Implementa a funcionalidade que altera a cena de acordo com o valor dessa variável.

A interface gráfica é definida na classe **MyInterface** (sub-classe de *CGFInterface*), da seguinte forma:

- Inicializa o objeto da GUI - classe **dat.GUI** (ver link fornecido anteriormente) - na função **init()**
- Adiciona um controlador à GUI, associado à cena e à variável *displayAxis* da mesma. Dado que esta foi inicializada com um valor booleano, a GUI **infere** que deverá ser representada por uma checkbox.

## 2. Geometria básica e estruturação

O desenho de objetos em **WebGL** e nas versões modernas de **OpenGL** é tipicamente baseado na definição de um conjunto de triângulos com um conjunto de características associadas.

Esses triângulos são definidos por um conjunto de vértices (e possivelmente algumas características associadas a cada vértice), e a forma como os vértices se ligam para formar os triângulos.

No código de base fornecido com o trabalho prático é providenciado o código necessário para criar um **losango (MyDiamond.js)**, e incluído na cena para que possa ser visível.

Considere que esse losango é definido pelos vértices A, B, C e D, que formam dois triângulos de vértices ABC e DCB (Figura 1).

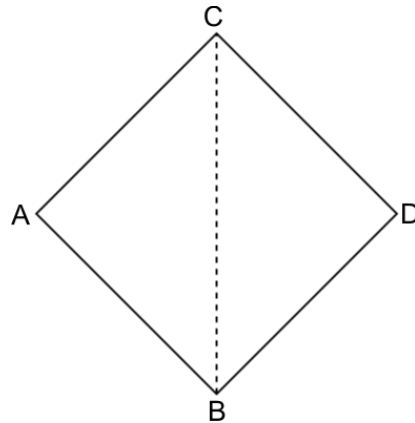


Figura 1: Geometria de exemplo (quadrilátero).

Para criarmos essa geometria, temos que, em primeiro lugar, criar um *array* de vértices com as coordenadas dos quatro cantos.

```
vertices = [  
    xA, yA, zA,  
    xB, yB, zB,  
    xC, yC, zC,  
    xD, yD, zD  
];
```

Em seguida, devemos indicar como estes vértices serão ligados entre si para formar triângulos.

Para isso, criamos um novo *array* que indique, através de índices referentes à ordem dos vértices, como agrupá-los de três a três. Neste caso, sendo os triângulos definidos pela ordem ABC e BDC, teremos:

```
indices = [  
    0, 1, 2,  
    1, 3, 2  
];
```

O uso de índices permite reduzir a quantidade de informação necessária para definir a geometria. Em vez de repetirmos 3 coordenadas na lista de vértices quando o mesmo vértice é usado mais do que uma vez, apenas repetimos o seu índice na lista de índices.

Quanto maior for a geometria e o número de vértices partilhados (algo bastante comum em modelos 3D compostos por uma malha de triângulos), mais benefício há em usar os índices para representar a conectividade.

Tendo esta informação definida, o desenho efetivo da geometria implica passar a informação assim declarada em **JavaScript** para buffers do **WebGL** (já alocados na memória gráfica), e instruir o mesmo para os desenhar considerando a sua conectividade como sequências de triângulos.

Na **WebCGF**, a complexidade desta fase final do desenho está encapsulada na classe **CGFObject**. Dessa forma, para criar um determinado objeto 3D, podemos simplesmente:

- criar uma sub-classe da **CGFObject**, p.ex. **MyObject**
- implementar o método **initBuffers()**, onde:
  - Declaramos os arrays acima referidos;
  - Invocamos a função **initGLBuffers** para a informação ser passada para o **WebGL**.

Na nossa cena:

- Criar e inicializar uma instância do novo objeto no método **init()** da cena;

- Invocar o método **display()** dessa instância do objeto no método **display()** da cena. Na classe **MyDiamond** encontram um exemplo completo, que servirá de base para a criação de outras formas.

## Exercícios

### Exercício 1

Seguindo o exemplo de **MyDiamond**, crie uma nova subclasse de **CGFobject** chamada **MyTriangle**, num ficheiro **MyTriangle.js**, que define um triângulo retângulo no plano XY, com lado de 2 unidades, demonstrado na Figura 2, onde o ponto vermelho representa a origem (0,0,0). **Acrescente** a importação deste novo ficheiro (e dos criados seguidamente) no ficheiro da classe **MyScene**.

**NOTA:** Respeite os nomes das classes indicados nesta alínea e em todas as outras.

Na GUI definida na classe **MyInterface**, adicione dois controladores do tipo **checkbox**, que deverão controlar a visibilidade do losango já apresentado na cena, assim como o triângulo criado na alínea anterior.

Crie uma nova subclasse de **CGFobject** chamada **MyParallelogram**, que cria um paralelogramo como demonstrado na Figura 3, com o vértice mais à esquerda na origem da cena (0,0,0). A sua altura é de 1 unidade, e a largura total é de 3 unidades. Deverá ser *double-sided*, isto é, visível em ambos lados (sugestão: deve ser explorada a repetição e ordem dos índices).

Adicione outra **checkbox** para controlar a visibilidade do paralelogramo.

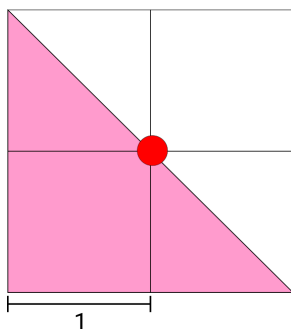


Figura 2: Triângulo retângulo (**MyTriangle**)

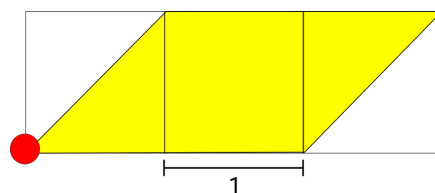


Figura 3: Paralelogramo inversível (**MyParallelogram**)

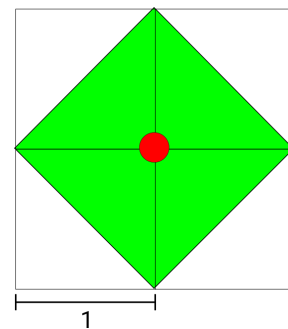




Figura 4: Losango já criado (**MyDiamond**)

### Exercício 2

Crie as peças adicionais representadas nas figuras seguintes, 5 e 6. (  1) (  1)

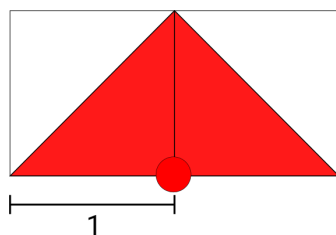


Figura 5: Triângulo rectângulo (pequeno)  
(MyTriangleSmall)

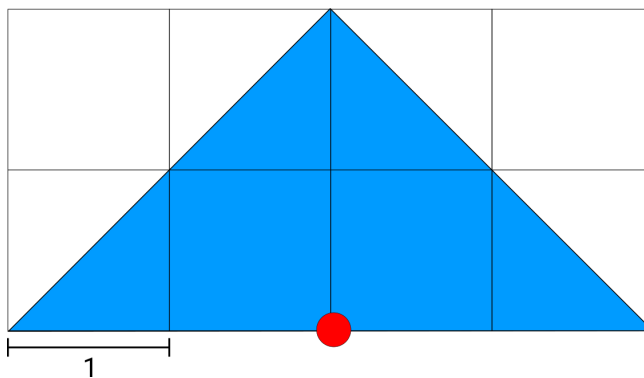


Figura 6: Triângulo rectângulo (grande)  
(MyTriangleBig)

## Checklist

Até ao final do trabalho deverá ter criado as seguintes imagens e *commits* do código, **respeitando estritamente a regra dos nomes**, e incluir as imagens no README.md do trabalho:



Imagens

(1):

n=1

(nomes

do

tipo

"CG-t<turma>g<grupo>-tp1-n.png")



GIT Commits/Tags (1): "tp1-1"