

## Computação Gráfica (L.EIC)

Tópico 2



### Geometria básica e transformações geométricas

## Objetivos

- Utilizar matrizes de transformação geométrica para manipular/modificar formas geométricas
- Utilizar funcionalidades da **WebCGF** para facilitar a definição e aplicação das transformações geométricas
- Criar objetos compostos

## Trabalho prático

Nesta aula prática, criaremos um novo projeto que utilizará os objetos criados na aula anterior para representar uma figura Tangram (<https://en.wikipedia.org/wiki/Tangram>). Cada grupo terá que replicar uma figura, **identificada no ficheiro dos grupos e fornecida no moodle**.

À semelhança do trabalho anterior, será necessário fazerem capturas de ecrã em alguns pontos do enunciado, bem como assinalar versões do código no *Git* com *Tags*. Os pontos onde tal deve ser feito estão assinalados ao longo do documento e listados numa check list no final deste enunciado, sempre assinalados com os ícones  (captura de uma imagem) e  (tags). No final deve ser submetido um zip no Moodle com a última versão.

## Preparação do Ambiente de Trabalho

Para a resolução de exercícios desta aula prática, serão usados os objetos desenvolvidos na aula prática anterior. Devem copiar os ficheiros da pasta **tp1** para a pasta **tp2**, mantendo separado o desenvolvimento dos diferentes trabalhos.

### 1. Matrizes de transformações geométricas

Num sistema de coordenadas 3D, as três transformações geométricas básicas - Translação, Rotação e Escalamento - são representáveis por matrizes quadradas, com 4 linhas e 4 colunas. A concatenação de um conjunto de transformações geométricas obtém-se pela multiplicação das matrizes respetivas.

Em **OpenGL/WebGL**, a ordem dos valores dos vetores que representam uma matriz de transformação geométrica corresponde ao transposto das matrizes definidas matematicamente; assim, ao pretender-se uma matriz com o conteúdo seguinte:

```
| 0 1 2 3 |
| 4 5 6 7 |
| 8 9 A B |
| C D E F |
```

deve declarar-se-se, em **OpenGL/WebGL**, da seguinte forma:

```
m = [
    0, 4, 8, C,
    1, 5, 9, D,
    2, 6, A, E,
    3, 7, B, F
];
```

Na cena **MyScene** utilizada na aula anterior, o método **display()** contém uma matriz de transformação geométrica que permite mudar a escala dos objectos desenhados a seguir. Essa matriz é passada para o comando **this.multMatrix(...)**. O método **multMatrix** da **CGFscene** permite acumular várias transformações à perspetiva da câmara, de forma a que os objetos sejam transformados relativamente à mesma.

### 2. Funções WebCGF para transformações geométricas

A **WebCGF** fornece na sua classe **CGFscene** um conjunto de instruções que permitem manipular transformações geométricas e aplicá-las à perspetiva da câmara, baseadas na biblioteca **gl-matrix.js**; com elas não é necessário declarar as matrizes. São elas:

- CGFscene.translate(x, y, z):** Gera uma matriz de translação e aplica-a;
- CGFscene.rotate(ang, x, y, z):** Gera uma matriz de rotação de *ang* **radianos** à volta do eixo (x, y, z) e aplica-a;
- CGFscene.scale(x, y, z):** Gera uma matriz de escalamento nas três direções e aplica-a; **Nota:** nenhum dos componentes de **scale()** deve ser zero, caso contrário a geometria será reduzida a algo planar, com efeitos indefinidos.

Para efetuar a conversão entre radianos e graus utilize a constante **Math.PI**. Para criar a matriz de rotação utilize as funções trigonométricas **Math.cos(ang)** e **Math.sin(ang)**.

seção correspondente do moodle.

**Nota:** Considere que a figura final deverá estar aproximadamente centrada na origem (0,0,0), podendo escolher o vértice mais central na sua figura para alinhar com a origem. As transformações geométricas aplicadas em cada alínea deverão ter esse ponto de referência (sugere-se que faça um rascunho em papel ou numa aplicação de desenho para determinar as orientações e posições da cada peça).

- De acordo com a figura de Tangram fornecida ao seu grupo, crie uma instância da classe **MyDiamond** e coloque-o em cena no plano XY utilizando **operações de multiplicação de matrizes tal como descrito na secção 1 (ou seja, declarando as matrizes e utilizando a função *multMatrix()*** ). Coloque o objeto tendo em conta que a figura final Tangram deverá estar aproximadamente centrada na origem (0,0,0).
- Recorrendo às instruções de transformações geométricas descritas na secção 2, coloque as restantes peças na cena. Todas estas peças deverão ser colocadas usando transformações geométricas tendo como ponto de partida a origem. Para tal, utilize as instruções ***CGFscene.pushMatrix()*** e ***CGFscene.popMatrix()*** para colocar o ponto de desenho na origem para cada objeto.
- Crie uma nova classe **MyTangram**, subclasse de **CGFObject**, que será um objeto composto que englobará os objetos criados nos exercícios anteriores. Crie a função ***MyTangram.display()*** para onde deve mover e ajustar o código respeitante à figura que criou na alínea 2. Na **MyScene** deve criar uma instância de **MyTangram** na função ***init***, e na função ***display*** da **MyScene** deve invocar a função ***display*** do objeto criado (em substituição do código do desenho das peças que moveu para **MyTangram**). Na captura de ecrã deverá mostrar a janela com a cena em WebGL lado a lado com outra janela com a imagem Tangram original, para facilitar a comparação final.





### 3. Geometria tridimensional - Cubo Unitário

Até agora, foram apenas consideradas superfícies coplanares. Neste exercício pretende-se a criação de um cubo unitário, ou seja, um **cubo centrado na origem e de aresta unitária**, ou seja, com coordenadas entre (-0.5, -0.5, -0.5) e (0.5, 0.5, 0.5), construído **com uma única malha de triângulos**.

Comece por comentar na função ***display()*** o código relacionado com o desenho do **MyTangram**, de forma a ter o método ***display()*** apenas a desenhar os eixos (ou seja, comente todo o código entre o desenho dos eixos, e o fim do método ***display()***).

#### Exercícios

- Crie um ficheiro **MyUnitCube.js** e defina nesse ficheiro a classe **MyUnitCube** como subclasse da **CGFObject** (pode usar uma cópia do código do **MyDiamond.js** como ponto de partida). Essa classe deve definir na função ***initBuffers*** os 8 vértices do cubo, e a conectividade entre eles de forma a formar os triângulos que constituem as faces quadradas do cubo. Recomenda-se que sejam inseridos comentários identificando os vértices e as faces que estão a ser definidas.
- Deve importar no ficheiro da classe **MyScene** o novo ficheiro **MyUnitCube.js**.
- Inclua um novo objeto do tipo **MyUnitCube** na função ***init*** da **MyScene**, e invoque o método ***display()*** de **MyUnitCube** no método ***display()*** da **MyScene**. Execute a aplicação. Deve ter um cubo unitário centrado na origem.
- Reative a instância da classe **MyTangram** novamente na cena, descomentando o código respetivo. Aplique transformações geométricas no cubo criado de forma a que este seja colocado por trás da figura Tangram criada, como uma base.
- Considerando o conjunto composto pela base e figura de Tangram, aplique transformações geométricas de forma a que seja colocado paralelo ao plano XZ, com o vértice superior

esquerdo da base na origem (0,0,0), ( 2) ( 2).

### 4. Geometria composta - Cubo composto por planos



Crie um novo cubo unitário utilizando planos desenhados várias vezes para definir as faces.

- Crie uma nova classe **MyQuad** como subclasse de **CGFObject**, que representará um quadrado unitário centrado na origem (0,0,0).
- Crie uma nova classe **MyUnitCubeQuad**, que será composta por um objeto da classe **MyQuad**. Na função ***display()*** desta classe, utilize as funções de transformações geométricas para desenhar o objeto de **MyQuad** como as seis faces do cubo unitário.
- Observe o resultado, substituindo o cubo criado no exercício anterior por este novo cubo

na cena (aplique as mesmas transformações geométricas).( 3) ( 3)

## Checklist

Até ao final do trabalho deverá ter criado as seguintes imagens e *commits* do código, **respeitando estritamente a regra dos nomes**:

-  Imagens (3): 1, 2, 3 (nomes do tipo "cg-t-turma-g<grupo>-tp2-n.png")
-  GIT Commits/Tags (3): 1,2,3 (Git Tags correspondentes: "tp2-1", "tp2-2" e "tp2-3")

Deve também submeter no final um **zip no moodle com o nome no formato "cg-t-turma-g<grupo>-tp2.zip"**