

# Computer Graphics (MIEIC)

## Topic 2

### *Basic geometry and geometric transformations*


## Objectives


- Use matrices to manipulate / modify geometric shapes
- Use **WebCGF** functionalities to facilitate the definition and application of geometric transformations
- Create composite 3D objects

## Practical work

In this class, we will create a new project where we will use the objects created in the previous class to represent a Tangram figure (<https://en.wikipedia.org/wiki/Tangram>). Each group will have to replicate a figure, identified in the group file and provided in moodle.

The following points describe the topics covered during this practical class, as well as the tasks

to be performed. Some of the tasks are noted with the icon  (image capture). At these points, you should capture and save an image of the application in the browser (e.g., using Alt-PrtScr on Windows or Cmd-Shift-3 on Mac OS X to capture to the clipboard and then save to file in an image management application of your choice). At the end of each class, students must rename the images to the format "**ex2-t<class> g<group>-n.png**", where **class** corresponds to the class number (e.g., 2MIEIC01 is class '**01**'), and **group** corresponds to student group number (defined in the TP group file in Moodle), and **n** corresponds to the number provided in the exercise (eg "**ex2-t1g01-1.png**").

In tasks marked with the icon  (code), students must create a **.zip file from the folder that contains your code (typically in the 'ex2' folder, if you have code being used in other folders, include it too)**, and name it "**ex2-t<class>g<group>-n.zip**", (with class, group and provided number identified as described above "**ex2-t1g01-1.zip**").

At the end (or throughout the work), one of the elements must submit the files via Moodle, through the link provided for that purpose. Only one member of the group will need to submit the work.

# Preparation of the Work Environment

For solving exercises in this practical class, the objects developed in the previous practical class will be used. It is suggested that you create a copy of the folder **ex1**, and rename the copy to **ex2**, keeping the development of the different lessons separate. It should have a folder structure similar to the following:

```
\ CGRA          (where the server is started)
  \ lib
  \ ex1
  \ ex2
```

## 1. Matrices of geometric transformations

In a 3D coordinate system, the three basic geometric transformations - Translation, Rotation and Scaling - are representable by square matrices, with 4 rows and 4 columns. The concatenation of a set of geometric transformations is obtained by multiplying the respective matrices.

In **OpenGL / WebGL**, the order of the values of the vectors that represent a geometric transformation matrix corresponds to the transposition of the matrices defined mathematically; thus, when aiming for a matrix with the following content:

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

it must be declared in **OpenGL / WebGL**, as follows:

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

In the **MyScene** used in the previous class, the **display()** method contains a geometric transformation matrix that allows you to change the scale of the objects drawn below. This array is passed to the **this.multMatrix(...)** function. The **multMatrix()** method from **CGFscene** allows accumulating several transformations from the camera's perspective, so that objects are transformed in relation to it.

## 2. WebCGF functions for geometric transformations

The **WebCGF** library provides in its **CGFscene** class a set of instructions for manipulating geometric transformations and apply them to the camera's perspective, based on **gl-matrix.js** library; these instructions do not require declaring matrices for defining transformations.

They are:

- **CGFscene.translate** (*x, y, z*): Generates a translation matrix and applies it;
- **CGFscene.rotate** (*ang, x, y, z*): Generates a rotation matrix of *ang* radians around the axis (*x, y, z*) and applies it;
- **CGFscene.scale** (*x, y, z*): Generates a scaling matrix in the 3 directions and applies it;  
**Note:** the **scale()** components must be non-zero, otherwise the geometry will be reduced to something planar, with undefined effects.

To convert between radians and degrees, use the constant **Math.PI**. To create the rotation matrix use the trigonometric functions **Math.cos (ang)** and **Math.sin (ang)**.

### Exercises



The following exercises will focus on applying geometric transformations to the objects created in the previous class in order to recreate a Tangram figure, provided to each group.

See the Tangram number on the group sheet and obtain the corresponding image in the image directory, available on Moodle.

**Note:** Consider that the final figure should be approximately centered on the origin (0,0,0), being able to choose the most central vertex in your figure to align with the origin. The geometric transformations applied in each exercise must have this reference point (as a suggestion, create a sketch on paper or in a drawing application to determine the orientations and positions of each piece).

1. According to the Tangram figure provided to your group, create an instance of the **MyDiamond** class and place it on the XY plane using **matrix multiplication operations** as described in section 1 (i.e. declaring the matrices and using the *multMatrix()* function). Place the object taking into account that the final Tangram figure should be approximately centered on the origin (0,0,0).
2. Using the geometric **transformation instructions provided by the WebCGF library** described in section 2, place the remaining pieces on the scene. All of these pieces must be placed using geometric transformations with the origin as a starting point. For this reason, use the **CGFscene.pushMatrix()** and **CGFscene.popMatrix()** instructions to place the drawing point at the origin for each object.
3. Create a new **MyTangram** class, as subclass of **CGFObject**, which will be a composite object that will include the objects created in previous exercises. Create the **MyTangram.display()** function, where you should place and adjust the code that draws the objects in the scene as defined in the previous exercises. In **MyScene**, create an

instance of **MyTangram** in the *init()* function, and in its *display()* function, invoke the *display* function of **MyTangram** (replacing the code previously used to draw the Tangram that was moved into **MyTangram**). In the screenshot, you should show the window with the scene in WebGL side by side with another window with the original

Tangram image, to facilitate the final comparison. (  1) (  1)



### 3. Three-dimensional geometry - Single Cube

Until now, only coplanar surfaces have been considered. In this exercise we intend to create a unitary cube, that is, a **cube centered on the origin and with a unitary side**, that is, built **with a single mesh of triangles** between the coordinates  $(-0.5, -0.5, -0.5)$  and  $(0.5, 0.5, 0.5)$ , .

Start by commenting in the *MyScene.display()* the code related to **MyTangram**, so that the *display()* function only draws the axes (that is, comment the code related to objects and geometric transformations between drawing of the axes and the end of the method *display()*).

#### Exercises

1. Create a file **MyUnitCube.js** and define the **MyUnitCube** class, as a subclass of **CGFObject** (you can use a copy of the code in **MyDiamond.js** as a starting point). This class must define in the function *initBuffers()* the 8 vertices of the cube, and the connectivity between them in order to form the triangles that make up the square faces of the cube. The use of comments is recommended to help identify the vertices and faces being defined.
2. You must add in the *main.js* file the inclusion of the new file **MyUnitCube.js**, in the line where the other files of the project are included.
3. Initialize a new **MyUnitCube** object in the *init()* function of **MyScene**, and invoke *display()* function of **MyUnitCube** in the *display()* function of **MyScene**. Run the application. It should have a unit cube centered on the origin.
4. Reactivate the instance of the class **MyTangram** again in the scene, uncommenting the respective code. Apply geometric transformations to the created cube so that it is placed behind the created Tangram figure, as a base.
5. Considering the set consisting of the base cube and Tangram figure, apply geometric transformations to the total set so that it is placed parallel to the XZ plane, with the upper

left corner of the base at the origin  $(0,0,0)$ . (  2) (  2).



## 4. Composite geometry - Cube composed of planes (Extra)

As an extra exercise, create a new unit cube using planes drawn several times to define the faces.

1. Create a new class **MyQuad** as a subclass of **CGFObject**, which will represent a unit square centered on the origin (0,0,0).
2. Create a new class **MyUnitCubeQuad**, which will be composed of an object of the class **MyQuad**. In the *display()* function of this class, use the geometric transformation functions to draw the MyQuad object as the six faces of the unit cube.
3. Observe the results by replacing the previously defined base cube (applying the same geometric transformations).

## Checklist

Until the end of the work you should save and later submit the following images and versions of the code via Moodle, **strictly respecting the naming rule**:

-  **Images (2): 1,2 (named as "ex2-t<class>g<group>-n.png")**
-  **Code in zip file (2): 1,2 (named as "ex2-t<class>g<group>-n.zip")**