

Computer Graphics (MIEIC)

Topic 4

Application of Textures

Objectives

- Define texture coordinates appropriately.
- Explore the different texture application modes.
- Combine the use of materials with textures to achieve a realistic look.


Preparation of the Work Environment


Students should download the code available for this topic on Moodle, and place the **ex4** folder contained in the .zip file at the same folder as the previous lessons.

If you developed the **MyUnitCubeQuad** class as an extra exercise for practical class 2 (**ex2**), you can copy its file to the folder of this lesson. The **MyQuad** class used to create the faces of the composite cube is provided in the base code of this practical lesson, so you may choose to use that file (check if the classes are consistent and compatible).

Practical work

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 "**ex4-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 (e.g., "**ex4-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 'ex4' folder, if you have code being used in other folders, include it too)**, and name it "**ex4-t<class>g<group>-n.zip**", (with class, group and provided number identified as described above "**ex4-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 is required to submit the work.

1. Application of textures

Texture mapping is a way of assigning information stored in *bitmap* format to different areas of the drawn 3D surfaces. One of its most common uses is to map sections or an entire image to a geometry, in order to add visual detail without increasing the number of vertices or adding complexity to the geometry (other types of mapping include *bump mapping* and *normal mapping*, which will not be explored in this lesson).

In the context of OpenGL / WebGL, a two-dimensional texture can result from loading a *bitmap* image, which is loaded into a buffer and can later be accessed using two dimensions commonly identified as *s* and *t* (or in other contexts like *u* and *v*), whose values are normalized between 0 and 1 (see fig. 1).

Note: It is important to note that the representation of the axes of texture coordinates is presented differently between the theoretical and practical classes. Specifically, the origin (0,0) corresponds to the *lower-left corner* in the theoretical class, since the *loader* texture **OpenGL** starts at that point. However, in the context of practical classes (**WebGL**) the origin corresponds to the *upper-left corner* of the image.

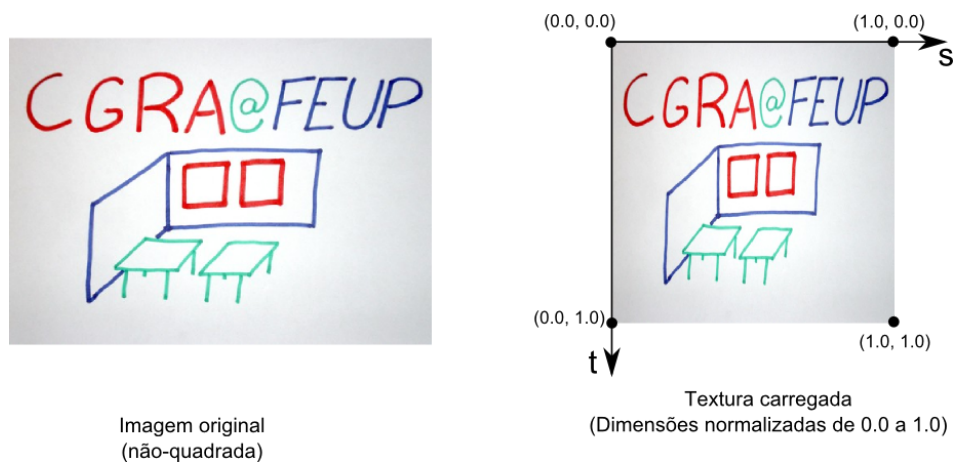


Figure 1: Image and corresponding loaded texture

A previously loaded texture can be applied to a given geometry - in the most basic case, a triangle - by mapping between the vertices of the geometry and the points of the image that will be associated with them, defining for each vertex a texture coordinate (see **fig. 2, a)** and **b)**).

Conceptually, we can consider that we are defining the "image clipping" that will be applied to the triangle in question, and if the "clipping" does not have the same proportions as the original triangle, the image will be distorted accordingly (see **fig. 2, c)** and **d)**).

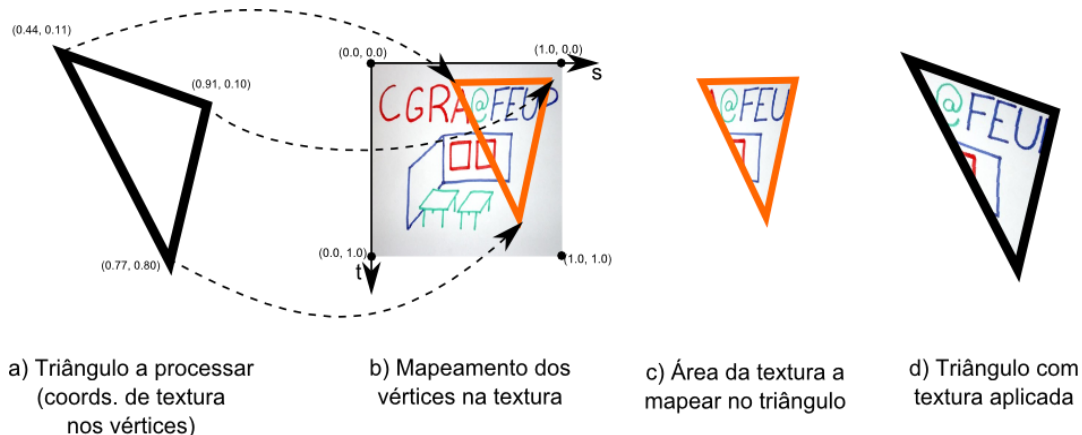


Figure 2: Mapping between triangle and texture defining texture coordinates by vertex.

2. Texture Wrapping Modes

In the examples explored so far, the texture coordinates associated with each vertex are in the normalized range of 0.0 to 1.0. However, it is possible to indicate values outside this range, when we want, for example, to have several repetitions of the same image in a polygon, or to map the entire image in only a part of the polygon.

The way in which values outside the range [0..1] are used when applying a texture is controlled by setting the **wrapping** mode. The supported **wrapping** modes vary slightly between versions of OpenGL, and in the case of WebGL the possible modes are **'REPEAT'**, **'CLAMP_TO_EDGE'** and **'MIRRORED_REPEAT'**.

Figure 3 shows some examples of how to manipulate the texture coordinates in each mode to obtain different effects. Note that the **wrapping** mode can be different in the two dimensions **s** and **t**.

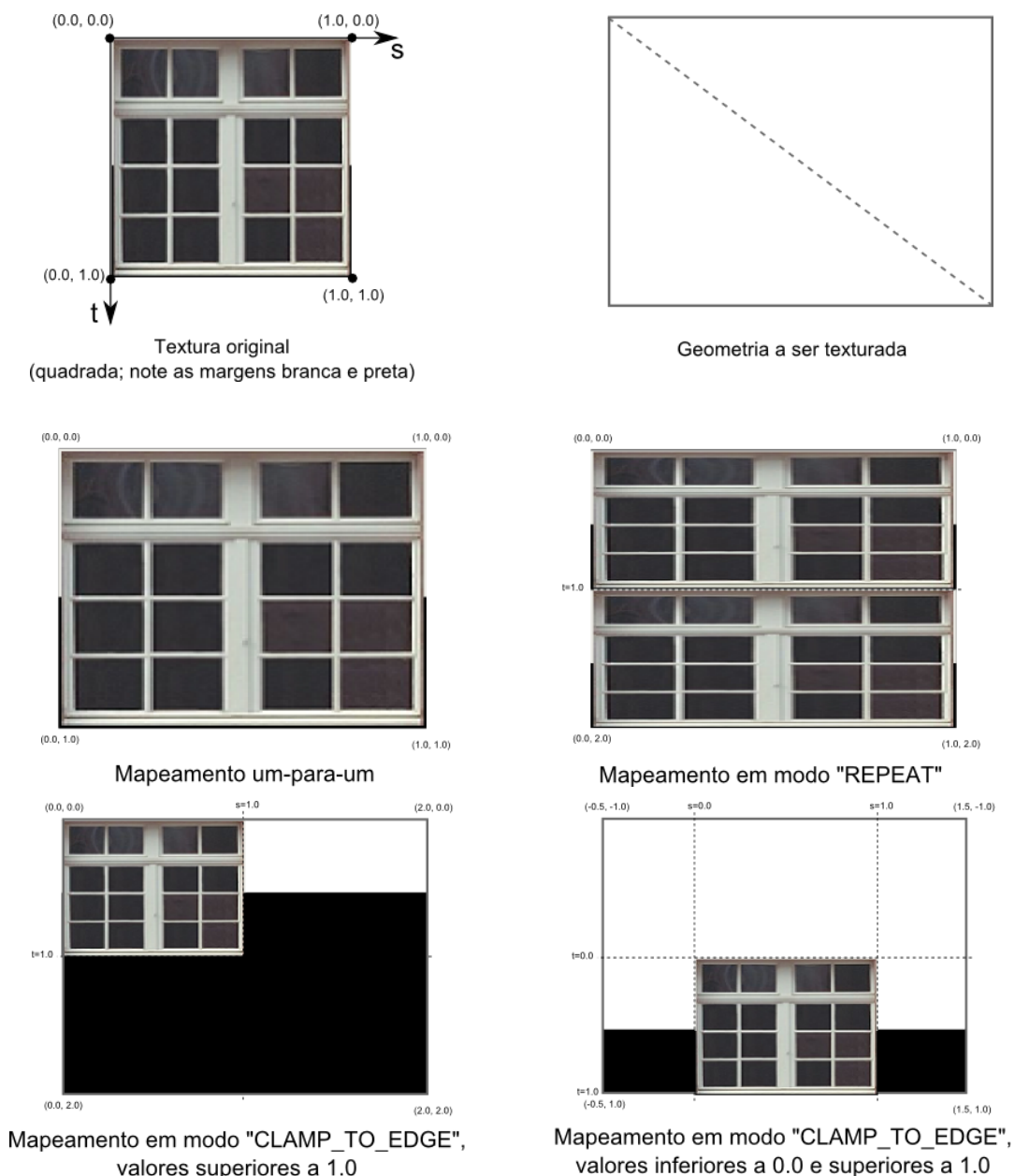


Figure 3: Application of a texture using repetition or *clamping*.

Note: Notice how in the **'CLAMP_TO_EDGE'** mode, the image margins are extended along the coordinate zones outside the range [0..1].

Experiences

In this lesson, we intend to explore the definition of texture coordinates in objects and the different wrapping modes for applying different textures.

The provided scene contains an object of type **MyQuad** to which a material of type **CGFappearance** has been applied called **quadMaterial**. The interface contains a dropdown menu for choosing textures (starting without selection), two dropdowns menus for selecting wrapping modes, and sliders for controlling the texture coordinates associated with the four corners of the geometry.

1. Select the **'Board'** texture in the interface. With the wrap mode of the S and T coordinates in **'Repeat'**, change the values of the texture coordinates so that you get three columns and two lines of the image on the object.
2. Restart the scene, and select the **'Floor'** texture in the interface. Keeping the wrap mode at **'Repeat'**, change the texture coordinate values so that the image is inverted vertically.
3. Change the wrap mode of the S and T coordinates to **'Clamp to Edge'** and see the differences in texture mapping.
4. Restart the scene, and select the **'Window'** texture in the interface. With the wrap mode of the S and T coordinates in **'Clamp to Edge'**, change the values of the texture coordinates so that the window appears centered on the geometry, occupying half the total height and width, as shown in figure 5.

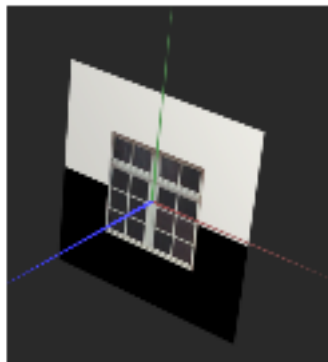


Figure 5: Window centered on the object **MyQuad**.

5. Try switching wrapping modes to S and T and observe the differences.

Exercises


Include in the exercise folder (**ex4**) the files of the classes **MyTangram** (and the classes of its pieces). Create a checkbox that allows you to hide the object **MyQuad** and its material **quadMaterial** so that the scene is empty.

Applying textures to Tangram

1. Create a new material at the initialization of the scene to be applied in the *display* function of **MyTangram** class, to the **MyDiamond** object. Define the image 'tangram.png' as the texture of this material (see code examples).
2. Set texture coordinates for **MyDiamond** so that the edges of the diamond piece in the image match the edges of the object. To help in the process of determining the texture coordinates to be assigned to each vertex, we suggest that you open a copy of 'tangram.png' in an image editor to annotate the S and T axes as seen in figure 1, identify the vertices of the diamond in this figure, and determine its coordinates in that space S, T (values between 0.0 and 1.0). The texture coordinates are defined by creating in the *initBuffers* function of the object a new array called **this.texCoords**.

In this array, a pair of S,T coordinates must be defined for each vertex previously declared in the **this.vertices** array:


```
    this.texCoords = [  
        s0, t0,  
        s2, t2,  
        ...  
        sn, tn  
    ];
```

3. Repeat the previous two steps for each of the other Tangram pieces, so that each piece is mapped to its representation of the image. (1 ) (1)

Applying textures to a cube composed of planes



4. Create a new class **MyUnitCubeQuad**, which defines a new unit cube using an object of type **MyQuad**, drawn several times to define the faces. Use the geometric transformation functions to draw the six faces, in the **display()** function of **MyUnitCubeQuad**.
Note: If you already created this class in the extra exercise of practical class 2 (**ex2**), you can copy the corresponding file to the folder of this class (**ex4**).
5. Apply the texture 'mineSide.png' to the side faces of **MyUnitCubeQuad**, and the textures 'mineTop.png' and 'mineBottom.png' to the top and bottom faces, respectively.
6. Notice how the textures are poorly defined. This is due to the fact that they originally had dimensions of 16x16 pixels, but are actually covering a much larger drawing area. By default, in these cases a **linear interpolation** is applied to the colors (**LINEAR FILTERING**, see filtering on the theoretical slides).

Find in the sample code the line of code that allows you to change the type of filtering used (originally commented on **display()** function of the scene). Use it for the cube textures to achieve the desired effect, applying this mode after activating the texture and before drawing the

faces to be affected. (2 ) (2)

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 "ex4-t<class>g<group>-n.png")
-  **Code in zip file (2):** 1,2 (named as "ex4-t<class>g<group>-n.zip")