FACULTY OF COMPUTER AND INFORMATION SCIENCES, AIN SHAMS UNIVERSITY

Lab 6

Texture Mapping

OBJECTIVES:

1. **Be able to add Textures to objects in the scene.**
2. **Be able to control Textures behavior.**
3. **Add Light Maps using multi texturing.**

1. BE ABLE TO ADD TEXTURES TO OBJECTS IN THE SCENE

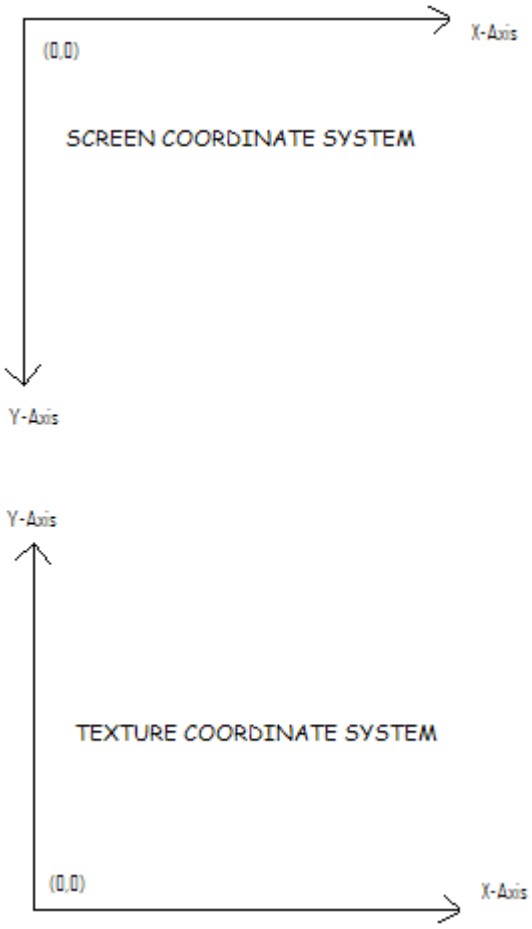
In this section we will go through the basic steps required for adding a texture to an object in the scene. What is needed is as follows:

1- Load the image's data in the memory.

2- Set the loaded data for the current texture in Open

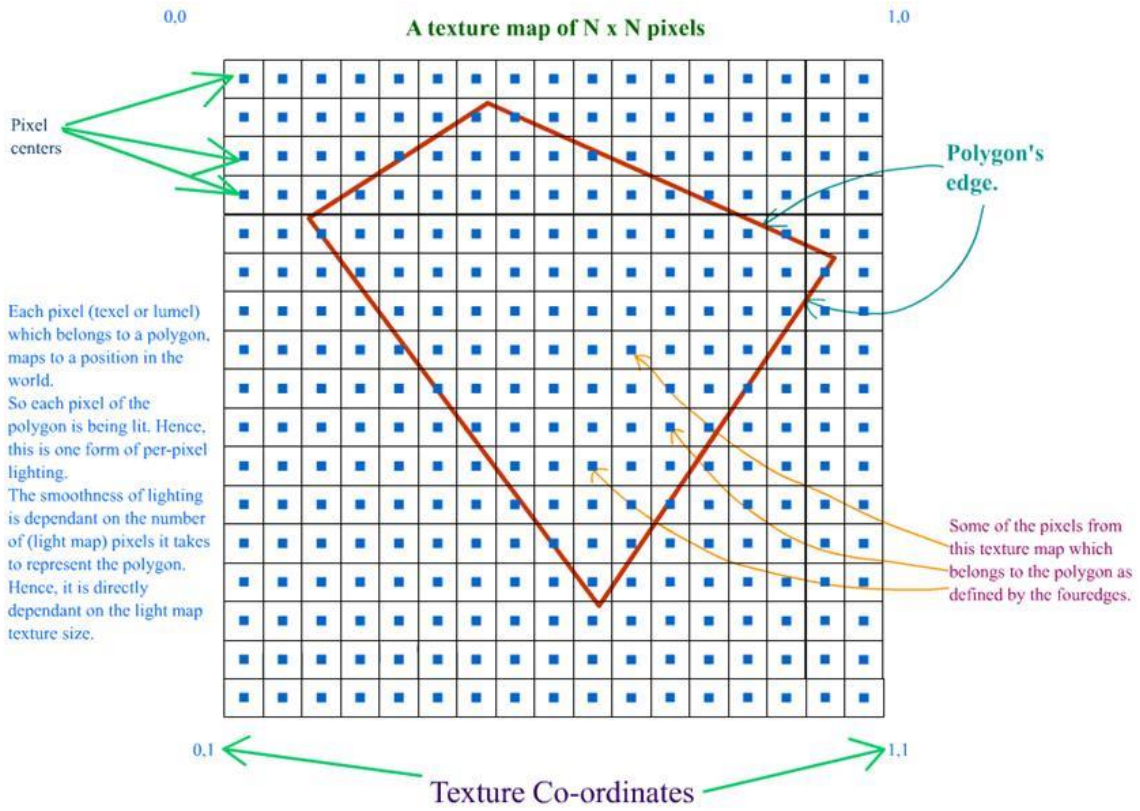
3- Set the S and T coordinates for the object.

Load the bitmap file using a new API called **auxDIBImageLoad** this API returns an image to be stored in a pointer that must be freed later when you done loading the image and store it in the OpenGl internal memory.




```
      AUX_RGBImageRec       *pBitmap = NULL;

      pBitmap = auxDIBImageLoad(pszFileName);
      if (pBitmap == NULL)
            // Error handling code should be here
```

To set the loaded data as texture for the current object, you need first to inform the OpenGL how this texture will be mapped over the surface of your object. Consider your texture as a unit square in dimension. This makes it easier to refer to pixels in images of different dimensions and aspect ratios. For each vertex you will draw, you need to inform the OpenGL about which pixel from the texture shall be mapped to this vertex and the rest of the geometry's pixels are interpolated through the pipeline.



This can be done by calling glTexCoord2f before setting your vertex position. This function takes two parameters S & T values, which correspond to X & Y respectively but in texture coordinates. The code sample below maps a texture exactly over the whole square.

```
void DrawSquare()
{
	glBegin(GL_QUADS);

	glTexCoord2f(0, 0);
	glVertex2f(-.5f, -.5f);

	glTexCoord2f(1, 0);
	glVertex2f(.5f, -.5f);

	glTexCoord2f(1, 1);
	glVertex2f(.5f, .5f);

	glTexCoord2f(0, 1);
	glVertex2f(-.5f, .5f);

	glEnd();
}
```

So far we just mentioned how the mapping will happen. But we need to provide the OpenGL with the texture object to map over this object. First we need to enable texture as it's disabled by default. Then we need to generate a name/id for the texture we will use. Then we need to bind this texture to a texturing target. Finally we set the data of the 2D texture we're using.

```
      glEnable(GL_TEXTURE_2D);
      glGenTextures(1, texture);
      glBindTexture(GL_TEXTURE_2D, texture[0]);
      gluBuild2DMipmaps(GL_TEXTURE_2D, 3, pBitmap->sizeX,
pBitmap->sizeY, GL_RGB, GL_UNSIGNED_BYTE, pBitmap->data);
```

The command `glGenTextures` generates a number of texture names/id, the first parameter is the number to generate and the second parameter is a UNIT [] to hold the generated names/ids.

The command `glBindTexture` binds a certain texture name to a texturing target. The first parameter is the type of texturing target and the second parameter is the texture name to bind. The possible texturing targets are `GL_TEXTURE_1D`, `GL_TEXTURE_2D`, `GL_TEXTURE_3D` and `GL_TEXTURE_CUBE_MAP`.

The command `gluBuild2DMipmaps` sets a 2 dimensional texture image; the parameters are listed in the table below.

Parameter	**Description**
target	Specifies the target texture. Must be `GL_TEXTURE_2D`, `GL_PROXY_TEXTURE_2D`, `GL_TEXTURE_CUBE_MAP_POSITIVE_X`, `GL_TEXTURE_CUBE_MAP_NEGATIVE_X`, `GL_TEXTURE_CUBE_MAP_POSITIVE_Y`, `GL_TEXTURE_CUBE_MAP_NEGATIVE_Y`, `GL_TEXTURE_CUBE_MAP_POSITIVE_Z`, `GL_TEXTURE_CUBE_MAP_NEGATIVE_Z`, or `GL_PROXY_TEXTURE_CUBE_MAP`.
Components	Specifies the level-of-detail number. Level 0 is the base image level. Level n is the nth mipmap reduction image.
Width	Specifies the width of the texture image including the border if any. If the GL version does not support non-power-of-two sizes, this value must be $2^n + 2$ for some integer n. All implementations support texture images that are at least 64 pixels wide.

height	Specifies the height of the texture image including the border if any. If the GL version does not support non-power-of-two sizes, this value must be $2^m + 2$ for some integer m. All implementations support texture images that are at least 64 pixels wide.
format	Specifies the format of the pixel data. The following symbolic values are accepted: `GL_COLOR_INDEX`, `GL_RED`, `GL_GREEN`, `GL_BLUE`, `GL_ALPHA`, `GL_RGB`, `GL_BGR`, `GL_RGBA`, `GL_BGRA`, `GL_LUMINANCE`, and `GL_LUMINANCE_ALPHA`.
Type	Specifies the data type of the pixel data.
data	Specifies a pointer to the image data in memory.

2. BE ABLE TO CONTROL TEXTURES BEHAVIOR

For now we loaded the texture and rendered it, but there are still other options that we can edit. Usually the dimensions of the texture are not exactly equal to those of the surface it will be mapped on. So, in this case the OpenGL will tend to do some Minimization or Magnification. These operations are performed using image processing filters. We can choose a fast filter which will give poor result or a good filter which will consume our rendering time and though will cost us frames per second.

Another thing is that, what if the geometry was given a T or S coordinate that exceeds 1. Will it repeat the texture again, mirror it or clamp it of just fill it with a solid color. The following piece of code allows you to modify such parameters.

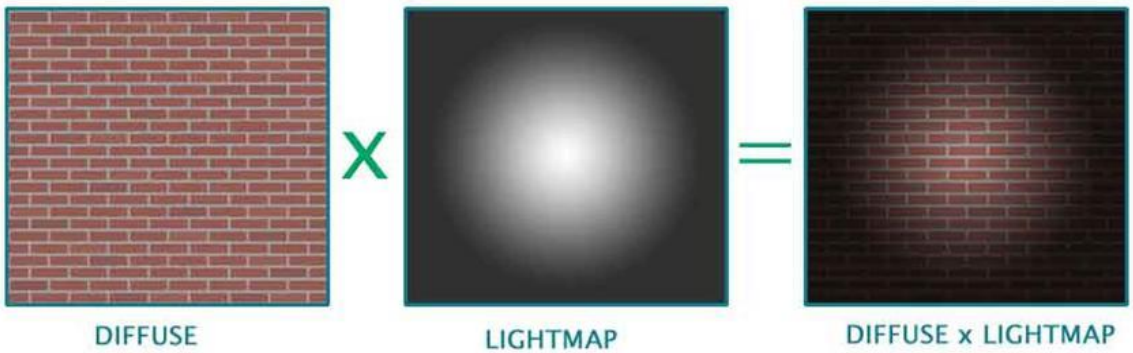
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
```

The command `glTexParameteri` is used to set the texture parameters. The table below contains a description for these parameters. For more information about the parameters and their possible values, check the function documentation in the above hyperlink.

Parameter	Description
target	Specifies the target texture. Must be `GL_TEXTURE_2D`, `GL_PROXY_TEXTURE_2D`, `GL_TEXTURE_CUBE_MAP_POSITIVE_X`, `GL_TEXTURE_CUBE_MAP_NEGATIVE_X`, `GL_TEXTURE_CUBE_MAP_POSITIVE_Y`, `GL_TEXTURE_CUBE_MAP_NEGATIVE_Y`, `GL_TEXTURE_CUBE_MAP_POSITIVE_Z`, `GL_TEXTURE_CUBE_MAP_NEGATIVE_Z`, or `GL_PROXY_TEXTURE_CUBE_MAP`.
pname	Specifies the symbolic name of a single-valued texture parameter. pname can be one of the following: `GL_TEXTURE_MIN_FILTER`, `GL_TEXTURE_MAG_FILTER`, `GL_TEXTURE_MIN_LOD`, `GL_TEXTURE_MAX_LOD`, `GL_TEXTURE_BASE_LEVEL`, `GL_TEXTURE_MAX_LEVEL`, `GL_TEXTURE_WRAP_S`, `GL_TEXTURE_WRAP_T`, `GL_TEXTURE_WRAP_R`, `GL_TEXTURE_PRIORITY`, `GL_TEXTURE_COMPARE_MODE`, `GL_TEXTURE_COMPARE_FUNC`, `GL_DEPTH_TEXTURE_MODE`, or `GL_GENERATE_MIPMAP`.
param	Specifies the number of color components in the texture.

3. ADD LIGHT MAPS USING MULTI TEXTURING.

New need to add another layer of texture, which represents our light map, though the final result will be our texture lightened with the chosen light map. Just as displayed in the figure below.



We need to define our 2 textures, the diffuse and the light map. For defining the texture we will do the same as we did in section one. Instead we will generate 2 texture names instead of one. For each texture we will set the parameters `GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MAG_FILTER, GL_TEXTURE_WRAP_S and GL_TEXTURE_WRAP_T`.

The case with multi texturing, is that it belongs to OpenGL extensions. This will add to us the headache of checking whether it's supported for the current version or not.

```
char*      pszExtenstions = (char*)glGetString(GL_EXTENSIONS);
string           str = pszExtenstions;
bMultiTexturingSupported = str.find("GL_ARB_multitexture") >= 0 ? true : false;
```

The command `glGetString Gets all supported extensions by this version when passing to it the flag GL_EXTENSIONS`. Then we check for the string if it contains ARB multi-texture or not. There are many types of extensions but the ARB extension has been officially approved by the OpenGL Architectural Review Board.

Now we need to define the texture coordinates for our new 2 textures, using ARB extensions.

```
void DrawSquare()
{
      glBegin(GL_QUADS);
      glTexCoord2f(0, 0);
      glMultiTexCoord2fARB(GL_TEXTURE0_ARB, 0, 0);
      glMultiTexCoord2fARB(GL_TEXTURE1_ARB, 0, 0);
      glVertex3f(-.5f, -.5f, 0);
```

Before we render our object which will be affected by this textures we need first to enable the first 2 ARB textures, then bind our textures to each one respectively. Then don't forget to disable every thing after you're done, as this won't affect any other objects.

```
      glActiveTextureARB(GL_TEXTURE0_ARB);
      glEnable(GL_TEXTURE_2D);
      glBindTexture(GL_TEXTURE_2D, texture[0]);

      glActiveTextureARB(GL_TEXTURE1_ARB);
      glEnable(GL_TEXTURE_2D);
      glBindTexture(GL_TEXTURE_2D, texture[1]);
      DrawSquare();
      glDisable(GL_TEXTURE_2D);
      glActiveTextureARB(GL_TEXTURE0_ARB);
      glDisable(GL_TEXTURE_2D);
```

The command `glActiveTextureARB` selects the active texture unit. So, we first select our first ARB texture then enable texturing 2D for it and bind our texture object to this rendering target. Then do the same with the second ARB texture.

Hint: You can use the key "L" to switch between single texture and multi-texture scenes.