Computer Graphics Labs

Abel J. P. Gomes

LAB. 7

Department of Computer Science and Engineering
University of Beira Interior
Portugal
2020

LAB. 7

TEXTURES

1. Learning goals
2. Handling Textures using OpenGL
3. Interesting Links
4. Example: The Texturized Cube
5. Programming Exercises

Lab. 7

TEXTURES

This laboratory unit covers the basics of texture mapping in OpenGL. Usually, this involves 3 major steps:
- The reading of texture file (e.g., a JPG file) in memory,
- the uploading of the texture to the video memory, and
- the application of the texture onto geometry.

The first step is not strictly necessary because we can create the texture directly in memory. Below, we use some snippets of code to clarify ideas and concepts.

1. Learning Goals

At the end of this lab **you should be able to**:

1. To map textures onto different geometric objects.
2. Eventually, to master more advanced texture techniques such as, for example, bump and lightmap texturing.

2. Handling Textures using OpenGL

Let us assume that we have some image data (i.e., texture) in memory, and we want to apply it to a given geometric object using OpenGL. Before using such texture data in our OpenGL application, we need to upload it to the video memory. Then, such texture can be used every time we need while the OpenGL application is running.

2.1 Setup

However, before uploading a texture to the video memory, some setup must take place to make sure that OpenGL engine knows what to do with it. Such a setup involves the following procedures:

glGenTextures and **glBindTexture:**

In the process of uploading the texture, the first step is to call the function `glGenTextures`. This function tells OpenGL which texture "id" we are going to work with. This texture "id" is just an integer that we use to access a specific texture. For example, the write down

```
unsigned int texID;
```

and call

```
glGenTextures(1, &texID);
```

Indeed, this function generates an array of identifiers for textures. In this case, the array has only a single identifier, because supposedly we intend to use only a single texture.

Just like other objects in OpenGL, before applying operations to textures, they must be bound, i.e., we need to specify the dimension of the target texture as an array of pixels. If an image is a 2D array of pixels, it must be bound to the `GL_TEXTURE_2D` target as follows:

```
glBindTexture(GL_TEXTURE_2D, texID);
```

glTexParameteri:

The function `glTexParameteri` allows us to set various parameters for the currently bound texture. This setup involves the parameters concerning *wrapping* modes and *filtering* modes.

For example, the following code snippet sets the wrapping mode of the currently bound texture as GL_REPEAT along the directions *s* and *t*. Concerning the filtering modes, both magnification and minification filters are set as GL_LINEAR.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

2.2 Load and generate texture

`glTexImage2D`:

This function uploads the texture to the graphics memory. Its arguments are as follows:

- *target* - The target or texture type; it will be GL_TEXTURE_2D, but also 1D and 3D textures are possible targets.
- *level* - The level of detail (LoD) number; for the time being, level should take on the value 0. However, this value may change in the future in more advanced graphics

applications.
- *internal format* - Internal components parameter. This argument indicates how many color components we will use in the graphics application internally from the texture. Typically, we will use GL_RGB, whose value is equal to 3.
- *width & height* - The width and height of the image data. These arguments must be integers, each one of which specified as a power of 2.
- *border* - Image border, must be 0 or 1. Usually, we use 0 when we do not use image border, and 1 otherwise.
- *format* - Format of the texture pixel data that will be uploaded. There are many possibilities, but GL_RGB is the constant value that is most widely used.
- *type* – Data type that will be uploaded. Again, there are several possibilities for this argument, but typically we will use GL_UNSIGNED_BYTE.
- *pixels* - Pointer to the image data. This indicates the image data that will be uploaded to the graphics memory.

Now that you know the parameters for `glTexImage2D` here is a sample call:

```
glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,imageWidth,imageHeight,0,GL_RGB,
GL_UNSIGNED_BYTE,imageData);
```

Note that after calling `glTexImage2D`, we can free the texture from RAM memory this function has just transferred the texture image into video memor y. For example, in the following code snippet, we use the function `stbi_texture` to read the texture in the file container.jpg in RAM memory, which is then uploaded into graphics memory using the function `glTexImage2D`. In the end, we use the function `stbi_image_free` to free RAM memory occupied by the texture. The function `glGenerateMipmap` creates a mipmap of the texture in graphics memory.

```
// load and generate the texture
int width, height, nrChannels;
unsigned char *data;

data = stbi_load("container.jpg", &width, &height, &nrChannels, 0);

if (data) {
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
      GL_RGB, GL_UNSIGNED_BYTE, data);
   glGenerateMipmap(GL_TEXTURE_2D);
}
else
      std::cout << "Failed to load texture" << std::endl;

stbi_image_free(data);
```

Concerning the functions `stbi*`, have a look at the webpage:

https://learnopengl.com/Getting-started/Textures

3. Interesting Links

Before proceeding any further, have a look at the following webpages:

https://learnopengl.com/Getting-started/Textures
http://www.opengl-tutorial.org/beginners-tutorials/tutorial-5-a-textured-cube/
https://open.gl/textures
https://learnopengl.com/Advanced-OpenGL/Cubemaps
https://learnopengl.com/In-Practice/Text-Rendering
http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-11-2d-text/

4. Example: The Texturized Cube

This example is available at:

http://www.di.ubi.pt/~agomes/cg/praticas/cap4-cookbook.zip

This code includes more 8 graphics applications to show textures can be used to model and represent several phenomena, including reflection and refraction effects. To see a cube textured with bricks, we need to keep the code line `string recipe = "texture";` in the file main.cpp as it stands. But, by changing the recipe to `"refract-cube"`, we can observe how texturing can used to simulate refraction effects.

Remarks:

The cap4-cookbook.zip code was adapted from the one available at:

https://github.com/daw42/glslcookbook

so, the student is referred to carefully read this webpage to better understand how lighting interacts with objects within a scene. This code was published together with the chapter 4 of the book entitled "*OpenGL 4 Shading Language Cookbook*", authored by David Wolff.

Also, if the GLAD files that are wrapped in lighting.zip do not work, and as explained in the web page above, you need to use GLAD web service (https://glad.dav1d.de/) to generate such GLAD files to interface OpenGL with your graphics card driver. These files need to be added to any graphics application onwards. But, before generating the GLAD files, you first need to check which OpenGL version is supported by your computer, using GLview (http://www.realtech-vr.com/home/glview) or similar package.

5. Programming Exercises

First read the web page about textures at:

https://learnopengl.com/Getting-started/Textures

which allows to wrap a texture onto a rectangle. Based on this example, let us then proceed as follows.

1) Draw a white cube with one of its faces mapped with the texture container.jpg.
2) Draw a white cube with all its faces mapped with the texture container.jpg. This texture is available from the webpage above.
3) Change the color of the cube vertices so that each vertex has a different color. Now, the cube must be drawn by blending cube colors with texture colors.
4) Change the texturing of the cube so that now 3 out of 6 faces are textured with awesomeface.png. This texture is also available from the webpage above.
5) Add a point light to the scene using Phong shading, assuming the cube is made of gold material. Loot at http://devernay.free.fr/cours/opengl/materials.html to know the RGB values for gold material. Now, the cube must be drawn by blending cube colors with texture colors. What is the difference in relation to 4)?
6) Add a spot light to the scene. What happens?