CS130 - LAB - Texture Mapping

Name: Patrick Dang SID: 862091714

Introduction

Texture mapping in GLSL consists of 3 parts

1. Uploading a texture: Handled in the OpenGL program (C/C++) part. In a typical OpenGL program, textures are read from an image file (.png,. tga etc.) and loaded in to OpenGL. The parameters of the texture (such as interpolation methods) are also set in the program.
2. Computing the texture coordinate of a vertex: In GLSL, the texture coordinate `glTexCoord[i]` for a texture i and a vertex is determined in the vertex shader. This is the coordinate of the vertices' corresponding texture positions in the image data of texture i, where i is the index of a texture (in case of multiple textures, $i = 0$ for a single texture).
3. Getting the texture color for a fragment: The texture coordinate, `glTexCoord[i]`, of texture i is readily interpolated to the fragment location by OpenGL. A lookup function such as texture2D is used to get the color from the texture.

Part I: Uploading a Texture

Read the tutorials about uploading a texture file in these links and answer the questions.

1. https://www.gamedev.net/articles/programming/graphics/opengl-texture-mapping-an-introduction-r947
2. http://www.opengl-tutorial.org/beginners-tutorials/tutorial-5-a-textured-cube/#how-to-load-texture-with-glfw (Until section "How to load texture with GLFW")

Question 1: Describe briefly with your own words each one of the following functions. Look at the OpenGL documentation for reference. Link: https://www.khronos.org/registry/OpenGL-Refpages/gl4/

Google: "OpenGL 4 references"

`glGenTextures`:

Input 1: The number of texture names generated
Input 2: The array the texture names are stored

`glBindTexture`:

Input 1: The target the texture is bound to
Input 2: The name of the texture

`glTexParameter`:

Input 1: The target the texture is bound to
Input 2: The symbolic name of the texture parameter
Input 3: The value(s) of input 2

`glTexImage2D`:

Input 1: The target texture
Input 2: The number specifying the amount of detail going into the texture
Input 3: The number of colors in the texture
Input 4: The width of the texture image
Input 5: The height/number of layers in the texture image/array
Input 6: A value of 0
Input 7: The format of pixel data
Input 8: The data type of the pixel data
Input 9: The pointer to the image data in memory

Question 2: Answer the question below, briefly. Hint: see `glTexParameter`'s reference page.

1. What do minifying and magnifying mean?

   Minifying means causing the object to move farther away from the viewer (less details needed). Magnifying means the object is getting closer to the viewer (more details needed).

2. What parameter name should be used in `glTexParameter` function in order to specify minifying function?

GL_TEXTURE_MIN_FILTER should be used to specify the minifying function.

3. What parameter name should be used in `glTexParameter` function in order to specify magnifying function?

   GL_TEXTURE_MAG_FILTER should be used to specify the magnifying function.

4. What are the possible minifying and magnifying functions defined by OpenGL?

   Answer: GL_LINEAR, GL_NEAREST, GL_NEAREST_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_LINEAR

Question 3: Read the comments and fill out the code accordingly.

```
// ========================================================================
// Inputs:
//    data: a variable that stores the image data in ``unsigned char*''
// GL_UNSIGNED_BYTE type
//    height: an integer storing the height of the image data
//    width: an integer storing the height of the image data
// Description:
//    A piece of code that uploads the image ``data'' to OpenGL
//
// ========================================================================
GLuint texture_id = 0;
// generate an OpenGL texture and store in texture_id variable
glGenTextures(1, &texture_id);

// set/``bind'' the active texture to texture_id
glBindTexture(GL_TEXTURE_2D, texture_id);

// Set the magnifying filter parameter of the active texture to linear
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

// Set the minifying filter parameter of the active texture to linear
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// Set the wrap parameter of "S" coordinate to GL_REPEAT
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);

// Set the wrap parameter of "T" coordinate to GL_REPEAT
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

// Upload the texture data, stored in variable ``data'' in RGBA format
glTexImage2d(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0,
             GL_RGBA, GL_UNSIGNED_BYTE, data);
```

Part II: Shading with Textures in GLSL

Read the tutorials below and answer the following questions

https://www.opengl.org/sdk/docs/tutorials/ClockworkCoders/texturing.php

Introduction section only

http://www.lighthouse3d.com/tutorials/glsl-12-tutorial/simple-texture/

1. Fill out the blanks in the vertex and fragment shaders below to compute the `gl_TexCoord[0]` using `gl_TextureMatrix[0]` and `glMultiTexCoord`.

`vertex.glsl`:

```
varying vec3 N;
varying vec4 position;

// Create a uniform 2D texture sampler variable, with name "tex";
uniform sampler2D tex;


void main()
{
    // compute gl_TexCoord[0] using gl_TextureMatrix[0] and glMultiTexCoord
    gl_TexCoord[0] = gl_TextureMatrix[0] * gl_MultiTexCoord;
    N = gl_NormalMatrix * gl_Normal;
    gl_Position = gl_ProjectionMatrix * gl_ModelViewMatrix * gl_Vertex;
    position = gl_ModelViewMatrix * gl_Vertex;
}
```

`fragment.glsl`:

```
// create a uniform 2D texture sampler variable, with name "tex", so that
// it will be forwarded from the vertex shader
uniform sampler2D tex;



void main()
{
    // get the texture color from "tex", using a texture lookup function
    // and the s,t coordinates of gl_TexCoord[0].


    vec4 tex_color = texture2D(tex,gl_TexCoord[0].st);
```

```
    // set gl_FragColor to tex_color
    gl_FragColor = tex_color;
}
```

Part III: Texture mapping coding practice

In this part of the lab you will be practicing texture mapping with GLSL with a skeleton code.

The skeleton code has texture files (`.tga` images) `monkey.tga` and `monkey_occlusion.tga`, as well as the base code for creating an OpenGL window, loading a monkey model and drawing it. Follow the steps below and implement texture mapping in the skeleton.

Step 0. Download the skeleton code from the lab webpage to your environment of choice (the ti-05 server works best for this) and unzip/untar it. Open the image files and observe their content.

Step 1. Uploading `monkey.tga` to OpenGL:

- Locate the TODO section towards the end of the `loadTarga` function in `application.cpp`
- Use the code in the answer to Part I Question 3, to upload "data" to OpenGL.

Note 1: the code at the beginning of the function reads the image file in "filename" to the "data" variable.

Note 2: `monkey.tga` is in RGBA format, and the data is a pointer to `UNSIGNED_BYTE` array.

Step 2. Computing texture coordinates

- Locate `vertex.glsl` and compare the code with the vertex shader code in Part II Question 2.
- Note that the code that computes the texture coordinate of the vertex is already computed, and so you have nothing to do and may go to step 3.

Step 3. Computing the color of the fragment using texture color.

- Locate fragment.glsl and compare the code with the phong shader you implemented in a previous lab. This is a phong shader without a specular component.

- Now compute the texture color just like in the fragment shader in Part II Question 3 and store it in a `tex_color` variable. But, do not assign it to `gl_FragColor`.
- Here we would like to use the texture color instead of the material color (`glFrontMaterial.diffuse.rgb`), while keeping the rest of the computation.
- Rewrite the line that computes the `gl_FragColor` to do this.