COSC363 Computer Graphics

Lab10: OpenGL-4 Texture Mapping

Aim:

This lab introduces the procedure for mapping textures to polygonal surfaces using the OpenGL-4 pipeline. It also demonstrates the use of the fragment shader in modifying a fragment's properties based on the colour of the applied texture.

I. CubeDraw.cpp:

1. The program `CubeDraw.cpp` provides the code for displaying the mesh model of a cube. It has a structure similar to `TorusDraw.cpp` (Lab09).

   - The file `Cube.h` contains the array definitions specifying vertex coordinates, normal components, texture coordinates and polygon indices.

   - The vertex shader (`Cube.vert`) implements a simple lighting model using only the diffuse component. It also outputs the texture coordinates and the diffuse lighting term (***n•l***) to the fragment shader.

   - The fragment shader (`Cube.frag`) defines a uniform variable "tSampler1" of type `Sampler2D` (a texture type). The function `texture()` returns a colour value obtained by sampling the texture using the input texture coordianates. The fragment shader outputs this colour for each fragment, thus generating the display of a texture mapped primitive.

   - The program `CubeDraw.cpp` loads the texture "Brick.tga" (Fig. 1(a)) and assigns it to texture unit 0. Using the function `glUniformi()` it assigns the same value 0 to the variable "tSampler1" in the fragment shader. The program generates the output of a texture mapped cube (Fig. 1(c)).

Fig. 1(a)

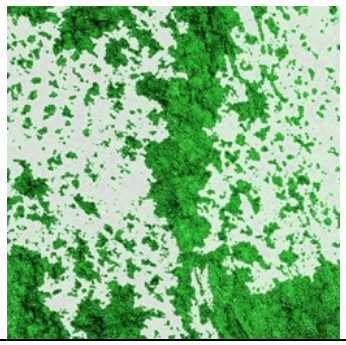

Fig. 1(b)

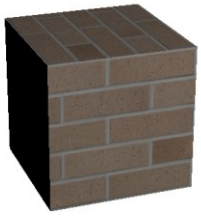

Fig. 1(c)

2. Modify the program `CubeDraw.cpp` to load a second texture "Moss.tga" (Fig. 1(b)). Select texture unit 1 for this texture, and assign the value 1 to a uniform variable "tSampler2" . Add this new uniform variable "tSampler2" in the fragment shader. Modify the shader's output by combining the colour values obtained from the two textures. The result of multi-texturing should look similar to that given in Fig. 2.

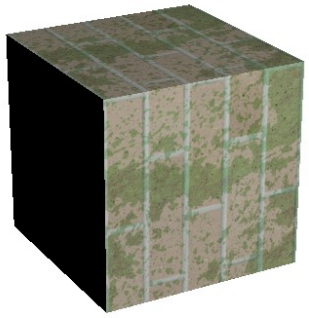

Fig. 2.

II. CylinderDraw.cpp:

1. The program `CylinderDraw.cpp` generates the output shown in Fig. 3(a).

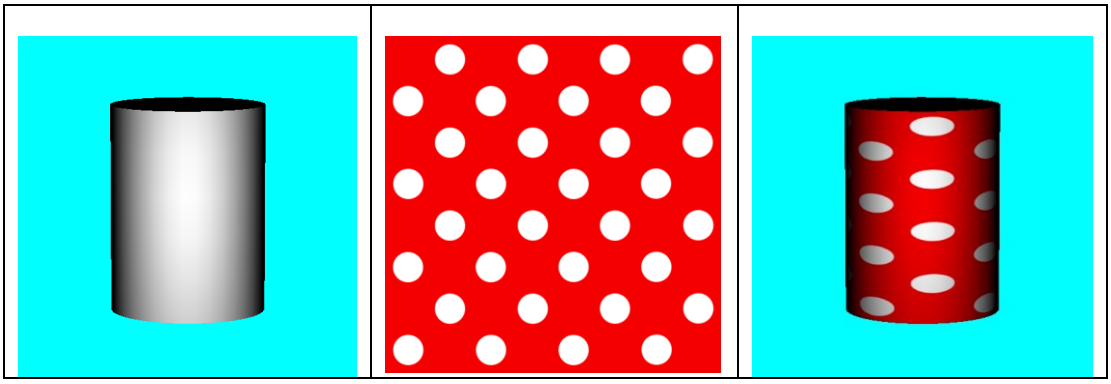

Fig. 3(a) Fig. 3(b) Fig. 3(c)

- The file `Cylinder.cpp` generates the buffer objects for a generic cylinder of radius *r*, height *h*, and the user specified number of slices and stacks.

- The vertex shader `Cylinder.vert` is just another copy of `Cube.vert`. It outputs the texture coordinates and the diffuse lighting term (***n•l***) to the fragment shader.

- The fragment shader `Cylinder.frag` simply outputs color (1,1,1,1) scaled by the diffuse term received from the vertex shader.

2. Modify the program `CylinderDraw.cpp` to load the texture "Dots.tga" (Fig. 3(b)). Also modify the fragment shader to include the necessary Sampler2D variable to access the texture, and to output the texture colour modulated by the diffuse term. The output should look similar to that given in Fig. 3(c).

3. After properly mapping the texture to the cylinder, we can remove parts of the cylinder where the colour value is white, thus carving holes in the cylinder! This interesting visual effect can be easily implemented in the fragment shader by discarding fragments that have white as the texture colour. Since the texture has only two colours red and white, we need only check the green component value to detect white pixels:

```
if(texColor.g > 0.9) discard;
```

The output is shown in Fig. 4 below.

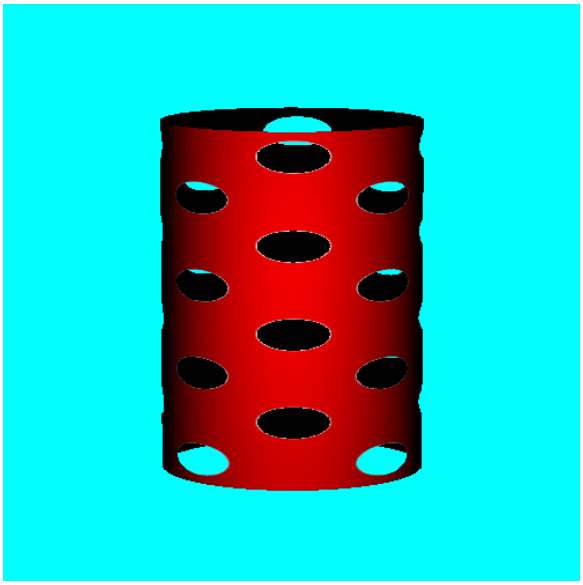

Fig. 4.

III. Quiz-10

This is the last quiz for this course!

The quiz will remain open until **5pm, 31-May-2019**.