Programming Assignment #6 - Textures

CS 412 – Computer Graphics – Spring 2018

Due: 4/20/2018

Goals for this assignment

- Learn about accessing textures from GLSL shaders.
- Implement a program to demonstrate textures.

Academic Honesty

Please read carefully the section titled "Academic Integrity" in the syllabus. If you have any questions, contact your instructor before you begin this assignment.

Overview

This assignment involves adding textures to objects in our scene. We will support objects that have textures applied and those that do not. To do so, we'll update our program to load texture files from disk and access them from a global object. The `Material` class will be updated to include textures and you'll update the GLSL shaders to access them. If an object has a texture, our shader code will use it as the diffuse reflectivity, otherwise it will use the uniform variable as in the previous assignment.

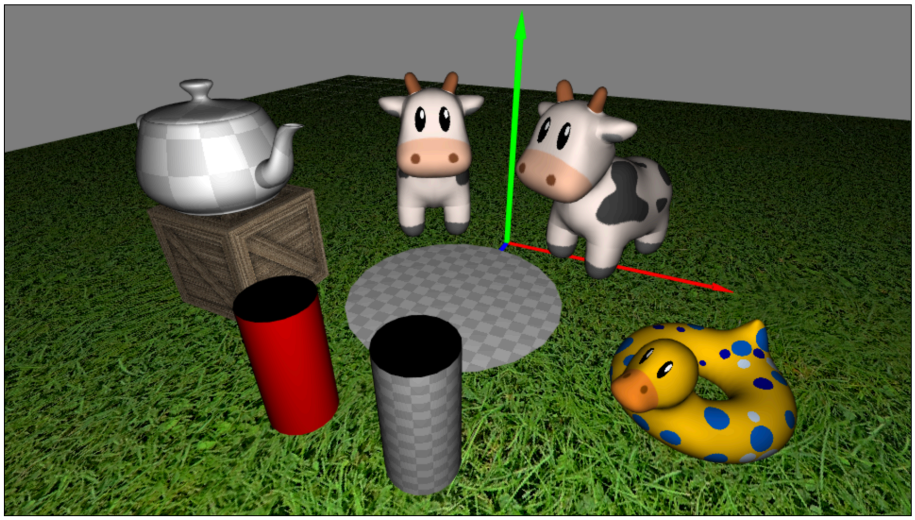

Figure 1: *Example of a final result. One cylinder has a texture applied, the other does not.*

Procedure

With the exception of a few updated files, there is no starter code for this assignment. You'll dig into the current code and make modifications. To do this, you'll need a strong understanding of the code. If you don't feel comfortable with all of the code yet, you should spend some time reading it and asking questions.

1. Download a new version of `objmodel.js`. There are very few changes this time. The only difference is that it supports the `map_Kd` and `map_Ks` commands in material files. If there is a `map_Kd` command in a material file, it will store the texture file name in the `Material` object's property

named `diffuseTexture`. For `map_Ks` it will store the file name in `specularTexture`. Note that this does not actually load the images. You'll need to do that yourself.

2. Download the `material.js` and `textures.js` files. You can replace the `setUniforms` method in `material.js` with the one you wrote in the previous assignment, but we'll be adding to it.
3. Update the GLSL code for the Phong shader from the previous assignment.
   (a) In the vertex shader, add a new `in` variable. Remember, `in` variables are vertex attributes and take on values from the vertex arrays. This one will take the texture coordinate, so it needs to be a `vec2`. Our `TriangleMesh` class sends texture coordinates via attribute location 2. Therefore, you'll use that location. So the declaration should be:

```
layout(location=2) in vec2 vTexCoord;
```

   (b) Also in the vertex shader, we need to add a new `out` variable. We'll use this to send the texture coordinate to the fragment shader.

```
out vec2 fTexCoord;
```

   (c) Update the `main` method in the vertex shader to assign the value of the `vTexCoord` to `fTexCoord`. No modification/transformation of the value is necessary here, although we could do so if desired. Remember that this value will be interpolated across the triangle before reaching the fragment shader.
   (d) In the fragment shader add a corresponding `in` variable to receive the texture coordinate:

```
in vec2 fTexCoord;
```

   note that the name must be the same in both stages.
   (e) We need some additional uniform variables to manage textures. In this assignment we'll use two. One indicates whether or not we're using a texture, and the other is the variable used to access the texture (called a `sampler2D`). We can add these to either (or both) of the shader stages, but since we'll be using them in the fragment shader, we'll add them there:

```
// Texture parameters
uniform bool uHasDiffuseTex;   // Whether or not we're using a texture
uniform sampler2D uDiffuseTex; // The diffuse texture
```

   (f) Update the code of the `main` method to compute the diffuse shading based on the value of `uHasDiffuseTex`. If the value is `false`, then the shader will use `uDiffuse` as the diffuse reflectivity. Otherwise, it will access the texture via `uDiffuseTex` and use the RGB value as the diffuse reflectivity. We'll cover details in class. You should only need a couple lines of code to make this change.
4. Update your `init` method to get the locations of all new uniform variables.
5. Review/recall the difference between a texture "unit" and a texture object. Texture units are selected using `gl.activeTexture()`. A texture unit can switch between multiple texture objects during rendering. However, if you want a shader to access more than one texture at the same time, you need at least two different texture units. The shader variable `uDiffuseTex` (or any

`sampler` variable) is set to the texture unit number. In our case, we'll use texture unit 0 for all of our textures. Use `gl.uniform1i` to set the uniform variable `uDiffuseTex` to 0. For this assignment, we won't need to change this, so you can do this only once in the `init` function.

6. Load all of the texture image files within your `init` function. I've provided you with a utility function for loading textures in `412-utils.js`. The function is: `Utils.loadTexture`. Read through the documentation. This is another one of those asynchronous functions that returns a `Promise` object, so we'll treat it the same way we did the `Obj.load` method. You might want to review that section of the previous assignment.

   The `Utils.loadTexture` function will return a `Promise` that resolves to a `WebGLTextureObject` (if successful). Store each loaded texture in the `Textures` object. We'll use this object as a simple map. Store each texture object as a property in the object with the property name being the same as the texture file name. Recall that we can use any string as a property name. To do so, we'll need to use the square bracket syntax for accessing/setting properties. For example:

```
Utils.loadTexture(gl, "media/texture_file_name.png")
  .then(function(val) {
    Textures["texture_file_name.png"] = val;
    ...
  });
```

7. Update `setUniforms` in `material.js`. If the `diffuseTexture` property is non null and exists as a property in the `Textures` object, then set the `uHasDiffuseTex` uniform to `true` (use `gl.uniform1i` with a value of 1), and bind the corresponding texture object to texture unit 0 (use `gl.activeTexture` and `gl.bindTexture`). Otherwise, set `uHasDiffuseTex` to `0` (`false`).

8. Add texture coordinates to your objects. You can skip the cone for this assignment. When adding coordinates to the cylinder, watch out for the "seam". You may see incorrect results along one polygon where it connects back to itself. This is because you need different texture coordinates there for each side of the last pair of points. You'll need to double that pair of verts along the seam.

   When adding texture coordinates to the disk, make sure that you don't warp the texture. Don't try to map the entire image to the disk. Instead, just map as much of the image as possible without warping. Use the checkerboard texture `default.png` to help debug.

9. Update any `Material` objects that you're using, adding the `diffuseTexture` property.

10. Finally, you can test! You'll probably run into lots of errors. Be patient and methodical. Ask questions.

11. Build a scene that includes several objects with textures and others without. See the screenshot above for an example.

Bonus

For a few bonus points, implement a specular texture map (also called a gloss map) and/or an occlusion map. A specular map is one that defines the specular reflectivity (instead of the diffuse reflectivity). An occlusion map is one that indicates how the surface of the object is exposed to ambient light. I've provided you with some textures that have both specular, occlusion and diffuse maps. You'll need to be able to access two (or more) different textures within the shader at the same time. In order to do this, you'll need multiple texture units available to the shader (therefore multiple `sampler2D` variables).

Submission Instructions

See the “Programming Assignment Requirements” document for details on submitting your assignment, and other important requirements.