Textures and Material Properties, or Fun with Fragment Shaders

CS 320

You may work in teams of cardinality two or three. For grading purposes, this assignment will count as half a project (graded out of 50 points.) Do all three of the following exercises. All of them use the TextureStarter Eclipse archive file as the basis. Add each exercise to the same TextureStarter instance. Note that pressing the 'f' key will cycle between the existing two shaders. This behavior should continue as you complete each exercise. The basic-gl2 vertex shader is used for all exercises. See the solution demo video on the course web site for the effect that you're working to achieve in each exercise.

1. Starting from a copy of the solid-gl2 fragment shader, implement the 8×8 checkerboard texture procedurally — based upon the x- and y-values of the texture coordinate passed to the shader in `vTexCoord`, shade the fragment `uColor` or black.
2. Starting from a copy of the solid-gl2 fragment shader, implement the 8×8 checkerboard texture using the included checkerboard.ppm image. This will require adding image texture-handling code to `asst2.cpp` from your HW2d project or the HW2d starter code.
3. Starting from a copy of the diffuse-gl2 fragment shader, add material specularity (shininess) and an ambient light component. Your fragment shader will implement diffuse, specular, and ambient material properties using both light sources passed to the shader. Your specularity calculation should use the bounce vector, **not** the halfway vector.

Use GitHub to turn in your work. Use the link in GoucherLearn to create your Texture team and repository. If your team hasn't changed, you may be able to recycle your team name. Your repository must include a `README.md` file containing, at a minimum, the names of the team members. There will be an 8% penalty if this file and information are missing.

Forgotten the git commands? See the *Git/GitHub Crash Course* document on the course web site.

GLSL shader language hints

1. The example shader programs in the textbook don't use quite the same GLSL dialect that we're using. See the appendix below for examples.
2. An operator's operands must be of the same type:

```
float x = 5.0;
int i = 5;
x = x + i;          // Won't compile.
i = x;              // Won't compile.
i = (int) x;        // Won't compile.
i = int(x);         // Use constructor in place of a cast.
x = x + float(i);   // Ditto.
```

3. The following bit-wise operators are available:
   (a) Unary negate: `~`
   (b) Binary and: `&`
   (c) Binary or: `|`
   (d) Binary xor: `^`

   The arithmetic modulo operator `%` is also available.
4. For GLSL details, see the GLSL language specification for version 1.10.59 and the `gpu_shader4` extension documentation on the course web site.

Appendix

Vertex shader written in our GLSL dialect:

```
uniform mat4 uProjMatrix;
uniform mat4 uModelViewMatrix;
uniform mat4 uNormalMatrix;

attribute vec3 aPosition;
attribute vec3 aNormal;

varying vec3 vNormal;
varying vec3 vPosition;

void main() {
  vNormal = vec3(uNormalMatrix * vec4(aNormal, 0.0));

  // send position (eye coordinates) to fragment shader
  vec4 tPosition = uModelViewMatrix * vec4(aPosition, 1.0);
  vPosition = vec3(tPosition);
  gl_Position = uProjMatrix * tPosition;
}
```

Same vertex shader written in the textbook's dialect:

```
#version 130

uniform mat4 uProjMatrix;
uniform mat4 uModelViewMatrix;
uniform mat4 uNormalMatrix;

in vec3 aPosition;
in vec3 aNormal;

out vec3 vNormal;
out vec3 vPosition;
```

```
void main() {
  vNormal = vec3(uNormalMatrix * vec4(aNormal, 0.0));

  // send position (eye coordinates) to fragment shader
  vec4 tPosition = uModelViewMatrix * vec4(aPosition, 1.0);
  vPosition = vec3(tPosition);
  gl_Position = uProjMatrix * tPosition;
}
```

Fragment shader written in our GLSL dialect:

```
uniform vec3 uLight, uLight2, uColor;

varying vec3 vNormal;
varying vec3 vPosition;

void main() {
  vec3 tolight = normalize(uLight - vPosition);
  vec3 tolight2 = normalize(uLight2 - vPosition);
  vec3 normal = normalize(vNormal);

  float diffuse = max(0.0, dot(normal, tolight));
  diffuse += max(0.0, dot(normal, tolight2));
  vec3 intensity = uColor * diffuse;

  gl_FragColor = vec4(intensity, 1.0);
}
```

Same fragment shader written in the textbook's dialect:

```
#version 130

uniform vec3 uLight, uLight2, uColor;

in vec3 vNormal;
in vec3 vPosition;

out vec4 fragColor;

void main() {
  vec3 tolight = normalize(uLight - vPosition);
  vec3 tolight2 = normalize(uLight2 - vPosition);
  vec3 normal = normalize(vNormal);

  float diffuse = max(0.0, dot(normal, tolight));
  diffuse += max(0.0, dot(normal, tolight2));
  vec3 intensity = uColor * diffuse;

  fragColor = vec4(intensity, 1.0);
}
```