11494 Interactive and GPU Computing Lab. 4 - 16/04/2015

GLSL - Illumination

Supervisor: Abel Gomes *Scribe: Orlando Pereira*

The goal of this assignment is to extend the Hello GLSL demo provided in the last class.

1 Exercises: Host (CPU) - Device (GPU) communication

1. Change the color of a triangle using an uniform variable (called uColor). Note that you should pay special attention to the data type of uColor.
2. Change the triangle intensity color through another uniform variable called uIntensity.
3. Change the color of a triangle using an attribute variable (called aColor).
4. Render one object (triangle, quad, or other) using only attribute vertex. Each object vertex should be defined using an attribute variable.
5. Manually interpolate your triangle color using an uniform variable called vColor.
6. Explain the main differences between uniform, attribute and varying variables. When you should use one instead of another?

2 Exercises: Variables, Statements, and Functions

1. Inside your vertex shader create the following structure.

```
struct sVertex {
    vec4 color;
    vec3 position;
    float intensity;
};
```

2. Create the required shader variables to populate the structure.
3. Use the structure values to shade your object.
4. Modify your program to receive user input interaction. The user should be able to change the triangle facet colors or intensity by pressing a keyboard key.

   The accepted inputs are: R, G, B, 1, 2, 3. When you press r, g, or b the triangle color should change to red, green or blue, accordingly. When you press 1, 2, or 3, you should change the color intensity. Note that, both input properties should cooperate in the shader program. Each interaction should be taken care by the shader program using a specific function (for instance, a function to change the triangle color to Red). The shader variables to use are a personal choice. you can use the following color variables:

```
vec4 Red = vec4(1,0,0,0);
vec4 Green = vec4(0,1,0,0);
vec4 Blue = vec4(0,0,1,0);
```

3 Exercises: Illumination

In this section you should modify your vertex shader program to recreate the Phong reflection model [1].

1. For simplicity, you can create a new shader program (vertex and fragment) or comment the unnecessary code from your current shader.

2. Replace your object by a solid teapot. Note that for correct rendering visualization you should enable the OpenGL back-face culling mechanism.

   For the material properties you should use the following:

```
float lpos[4] = {3,3,1,0}; // OpenGL variable
// Shader variables
vec4 materialKA = vec4(0.0215, 0.1745, 0.0215, 0);
vec4 materialKD = vec4(1, 0.31424, 0.07568, 0);
vec4 materialKS = vec4(0.633, 0.727811, 0.633, 0);
float materialSH = 0.6;
```

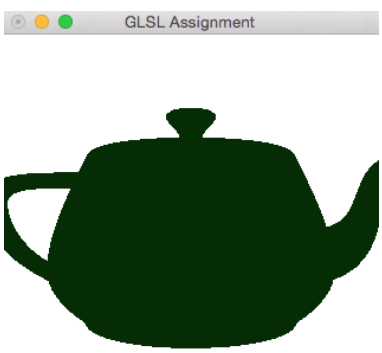

Figure 1: Ambient

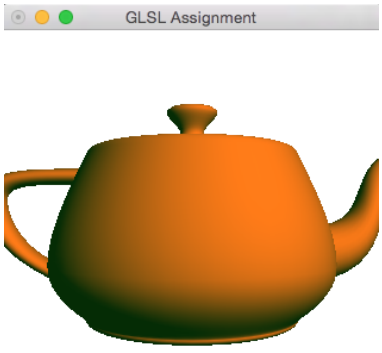

Figure 2: Diffuse

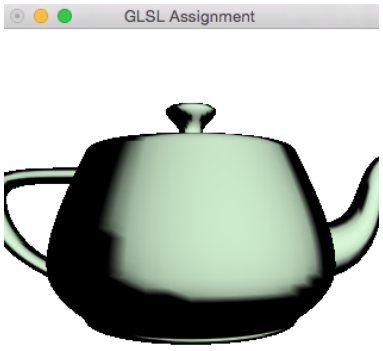

Figure 3: Specular

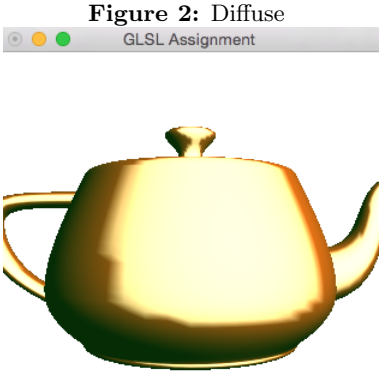

Figure 4: Final

Figure 5: Phong reflection model

3. Render the teapot using only the ambient color component.

4. Render the teapot using only the ambient and diffuse color component.

5. Render the teapot using all three ambient, diffuse and specular color component.

4 Challenge

1. Create a Phong reflection model using the fragment shader. Your render should be similar to the next Figure.

Figure 6: per-fragment Phong illumination.

References

[1] Phong reflection model `http://en.wikipedia.org/wiki/Phong_reflection_model`, last access on 15/04/2015.

[2] Shading Language Reference `http://www.shaderific.com/glsl/`, last access on 15/04/2015.

[3] GLSL common mistakes `https://www.opengl.org/wiki/GLSL_:_common_mistakes`, last access on 15/04/2015.

[4] The OpenGL Shading Language `https://www.opengl.org/registry/doc/GLSLangSpec.4.40.pdf`, last access on 15/04/2015.