Programming Assignment #7

CS 200, Fall 2024

Due Friday, November 8

In this assignment, you will write the vertex and fragment shaders for a GLSL shader program for rendering textured objects that are illuminated using a 2D spot light.

Spot light (with ambient light)

A spot light concentrates light in a circular region. Very little light reaches points outside of the region. Points near the boundary of the spot light receive some light, while points far away receive almost no light at all. We can model the fraction of light received by the spot light by the function

$$f_{\text{spot}}(P) \doteq \begin{cases} 1 & \text{if } \|P - C\| \leq r \\ \left(\frac{r}{\|P-C\|}\right)^2 & \text{if } \|P - C\| > r \end{cases}$$

where C is the center point of the spot light, and r is the radius of the spot light. Using the minimum value function, we can write this as

$$f_{\text{spot}}(P) = \min\left\{\left(\frac{r}{\|P - C\|}\right)^2, 1\right\}. \tag{1}$$

Points outside of the spot light are not illuminated, so would be colored black. As this is not usually desirable, we also introduce an *ambient light* factor. This will allow points to receive light, regardless whether they are in the spot light or not. With such a factor, the total light fraction receive by both the spot light and ambient light is

$$f_{\text{tot}} = f_{\text{amb}} + \mu f_{\text{spot}} \tag{2}$$

where μ controls the spot light intensity, $0 \leq \mu \leq 1$. Note that as $0 \leq f_{\text{amb}} \leq 1$ and $0 \leq f_{\text{spot}} \leq 1$, it is possible for f_{spot} to be greater that 1.

Task #1: vertex shader

The spot light position and radius are given in terms of world coordinates, so the vertex shader needs to send the (interpolated) position a pixel/fragment in world coordinates, as well as the (interpolated) texture coordinates of the pixel/fragment.

In particular, the vertex shader should have two attribute (in) variables: one for the mesh vertex coordinates in object space, and one for the texture coordinates for the vertices.

```
in vec4 position;
in vec2 texcoord;
```

(you can also use the older keyword `attribute` in place of `in`).

There should also be two uniform variables:

```
uniform mat4 object_to_world;
uniform mat4 world_to_ndc;
```

The first uniform variable is for the modeling transformation, which will transform the mesh vertices from object space to world space. The second is for the (net) transformation from world space to NDC. Recall that this is a combination of the world–to–camera (viewing) transformation and the camera–to–NDC transformation.

The vertex shader should have two output (varying) variables: one for the (interpolated) world space coordinates of the pixel, and one for the (interpolated) texture coordinates of the pixel.

```
out vec2 interp_world_position;
out vec2 interp_texcoord;
```

The `main` portion of the vertex shader should do two things: (1) compute the NDC coordinates of the mesh vertices, and (2) compute the values of the output variables. For the first task, you will store the result in the OpenGL–defined variable `gl_Position`. For the second task, the value of `interp_world_position` is obtained by converting the value of the attribute `position` from object space to world space. Whereas the value of `interp_texcoord` is just the same as that of the attribute `texcoord`. However, note the variable types for the output variables, namely `vec2`. The "swizzling" operator in GLSL is useful here. For example, the fragment

```
vec4 v(1,2,3,4);
vec2 u = v.xz;
```

will result in $u = \langle 1, 3 \rangle$ (there is no good reason to have `interp_world_position` be of type `vec2` instead of the usual `vec4` — it is just a vehicle to introduce swizzling).

Task #2: fragment shader

The purpose of the fragment shader is to assign a color to a pixel within a mesh. In our case, we will use a texture map look–up to obtain the basic color, and use equation (2) to modify the intensity of the basic color.

The vertex shader described above has two output variables: one for the position of the pixel in world coordinates, and one for the texture coordinate of the pixel. We use these two variables as input variables in the fragment shader:

```
in vec2 interp_world_position;
in vec2 interp_texcoord;
```

In general, for each output variable of the vertex shader, we must have a corresponding input variable in the fragment shader, with the *same* exact name.

To facilitate texture look–up, we need a uniform sampler variable. We also need uniform variables for each of the parameters in equations (1) and (2). Specifically, we have the following.

```
uniform vec4 light_position;
uniform float light_radius;
uniform float light_factor;
uniform float ambient_factor;
uniform sampler2D usampler;
```

Finally, we need an output variable to to store the actual computed pixel color:

```
out vec4 frag_color;
```

However, you can choose any name for this variable, as long as it does not conflict with any other variable names.

In the `main` function of the fragment shader, you will need to (1) get the texture color corresponding to the input texture coordinate, and (2) multiply this color by the lighting factor f_{tot} in equation (2). Note that the value of f_{tot} can be greater than unity, which can result in RGB components outside of the range $[0, 1]$. While you can explicitly use the GLSL `min` or `clamp` functions, the graphics card will automatically clamp final RGB values.

What to turn in

You should submit two files: (1) the vertex shader source file, which should be named `light.vert`, and (2) the fragment shader source file, named `light.frag`. You may only use GLSL version 1.3 for this assignment.