1. Consider the OpenGL code diagram depicted in the last page. Describe briefly with your own words each one of the following functions. Look at the OpenGL documentation for reference.

Llink: https://www.khronos.org/registry/OpenGL-Refpages/gl4/
Google: "opengl 4 references"

`glCreateShader`

input: GLenum shaderType (GL_Compute_Shader, GL_Vertex_Shader, ...)

output: GLuint type with reference to newly made Shader of type shaderType

`glShaderSource`

input: GLuint shader, GLsizei count, const GLchar ** strng, const GLint* length

(int shader type, sizei type, char type, int length)

`glCompileShader`

input: GLuint shader

`glCreateProgram`:

output: GLuint type (an empty Program object)

`glAttachShader` (attaches Shader object to program object)

input: GLuint program, GLuint shader

`glLinkProgram` Links a program object

input: GLuint program

`glUseProgram`

input: GLuint program

2. Read the comments and order the lines of code in correct order for loading shaders.
Fill in the blanks afterwards.

```
A.glCompileShader( fragment_shader ); // compile fragment shader
B.glAttachShader( program , vertex-id ); //attach vertex shader to program
C.GLuint vertex_id = glCreateShader( vertex shader ); // create vertex shader
D.glCompileShader( vertex-id ); //compile vertex shader
E.glAttachShader( program ,fragment_id); //attach program shader to program
G.glShaderSource( vertex-id ,1,&vertex_shader_file,NULL); //source vertex shader
F.glLinkProgram( program ); //link program
H.GLuint fragment_id=glCreateShader( fragment-shader); // create fragment shader
I.glShaderSource( fragment_id ,1,&fragment_shader_file,NULL); //source fragment shader
J.GLuint program = glCreateProgram();
```

Ordering: J, C & H, G & I, A & D, B, E, F
(The answer may vary.)

3. Consider the following GLSL vertex (left) and fragment (right) GLSL codes.

```
void main() {
  gl_Position = gl_ProjectionMatrix
      * gl_ModelViewMatrix
      * gl_Vertex;
  gl_FrontColor =
      vec4(0, 1, 0, 1);
}
```

```
vec4 light_color = vec4(1, 0, 0, 1);

void main() {
  gl_FragColor = gl_FrontColor;
}
```

The vertex shader receives a `vec4` **`gl_Vertex`** and returns a `vec4` **`gl_Position`**.
`gl_ProjectionMatrix` and `gl_ModelViewMatrix` are transformation matrices given by OpenGL.

The fragment shader receives **`gl_FrontColor`** from the vertex shader and returns the color of the fragment as **`gl_FragColor`**.

3.1. What is the output color of the fragment shader?

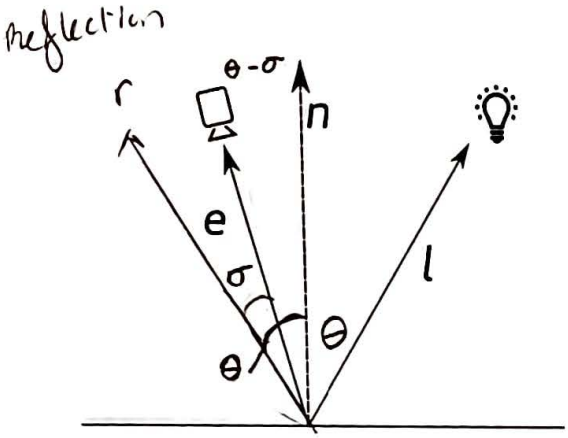
`gl_FragColor = (1, 1, 1, 3)`

3.2. Consider an object with color green represented by the RGB color vector (0, 1, 0) and a blue light source with color (0, 0, 1). If we illuminate the object with the light, what is the output color? Green w/ blue

(0, 1, 1)

CS130 LAB: 4 – Part 2: Phong model

Write the equations for the the Phong model components. Draw any missing vectors in the figure below.

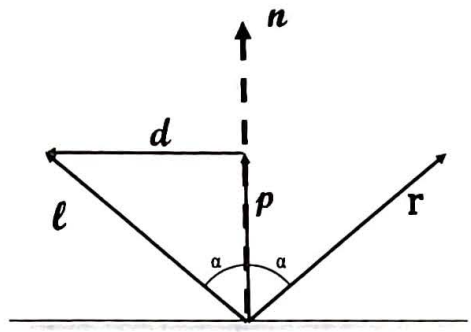


Ambient: $C_r(C_a + C_L \max(0, n \cdot l))$

Diffuse: $C_r C_l |n \cdot l|$

Specular: $r = d - 2(d \cdot n)n$

In the figure below, vector r is the reflection of vector l from the surface, and n vector is the unit-length normal of the surface.



Write the reflection vector r in terms of n and l, following the steps below:

Step 1: Formulate vector p, which is the projection of l on n, in terms of l and n.

$p = \frac{n}{\|n\|}(n \cdot l)$

Step 2: Formulate vector d, in terms of l and p

$d = (l - p)$

Step 3: Write vector r in terms of d, p and l (you do not have to use all of them)

$r = l - 2(l \cdot p)p$

Step 4: Substitute p and d, with your results from steps 1 and 2, and write r in terms of l and n only.

$r = l - 2\left(l \cdot \left(\frac{n}{\|n\|}(n \cdot l)\right)\right)\left(\frac{n}{\|n\|}(n \cdot l)\right)$