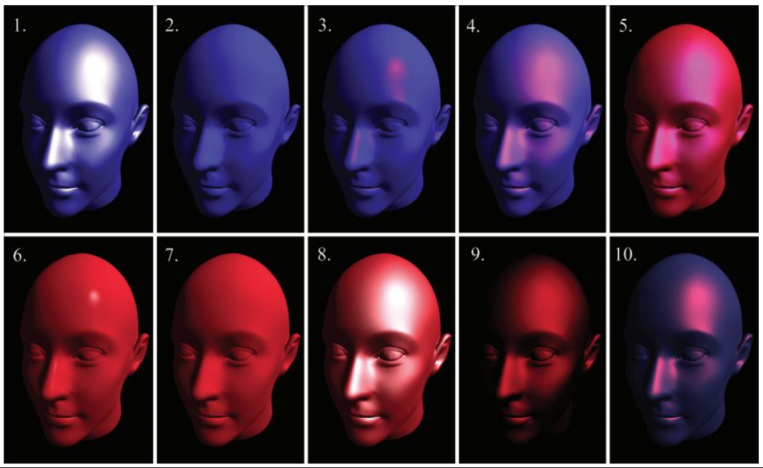
Homework 4 - Lighting

1. **Define the three components of light used in the Phong lighting model: diffuse, specular, and ambient.**

2. **Match the images below to the properties defined in the table. Write the image number in the correspondent table row.**



	Diffuse Color	Specular Color	Specular Exponent (giving the size of the highlight)	Image Number
(a)	Red	Black	Any	
(b)	Blue	Red	Small	
(c)	Blue	White	Big	
(d)	Darker Blue	Red	Medium-Big	
(e)	Blue	Red	Big	
(f)	Blue	Black	Any	
(g)	Black	Red	Big Red	
(h)	Red	White	Small	
(i)	Red	White	Big	
(j)	Red	Blue	Big	

3. **Consider a scene with a single sphere of radius 1 centered at the origin (0,0,0). The sphere's ambient coefficient is k_a = RGB(0.1,0.1,0.1), diffuse coefficient k_d = RGB(0.5,0.5,0.5), and specular coefficient k_s = RGB(0.5,0.5,0.5), with specular exponent s = 2. There is a light at position p_1 = (0,2,0) with color c_1 = RGB(1.0, 0.0, 0.0). The camera is located at position p_2 = (3,1,0). Calculate the Phong Illumination for the sphere fragment located at position p_3 = (0,1,0).**

   (a) **Calculate the ambient portion of the color of this fragment.**

   (b) **Calculate the diffuse portion of the color of this fragment.**

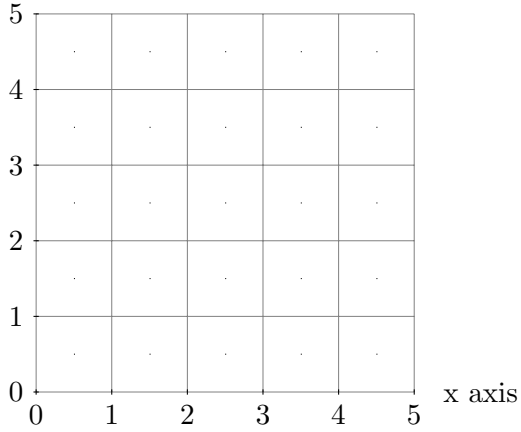
(c) **Calculate the specular portion of the color of this fragment.**

(d) **Calculate the total color of this fragment.**

(e) **Now suppose the camera moves to p_4 = (0,3,0). What is the total color of the fragment?**

4. **Consider an orthographic scene with objects listed below being rendered at 5 x 5 pixels with with the view plane at z=0 (near=0, far=99). Show the state of the z-buffer (with numbers) once these squares have been rendered (without using anti-aliasing or sub-sampling). Note that the pixels are drawn such that the z-buffer values are stored halfway between integers (look at the labels below the z-buffer).**

   Red square with vertices: (0,0,1), (1,0,1), (1,1,1), (0,1,1)
   Green square with vertices: (2,2,2), (4,2,2), (4,4,2), (2,4,2)
   Blue square with vertices: (0,0,3), (4,0,3), (4,4,3), (0,4,3)



5. Consider the following vertex and fragment shaders:

```
// Vertex Shader
uniform mat4 u_NormalMatrix;
uniform mat4 u_ModelMatrix;
uniform mat4 u_ViewMatrix;
uniform mat4 u_ProjectionMatrix;

attribute vec4 a_Position;
varying   vec3 v_Position;
attribute vec4 a_Normal;
varying   vec3 v_Normal;

void main() {
    v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));
    v_Position = vec3(u_ModelMatrix * a_Position);
    gl_Position = u_ProjectionMatrix * u_ViewMatrix * u_ModelMatrix * a_Position;
}‘;

// Fragment Shader
varying vec3 v_Position;
varying vec3 v_Normal;
uniform vec3 u_LightPos;
```

```
uniform vec3 u_LightColor;

void main() {
    vec3 l = normalize(u_LightPos - v_Position);
    float nDotL = max(dot(l, v_Normal), 0.0);
    gl_FragColor = vec4(u_LightColor * nDotL);
}
```

(a) **In the vertex shader, what is the normal matrix and why do we need to multiply it by the normal vector?**

(b) **In the vertex shader, why do we multiply the vertex position by the model matrix?**

(c) **What is the fragment shader computing?**