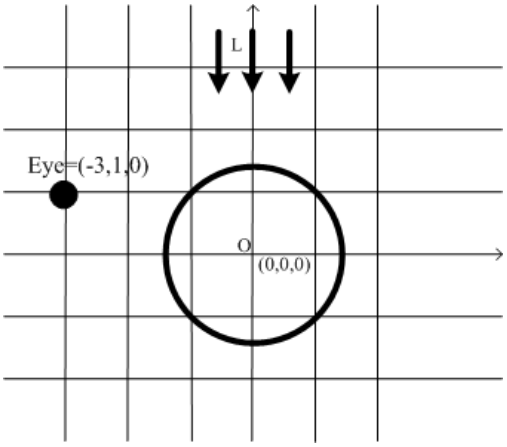
CPSC 314
Theory Assignment 3 - Solution

November 7, 2013

1. Lighting:

   The scene below consists of: a sphere of radius $\sqrt{2}$ centered at origin with $k_d = (1, 0, 0)$ and $k_s = (1, 1, 1)$; a parallel (directional) light $L = (0, -1, 0)$ with $I_d = I_s = (1, 1, 1)$; and an eye location, as shown, at $(-3, 1, 0)$. Assume there are no other light-sources.



   (a) At what point (coordinates) on the sphere will we get maximal specular reflection (white dot)? Explain your answer.

   In any common model of lighting the point with maximal specular reflection is the one that follows Snell's law. I.e. maximal illuminance reaches at the point P such that a ray, after collision with surface at P, is being reflected exactly to the eye. In other words, light direction and look direction towards P have the same angles with the normal at P. In order to find P with that property, the very first thing we can notice is that the point P should have $z = 0$. This leaves us with 2D problem overall - sphere now is a circle. Now in general case we would have to the following equation:

$$\frac{\mathbf{n_P} \cdot (P - eye)}{||P - eye||} = \frac{\mathbf{n_P} \cdot LightDir}{,}$$

   where $\mathbf{n_P}$ is normal in P, eye is eye location, LightDir is normalized light direction. But in our case it's very easy to find the only visible point on the sphere with the

sought property: it's $P = (-1, 1)$. *We can easily make sure that normal* $\mathbf{n_P} = (-1/\sqrt{2}, 1/\sqrt{2})$ *makes equal angles with* $Eye - P$ *and* $LightDir = (0, -1)$.

(b) At what point (coordinates) on the sphere will we get maximal diffuse illumination (red dot)? Explain your answer.

For diffusion, the point of maximum illumination is such that normal is collinear with the light direction. Here the location of the eye is ***not*** *important. Here the lit point with such property is the top of the sphere* $(0, \sqrt{2})$, *this is our final answer.*

(c) Given a single ambient light source with $I_a = (1, 0, 0)$ and a triangle P_1, P_2, P_3 with $k_a = (0, 0, 1)$, what color will be assigned to P_1 using the light equation? Show your work.

As we know, the ambient light influence on surface's color does not depend on the normal of the surface. To be more precise, the influence of ambient light is just a dot product of light's color/intensity and object's k_a, *i.e.* $I = I_a \cdot k_a$. *Here,* $I = (1, 0, 0) \cdot (0, 0, 1) = (0, 0, 0)$, *i.e. the whole triangle will be black.*

2. Light and shading

(a) Given a scene with two non specular objects, one yellow ($k_a = k_d = (1, 1, 0)$) and one red ($k_a = k_d = (1, 0, 0)$), classify the following statement as true or false. Explain.

i. Given a single point light source with intensity $I_p = (1, 0, 0)$ the objects will have the same shading. *False. As the shading of an object induced by a point light depends on the normals of the object, the shading will be the same iff the objects are exactly the same and their positions coincide.*

ii. Given a single ambient light source with intensity $I_a = (1, 0, 0)$ the objects will have the same shading. *True, as abmient shading does not depends on object normals and both objects will glow with red:* $I_1 = k_a^1 \cdot I_p = k_a^2 \cdot I_p = I_2 = (1, 0, 0)$

(b) Write the openGL code for defining the following lighting scenario with three light sources: ambient light source with intensity $I_a = (0.3, 0, 0)$; directional light with direction $(1, 0, 0)$ and intensity $(0.6, 0.6, 0.6)$; point light at $(10, 0, 0)$.

The minimal answer to the question would be the following. Ambient shading doens't depend on the ambient light's position, so we don't have to specify the position for the first light.

```
glLightfv(GL_LIGHT0,GL_AMBIENT,{0.3,0,0,1});
```

To add the direction light, the only trick is that we specify the direction in the GL_POSITION with forth component of the position equal to 0 - that makes OpenGL treat the coordinates as direction, not location:

```
glLightfv(GL_LIGHT1,GL_POSITION,{1,0,0,0});
glLightfv(GL_LIGHT1,GL_DIFFUSE,{0.6,0.6,0.6,1});
```

And finally, adding the third light is easy:

```
glLightfv(GL_LIGHT2,GL_POSITION,{10,0,0,1});
```

If you want to be exceptionally correct, according to OpenGL specification, default GL_DIFFUSE and GL_SPECULAR values for GL_LIGHT0 are (1,1,1,1), though for all the other lights it's (0,0,0,1). So to make our GL_LIGHT0 only ambient, we will have to overwrite the default values by adding two more function calls:

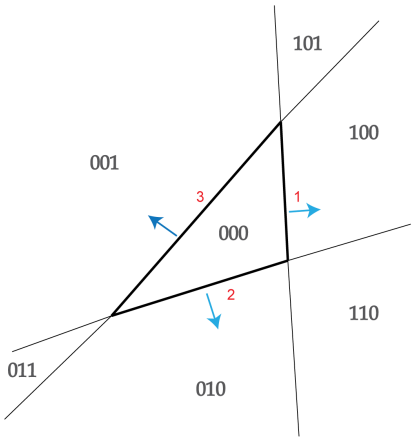
```
glLightfv(GL_LIGHT0,GL_DIFFUSE,{0,0,0,1});
glLightfv(GL_LIGHT0,GL_SPECULAR,{0,0,0,1});
```

(c) In openGL define the material properties for a triangle with $k_a = (1, .5, .5)$, $k_d = (1, .5, .5)$, $k_s = (.5, .5, .5)$ and specularity coefficient $n = 16$.

```
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, {1, .5, .5, 1}
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, {.5, .5, .5, 1});
glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, 16);
```

3. Clipping

(a) Write an algorithm (pseudo-code) for clipping a line $L = P_1P_2$ ($P_1 = (P_1^x, P_1^y)$, $P_2 = (P_2^x, P_2^y)$) against a triangle $T = (T_1, T_2, T_3)$ with $T_1 = (T_1^x, T_1^y)$, $T_2 = (T_2^x, T_2^y)$, $T_3 = (T_3^x, T_3^y)$ (in 2D). Follow the framework of the Cohen-Sutherland algorithm for clipping a line against a window.



First we enumerate all the sides of the triangle in an arbitrary way (red numbers on the figure). Then we orient each side, i.e. choose normal direction. For convenience, we for each side we choose outer normals with regard to the triangle - that will give us later code 000 inside the triangle. Then for any point on the plane we can calculate its 3-digit binary code, where i-th digit means whether the point is to the left or to the right of i-th line. Then the algorithm goes like that:

```
CsClip(P_1, P_2, T_1, T_2, T_3)
C_1 <- Code(P_1, T_1, T_2, T_3);
C_2 <- Code(P_2, T_1, T_2, T_3);
if (C_1 & C_2) != 0) return;
if (C_1 | C_2) == 0) DrawLine(P_1,P_2);
// Here we assume that P_1 has maximum code, i.e. C_1 > C_2
Edge <- triangle boundary corresponding to
the leftmost non-zero component of C_1;
P_3 <- P_1 P_2 intersect with Edge
return CsClip(P_2,P_3,T_1,T_2,T_3);
```

(b) Explain how to extend your algorithm for clipping the line L against a convex polygon $T = (T_1, T_2, \ldots, T_n)$.

The only thing that changes in the algorithm above is that we will have n-digit code. It's easy to notice that because of the recursion and the fact we're finding location of each queried point with regards to every line, the algorithm becomes inefficient with large ns.

(c) Will your algorithm work for non-convex polygons? Explain.

No, it won't work: here we use the fact that if the point lies on the same side of every line forming the polygon, then the point is inside - which is not true for non-convex polygons.

4. Bresenham

Write the Bresenham algorithm for rasterizing a line from (x_1, y_1) to (x_2, y_2) where $x_1 \geq x_2$, $y_2 > y_1$ and $x_1 - x_2 < y_2 - y_1$.

```
Line (x1, y1, x2, y2)
{
int x,y,dx,dy,d,delta_w, delta_nw;
x = x1; y = y1;
dx = x2 - x1;
dy = -(y2 - y1); // We take dy with negative sign
//so that we keep the rest of the algorithm untouched
d = -2*dy - dx;
delta_w = 2*dy;    delta_nw = 2*(dy-dx);
PlotPixel (x,y);

//and now the only difference is that we're moving UP
//i.e. increase y each time
while (y < y2)
{
 if (d<0)
  {
   d = d + delta_w;
   }
  else
  {
   d = d + delta_nw;
   x--;
  }
 y++;
 PlotPixel(x,y);
}
}
```

5. Clipping (Bonus question)

Use the definition of convexity to prove that the intersection of two convex objects is convex.

Let's take two convex objects A and B. By definition,

$$\forall x, y \in A \text{ and } \alpha \in [0,1], (1-\alpha)x + \alpha y \in A$$

and the same holds for B. Now let's take arbitrary $x, y \in A \cap B$ and $\alpha \in [0, 1]$. Then by definition of convexity of A, $p = (1 - \alpha)x + \alpha y \in A$, since $x, y \in A$. In the same way, $p = (1 - \alpha)x + \alpha y \in B$, since $x, y \in B$. Therefore, $p \in Aandp \in B$, so $p \in A \cap B$.