Saleema Amershi
75880997
Math 309 Presentation
March 15, 2004

Light Issues in Computer Graphics

An important goal for many computer graphics artists is generating realistic images and scenes. As a result, interaction of light with objects in a scene is as important as the objects themselves in synthesizing realistic images. In modeling these real world interactions in computer graphics, we can illustrate 3D properties of objects, shadows from occlusion, refraction from translucent object and more.

Local Illumination models are used in computer graphics to understand and demonstrate the direct interaction of a single light source with a 3D object's surface.

Global Illumination models take into consideration that in the real world, what we see is not just the effect of one light source and an object, but a complex composition of light reflecting and refracting off all surfaces in a scene and onto the object.

Local Illumination Models

Physically based models of interaction between light sources and the surface properties of objects are very complex. Computer graphics uses approximations in their models to simulate these interactions.

The Phong model is a basic local illumination model used in many computer graphics programs because of its relatively low computational overhead and its realistic effects.

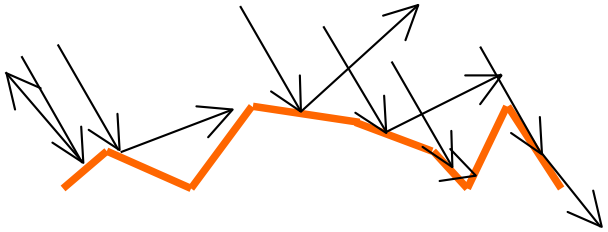
The Phong model breaks down the illumination from an object into ambient, diffuse and specular reflection components.

Ambient reflection is used as a crude approximation to global effects of light being scattered in all directions from all surfaces which gives the effect of an almost even spread of brightness. (Think about turning a light on and off). To do this in computer graphics, we simply assign a constant amount of light to every point in the scene.

$$I_{out} = k_{ambient} \cdot I_{ambient}$$

Where I is light intensity (power per unit area) or illumination, I_{out} is the light coming out from a single point on an object, $k_{ambient}$ is the ambient reflectance factor (which the user specifies when rendering graphics) which is the proportion of incoming light reflected ambiently, and $I_{ambient}$ is the intensity of light directly emitted from the light source.

Diffuse reflection models light interactions with different material properties of surfaces. It takes into account the direction and angle that the light source is relevant to the surface point, and also the roughness or smoothness of the surface. All surfaces have some roughness at the microscopic level. Here is a close up model of a surface.



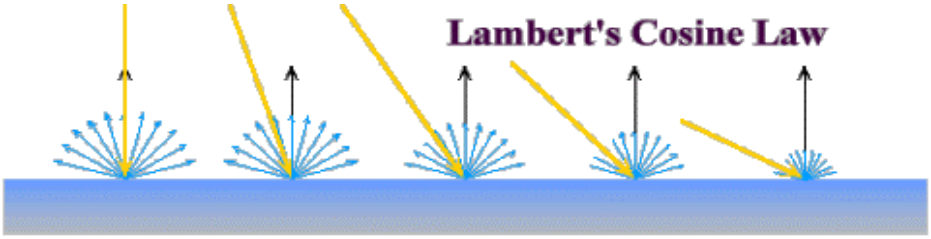
The surface is really broken up into small pieces, all facing different directions. If we assume that light reflects off of each piece like a perfect mirror, then some of the reflected light will be obstructed by other pieces causing shadows. Also, since the pieces of the surface face different directions, rays from a single light source in one direction will reflect in different directions. In an ideal situation, we can think of light coming from a single source being scattered or reflected equally in all directions from a surface.

But we haven't accounted for the intensity of this diffuse reflection, which depends on the angle that the light source is relative to the surface. Johann Friedrich Lambert described that the energy reflected off of a surface from a light source in a given direction relative to the surface is proportional to the cosine of the incident angle, *i*, between the light and the normal to the surface.

$$I_{out} \propto cos(i)$$

$$I_{out} = k_{diffuse} \cdot I_{light} \cdot cos(i)$$

Where here, $k_{diffuse}$ is the proportion of incoming light that is reflected due to diffuse interaction with a light source. This constant is attributed to the material properties of the surface. It is higher for shinier surfaces like metallics and lower for duller surfaces such as carpets and walls.



Consider a beam of light of width *w* coming from a light source onto a small area of a surface *dA*.

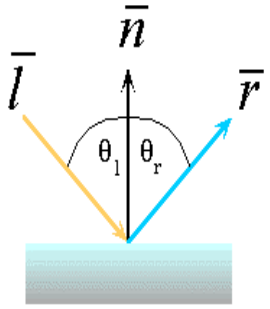
We can see that $cos(i) = w/dA$. So we have $I_{out} = k_{diffuset} \cdot I_{light} \cdot w/dA$. As the angle of incidence increases, the width of the beam decreases and thus the intensity of light coming out of the surface decreases.

So we can add the diffuse component of light interaction with a surface to the total light intensity reflecting from a small surface area:

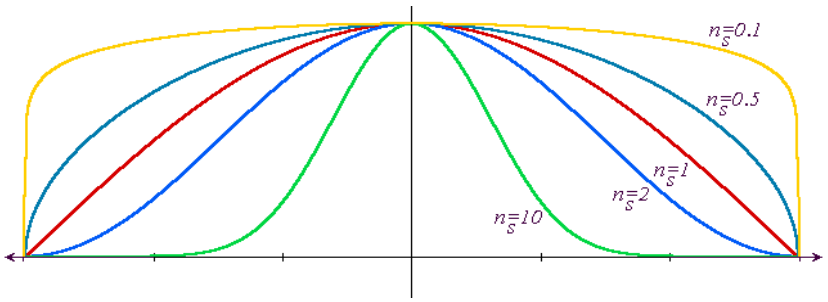
$$I_{out} = k_{ambient} \cdot I_{ambient} + k_{diffuse} \cdot I_{light} \cdot cos(i)$$

So far we have only considered direction that a light source is relative to a surface, but have not considered where the viewer is relative to the surface. The viewer direction is necessary to achieve what is known as specular reflection, which occurs from highlights of light sources off of shiny surfaces in certain directions.

Specular reflection tries to model the reflections of shiny surfaces more accurately, taking into account not only the light source, but the viewer direction from the surface as well. On an ideal mirror, the light from some source will be perfectly reflected at an angle equal to the incident angle about the normal. However, as we said earlier, surfaces are not smooth at the microscopic level, so not all light will be reflected perfectly even for smooth surfaces. Empirical observations of this suggest that off of smooth surfaces *most* light will be reflected in the direction of a perfectly reflected ray, but with slight variations around the perfect ray.



Phong Bui-Tuong modeled these observations by raising the cosine of the angle of deviation from the perfect ray by a shininess constant, *s*, which the user defines and reflects empirical observations of the shininess of materials. Below are plots of the cosine of an angle raised to different powers. We can see that the higher the exponent the sharper the plot around the ideal reflection ray. This gives the effect of specular lighting.



We can calculate the amount of light received from the viewer, by using the angle, θ, that the viewer is relative to the ideal reflectance angle in cosine term. So now we have

$$I_{out} = k_{specular} \cdot I_{light} \cdot \cos(\theta)^s$$

Here $k_{specular}$ is, as always, the constant given to the surface which describes the proportion of light reflected due to specular reflection and the surface material. So as the angle between the viewer and the perfect ray increases the amount of light received decreases. And as the shininess factor increases the more concentrated the light is around the reflected ray.

So putting it all together we have the amount of light from a small surface area being the combination of ambient, diffuse and specular reflection. This is commonly called the Phong lighting model/

$$I_{out} = k_{ambient} \cdot I_{ambient} + k_{diffuse} \cdot I_{light} \cdot \cos(i) + k_{specular} \cdot I_{light} \cdot \cos(\theta)^s$$

This only models light from one light source on a surface. In reality, however, there could be many light sources in a scene, each contributing to the light reflected from a surface. So to calculate accurately, we must sum the diffuse and specular components from all light sources in a scene (note that ambient reflection is overall lighting and does not depend on the light sources).

$$I_{out} = k_{ambient} \cdot I_{ambient} + \sum(k_{diffuse} \cdot I_{light} \cdot \cos(i) + k_{specular} \cdot I_{light} \cdot \cos(\theta)^s)$$

$$= k_{ambient} \cdot I_{ambient} + \sum I_{light} (k_{diffuse} \cdot \cos(i) + k_{specular} \cdot \cos(\theta)^s)$$

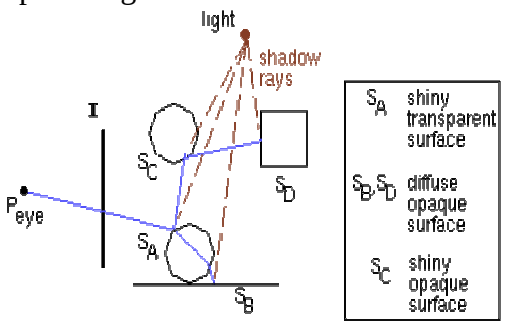
Global Illumination

So far we have introduced a basic model for calculating the light intensity reflected from a small surface area of an object. This model can itself be used to render scenes in computer graphics using the z-buffer algorithm.

Z-buffering is a very common algorithm for checking visibility in computer graphics. In a nutshell, in a simulated scene, visible objects lie within a viewing frustrum. These objects are then projected onto a viewing plane between the viewing frustrum and the camera position (ie. the viewer). In z-buffering, each pixel is first set to the background color of the scene. Then for each object in the scene, 3D positions (x,y,z) and directions are calculated fore each point (or small surface area) on the object. If the z-value of this point is greater then any z-value of previously rendered objects (ie. it is in front of all other objects already rendered) then the pixel that the point projects onto is set to the illumination of the point. If the z-value is not greater then all other points projecting to the current pixel, then the point is ignored (ie. there is something in front of it).

Using the z-buffering algorithm and the local illumination model, we can light up a scene, however, we do not get the realistic effects of shadows, reflections between objects and translucency of objects.

To see these global illumination effects, a better method uses *ray tracing!* With z-buffering, we modeled light coming from an object to the camera (or the eye). But with ray tracing we cast a ray from the camera through every pixel in the image plane until it intersects an object. A ray from the camera to an object is called a primary ray. Once we get an intersection point, we can calculate the local illumination for that point and then set the pixel to that value. But to make use of the benefits of ray tracing, we can shoot off secondary rays from this point using refraction and reflection and add these effects to the color value of the pixel to get realistic effects.



The largest overhead for using ray tracing methods in computer graphics is the cost of calculating intersections of rays with objects. Most objects can not be represented by simple equations such as spheres, where we can easily calculate intersections with rays. Instead, objects are modeled as collections of polygonal planar surfaces.

To calculate the intersections, we use an implicit plane equation and a parametric equation for the ray and solve for *t:*

$F(x,y,z) = Ax + By + Cz + D = 0$ for points (x,y,z) on the plane defined by F

Or

$F(P) = N \cdot P + D = 0$ where N is the normal vector for the plane which can be computed by taking the cross product of two vectors making up the plane.

$N = (P_2 - P_1) \times (P_3 - P_1)$ where P_1, P_2, P_3 are non-collinear points on the plane.

And

$P(t) = P_a + tv$ where P_a is the camera position or the pixel position (where the ray begins) and v is the unit vector in the direction of the ray.

$v = P_b - P_a$ where P_a and P_b are points on the ray.

Then to find the intersection we plug the ray equation into the plane equation and solve for t.

$$F(P(t)) = N \cdot (P_a + t(P_b - P_a)) + D = 0$$

$$t = (-D - N \cdot P_a) / (N \cdot P_b - N \cdot P_a)$$

Then plug in t back into $P(t)$ will give us the intersection point. (Remember that when we solve for t we are solving a quadratic, so we will get two possibilities for t. We want to compute the first intersection of a ray to an object, so use the smallest t value).

After we have computed the point of intersection of the ray with the plane, we need to test whether this point is actually within the bounds of the polygonal surface that lies on the plane. The way this is usually done is by a method of calculating what is known as barycentric coordinates. This is out of the scope of this lecture.

Like we said earlier, computing these intersections is very computationally high. There are many ways we can reduce the number of these calculations, by doing checks as to whether there actually will be an intersection point between a plane and a ray before we actually compute the intersection. We wont cover these methods here.

Now that we have an intersection point, we want to use ray tracing to show realistic effects of illumination. One major effect is shadows. Shadows can be calculated by combining ray tracing and the Phong illumination model. Once we have an intersection point, we look at every light source in the scene and cast a ray from the light source to this point of intersection. If this ray hits another object before it hits our point of intersection, that object obstructs the effect of light from that light source so we disregard that light source in our illumination model. If the ray from the light source is unobstructed then it is added to the illumination of the point in question. So adding this effect to our local illumination model, we get:

$$I_{out} = k_{ambient} \cdot I_{ambient} + \sum b_i \, I_{light(i)} \, (\, k_{diffuse} \cdot cos(i) + k_{specular} \cdot cos \, (\theta)^s \,)$$

Here b_i is a Boolean value associated with each light source for the current point. It is set to 0 if light source i is obstructed (so that light sources illumination effects are ignored) and set to 1 if the light source is unobstructed.

We can also recursively calculate reflected light off of objects who are obstructing light sources to get an even more realistic effect.

A reflected ray R can be expressed as a vector, by:

$R = L - (2L \cdot N)N$ where L is the vector of light from a source to the surface (the incident ray) and N is the normal vector to the surface

Another realistic effect that we can show using ray tracing is that of refraction of light through translucent materials. Just as we sent secondary rays out towards other objects and light sources due to reflection, we can send secondary rays through transparent surfaces using refraction.

To calculate refraction angles, we use Snell's law, which we all know is:

$$n_i \sin(i) = n_r \sin(r)$$

In computer graphics, we use vector calculations so we replace the above equation by:

$$n_i (L \times N) = n_r (R \times N)$$

To calculate the refracted vector R is quite difficult using vector math. But in a nutshell we get:

$$R = n_i / n_r (\cos(i)) - \cos(r)N - n_i / n_r L$$

$$= n_i / n_r (N \cdot L) - \sqrt{(1 - (n_i / n_r)^2 (1 - (N \cdot L)^2)} * N - n_i / n_r I$$

After we calculate the reflected or refracted vector, we can cast a secondary ray in the direction of this vector starting from out original point of intersection. We stop when this ray hits another object surface and then add the illumination of this point using our illumination model to the original point and recurse.

However, the secondary intersection point illuminations don't carry as much weight on the illumination of the pixel from which the primary ray generated because as distances between objects increases, the intensity of light from those objects decrease quadratically with distance. This is called light source attenuation in computer graphics. Therefore, when we are accumulating intensities, we add a weighting factor for each additional intensity which is based on the distance between the point and the previous intersection.

Putting it all together, we get a ray tracing algorithm for lighting a scene in computer graphics using Phong illumination which can render scenes with basic realistic effects:

```
raytrace(ray from a pixel)
        -calculate closest intersection
        -calculate local illumination for intersection point   //take into account shadows
        -reflected_component = raytrace(reflected_ray)         // if an object surface has
                                                                reflection properties (ie. is
                                                                not completely diffuse)
        -refracted_component = raytrace(refracted_ray)         //if an object surface is
                                                                transparent
```

$$\begin{aligned} \textit{-color\_of\_pixel} &= c_1 * \textit{localcolor} \\ &+ c_2 * \textit{reflected\_component} \\ &+ c_3 * \textit{refracted\_component} \end{aligned}$$

References:

Angel, Edward. Interactive Computer Graphics: A Top-Down Approach Using OpenGL. Addison Wesley, Boston (2003). Ch.6.

Angel, Edward. OpenGL: A Primer. Addison Wesley, Boston (2002).Ch.6.

http://www.doc.ic.ac.uk/~dfg/graphics/GraphicsLecture11+12.pdf

http://www.siggraph.org/education/materials/HyperGraph/illumin/similum0.htm

http://www.dgp.toronto.edu/~karan/courses/csc418/fall_2002/notes/illum_local.html

http://www.ugrad.cs.ubc.ca/~cs414/Vjan2004/S-lighting.pdf

http://www.ugrad.cs.ubc.ca/~cs414/notes/raytrace.html

Pictures

http://www.cl.cam.ac.uk/Teaching/2003/Graphics/shutterbug.html

http://www.cg.tuwien.ac.at/research/rendering/rays-radio/

http://www.electricimage.com/community/gallery_illumination2.html

http://www-graphics.stanford.edu/~cek/rayshade/rayshade.html

http://www.wis.co.uk/justin/raytrace-gallery.html