

Lighting and Shading

Notes, CMP 80c, Winter 2001

1 Shading

To render an object with a surface, you have to somehow find color values to draw to the screen. In the simplest case, you can just assign color to the object. But then the object will be one color and won't look realistic, or take into account lights. It will appear flat and two-dimensional.

So, to use *shading*, we associate characteristics with the surface that indicate how it responds to light, and also specify light in the scene. The process of creating the 2D image is called *rendering*. In the simplest case, we associate what looks like a color, that is, a value for red, green, and blue. Actually, these colors are *coefficients of reflectivity* that determine what color is reflected from the surface given the light in the scene. Because computer graphics normally considers colors as combinations of red, green, and blue, calculations for shading are done independently in terms of red, green, and blue.

By using a mathematical model of how light is reflected from surfaces, computer graphical programs can take the lights in a scene and the properties of surfaces and create a quite realistic image. Realism is directly related to the amount of calculations that must be done, so a highly realistic image may take several minutes or even hours to render.

1.1 Lights

Lights make the scene appear realistic and three-dimensional, and further create the mood for the scene. The same models can look very different under different lighting conditions. In the real world, photographers use a *light meter* to measure the light in a scene, and must carefully control the light by altering the *f/stop* on their camera and using the appropriate film speed. Computer graphical scenes have the major advantage that lighting conditions can be easily altered iteratively until the desired effect is found, without wasting film or trying the patience of the subjects or actors. Sometimes, when computer graphical effects are added to real images, the lighting conditions in the real world must be carefully imitated.

Lights normally have a location (or direction) and a color and possibly other characteristics. Different kinds of lights can be simulated.

1. A *point* light is a light at a certain position in space that radiates in all directions from that point. An example of a point light might be a small bulb.
2. A *spot* light is similar, but it can radiate out in a limited range, spreading away from a certain direction. It has special parameters such as angle and drop-off. A *fresnel lens* can be simulated to create a particularly sharp beam. Light comes out as a cone or pyramid. Examples of spot lights are flood lights and flashlights.
3. A *directional (infinite)* light is like a point light that is very very far away (at infinity) so all the rays come in along a certain direction. The sun is an example of a directional light.

4. A *linear* light has length but no width, and can be simulated by a continuous series of point lights. A long fluorescent tube can be simulated by a linear light.
5. A *area* light emits light from a rectangular area. This can be used to simulate light coming through a window, or a light box, or fluorescent lights covered by a diffusing panel.
6. An *ambient* light is thoroughly unrealistic and provides a certain amount of background illumination irrespective of direction. It crudely imitates the light that is continually bounced around off of surfaces, so that all surfaces receive some light, even if they are not directly in the path of a light. A more realistic and expensive way to model this environmental light as well as reflected light is called *radiosity* in computer graphics.

1.1.1 Light Characteristics

The light characteristics light can be specified as a color triplet (for red, green, and blue). Think of 0 as no light and 1 as maximum light. A bright white light has a RGB value of (1,1,1) (R,G,B). Black (no light) is (0,0,0). A bright red light is (1,0,0), yellow is (1,1,0), etc. A dimmer bluish light with some green might be (0,0.3,0.5). While we speak of the sun and interior light as being “white”, in fact it is usually slightly tinted, and the tints can create important effects. I.e., blue tints can appear cold, and red tints warm and inviting.

In the real world, light can be measured in watts (the power of the lamp), or footcandles (the light output). Unfortunately, such measurements are not yet commonly used in graphics software.

Specifying the color of a light is similar to specifying the color of a surface. The HSV color model, or any other, can be used. Lights are *additive*, in that the effect of each light is added up to produce a sum over all lights in the environment. Light may *decay* over distance so that distant objects receive less of it. Sometimes lights are given a *glow*, so that they are visible in a scene. Otherwise, only the effect of light on surfaces is produced.

Other characteristics of lights depend on their type, and may include *position*, *direction*, *drop – off*, *beam angle*. Lights may be positioned in a spherical coordinate system rather than the normal Cartesian one. The *altitude* would be the height above the horizon, or the *latitude* is the “north pole” is directly above. The *azimuth* is the angle relative to some starting position on the horizon, or the *longitude* in earth coordinates. Lights may be animated. In software packages, lights can generally be turned on and off interactively to determine the effect of each. One must be careful not to introduce too many lights into a scene, or a maximum will be reached and all objects will appear bright light.

1.1.2 Shadows

In principle, all light sources produce *shadows*. However, shadows are quite expensive to simulated with a computer, because one must test whether an object is blocking a light for each point of the surface being rendered. Therefore, modeling software generally allows you to turn shadows off; sometimes this can be set independently for each light. Shadows have two regions: an *umbra* is the part that completely blocks a light, and the *penumbra* is the edge of this where light is only partially blocked. If the shadows are cast by stencils to create a pattern, these are called *gobo lights*.

1.1.3 Scene Lighting

Lights should depend on the effect one wishes to achieve with them. Objects of importance should be brought out, and shadows should not obscure them. Placement of light sources based on traditional stage lighting

are useful.

These categories of lighting should be considered:

1. The *main action area* by *key lights*. These lights set the mood and ensure that actions will be understood. Light may have to vary some depending on the camera placement, but should not give the effect of the being a different scene.
2. The *secondary action area* where some action takes place. Usually fewer lights are used and again they may vary with the action. Examples might be a character wandering to a different corner of the room.
3. The *background* is like the stage in theatre, and may be a room, or the outdoor environment.
4. *Key lights* illuminate the main action and *fill lights* affect the overall scene.

Certain light arrangements have also been defined:

1. A *45-degree pair* (“ordinary lighting”) is two spot lights above, to the sides, and in front of the point of interest shining at it. Their beams are 90 degrees apart, and rotated 45 degrees down from horizontal and 45 degrees in front. This arrangements produces plenty of light on the main subjects and creates interesting shadows.
2. A *frontal light from below* is unusual in the real world and casts interesting shadows and may appear artificial and dramatic.
3. A *frontal light at subject level* tends to make the subject appear rather homogeneous and flat, but a small amount can aid spot lights effectively.
4. A *lateral light at subject level* increases contrast across the subject.
5. A *lateral light from above* produces shadows on the floor and can act rather like the 45-degree spot pairs.
6. A *lateral light from above and behind* defines the subject against the background, may create a halo effect, and can produce specular highlights (more later).
7. *Overhead lights* can add drama by creating top to bottom contrast on the subject.

1.2 Surface Reflection

What happens when a particular color light strikes a surface? For example, a slightly unsaturated (pastel-ish) orange surface might be (0.8, 0.5, 0.2) in (R,G,B). This surface is orange-ish because it reflects mostly red and quite a bit of green, but little blue. (Red and green make yellow.) The little bit of blue means there is at least 0.2 worth of all three colors. Equal amounts of all colors is like white, so the reflected light isn't fully saturated. If one of the colors was zero, it would be. The colors are all less than 1, so it is doesn't have the greatest possible value (brightness) either.

You can find the amount of light that can be reflected from a surface due to a single light by multiplying the RGB values for the light by the RGB values of the surface, component-wise. That is, multiply the coefficient of reflectivity for the red of the surface times the value for red in the light and this is the most red that can be reflected. Do the same for green, and for blue.

Given the above orange surface, and a white (but not full value) light (.5,.5,.5), the reflection of light from the surface will be:

$$(R, G, B)_{surface} = (R, G, B)_{light} * (R, G, B)_{reflectivity}$$

$$(0.4, 0.25, 0.1) = (0.5, 0.5, 0.5) * (0.8, 0.5, 0.2)$$

However, if the light itself is pure red (1,0,0), only red light can be reflected, and the green and blue are absorbed into the surface. The amount reflected will be

$$(0.8, 0.0, 0.0) = (1.0, 0.0, 0.0) * (0.8, 0.5, 0.2)$$

If there is more than one light, the contributions of each light on each surface that it strikes are calculated and summed.

However, just because light is reflected from the surface does not mean that it all reaches the camera. Therefore, the above equation only give the maximum amount that can reach the camera. How much of it does takes into account the orientation of the light rays, the surface, and the camera relative to each other. The amount of light that is reflected from a surface depends on whether it is oriented toward the light directly, or at an angle. The mathematics of shading takes this into account and there is a drop-off in the amount of light reflected when the surface is not perpendicular to the incoming light rays.

Remember that this shading model is a very simplistic method compared to the behavior of light and surfaces in the real world. However, it can be calculated quite fast and produces reasonably good images.

1.3 Ambient, Diffuse, and Specular Light

More specifically, the reflectivity model most often used in computer graphics pretends that the incoming light is reflected from a surface partly as *ambient*, partly as *diffuse*, and partly as *specular* light. There is some variation between modeling programs concerning how to handle the ambient light in this model. Here we try to remain consistent

First, let's consider *ambient* light, which was defined above. If there is any ambient light in a scene, all surfaces will be affected by it. The reflection of ambient light is calculated using the component-wise multiplication already explained. Ambient light acts as if it hits the surface from all directions equally and reflects in all direction equally. Position and orientation have no effect.

Individual light sources may contribute *diffuse* light and/or *specular* light. In both cases, there is more light reflected off as the surface is oriented more nearly perpendicular to the light rays coming in.

Diffuse light acts as if the surface were *matte* (non-shiny). The light coming in reflects off in all directions. This simulates light hitting a surface that is very bumpy, and randomly bumpy, so the light hits the little nooks and crannies and bounces off in all directions. If only matte surfaces are simulated, this is called the *Lambert* shading model in graphics. To calculate the diffuse contribution, you need to know the color of the light and the reflectivity of the surface, the direction of the incoming light rays relative to the surface, and the orientation to the surface (the *normal* vector). The orientation of the surface is given by a *normal vector*. Each of these can be represented as a 3D vector (a triplet of real numbers).

Specular light models light from a very shiny surface. In this case, light reflects such that the angle of incidence is equal to the angle of reflection. This is more or less what a pool ball does when it hits the side of the table. The effect of specular light is that more light is seen if the eye is looking nearly down the reflected light vector. In this case, you see *highlights* (shiny spots). For surface such as metals, the highlights are usually the color of the surface; while in plastics, the highlights are white. To model specular light, you need to know the color of the light and surface reflectivity, the location of the light, the normal to the surface,

and also the location of the camera relative to the surface. These are all 3D vectors. There is also an extra *shininess* parameter used in specular lighting that models the size of the highlight. The shinier the surface, the smaller the highlight.

Surfaces that are between perfectly matte and shiny are modeled as a weighted combination of diffuse and specular light.

If there are several lights in a scene, each contribution is found separately and they are summed to find the final color.

1.4 Other Parameters

Surfaces may also be *transparent*. In this case, light coming from behind can get through. The total amount of light coming from the surface is then a weighted sum of the amount from behind and the amount from reflections off the surface itself. If a surface is not transparent it is *opaque*. This characteristic can be stored as a number from 0 to 1, where 0 is transparent and 1 is opaque and the numbers between are varying amounts of transparency.

When light goes through a transparent material it can be bent on leaving or entering. This is called *refraction*. (You see this when you look at an object under water while you are in the air.) The amount of refraction can be controlled.

Surfaces may be *emitters*. This means they emit light which is added in to the other contributions. For reasons of simplicity, though, the light from an emitting surface usually doesn't light other objects. If you want to see an emitter and have it affect other objects, you have to make it both a light-emitting object and place a light source in the same location.

Some shiny surfaces are very *reflective*. In this case, light that was reflected off of other objects hits the surface and reflects off toward your eye. This effect is normally only seen if you are using a careful renderer such as a *ray-tracer*. Lightwave 3D lets you choose whether to use reflections in the render menu.

Surfaces can have other characteristics, like bumpiness, and texture can be associated with them. More later.

2 Rendering Methods

Most graphical models are rendered as polygons. Even complex curved surfaces are usually subdivided into polygonal facets for drawing. You can usually tell if this is the case by looking at the *silhouette edges* of the object. If an object that you would expect to be curved looked like it has some straight edges along the boundary, it has been rendered as polygons. If the polygons are smaller than a pixel, of course, you won't see this effect.

There are various ways to render (light and shade) polygons. The simplest is called *flat shading (faceted shading)*. In this case, the shading model is done once for the whole polygon face, so the whole face is just one color. This creates objects that appear to be made of planar faces or facets. It is appropriate for objects like boxes, but if you use flat shading on something like a sphere, it will look more like a geodesic dome. However, it is very quick, because you only have to calculate the color once per polygon.

The next more complex rendering method is to use the shading model on each of the vertices of polygons. The color is calculated for the vertices using a normal vector that is the average of the normal vectors of the neighboring polygons. The color of each vertex is interpolated across the face of the polygon. This is called *Gouraud shading (smooth shading)*. This produces a smooth change in color across a tessellated sphere. (A tessellated sphere is one where the surface is divided into polygons.)

A slightly more expensive method calculates the shading at each pixel that the polygon *projects* onto, using a normal vector there that is an interpolating of the normal vector at the polygon vertices. This is usually called *Phong shading*). It looks somewhat more realistic than Gouraud shading, and can clearly create small highlights that indicate a shiny surface. For this reason, the text calls this *specular shading*.

Often renderers can simulate *fog* by gradually blending the calculated color of surfaces with a fog color, such that fog gradually becomes dominant with distance from the camera.

2.1 Hidden Surface Removal

Calculating the color of objects is often combined with determining which surfaces are in front relative to the camera. This is called *hidden surface removal*. For fairly fast, moderate quality rendering, a *scan-conversion* rendering method is used, in combination with *Z-buffering*.

In scan conversion, each polygon is examined one by one, and by mapping the polygon from world space to screen (pixel) space, the pixels covered by this polygon are determined. The color for each of these pixels is found one by one. A *Z-buffer* is like a duplicate frame buffer where, instead of storing color, a value is stored (the *Z-value*) that represents a distance from your eye to whatever polygon was last drawn at that pixel. If a new polygon is closer to the last one drawn there, it replaces it, both in the frame buffer with color, and in the *Z-buffer* with distance. Much of this is done “in hardware”, meaning the machine was built to do this very fast, with specialized components.

Polygon scan conversion combined with *Z-buffering* is the most commonly used rendering method. It can be quite fast and produce shaded images in *realtime*, meaning fast enough to create the illusion of animation if the camera or objects are moving. However, it is hard or impossible to do very realistic effects, such as transparency and inter-object reflections and refraction and shadows. Accurate transparency and refraction require that the polygons be drawn in order of their distance from the camera, and sorting them is expensive and generally not done using *Z-buffers*. Shadows and reflections requires searching all the objects in the scene to determine their relationship to either other and to the lights and camera.

2.2 Ray Tracing, Ray Casting, and Radiosity

Ray tracing is a more expensive rendering method that calculates the shading model for each pixel an object projects onto, but does even more than that. It can be envisioned as passing a ray from your eye through each pixel and finding which objects this ray intersects. If reflections aren't calculated, the usual shading model is calculated and you are done. This is usually called *ray-casting*, when no further secondary rays are propagated. Ray tracing can be done fairly easily with non-polygonal objects, especially spheres, because it is mathematically simple to calculate the intersection of a sphere and a ray. This is why you see so many images of shiny and transparent spheres in computer graphics.

If reflections are being calculated, the ray that strikes the object reflects off the surface of the object in the *specular* direction, and the intersections of the reflected ray with other objects are calculated. The shading model for these object-ray intersections is found, and the color from this shading model used to find the reflection of that object in the original object. These rays can continue to bounce off multiple objects, and the net result is the image of objects in other objects. This is, as you might expect, rather expensive to calculate. If these secondary rays are followed, the renderer is a *ray-tracer*. These secondary rays only contribute to the *specular* part of the color calculated.

Also, if the object is *transparent*, the ray may continue through the object and contributions from other objects can be found. The ray passing through an object may bend simulating *refraction*.

If you want *shadows*, it is possible to check if there are objects between the light source and the object being lit. This is usually not done with other renderers.

Radiosity is a method for finding reflections between diffuse objects (*color bleeding*). For example, if a white room has a very red sofa in it, the wall near the sofa will be pink. This is because light is diffusely reflected off the sofa to the wall. Radiosity models this, but ray-tracing doesn't. On the other hand, radiosity doesn't model specular light. In combination, very realistic images can be done, but they take many minutes or hours.

3 Texture Mapping and Other Mappings

A good way to get the appearance of complex surfaces without paying the computational cost is to use a mapping. The most common is called *texture mapping* or *image mapping*. In the simplest case, a two-dimensional digital image is associated (“mapped”) onto any given surface, so that it appears to be part of the surface. For example, a brick wall can be mapped onto a single polygon, and the polygon then appears to be a brick wall. This is done by associating the corners of the polygon with positions in the texture and taking the color of the polygon at any point within it as the color of the texture at that associated point. Any 2D image can be used as a texture map. The size of the texture relative to the object can be controlled. Texture mapping can be difficult if one maps to a non-planar surface, such as a sphere, or to a complex surface like the face. The texture map completely replace the reflectivity values of the surface or be combined with them in various ways.

In *environment mapping*, a complex environment around a scene is mapped onto objects in a scene. This is a less expensive way to create the appearance of reflections in objects. For example, the image of a skyline could be mapped onto a shiny glass office building. Often environment maps completely surround the 3D scene being rendering, either as an enclosing sphere, or as a box with six sides.

Bump mapping simulates a bumpy surface. This is done by *perturbing* (tilting, in this case) the normal vector for an object as you render it. This makes what should be a flat surface appear bumpy, because the amount of light reflected from a surface changes depending on the normal vector. For example, to create a randomly bumpy surface, the real orientation is randomly altered just a little for each pixel the polygon projects onto. This way, the light affects each pixel slightly differently, and the surface looks bumpy. This can also be done using a *bump map* texture, which is a 2D map that stores information about of change in orientation of the surface for each position. Vertices of a polygon are associated with positions in this grid, and the positions in the grid are used to give surface orientation at each position. This way, regular bumpy patterns, like waves or wood grain, can be made. If you look at a silhouette edge, however, you will see there really aren't any bumps at all; rather it is just an illusion.

Displacement mapping is a similar but more realistic (and expensive) method. Instead of just changing the orientation of the surface for shading, in order to make bumps, the actual surface is slightly moved because shading is done. In this case, the bumps are “really” there and even silhouette edges appear non-flat.

A *solid texture* or *3D texture* is one that is represented as values in 3D dimensions, not just 2. It can be a series of 2D digital images that represent slices in depth, or a procedure that gives values at any point in 3 dimensions. Solid textures are good for creating things that look like they were carved out of a material, such as wood or stone, rather than having a surface wrapped around them.

Procedural textures never create an actual 2D image, but call a special *procedure* (a module of a computer graphics program that does something) that returns a color given the position on the polygon. This is good for simple repetitive patterns such as grids and tablecloths, and also for some very random *noisy* patterns like granite.