Shading

Brian Curless
CSE 557
Fall 2013

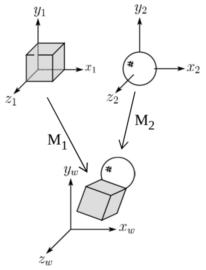
Reading

Required:

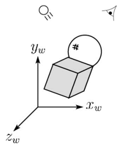
- Shirley, Chapter 10

Basic 3D graphics

With affine matrices, we can now transform virtual 3D objects in their local coordinate systems into a global (world) coordinate system:

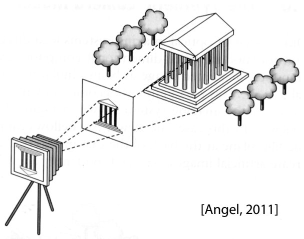


To synthesize an image of the scene, we also need to add light sources and a viewer/camera:



Pinhole camera

To create an image of a virtual scene, we need to define a camera, and we need to model lighting and shading. For the camera, we use a **pinhole camera**.



[Angel, 2011]

The image is rendered onto an **image plane** (usually in front of the camera).

Viewing rays emanate from the **center of projection** (COP) at the center of the pinhole.

The image of an object point **P** is at the intersection of the viewing ray through **P** and the image plane.

Shading

Next, we'll need a model to describe how light interacts with surfaces.

Such a model is called a **shading model**.

Other names:

- Lighting model
- Light reflection model
- Local illumination model
- Reflectance model
- BRDF

An abundance of photons

Given the camera and shading model, properly determining the right color at each pixel is *extremely hard.*

Look around the room. Each light source has different characteristics. Trillions of photons are pouring out every second.

These photons can:

- interact with molecules and particles in the air ("participating media")
- strike a surface and
  - be absorbed
  - be reflected (scattered)
  - cause fluorescence or phosphorescence.
- interact in a wavelength-dependent manner
- generally bounce around and around

Our problem

We're going to build up to a *approximations* of reality called the **Phong and Blinn-Phong illumination models**.

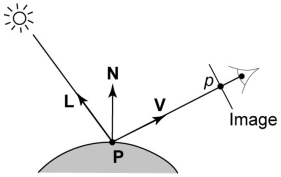
They have the following characteristics:

- *not* physically correct
- gives a "first-order" *approximation* to physical light reflection
- very fast
- widely used

In addition, we will assume **local illumination**, i.e., light goes: light source -> surface -> viewer.

No interreflections, no shadows.

Setup…



Given:

- a point **P** on a surface visible through pixel *p*
- The normal **N** at **P**
- The lighting direction, **L**, and (color) intensity, I_L, at **P**
- The viewing direction, **V**, at **P**
- The shading coefficients at **P**

Compute the color, *I*, of pixel *p*.

Assume that the direction vectors are normalized:

$$\|\mathbf{N}\| = \|\mathbf{L}\| = \|\mathbf{V}\| = 1$$

"Iteration zero"

The simplest thing you can do is…

Assign each polygon a single color:

$$I = k_e$$

where

- I is the resulting intensity
- k_e is the **emissivity** or intrinsic shade associated with the object

This has some special-purpose uses, but not really good for drawing a scene.

[Note: k_e is omitted in Shirley.]

"Iteration one"

Let's make the color at least dependent on the overall quantity of light available in the scene:

$$I = k_e + k_a I_{La}$$

- k_a is the **ambient reflection coefficient.**
  - really the reflectance of ambient light
  - "ambient" light is assumed to be equal in all directions
- I_{La} is the **ambient light intensity**.

Physically, what is "ambient" light?

[Note: Shirley uses c_a instead of I_{La}.]

Wavelength dependence

Really, k_e, k_a, and I_{La} are functions over all wavelengths λ.

Ideally, we would do the calculation on these functions. For the ambient shading equation, we would start with:

$$I(\lambda) = k_a(\lambda) I_{La}(\lambda)$$

then we would find good RGB values to represent the spectrum $I(\lambda)$.

Traditionally, though, k_a and I_{La} are represented as RGB triples, and the computation is performed on each color channel separately:

$$I^R = k_a^R \, I_{La}^R$$
$$I^G = k_a^G \, I_{La}^G$$
$$I^B = k_a^B \, I_{La}^B$$

Diffuse reflection

Let's examine the ambient shading model:

- objects have different colors
- we can control the overall light intensity
  - what happens when we turn off the lights?
  - what happens as the light intensity increases?
  - what happens if we change the color of the lights?

So far, objects are uniformly lit.

- not the way things really appear
- in reality, light sources are localized in position or direction

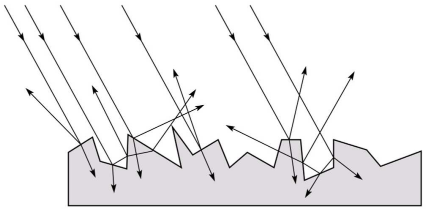
Diffuse, or **Lambertian** reflection will allow reflected intensity to vary with the direction of the light.

Diffuse reflectors

Diffuse reflection occurs from dull, matte surfaces, like latex paint, or chalk.

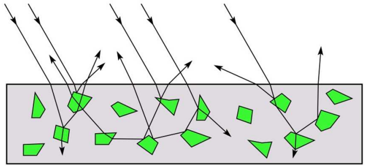
These **diffuse** or **Lambertian** reflectors reradiate light equally in all directions.

Picture a rough surface with lots of tiny **microfacets**.



Diffuse reflectors

…or picture a surface with little pigment particles embedded beneath the surface (neglect reflection at the surface for the moment):



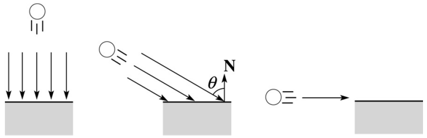
The microfacets and pigments distribute light rays in all directions.

Embedded pigments are responsible for the coloration of diffusely reflected light in plastics and paints.

Note: the figures above are intuitive, but not strictly (physically) correct.

Diffuse reflectors, cont.

The reflected intensity from a diffuse surface does not depend on the direction of the viewer. The incoming light, though, does depend on the direction of the light source:



"Iteration two"

The incoming energy is proportional to ______, giving the diffuse reflection equations:

$$I = k_e + k_a I_{La} + k_d I_L B \underline{\qquad\quad}$$

$$= k_e + k_a I_{La} + k_d I_L B(\qquad)$$

where:

- k_d is the **diffuse reflection coefficient**
- I_L is the (color) intensity of the light source
- **N** is the normal to the surface (unit vector)
- **L** is the direction to the light source (unit vector)
- B prevents contribution of light from below the surface:

$$B = \begin{cases} 1 & \text{if } \mathbf{N} \cdot \mathbf{L} > \mathbf{0} \\ 0 & \text{if } \mathbf{N} \cdot \mathbf{L} \le \mathbf{0} \end{cases}$$

[Note: Shirley uses c_r and c_l instead of k_d and L.]

Specular reflection

Specular reflection accounts for the highlight that you see on some objects.

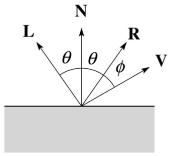
It is particularly important for *smooth, shiny* surfaces, such as:

- metal
- polished stone
- plastics
- apples
- skin

Properties:

- Specular reflection depends on the viewing direction **V**.
- For non-metals, the color is determined solely by the color of the light.
- For metals, the color may be altered (e.g., brass)

Specular reflection "derivation"



For a perfect mirror reflector, light is reflected about ***N***, so

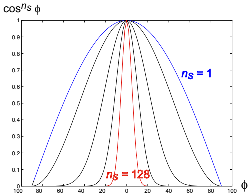
$$I = \begin{cases} I_L & \text{if } \mathbf{V} = \mathbf{R} \\ 0 & \text{otherwise} \end{cases}$$

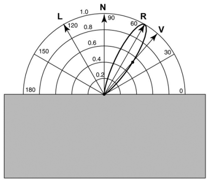
For a near-perfect reflector, you might expect the highlight to fall off quickly with increasing angle ϕ.

Also known as:

- **"rough specular" reflection**
- **"directional diffuse" reflection**
- **"glossy" reflection**

Phong specular reflection





One way to get this effect is to take (**R·V**), raised to a power n_s.

As n_s gets larger,

- the dropoff becomes {more,less} gradual
- gives a {larger,smaller} highlight
- simulates a {more,less} mirror-like surface

Phong specular reflection is proportional to:

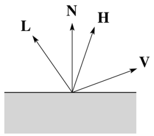
$$I_{\text{specular}} \sim B(\mathbf{R} \cdot \mathbf{V})_{+}^{n_s}$$

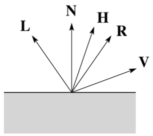
where $(x)_{+} \equiv \max(0, x)$.

Blinn-Phong specular reflection

A common alternative for specular reflection is the **Blinn-Phong model** (sometimes called the **modified Phong model**.)

We compute the vector halfway between **L** and **V** as:





Analogous to Phong specular reflection, we can compute the specular contribution in terms of (**N·H**), raised to a power n_s:

$$I_{\text{specular}} \sim B(\mathbf{N} \cdot \mathbf{H})_{+}^{n_s}$$

where, again, $(x)_{+} \equiv \max(0, x)$.

"Iteration three"

The next update to the Blinn-Phong shading model is then:

$$I = k_e + k_a I_{La} + k_d I_L B(\mathbf{N}\cdot\mathbf{L}) + k_s I_L B(\mathbf{N}\cdot\mathbf{H})^{n_s}$$

$$= k_e + k_a I_{La} + I_L B\left[k_d(\mathbf{N}\cdot\mathbf{L}) + k_s(\mathbf{N}\cdot\mathbf{H})^{n_s}\right]$$

where:

- k_s is the **specular reflection coefficient**
- n_s is the **specular exponent** or **shininess**
- **H** is the unit halfway vector between **L** and **V**, where **V** is the viewing direction.

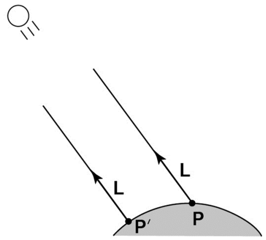
[Note: Shirley uses **e**, **r**, **h**, and p instead of **V**, **R**, **H**, and n_S.]

Directional lights

The simplest form of lights supported by renderers are ambient, directional, and point. Spotlights are also supported often as a special form of point light.

We've seen ambient light sources, which are not really geometric.

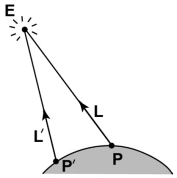
Directional light sources have a single direction and intensity associated with them.



Using affine notation, what is the homogeneous coordinate for a directional light?

Point lights

The direction of a **point light** sources is determined by the vector from the light position to the surface point.



Physics tells us the intensity must drop off inversely with the square of the distance:

$$f_{\text{atten}} = \frac{1}{r^2}$$

Sometimes, this distance-squared dropoff is considered too "harsh." A common alternative is:

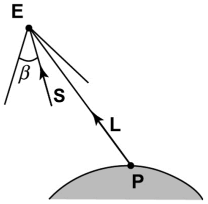
$$f_{\text{atten}} = \frac{1}{a + br + cr^2}$$

with user-supplied constants for a, b, and c.

Using affine notation, what is the homogeneous coordinate for a point light?

Spotlights

We can also apply a *directional attenuation* of a point light source, giving a **spotlight** effect.



A common choice for the spotlight intensity is:

$$f_{\text{spot}} = \begin{cases} \dfrac{(\mathbf{L}\cdot\mathbf{S})^e}{a + br + cr^2} & \text{if } \mathbf{L}\cdot\mathbf{S} \le \cos\beta \\ 0 & \text{otherwise} \end{cases}$$

where

- **L** is the direction to the point light.
- **S** is the center direction of the spotlight.
- β is the cutoff angle for the spotlight
- e is the angular falloff coefficient

"Iteration four"

Since light is additive, we can handle multiple lights by taking the sum over every light.

Our equation is now:

$$I = k_e + k_a I_{La} + \sum_j \frac{\left(\mathbf{L}_j \cdot \mathbf{S}_j\right)^{e_j}_{\beta_j}}{a_j + b_j r_j + c_j r_j^{\,2}} I_{L,j} B_j \left[k_d \left(\mathbf{N} \cdot \mathbf{L}_j\right) + k_s \left(\mathbf{N} \cdot \mathbf{H}_j\right)^{n_s} \right]$$

This is the Phong illumination model.

Which quantities are spatial vectors?

Which are RGB triples?

Which are scalars?

Choosing the parameters

Experiment with different parameter settings. To get you started, here are a few suggestions:

- Try n_s in the range [0,100]
- Try $k_a + k_d + k_s < 1$
- Use a small k_a (~0.1)

	n_s	k_d	k_s
Metal	large	Small, color of metal	Large, color of metal
Plastic	medium	Medium, color of plastic	Medium, white
Planet	0	varying	0

BRDF

The diffuse+specular parts of the Blinn-Phong illumination model are a mapping from light to viewing directions:

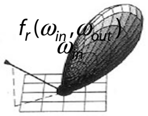
$$\begin{aligned} I &= I_L B \left[k_d (\mathbf{N} \cdot \mathbf{L}) + k_s \mathbf{N} \cdot \left(\frac{\mathbf{L} + \mathbf{V}}{\|\mathbf{L} + \mathbf{V}\|} \right)^{n_s} \right] \\ &= I_L \; f_r(\mathbf{L}, \mathbf{V}) \end{aligned}$$

The mapping function f_r is often written in terms of incoming (light) directions ω_{in} and outgoing (viewing) directions ω_{out}:

$$f_r(\omega_{in}, \omega_{out}) \quad \text{or} \quad f_r(\omega_{in} \to \omega_{out})$$

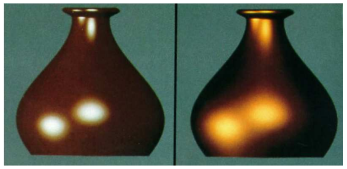
This function is called the **Bi-directional Reflectance Distribution Function** (**BRDF**).

Here's a plot with ω_{in} held constant:

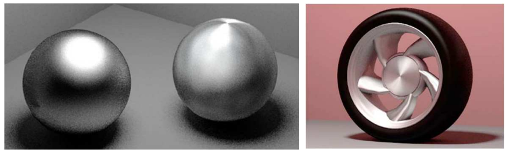


BRDF's can be quite sophisticated…

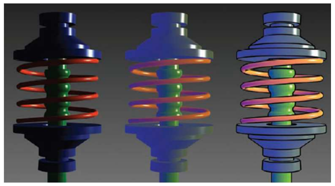
More sophisticated BRDF's



[Cook and Torrance, 1982]



Anisotropic BRDFs [Westin, Arvo, Torrance 1992]



Artistics BRDFs [Gooch]

Gouraud vs. Phong interpolation

Now we know how to compute the color at a point on a surface using the Blinn-Phong lighting model.

Does graphics hardware do this calculation at every point? Not by default...

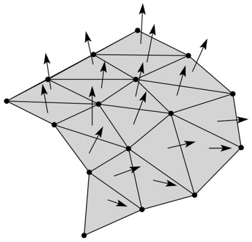
Smooth surfaces are often approximated by polygonal facets, because:

- Graphics hardware generally wants polygons (esp. triangles).
- Sometimes it easier to write ray-surface intersection algorithms for polygonal models.

How do we compute the shading for such a surface?

Faceted shading

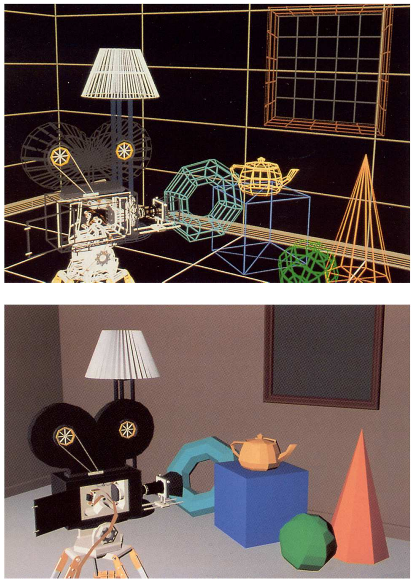
Assume each face has a constant normal:



For a distant viewer and a distant light source and constant material properties over the surface, how will the color of each triangle vary?

Result: faceted, not smooth, appearance.

Faceted shading (cont'd)



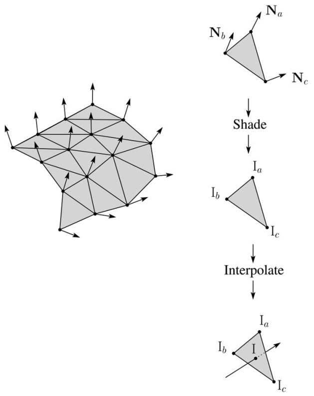
[Williams and Siegel 1990]

Gouraud interpolation

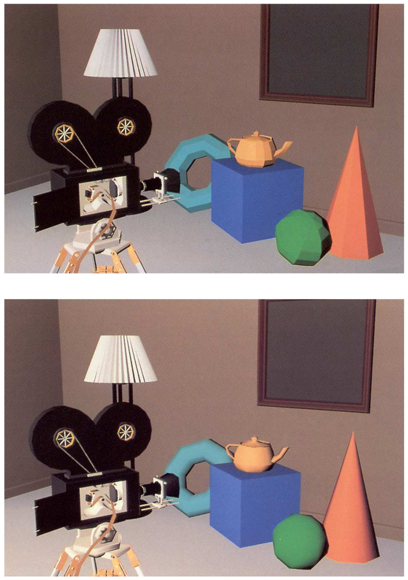
To get a smoother result that is easily performed in hardware, we can do **Gouraud interpolation**.

Here's how it works:

1. Compute normals at the vertices.
2. Shade only the vertices.
3. Interpolate the resulting vertex colors.



Facted shading vs. Gouraud interpolation

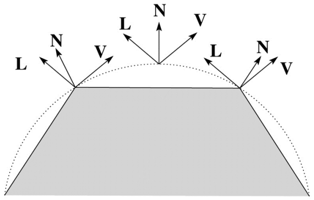


[Williams and Siegel 1990]

Gouraud interpolation artifacts

Gouraud interpolation has significant limitations.

1. If the polygonal approximation is too coarse, we can miss specular highlights.



2. We will encounter **Mach banding** (derivative discontinuity enhanced by human eye).

This is what graphics hardware does by default.

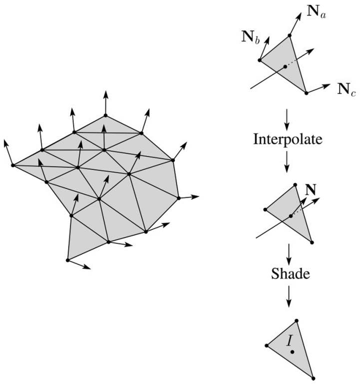
A substantial improvement is to do…

Phong interpolation

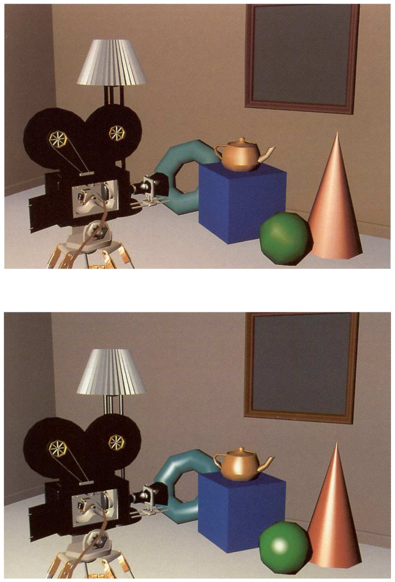
To get an even smoother result with fewer artifacts, we can perform **Phong *interpolation***.

Here's how it works:

1. Compute normals at the vertices.
2. Interpolate normals and normalize.
3. Shade using the interpolated normals.



Gouraud vs. Phong interpolation



[Williams and Siegel 1990]