Shading

Brian Curless

CSE 457

Spring 2013

Reading

Required:

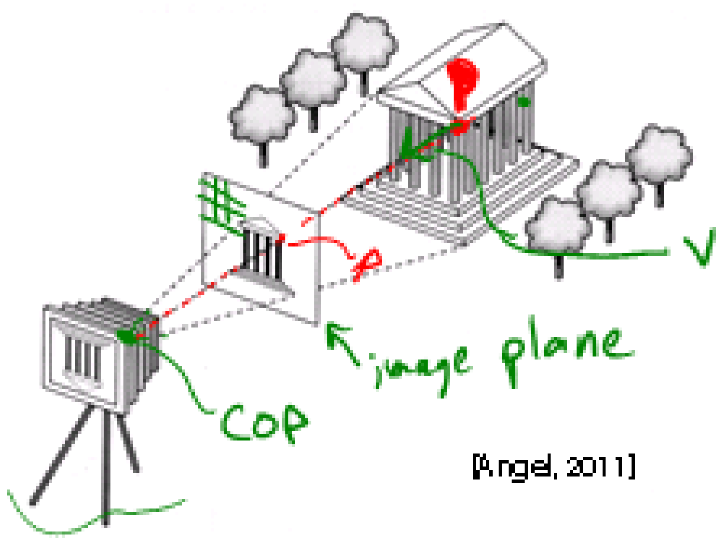
- Angel chapter 5.

Optional:

- OpenGL red book, chapter 5.

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.**



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.

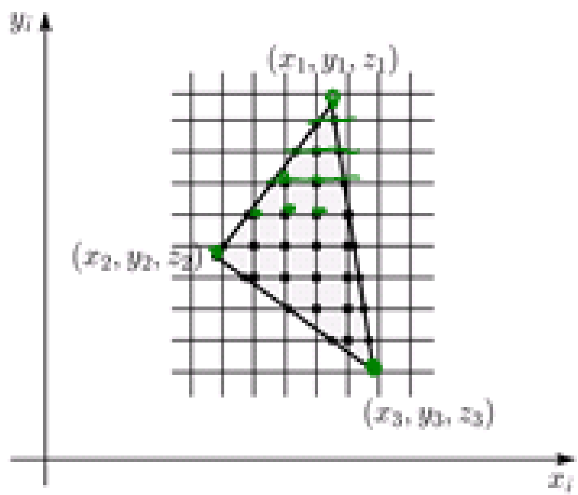
But is P visible? This the problem of **hidden surface removal** (a.k.a., **visible surface determination**).

Rasterization

Graphics hardware assumes the world is made of triangles.

It does hidden surface determination by projecting every triangle to the image plane, and "smearing" properties stored at each vertex across each projected triangle.

The process of filling in the pixels inside of a polygon is called **rasterization**.



Graphics hardware smears the z-values of the triangle vertices, and uses a Z-buffer to determine if a point inside a triangle is visible. More on this in another lecture...

Shading

So far, we've talked exclusively about geometry.

- What is the shape of an object?
- How do I place it in a virtual 3D space?
- How do I know which pixels it covers?
- How do I know which of the pixels I should actually draw?

Once we've answered all those, we have to ask one more important question:

- To what value do I set each pixel?

Answering this question is the job of the **shading model**.

Other names:

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

Our problem

Modeling the flow of light in a scene is very complex: photons pour out of light sources and bounce around and around before reaching a camera.

Here we focus on **local illumination**, i.e., what happens for a single bounce:

light source → surface → viewer

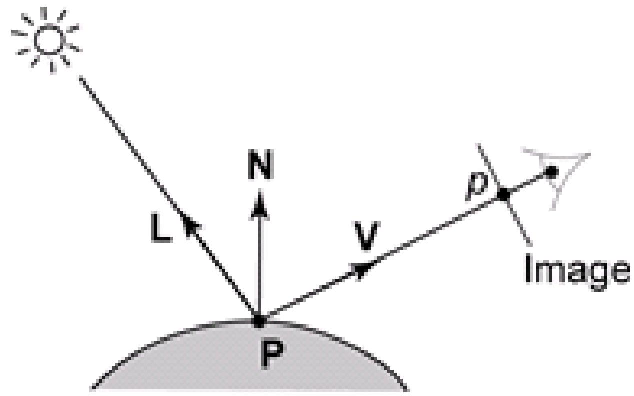
No interreflections, no shadows.

We're going to explore two models: the **Phong** and **Blinn-Phong illumination models**.

They have the following characteristics:

- physically plausible (albeit not strictly correct)
- very fast
- widely used

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 Angel.]

"Iteration one"

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

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

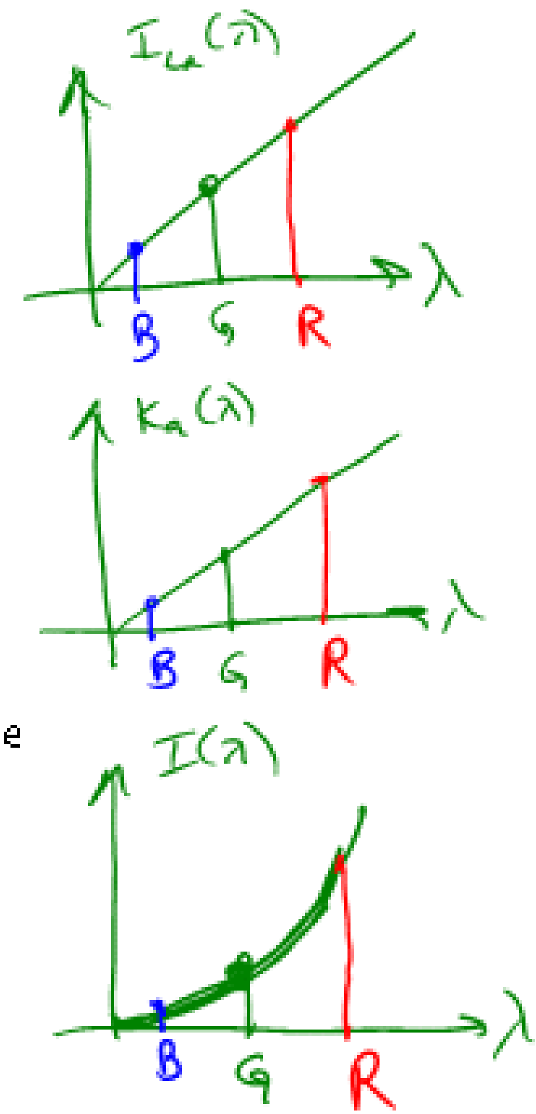
- k_a is the **ambient reflection coefficient.**
  - really the reflectance of ambient light
  - "ambient" light is assumed to be equal in all directions
- I_{La} is the **ambient light intensity**.

Physically, what is "ambient" light?

Poor man's interreflection

[Note: Angel uses L_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

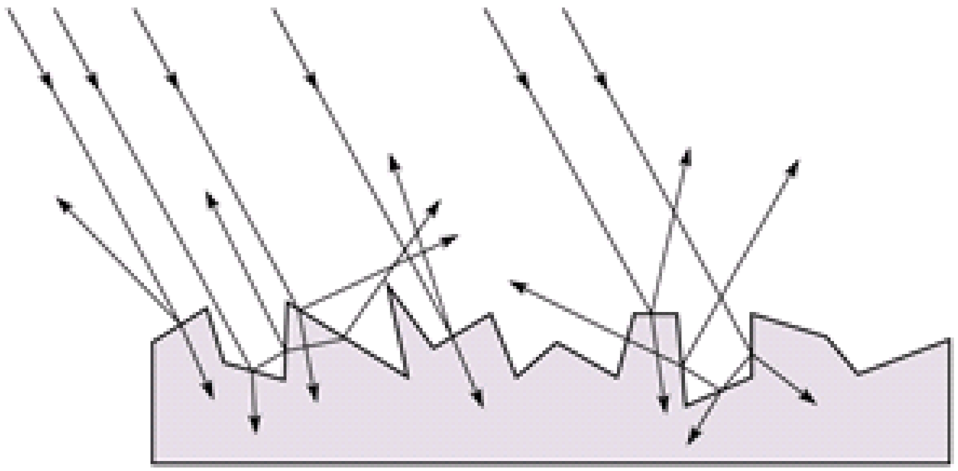
Diffuse, or **Lambertian** reflection will allow reflected intensity to vary with the direction of the light.

Diffuse reflectors

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

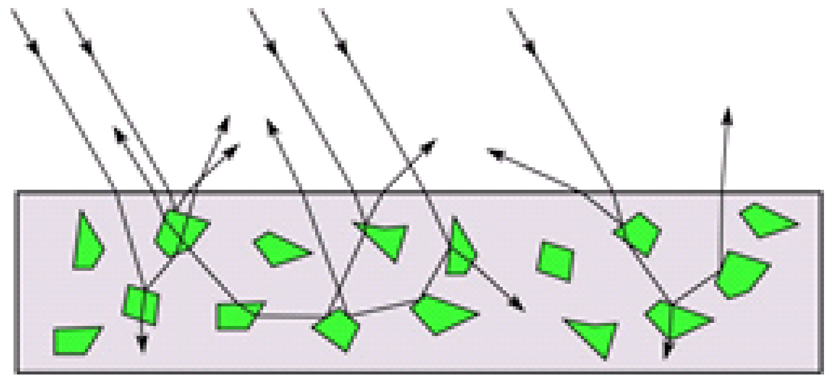
These **diffuse** or **Lambertian** reflectors reradiate light equally in all directions.

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):



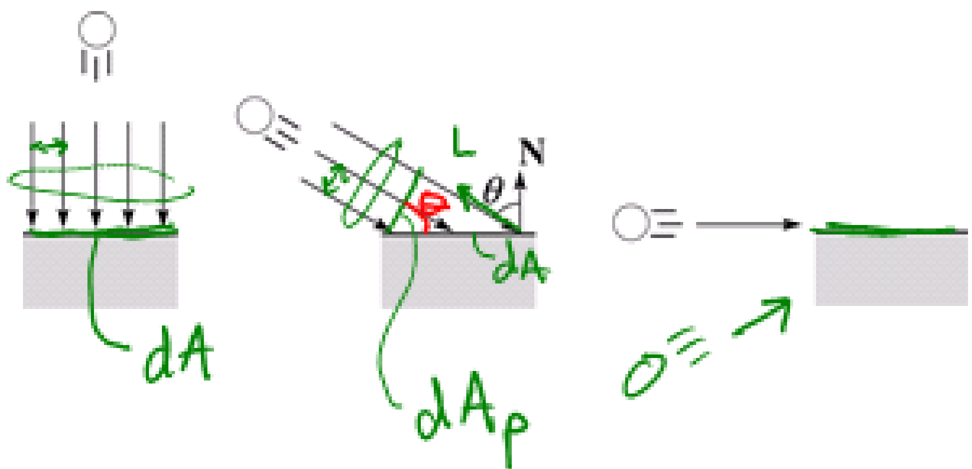
The microfacets and pigments distribute light rays in all directions.

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:



$$\cos\theta = \frac{dA_p}{dA}$$

$$dA_p = \cos\theta \, dA$$

$$\cos\theta = N \cdot L$$

$$(\text{if } \|N\| = \|L\| = 1)$$

"Iteration two"

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

$$I = k_e + k_a I_{La} + k_d I_L B \underline{\cos\theta}$$

$$= k_e + k_a I_{La} + k_d I_L B(N \cdot L)$$

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:

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

[Note: Angel uses L_d instead of I_L and f instead of B]

Specular reflection

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

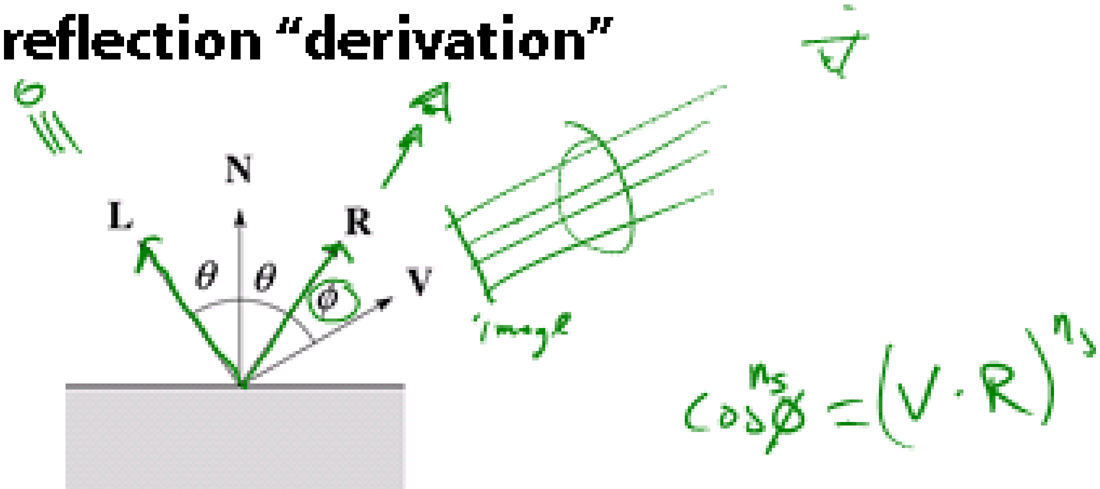
It is particularly important for *smooth, shiny* surfaces, such as:

- 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

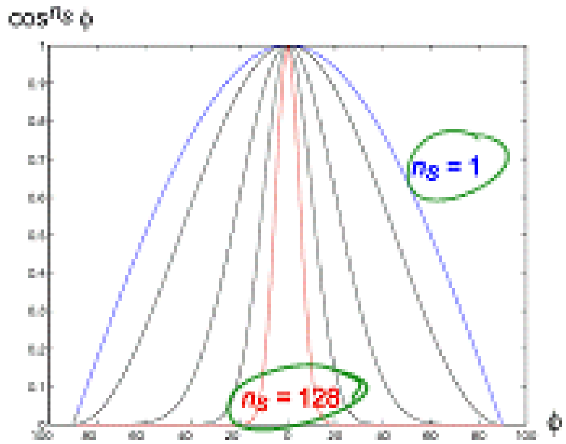
$$I = \begin{cases} I_L & \text{if } \mathbf{V} = \mathbf{R} \\ 0 & \text{otherwise} \end{cases}$$

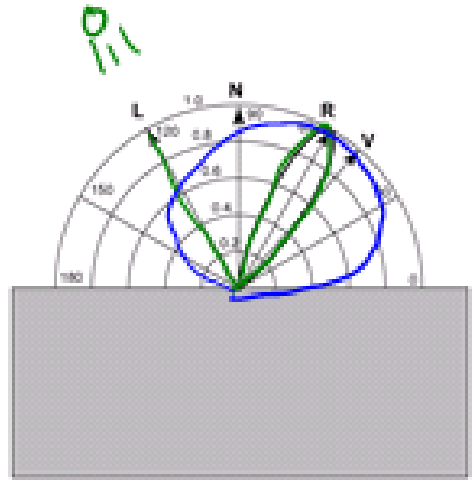
For a near-perfect reflector, you might expect the highlight to fall off quickly with increasing angle ϕ.

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:

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

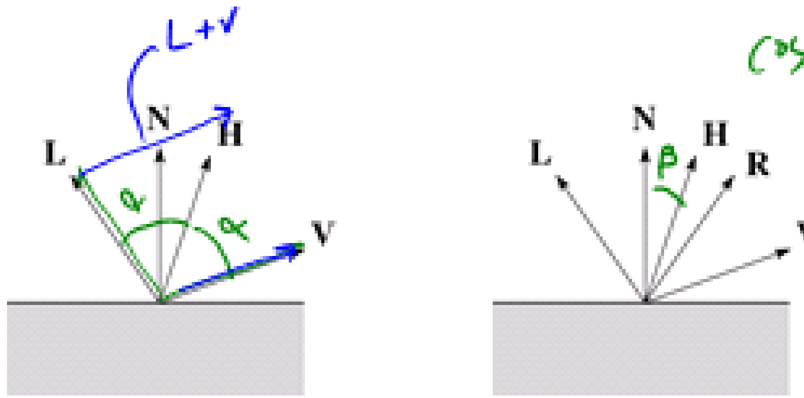
where $(x)_+ \equiv \max(0, x)$.

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:

$H = \frac{L+V}{\|L+V\|}$



$\cos\beta = N \cdot H$

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

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

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.

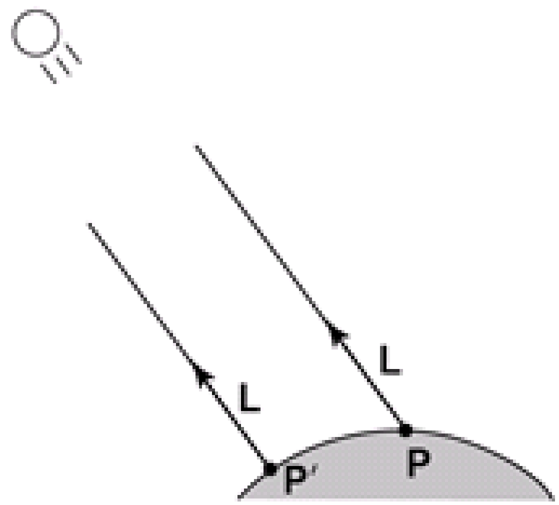
[Note: Angel uses α instead of n_s and maintains a separate L_d and L_s, instead of a single I_L. This choice reflects the flexibility available in OpenGL.]

Directional lights

OpenGL supports three different kinds of lights: ambient, directional, and point. Spot lights are also supported as a special form of point light.

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

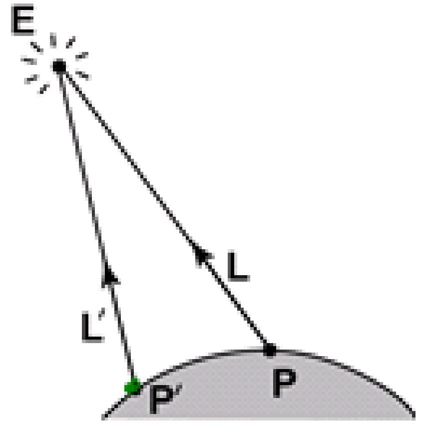
Directional light sources have a single direction and intensity associated with them.



Using affine notation, what is the homogeneous coordinate for a directional light? 0

Point lights

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



$$\mathbf{L} = \frac{\mathbf{E} - \mathbf{P}}{\|\mathbf{E} - \mathbf{P}\|}$$

$$r = \|\mathbf{E} - \mathbf{P}\|$$

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

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

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

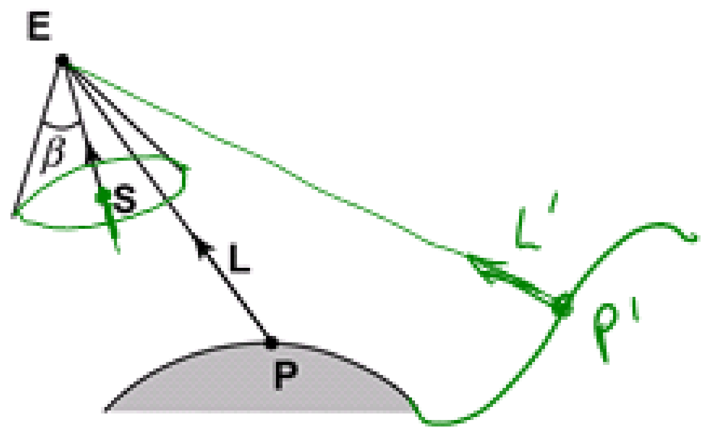
$$f_{atten} = \frac{1}{a + br + cr^2}$$

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

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

Spotlights

OpenGL also allows one to apply a *directional attenuation* of a point light source, giving a **spotlight** effect.



The spotlight intensity factor is computed in OpenGL as:

$$f_{spot} = (\mathbf{L} \cdot \mathbf{S})^{e}_{\beta}$$

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
- $(x)^{e}_{\beta} = \left[\max\{\operatorname{acos}(x) - \beta, 0\}\right]^{e}$

"Iteration four"

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

Our equation is now (for point lights):

$$I = k_e + k_a I_{La} + \sum_j \frac{1}{a_j + b_j r_j + c_j r_j^2} I_{L_j} B_j \left[k_d \left(\mathbf{N} \cdot \mathbf{L}_j \right) + k_s \left(\mathbf{N} \cdot \mathbf{H}_j \right)_+^{n_s} \right]$$

This is the Blinn-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

Materials in OpenGL

The OpenGL code to specify the surface shading properties is fairly straightforward. For example:

```
GLfloat ke[] = { 0.1, 0.15, 0.05, 1.0 };
GLfloat ka[] = { 0.1, 0.15, 0.1, 1.0 };
GLfloat kd[] = { 0.3, 0.3, 0.2, 1.0 };
GLfloat ks[] = { 0.2, 0.2, 0.2, 1.0 };
GLfloat ns[] = { 50.0 };
glMaterialfv(GL_FRONT, GL_EMISSION, ke);
glMaterialfv(GL_FRONT, GL_AMBIENT, ka);
glMaterialfv(GL_FRONT, GL_DIFFUSE, kd);
glMaterialfv(GL_FRONT, GL_SPECULAR, ks);
glMaterialfv(GL_FRONT, GL_SHININESS, ns);
```

Notes:

- The `GL_FRONT` parameter tells OpenGL that we are specifiying the materials for the front of the surface.
- Only the alpha value of the diffuse color is used for blending. It's usually set to 1.

Shading in OpenGL

The OpenGL lighting model allows you to associate different lighting colors according to material properties they will influence.

Thus, our original shading equation:

$$I = k_e + k_a I_{La} + \sum_j \frac{1}{a_j + b_j r_j + c_j r_j^2} I_{L,j} B_j \left[k_d (\mathbf{N} \cdot \mathbf{L}_j)_+ + k_s (\mathbf{N} \cdot \mathbf{H}_j)_+^{n_s} \right]$$

becomes:

$$I = k_e + k_a I_{La} + \sum_j \frac{1}{a_j + b_j r_j + c_j r_j^2} \left[k_a I_{La,j} + B_j \left\{ k_d I_{Ld,j} (\mathbf{N} \cdot \mathbf{L}_j)_+ + k_s I_{Ls,j} (\mathbf{N} \cdot \mathbf{H}_j)_+^{n_s} \right\} \right]$$

where you can have a global ambient light with intensity I_{La} in addition to having an ambient light intensity $I_{La,j}$ associated with each individual light, as well as separate diffuse and specular intensities, $I_{Ld,j}$ and $I_{Ls,j}$, repectively.

Shading in OpenGL, cont'd

In OpenGL this equation, for one light source (the 0th) is specified something like:

```
GLfloat La[] = { 0.2, 0.2, 0.2, 1.0 };

GLfloat La0[] = { 0.1, 0.1, 0.1, 1.0 };
GLfloat Ld0[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat Ls0[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat pos0[] = { 1.0, 1.0, 1.0, 0.0 };
GLfloat a0[] = { 1.0 };
GLfloat b0[] = { 0.5 };
GLfloat c0[] = { 0.25 };
GLfloat S0[] = { -1.0, -1.0, 0.0 };
GLfloat beta0[] = { 45 };
GLfloat e0[] = { 2 };

glLightModelfv(GL_LIGHT_MODEL_AMBIENT, La);
glLightfv(GL_LIGHT0, GL_AMBIENT, La0);
glLightfv(GL_LIGHT0, GL_DIFFUSE, Ld0);
glLightfv(GL_LIGHT0, GL_SPECULAR, Ls0);
glLightfv(GL_LIGHT0, GL_POSITION, pos0);
glLightfv(GL_LIGHT0, GL_CONSTANT_ATTENUATION, a0);
glLightfv(GL_LIGHT0, GL_LINEAR_ATTENUATION, b0);
glLightfv(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, c0);
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, S0);
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, beta0);
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, e0);
```

Shading in OpenGL, cont'd

Notes:

You can have as many as GL_MAX_LIGHTS lights in a scene. This number is system-dependent.

For directional lights, you specify a light direction, not position, and the attenuation and spotlight terms are ignored.

The directions of directional lights and spotlights are specified in the coordinate systems *of the lights*, not the surface points as we've been doing in lecture.

BRDF

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

$$I = I_L B\left[k_d(\mathbf{N}\cdot\mathbf{L}) + k_s\left(\mathbf{N}\cdot\frac{\mathbf{L}+\mathbf{V}}{\|\mathbf{L}+\mathbf{V}\|}\right)_+^{n_s}\right]$$

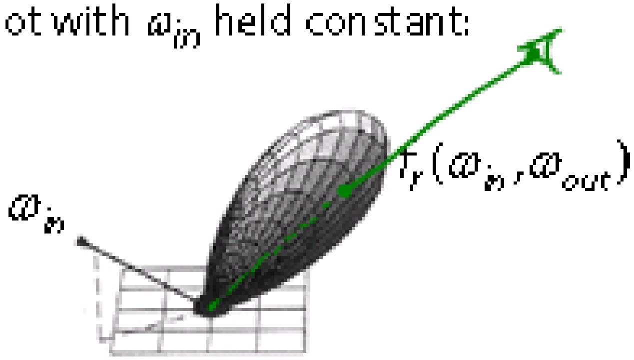
$$= I_L \; f_r(\mathbf{L},\mathbf{V})$$

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

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

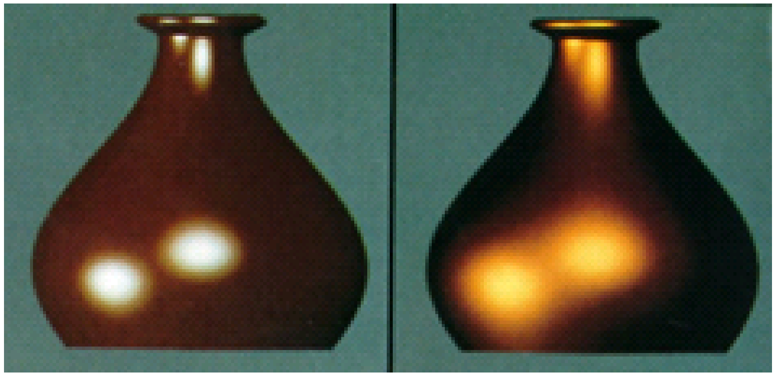
This function is called the **Bi-directional Reflectance Distribution Function** (**BRDF**).

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

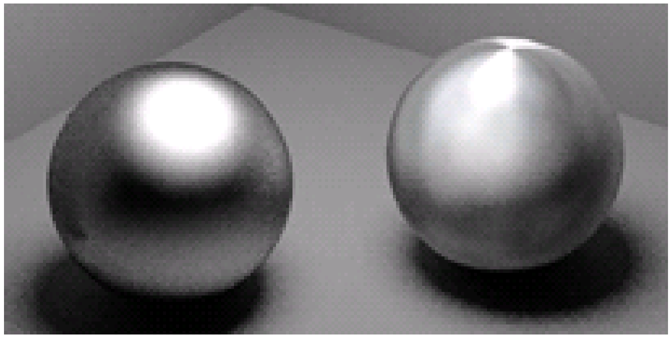


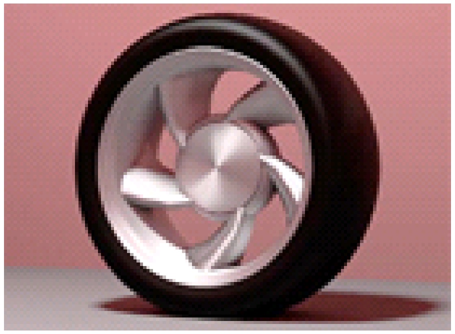
BRDF's can be quite sophisticated…

More sophisticated BRDF's

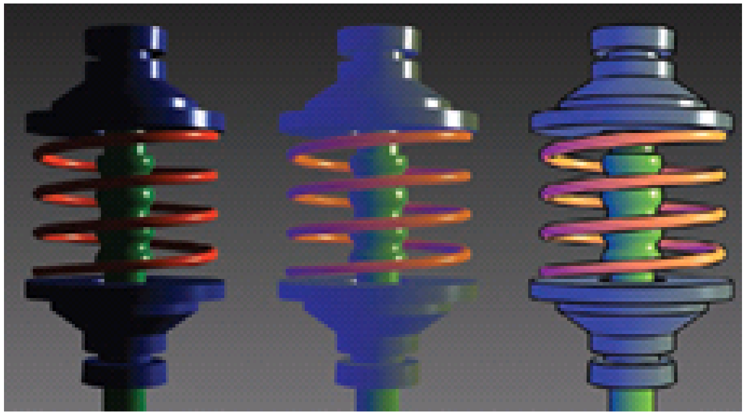


[Cook and Torrance, 1982]





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:

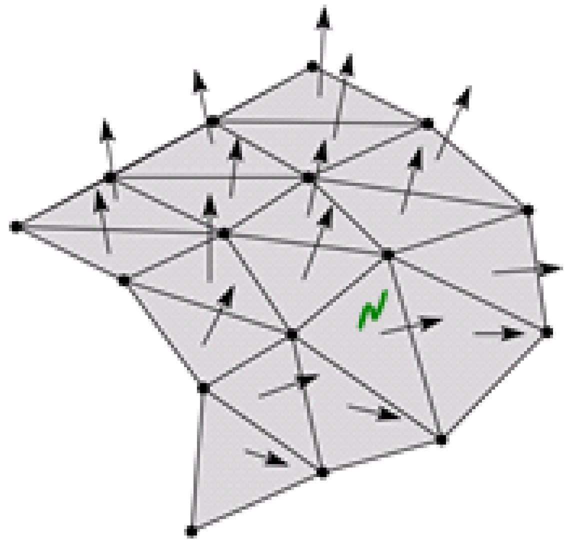
- Graphics hardware generally wants polygons (esp. triangles).
- Sometimes it easier to write ray-surface intersection algorithms for polygonal models.

How do we compute the shading for such a surface?

Faceted shading

Assume each face has a constant normal:

L V



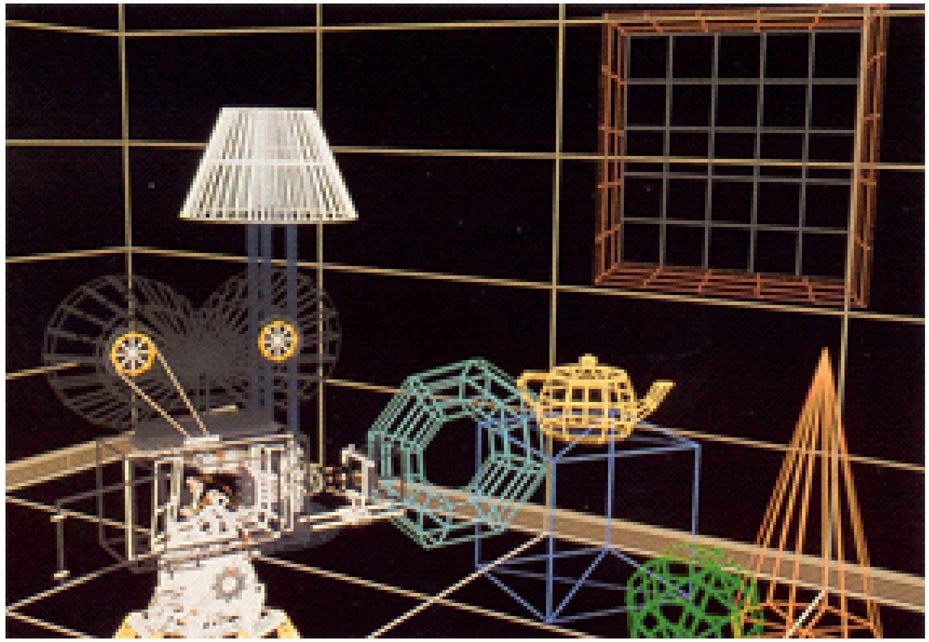
For a distant viewer and a distant directional light source and constant material properties over the surface, how will the color of each triangle vary?

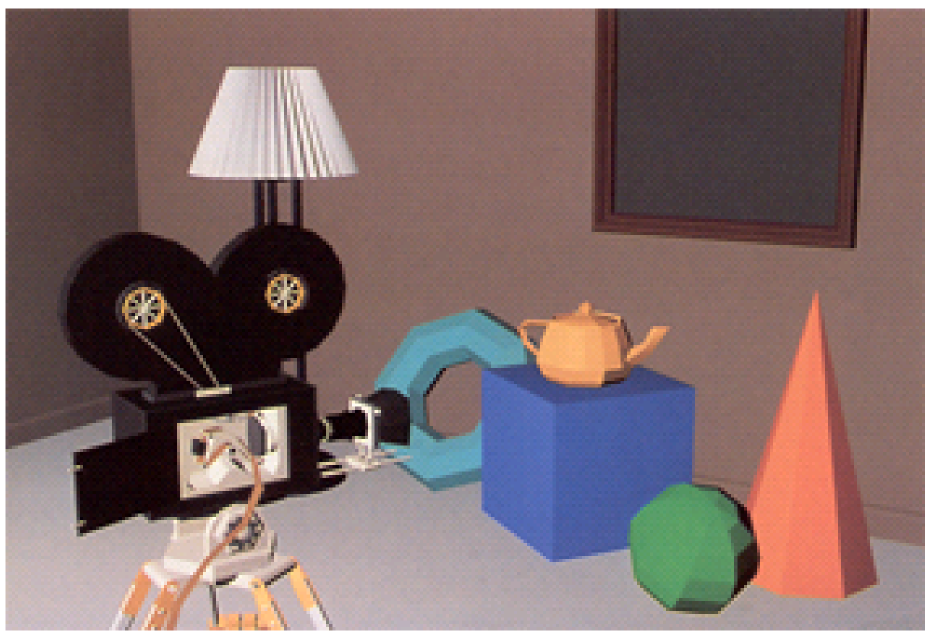
Result: faceted, not smooth, appearance.

$I = \ldots (N \cdot L) \ldots$

$\left(N \cdot \frac{L+V}{\|L+V\|}\right)^{n_s} \ldots$

Faceted shading (cont'd)





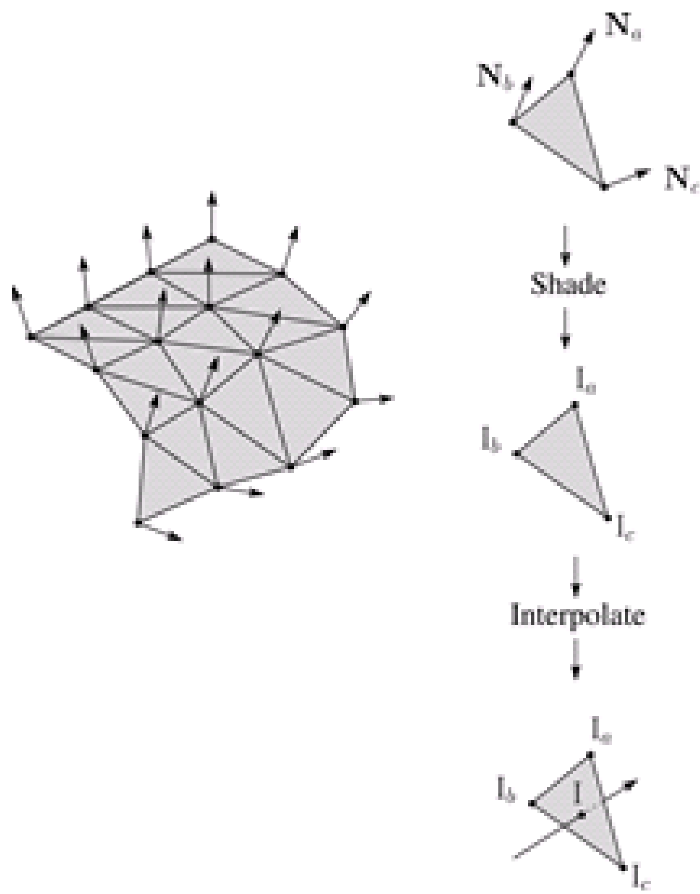
[Williams and Siegel 1990]

Gouraud interpolation

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

Here's how it works:

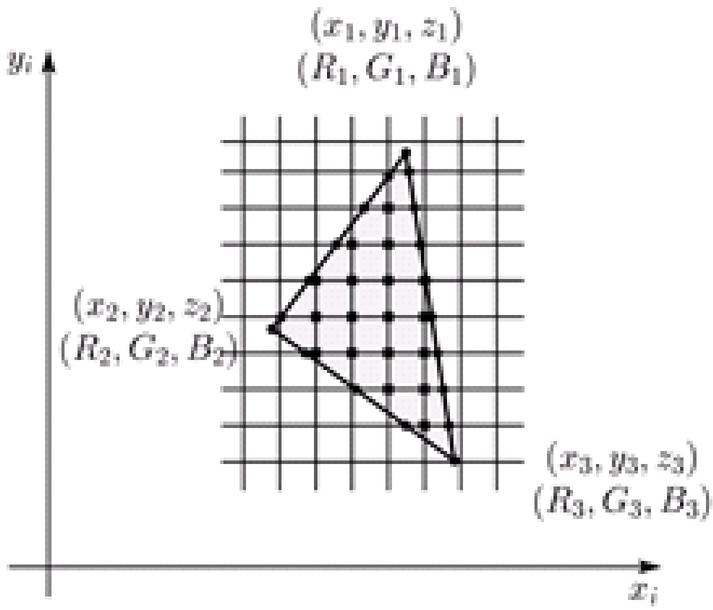
1. Compute normals at the vertices.
2. Shade only the vertices.
3. Interpolate the resulting vertex colors.



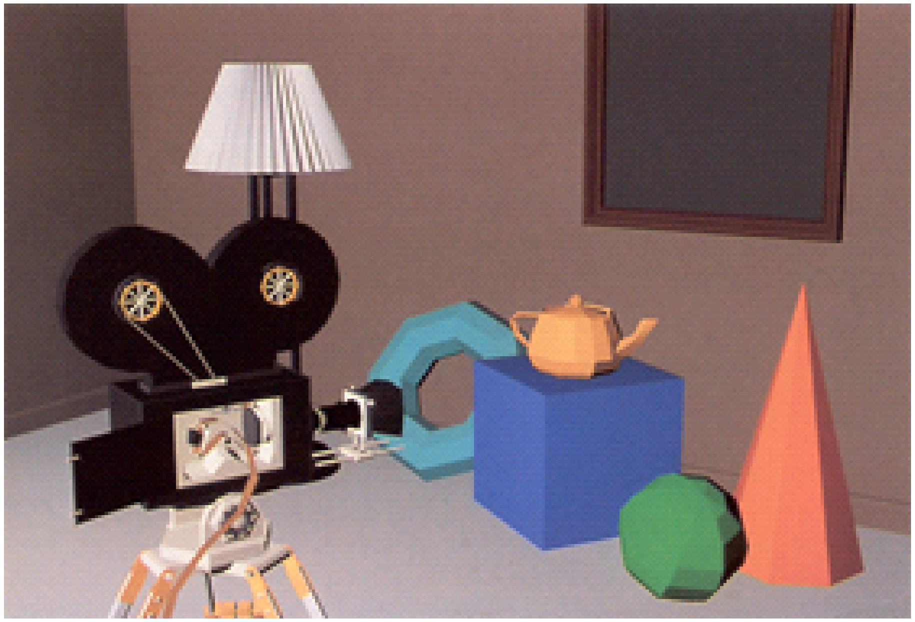
Rasterization with color

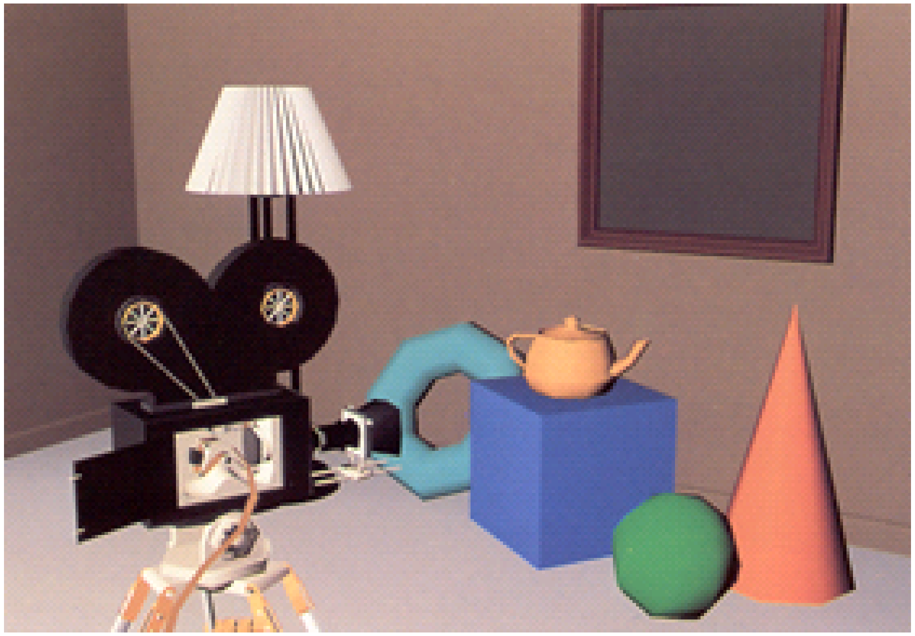
Recall that the z-buffer works by interpolating z-values across a triangle that has been projected into image space, a process called rasterization.

During rasterization, colors can be smeared across a triangle as well:



Facted shading vs. Gouraud interpolation



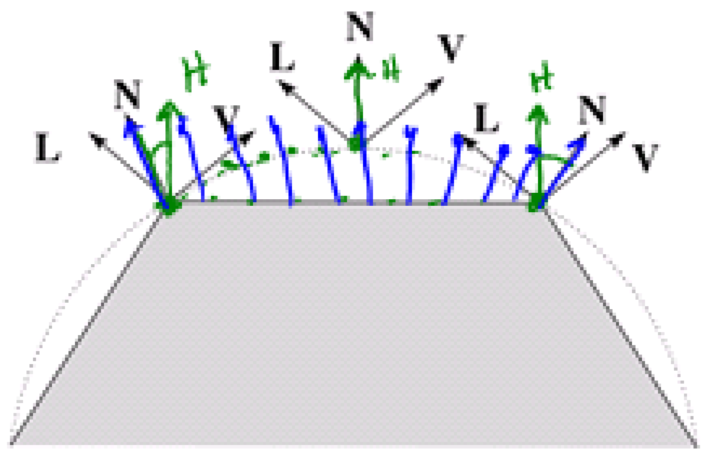


[Williams and Siegel 1990]

Gouraud interpolation artifacts

Gouraud interpolation has significant limitations.

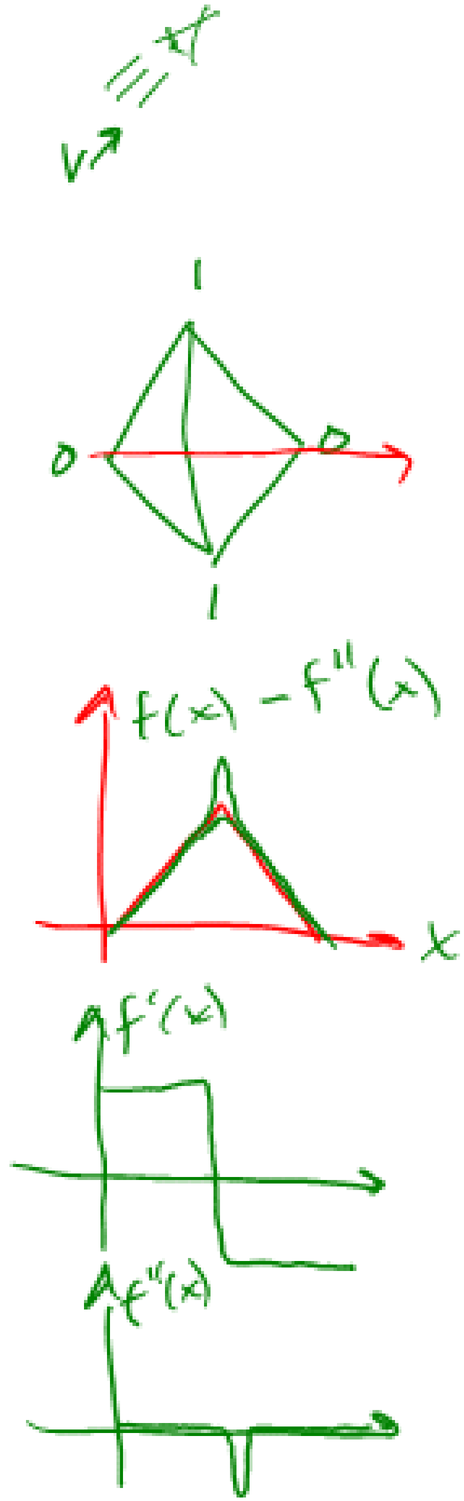
1. If the polygonal approximation is too coarse, we can miss specular highlights.



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…

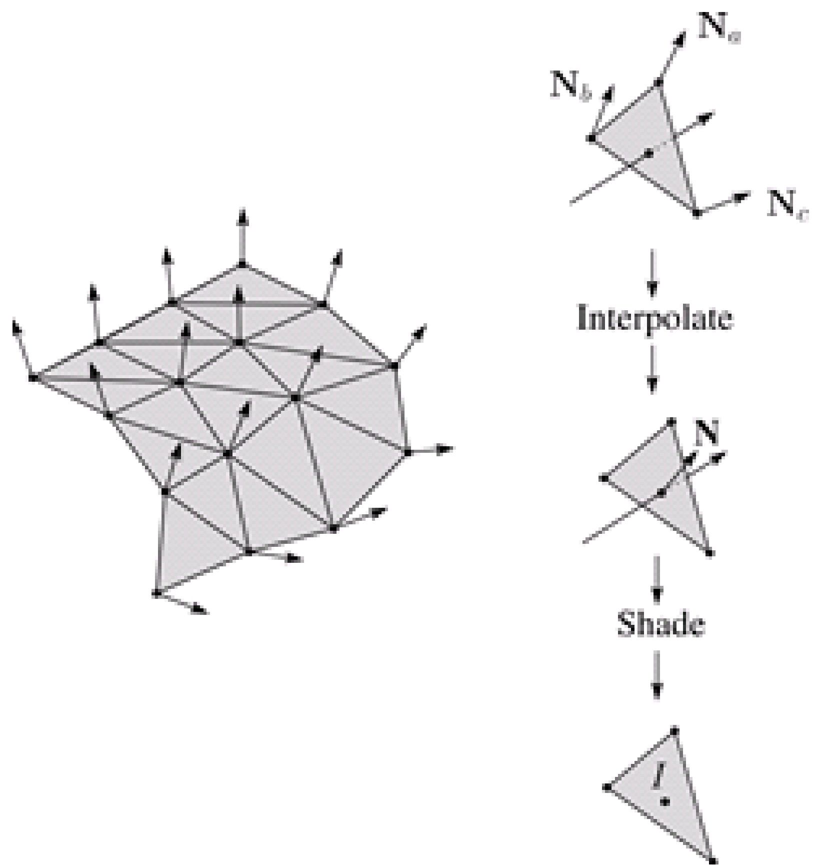


Phong interpolation

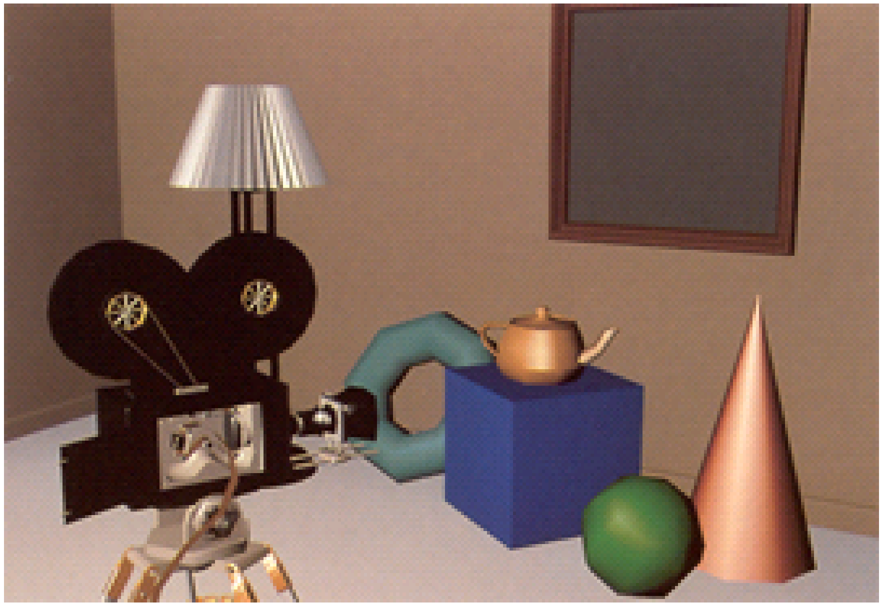
To get an even smoother result with fewer artifacts, we can perform **Phong *interpolation***.

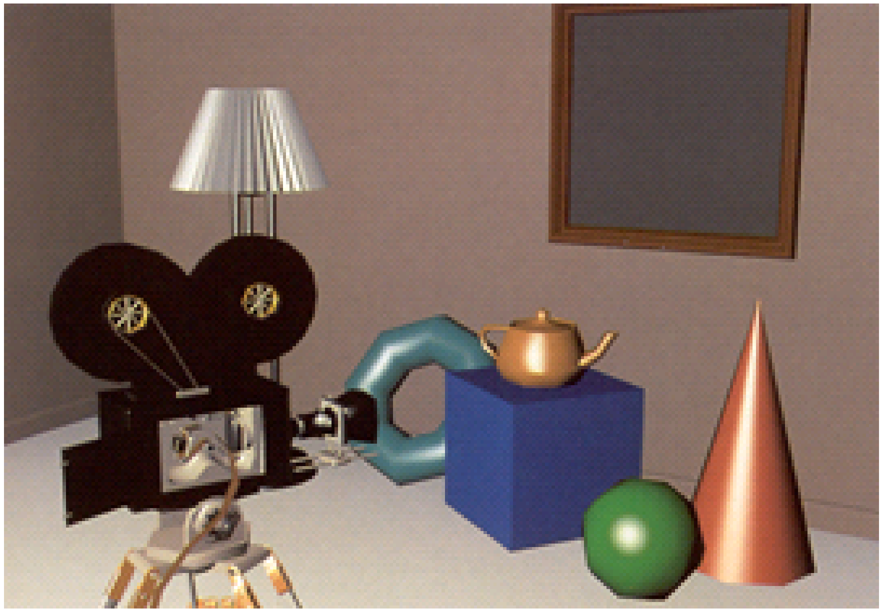
Here's how it works:

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



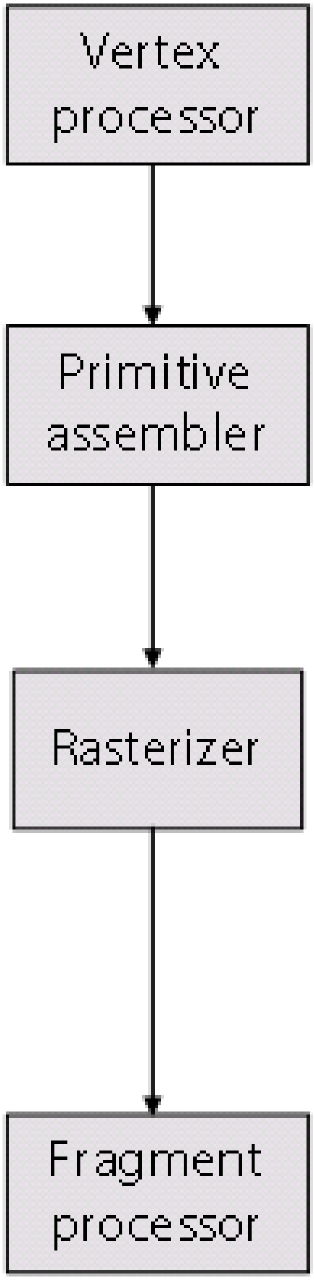
Gouraud vs. Phong interpolation





[Williams and Siegel 1990]

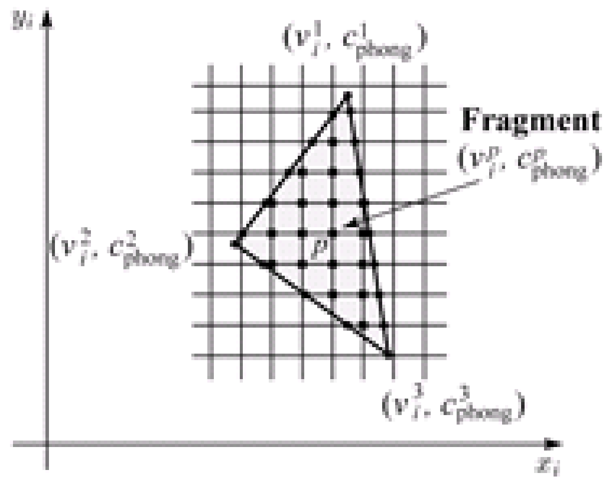
Default pipeline: Gouraud interpolation



Default vertex processing:

$L \leftarrow$ determine lighting direction
$V \leftarrow$ determine viewing direction
$N \leftarrow$ normalize(n_v)
$c_{\text{phong}} \leftarrow$ shade with L, V, N, k_d, k_s, n_s
attach c_{phong} to vertex as "varying"
$v_i \leftarrow$ project v to image

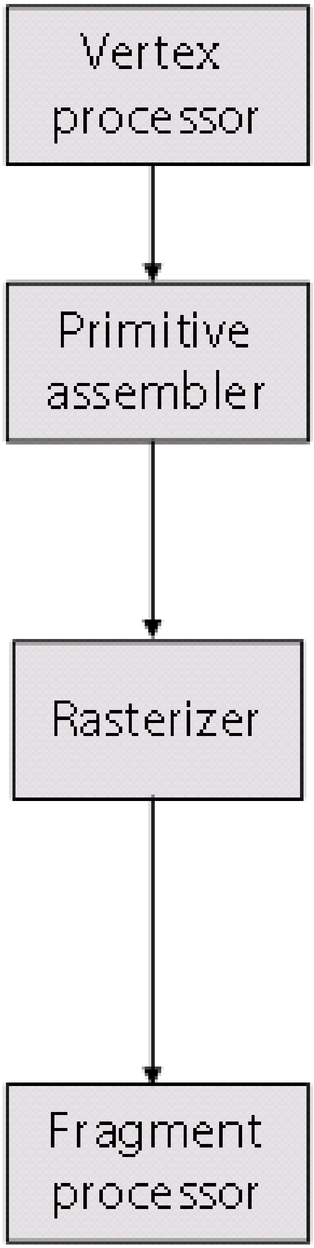
$v_i^1, v_i^2, v_i^3 \rightarrow$ triangle



Default fragment processing:

color $\leftarrow c_{\text{phong}}^p$

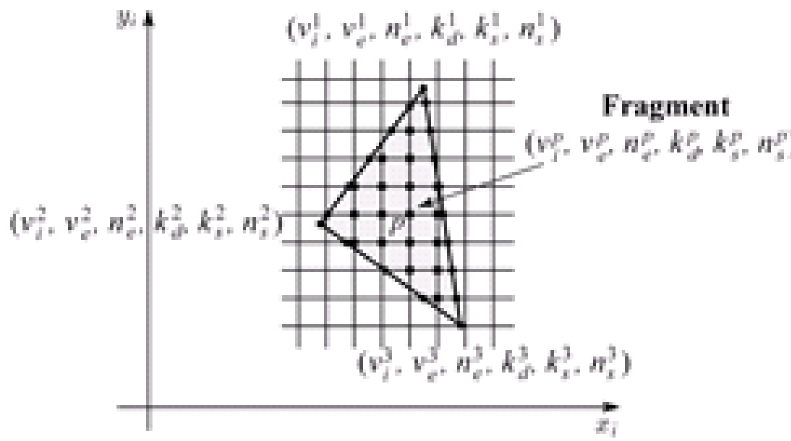
Programmable pipeline: Phong-interpolated normals!



Vertex shader:

attach n_e to vertex as "varying"
attach v_e to vertex as "varying"
$v_i \leftarrow$ project v to image

$v_i^1, v_i^2, v_i^3 \rightarrow$ triangle



Fragment shader:

$L \leftarrow$ determine lighting direction
$V \leftarrow$ determine viewing direction
$N \leftarrow$ normalize(n_e^p)
color $\leftarrow$ shade with $L, V, N, k_d^p, k_s^p, n_s^p$

Summary

You should understand the equation for the Blinn-Phong lighting model described in the "Iteration Four" slide:

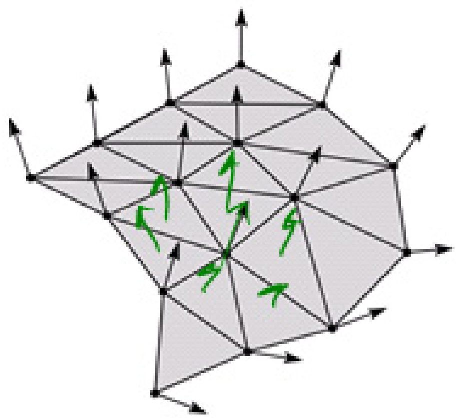
- What is the physical meaning of each variable?
- How are the terms computed?
- What effect does each term contribute to the image?
- What does varying the parameters do?

You should also understand the differences between faceted, Gouraud, and Phong *interpolated* shading.

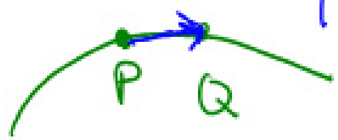
And you should understand how to compute the normal to a surface of revolution.

Surface normals

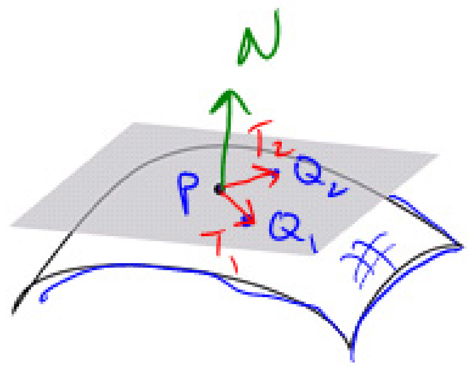
How can we compute the normal to a surface at a given point?



Tangent vectors and tangent planes



$$\vec{T} = \lim_{Q \to P} \frac{Q - P}{\|Q - P\|}$$

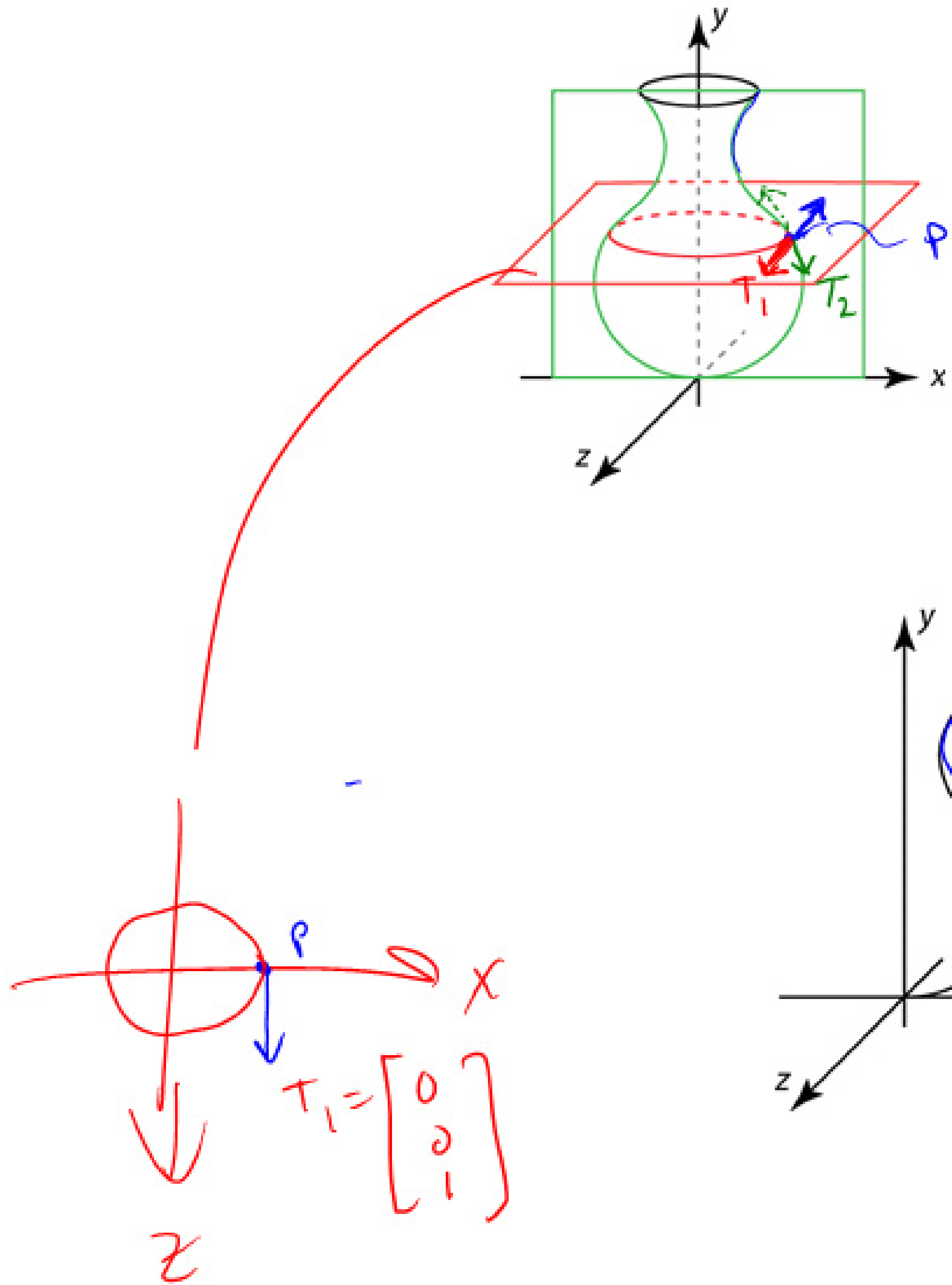


$$T_1 = Q_1 - P$$

$$T_2 = Q_2 - P$$

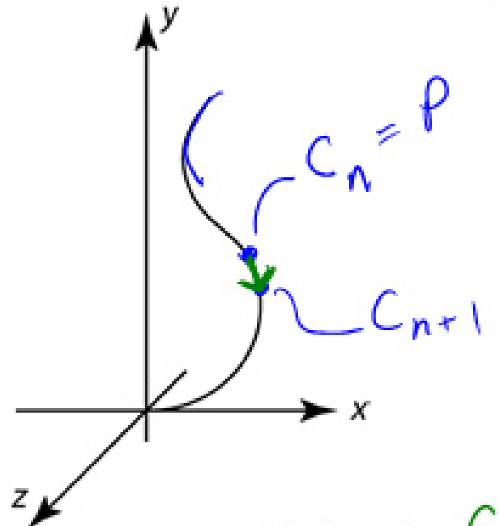
$$N \approx T_1 \times T_2$$

Normals on a surface of revolution



$N \sim T_1 \times T_2$

rotate around to get normals everywhere



$T_2 = C_{n+1} - C_n$