

# CSC 4356

## Interactive Computer Graphics

### Lecture 16: Illumination (Part 3)

Jinwei Ye

<http://www.csc.lsu.edu/~jye/CSC4356/>

Tue & Thu: 10:30 - 11:50am  
218 Tureaud Hall

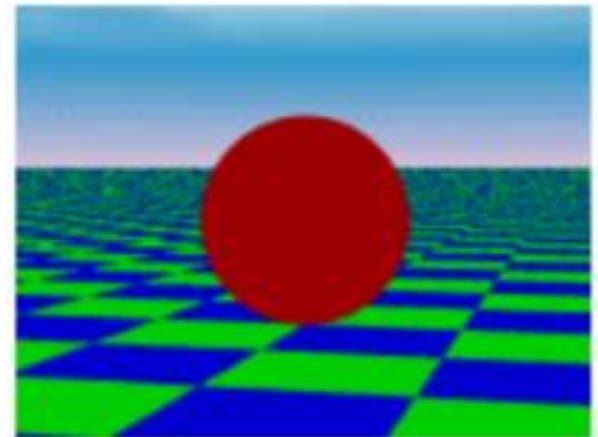
# Ambient Reflectance

- The amount of ambient light that is reflected by an object is independent of the object's position or orientation
- Surface properties are used to determine how much ambient light is reflected

$$I_{ambient} = k_a I_a$$

Ambient Reflectance      Ambient Reflectivity      Ambient Light Intensity

The diagram shows the equation  $I_{ambient} = k_a I_a$  with three red arrows pointing upwards from the labels below to the corresponding terms in the equation: one from 'Ambient Reflectance' to  $I_{ambient}$ , one from 'Ambient Reflectivity' to  $k_a$ , and one from 'Ambient Light Intensity' to  $I_a$ .



# Computing Diffuse Reflection

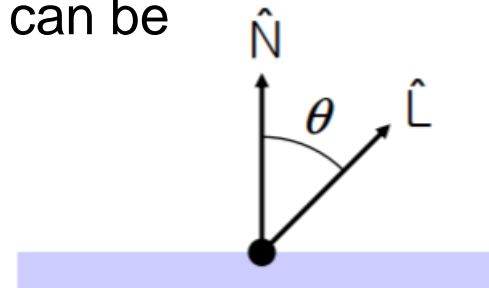
- The angle between the surface normal and the incoming light ray is called the angle of incidence and we can express a intensity of the light in terms of this angle  $\theta$

$$I_{diffuse} = k_d I_l \cos \theta$$

Diffuse Reflectance      Diffuse Reflectivity      Light Intensity      Incident Angle

- In practice, we can use dot product to compute  $\cos\theta$ 
  - If both the surface normal and the lighting direction are normalized (unit length) then diffuse reflectance can be computed as

$$I_{diffuse} = k_d I_l (\hat{n} \cdot \hat{l})$$



# Diffuse Light Examples

- Below are several examples of a spherical diffuse reflector with a varying lighting angles.
  - Why consider a spherical surface?
  - We need only consider angles from 0 to 90 degrees
  - Greater angles (where the dot product is negative) are blocked by the surface and the reflectance is zero



# Specular Reflection

- A second surface type is called a specular reflector (e.g. mirror, polished metal)
  - Have bright, *view-dependent* highlight
  - At microscopic level, a specular surface is very smooth



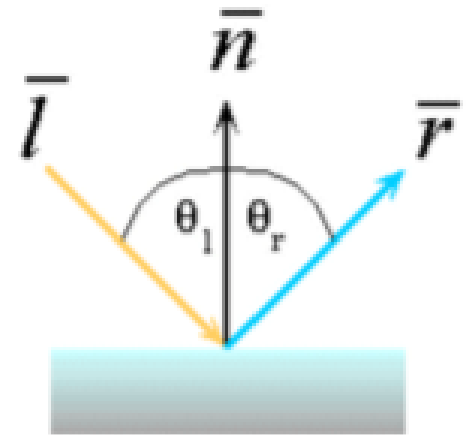
# Snell's Law

- Specular reflection follows the Snell's Law
  - The incoming ray, the surface normal, and the reflected ray all lie in a common plane.
  - The angle that the reflected ray forms with the surface normal is determined by the angle that the incoming ray forms with the surface normal, and the relative speeds of light of the mediums in which the incident and reflected rays propagate

$$n_l \sin \theta_l = n_r \sin \theta_r$$

- In reflection, the medium is the same ( $n_l = n_r$ )
  - So we simplify the expression to

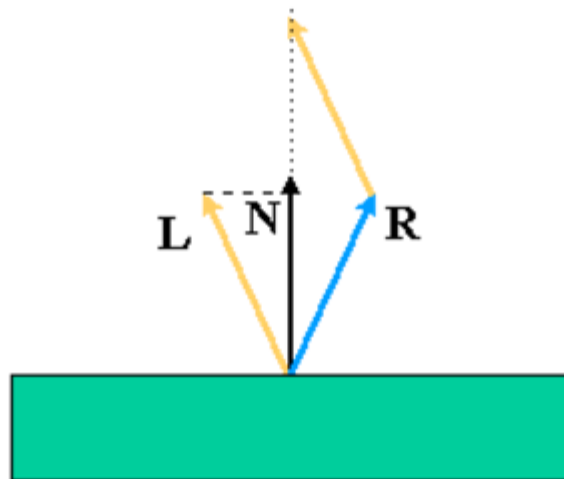
$$\theta_l = \theta_r$$



# How to Compute Reflection Vector?

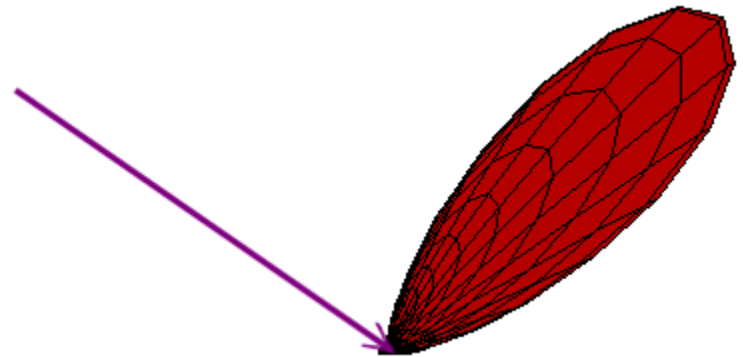
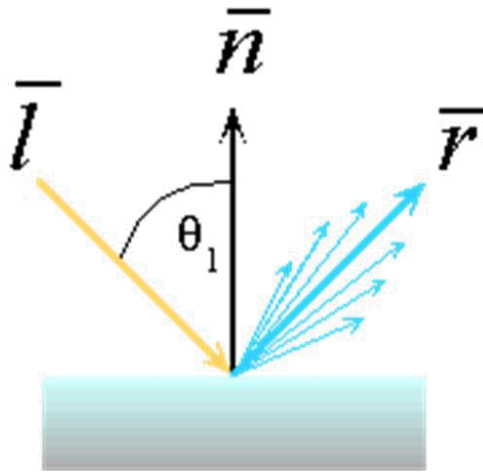
- The vector reflection vector  $R$  can be computed from the incoming light direction and the surface normal as shown below:

$$\hat{r} + \hat{l} = (2(\hat{n} \cdot \hat{l}))\hat{n} \quad \longrightarrow \quad \hat{r} = (2(\hat{n} \cdot \hat{l}))\hat{n} - \hat{l}$$



# Non-ideal Specular Reflector

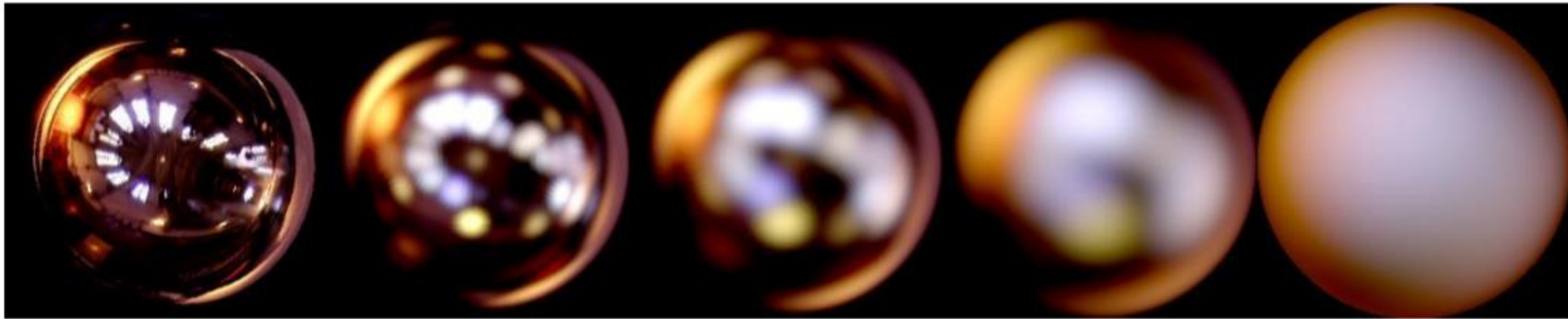
- Snell's law applies only to ideal specular reflectors
- Real materials, other than mirrors and chrome tend to deviate significantly from ideal specular reflectors
  - Roughness of surfaces causes highlight to “spread out”
- Now we introduce an empirical model that is consistent with our experience (but without capture the physics of it)





# Blurred Highlights

- Blurred highlights caused by surface roughness



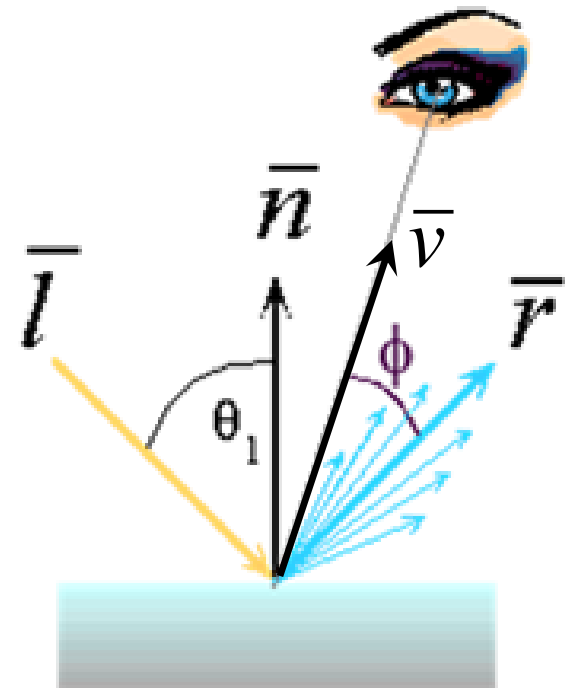
Roughness

# Phong Illumination

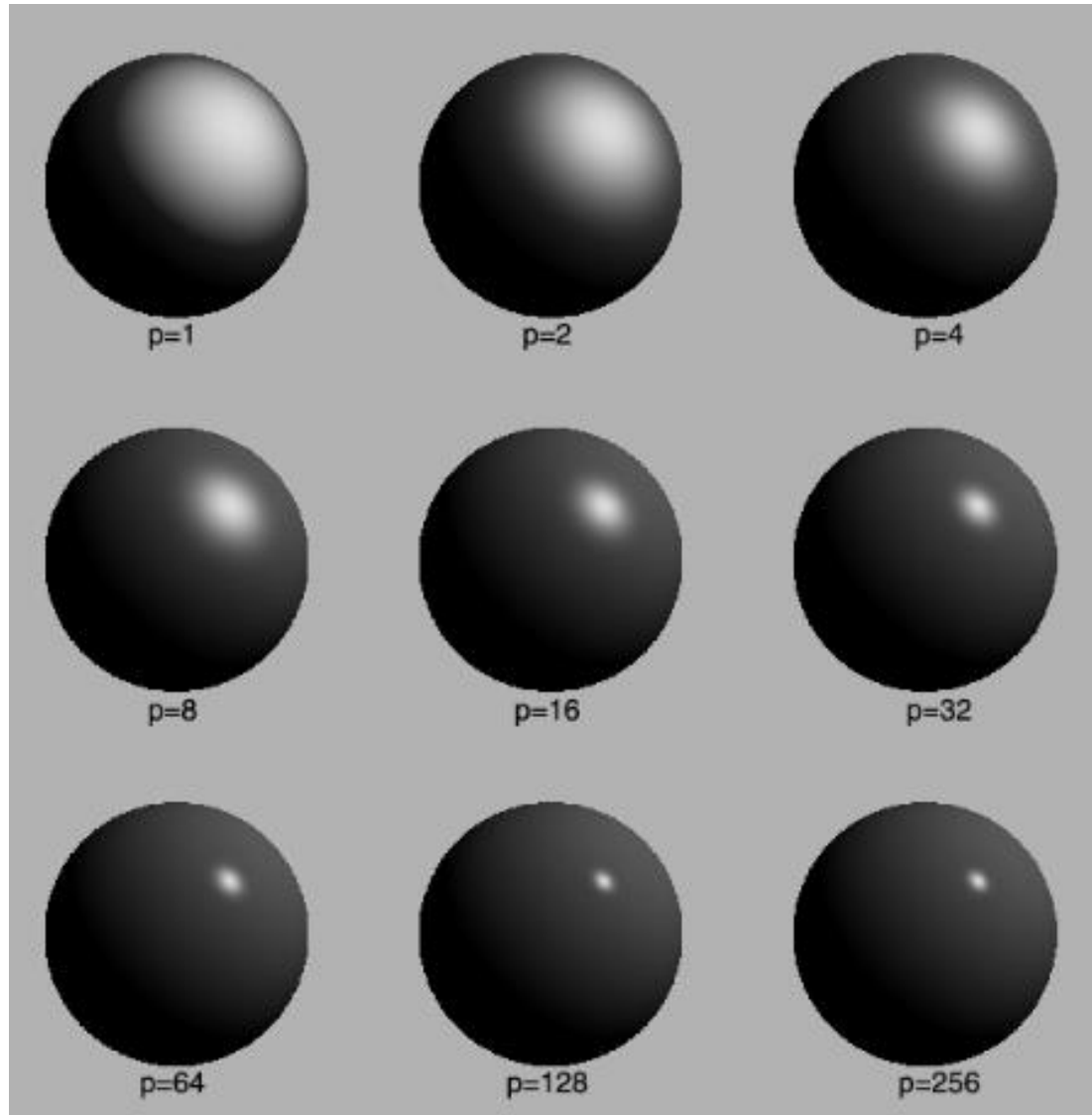
- Phong Illumination model approximates this fall off
  - This model has no physical basis
  - Yet it is one of the most commonly used illumination models in computer graphics

$$\begin{aligned} I_{specular} &= k_s I_l (\cos \phi)^{n_{shiny}} \\ &= k_s I_l (\hat{v} \cdot \hat{r})^{n_{shiny}} \end{aligned}$$

- $\hat{v}$  is the direction to the viewer
- The  $n_{shiny}$  controls how quickly the highlight falls off
  - The larger the exponent, the faster fall off



# Effect of Shininess

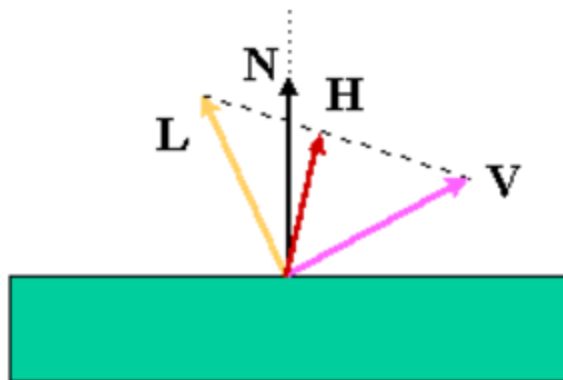


# Blinn & Torrance Variation

- Jim Blinn introduced another approach for computing Phong-like illumination based on the work of Ken Torrance

$$I_{specular} = k_s I_l (\hat{n} \cdot \hat{H})^{n_{shiny}}$$

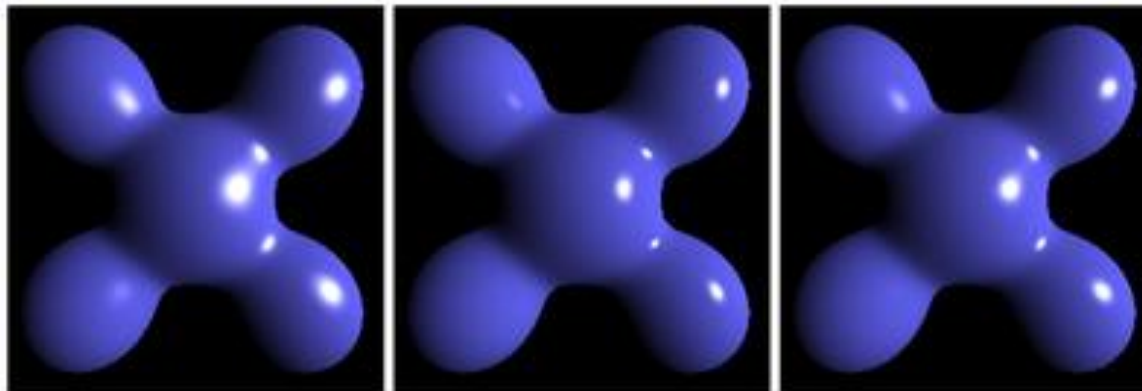
- Halfway vector H: a vector bisecting the incoming light direction and the viewing direction



$$\hat{H} = \frac{\hat{l} + \hat{v}}{|\hat{l} + \hat{v}|}$$

# What is the difference?

- The angle between the halfway vector and the surface normal is likely to be smaller than the angle between  $R$  and  $V$  used in Phong's model
  - unless the surface is viewed from a very steep angle, the angle between  $H$  and  $N$  is likely to be larger
- We can set larger exponent (shininess) for Blinn



Blinn






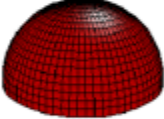

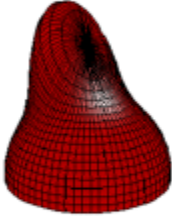



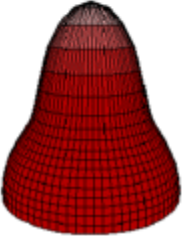
Phong

Blinn  
(with  $4 \times n_{\text{shiny}}$ )

# Putting It All Together

- Our final empirical illumination model is:

$$I_{total} = k_a I_a + \sum_{i=1}^{lights} I_i (k_d (\hat{n} \cdot \hat{l}) + k_s (\hat{v} \cdot \hat{r})^{n_{shiny}})$$

Phong	$\rho_{ambient}$	$\rho_{diffuse}$	$\rho_{specular}$	$\rho_{total}$
$\phi_i = 60^\circ$				
$\phi_i = 25^\circ$				
$\phi_i = 0^\circ$				

# Lighting in OpenGL

```
# define a directional light
float lightDirection[] = { 2.0f, 0.0f, 1.0f, 0 };
glLightfv(GL_LIGHT0, GL_POSITION, lightDirection);
glEnable(GL_LIGHT0);

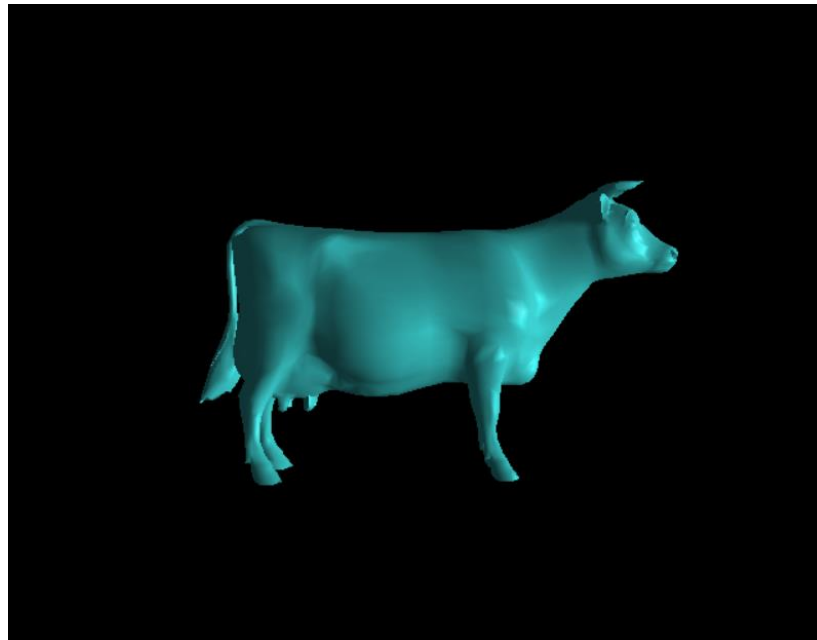
# set up light colors
float ambientIntensity[] = { 0.1f, 0.1f, 0.1f, 1.0f };
float lightIntensity[] = { 0.9f, 0.9f, 0.9f, 1.0f };
float lightSpec[] = { 1.0f, 1.0f, 1.0f, 1 };

glLightfv(GL_LIGHT0, GL_AMBIENT, ambientIntensity);
glLightfv(GL_LIGHT0, GL_DIFFUSE, lightIntensity);
glLightfv(GL_LIGHT0, GL_SPECULAR, lightSpec);
```

# Lighting in OpenGL

```
# set up surface material properties
float frontColor[] = { 0.2f, 0.7f, 0.7f, 1.0f };

glMaterialfv(GL_FRONT, GL_AMBIENT, frontColor);
glMaterialfv(GL_FRONT, GL_DIFFUSE, frontColor);
glMaterialfv(GL_FRONT, GL_SPECULAR, frontColor);
glMaterialf(GL_FRONT, GL_SHININESS, 100);
```





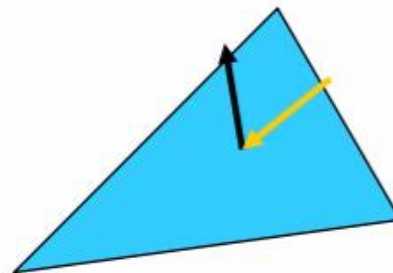
# Where do we Illuminate?

- To this point, we have discussed how to compute an illumination model at a point on a surface. But, at **which points on the surface** is the illumination model applied?
- Where and how often it is applied has a noticeable effect on the result
- For models defined by collections of polygonal facets or triangles:
  - Each facet has a common surface normal
  - If the light is directional then the diffuse contribution is constant across the facet
  - If the eye is infinitely far away and the light is directional then the specular contribution is constant across the facet

# Flat Shading

- The simplest shading method applies only one illumination calculation for each primitive
  - This technique is called *constant* or *flat shading*
  - Often used on polygonal primitives
- Which point on the facet should we use for computing illumination?

- The centroid  $centroid = \frac{1}{vertices} \sum_{i=1}^{vertices} \bar{P}_i$



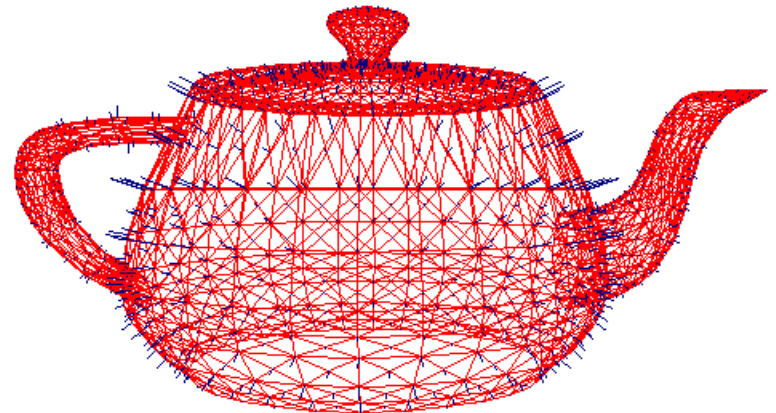
# Issues with Flat Shading

- Polygonal shape is still apparent
  - Constant color for each facet
- For point light sources, the direction to the light source varies over the facet
- For specular reflections, the direction to the eye varies over the facet
- To overcome this limitation, normals are introduced at each vertex
  - Usually different than the polygon normal
  - Used only for shading (not backface culling or other geometric computations)
  - Better approximates the "real" surface

# Vertex Normals

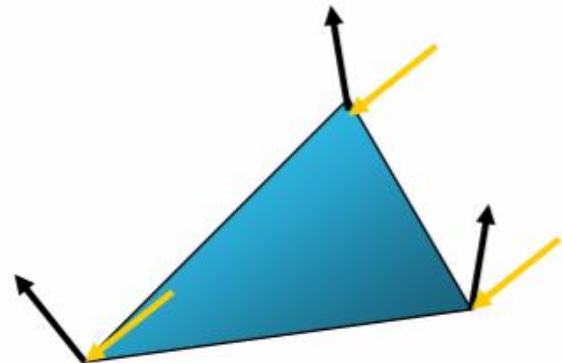
- In most obj files, the vertex normals are provided in object specifications
  - f int//int int//int int//int ...
- If vertex normals are not provided, they can often be approximated by averaging the normals of the facets which share the vertex
  - This only works if the polygons reasonably approximate the underlying surface

$$\bar{n}_v = \sum_{i=1}^k \frac{\bar{n}_i}{|\bar{n}_i|}$$



# Gouraud Shading

- The Gouraud Shading applies the illumination model on a subset of surface points and *interpolates the intensity* of the remaining points on the surface
  - In the case of a polygonal mesh the illumination model is applied at each vertex and the colors in the triangles interior are linearly interpolated from these vertex values
  - The linear interpolation can be accomplished using the plane equation method discussed in the lecture on rasterizing polygons
  - Notice that facet artifacts are still visible



# Interpolating Color

- Now we know how to draw a solid triangle (All vertices have the same color)
- What if they have different colors (or other parameters, e.g. depth)? How to interpolate?
- Idea: triangles are planar in any space:

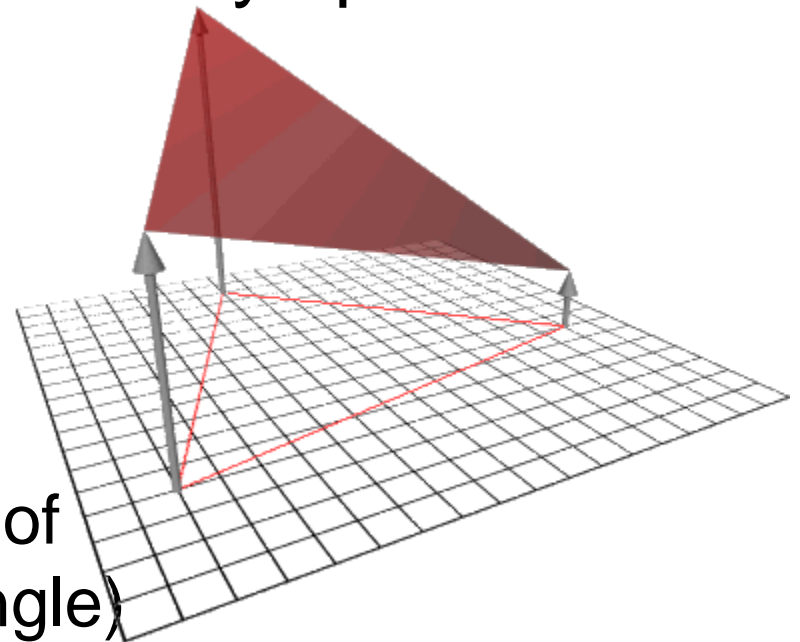
- This is the “redness” parameter space

- Also need to do this for green and blue

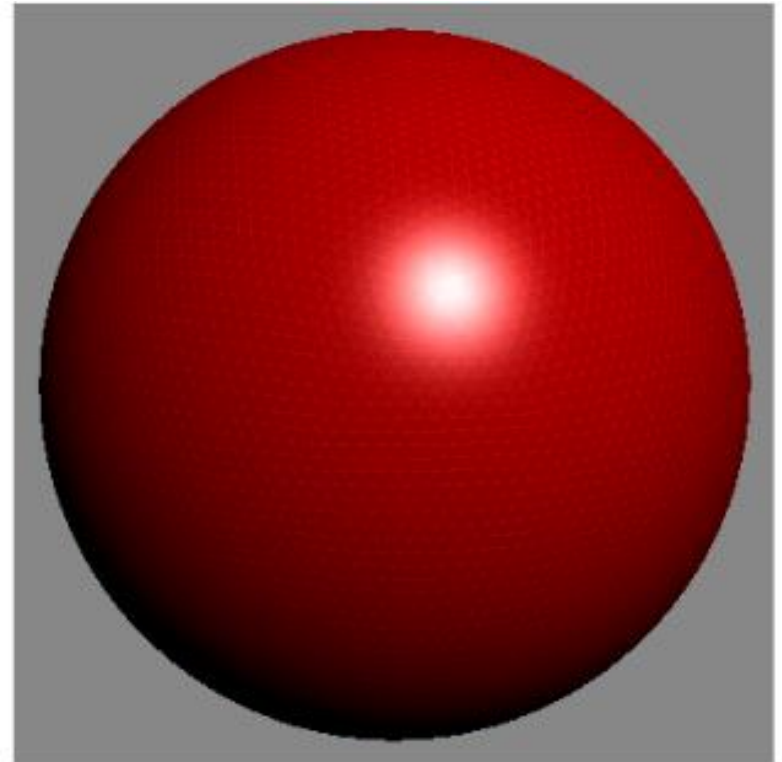
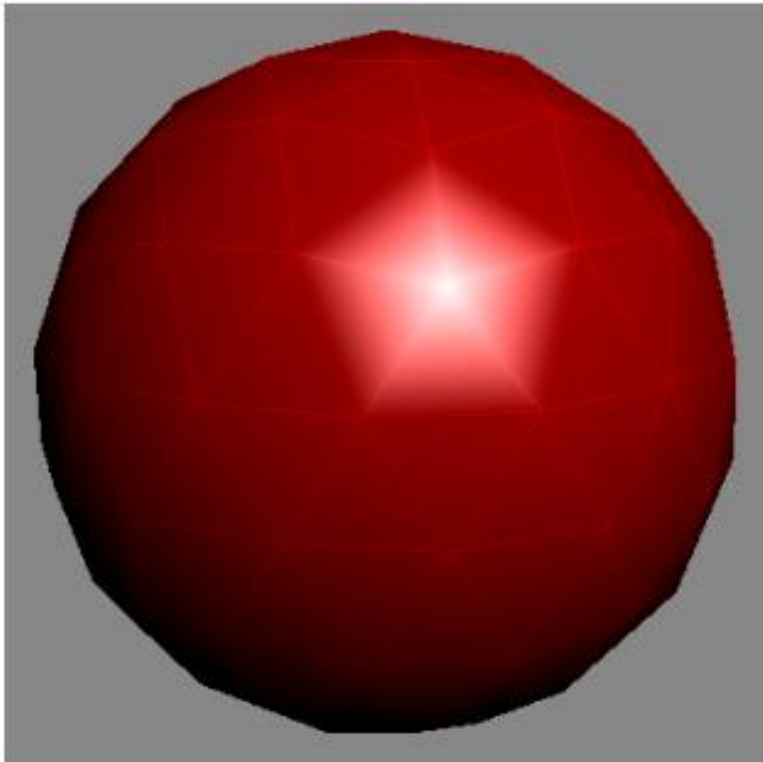
- Plane equation

$$z = A_r x + B_r y + C_r$$

(here  $z$  stands for redness of a point  $(x,y)$  inside the triangle)



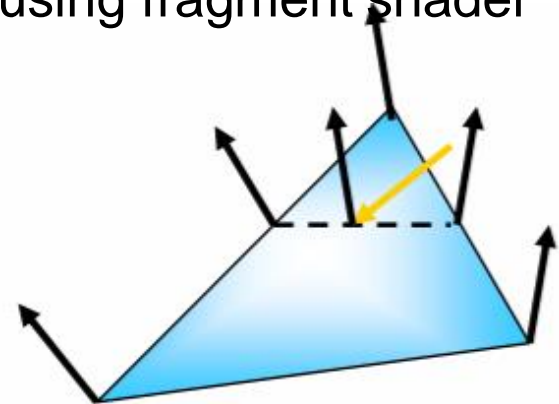
# Gouraud Shading Artifacts



- Poor handling on specular highlights
  - Issue lessens by using finer detailed geometry

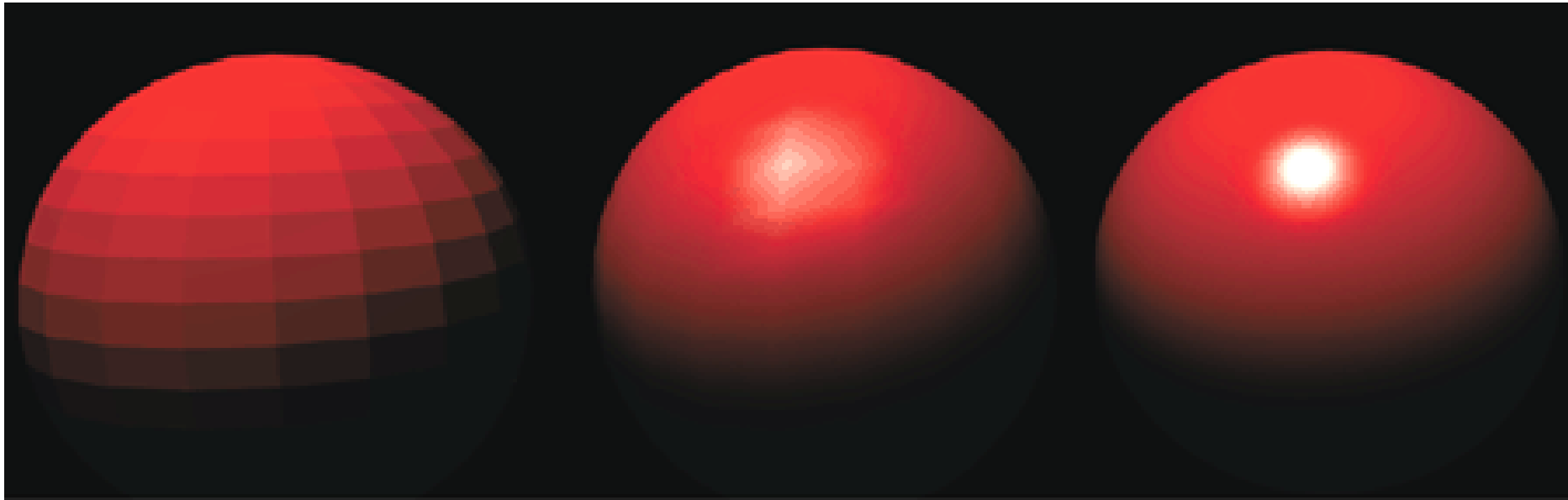
# Phong Shading

- In Phong shading (not to be confused with Phong's illumination model), the *surface normal is linearly interpolated* across polygonal facets, and the illumination model is applied at every point
  - Better handling on specular high lights and usually results in a very smooth appearance
  - Slower than Gouraud shading
  - NOT built into OpenGL (OpenGL uses Gouraud)
  - Can be implemented on graphics card using fragment shader





# Flat vs. Gouraud vs. Phong



**Flat**

Compute illumination model once on facet centroid

**Gouraud**

Compute illumination model on the vertices of a facet and then **interpolate color** for interior points

**Phong**

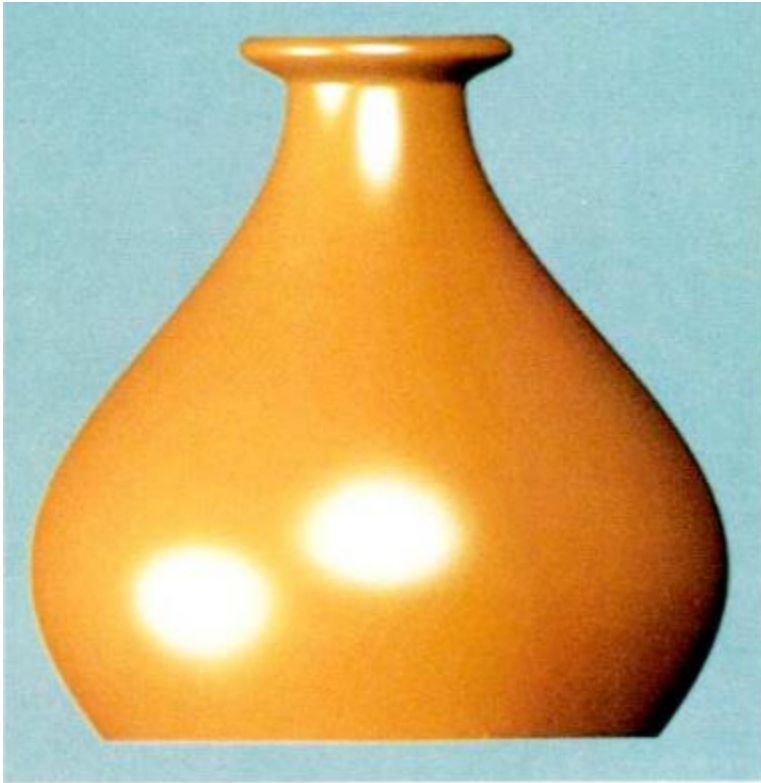
**Interpolate normal** for every point and then compute illumination model for each point on a facet

# Revisit The Empirical Model

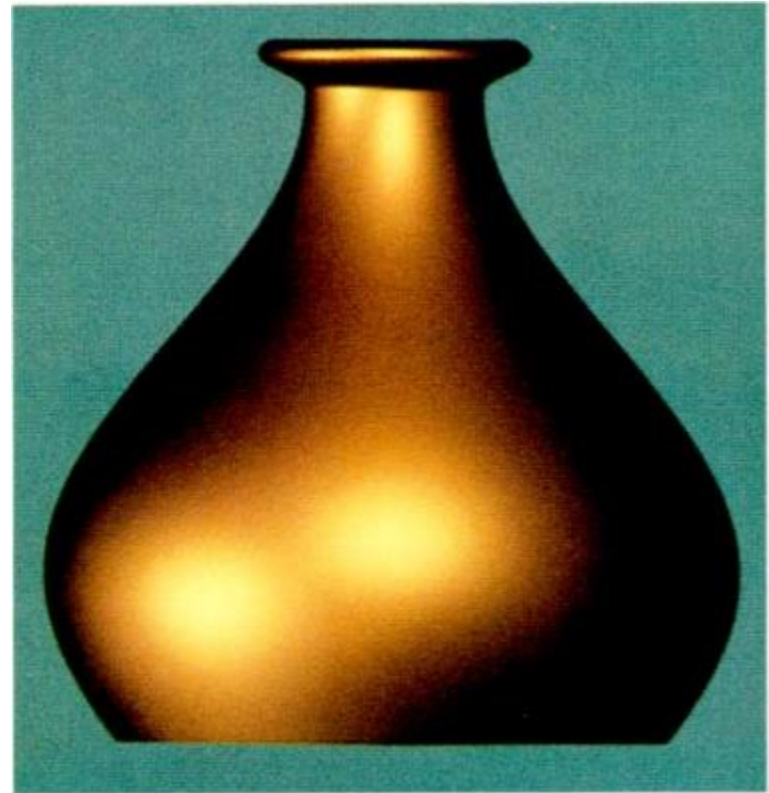
$$I_{total} = k_a I_a + \sum_{i=1}^{lights} I_i (k_d (\hat{n} \cdot \hat{l}) + k_s (\hat{v} \cdot \hat{r})^{n_{shiny}})$$

- Now we know where to apply this illumination model
- Even with Phong shading, any problem with this model?
  - What are  $k_a$ ,  $k_s$ , and  $n^{shiny}$ ? Are they measurable quantities?
  - What are the coefficients for copper?
  - Is my picture accurate?

# Cook-Torrance Result



**Plastic-looking copper vase rendered using Phong model**

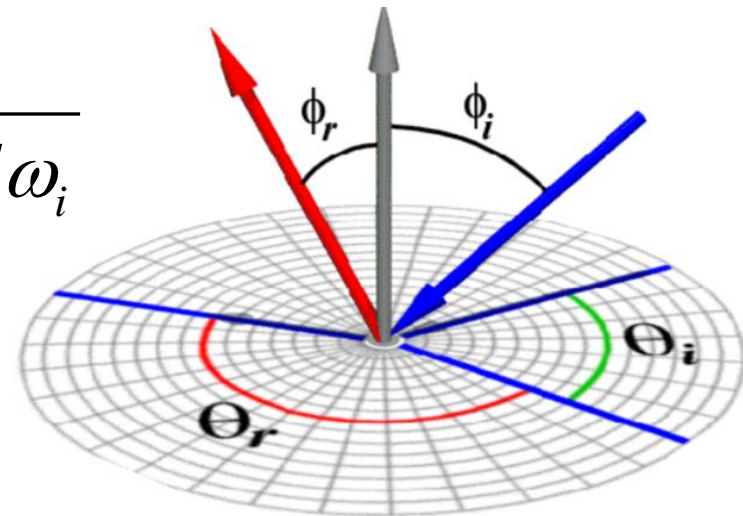


**A copper vase with a more metallic appearance**

# BRDF

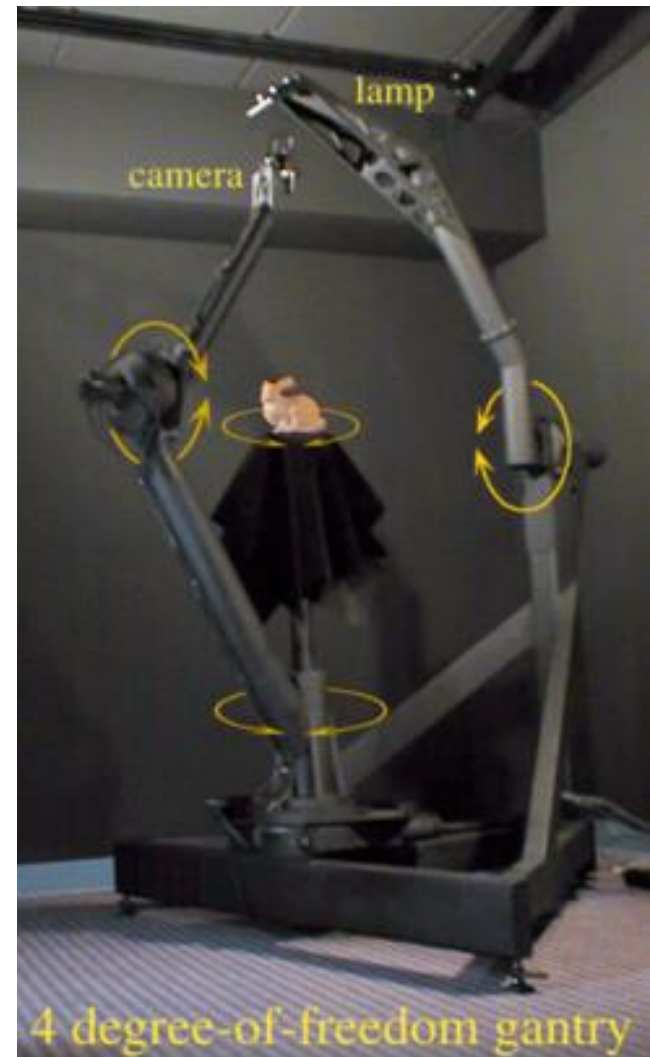
- **Bi-directional Reflectance Distribution Function** describes the transport of incoming light to outgoing light (surface radiance)
  - Ratio between outgoing surface radiance and incoming irradiance
  - Characterize surface reflectance/material

$$\rho(\theta_r, \phi_r, \theta_i, \phi_i) = \frac{L(\theta_r, \phi_r)}{L(\theta_i, \phi_i) \cos \theta_i d\omega_i}$$



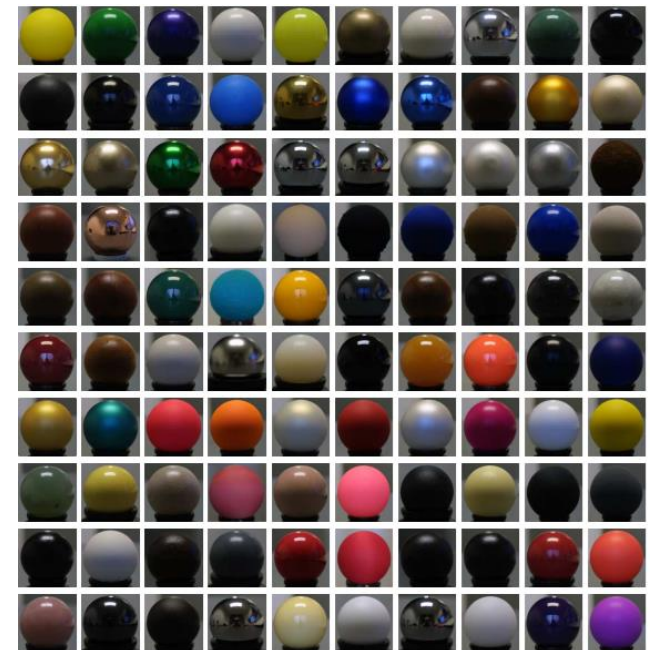
# Measuring BRDF

- Goniophotometer
  - 4 degree-of-freedom gantry
  - Measure one incoming/ outgoing light pair at time
  - Slow
  - Accurate



# Data Driven BRDF Modeling

- Acquisition system
  - Homogeneous sphere
  - Fixed camera
  - Orbiting light source
- 20 – 80 million reflectance per material sample
  - Each camera pixel in an image samples a BRDF
  - 330 pictures to cover half circle



# How to use BRDF data

- Directly use the BRDF measurements for rendering



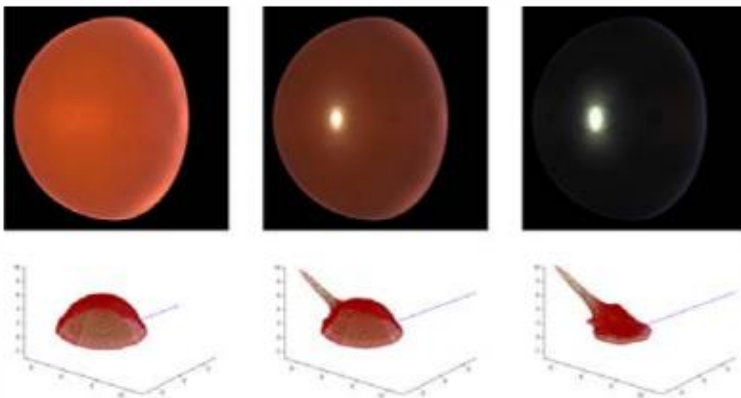
**Nickel**

**Hematite**

**Gold Paint**

**Pink Felt**

- Linear combination of acquired BRDFs can be used to synthesize new materials



# BRDF Editing

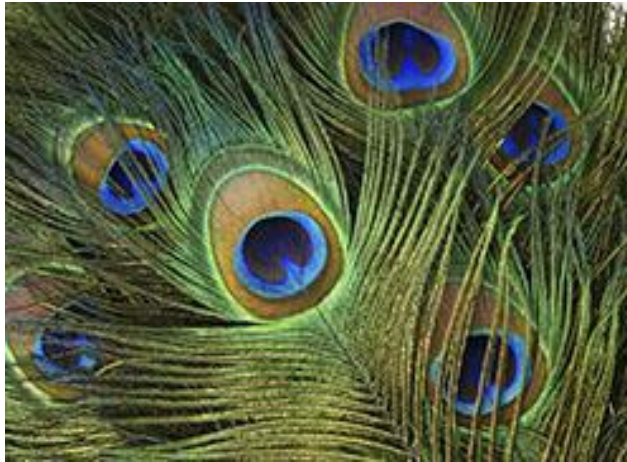
- BRDF Shop

- Video:

- <https://www.youtube.com/watch?v=HmvkNqrGvxs>



# Complex BRDFs/Materials



# Next Time ...

- Programming the GPU
  - Shading Language
  - Fragment shader
  - Vertex shader
- Reading:
  - Textbook Chapter 22
- Office Hour Change (This Week Only)
  - Thursday 2-4pm -> Tuesday 2-4pm