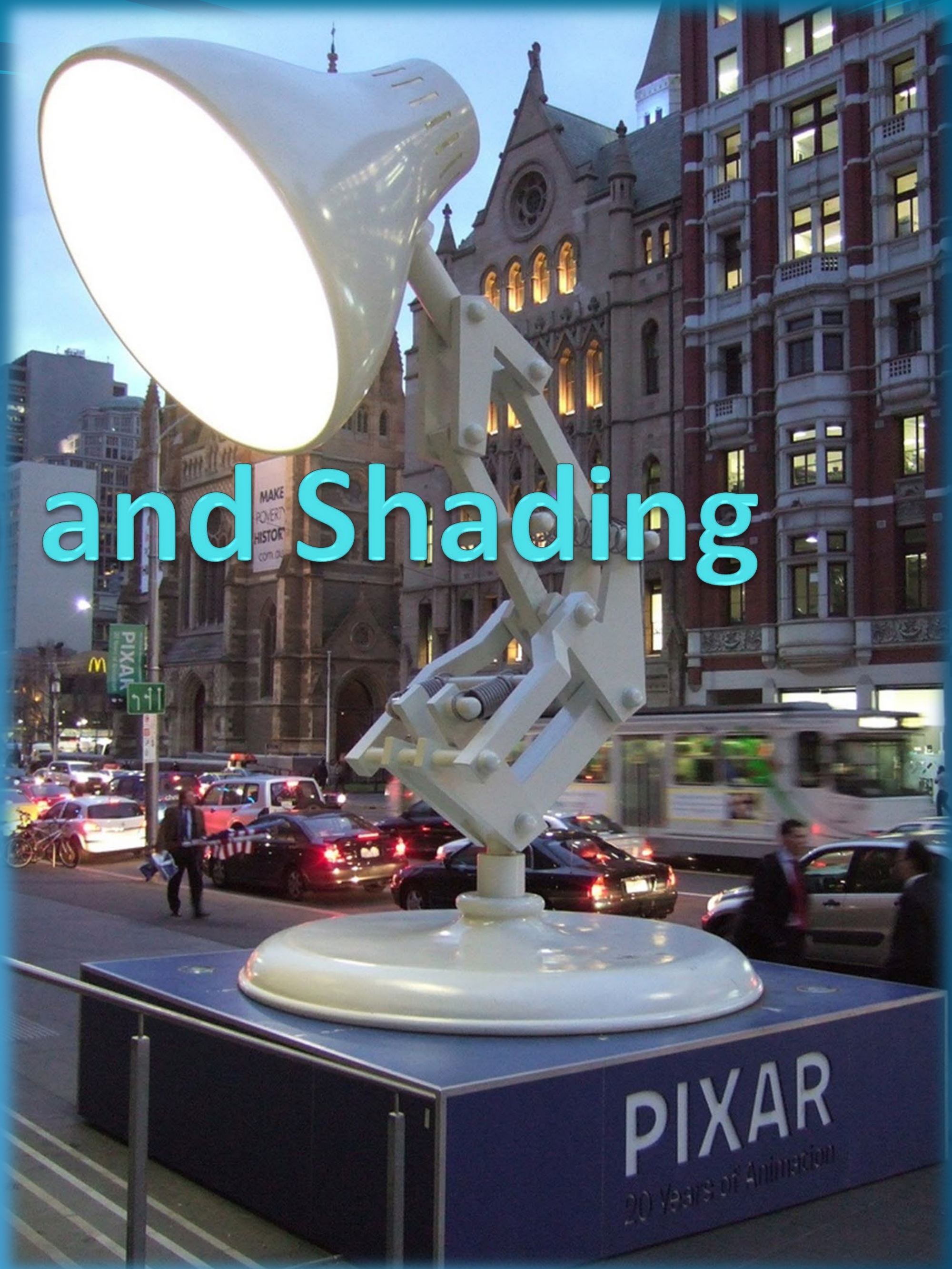
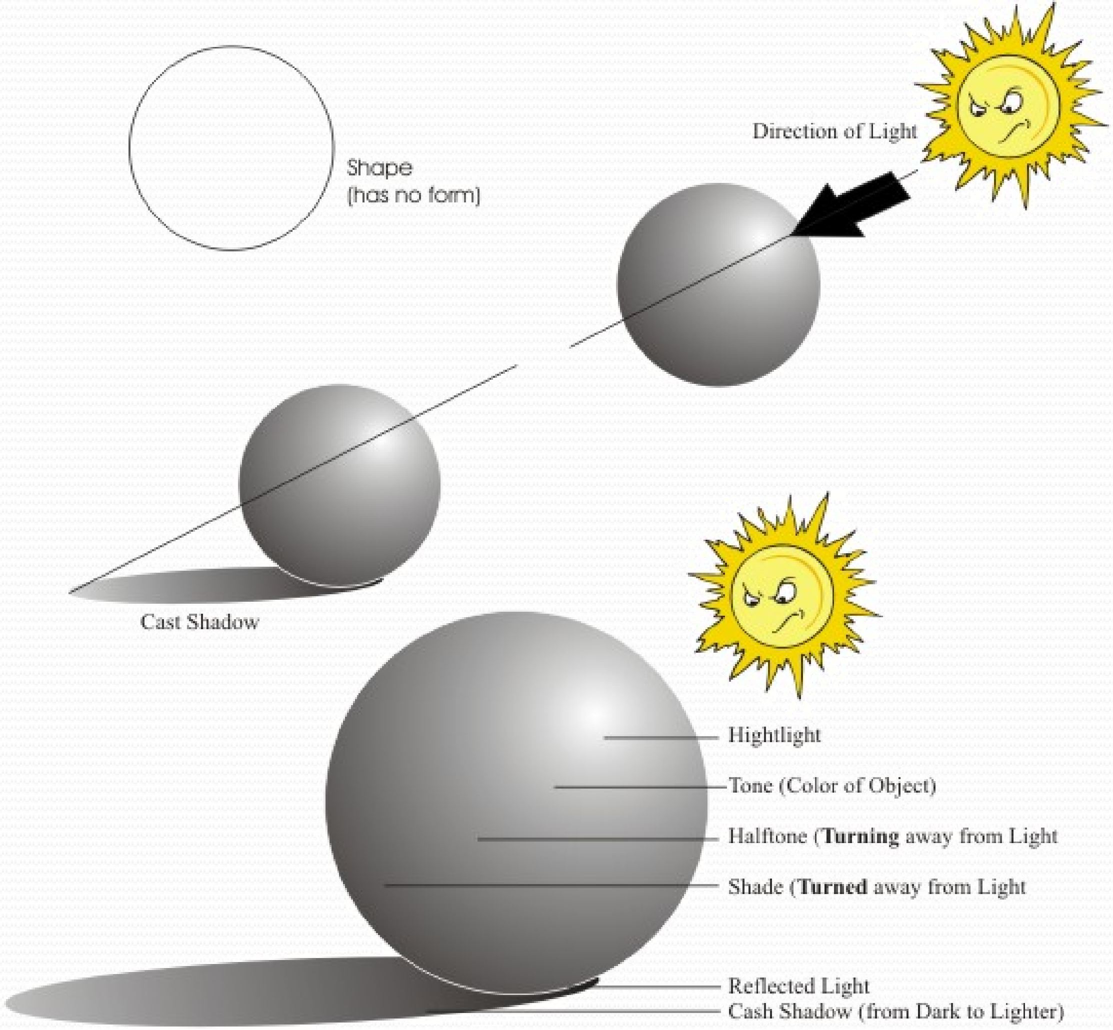
Illumination and Shading

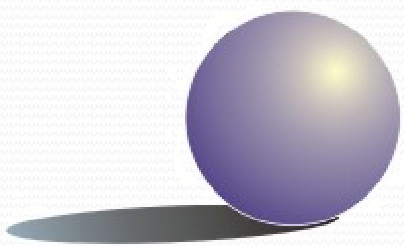
"Fiat Lux!"

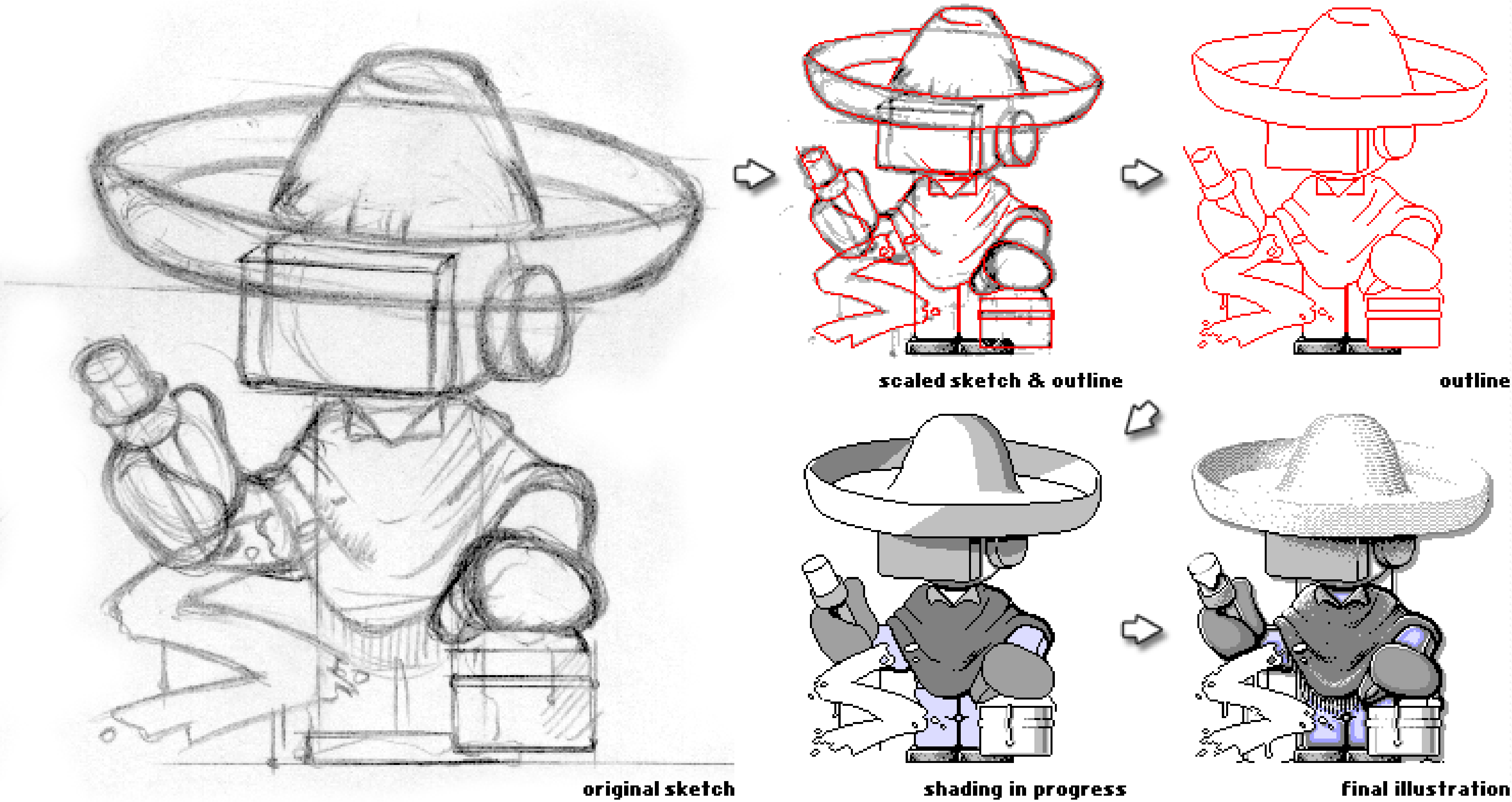


Illumination and Shading

- Shading (dictionary definition)
  - The use of marking made within outlines to suggest three-dimensionality, shadow, or degrees of light and dark in a picture or drawing (Merriam-Webster)


Shape
(has no form)
Direction of Light
Cast Shadow
Hightlight
Tone (Color of Object)
Halftone (**Turning** away from Light
Shade (**Turned** away from Light
Reflected Light
Cash Shadow (from Dark to Lighter)




original sketch
scaled sketch & outline
outline
shading in progress
final illustration

Wireframe

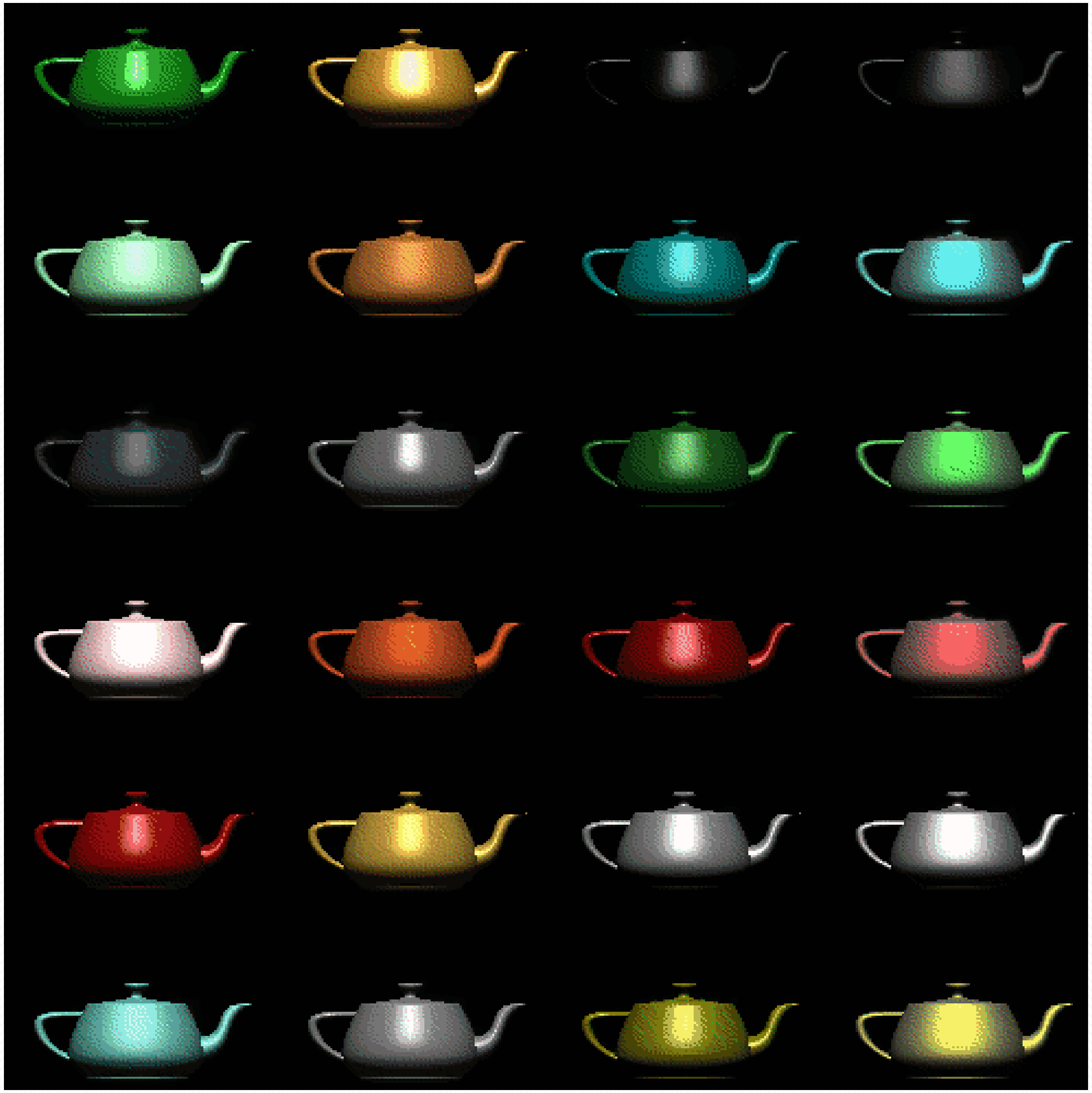
Ambient Lighting

+ Diffuse Lighting

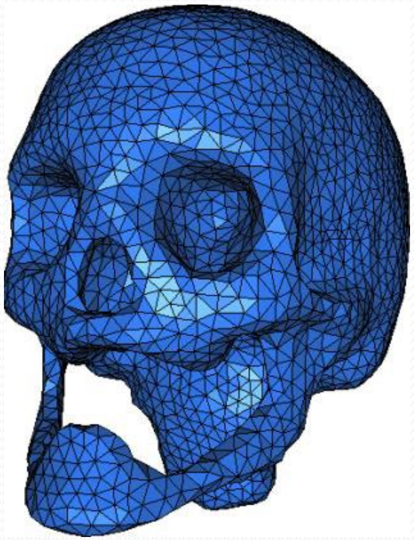

+ Specular Lighting

Materials

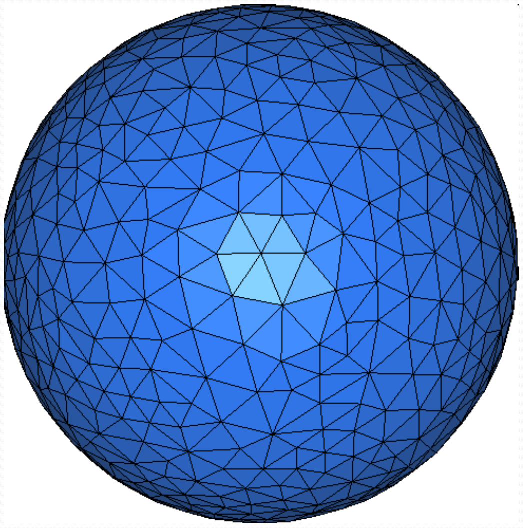
- By different colors and <u>reflectivity</u>, we can deceit the viewers that the objects are made of different materials
  - E.g. plastic, metals, etc.



Illumination and Shading

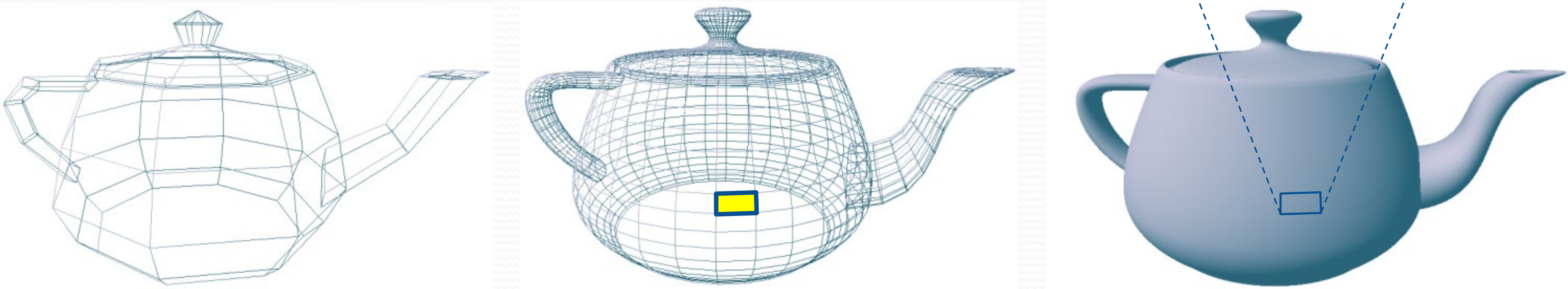


- Input:
  - Objects are in polygonal mesh representation
  - With their connectivity (topology)
    - Namely, we know the neighboring polygons of each polygon
- Create a raster-scan image that
  - Looks realistic
  - And do it fast, namely, in real time



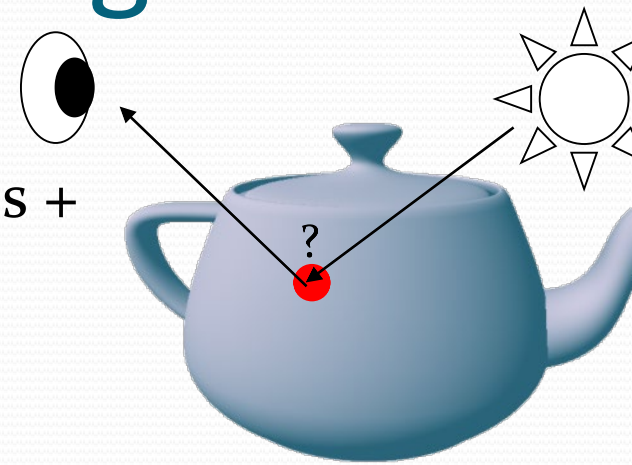
Polygonal Models

- Polygonal meshes are <u>**approximation**</u> of curve surfaces
  - If you have “fine enough” polygons, they look like a curve surface

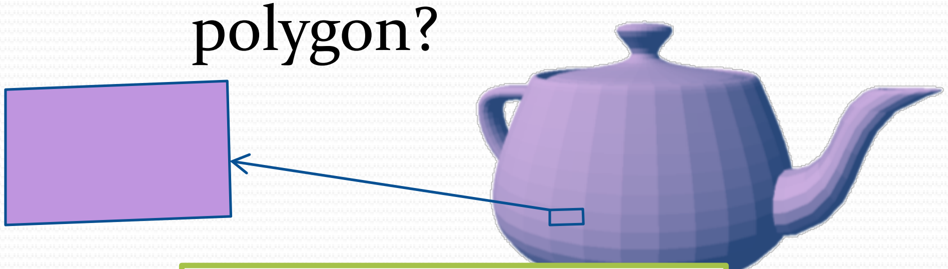


- Question:
  - How do you “color” every polygon to make it looks smooth

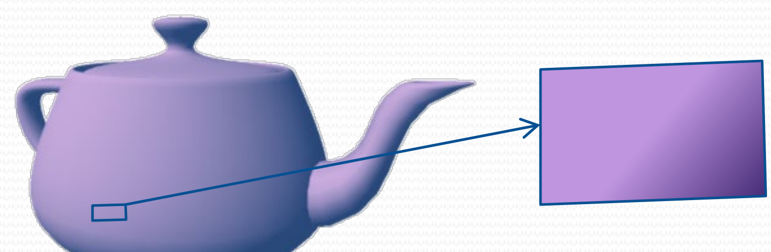
Illumination and Shading



- **Illumination model**
  - Given the surface **point** + light sources + the viewer
  - How to compute the colour of that surface point
- **Shading**
  - How do we determine the color of a polygon?

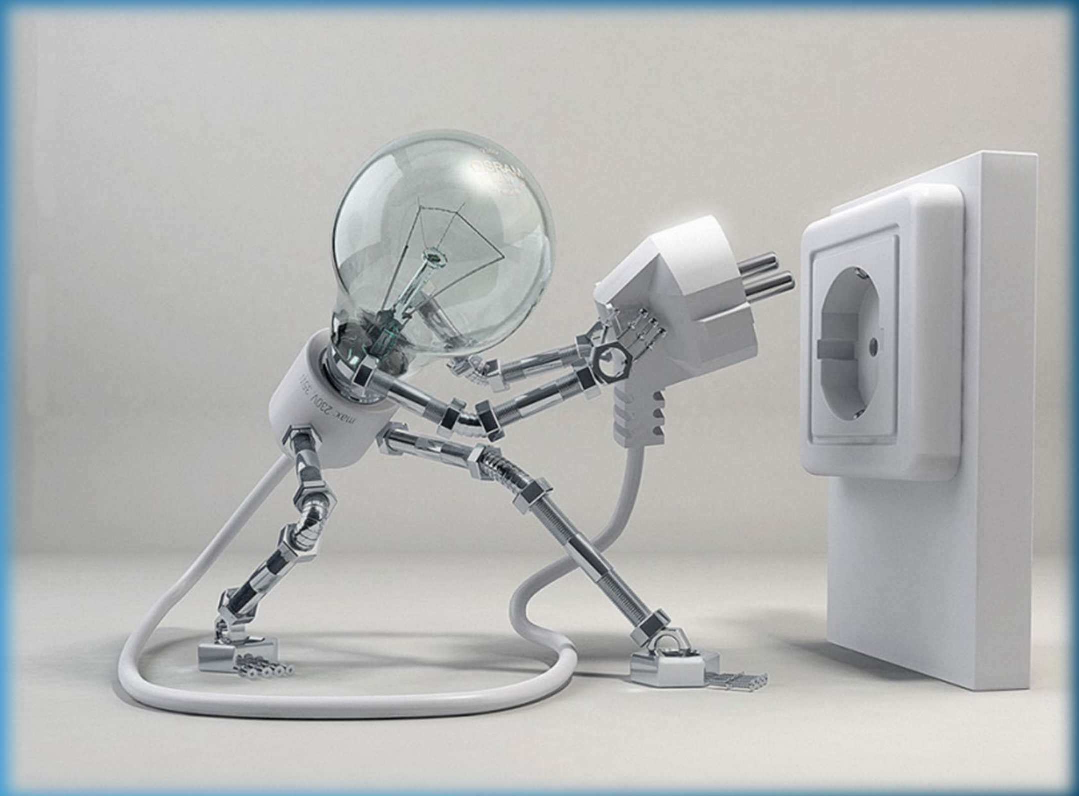


One single color for each polygon?

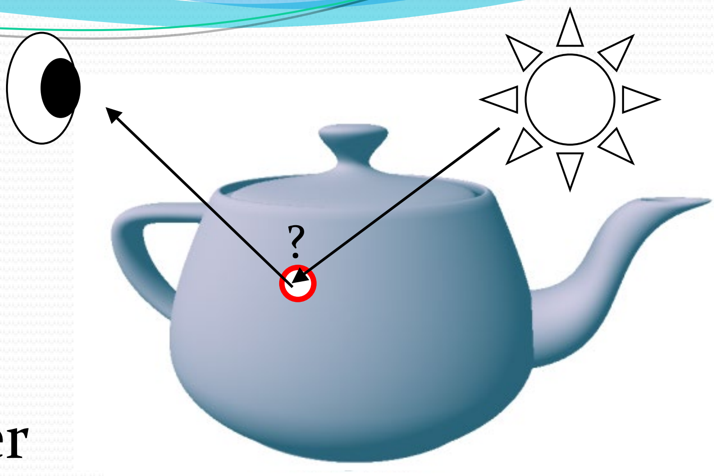


Blending colors for each polygon?

Illumination Model



Illumination Model



- Given the configurations of
  - Lights + a surface point + the viewer

- Compute the color of **a surface point** by the **Phong Illumination Equation**:

Final color of a point on the surface

$$I_{Phong} = \begin{pmatrix} r \\ g \\ b \end{pmatrix} = I_a K_a + f_{att} I_p K_d (N \cdot L) + f_{att} I_p K_s (R \cdot V)^n$$

PIE

Phong Illumination Model

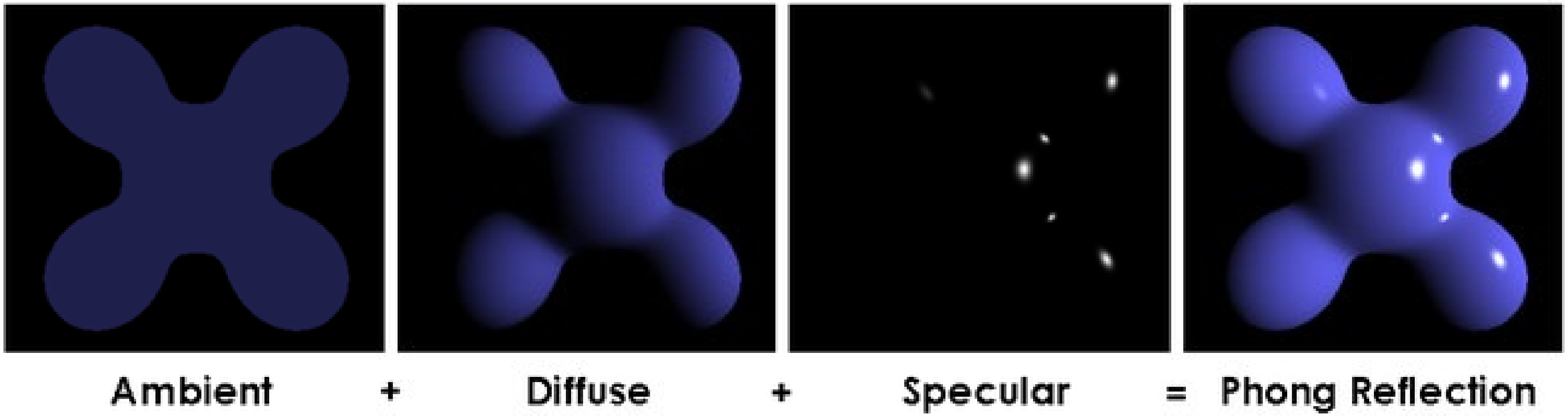
- Phong Model (1975) - A *local point reflection model* that compromises both acceptable *image quality* and *processing speed*.

Bùi Tường Phong (December 14, 1942 – July 1975) was a **Vietnamese-born French-American** computer graphics researcher and pioneer.

Phong Illumination Equation (PIE)

- For every point on a surface, we can compute its color/intensity by the Phone Illumination Equation:

$$I_{Phong} = \begin{bmatrix} i_r \\ i_g \\ i_b \end{bmatrix} = \boxed{I_a K_a} + f_{att} I_p K_d (N \cdot L) + f_{att} I_p K_s (R \cdot V)^n$$



Capital letters are 3 x 1 vectors and small caps are just constants

Ambient Term

- A ambient light is used to produce an uniform lighting effect on every point on every surface in the scene. Its luminance I_a is specified by

$$I_a = \begin{bmatrix} i_{ar} \\ i_{ag} \\ i_{ab} \end{bmatrix}$$

- Note that this light is "universal", namely, every surface receive the same color and intensity of light

Ambient Lighting

Under ONE Single white ambient lighting but every object has a different color

Material Property

- Then, for different objects, how can they appear as different colors?
- For **EACH** object/surface, we specify its own ambient material property

$$K_a = \begin{bmatrix} k_{ar} \\ k_{ag} \\ k_{ab} \end{bmatrix}$$

- Referring to the PIE, the color on a specific object is

$$I_a K_a = \begin{bmatrix} i_{ar} k_{ar} \\ i_{ag} k_{ag} \\ i_{ab} k_{ab} \end{bmatrix}$$

*Note that also there are other "*k*" for the remaining two terms

*Note that $I_a K_a$ is NOT the dot product
Actually most of the multiplications in PIE are NOT dot products

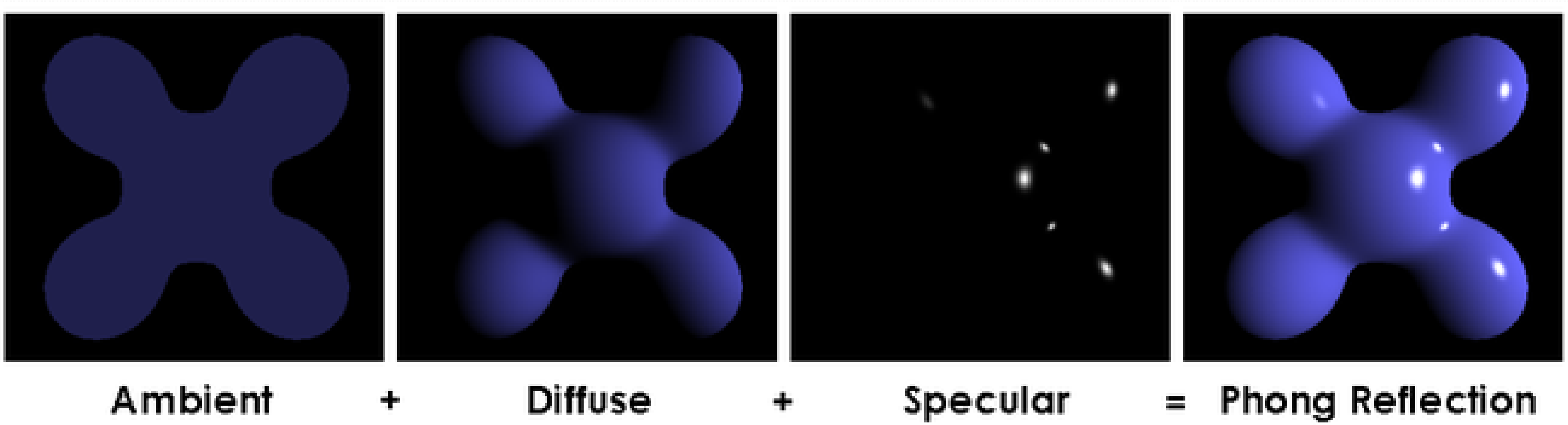
Ambient Lighting

However, it does not look nice

Diffuse Term of PIE

- The diffuse term give different colors to different points on the surface according to the <u>light positions</u> and surface normals

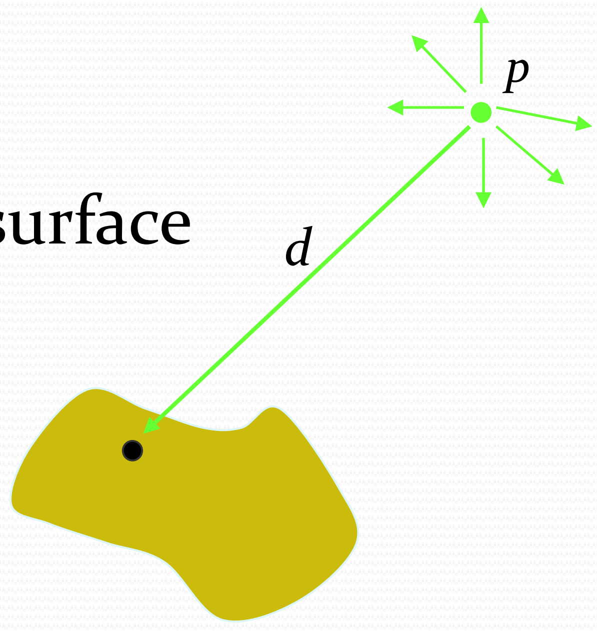
$$I_{Phong} = I_a K_a + \boxed{f_{att} I_p K_d (N \cdot L)} + f_{att} I_p K_s (R \cdot V)^n$$



Point Light Source

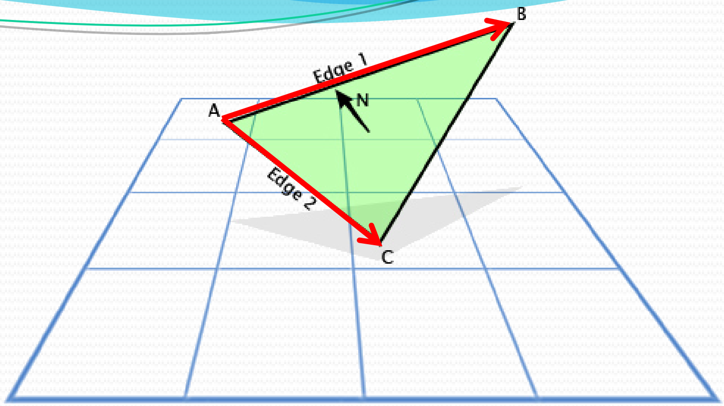
- Assuming the light is a point at position p
- Emitting light in every direction
- Similar to ambient light, its color/intensity is specified by a vector I_p
- However, the light received will be weaker if the object is far away from the light
  - When distance d ↑, light received ↓
- So the light intensity received by the surface point is

$$f_{att} I_p = \frac{1}{a + bd + cd^2} I_p$$



- a, b, c are user defined constants

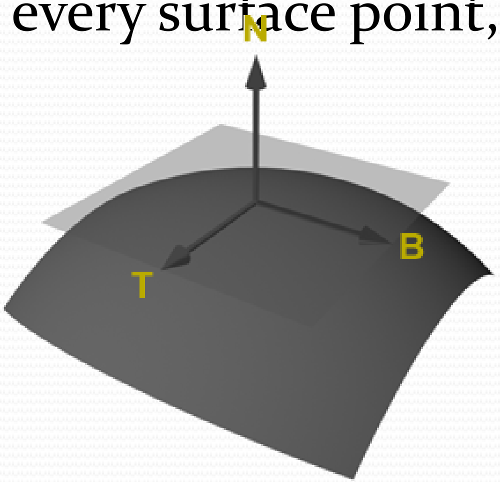
Surface Normal

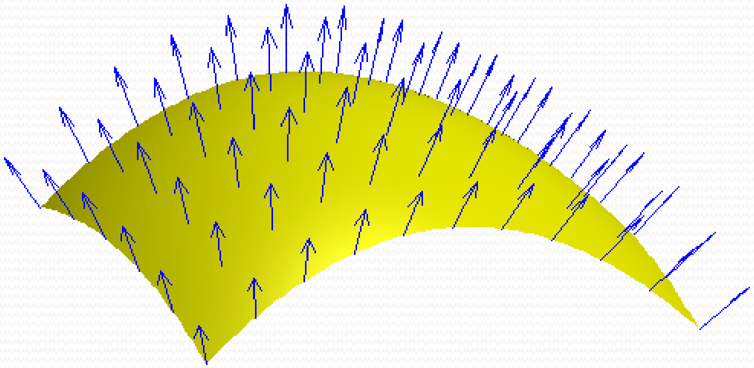


- For a triangle
  - The triangle spans a plane
  - There will be a normal vector N that is perpendicular

$$N = (B - A) \times (C - A)$$

  - (Note that the normal vector has two choices, N or $-N$)
- For a curved surface
  - For every surface point, there is a plane “parallel” to it

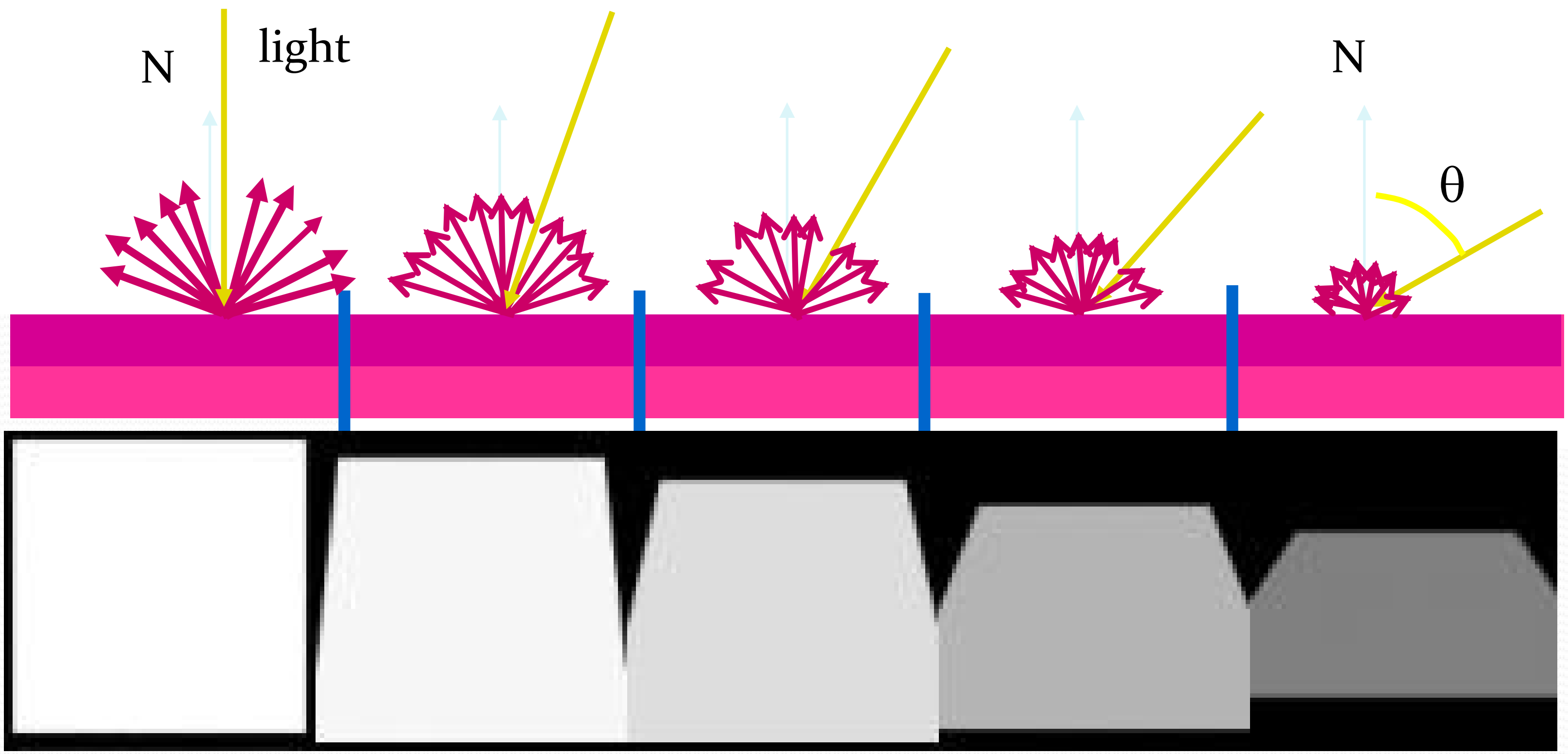




Diffuse Reflection

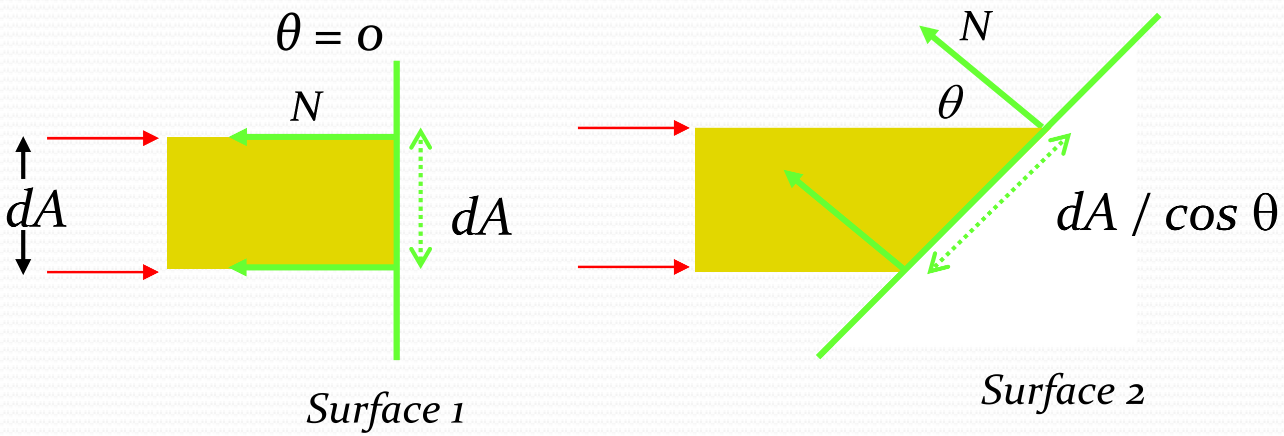
This IS the dot product

- Lambert's Cosine Law
  - $diffuse\ reflection \propto \cos\theta = N \bullet L$



Diffuse Term

- Consider 2 surfaces of different orientation, a light beam of infinitesimal cross-sectional area *dA* intercepts the surface at different angles of incident θ .



- Average amount of light energy that falls on a surface $\propto \cos\theta$

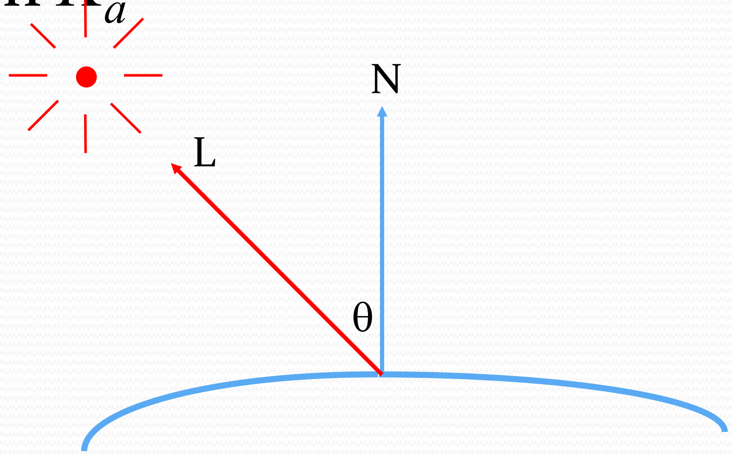
Diffuse Term

The direction **unit** vector from the surface point to the light source

- This accounts for the cos $\theta = N \bullet L$ term in the Lambert's cosine law mentioned earlier

$$I_{Phong} = I_a K_a + \boxed{f_{att} I_p K_d (N \cdot L)} + f_{att} I_p K_s (R \cdot V)^n$$

- K_d is the diffuse material property $[k_{dr} \;\; k_{dg} \;\; k_{db}]^T$ for the object, usually it is different from K_a

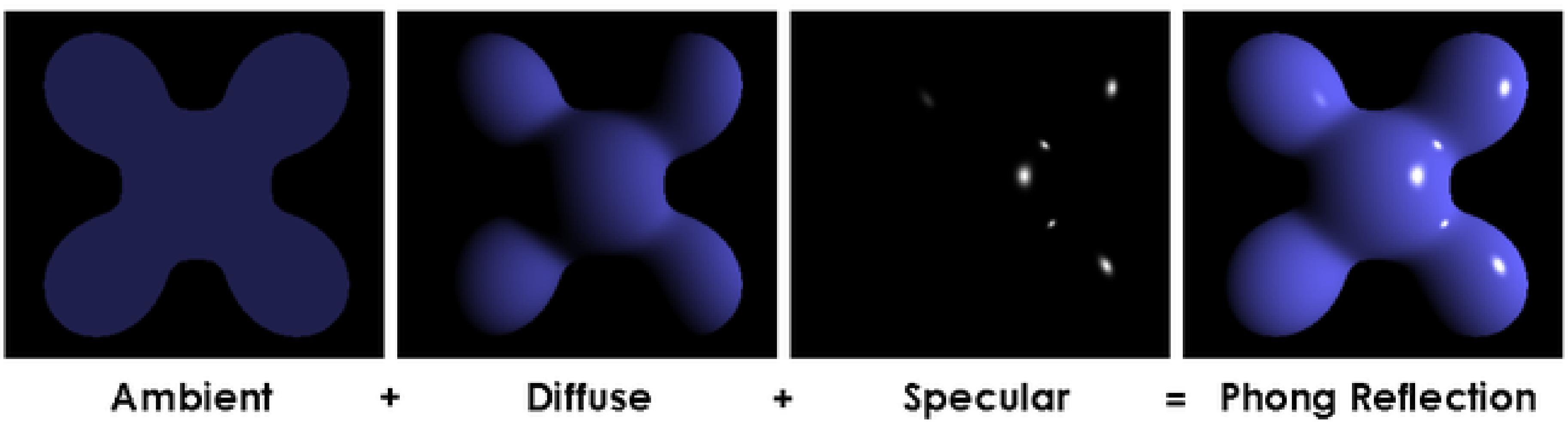


*Remember to normalize every directional vector

Specular Term of PIE

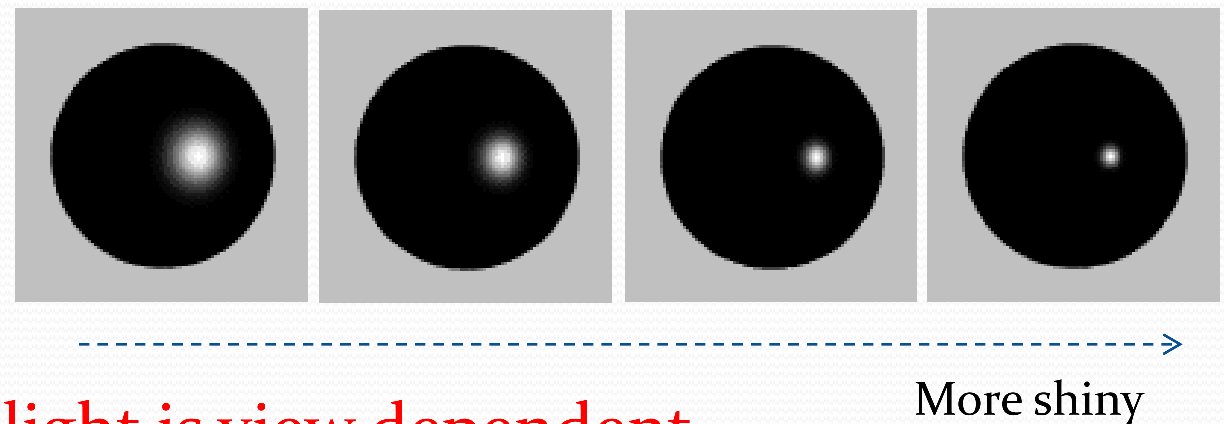
- Adding highlight for shiny objects

$$I_{Phong} = I_a K_a + f_{att} I_p K_d (N \cdot L) + \boxed{f_{att} I_p K_s (R \cdot V)^n}$$



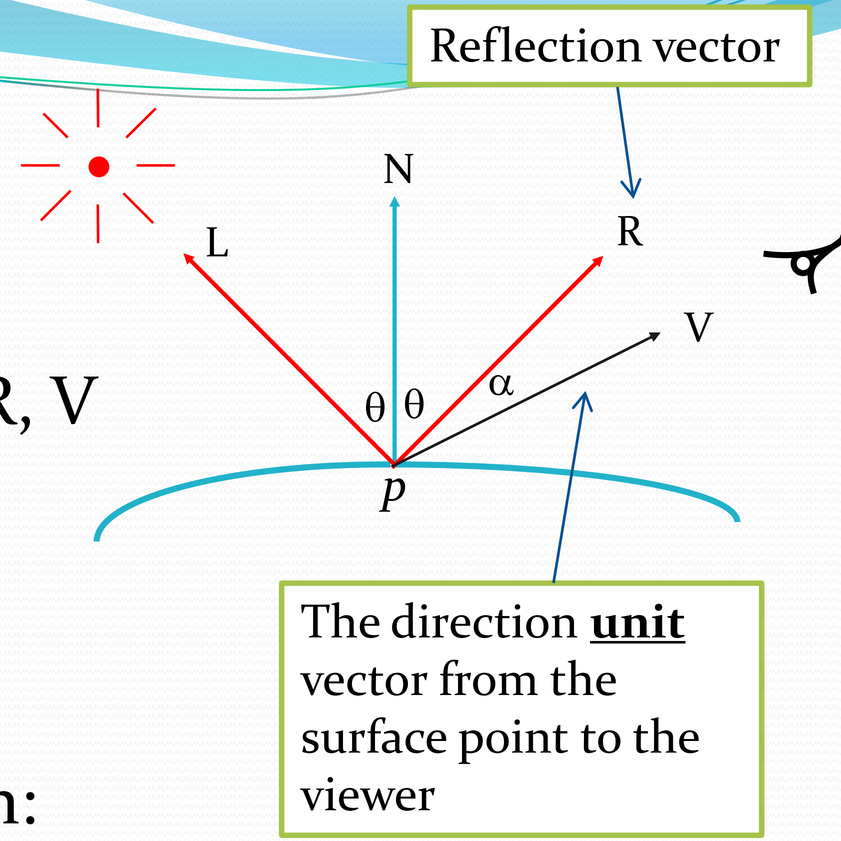
Specular Reflection

- Because we assume that the light source is a point, shininess is *inversely proportional* to the size of the highlight



- Highlight is view dependent.
  - The highlight on the object will "move on the object" when the viewer moves

Specular Term



- Define 4 unit vectors: N, L, R, V
- Then

$$\alpha = \cos^{-1}(R \cdot V)$$
$$R = 2(N \cdot L)N - L$$

- Phong Illumination Equation:

$$I_{Phong} = I_a k_a + f_{att} I_p k_d (N \cdot L) + \boxed{f_{att} I_p k_s (R \cdot V)^n}$$

- With *n*: shininess coefficient
  - *n* increases, highlights become smaller and sharper

Computing the Reflection Vector

$$|N'| = N \bullet L$$

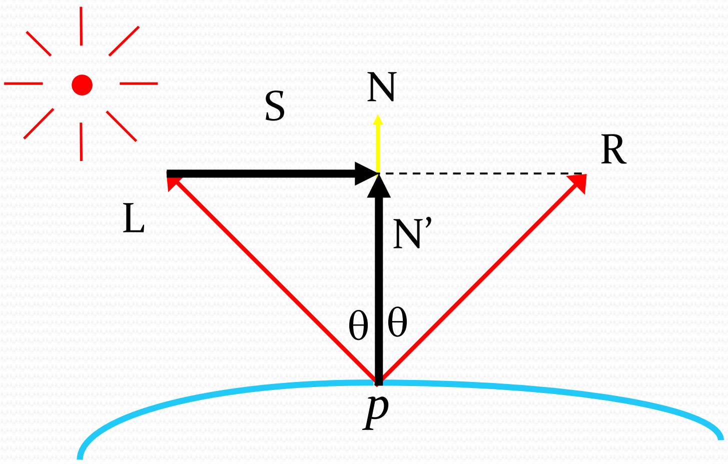
$$N' = (N \bullet L)\, N$$

$$S = N' - L$$

$$R = L + 2S$$

$$= 2N' - L$$

$$= 2(N \bullet L)\, N - L$$



Adjusting Parameters

Material Properties

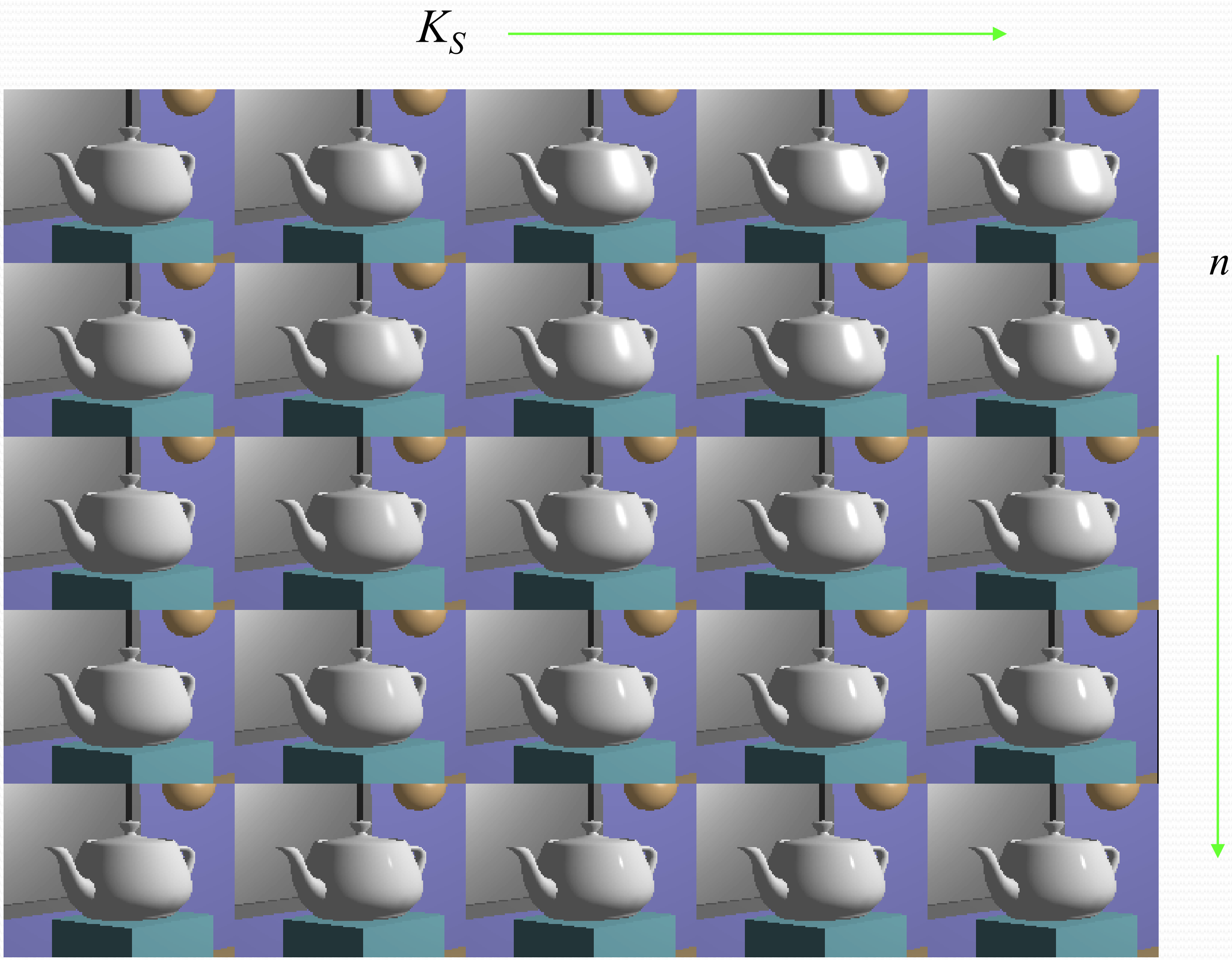
More types of light sources

Material Properties

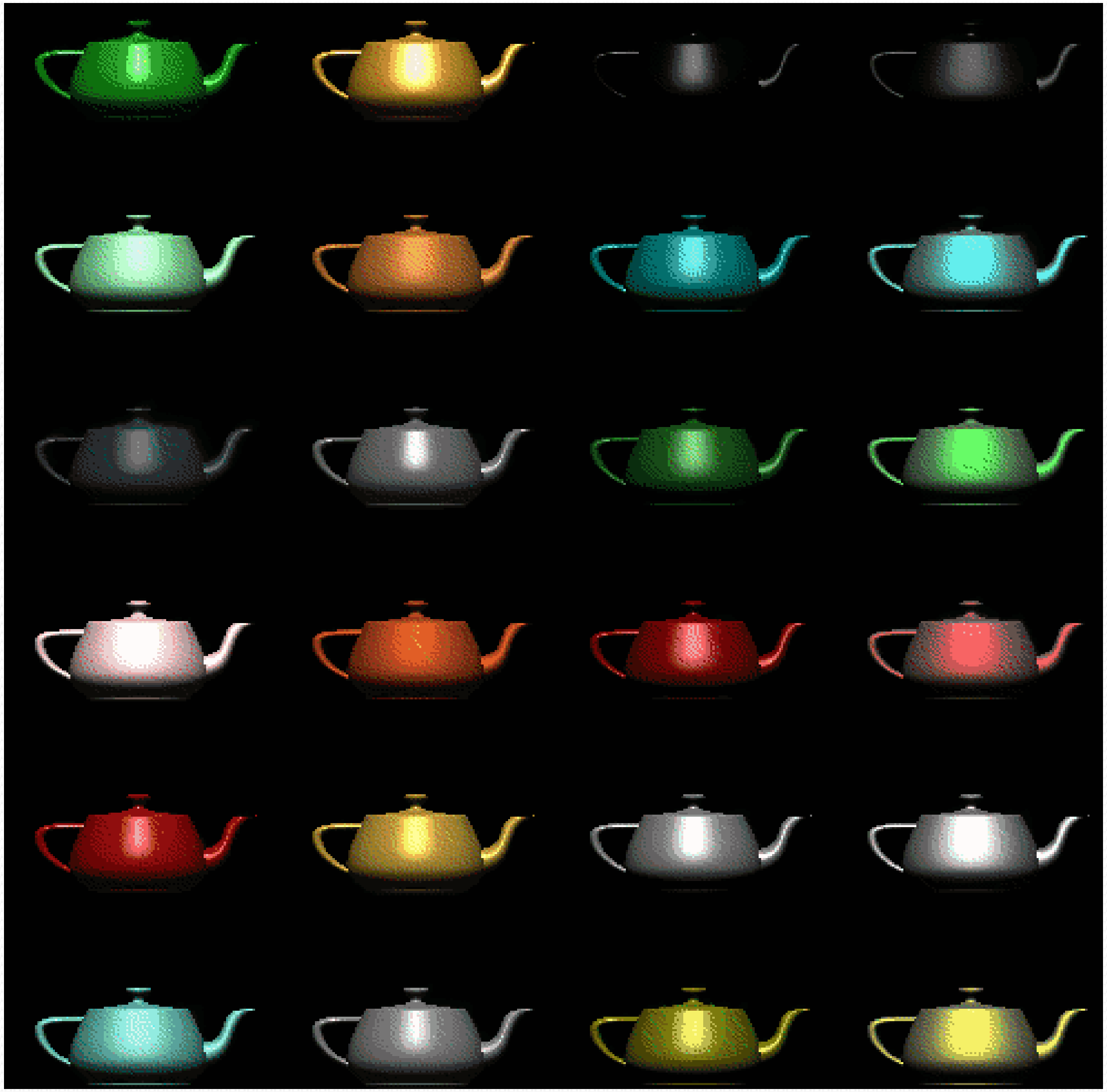
$$I_{Phong} = I_a K_a + f_{att} I_p K_d (N \cdot L) + f_{att} I_p K_s (R \cdot V)^n$$

- <u>*Material properties*</u> are modeled by *ambient, diffuse, specular reflection coefficients* K_a, K_d, K_s
- Each of them is a vector of 3 colors with values between 0 and 1
- The shininess coefficient *n* could be between 1 and ~500

Changing K_s and n



- Using Phong model, can shade object with all kinds of colors with different material properties
  - K_a, K_d, K_s



Multiple Lightings

- Original:

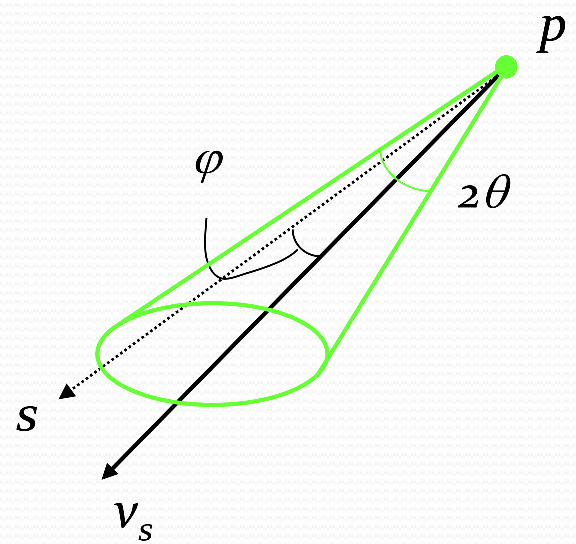
$$I_{Phong} = I_a K_a + f_{att} I_p [K_d (N \cdot L) + K_s (R \cdot V)^n]$$

- More than one light source

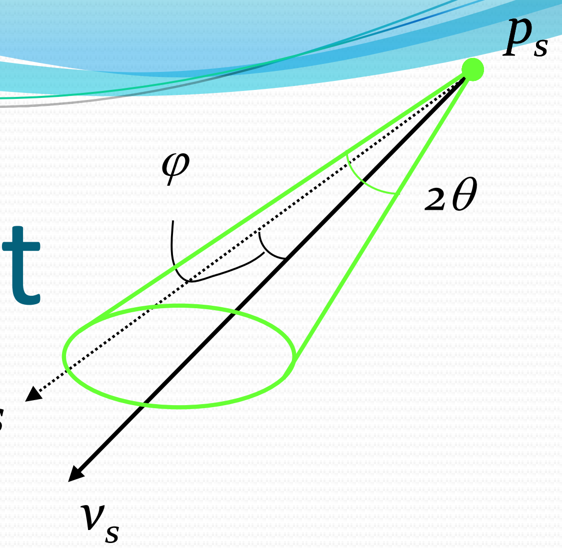
$$I_{Phong} = I_a K_a + \sum f_{att,i} I_{p,i} [K_d (N \cdot L) + K_s (R \cdot V)^n]$$

More Light Source: Spot Light

- Constructed from a point light source p by *defining a cone* with a limit of an angle of 2θ at which light from the source can be seen
  - θ : spotlight cutoff angle
- v_s : spotlight direction
- s : vector from p to a point
  on the illuminated surface
- $\varphi = \cos^{-1}(s \bullet v_s)$
- With s and v_s being unit vectors.



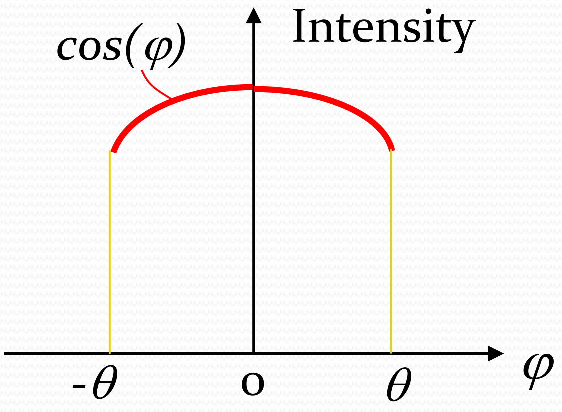
Light Source: Spot Light

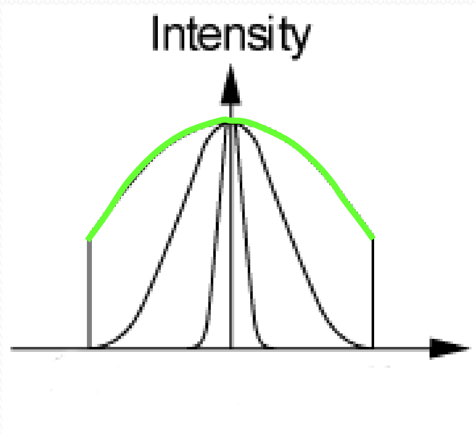


- To model more realistic spot lights…
  - Soft cutoff
- Light intensity received by the surface point (at the s direction) is

$$I_p' = I_p(\cos\varphi)^n = I_p(\, s \bullet v_s \,)^n$$

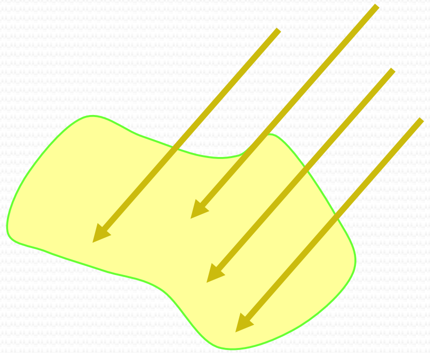
- n : spotlight exponent



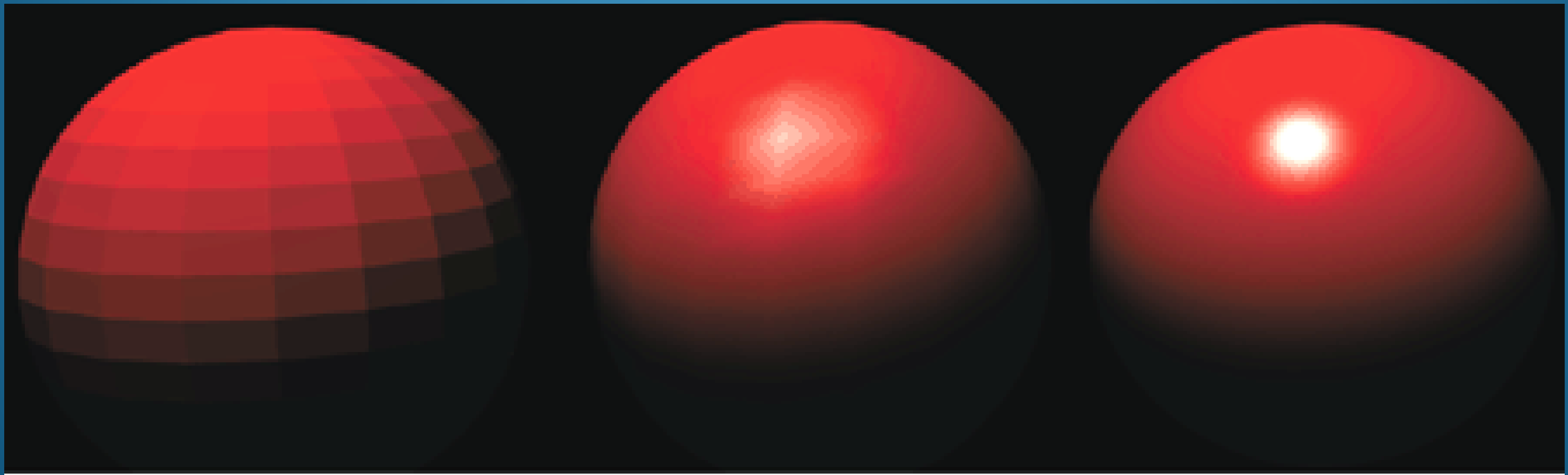


Light Source: Parallel Light

- If a light is too far away (e.g. the sun)
- We treat that every beam of light is in the same direction



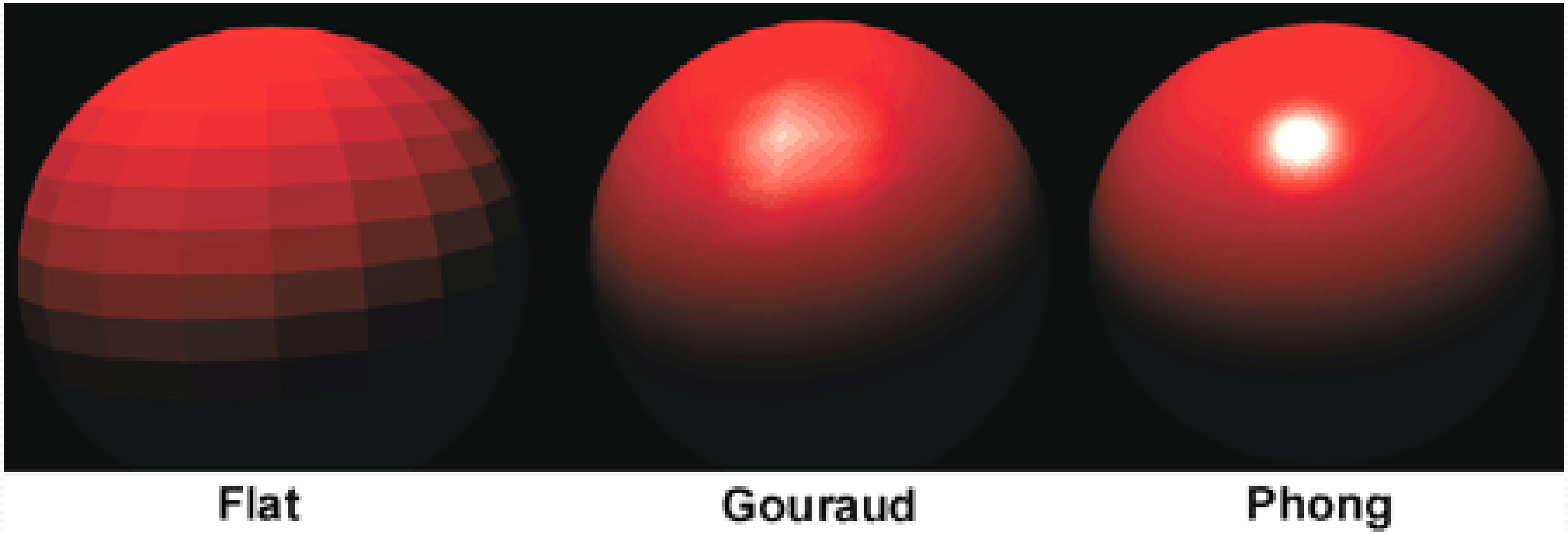
Shading



Flat	Gouraud	Phong

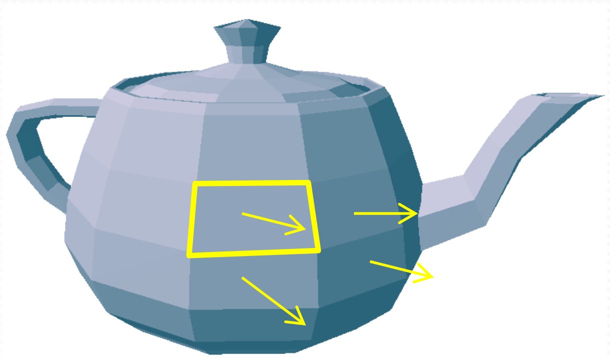
Three Types of Shading

- Flat shading
- Gouraud shading
- Phong Shading
    - (Don't mix up with Phong illumination Equation)

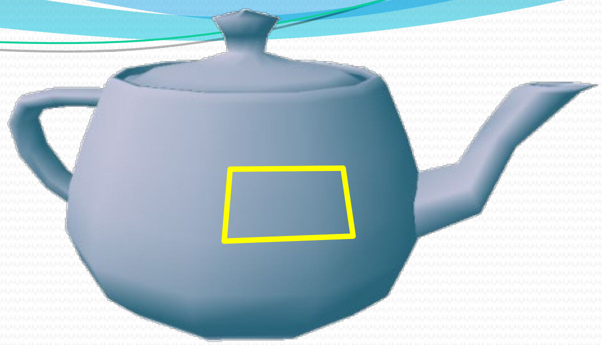


Flat Shading

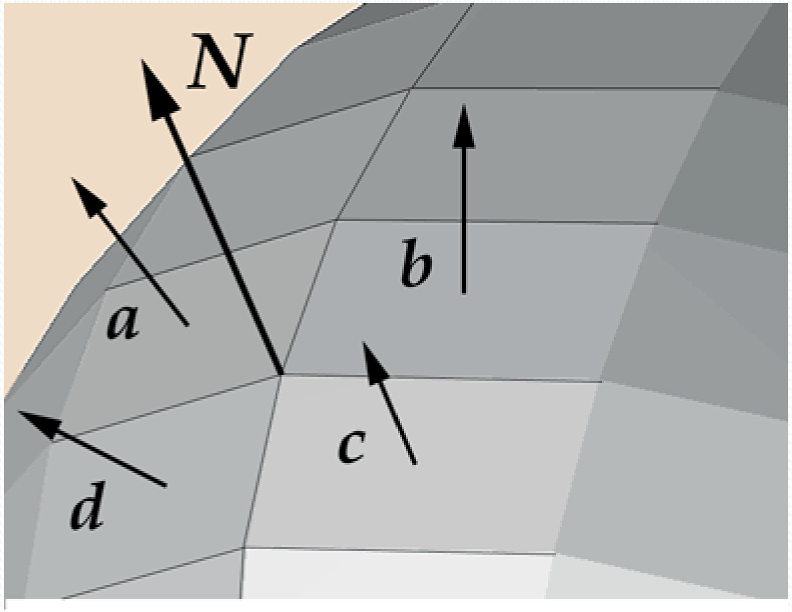
- For every polygon, we color the whole polygon with one color only
- Just pick <u>any</u> point on each polygon (e.g. the middle point, or a corner) and calculate its color by the normal of the polygon with PIE, and use this color for the whole polygon
- Distinctive color difference between each neighboring polygons
- OpenGL:
  - `glShadeModel(GL_FLAT);`



Gouraud Shading

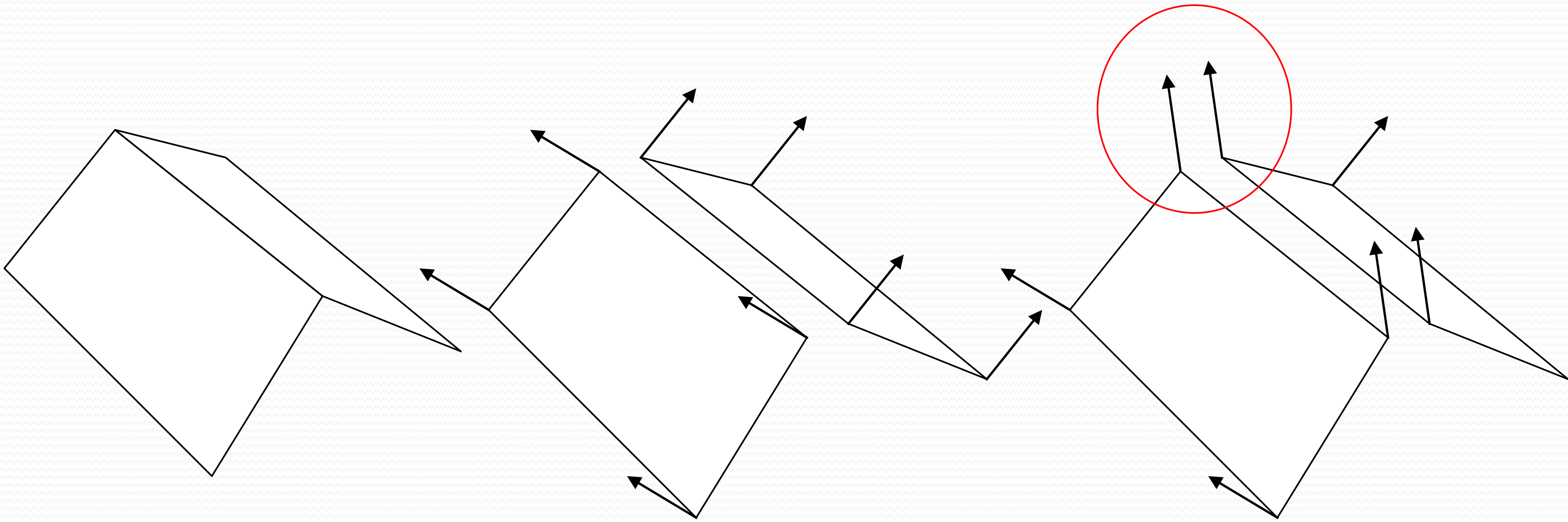


- For each **vertex**, compute the **average** normal vector of the polygons that share with the vertex
  - That’s why we need to know the connectivity/topology
  - Because we need to know which are the neighboring polygon
- Then compute the average vector as the vertex normal

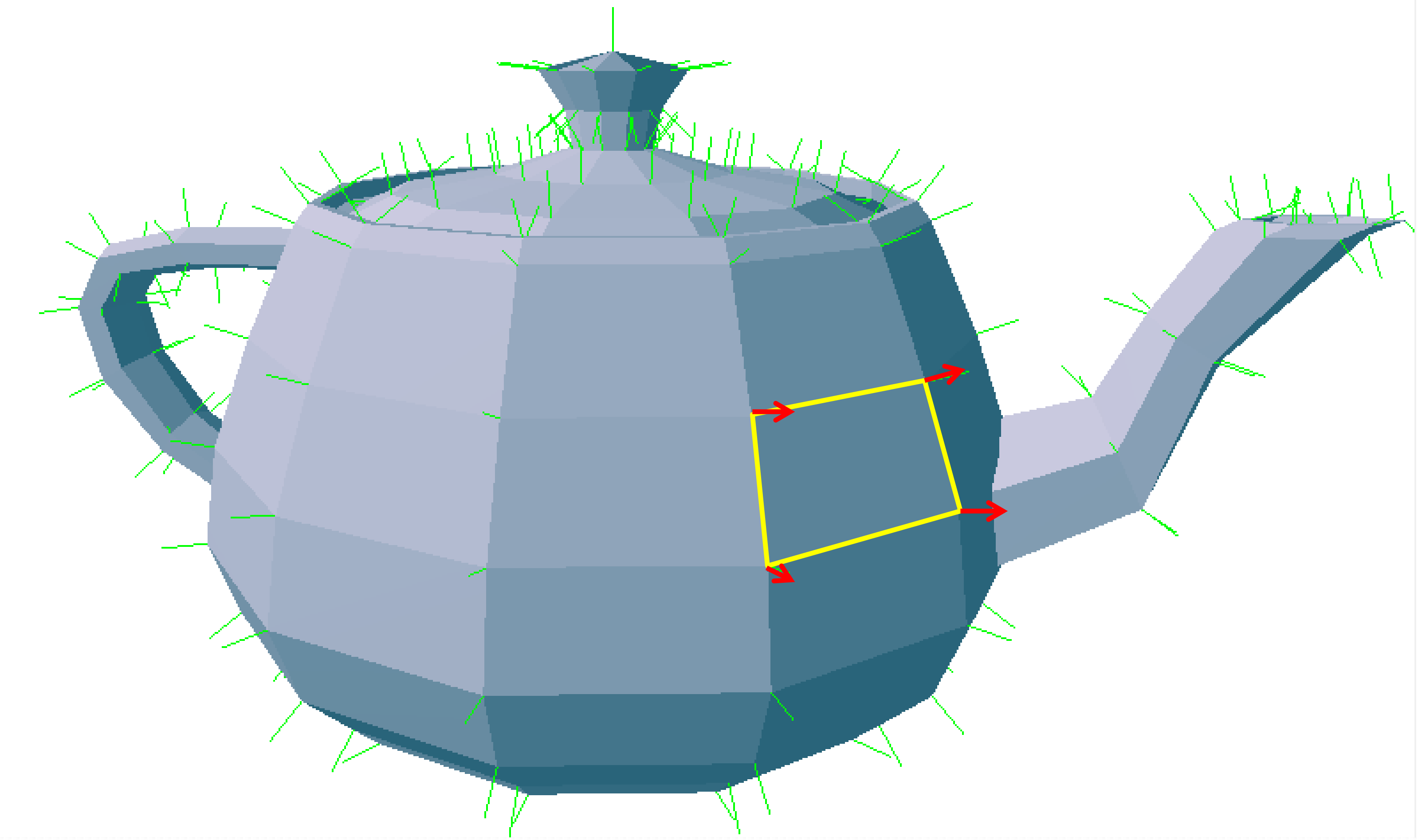


$$N = \frac{a+b+c+d}{4}$$

Normals at the shared edge

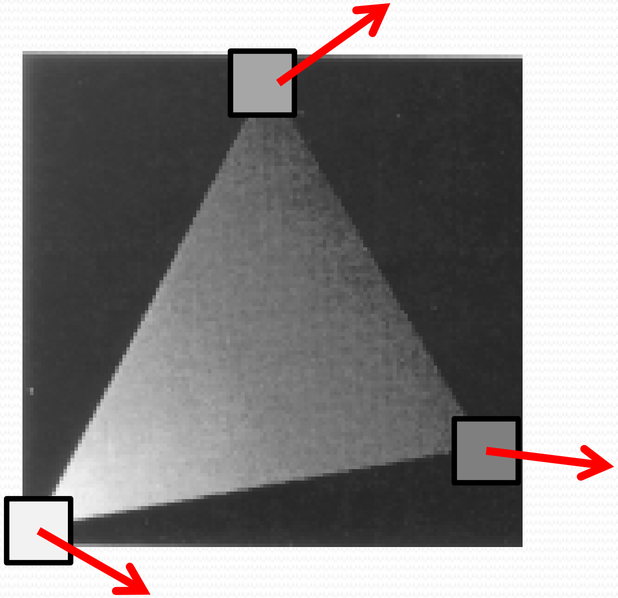


One Polygon with Different Vertex Normal Vectors

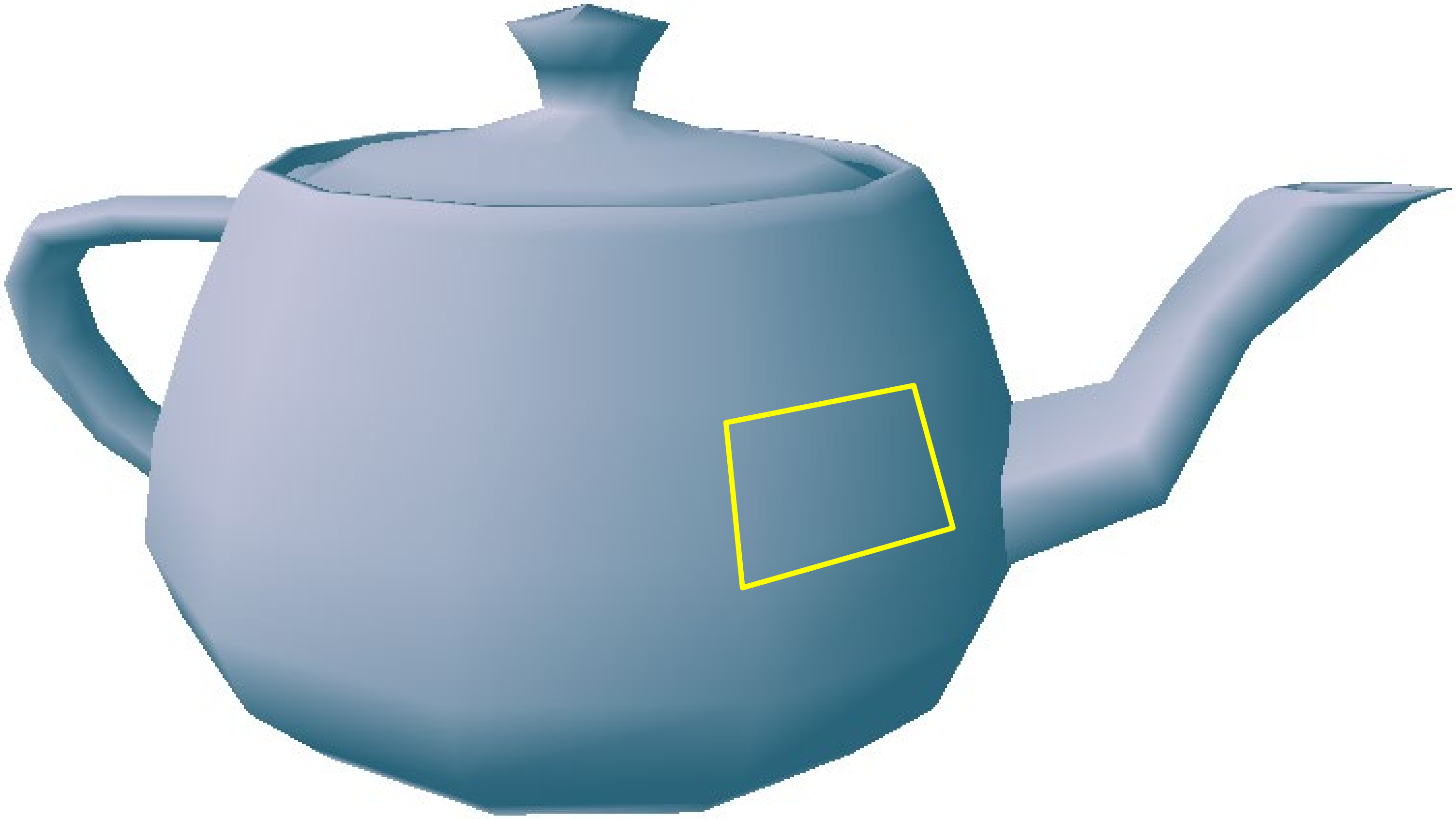


Gouraud Shading

- For each polygon, each vertex has a different vertex normal and position
  - Thus each vertex will have a different color by PIE
- Then, in SCA, interpolate the color as in Lecture 3



Gouraud Shading

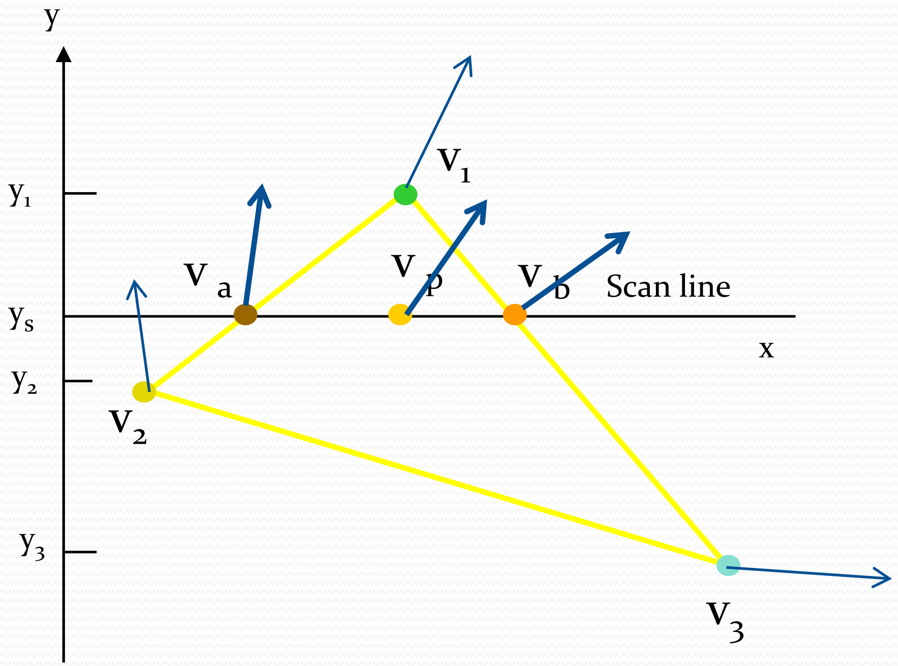


- OpenGL: **`glShadeModel(GL_SMOOTH)`**

Phong Shading

- Same as Gouraud Shading, every vertex of a polygon has a different vertex normal vector
- Except that, in Phong Shading, we do NOT compute the colors of the vertices for interpolation
- Instead, for each pixel in SCA, we interpolate the **normal vectors**
- Finally, for each pixel, we have it's screen coordinate (with z value also)
  - We inversely transform it back to the world coordinate to compute it's color by PIE

Vector Interpolation



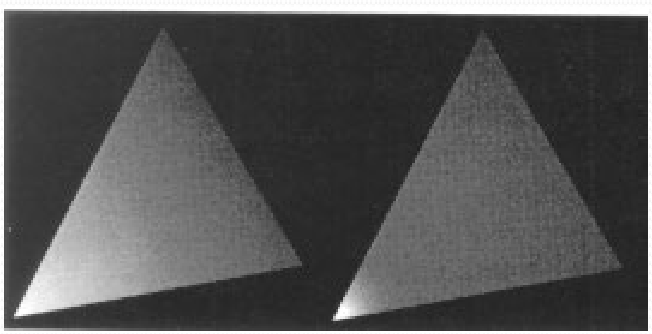
- Then, transform p (and the normal vector v_p) back to the world coordinate to compute the color/intensity

Gouraud shading

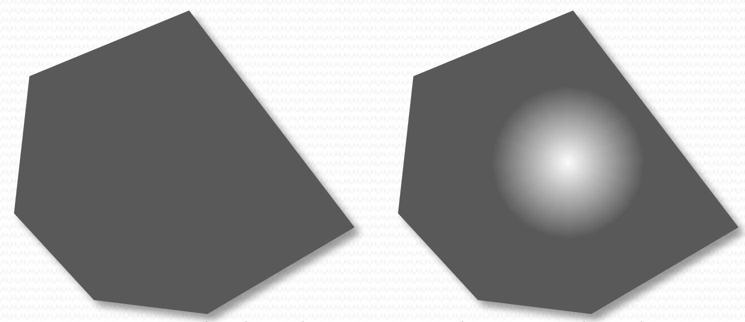
Phong shading

Gouraud vs Phong Shadings

- Highlights are produced more faithfully with Phong shading
  - Gouraud shading produces only "linear interpolation" of colors
  - Gouraud shading may even miss the highlight



Gouraud shading Phong shading



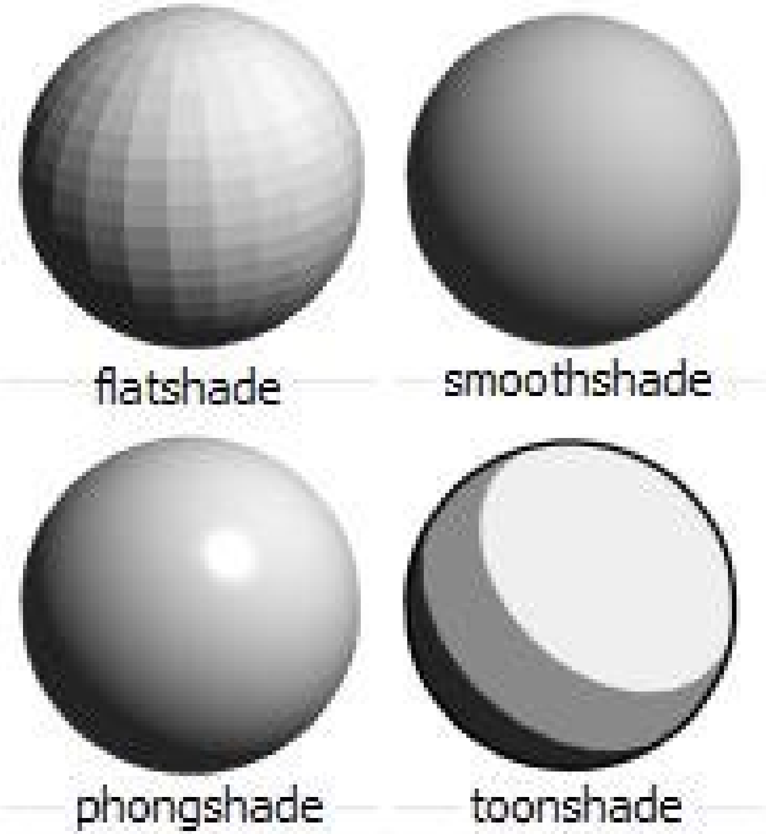
Gouraud shading Phong shading

- But… too bad, OpenGL does not support Phong Shading
  - SL does support, though…

Non-realistic Shading

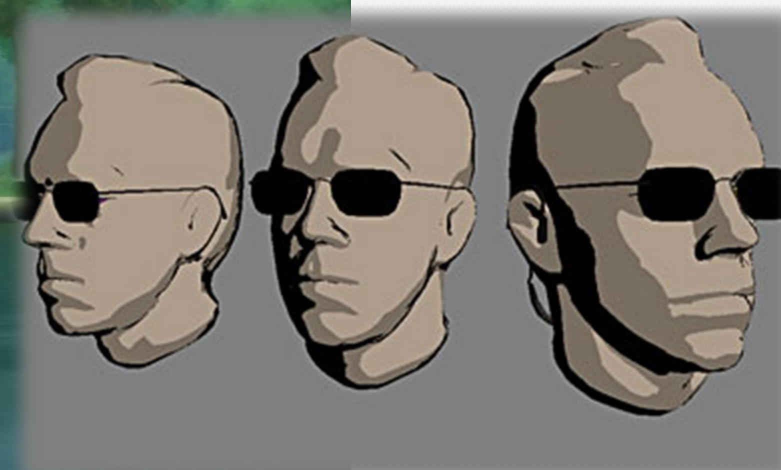
Extra: Toon Shading

- Or “Cel-shading”




171
60

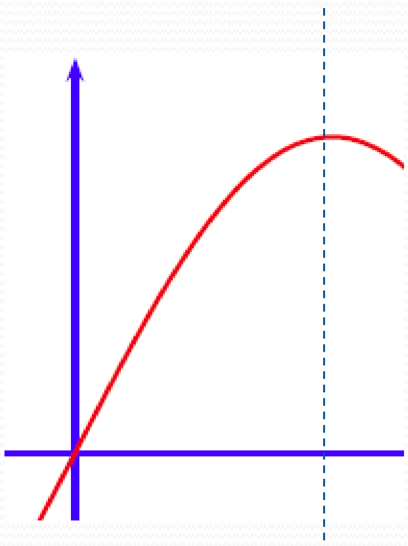

68
Naruto
L1
R1



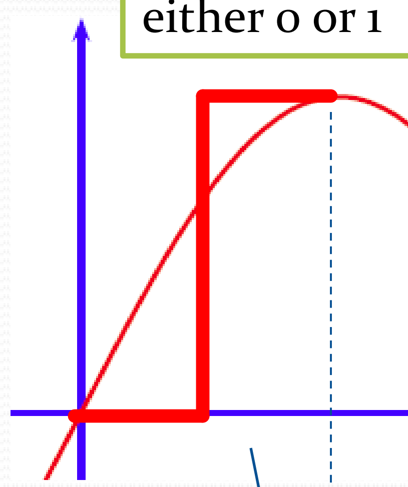
Toon Shading

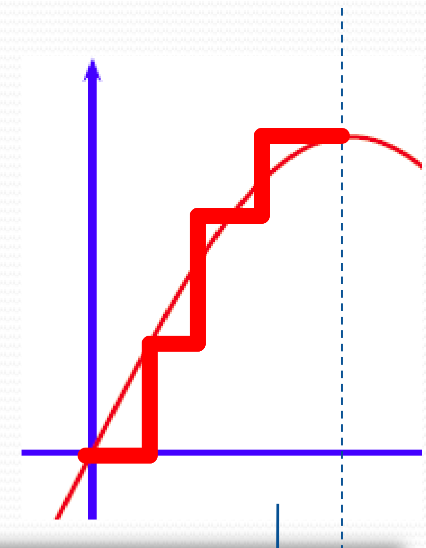
$$I_{Phong} = I_a K_a + f_{att} I_p K_d (N \cdot L) + f_{att} I_p K_s (R \cdot V)^n$$

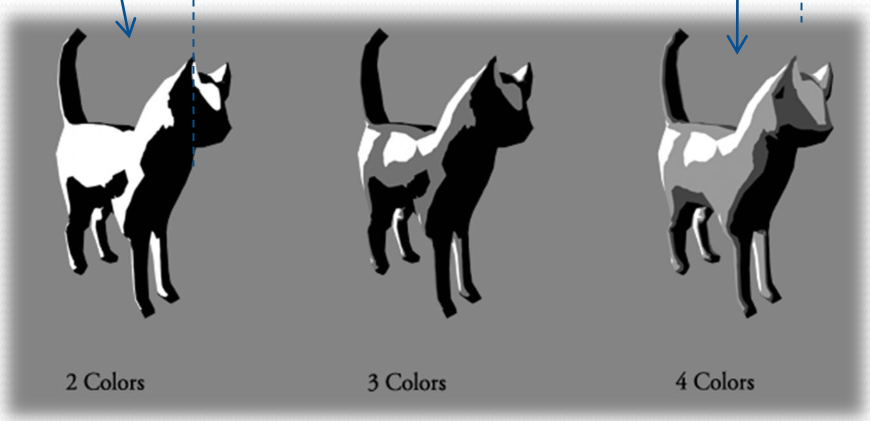
E.g. round up to either 0 or 1



Smooth Shading





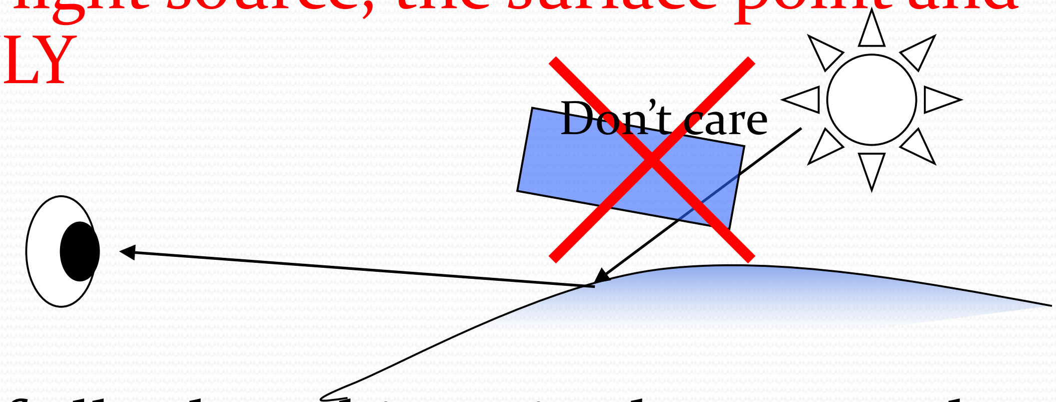


Problems with Interpolated Shading with Polygonal Models

- Non-global effects
  - No shadow
- Polygonal silhouette
- Orientation dependence
- Shared vertices
- Misleading vertex normals

Non-global Illumination

- ONLY consider the light source, the surface point and the viewer DIRECTLY



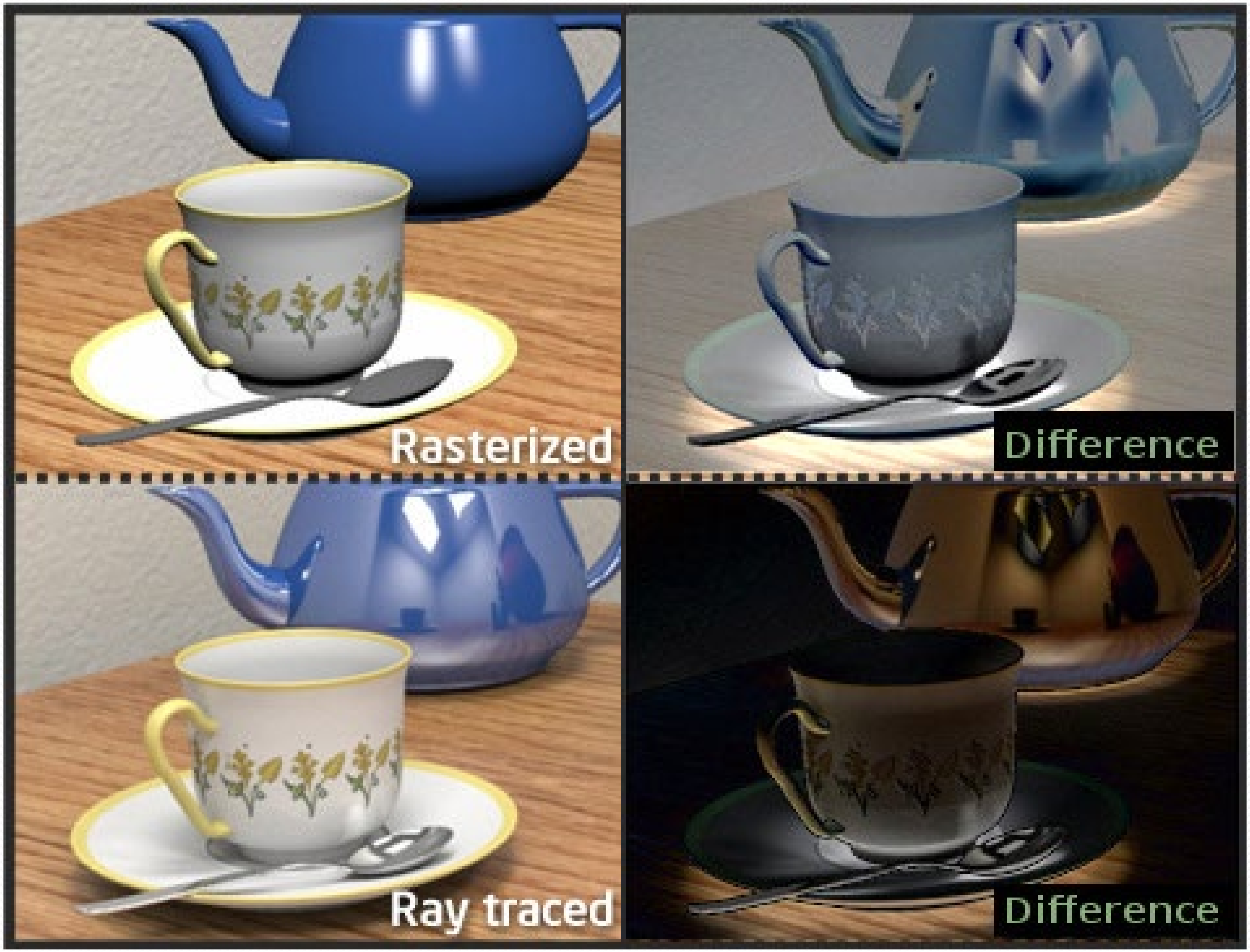
- Ignore the effects of all other objects in the scene when considering a particular surface element
- Trade-offs - pay a price in <u>lost of realism</u> for rendering speed
  - i.e. interesting light effects, e.g. shadows, inter-object
  - reflection, refraction, etc will be lost.

Global vs Non-global Illumination

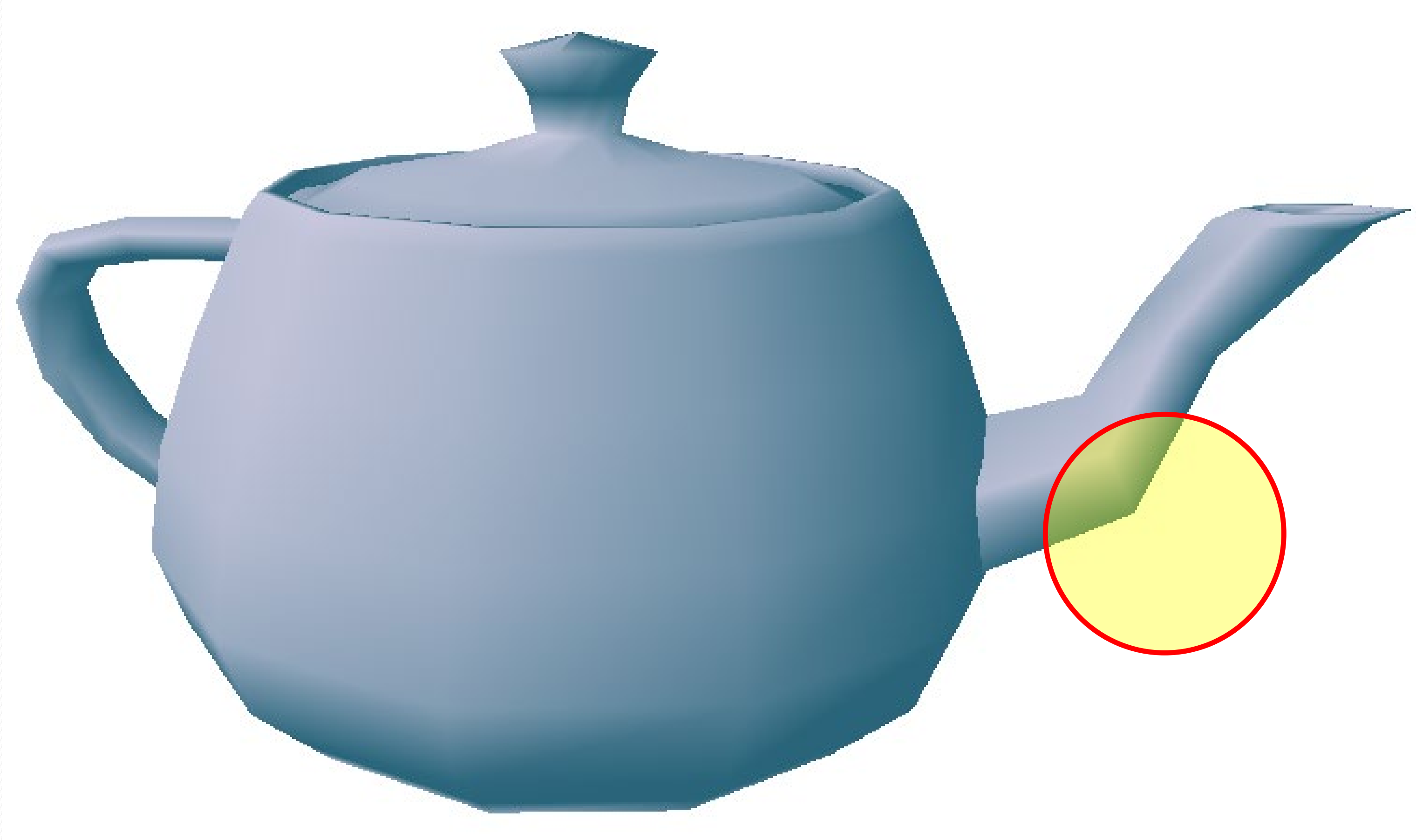
- Global (Ray Tracing)
  - More photorealistic/complex
    - Compute ALL type of physical simulation of light interactions
  - Slow
  - E.g. CG movies
- Non-global (e.g. Phong Shading)
  - ONLY consider
    - The light source, the surface point and the viewer DIRECTLY
  - Faster but not so realistic
    - E.g. no shadow/reflection
  - E.g. Real time 3D games

Phong Shading

Ray Tracing

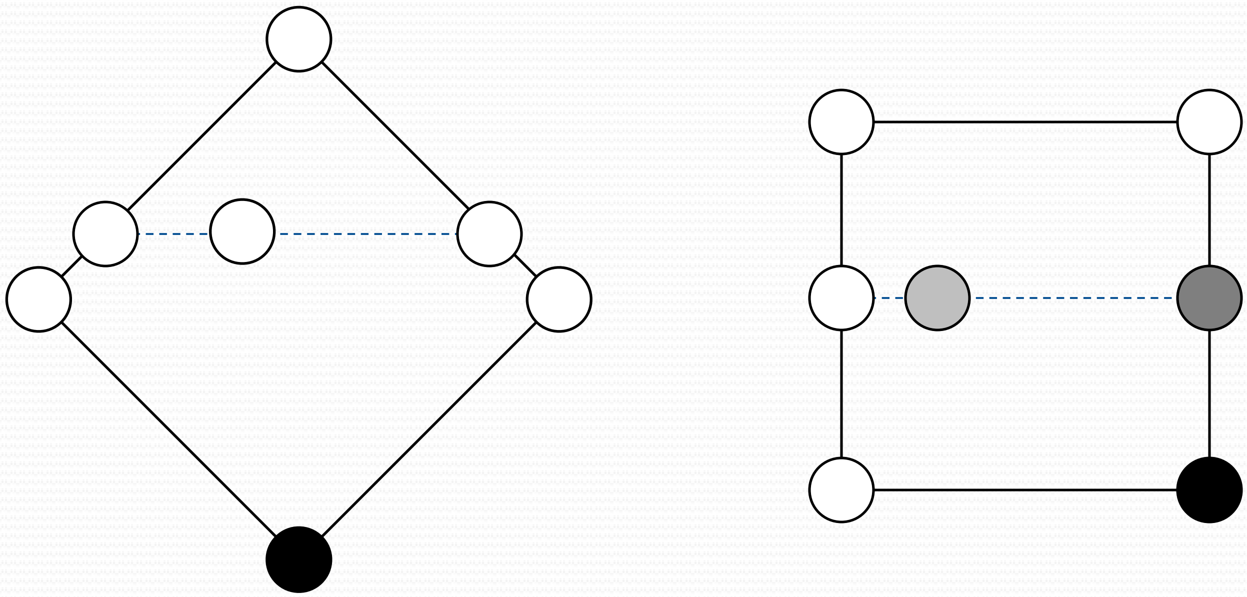

Rasterized
Difference
Ray traced
Difference

Polygonal Silhouette



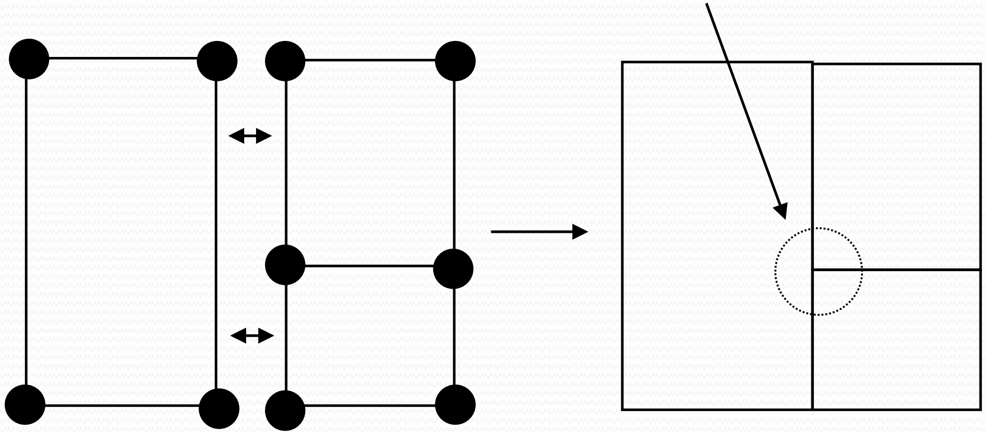
Problems with Interpolated Shading

- Orientation dependence

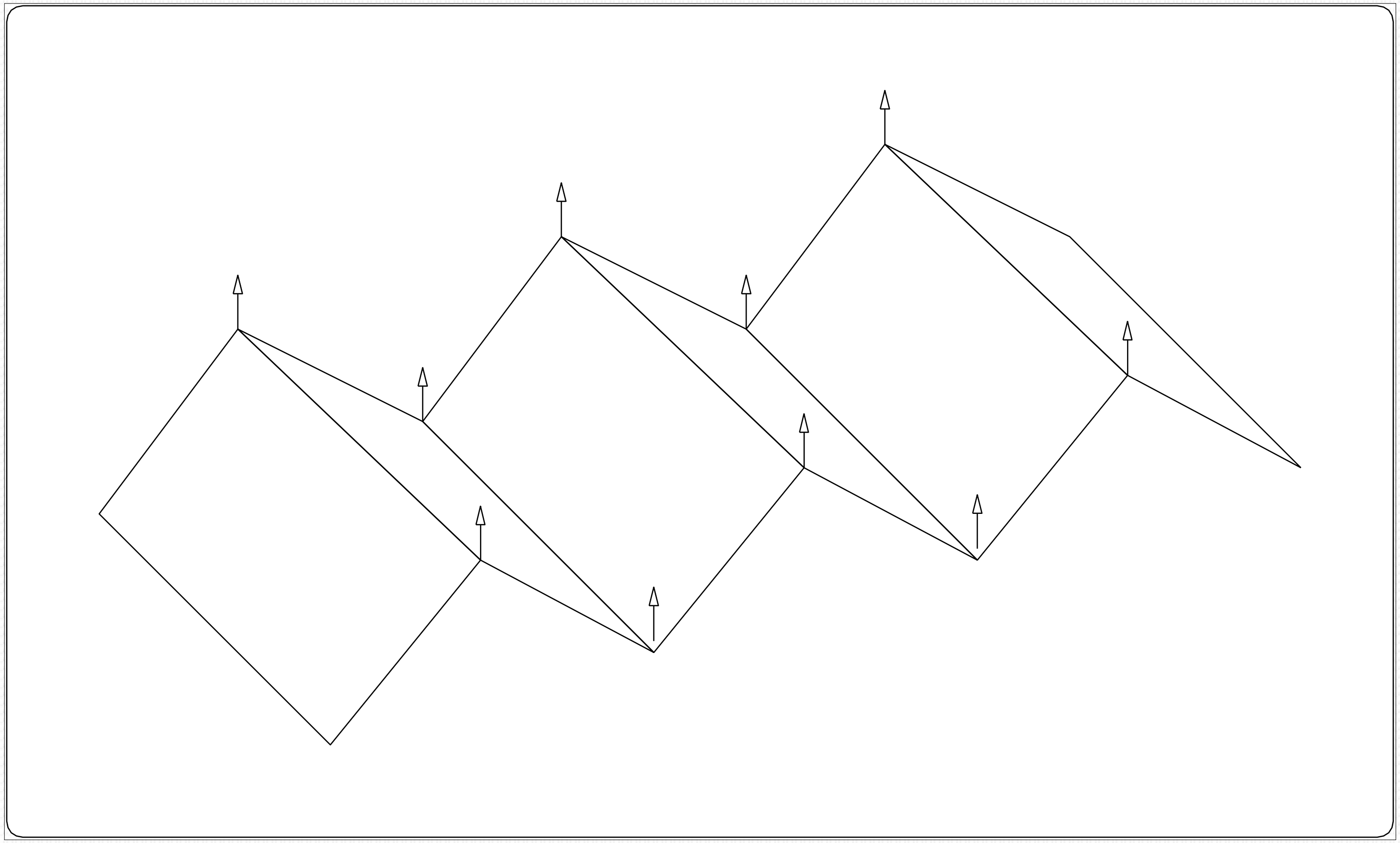


Problems with Interpolated Shading

- Shared vertices



Disadvantage (Misleading Normal Vectors)



Bump Mapping

Bump Mapping

- Similar to Texture mapping
- Instead of mapping colors onto a surface, we map distortion of normals
- More details later.....