NUSRI Summer Programme 2016

# RI3004A
# 3D Graphics Rendering

## Lecture 7
## Illumination & Shading

School *of* Computing
National University *of* Singapore

# Objectives

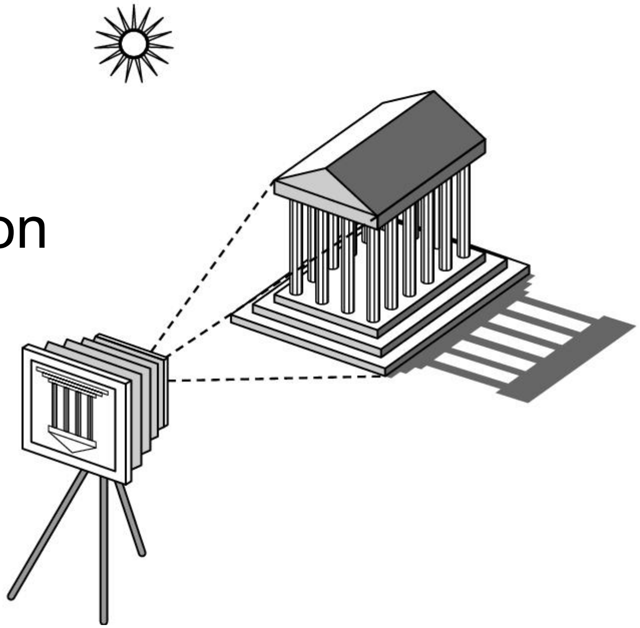
- Illumination
  - Given a point on the surface, the light source, and the viewpoint
  - How to compute its color

- Shading
  - Given a polygon and its rasterization
  - How to compute the color on each fragment of the polygon

# Illumination

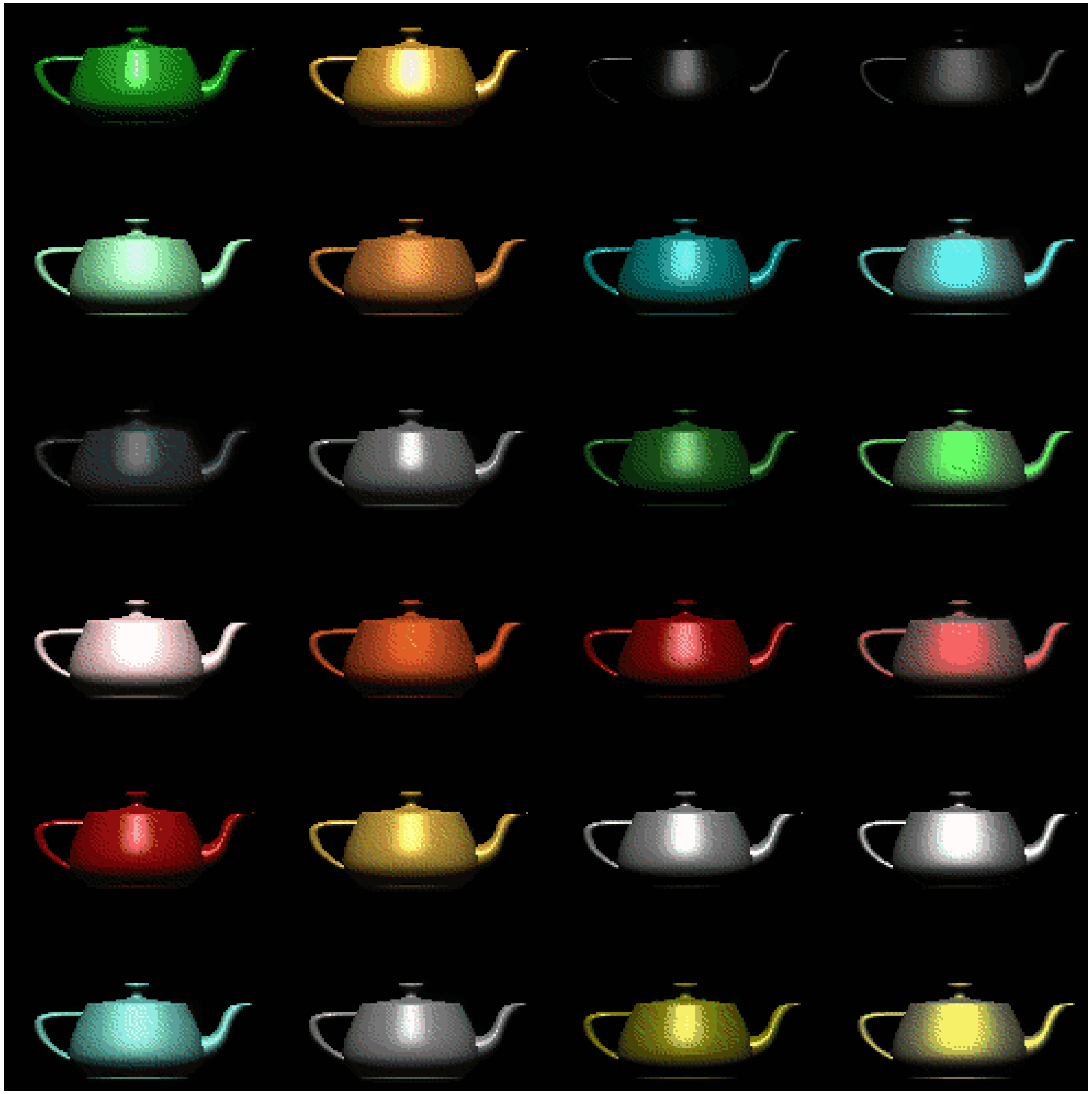
# Elements of Image Formation

- Objects ✓
  - Define objects' geometry
- Viewer ✓
  - Set up view transformation, projection and viewport transformation
- Light source(s)
  - This lecture
- Materials
  - This lecture



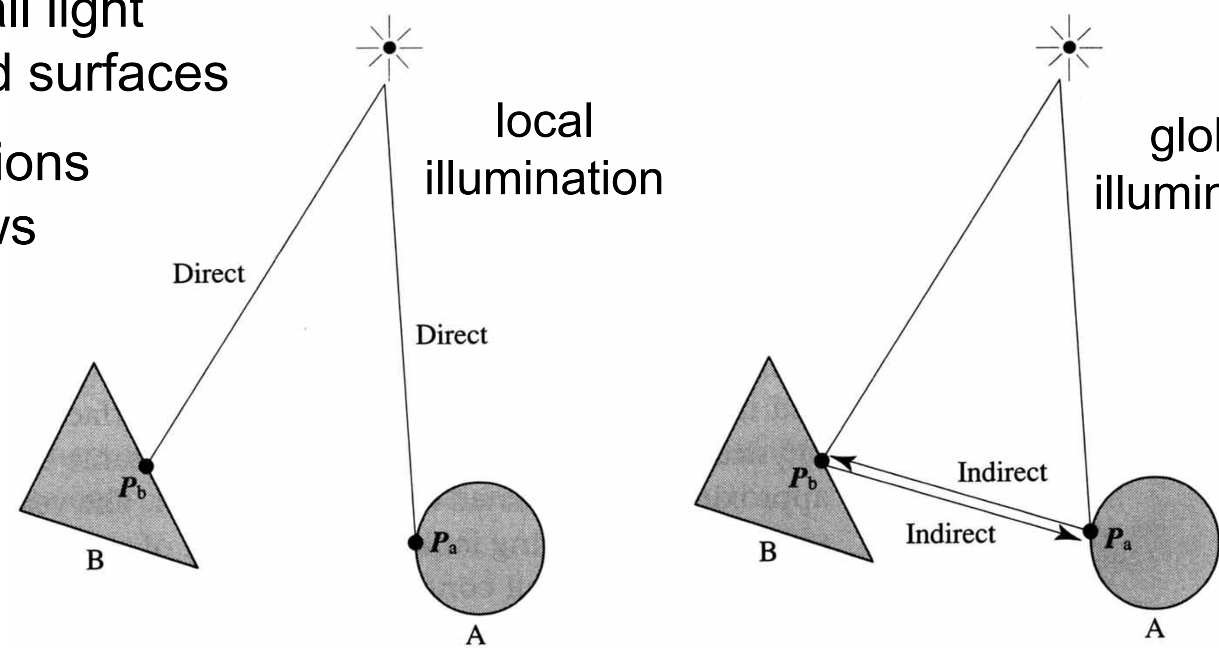
# Goal

- We want to "color" the object in such a way to better show its shape and material



# Local Reflection vs Global Illumination

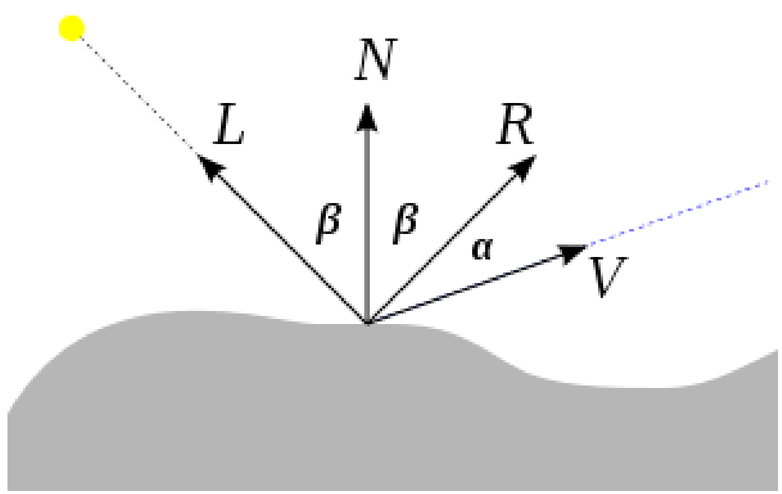
- Local reflection
  - Considers relationship between a **light source**, a **single surface point**, and a **view point**
  - No interaction with other objects
- Global illumination
  - Considers all light sources and surfaces
  - Inter-reflections and shadows



# Illumination Model

- Phong Illumination Equation (PIE)
  - Given surface point + light source + viewer
  - Compute color at surface point using

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



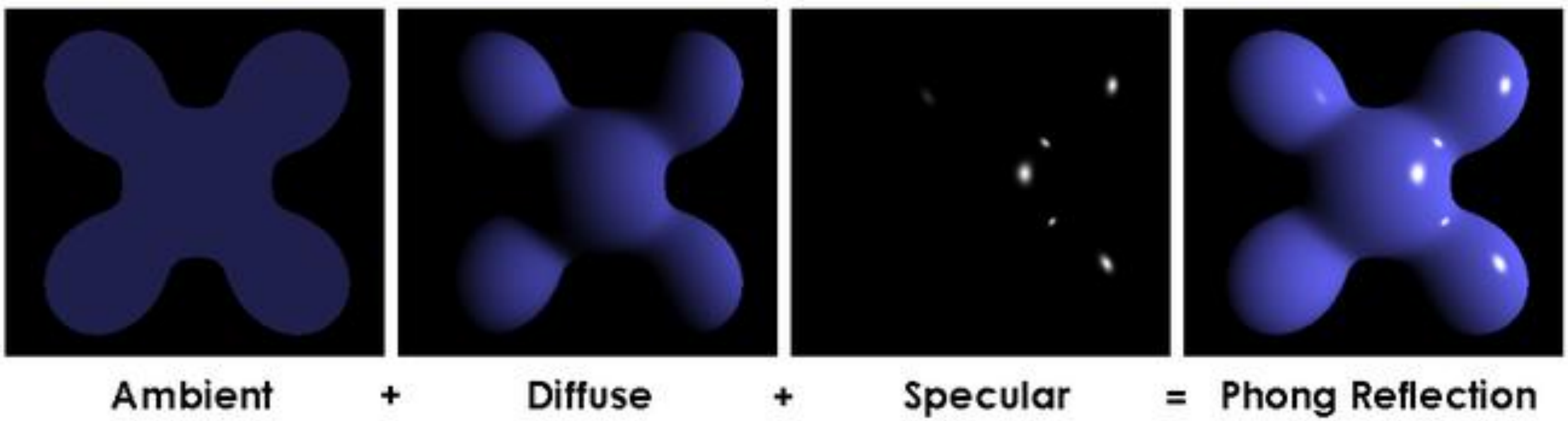
# Phong Illumination / Reflection Model

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

# Phong Illumination Equation

- The 3 terms in the PIE

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



# Ambient Light

- 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 Material

- Then, for different surfaces, how can they appear as different colors?
- For **each surface**, we specify its own ambient material property

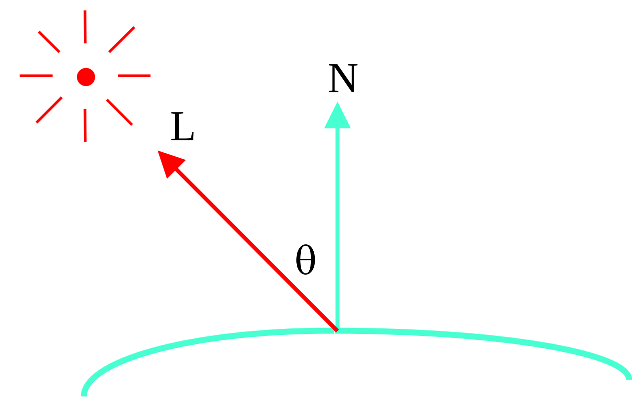
$$k_a = \begin{bmatrix} k_{ar} \\ k_{ag} \\ k_{ab} \end{bmatrix}$$

- Referring to the PIE, the ambient color of the surface is

$$I_a k_a = \begin{bmatrix} I_{ar} k_{ar} \\ I_{ag} k_{ag} \\ I_{ab} k_{ab} \end{bmatrix}$$

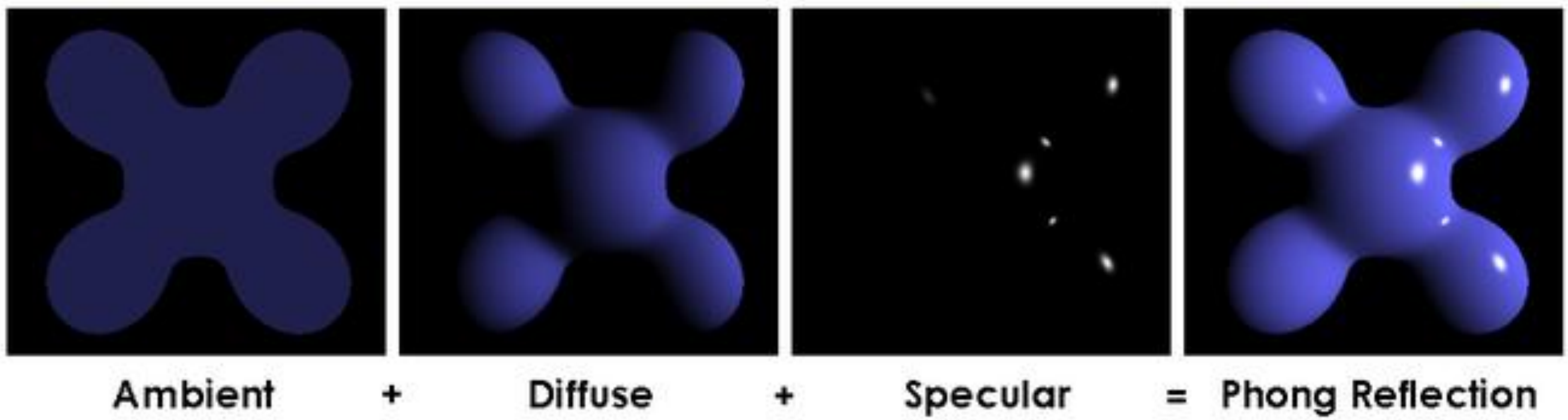

Ambient Lighting Only

# Diffuse Term of PIE




- The diffuse term gives color to the surface point according to the light position and the surface normal

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

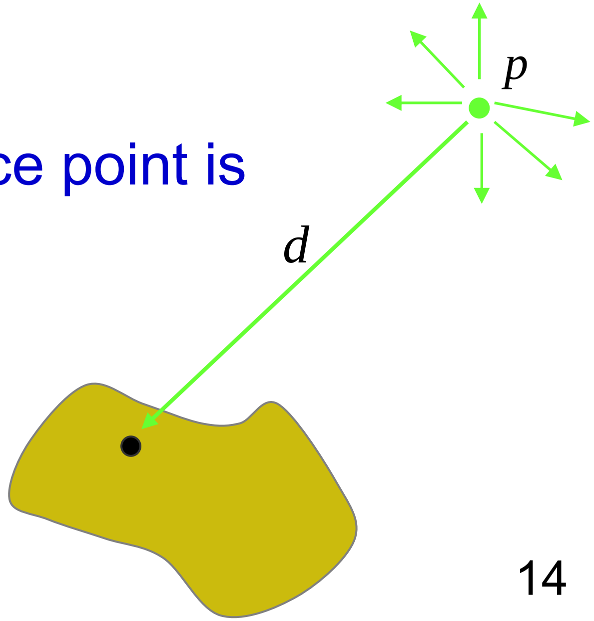


# Point Light Source

- 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 farther 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

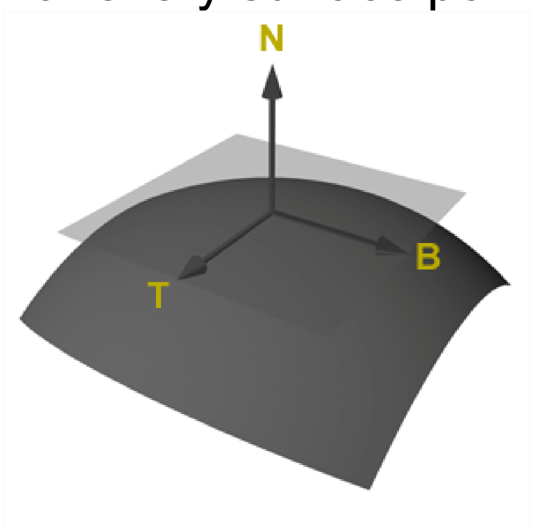


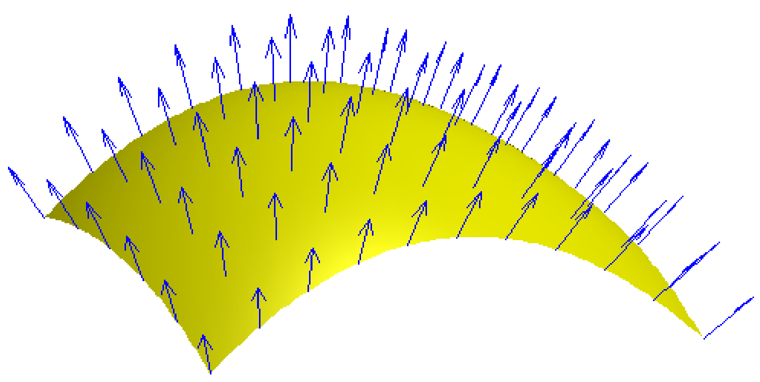
# Surface Normal

- For a triangle
  - The triangle, with vertices $A$, $B$ and $C$, spans a plane
  - There is a normal vector $N$ that is perpendicular to the plane

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

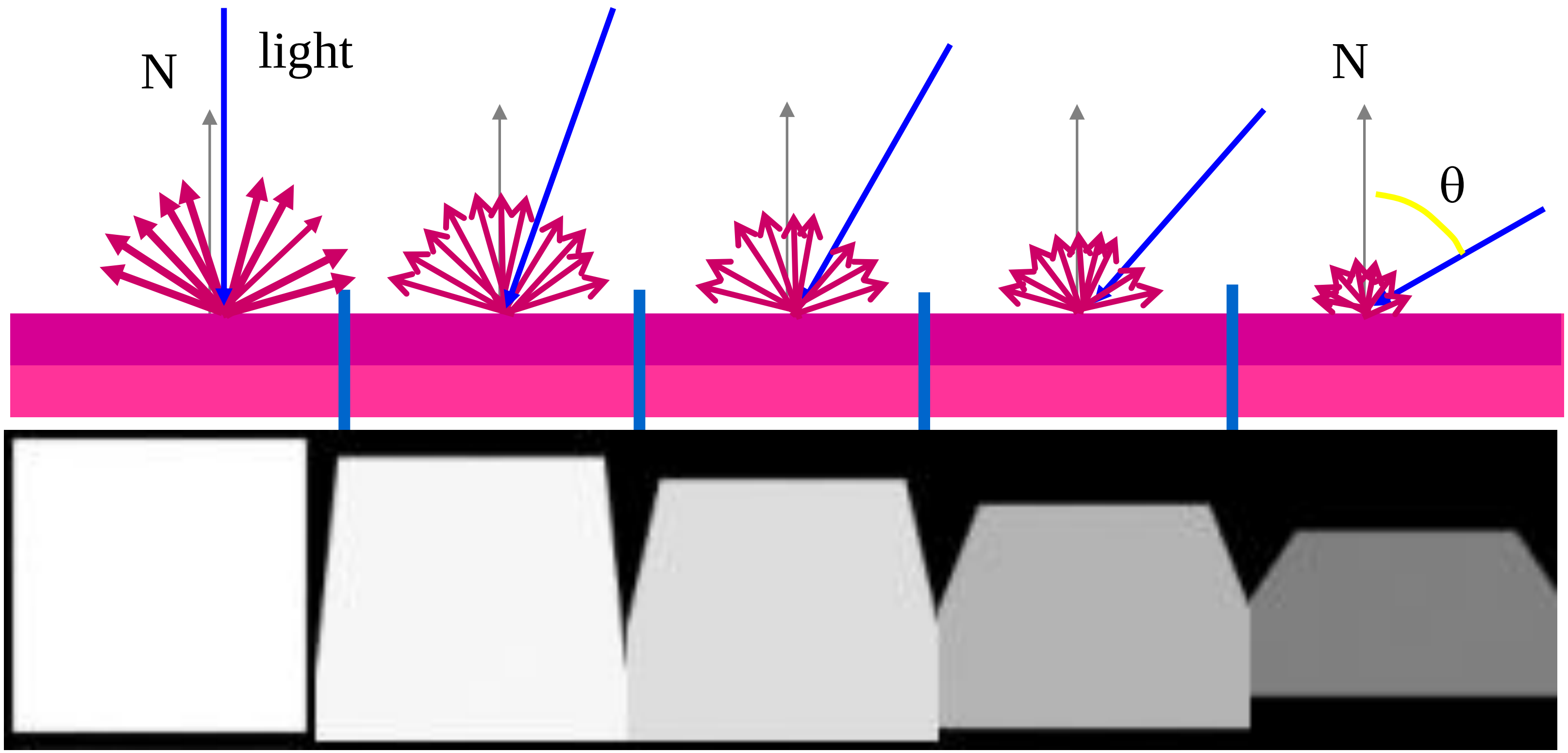
  - (Note that the normal vector has two choices, $N$ or $-N$)
- For a curved surface
  - For every surface point, there is a plane “parallel” to it



# Diffuse Reflection

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



# Diffuse Term of PIE

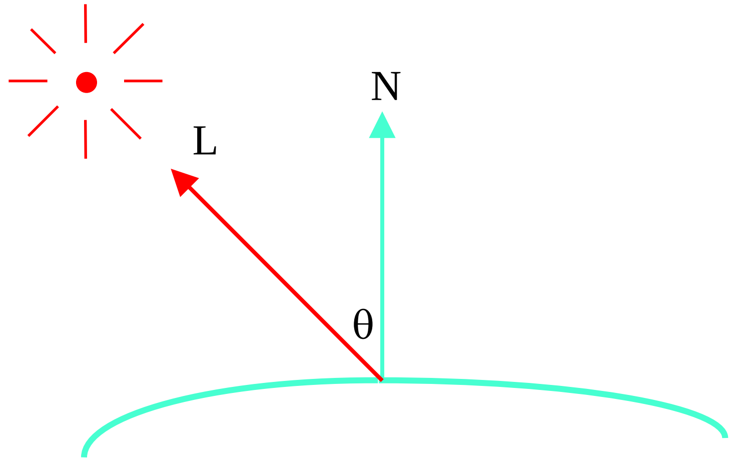
The direction **unit** vector from the surface point to the light source

- 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 surface, usually it is different from $k_a$

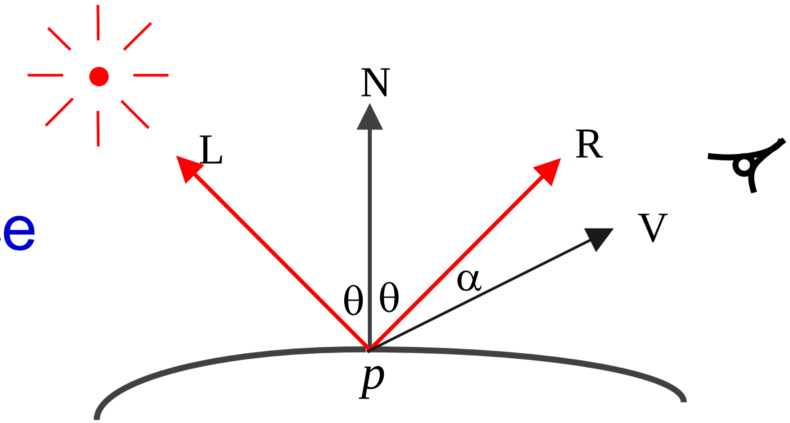
Remember to normalize every directional vector



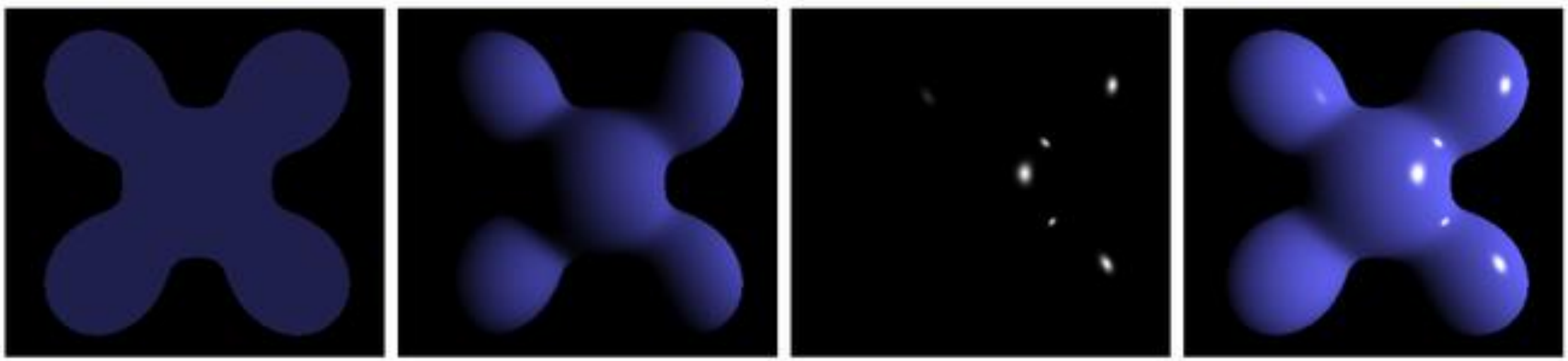

+ Diffuse Lighting

# Specular Term of PIE

- Adding highlights to shiny surface



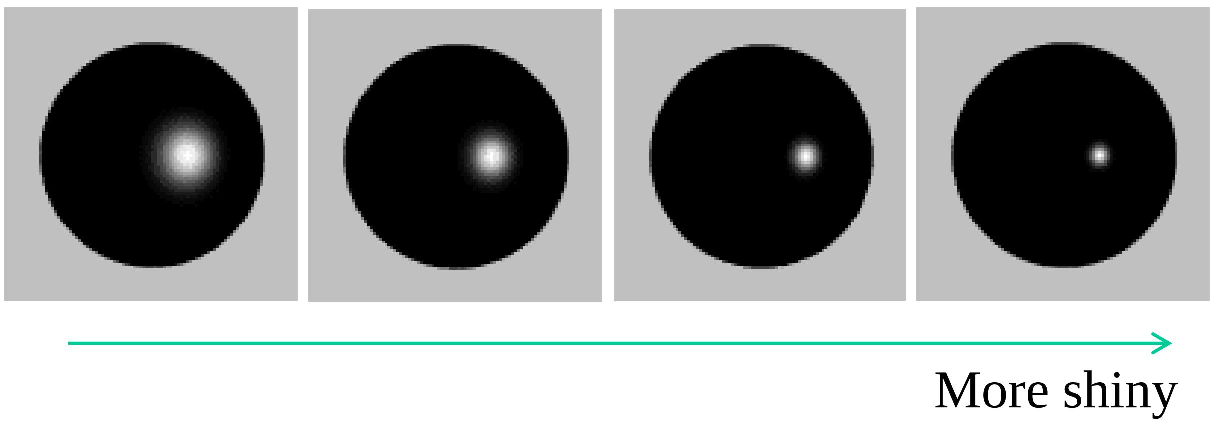

$$I_{Phong} = I_a k_a + f_{att} I_p k_d (N \cdot L) + \boxed{f_{att} I_p k_s (R \cdot V)^n}$$



Ambient + Diffuse + Specular = Phong Reflection

# Specular Reflection

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

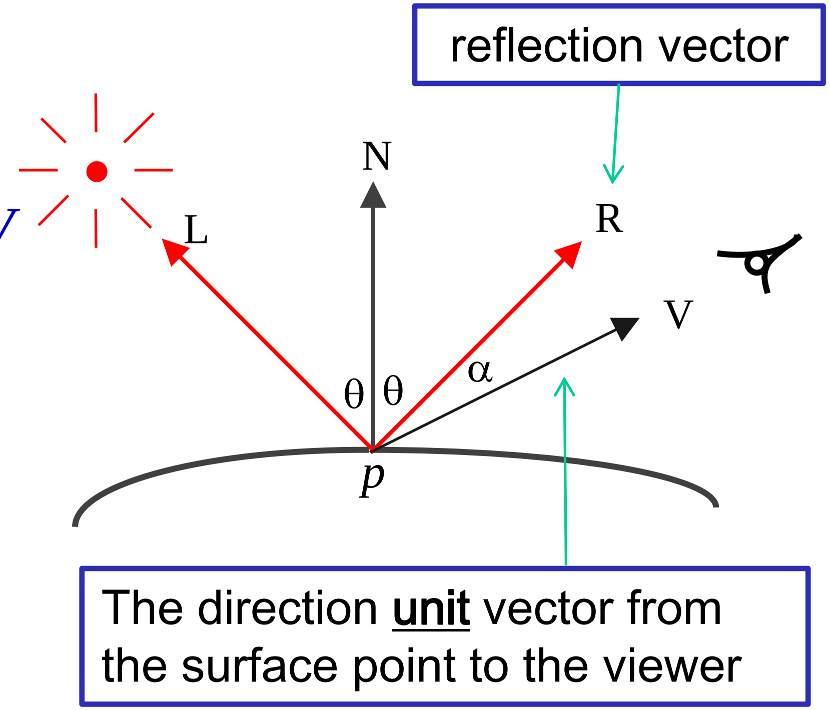



- Highlight is *view dependent*
  - The highlight on the object will “move on the object” when the viewer moves

# Specular Term of PIE

- 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

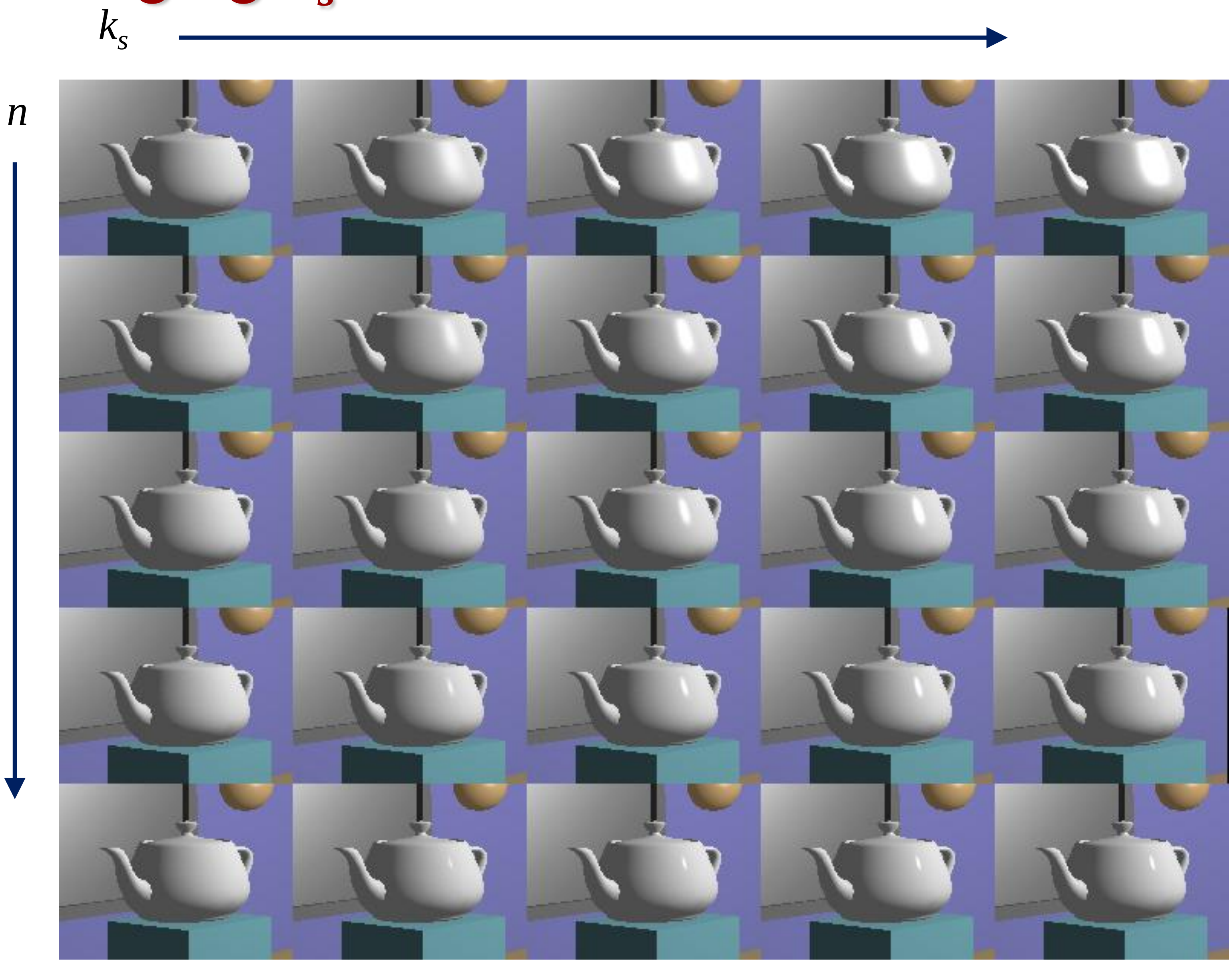

+ Specular Lighting

# Material Properties

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

- *Material properties* of a surface 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 $n$ can have value from 1 to 128 in OpenGL

# Changing $k_s$ and $n$



# Multiple Light Sources

- For one light source

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

- For multiple light sources

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

# Illumination in OpenGL

# Specifying Vertex Normal Vectors

- In OpenGL the normal vector is part of the state
  - Usually set just before specifying each vertex
- Set by `glNormal*()`
  - `glNormal3f(x, y, z);`
  - `glNormal3fv(p);`
- Usually we want to set the normal to have unit length so cosine calculations are correct
  - Length can be affected by transformations
  - Note that scaling does not preserved length
  - `glEnable(GL_NORMALIZE)` allows for autonormalization at a performance penalty

# Enabling Lighting Computation

- Shading calculations are enabled by
  - `glEnable(GL_LIGHTING)`
  - Once lighting is enabled, glColor() ignored
- Must enable each light source individually
  - `glEnable(GL_LIGHTi)` `i` = 0, 1, 2, …
- Can choose light model parameters
  - `glLightModeli(parameter, GL_TRUE)`
    - `GL_LIGHT_MODEL_LOCAL_VIEWER` do not use simplifying distant viewer assumption in calculation
    - `GL_LIGHT_MODEL_TWO_SIDED` shades both sides of polygons independently

# Defining a Point Light Source

- For each light source, we can set an RGBA for the diffuse, specular, and ambient components, and for the position

```
GLfloat diffuse0[] = {1.0, 0.0, 0.0, 1.0};
GLfloat ambient0[] = {1.0, 0.0, 0.0, 1.0};
GLfloat specular0[] = {1.0, 0.0, 0.0, 1.0};
GLfloat light0_pos[] = {1.0, 2.0, 3,0, 1.0};

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightfv(GL_LIGHT0, GL_POSITION, light0_pos);
glLightfv(GL_LIGHT0, GL_AMBIENT, ambient0);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular0);
```

# Global Ambient Light

- Ambient light depends on color of light sources
  - A red light in a white room will cause a red ambient term that disappears when the light is turned off
- OpenGL also allows a global ambient term that is often helpful for testing
  - **`glLightModelfv(GL_LIGHT_MODEL_AMBIENT, global_ambient)`**

# Moving Light Sources

- Light sources are geometric objects whose positions or directions are affected by the model-view matrix
- Depending on where we place the position (direction) setting function, we can
  - Move the light source(s) with the object(s)
  - Fix the object(s) and move the light source(s)
  - Fix the light source(s) and move the object(s)
  - Move the light source(s) and object(s) independently

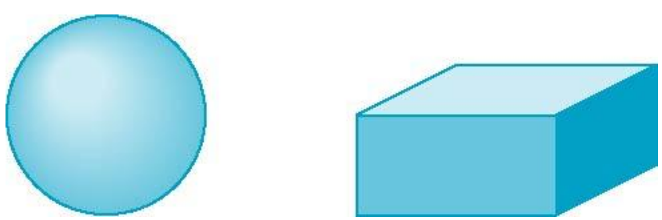
# Material Properties

- Material properties are also part of the OpenGL state and match the terms in the Phong model
- Set by **`glMaterialfv()`**

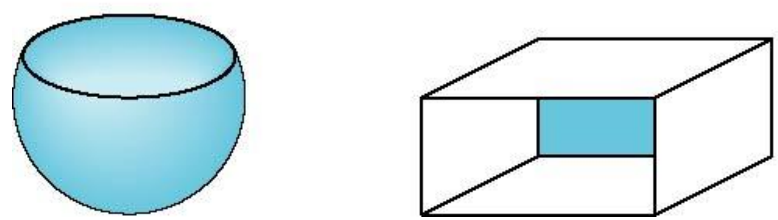
```
GLfloat ambient[] = {0.2, 0.2, 0.2, 1.0};
GLfloat diffuse[] = {1.0, 0.8, 0.0, 1.0};
GLfloat specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat shine = 100.0

glMaterialfv(GL_FRONT, GL_AMBIENT, ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, specular);
glMaterialfv(GL_FRONT, GL_SHININESS, shine);
```

# Front and Back Faces

- The default is shade only front faces which works correctly for convex objects
- If we set two sided lighting, OpenGL will shade both sides of a surface
- Each side can have its own properties which are set by using **`GL_FRONT`**, **`GL_BACK`**, or **`GL_FRONT_AND_BACK`** in **`glMaterialfv`**



back faces not visible



back faces visible

# Emissive Term

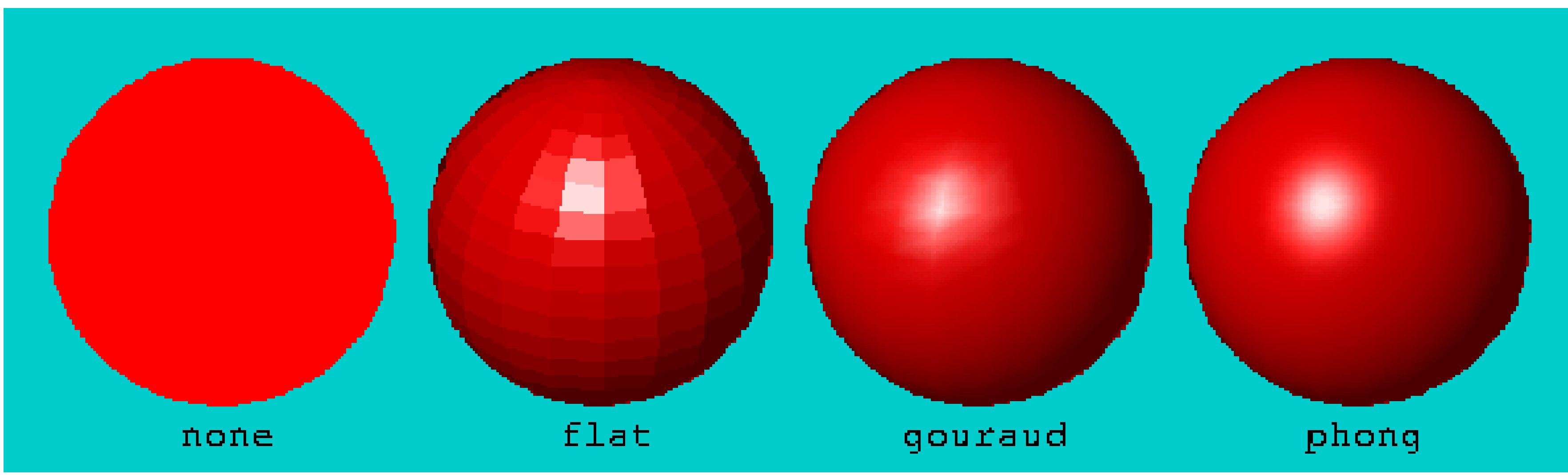
- We can simulate a light source in OpenGL by giving a material an emissive component
- This component is unaffected by any sources or transformations

```
GLfloat emission[] = {0.0, 0.3, 0.3, 1.0};
glMaterialfv(GL_FRONT, GL_EMISSION, emission);
```

# Shading

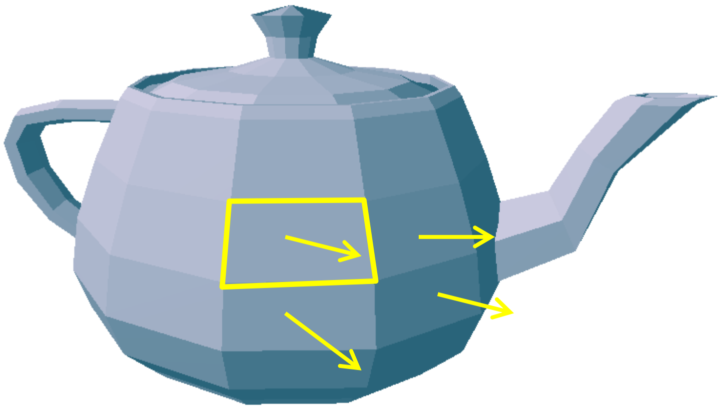
# Three Types of Shading

- Flat shading
- Gouraud shading
- Phong Shading
  - Don’t mix up with Phong Illumination Equation



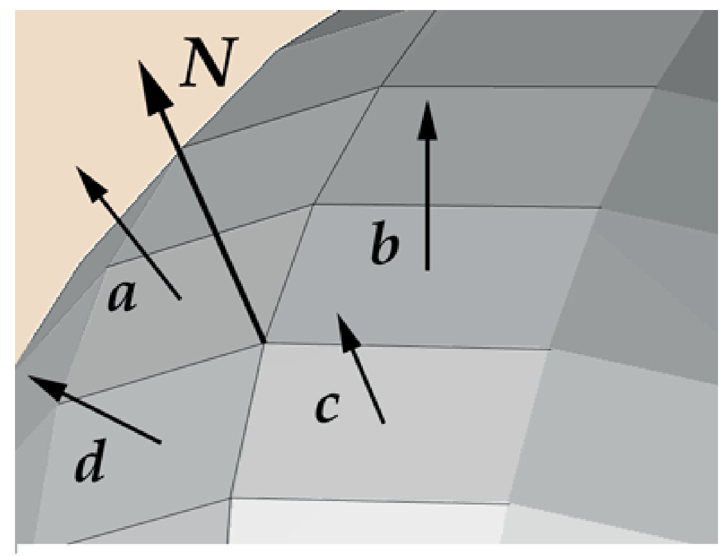
# Flat Shading

- For each polygon, we color the whole polygon with one color only
- Just pick <u>any</u> point on each polygon (e.g. a corner) and compute its color using PIE (with the surface normal at that point), and use this color for the whole polygon
- Distinctive color difference between each neighboring polygons
- To use flat shading in OpenGL
  - `glShadeModel(GL_FLAT);`



# Gouraud Shading

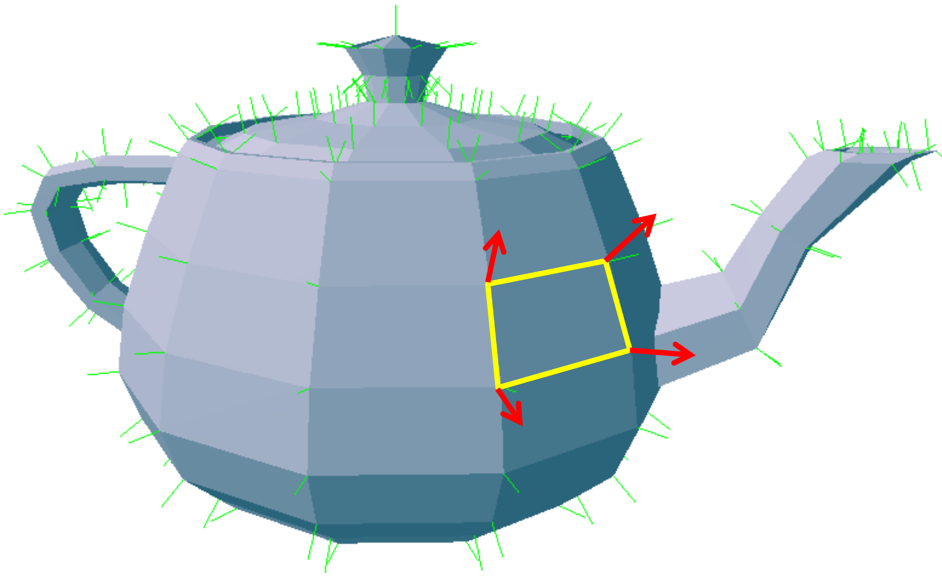
- For each **vertex**, compute the average normal vector of the polygons that share the vertex
  - We need to know the connectivity so that we can find the neighboring polygons
- Apply PIE at the vertex using its average normal vector
- Smoothly interpolate the computed colors at the vertices of the polygon to the interior of the polygon

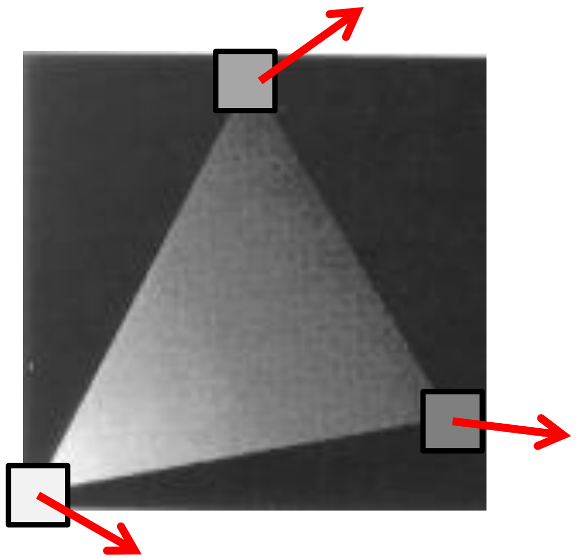



$$N = \frac{a+b+c+d}{4}$$

# 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, smoothly interpolate these vertex colors across the polygon during rasterization





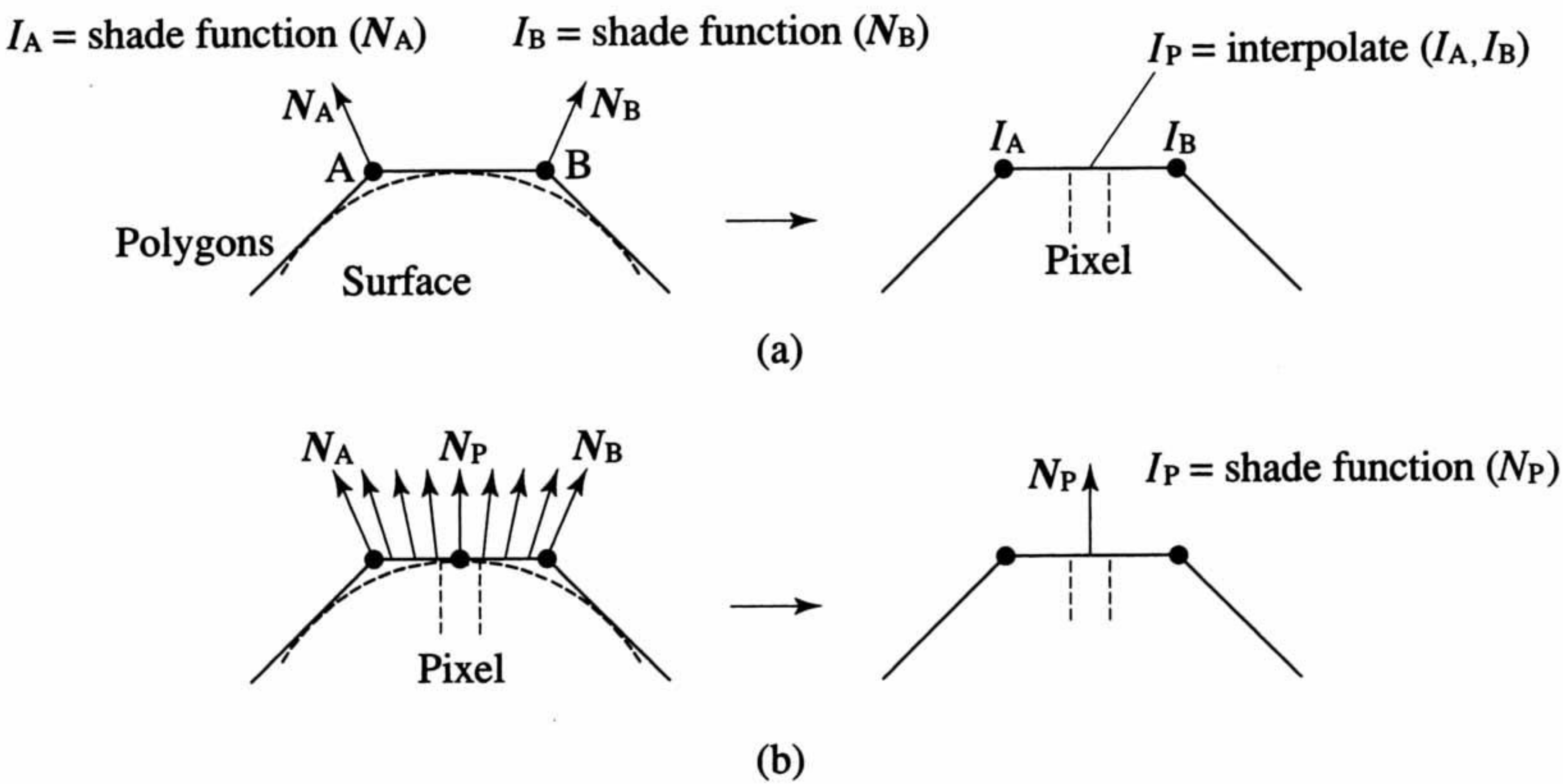
# Gouraud Shading

- To use Gouraud Shading in OpenGL
  - `glShadeModel(GL_SMOOTH);`

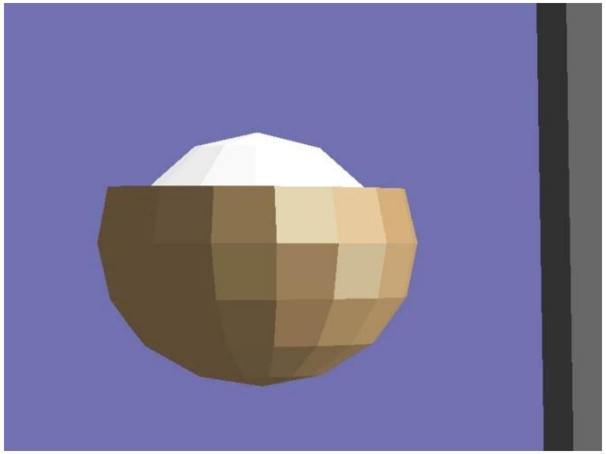
# Phong Shading

- 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 fragment in the polygon, we interpolate the normal vectors from the vertices
- Then, at each fragment, we apply PIE on the interpolated normal vector to compute a color for the fragment
  - A.k.a. per-pixel lighting computation
  - In Gouraud Shading, per-vertex lighting computation is used

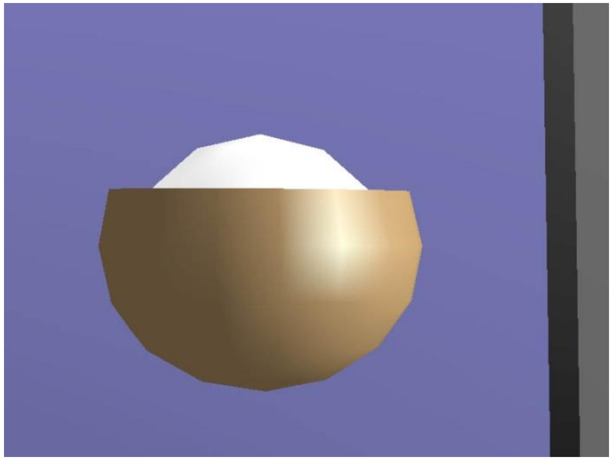
# Gouraud Shading & Phong Shading



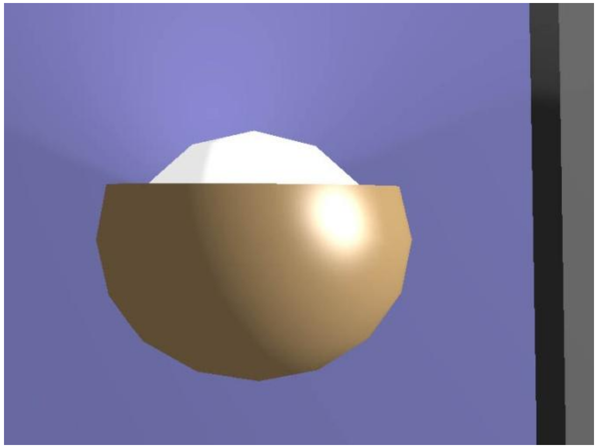
# Gouraud Shading vs Phong Shading



Flat Shading



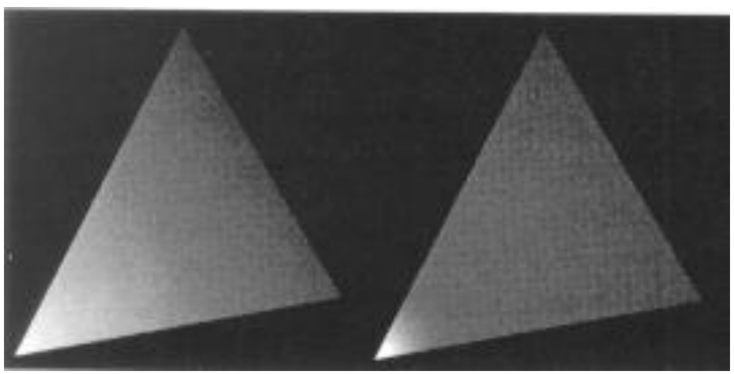
Gouraud Shading



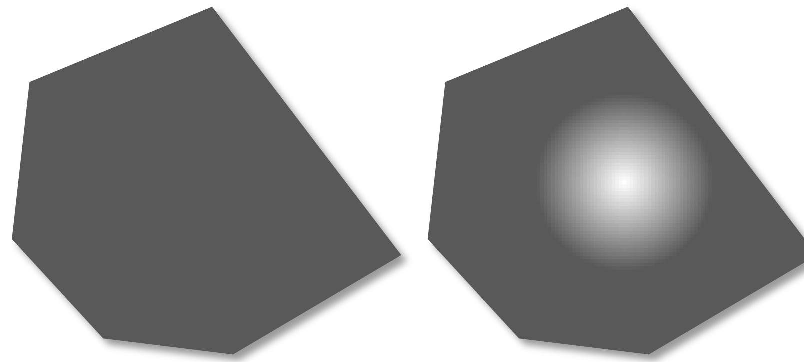
Phong Shading

# Gouraud Shading vs Phong Shading

- 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

- OpenGL does not support Phong Shading
  - But can be done by reprogramming the rendering pipeline using shaders

# End of Lecture 7