




# Chapter 6

## Lighting and Shading

Making images with a  
more naturalistic  
appearance



# Key Contents

1. Lighting Conception
2. Lighting Model
3. Surface orientation and Material
4. Light Properties
5. Shading
6. Computing Light in OpenGL
7. Front and Back Faces
8. Implementing Lighting Model
9. Other Illuminated Techniques

# 1. Lighting Conception

Actually, it is different colors(R,G,B) of the pixels on the surface



Utah teapot by Martin Newell in 1974

Light-material interactions cause each point to have a different color or shade

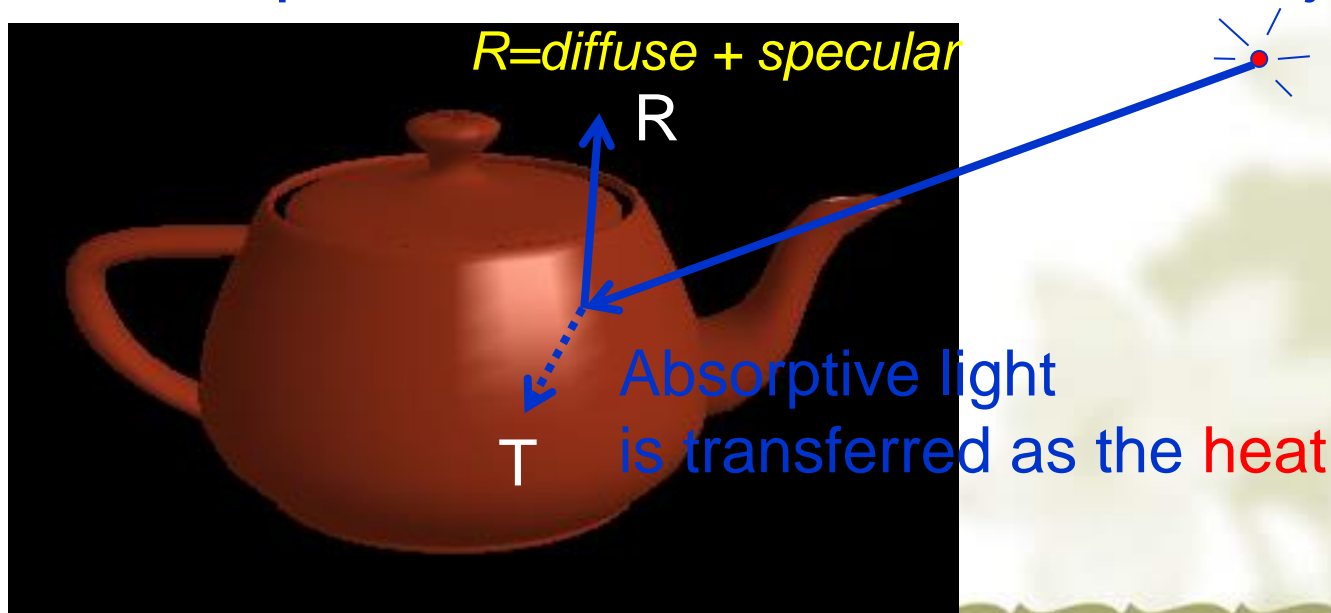


Need to consider

- Surface orientation--normal
- Material properties
- Light sources
- Location of viewer

# Lighting Conception

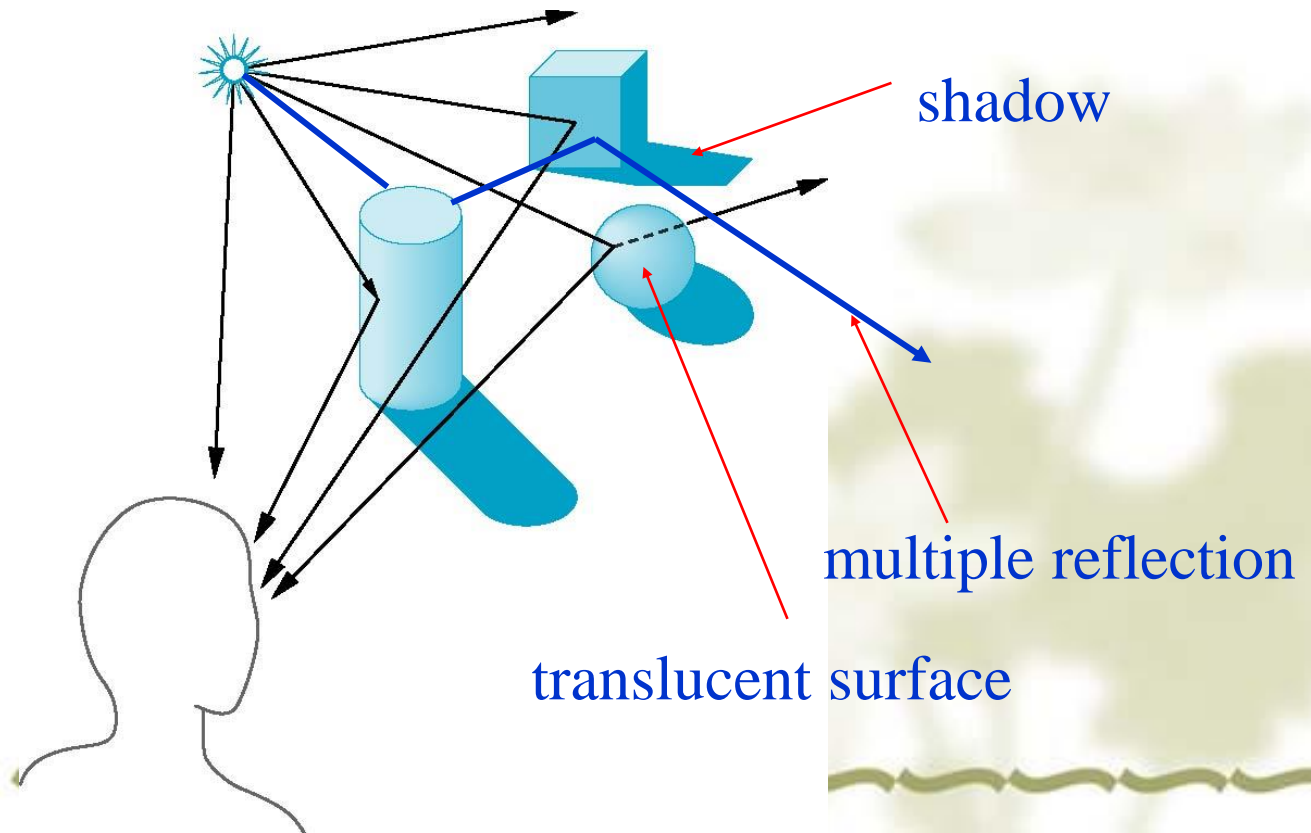
- ❖ Light which shines from light source to the objects transmits the **reflective** light, **transparent** light and **absorptive** light(not be seen)
- ❖ Reflective light includes *diffuse* and *specular* which more depends on the material of the object



# Local vs Global Lighting

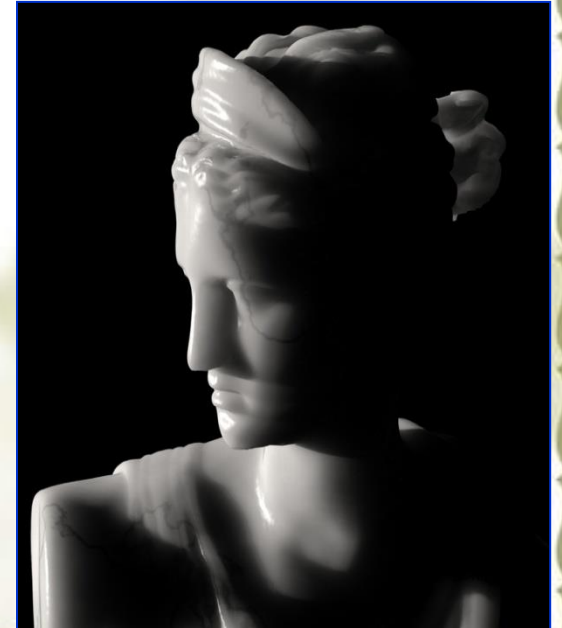
- ❖ In computer graphics, especially real time graphics, we are happy if things “look right”
- ❖ Exist many techniques for approximating effects

Lighting  
Shading  
Rendering



## 2. Lighting Model

- ❖ We will use the *local lighting model*: all light comes from lights defined within the scene
- ❖ Light is seen to have three components:
  - ↪ Ambient light
  - ↪ Diffuse light
  - ↪ Specular light
- ❖ All light uses the RGB color model,  $I = (I_R, I_G, I_B)$

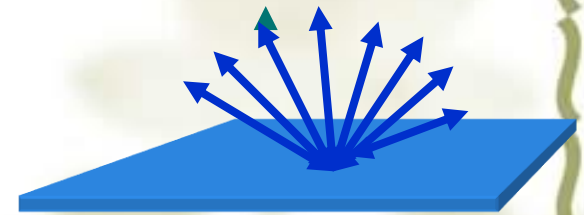




# Ambient Reflection

- ❖ Light is simply present but involves objects
- ❖ Total ambient light  $A$  is a scene-defined ambient value  $L_0$  plus the sum of the individual ambient lights

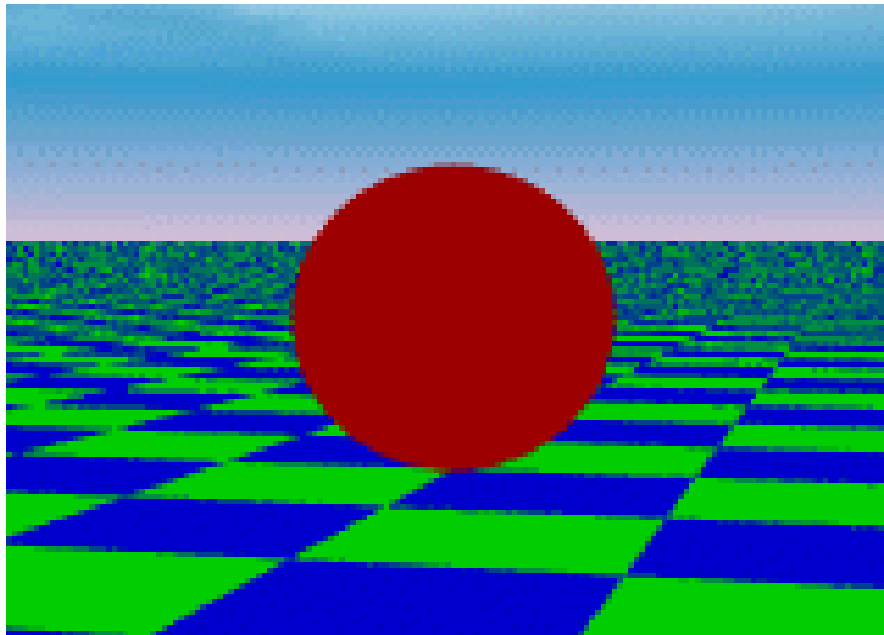
$$A = L_0 + \sum_{lights} (L_A * C_A)$$



- ❖ Individual light contributions come from ambient light  $L_A$  and material ambient value  $C_A$
- ❖ Here  $L_A$  and  $C_A$  are triples of (R, G, B)

# Ambient Reflection

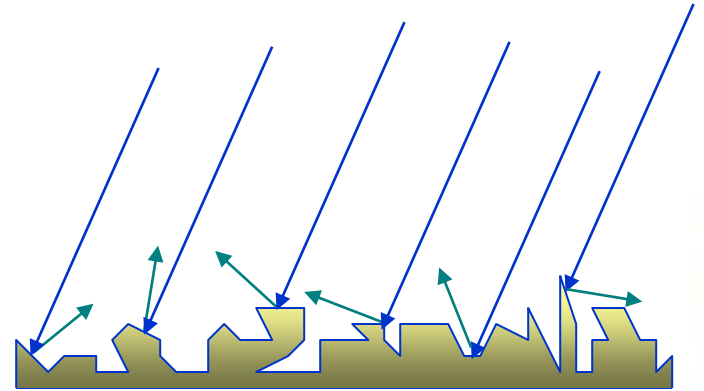
- ❖ Do not care Light Position
- ❖ Do not care Viewer Position
- ❖ Do not care Normals of Surface



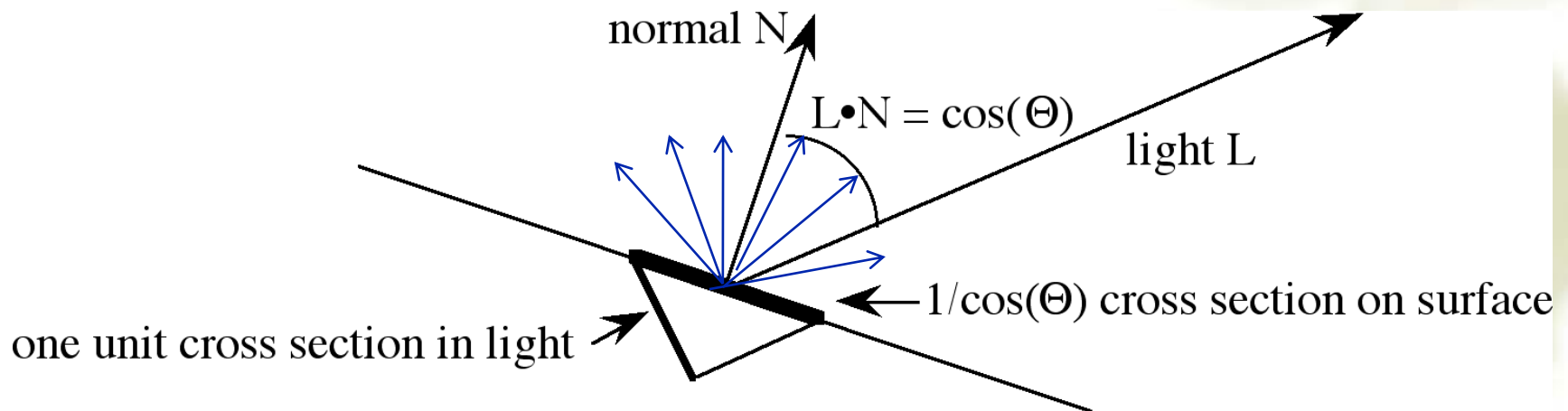


# Lambertian Diffuse Reflection

- ❖ Perfectly diffuse reflector



- ❖ Diffuse light depends on the direction of light because of area seen by light



# Lambertian Diffuse Reflection

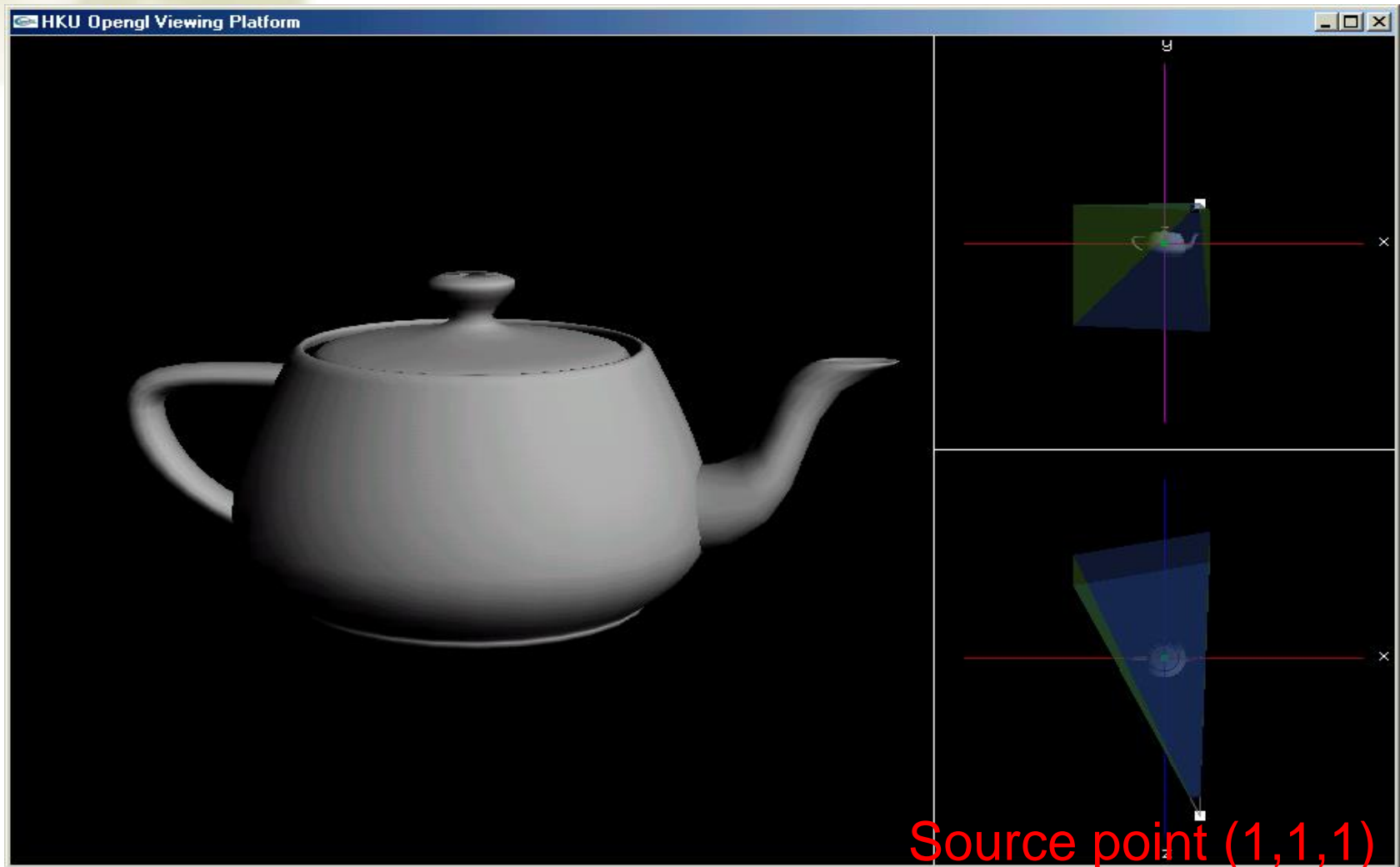
- ❖ The area visible to light depends on the cosine of the angle to the light -- the dot product of the light vector  $L$  and normal vector  $N$
- ❖ With light diffuse value  $L_D$  and materials diffuse value  $C_D$  total diffuse light  $D$  is given by

$$D = \sum_{lights} L_D * C_D * (L \bullet N)$$

- ❖ Here  $L_D$  and  $C_D$  are triples of (R, G, B)

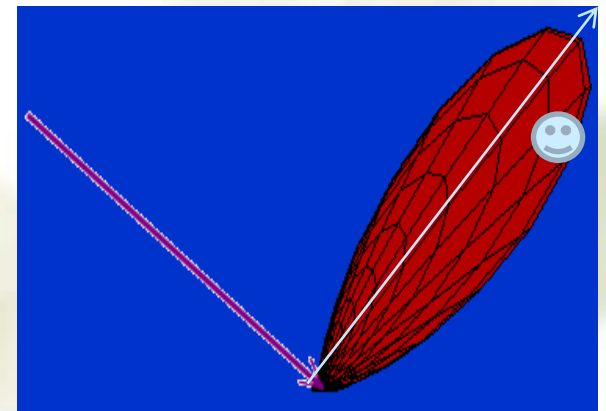
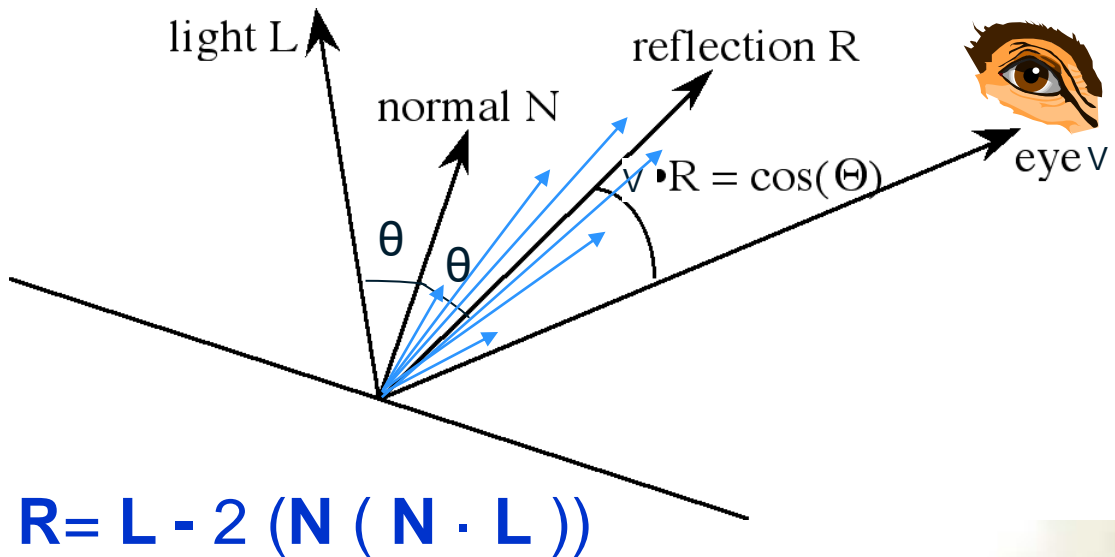
# Lambertian Diffuse Reflection

❖ Diffuse from directional source



# Phong Specular Reflection

Phong(1973) proposed using a term that dropped off as the angle between the viewer and the ideal reflection increased



B. T. Phong, Illumination for Computer Generated Pictures,(1975)

# Phong Specular Reflection

- ❖ Specular light also depends on the shininess  $K$  of the surface; this is modeled by a cosine function  $\cos^K(\phi)$
- ❖ Since the cosine is the dot product  $V \cdot R$ , for light specular value  $L_s$ , material value  $C_s$ , and material shininess  $K$ , we have

$$S = \sum_{lights} L_s * C_s * (V \bullet R)^K$$

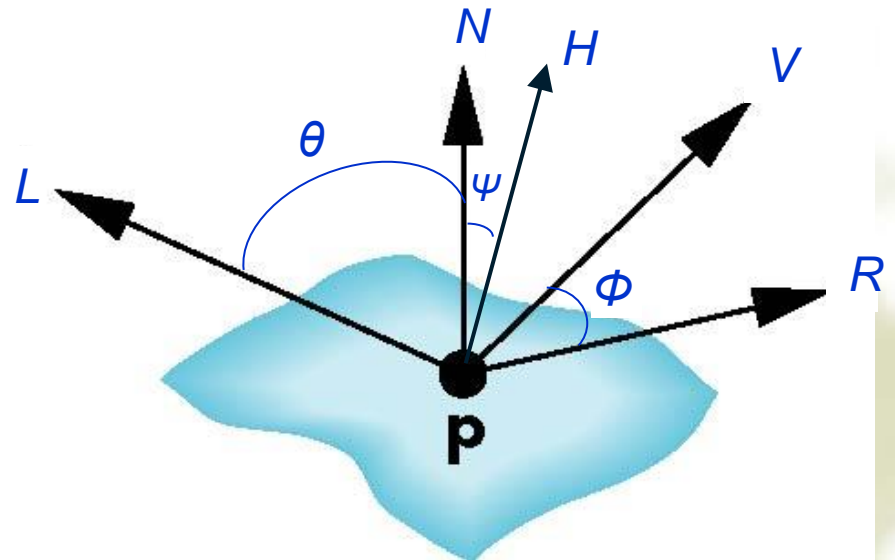
- ❖ Here  $L_s$  and  $C_s$  are triples of (R, G, B)

# Blinn-Phong Specular

Using the unit vector halfway between the viewer vector and the light source vector

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

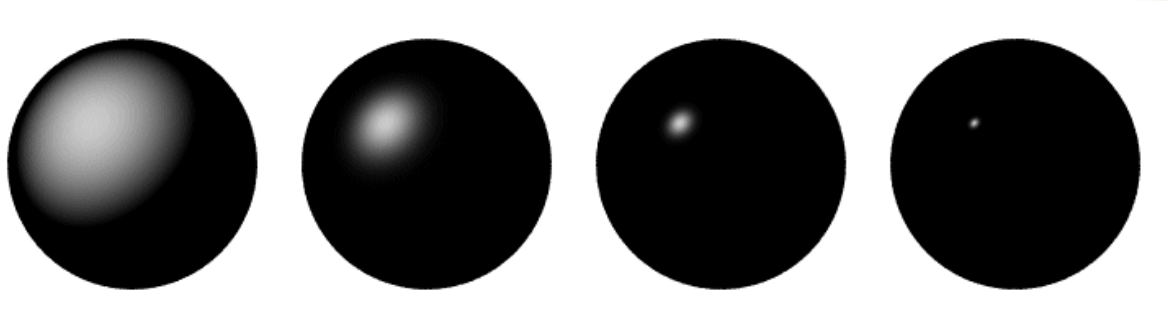
$$S = \sum_{lights} L_S * C_S * (N \cdot H)^K$$





# Shininess -- Phong Specular

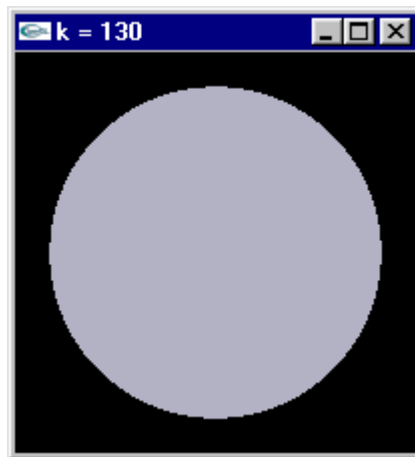
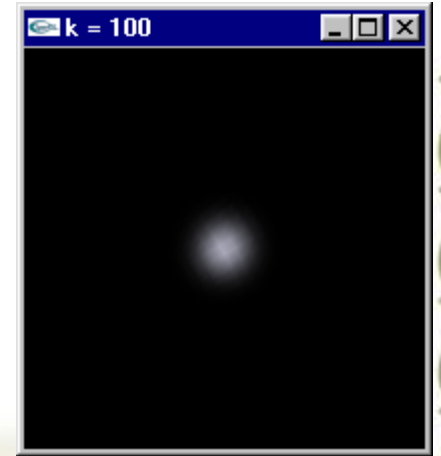
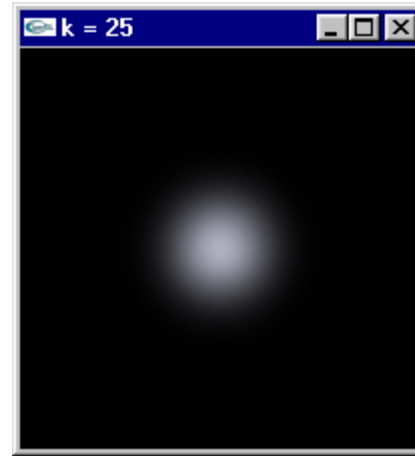
- ❖  $K$  : purely empirical constant, varies the rate of falloff
- ❖ no physical basis, works ok in practice



Higher values of the shininess coefficient  $K$  lead to smaller and brighter shiny spots

# Shininess -- Phong Specular

Shininess of a material –  $k$

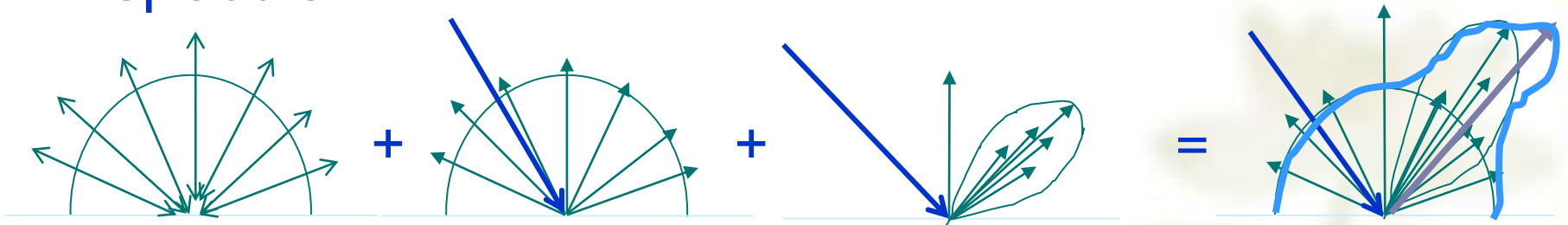


The larger the  $k$ , the higher the shininess.

*But the  $k$  shall less than 128.*

# Local Lighting Model

- ❖ We consider only lights in the scene, ignore reflected lights from each object or each surface
- ❖ Each light is composed of ambient, diffuse and specular



$$I = L_0 + \sum_{lights} (L_A * C_A) + \sum_{lights} L_D * C_D * (L \bullet N) + \sum_{lights} L_S * C_S * (N \bullet H)^K$$

# Summary of Lighting and Vector

- ❖ Has three components

  - ↪ Diffuse

  - ↪ Specular

  - ↪ Ambient

- ❖ Uses five vectors

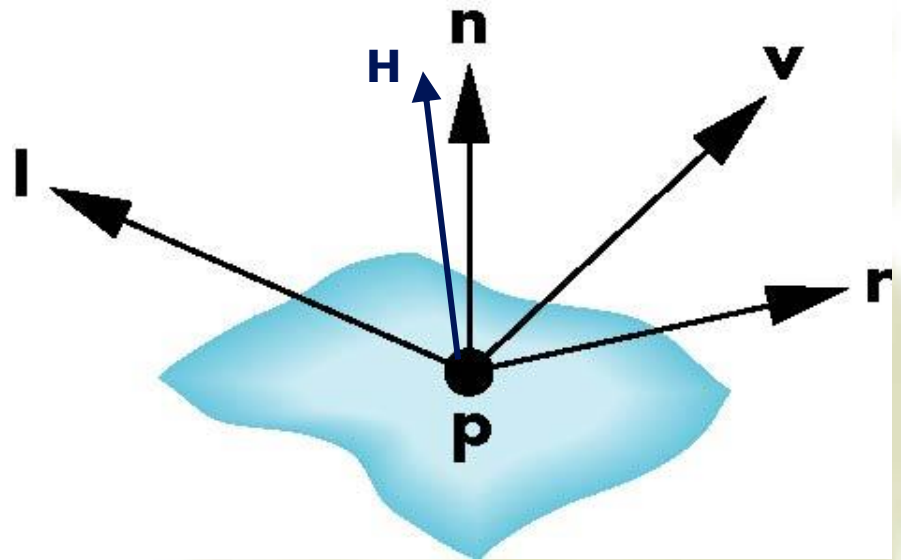
  - ↪ To source

  - ↪ To viewer

  - ↪ Normal

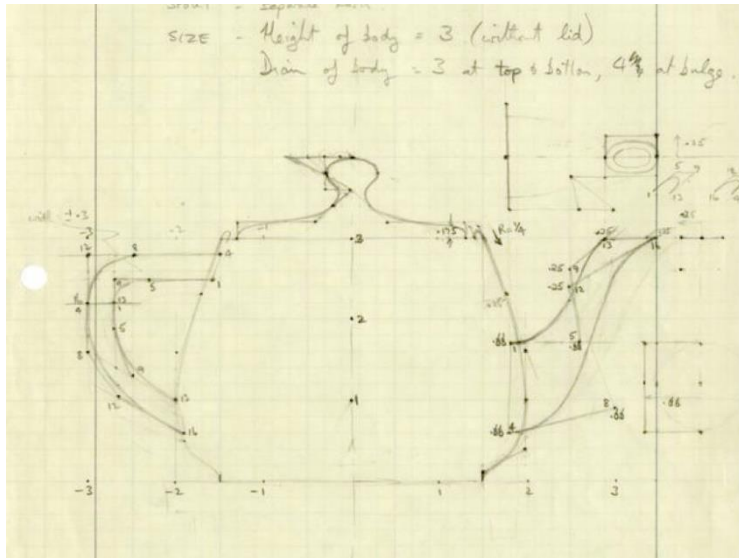
  - ↪ Perfect reflector

  - ↪ Halfway vector between viewer and source

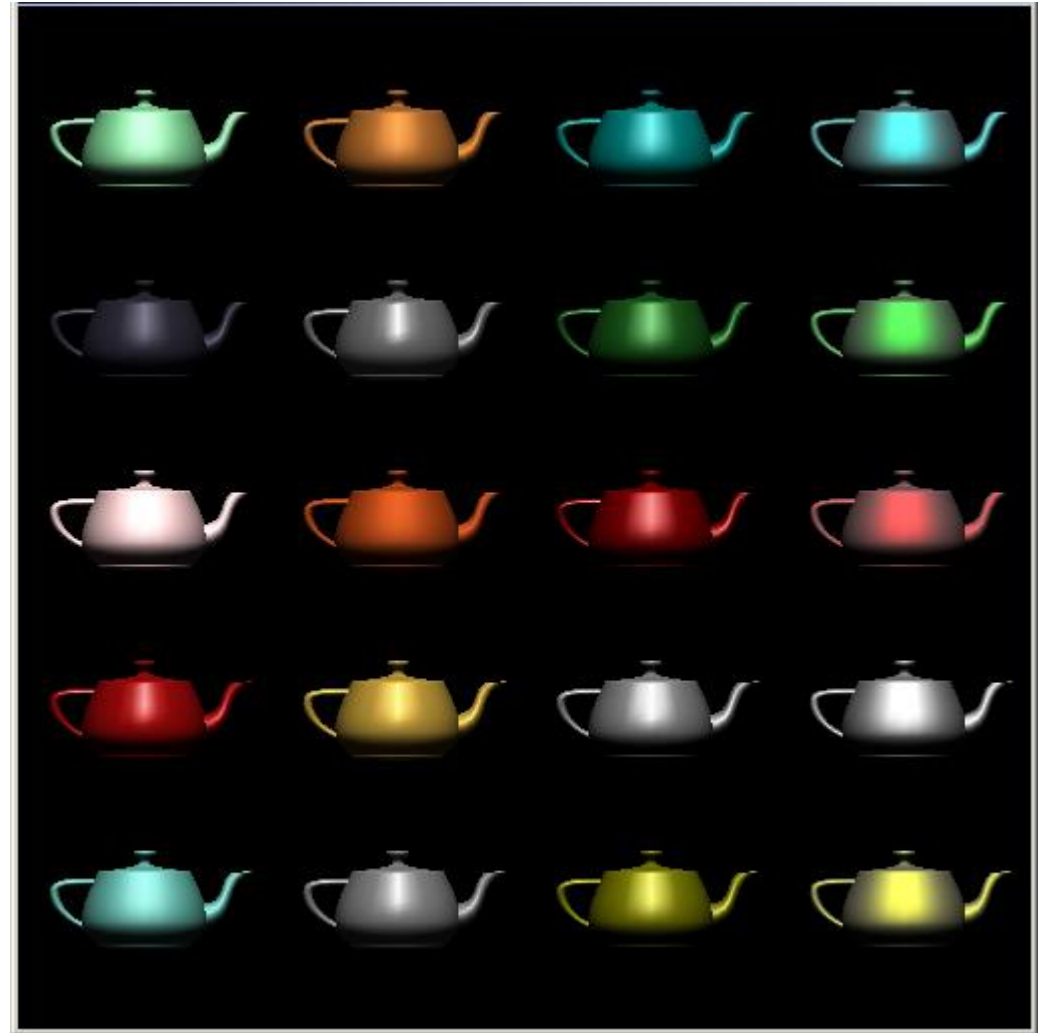


# Blinn-Phong Lighting Model

Only differences in these Utah teapots are the parameters in the modified Blinn-Phong lighting model



Utah teapot by Martin Newell in 1974



### 3. Surface orientation and Material

- ❖ In order to use lighting, you must have two things
  - ⌘ You must have **normals** for each vertex of your objects
  - ⌘ You must define the **material** for each object
- ❖ When you enable lighting, the lighting calculations we saw above will be done to create the color for each object

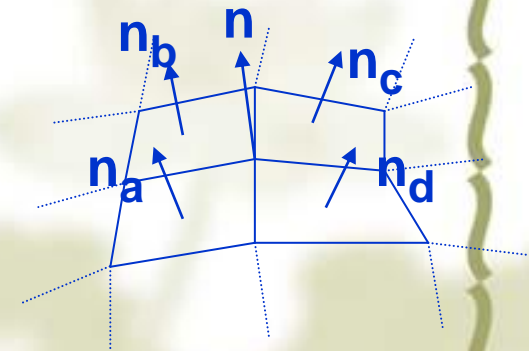
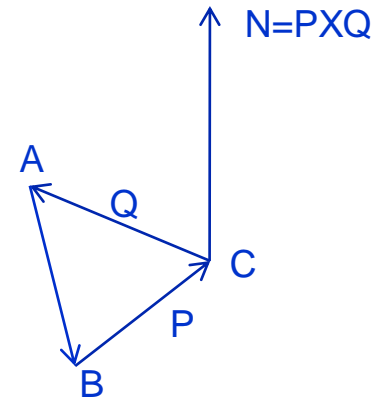


# Normals

- ❖ Can be computed as the cross product of two adjacent edges
- ❖ If several polygons meet at a vertex, the normals for each polygon can be averaged

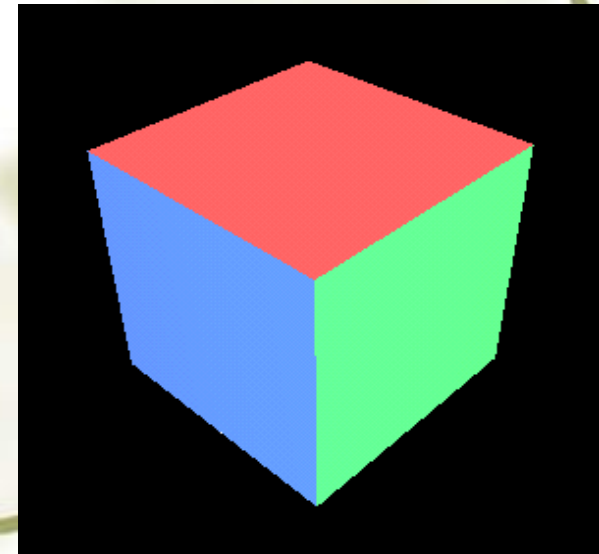
$$n = \frac{(n_a + n_b + n_c + n_d)}{4}$$

- ❖ Normals can also be computed exactly from directional derivatives if the object geometry is analytic
- ❖ Normalized surface normals
- ❖ The normal at the vertex could be defined for shading



# Defining Materials

- ❖ Materials are part of the appearance in the shape node:  $C_A, C_D, C_S$
- ❖ To define the material properties for your objects, you simply define the color for each lighting component and the value of the shininess
- ❖ These are all RGBA colors  
 $C_D = (\text{Red}_D, \text{Green}_D, \text{Blue}_D)$   
each component is  $[0, 1]$



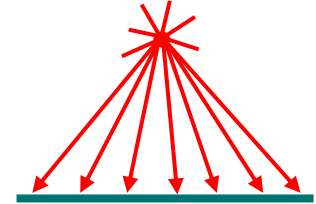
## 4. Light Properties

1. Types of Light Sources
2. Position - point in space
3. Directional - direction of light
4. Spotlight - direction and cutoff
5. Attenuation - dropoff of values with distance
6. Color for a Light Source in OpenGL

# 4.1 Types of Light Sources

- ❖ point lights

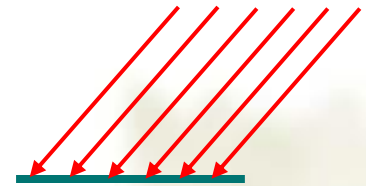
- ❖ finite position:  $(x,y,z,1)^T$



- ❖ directional/parallel lights

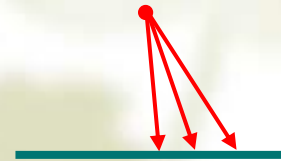
- ❖ point at infinity:  $(x,y,z,0)^T$

- ❖ from the origin to the point

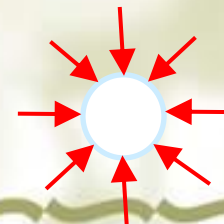


- ❖ spotlights

- ❖ position, direction, angle(cutoff)



- ❖ ambient lights

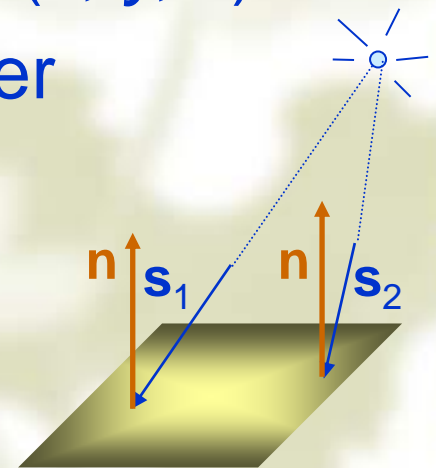
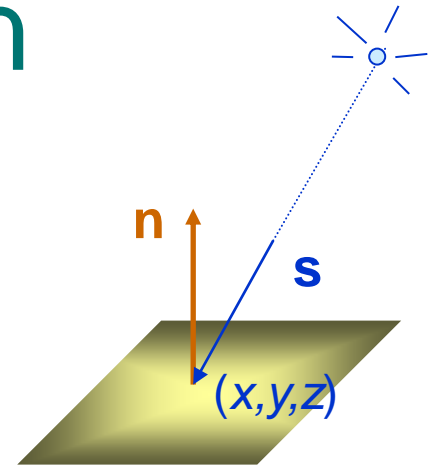


## 4.2 Light Position

### Point sources

- ❖  $\mathbf{n}$  is the normal of the surface.  
 $\mathbf{n}$  remains the same at various locations on a plane
- ❖  $(x, y, z)$  are the coordinates of a point on the surface
- ❖  $\mathbf{s}$  is the direction of the light source at  $(x, y, z)$
- ❖  $\mathbf{s}$  changes from one location to another  
eg,  $\mathbf{s}_1 \neq \mathbf{s}_2$  for point sources

The closer the light source is to a surface,  
the larger the change of  $\mathbf{s}$  is.

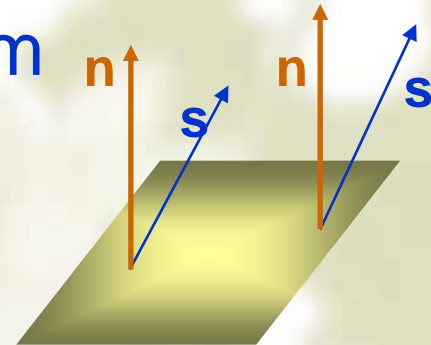


## 4.3 Directional sources

(parallel sources or distributed sources)



- ❖ When the light source is farther away, the change of  $\mathbf{s}$  becomes less significant.
- ❖ When the distance becomes very large,  $\mathbf{s}$  remains unchanged across a surface
- ❖ The direction of the light ray is from object to source  
e.g., the sun.





## 4.4 The Spot of The Light Source

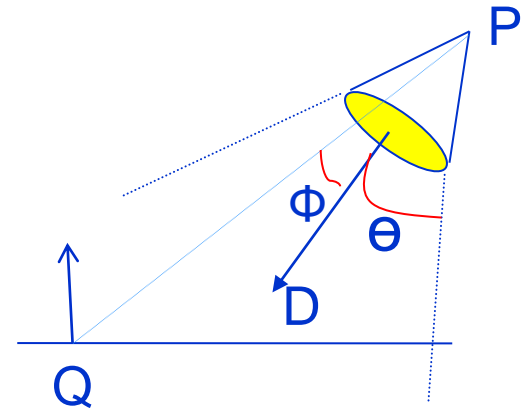
To define a spot light (a point source)  
the position  $P$ , direction  $D$ , cutoff  $\theta$ , dropoff  $d$

$Q$  is in cutoff  $\theta$

$$I_Q = \cos^d(\phi) = ((Q-P) \cdot D)^d \quad \text{if } \phi < \theta$$

$Q$  is out of range cutoff  $\theta$

$$I_Q = 0 \quad \text{if } \phi > \theta$$



## 4.5 Light Attenuation

- ❖ When a light is moving away from a surface, the intensity of light received decreases
- ❖ For a point light source, the attenuation factor is

$$A_f = \frac{1}{d^2} \quad (\text{in theory})$$

$$A_f = \frac{1}{a + bd + cd^2} \quad (\text{in practice})$$

$d$  is the distance from the point light source.

$a$ ,  $b$ , and  $c$  are heuristic parameters chosen empirically.

- ❖ For a directional light source,  $A_f = 1$

$$I = L + \sum_{lights} A_f \times L_A \times C_A + \sum_{lights} A_f \times L_D \times C_D \times (L \bullet N) + \sum_{lights} A_f \times L_S \times C_S \times (N \bullet H)^K$$

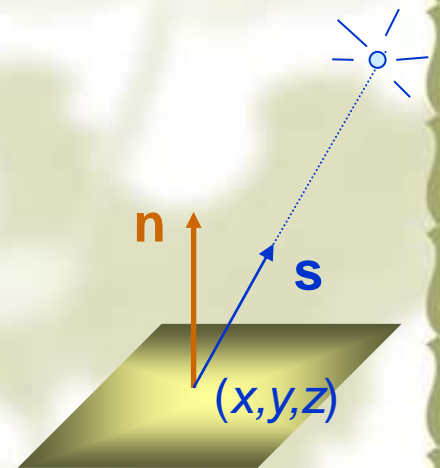
## 4.6 Color for a Light Source

### - RGB values

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

```
vec4 diffuse0 =vec4(1.0, 1.0, 1.0, 1.0); //white color
vec4 ambient0 = vec4(1.0, 1.0, 1.0, 1.0); //white color
vec4 specular0 = vec4(1.0, 1.0, 1.0, 1.0); //white color
// specify the position of a point light source
vec4 light0_pos =vec4(1.0, 2.0, 3.0, 1.0);
//specify the direction of a directional light source
vec4 light1_pos =vec4(1.0, 1.0, 1.0, 0.0);
```

$s$  is the direction from light point to the origin.

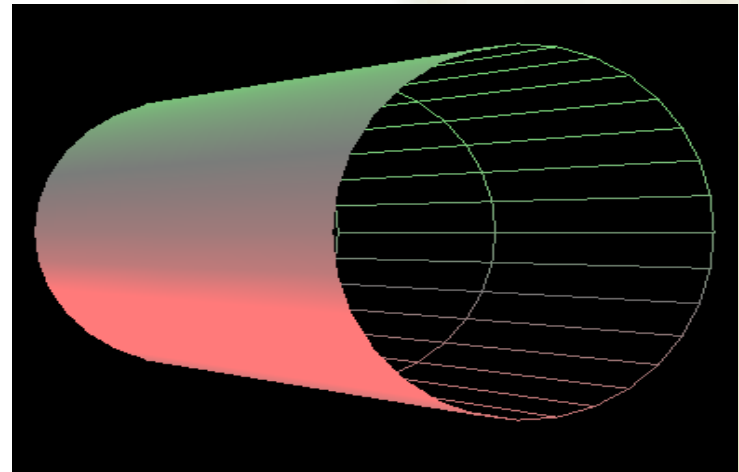
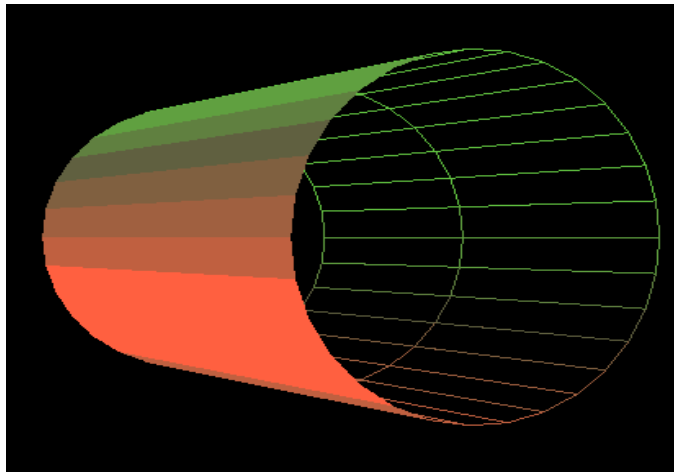


## 5. Shading

- ❖ How to compute the color of each part of a graphic object, especially a polygon
- ❖ We have properties at each vertex, and shading uses these to compute the color at each pixel in the object
- ❖ The shade of a surface is the amount of ambient, diffuse and specular light reflected from the surface
- ❖ OpenGL can offer flat and smooth (Gouraud) shading

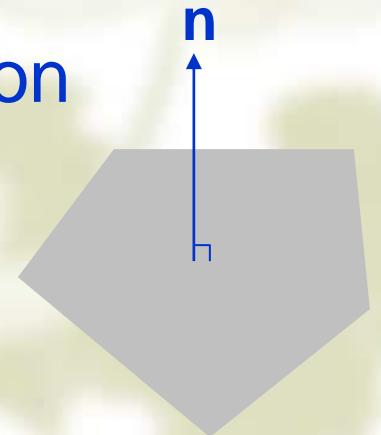
## 5.1 Flat and Smooth Shading

- ❖ Shading means the color can be set or can be computed with the lighting model
- ❖ Smooth shading interpolates vertex colors across the polygon



# Constant (Flat) Shading

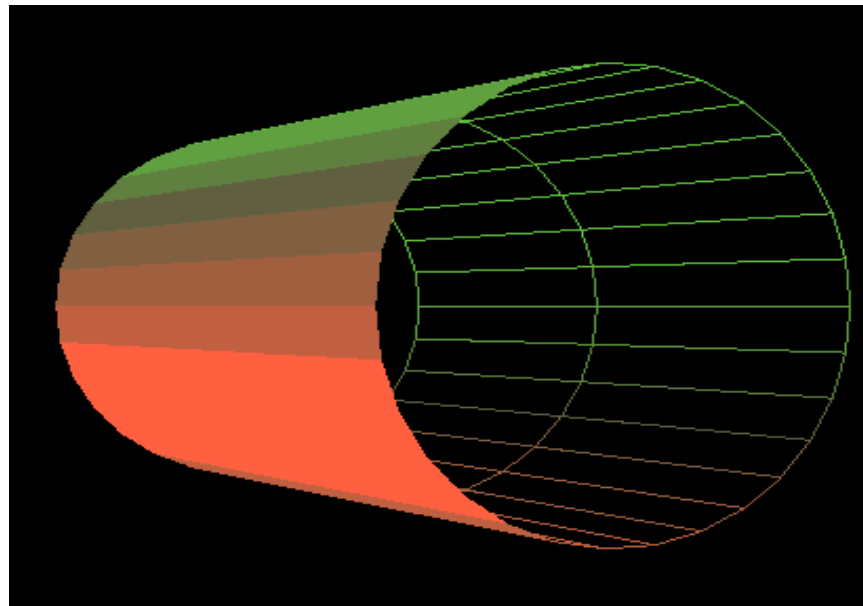
- ❖ In constant shading, the reflection calculation only carried out once for a point on a polygon, e.g., the first vertex. The resulting shade is assigned to all pixels covered by the polygon
- ❖ Flat shading simply applies a single color to an entire polygon
- ❖ The surface normal is used in calculation
- ❖ Efficient but often too coarse





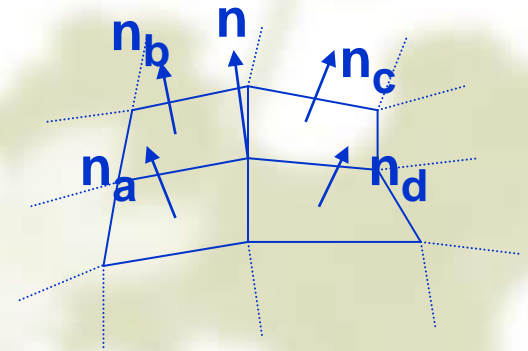
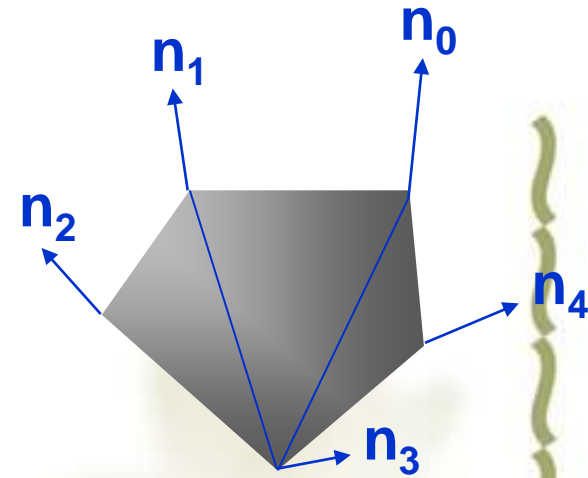
# Discrete Change of Shade

- ❖ A *Mach band* is a stripe found along the edge of two polygons of different shades. *Mach bands* often occur in constant shading
- ❖ Our eyes are particularly sensitive to the discrete change of shade



## 5.2 Gouraud (Smooth) Shading

- ❖ Shading calculation is carried out for each vertex of a triangle
- ❖ Each vertex is assigned a normal that may be the normal of the triangle, or the average of the normals of adjacent triangles
- ❖ The light color of the pixel at a vertex is calculated using the normal  $N$  assigned to the vertex

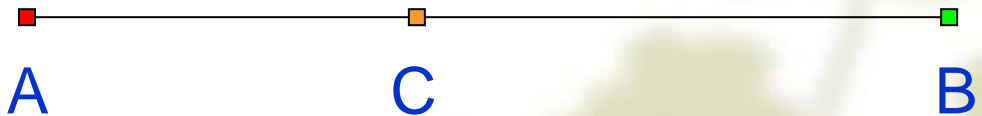


# Gouraud (Smooth) Shading

- ❖ The shades of other pixel covered by a triangle is obtained through interpolation, as in the case of colors



- ❖ The color of an in-between point, C, is determined by



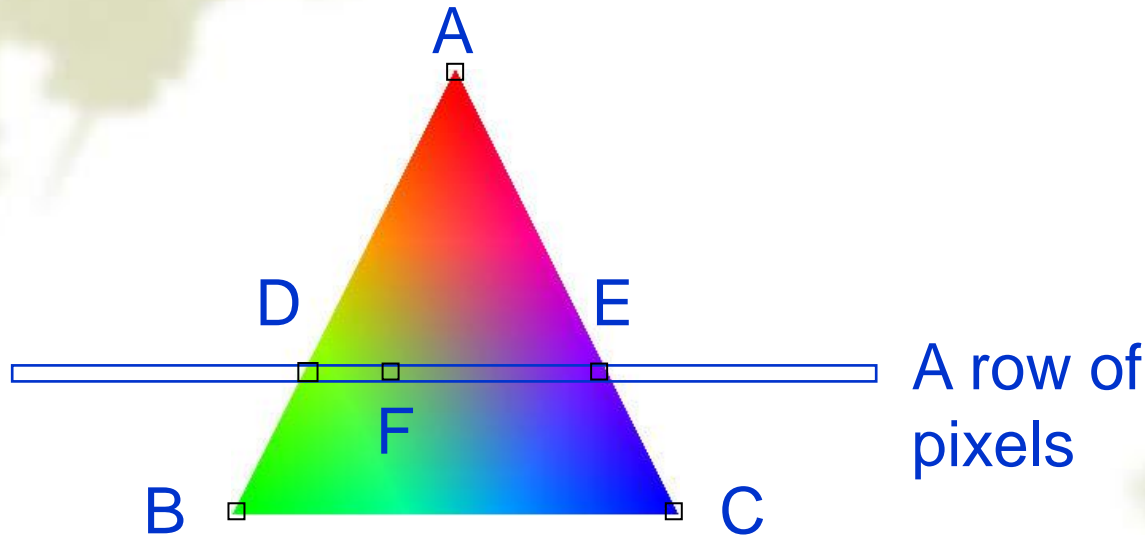
$$\alpha = \frac{\text{distance of AC}}{\text{distance of AB}}$$



$$Color(C) = (1 - \alpha) \times Color(A) + \alpha \times Color(B)$$

# Gouraud (Smooth) Shading

- ❖ Determine the colors of a pixel covered by a triangle



- ❖ The color of D is determined from that of A and B as in the case of a line
- ❖ The color of E is determined from A and C in the same way
- ❖ The color of F is then determined from D and E in the same way

# illumination Interpolation (Gouraud Shading)

$$\diamond I_Q = (1-u) \times I_A + u \times I_B$$

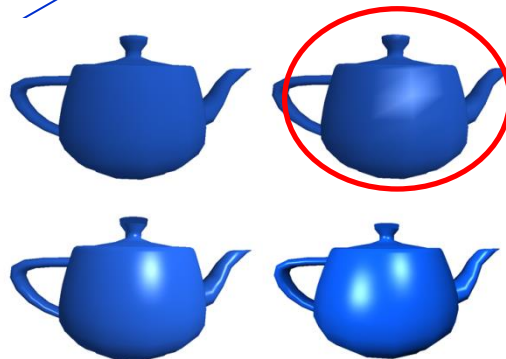
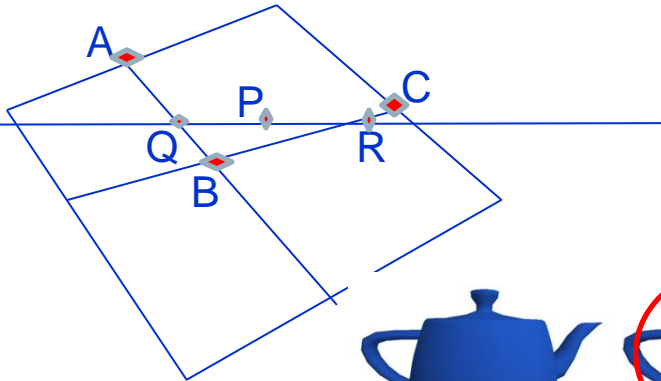
$$0 \leq u \leq 1, u = AQ/AB$$

$$\diamond I_R = (1-w) \times I_B + w \times I_C$$

$$0 \leq w \leq 1, w = BR/BC$$

$$\diamond I_P = (1-t) \times I_Q + t \times I_R$$

$$0 \leq t \leq 1, t = QP/QR$$



Increment Computing:

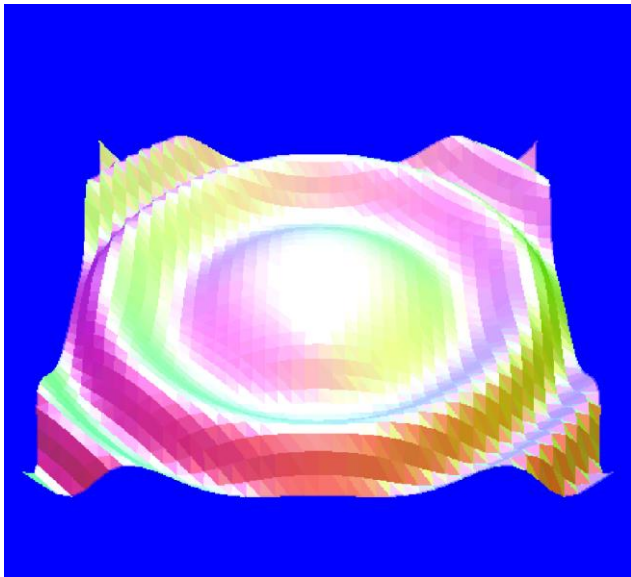
$$I_{P2} = (1-t_2) \times I_Q + t_2 \times I_R$$

$$I_{P1} = (1-t_1) \times I_Q + t_1 \times I_R$$

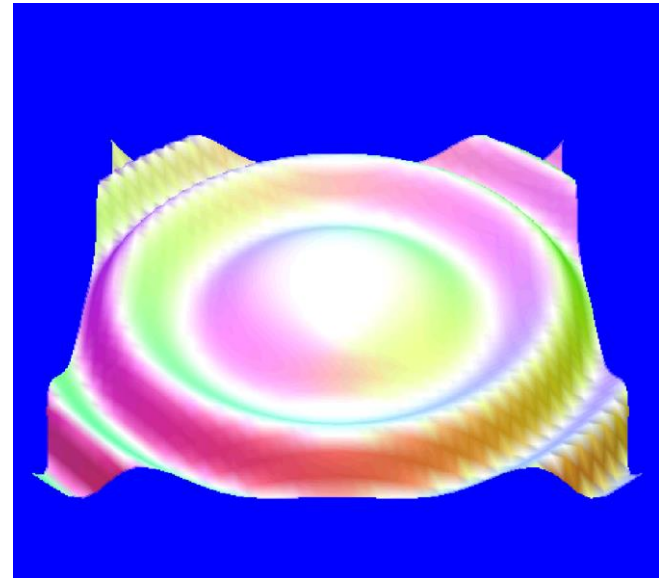
$$\begin{aligned} I_{P2} &= I_{P1} + (I_R - I_Q) (t_2 - t_1) \\ &= I_{P1} + \Delta I_{RQ} \times \Delta t \end{aligned}$$

# Shading Examples

- ❖ Flat shading on each polygon



- ❖ Smooth shading on each polygon

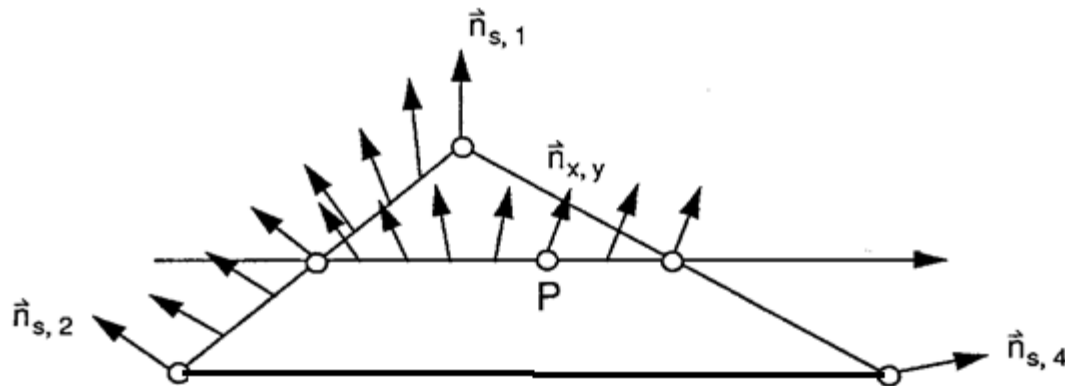


$$f(x,y) = 0.3 \cdot \cos(x^2 + y^2 + t)$$

## 5.3 Other Shading Models

Phong shading:

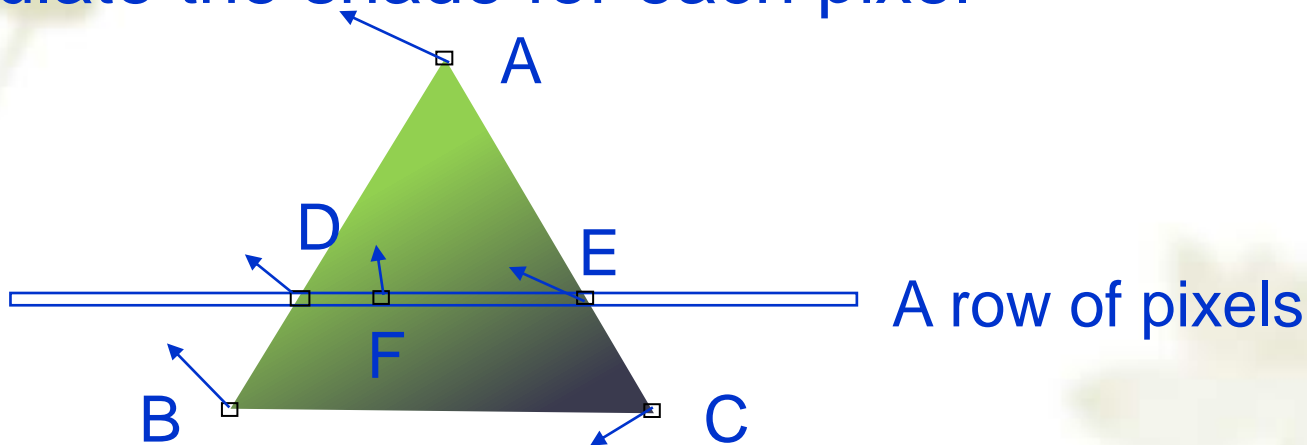
- ❖ interpolate the normal across a triangle
- ❖ compute the color for each pixel directly with local lighting model





# Phong Shading

- ❖ To interpolate the normal across a triangle, and calculate the shade for each pixel



- ❖ Using Gouraud shading, we often need to cut a surface into very small pieces in order to obtain satisfactory result. Phong shading reduces the excessive cutting but requires more calculation
- ❖ It may produce quality pictures but much slower

# Normal Interpolation (Phong Shading)

$$\diamond n_Q = (1-u) \times n_A + u \times n_B$$

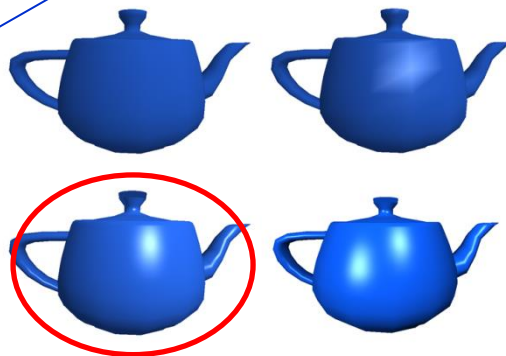
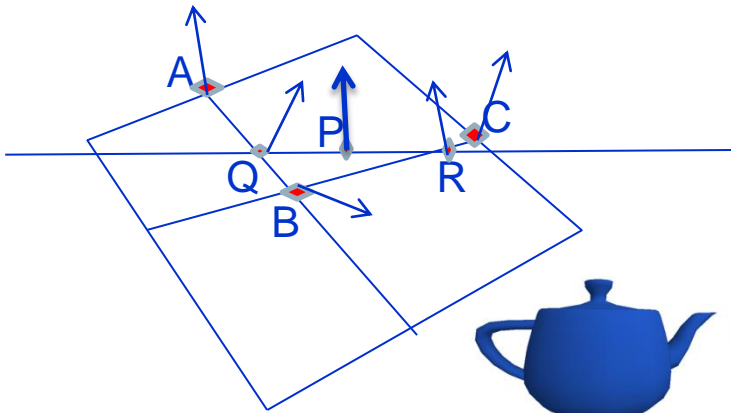
$$0 \leq u \leq 1, u = AQ/AB$$

$$\diamond n_R = (1-w) \times n_B + w \times n_C$$

$$0 \leq w \leq 1, w = BR/BC$$

$$\diamond n_P = (1-t) \times n_Q + t \times n_R$$

$$0 \leq t \leq 1, t = QP/QR$$



Incremental Computing:

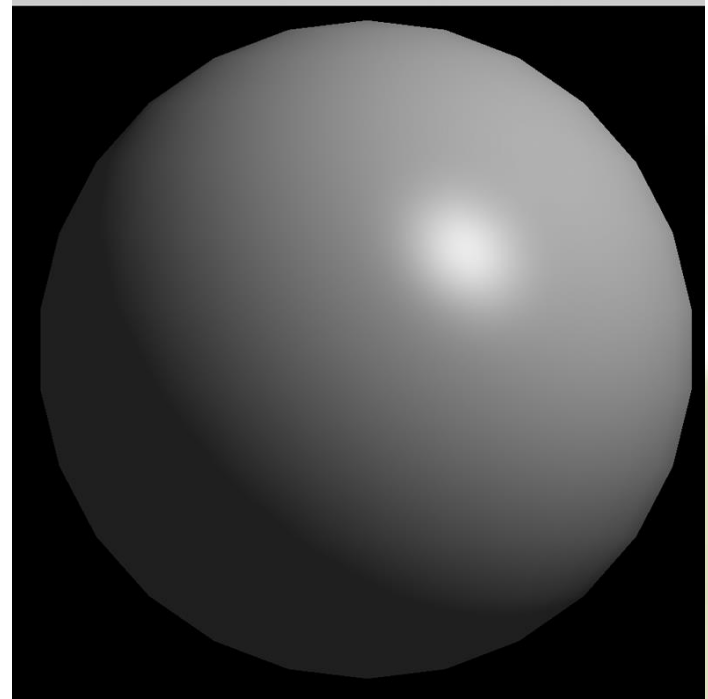
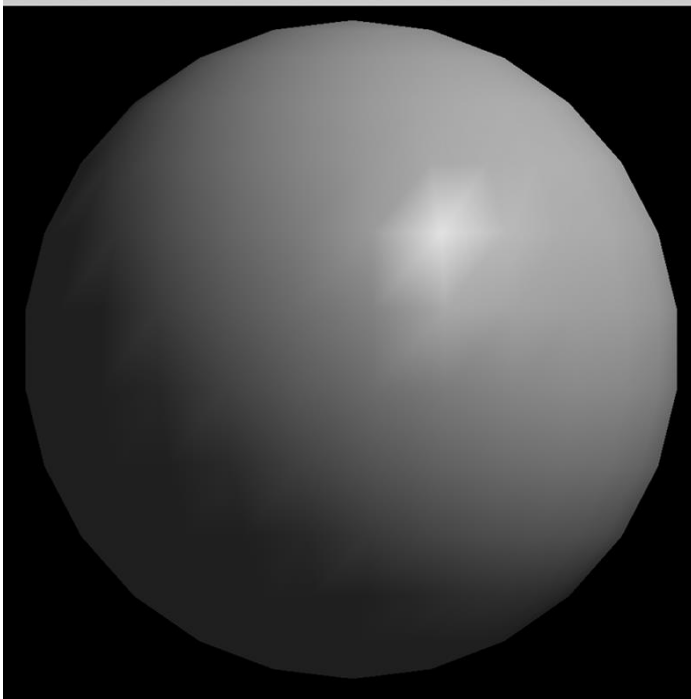
$$n_{P2} = (1-t_2) \times n_Q + t_2 \times n_R$$

$$n_{P1} = (1-t_1) \times n_Q + t_1 \times n_R$$

$$\begin{aligned} n_{P2} &= n_{P1} + (n_R - n_Q) (t_2 - t_1) \\ &= n_{P1} + \Delta n_{RQ} \times \Delta t \end{aligned}$$

# Compare Smooth and Phong

❖ Smooth (left) and Phong (right) compared



## 6. Computing Light in OpenGL

- ❖ If an OpenGL **light** has components (LR, LG, LB), and a **material** has corresponding components (MR, MG, MB), then, ignoring all other reflectivity effects, the light that arrives at the eye is given by  $(LR*MR, LG*MG, LB*MB)$ .
- ❖ Similarly, if you have **two lights** that send (R1, G1, B1) and (R2, G2, B2) to the eye, OpenGL adds the components, giving  $(R1+R2, G1+G2, B1+B2)$ .
- ❖ If any of the sums are greater than 1 (corresponding to a color brighter than the equipment can display), the component is clamped to 1

# Computing Light in OpenGL

- ❖ The color of a material mainly depends on the diffuse reflectance.
- ❖ The diffuse reflection alone provides most information about the curvature and the depth of an object
- ❖ The ambient reflectance of a material is often the same as the diffuse reflectance
- ❖ The highlights produced in specular reflections are the blurred images of the light sources.

# Computing Light in OpenGL

```
color4 light_ambient = color4(0.4,0.4,0.4,1.0); // gray light
color4 material_ambient = color4(1.0,0.0,0.0,1.0); //red surface

color4 ambient_product = product(light_ambient, material_ambient);
=====
color4 light_diffuse = color4(1.0,1.0,1.0,1.0); // white light
color4 material_diffuse = color4(1.0,0.1,0.1,1.0); //red surface

color4 diffuse_product = product(light_diffuse, material_diffuse);
=====
color4 light_specular = color4(1.0,1.0,1.0,1.0); // white light
color4 material_specular = color4(1.0,0.1,0.1,1.0); //red surface

color4 specular_product = product(light_specular, material_specular);
```

Usually, material\_diffuse and material\_specular are the same

## Pure Ambient



## Pure Diffuse



## Pure Specular



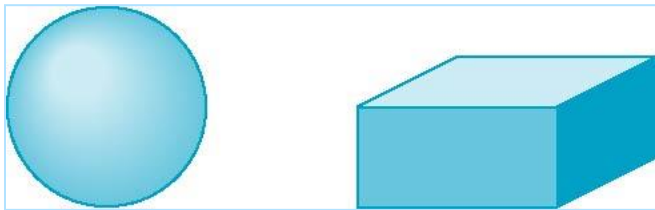
## Overall



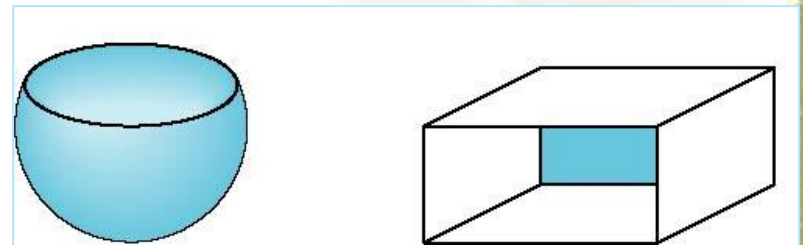


## 7. Front and Back Faces

- ❖ Every face has a front and back
- ❖ For many objects, we never see the back, so we don't care how or if it's rendered
- ❖ If it matters, we can handle in shader



back faces not visible



back faces visible

# Emissive Term

- ❖ If we want to simulate a lamp, light source in OpenGL is defined as an emissive component
- ❖ This component is unaffected by any sources or transformations

```
color4 emission = color4(0.0,1.0,1.0,1.0); // cyan color
```

the material  
attributes on the  
vertex of the  
surface:

```
typedef struct material_vertex {  
    color4 ambient;  
    color4 diffuse;  
    color4 specular;  
    color4 emission;  
    float shininess;  
};
```

# Transparency

- ❖ Material properties are specified as RGBA values
- ❖ The A value can be used to make the surface translucent
- ❖ The default is that all surfaces are opaque

```
color4 light_ambient = color4(0.4,0.4,0.4,1.0);  
color4 material_ambient = color4(1.0,0.0,0.0,1.0);
```

opaque surface

```
color4 material_ambient = color4(1.0,0.0,0.0,0.5);
```

translucent

# 8. Implementing Lighting Model

- ❖ Need
  - Normals
  - material properties
  - Lights
- ❖ State-based shading functions have been deprecated (`glNormal`, `glMaterial`, `glLight`)
- ❖ computing in shaders

# method I: Vertex Lighting Shader

```
// vertex shader
in vec4 vPosition;
in vec3 vNormal;
out vec4 color;           //vertex shader => fragment shader
```

```
// light and material properties,  $L_a * C_a$ ;  $L_d * C_d$ ;  $L_s * C_s$ 
uniform vec4 AmbientProduct, DiffuseProduct, SpecularProduct;
uniform mat4 ModelView;
uniform mat4 Projection;
uniform vec4 LightPosition;
uniform float Shininess;
```

# method I: Vertex Lighting Shader

```
void main()
{
    // Transform vertex position into eye coordinates
    vec3 pos = (ModelView * vPosition).xyz;    //a vertex

    vec3 L = normalize( LightPosition.xyz - pos );
    vec3 E = normalize( -pos ); //eye on the origin
    vec3 H = normalize( L + E ); //middle vector

    // Transform vertex normal into eye coordinates
    vec3 N = normalize( ModelView*vec4(vNormal, 0.0) ).xyz;
```

注：当物体变换时，其顶点的法向量vNormal也要随之变换

# method I: Vertex Lighting Shader

```
// Compute terms in the illumination equation
```

```
vec4 ambient = AmbientProduct;
```

```
float Kd = max( dot(L, N), 0.0 );
```

```
vec4 diffuse = Kd*DiffuseProduct;
```

```
float Ks = pow( max(dot(N, H), 0.0), Shininess );
```

```
vec4 specular = Ks * SpecularProduct;
```

```
if( dot(L, N) < 0.0 ) specular = vec4(0.0, 0.0, 0.0, 1.0);
```

```
gl_Position = Projection * ModelView * vPosition;
```

```
color = ambient + diffuse + specular;
```

```
color.a = 1.0;
```

```
}
```



# method I: Fragment Lighting Shaders

```
// fragment shader
```

```
in vec4 color;
```

```
void main()
```

```
{
```

```
    gl_FragColor = color;
```

```
}
```

to take the interpolated colors from the rasterizer and assign them to fragments

# method II: Vertex Lighting Shader

```
// vertex shader
in vec4 vPosition;
in vec3 vNormal;

// output values that will be interpolated per-fragment
out vec3 fN;
out vec3 fE;
out vec3 fL;

uniform mat4 ModelView;
uniform mat4 Projection;
uniform vec4 LightPosition;
```

# method II: Vertex Lighting Shader

```
void main()
{
    fN = vNormal;
    fE = vPosition.xyz;
    fL = LightPosition.xyz;

    if( LightPosition.w != 0.0 ) { // point light source
        fL = LightPosition.xyz - vPosition.xyz;
    }

    gl_Position = Projection*ModelView*vPosition;
}
```

# method II: Fragment Lighting Shaders

```
// fragment shader
```

```
// per-fragment interpolated values from the vertex shader
```

```
in vec3 fN;
```

```
in vec3 fL;
```

```
in vec3 fE;
```

```
uniform vec4 AmbientProduct, DiffuseProduct, SpecularProduct;
```

```
uniform float Shininess;
```

# method II: Fragment Lighting Shaders

```
void main()
{
    // Normalize the input lighting vectors
    vec3 N = normalize(fN);
    vec3 E = normalize(fE);
    vec3 L = normalize(fL);

    vec3 H = normalize( L + E );
    vec4 ambient = AmbientProduct;
```

# method II: Fragment Lighting Shaders

```
float Kd = max(dot(L, N), 0.0);  
vec4 diffuse = Kd*DiffuseProduct;
```

```
float Ks = pow(max(dot(N, H), 0.0), Shininess);  
vec4 specular = Ks*SpecularProduct;
```

```
// discard the specular highlight if the light's behind the vertex  
if( dot(L, N) < 0.0 )  
    specular = vec4(0.0, 0.0, 0.0, 1.0);
```

```
gl_FragColor = ambient + diffuse + specular;  
gl_FragColor.a = 1.0;
```

```
}
```

# Exercise 6

1. 画一个材质是白色的**cube**，再定义三个点光源，分别是红色光、绿色光、蓝色光，光源位置在**cube**的三个可见面的前方。用fragment shader编程实验：

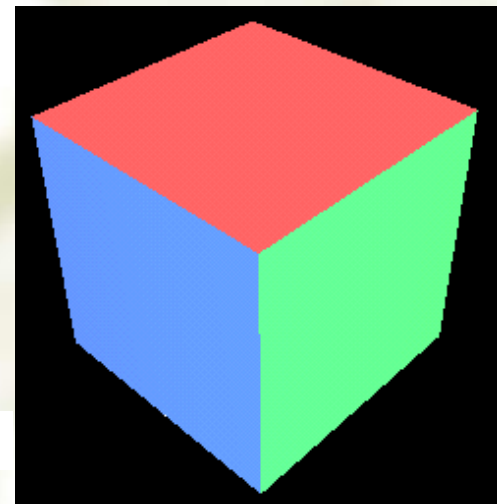
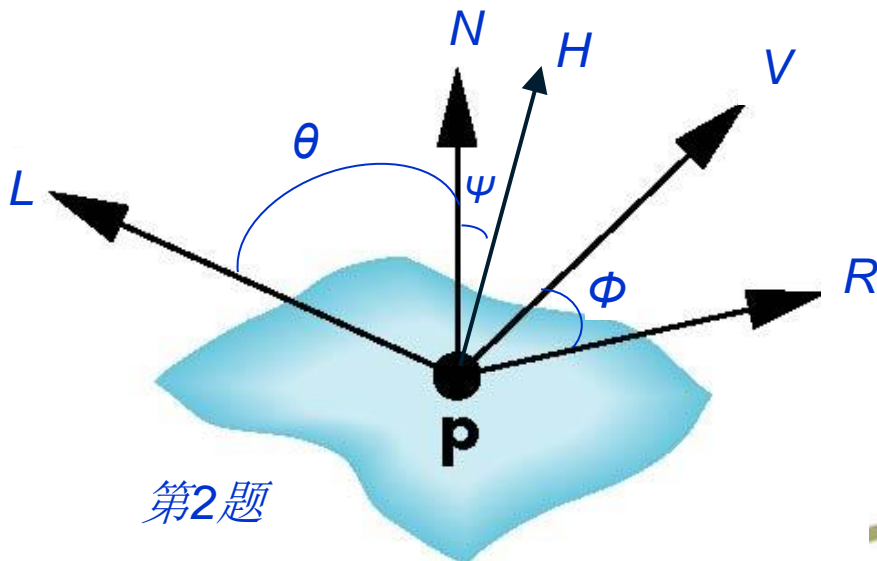
1) 只定义漫反射光

2) 只定义漫反射光和镜面反射光

3) 定义漫反射光、镜面反射光，以及环境光

比较三种情况的结果，并讨论原因。

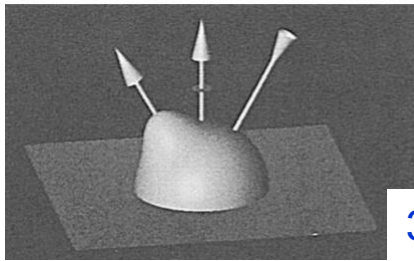
2. #6.7 to prove  $2\psi = \phi$  by coplanar





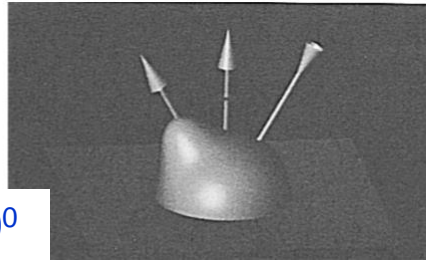
# 9. Other Illuminated Techniques

- ❖ Micro facets for specular reflection
- ❖ Peak specular reflection depends on the incident angle by a collection of mirror like micro facets

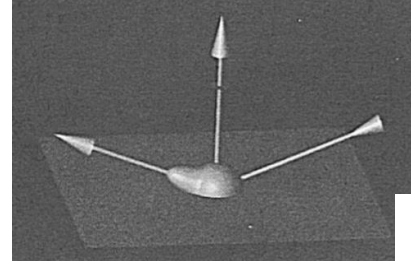


Phong

30°

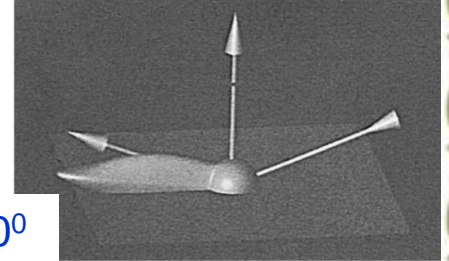


Torrance-Sparrow

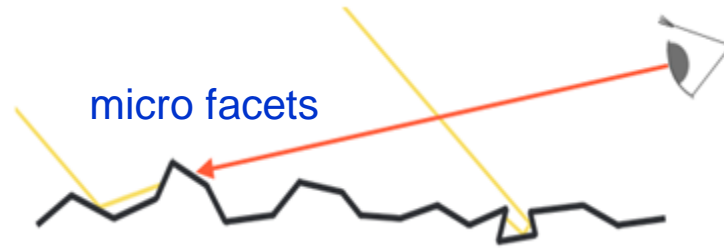


Phong

70°



Torrance-Sparrow



- ❖ To take into account the random directions of micro facets all over the surface

# More Accurate Specular Reflection

- ❖ BRDF(Bidirectional Reflectance Distribution Function)

$$R_{bd} = \frac{I(v)}{I(l)} \quad \text{即：出射光强/入射光强}$$

- ❖ BRDF for Blinn model(1977) and Cook-Torrance model(1981):

$$R_{bd} = k_d R_d + k_s R_s \quad \text{where} \quad k_d + k_s = 1$$

- ❖ Specular Reflection Model:

$$I = f I_a R_a + \sum_{i=1}^M I_{li} (N \cdot L_i) d\omega_{li} (k_d R_d + k_s R_s)$$

where  $R_s = \frac{F D G}{\pi (N \cdot V)(N \cdot L)}$  Torrance-Sparrow model(1967)

F is the Fresnel reflection law. D is the distribution function of the directions of the micro facets. G is the amount by which the facets shadow and mask each other.

# Other BRDF

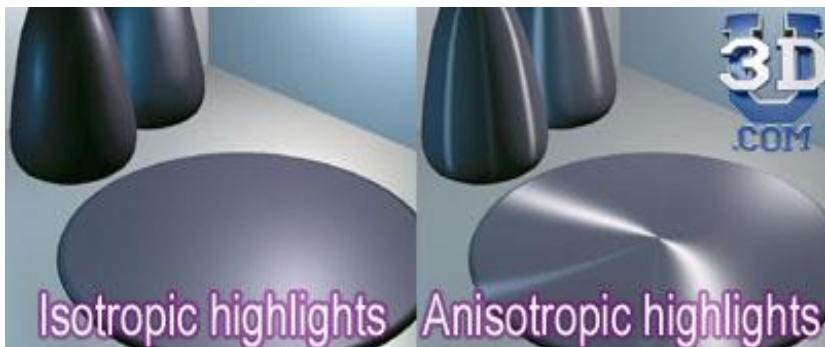
- ❖ BTDF(Bidirectional Transmittance Distribution Function)
- ❖ BSSRDF (Bidirectional Surface Scattering Reflectance Distribution Function)



Jensen H.W, Marschner S.R, Levoy M, et al. A practical model for subsurface light transport[C]. Proceedings of the 28<sup>th</sup> annual conference on Computer graphics and interactive techniques.ACM, 2001

# Anisotropic & Isotropic Lighting

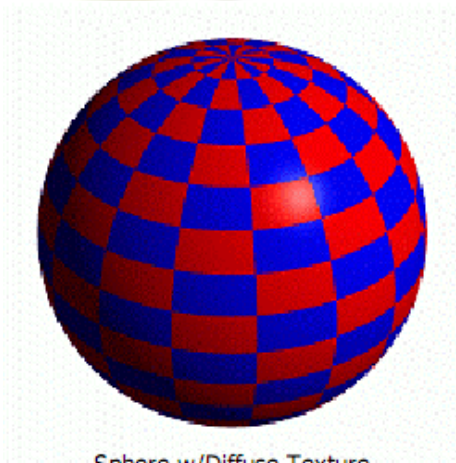
- ❖ Microfacets distributed regular normals stretch reflections and highlights in a direction that runs perpendicular to the grain or grooves in a surface
- ❖ The human hair produces highlights that run perpendicular to the hair direction, because there is more curvature running around these hairs than running along the hairs, hair in this photograph is shown scattering light anisotropically
- ❖ Contrast *anisotropic* lighting with the more common *isotropic* lighting, which scatter light evenly in all directions



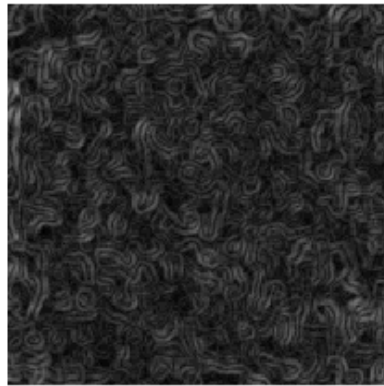


# Bump Mapping By Normal

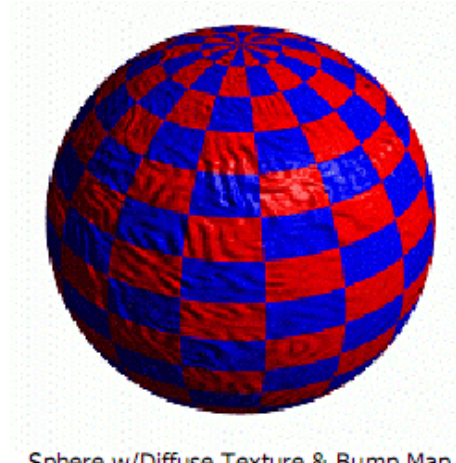
- ❖ This does not change the actual shape of the surface, we are only shading it as if it were a different shape!



Sphere w/Diffuse Texture

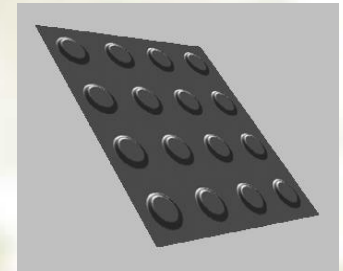
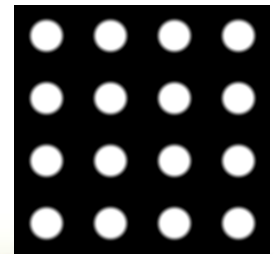


Swirly Bump Map



Sphere w/Diffuse Texture & Bump Map

Courtesy of Leonard McMillan. Used with permission.



bump map shading

- ❖ Bump map itself is 2D image wherein the height of each point is defined by the gray level
- ❖ Normals are computed from the changes in color in the height field

James F. Blinn, Simulation of Wrinkled Surfaces,(1978)

# Global Illumination

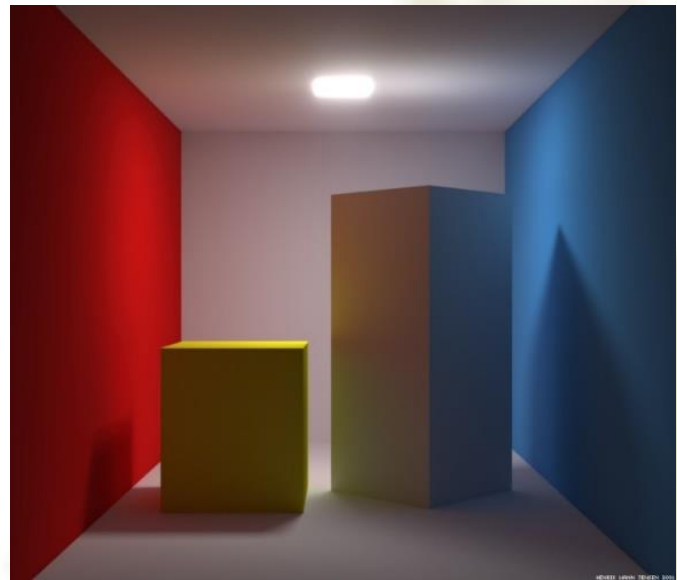
- ❖ local illumination: let all light come only from lights
- ❖ Global illumination: considers not only how light comes into a scene, but also how light is reflected within a scene
- ❖ More complex, but more realistic
- ❖ Main algorithms:
  - Ray Tracing
  - Radiosity

# Comparison

- ❖ Ray-Tracing: great specular, approx diffuse
- ❖ Radiosity: great diffuse, ignore specular
- ❖ advanced hybrids: combine them



Ray Tracing



Radiosity