



Computer Graphics

# Lighting & Shading

---

Teacher: A.prof. Chengying Gao(高成英)

E-mail: [mcsgcy@mail.sysu.edu.cn](mailto:mcsgcy@mail.sysu.edu.cn)

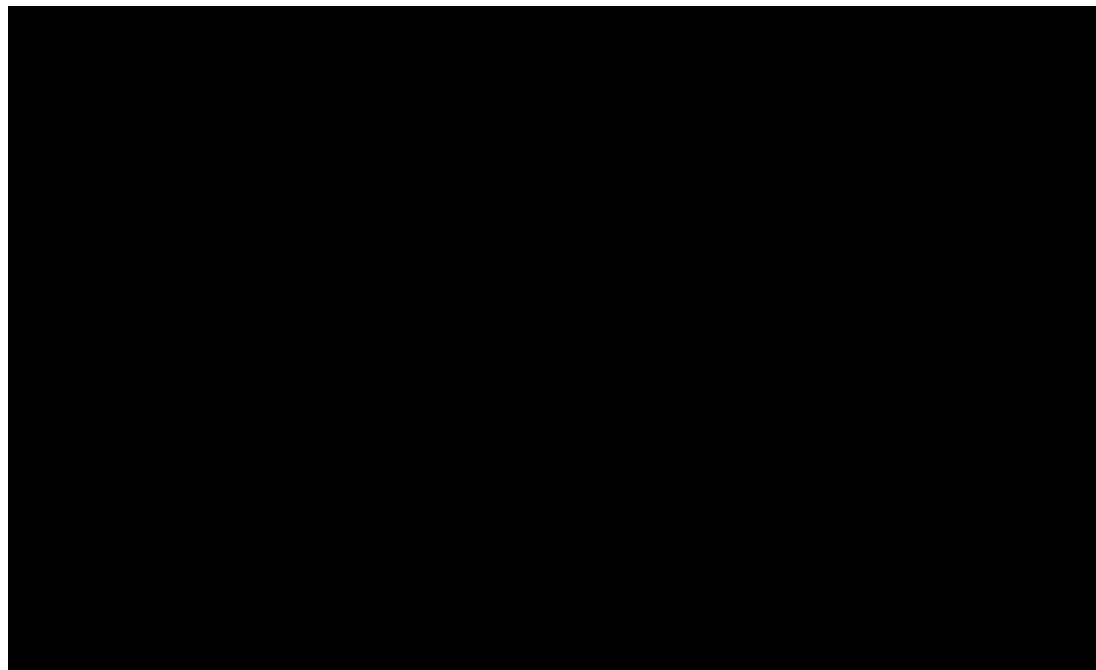
School of Data and Computer Science



# Lighting & Shading

---

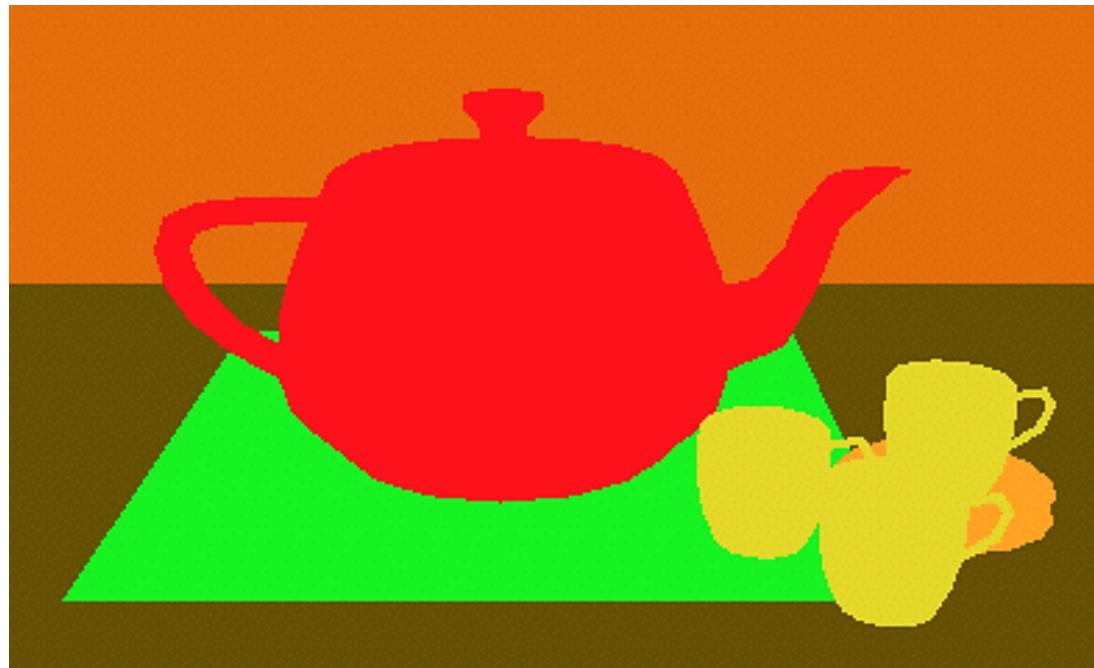
- Without light...
- We do not see much of our scene!



# Lighting & Shading

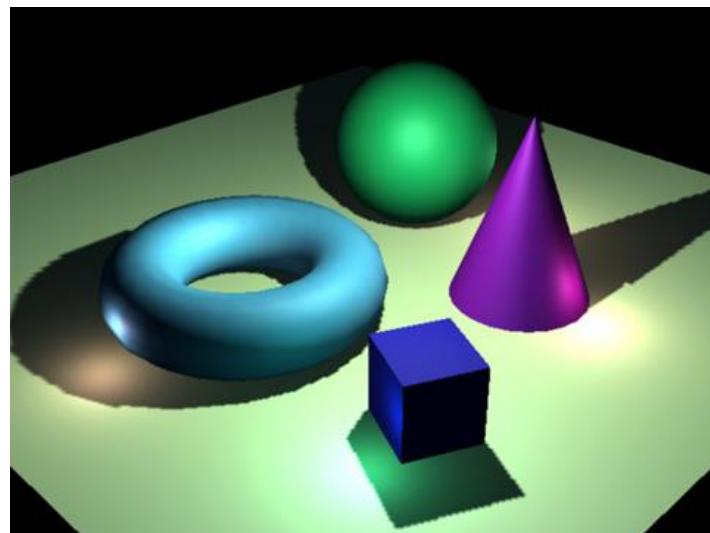
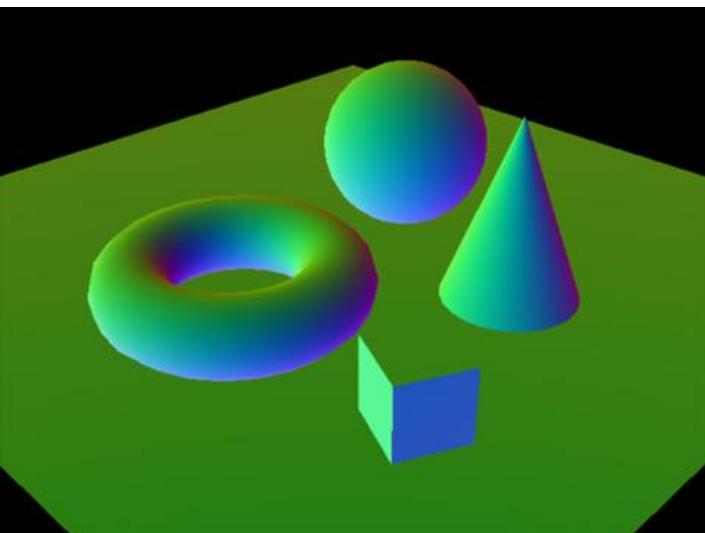
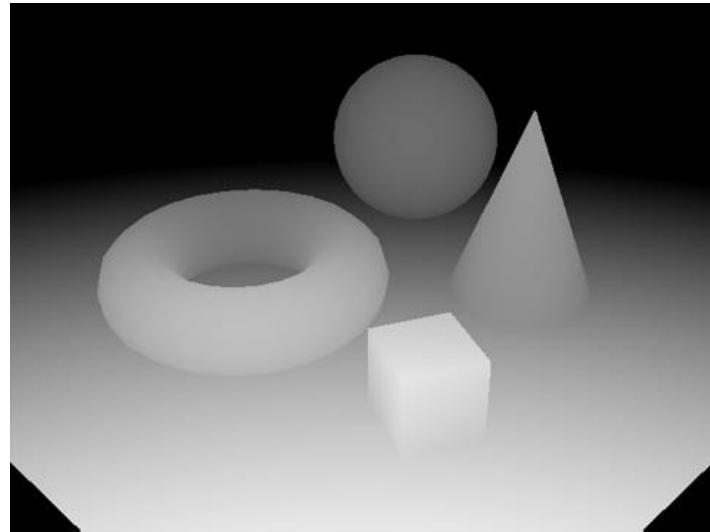
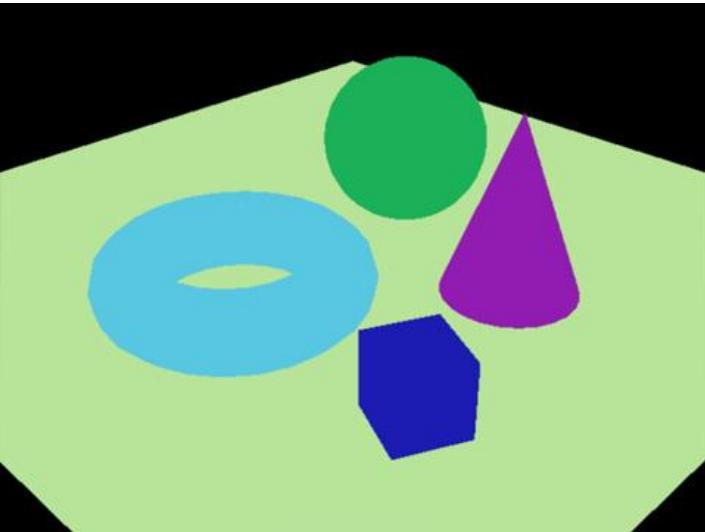
---

- Without shading...
- Objects do not look three dimensional



# Lighting & Shading

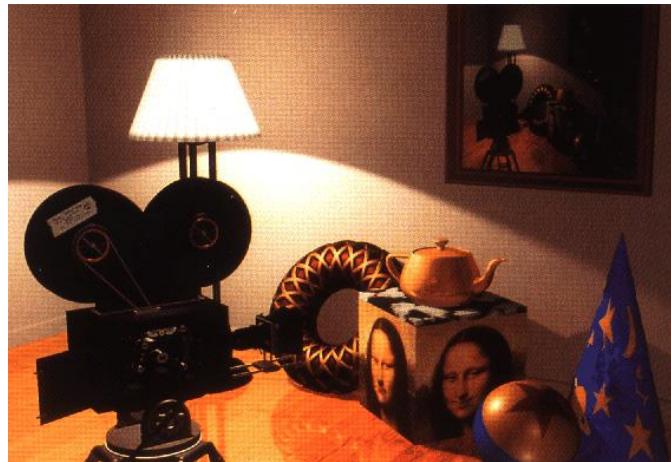
---



# Lighting/Shading

---

- Goal
  - Model the interaction of light with surfaces to render realistic images
  - Generate per (vertex/pixel) color

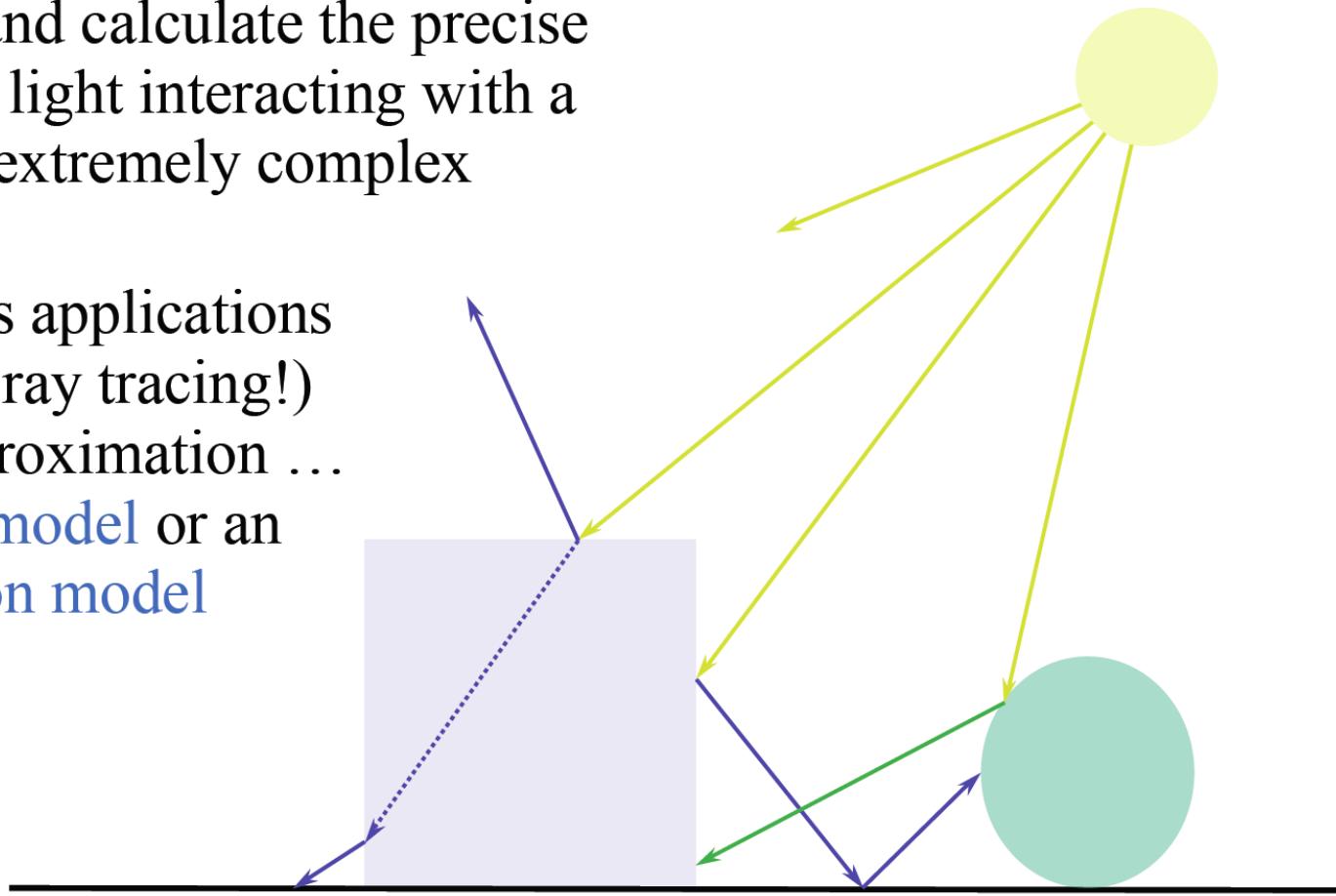


# Light interaction in a Scene

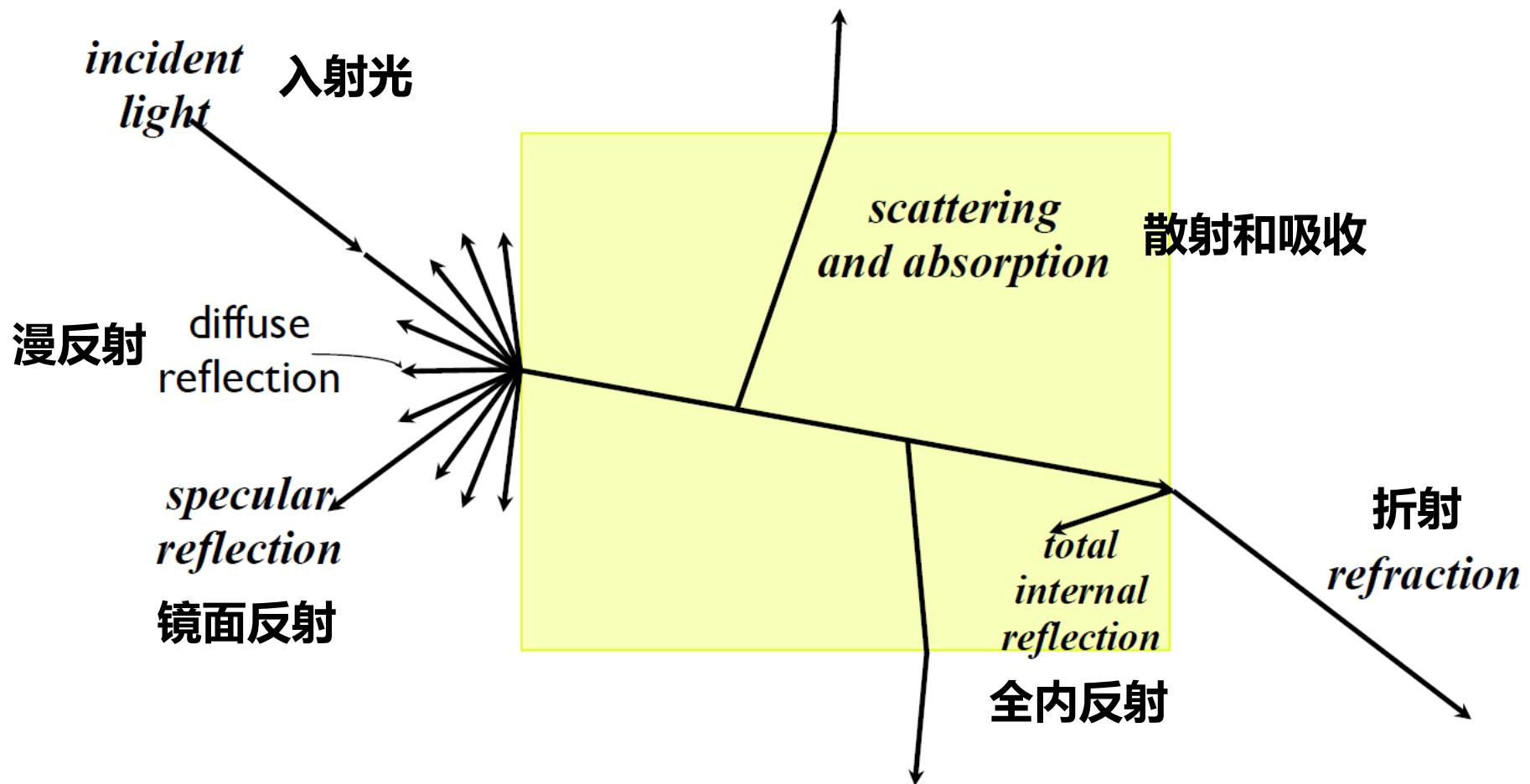
---

To simulate and calculate the precise physics of light interacting with a surface is extremely complex

Most graphics applications  
(including ray tracing!)  
use an approximation ...  
a **lighting model** or an  
**illumination model**



# Interaction of light with a Solid



# Illumination model

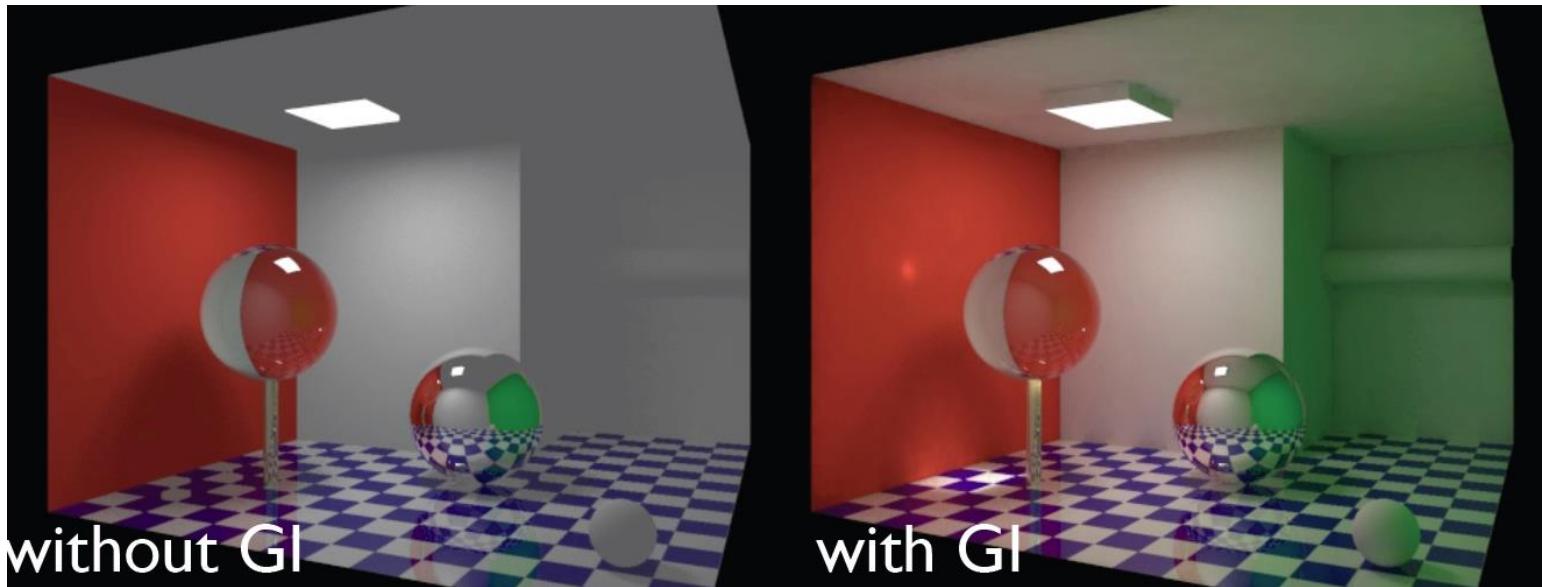
---

- Illumination model(also lighting model):
  - To calculate the color of an illuminated position on the surface of an object.
  - express the factors which determine the surface color at a given point on a surface
  - compute the color at a point in terms of both local and global illuminiaiton



# Illumination Models

- A surface point could be illuminated by
  - **local illumination**, light directly emitted by light sources
  - **global illumination**, light reflected from and transmitted through its own and other surfaces



# Illumination models

---

- Local illumination – Fast
  - “Fake (假伪)” –Ignore real physics, approximate the look
  - Compute at material, from light to viewer
  - Only direct illumination from emitters to surfaces
- Global illumination – Slow
  - It is illuminated by all the emitters and reflectors in the global scene
  - Physically based



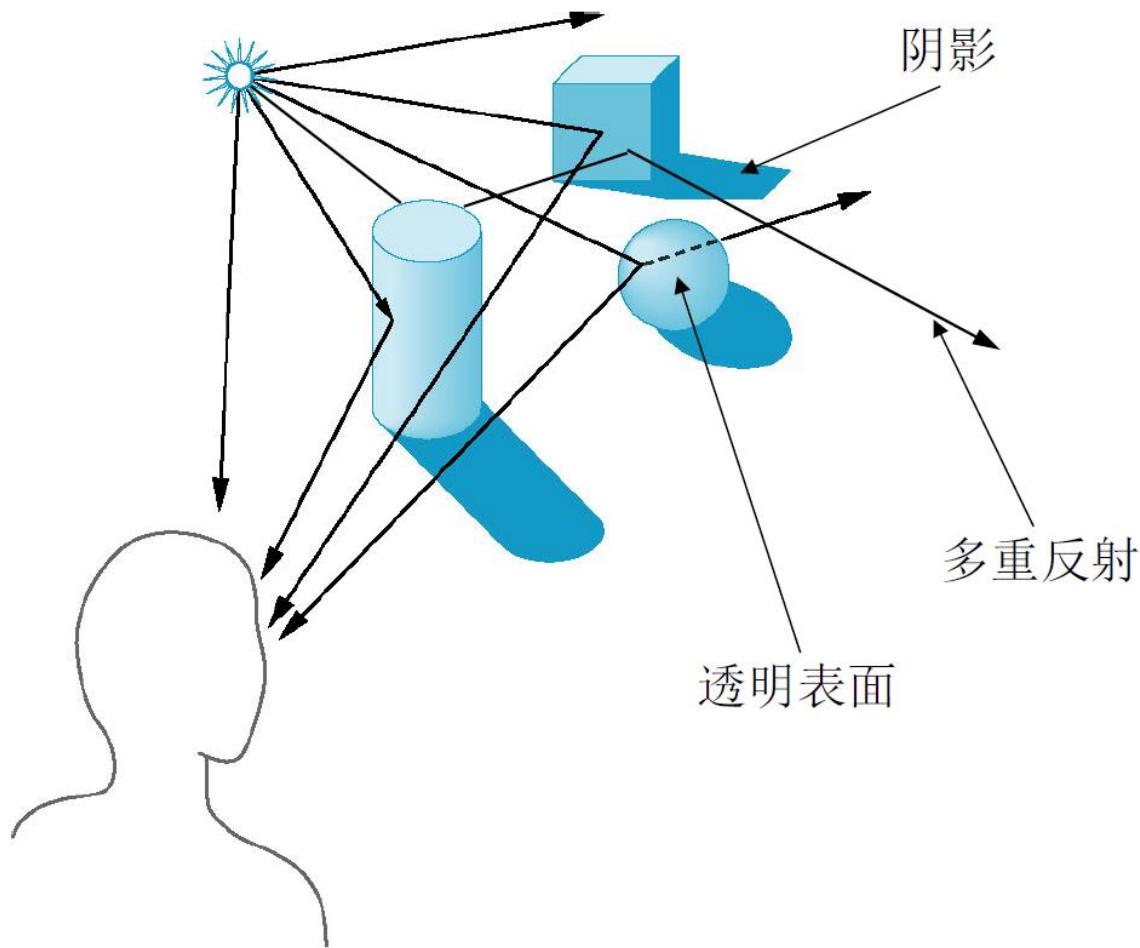
local



global



# Global Illumination



# Factors

---

- Light sources
  - Location, type & color
- Surface materials
  - How surfaces reflect light
- Transport of light
  - How light moves in a scene
- Viewer position



# Factors

---

- Light sources
  - Location, type & color
- Surface materials
  - How surfaces reflect light
- Transport of light
  - How light moves in a scene
- Viewer position
- How can we do this in the pipeline?



# The big picture (basic)

---

- Light: energy in a range of wavelengths
  - White light – all wavelengths
  - Colored (e.g. red) – subset of wavelengths
- Surface “color” – reflected wavelength
  - White – reflects all lengths
  - Black – absorbs everything
  - Colored (e.g. red) – absorbs all but the reflected color
- Multiple light sources add (energy sums)



# Color of Light

---

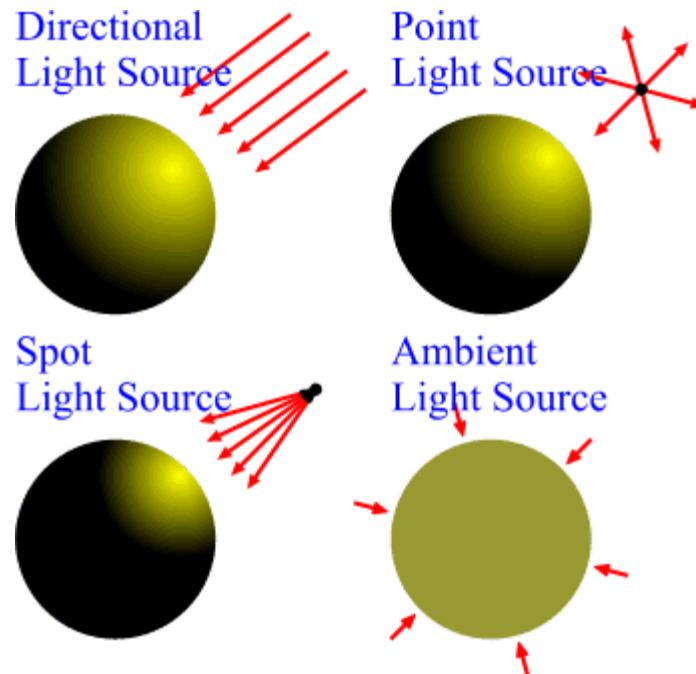
- Light not only emits a different amount of different frequencies of light, and their direction property with frequency can also be different
  - The real physical model will be very complicated
- Human visual system is based on the theory of the three primary colors
- In most applications, light could be represented as the intensities of Red, Green and Blue.
  - luminance function is :  $\mathbf{I} = [I_r, I_g, I_b]$



# Types of Light Sources

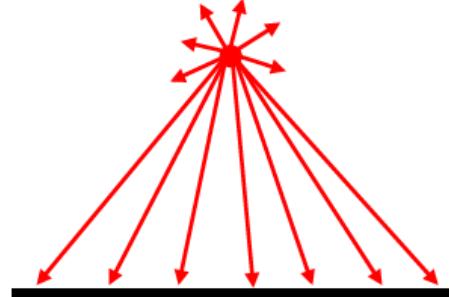
---

- Point source
- Parallel source
- Ambient lights
- Spotlights

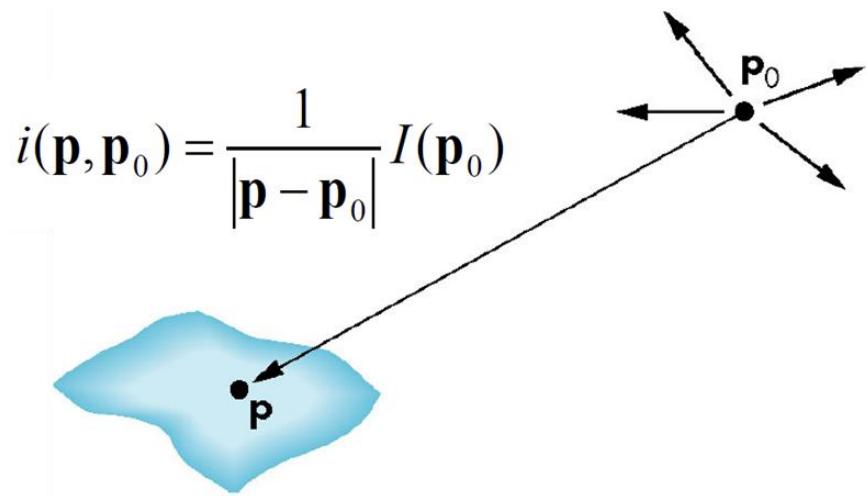


# Point source

- A point light source emits light equally in all directions from a single point
- The intensity function of point source:  $I(p_0) = [I_r(p_0), I_g(p_0), I_b(p_0)]$
- The intensity of P is inversely proportional to the distance between  $P_0$  and P.
- defined by **location only**



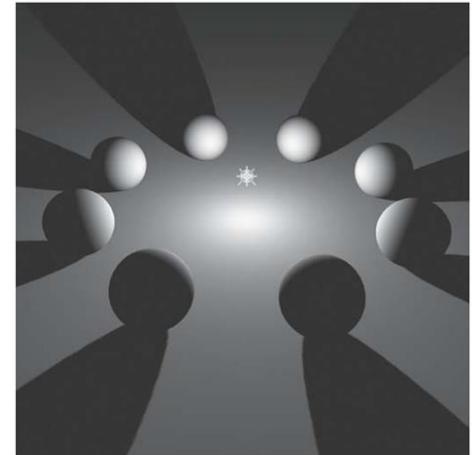
$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



# Application of Point Light

---

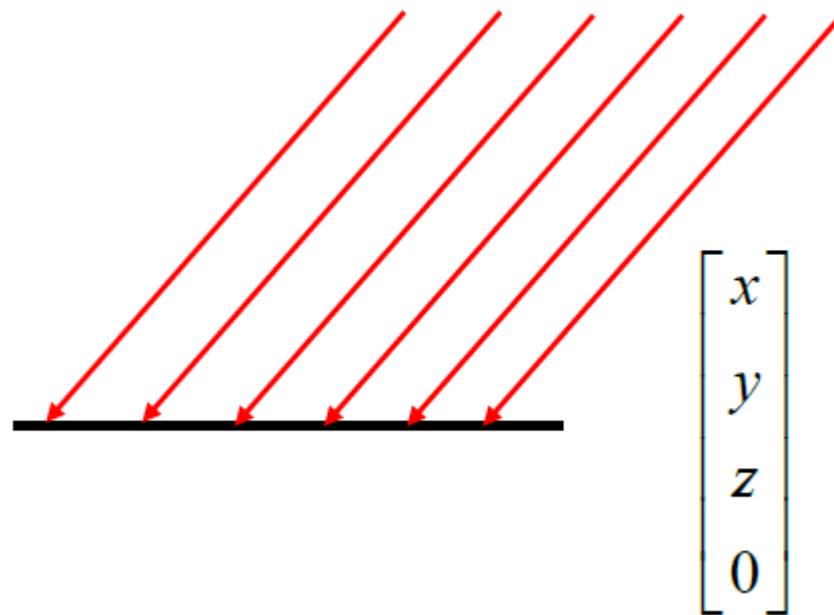
- Point sources are widely used in computer graphics for it is easy to use.
- Point sources can't simulate well the physical reality.
  - The contrast in the image is high when only has point sources in a sence: Objects appear too bright, or very dark.



# Parallel source

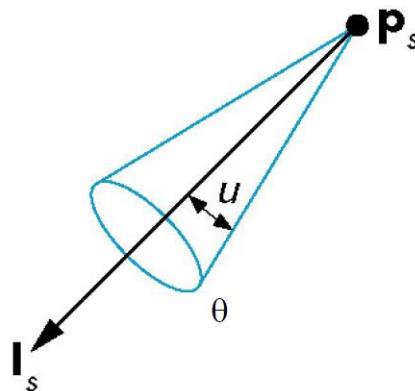
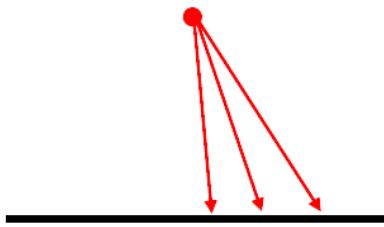
---

- light rays are parallel
- Rays hit a planar surface at **identical** angles
- May be modeled as point source at infinity
- Directional light
- defined by **direction** only

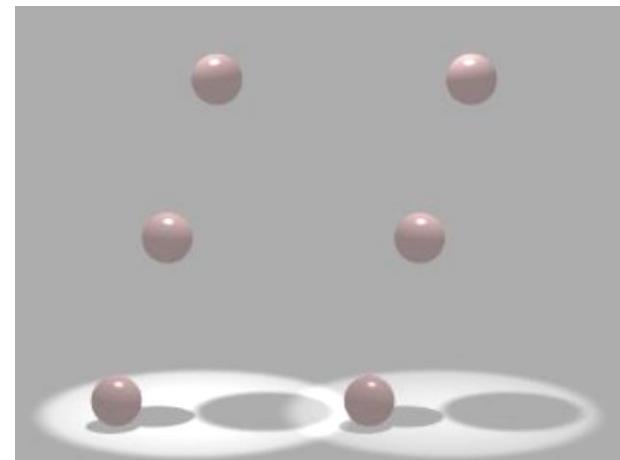


# Spotlights

- Spotlights are point sources whose intensity falls off directionally. **Point + direction + cutoff angle**
  - Requires color, point direction, falloff parameters
  - Supported by OpenGL

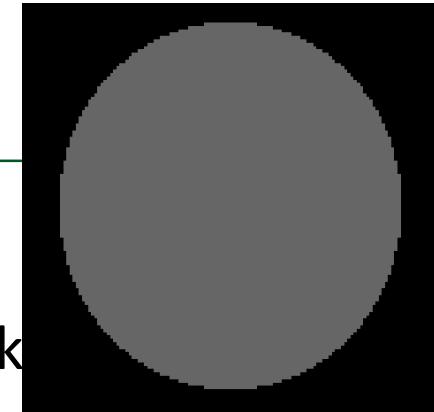


- 可以给点光源加上一定的限制得到
- 锥的顶点在  $P_s$ , 而中心轴方向为  $l_s$ 。
- 如果  $\theta = 180^\circ$ , 聚光灯成为点光源



# Ambient Lights

---



- In reality, parts of the objects not directly illuminated by the light source are not completely dark
  - e.g., the ceiling in this room, undersides of desks
- These parts are lit by global illumination i.e. light reflected by the surrounding environment; light that is reflected so many times that doesn't seem to come from anywhere
- Too expensive to calculate (in real time), so we add a **constant** light called an ambient light
  - No spatial or directional characteristics; illuminates all surfaces equally
  - Amount reflected depends on surface properties



# Ambient Lights

---

- For each sampled wavelength(R,G,B), the ambient light reflected from a surface depends on
  - The surface properties,  $K_{\text{ambient}}$
  - The intensity,  $I_{\text{ambient}}$ , of the ambient light source (constant for all points on all surfaces)
  - $I_{\text{reflected}} = K_{\text{ambient}} I_{\text{ambient}}$

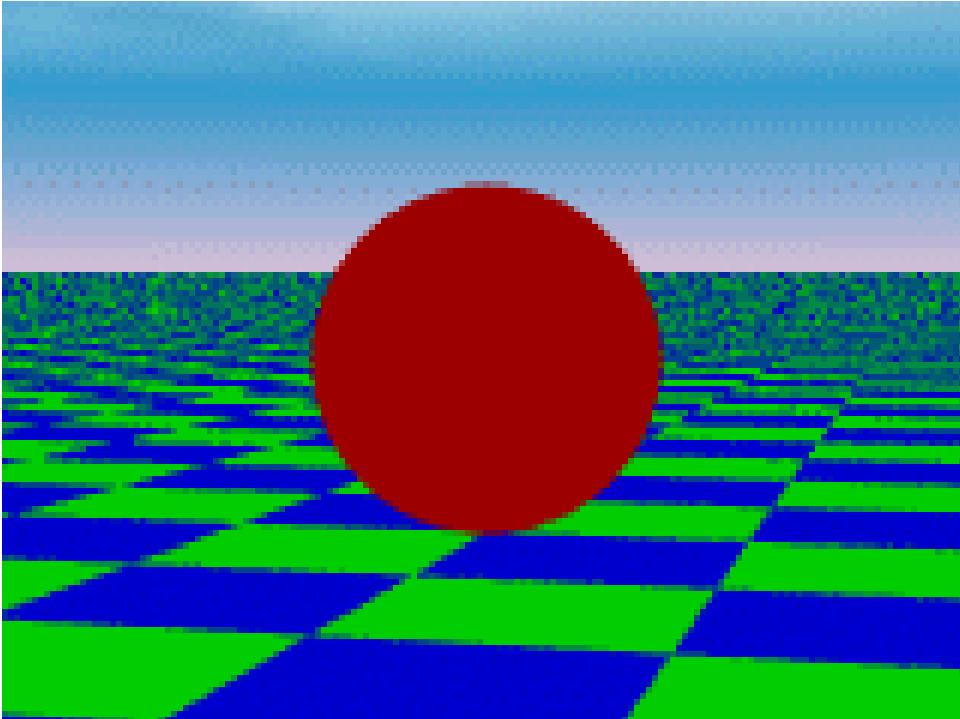
环境光强是由  $I_a = [I_{ar}, I_{ag}, I_{ab}]$  确定的，在每点的值完全相同



# Ambient Light Sources

---

- scene lit only with an ambient light source



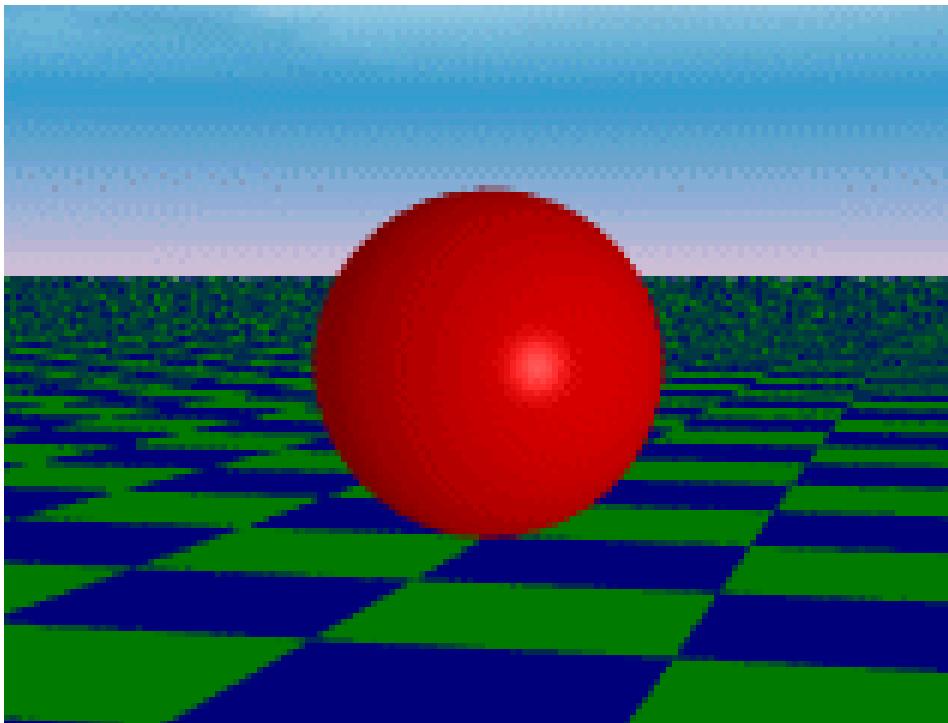
Light Position  
Not Important

Viewer Position  
Not Important

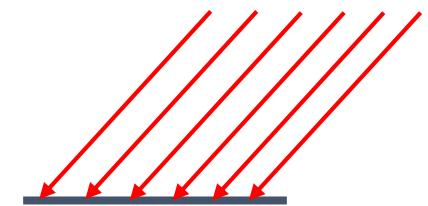
Surface Angle  
Not Important

# Directional Light Sources

- scene lit with ambient and directional light



Surface Angle  
Important



Light Position  
Not Important

Viewer Position  
Not Important

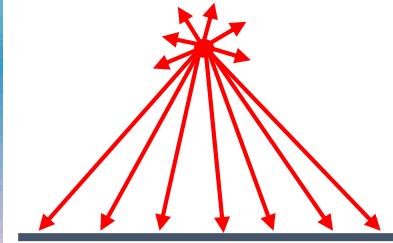
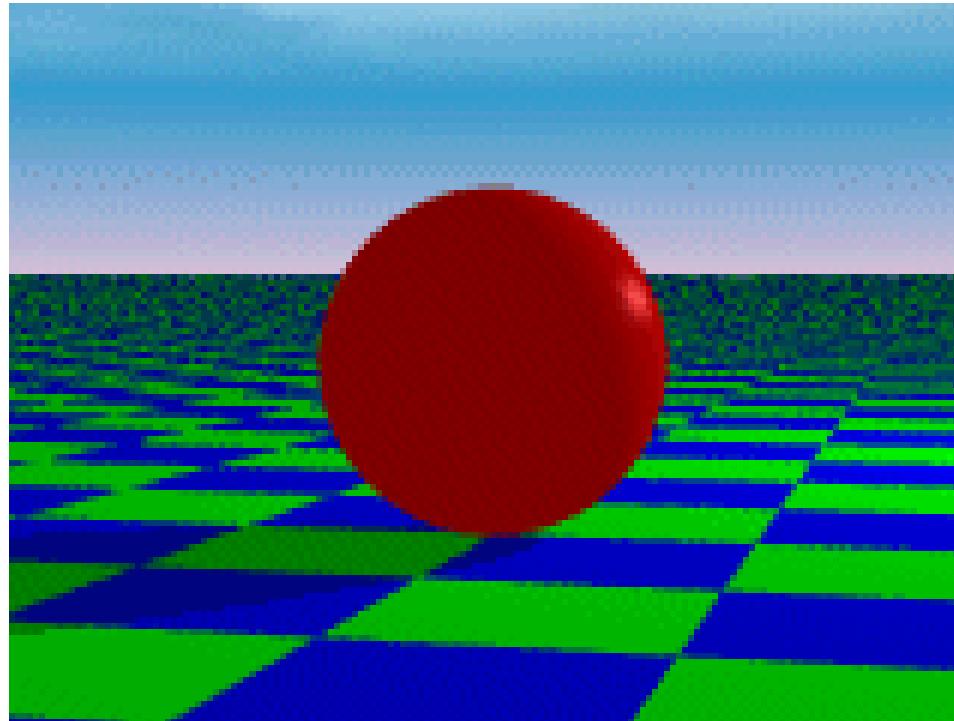
# Point Light Sources

- scene lit with ambient and point light source

Light Position  
Important

Viewer Position  
Important

Surface Angle  
Important



# Interaction Light Sources & Materials

---

- 照射在对象上的光线部分被吸收，部分被反射
- 如果对象是透明的，有些光被折射
- 反射部分的多少确定对象的颜色与亮度
- 对象表面在白光上看起来是红的，就是因为光线中的红色分量被反射，而其它分量被吸收
- 反射光被反射的方式是由表面的光滑程度和定向确定的



# Materials

---

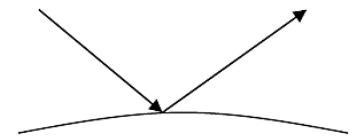
- Surface reflectance:
  - Illuminate surface point with a ray of light from different directions
  - How much light is reflected in each direction?



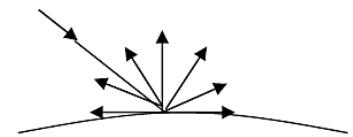
# Types of Reflection

---

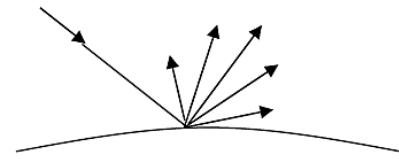
- **specular reflection** (a.k.a. *mirror* or *regular*) causes light to propagate without scattering.



- **diffuse reflection** sends light in all directions with equal energy.



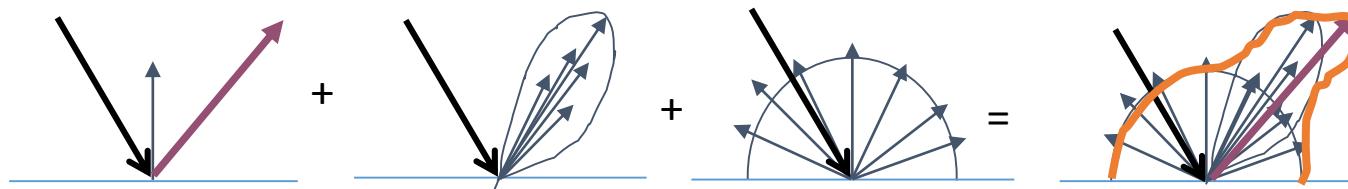
- **glossy/mixed reflection** is a weighted combination of specular and diffuse.



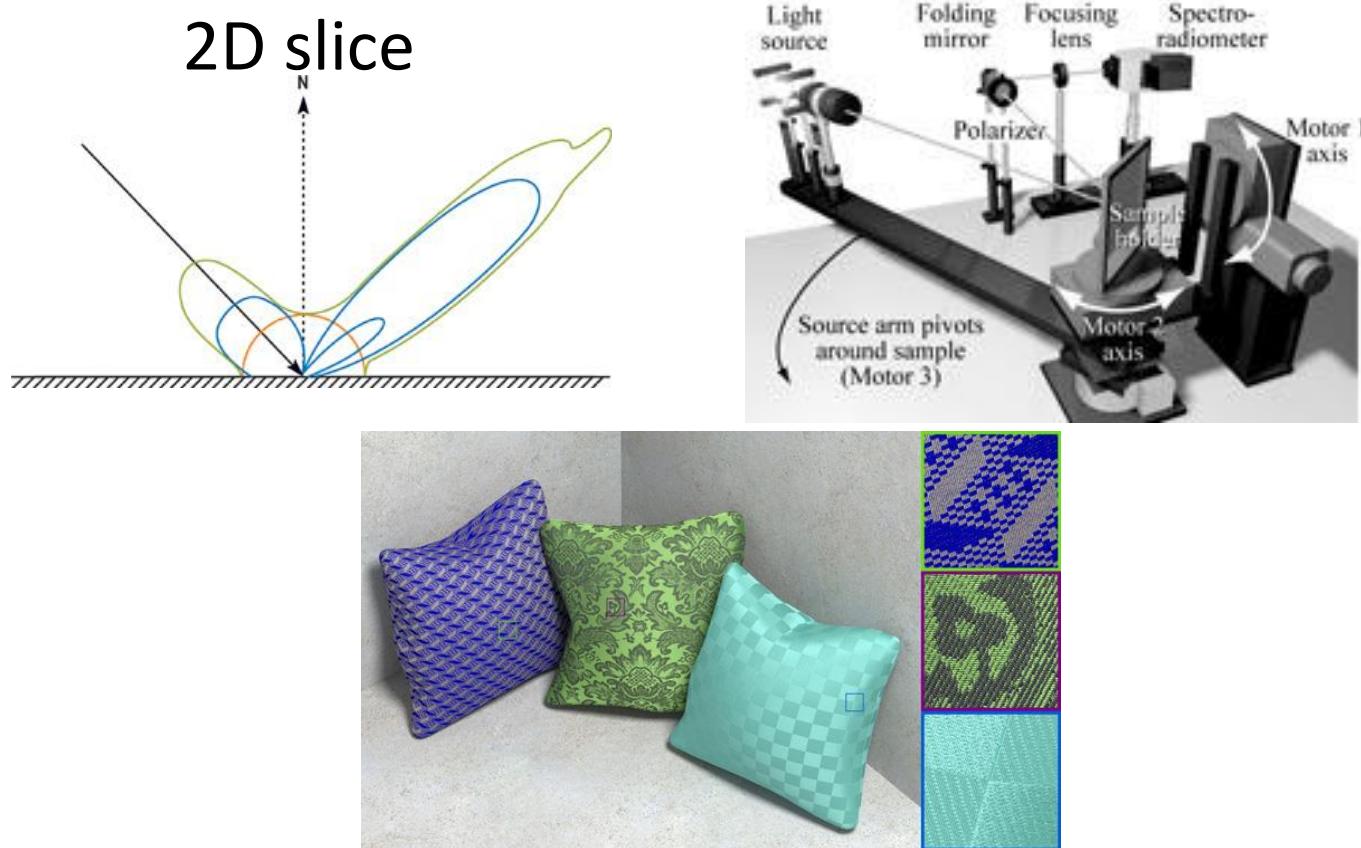
# Reflectance Distribution Model

---

- Most surfaces exhibit complex reflectance
  - Vary with incident and reflected directions.
  - Model with combination – known as BRDF
    - BRDF: *Bidirectional Reflectance Distribution Function*



# BRDF MEASUREMENTS/PLOTS(测量模型 )

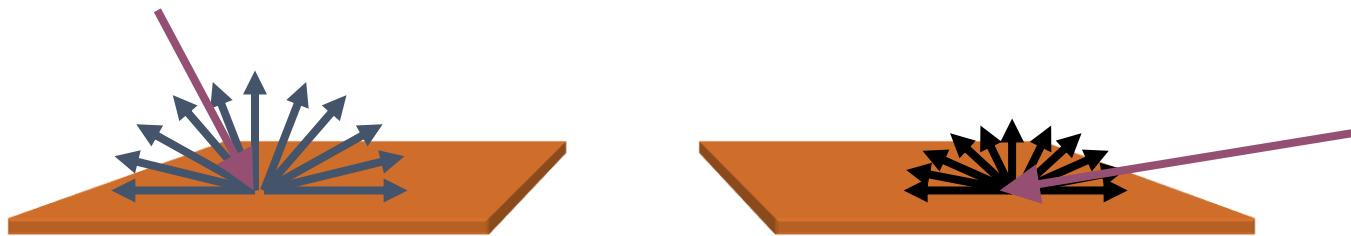


MERL等实验室使用仪器测量了上百种真实材质表面在不同光照角度和观察角度下的反射数据，并记录在[MERL BRDF Database](#)等数据库中。

# Physics of Diffuse Reflection

---

- Ideal diffuse reflection
  - very rough surface at the microscopic level
    - real-world example: chalk
  - microscopic variations mean incoming ray of light equally likely to be reflected in any direction over the hemisphere
  - what does the reflected intensity depend on?
    - Only depends on light direction!



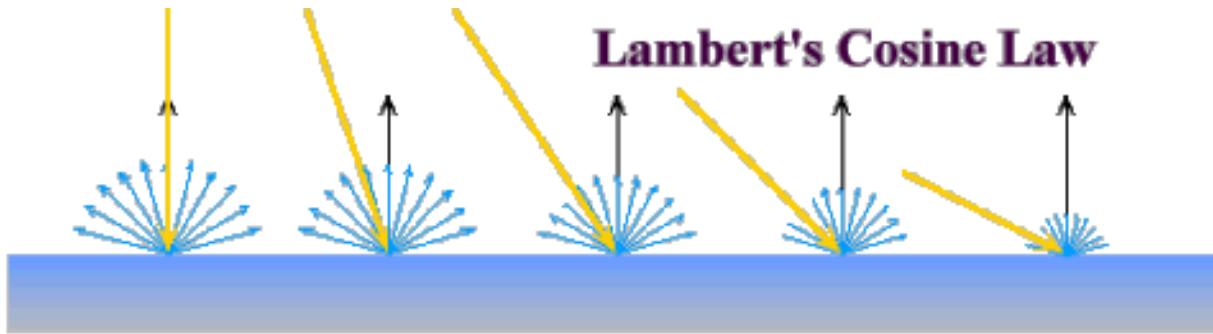
# Lambert's Cosine Law

---

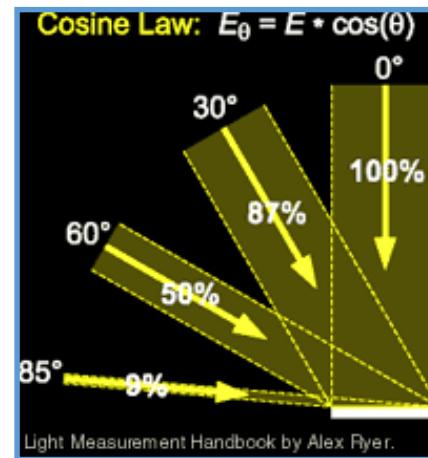
- Ideal diffuse surfaces reflect according to Lambert's cosine law:
  - the energy reflected by a small portion of a surface from a light source in a **given direction** is proportional to **the cosine of the angle between that direction and the surface normal**
- Reflected intensity
  - independent of viewing direction
  - depends on surface orientation w.r.t. light
- often called **Lambertian surfaces**



# Lambert光照模型



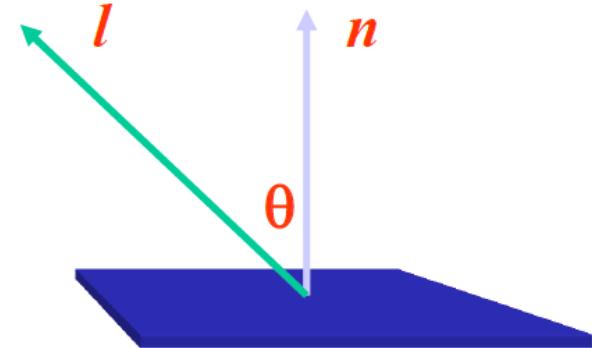
Intuitively: cross-sectional area of the “beam” intersecting an element of surface area is smaller for greater angles with the normal.



# Computing Diffuse Reflection

- depends on **angle of incidence**: angle between surface normal and incoming light

- $I_{\text{diffuse}} = k_d I_{\text{light}} \cos \theta$



- in practice use vector arithmetic

- $I_{\text{diffuse}} = k_d I_{\text{light}} (\mathbf{n} \cdot \mathbf{l})$

- always normalize vectors used in lighting!!!

- $\mathbf{n}$ ,  $\mathbf{l}$  should be unit vectors

- scalar (B/W intensity) or 3-tuple or 4-tuple (color)

- $k_d$ : diffuse coefficient

- $I_{\text{light}}$ : incoming light intensity

- $I_{\text{diffuse}}$ : outgoing light intensity (for diffuse reflection)



# Diffuse Lighting Examples

---

- Lambertian sphere from several lighting angles:



- need only consider angles from  $0^\circ$  to  $90^\circ$



# Specular Highlights

---



Michiel van de Panne

# Specular Reflection

---

- shiny surfaces (光泽曲面) exhibit specular reflection

- polished metal
- glossy car

- specular highlight

- bright spot from light shining on a **specular surface** (镜面)

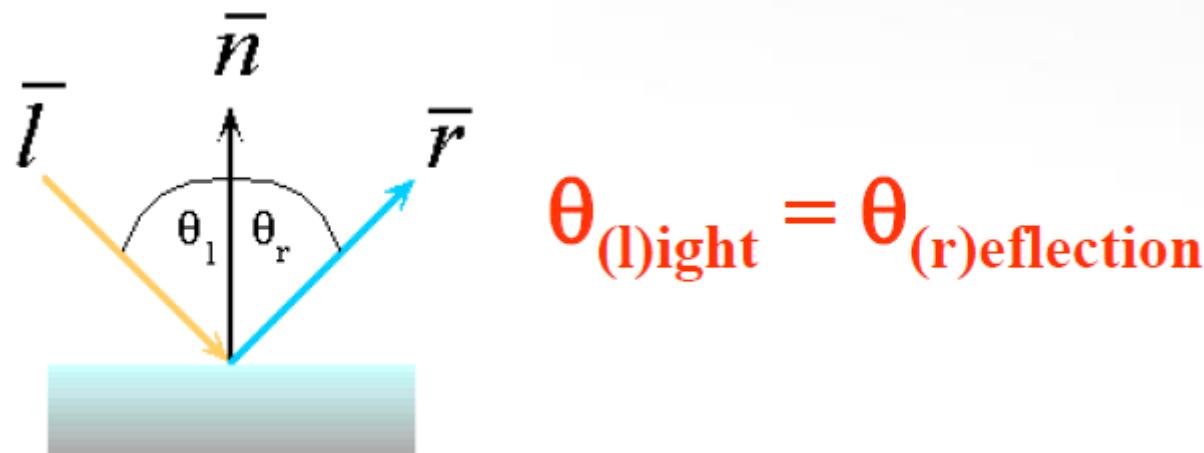
- view dependent

- highlight position is function of the viewer's position



# Optics of Reflection

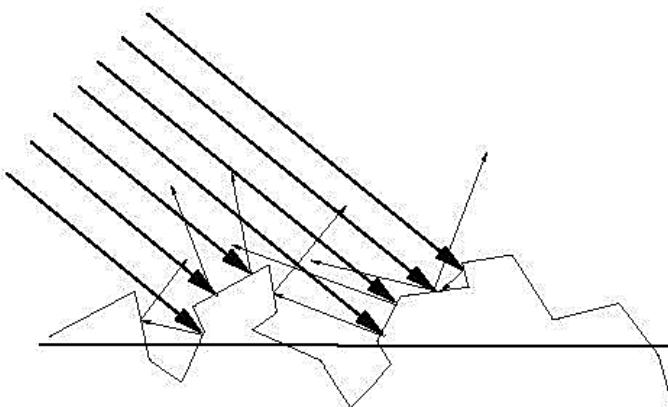
- Reflection follows **Snell's Law**:
  - incoming ray and reflected ray lie in a plane with the surface normal
  - angle the reflected ray forms with surface normal equals angle formed by incoming ray and surface normal



# Non-Ideal Specular Reflectance (Glassy Reflectance)

---

- Snell's law applies to perfect mirror-like surfaces, but aside from mirrors few surfaces exhibit perfect specularity
- How can we capture the “softer” reflections of surface that are glossy, not mirror-like?
- One option: model the **microgeometry** of the surface and explicitly bounce rays off of it



# Empirical Approximation

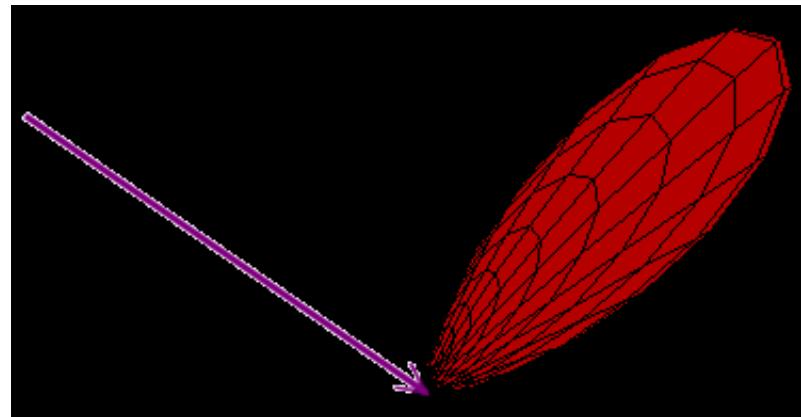
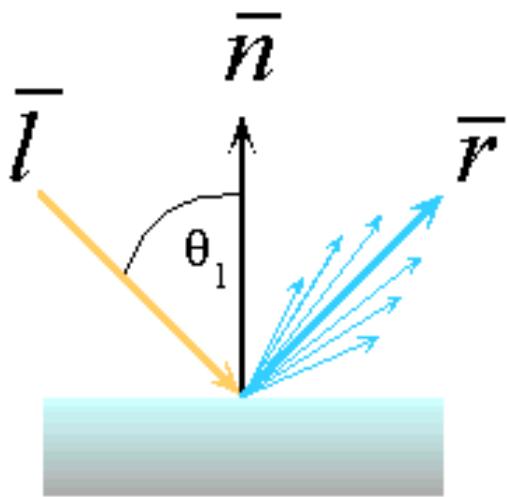
---

- we expect most reflected light to travel in direction predicted by Snell's Law
- but because of microscopic surface variations, some light may be reflected in a direction slightly off the ideal reflected ray
- as angle from ideal reflected ray increases, we expect less light to be reflected



# Empirical Approximation

- Angular falloff (角度下降)



- No physical basis, works ok in practice

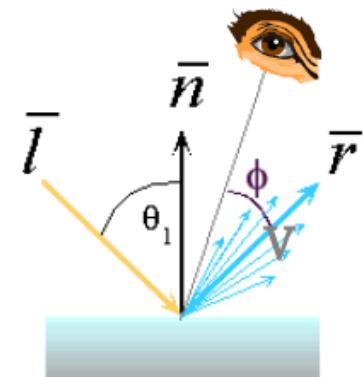


# Empirical Approximation

- The cos term of lighting can be computed using vector arithmetic:

$$I_{specular} = k_s I_{light} (\bar{v} \cdot \bar{r})^{n_{shiny}}$$

- $\bar{v}$  is the unit vector towards the viewer
- $\bar{r}$  is the ideal reflectance direction
- $K_s$ : specular component
- $I_{light}$ : incoming light intensity
- $n_{shiny}$ : purely empirical constant, varies rate of falloff(材质发光常数, 值越大, 表面越接近镜面, 高光面积越小。)



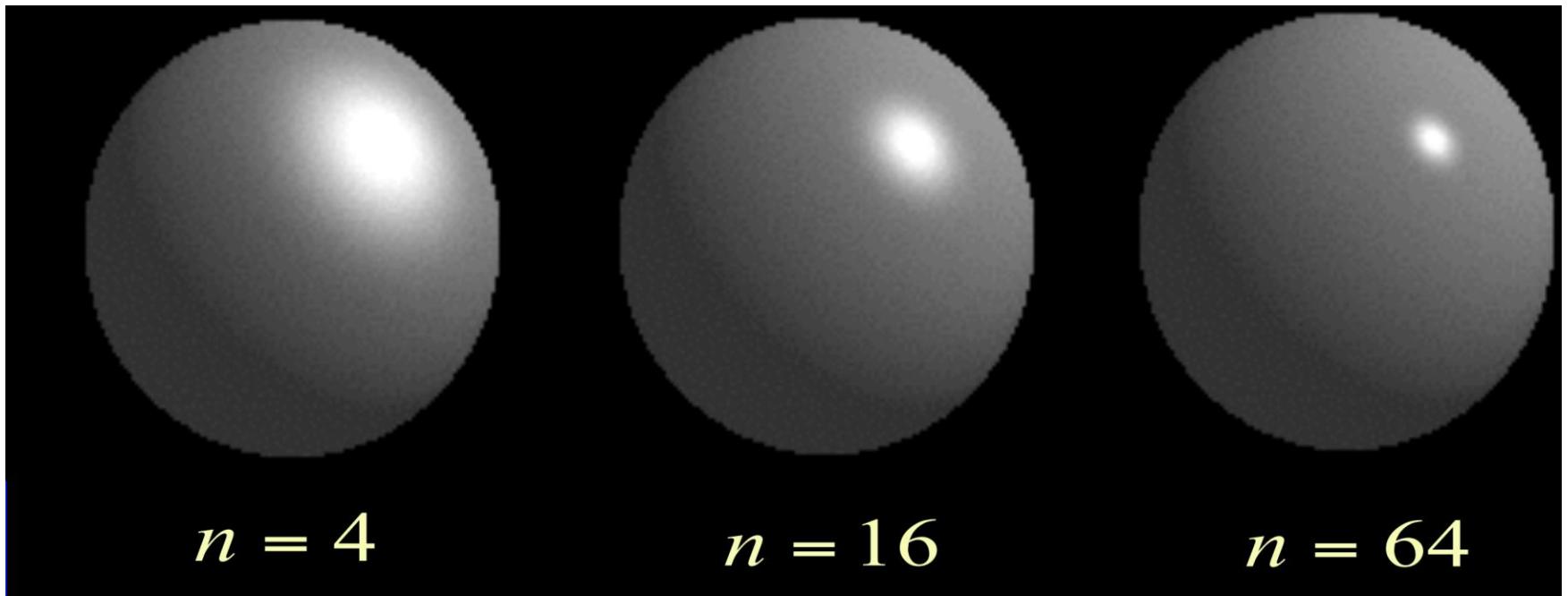
- How to efficiently calculate  $r$  ?



# Empirical Approximation

---

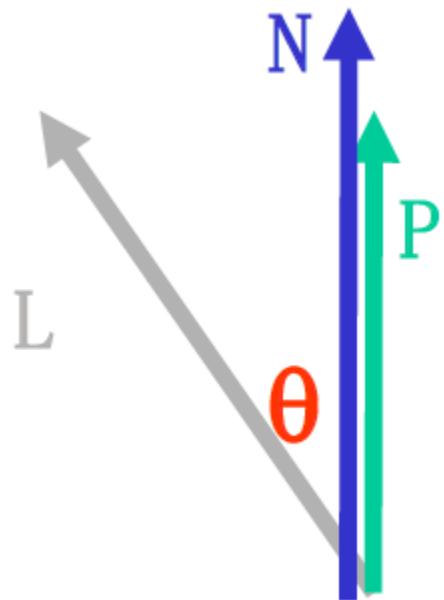
- The specular reflection exponent  $n$  controls the shine of the surface. The shine of the surface becomes sharper as  $n$  increases.



# Calculating R Vector

---

- $P = (L \cdot N) N$  :
  - projection of L onto N, L, N are unit length



# Calculating R Vector

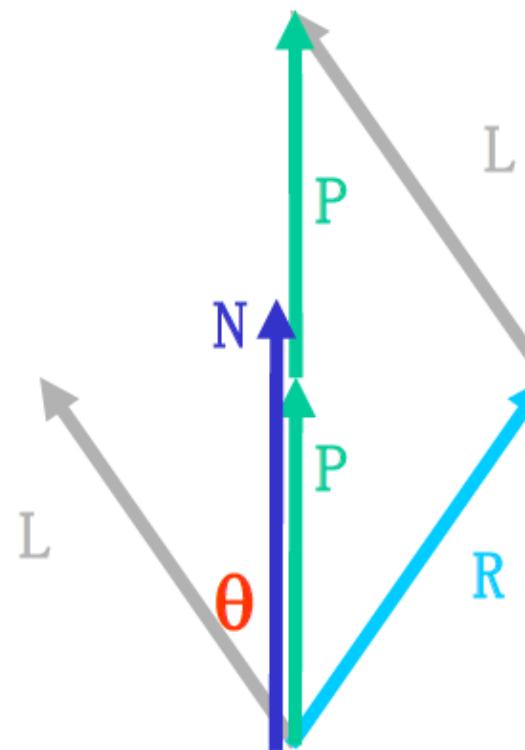
---

- $P = (L \cdot N) N$  :
  - projection of L onto N, L, N are unit length

$$2P = R + L$$

$$2P - L = R$$

$$2(N(N \cdot L)) - L = R$$

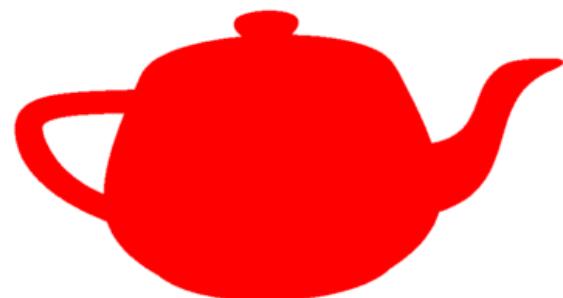


# Ambient Light

---

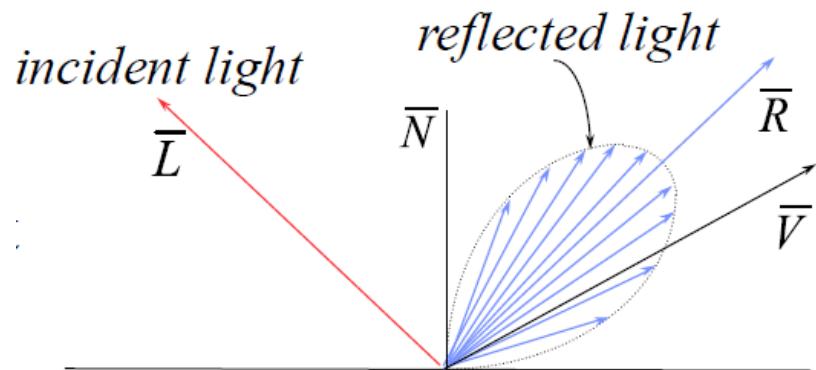
- No identifiable source or direction – environment light
  - hack for replacing true global illumination (diffuse interreflection: light bouncing off from other objects)
- Object illuminated with same light everywhere
  - Looks like silhouette
- Illumination equation
  - $I_a$  - ambient light intensity
  - $k_a$  - fraction of this light reflected from surface

$$I = I_a k_a$$



# Phong Illumination Model (经验模型)

- Developed by Phong Bui-Tuong (1975) is a popular model for non-perfect reflectors
- The light calculation depends on the viewing direction
- Reflected intensity is modeled in terms of
  - Diffuse reflection component
  - Specular reflection component
  - Ambient component
- 经验模型并不是基于物理原理，而是提出经验公式，通过调整参数来模拟光照。



# Phong Illumination Model

- The illumination equation in its simplest form is given as(综合了环境光、漫反射及镜面反射)

$$I = K_a I_a + K_d I_e \cos\alpha + k_s I_e \cos^n \gamma$$

The diagram illustrates the components of the Phong illumination model. At the top, the total illumination equation is shown with arrows pointing to four spheres representing different components: Ambient, Diffuse, Specular, and Phong Reflection. Below each sphere is a breakdown showing the individual components contributing to it. The breakdowns are: Ambient (a blue cross shape), Diffuse (a blue blob shape), Specular (a small white dot), and Phong Reflection (a large blue blob shape). The final result is the sum of these components.

Ambient      +      Diffuse      +      Specular      =      Phong Reflection



# Multiple Lights

---

- Light is **linear**
  - If multiple rays illuminate the surface point the result is just the sum of the individual reflections for each ray

$$\sum_p I_p (k_d (n \cdot l_p) + k_s (r_p \cdot v)^n)$$

- Multiple lights of Phong Illumination Model

$$I = K_a I_a + \sum_{i=1}^m I_i (K_d (\vec{n}, \vec{l}) + k_s (\vec{v}, \vec{r})^n)$$



# Colored objects

---

- Light has color
- Interacts with object color (r,g,b)

$$I = I_a k_a$$

$$I_a = (I_{ar}, I_{ag}, I_{ab})$$

$$k_a = (k_{ar}, k_{ag}, k_{ab})$$

$$I = (I_r, I_g, I_b) = (I_{ar}k_{ar}, I_{ag}k_{ag}, I_{ab}k_{ab})$$

- Blue light on white surface?
- Blue light on red surface?



# Light and Material Specification

---

- Light source: amount of RGB light emitted
  - value = intensity per channel
    - e.g., (1.0,0.5,0.5)
  - every light source emits ambient, diffuse, and specular light
- Materials: amount of RGB light reflected
  - value represents percentage reflected
    - e.g., (0.0,1.0,0.5)
- Interaction: multiply components
  - Red light (1,0,0) x green surface (0,1,0) = black (0,0,0)



# Colored objects

---

- The color of objects is set by appropriate setting of the ambient and the diffused reflection coefficients
- Specular coefficient is not decided by the color
- There are now three intensity equations

$$I_r = I_a k_{ar} + I_p [k_{dr} (\bar{N} \bullet \bar{L}) + k_s (\bar{R} \bullet \bar{V})^n]$$

$$I_g = I_a k_{ag} + I_p [k_{dg} (\bar{N} \bullet \bar{L}) + k_s (\bar{R} \bullet \bar{V})^n]$$

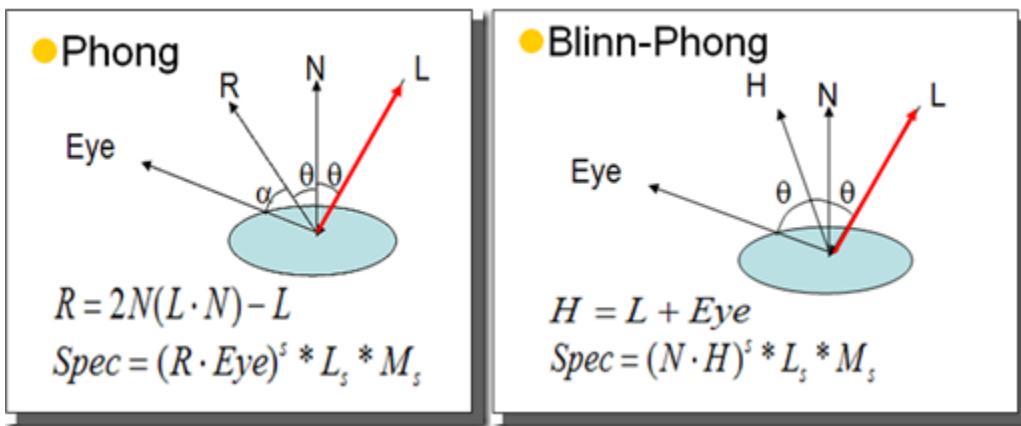
$$I_b = I_a k_{ab} + I_p [k_{db} (\bar{N} \bullet \bar{L}) + k_s (\bar{R} \bullet \bar{V})^n]$$

- Summarizing these three equations as single expression

$$I(r, g, b) = I_a k_a(r, g, b) + I_p [k_d(r, g, b)(\bar{N} \bullet \bar{L}) + k_s (\bar{R} \bullet \bar{V})^n]$$



# Blinn-Phong Illumination Model



- 相比较Phong模型, Blinn-Phong模型只适用 $N \bullet H$ 替换了 $V \bullet R$ , 但却获得了明显的提高, 它能提供比Phong更柔和、更平滑的高光, 而且速度上也更快, 因此成为很多CG软件中默认的光照渲染方法, 同时也被集成到大多数的图形芯片中, 而且在OpenGL和DirectX 3D的渲染管线中, 它也是默认的光照模型。



# When to apply Lighting Model?

---

per polygon  
“flat shading”

per vertex  
“Gouraud  
shading”

per pixel  
“per pixel lighting”  
“Phong shading”

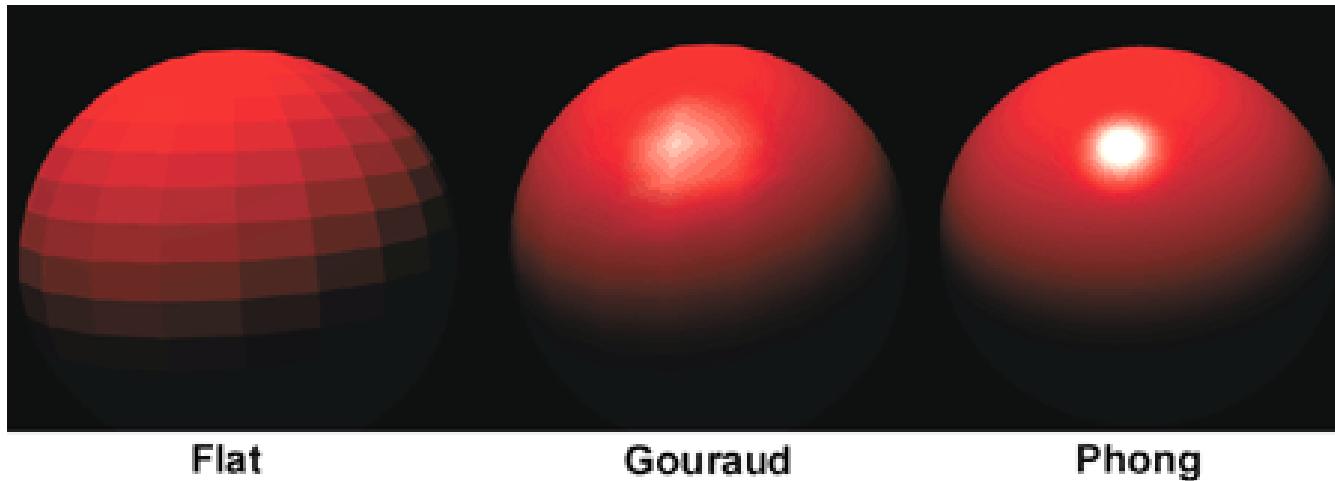


Image © Intergraph Computer Systems

# Shading

---

- Shading is the process of determining the colors of all the **pixels** covered by a surface using an illumination model
- Simplest method is to
  - determine surface visible at each pixel
  - compute normal of the surface
  - evaluate light intensity and color using an illumination model
- This is quite expensive. The shading methods could be made efficient by customizing for specific surface representation



# Shading

---

- Shading Models give a technique to determine the colors of all the pixels covered by a surface using appropriate illumination model
- Polygonal meshes are commonly used for representing complex surfaces
- The geometric information is available only at the vertices of a polygon
- Interpolative shading models could be used to increase the efficiency substantially



# Applying Illumination

---

- we now have an illumination model (lighting model ) for a point on a surface
- if surface defined as mesh of polygonal facets, **which points should we use?**
  - fairly expensive calculation
  - several possible answers, each with different implications for visual quality of result



# Flat Shading ( Constant Shading )

---

- Simplest approach calculates illumination at a single point for each polygon
- constant color



- obviously inaccurate for smooth surfaces

# Flat Shading

---

- One polygon receives only one intensity value
- Illumination model is applied only once for each polygon
- Makes the following assumptions
  - light source is at infinity, so  $\bar{N} \cdot \bar{L}$  is constant across a polygon face
  - viewer is at infinity, so  $\bar{R} \cdot \bar{V}$  is constant across the polygon face
  - polygon represents the actual surface being modeled



# Flat Shading

---

- It is a fast technique for shading as it involves very less calculations
- If the polygons are very small(say one pixel large) when projected on the screen then the result is as good as any interpolative technique
- Usually used of coarse preview of scenes
- obviously inaccurate for smooth surfaces for most cases

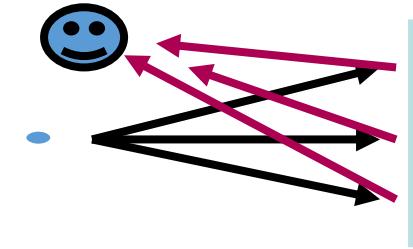
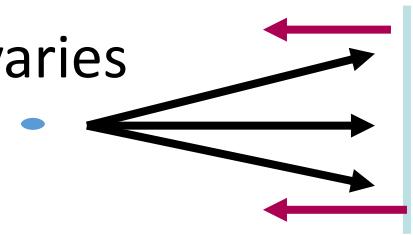


# Flat Shading Approximations

- if an object really is faceted, is this accurate?
- no!
  - for point sources, the direction to light varies across the facet

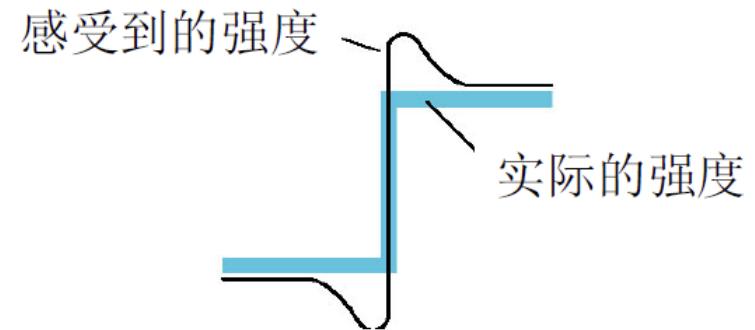
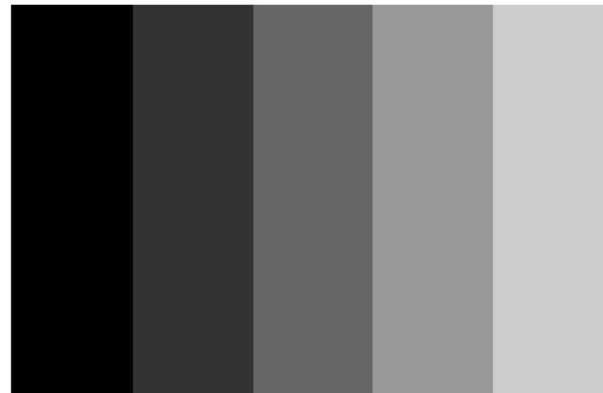


- for specular reflectance, direction to eye varies across the facet



# Flat Shading

- 人类视觉系统对光强的变化非常敏感
  - 称为lateral inhibition (侧抑制) 性质
- 观察到下图边界上的条状效果，称为Mach带
- 没有办法避免这种情形，只有给出更光滑的明暗处理方法

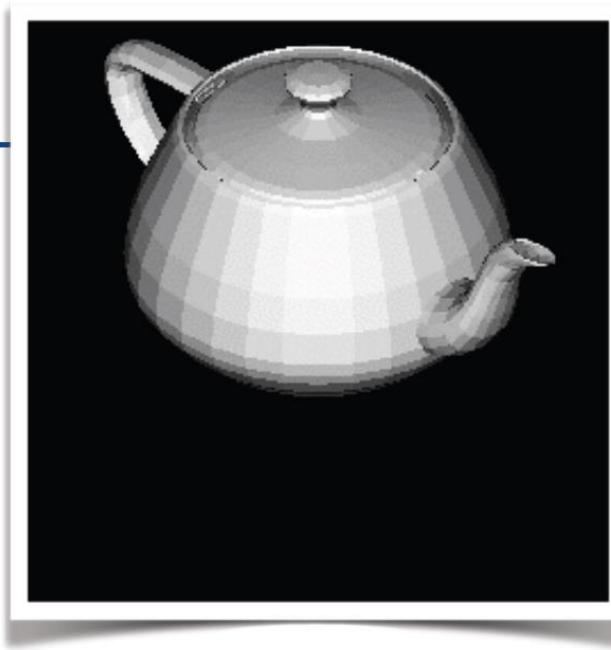


# Flat Shading in OpenGL

---

OpenGL uses the normal of the **first vertex** of a single polygon to determine the color.

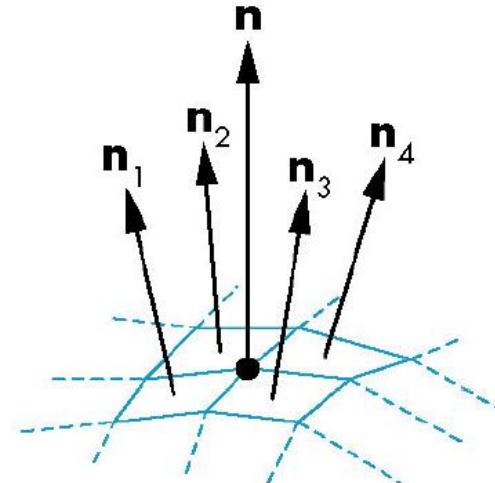
```
glShadeModel(GL_FLAT);
```



# Gouraud Shading

- It is an interpolative shading method, also called **intensity interpolation shading** or **color interpolation shading**
- Involves the following steps
  - Normals are computed at the vertex as **the average of the normals of all the faces meeting at that vertex**
  - **Intensity** at each vertex is calculated using **the normal** and an illumination model
  - For each polygon the intensity values for the interior pixels are calculated by linear interpolation of the intensities at the vertices

$$n = \frac{n_1 + n_2 + n_3 + n_4}{|n_1 + n_2 + n_3 + n_4|}$$



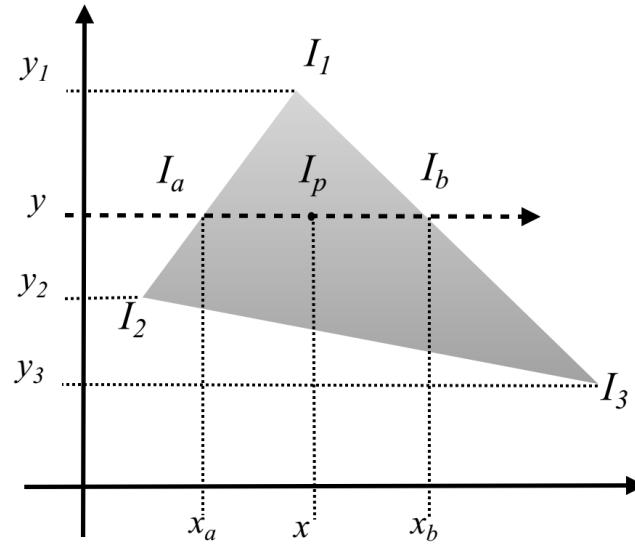
# Gouraud Shading

- This is the most common approach
  - perform Phong lighting at the vertices
  - Linearly interpolate the resulting colors over faces
    - Along edges
    - Along scanlines
  - Intensity  $I_p$  at a point is calculated as

$$I_a = I_1 + (I_2 - I_1) \frac{y - y_1}{y_2 - y_1}$$

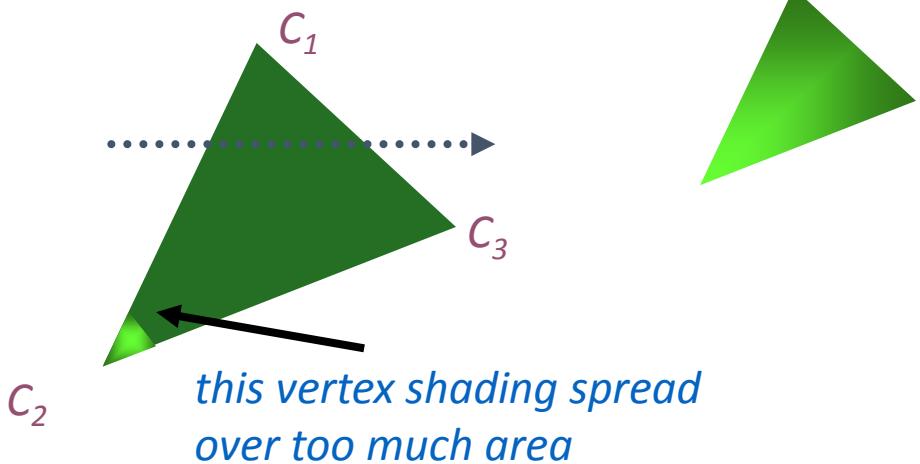
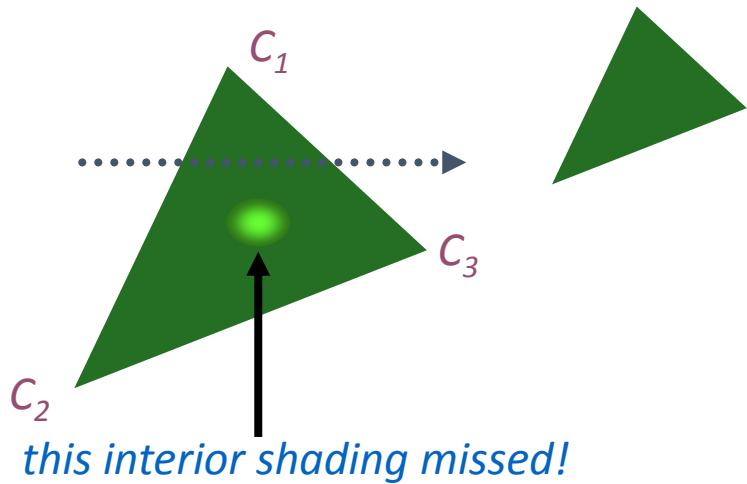
$$I_b = I_1 + (I_3 - I_1) \frac{y - y_1}{y_3 - y_1}$$

$$I_p = I_a + (I_b - I_a) \frac{x - x_a}{x_b - x_a}$$



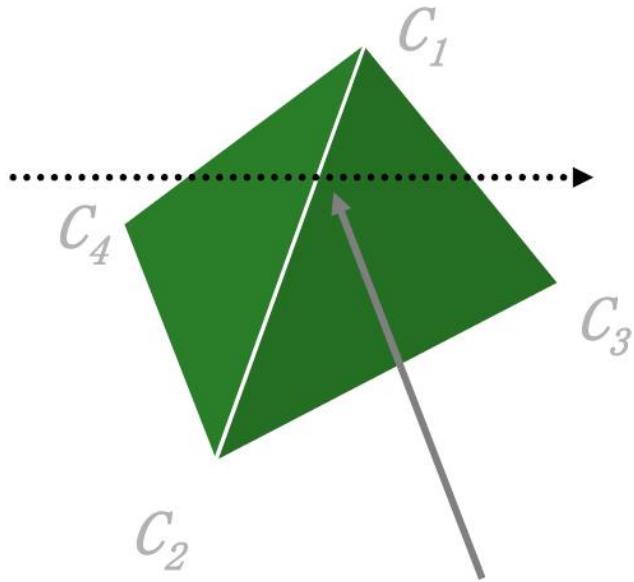
# Gouraud Shading Artifacts

- Might miss specular highlights, if the highlight doesn't fall at the vertex
- Lacks accurate specular component
  - if included, will be averaged over entire polygon

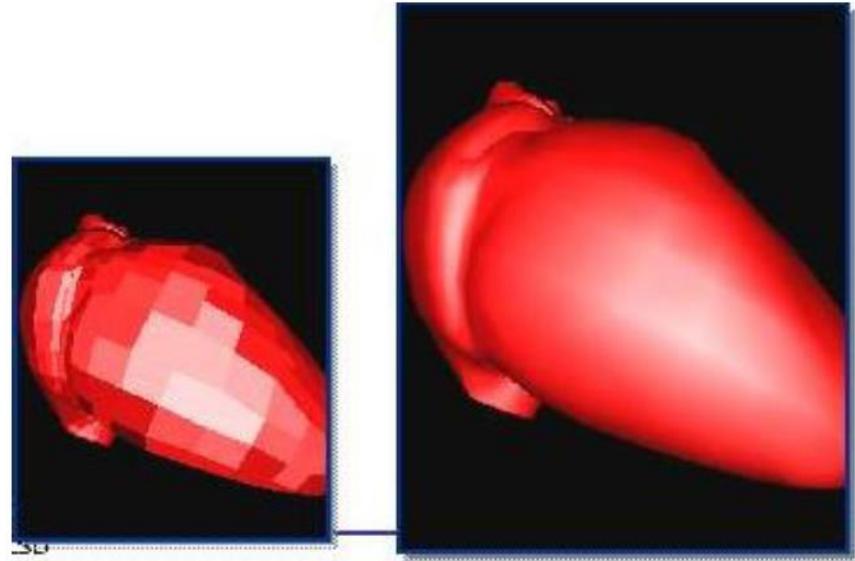


# Gouraud Shading Artifacts

- Mach bands(马赫带效应)



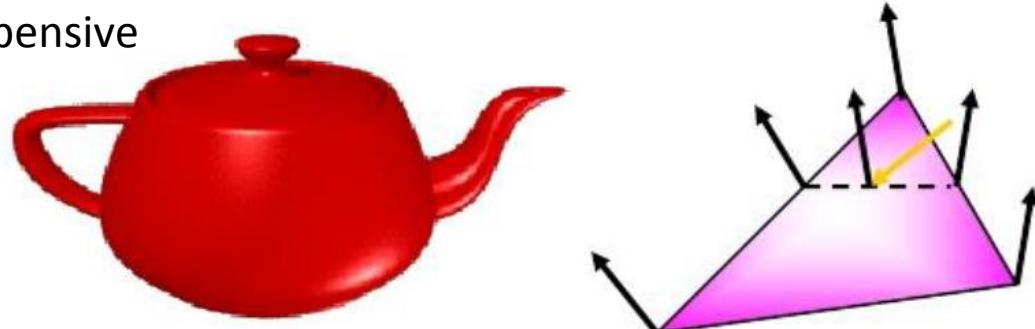
*Discontinuity in rate  
of color change  
occurs here*



`glShadeModel(GL_SMOOTH);`

# Phong Shading

- Phong shading is not the same as Phong Illumination Model, though they are sometimes mixed up
  - Phong Illumination(Phong lighting): the empirical model we've been discussing to calculate illumination **at a point** on a surface
  - Phong shading: linearly interpolating the surface normal across the facet, applying the Phong Illumination model **at every pixel**(also called normal-vector interpolation shading)
    - Same input as Gouraud shading
    - Usually more smooth-looking results:
    - But, considerably more expensive



# Phong Shading

---

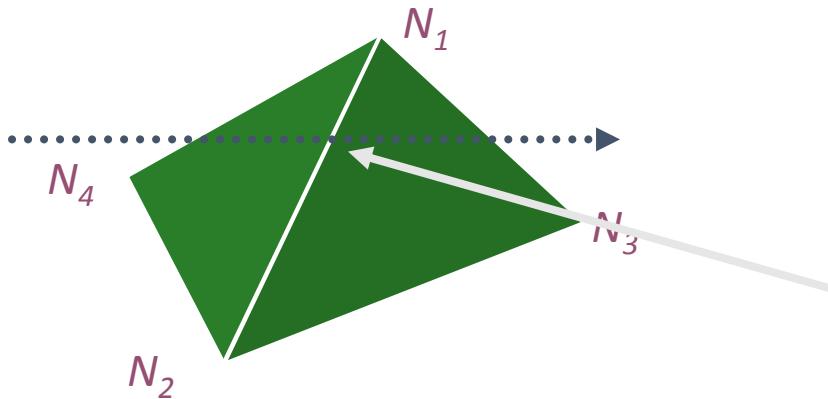
- Involves the following steps
  1. Normals are computed at the vertex as the average of the normals of all the faces meeting at that vertex
  2. For each polygon  
the value of the normal for the surface occupied by each interior pixel is calculated by linear interpolation of the normals at the vertices
- Specular reflections are also incorporated
- Interpolation of normals is done exactly like intensity interpolation in Gouraud shading



# Phong Shading

- linearly interpolate the vertex normals
  - compute lighting equations at each pixel

$$I_{total} = k_a I_{ambient} + \sum_{i=1}^{\# lights} I_i \left( k_d (\mathbf{n} \times \mathbf{l}_i) + k_s (\mathbf{v} \times \mathbf{r}_i)^{n_{shiny}} \right)$$



remember: normals used in diffuse and specular terms

discontinuity in normal's rate of change harder to detect

# Phong Shading Difficulties

---

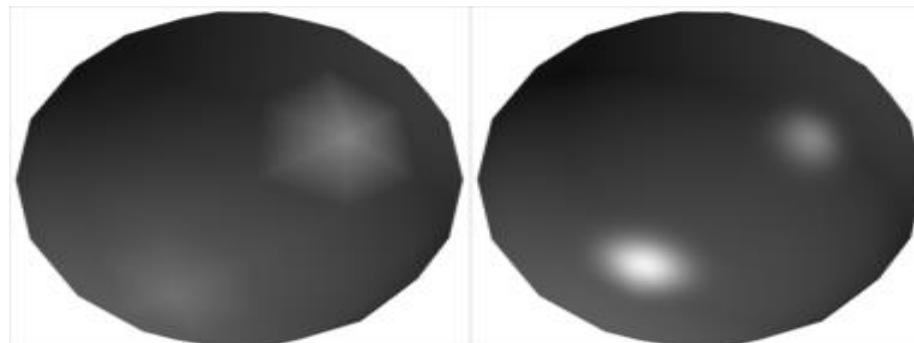
- computationally expensive
  - per-pixel vector normalization and lighting computation!
  - floating point operations required
- no direct support in pipeline hardware
  - but can be simulated with texture mapping



# Shading Artifacts: Silhouettes

---

- polygonal silhouettes remain
  - The silhouette edge of the mesh is always a polygon
  - Solution :
    - finer subdivision for the entire surface
    - finer subdivision only along silhouette (view dependent)

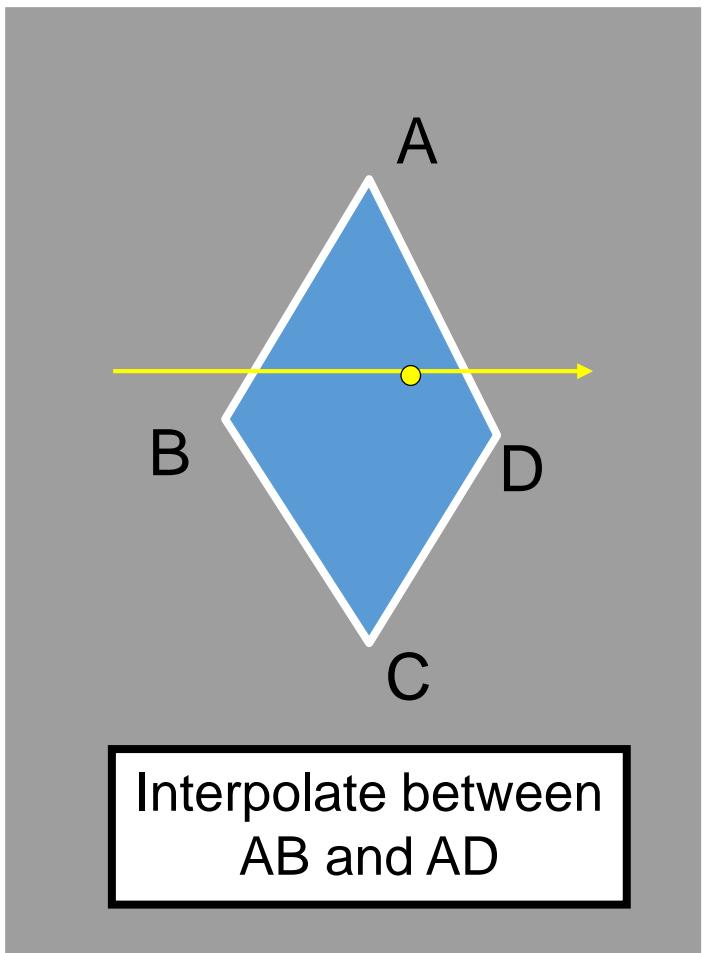


*Gouraud*

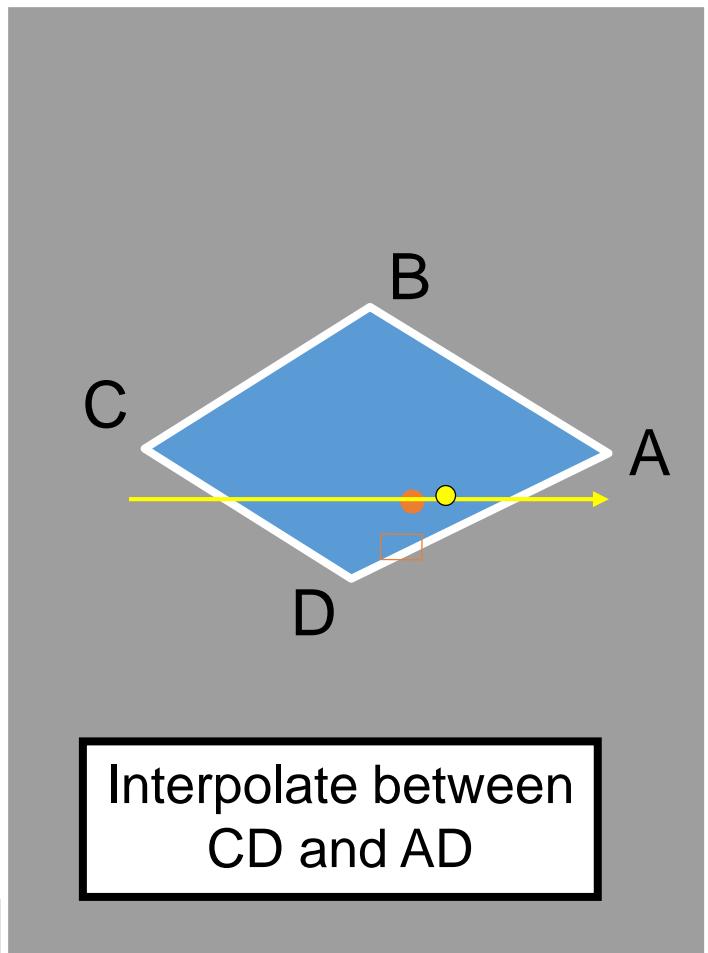
*Phong*

# Shading Artifacts: Orientation

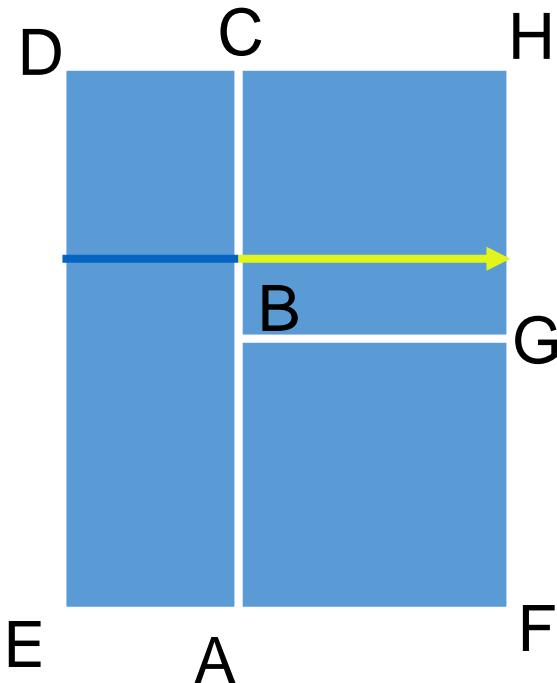
- interpolation dependent on polygon orientation
  - view dependence!



Rotate -90°  
and color  
same point



# Shading Artifacts: Shared Vertices



vertex B shared by two rectangles

first portion of the scanline  
is interpolated between DE and AC

second portion of the scanline  
is interpolated between BC and GH

a large discontinuity could arise



# Shading Models Summary

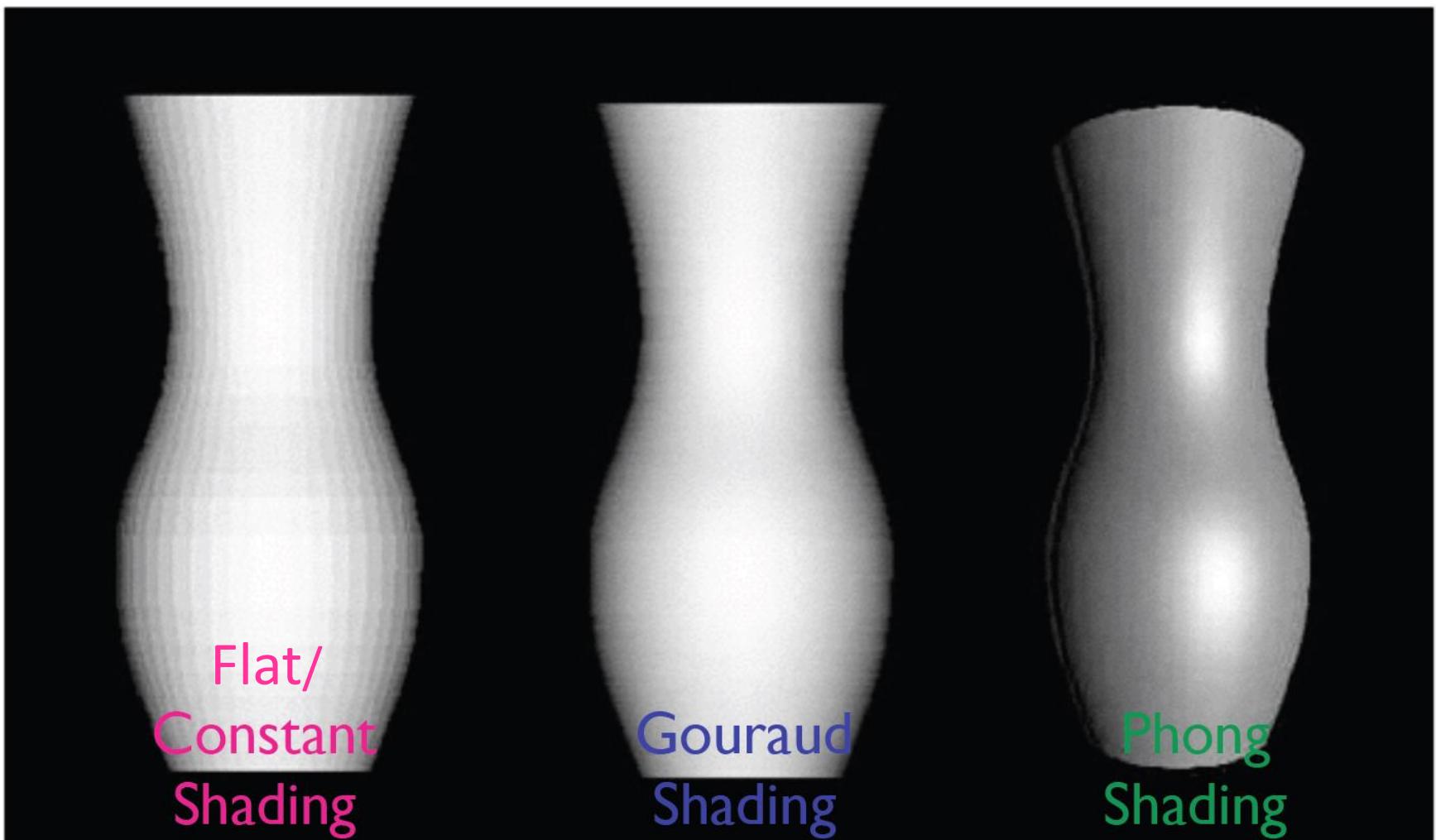
---

- Flat shading
  - compute Phong lighting once for entire polygon
- Gouraud shading
  - **compute Phong lighting at the vertices** and interpolate lighting values across polygon
- Phong shading
  - compute averaged vertex normals
  - interpolate normals across polygon and perform Phong lighting across polygon



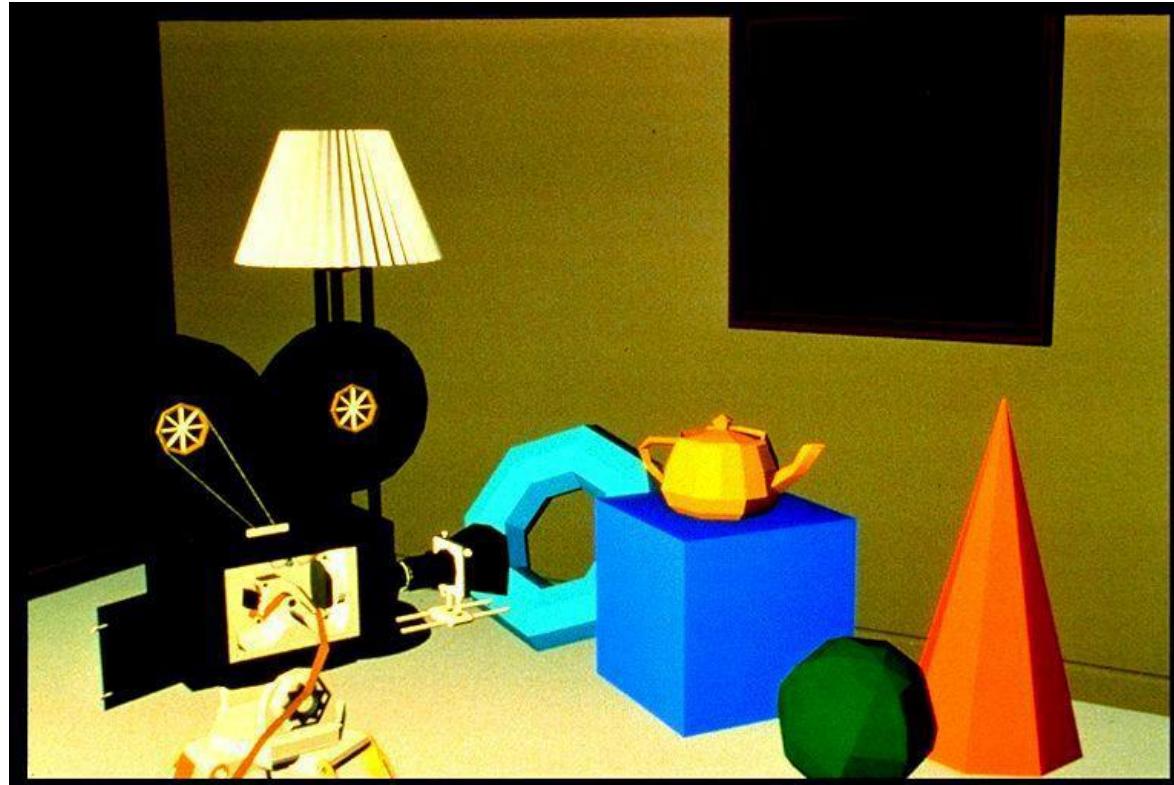
# Phong Shading

---



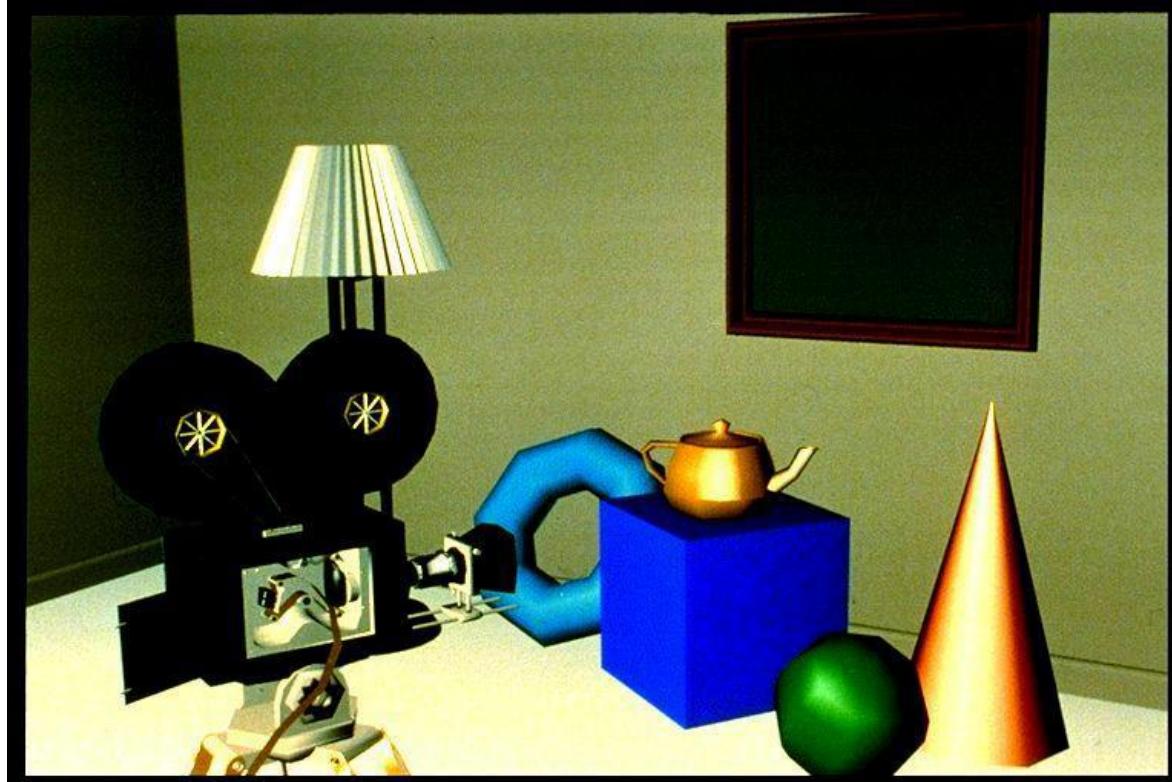
# Flat Shading

---



# Gouraud Shading

---



# Phong Shading

---



# Lighting in OpenGL



# OpenGL Lighting Model

---

Steps of specifying lighting in OpenGL

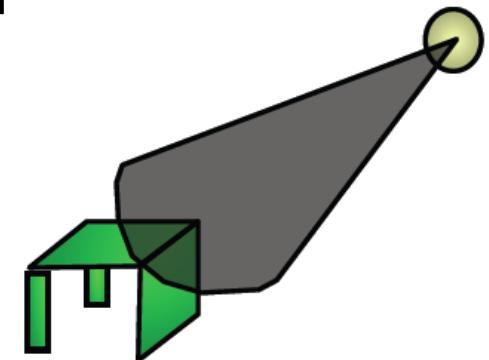
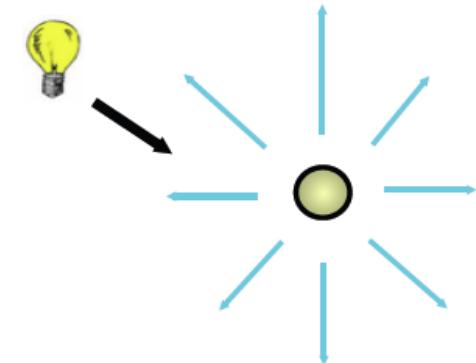
1. Define **normal vectors** for each vertex of every object.
2. Create, position, and enable one or more **light sources**.
3. Select a **lighting model**.
4. Define **material properties** for the objects in the scene.



# OpenGL Lighting Model

---

- The OpenGL lighting model is simple.
- Types of lights
  - **Ambient light** is light that has been reflected so much that it doesn't seem to come from anywhere and illuminates from all directions equally
  - **Point lights** – rays emanate in all directions. Small compared to objects in the scene
  - **Spot lights** – rays emanate in a narrow range of angles



# Lights in OpenGL

---

- Most implementations of OpenGL can have up to 8 lights in the scene
  - Each light can have a diffuse and a specular component
  - Each light can also have an ambient component
  - Lights are referred to by the macros
    - `GL_LIGHT0`, `GL_LIGHT1`, ... , `GL_LIGHT8`
  - We set the properties of lights with calls to the function “`glLightfv`” (`v` stands for vector)



# Light properties

---

- 1、Create light sources :

```
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

其中light\_position是一个指针，指向定义的光源位置齐次坐标数组。  
其它几个光源特性都为缺省值。

同样，我们也可用类似的方式定义光源的其他几个特性值，例如：

```
GLfloat light_ambient [] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat light_diffuse [] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
glLightfv(GL_LIGHT0, GL_AMBIENT , light_ambient );
glLightfv(GL_LIGHT0, GL_DIFFUSE , light_diffuse );
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
```



# Specifying Light properties in OpenGL

- `void glLightfv( GLenum lighti, GLenum pname, GLfloat *params );`
  - ✓ `lighti`: 光源编号:0~8,
  - ✓ `pname`: Specifies the light source properties parameter that is being updated.
  - ✓ `Params`: parameter values.

<b>pname</b> 参数名	缺省值	说明
<code>GL_AMBIENT</code>	(0.0, 0.0, 0.0, 1.0)	RGBA模式下环境光
<code>GL_DIFFUSE</code>	(1.0, 1.0, 1.0, 1.0)	RGBA模式下漫反射光
<code>GL_SPECULAR</code>	(1.0,1.0,1.0,1.0)	RGBA模式下镜面光
<code>GL_POSITION</code>	(0.0,0.0,1.0,0.0)	光源位置齐次坐标 (x,y,z,w)
<code>GL_SPOT_DIRECTION</code>	(0.0,0.0,-1.0)	点光源聚光方向矢量 (x,y,z)
<code>GL_SPOT_EXPONENT</code>	0.0	点光源聚光指数
<code>GL_SPOT_CUTOFF</code>	180.0	点光源聚光截止角
<code>GL_CONSTANT_ATTENUATION</code>	1.0	常数衰减因子
<code>GL_LINEAR_ATTENUATION</code>	0.0	线性衰减因子
<code>GL_QUADRATIC_ATTENUATION</code>	0.0	平方衰减因子



## 2. 启动光照：

- 要使光照有效，首先得启动光照，即 `glEnable(GL_LIGHTING)`；
- 若使光照无效，则调用：`glDisable(GL_LIGHTING)` 可关闭当前光照。
- 然后，必须使所定义的每个光源有效，即：

`glEnable(GL_LIGHT0); glEnable(GL_LIGHT1);.....`



# Lighting in OpenGL

---

- **light source: amount of RGB light emitted**
  - value represents percentage of full intensity  
e.g., (1.0,0.5,0.5)
  - every light source emits ambient, diffuse, and specular light
- **materials: amount of RGB light reflected**
  - value represents percentage reflected  
e.g., (0.0,1.0,0.5)
- **interaction: component-wise multiply**
  - red light (1,0,0) x green surface (0,1,0) = black (0,0,0)



# Specifying Material properties in OpenGL

- `void glMaterialfv( GLenum face, GLenum pname, const GLfloat * params);`
  - ✓ **face**: Specifies which face or faces are being updated. Must be one of GL\_FRONT, GL\_BACK, or GL\_FRONT\_AND\_BACK
  - ✓ **pname**: Specifies the material parameter of the face or faces that is being updated.
  - ✓ **Params**: parameter values.

参数名	缺省值	说明
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	材料的环境光颜色
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	材料的漫反射光颜色
GL_AMBIENT_AND_DIFFUSE		材料的环境光和漫反射光颜色
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)	材料的镜面反射光颜色
GL_SHININESS	0.0	镜面指数（光亮度）
GL_EMISSION	(0.0, 0.0, 0.0, 1.0)	材料的辐射光颜色
GL_COLOR_INDEXES	(0, 1, 1)	材料的环境光、漫反射光和镜面光颜色



# Example

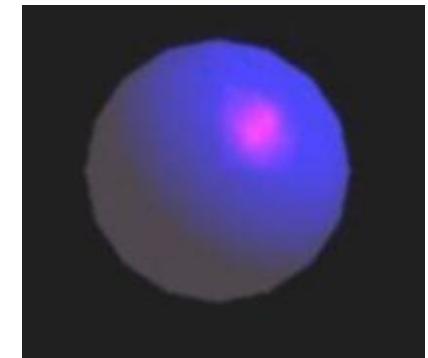
```
void myinit(void)
{
    GLfloat mat_ambient[] = { 0.8, 0.8, 0.8, 1.0 };
    GLfloat mat_diffuse[] = { 0.8, 0.0, 0.8, 1.0 };
    GLfloat mat_specular[] = { 1.0, 0.0, 1.0, 1.0 };
    GLfloat mat_shininess[] = { 50.0 };

    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

    GLfloat light_diffuse[] = { 0.0, 0.0, 1.0, 1.0 };
    GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };

    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);

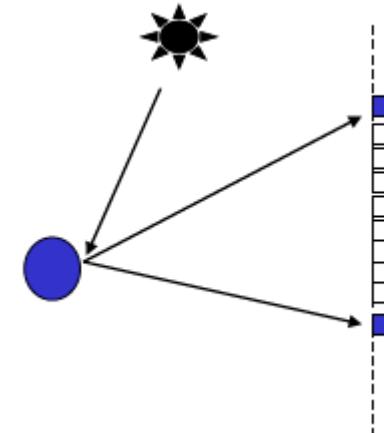
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glDepthFunc(GL_LESS);
    glEnable(GL_DEPTH_TEST);
}
```



# Lighting vs. Shading

---

- Lighting (光照)
  - process of computing the luminous intensity (i.e., outgoing light) at a particular 3D point, usually on a surface
- Shading (描影，着色)
  - the process of assigning colors to pixels



# Advanced Rendering



# Global Illumination Models

---

- Simple lighting/shading methods simulate local illumination models
  - No object-object interaction
- global illumination models
  - More realism, more computation
- Approaches
  - Ray tracing (光线追踪)
  - Radiosity (辐射度)
  - Photon mapping (光子映射)
  - Subsurface scattering (次表面散射)



# Ray Tracing

---

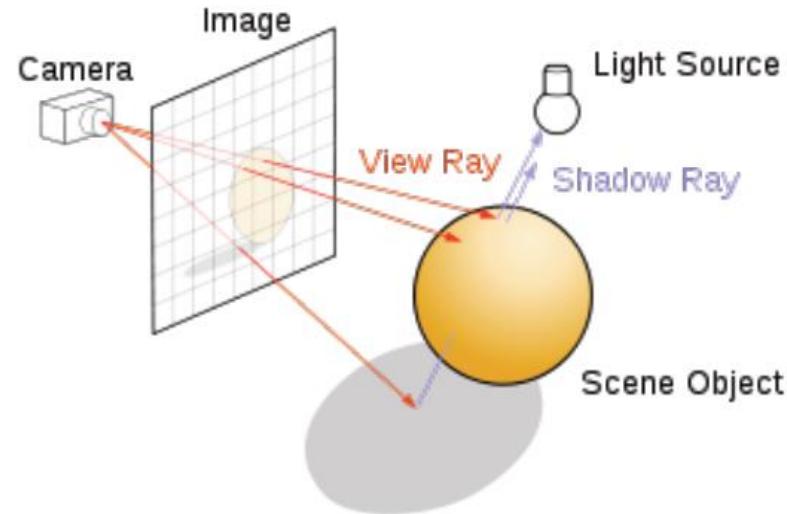
- Ray Tracing is a technique for image synthesis helps create a 2D picture of a 3D world
- An algorithm for visible surface determination, which combines following factors in a single model
  - hidden surface removal
  - shading due to local illumination
  - shading due to global illumination
  - shadows



# Features

---

- Best known for handling **shadows, reflections and refractions**
- It is the most complete simulation of an **illumination-reflection model** in computer graphics
- **Ray tracing** has produced some of the most realistic images in computer graphics



# Ray Tracing

---



# Ray Tracing

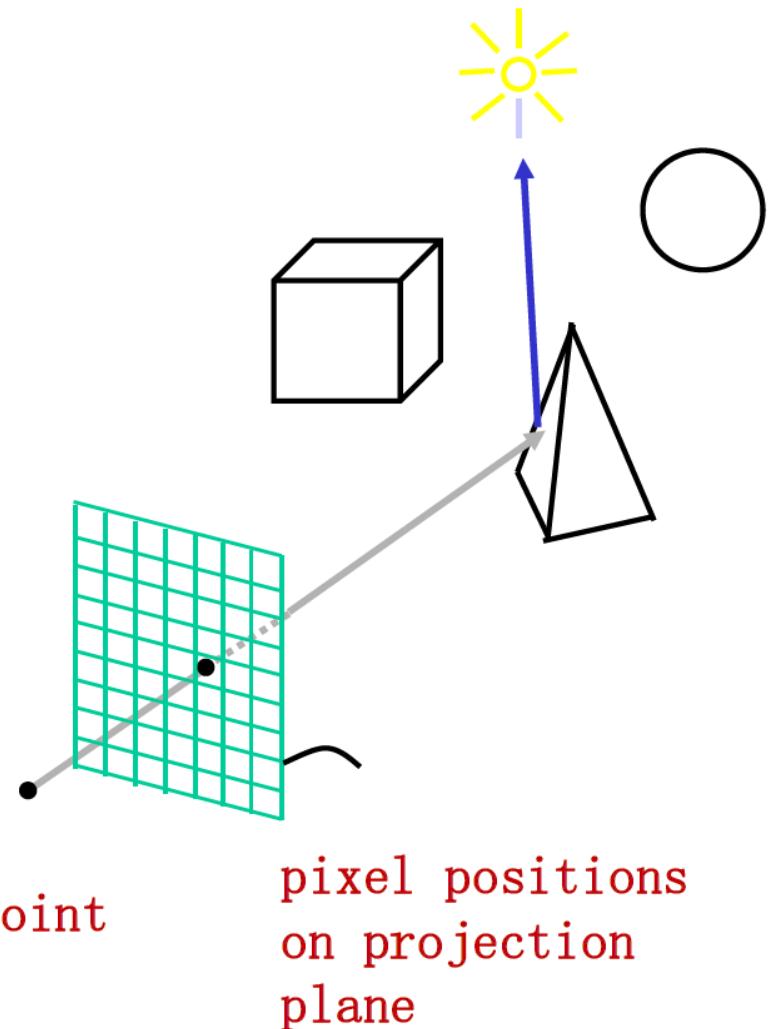
---



# Simple Ray Tracing

---

- **view dependent method**
  - cast a ray from viewer's eye through each pixel
  - compute intersection of ray with first object in scene
  - cast ray from intersection point on object to light sources



# Representing a Ray

---

- Ray tracing is based on ray-object intersection algorithms

*Representing a ray becomes essential:*

A point  $P$  on a ray is given by the parametric equation

$$P = O + t \bullet D \quad , \quad \text{for } t > 0$$

where  $O$  is the ray origin,  $D$  is the ray direction

If the direction  $D$  is normalized then  $t$  is the distance of the point from the origin



# Representing a Ray

---

- Given a ray with *origin*  $O(x_o, y_o, z_o)$  and *direction*  $D(x_d, y_d, z_d)$  any point on the ray is given as

$$P(x_o + t \bullet x_d, y_o + t \bullet y_d, z_o + t \bullet z_d)$$

- This equation forms the basis of calculating intersections with some of the common primitives like sphere, plane etc..



# Ray-Sphere Intersection

---

- Sphere Representation:

- center  $\mathbf{C}(x_c, y_c, z_c)$ , radius  $r$

- Equation of the sphere is

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = r^2$$

- Substituting the ray equation into the sphere equation we have

$$(x_o + t \cdot x_d - x_c)^2 + (y_o + t \cdot y_d - y_c)^2 + (z_o + t \cdot z_d - z_c)^2 = r^2$$



# Ray-Sphere Intersection

- This is a quadratic equation of the form

$$A \cdot t^2 + B \cdot t + C = 0$$

where,

$$A = x_d^2 + y_d^2 + z_d^2 = 1$$

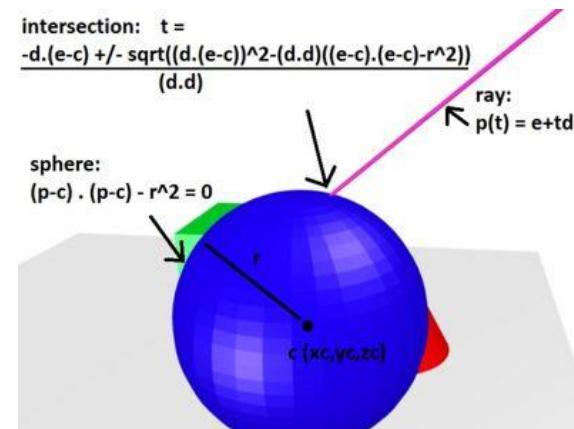
$$B = 2 \cdot (x_d \cdot (x_o - x_c) + y_d \cdot (y_o - y_c) + z_d \cdot (z_o - z_c))$$

$$C = (x_o - x_c)^2 + (y_o - y_c)^2 + (z_o - z_c)^2 - r^2$$

- the two roots are given by

$$t_1 = \frac{-B - \sqrt{B^2 - 4 \cdot C}}{2} \quad t_2 = \frac{-B + \sqrt{B^2 - 4 \cdot C}}{2}$$

- The smallest positive  $t$  value gives the nearest point of intersection



# Ray-Plane Intersection

- The plane is represented by the equation

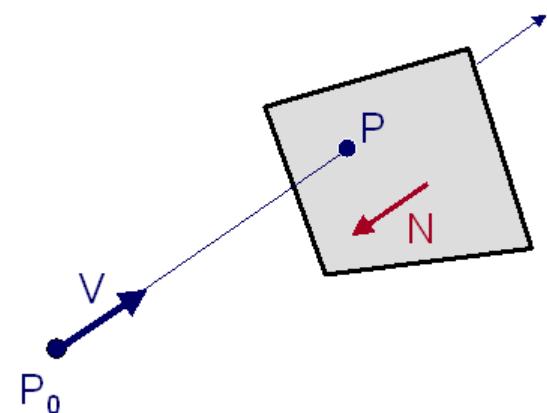
$$a \bullet x + b \bullet y + c \bullet z + d = 0$$

- Substituting the ray equation into the plane equation we have

$$a \bullet (x_o + t \bullet x_d) + b \bullet (y_o + t \bullet y_d) + c \bullet (z_o + t \bullet z_d) + d = 0$$

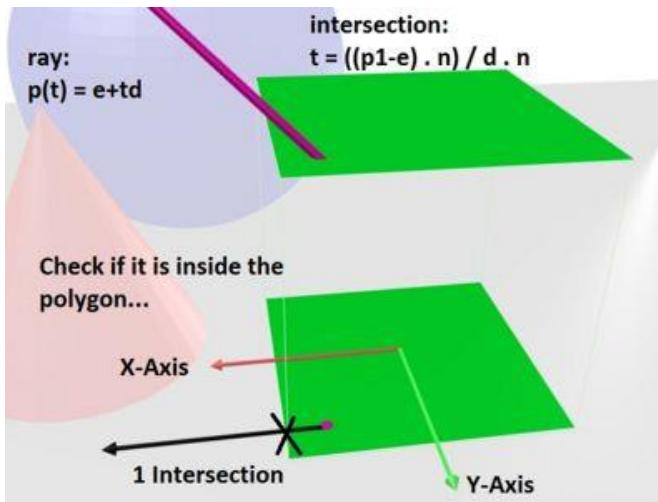
- Solving for t

$$t = \frac{-(a \bullet x_o + b \bullet y_o + c \bullet z_o + d)}{(a \bullet x_d + b \bullet y_d + c \bullet z_d)}$$



# Ray-Polygon Intersection

- Involves two steps
  - Find the point of intersection of the ray with the plane of the polygon
  - Check if the point is inside or outside the polygon (even-odd rule)



# Efficiency in Ray Tracing

---

- 95% of the time is spent in ray-object intersection
- So to increase speed
  - write faster intersection algorithms
  - reduce number of intersection calculations
- Intersection algorithms are always written to work efficiently. Reducing the number of intersection calculation is the key to increase speeds



# Some observations of ray tracing

---

- computationally intensive
  - may take hours to generate a scene of reasonable complexity
- view dependent
  - For every change in view the image has to be recomputed
- Ray tracing in real-time is a challenge even today
  - GPU based ~ or Cloud based ~
  - Use of parallel machines and dedicated ray tracing chips are some methods being investigated to do real-time ray tracing



# More about ray tracing

---

- LuxRender is a physically based and unbiased rendering engine. Based on state of the art algorithms, LuxRender simulates the flow of light according to physical equations, thus producing realistic images of photographic quality.



<http://luxrender.net/>

