Computer Graphics

Lighting & Shading

Teacher: A.prof. Chengying Gao(高成英)

E-mail: mcsgcy@mail.sysu.edu.cn

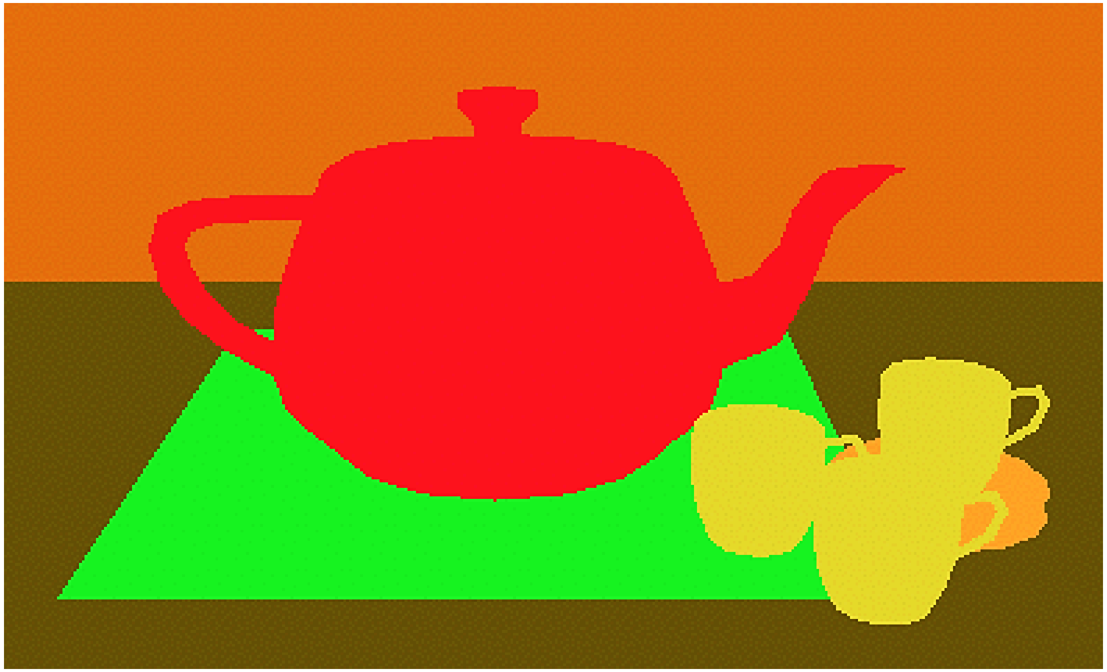
School of Data and Computer Science

Lighting & Shading

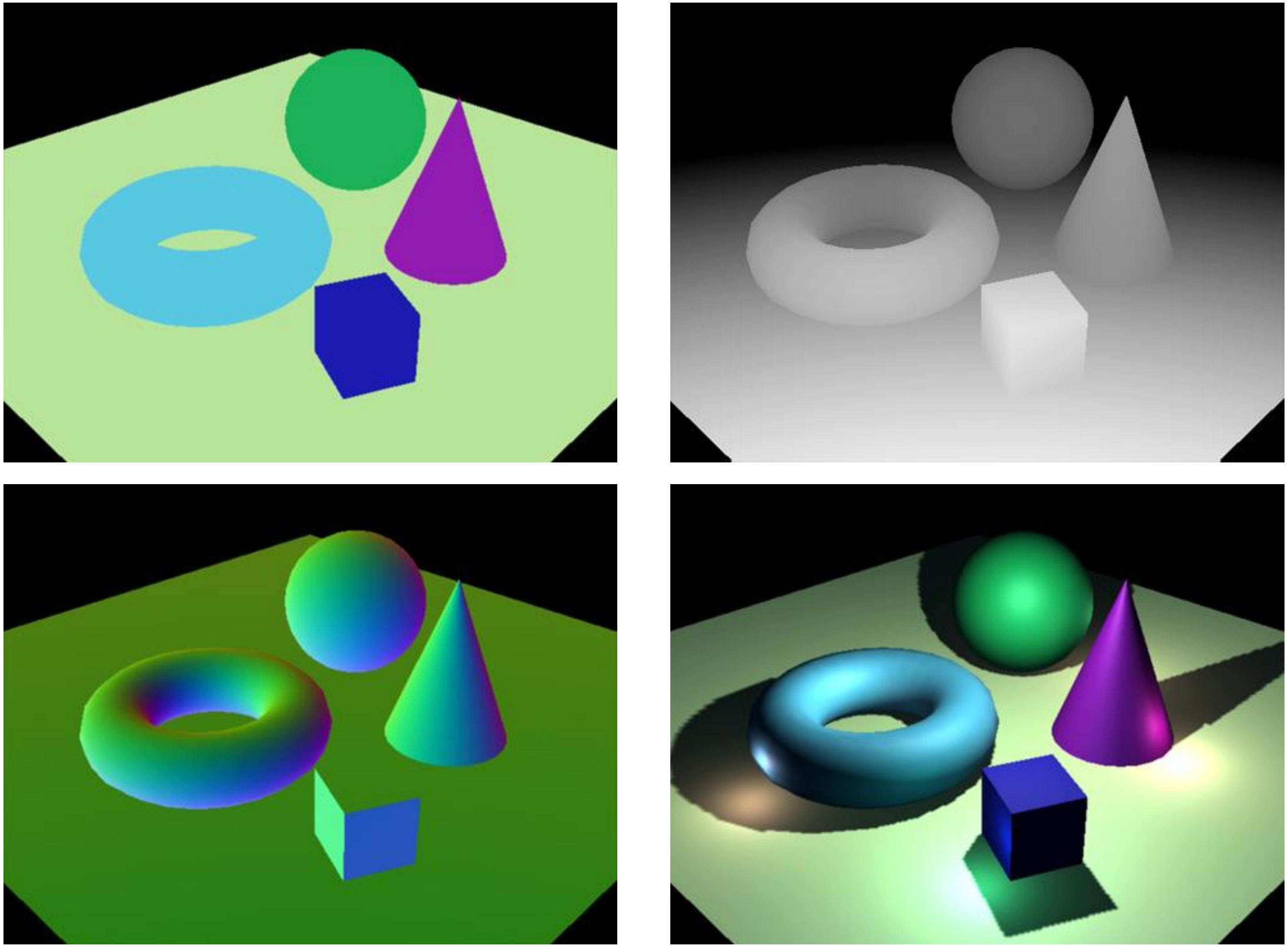
- Without light…
- We do not see much of our scene!

Lighting & Shading

- Without shading…
- Objects do not look three dimensional

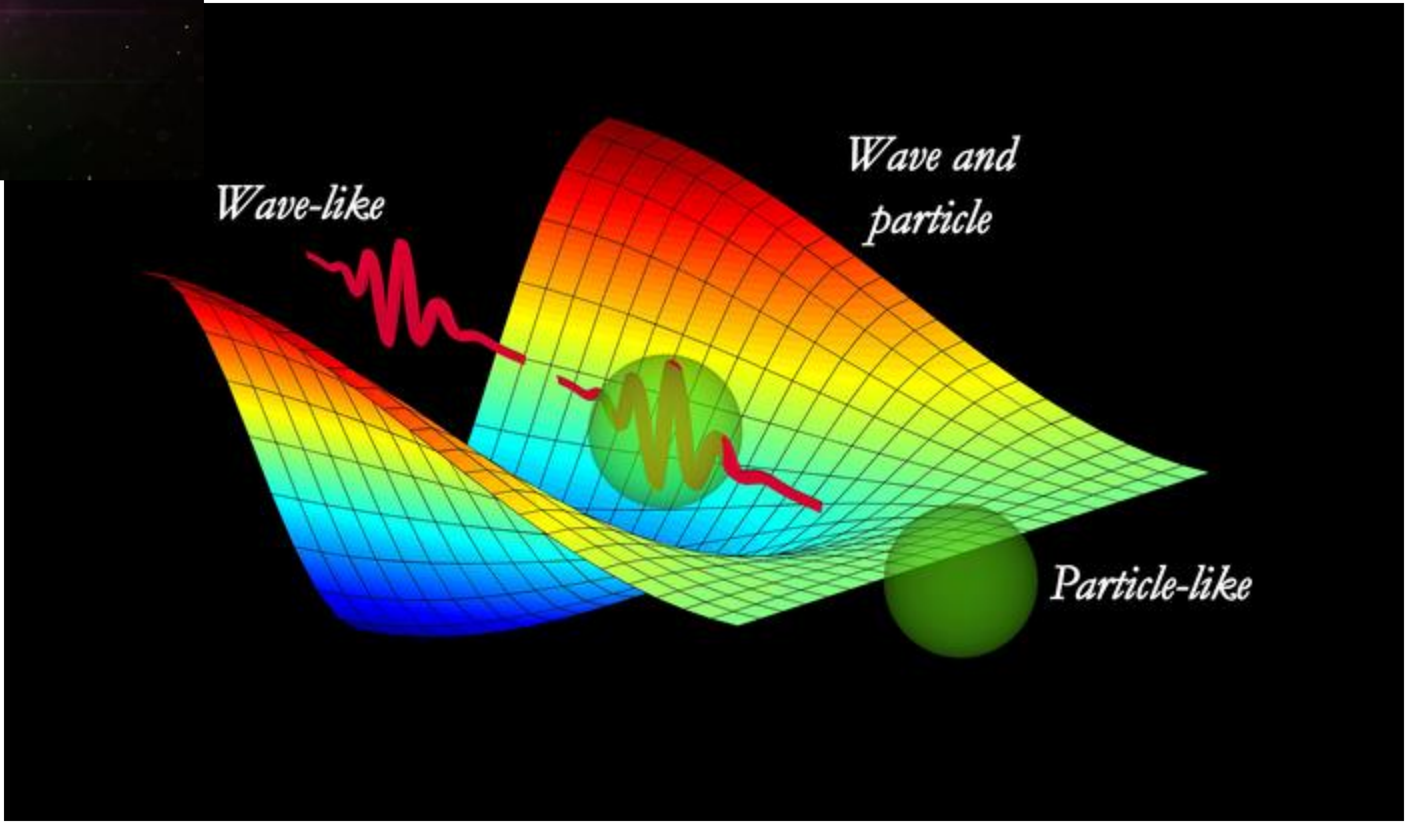


Lighting & Shading

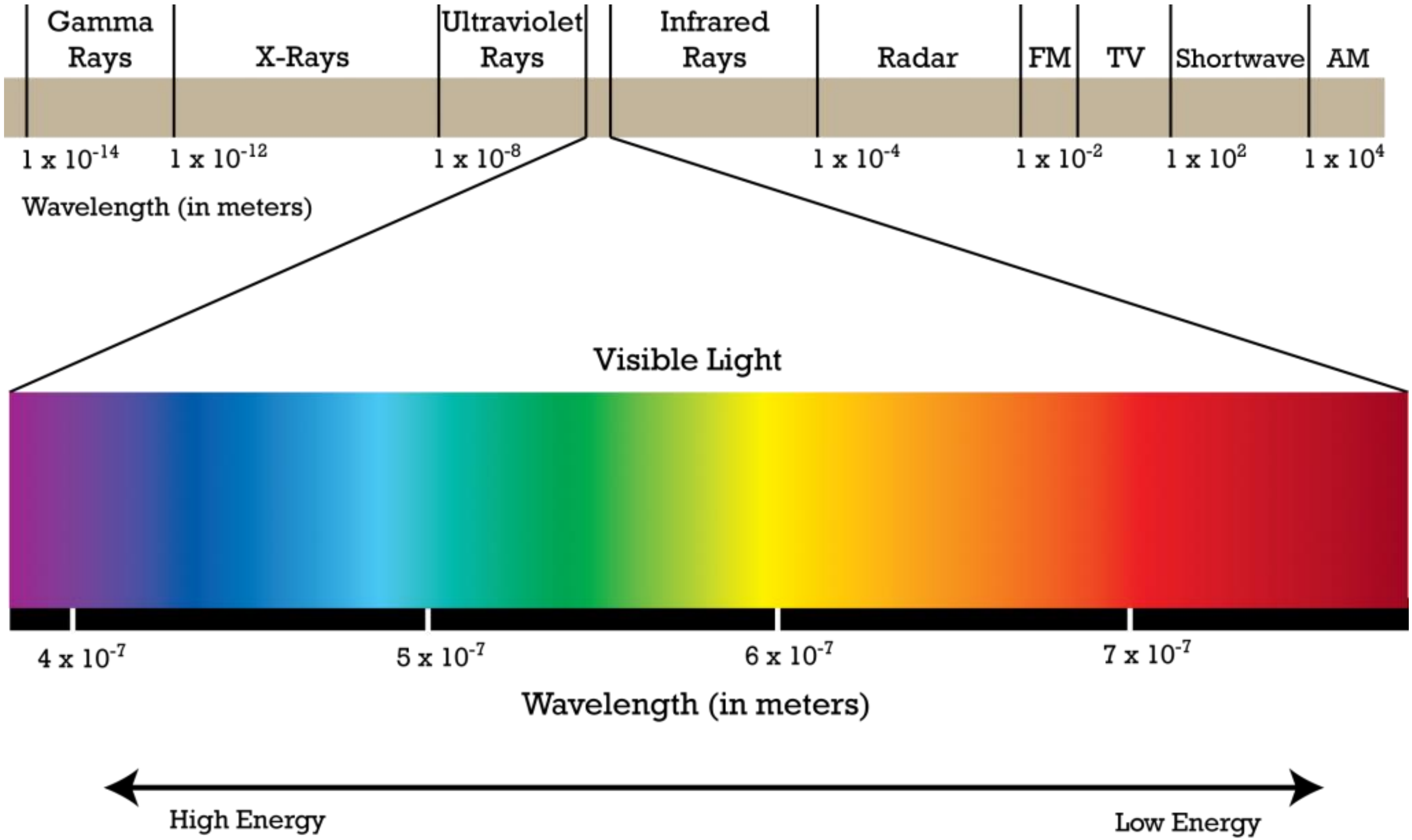


Basic concepts of light

- wave-particle duality (波粒二象性)



Light or visible light



Light source

- Natural

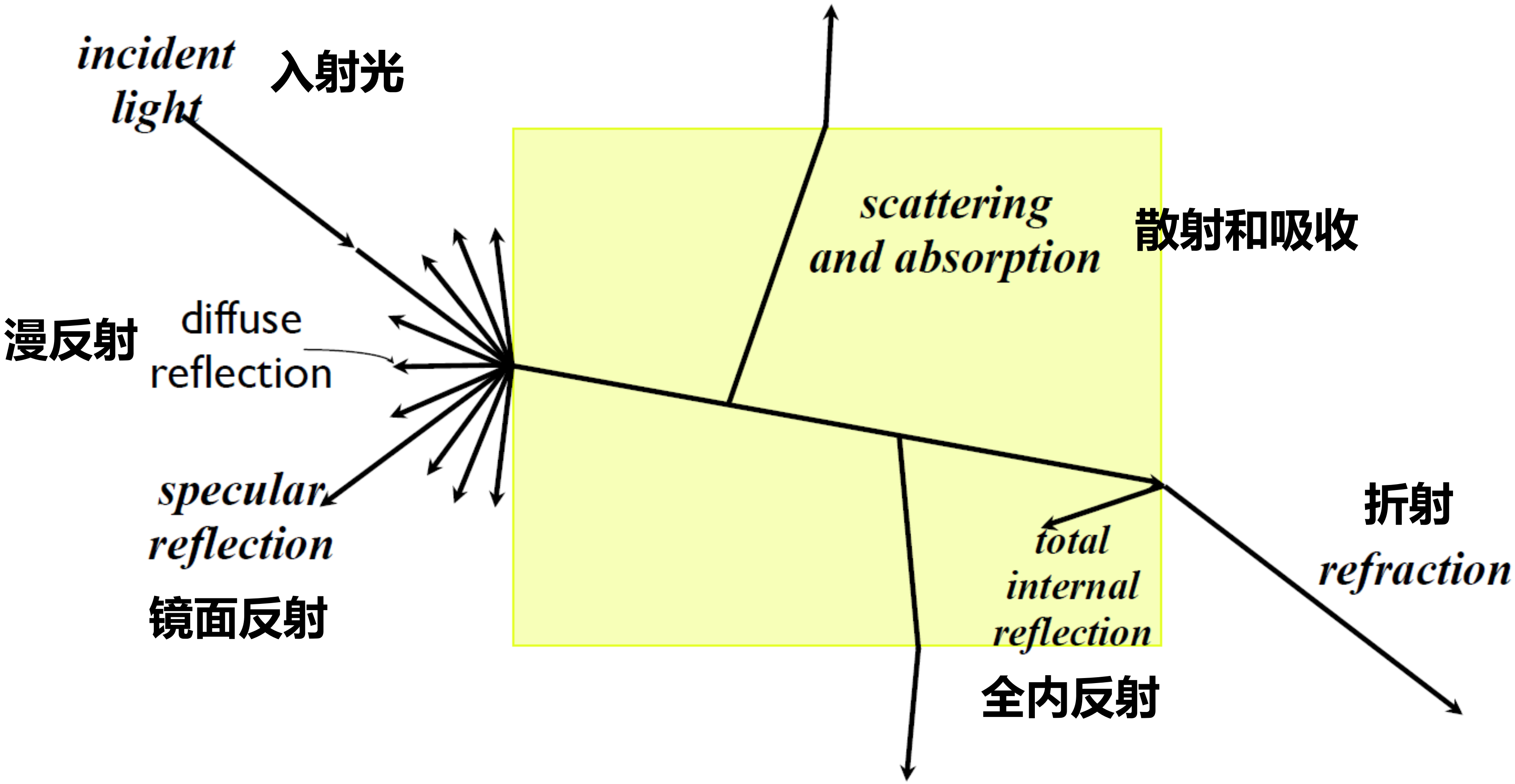
Light source

- Man-made

Illumination (照度)

- ***Illumination*** is the complete description of all the light striking a particular point on a particular surface
- ***Color*** at a point on an object is decided by the properties of the light leaving that point
- Knowing the ***illumination*** and the ***surface physics*** at a point on a surface, we can determine the properties of the light leaving that point
- In order to generate realistic images we need to understand how light interacts with the surface of objects

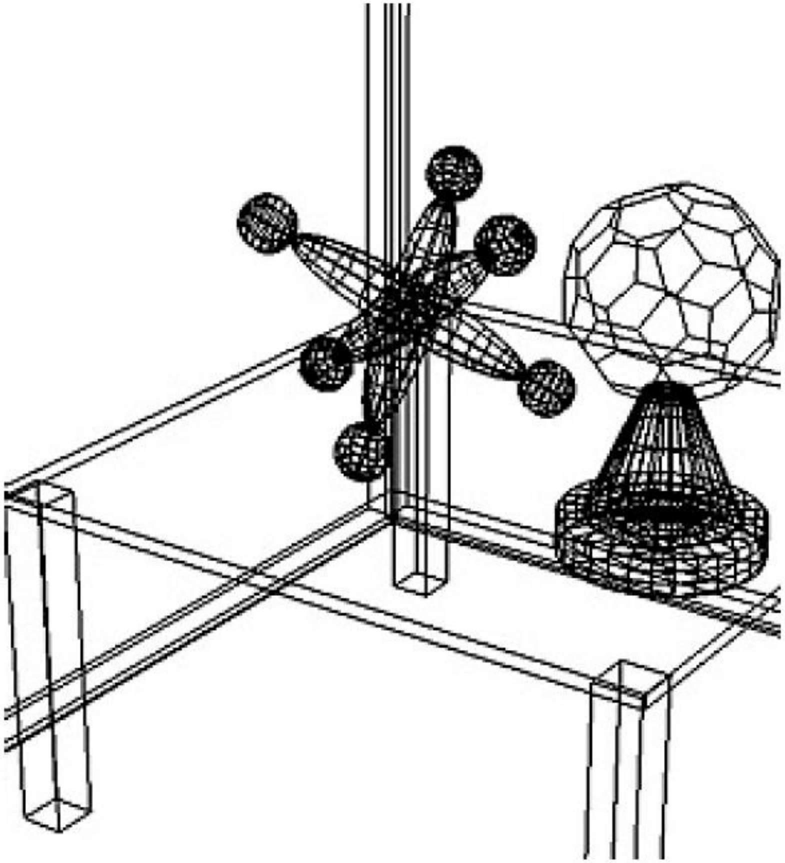
Interaction of light with a Solid



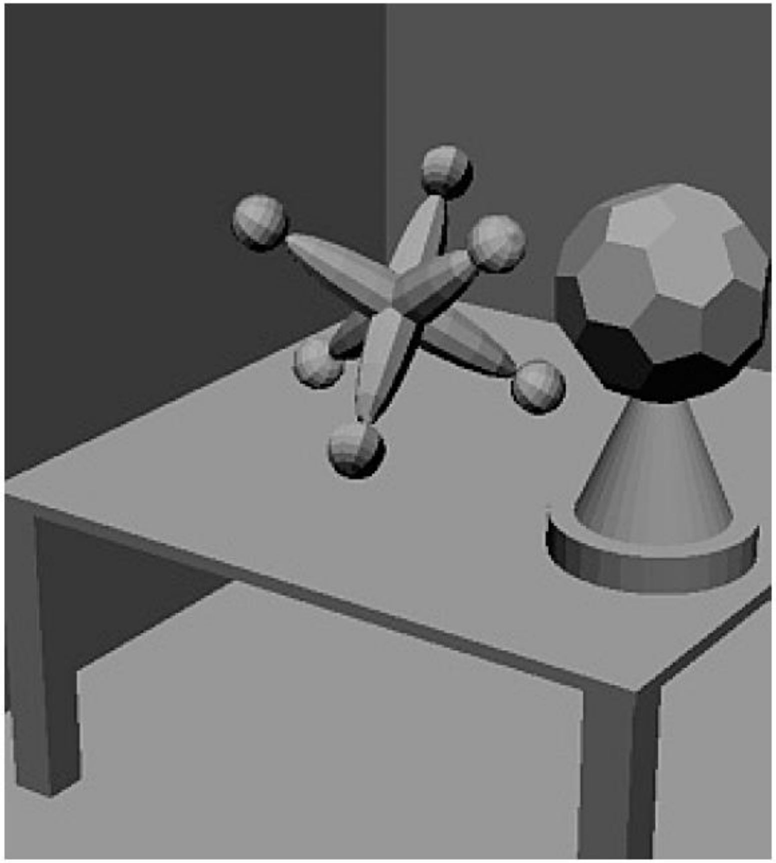
Lighting Simulation

- 在隐藏面消除后，为了使对象看起来更真实，应当模拟光照在物体的状态，即应当通过计算确定表示对象的像素的适当状态。
- 在这种计算中应充分根据对象表面的状态，光源的位置以及视点的位置。
- 需要计算每帧图像中各个像素的颜色亮度，而不是由用户直接指定。

Example



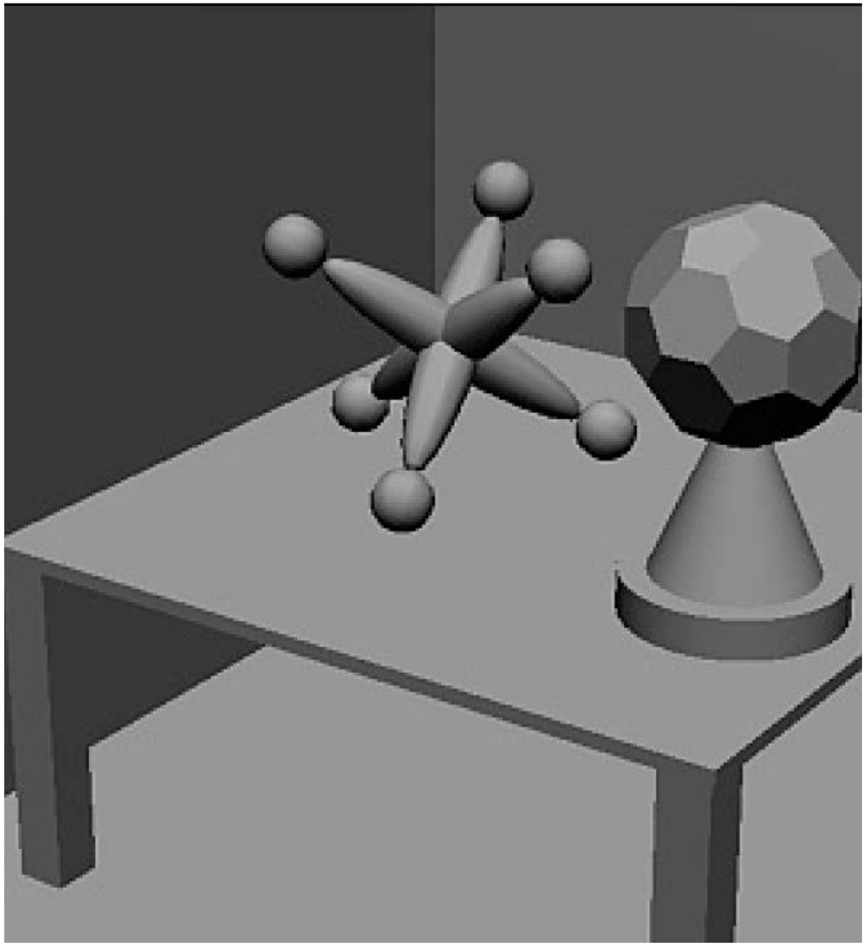
Wireframe



Simple lighting & flat shading

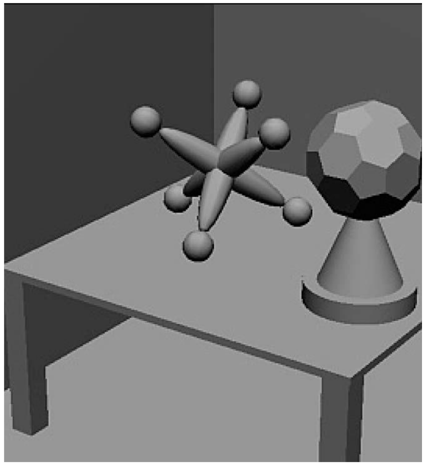
Smooth Shading

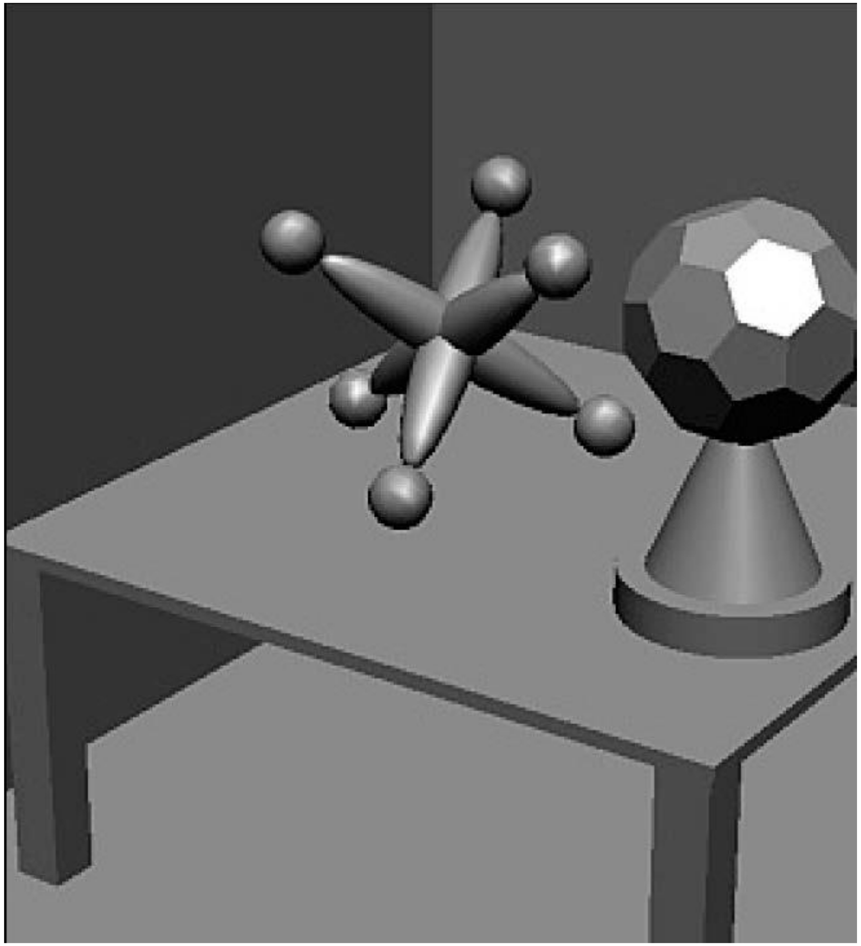
- 如果多边形网格表示的对象为光滑对象，那么显示的结果应当反映这种光滑性
- 在计算了每个顶点处的亮度后，应用线性插值计算出内部的亮度
  - 称为Gouraud明暗处理算法



Specular Light

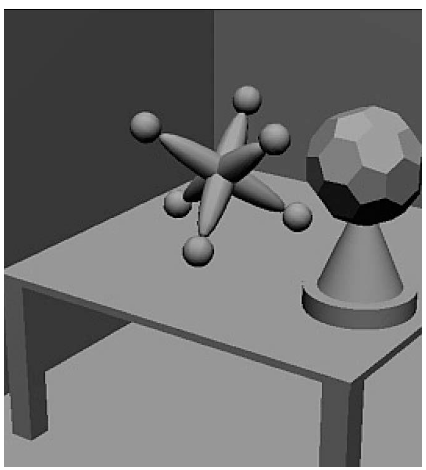
- 为了得到更真实的效果，可以加入镜面光的效果

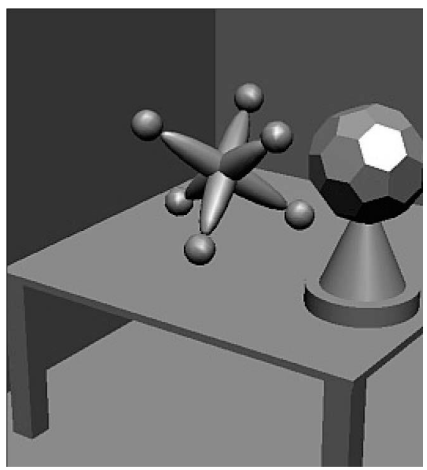


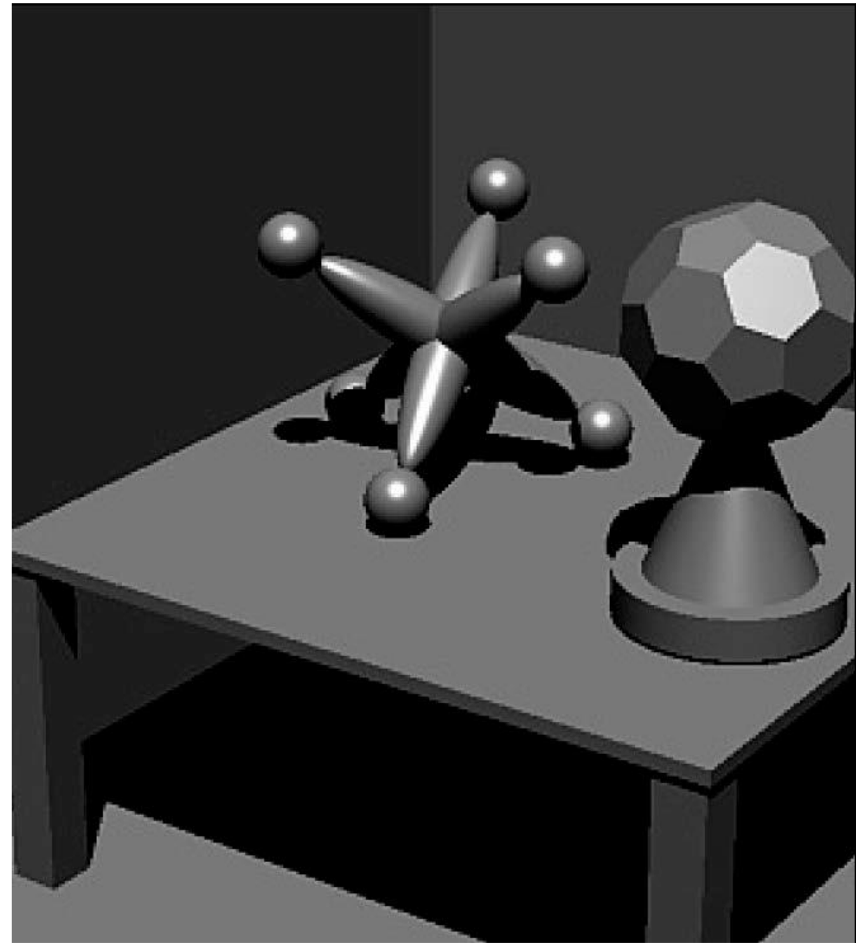


Shadow

- 在加入阴影后，可以进一步提高图像的真实感

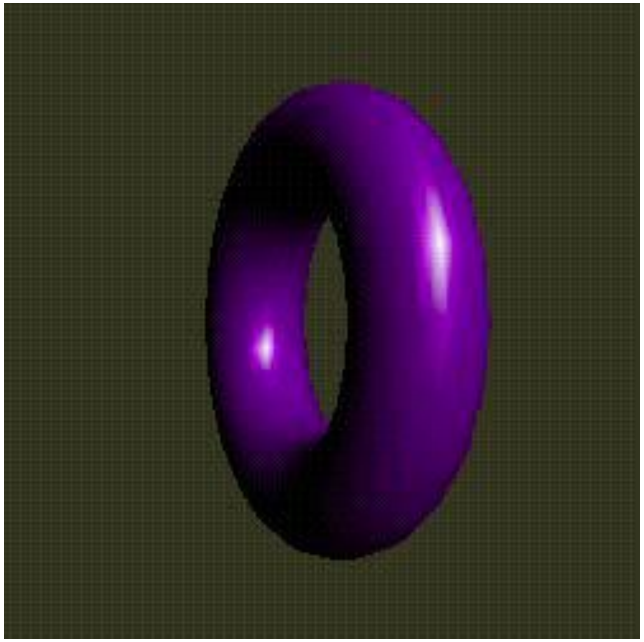




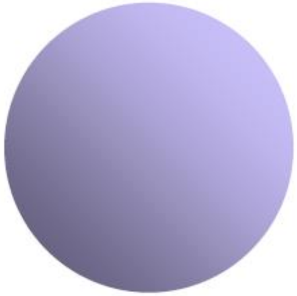


shading

- Assume a ball model is constructed by using polygon mesh, its color is defined by **glColor**, then we obtain,

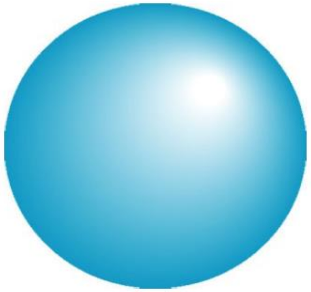


- However, we expect

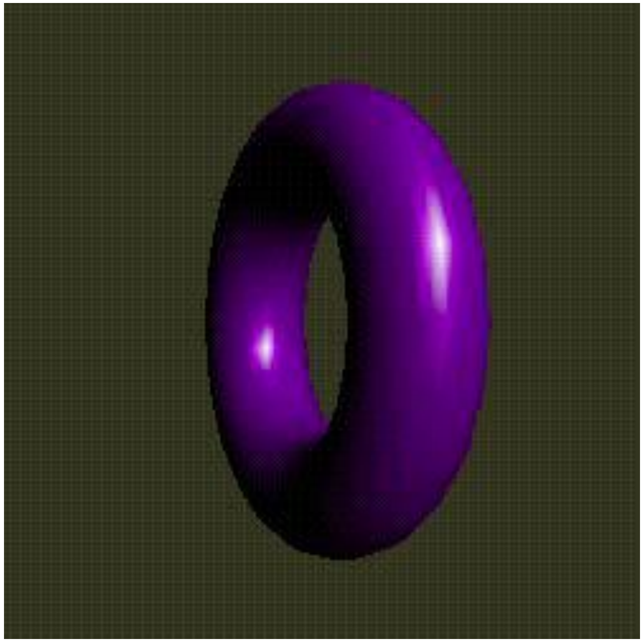


Why?

- 为何实际中球的图像应当类似于



- 光源与材料的交互作用导致每点有不同的颜色或者明暗效果
- 这时需要考虑
  - 光源
  - 材料属性
  - 观察者位置
  - 曲面定向



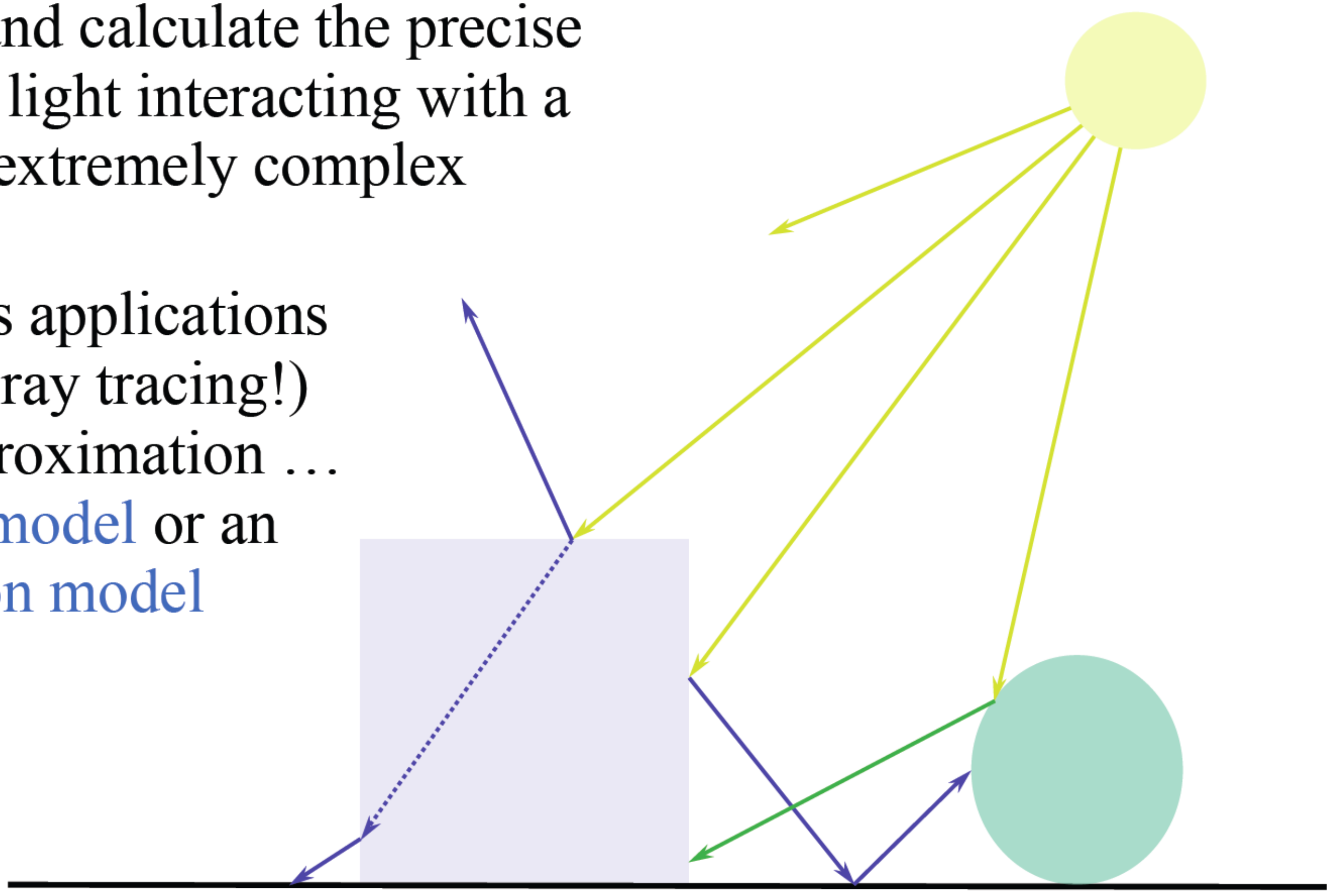
Interaction of Light

- There are two illumination phenomena of importance
  - interaction of light with the boundaries between materials
  - Scattering (发散) and absorption (吸收) of light as it passes through the material
- Boundaries between materials are surfaces which make up the environment
- Light striking a boundary is either reflected (反射) or transmitted (透射). For opaque materials, light is absorbed on passing through the boundary

Light interaction in a Scene

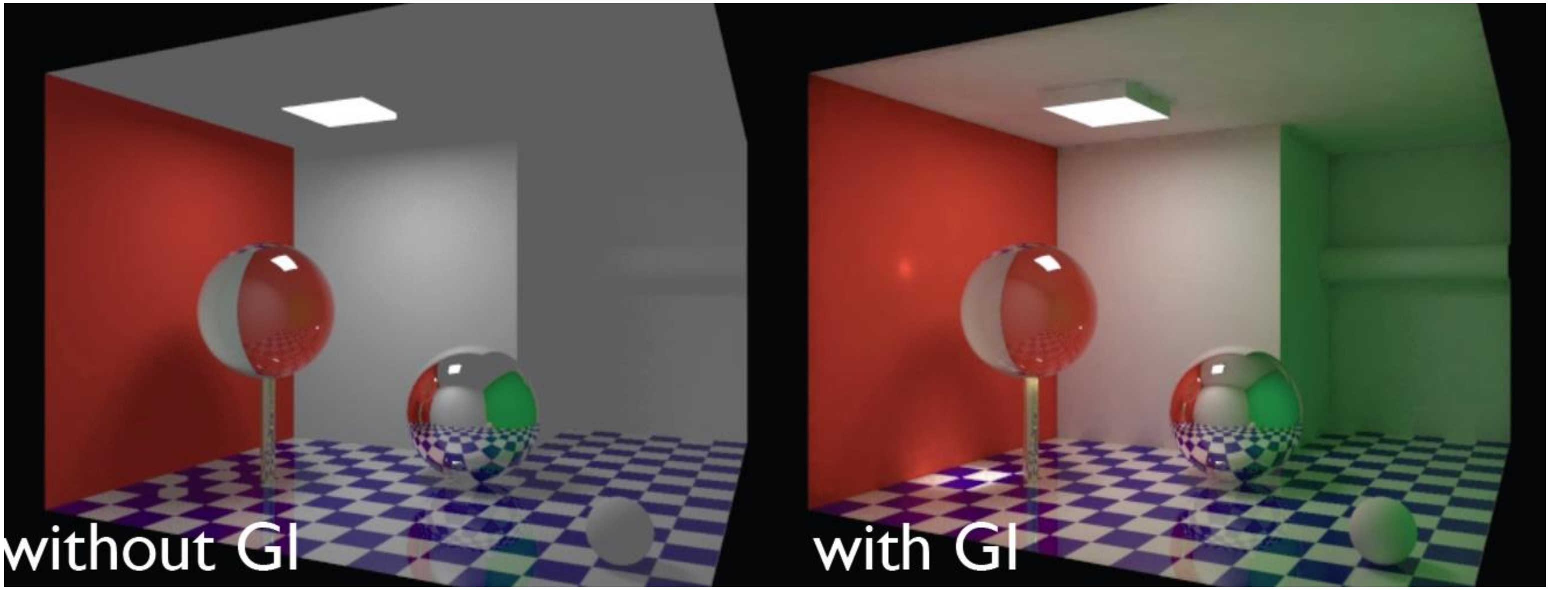
To simulate and calculate the precise physics of light interacting with a surface is extremely complex

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



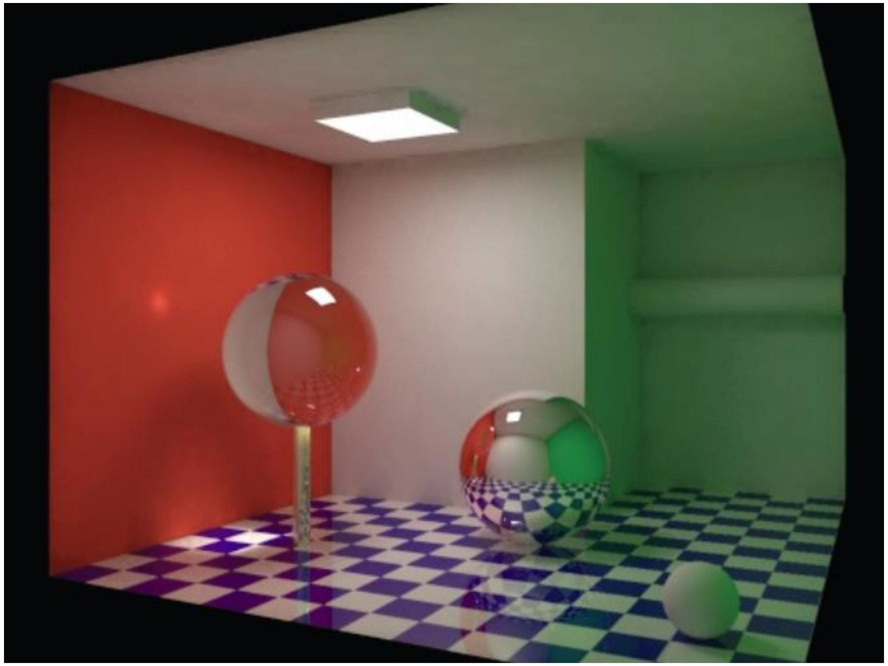
Illumination Models

- 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

- Illumination models
  - 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 illumination



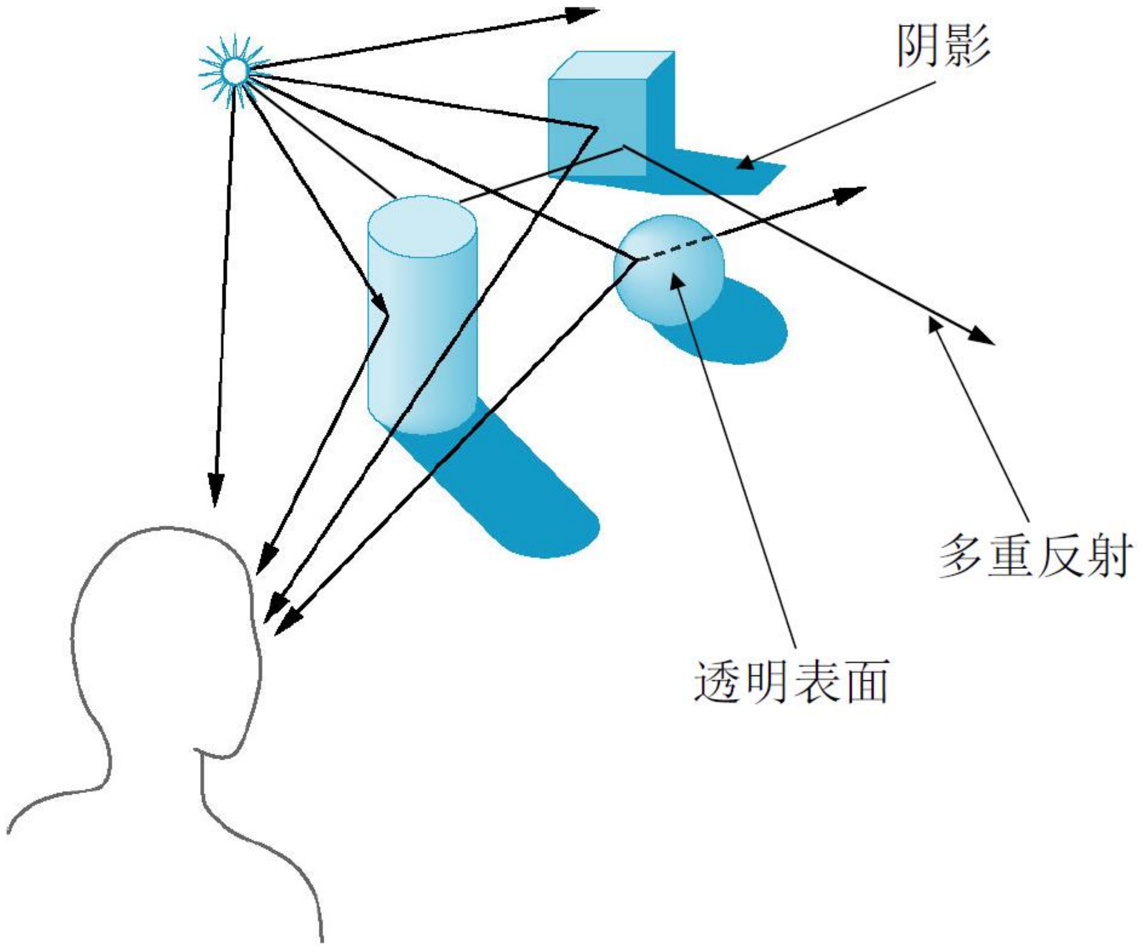
Illumination models

- 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

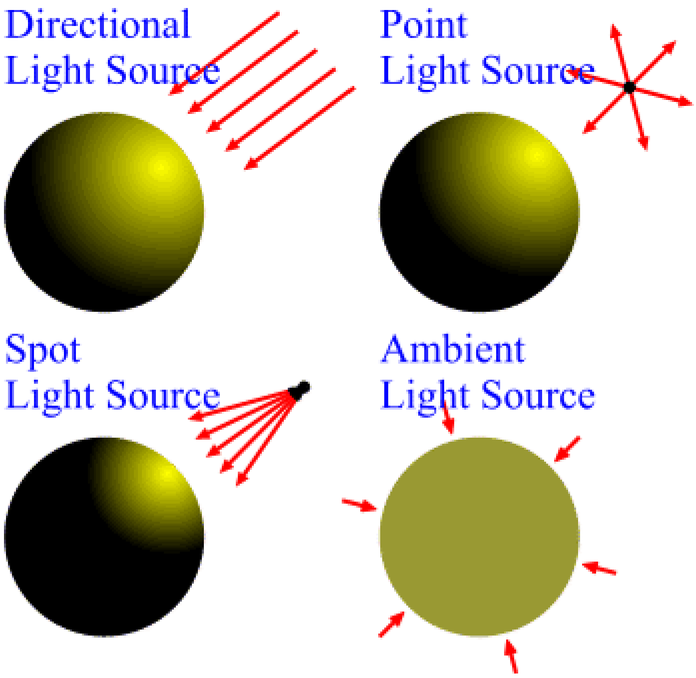


Color of Light

- 光源不但发射出不同量的不同频率光，而且它们的方向属性随着频率也可能不同
  - 真正的物理模型将非常复杂
- 人的视觉系统是基于三原色理论的
- 在大多数应用中，可以用三种成分—红、绿、蓝—的强度表示光源
  - 光亮度(luminance)函数为$\mathbf{I} = [I_r, I_g, I_b]$

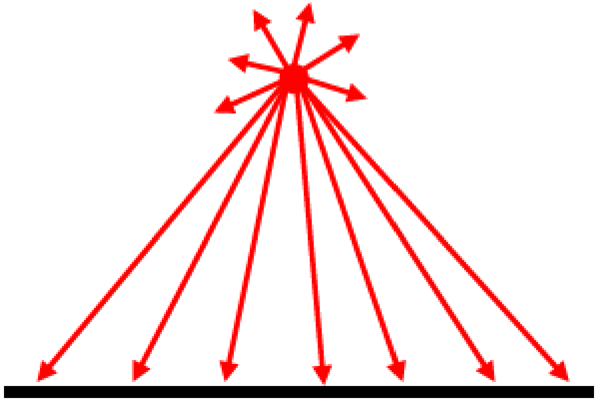
Light Sources

- Point source
- Parallel source
- Ambient lights
- Spotlights



Point source

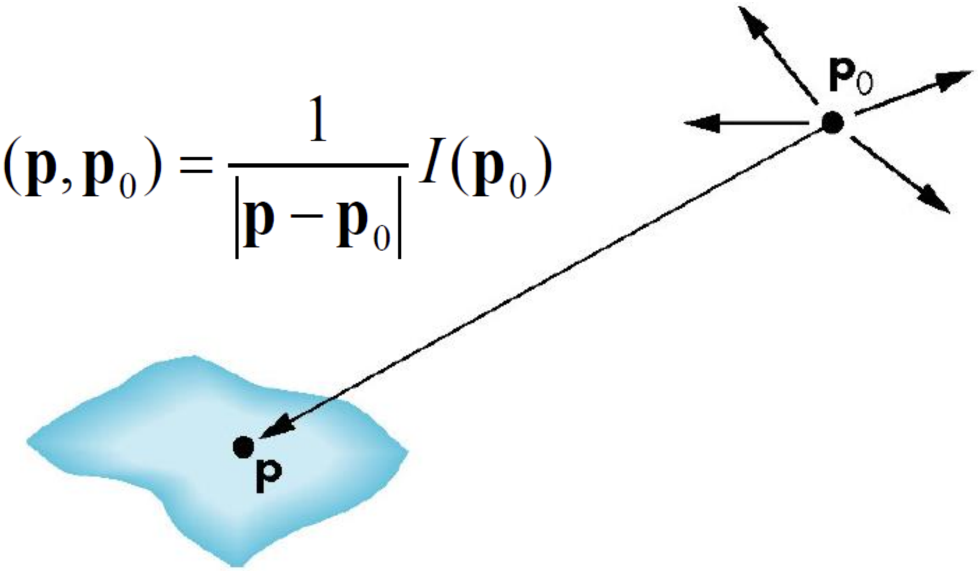
- A point light source emits light equally in all directions from a single point
- The direction to the light from a point on a surface thus differs for different points



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

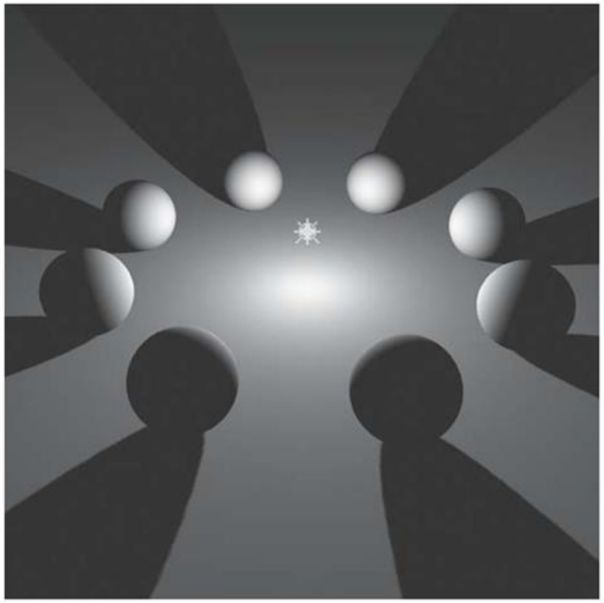
Luminance of Point Light

- 由位置和颜色表示
- 理想的点光源向各个方向发射光
- 远光源--即在无穷远处的光源，光线为平行线
- 点光源的亮度函数 $\mathrm{I}(\mathbf{p}_0) = [I_r(\mathbf{p}_0), I_g(\mathbf{p}_0), I_b(\mathbf{p}_0)]$
- 在点**p**接受的光亮度
- 反比于光源与点的距离 $i(\mathbf{p},\mathbf{p}_0) = \dfrac{1}{|\mathbf{p}-\mathbf{p}_0|} I(\mathbf{p}_0)$



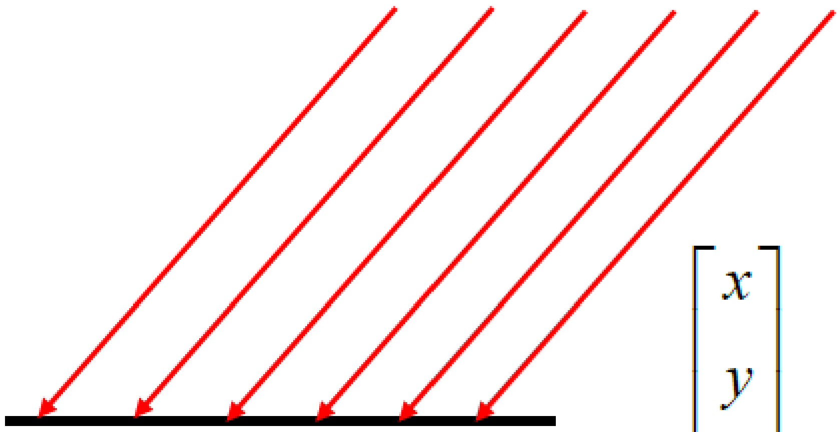
Application of Point Light

- 在计算机图形学中大量应用点光源，是因为它易于使用，
- 但不能很好地反映物理现实
  - 只有点光源的场景得到的图像中对比度比较高；对象显得要么很亮，要么很暗
  - 而真实的光源由于尺寸较大，因此场景的结果比较柔和



Parallel source

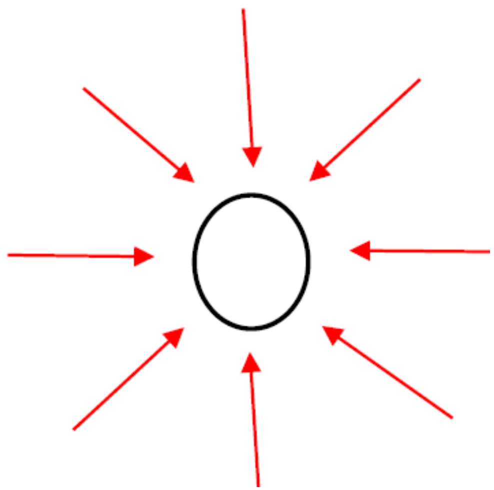
- light rays are parallel
- Rays hit a planar surface at identical angles
- May be modeled as point source at infinity
- Directional light



$$\begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

Ambient Lights

- Objects not directly lit are typically still visible
  - e.g., the ceiling in this room, undersides of desks
- This is the result of indirect illumination from emitters, bouncing off intermediate surfaces
- Too expensive to calculate (in real time), so we use a hack called an ambient light source
  - No spatial or directional characteristics; illuminates all surfaces equally
  - Amount reflected depends on surface properties



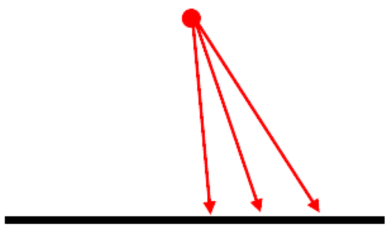
Ambient Lights

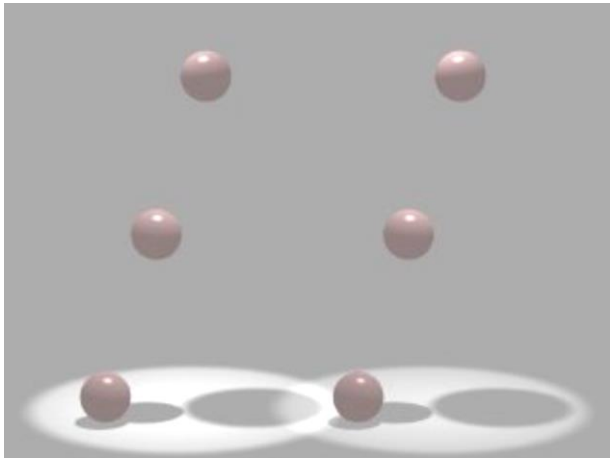
- **For each sampled wavelength (R, G, B), the ambient light reflected from a surface depends on**
  - The surface properties, $k_{ambient}$
  - The intensity, $I_{ambient,}$ of the ambient light source (constant for all points on all surfaces)
  - $I_{reflected} = k_{ambient} \, I_{ambient}$

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

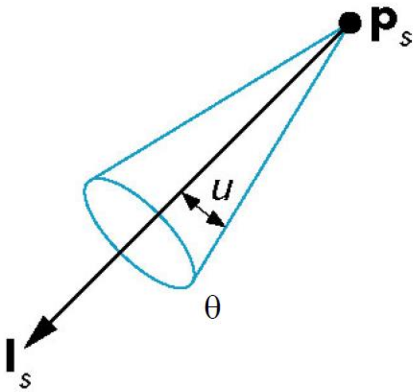
Spotlights

- 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$\theta = 180°$，聚光灯成为点光源



Interaction of Light Sources & Materials

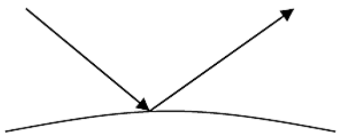
- 照射在对象上的光线部分被吸收，部分被反射
- 如果对象是透明的，有些光被折射
- 反射部分的多少确定对象的颜色与亮度
- 对象表面在白光上看起来是红的，就是因为光线中的红色分量被反射，而其它分量被吸收
- 反射光被反射的方式是由表面的光滑程度和定向确定的

Materials

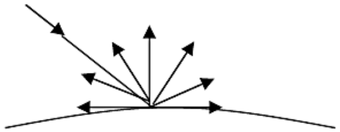
- Surface reflectance (反射比)
  - Illuminate surface point with ray of light from different directions
  - How much light is reflected in each direction

Types of Reflection

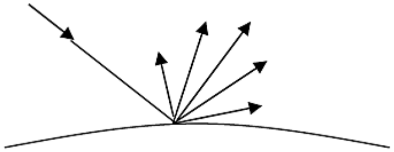
- **specular reflection** (a.k.a. *mirror* or *regular*) causes light to propagate without scattering.



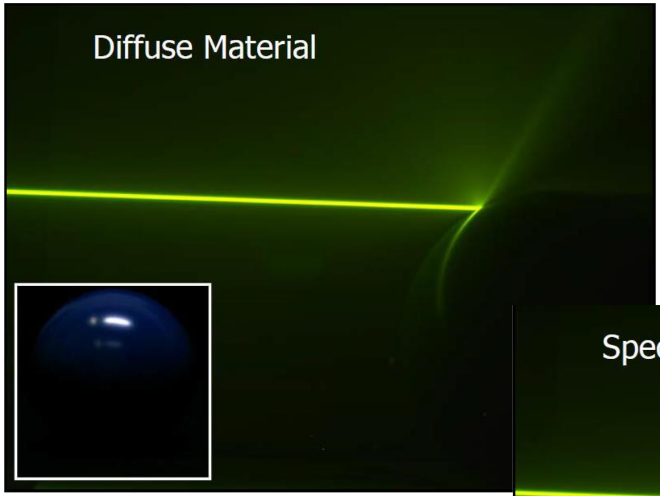
- **diffuse reflection** sends light in all directions with equal energy.

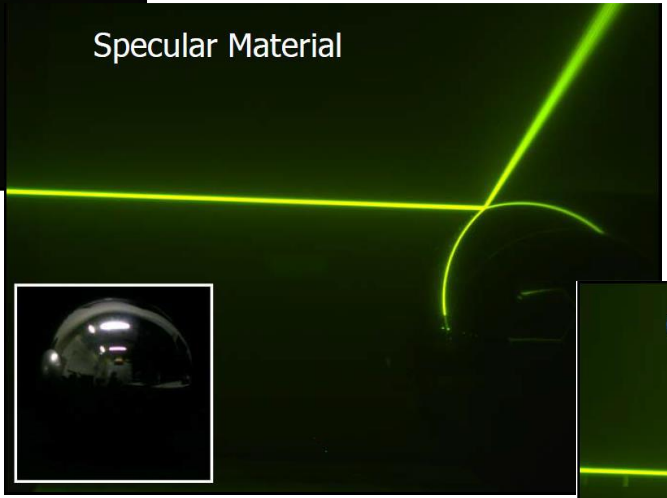


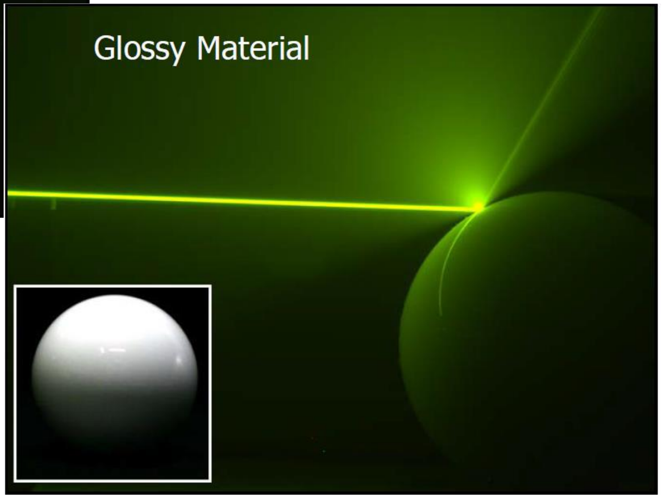
- **glossy/mixed reflection** is a weighted combination of specular and diffuse.



Materials

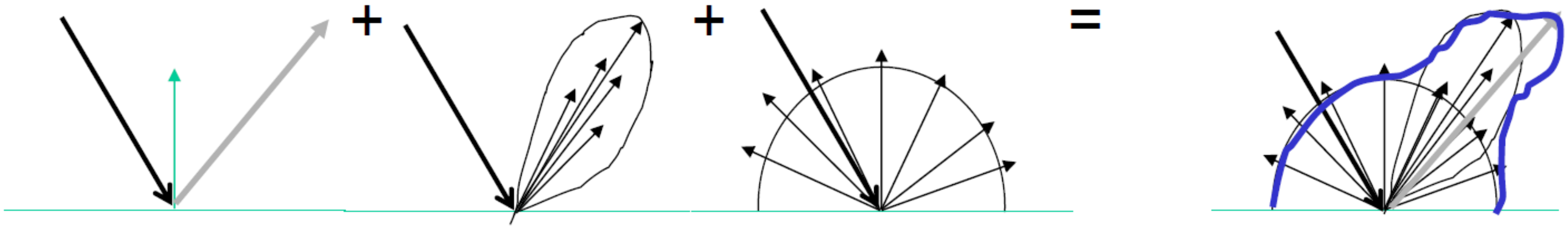






Reflectance Distribution Model

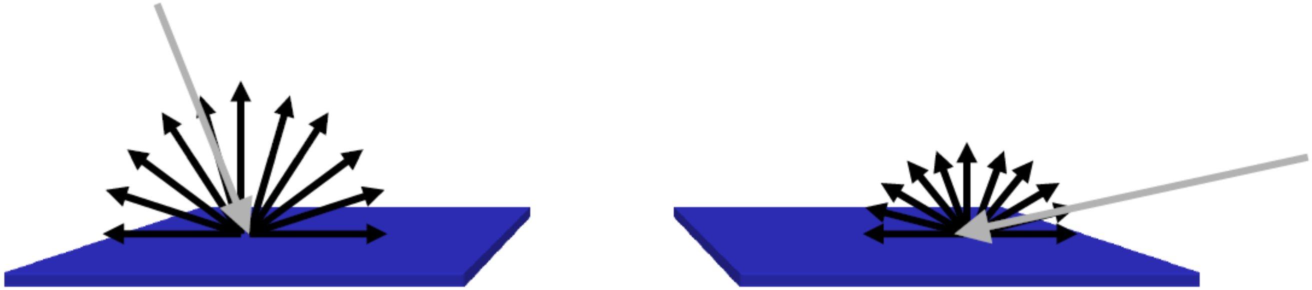
- **most surfaces exhibit complex reflectances**
  - vary with incident and reflected directions.
  - model with combination



specular + glossy + diffuse = reflectance distribution

Physics of Diffuse Reflection

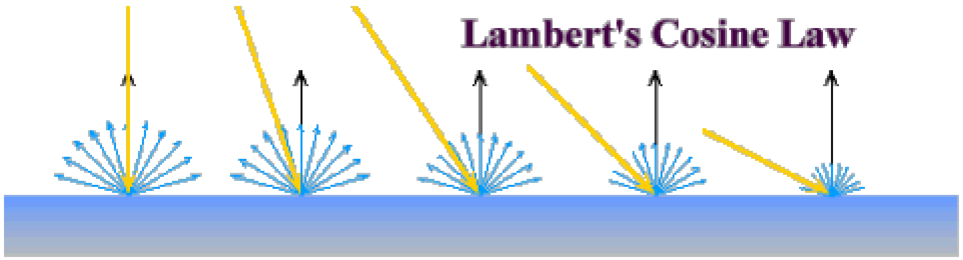
- Ideal diffuse reflection
  - very rough surface at the microscopic level (微观级别)
    - real-world example: chalk
  - microscopic variations mean incoming ray of light equally likely to be reflected in any direction over the hemisphere
  - what does the reflected intensity depend on?
    - Only depends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

Lambert光照模型



- Lambert光照模型用于纯粹的漫反射表面的物体，比如磨砂的玻璃表面，观察者的所看到的反射光和观察的角度无关，这样的表面称为Lambertian。直观地说，就是他表面的亮度是各向同性的，亮度的计算遵循 Lambert‘s cosine 法则。

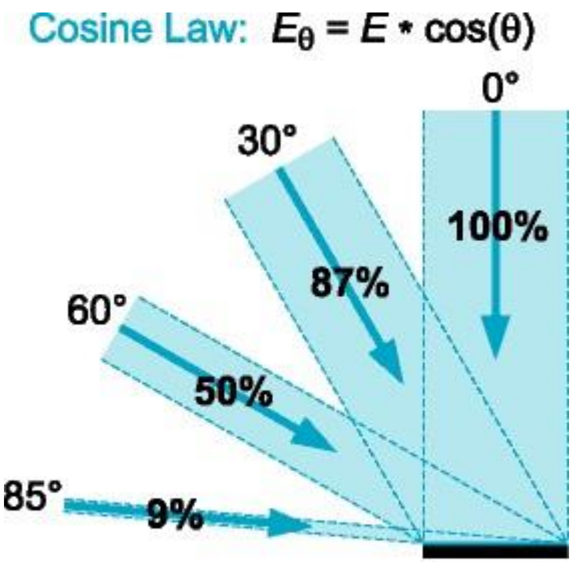
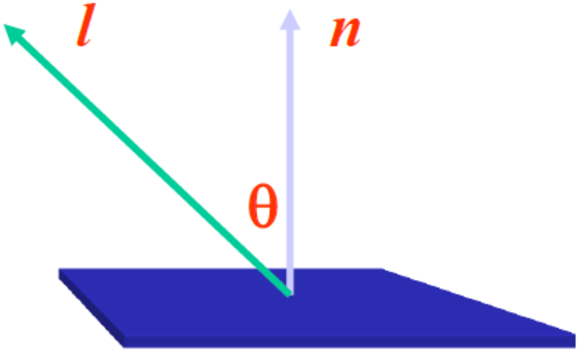


Fig. 6.3 Lambert's cosine law.

Computing Diffuse Reflection

- depends on **angle of incidence**: angle between surface normal and incoming light
  - $I_{diffuse} = k_d \, I_{light} \cos\theta$



- in practice use vector arithmetic
  - $I_{diffuse} = k_d \, I_{light} \, (\mathbf{n} \bullet \mathbf{l})$

- **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_{light}: incoming light intensity
  - $I_{diffuse}$: outgoing light intensity (for diffuse reflection)

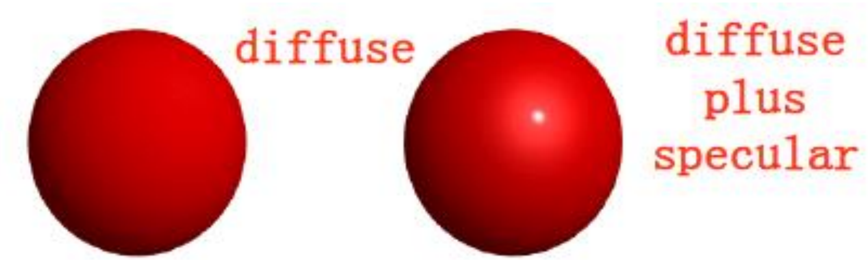
Diffuse Lighting Examples

- Lambertian sphere from several lighting angles:

- need only consider angles from 0° to 90°

Specular Reflection

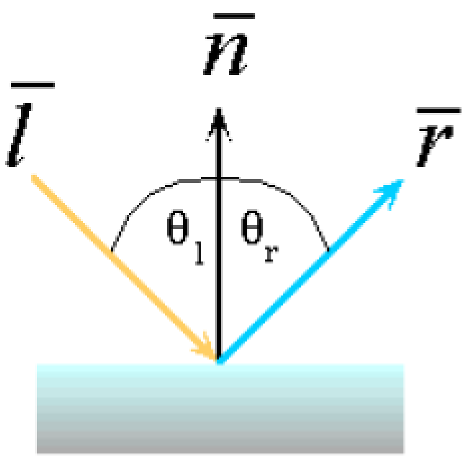
- shiny surfaces (光泽曲面) exhibit specular reflection
  - polished metal
  - glossy car



- 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

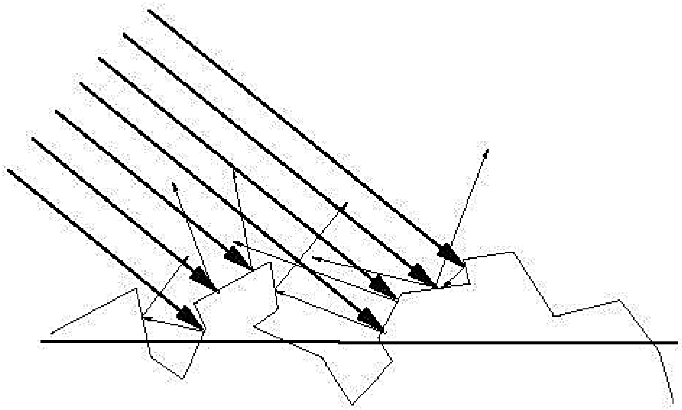
- Reflection follows **Snell's Law**:
  - incoming ray and reflected ray lie in a plane with the surface normal
  - angle the reflected ray forms with surface normal equals angle formed by incoming ray and surface normal



$$\theta_{(l)ight} = \theta_{(r)eflection}$$

Non-Ideal Specular Reflectance (Glossy 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

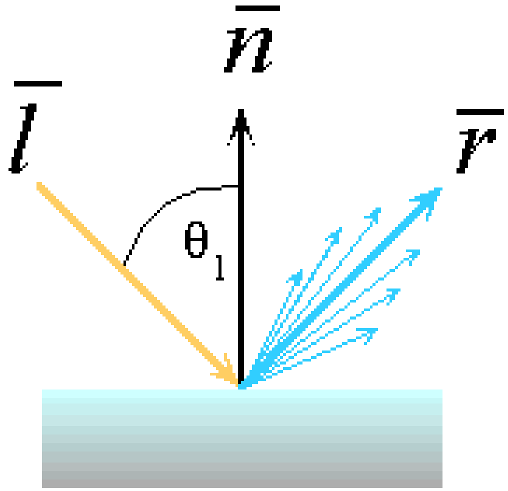


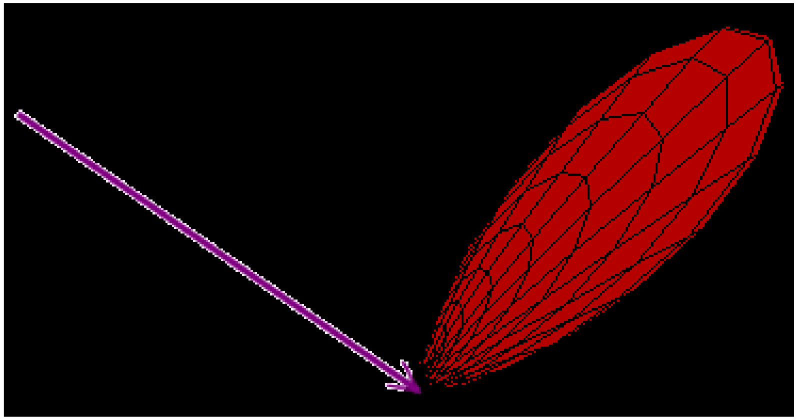
Empirical Approximation

- 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)



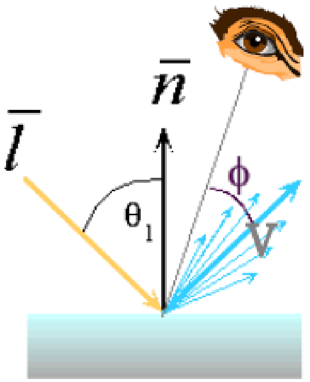


- No physical basis, works ok in practice

Empirical Approximation

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

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

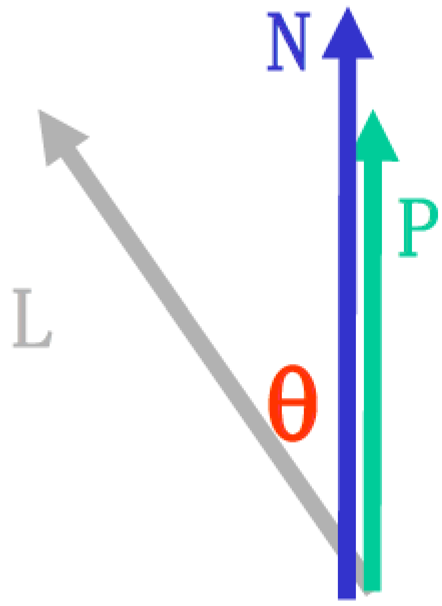


  - V is the unit vector towards the viewer
  - r is the ideal reflectance direction
  - K_s: specular component
  - I_{light}: incoming light intensity
  - n_{shiny}: purely empirical constant, varies rate of falloff(材质发光常数，值越大，表面越接近镜面，高光面积越小。)

- **How to efficiently calculate r ?**

Calculating R Vector

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



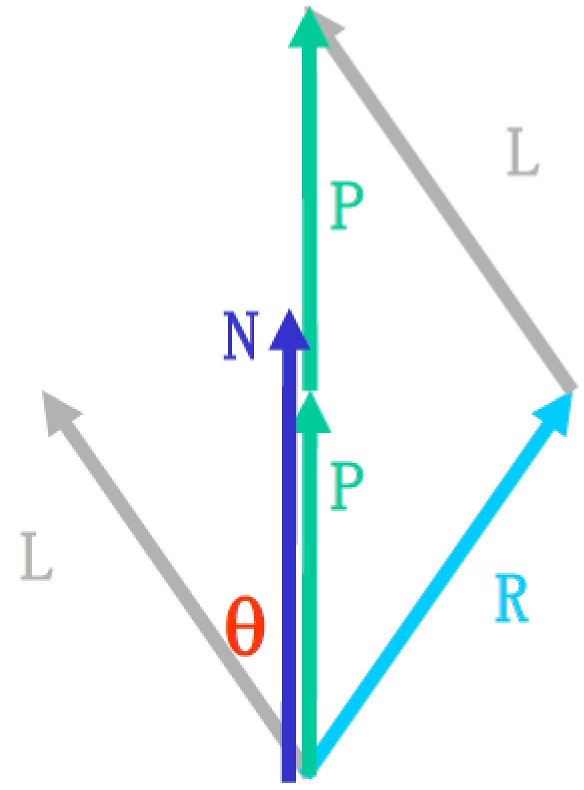
Calculating R Vector

- P = (N · L) N :
  - projection of L onto N, L, N are unit length

2 P = R + L

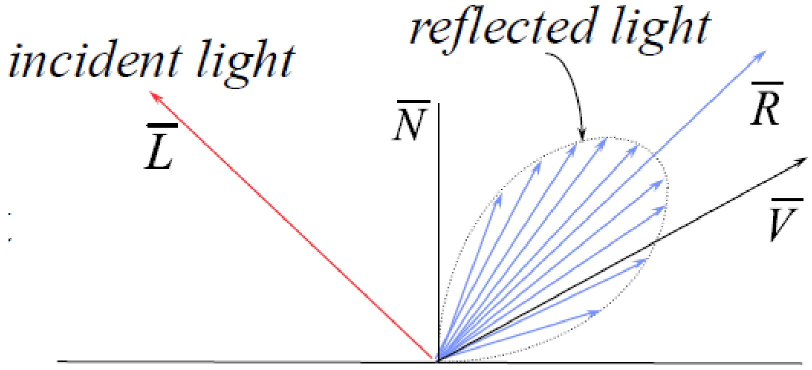
2 P – L = R

2 (N(N · L)) - L = R



Phong Illumination Model

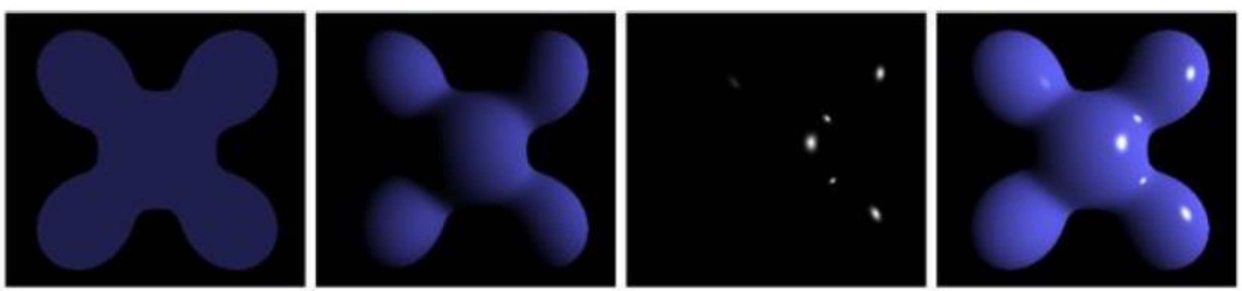
- Developed by Phong Bui-Tuong (1975) is a popular model for non-perfect reflectors
- Specular reflection of shiny objects is considered. It assumes that maximum specular reflection occurs at $\alpha = 0$
- The light calculation depends on the viewing direction
- Reflected intensity is modeled in terms of
  - Ambient component
  - Diffuse reflection component
  - Specular reflection component



Phong Illumination Model

- The illumination equation in its simplest form is given as(综合了漫反射、泛光反射分量及镜面反射)

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



Ambient + Diffuse + Specular = Phong Reflection

Phong Illumination Model

- Multilights (多点光源)

$$I = K_a I_a + \sum_{i=1}^{m} I_i (K_d \cos\alpha + k_s \cos^n \gamma)$$

- Vector arithmetic (矢量积形式)

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

Colored objects

- 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

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

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

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

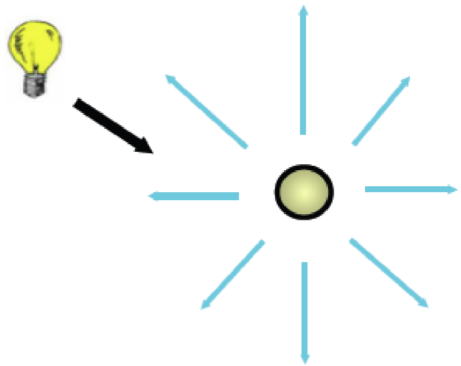
- Summarizing these three equations as single expression

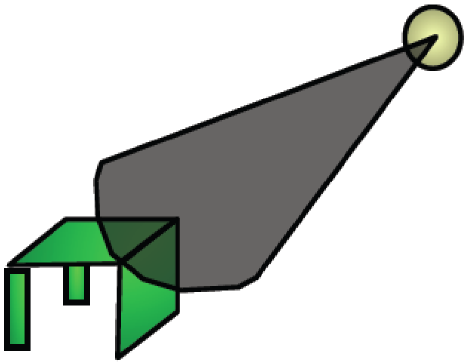
$$I(r, g, b) = I_a k_a(r, g, b) + I_p[k_d(r, g, b)(\overline{N} \bullet \overline{L}) + k_s(\overline{R} \bullet \overline{V})^n]$$

Lighting in OpenGL

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 (light that is reflected off of so many surfaces, we can't tell where it comes from)
  - 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、创建光源：

```
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);
```

Light properties

- 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.

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

Light properties

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：gDisable(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)

Material Properties

- 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_SHINESS	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

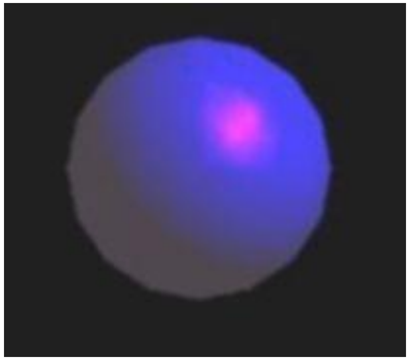
```
void myinit(void)
    {
        GLfloat mat_ambient[] = { 0.8, 0.8, 0.8, 1.0 };
        GLfloat mat_diffuse[] = { 0.8, 0.0, 0.8, 1.0 };
        GLfloat mat_specular[] = { 1.0, 0.0, 1.0, 1.0 };
        GLfloat mat_shininess[] = { 50.0 };

        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);
    }
```

Example

- 以上程序运行结果是一个蓝色的球，其中高光部分仍为上一例的亮紫色。从上可看出，球漫反射光的结果是**mat_diffuse[]**与**light_diffuse[]**中的三个颜色分量值相乘，即:

  (0.0*1.0, 0.0*1.0, 0.8*1.0, 1.0*1.0) = (0.0, 0.0, 0.8, 1.0)，

- 所以球大部分呈现蓝色。

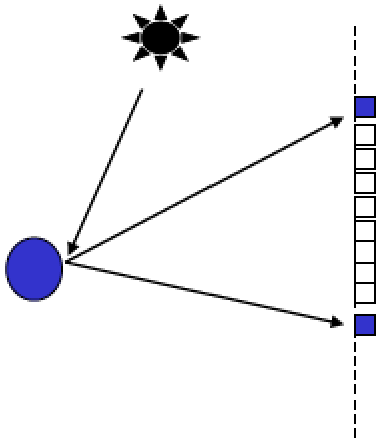


Shading

- 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

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



Applying Illumination

- we now have an illumination model for a point on a surface
- if surface defined as mesh of polygonal facets, which points should we use?
- Three shading methods:
  - Flat Shading（平面描影处理）
  - Gouraud Shading（光滑描影处理）
  - Phong Shading （Phong描影处理）

Shading Models

- Flat Shading
  - Compute Phong lighting once for entire polygon
  - Constant color

- Gouraud Shading
  - Compute Phong lighting at the vertices and interpolate lighting values across polygon
  - Interpolate colors

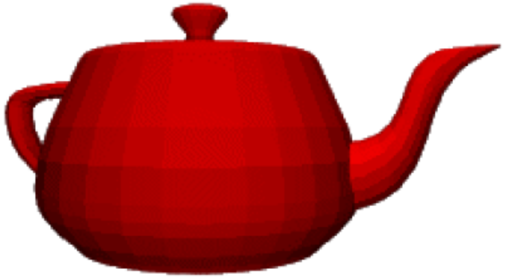
- Phong Shading
  - Compute averaged vertex normals
  - Interpolate normals across polygon and perform Phong lighting across polygon

Flat Shading

- It is the simplest of the shading models and is also called as ***faceted shading*** or ***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

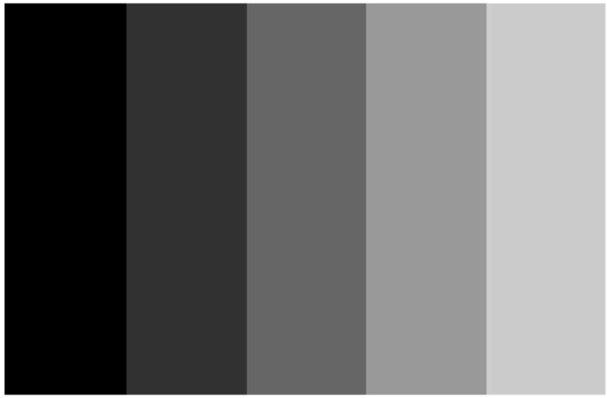
Flat Shading

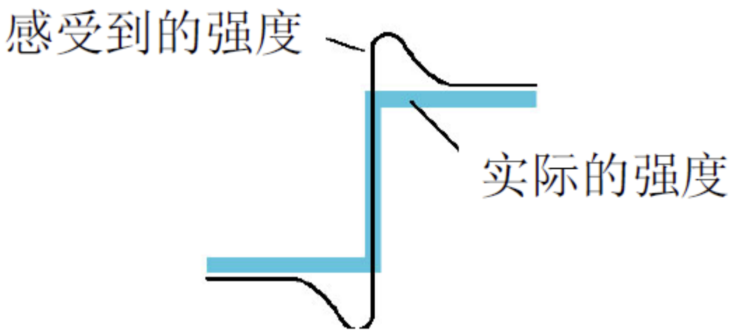
- It is a fast technique for shading as it involves very less calculations
- If the polygons are very small(say one pixel large) when projected on the screen then the result is as good as any interpolative technique
- Usually used of coarse preview of scenes
- obviously inaccurate for smooth surfaces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



Flat Shading

- 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

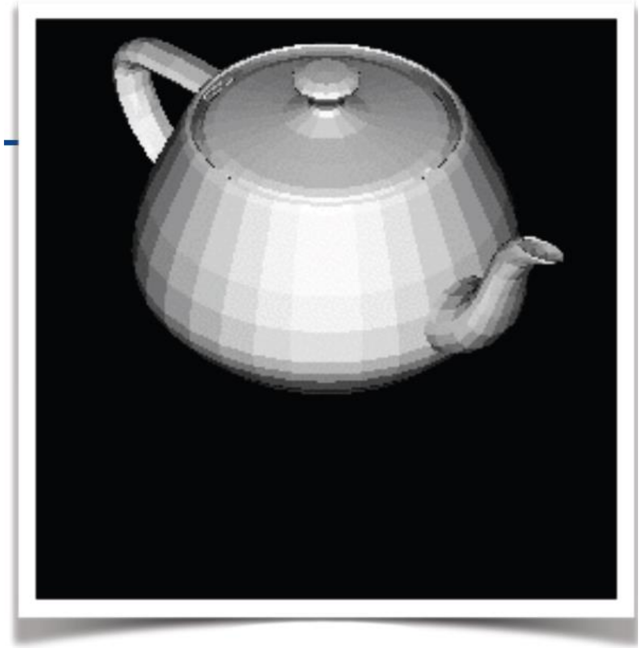




Flat Shading in OpenGL

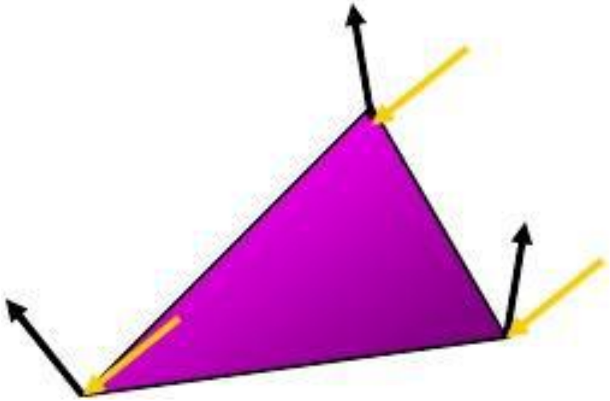
OpenGL uses the normal of the **first vertex** of a single polygon to determine the color.

glShadeModel(GL_FLAT);



Gouraud Shading

- Performs the illumination model on vertices and interpolates the intensity of the remaining points on the surface

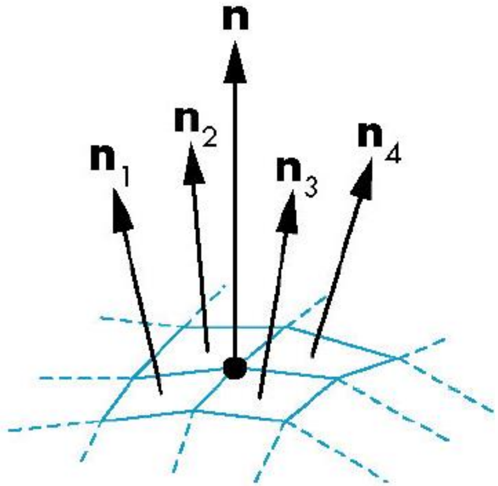


Notice that facet artifacts are still visible

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

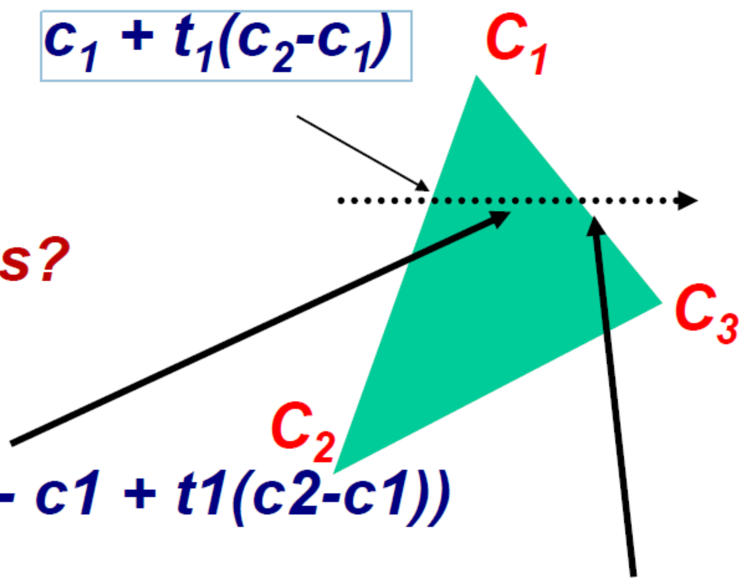
$$n = \frac{n_1 + n_2 + n_3 + n_4}{\left| n_1 + n_2 + n_3 + n_4 \right|}$$



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

$c_1 + t_1(c_2-c_1)$



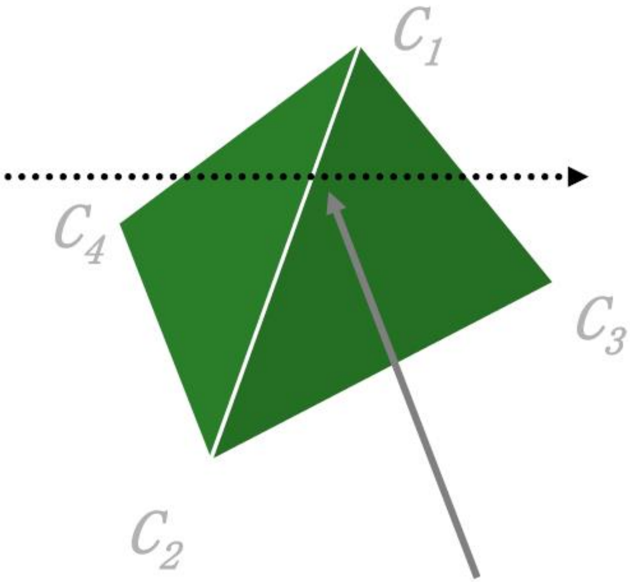
Does this eliminate the facets?

$c1 + t1(c2-c1) + t3(c1 + t2(c3-c1)- c1 + t1(c2-c1))$

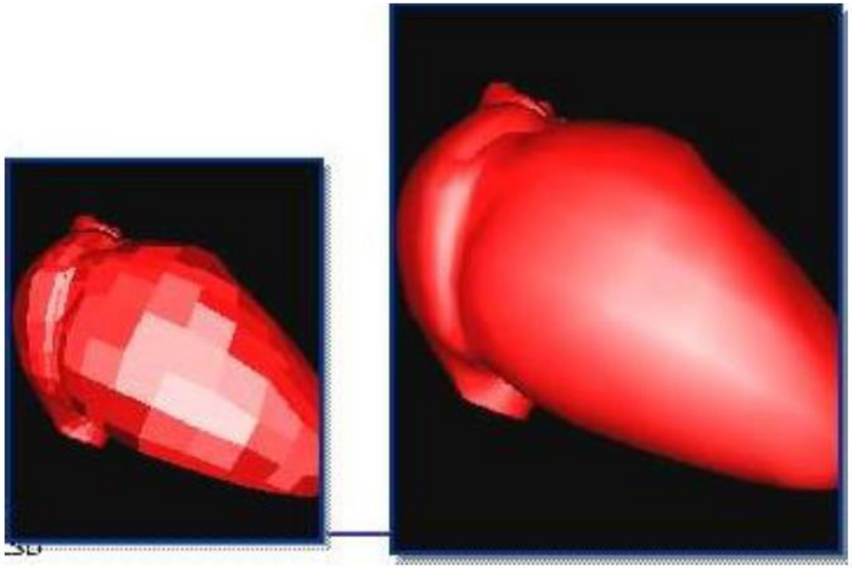
$c_1 + t_2(c_3-c_1)$

Gouraud Shading Artifacts

- **Mach bands(马赫带效应)**



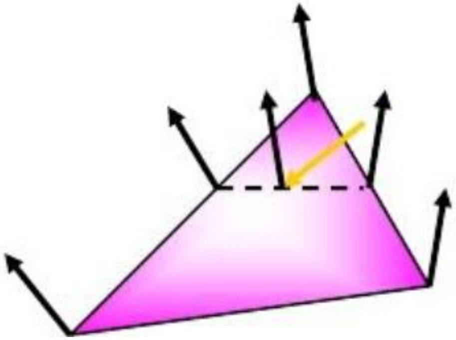
Discontinuity in rate of color change occurs here



glShadeModel(GL_SMOOTH);

Phong Shading

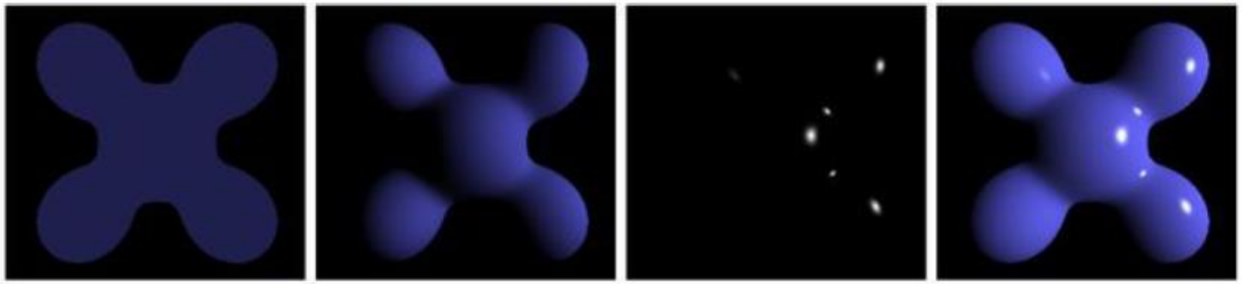
- Phong shading is <u>not</u> the same as Phong Illumination Model, though they are sometimes mixed up
  - Phong Illumination: the empirical model we've been discussing to calculate illumination at a point on a surface
  - Phong shading: linearly interpolating the surface normal across the facet, applying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



Review: Phong Illumination Model

- The illumination equation in its simplest form is given as(综合了漫反射、泛光反射分量及镜面反射)

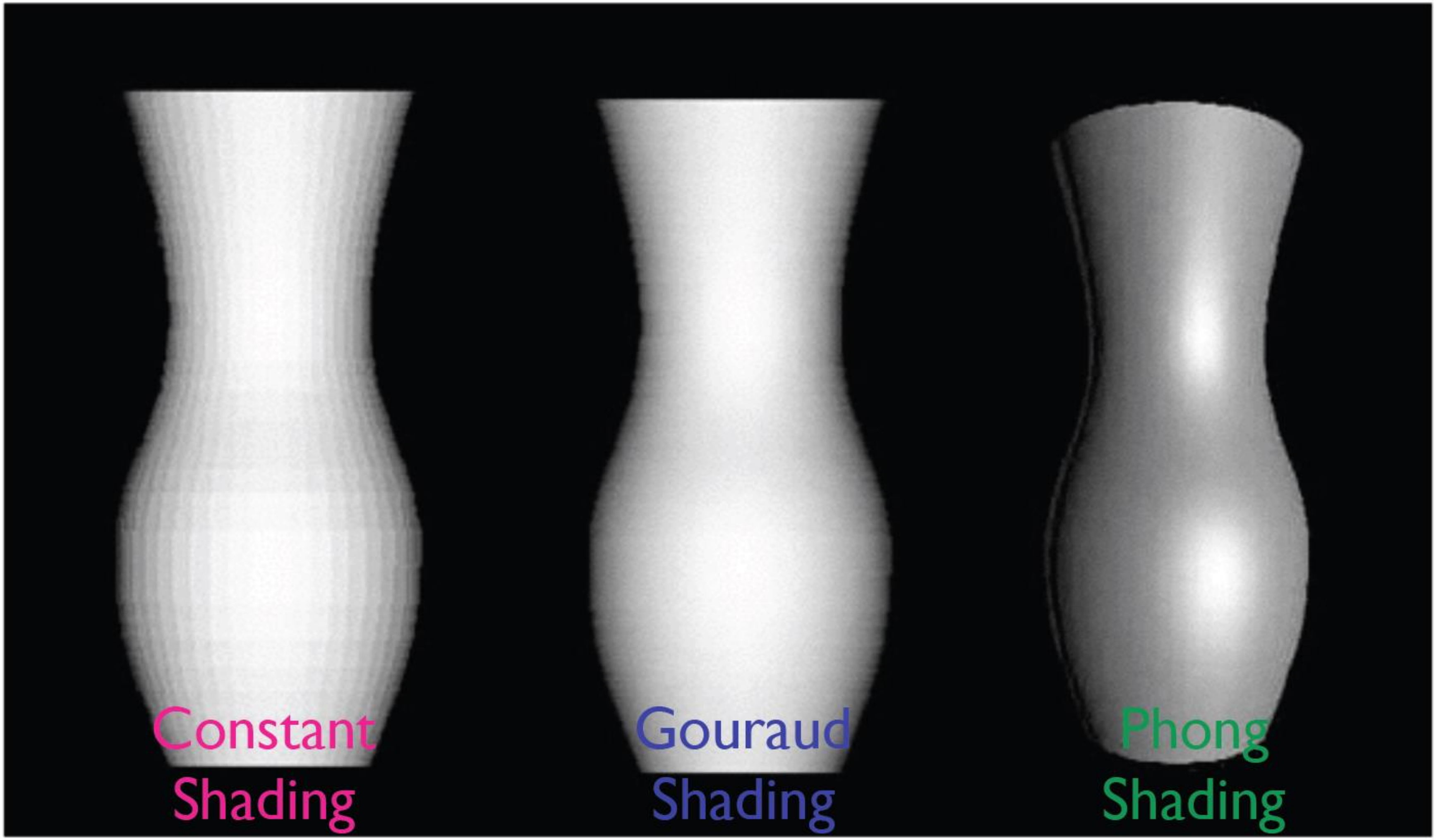
$$I = K_a I_a + K_d I_e \cos\alpha + k_s I_e \cos^n \gamma$$



Phong Shading

- It is an interpolative shading method, also called:
  - **normal-vector interpolation 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

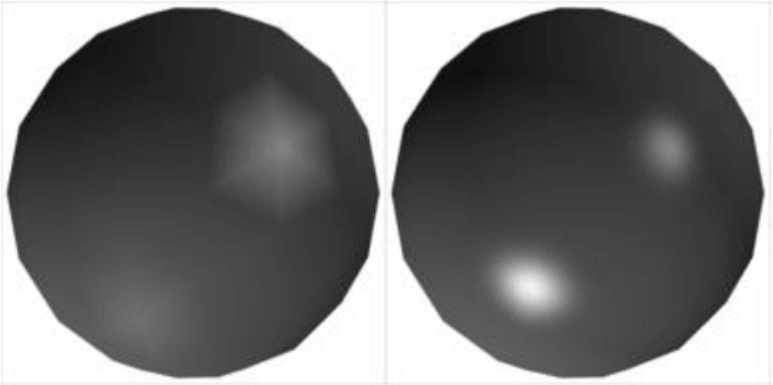


Pros and cons

– **Light intensity** changes more naturally

– **Computational cost** is higher than that of Gouraud Shading (6~8 times)

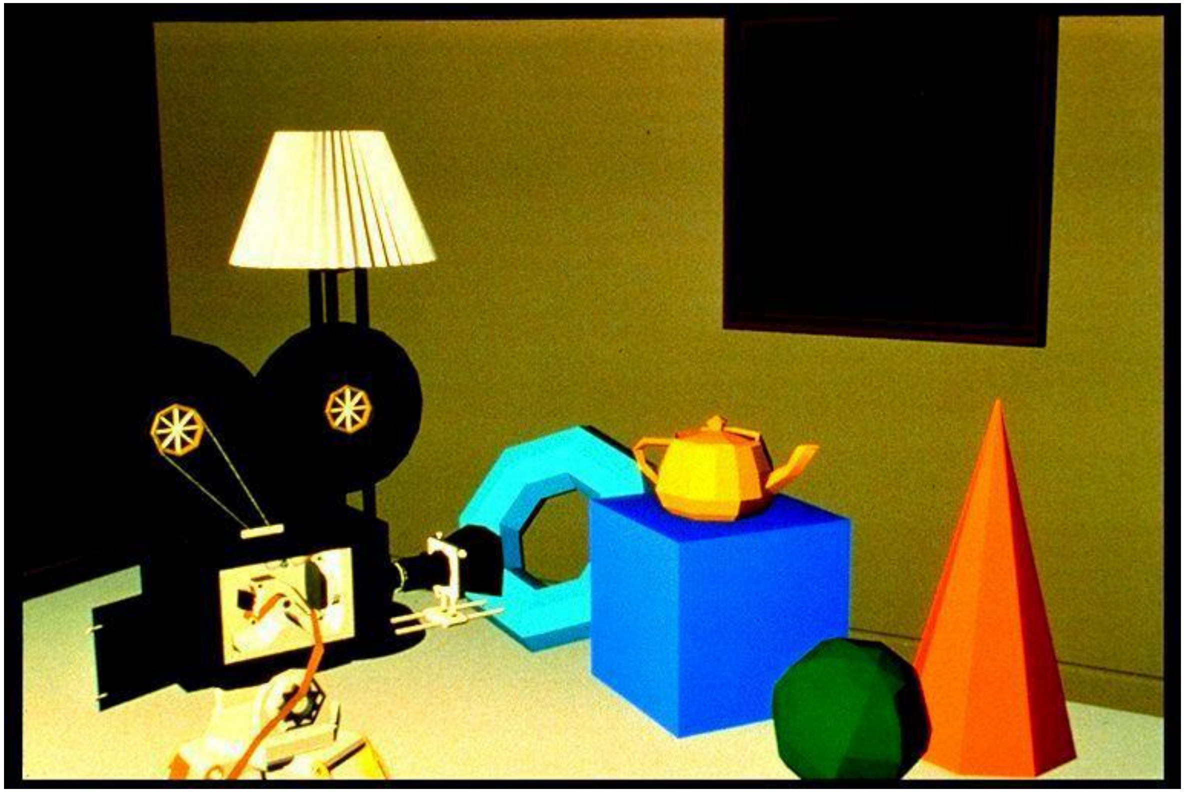
– Polygonal silhouettes remain



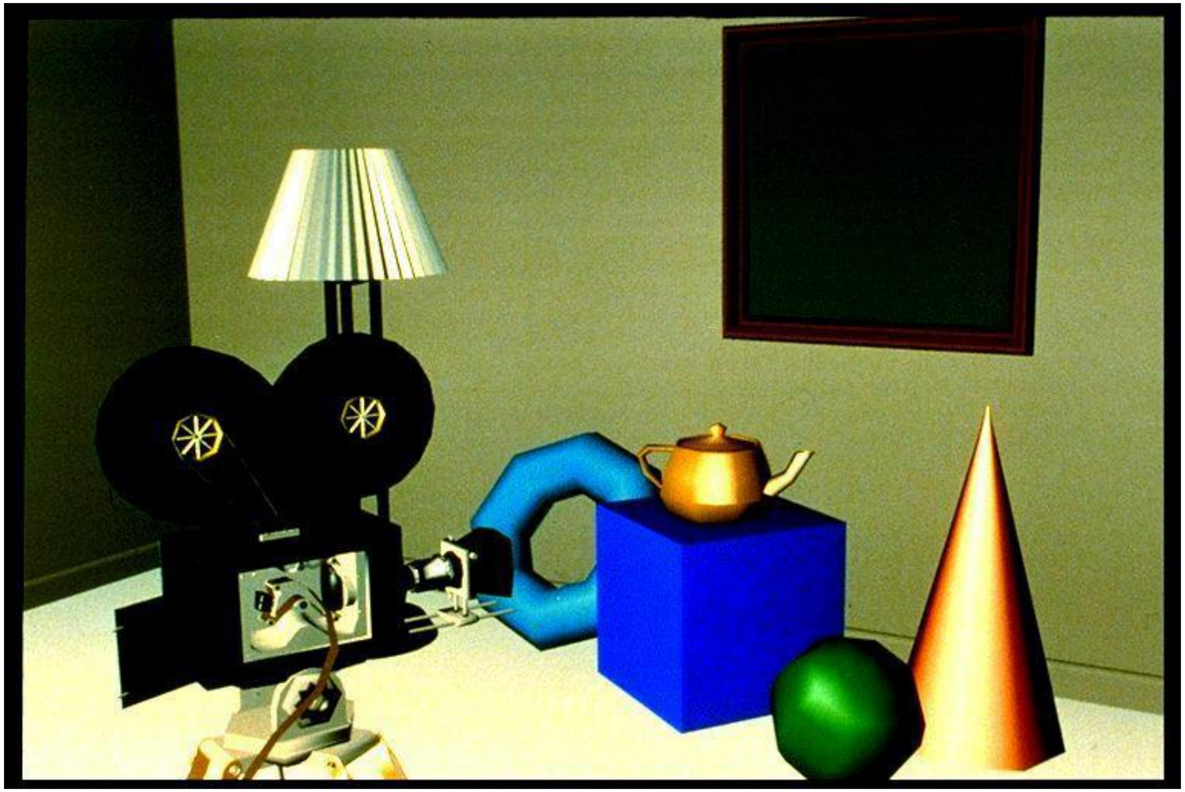
Gouraud Phong

- Solution :
  - finer subdivision for the entire surface
  - finer subdivision only along silhouette (view dependent)

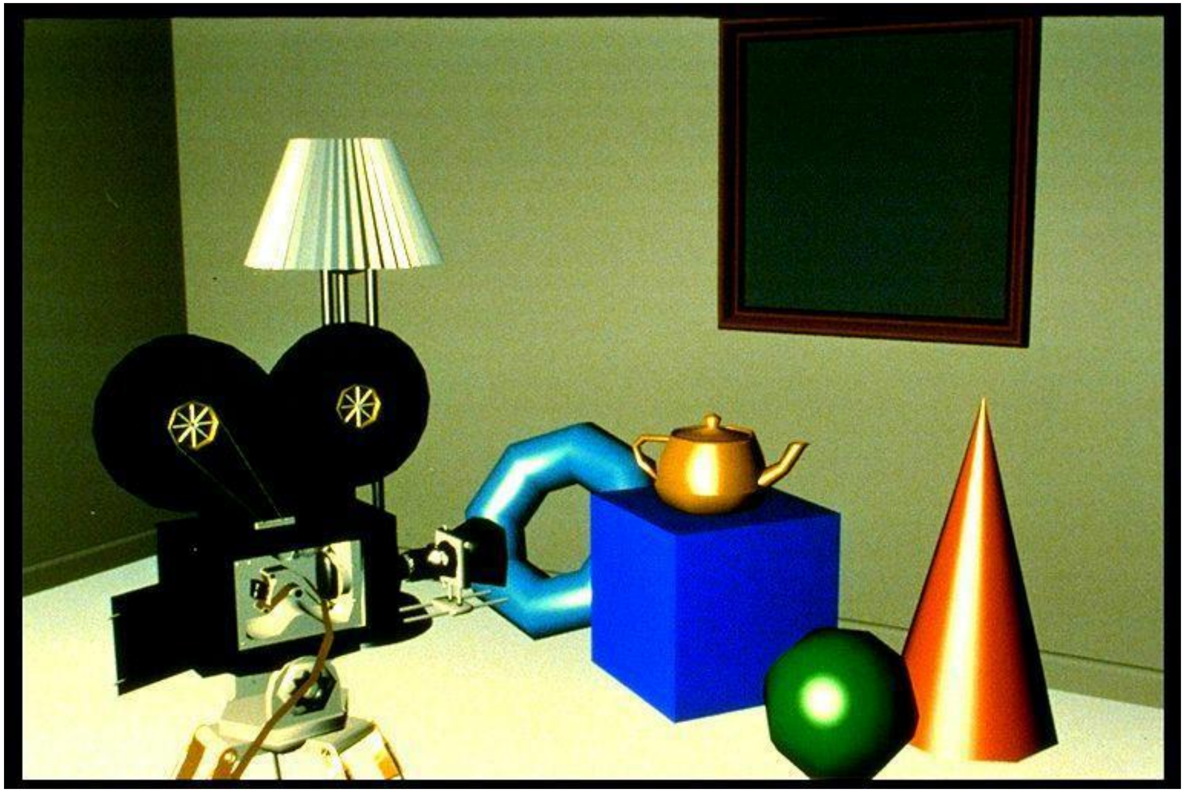
Flat Shading



Gouraud Shading



Phong Shading



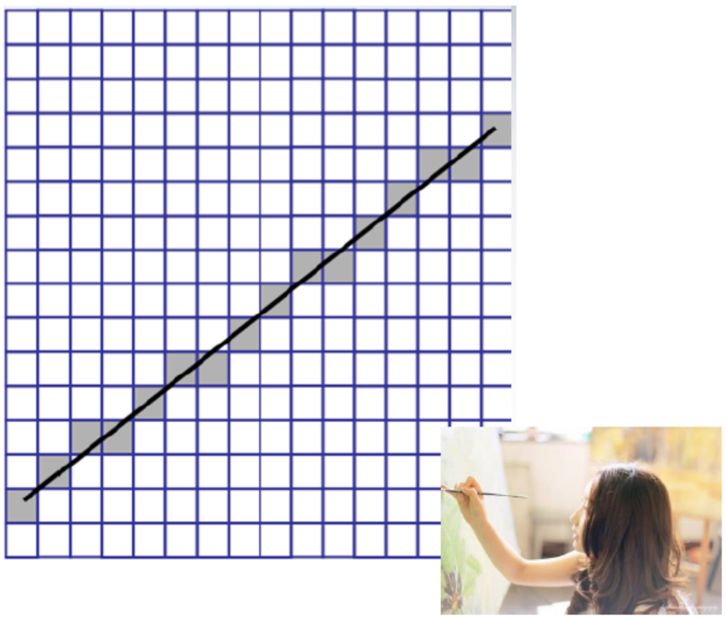
Advanced Rendering

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)

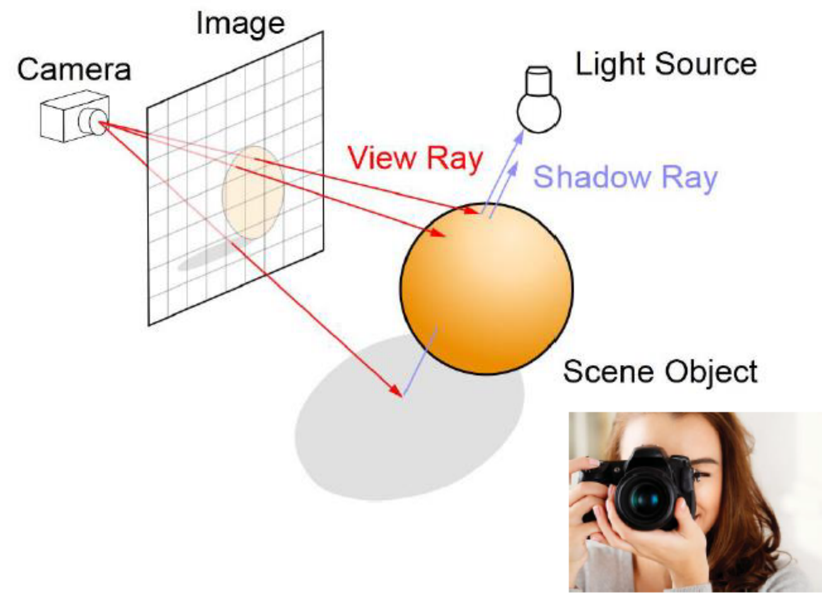
Recall : what is ray tracing

In CG, drawing is...



scan conversion(rasterization)

photography is...



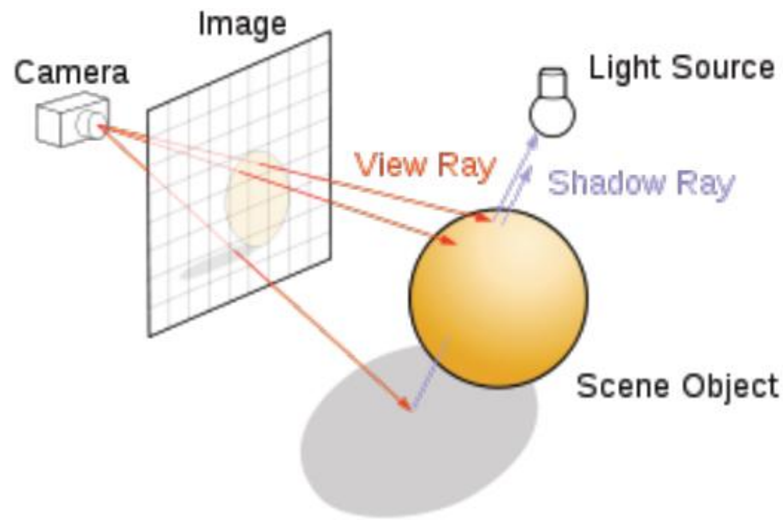
ray tracing

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 direct illumination
  - shading due to global illumination
  - shadows

Features

- Best known for handling shadows, reflections and refractions
- It is an algorithm that works entirely in object space, hence accurate
- Partial solution to global illumination problem and 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



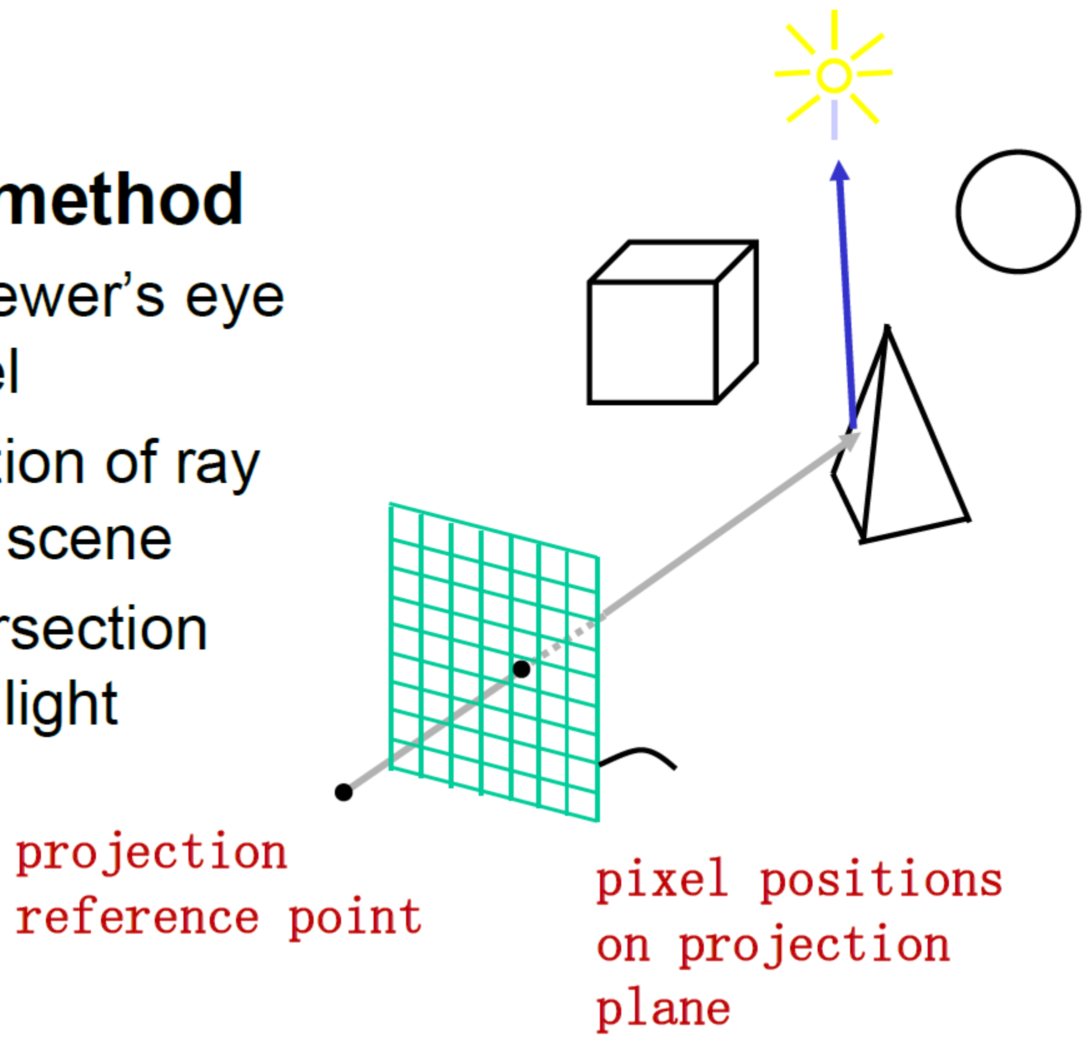
Ray Tracing

Ray Tracing


Canon
AE-1
PROGRAM
No. 213987
ø 58
f=28-80 mm
Niccor-XR7
1:3.9/4.9
Credits: Mike Miller using Pov-Ray

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

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

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

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

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

Ray-Sphere Intersection

- Sphere Representation:
  - center **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

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

Ray-Sphere Intersection

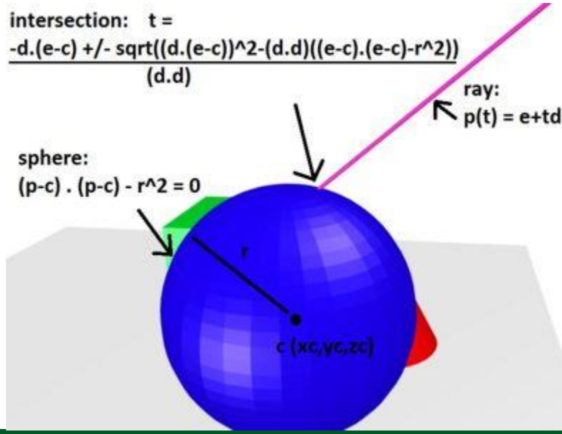
- This is a quadratic equation of the form

$$
\begin{aligned}
&A \bullet t^2 + B \bullet t + C = 0 \\
&\textit{where,} \\
&A = x_d^{\ 2} + y_d^{\ 2} + z_d^{\ 2} = 1 \\
&B = 2 \bullet \left(x_d \bullet (x_o - x_c) + y_d \bullet (y_o - y_c) + z_d \bullet (z_o - z_c)\right) \\
&C = (x_o - x_c)^2 + (y_o - y_c)^2 + (z_o - z_c)^2 - r^2
\end{aligned}
$$

- the two roots are given by

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

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



Ray-Plane Intersection

- The plane is represented by the equation

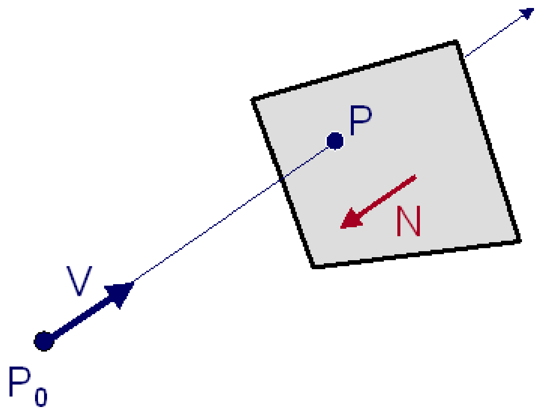
$$a \bullet x + b \bullet y + c \bullet z + d = 0$$

- Substituting the ray equation into the plane equation we have

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

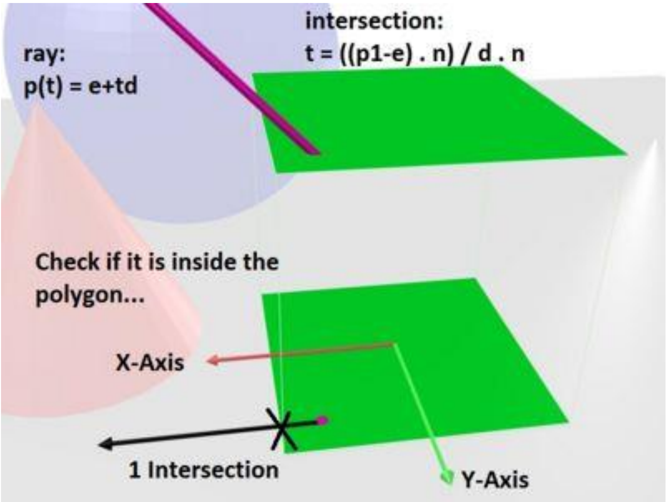
- Solving for t

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



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/

Fantastic work from CAD Lab

- RenderAnts Pro (GPU based)
  - http://www.gaps-zju.org/project/renderants.html

Aliyun Render

- Rendering Cloud System (cloud based @ aliyun)
  - https://rendering.aliyun.com/