Computer Graphics

Lighting & Shading

Teacher: A.prof. Chengying Gao(高成英)

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

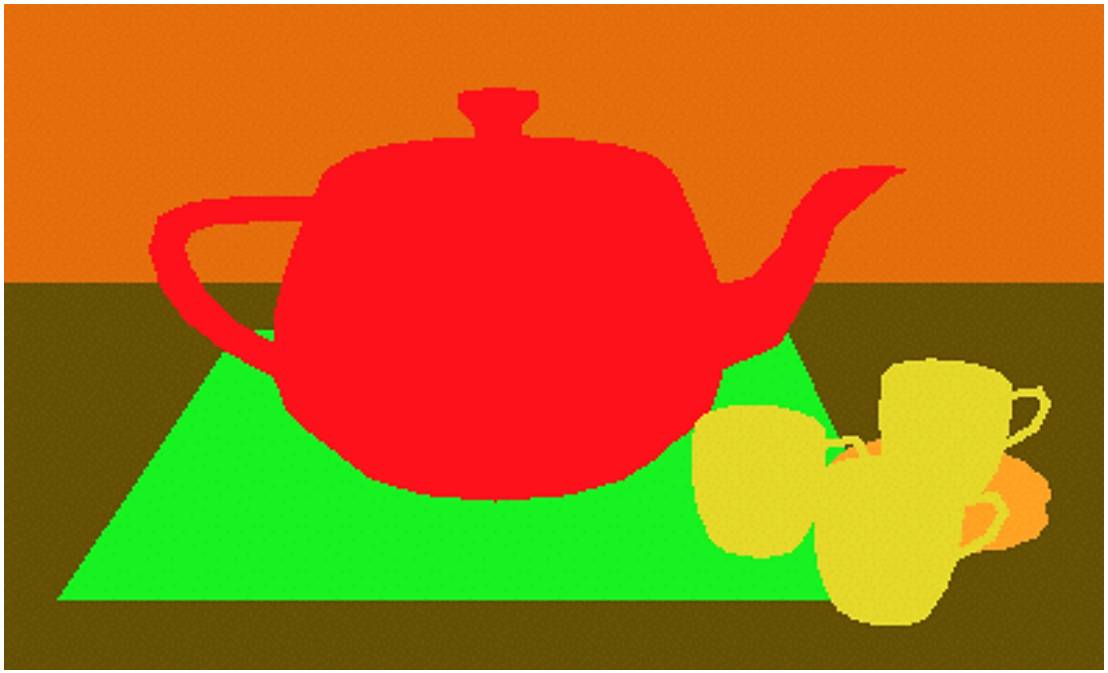
School of Data and Computer Science

Lighting & Shading

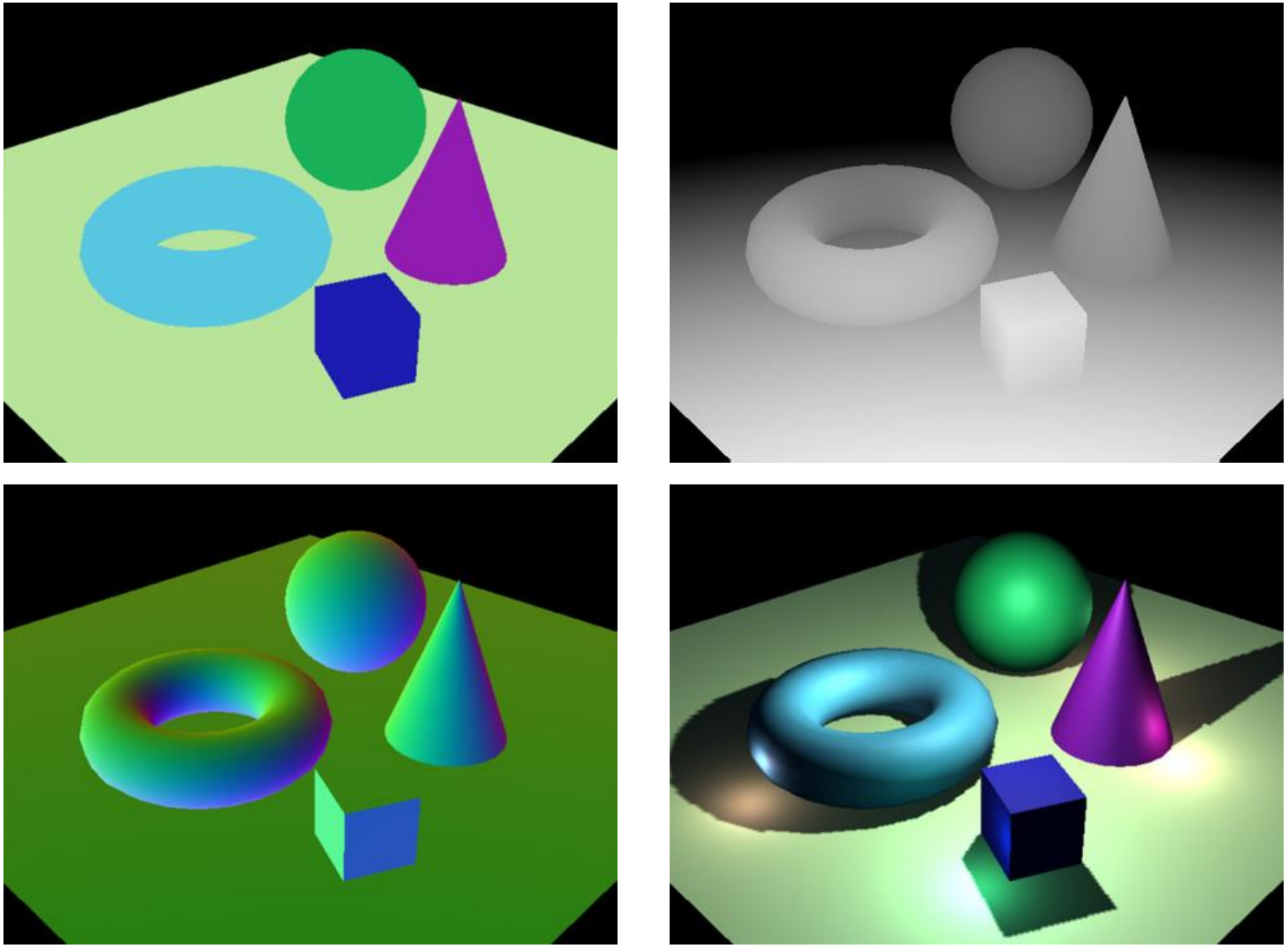
- Without light…
- We do not see much of our scene!

Lighting & Shading

- Without shading…
- Objects do not look three dimensional

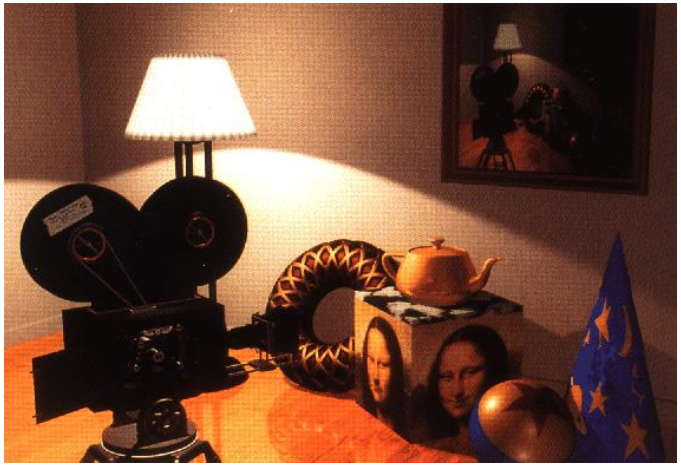


Lighting & Shading



Lighting/Shading

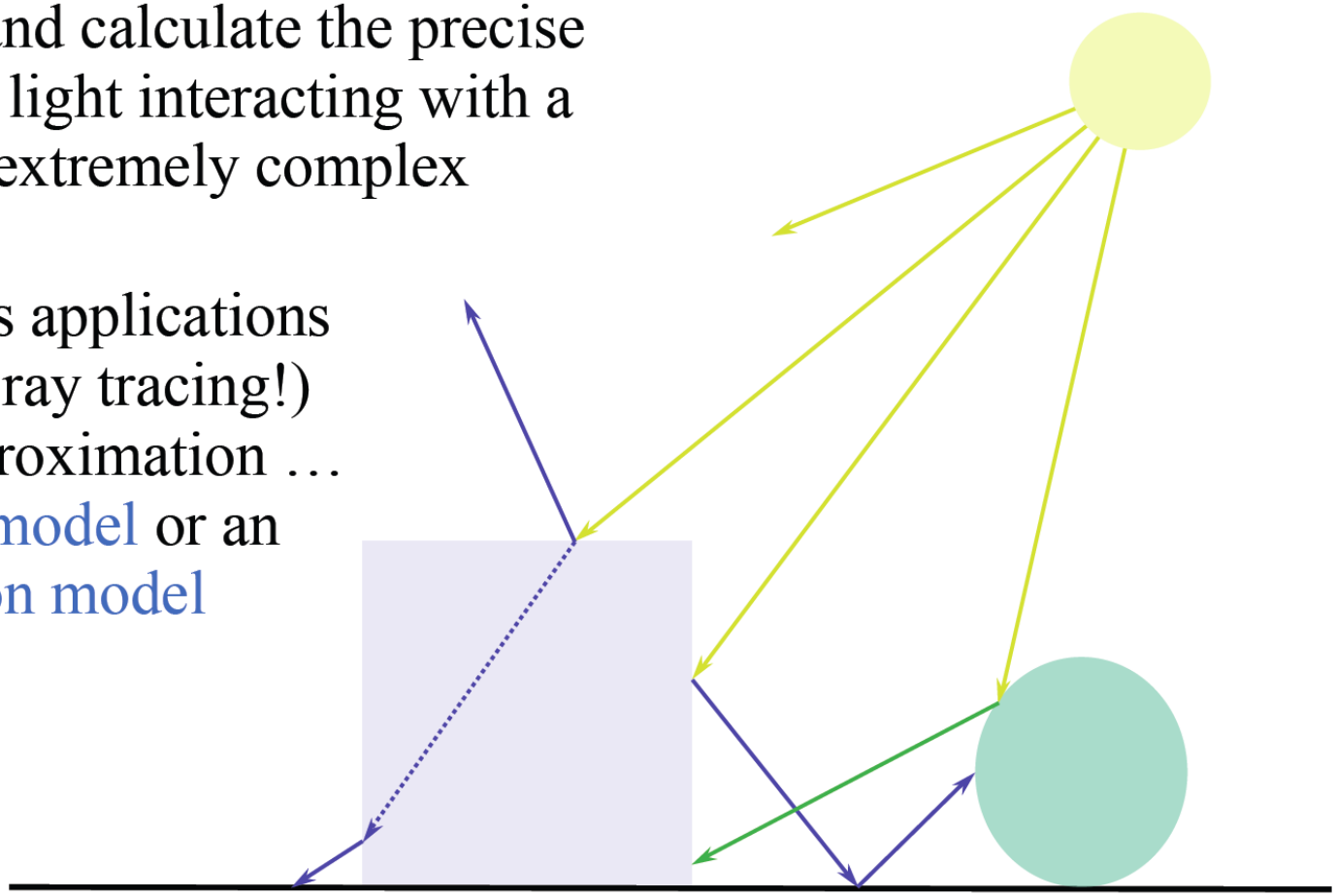
- Goal
  - Model the interaction of light with surfaces to render realistic images
  - Generate per (vertex/pixel) color



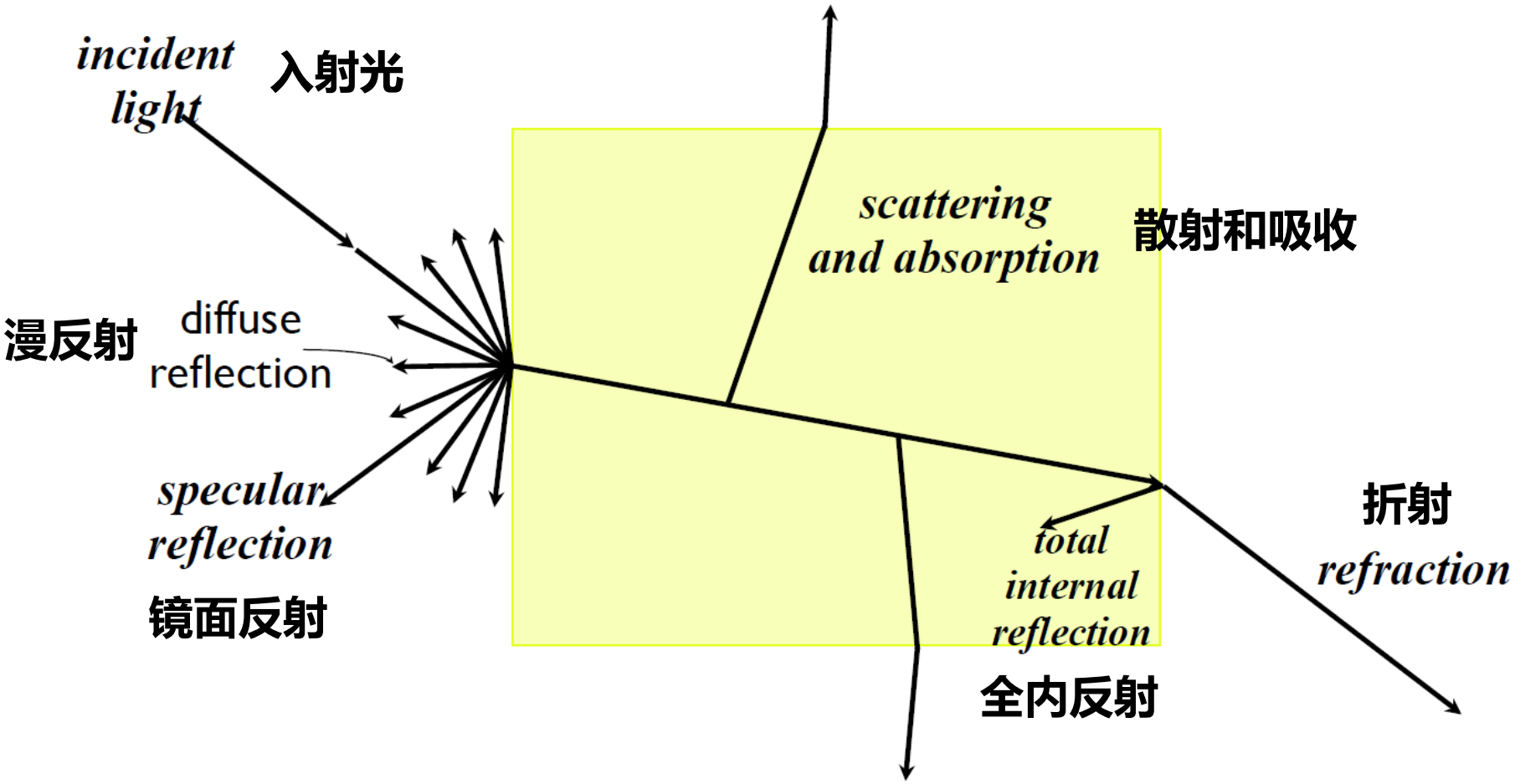
Light interaction in a Scene

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

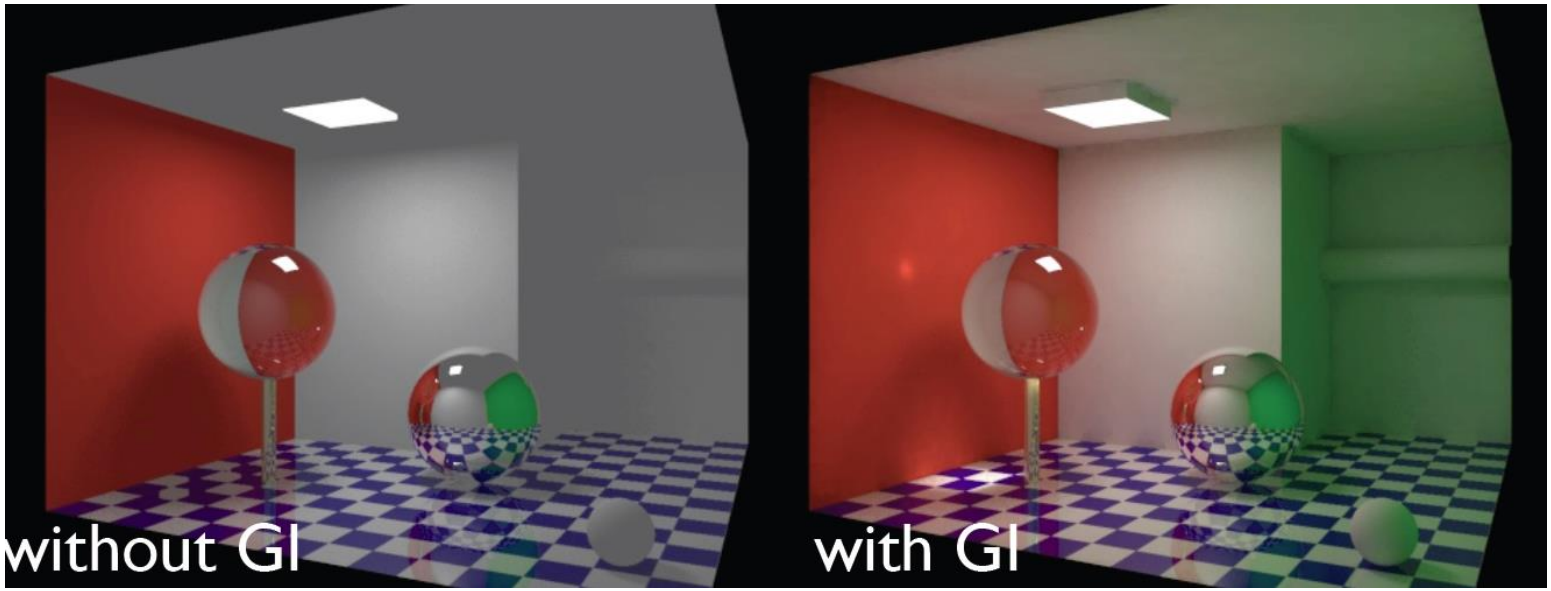


Illumination model

- 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

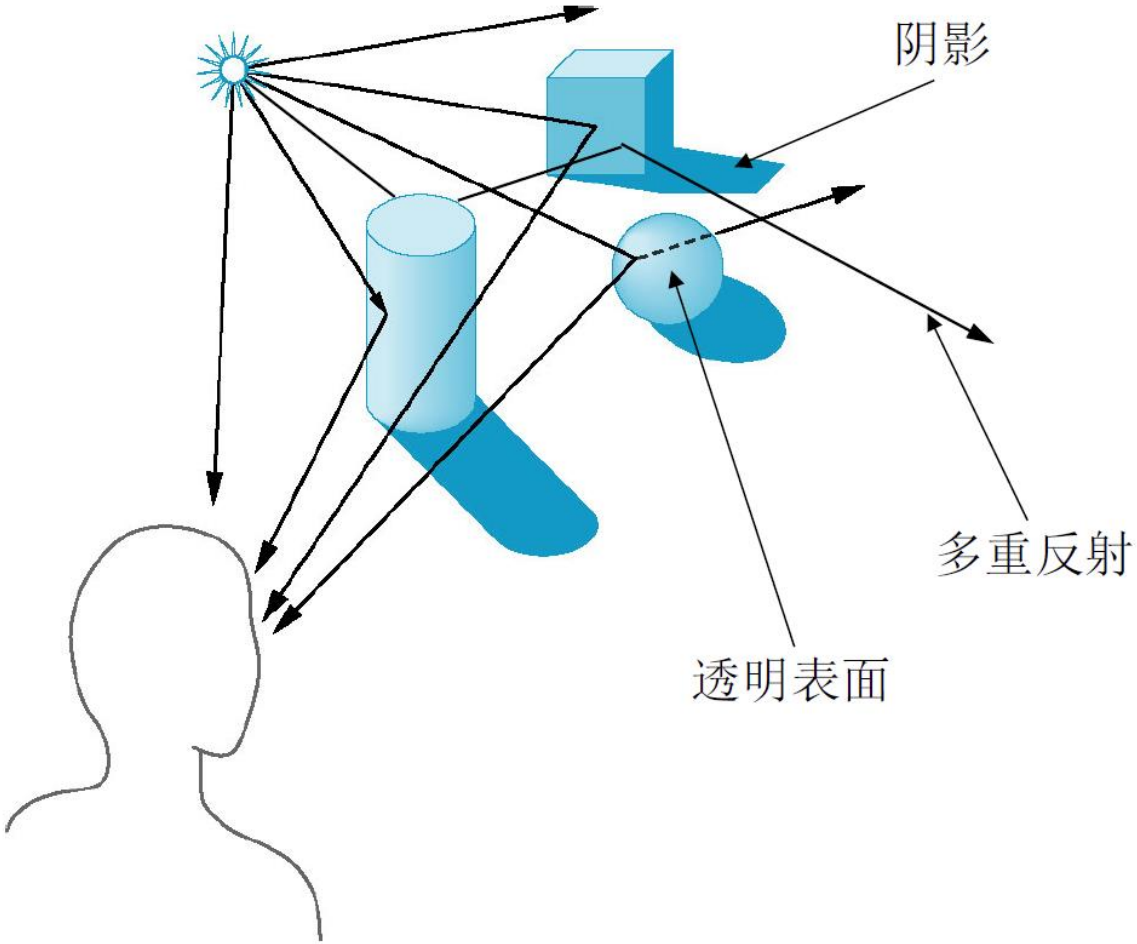
- A surface point could be illuminated by
  - **local illumination**：Concerned with how objects are directly illuminated by light sources
  - **global illumination**, light reflected from and transmitted through its own and other surfaces



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

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)

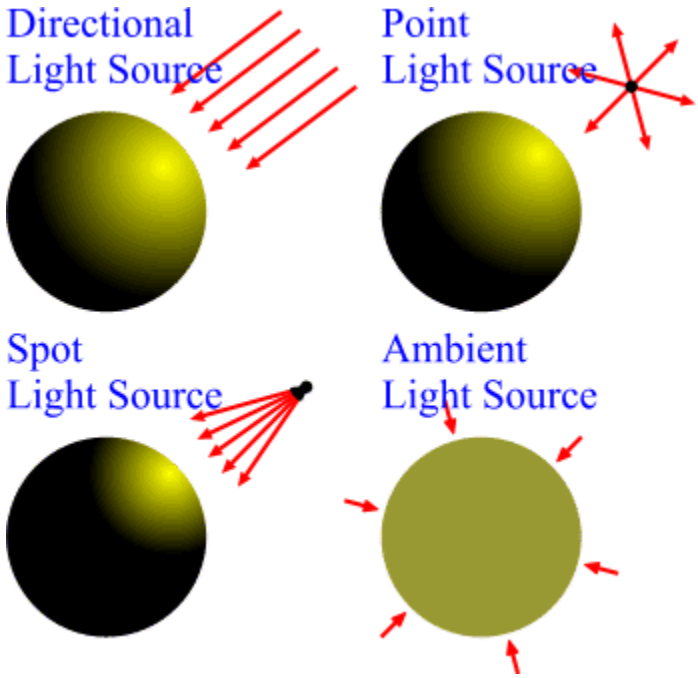
- Light: energy in a range of wavelengths
  - White light – all wavelengths
  - Colored (e.g. red) – subset of wavelengths
- Surface “color” – reflected wavelength
  - White – reflects all lengths
  - Black – absorbs everything
  - Colored (e.g. red) – absorbs all but the reflected color
- Multiple light 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 : $\mathbf{I} = [I_r, I_g, I_b]$

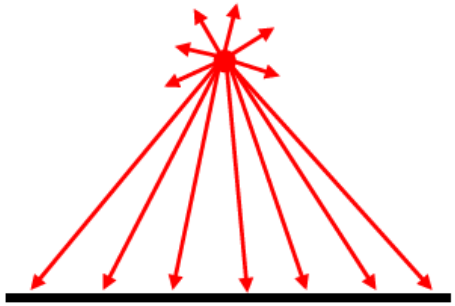
Types of Light Sources

- Point source
- Parallel source
- Ambient lights
- Spotlights



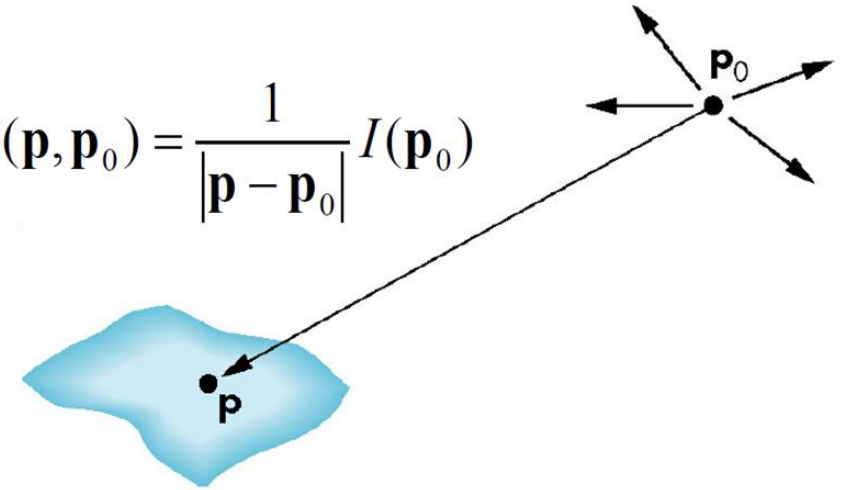
Point source

- A point light source emits light equally in all directions from a single point
- The intensity function of point source: $\mathrm{I}(\mathbf{p}_0) = [I_r(\mathbf{p}_0), I_g(\mathbf{p}_0), I_b(\mathbf{p}_0)]$
- The intensity of P is inversely proportional to the distance between P_0 and P.
- defined by **location** only



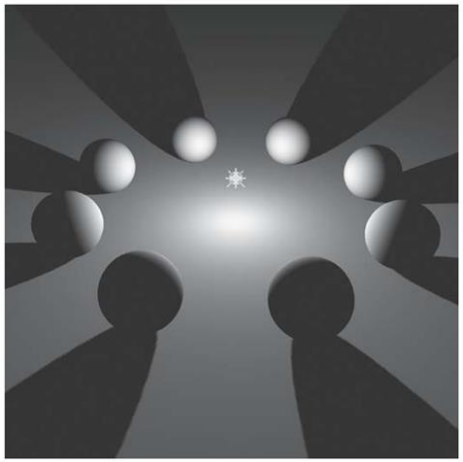
$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$i(\mathbf{p}, \mathbf{p}_0) = \frac{1}{|\mathbf{p} - \mathbf{p}_0|} I(\mathbf{p}_0)$$



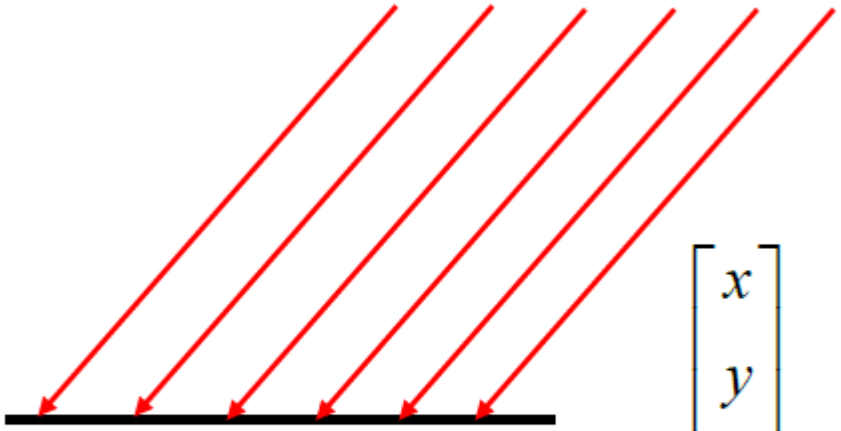
Application of Point Light

- 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 scene: Objects appear too bright, or very dark.



Parallel source

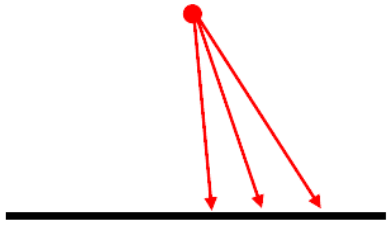
- light rays are parallel
- Rays hit a planar surface at identical angles
- May be modeled as point source at infinity
- Directional light
- defined by **direction** only

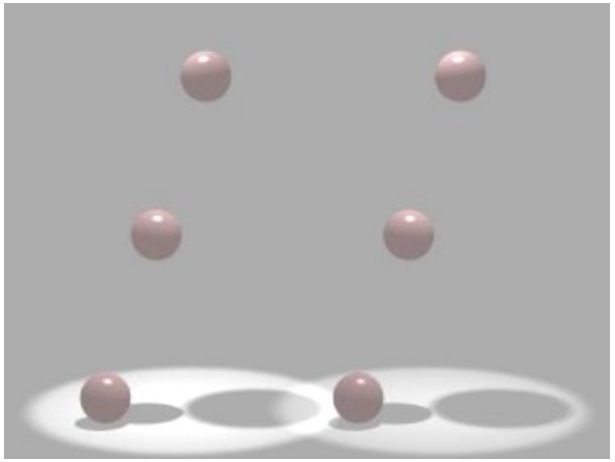


$$\begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

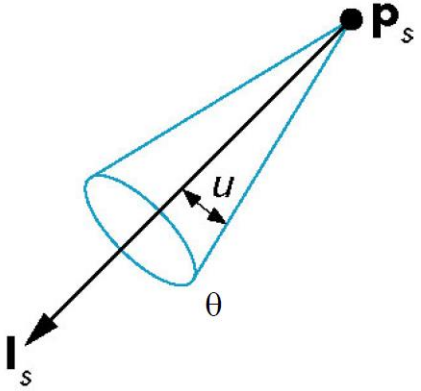
Spotlights

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

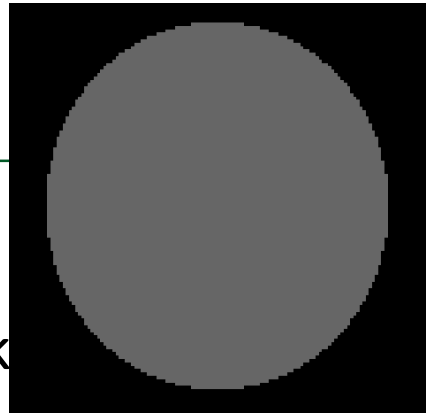




- 可以给点光源加上一定的限制得到
- 锥的顶点在P_s，而中心轴方向为l_s。
- 如果$\theta = 180°$，聚光灯成为点光源



Ambient Lights（环境光）



- 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 any where
- 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, Kambient
  - The intensity, Iambient, of the ambient light source (constant for all points on all surfaces)
  - $I_{reflected} = K_{ambient} I_{ambient}$

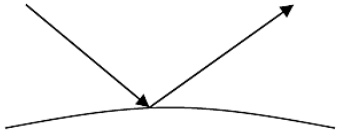
Materials

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

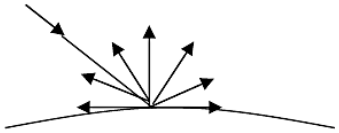


Types of Reflection

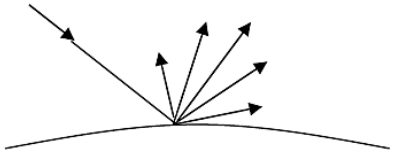
- **specular reflection** (a.k.a. *mirror* or *regular*) causes light to propagate without scattering.



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

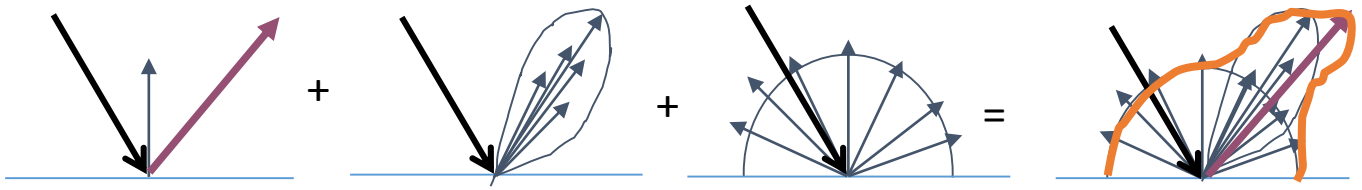


- **glossy/mixed reflection** is a weighted combination of specular and diffuse.

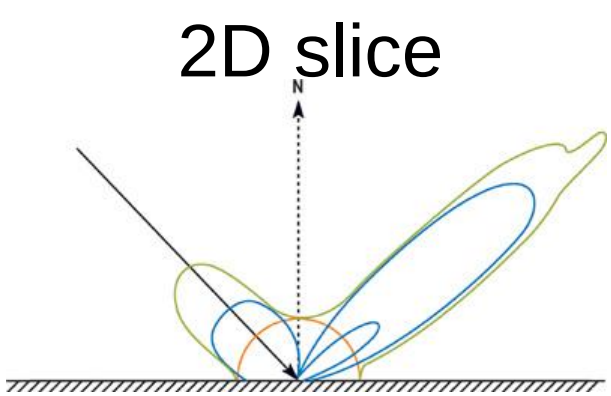


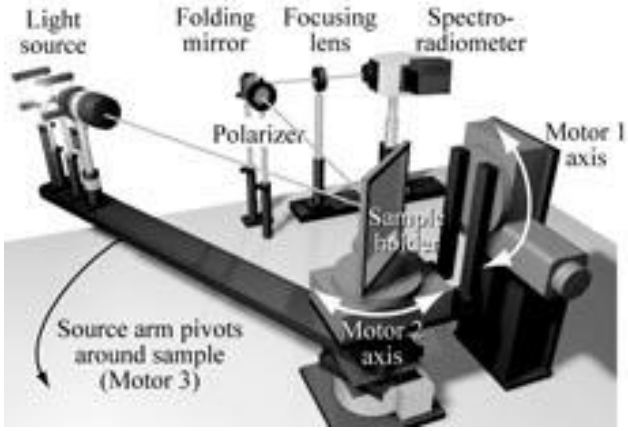
Reflectance Distribution Model

- 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。

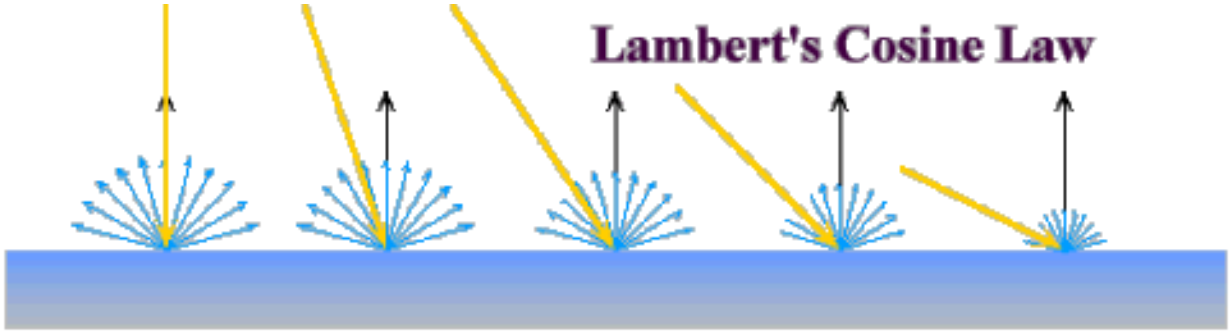
Physics of Diffuse Reflection

- 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!

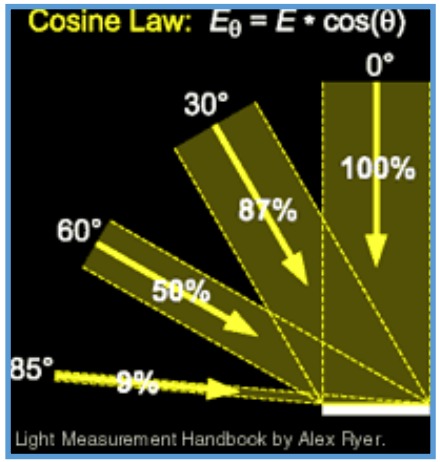
Lambert's Cosine Law（朗伯余弦定律）

- 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

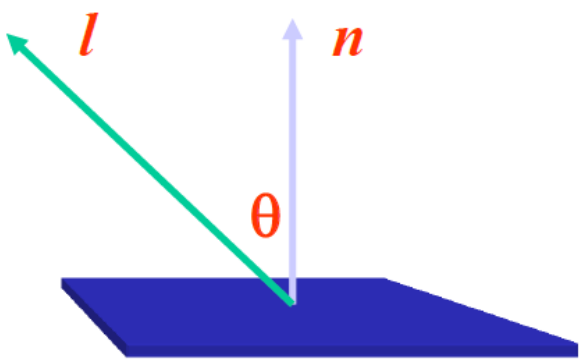


Intuitively: cross-sectional area of the “beam” intersecting an element of surface area is smaller for greater angles with the normal.



Computing Diffuse Reflection

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



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

- **always normalize vectors used in lighting!!!**
  - **n**, **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:

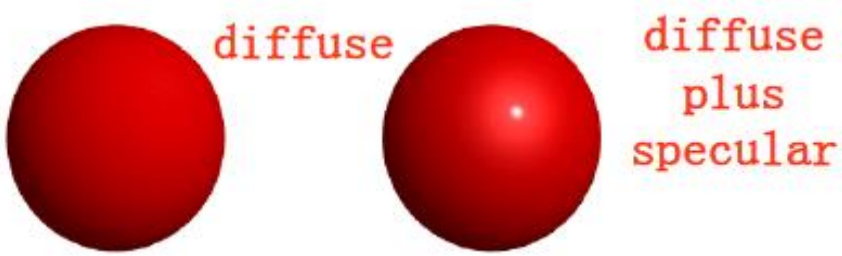
- need only consider angles from 0° to 90°

Specular Highlights

Michiel van de Panne

Specular Reflection

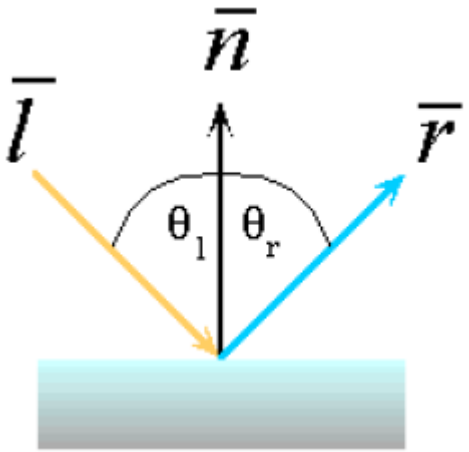
- shiny surfaces (光泽曲面) exhibit specular reflection
  - polished metal
  - glossy car



- 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

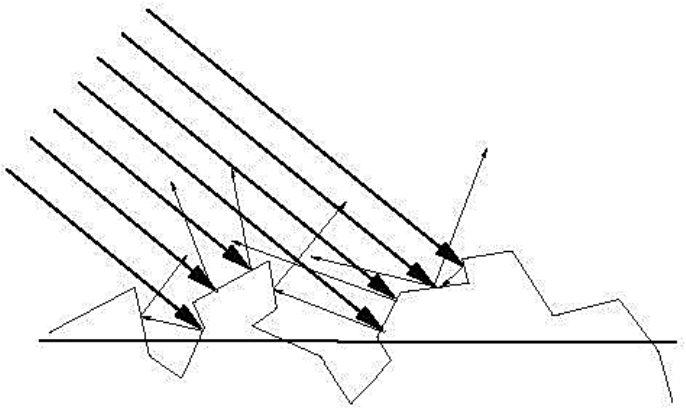
- Reflection follows **Snell's Law**:
  - incoming ray and reflected ray lie in a plane with the surface normal
  - angle the reflected ray forms with surface normal equals angle formed by incoming ray and surface normal



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

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

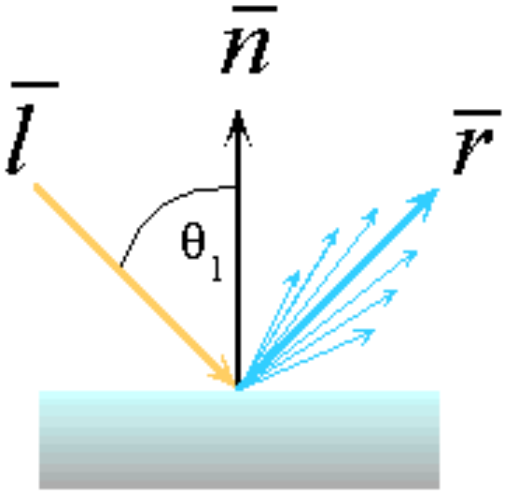


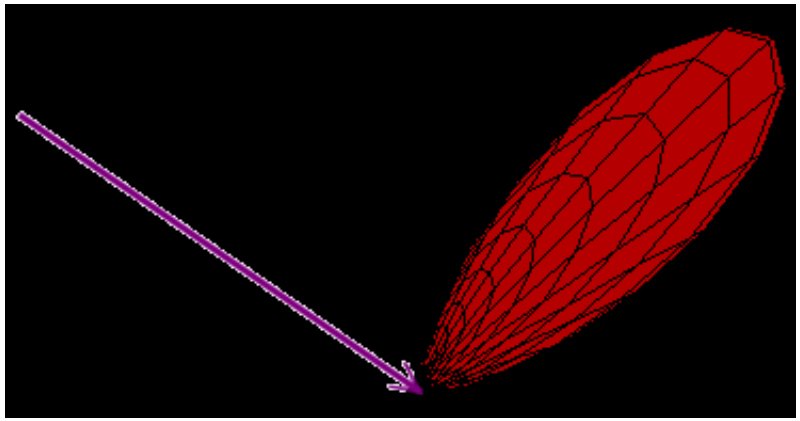
Empirical Approximation

- 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)



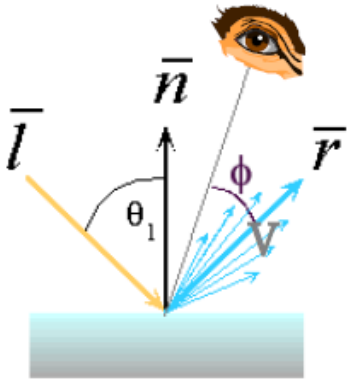


- No physical basis, works ok in practice

Empirical Approximation

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

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

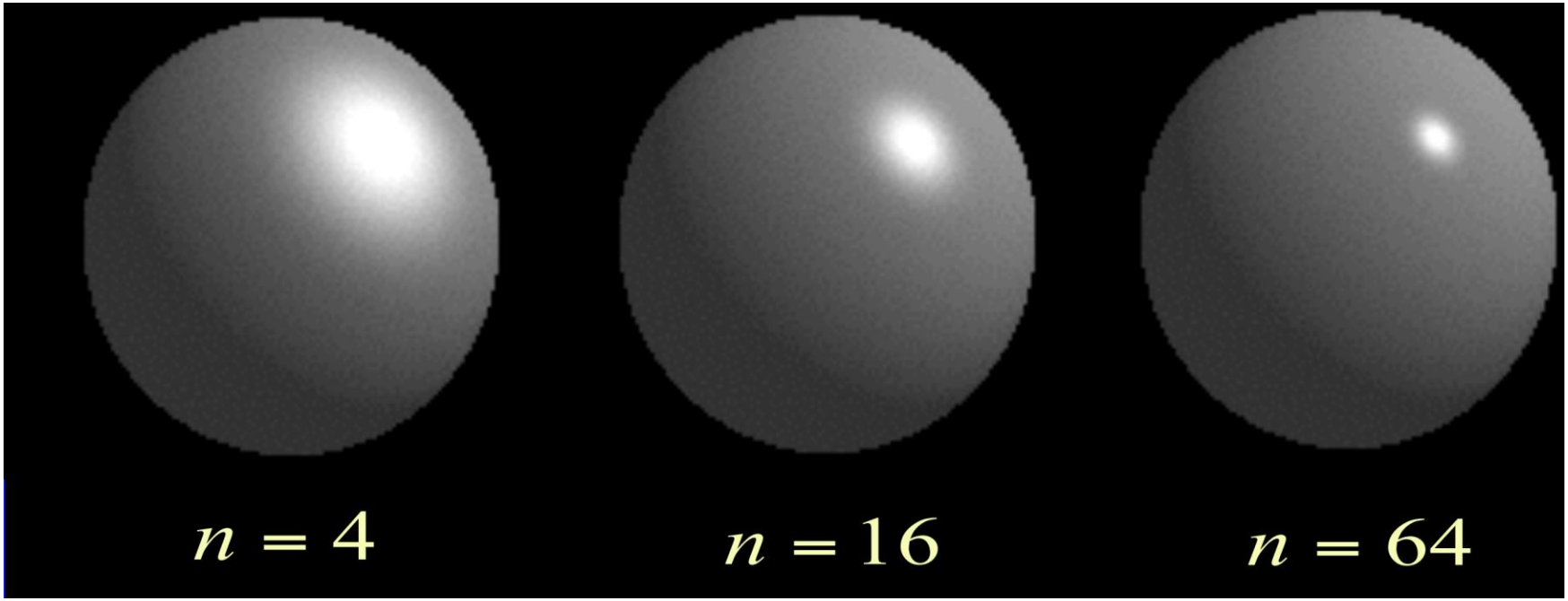


  - V is the unit vector towards the viewer
  - r is the ideal reflectance direction
  - K_s: specular component
  - I_{light}: incoming light intensity
  - n_{shiny}: purely empirical constant, varies rate of falloff(材质发光常数，值越大，表面越接近镜面，高光面积越小。）

- **How to efficiently calculate r ?**

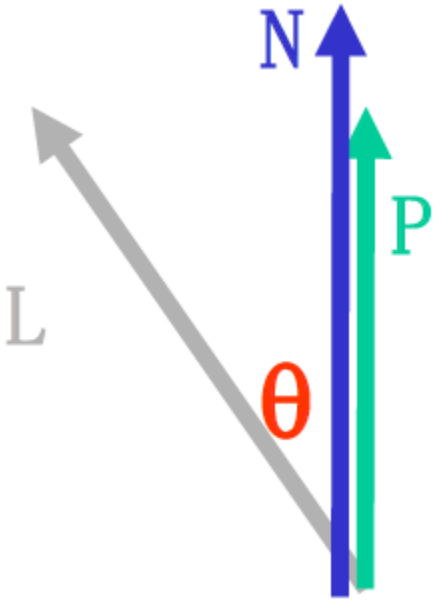
Empirical Approximation

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



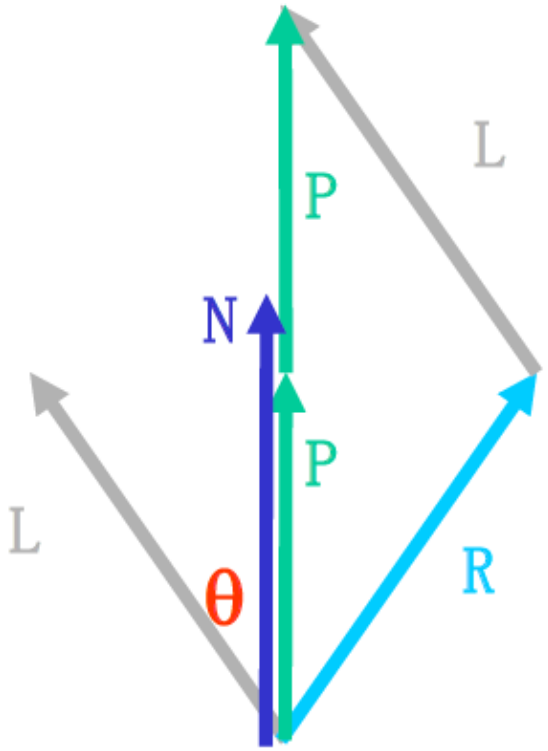
Calculating R Vector

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

$$2P = R + L$$

$$2P - L = R$$

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



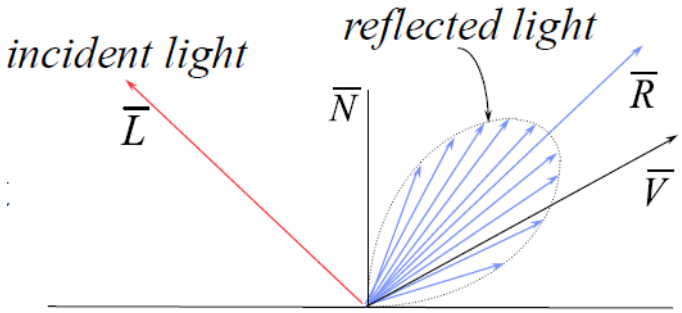
Ambient Light（环境光）

- 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
- Local lighting model
- Reflected intensity is modeled in 3 parts of
  - Diffuse reflection component
  - Specular reflection component
  - Ambient component(to approximate the global Illumination effects)

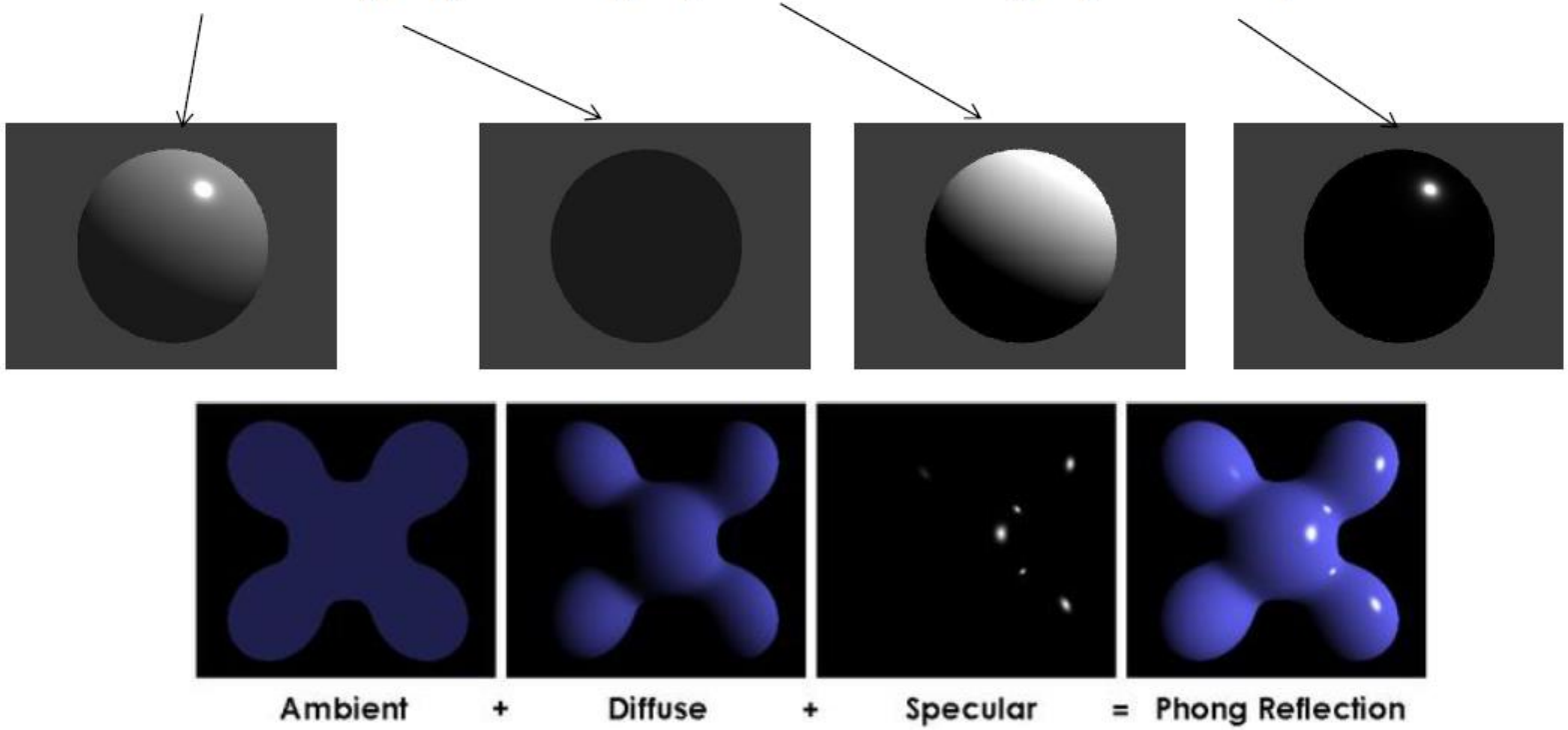


- 经验模型并不是基于物理原理，而是提出经验公式，通过调整参数来模拟光照。

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)

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



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

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

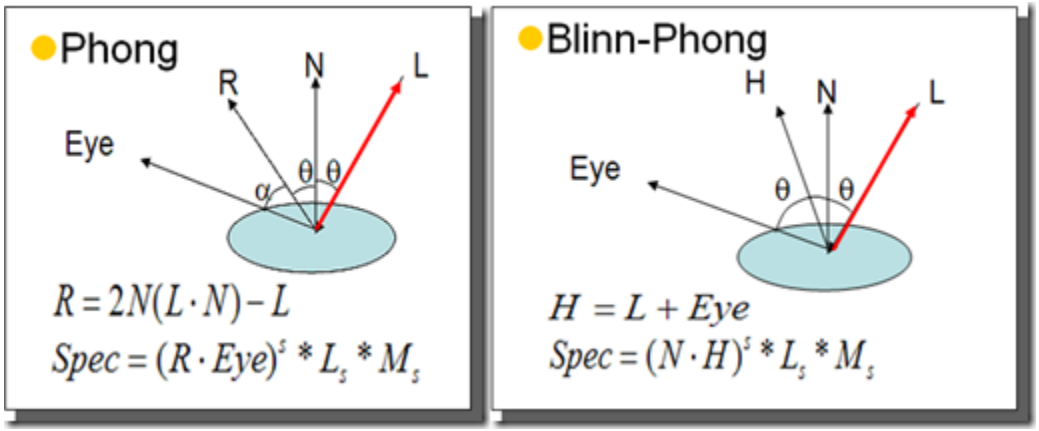
$$I_g = I_a k_{ag} + I_p[k_{dg}(\overline{N} \bullet \overline{L}) + k_s(\overline{R} \bullet \overline{V})^n]$$

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

- Summarizing these three equations as single expression

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

Blinn-phong Illumination Model



- 相比较Phong模型, Blinn-phong模型只是用N•H替换了V•R, 其中H是V和L的中间向量。求解H的计算量大大小于R的求解计算量，同时Blinn-phong模型能提供比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。

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

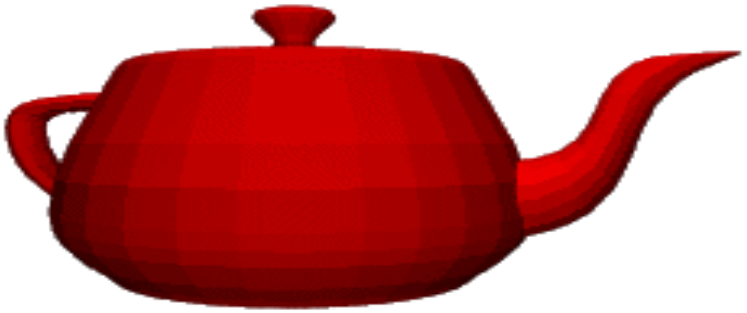
- Shading Models give a technique to determine the colors of all the pixels covered by a surface using appropriate illumination model
- Polygonal meshes are commonly used for representing complex surfaces
- The geometric information is available only at the vertices of a polygon
- Interpolative shading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



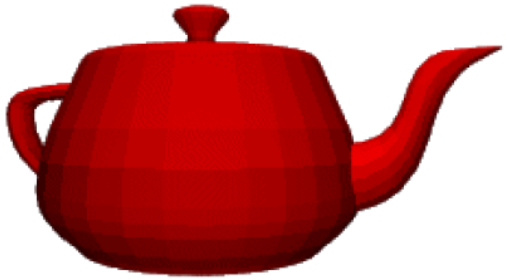
- obviously inaccurate for smooth surfaces

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

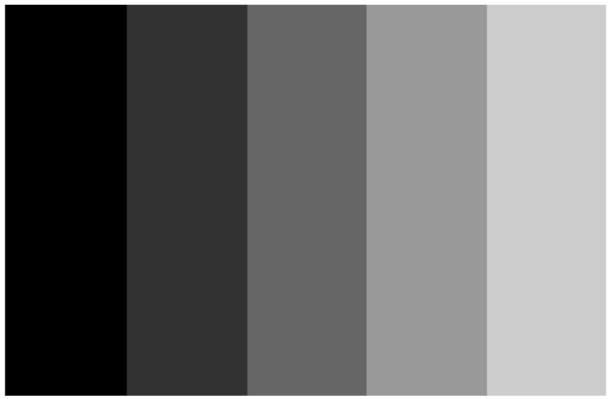
Flat Shading

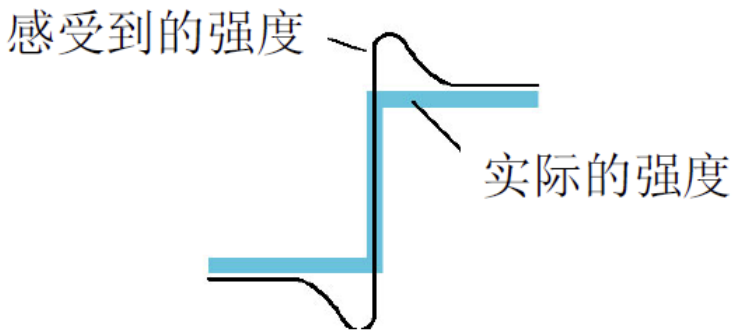
- It is a fast technique for shading as it involves very less calculations
- If the polygons are very small(say one pixel large) when projected on the screen then the result is as good as any interpolative technique
- Usually used of coarse preview of scenes
- obviously inaccurate for smooth surfaces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



Flat Shading

- 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

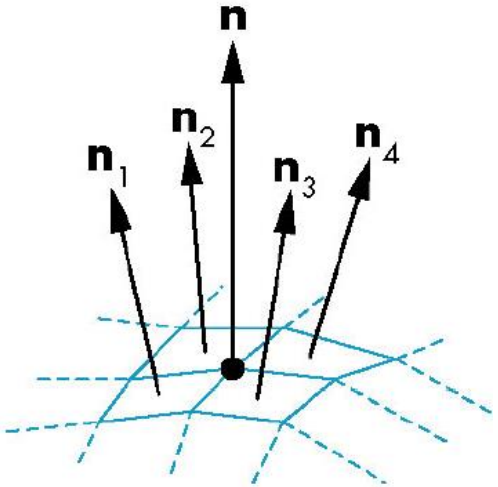




Gouraud Shading

- It is an interpolative shading method, also called **intensity interpolation shading** or **color interpolation shading**. Proposed by Gouraud in 1971.
- 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

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



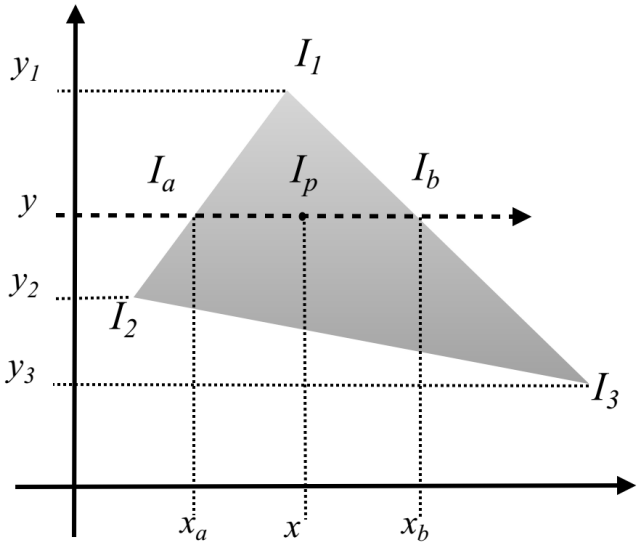
Gouraud Shading

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

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

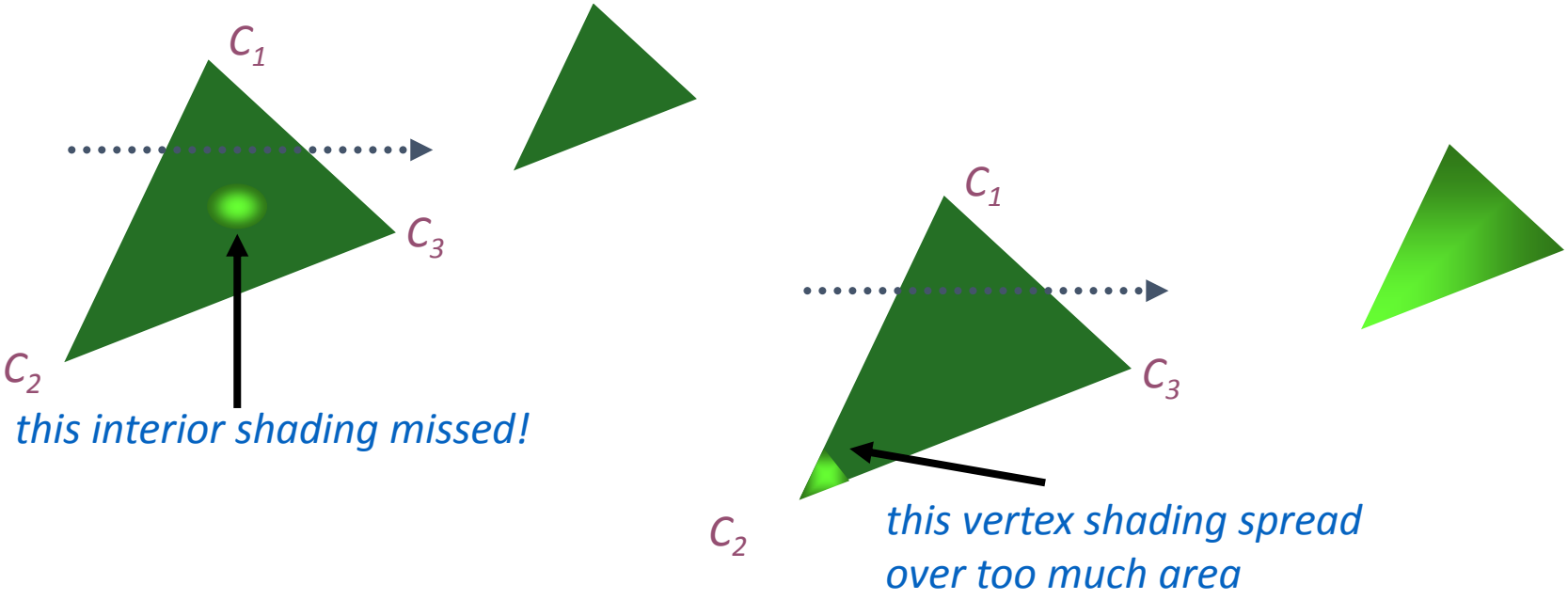
$$I_b = I_1 + (I_3 - I_1)\frac{y - y_1}{y_3 - y_1}$$

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



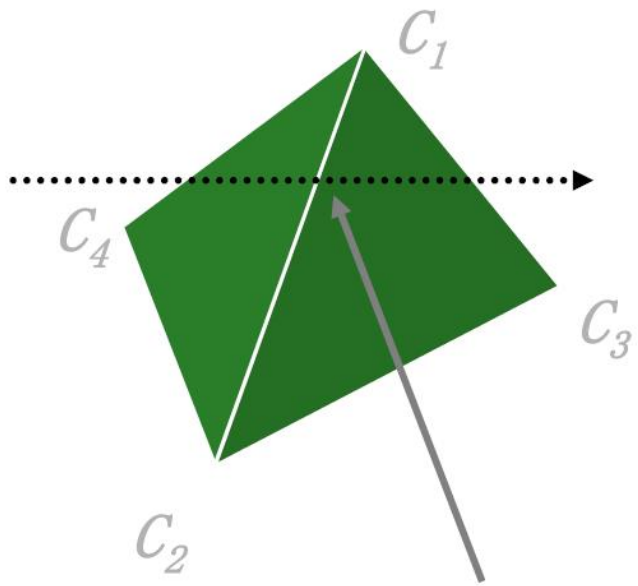
Gouraud Shading Artifacts

- 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

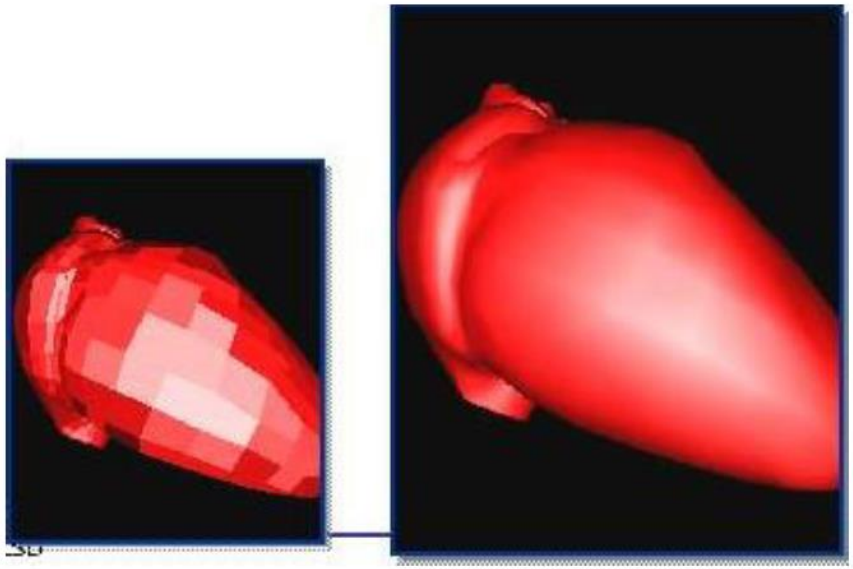


Gouraud Shading Artifacts

- **Mach bands(马赫带效应)**

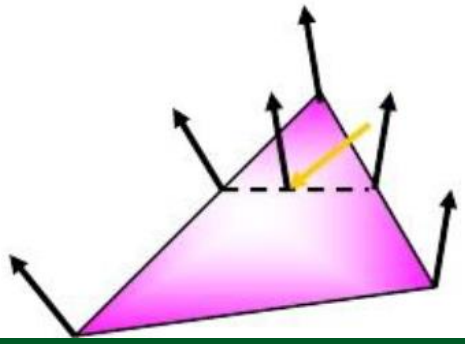


Discontinuity in rate of color change occurs here



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



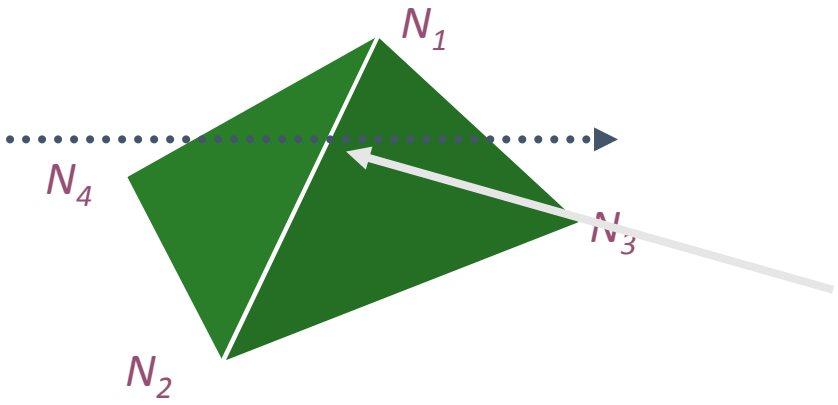
Phong Shading

- 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

$$I_{total} = k_a I_{ambient} + \sum_{i=1}^{\#\,lights} I_i \left(k_d (\mathbf{n} \cdot \mathbf{l_i}) + k_s (\mathbf{v} \cdot \mathbf{r_i})^{n_{shiny}} \right)$$

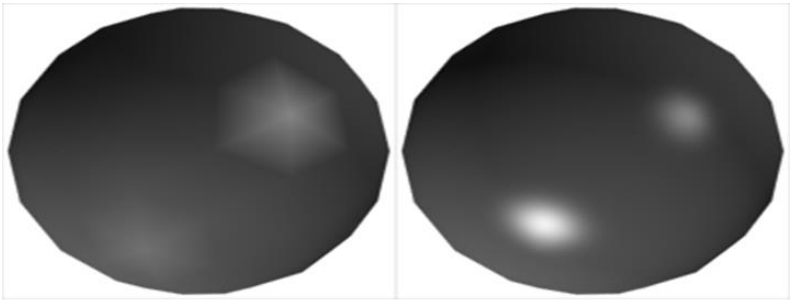


remember: normals used in diffuse and specular terms

discontinuity in normal's rate of change harder to detect

Phong Shading Difficulties

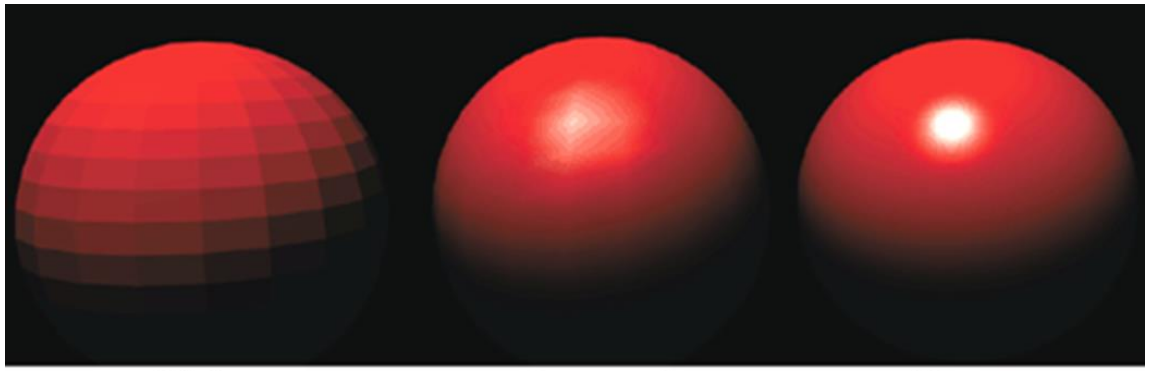
- computationally expensive
  - per-pixel vector normalization and lighting computation!
  - floating point operations required
- polygonal silhouettes remain
  - The silhouette edge of the mesh is always a polygon
  - Solution :
    - finer subdivision for the entire surface
    - finer subdivision only along silhouette (view dependent)



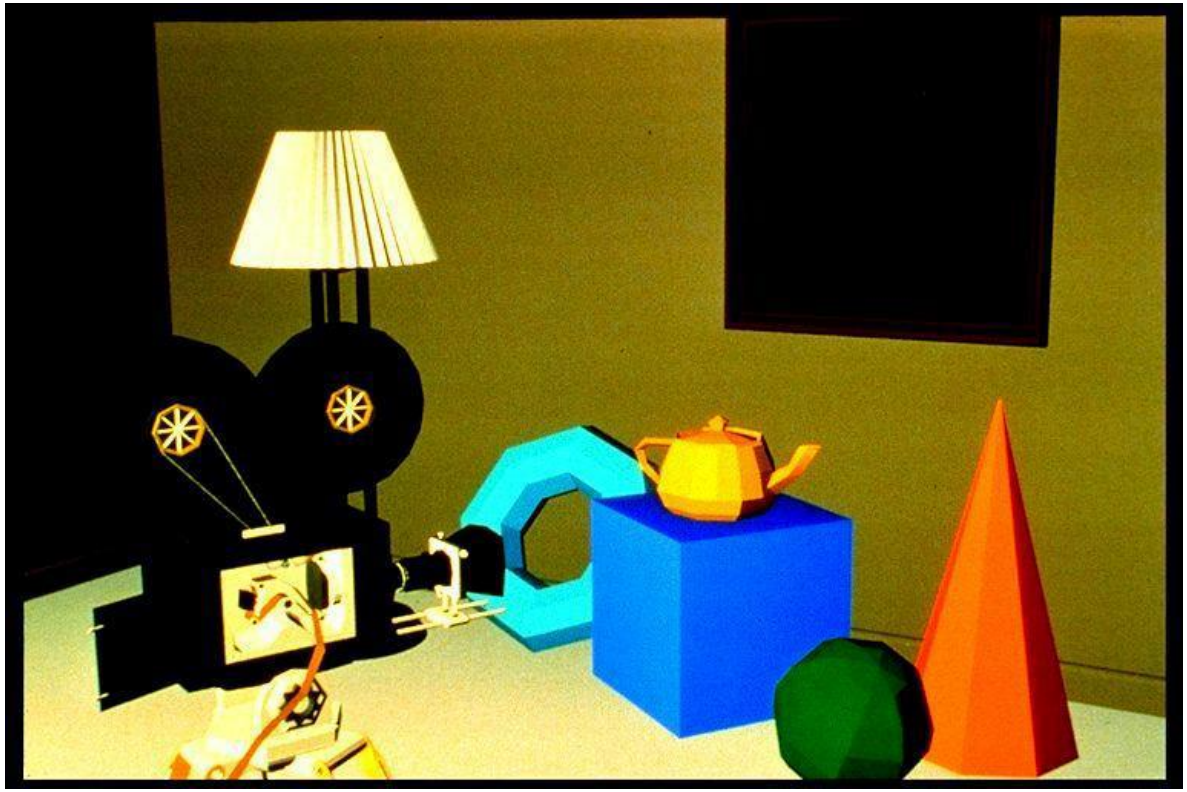
Gouraud *Phong*

Shading Models Summary

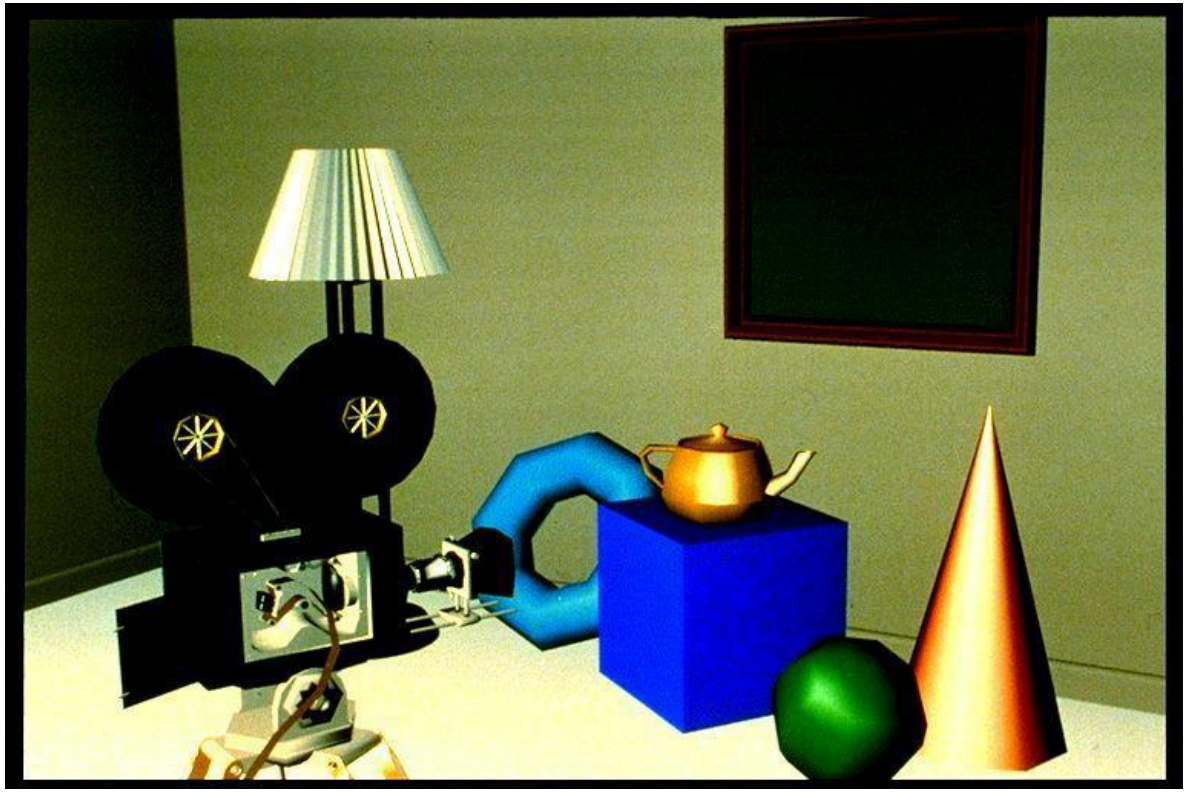
- Flat shading: per polygon
  - compute Phong lighting once for entire polygon
- Gouraud shading: per vertex
  - compute Phong lighting at the vertices and interpolate lighting values across polygon
- Phong shading: per pixel
  - compute averaged vertex normals
  - interpolate normals across polygon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



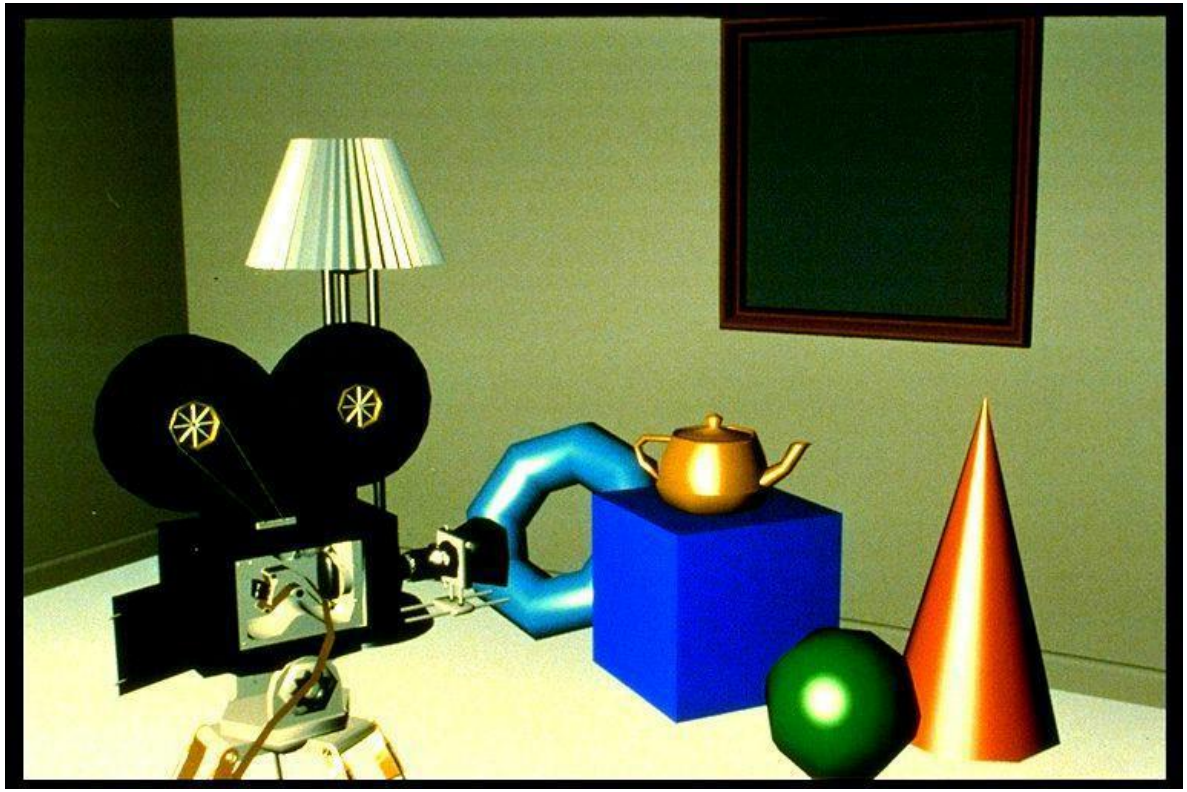
Flat Shading



Gouraud Shading



Phong Shading



OpenGL 中的实现

Colors

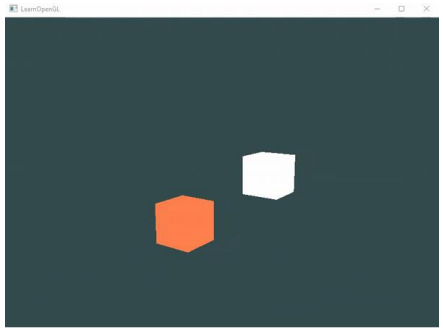
- A simple model to simulate relationship between light color and object color
- vec3 light_color(rl, gl, bl) rl, gl, bl ∈[0, 1]
- vec3 object_color(ro, go, bo) ro, go, bo ∈[0, 1]
- vec3 result = light_color · object_color

```
glm::vec3 lightColor(1.0f, 1.0f, 1.0f);
glm::vec3 toyColor(1.0f, 0.5f, 0.31f);
glm::vec3 result = lightColor * toyColor; // = (1.0f, 0.5f, 0.31f);
```

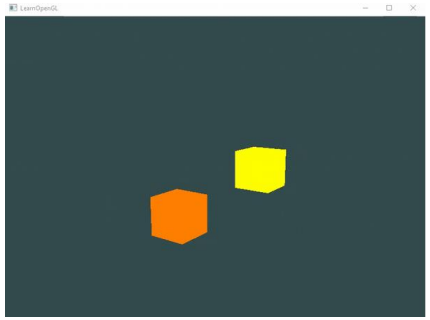
```
glm::vec3 lightColor(0.0f, 1.0f, 0.0f);
glm::vec3 toyColor(1.0f, 0.5f, 0.31f);
glm::vec3 result = lightColor * toyColor; // = (0.0f, 0.5f, 0.0f);
```

Build a lighting scene

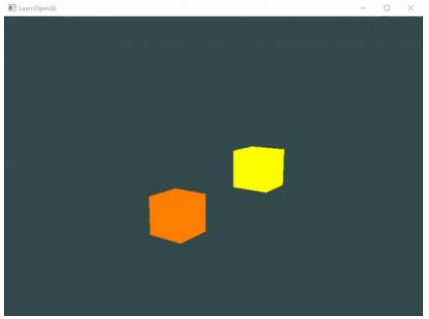
- light_color=from (1,1,1) white to (1, 1, 0) yellow
- object_colot=(1, 0.5, 0.31)
- result = from (1, 0.5, 0.31) to (1, 0.5, 0)



white light



yellow light

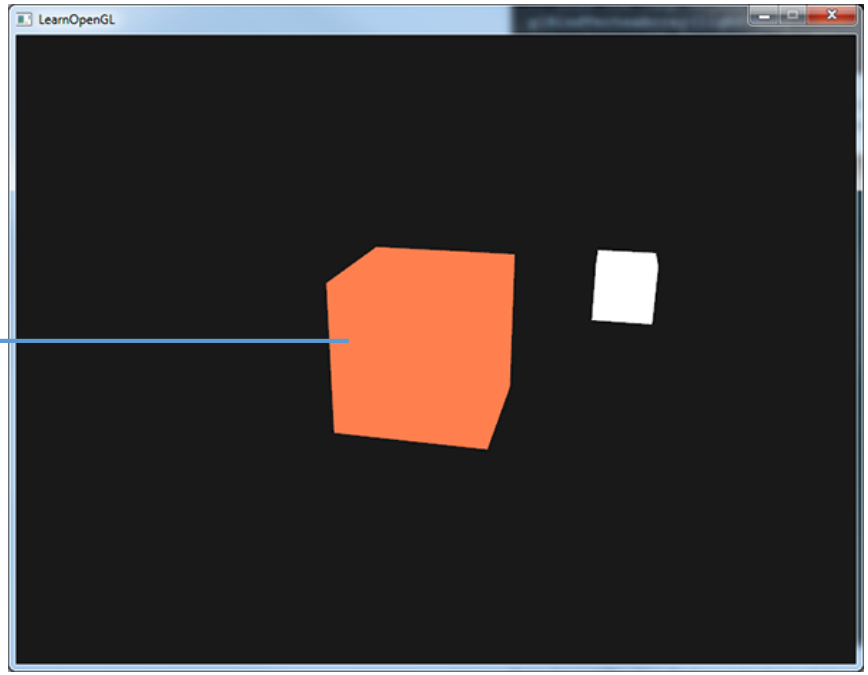


gif

Build a lighting scene

- 物体看起来不真实
  - 颜色无明暗变化，没有棱角，没有立体感
- 解决方法：使用更复杂的光照模型（环境光，漫反射，镜面反射）

一点都不“立体”



Basic lighting—Ambient lighting

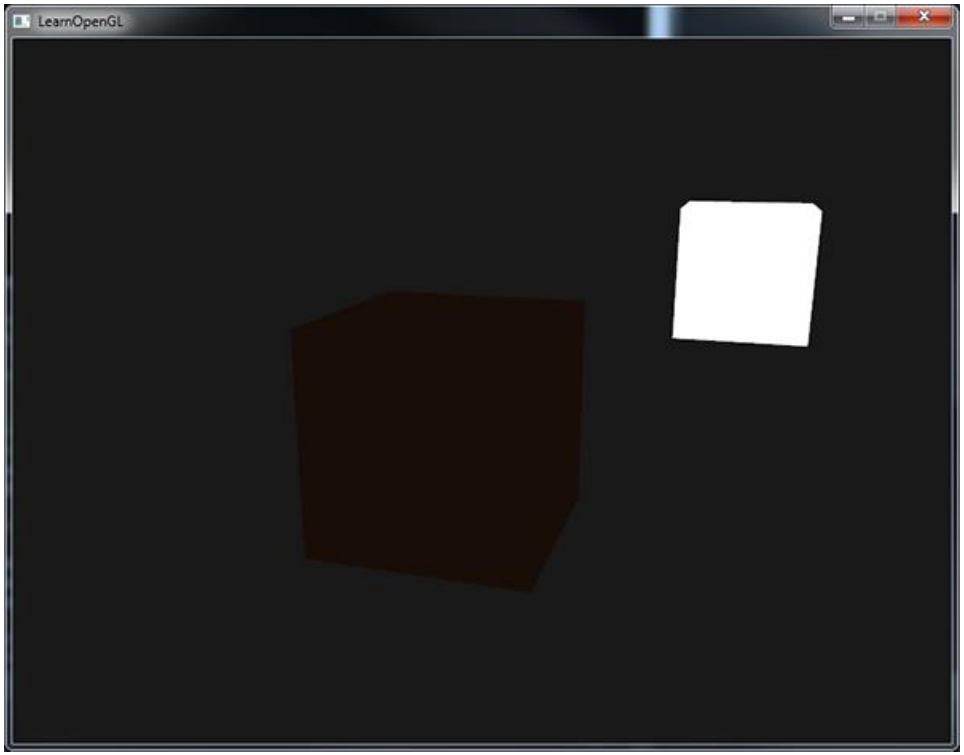
- 环境光（Ambient lighting ）是没有光源、没有方向并且对场景中的所有物体产生相同的点亮效果的一种微弱的光。
- 简单模拟光在物体之间的复杂传递，造成所有物体都有一定的点亮效果。
- First use only Ambient lighting
- Ambient lighting is week. We can set it 10% of original light
- in fragment shader：

```
void main()
{
    float ambientStrength = 0.1;
    vec3 ambient = ambientStrength * lightColor;

    vec3 result = ambient * objectColor;
    FragColor = vec4(result, 1.0);
}
```

齐次坐标

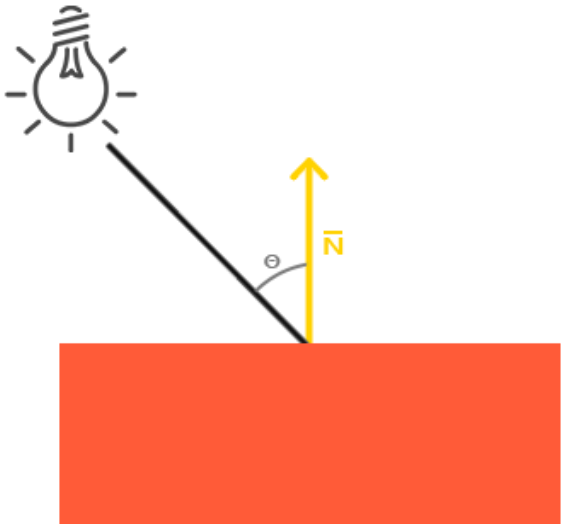
Basic lighting—Ambient lighting

- Result: Using only Ambient lighting



Basic lighting—Diffuse lighting

- Add Diffuse lighting to simulate this situation
- what do we need to calculate diffuse lighting?
  - Normal vector
  - The directed light ray （ light position – fragment position ）
- diffuse lighting = max(cosΘ, 0) * original lighting



Basic lighting—Diffuse lighting

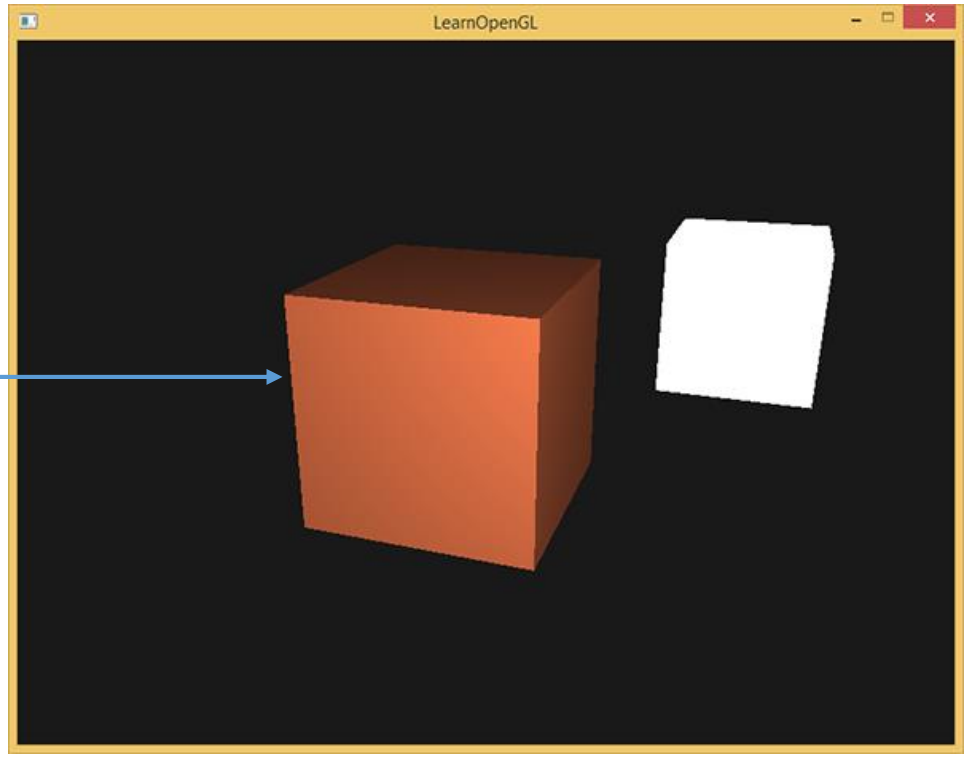
- Calculating the diffuse color
- In fragment shader：

```
#version 330 core
out vec4 FragColor;

in vec3 Normal;
in vec3 FragPos;

uniform vec3 lightPos;
uniform vec3 lightColor;
uniform vec3 objectColor;

void main()
{
    // ambient
    float ambientStrength = 0.1;
    vec3 ambient = ambientStrength * lightColor;

    // diffuse
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(lightPos - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * lightColor;

    vec3 result = (ambient + diffuse) * objectColor;
    FragColor = vec4(result, 1.0);
}
```

Basic lighting—Diffuse lighting

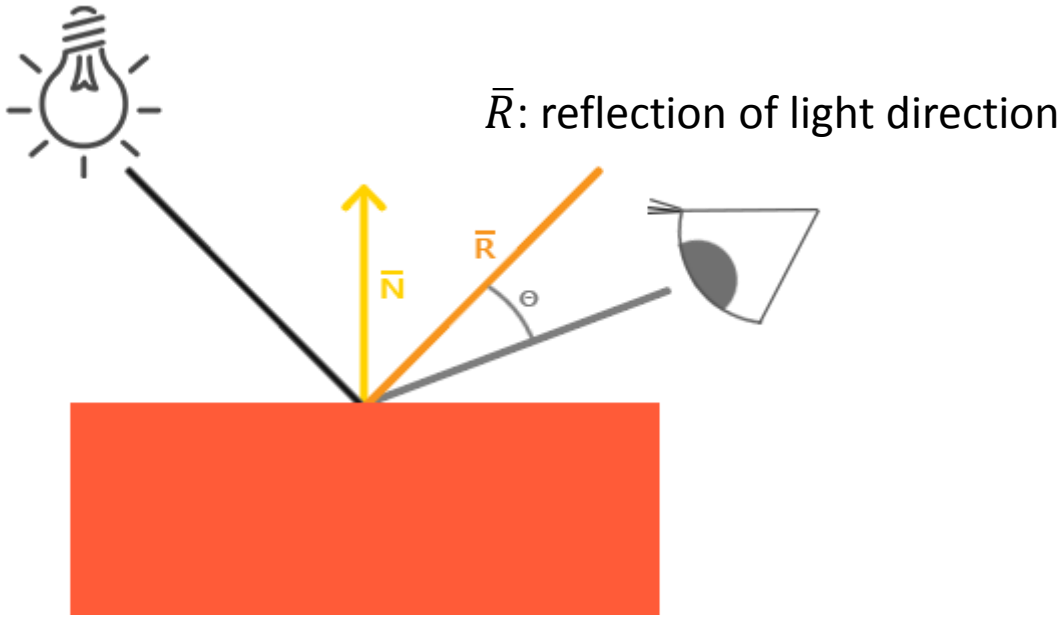
- Ambient lighting and Diffuse lighting
- 漫反射是表现真实感中很重要的一部分

真实感强了很多



Basic lighting—Specular lighting

- 越靠近光源的面应该更加亮，更加接近光的颜色，会出现亮点。视角不同，效果也会不一样。
- Add Specular lighting to simulate this situation
- specular lighting = strength * pow(max(cosΘ, 0), shininess) * original lighting

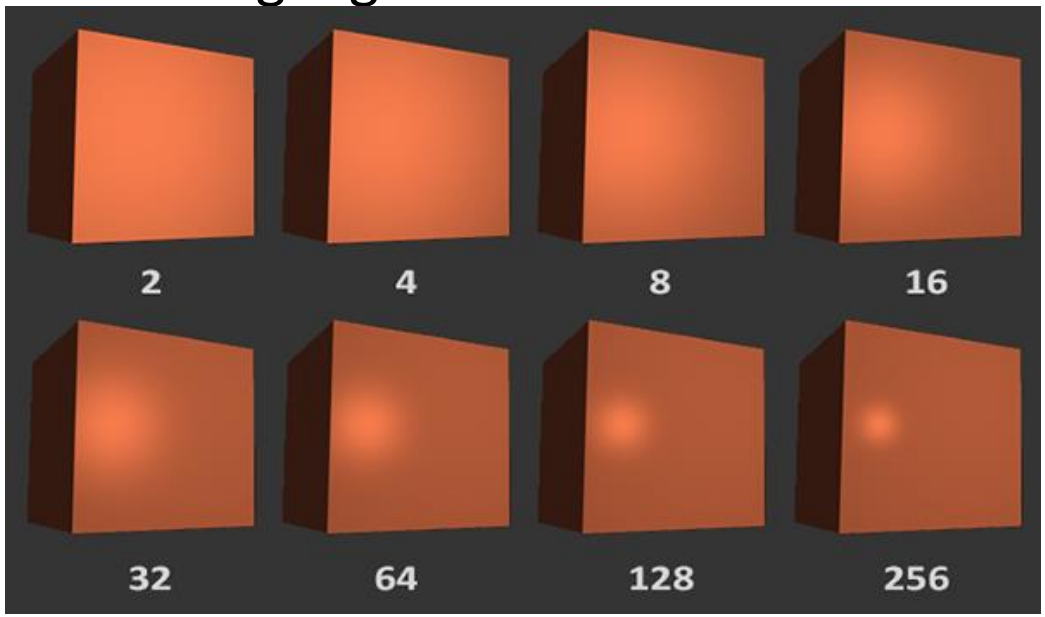


Basic lighting—Specular lighting

- Calculating the specular color in OpenGL

```
// specular
float specularStrength = 0.5;
vec3 viewDir = normalize(viewPos - FragPos);
vec3 reflectDir = reflect(-lightDir, norm);
float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
vec3 specular = specularStrength * spec * lightColor;
```

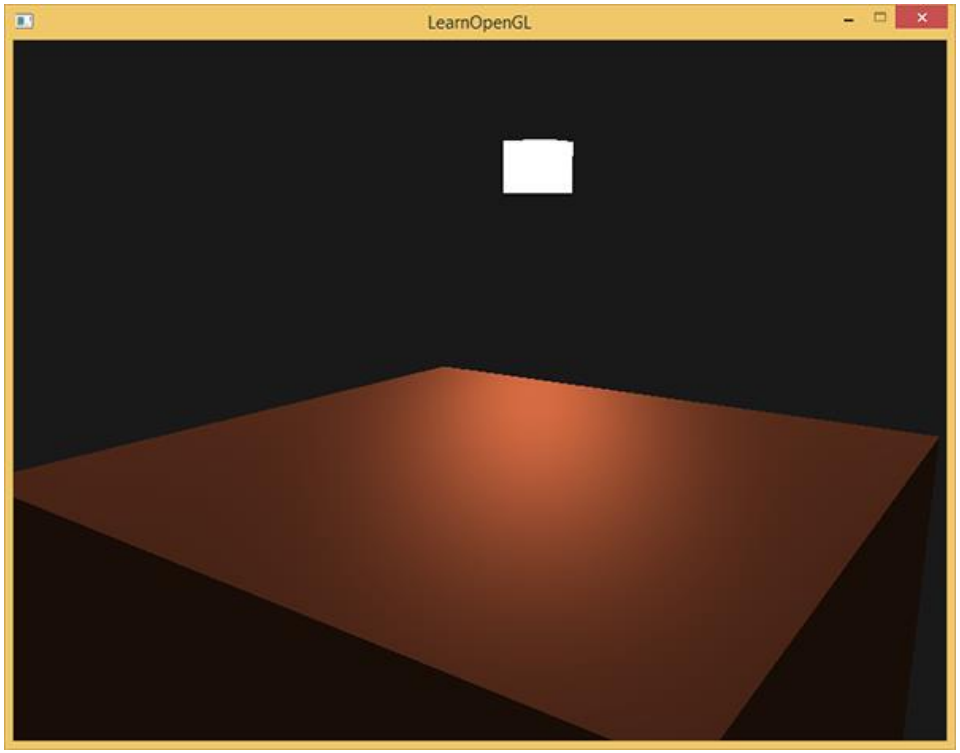
- about shininess of highlight



Basic lighting

- Finally, combine ambient, diffuse and specular lighting

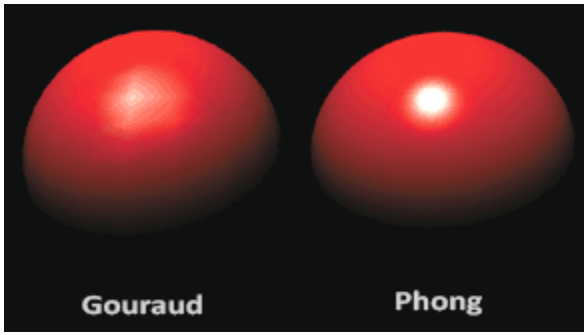
```
vec3 result = (ambient + diffuse + specular) * objectColor;
FragColor = vec4(result, 1.0);
```



Gouraud shading and Phong shading in OpenGL

- **Phong lighting model** is implemented in the **vertex shader** it is called **Gouraud shading**

- **Phong lighting model** is implemented in the **fragment shader** it is called **Phong shading**



Advanced Rendering

Global Illumination Models

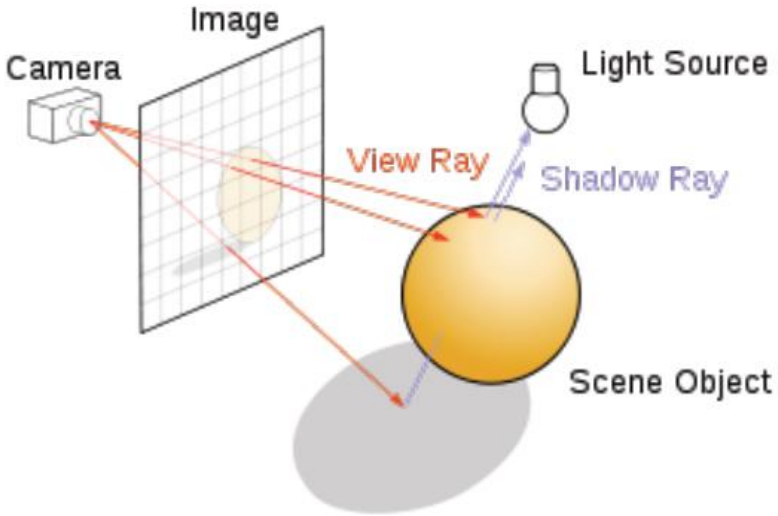
- Simple lighting/shading methods simulate local illumination models
  - No object-object interaction
- global illumination models
  - More realism, more computation
- Approaches
  - Ray tracing (光线追踪)
  - Radiosity (辐射度)
  - Photon mapping (光子映射)
  - Subsurface scattering (次表面散射)

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



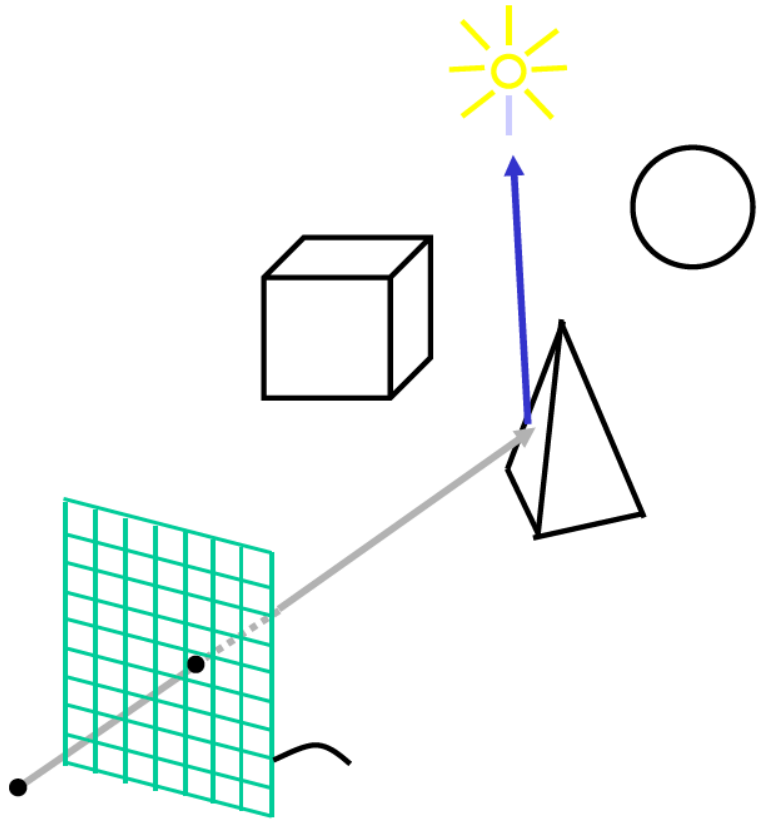
Ray Tracing

Ray Tracing


Canon
AE-1
PROGRAM
No. 213987
f=28-80 mm
1:3.5-4.9
Nikkor-XR7
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 $\mathbf{O(x_o, y_o, z_o)}$ and *direction* $\mathbf{D(x_d, y_d, z_d)}$

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

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

where,

$$A = x_d^{\,2} + y_d^{\,2} + z_d^{\,2} = 1$$

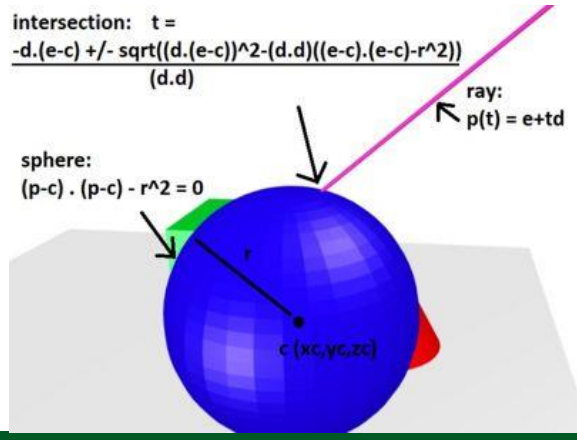
$$B = 2 \bullet \left(x_d \bullet \left(x_o - x_c \right) + y_d \bullet \left(y_o - y_c \right) + z_d \bullet \left(z_o - z_c \right) \right)$$

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

- 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

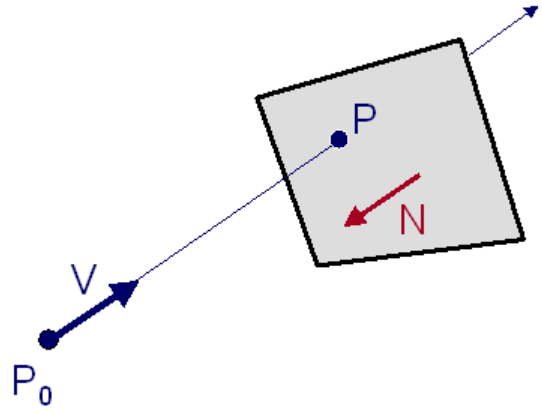
$$a \bullet x + b \bullet y + c \bullet z + d = 0$$

- Substituting the ray equation into the plane equation we have

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

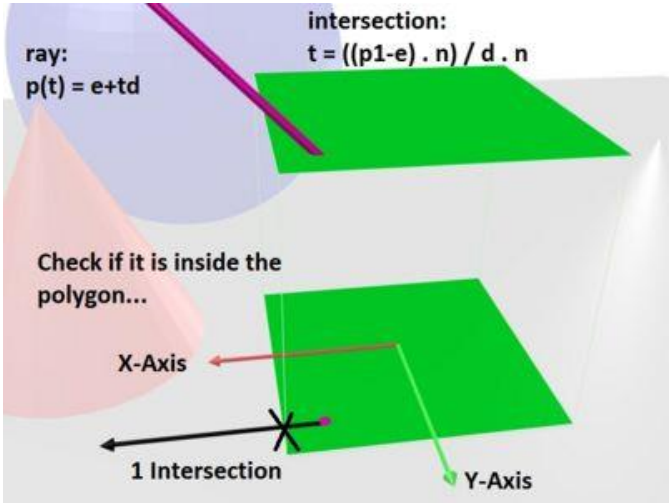
- Solving for t

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



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/