



中山大學
SUN YAT-SEN UNIVERSITY



国家超级计算广州中心
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

计算机图形学

光照与着色

陶钧

taoj23@mail.sysu.edu.cn

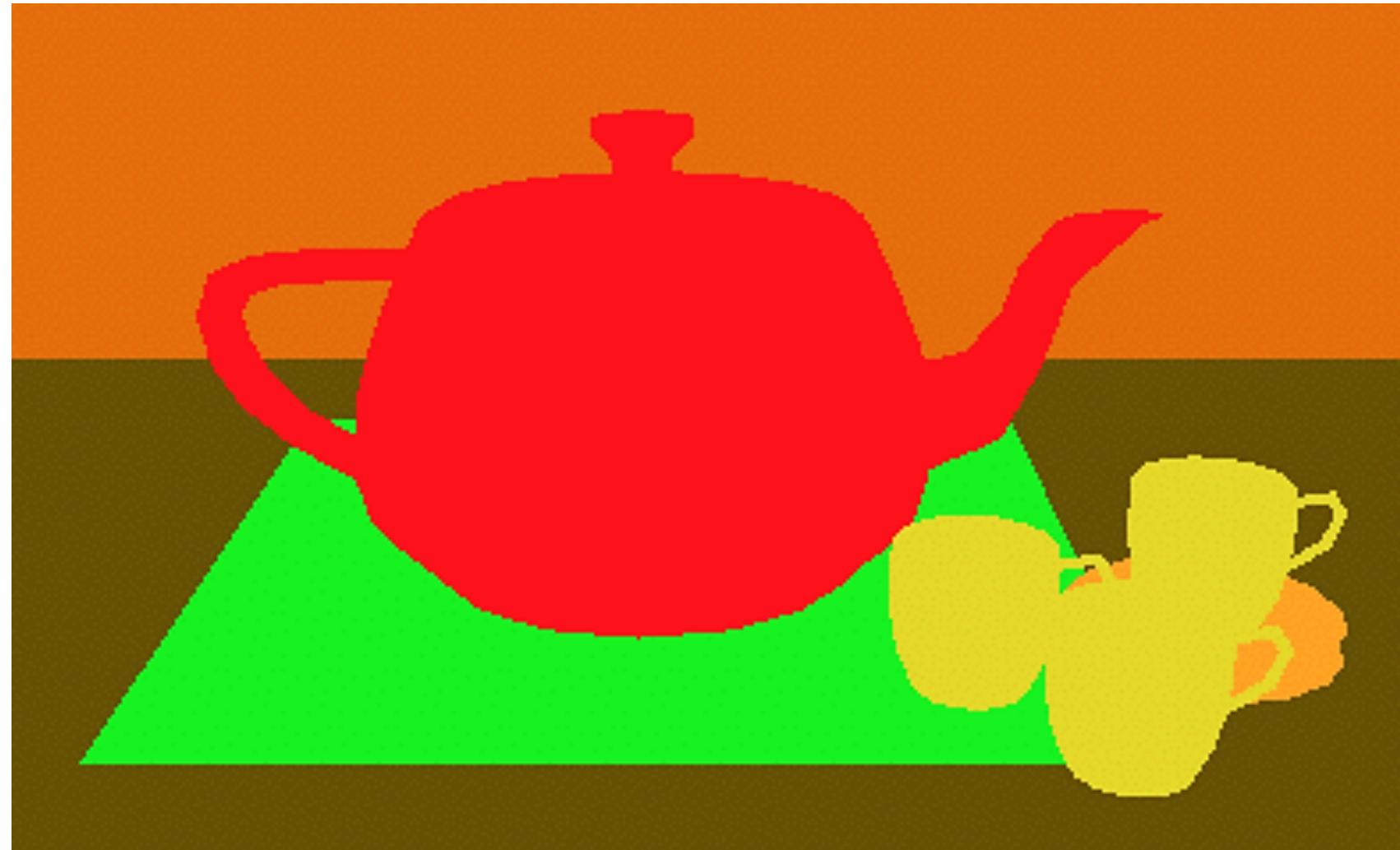
中山大学 数据科学与计算机学院
国家超级计算广州中心

- 简介
- 光源
- Phong反射模型
- OpenGL中的光照
- 其他



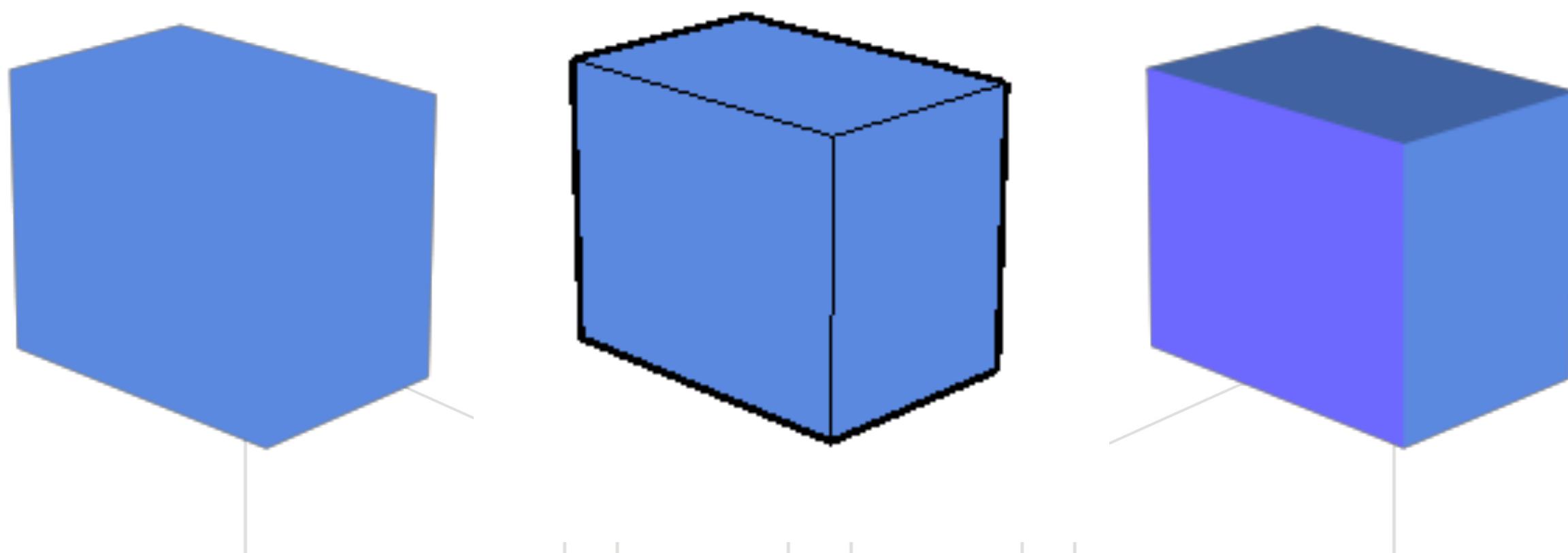
此前，我们已经学习过如何指明顶点的位置和颜色

- 然而，三维的顶点投影到屏幕上，却没有深度感
- 缺少了什么？

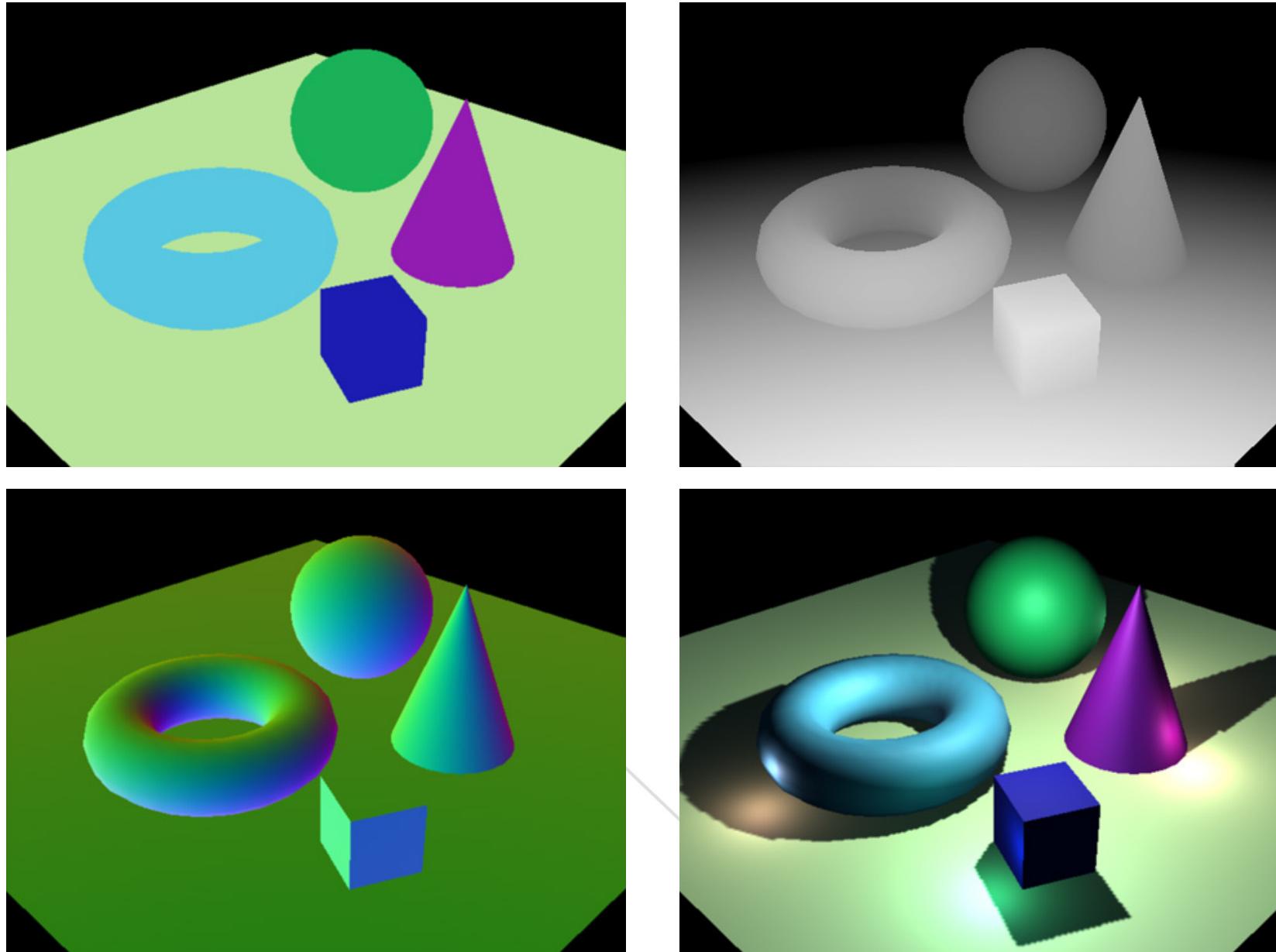


此前，我们已经学习过如何指明顶点的位置和颜色

- 当拼接在一起的多个表面具有同样颜色时，我们无法区分每一个表面，而物体的投影也失去了其三维观感
 - 加上边缘线区分表面如何？



着色 (shading) 在表现立体形状上有着重要的作用



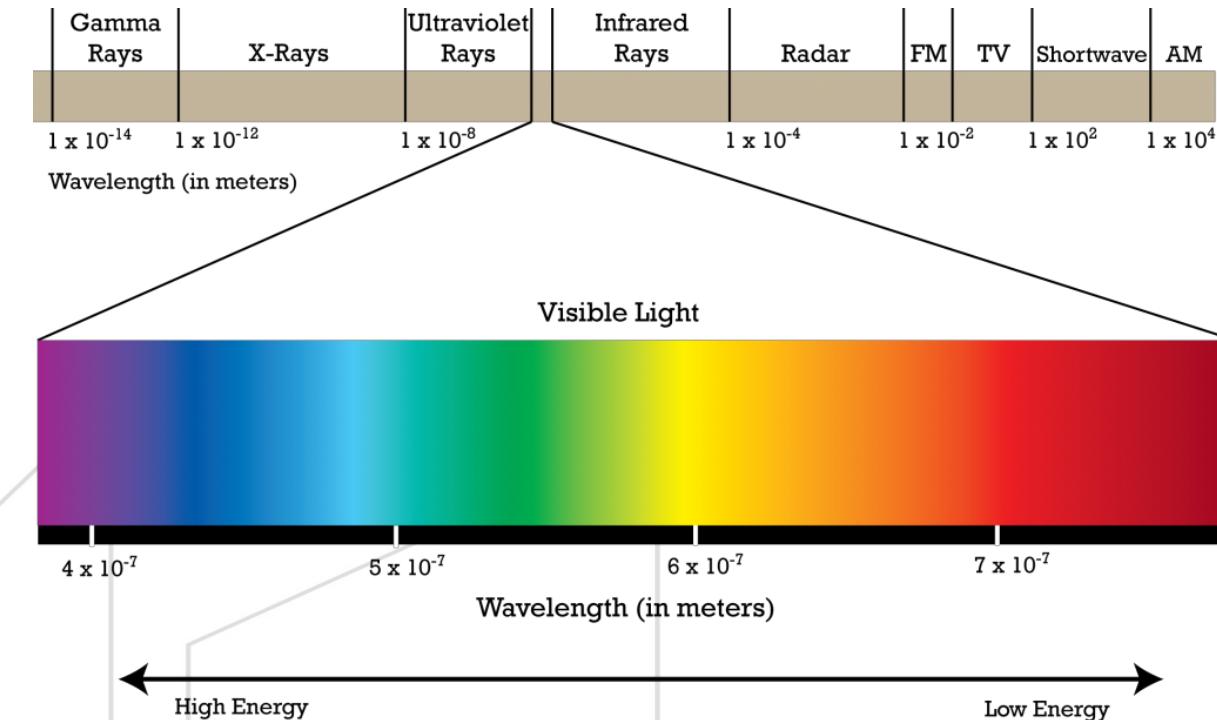
光是绚烂世界的重要组成部分

- 自然光、人造光
- 绘制绚丽的图像离不开光照



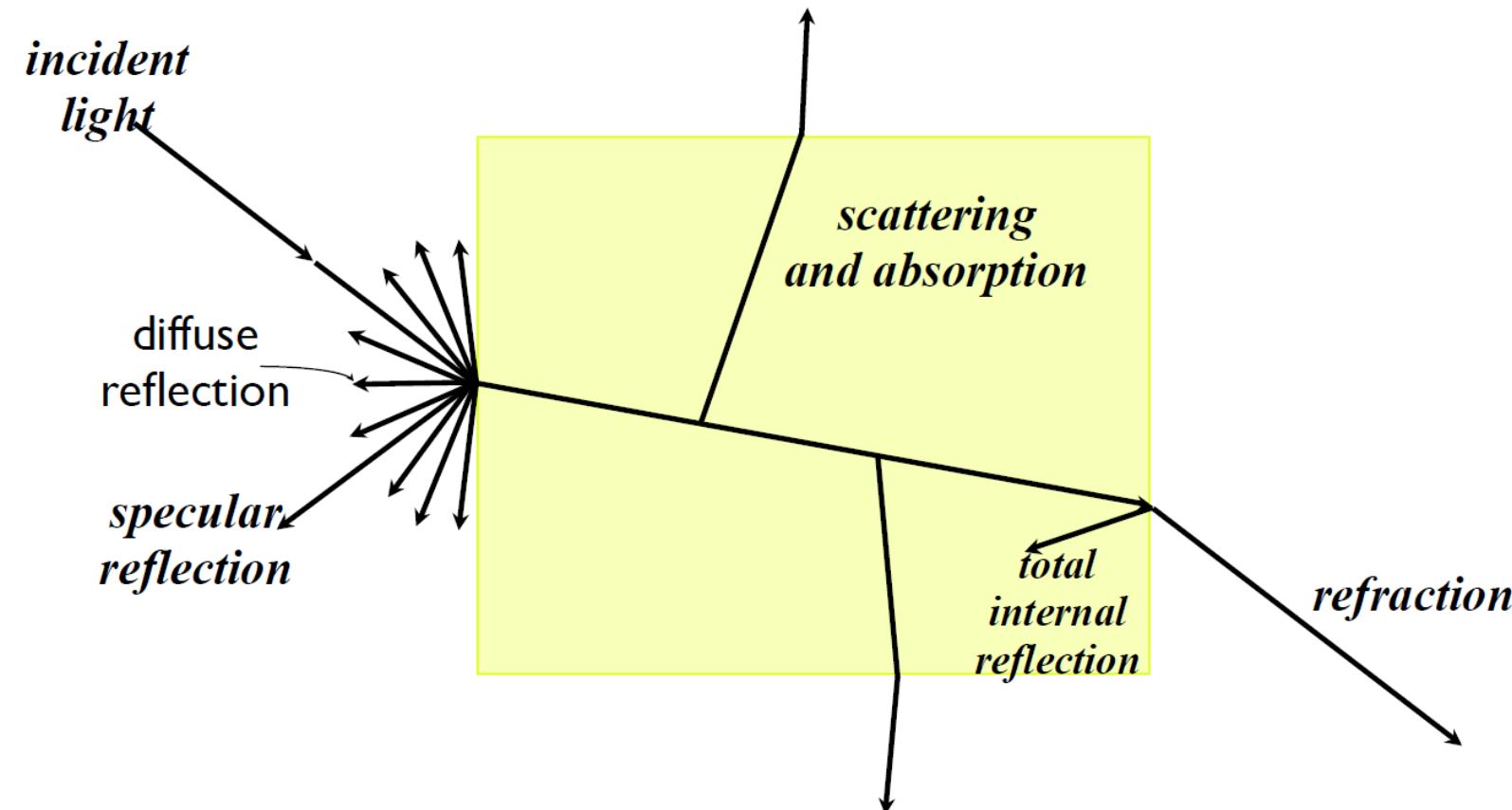
● 照明 (illumination)

- 照明是对光射在特定物体的特定点所发生现象的完整描述
- 物体上任一点的颜色由离开该点的光的属性所决定
 - 不同波长的光的强度、角度
 - 由入射光的属性及物体表面物理特性所决定
- 为了产生具有真实感的图像，我们必须理解入射光与物体表面之间的相互作用



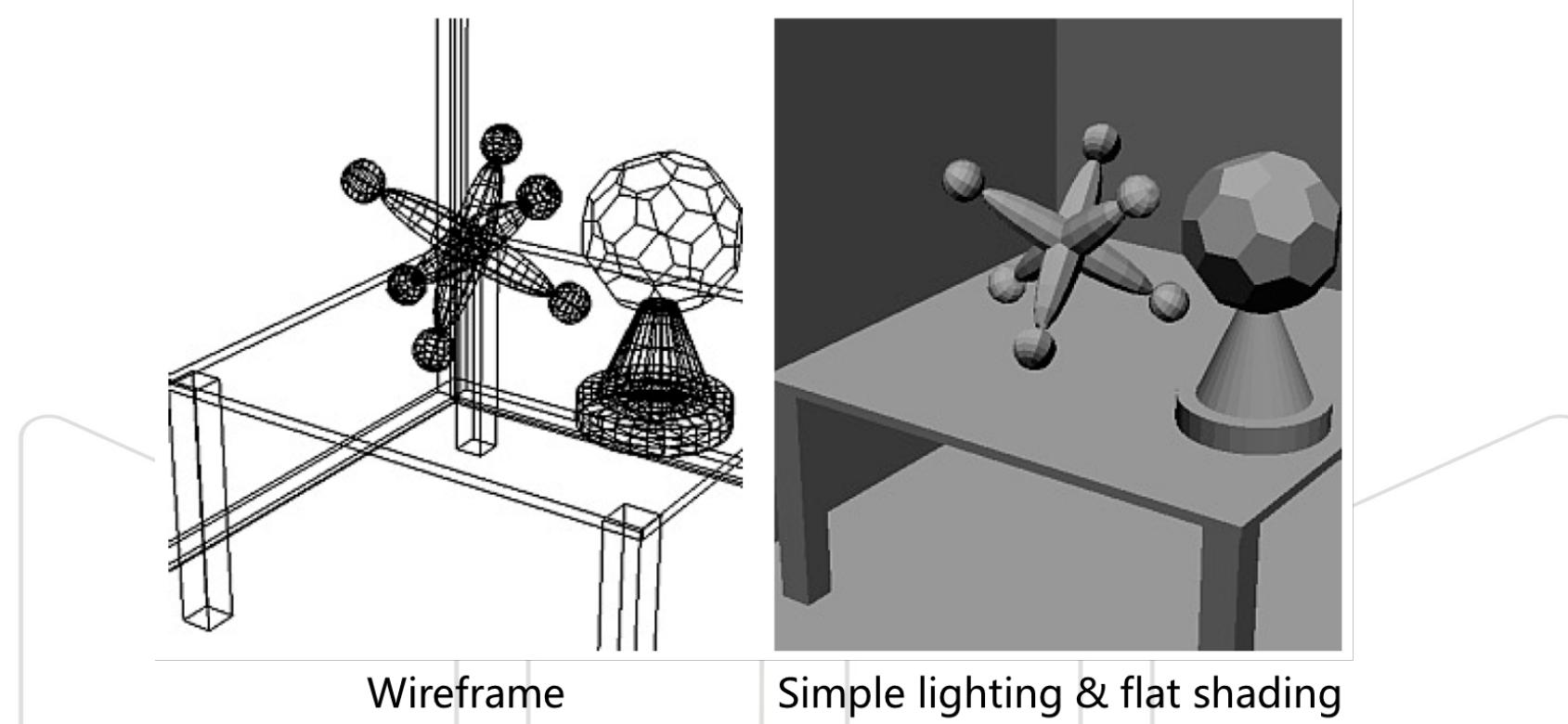
光与物体的相互作用

- 光与物体表面的相互作用
 - 反射与透射
 - 物体表面即场景中物体的几何模型（多边形网格）
- 光通过物体时产生的发散与吸收



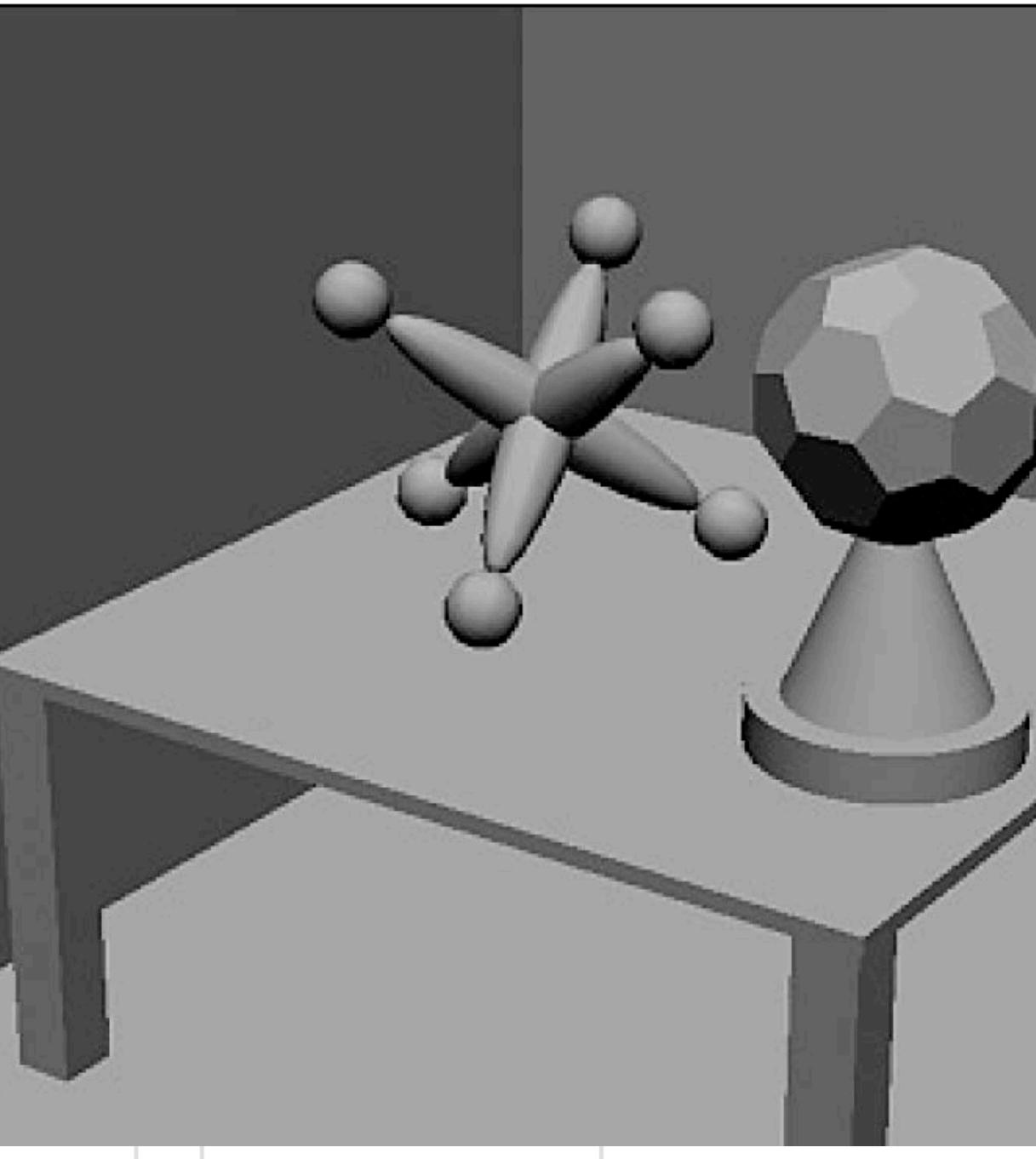
● 光照模拟

- 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
- 在计算中应充分考虑：对象表面状态，光源位置，及视点位置
- 需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



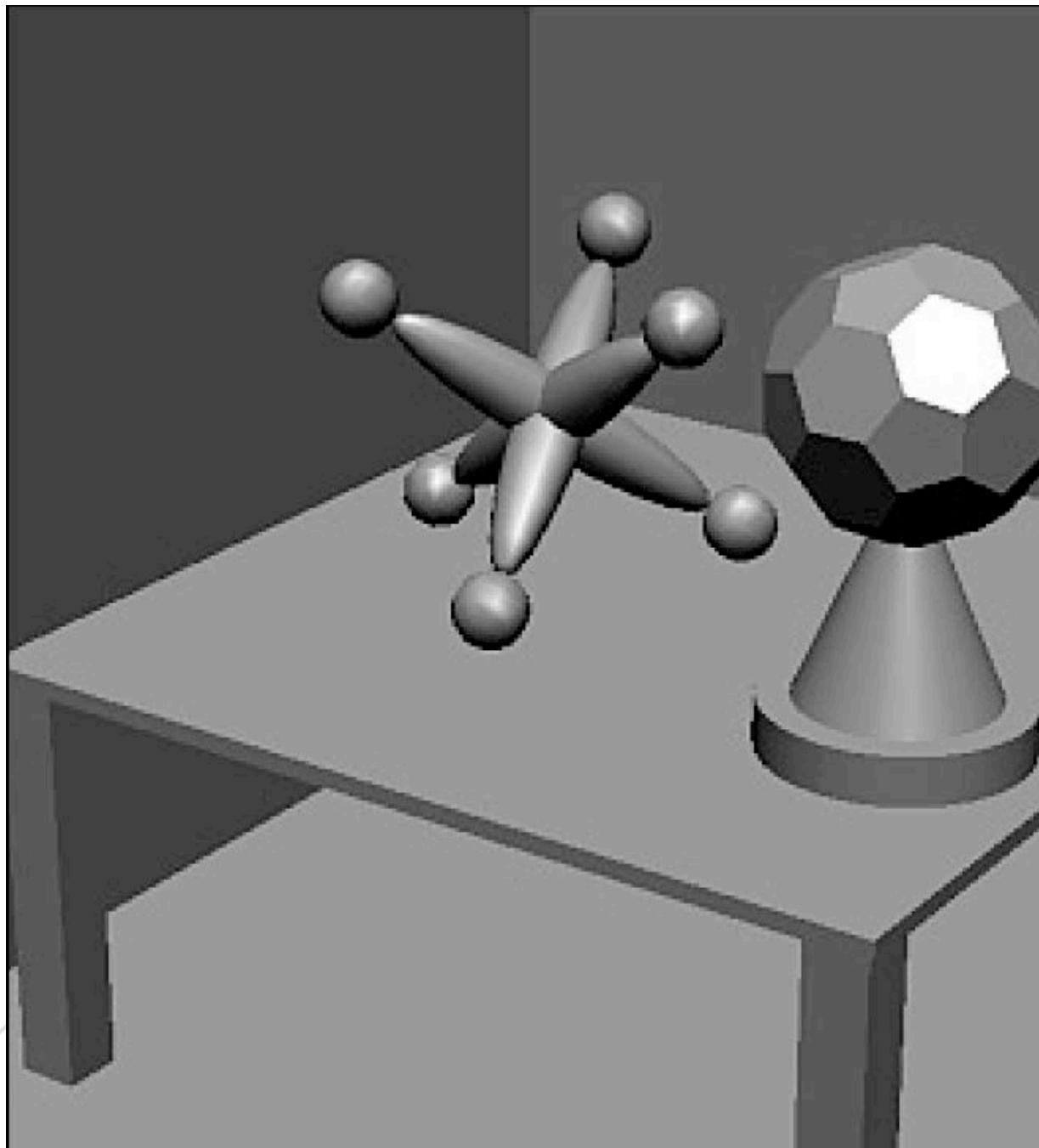
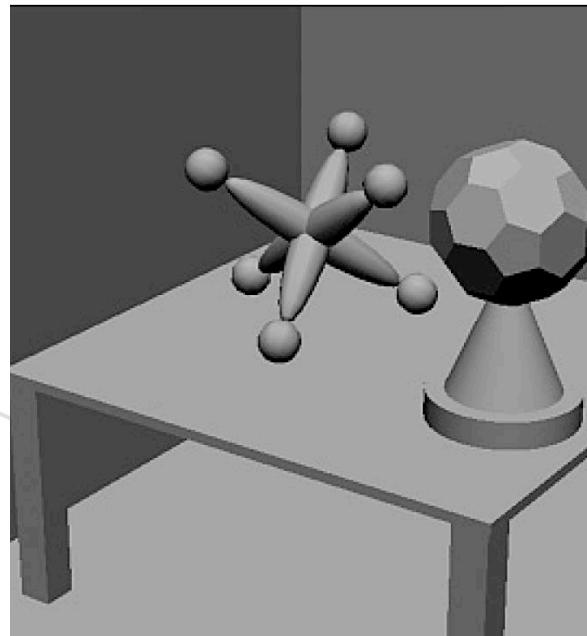
Smooth shading

- 如果多边形网格表示的对象为光滑对象，那么现实的结果也应当反映这种光滑性
- 在计算每个顶点的亮度后，用线性插值计算出内部的亮度
 - Gouraud shading



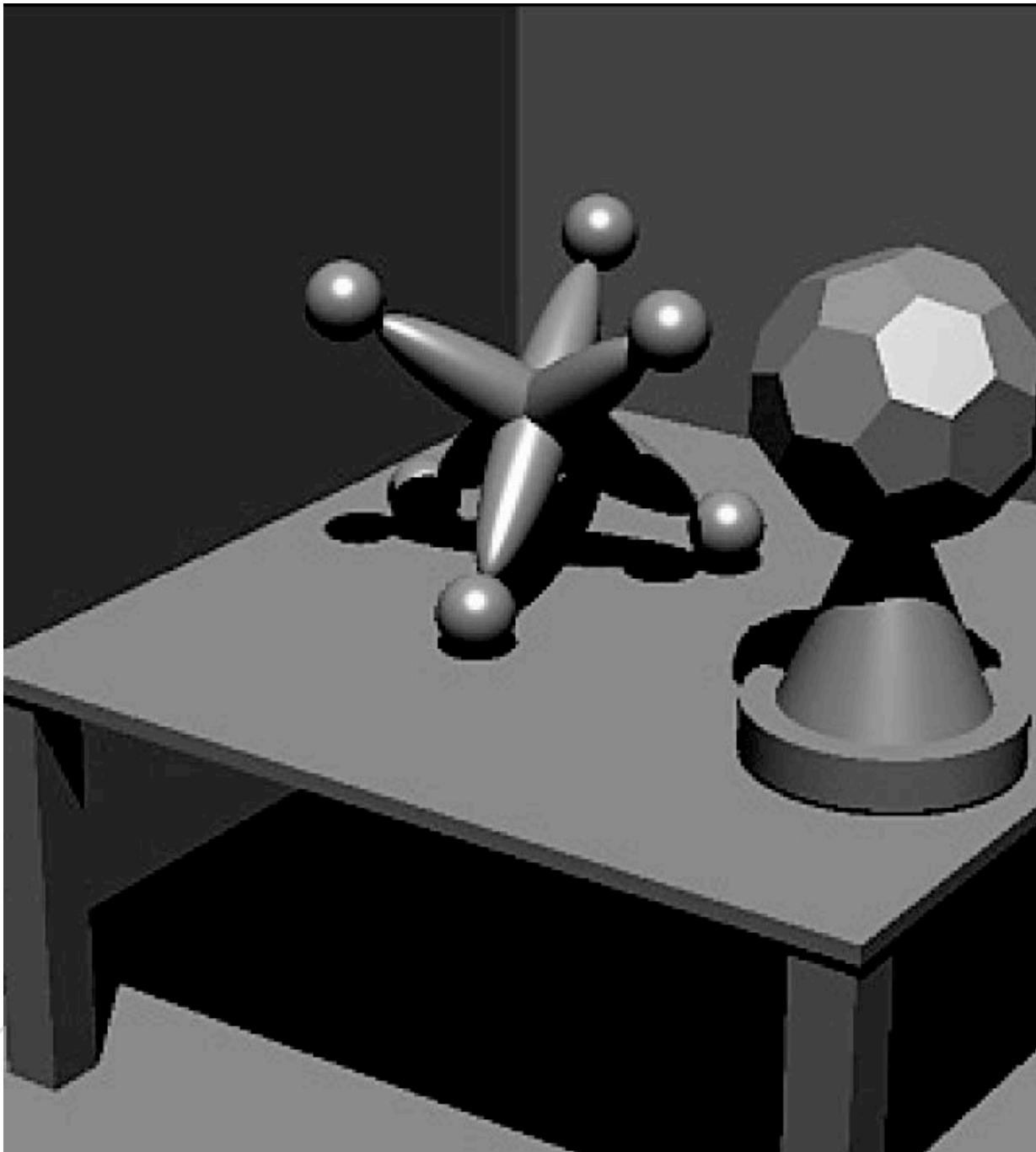
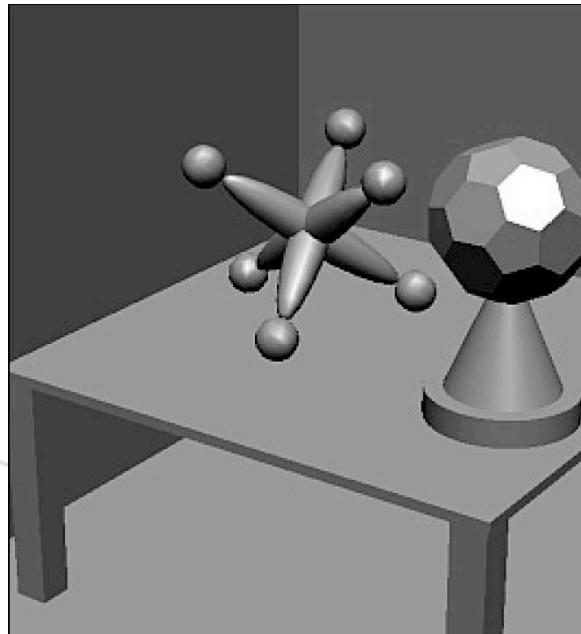
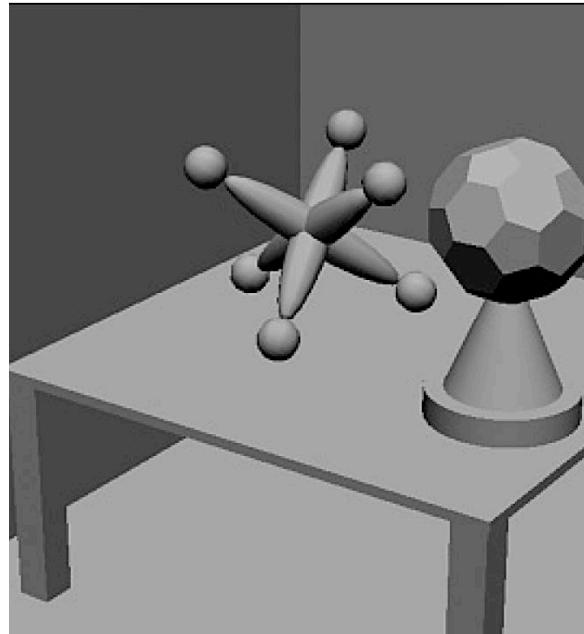
Smooth shading

- 为了得到更真实的效果，可以加入镜面光效果



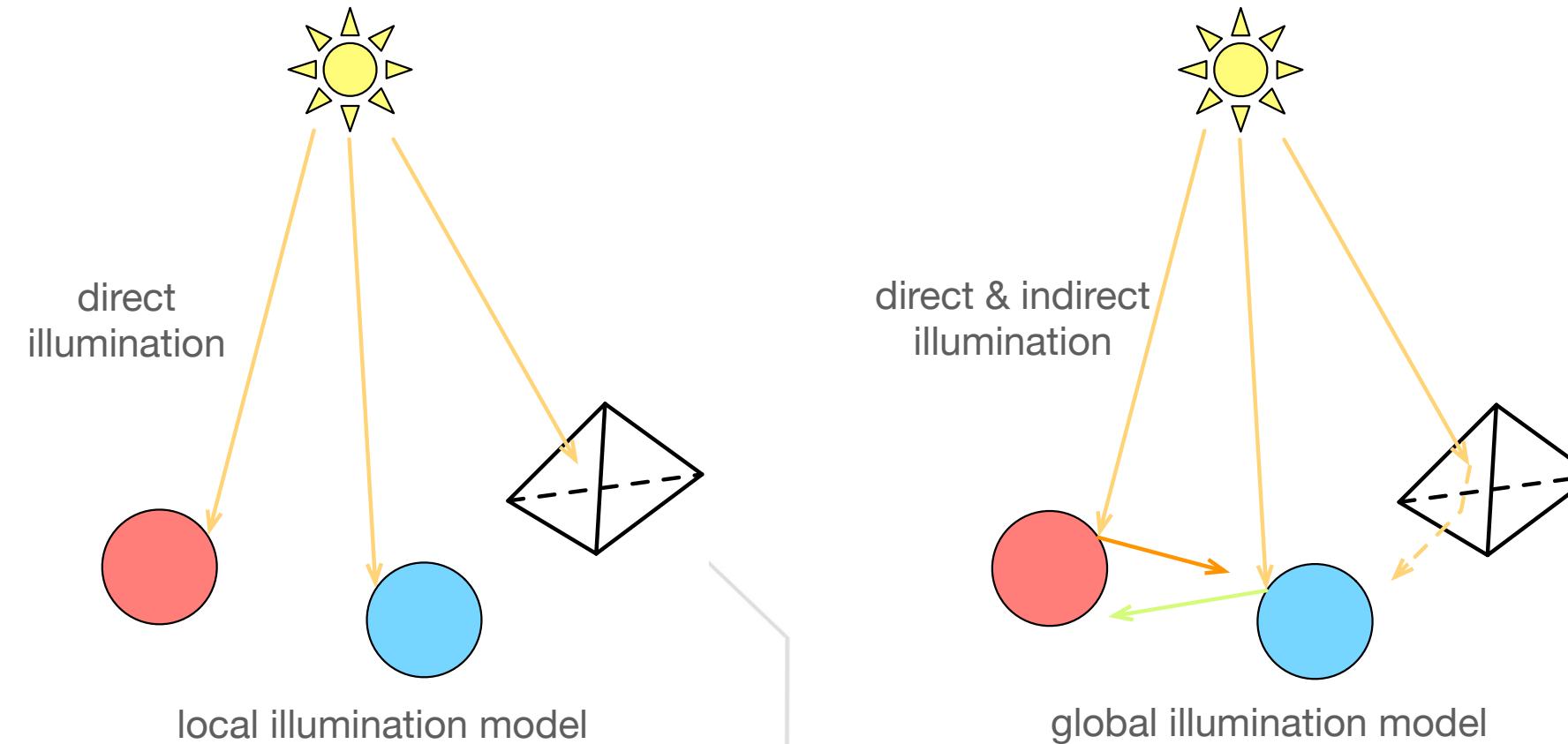
- Smooth shading

- 在加入阴影后，可以进一步提升图像的真实感



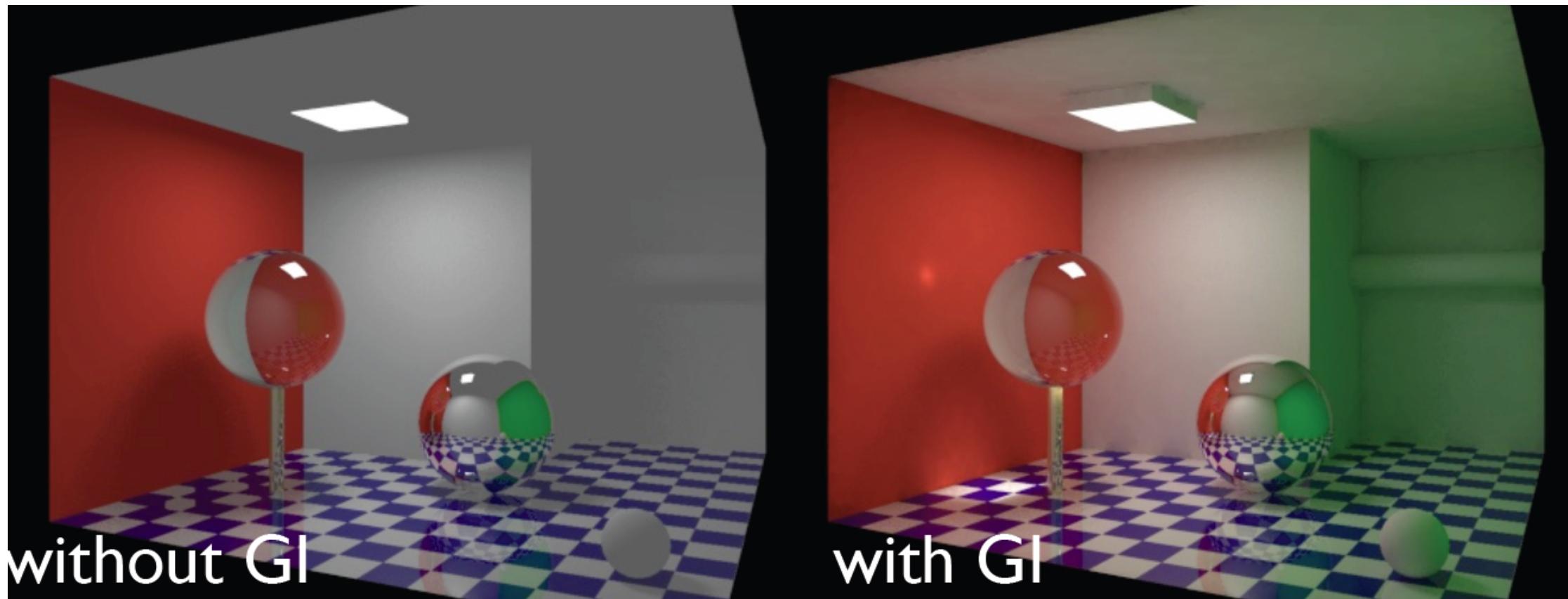
光照模型

- 精准地模拟真实的光与物体表面的相互作用复杂度极高
- 大多数计算机图形应用使用的都为近似方法
 - Lighting model或illumination model



● 光照模型

- 在计算任意点的颜色时，我们需要考虑照射该点的光
 - 局部光照模型：只考虑由光源直接发出的光
 - 全局光照模型：考虑通过别的物体/自身对光的透射和反射到达该点的光



- 简介
- 光源
- Phong反射模型
- OpenGL中的光照
- 其他

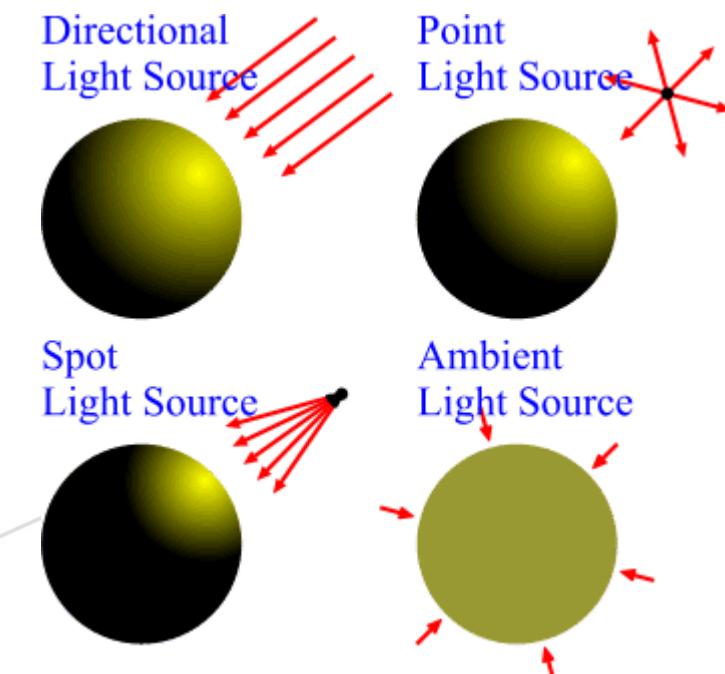


颜色

- 光源发射出不同频率的光的强度不同，其方向也可能随频率变化
 - 真实的物理模型非常复杂
- 人的视觉系统基于三原色理论
 - 在大多数应用中，可以用三种成分（红绿蓝）强度表示光源
 - 光的亮度可表示为 $I = [I_r, I_g, I_b]^T$

类型

- 环境光 (ambient light)
- 点光源 (point source)
- 聚光灯 (spotlight)
- 平行光/远距离光源 (directional light)



环境光

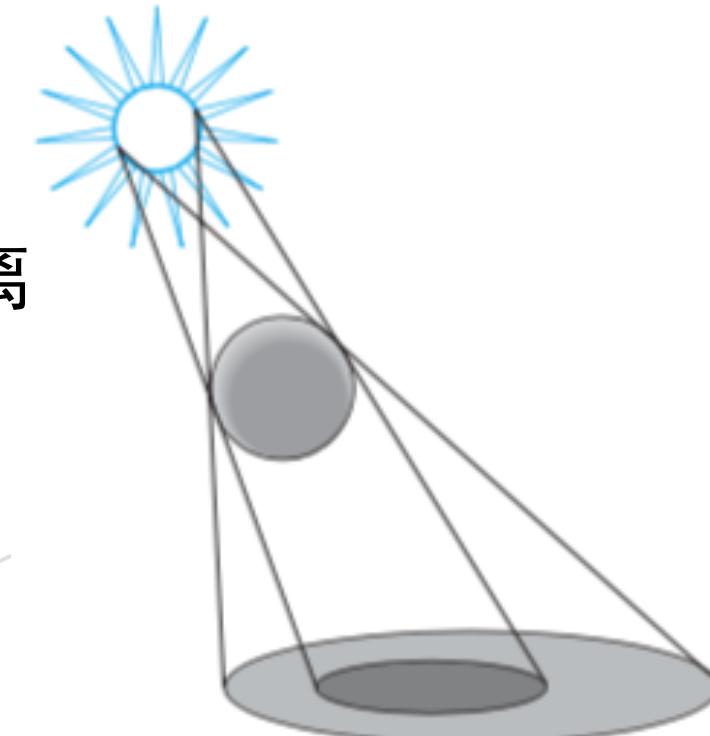
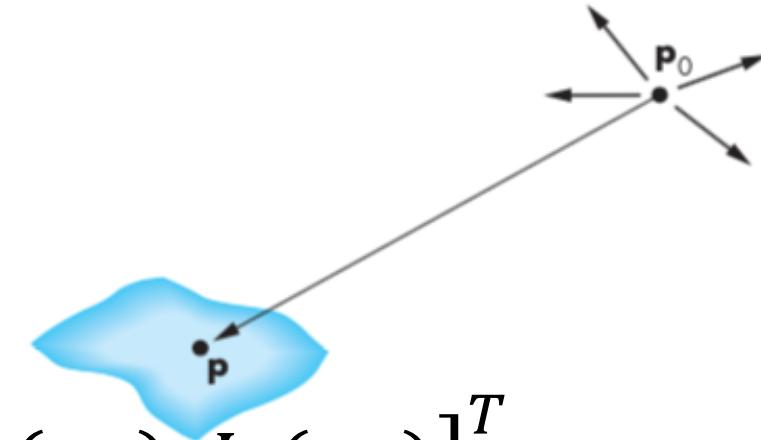
– 对整个场景的均匀照明

- 如，房间中的光源布置，使整个房间得到（尽可能）均匀的照明效果
 - 多个大尺寸光源，光源内加装散光器，使光线向各个方向散射
- 完整地模拟环境光过于复杂，在实际应用中，常假设场景中每个点获得的光照强度一致，表示为 $I_a = [I_{ar}, I_{ag}, I_{ab}]^T$
- 但由于材质属性不同，每个点可以反射出不同强度的光线，因而呈现不同颜色



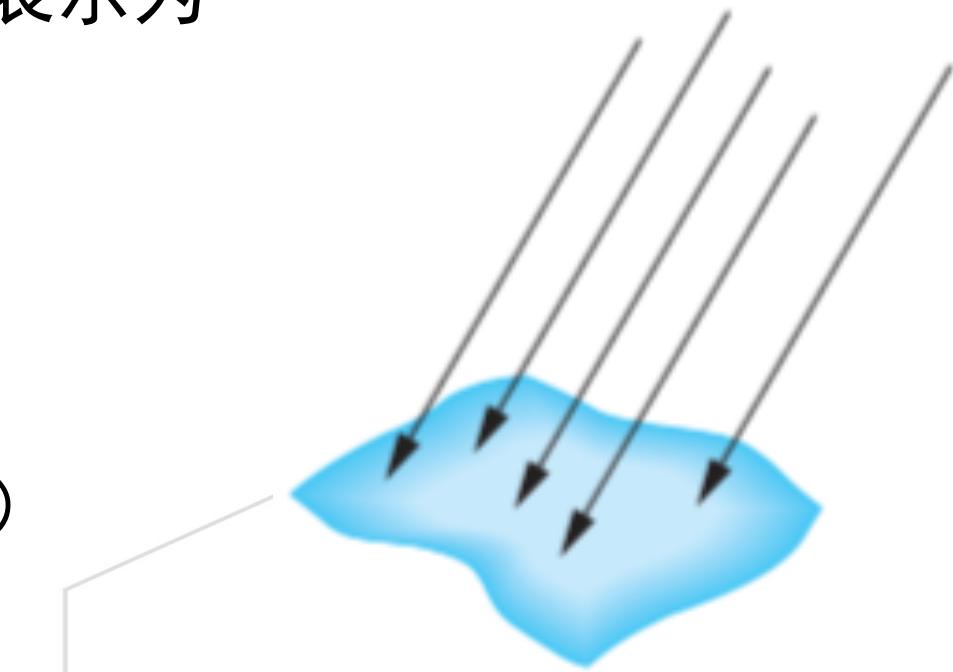
点光源

- 点光源向各个方向均匀地发射光线
 - 光线强度一致，但方向不同
- 点光源可由其颜色和位置表示： $I(p_0) = [I_r(p_0), I_g(p_0), I_b(p_0)]^T$
- 点 p 从点光源 p_0 接收到的光线强度为： $I(p, p_0) = \frac{1}{|p-p_0|^2} I(p_0)$
 - 简单，易于计算，但往往使场景明暗反差过大
 - 光线强度随距离衰减，一定程度缓解此现象
 - 计算中常用 $(a + bd + cd^2)^{-1}$ 代替平方反比距离获得柔和的照明效果
 - 真实环境中，光源具有一定尺寸，使得场景中亮度变化比较柔和
 - 本影、半影



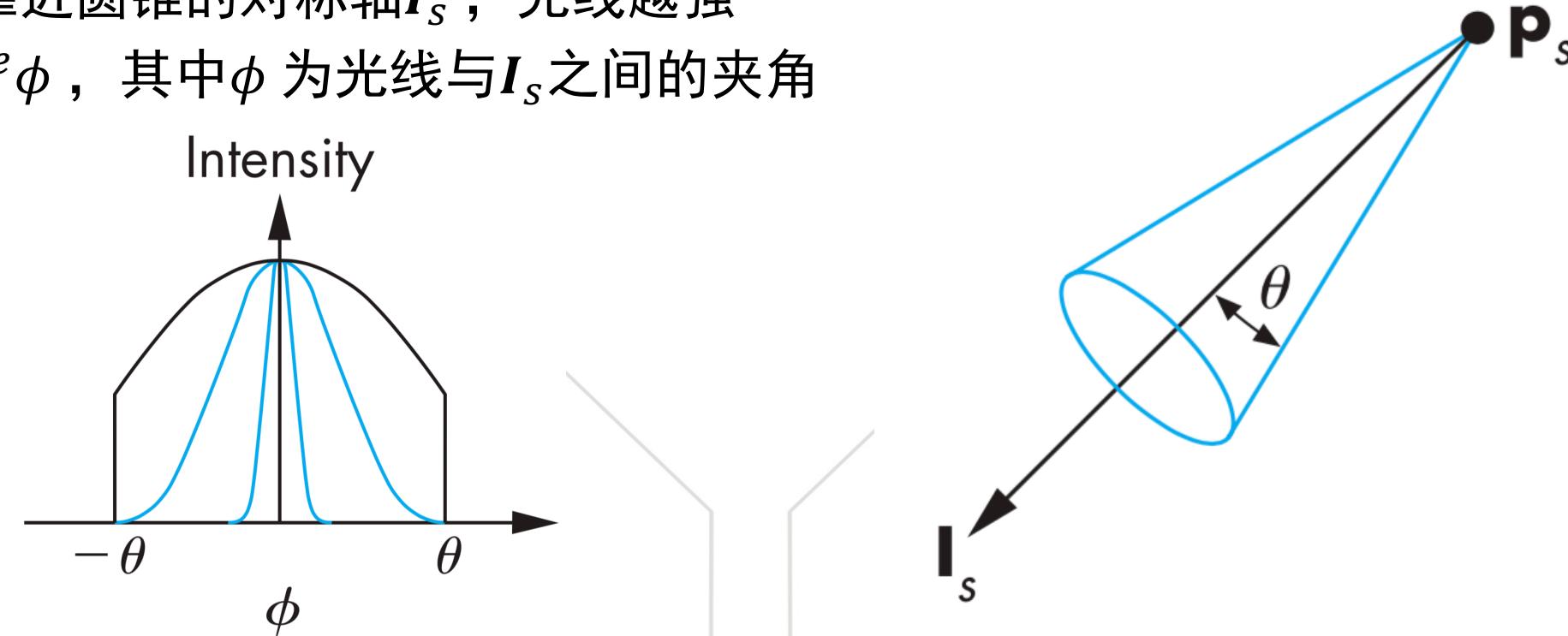
平行光源

- 光线方向一致，强度相同
 - 如，远距离光源（多远算远距离？）
- 在表示平行光源时，考虑光源方向而忽略其位置
 - 与平行投影的计算类似
 - 从齐次坐标的角度理解，位于 p_0 的点光源表示为
 - $p_0 = [x, y, z, 1]^T$
 - 而位于无穷远点 p_0 的平行光源为
 - $p_0 = [x, y, z, 0]^T$
 - 图形系统使用远距离光源进行绘制比近距离光源更有效（更接近于真实模型）
 - 现实中使用哪个模型取决于场景需要



● 聚光灯

- 聚光灯只在一个小角度范围内发射光线
- 可通过限制点光源发射光线的角度构造聚光灯模型
 - 位置 (p_s) , 朝向 (I_s) , cutoff angle (θ)
 - 常考虑发光强度在圆锥内的分布
 - 越靠近圆锥的对称轴 I_s , 光线越强
 - $\cos^e \phi$, 其中 ϕ 为光线与 I_s 之间的夹角



简介

光源

Phong反射模型

OpenGL中的光照

其他



● 材料与反射

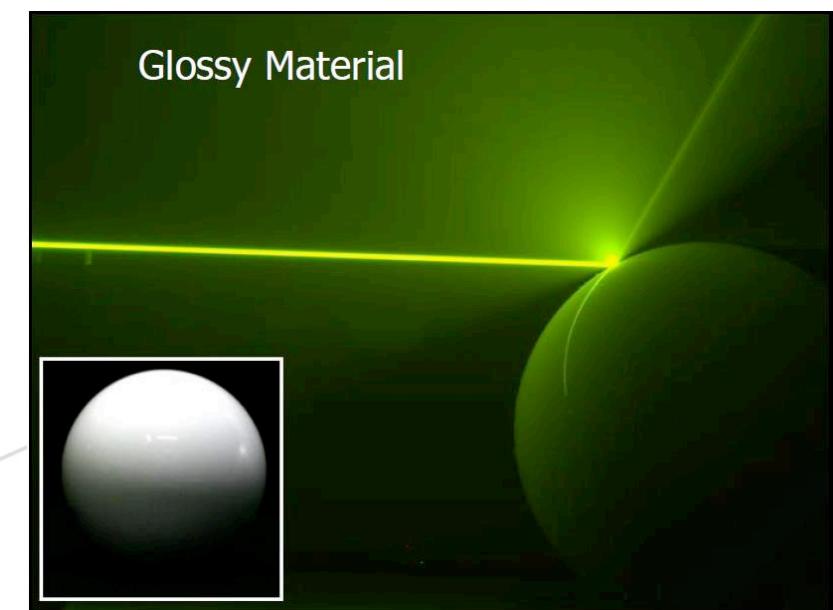
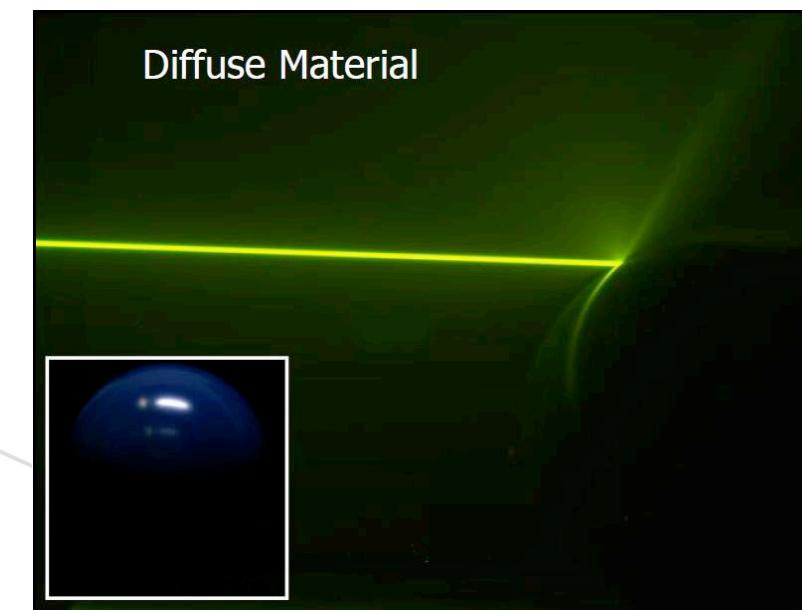
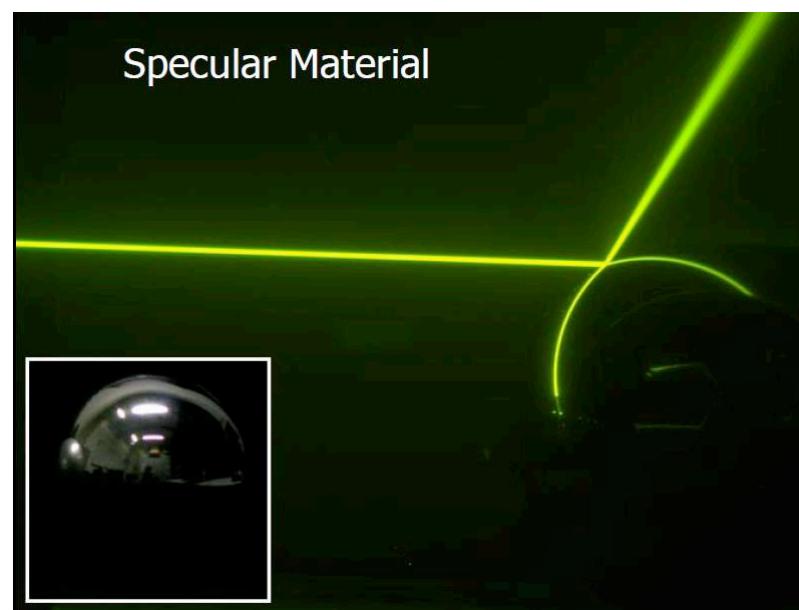
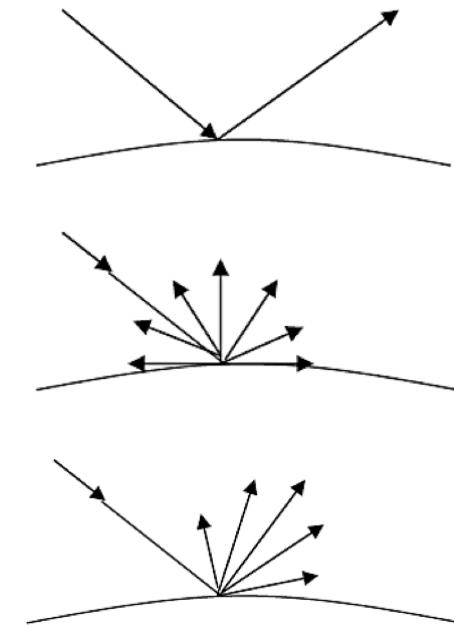
– 实际看到的颜色，不但取决于光源颜色也取决于物体自身材质

- 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



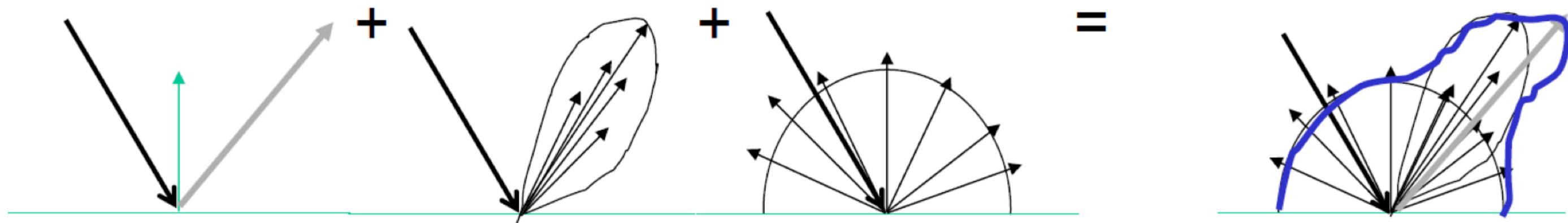
材料与反射

- 镜面反射 (specular reflection) 使入射光沿固定角度反射，而不发生散射
- 漫反射 (diffuse reflection) 把光线向所有方向均匀地散射
- 混合反射 (glossy/mixed reflection) 以上两者的合成



材料与反射

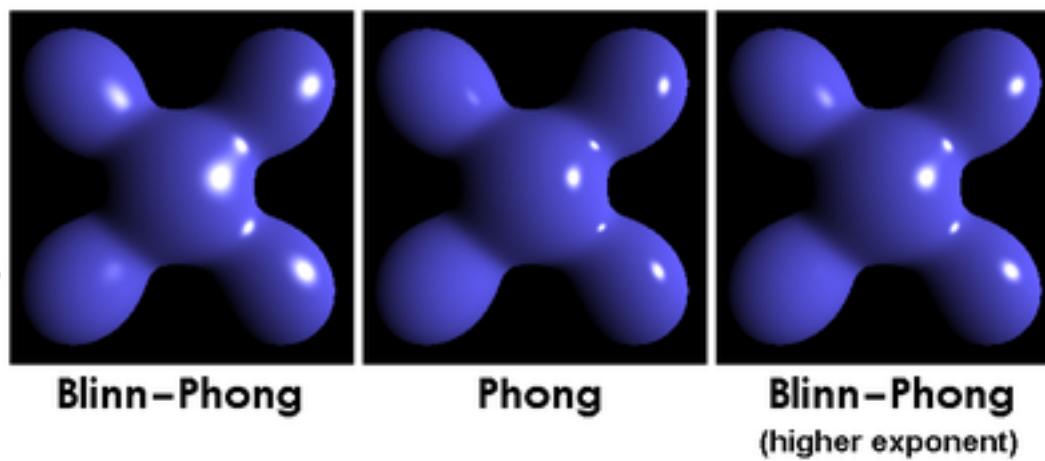
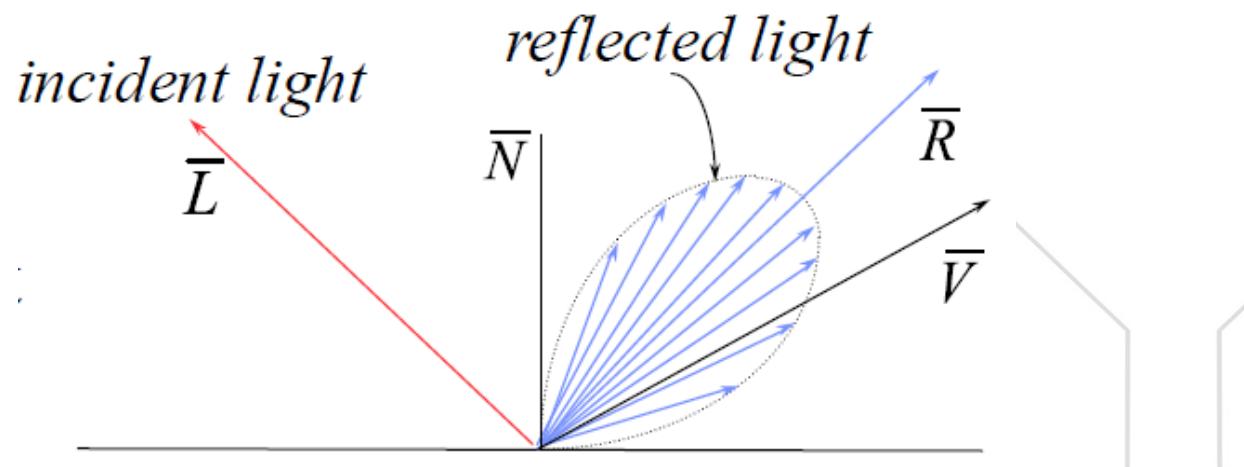
- 大多数物体表面都呈现一种复杂的反射模式
 - 随入射角与反射方向而异
 - 多种模型的混合



specular + glossy + diffuse = reflectance distribution

● Phong反射模型

- 由越南裔Bùi Tường Phong（裴祥风）在其1973年的博士论文（犹他大学）及1975年的论文中提出，并由Jim Blinn改进
 - OpenGL与Direct3D渲染管线中的默认反射模型
- 考虑三个分量
 - 环境光分量，漫反射分量，及镜面反射分量
 - 通过镜面反射更好地表示光泽表面
 - 光照计算依赖于观察方向（viewing direction）

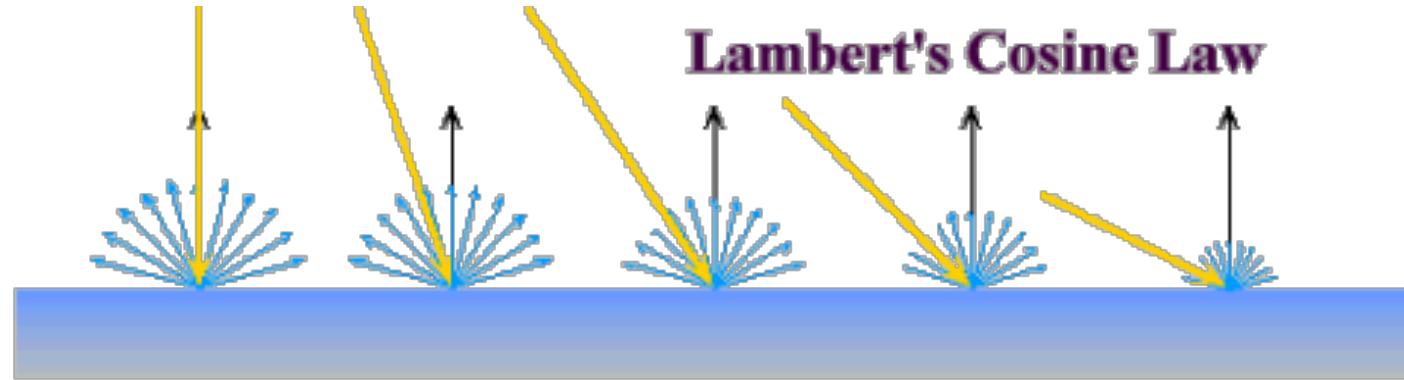


环境光反射

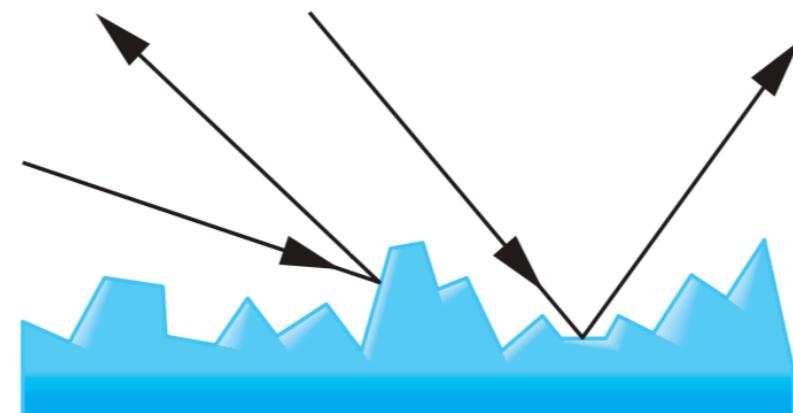
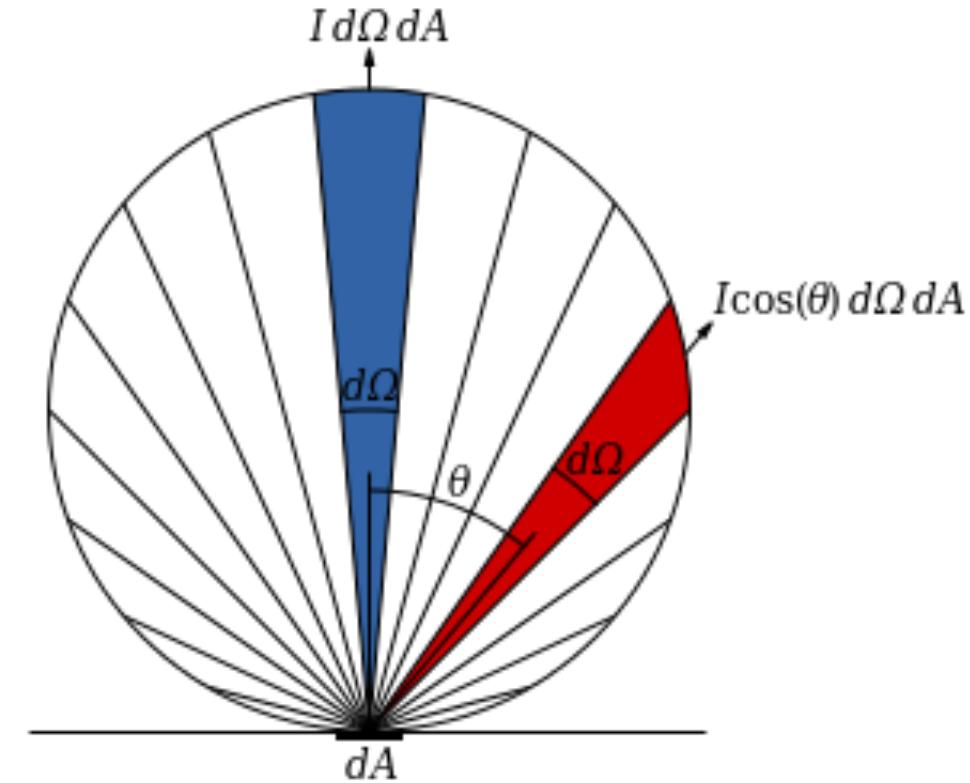
- 场景中所有点处环境光强度 L_a 一致
- 当环境光照射在物体表面时，一部分被表面吸收，另一部分被表面反射，反射部分强度由环境光反射系数 k_a 决定
 - 因此环境光反射强度为 $I_a = k_a L_a$
- 物体表面可以有三个环境光反射系数 k_{ar}, k_{ag}, k_{ab}
 - 如，物体表面反射系数 k_{ar}, k_{ag} 大而 k_{ab} 小时，其表面反射红光及绿光而吸收蓝光，在白色环境光照射下，物体呈现黄色



- 漫反射-Lambert's Cosine Law



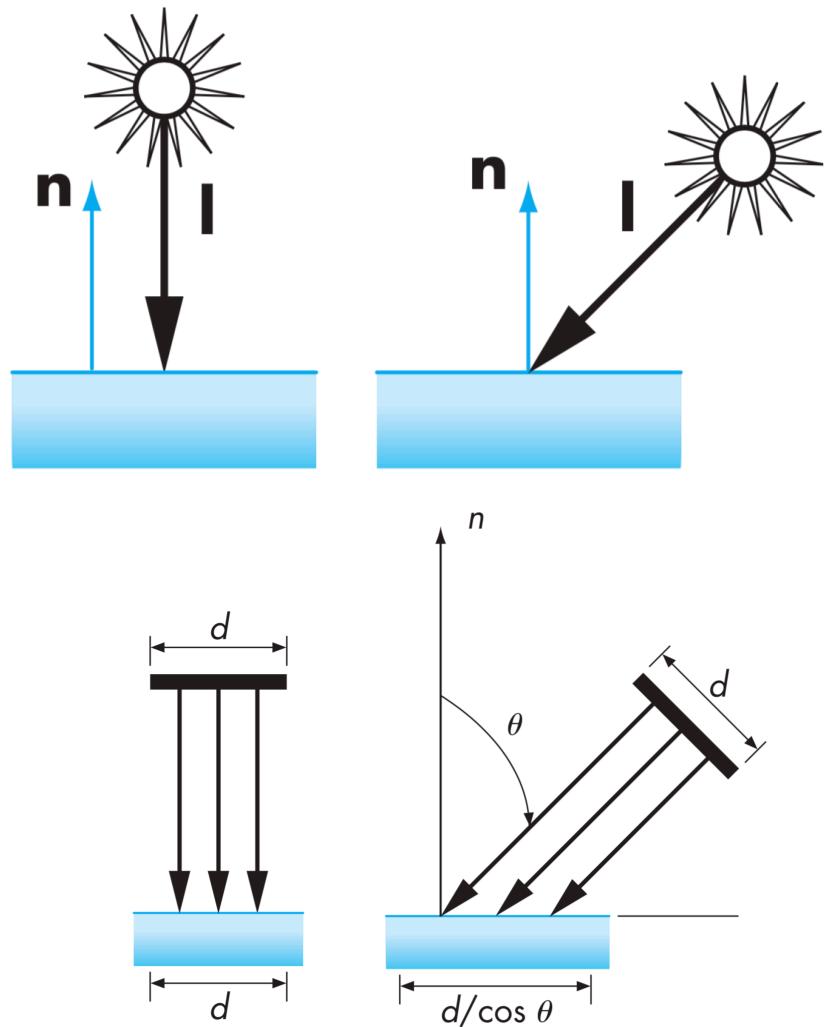
- Lambert定律认为只有入射光线的垂直分量对照明其作用
- Lambert光照模型用于粗糙表面
 - 粗糙表面将光反射至各个方向
 - 也称为Lambertian surface
 - 漫反射光的强度与观察角度无关



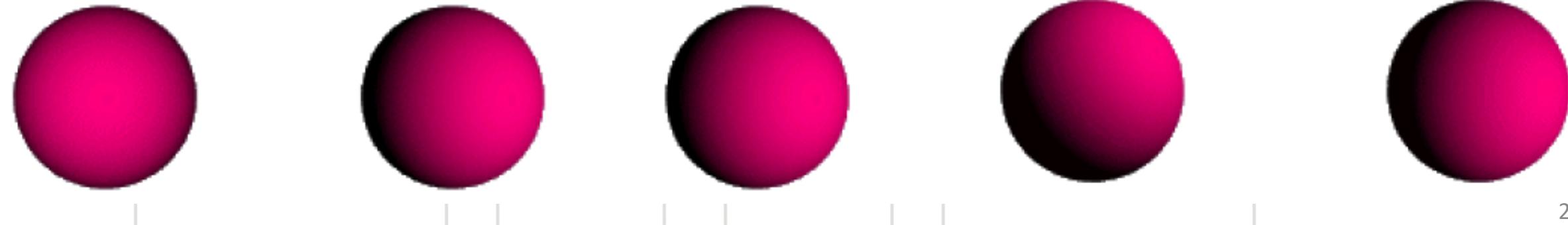
漫反射-Lambert's Cosine Law

– 漫反射光强度计算

- $I_d = k_d L_d \cdot \cos \theta$, θ 为照射点处法向量 n 与光照角度 l 的夹角
- 常写作点积形式 $I_d = k_d L_d (n \cdot l)$
- 注意与光照相关的所有计算中，向量都需要归一化（需要单位向量）
- 同样，漫反射光可以有三个分量

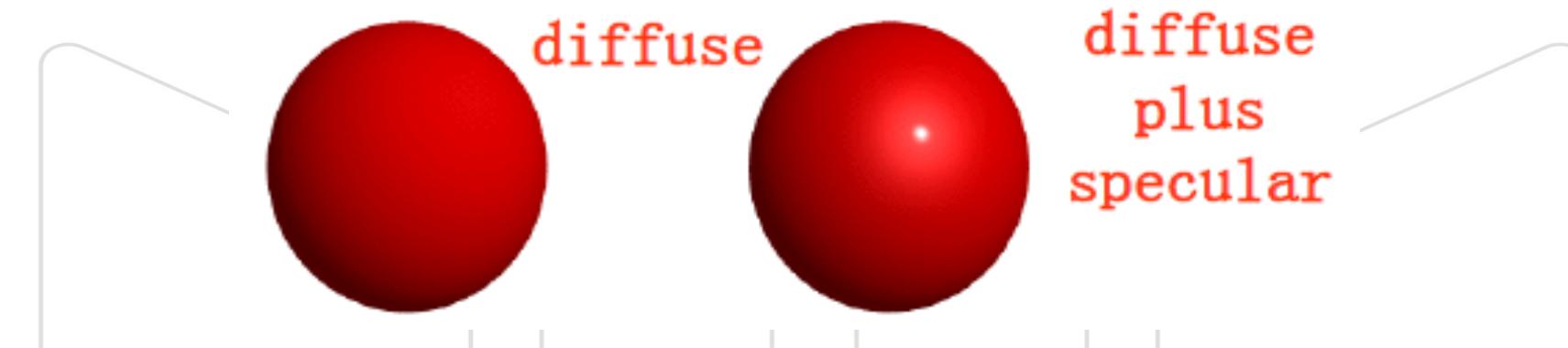
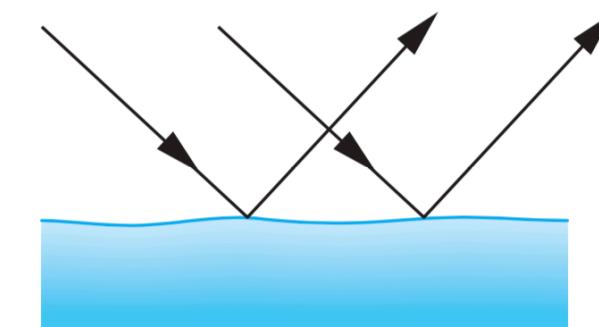


不同光照角度产生的
Lambertian
sphere



镜面反射

- 光泽曲面上产生镜面反射
 - 如，抛光金属，特殊涂料，等
 - 镜面反射假设表面光滑
- Specular highlight
 - Specular surface上产生的高亮点
- 镜面反射依赖于观察角度
 - 高亮位置是以观察角度为参数的函数



● 镜面反射

– 镜面反射的反射角等于入射角

- 公元1世纪，亚历山大城的古希腊数学家海伦（Heron of Alexandria）提出
- 因此当观察者靠近反射光时，镜面反射强度大

– 反射光强度计算

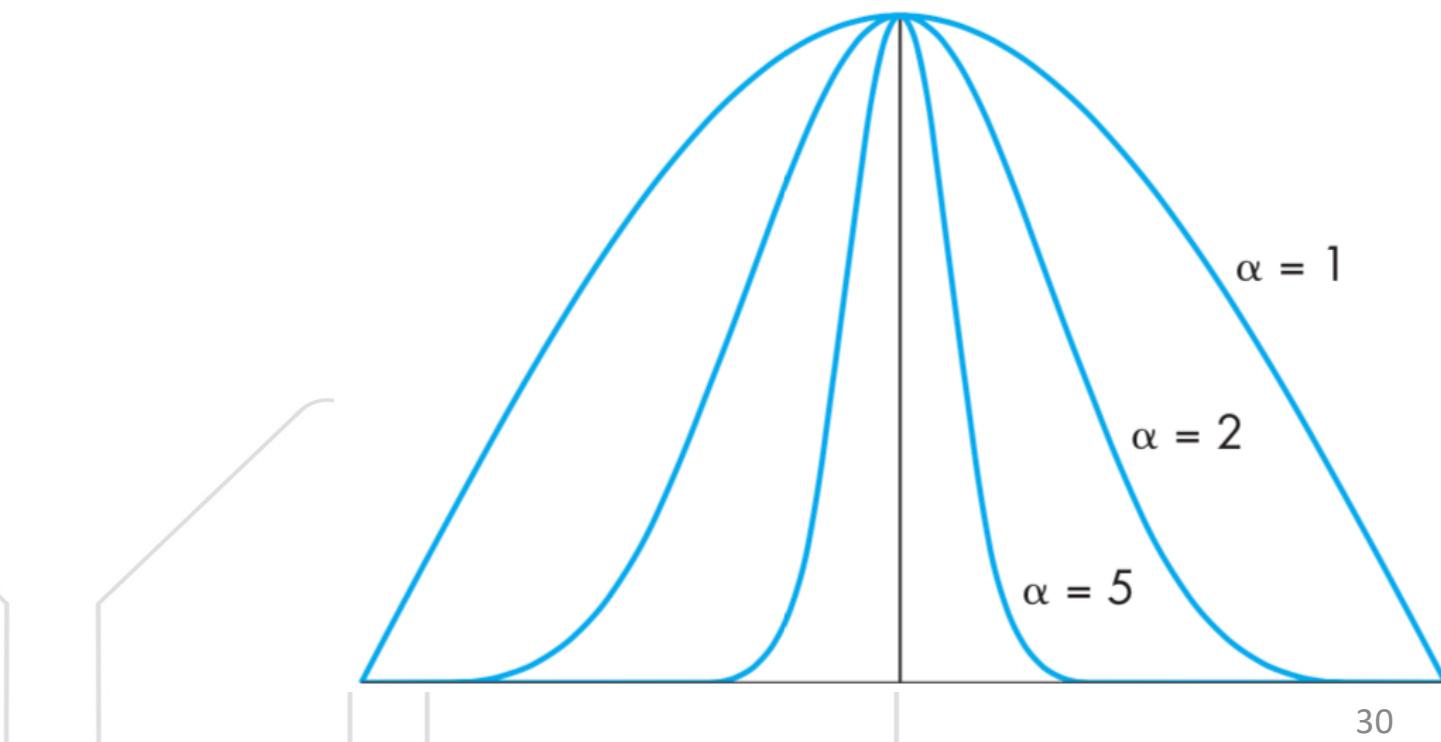
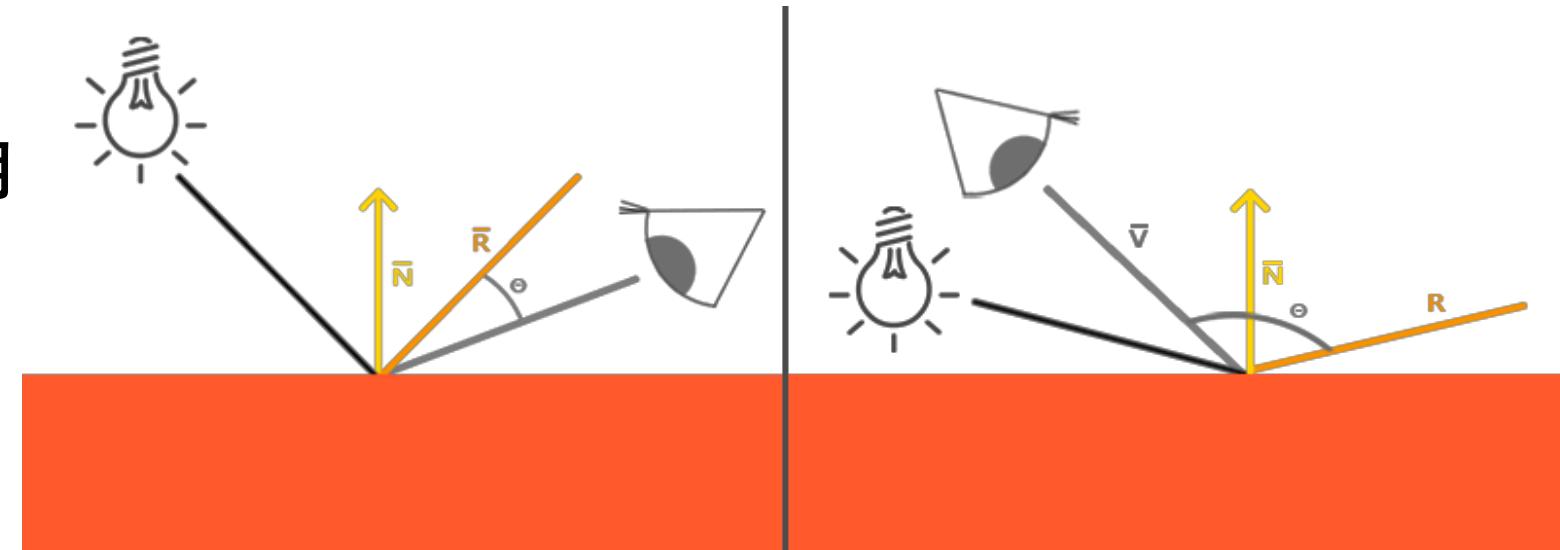
$$\bullet I_s = k_s L_s \cdot \max((\mathbf{r} \cdot \mathbf{v})^\alpha, 0)$$

• \mathbf{r} : 反射方向

• \mathbf{v} : 观察方向

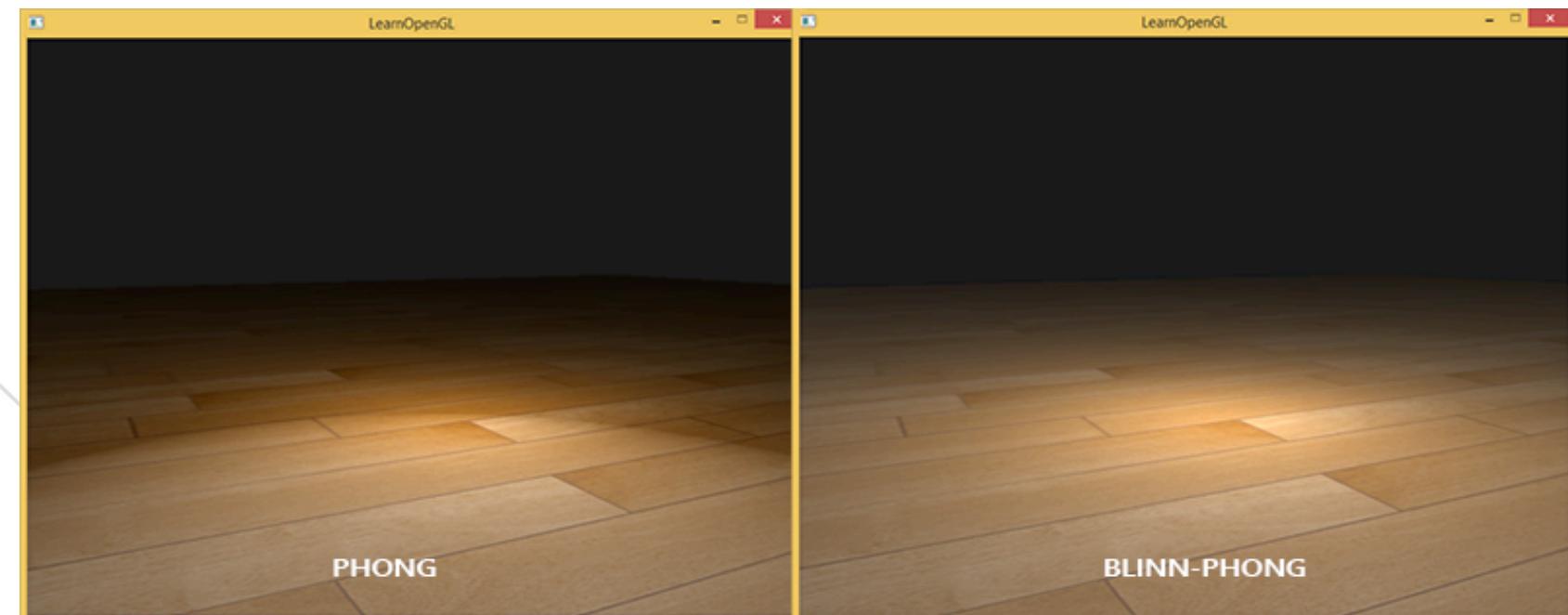
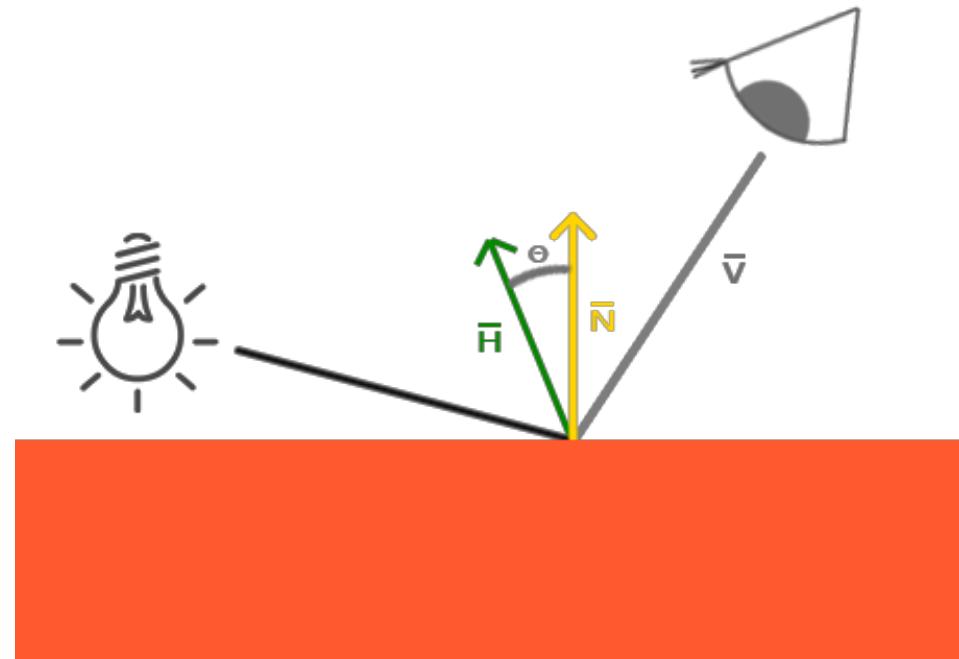
• α : 高光系数 (shininess)

• $\mathbf{r} \cdot \mathbf{v}$ 为负时镜面反射不起作用



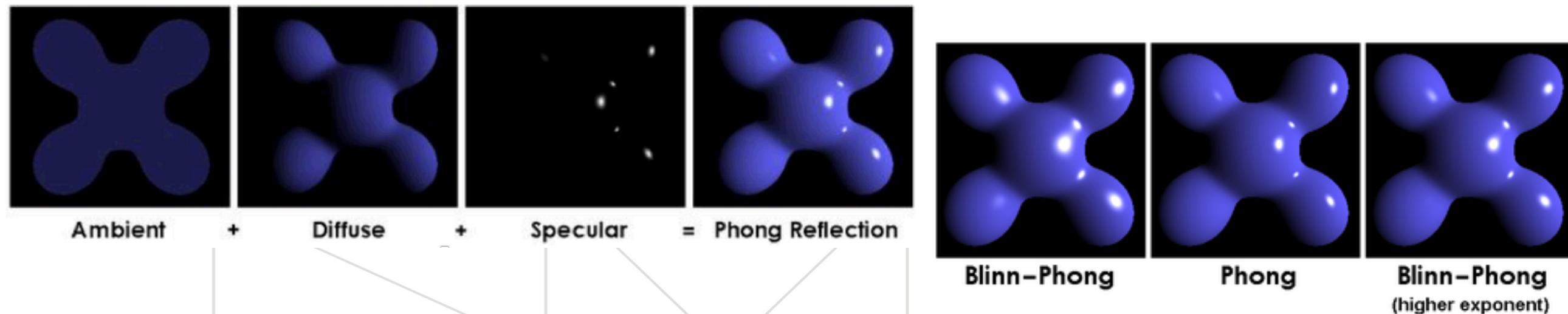
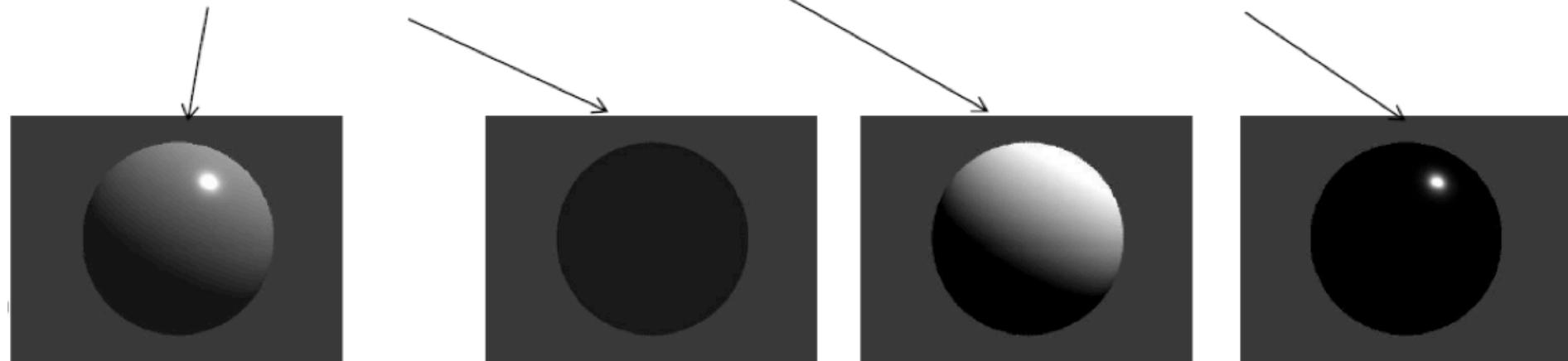
镜面反射

- $r \cdot v$ 为负时镜面反射不起作用
 - 看似合理，但在 $r \cdot v = 0$ 处不连续
- Blinn-Phong反射模型
 - 使用半角向量近似Phong模型中的镜面反射
 - $h = \frac{l+v}{|l+v|}$
 - $I_s = k_s L_s \cdot \max((n \cdot h)^\alpha, 0)$
 - 同时在某些情况下能减少计算量



环境光反射、漫反射、镜面反射合成

$$I = k_a L_a + k_d L_d(n \cdot l) + k_s L_s \cdot \max((n \cdot h)^\alpha, 0)$$



● 环境光反射、漫反射、镜面反射合成

$$- I = k_a \mathbf{L}_a + k_d \mathbf{L}_d (\mathbf{n} \cdot \mathbf{l}) + k_s \mathbf{L}_s \cdot \max((\mathbf{n} \cdot \mathbf{h})^\alpha, 0)$$

- 对漫反射及镜面反射可考虑衰减

$$\bullet I = \frac{1}{a+bd+cd^2} \left(k_d \mathbf{L}_d (\mathbf{n} \cdot \mathbf{l}) + k_s \mathbf{L}_s \cdot \max((\mathbf{n} \cdot \mathbf{h})^\alpha, 0) \right) + k_a \mathbf{L}_a$$

- 多光源合成

$$\bullet I = k_a \mathbf{L}_a + \sum_i \left(k_d \mathbf{L}_d (\mathbf{n} \cdot \mathbf{l}_i) + k_s \mathbf{L}_s \cdot \max((\mathbf{n} \cdot \mathbf{h}_i)^\alpha, 0) \right)$$

- 由于三原色可能具有不同的反射系数及光源强度

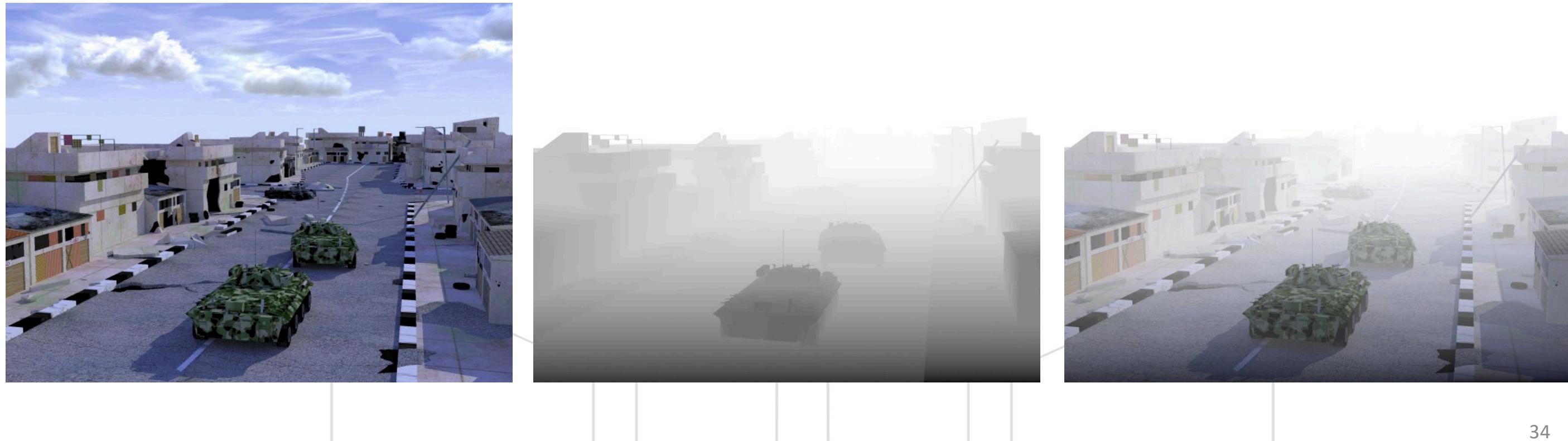
$$\bullet I = k_{ar} \mathbf{L}_{ar} + k_{dr} \mathbf{L}_{dr} (\mathbf{n} \cdot \mathbf{l}) + k_{sr} \mathbf{L}_{sr} \cdot \max((\mathbf{n} \cdot \mathbf{h})^\alpha, 0)$$

$$\bullet I = k_{ag} \mathbf{L}_{ag} + k_{dg} \mathbf{L}_{dg} (\mathbf{n} \cdot \mathbf{l}) + k_{sg} \mathbf{L}_{sg} \cdot \max((\mathbf{n} \cdot \mathbf{h})^\alpha, 0)$$

$$\bullet I = k_{ab} \mathbf{L}_{ab} + k_{db} \mathbf{L}_{db} (\mathbf{n} \cdot \mathbf{l}) + k_{sb} \mathbf{L}_{sb} \cdot \max((\mathbf{n} \cdot \mathbf{h})^\alpha, 0)$$

Depth cueing/Atmospheric attenuation

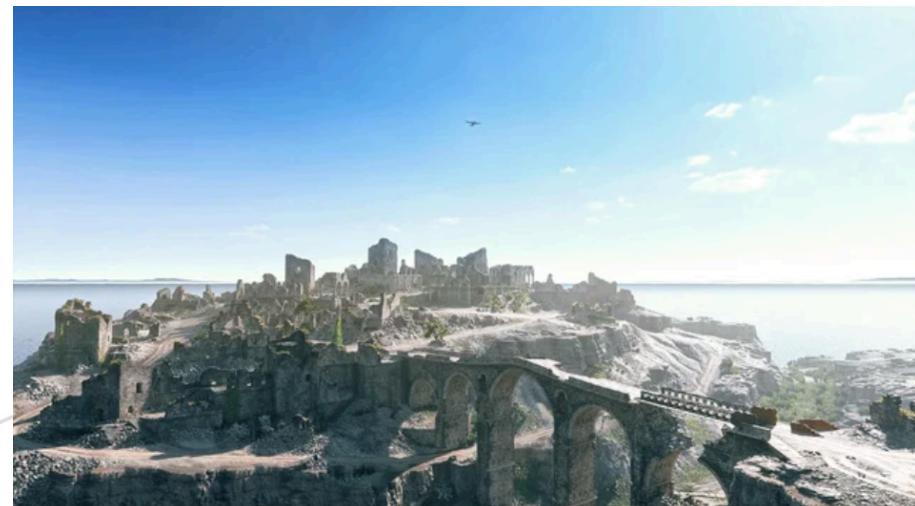
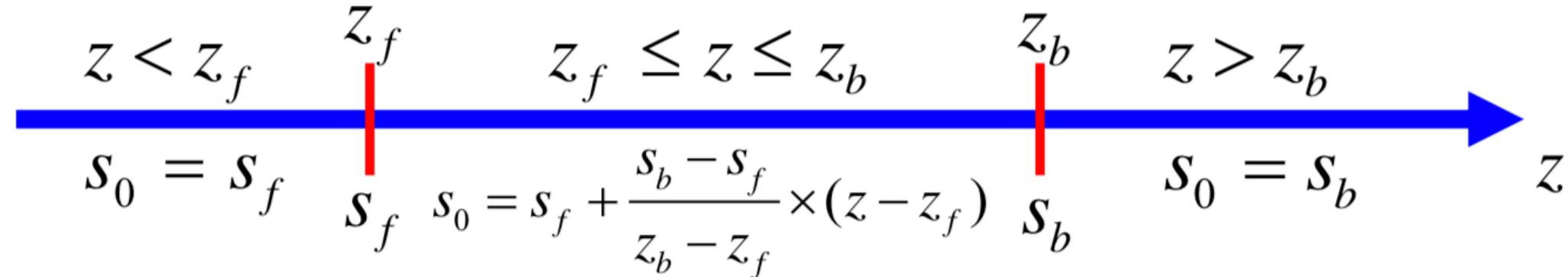
- 光照强度随距离变化
- 常用来产生近似雨雪雾天气效果
- 使用物体渲染过程中的深度信息近似距离



Depth cueing/Atmospheric attenuation实现

- 将渲染得到的颜色 I 与 depth cue 颜色（背景） I_{dc} 融合
 - 融合时的比例由参数 s 决定： $I_{new} = sI + (1 - s)I_{dc}$

- s 由深度决定



简介

光源

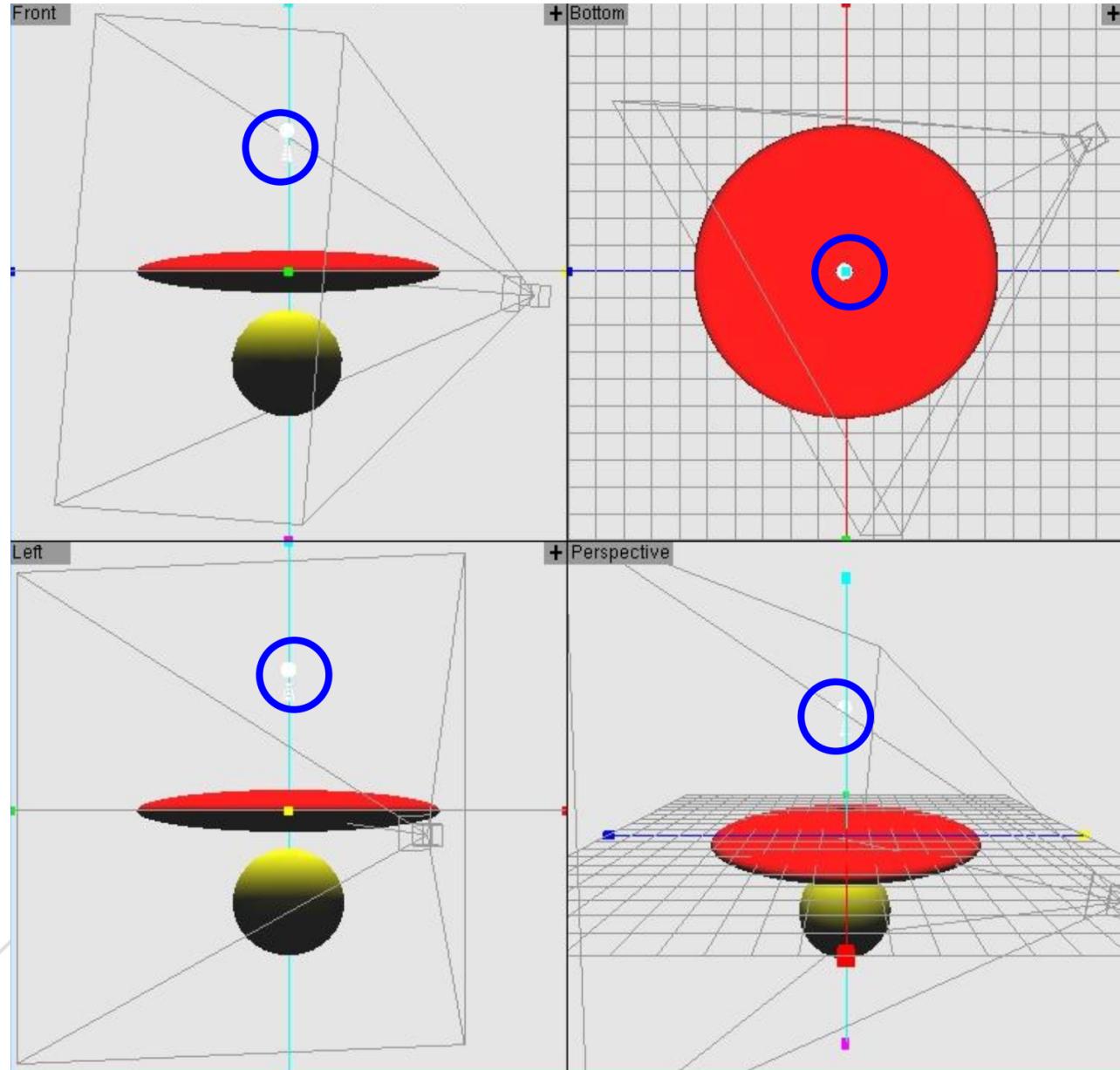
Phong反射模型

OpenGL中的光照

其他



- OpenGL使用的是局部光照模型
 - Local illumination
 - 每个点的颜色取决于
 - 光源、视点位置、该点处法向量、材料
 - 与物体间的关系、物体自身不同部分之间的关系无关
 - 无间接光照
 - 无阴影



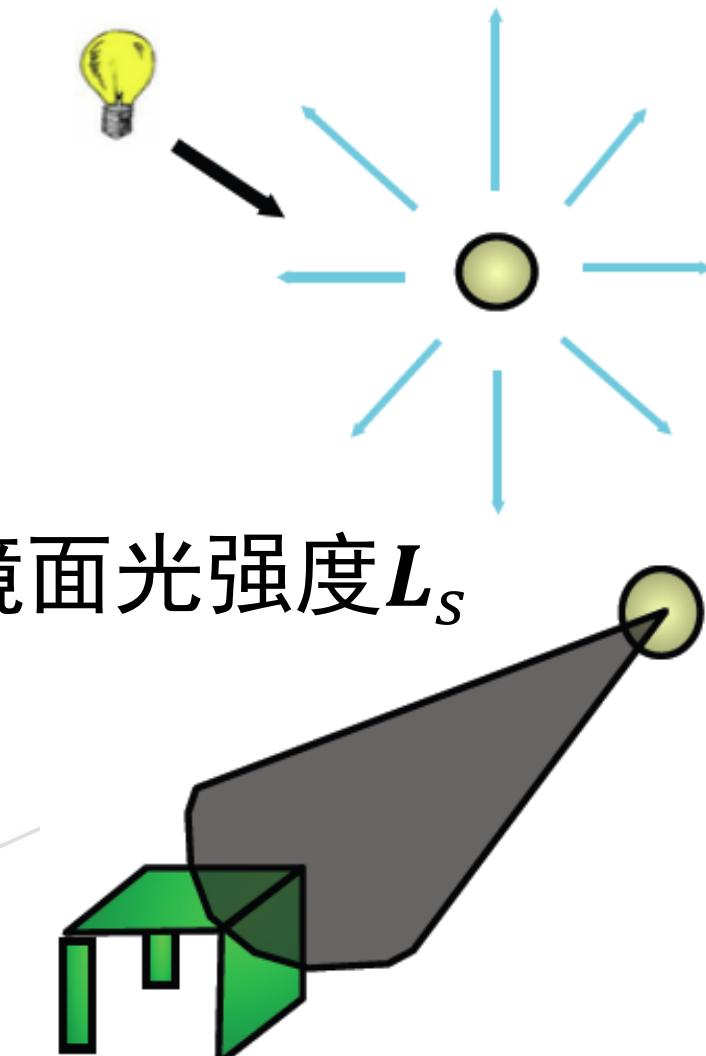
图片来自Ching-Kuang Shene课件

● 光源类型

- 环境光: Phong反射模型中的 L_a , 与光源位置, 方向无关
- 点光源: 向所有方向发射光, 有位置, 无大小
- 聚光灯: 在一个圆锥体范围内发光

● 多光源

- 大多OpenGL实现中有8个光源
- 每个光源都可以设置独立的漫反射光强度 L_d 与镜面光强度 L_s
- 每个光源也可以设置独立的环境光强度 L_a
 - 但环境光对物体的影响与光源位置无关
- 不同的光源可使用宏定义引用
 - `GL_LIGHT0, GL_LIGHT1, ... , GL_LIGHT8`



创建光源

- 使用**glLightfv(GL_LIGHTX, PROPERTY, value)**进行设置
- 设置光源位置

```
GLfloat light_position[]={1.0f, 1.0f, 1.0f, 0.0f};
```

```
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

- 设置光源强度

```
GLfloat ambient[]={0.2f,0.2f,0.2f,1.0f};
```

```
GLfloat diffuse[]={1.0f,1.0f,1.0f,1.0f};
```

```
GLfloat specular[]={1.0f,1.0f,1.0f,1.0f};
```

```
glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
```

```
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
```

```
glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
```

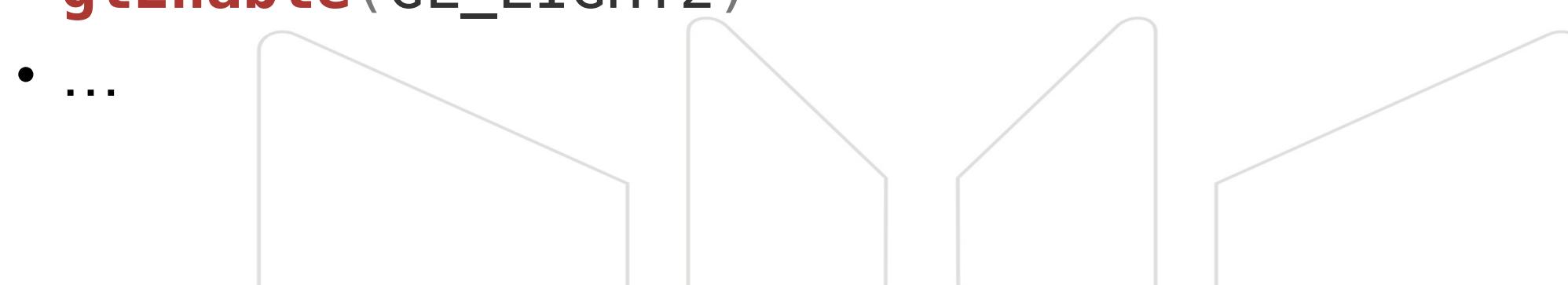
设置光源

- 所有光源属性
 - OpenGL通过适当参数构造前述光源类型

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)	光源位置其次坐标 (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_LINEAR_ATTENUATION	0.0	线性衰减因子
GL_QUADRATIC_ATTENUATION	0.0	平方衰减因子

启动光源

- 使光照生效，首先得启动光照
 - **glEnable(GL_LIGHTING)**
- 使光照无效，则需要关闭光照
 - **glDisable(GL_LIGHTING)**
- 在光照生效的情况下，使特定光源生效
 - **glEnable(GL_LIGHT0)**
 - **glEnable(GL_LIGHT1)**
 - **glEnable(GL_LIGHT2)**
 - ...



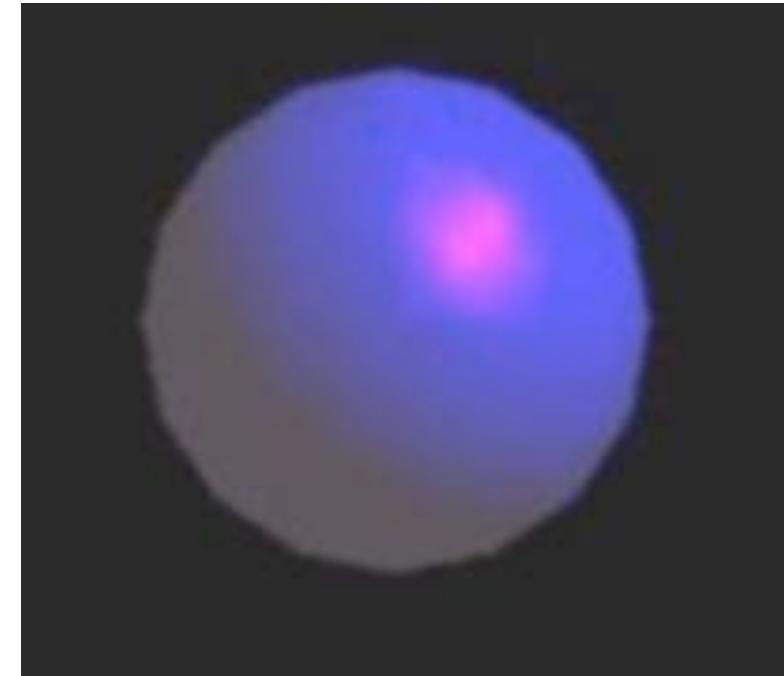
设置材料

- 光源决定光照强度，材料特性决定对光的反射
 - 如，光源强度为(1.0, 0.5, 0.5)，材料的反射为(0.0, 1.0, 0.5)时，最终显示颜色为rgb每个分量分别相乘(1.0x0.0, 0.5x1.0, 0.5x0.5)
- 使用**glMaterialfv(face, PROPERTY, value)**进行设置
 - 其中face为GL_FRONT/GL_BACK/GL_FRONT_AND_BACK

参数名	缺省值	说明
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	材料环境光颜色
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	材料漫反射光颜色
GL_AMBIENT_AND_DIFFUSE	?	材料环境光和漫反射光颜色
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)	材料镜面反射光颜色
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、漫反射光和镜面反射光

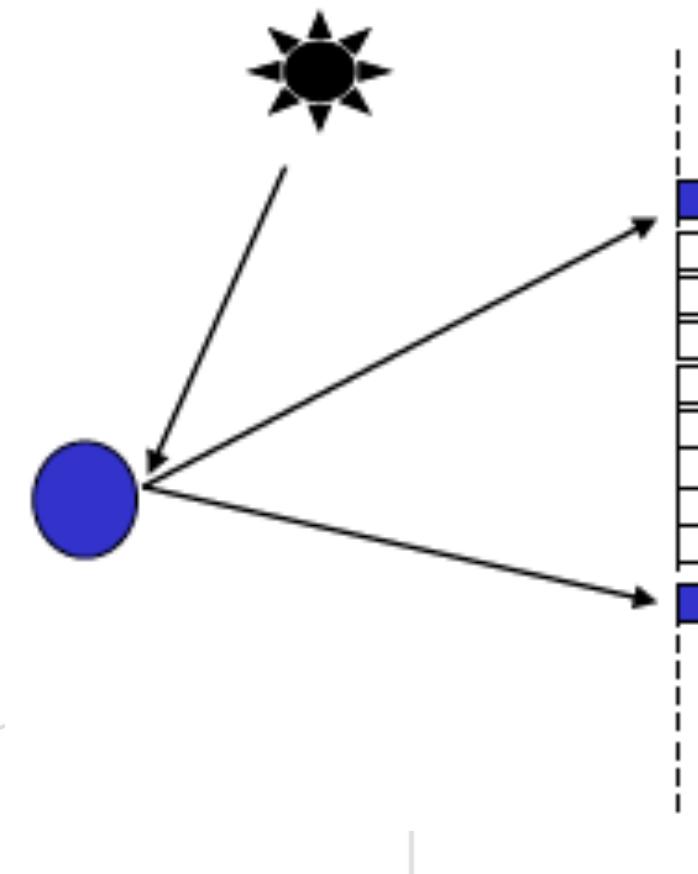
● 应用举例

```
void myinit(void){  
    GLfloat mat_ambient[] = {0.8f, 0.8f, 0.8f, 1.0f};  
    GLfloat mat_diffuse[] = {0.8f, 0.0f, 0.8f, 1.0f};  
    GLfloat mat_specular[] = {1.0f, 0.0f, 1.0f, 1.0f};  
    GLfloat mat_shininess[] = {50.0f};  
  
    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[] = {0.0f, 0.0f, 1.0f, 1.0f};  
    GLfloat light_position[] = {1.0f, 1.0f, 1.0f, 0.0f};  
  
    glEnable(GL_LIGHTING);  
    glEnable(GL_LIGHT0);  
    glDepthFunc(GL_LESS);  
    glEnable(GL_DEPTH_TEST);  
}
```



光照 (lighting) 与着色 (shading)

- 光照是根据光照模型计算离开物体上每一个三维点的光照强度
- 着色是计算物体表面所对应的每一个像素的显示颜色的过程
 - 直接思路：
 - 对每个像素计算其对应的三维位置/法向量
 - 根据三维位置计算光照方向及观察方向
 - 应用光照模型计算颜色
 - 开销过于庞大！
 - 在OpenGL中使用的是插值方法
 - Flat Shading
 - Gouraud Shading
 - Phong Shading



着色模型

– Flat Shading

- 对于每个多边形根据Phong反射模型计算颜色
- 多边形上的任意具有相同颜色



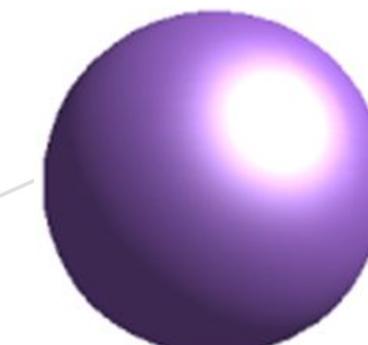
– Gouraud Shading

- 每个顶点根据Phong反射模型计算颜色
- 在多边形内部对顶点颜色插值



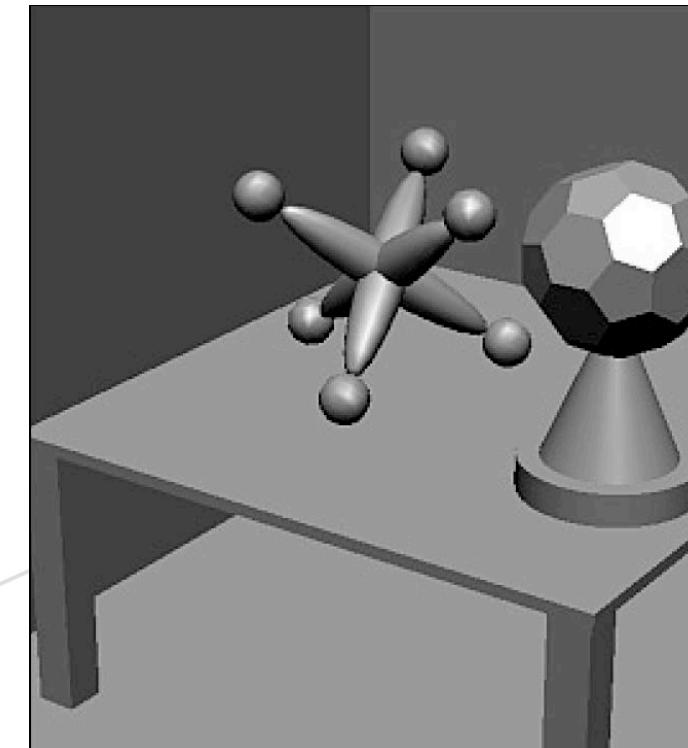
– Phong Shading

- 计算每个顶点的法向量
- 在多边形内部对顶点法向量进行插值
- 根据Phong反射模型计算多边形内部fragment颜色



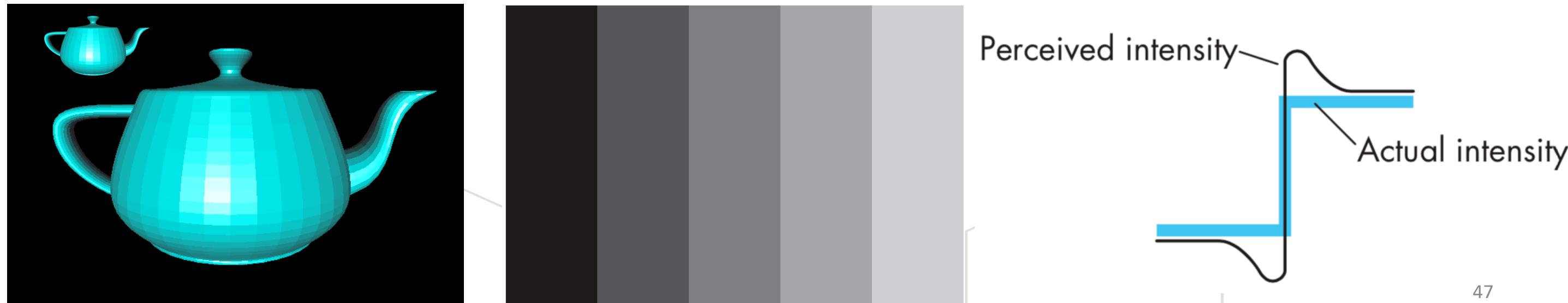
Flat Shading

- 通过**glShadeModel(GL_FLAT)**指定OpenGL使用flat shading
- 最简单的shading模型，也称为faceted shading, constant shading
- 每个多边形具有单一颜色
- 对每个多边形只需要应用光照模型计算一次颜色
- 其隐含假设为
 - 光源位于无穷远处
 - 因此 $n \cdot l$ 在整个多边形上的任意点为常数
 - 观察者位于无穷远处
 - 因此 $r \cdot v$ 在整个多边形上的任意点为常数
 - 多边形如实还原了场景中的物体表面



Flat Shading

- 计算量显著地比其他基于插值的着色模型小：快
 - 最常见的应用情景为初略地预览整个场景
- 如果多边形相对屏幕而言很小（例如，一个像素大小），其着色效果与其他插值模型相当
- 但在绝大多数情况下，用其显示光滑表面的效果不佳
 - 人类视觉系统对光强变化非常敏感（侧抑制性质，lateral inhibition）
 - 在明暗变化的边界产生马赫带（Mach bands）



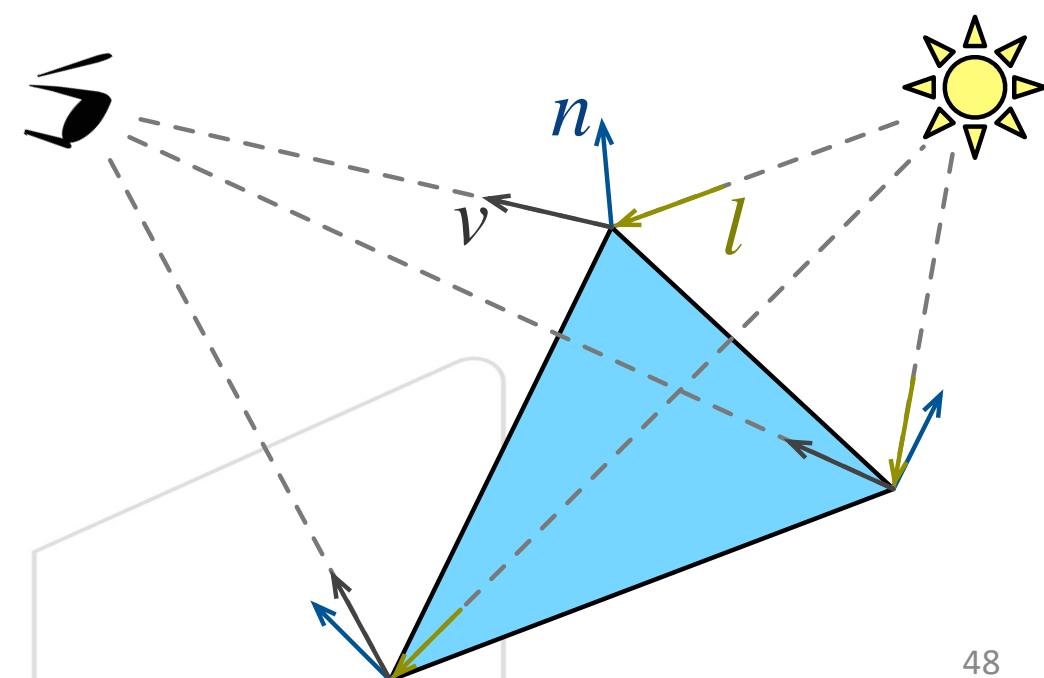
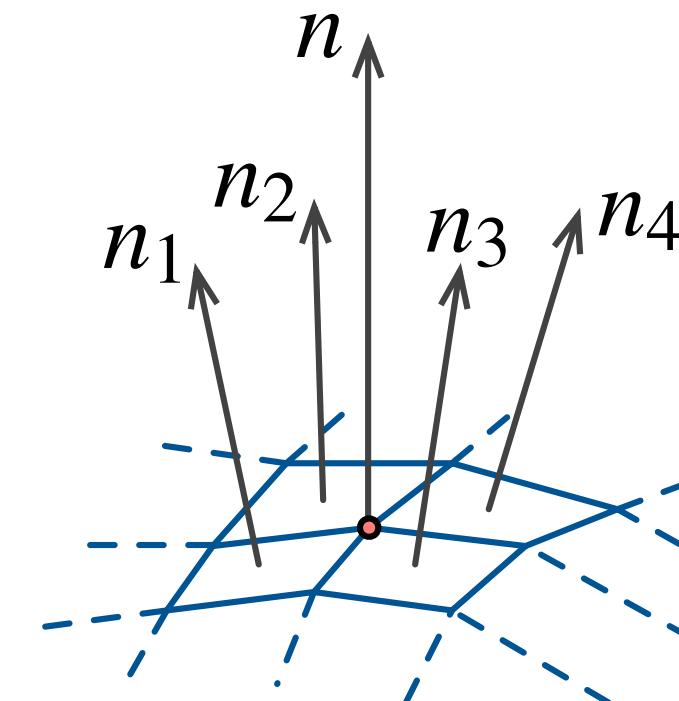
● Gouraud Shading

- `glShadeModel(GL_SMOOTH)`
- 为基于插值的方法，也称为intensity/color interpolation shading
- 计算步骤

- 1. 计算每个顶点的法向量方向（如果未知）
 - 相邻多边形法向量的平均

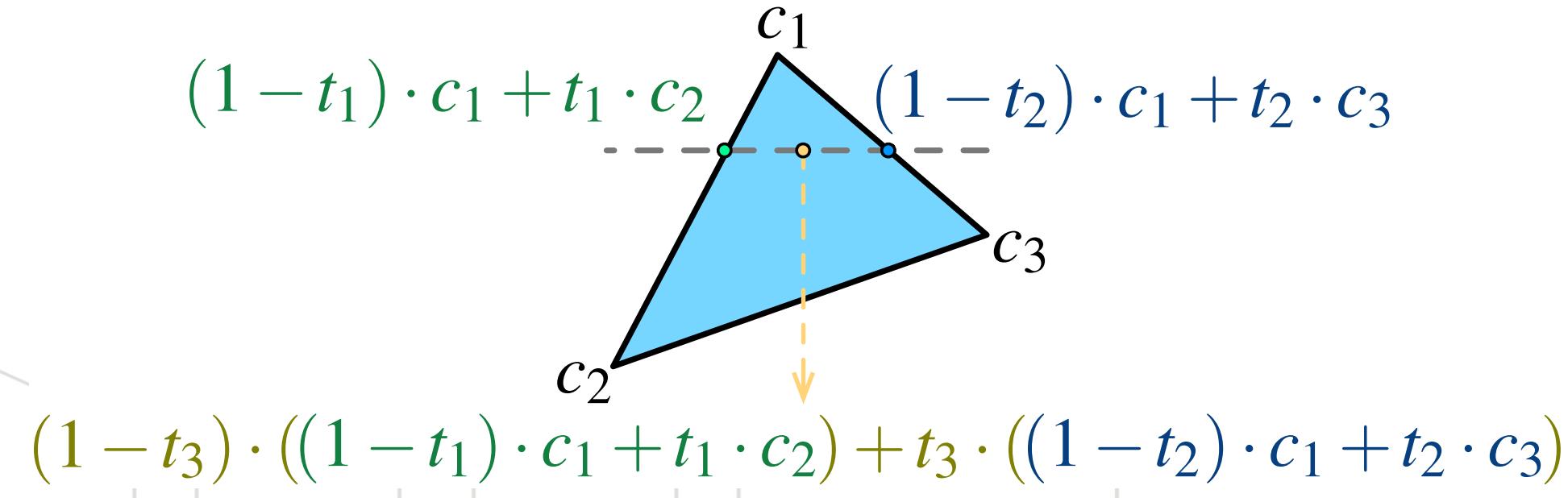
$$-\mathbf{n} = \frac{\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4}{|\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4|}$$

- 2. 计算每个顶点的颜色
 - 使用Phong反射模型
- 3. 计算多边形内部fragment颜色
 - 对顶点颜色进行插值



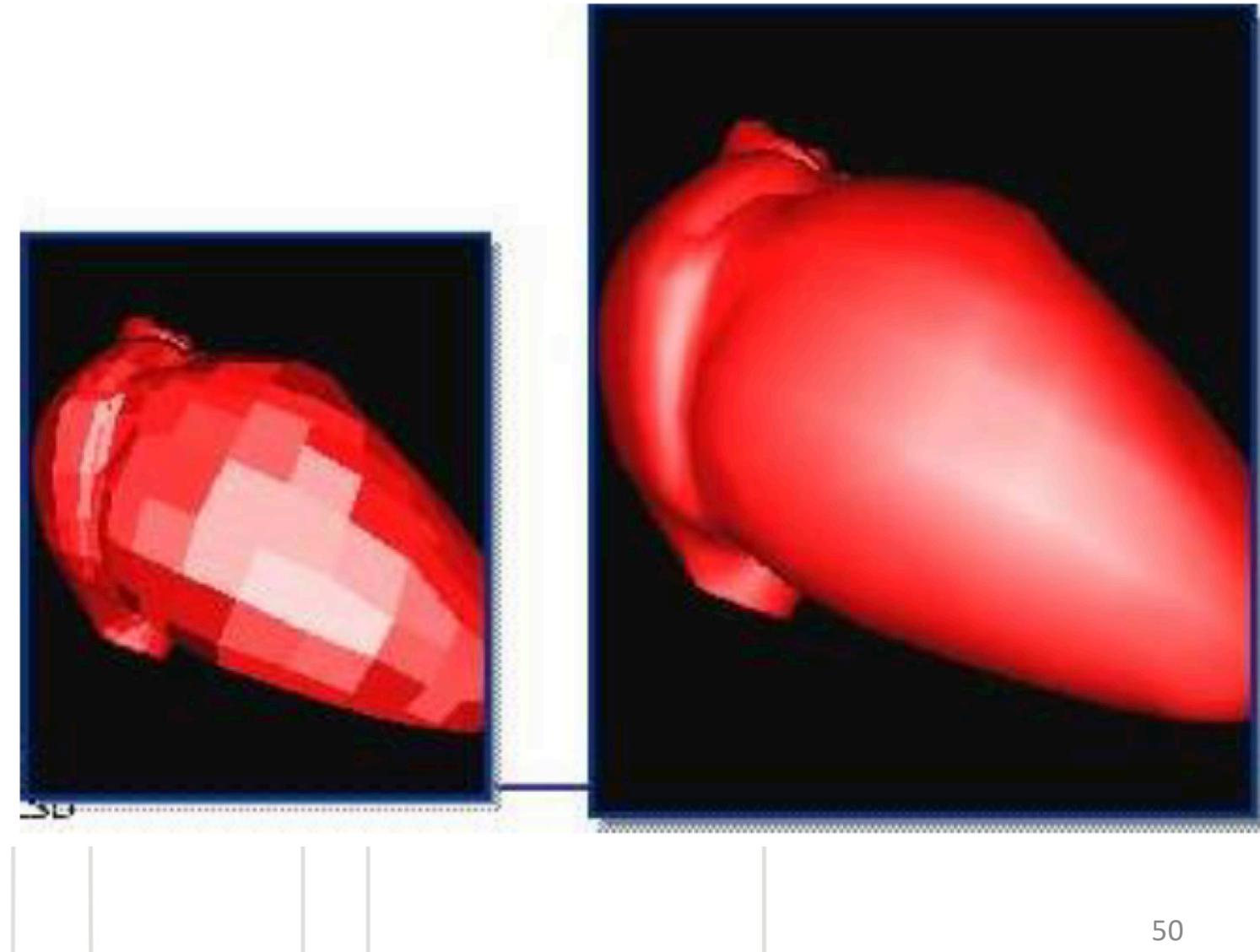
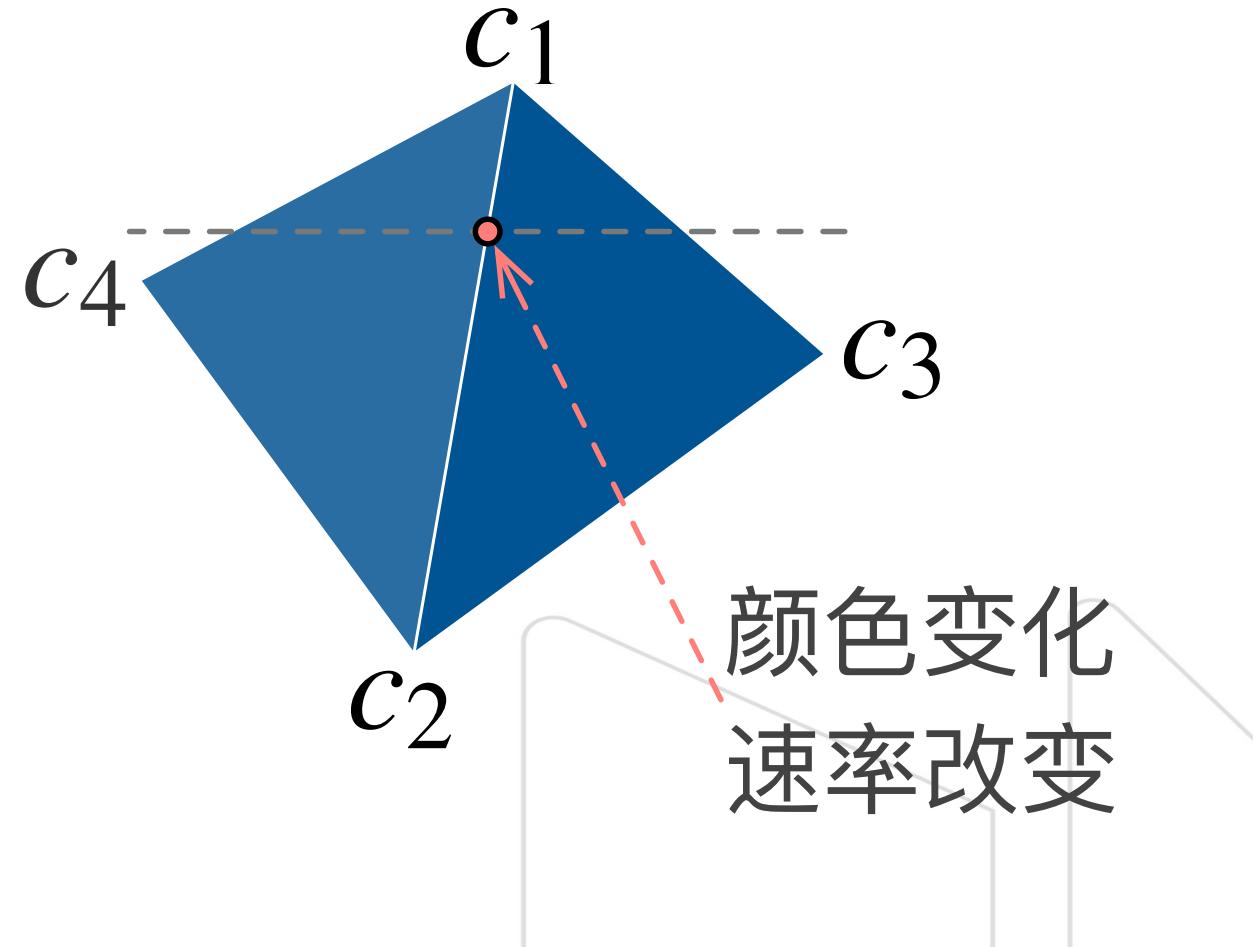
Gouraud Shading

- 使用线性插值计算fragment颜色
 - 沿三角形边对顶点颜色进行插值得到边上fragment颜色
 - 沿扫描线对边上fragment颜色进行插值得到内部fragment颜色
- Gouraud Shading并不能消除马赫带



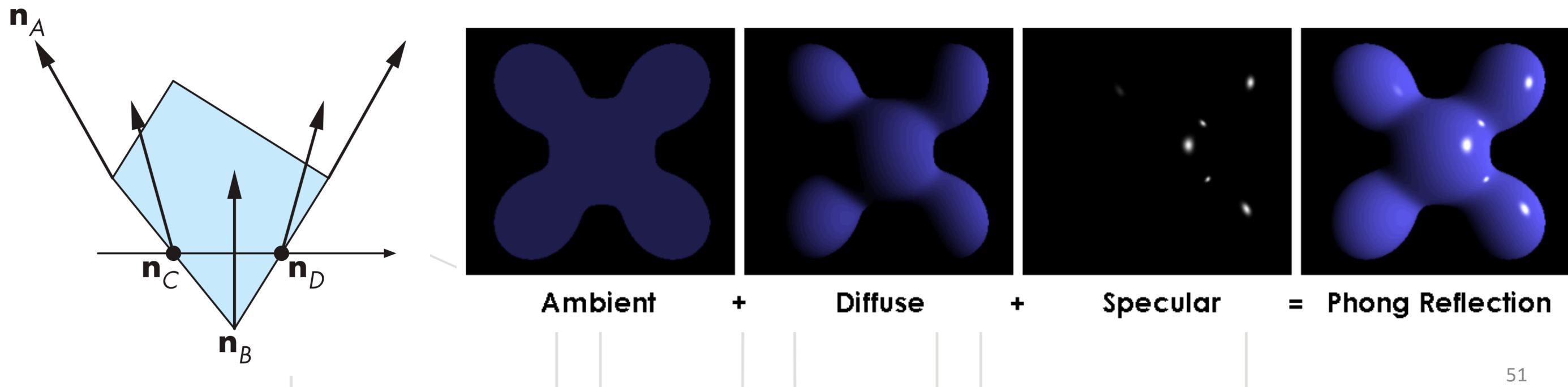
- Gouraud Shading

- Gouraud Shading并不能消除马赫带



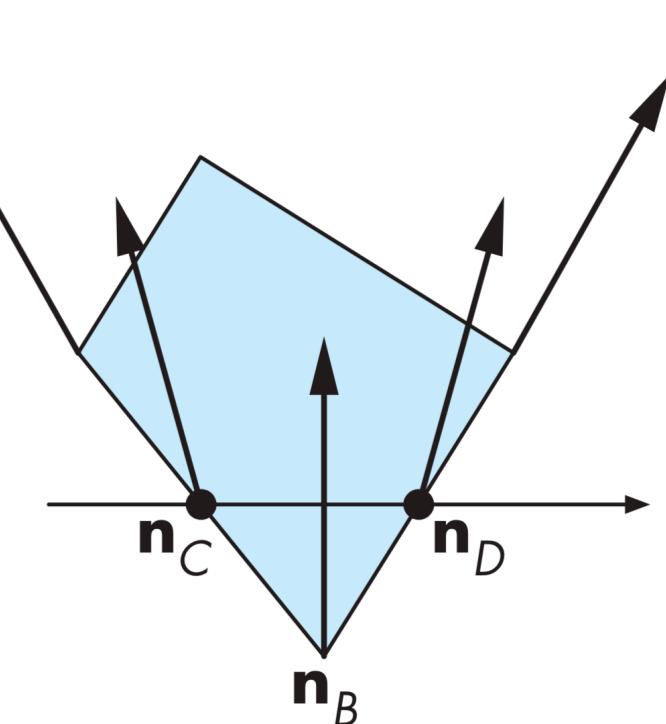
● Phong Shading

- Phong shading不是Phong反射模型
 - 尽管Phong shading与Phong反射模型常混合使用
 - Phong反射模型：根据光源位置、观察点位置、照射点位置计算反射光强度的光照模型
 - Phong shading：基于法向量线性插值的着色方法

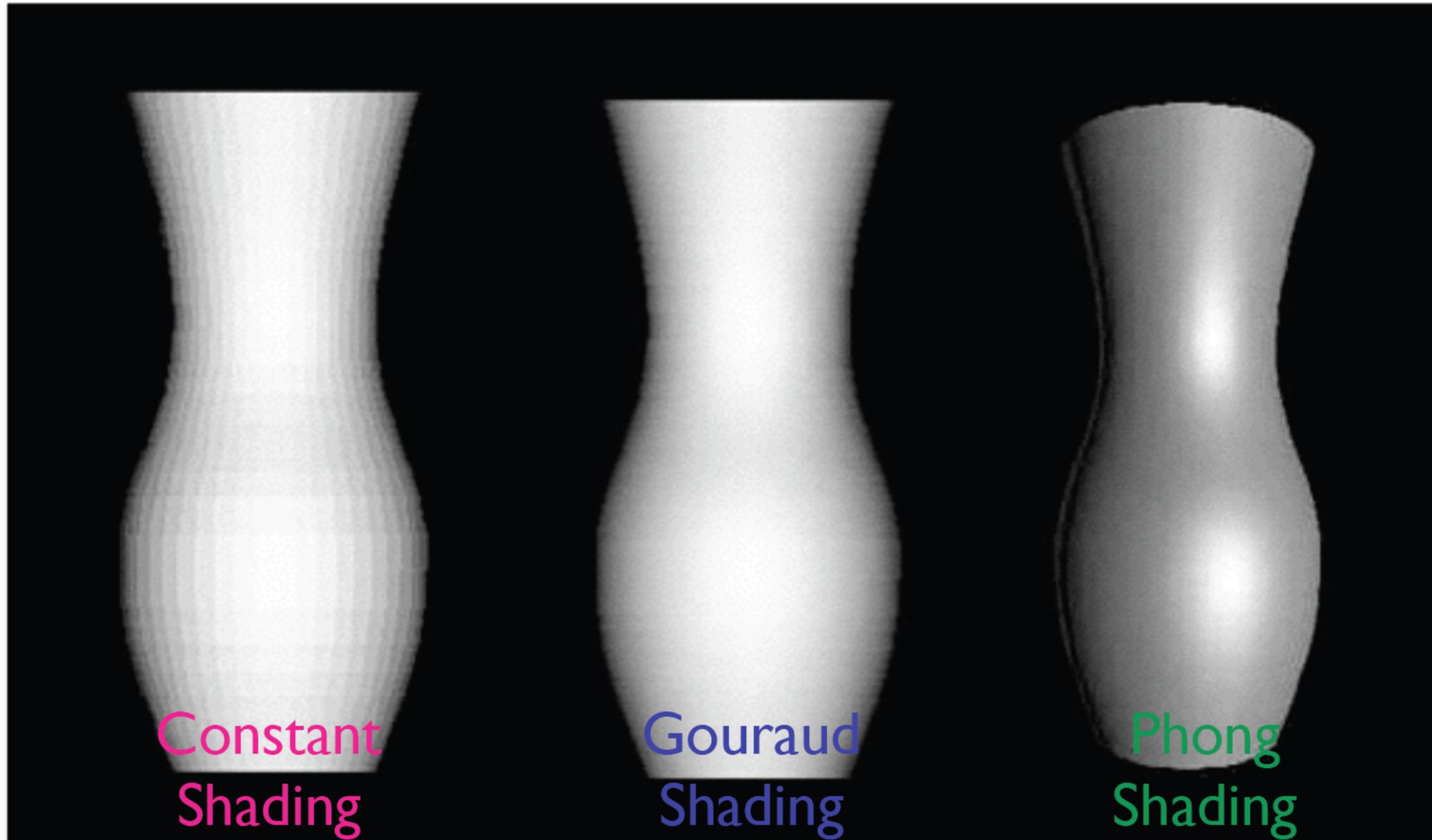


Phong Shading

- 输入：顶点法向量
 - 如果未知则通过相邻多边形法向量计算
 - 实际计算中多使用临边求向量积
 - 与Gouraud shading一致
- 计算过程：
 - 1. 通过对顶点法向量线性插值计算内部fragment法向量 n
 - 与Gouraud shading计算颜色方法一致
 - 2. 通过对顶点位置线性插值计算fragment对应的三维位置
 - 由此计算光照方向 l 及观察方向 v
 - 3. 通过光照模型计算fragment颜色

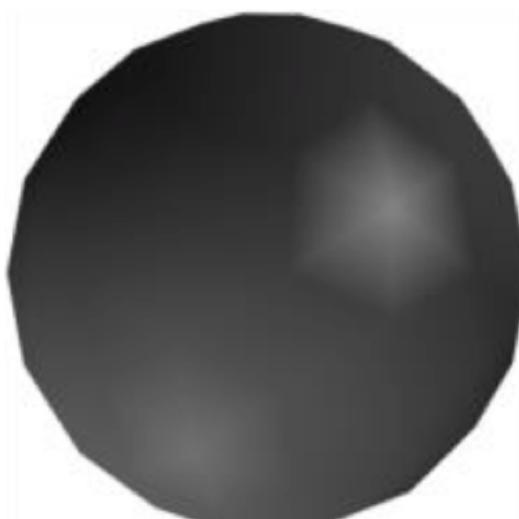


Phong Shading

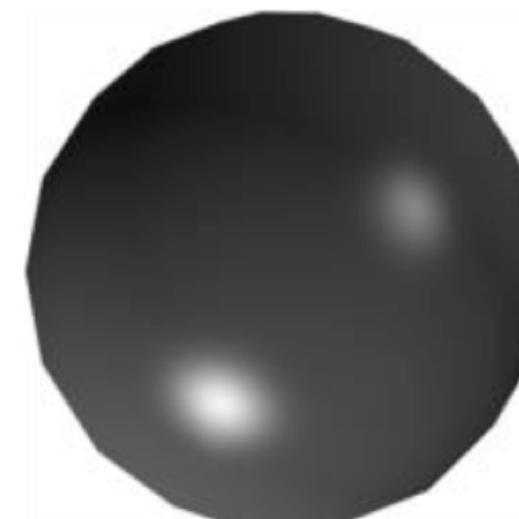


Phong Shading

- 光照强度变化最为平滑自然
- 计算开销比Gouraud shading更大 (6~8倍)
 - Gouraud: per vertex; Phong: per fragment
 - 需要通过glsl实现
- 实际渲染效果仍然受制于模型 (尤其是剪影线)
 - 解决方案: 对整个曲面进行细分; 对剪影线进行细分 (依赖于观察视角)



Gouraud



Phong

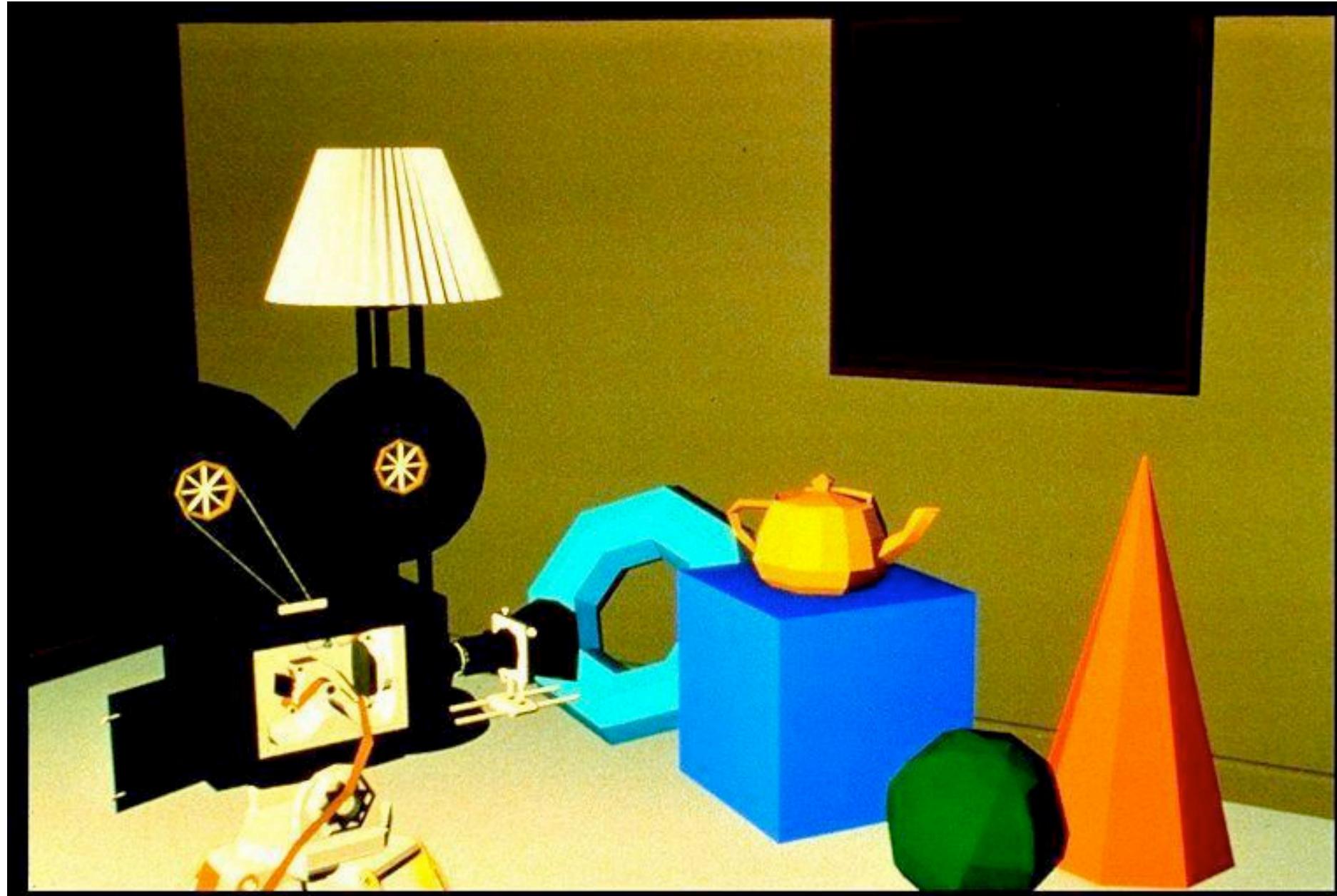


Gouraud

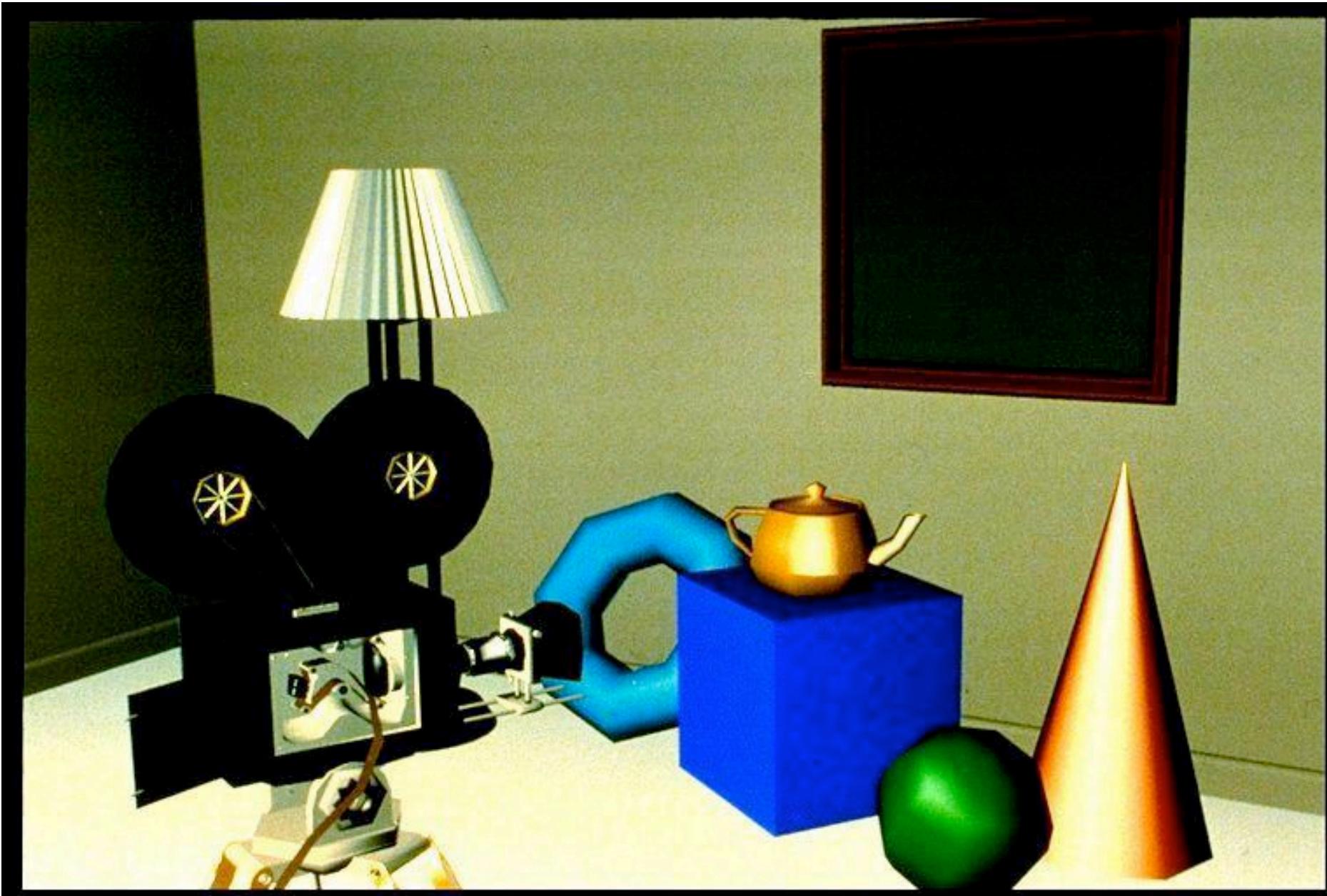


Phong

- Flat shading vs Gouraud shading vs Phong shading



Flat shading vs Gouraud shading vs Phong shading



Flat shading vs Gouraud shading vs Phong shading



简介

光源

Phong反射模型

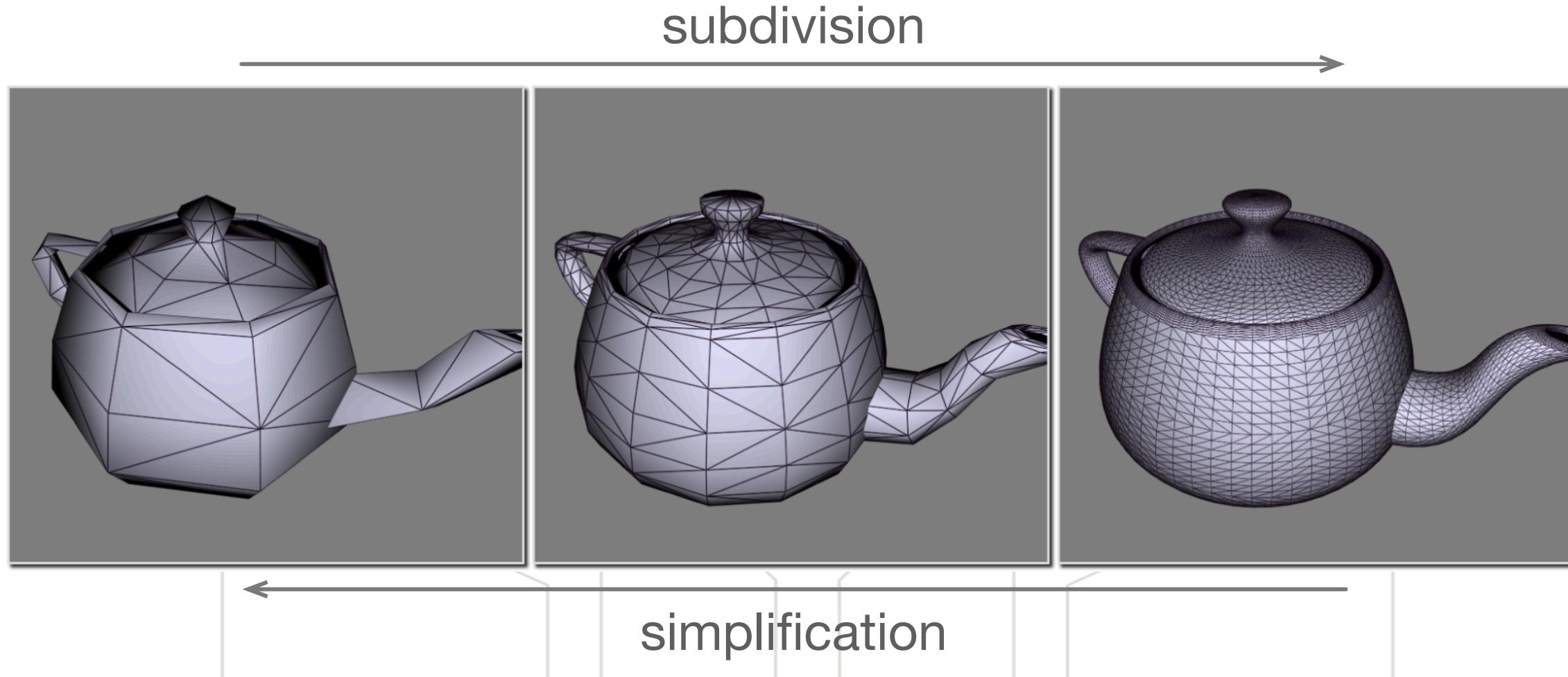
OpenGL中的光照

其他



光滑曲面可以有不同粒度的多边形近似

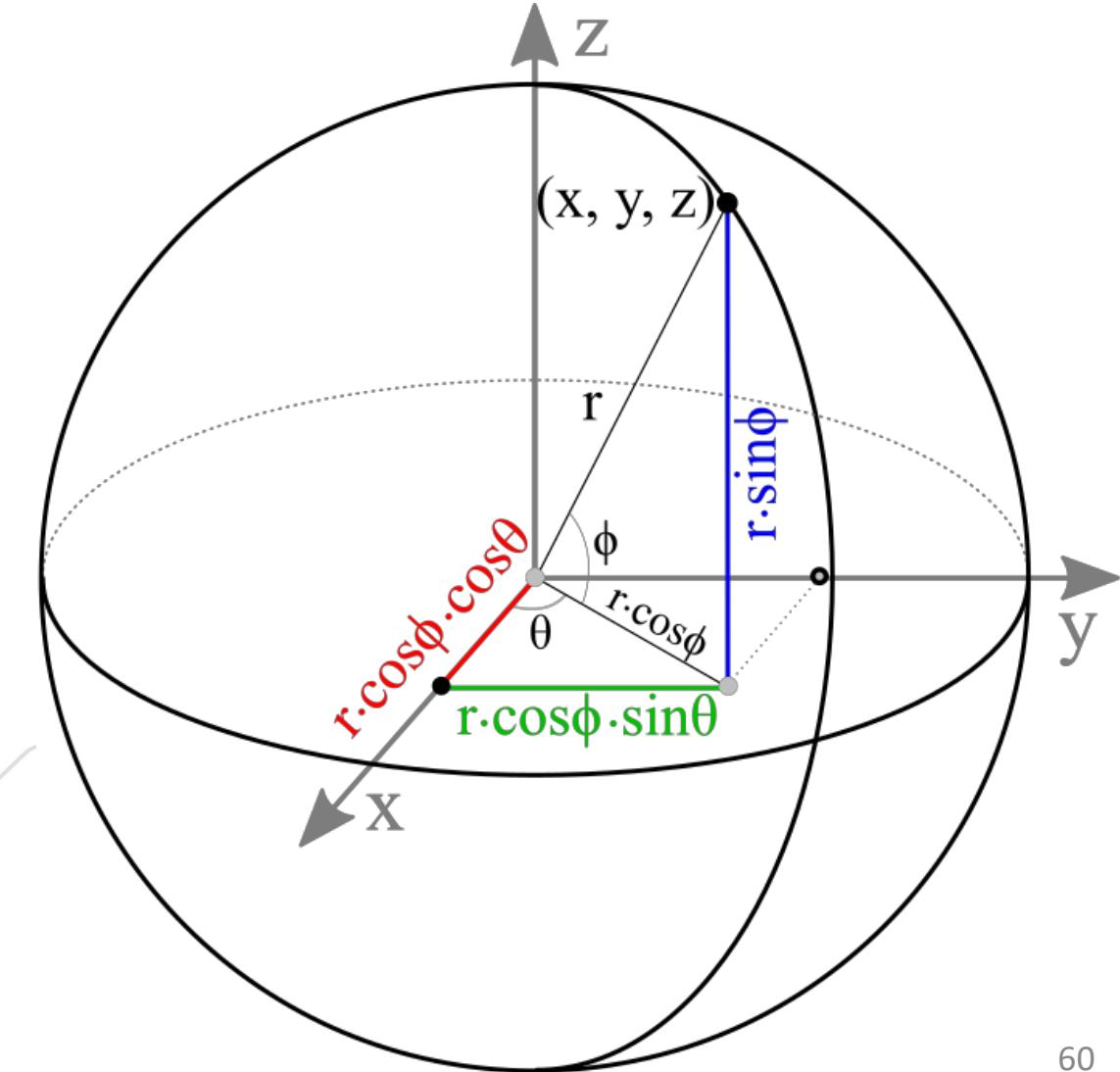
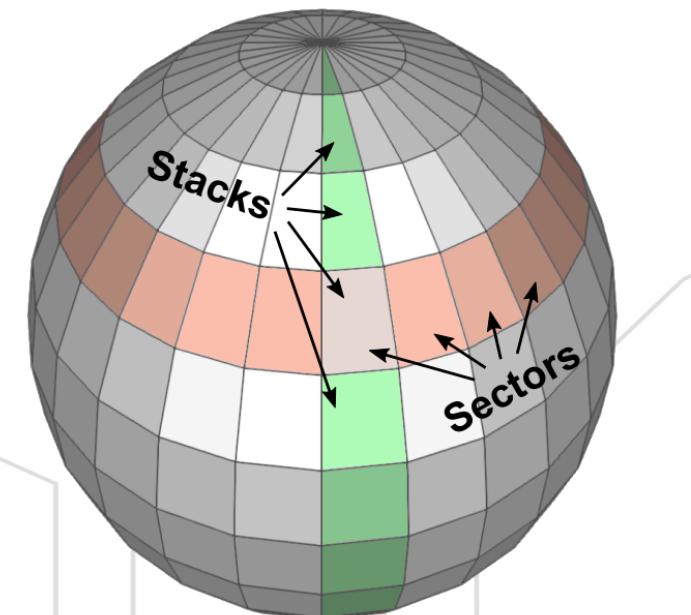
- 由粗到细：曲面细分 (subdivision)
- 由细到粗：simplification



- 如何产生一个光滑的球面?

- 通过旋转角指明球面上的顶点

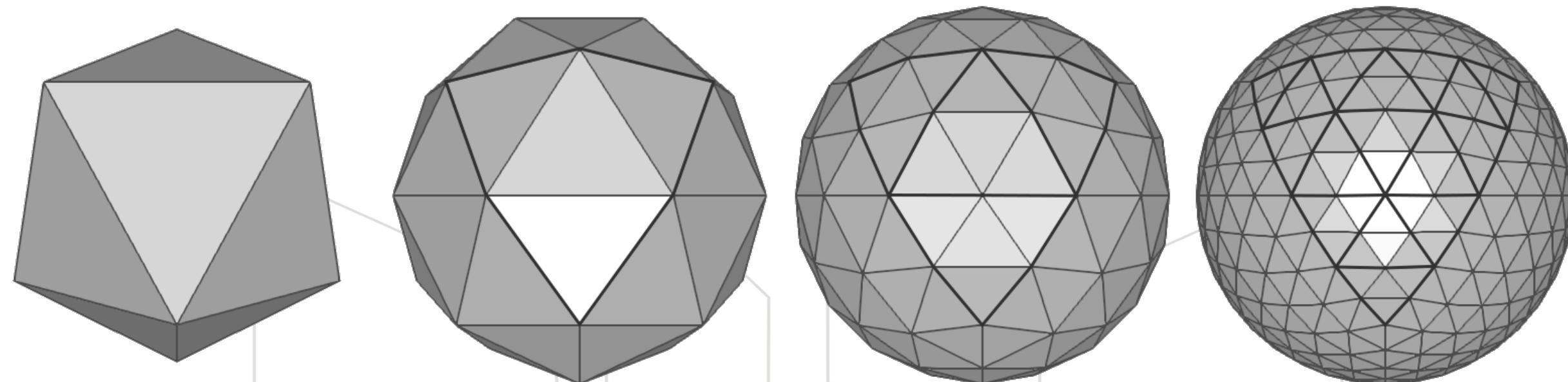
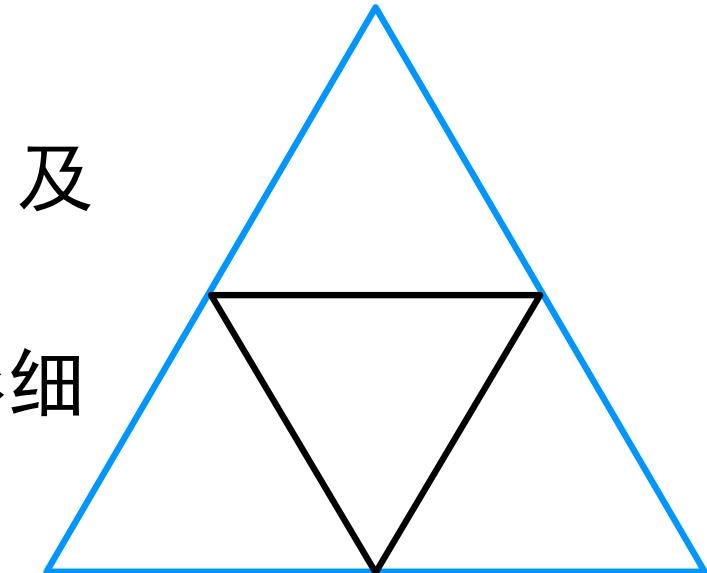
- 类似经纬度
 - $x = (r \cdot \cos \phi) \cdot \cos \theta$
 - $y = (r \cdot \cos \phi) \cdot \sin \theta$
 - $z = r \cdot \sin \theta$
 - 两层循环
 - 这个方法的问题是?



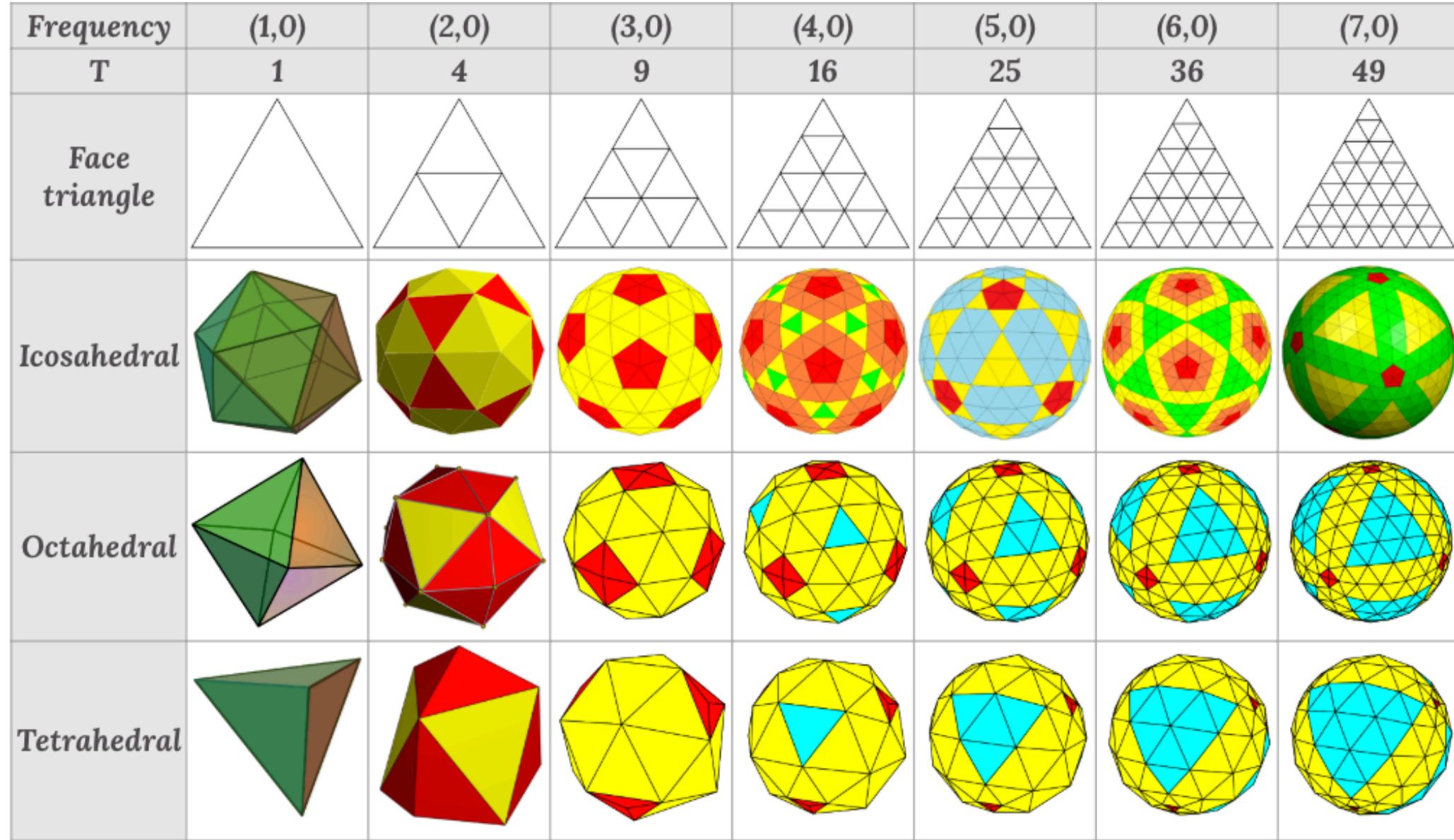
如何产生一个光滑的球面？

- 对正多面体进行递归细分

- 最常用于细分产生球面的为正四面体（tetrahedron）及正二十面体（icosahedron）
- 每次细分时，连接三角形的三边中点，将一个三角形细分为四个小三角形
- 由于新增加的点不在圆上，需要将点投影至圆面上

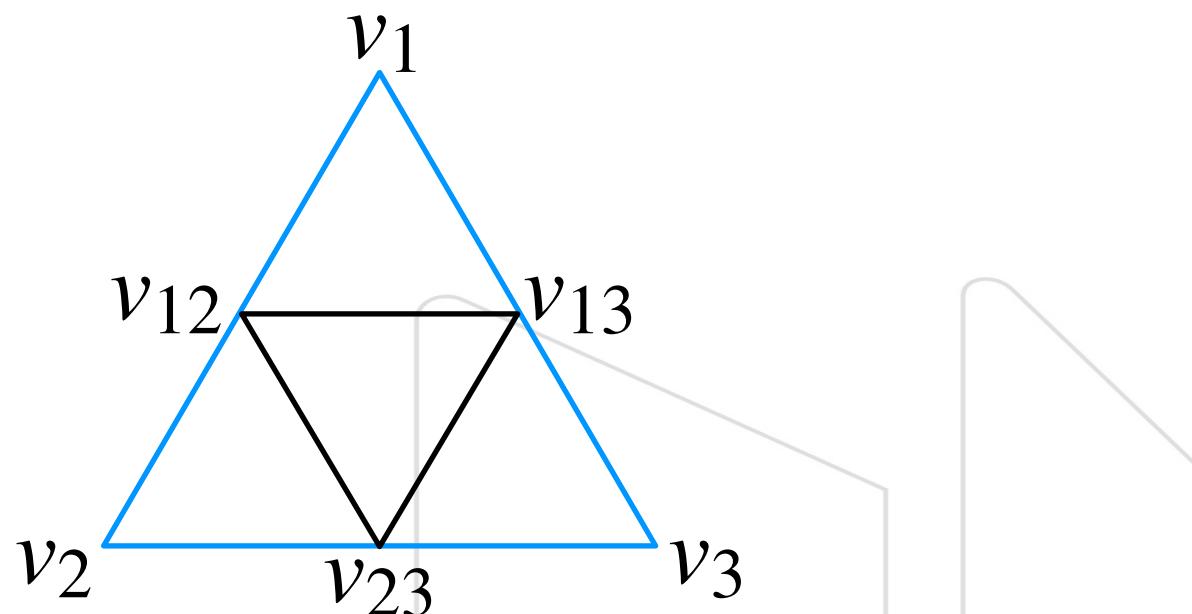


- 如何产生一个光滑的球面?
 - 对正多面体进行递归细分



对正多面体进行递归细分 OpenGL实现

```
vec3 computeHalfVertex(const vec& v1,
                      const vec& v2,
                      const float& radius)
{
    vec3 ret = v1+v2;
    ret *= radius/length(ret);
    return ret;
}
```



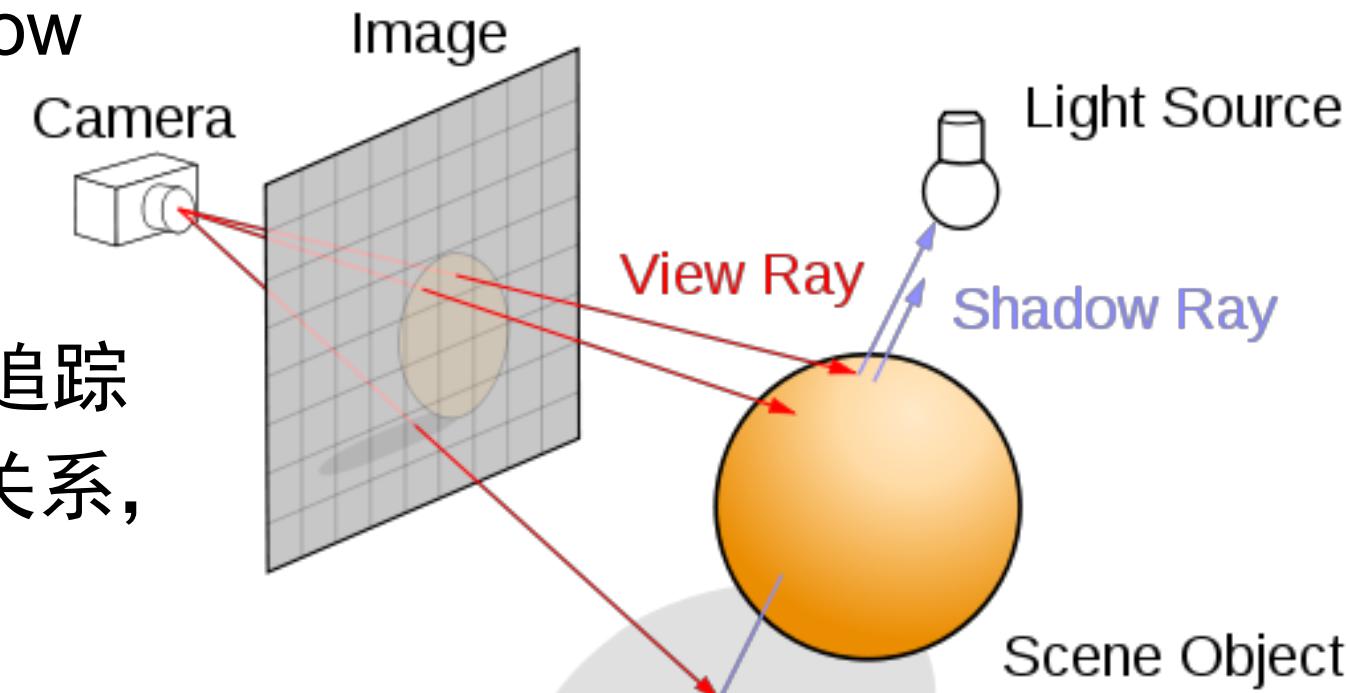
```
init_sphere(vertices);
// iterate all subdivision levels
for(int i = 0; i < subdivision; ++i){
    tmp_triangles.clear();
    // perform subdivision for each triangle
    for(int j = 0; j < triangles.size(); j += 3){
        // get 3 vertices of a triangle
        v1 = vertices[triangles[j].i];
        v2 = vertices[triangles[j].j];
        v3 = vertices[triangles[j].k];
        //add half vertices
        v12 = computeHalfVertex(v1, v2, radius);
        v13 = computeHalfVertex(v1, v3, radius);
        v23 = computeHalfVertex(v2, v3, radius);
        // add vertices to vertex array
        vertices.push_back(v12...v13...v23);
        // add indices of 4 new triangles
        //addTriangle: tmpTriangle.push_back()
        addTriangle(i1, i12, i13);
        addTriangle(i2, i12, i23);
        addTriangle(i3, i13, i23);
    }
    triangles = tmp_triangles;
```

- 常用于更具风格化的场景

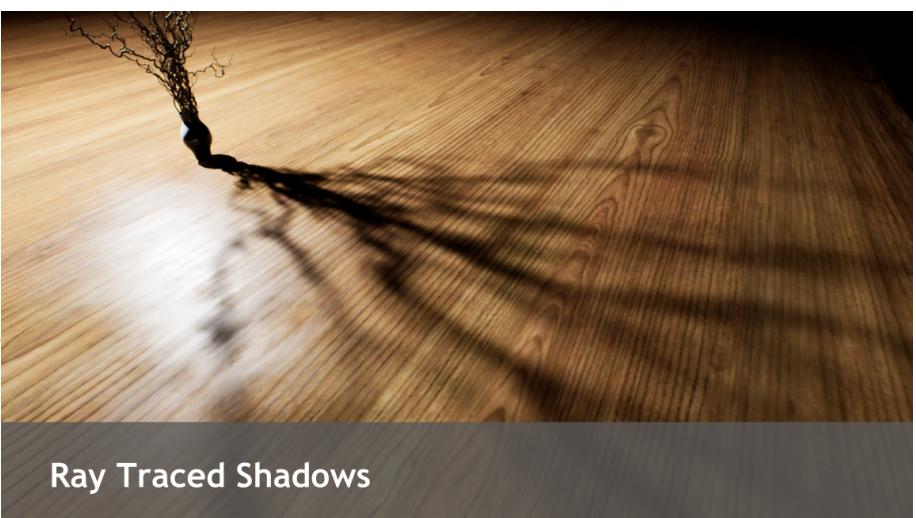
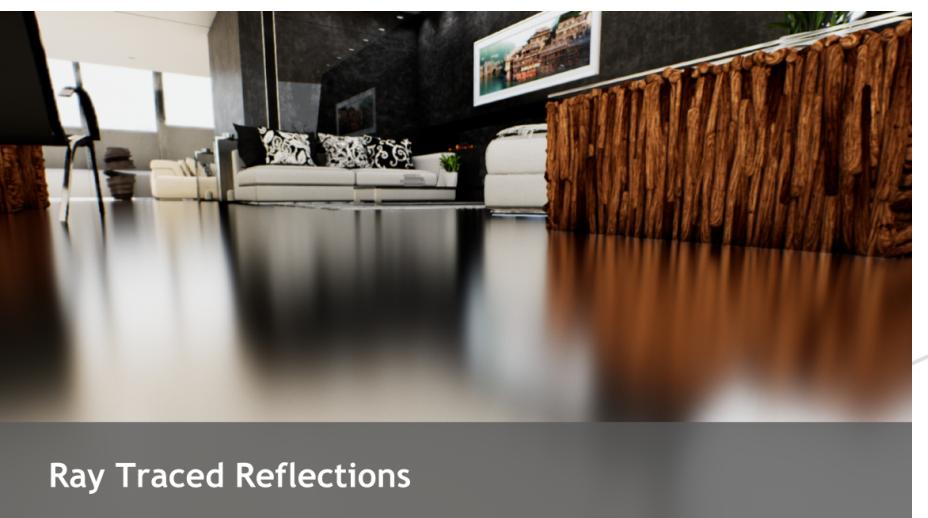


● 全局光照模型

- 局部光照模型无法产生具有真实感的光照细节
 - 间接光照, color bleeding, soft shadow
- 产生全局光照的方法有多种
 - Ray tracing: 从视角出发追踪光线
 - Photon mapping: 从光源光源出发追踪
 - Radiosity: 计算表面上小块之间的关系,求解每个小块的亮度

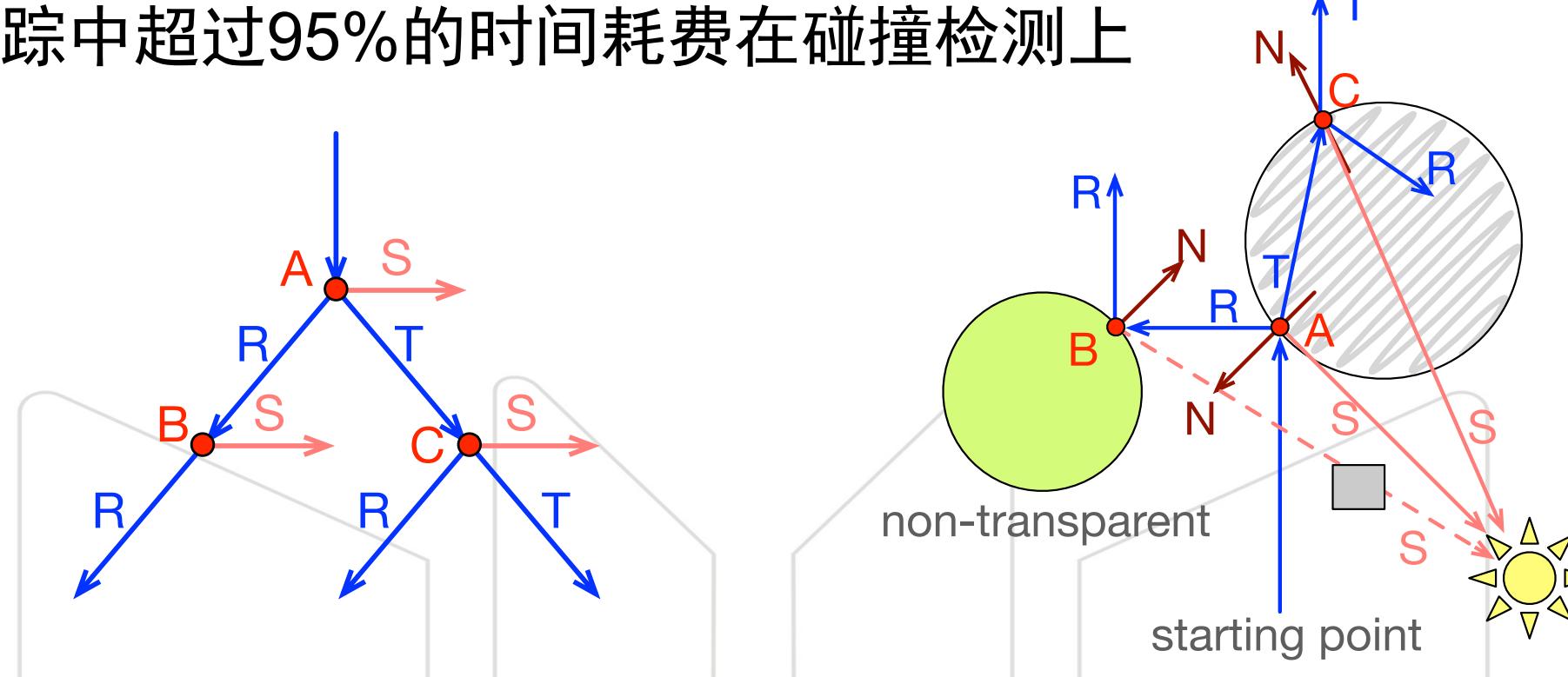


Ray tracing



Ray tracing

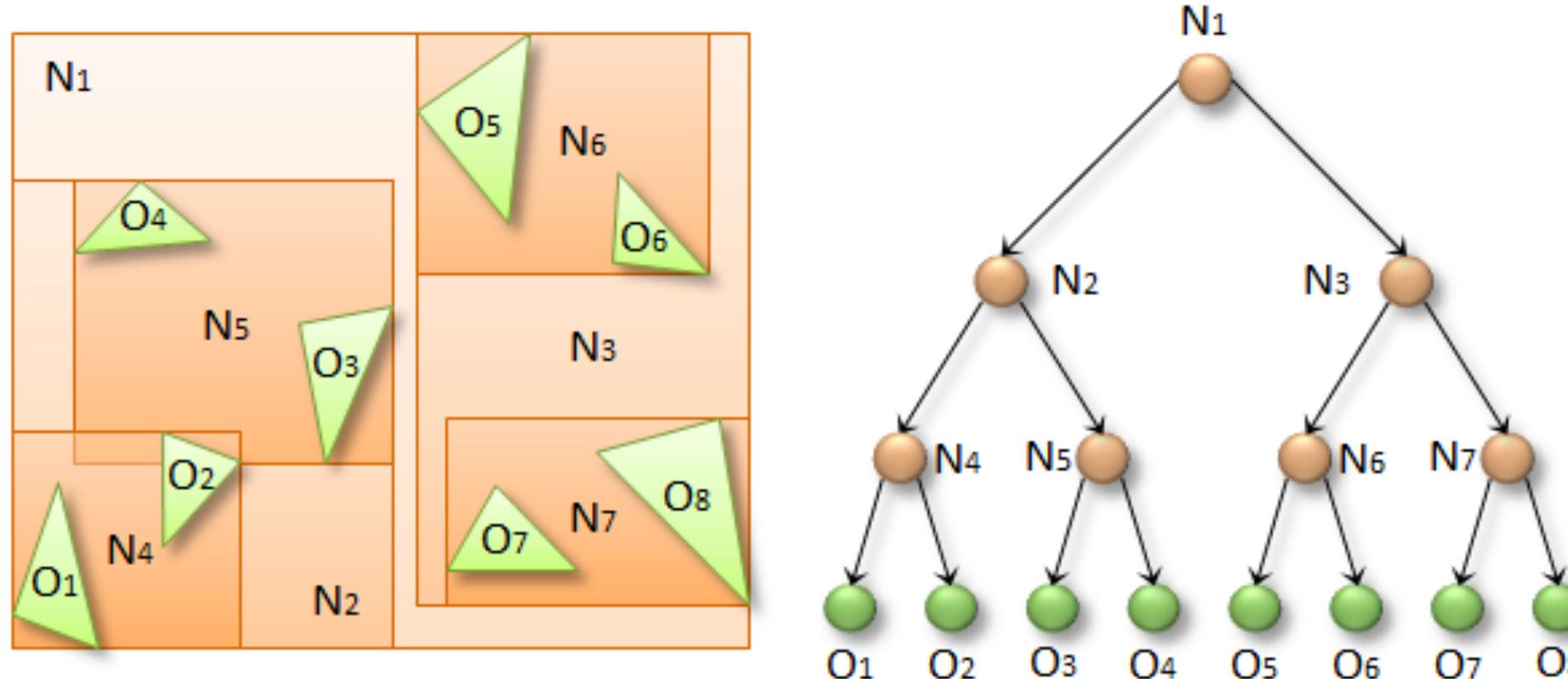
- 初始化时产生与图像像素相同数目的光线数目进行追踪
 - 当光线打在物体上时，产生三道光线进行进一步追踪
 - 反射光R，折射光T，以及阴影光S
- 光线追踪过程的关键为直线与几何物体的碰撞检测
 - 光线追踪中超过95%的时间耗费在碰撞检测上



碰撞检测

– bounding volume hierarchy (BVH)-tree

- 使用包围盒 (bounding volume) 将物体分配到树的节点中
- 在查找最近邻时，先判断与包围盒的距离



图片来自 <https://devblogs.nvidia.com/thinking-parallel-part-ii-tree-traversal-gpu/> 68

● 碰撞检测

- NVIDIA RTX平台上提供了硬件加速的ray tracing
 - 支持DirectX及Vulkan
 - <https://developer.nvidia.com/rtx/raytracing>



- 光照和着色是产生真实感的关键
 - 立体感, 光影
- 光源
 - 环境光、点光源、平行光源、聚光灯
- Phong反射模型
 - 环境光、漫反射、镜面反射
- OpenGL中的光照
 - 设置光源及材料参数
 - 着色模型: flat, Gouraud, Phong
- 其他
 - 曲面细分、非真实感着色、全局光照

Questions?

