第5讲：着色

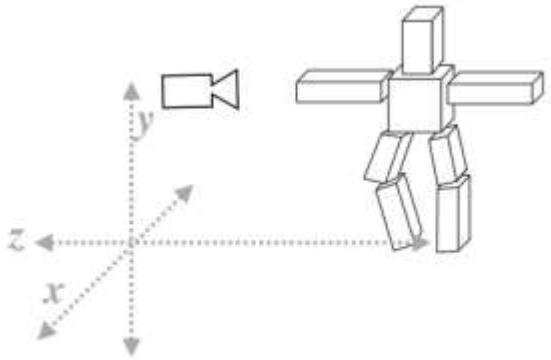
上次课程内容

- 光栅化
  - ➢ 什么是采样?
  - ➢ 什么是走样?
  - ➢ 反走样
  - ➢ 遮挡/可见性
  - ✓ 画家算法
  - ✓ Z-buffer算法

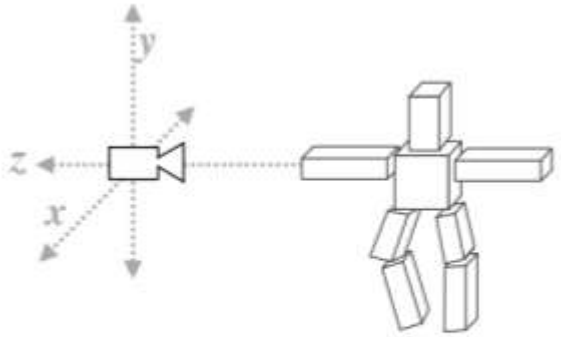
本次课程内容

- 什么是着色（Shading）？
- 一个简单的着色模型：Blinn-Phong反射模型
- 着色频率
- 图形管线（Graphics Pipeline）
- 纹理映射（Texture Mapping）

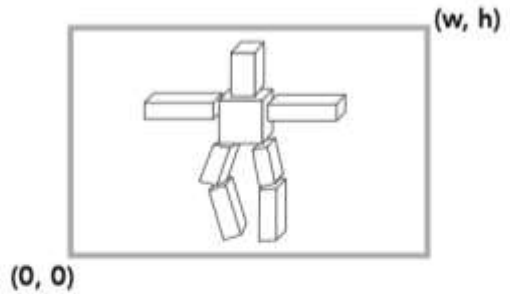
目前已学习内容



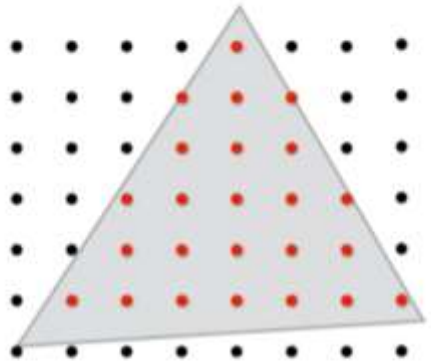
Position objects and the camera in the world



Compute position of objects relative to the camera



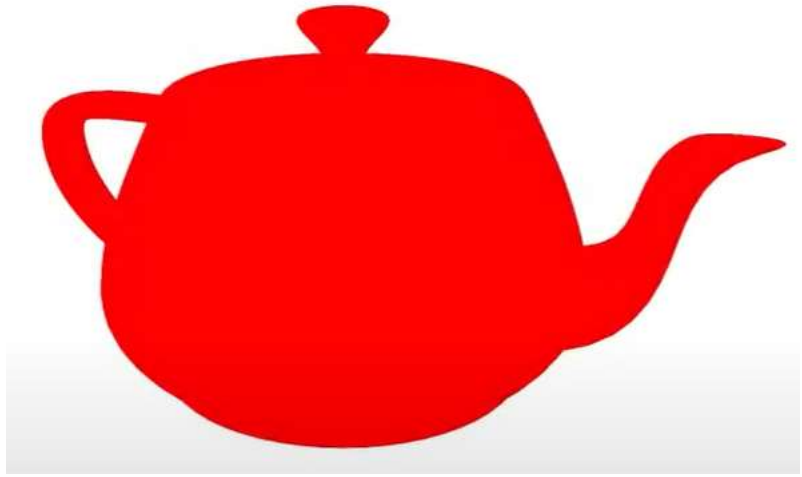
Project objects onto the screen



Sample triangle coverage

?

着色



着色

Credit: Bertrand Benoit. “Sweet Feast,” 2009. [Blender /VRay]

什么是着色？

- 在韦氏词典中

shad·ing, ['ʃeɪdɪŋ], noun
The darkening or coloring of an illustration or diagram with parallel lines or a block of color.

- 在计算机图形学中

The process of applying a material to an object.

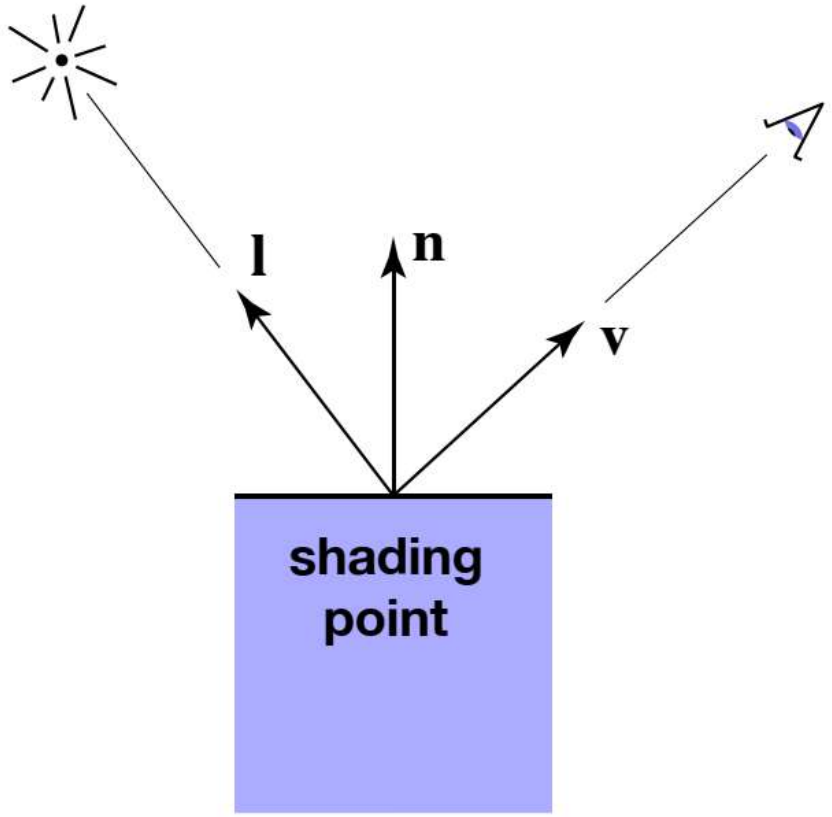
什么是着色？

Photo credit: Jessica Andrews, flickr

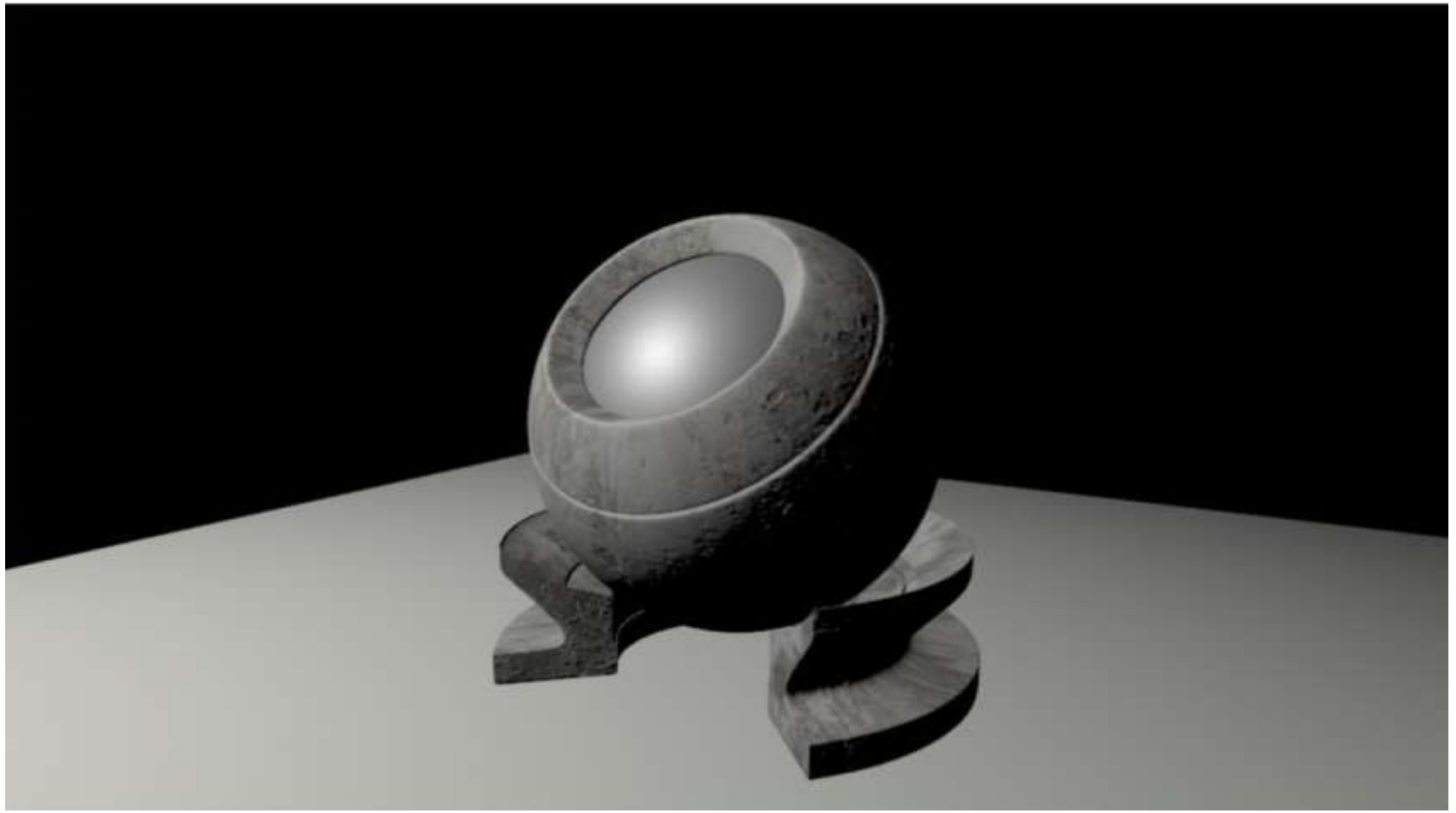
着色是局部的

- 在特定着色点计算反射到相机的光

- 输入：
  - 观察方向v
  - 表面法向n
  - 光线方向l
  - 表面参数（颜色、光泽度等等）

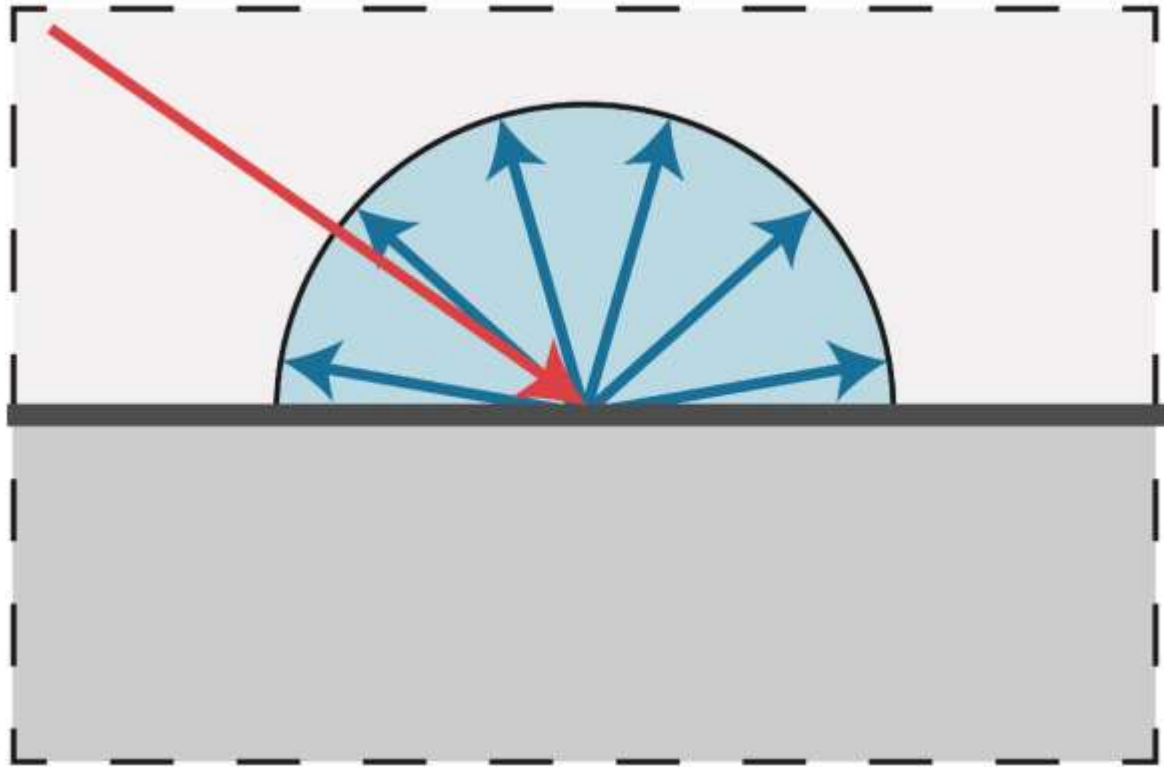


shading≠shadow



漫反射（Diffuse Reflection）

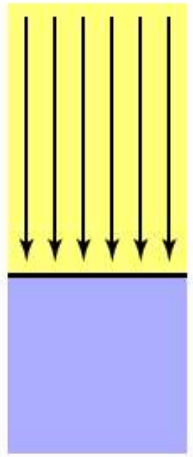
- 光向各个方向均匀散射
  - 所有观察方向看到的表面颜色是一致的



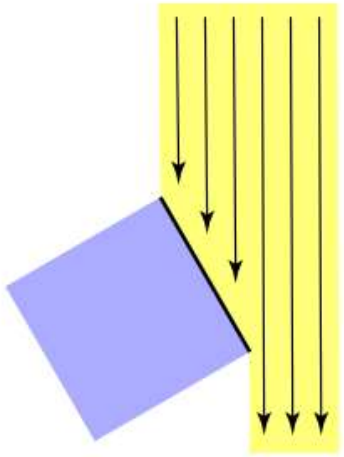
漫反射

Q:有多少光/能量被接收到？

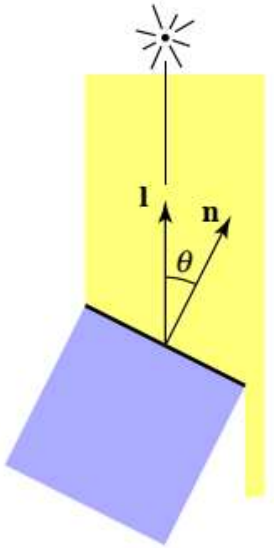
- 兰伯特余弦定理（Lambert’s cosine law）



Top face of cube receives a certain amount of light

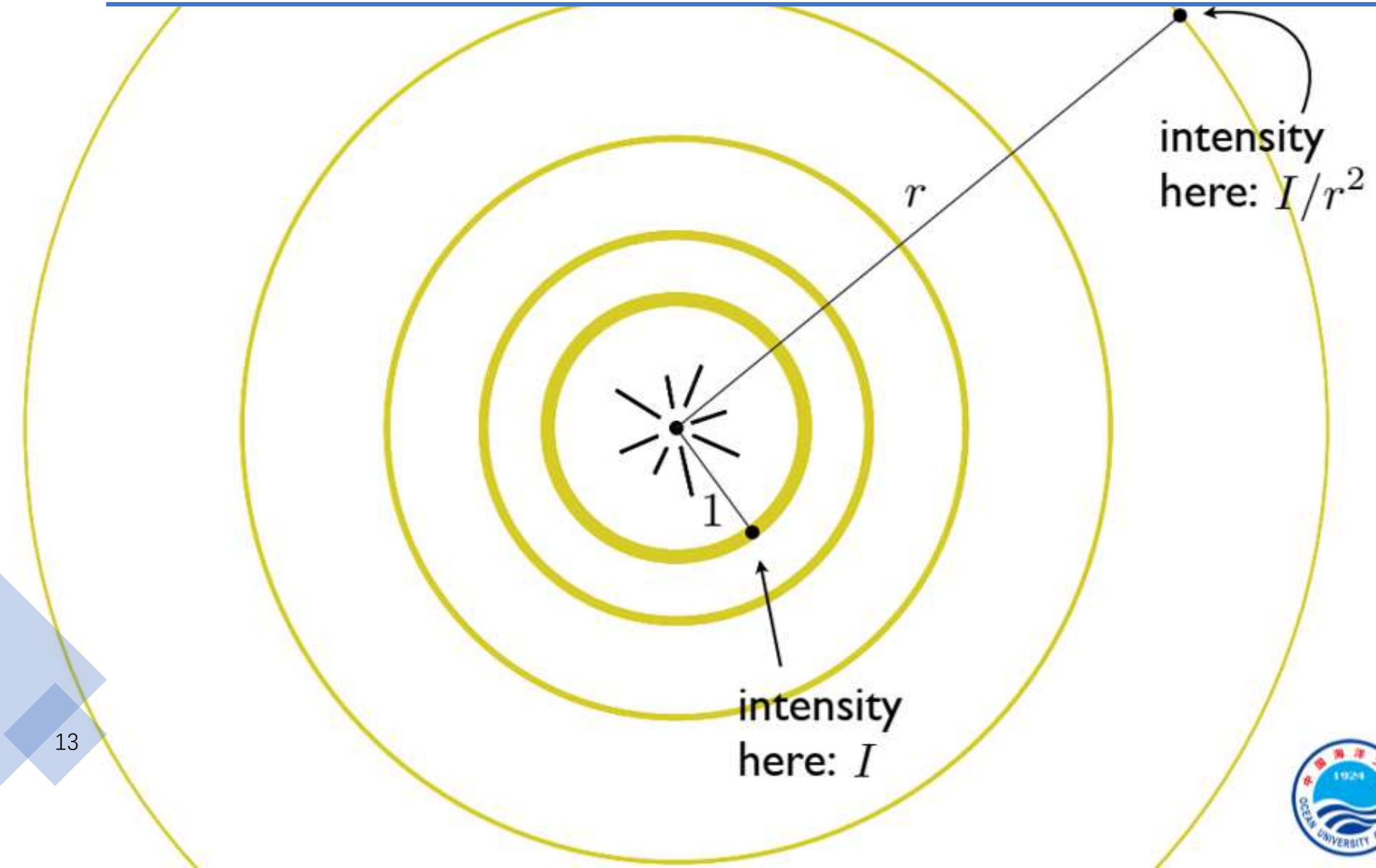


Top face of 60° rotated cube intercepts half the light



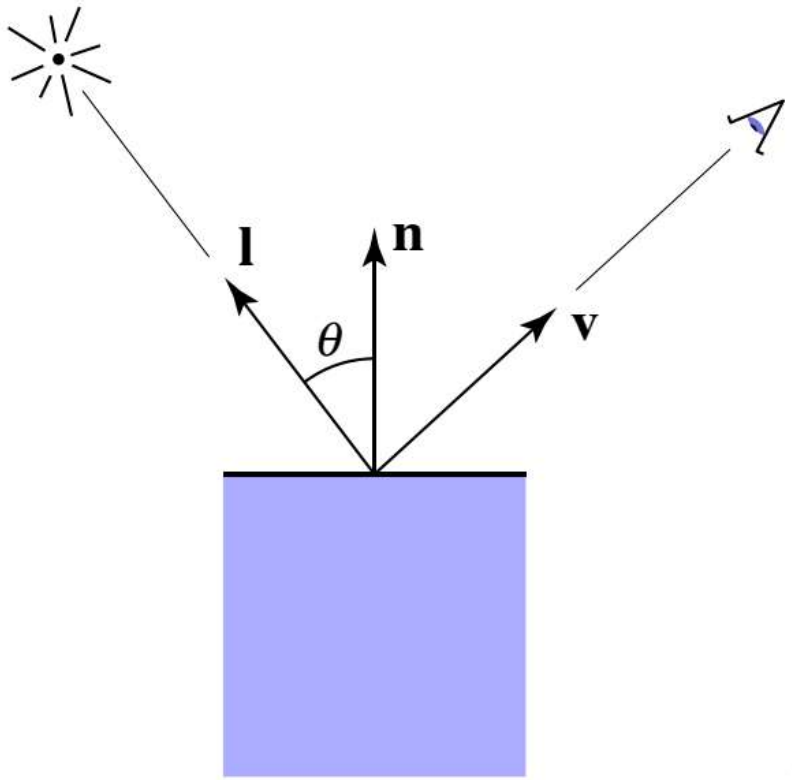
In general, light per unit area is proportional to $\cos\theta = l \cdot n$

光的衰减



Blinn-Phong反射模型：漫反射(Diffuse)

- 着色与观察方向无关



energy arrived
at the shading point

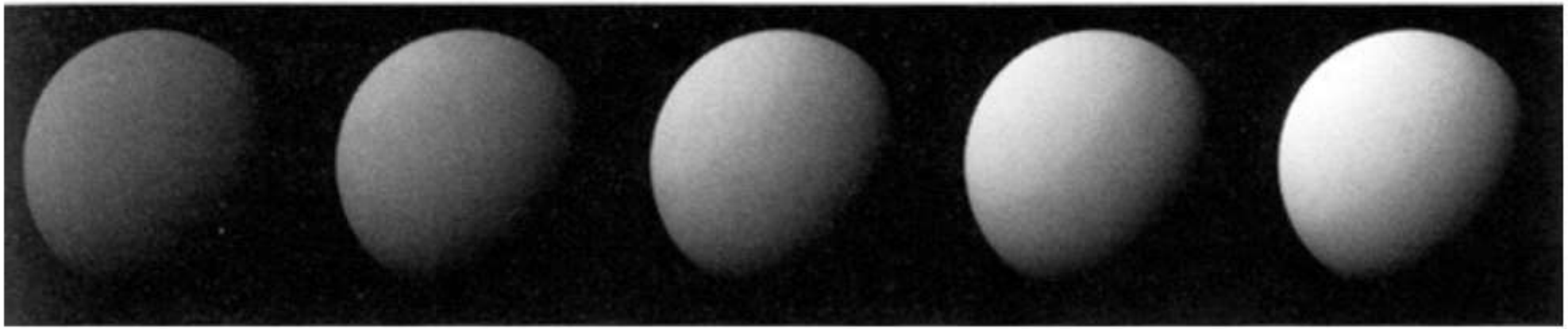
$$L_d = k_d\,(I/r^2)\,\max(0, \mathbf{n}\cdot\mathbf{l})$$

diffusely
reflected light

diffuse
coefficient
(color)

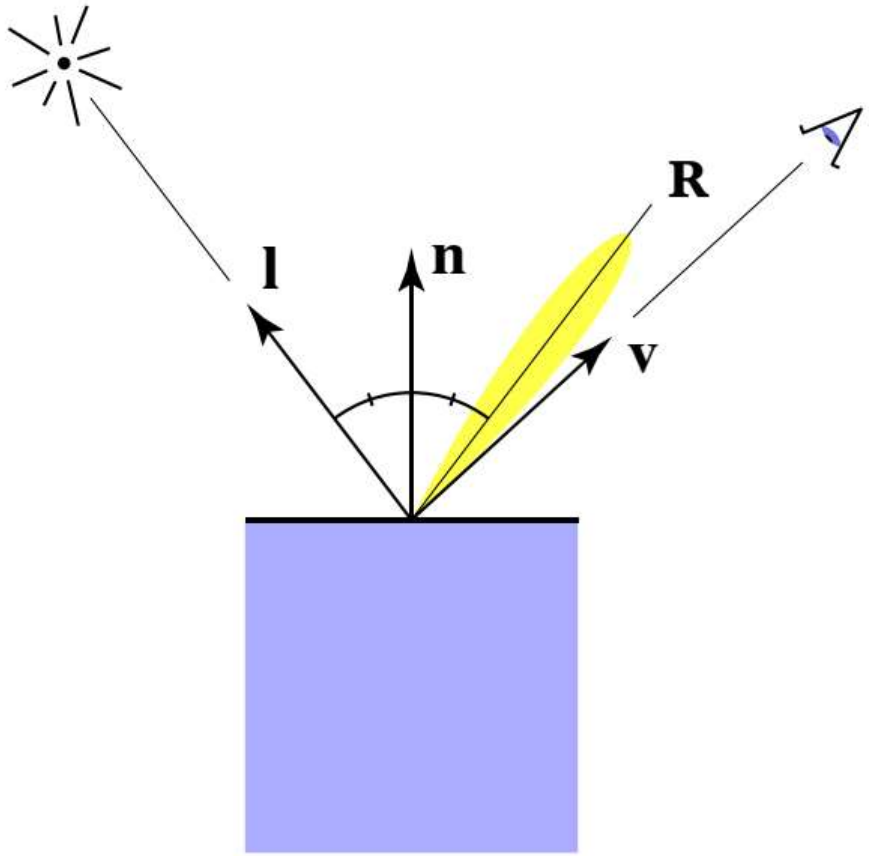
energy received
by the shading point

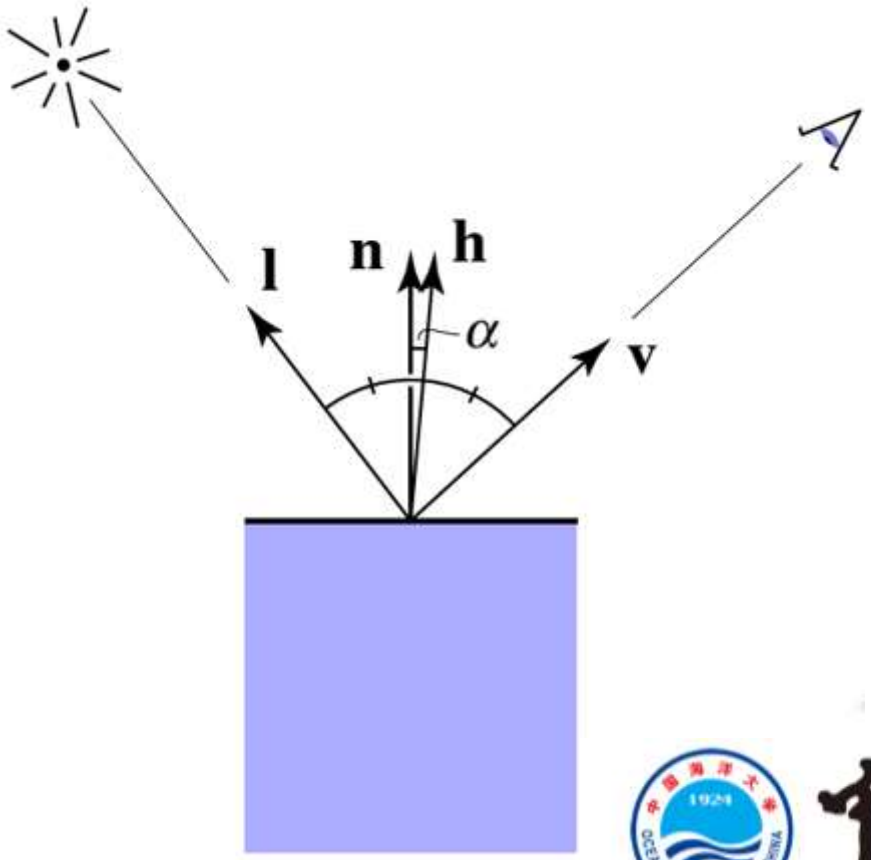
Blinn-Phong反射模型：漫反射



Blinn-Phong反射模型：镜面高光(Specular)

- 强度取决于观察方向：近镜面反射方向更加明亮
- v接近镜面反射方向等价于半程向量接近法线方向

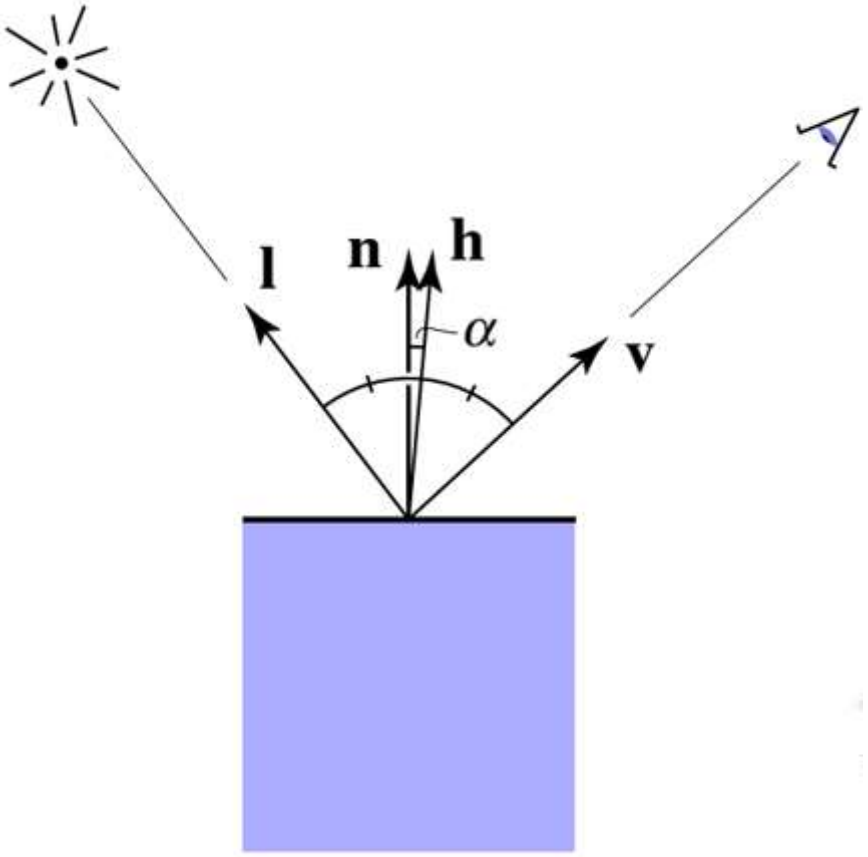




Blinn-Phong反射模型：镜面高光

Q:半程向量如何计算？

Q:如何衡量“接近”程度？



$$\mathbf{h} = \text{bisector}(\mathbf{v}, \mathbf{l})$$

(半程向量)

$$= \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|}$$

$$L_s = k_s \left(I/r^2\right) \max(0, \cos\alpha)^p$$

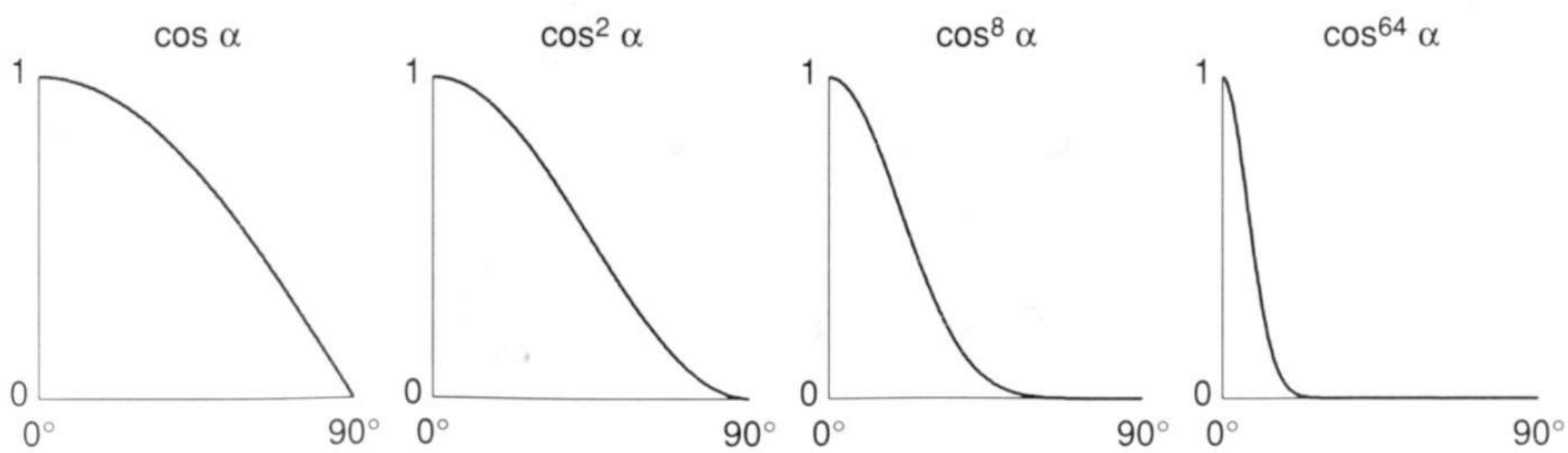
$$= k_s \left(I/r^2\right) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

specularly reflected light

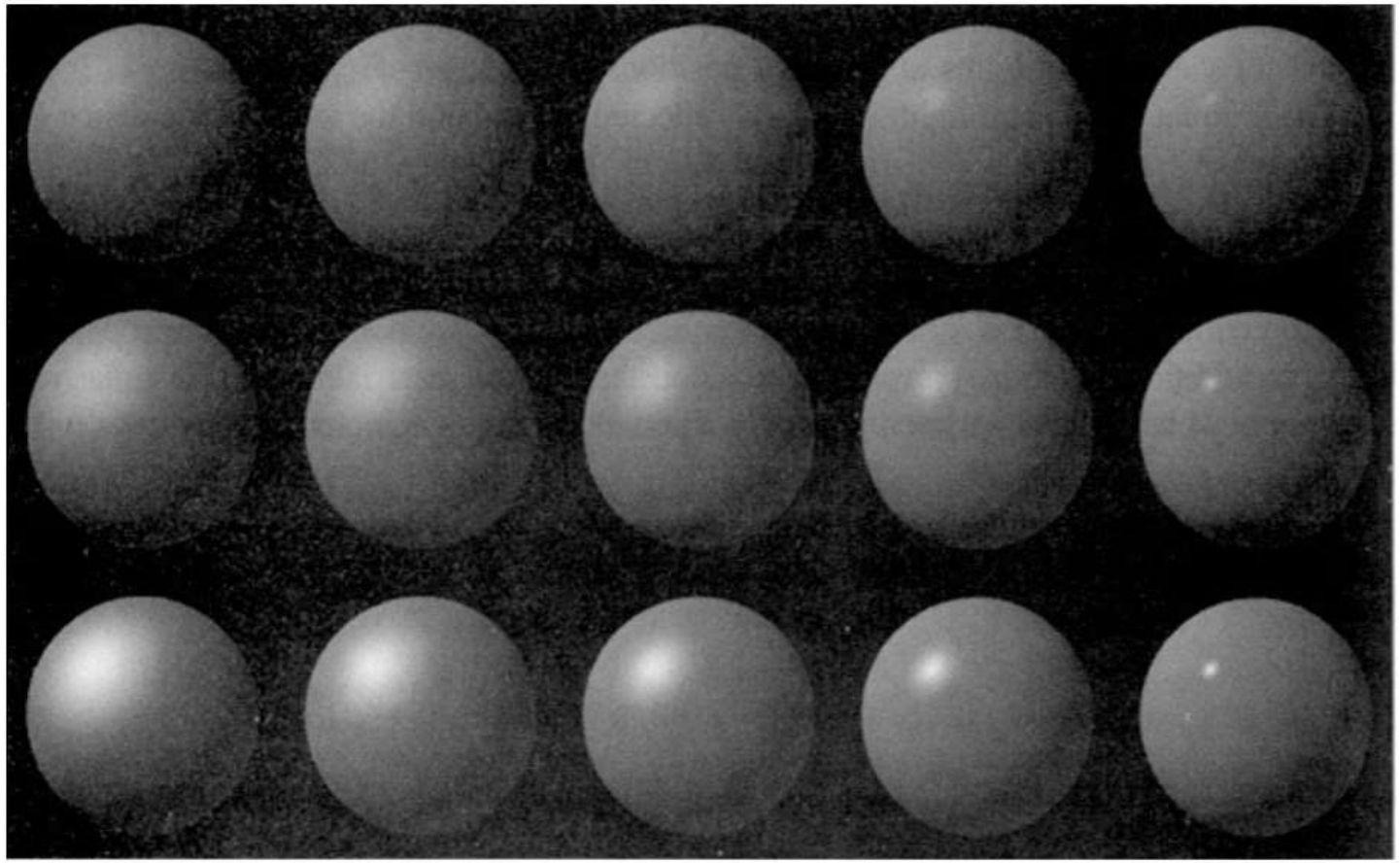
specular coefficient

余弦幂图

Q：随着p增大，余弦幂图如何变化？



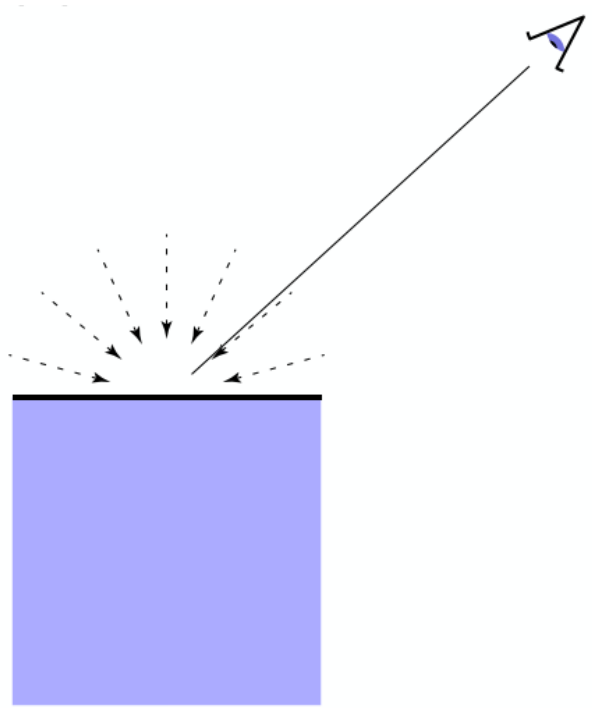
Blinn-Phong反射模型：镜面高光



Note: showing
Ld + Ls together

Blinn-Phong反射模型：环境光（Ambient）

- 着色不取决于任何其他因素
  - 添加常量颜色
  - 这是一种近似，并不真实

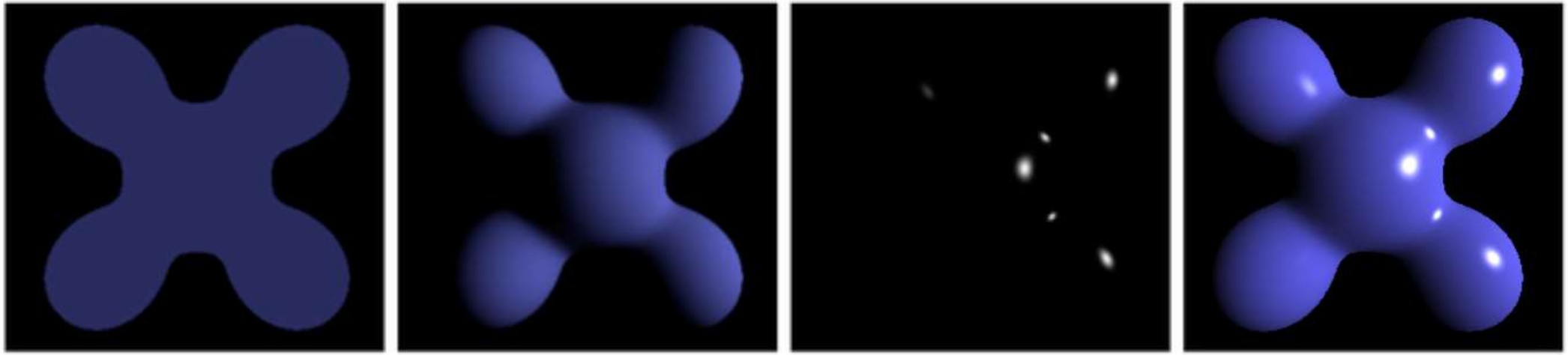


$$L_a = k_a I_a$$

ambient coefficient

reflected ambient light

Blinn-Phong反射模型



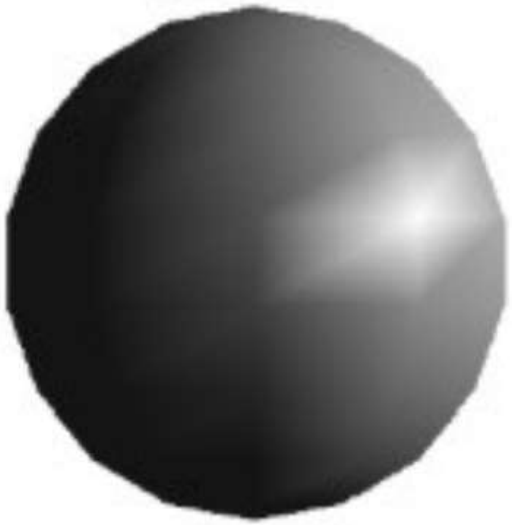
Ambient + Diffuse + Specular = Blinn-Phong Reflection

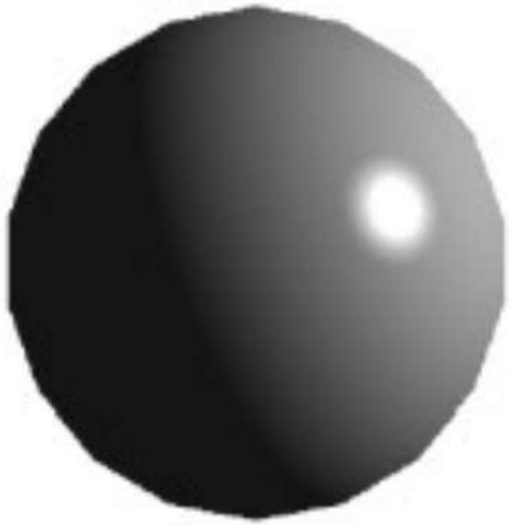
$$L = L_a + L_d + L_s$$
$$= k_a I_a + k_d (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

Q&A

着色频率

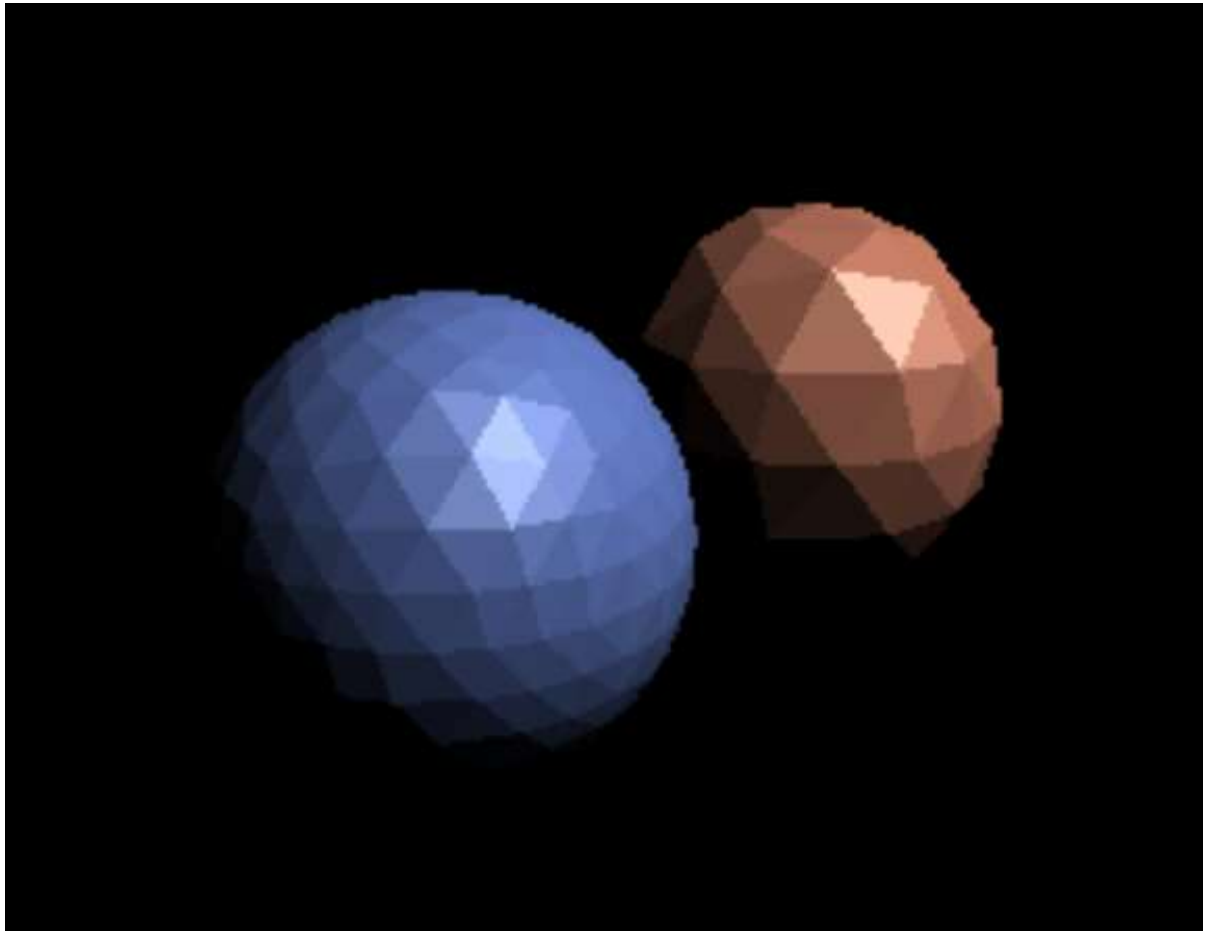
Q:什么导致了着色结果的不同?





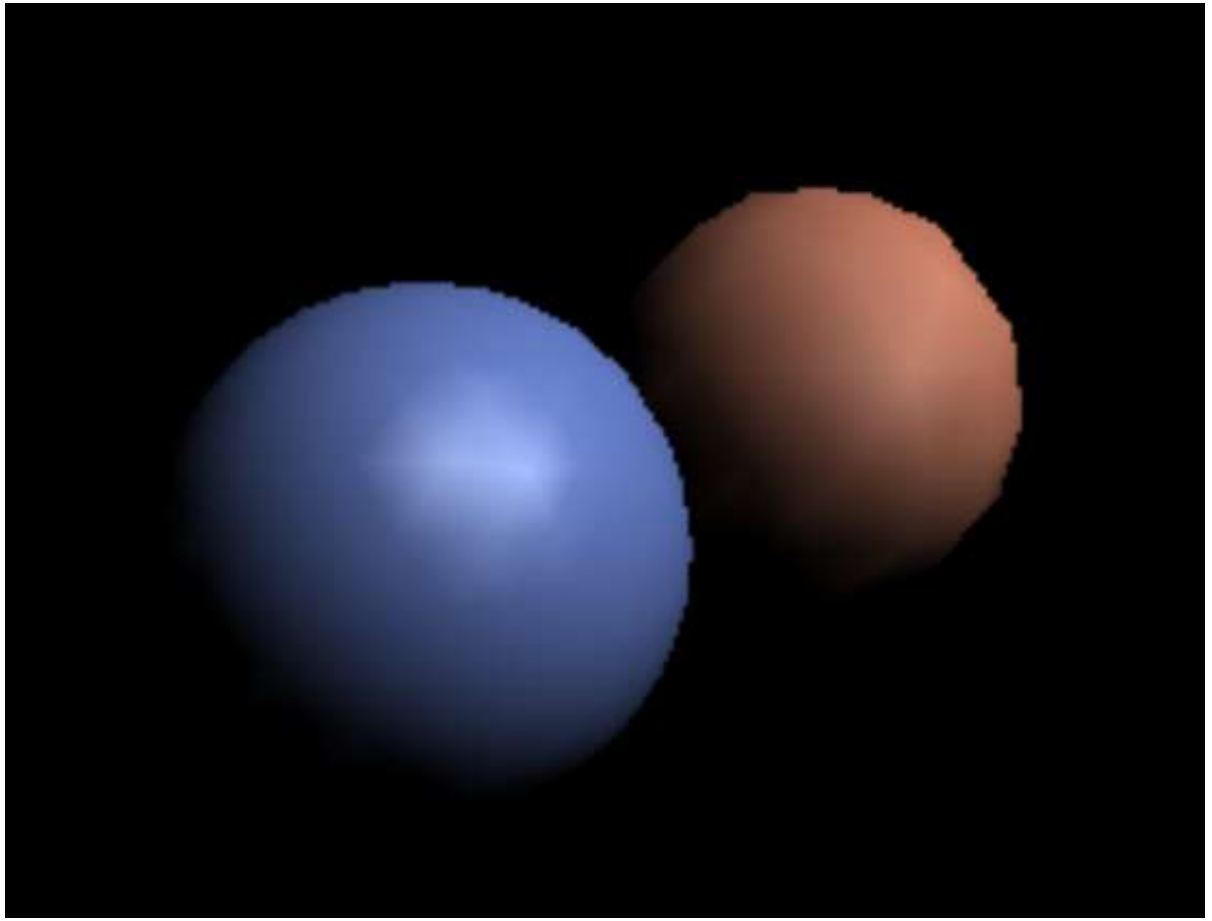
对每一个三角形着色

- Flat shading
  - 三角形是平面的：得到一个面的法向量
  - 不适合光滑的表面



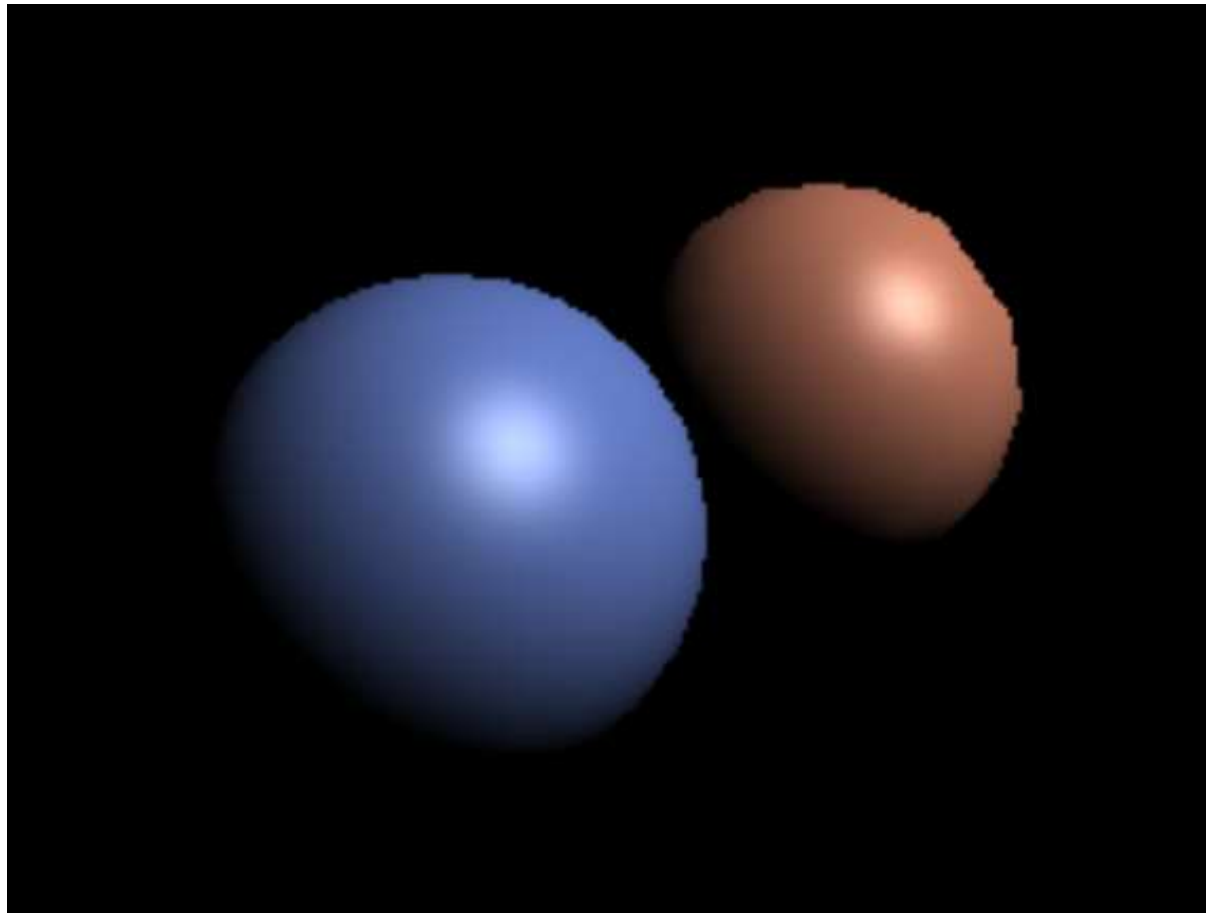
对每一个顶点着色

- Gouraud shading
  - 三角形的顶点携带颜色信息
  - 每一个顶点上有一个法向量（如何计算？）

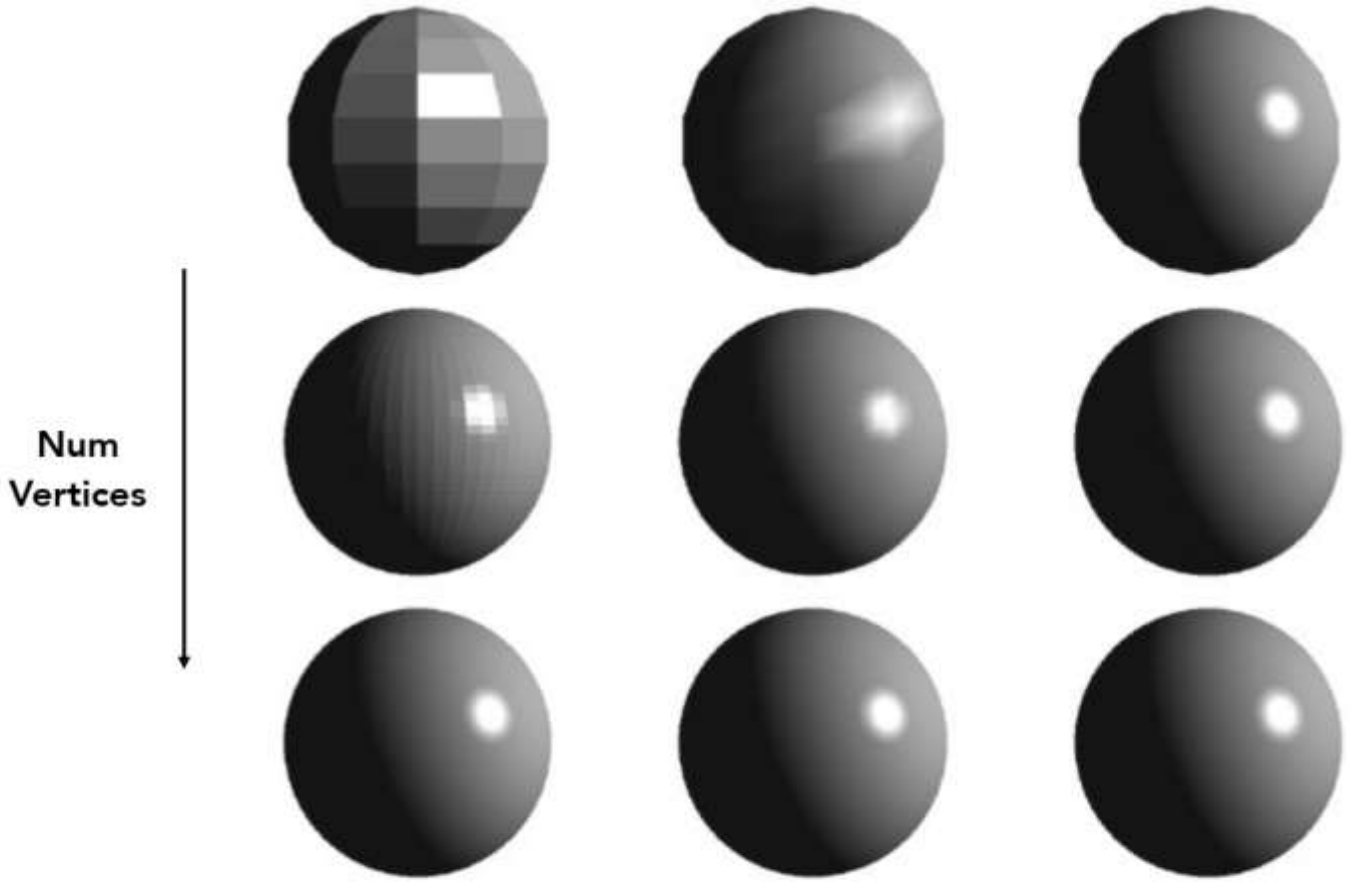


对每一个像素着色

- Phong shading
  - 插值得到法向量
  - 在每个像素上计算着色模型
  - 与Blinn-Phong反射模型区别开

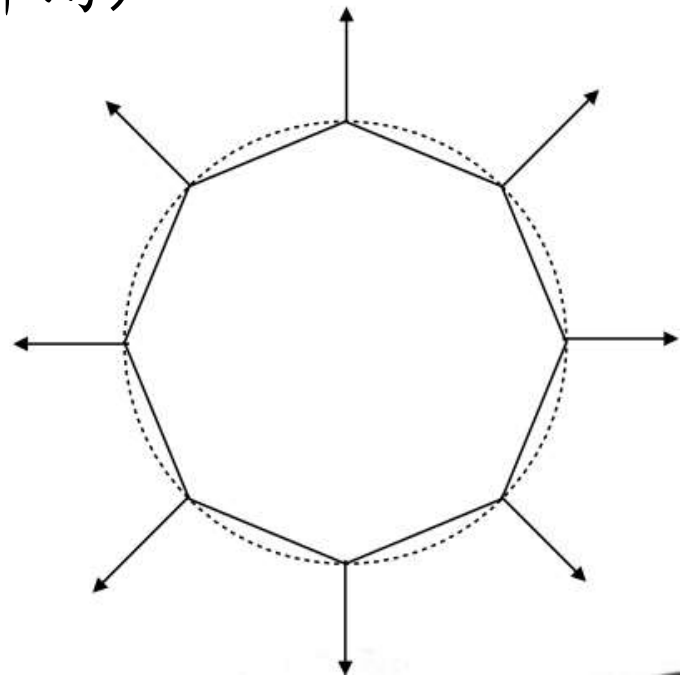


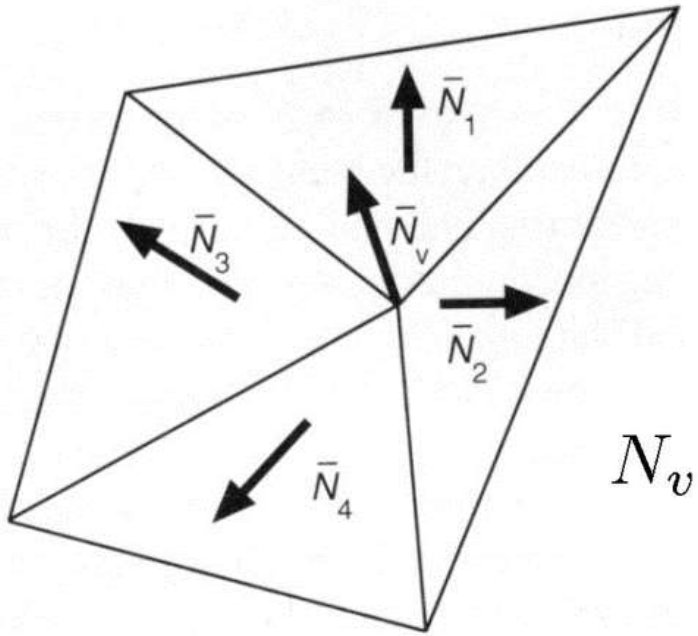
着色频率：面、顶点、像素



定义每个顶点上的法向量

- 最好是从想要表示的几何体获取顶点法向（例如：球体）
- 否则，需要从三角形面获取顶点法向（例如：利用环绕顶点的面法向取平均）

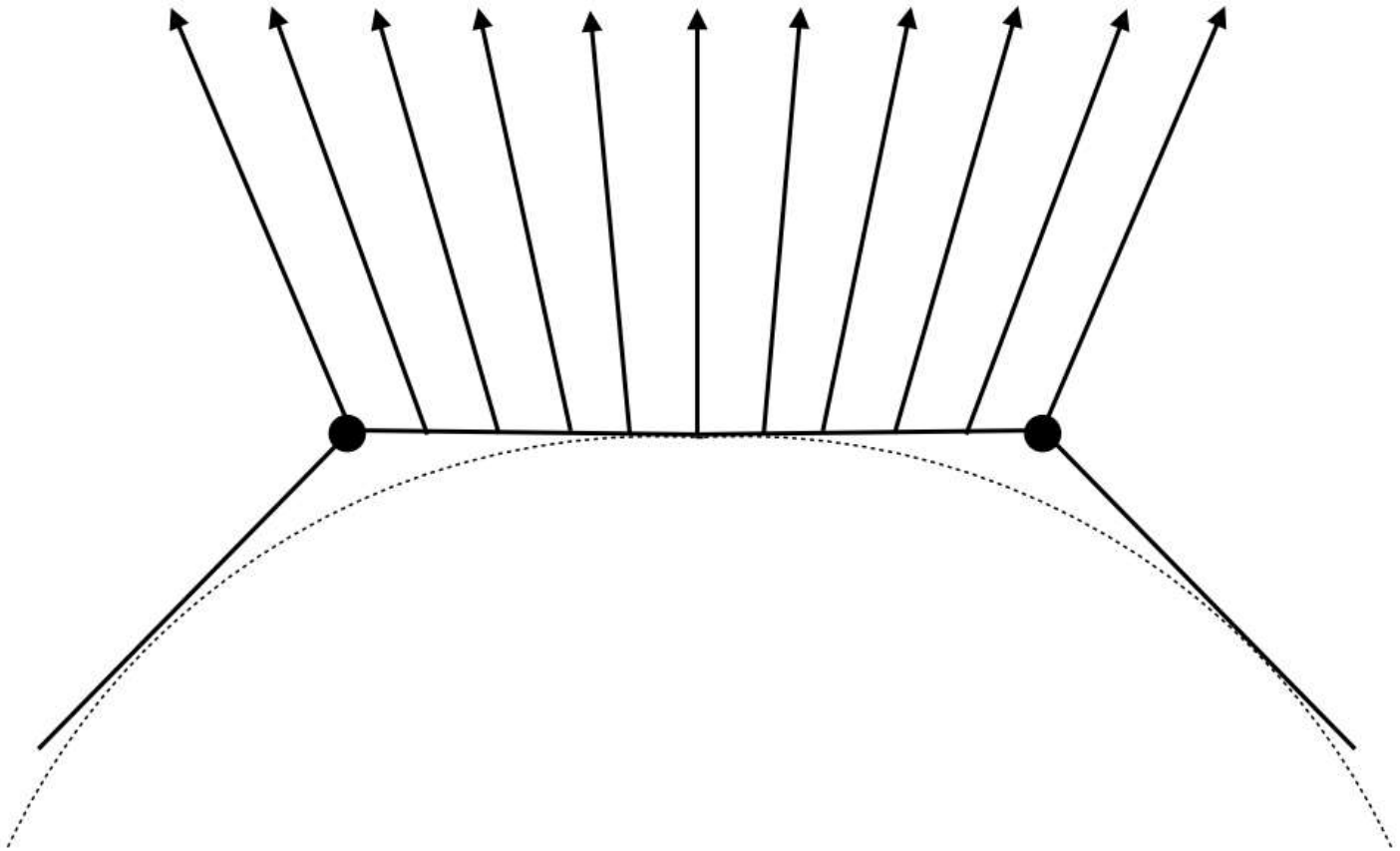




$$N_v = \frac{\sum_i N_i}{\|\sum_i N_i\|}$$

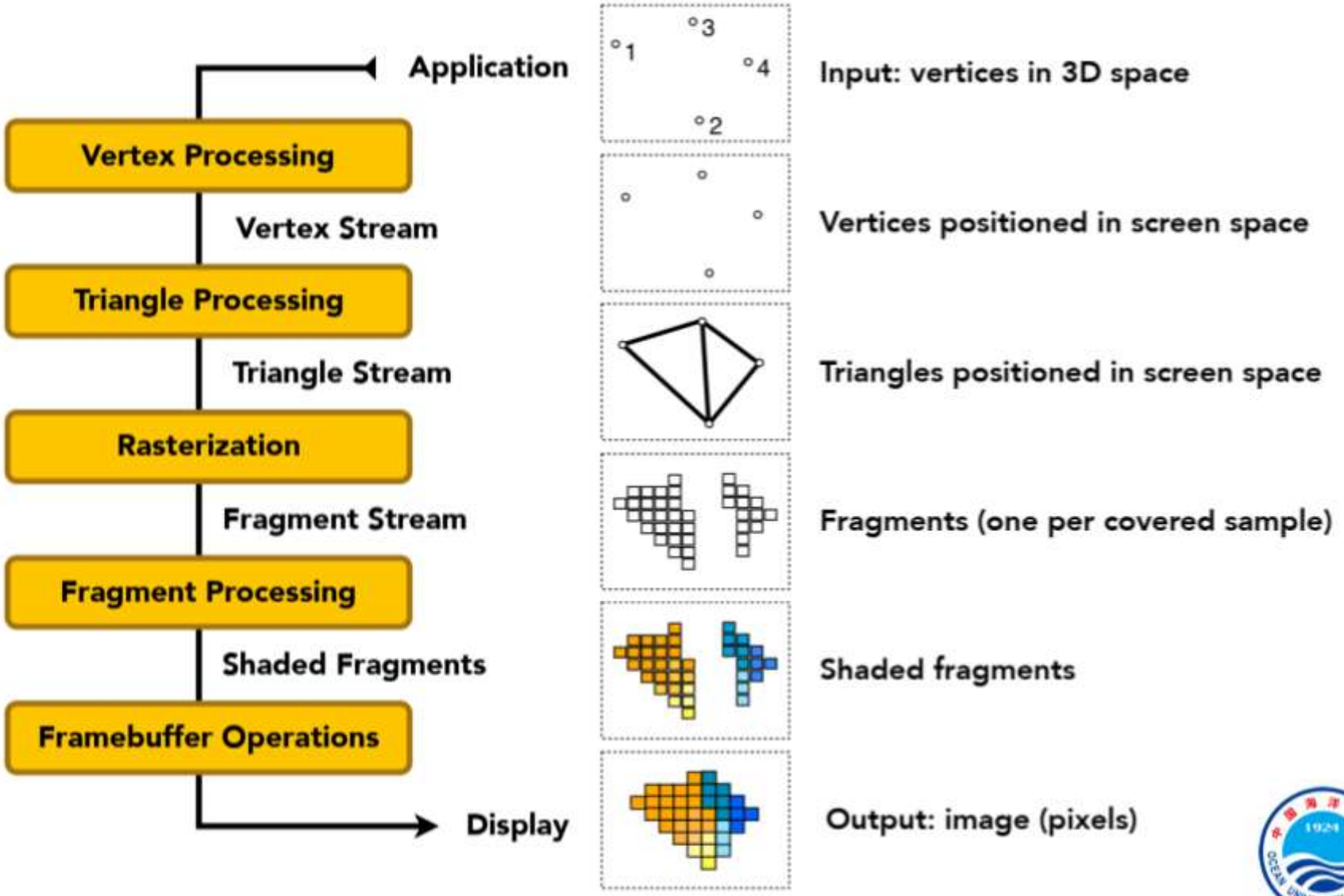
定义每个像素上的法向量

- 利用顶点法向的重心插值得到（注意插值结果的归一化）

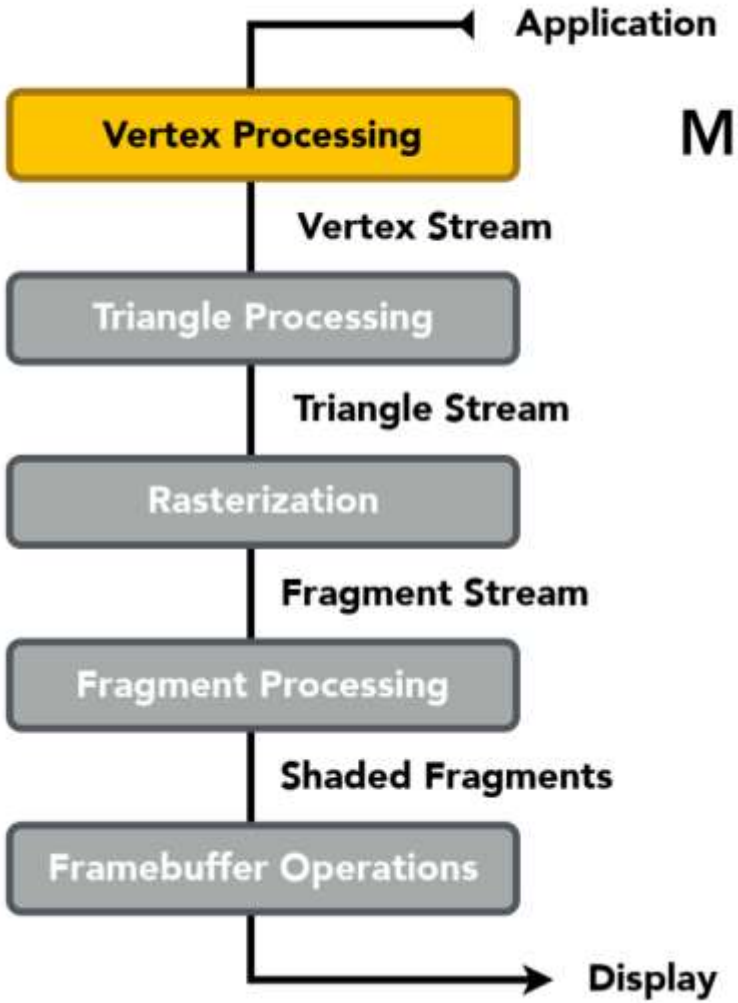


Q&A

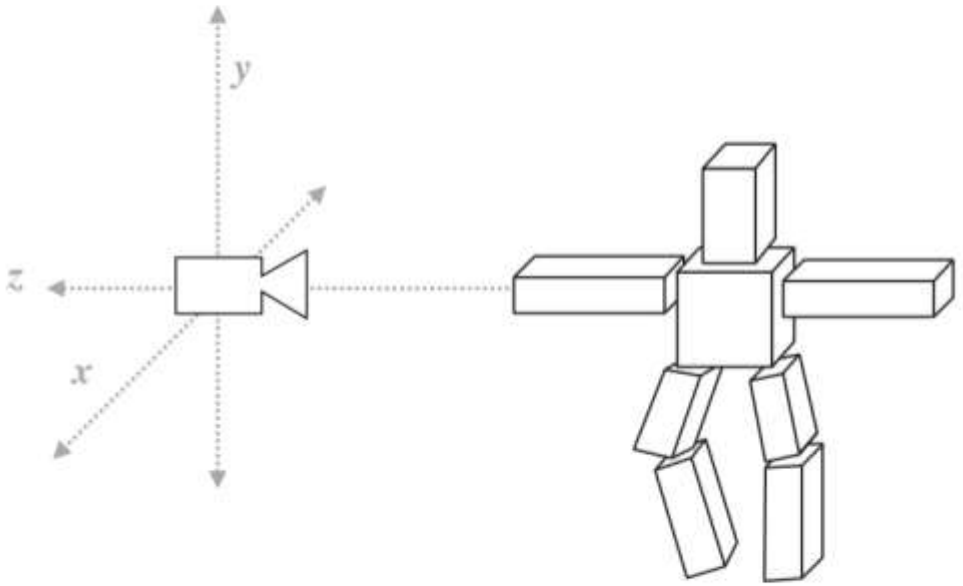
图形（实时渲染）管线



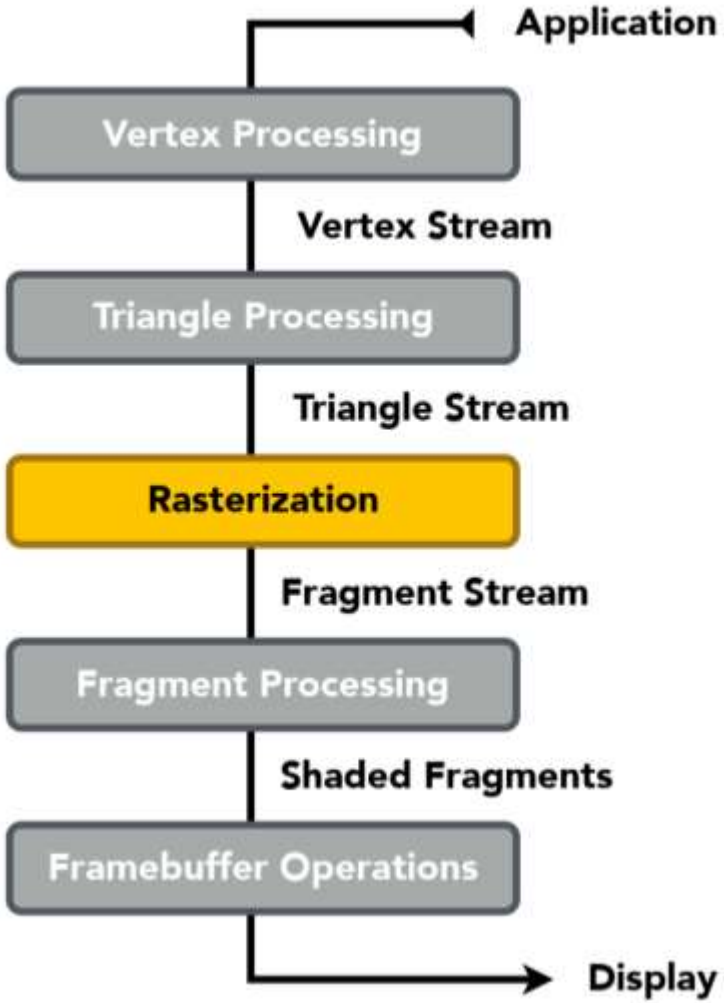
图形管线

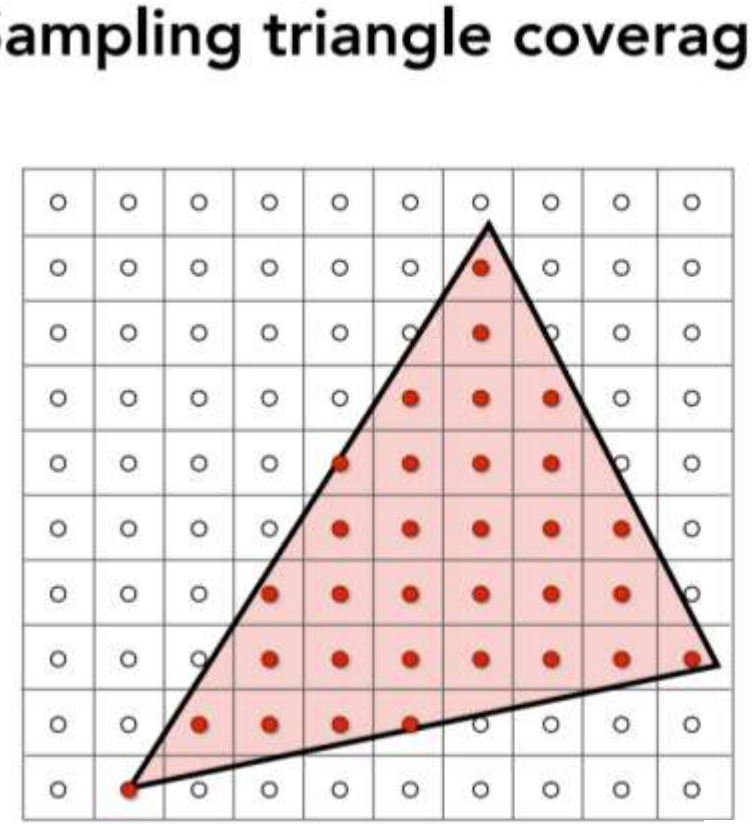


Model, View, Projection transforms

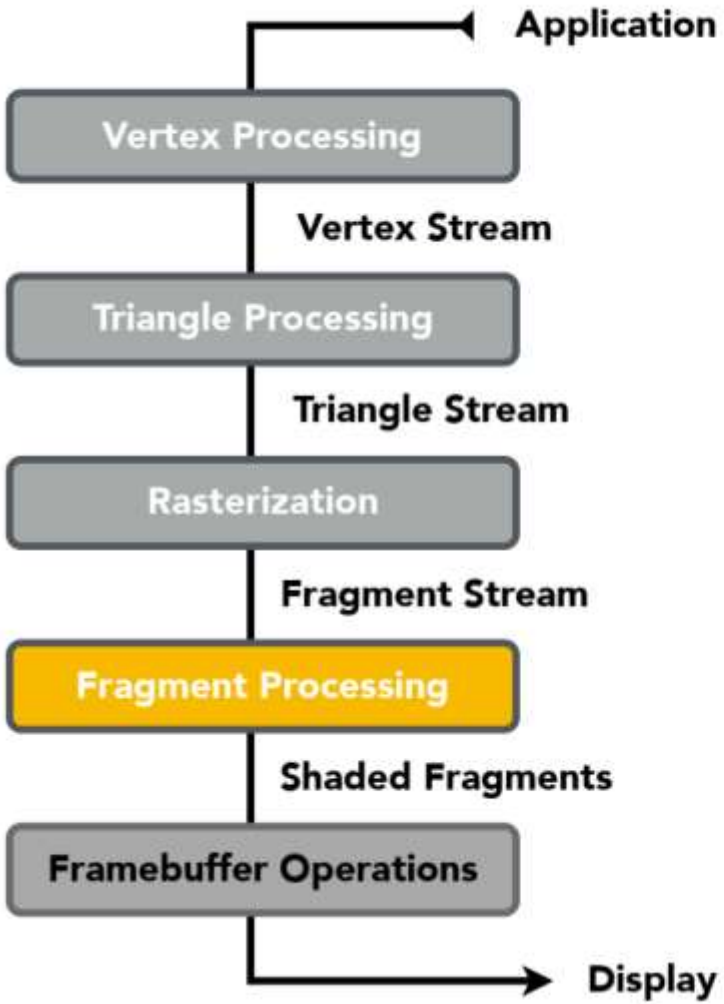


图形管线



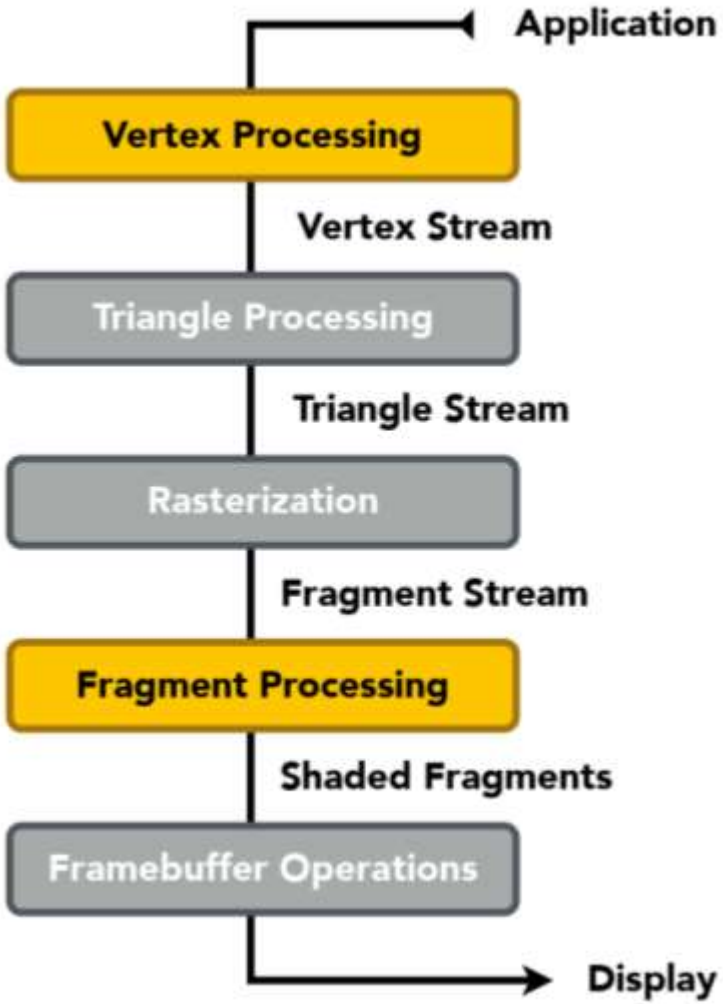


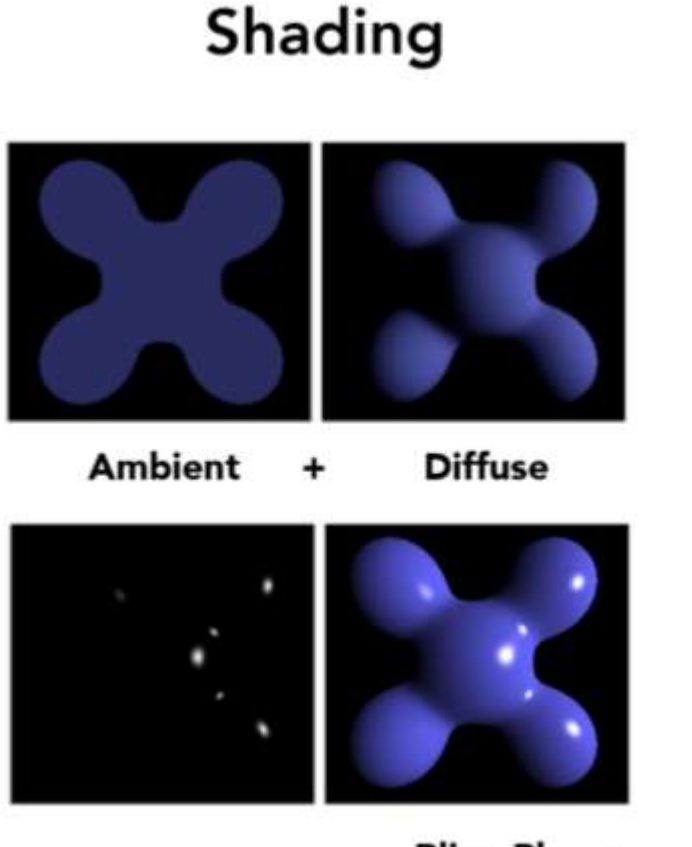
图形管线



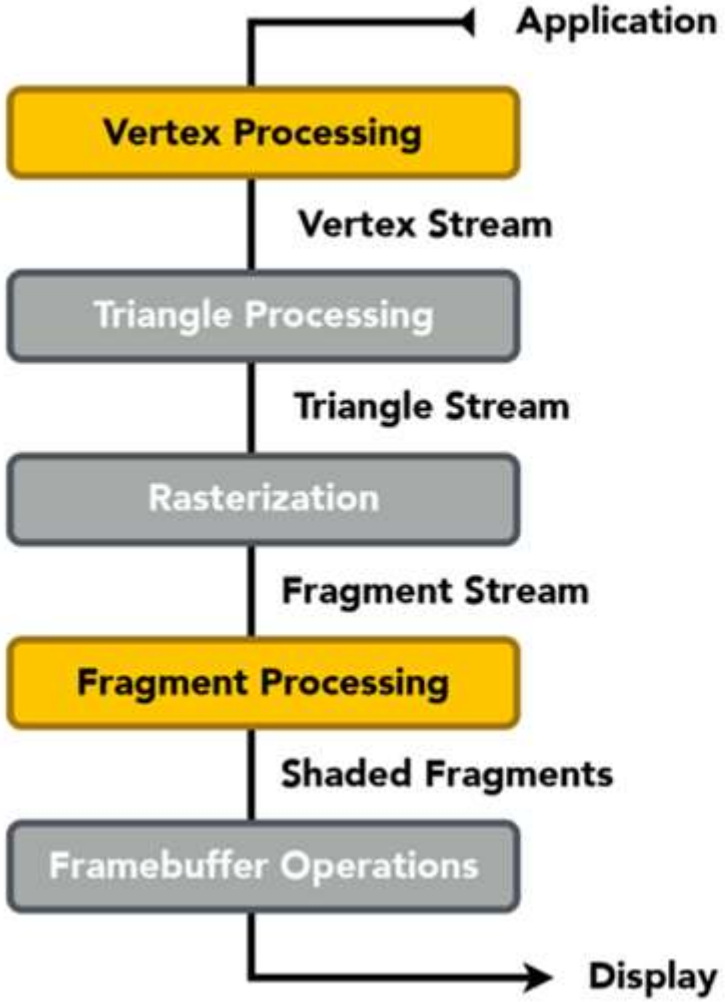
Z-Buffer Visibility Tests

图形管线





图形管线



Texture mapping

着色器程序

- 对顶点或片段进行着色编程
- 描述单个顶点（或片段）上的操作

- 右图是一个GLSL片段着色器程序
  - 着色器函数每个片段执行一次
  - 输出当前片段在屏幕采样位置处的表面颜色
  - 该着色器执行纹理查找以获得当前点表面的材质颜色，然后执行漫反射光照计算

```
uniform sampler2D myTexture;
uniform vec3 lightDir;
varying vec2 uv;
varying vec3 norm;

void diffuseShader()
{
  vec3 kd;
  kd = texture2d(myTexture, uv);
  kd *= clamp(dot(–lightDir, norm), 0.0, 1.0);
  gl_FragColor = vec4(kd, 1.0);
}
```

着色器程序

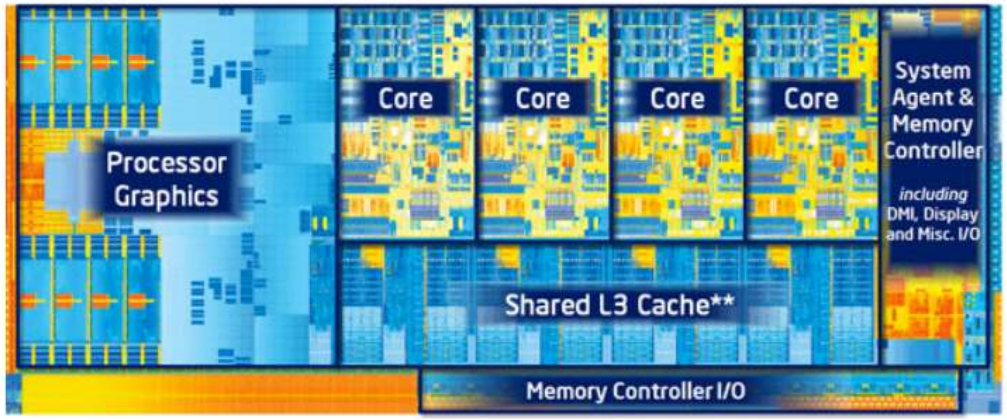
```
uniform sampler2D myTexture;      // program parameter
uniform vec3 lightDir;            // program parameter
varying vec2 uv;                  // per fragment value (interp. by rasterizer)
varying vec3 norm;                // per fragment value (interp. by rasterizer)

void diffuseShader()
{
  vec3 kd;
  kd = texture2d(myTexture, uv);                    // material color from texture
  kd *= clamp(dot(–lightDir, norm), 0.0, 1.0);      // Lambertian shading model
  gl_FragColor = vec4(kd, 1.0);                     // output fragment color
}
```

图形管线的实现：GPU

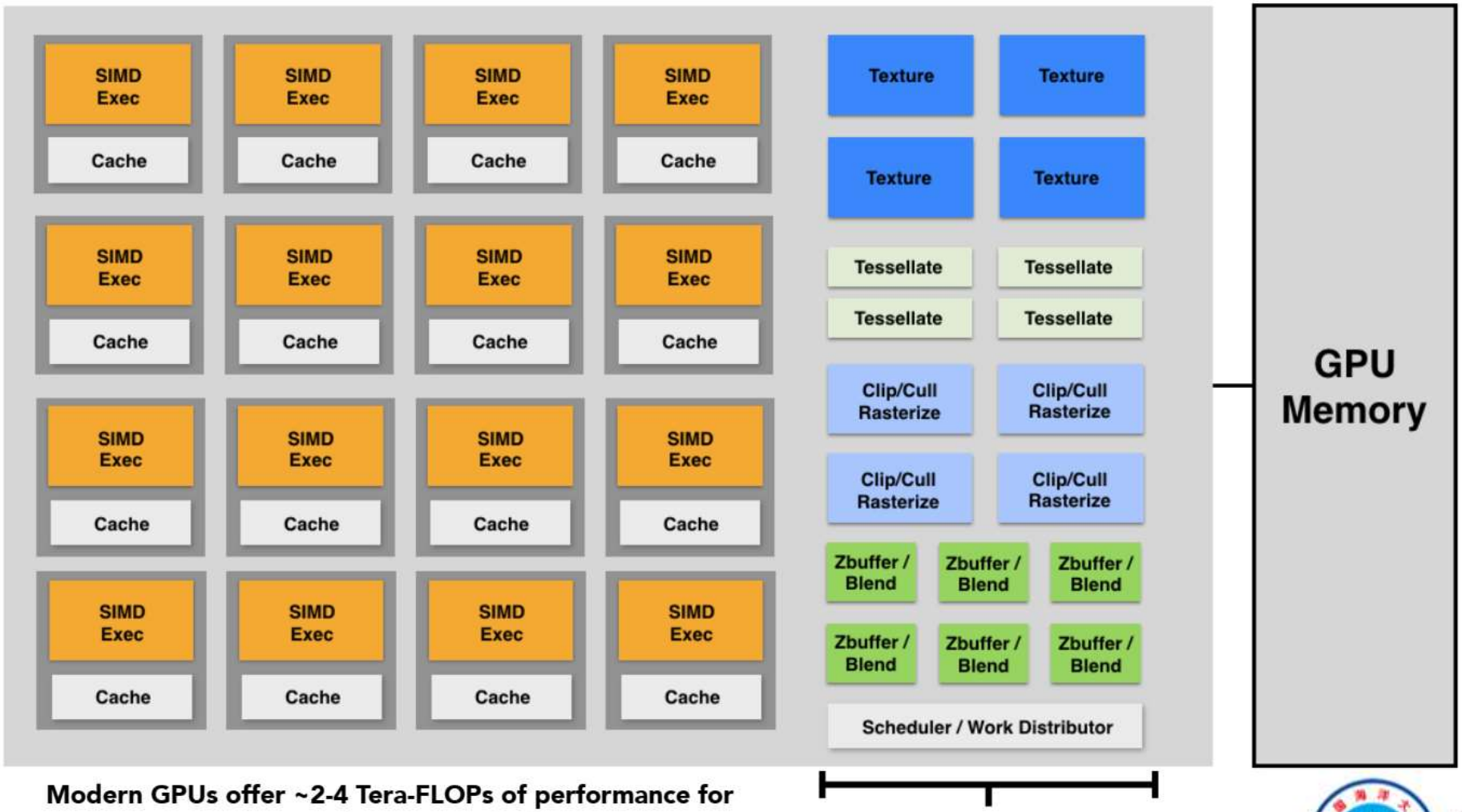
- 用于执行图形管线计算的专用处理器

Discrete GPU Card
(NVIDIA GeForce Titan X)



Integrated GPU:
(Part of Intel CPU die)

GPU：异构多核处理器



Q&A

纹理映射（Texture Mapping）

表面（surface）是二维的

- 三维空间中的表面上的一点总可以对应于二维图像（纹理）上的一点