

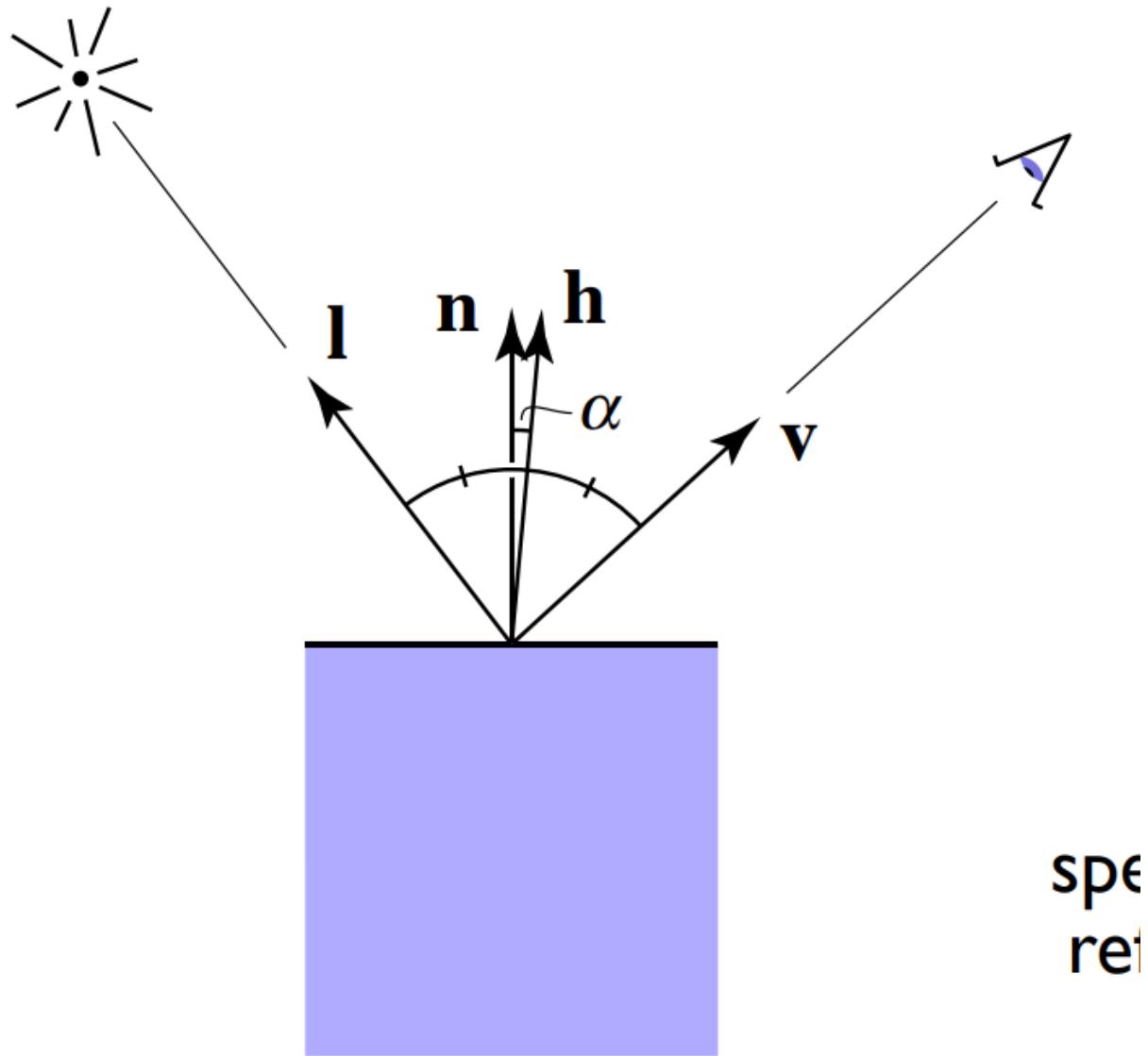
# Lecture 08

上一节了解了 Blinn-Phong Reflection Model 的三种光照模型，即

- Specular highlights 高光/镜面光
- Diffuse reflection 漫反射
- Ambient lighting 环境光照，即光源无法直接照到的地方，需要经过光线的反射

本节详细介绍该三类光照模型的数学过程

## 1.Specular Term



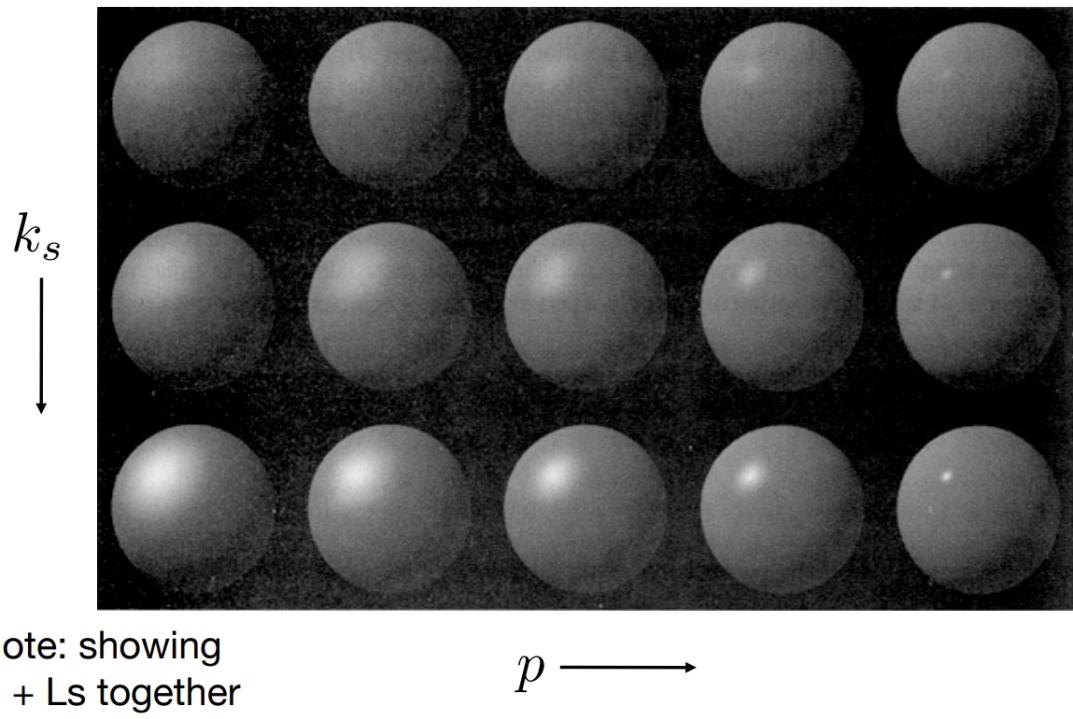
- 当观察方向  $v$  越接近镜面反射/高光  $R$  的方向，所观察到的亮度会变大。

- 定义  $\mathbf{h} = \text{bitsector}(\mathbf{v}, \mathbf{l})$  为  $\mathbf{l}, \mathbf{v}$  的 **半程向量**， $R$  与  $v$  接等价于  $\mathbf{h}$  与  $n$  接近，即可以用半程向量与法线的夹角大小衡量观察方向与高光方向的接近程度。
- 以下为高光反射亮度的计算，此公式忽略了 **物体吸收光照** 的参数，默认其都反射

$$\mathbf{h} = \text{bitsector}(\mathbf{v}, \mathbf{l}) = \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|}$$

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

- $\max(0, \mathbf{n} \cdot \mathbf{h})^p$  的  $p$  次方是为了控制高光的范围，由于  $\cos$  函数的一次方随角度增大数值降低缓慢，我们可以调整  $p$  值得大小设定高光的范围，同理  $k_s$  也会影响高光的范围



## 2.Ambient Term 环境光照

- **公式：**  $L_a = k_a I_a$
- 环境光照与光线入射，表面法线无关，仅与入射光线强度和系数  $k_a$  相关。此公式为简化后的结果，实际更复杂。

## 3.Blinn-Phong Reflection Model 完整公式

$$L = L_a + L_d + L_s$$

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

# Shading Frequency 着色频率



## 1. 着色频率

- 不同的着色方式，质量会取决于顶点和三角形的数量

## 2. 着色方式

- **Flat shading** 按 多边形/三角形平面 着色，求每个三角形的法线
- **Gouraud shading** 按 顶点 着色，求每个顶点的法线，三角形内部通过插值计算着色
- **Phong shading** 按 像素 着色，求顶点法线，再每个像素插值算出其法线向量
- 按顶点着色中求顶点法线
  - 求顶点的法线，通过求所有通过该点的平面的法线的加权平均，得到该点的法线

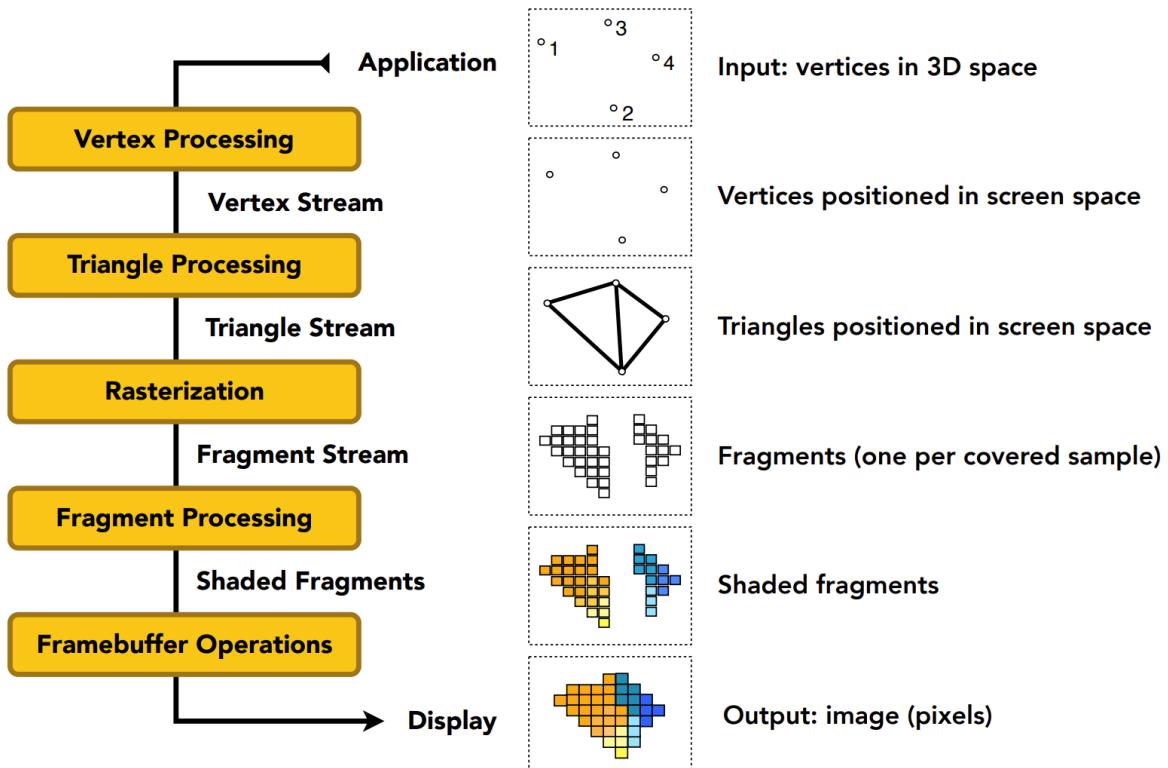
$$N_v = \frac{\sum_i N_i}{\|\sum_i N_i\|}$$

- 按像素着色中求像素法线
  - 求两顶点之间的像素的颜色需要经过**重心坐标**插值。
  - 在对法线操作时需要**归一化**。

### 3. Graphics Pipeline(Real-time Rendering) 管线(实时渲染)

- 管线

## Graphics Pipeline



- 在其中需要经历以下几步：
  - 向量处理 (MVP 变换)
  - 三角形处理 (判断像素是否在三角形内)
  - 光栅化 (求 Z-Buffer 并绘制图像)
  - 像素处理 (着色 / 纹理映射)
  - **帧缓冲操作**
- Shader 着色器
  - 分 顶点着色器/像素着色器
  - 描述在单个顶点 / 单个像素上如何操作并输出最终颜色

- Example GLSL fragment shader program

```

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)

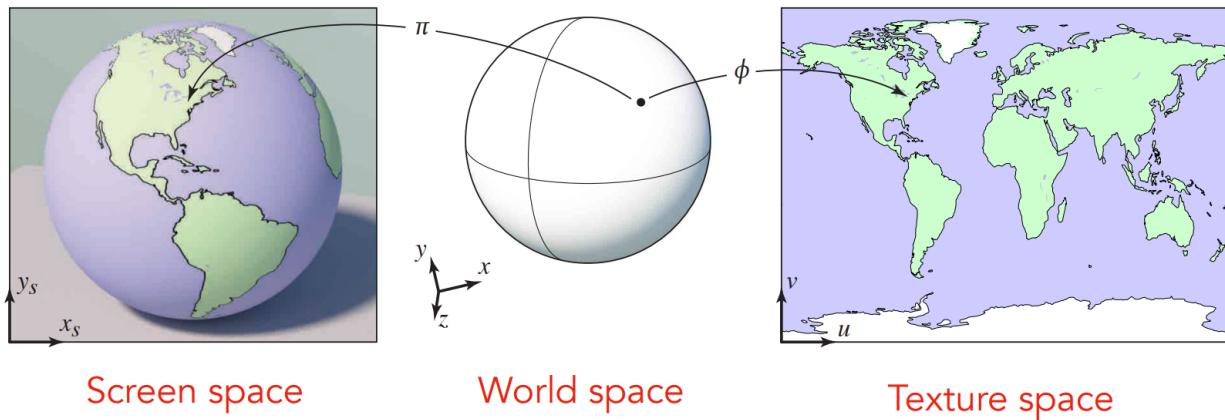
void diffuseShader()
{
    vec3 kd;
    kd = texture2d(myTexture, uv);          // material color from texture
    kd *= clamp(dot(-lightDir, norm), 0.0, 1.0); // Lambertian shading model
    gl_FragColor = vec4(kd, 1.0);           // output fragment color
}

```

- GPU
  - 图形管线的硬件实现

## 4. Texture Mapping 纹理映射

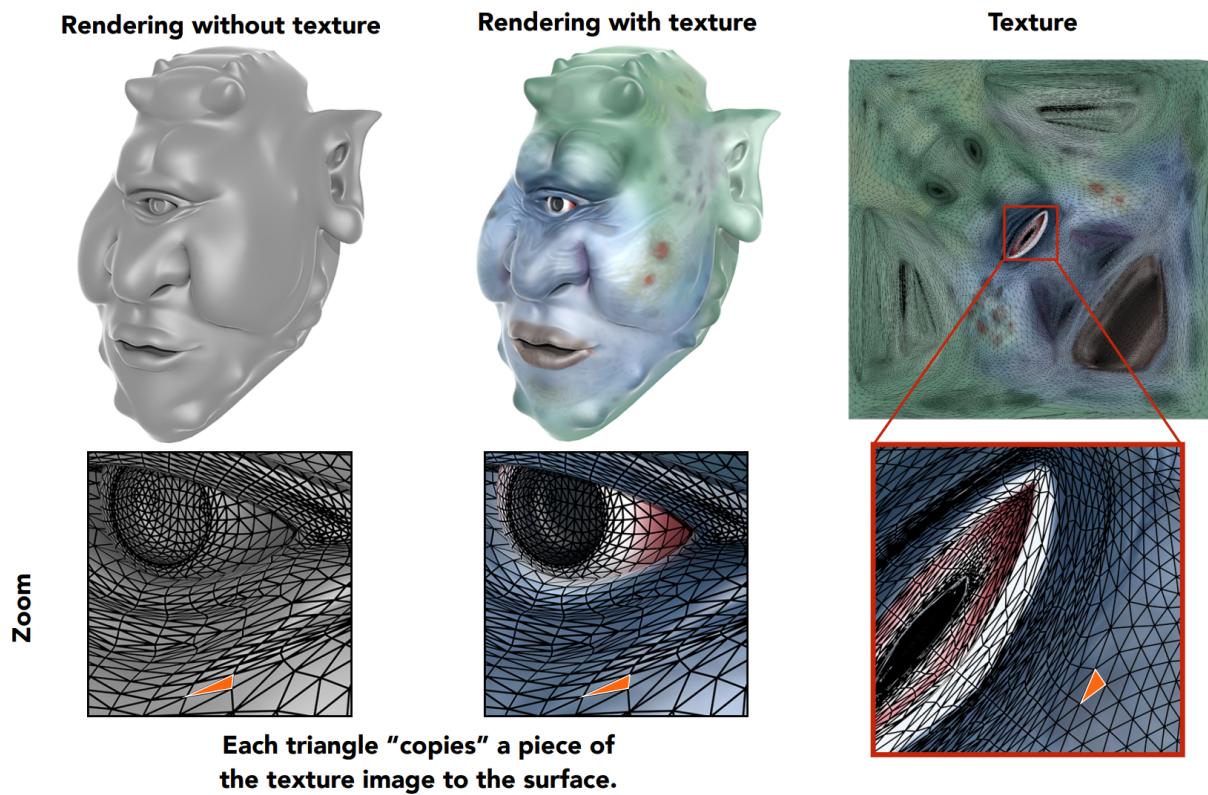
三维空间中的物体表面非常复杂，为了简化纹理映射，将其表面视作一个平面，比如地球的纹理图就是一张平面的世界地图



纹理映射的算法非常复杂，此处只解释纹理映射的大体流程的结果

- 物体表面的每一个三角形就可以看作是物体表面的一个极小面
- 运用 映射算法，可以把纹理图案覆盖到物体表面

- 从物体上的三角形到纹理图案中的三角形



纹理图使用  $u/v$  坐标系表示点的位置

- $u$  增大, 颜色变红
- $v$  增大, 颜色变绿
- 

