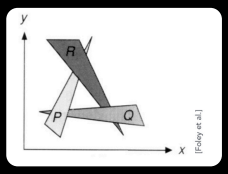
Visibility/Occlusion <Z-buffering>

Painter's Algorithm — paint from back to front, overwrite in the framebuffer

- requires sorting in depth <O(n log n) for n triangles>
- could have unresolvable depth order

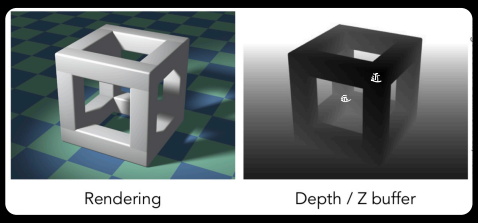


Z-Buffer

This is the algorithm that eventually won.

Idea:
- Store current min. z-value for each sample (pixel)
- Needs an additional buffer for depth values
  - frame buffer stores color values
  - depth buffer (z-buffer) stores depth

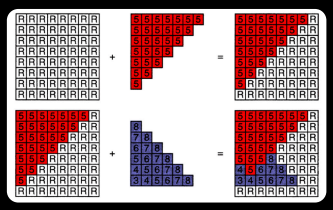
IMPORTANT: For simplicity we suppose z is always positive (smaller z -> closer, larger z -> further)




```
Initialize depth buffer to ∞
During rasterization:

for (each triangle T)
    for (each sample (x,y,z) in T)
        if (z < zbuffer[x,y])          // closest sample so far
            framebuffer[x,y] = rgb;    // update color
            zbuffer[x,y] = z;          // update depth
        else
            ;                          // do nothing, this sample is occluded
```

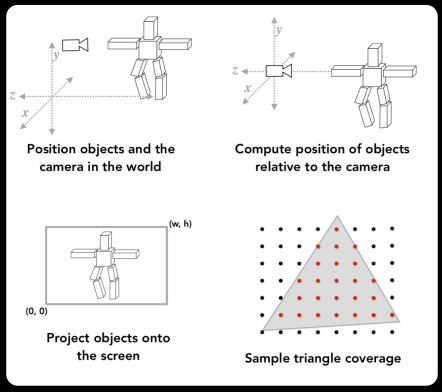
在每一个像素内记录最浅深度



- O(n) for n triangles <Cn assuming constant coverage> 并未现行排序
- Most important visibility algorithm — implemented in hardware for all GPUs
- 深度完全相等 <float> 几乎不会出现，在此不作讨论

Illumination, Shading & Graphic Pipeline

回顾 ⇨

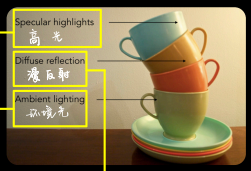


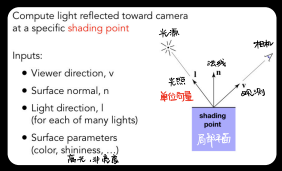
- In Merriam-Webster Dictionary

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

- In this course

  The process of applying a material to an object.



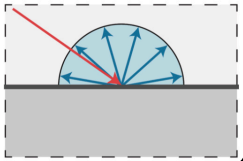


▲ Shading is local. NO SHADOW is generated

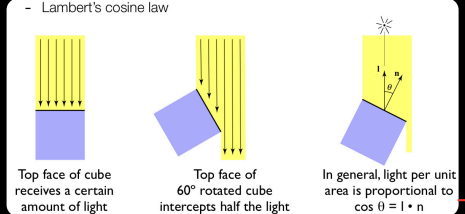
Diffuse Reflection

How much light is received?

- Light is scattered uniformly in all directions
  - Surface color is the same for all viewing directions

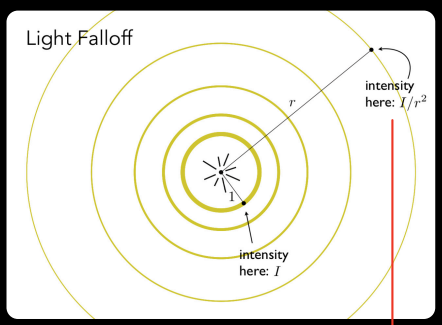


- Lambert's cosine law

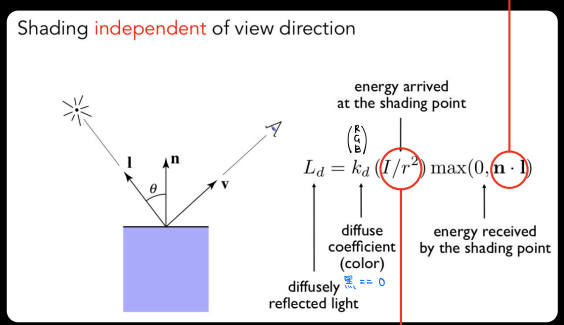


$\vec{a} \cdot \vec{b} = \|\vec{a}\| \cdot \|\vec{b}\| \cdot \cos\theta$ (b is normal vector)

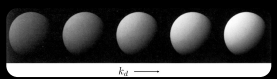
Diffusely Reflected Light Formula



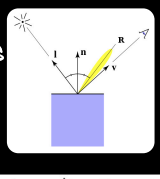
球体面积 $4\pi r^2$



漫反射成像与观测角度无关



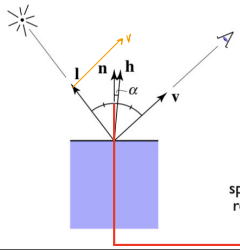
Specular Highlights



Blinn-Phong

V close to mirror direction ⇔ **half vector** near normal

- Measure "near" by dot product of unit vectors



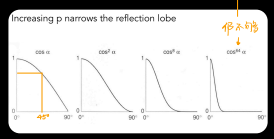
$$\mathbf{h} = \text{bisector}(\mathbf{v},\mathbf{l}) \quad \text{(半程向量)} = \frac{\mathbf{v}+\mathbf{l}}{\|\mathbf{v}+\mathbf{l}\|}$$ 方便计算

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

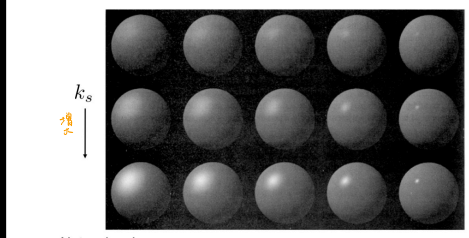
specularly reflected light ← usually white ← specular coefficient

法线方向与半程向量接近

使用 cos 函数时 "容忍度过高"，即使相差达到 45° 仍为明显差值



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

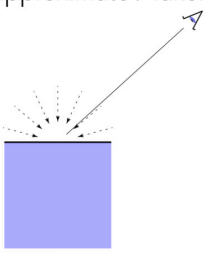


Ambient Lighting 环境光

与光源，观测角度无关

Shading that does not depend on anything

- Add constant color to account for disregarded illumination and fill in black shadows
- This is approximate / fake!



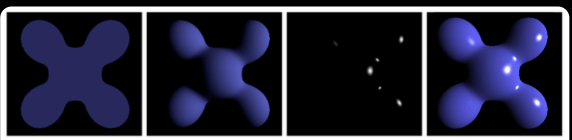
$$L_a = k_a\,I_a$$

k_a: ambient coefficient

I_a: reflected ambient light

Blinn-Phong Reflection Model

Intensity



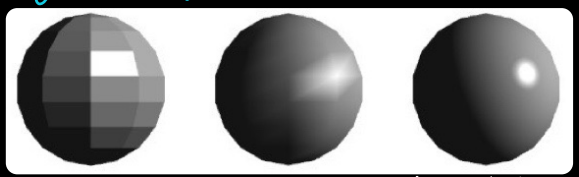
Ambient + Diffuse + Specular = Blinn-Phong Reflection

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

coefficient-wise product 对应相乘

floor cos θ

Shading Frequency 着色频率

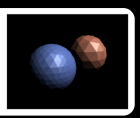


顶点着色 差值过渡

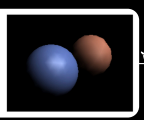
每一个像素计算法线并进行上色

Flat Shading | Gouraud Shading 高洛德着色 | Phong Shading

Flat shading
- Triangle face is flat — one normal vector
- Not good for smooth surfaces



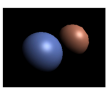
Gouraud shading
- Interpolate colors from vertices across triangle
- Each vertex has a normal vector (how?)



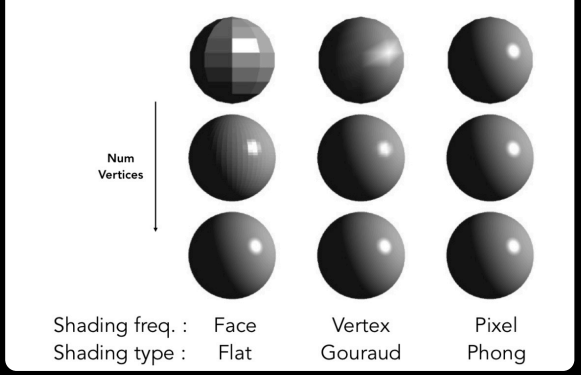
如何求

顶点法线

Phong shading
- Interpolate normal vectors across each triangle
- Compute full shading model at each pixel
- Not the **Blinn-Phong Reflectance Model**



Shading Frequency: Face, Vertex or Pixel

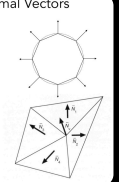


Defining Per-Vertex Normal Vectors

Best to get vertex normals from the underlying geometry
- e.g. consider a sphere

Otherwise have to infer vertex normals from triangle faces
- Simple scheme: **average surrounding face normals**

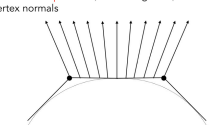
$$N_v = \frac{\sum_i N_i}{\|\sum_i N_i\|}$$



相邻三角形（加面积权）法线的平均

Defining Per-Pixel Normal Vectors

Barycentric interpolation (introducing soon) of vertex normals

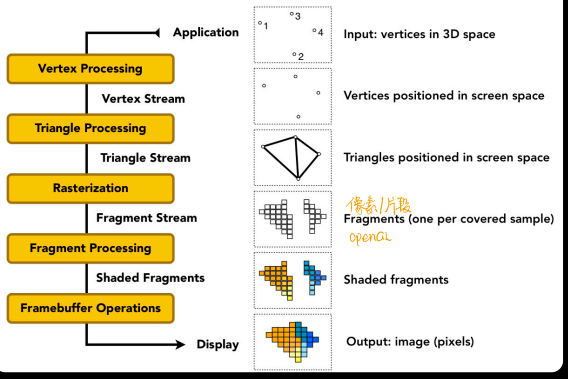


Don't forget to normalize the interpolated directions

Graphics < Real-time Rendering > Pipeline

Practice = Shader toy

Graphics Pipeline



投影

形成三角形

光栅化

着色

Shader Programs

- Program vertex and fragment processing stages
- Describe operation on a single vertex (or fragment)

Example GLSL fragment shader program

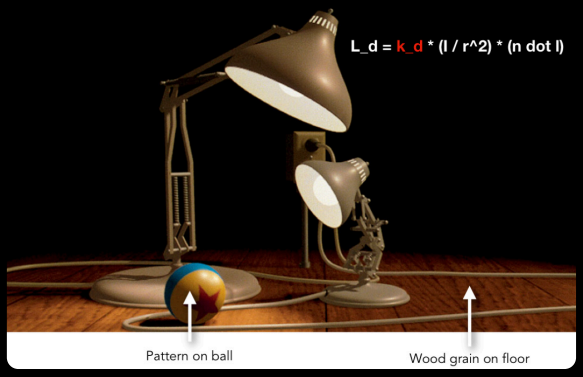
```
uniform sampler2D myTexture;
uniform vec3 lightDir;
varying vec2 uv;
varying vec3 norm;

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

- Shader function executes once per fragment.
- Outputs color of surface at the current fragment's screen sample position.
- This shader performs a texture lookup to obtain the surface's material color at this point, then performs a diffuse lighting calculation.

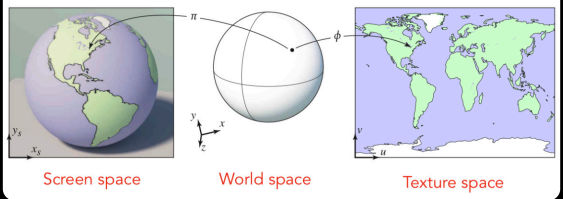
Texture Mapping

How to define K_d for every pixel



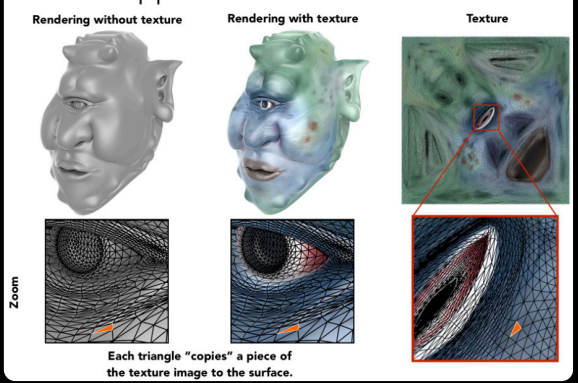
Surface lives in 3D world space

Every 3D surface point also has a place where it goes in the 2D image (**texture**).



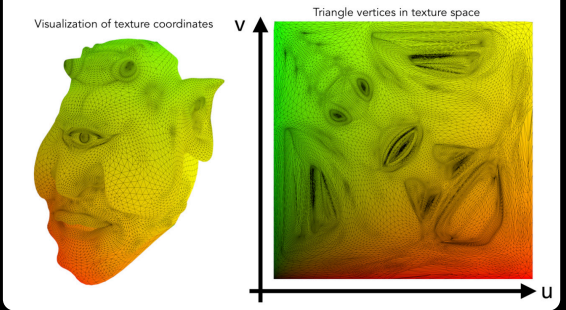
三维物体的表面是二维的

Texture Applied to Surface

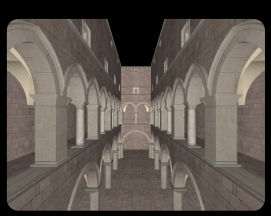


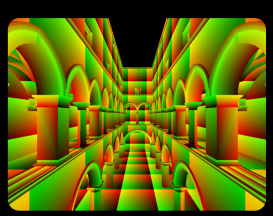
Visualization of Texture Coordinates

Each triangle vertex is assigned a texture coordinate (u,v)



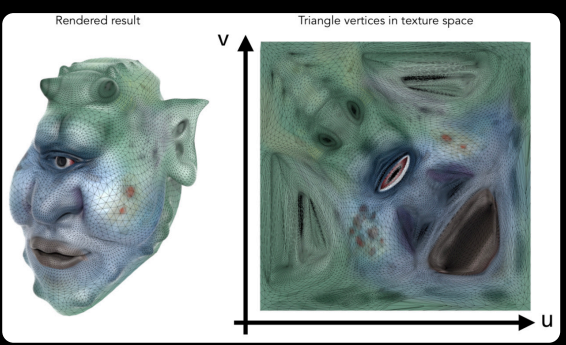
0~1 always regardless original aspect ratio





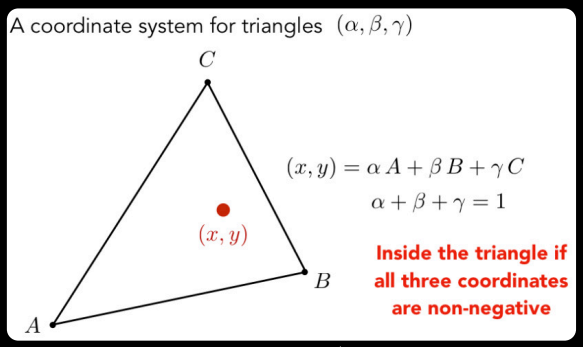
texture coordinates

tileable texture



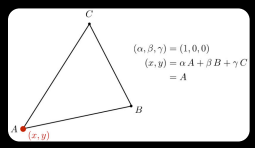
Barycentric Coordinates 重心坐标 - 计算三角形内部的插值 <像素的法线>

Interpolation across triangle - 一个三角形对应一个重心坐标

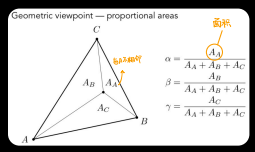


通过三个顶点之线性组合来得到点 (x, y)

$\gamma = 1 - \alpha - \beta$

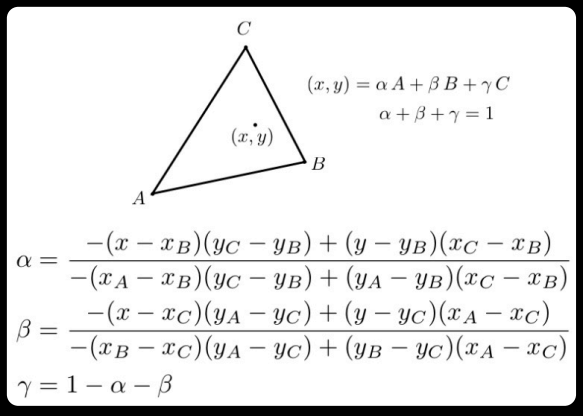


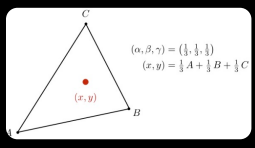
例：1点是重心坐标



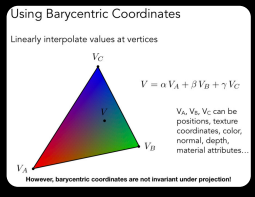
通过面积形重心坐标

公　　式





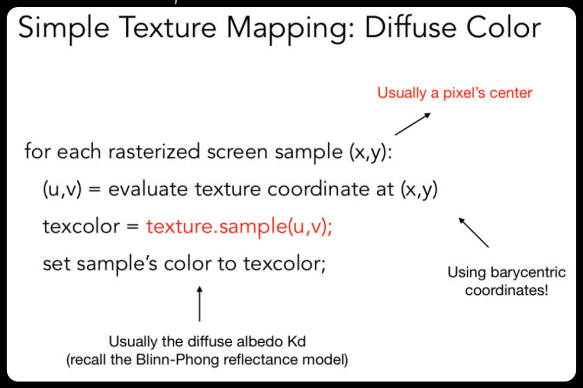
重心 <centroid> 是重心坐标



使用重心坐标计算属性

- 法线
- 颜色
- 纹理
- 位置
- etc

▲ 重心坐标在投影 <变形> 后会改变，因此要在三维空间内先算好重心坐标，再投影成二维



三维模型

$P = \alpha A + \beta B + \gamma C$

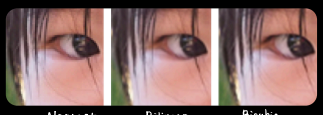
放缩三角

$P' = \alpha a + \beta b + \gamma c$

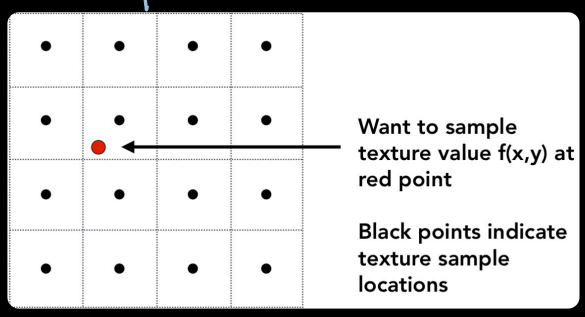
得到该三角内的颜色

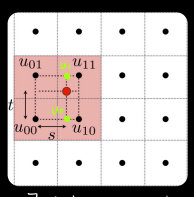
Texture Magnification 纹理放大

a pixel on texture — texel 纹理元素，纹素



Bilinear Interpolation 双线性插值





寻找邻近的四个点，并量距离左下角点的水平竖直距离

Linear interpolation (1D)

$$\text{lerp}(x, v_0, v_1) = v_0 + x(v_1 - v_0)$$

Two helper lerps

$$u_0 = \text{lerp}(s, u_{00}, u_{10})$$

$$u_1 = \text{lerp}(s, u_{01}, u_{11})$$

Final vertical lerp, to get result:

$$f(x, y) = \text{lerp}(t, u_0, u_1)$$

采样点非整数

Bicubic Interpolation 双三次插值 - 取周围16格计算