# Ray Tracing

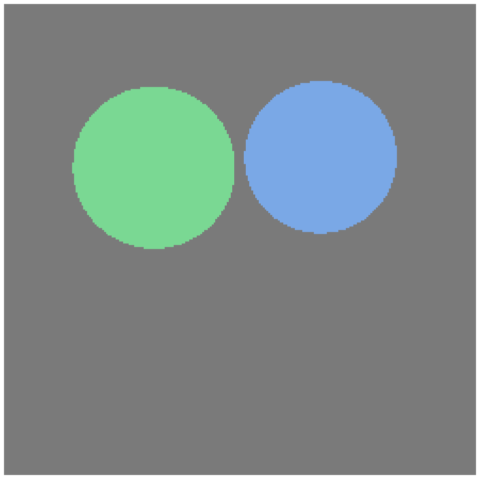
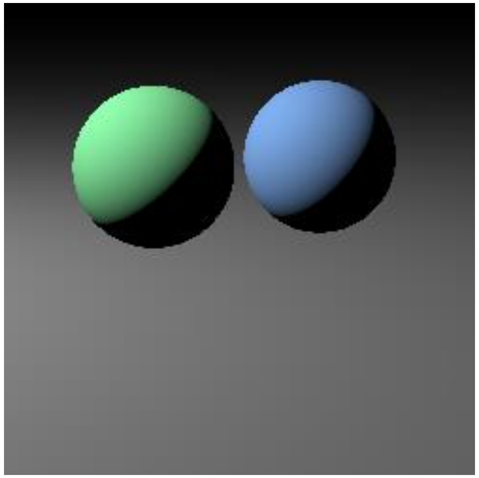
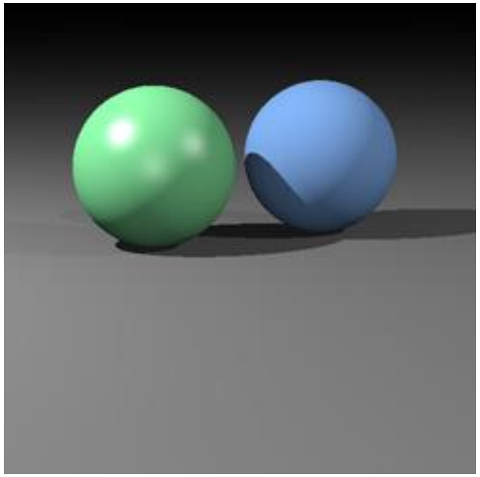
Lecture 4



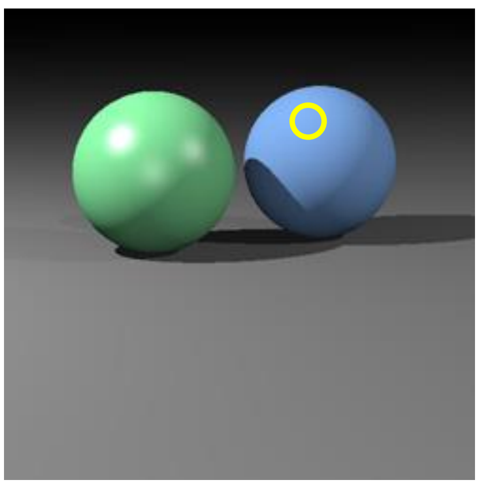
 

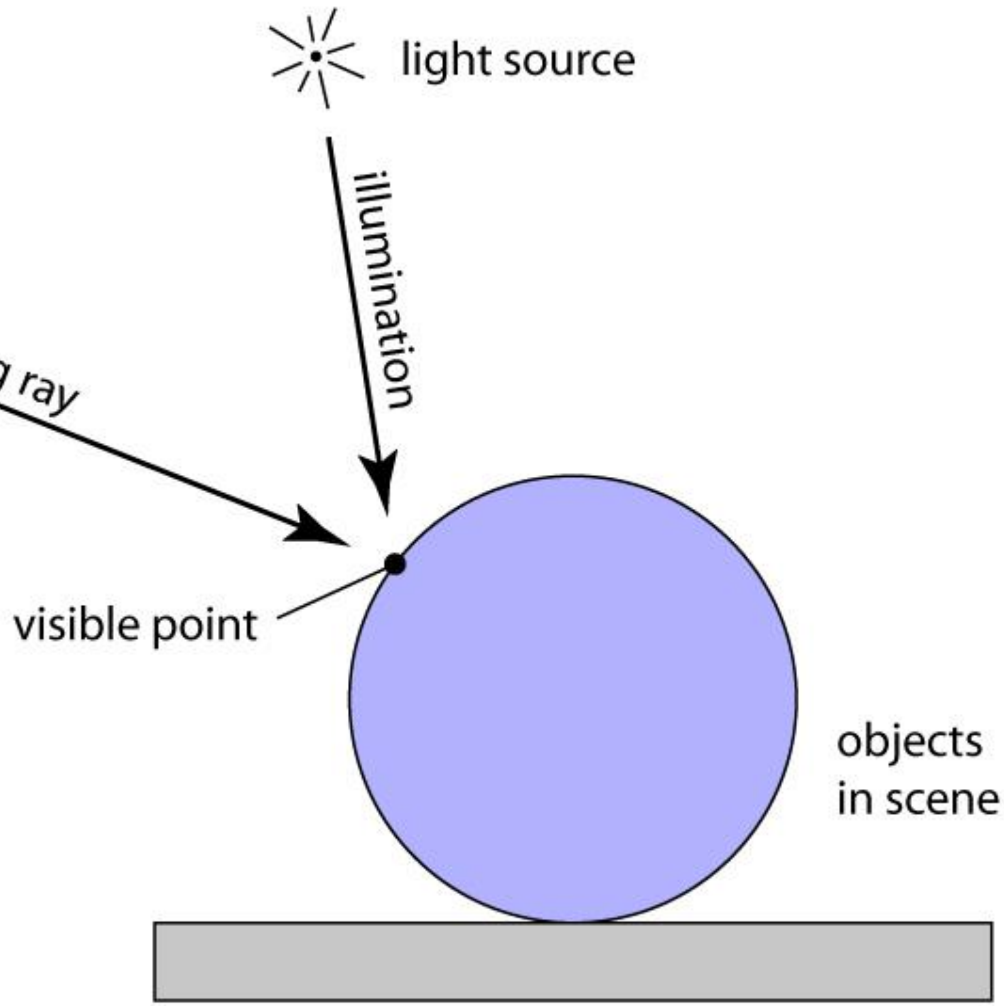
# Ray Tracing

- PA#1

# Ray tracing idea

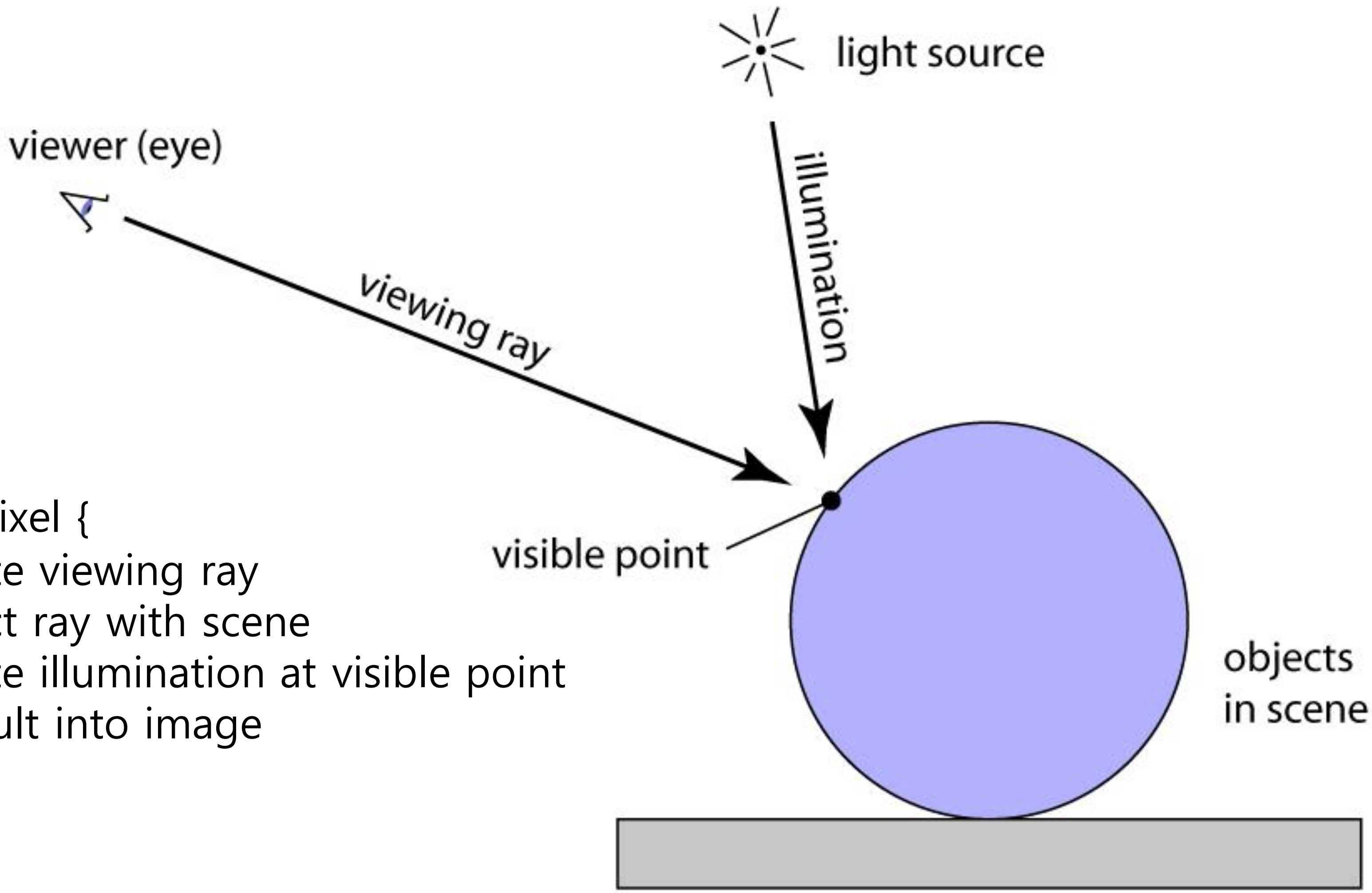






# Ray tracing algorithm



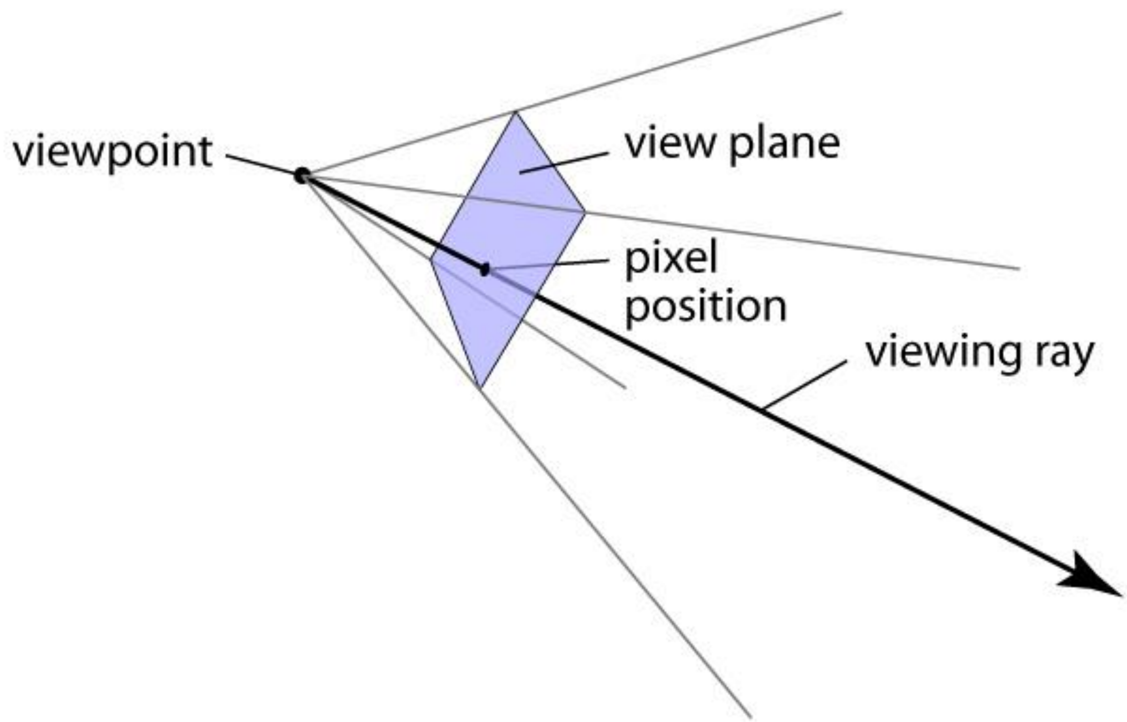

**for** each pixel {
compute viewing ray
intersect ray with scene
compute illumination at visible point
put result into image
}



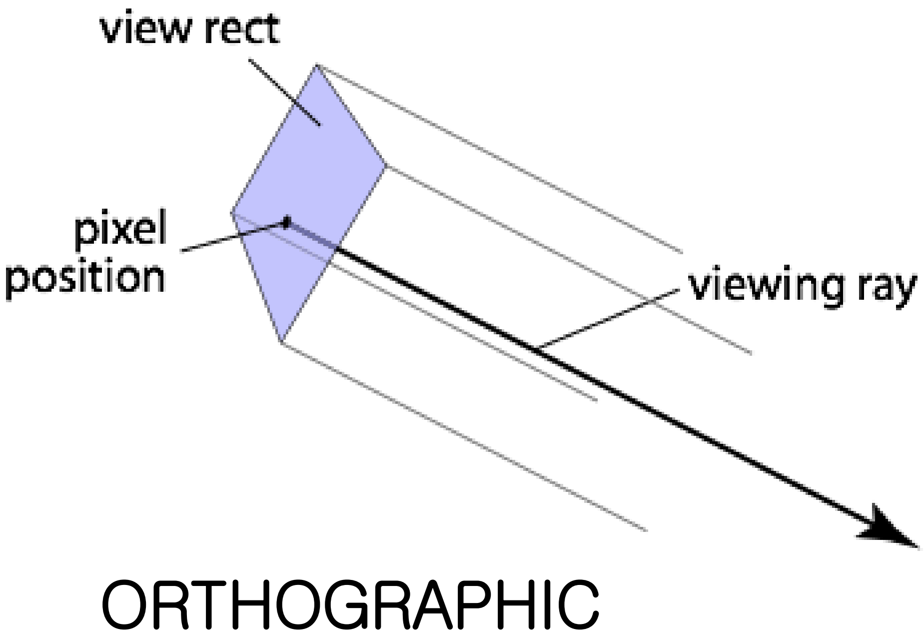
 

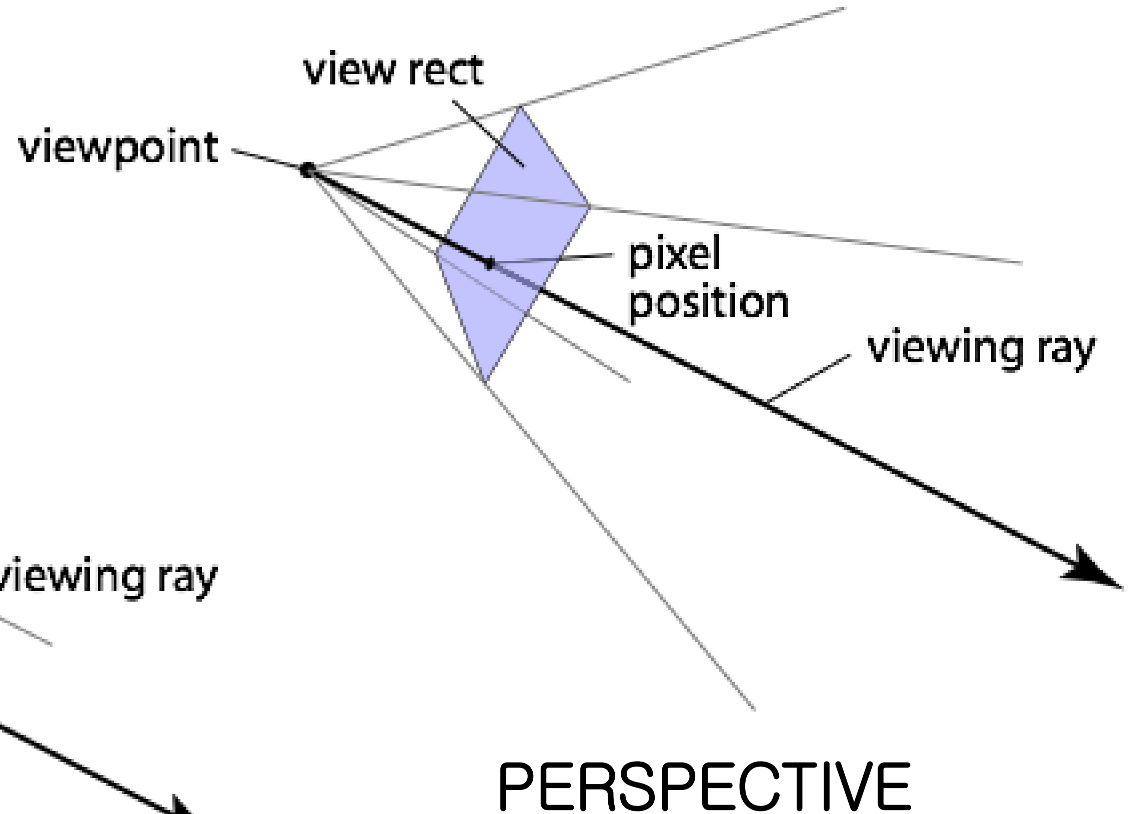
# Generating eye rays

- Use window analogy directly



# Generating eye rays





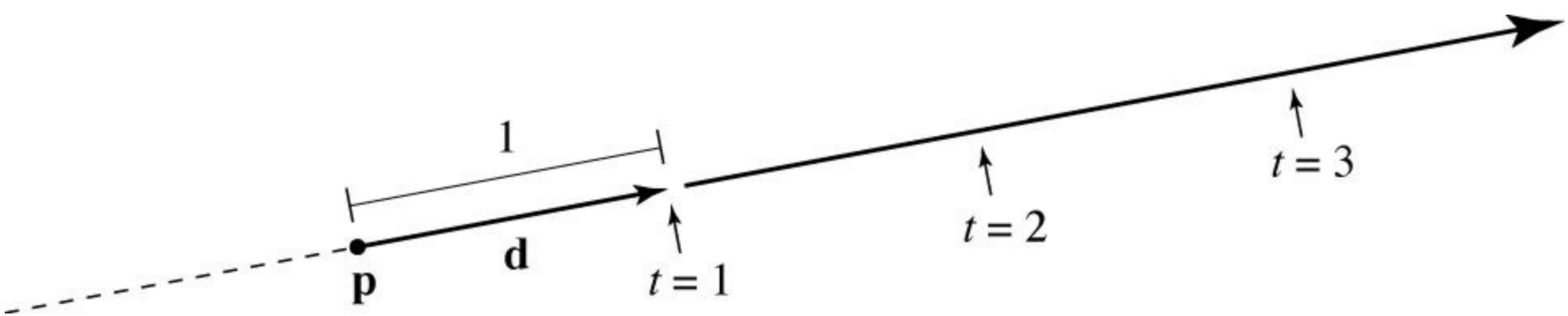

 

# Ray: a half line

- Standard representation: point **p** and direction **d**

  $$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$

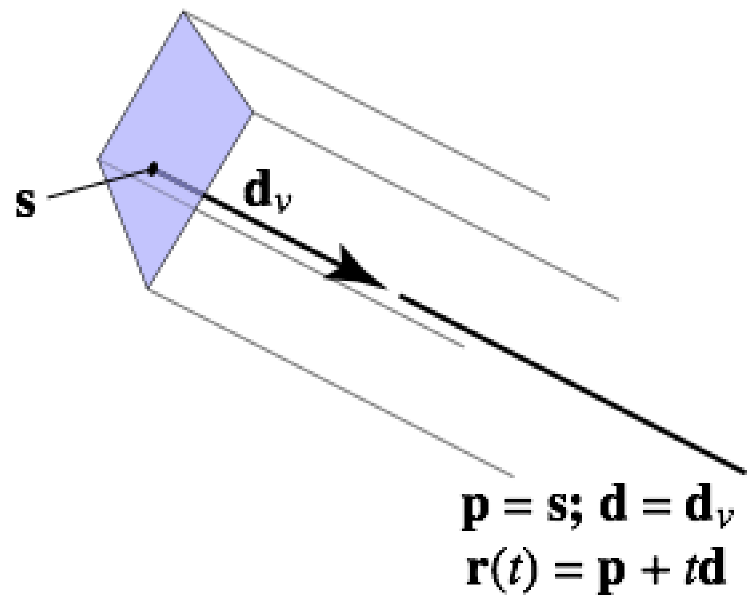
  - this is a *parametric equation* for the line
  - lets us directly generate the points on the line
  - if we restrict to $t > 0$ then we have a ray
  - note replacing **d** with $a$**d** doesn't change ray ($a > 0$)



# Generating eye rays—orthographic

- Just need to compute the view plane point **s**:



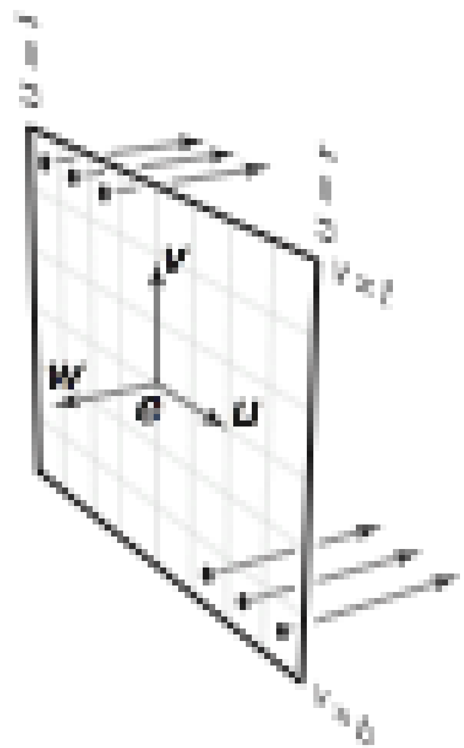

  – but where exactly is the view rectangle?

# Generating eye rays—orthographic

$$\mathbf{s} = \mathbf{e} + u\mathbf{u} + v\mathbf{v}$$

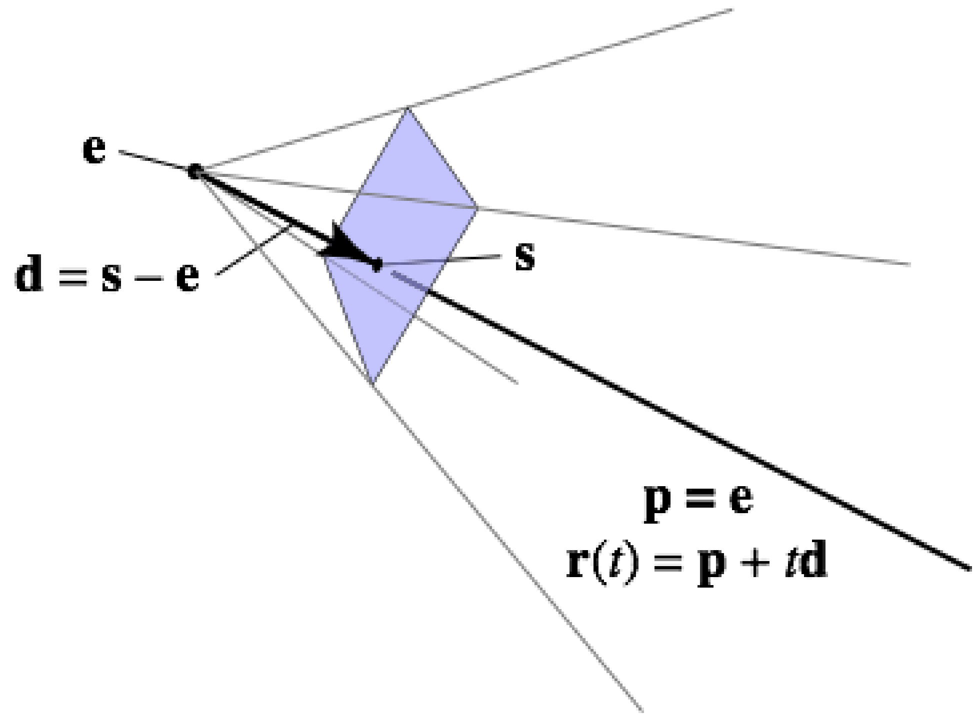
$$\mathbf{p} = \mathbf{s};\ \mathbf{d} = -\mathbf{w}$$

$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$



# Generating eye rays—perspective

- Distance is important: "focal length" of camera
  - ray origin always **e**
  - ray direction
    controlled by **s**



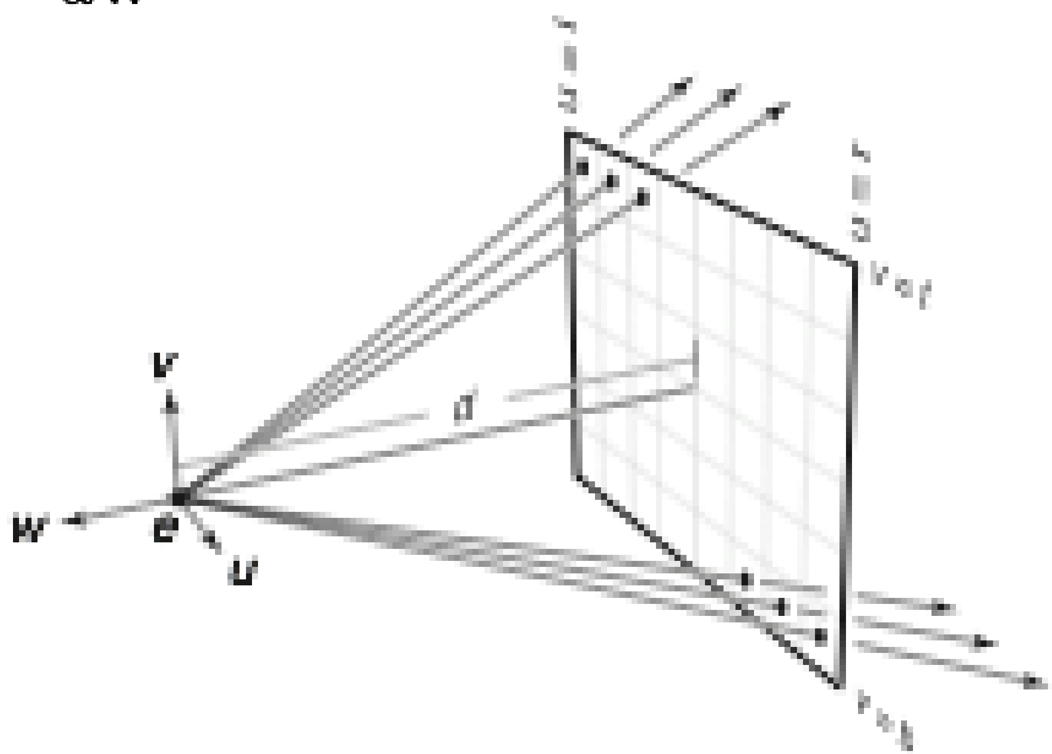
# Generating eye rays—perspective

- Compute **s** in the same way; just subtract *d***w**
  - coordinates of **s** are (*u*, *v*, −*d*)

$$\mathbf{s} = \mathbf{e} + u\mathbf{u} + v\mathbf{v} - d\mathbf{w}$$

$$\mathbf{p} = \mathbf{e};\ \mathbf{d} = \mathbf{s} - \mathbf{e}$$

$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$

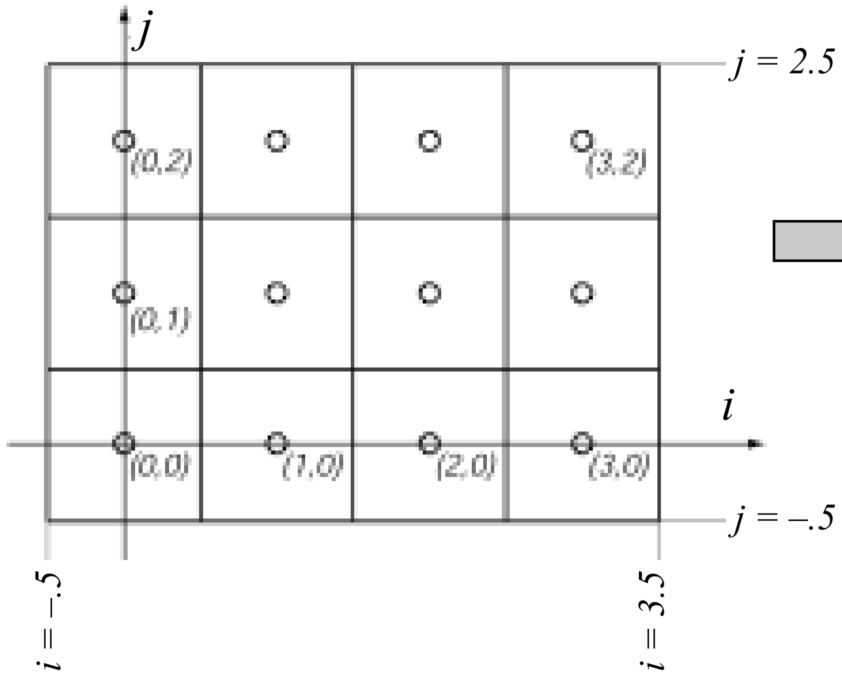


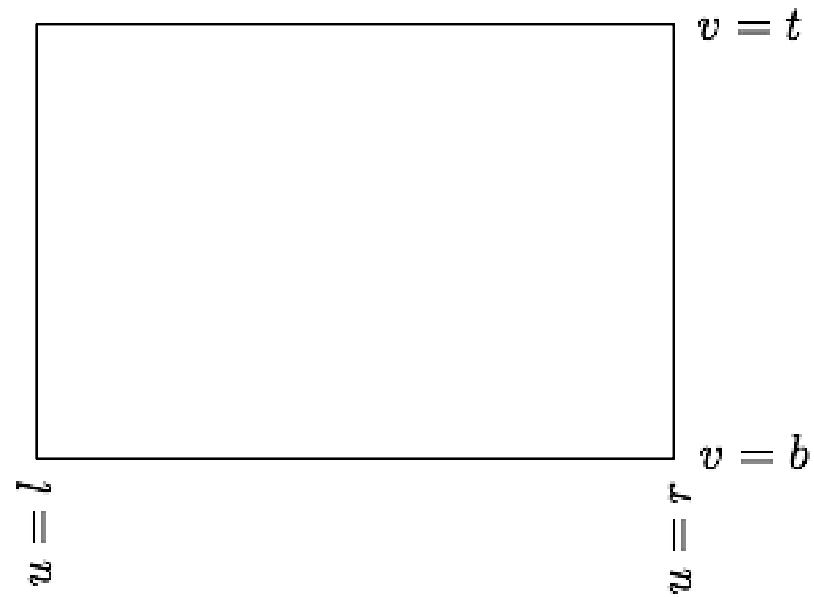


# Pixel-to-image mapping

- One last detail: ($u$, $v$) coords of a pixel

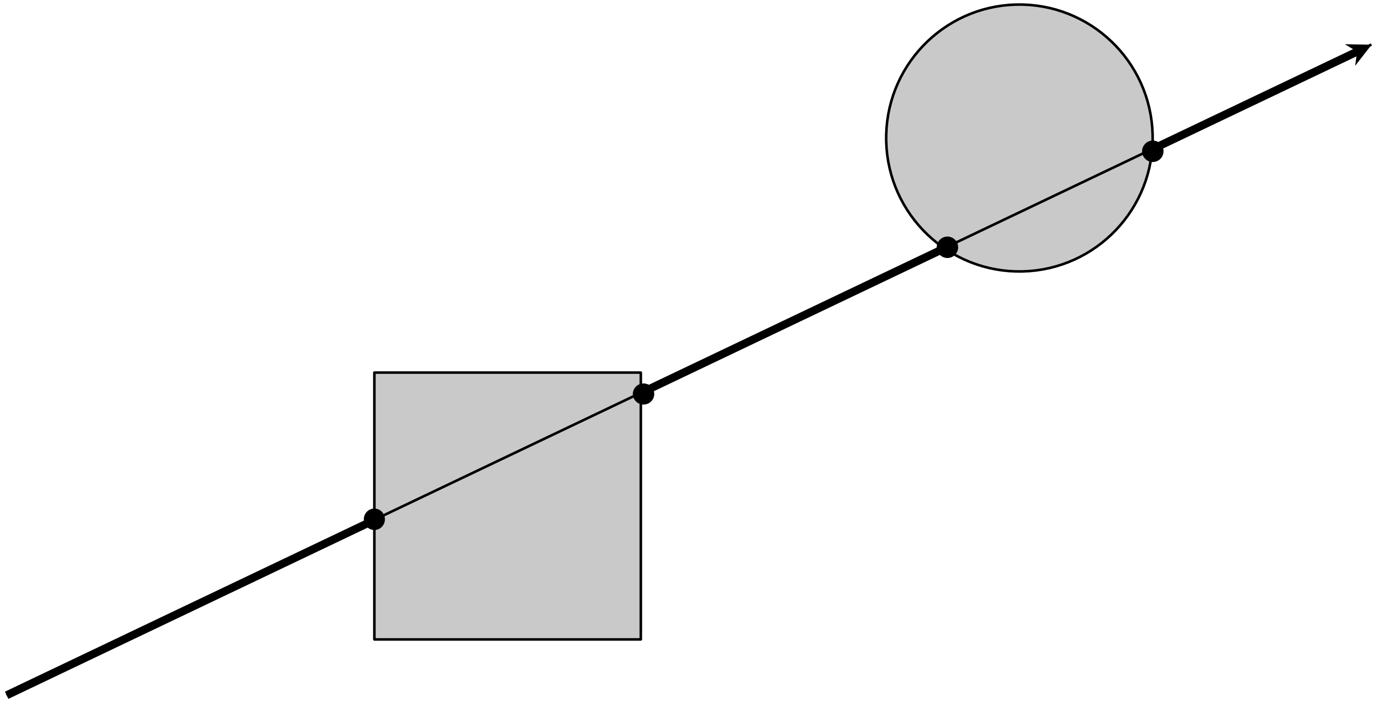



$$u = l + (r - l)(i + 0.5)/n_x$$

$$v = b + (t - b)(j + 0.5)/n_y$$

# Ray intersection



# Ray–sphere intersection: algebraic

- Condition 1: intersection point r is on ray

$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$

- Condition 2: point is on sphere
  - assume unit sphere; see Shirley or notes for general

$$\|\mathbf{x}\| = 1 \Leftrightarrow \|\mathbf{x}\|^2 = 1$$

$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{x} - 1 = 0$$

- Substitute:

$$(\mathbf{p} + t\mathbf{d}) \cdot (\mathbf{p} + t\mathbf{d}) - 1 = 0$$

  - this is a quadratic equation in *t*

# Ray-sphere intersection: algebraic

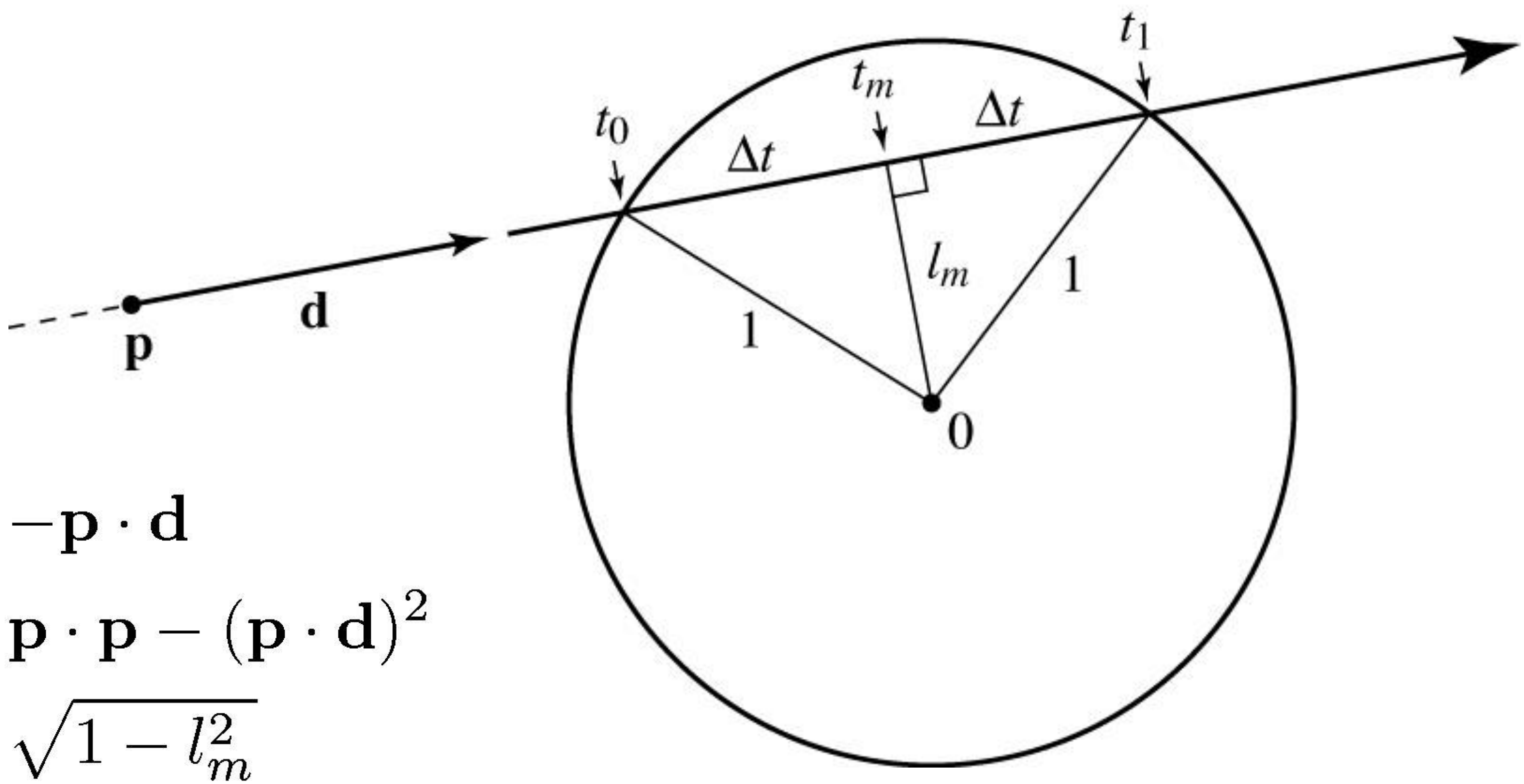
- Solution for *t* by quadratic formula:

$$t = \frac{-\mathbf{d}\cdot\mathbf{p} \pm \sqrt{(\mathbf{d}\cdot\mathbf{p})^2 - (\mathbf{d}\cdot\mathbf{d})(\mathbf{p}\cdot\mathbf{p} - 1)}}{\mathbf{d}\cdot\mathbf{d}}$$

$$t = -\mathbf{d}\cdot\mathbf{p} \pm \sqrt{(\mathbf{d}\cdot\mathbf{p})^2 - \mathbf{p}\cdot\mathbf{p} + 1}$$

  - simpler form holds when **d** is a unit vector

# Ray-sphere intersection: geometric




$$t_m = -\mathbf{p} \cdot \mathbf{d}$$

$$l_m^2 = \mathbf{p} \cdot \mathbf{p} - (\mathbf{p} \cdot \mathbf{d})^2$$

$$\Delta t = \sqrt{1 - l_m^2}$$

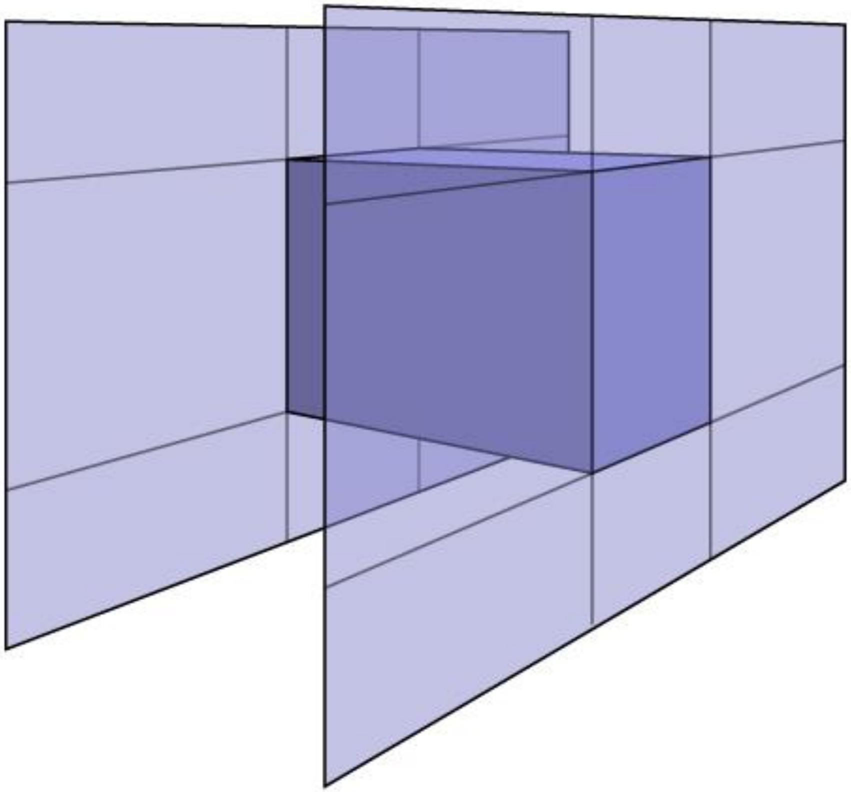
$$= \sqrt{(\mathbf{p} \cdot \mathbf{d})^2 - \mathbf{p} \cdot \mathbf{p} + 1}$$

$$t_{0,1} = t_m \pm \Delta t = -\mathbf{p} \cdot \mathbf{d} \pm \sqrt{(\mathbf{p} \cdot \mathbf{d})^2 - \mathbf{p} \cdot \mathbf{p} + 1}$$

# Ray-box intersection

- Could intersect with 6 faces individually
- Better way: box is the intersection of 3 slabs

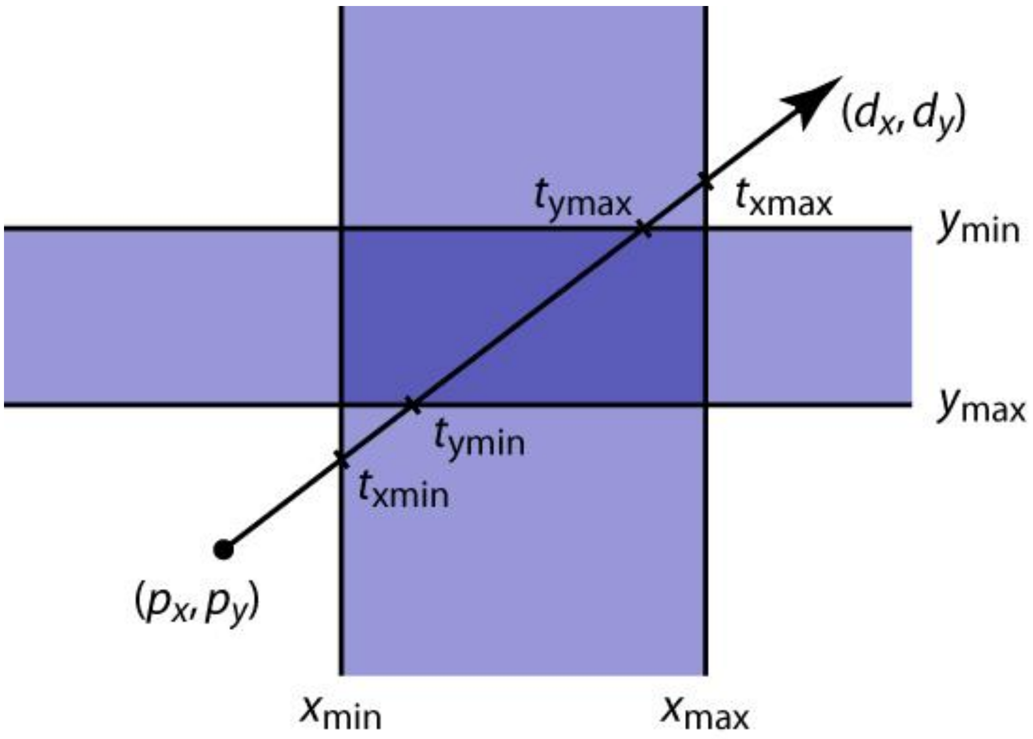


# Ray-slab intersection

- 2D example
- 3D is the same!

$$p_x + t_{x\min}\, d_x = x_{\min}$$
$$t_{x\min} = (x_{\min} - p_x)/d_x$$

$$p_y + t_{y\min}\, d_y = y_{\min}$$
$$t_{y\min} = (y_{\min} - p_y)/d_y$$

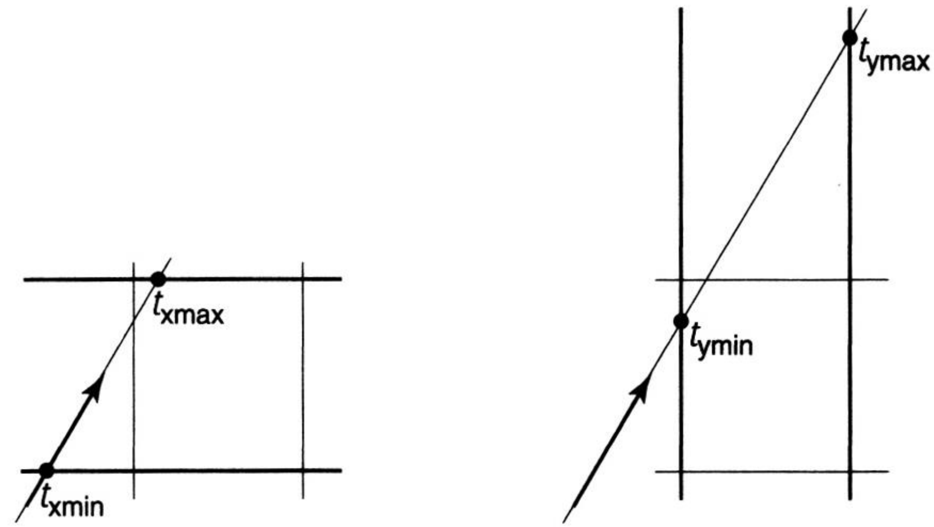




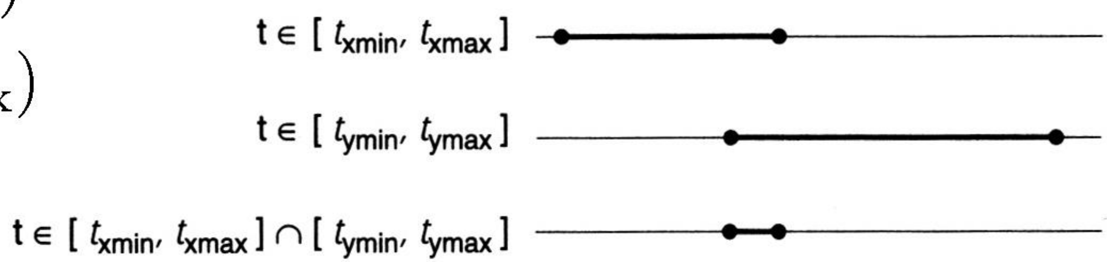

 

# Intersecting intersections

- Each intersection is an interval
- Want last entry point and first exit point




$$t_{\min} = \max(t_{x\min}, t_{y\min})$$
$$t_{\max} = \min(t_{x\max}, t_{y\max})$$




Shirley fig. 10.16

# Ray–triangle intersection

- Condition 1: point is on ray

$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$

- Condition 2: point is on plane

$$(\mathbf{x} - \mathbf{a}) \cdot \mathbf{n} = 0$$

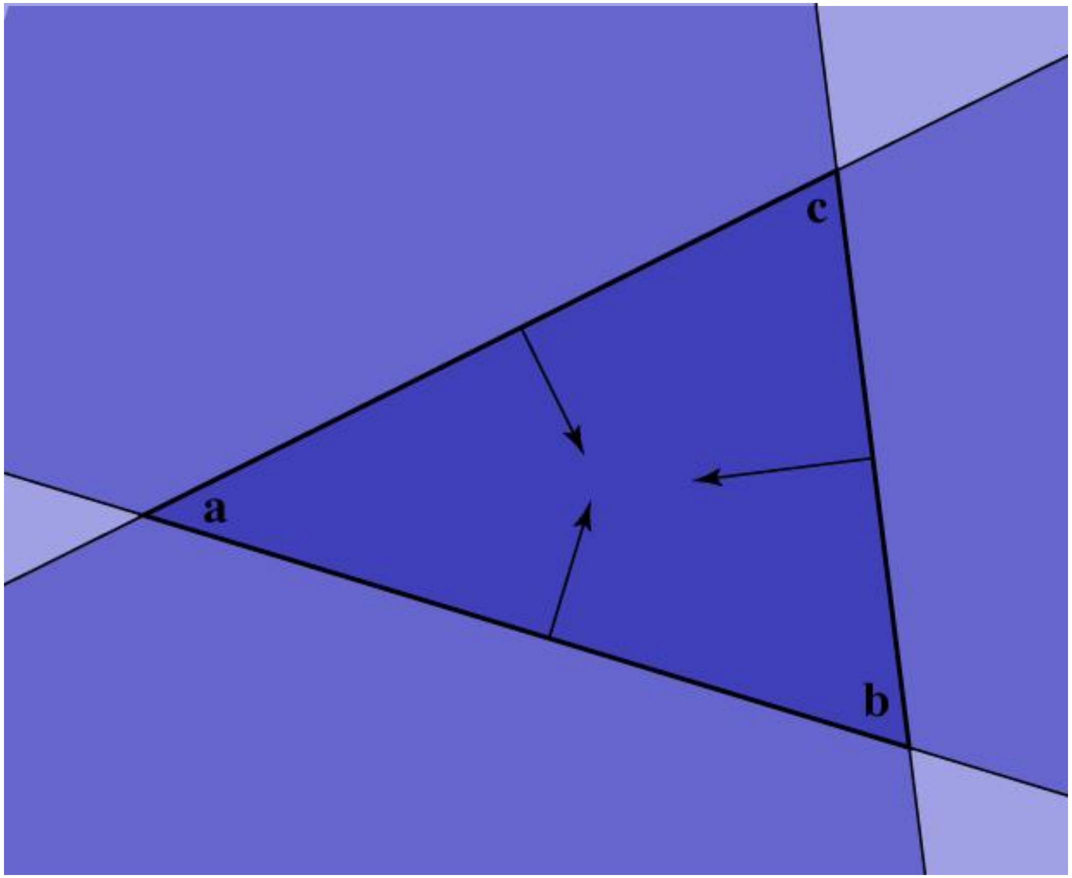
- Condition 3: point is on the inside of all three edges
- First solve 1&2 (ray–plane intersection)
  - substitute and solve for *t*:

$$(\mathbf{p} + t\mathbf{d} - \mathbf{a}) \cdot \mathbf{n} = 0$$

$$t = \frac{(\mathbf{a} - \mathbf{p}) \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}$$

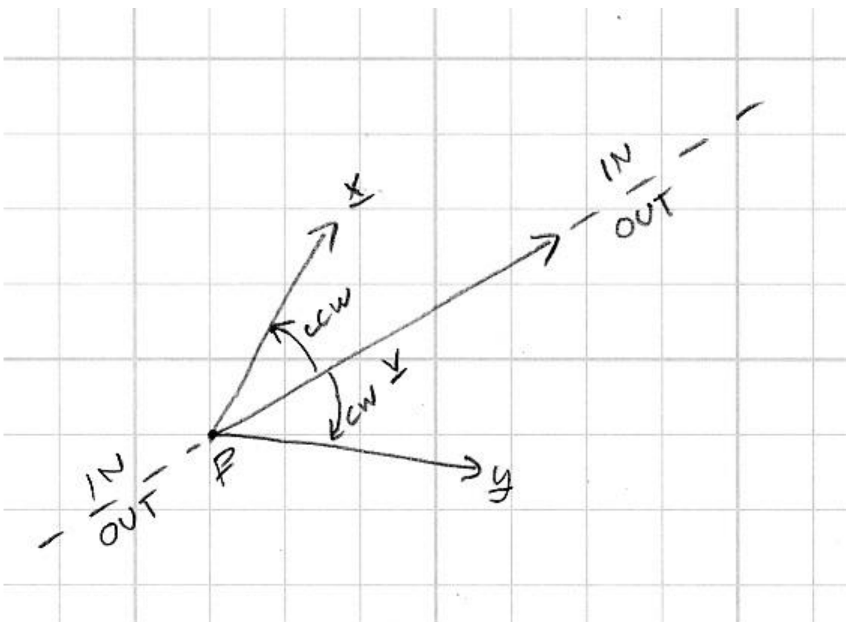
# Ray-triangle intersection

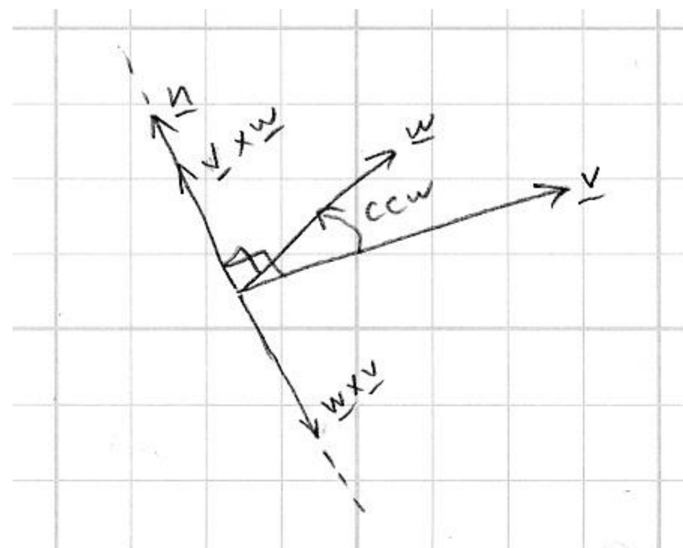
- In plane, triangle is the intersection of 3 half spaces



# Inside-edge test

- Need outside vs. inside
- Reduce to clockwise vs. counterclockwise
  - vector of edge to vector to **x**
- Use cross product to decide



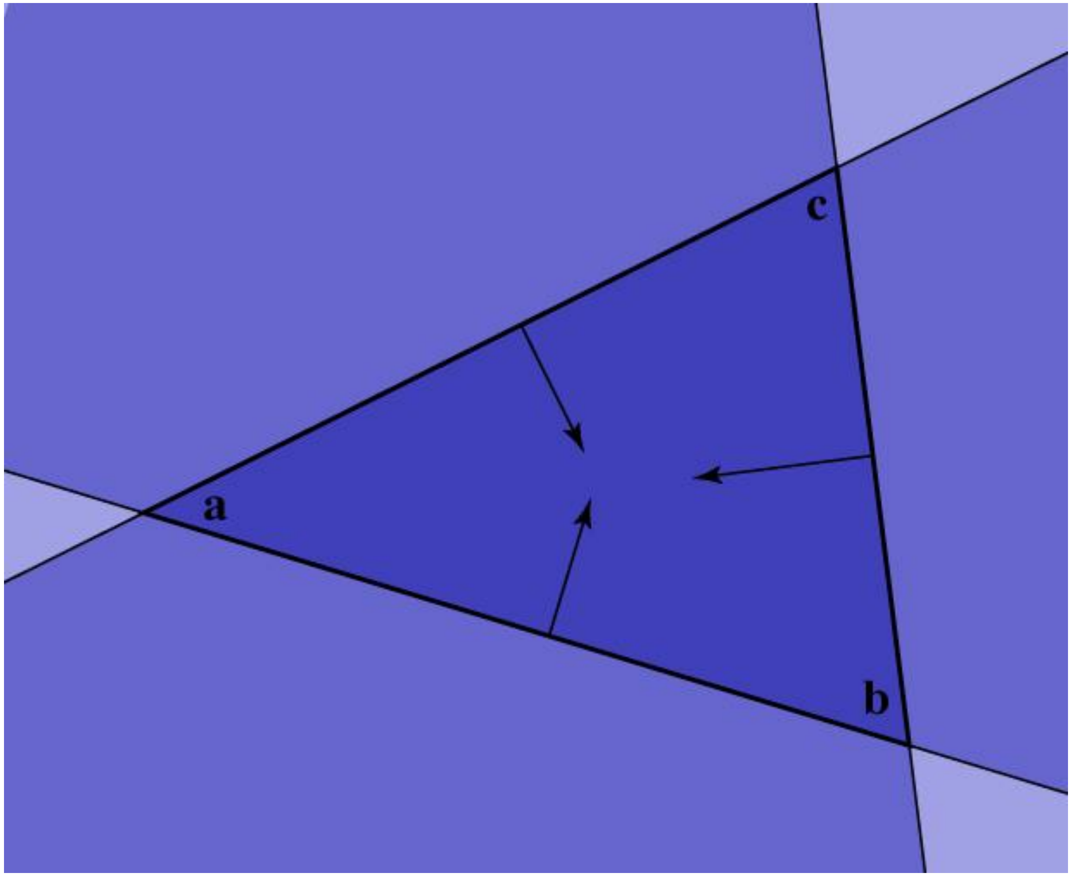



# Ray-triangle intersection

$$(\mathbf{b} - \mathbf{a}) \times (\mathbf{x} - \mathbf{a}) \cdot \mathbf{n} > 0$$

$$(\mathbf{c} - \mathbf{b}) \times (\mathbf{x} - \mathbf{b}) \cdot \mathbf{n} > 0$$

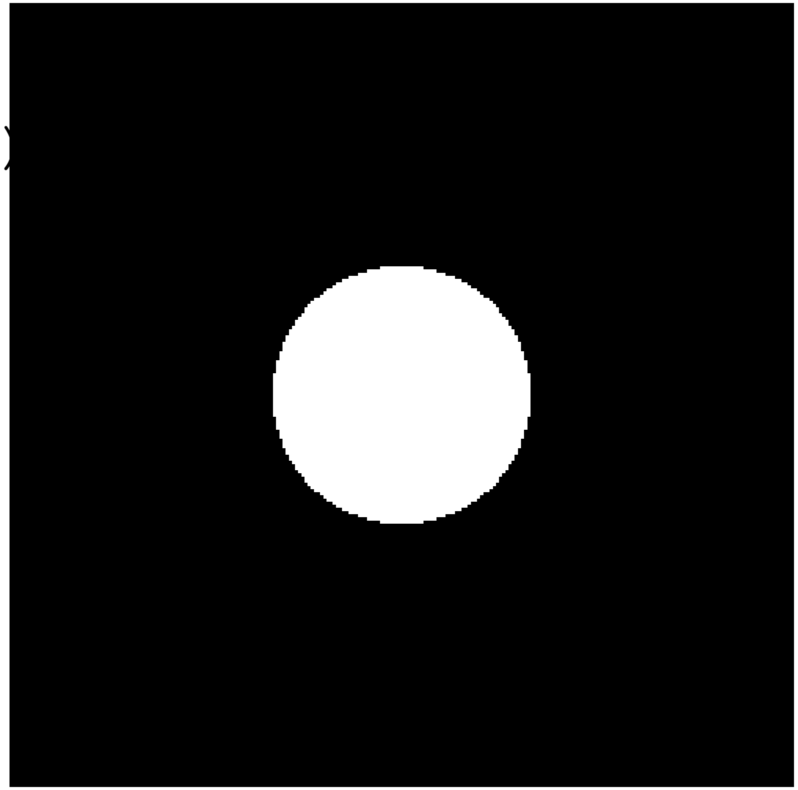
$$(\mathbf{a} - \mathbf{c}) \times (\mathbf{x} - \mathbf{c}) \cdot \mathbf{n} > 0$$



# Image so far

- With eye ray generation and sphere intersection

```
Surface s = new Sphere((0.0, 0.0, 0.0), 1.0);
for 0 <= iy < ny
    for 0 <= ix < nx {
        ray = camera.getRay(ix, iy);
        hitSurface, t = s.intersect(ray, 0, +inf)
        if hitSurface is not null
            image.set(ix, iy, white);
    }
```



# Intersection against many shapes

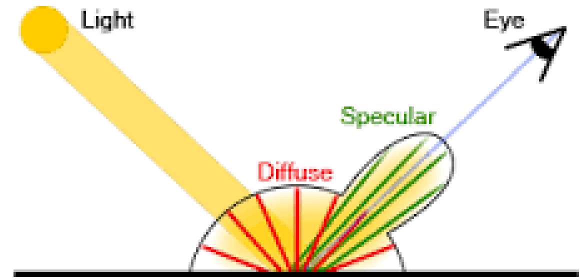
```
Group.intersect (ray, tMin, tMax) {
    tBest = +inf; firstSurface = null;
    for surface in surfaceList {
        hitSurface, t = surface.intersect(ray, tMin, tBest);
        if hitSurface is not null {
            tBest = t;
            firstSurface = hitSurface;
        }
    }
return hitSurface, tBest;
}
```

# Image so far

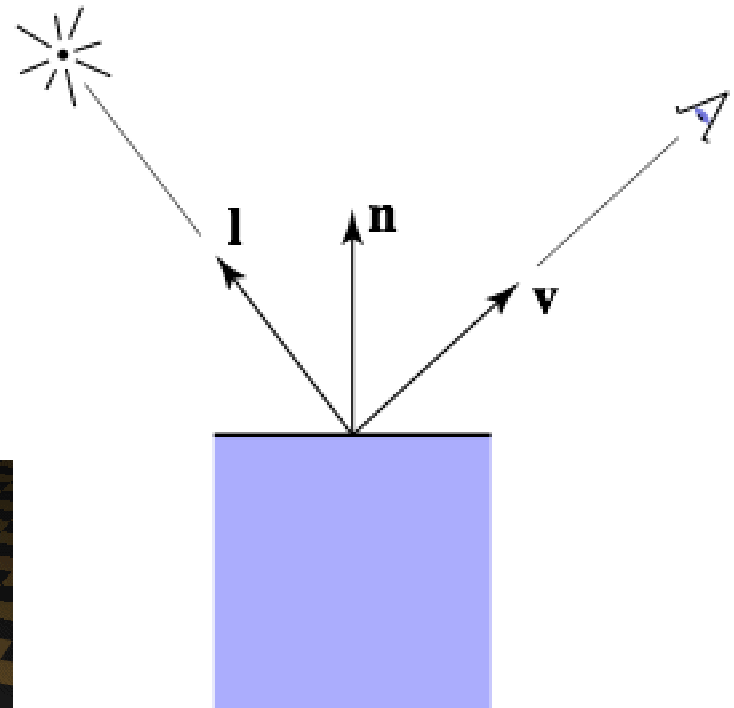
- With eye ray generation and scene intersection

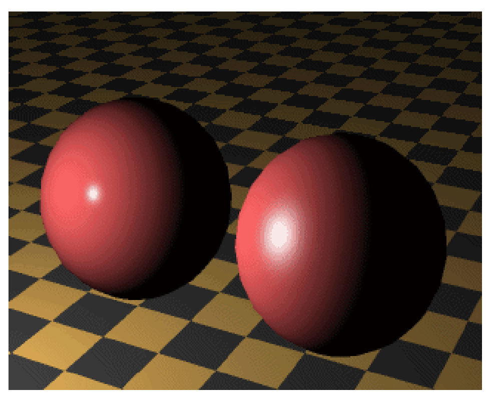
```
for 0 <= iy < ny
    for 0 <= ix < nx {
        ray = camera.getRay(ix, iy);
        c = scene.trace(ray, 0, +inf);
        image.set(ix, iy, c);
    }

...

Scene.trace(ray, tMin, tMax) {
    surface, t = surfs.intersect(ray, tMin, tMa
    if (surface != null) return surface.color()
    else return black;
}
```

# Shading




- Compute light reflected toward camer
- Inputs:
  - eye direction
  - light direction
    (for each of many lights)
  - surface normal
  - surface parameters
    (color, shininess, …)

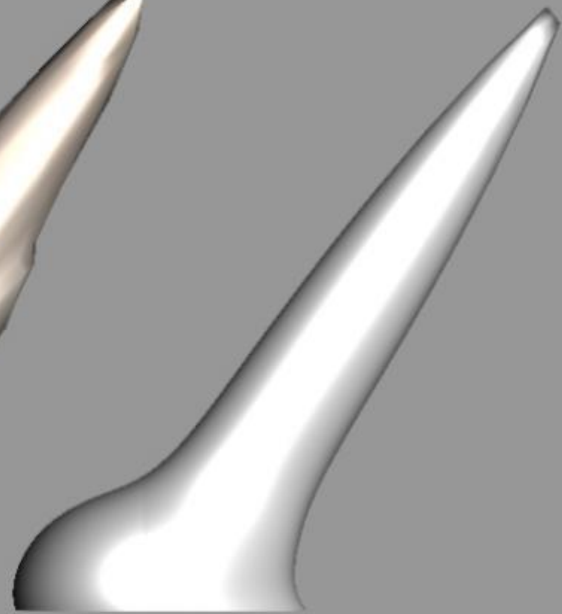




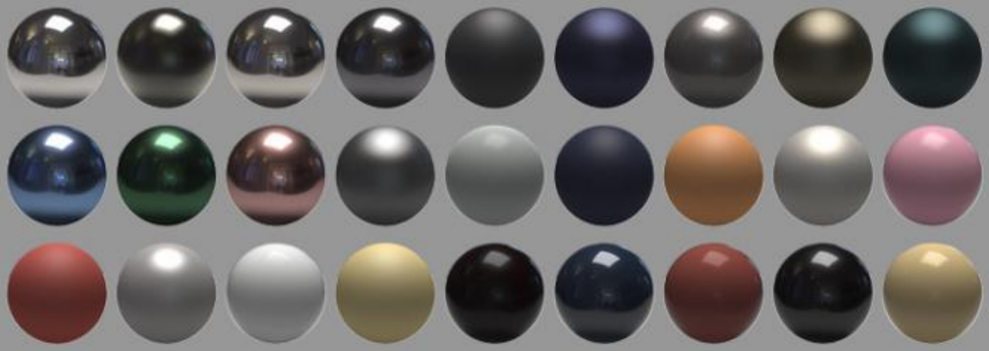
 

measured BRDF



approximated with
PBR layered Material

Mantra render



all Presets are either "eyeballed" from measured BRDF or physical measurement of IOR (Glass) and komplex IOR (Metals)

# Diffuse reflection

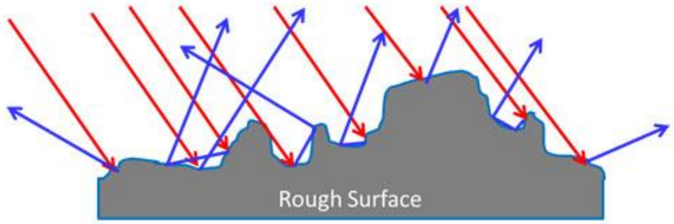
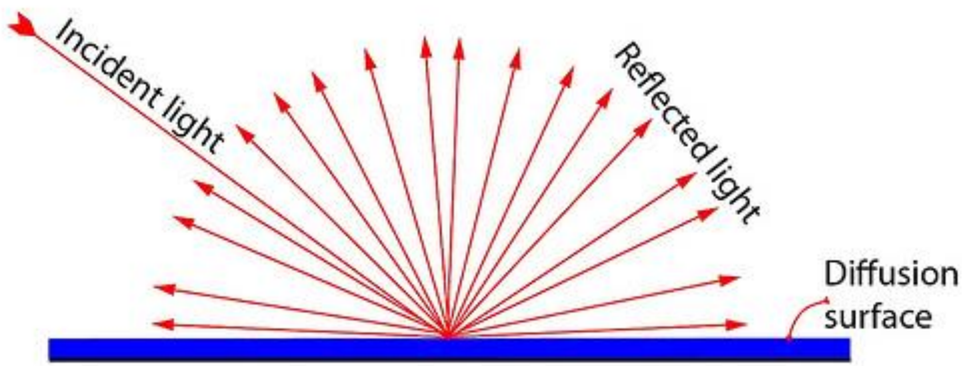
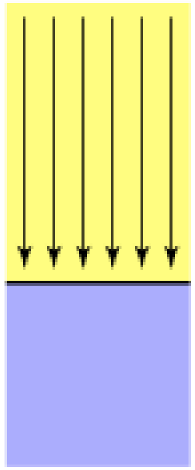


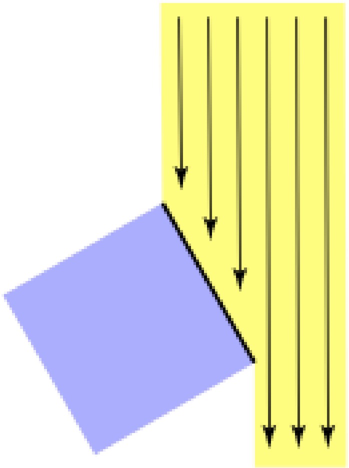

Diagram showing "Diffuse Reflection"



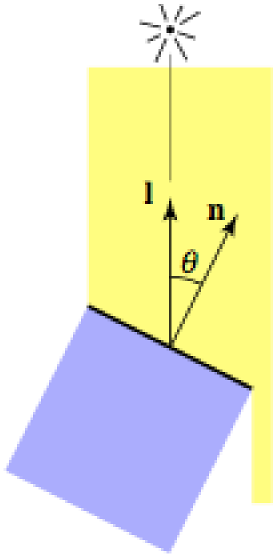

Incident light striking a perfect 'Diffusion Surface' creates a hemisphere of even illumination around the strike point.



Top face of cube receives a certain amount of light

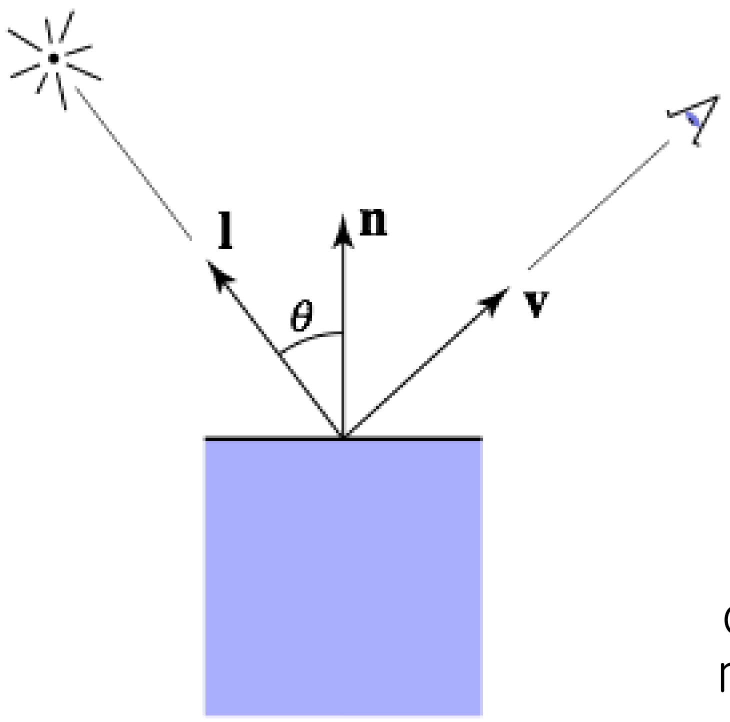


Top face of 60º rotated cube intercepts half the light




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

# Lambertian shading




illumination
from source

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

diffuse
coefficient

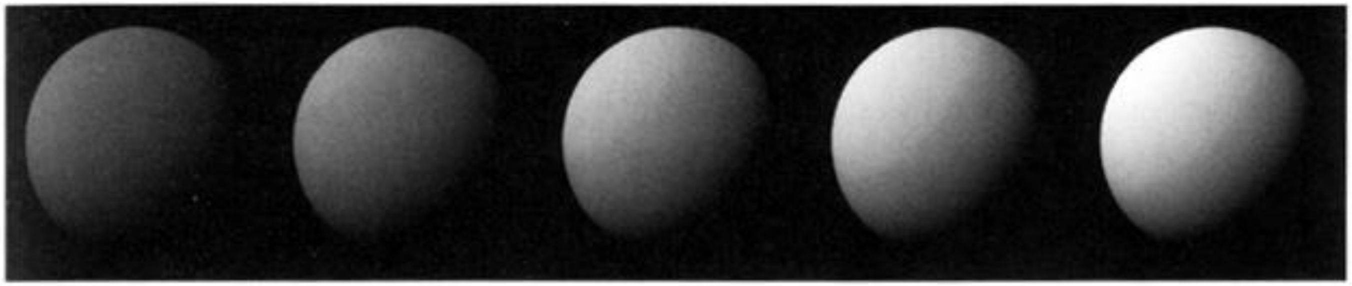
diffusely
reflected
light



# Lambertian shading

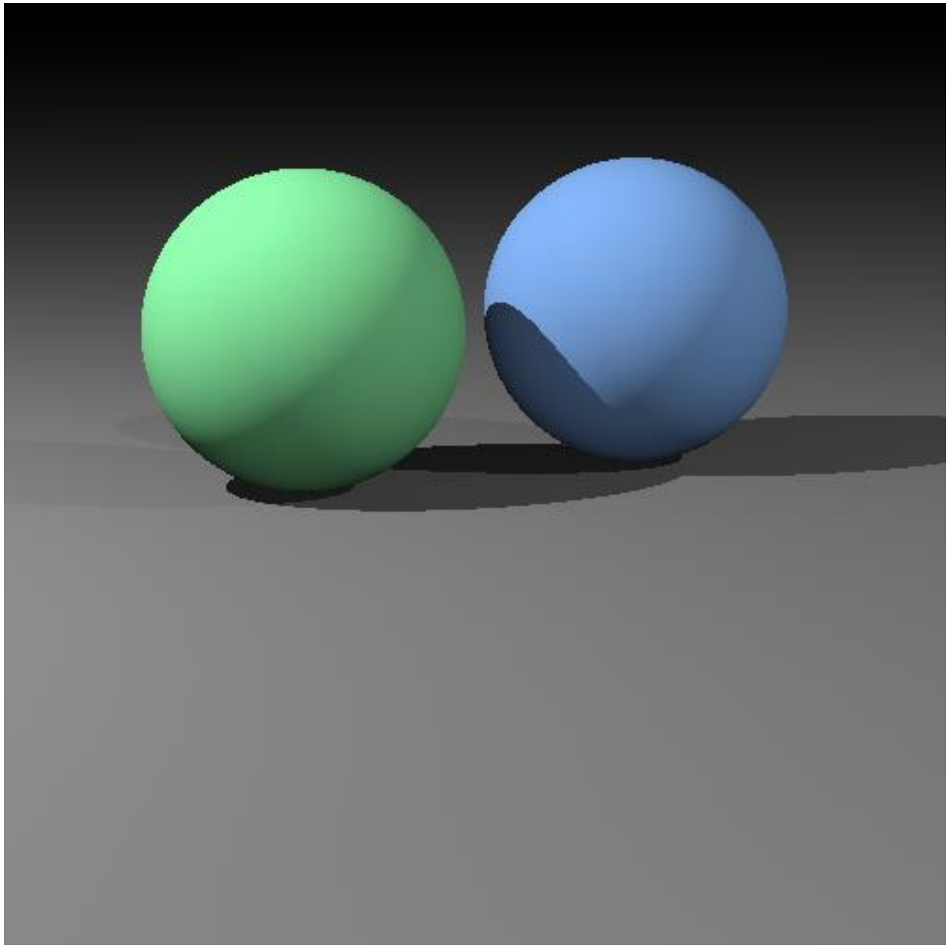
- Produces matte appearance



$k_d \longrightarrow$

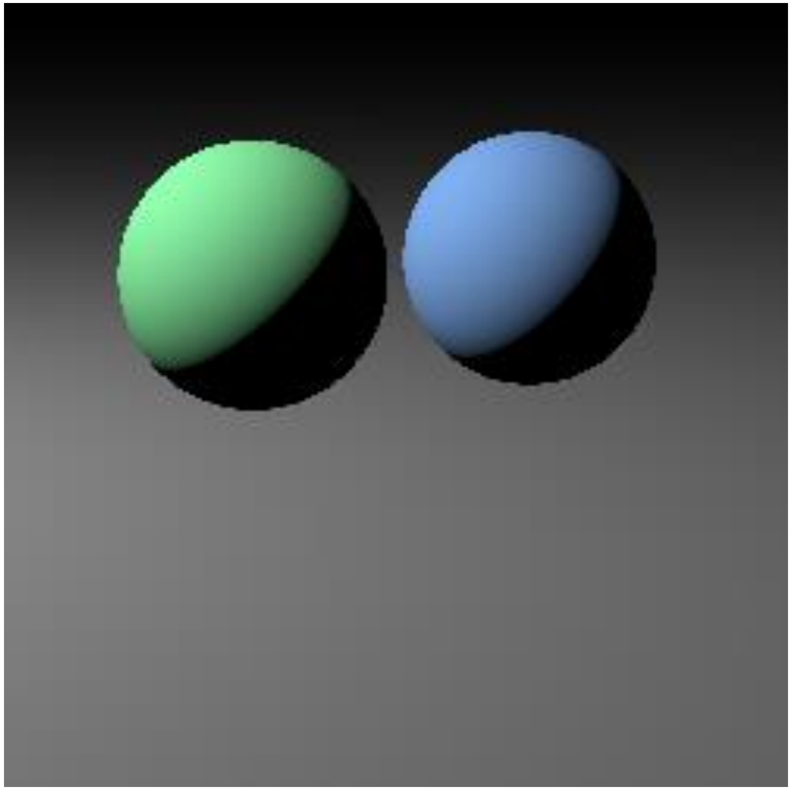
[Foley et al.]

# Diffuse shading



# Image so far

```
Scene.trace(Ray ray, tMin, tMax) {
    surface, t = hit(ray, tMin, tMax);
    if surface is not null {
        point = ray.evaluate(t);
        normal = surface.getNormal(point);
        return surface.shade(ray, point,
            normal, light);
    }
    else return backgroundColor;
}

...

Surface.shade(ray, point, normal, light) {
    v = –normalize(ray.direction);
    l = normalize(light.pos – point);
    // compute shading
}
```

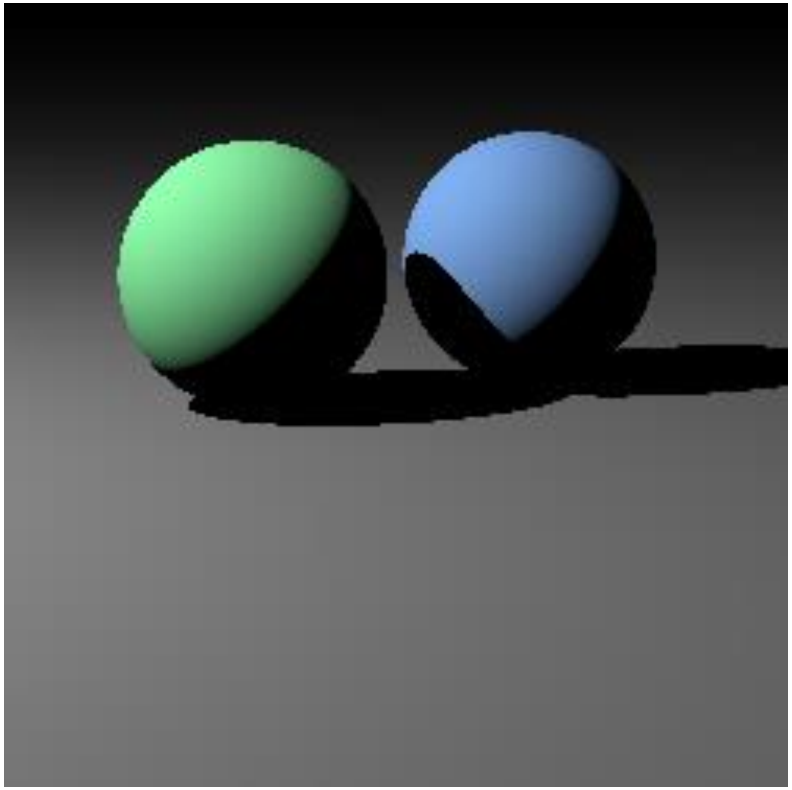


# Shadows

- Surface is only illuminated if nothing blocks its view of the light.
- With ray tracing it's easy to check
  - just intersect a ray with the scene!

# Image so far

```
Surface.shade(ray, point, normal, light) {
    shadRay = (point, light.pos – point);
    if (shadRay not blocked) {
        v = –normalize(ray.direction);
        l = normalize(light.pos – point);
        // compute shading
    }
    return black;
}
```

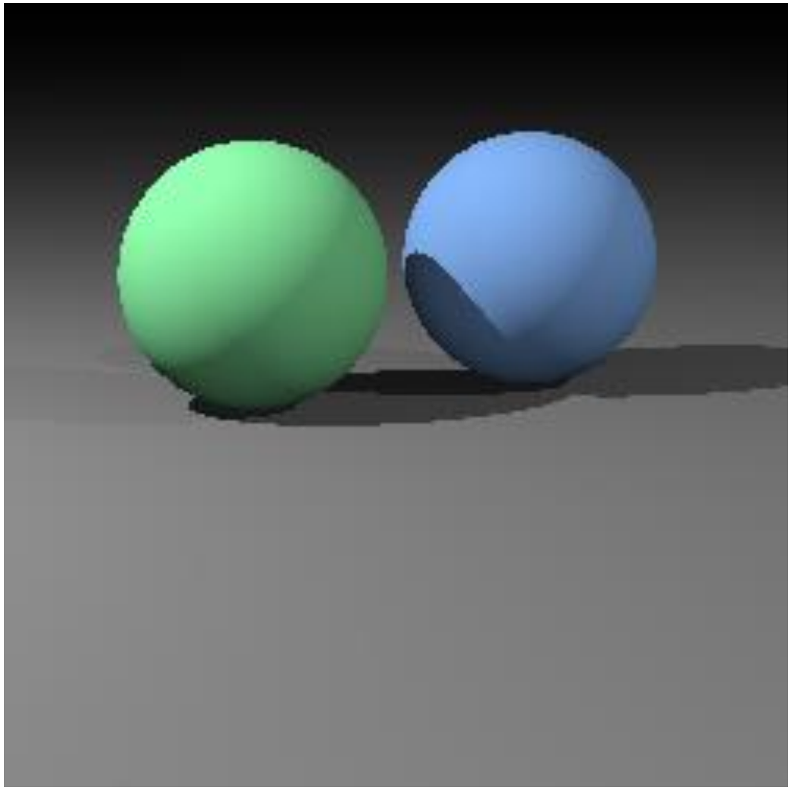


# Multiple lights

- Important to fill in black shadows
- Just loop over lights, add contributions
- Ambient shading
  - black shadows are not really right
  - add a constant “ambient” color to the shading…

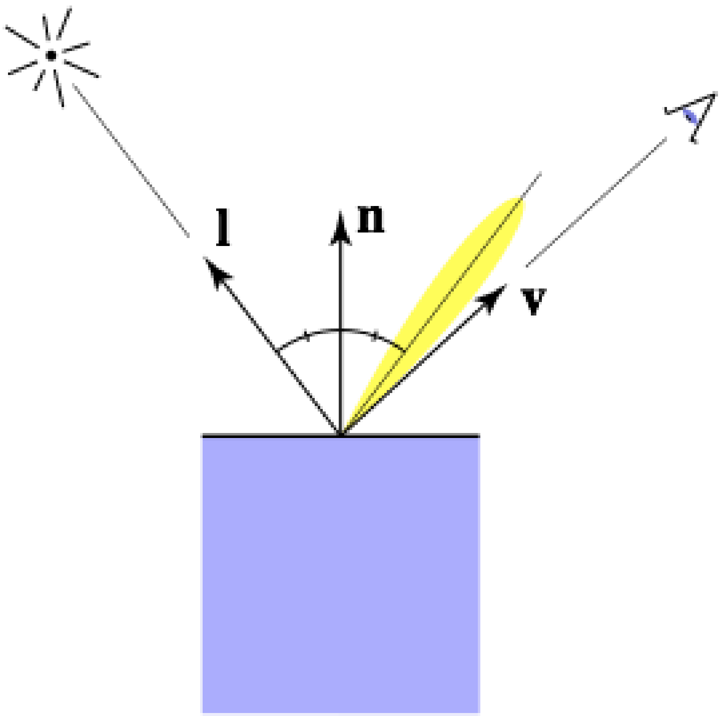
# Image so far

```
shade(ray, point, normal, lights) {
    result = ambient;
    for light in lights {
        if (shadow ray not blocked) {
            result += shading contribution;
        }
    }
    return result;
}
```



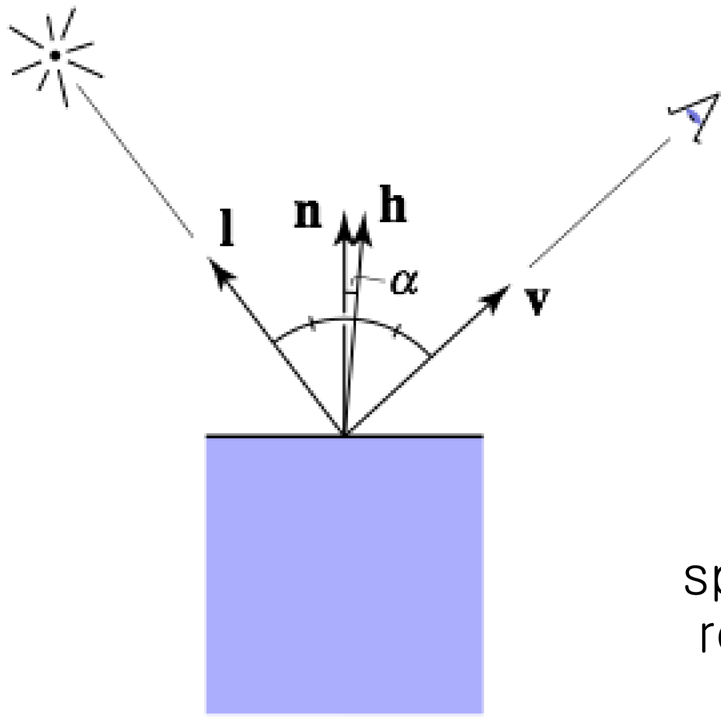
# Specular shading (Blinn-Phong)

- Intensity depends on view direction
  - bright near mirror configuration




# Specular shading (Blinn-Phong)

- Close to mirror ⇔ half vector near normal
  - Measure “near” by dot product of unit vectors




$$\mathbf{h} = \mathrm{bisector}(\mathbf{v}, \mathbf{l}) = \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|}$$

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

specularly reflected light ($L_s$)

specular coefficient ($k_s$)

# Phong model—plots

- Increasing *n* narrows the lobe

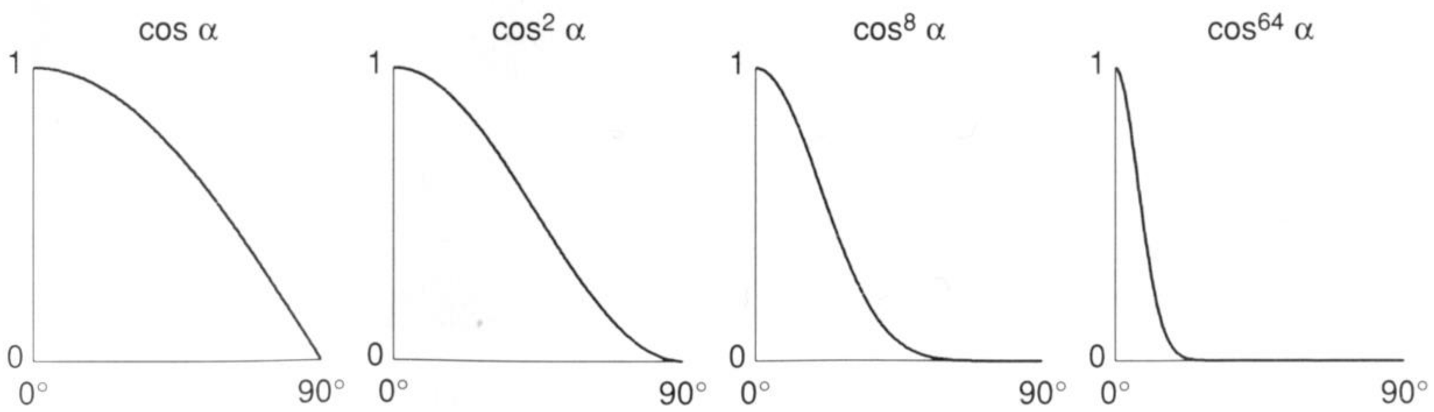
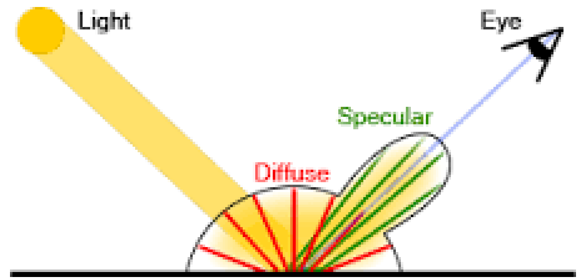


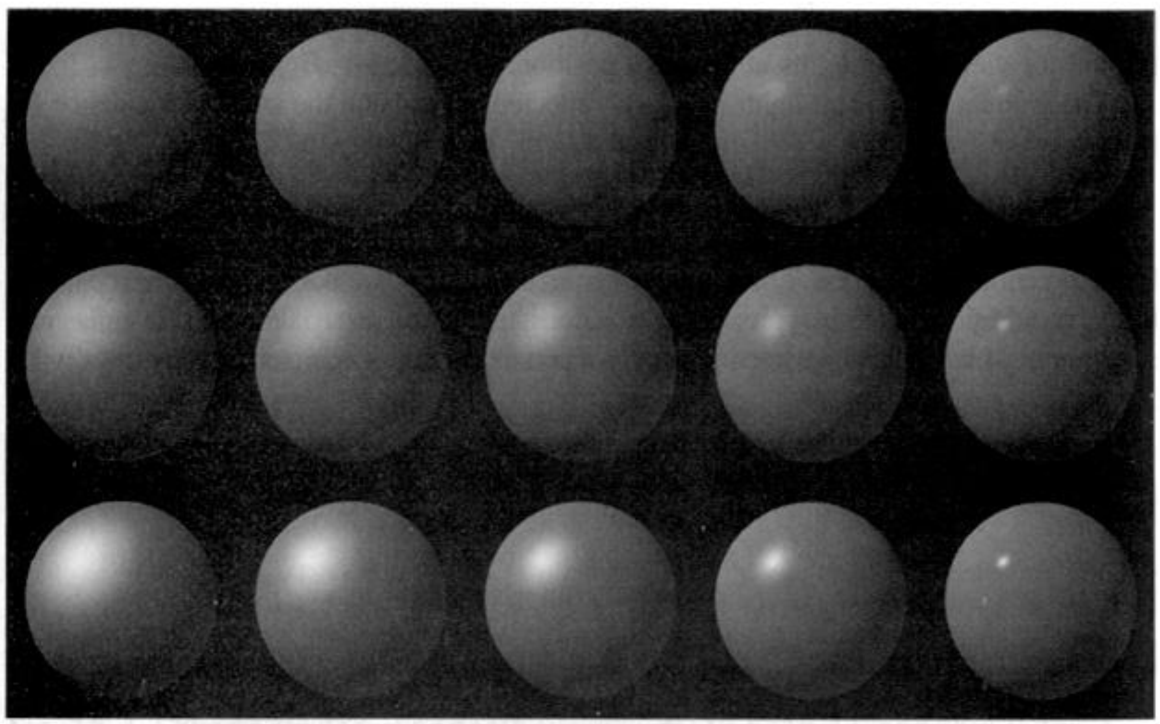

**Fig. 16.9** Different values of $\cos^n \alpha$ used in the Phong illumination model.

[Foley et al.]

# Specular shading




$k_s$



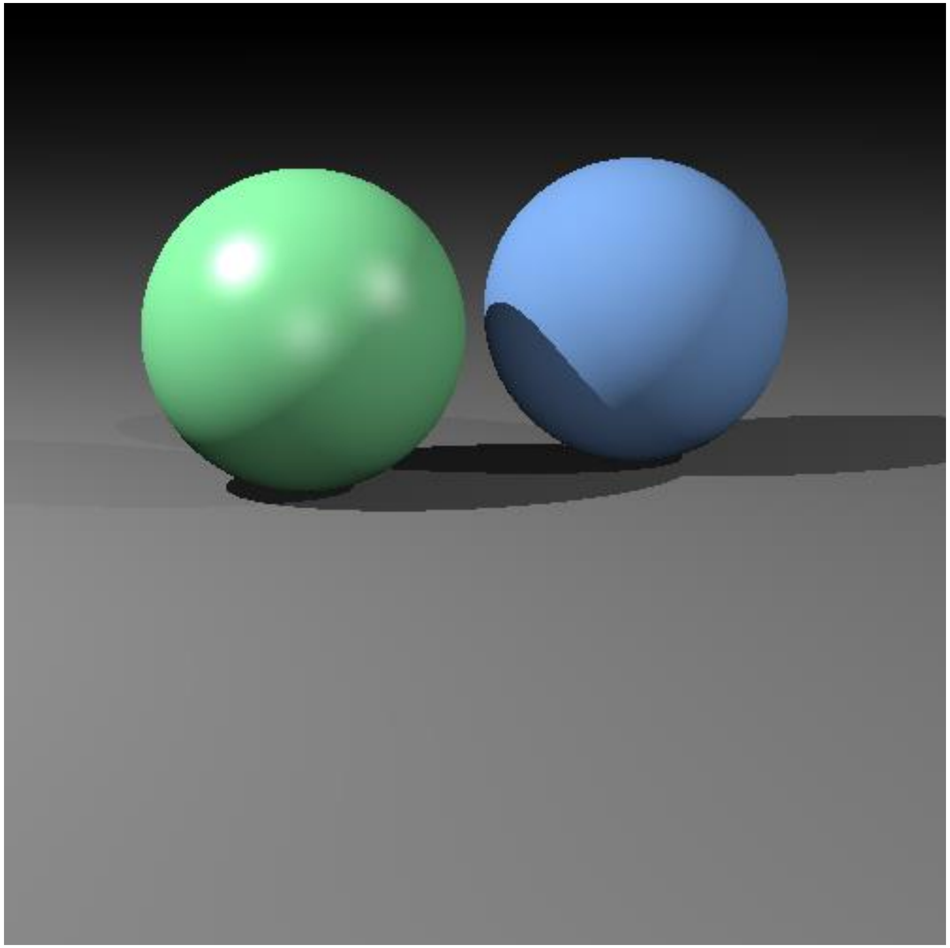
$p$

[Foley et al.]



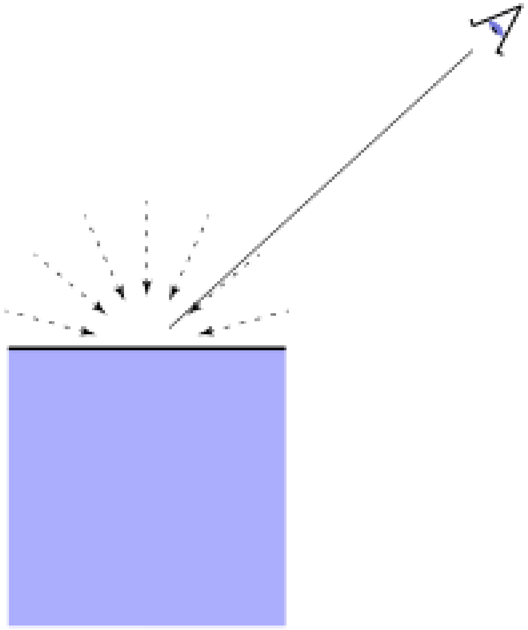
 

# Diffuse + Phong shading



# Ambient shading

- Shading that does not depend on anything
  - add constant color to account for disregarded illumination and fill in black shadows



$$L_a = k_a \, I_a$$

ambient coefficient

reflected ambient light

# Putting it together

- Usually include ambient, diffuse, Phong in one model

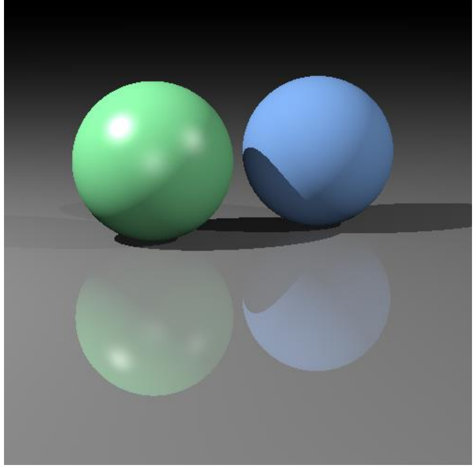
$$
\begin{aligned}
L &= L_a + L_d + L_s \\
&= k_a\, I_a + k_d\, I \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s\, I \max(0, \mathbf{n} \cdot \mathbf{h})^p
\end{aligned}
$$

- The final result is the sum over many lights

$$
L = L_a + \sum_{i=1}^{N} \left[ (L_d)_i + (L_s)_i \right]
$$

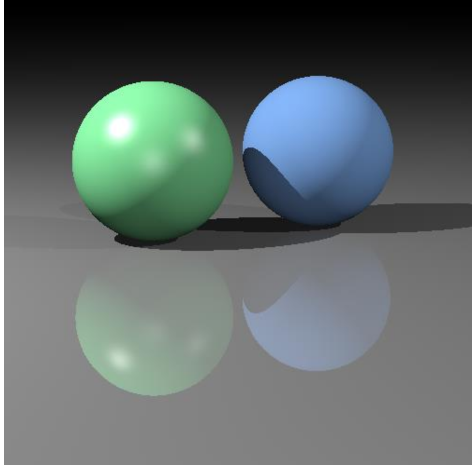
$$
L = k_a\, I_a + \sum_{i=1}^{N} \left[ k_d\, I_i \max(0, \mathbf{n} \cdot \mathbf{l}_i) + k_s\, I_i \max(0, \mathbf{n} \cdot \mathbf{h}_i)^p \right]
$$

# Mirror reflection

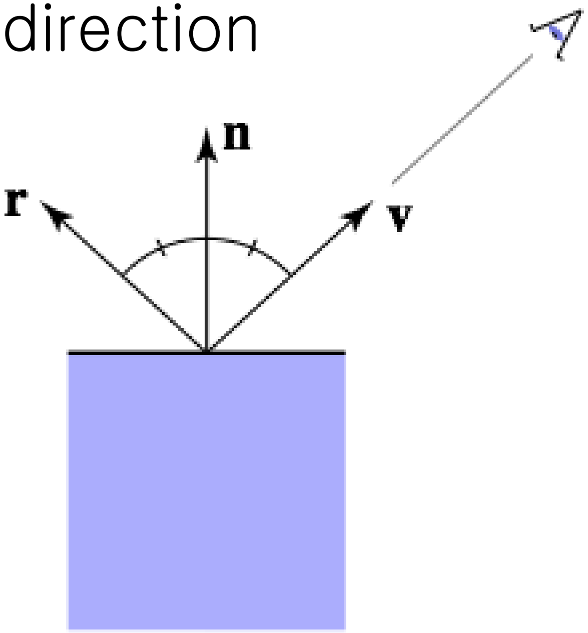


- Consider perfectly shiny surface
  - there isn't a highlight
  - instead there's a reflection of other objects
- Can render this using recursive ray tracing
  - to find out mirror reflection color, ask what color is seen from surface point in reflection direction
  - already computing reflection direction for Phong…
- "Glazed" material has mirror reflection and diffu$L = L_a + L_d + L_m$
  - where $L_m$ is evaluated by tracing a new ray

# Mirror reflection

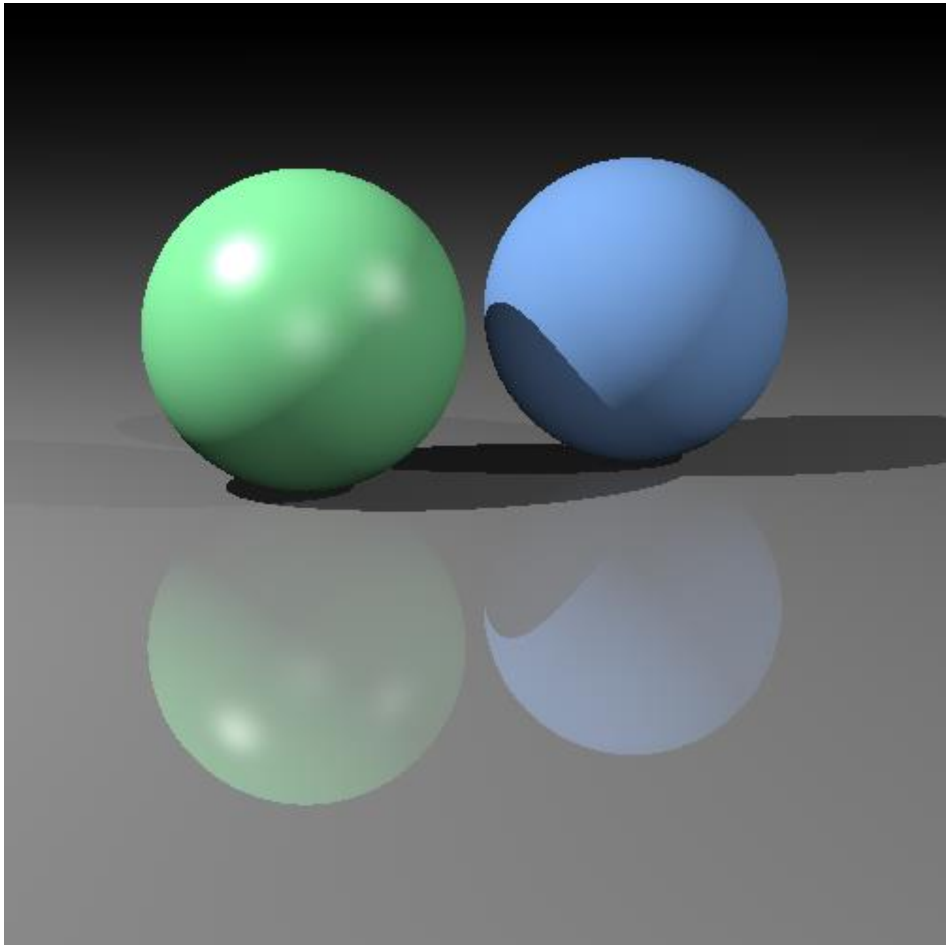


- Intensity depends on view direction
  - reflects incident light from mirror direction




$$\mathbf{r} = \mathbf{v} + 2((\mathbf{n} \cdot \mathbf{v})\mathbf{n} - \mathbf{v})$$
$$= 2(\mathbf{n} \cdot \mathbf{v})\mathbf{n} - \mathbf{v}$$

# Diffuse + mirror reflection (glazed)



(glazed material on floor)