Basic Ray Tracing

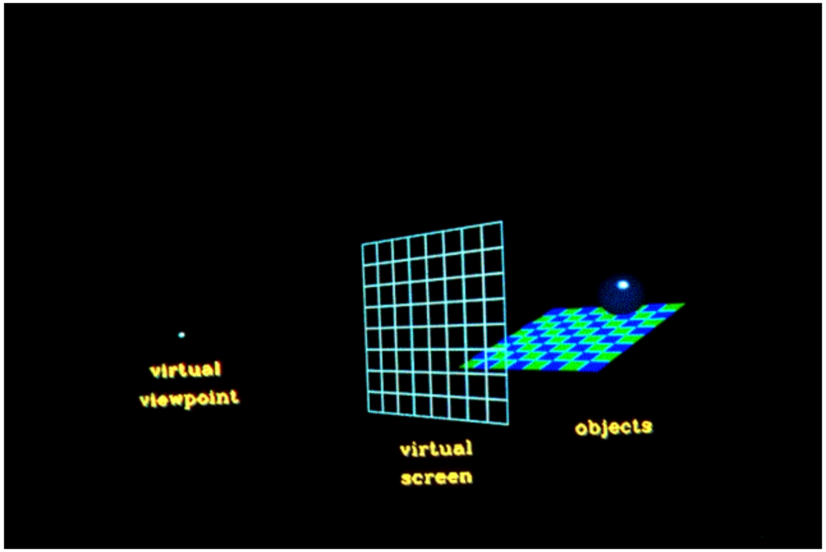
CMSC 435/634

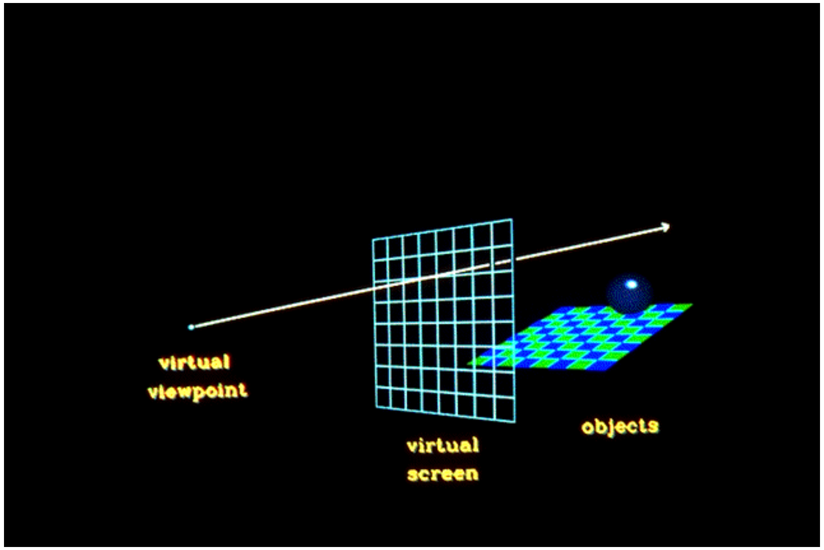
Visibility Problem

- Rendering: converting a model to an image
- Visibility: deciding which objects (or parts) will appear in the image
  - Object-order
    - OpenGL (later)
  - Image-order
    - Ray Tracing (now)

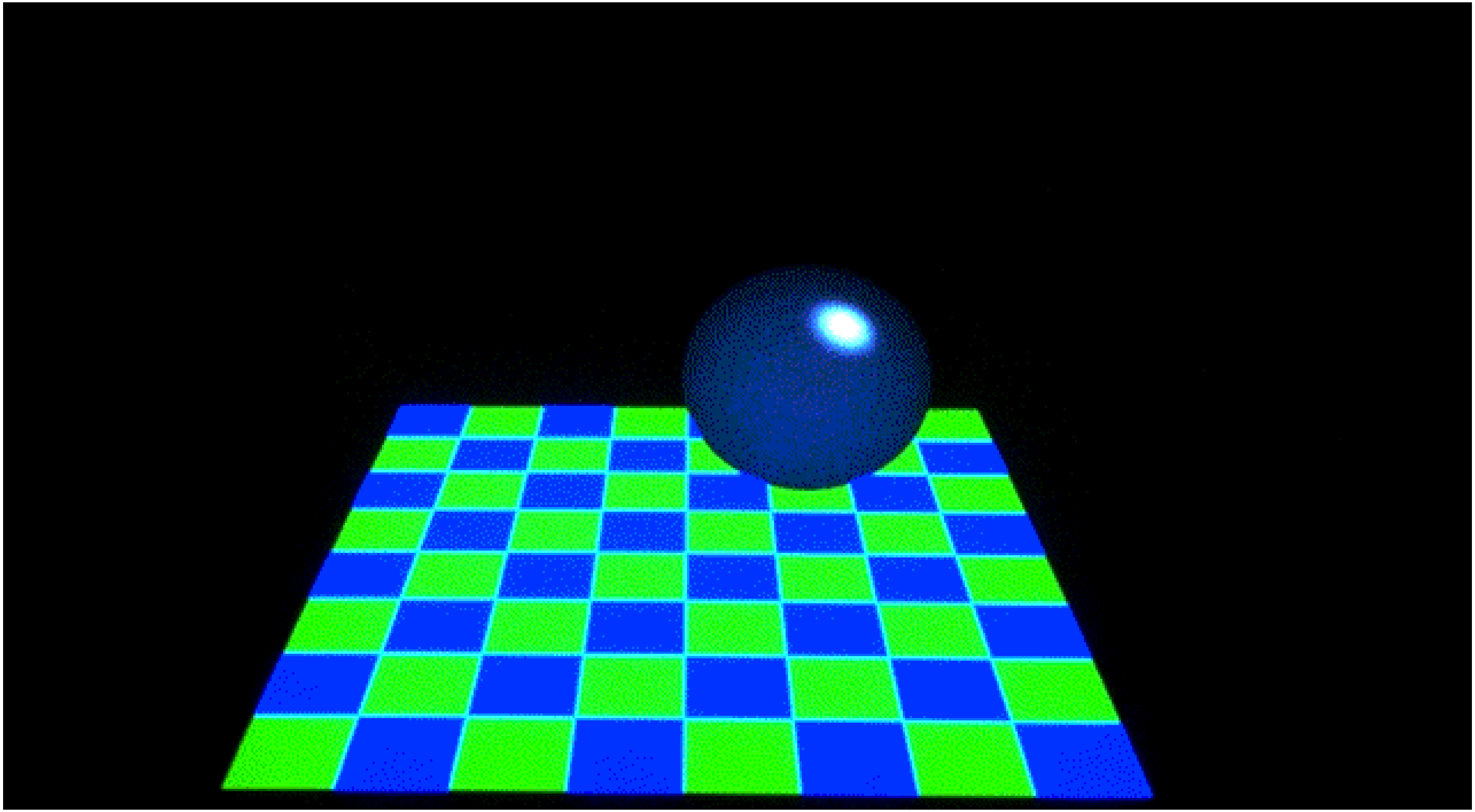
Raytracing

- Given
  - Scene
  - Viewpoint
  - Viewplane
- Cast ray from viewpoint through pixels into scene





View



Computing Viewing Rays

- Parametric ray

  $\vec{p}(t) = \vec{e} + t(\vec{s} - \vec{e})$

- Camera frame

  $\vec{e}$: eye point

  $\vec{u}, \vec{v}, \vec{w}$: basis vectors

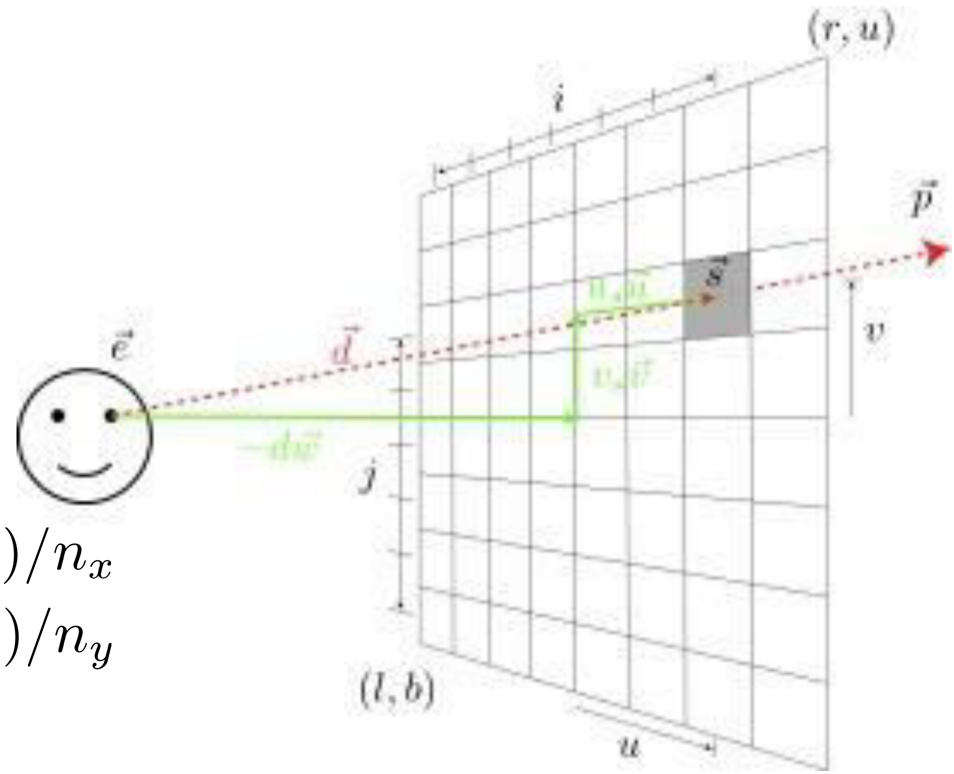
  - right, up, **backward**
    - Right hand rule!

- Screen position

  $u_s = left + (right - left)(i + 0.5)/n_x$

  $v_s = top + (bottom - top)(j + 0.5)/n_y$

  $\vec{s} = \vec{e} + u_s\vec{u} + v_s\vec{v} - d\,\vec{w}$



Calculating Intersections

- Define ray parametrically:

$$\vec{p} = \vec{e} + t\ (\vec{s} - \vec{e})$$

$$\begin{aligned} x &= e_z + t\ (s_x - e_x) &= e_x + t\ d_x \\ y &= e_z + t\ (s_y - e_y) &= e_y + t\ d_y \\ z &= e_z + t\ (s_z - e_z) &= e_z + t\ d_z \end{aligned}$$

- If (e_x, e_y, e_z) is center of projection and (s_x, s_y, s_z) is center of pixel, then

  $0 \leq t \leq 1$: points between those locations

  $t < 0$: points behind viewer

  $t > 1$: points beyond view window

Ray-Sphere Intersection

- Sphere in vector form

$$f(\vec{p}) = (\vec{p} - \vec{c}) \cdot (\vec{p} - \vec{c}) - r^2 = 0$$

- Ray

$$\vec{p}(t) = \vec{e} + t\,\vec{d}$$

- Intersection when

$$f(\vec{p}(t)) = 0$$

$$((\vec{e} + t\,\vec{d}) - \vec{c}) \cdot ((\vec{e} + t\,\vec{d}) - \vec{c}) - r^2 = 0$$

$$(t\,\vec{d} + \overrightarrow{ec}) \cdot (t\,\vec{d} + \overrightarrow{ec}) - r^2 = 0$$

$$\vec{d} \cdot \vec{d}\, t^2 + 2\vec{d} \cdot \overrightarrow{ec}\ t + (\overrightarrow{ec} \cdot \overrightarrow{ec} - r^2) = 0$$

$$t = \frac{-\vec{d} \cdot \overrightarrow{ec} \pm \sqrt{(\vec{d} \cdot \overrightarrow{ec})^2 - \vec{d} \cdot \vec{d}(\overrightarrow{ec} \cdot \overrightarrow{ec} - r^2)}}{\vec{d} \cdot \vec{d}}$$

Ray-Polygon Intersection

- Given ray and plane containing polygon

$$\vec{p}(t) = \vec{e} + t\,\vec{d}$$

$$f(\vec{p}) = \vec{n} \cdot \vec{p} - \vec{n} \cdot \vec{p}_0 = 0$$

- What is ray/plane intersection?

$$f(\vec{p}(t)) = \vec{n} \cdot (\vec{e} + t\,\vec{d}) - \vec{n} \cdot \vec{p}_0 = 0$$

$$t = \frac{\vec{n} \cdot \vec{p}_0 - \vec{n} \cdot \vec{e}}{\vec{n} \cdot \vec{d}}$$

- Is intersection point inside polygon?

Ray-Triangle Intersection

- Intersection of ray with *barycentric* triangle

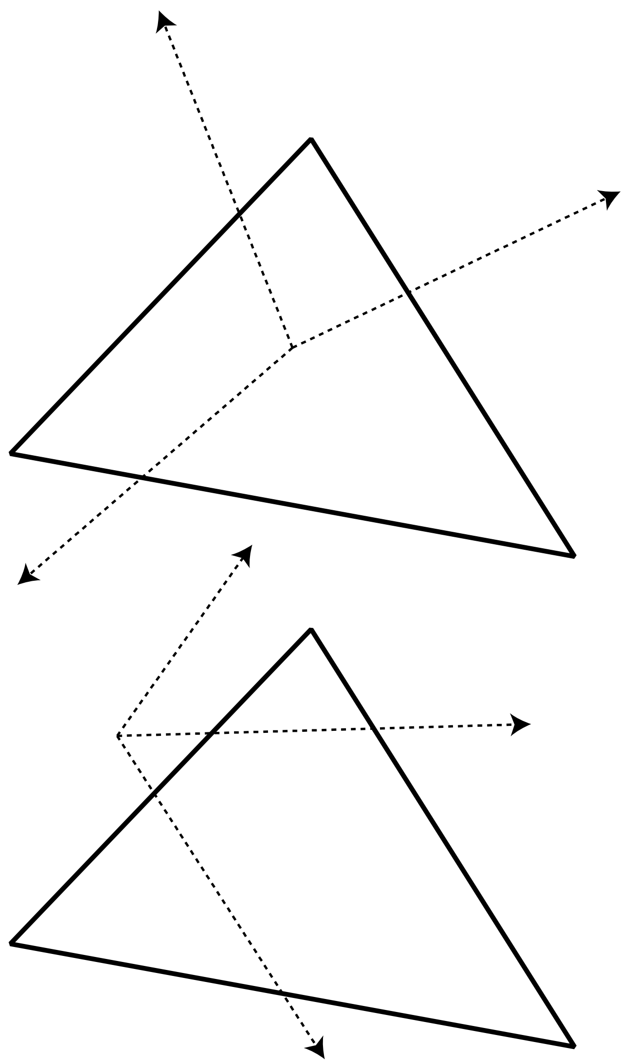
$$\vec{p} = \vec{e} + t\vec{d} = \alpha\vec{p}_0 + \beta\vec{p}_1 + \gamma\vec{p}_2 \qquad \alpha, \beta, \gamma > 0;\ \alpha + \beta + \gamma = 1$$

  - In triangle if $\alpha \geq 0$, $\beta \geq 0$, $\gamma \geq 0$
  - To avoid computing all three, can replace $\alpha \geq 0$ with $\beta + \gamma \leq 1$

```
boolean raytri (ray r, vector p0, p1, p2, interval [t0,t1] )
{
    compute t
    if (( t < t0 ) or (t > t1))
        return ( false )
    compute γ
    if ((γ < 0 ) or (γ > 1))
        return ( false )
    compute β
    if ((β < 0 ) or (β+γ > 1))
        return ( false )
   return true
}
```

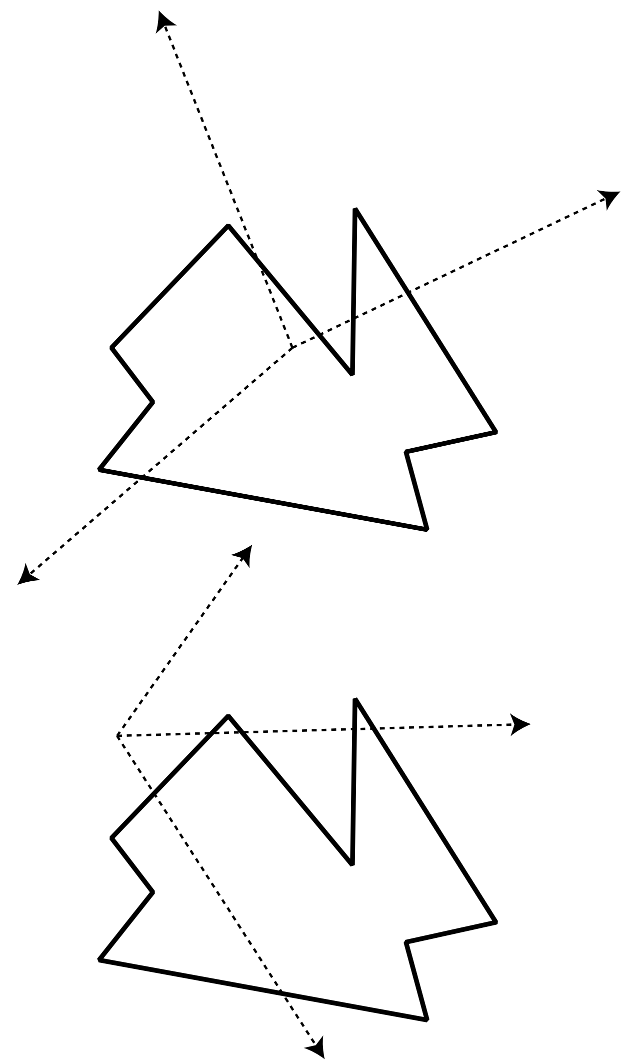
Point in Polygon?

- Is P in polygon?
- Cast ray from P to infinity
  - 1 crossing = inside
  - 0, 2 crossings = outside

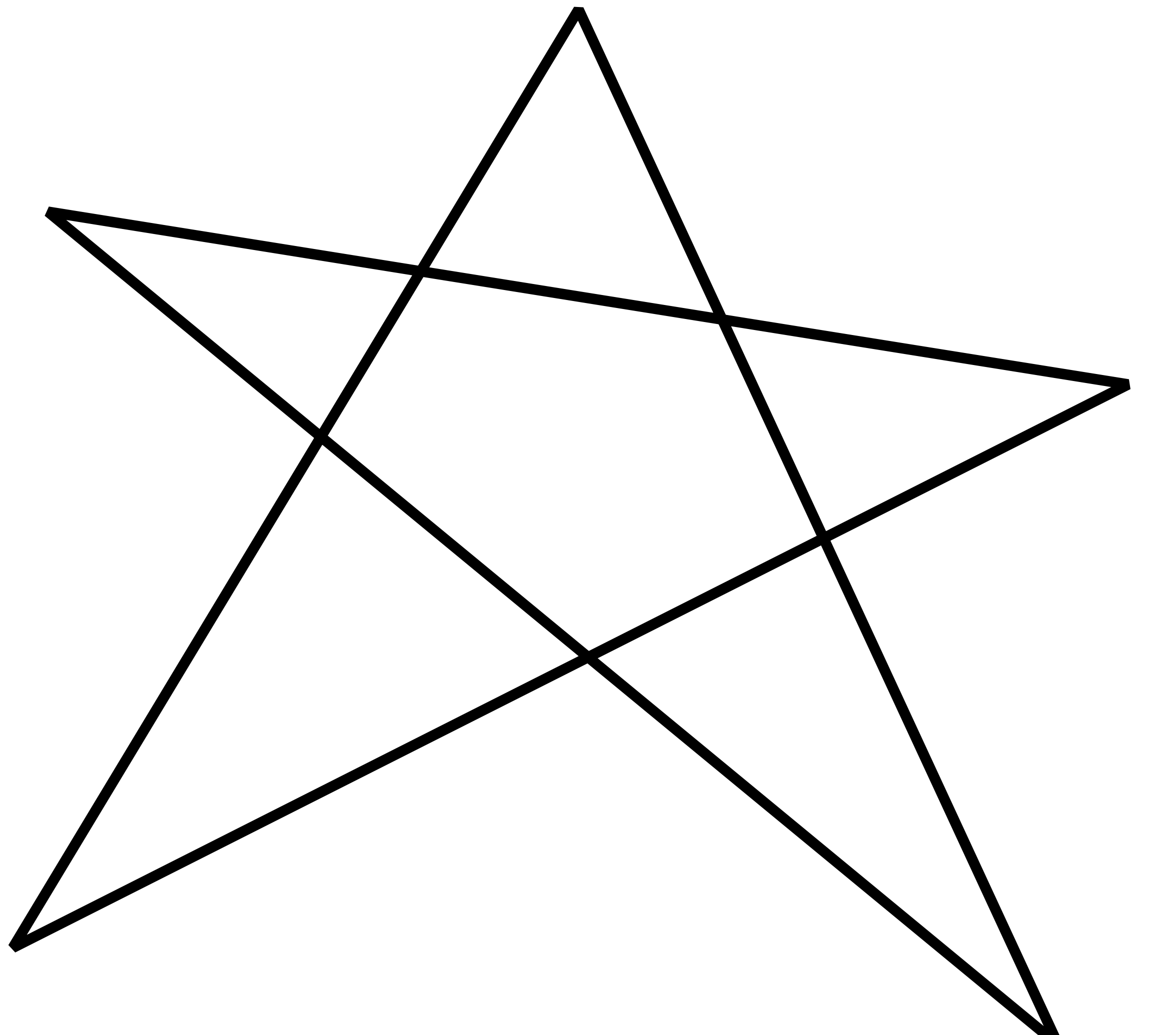


Point in Polygon?

- Is P in concave polygon?
- Cast ray from P to infinity
  - Odd crossings = inside
  - Even crossings = outside



What Happens?



Raytracing Characteristics

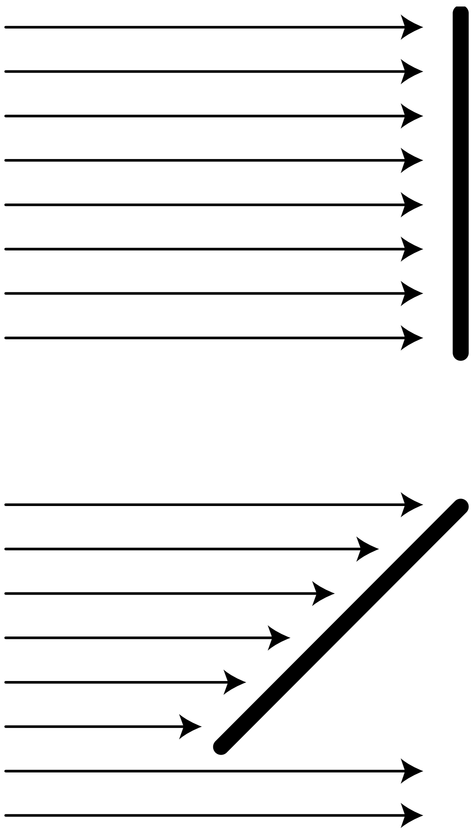
- Good
  - Simple to implement
  - Minimal memory required
  - Easy to extend
- Bad
  - Aliasing
  - Computationally intensive
    - Intersections expensive (75-90% of rendering time)
    - Lots of rays

Basic Illumination Concepts

- Terms
  - Illumination: calculating light intensity at a point (object space; equation) based loosely on physical laws
  - Shading: algorithm for calculating intensities at pixels (image space; algorithm)
- Objects
  - Light sources: light-emitting
  - Other objects: light-reflecting
- Light sources
  - Point (special case: at infinity)
  - Area

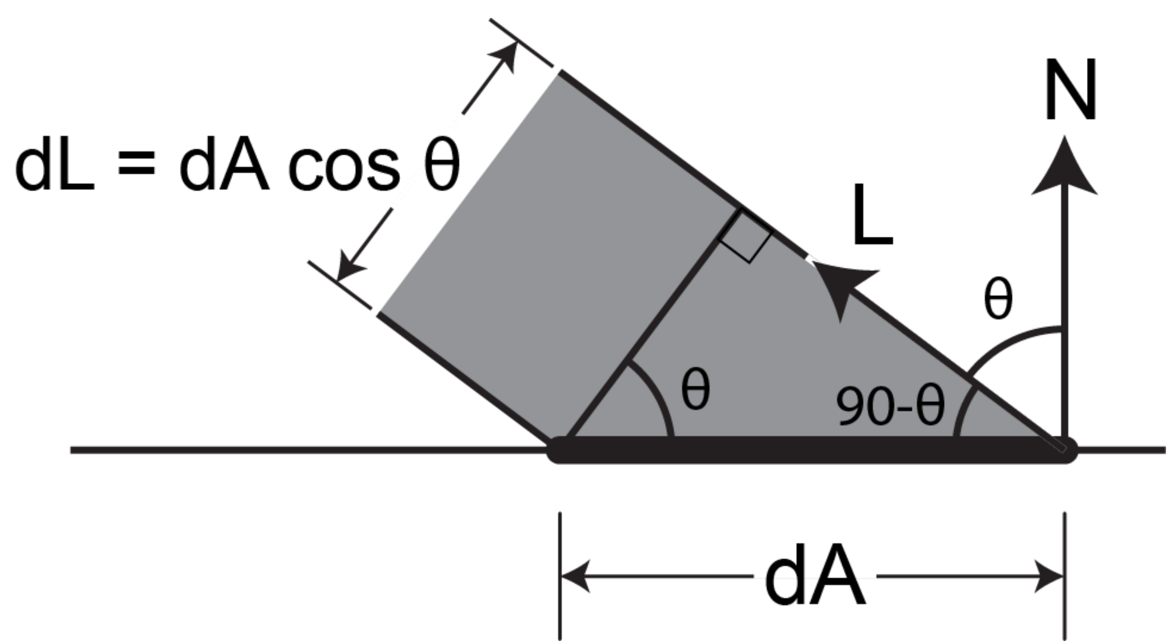
Lambert’s Law

- Intensity of reflected light related to orientation



Lambert’s Law

- Specifically: the radiant energy from any small surface area dA in any direction θ relative to the surface normal is proportional to $\cos\theta$



$$
\begin{aligned}
I_{\text{diff}} &= K_d I_l \cos\theta \\
&= K_d I_l (N \cdot L)
\end{aligned}
$$

$$I_{\text{diff}} = K_d I_l \max(0, N \cdot L)$$

Ambient Light

- Additional light bounces we're not counting
- Approximate them as a constant

I_a = Amount of extra light coming into this surface

K_a = Amount that bounces off of this surface

$$I_{\mathrm{amb}} = K_a I_a$$

Total extra light bouncing off this surface

Combined Model

$$\begin{aligned} I_{\text{total}} &= I_{\text{amb}} + I_{\text{diff}} \\ &= K_a I_a + K_d I_l \max(0, N \cdot L) \end{aligned}$$

Adding color:

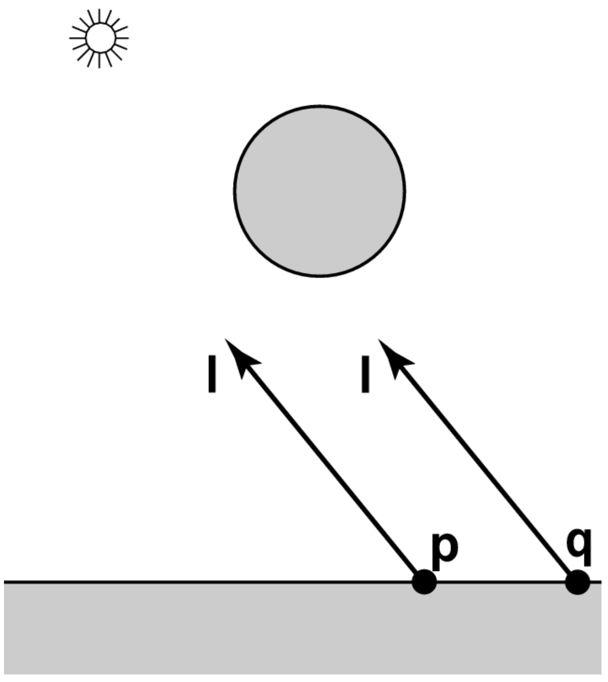
$$\begin{aligned} I_{\text{R}} &= K_{aR} I_{aR} + K_{dR} I_{lR} \max(0, N \cdot L) \\ I_{\text{G}} &= K_{aG} I_{aG} + K_{dG} I_{lG} \max(0, N \cdot L) \\ I_{\text{B}} &= K_{aB} I_{aB} + K_{dB} I_{lB} \max(0, N \cdot L) \end{aligned}$$

For any wavelength λ:

$$I_{\lambda} = K_{a\lambda} I_{a\lambda} + K_{d\lambda} I_{l\lambda} \max(0, N \cdot L)$$

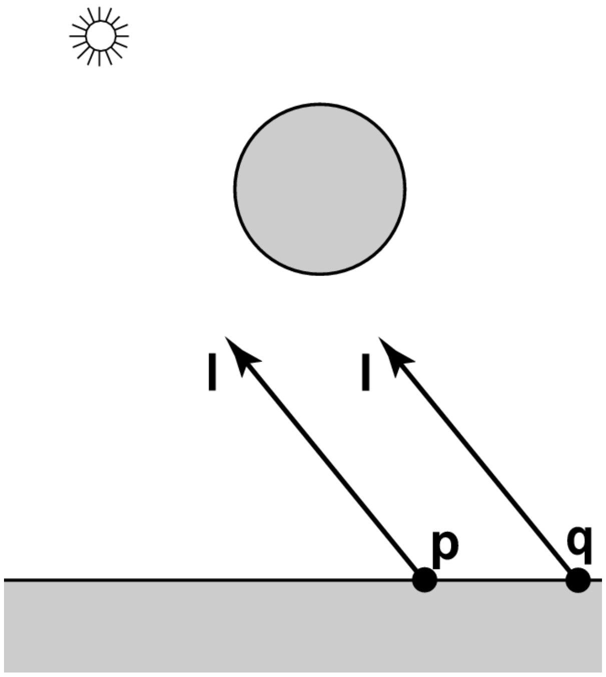
Shadows

- What if there is an object between the surface and light?

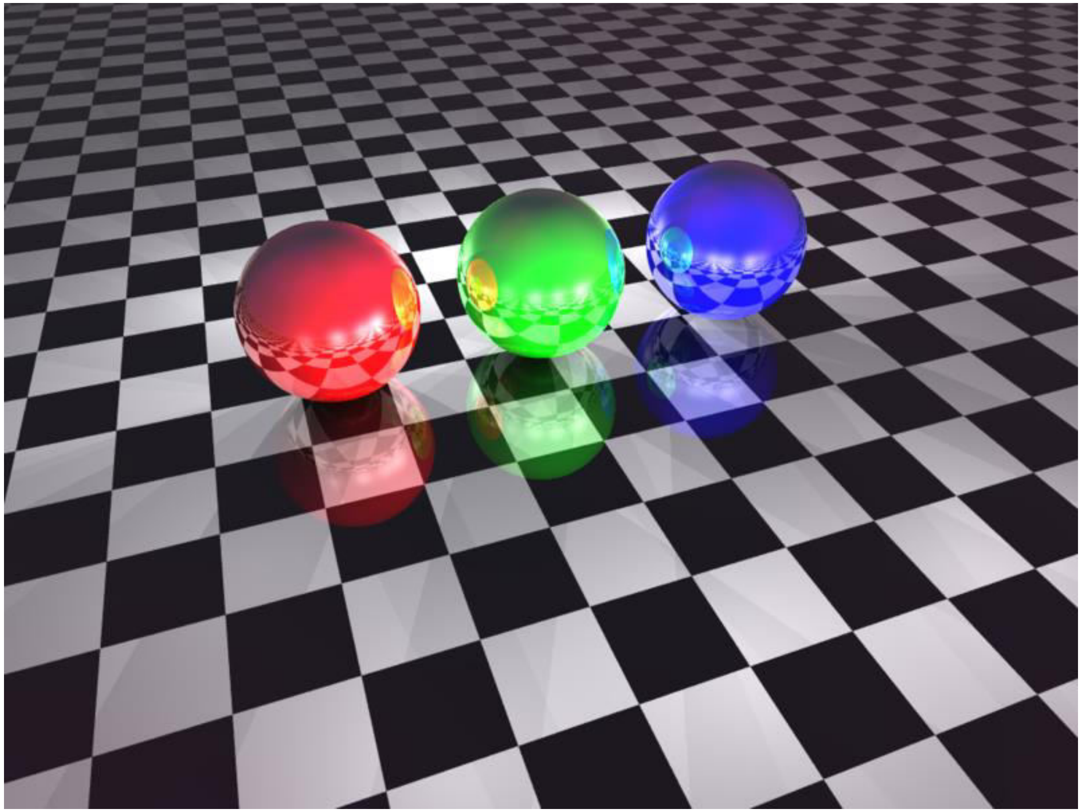


Ray Traced Shadows

- Trace a ray
  - Start = point on surface
  - End = light source
  - t=0 at Suface, t=1 at Light
  - "Bias" to avoid *surface acne*
- Test
  - Bias ≤ t ≤ 1 = shadow
  - t < Bias or t > 1 = use this light



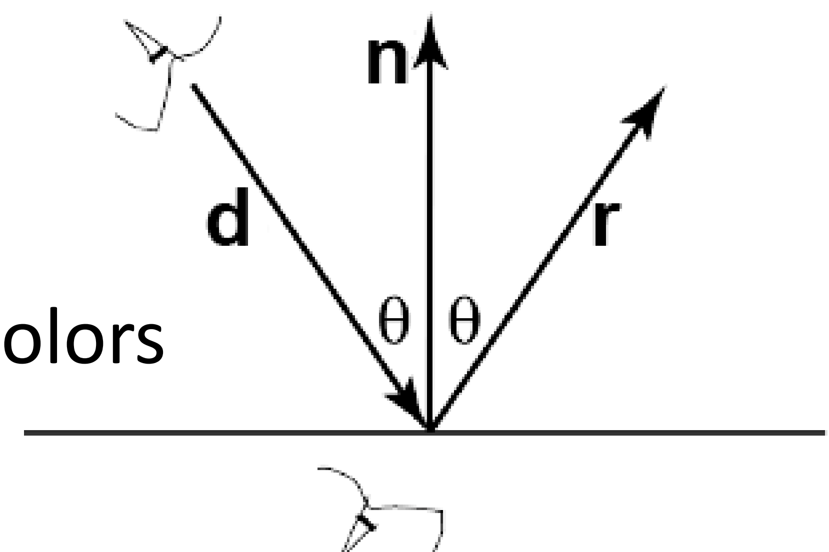
Mirror Reflection



The Dark Side of the Trees - Gilles Tran,
Spheres - Martin K. B.

Ray Tracing Reflection

- Viewer looking in direction ***d*** *sees whatever the viewer "below" the surface sees looking in direction* ***r***
- In the real world
  - Energy loss on the bounce
  - Loss different for different colors
- New ray
  - Start on surface, in reflection direction



Calculating Reflection Vector

- Angle of of incidence = angle of reflection

  $\hat{v} = -\hat{d}$

- Decompose $\hat{v}$

  $\vec{v}_n = (\hat{n} \cdot \hat{v})\hat{n}$

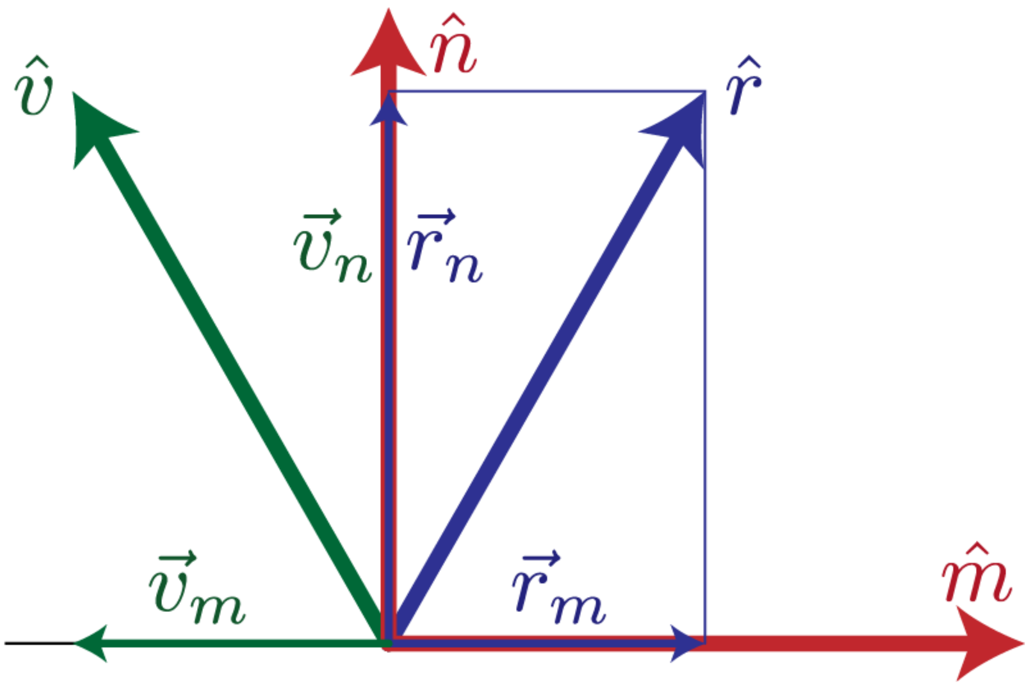
  $\vec{v}_m = \hat{v} - (\hat{n} \cdot \hat{v})\hat{n}$

- Recompose $\hat{r}$

  $\vec{r}_n = \vec{v}_n; \ \vec{r}_m = -\vec{v}_m$

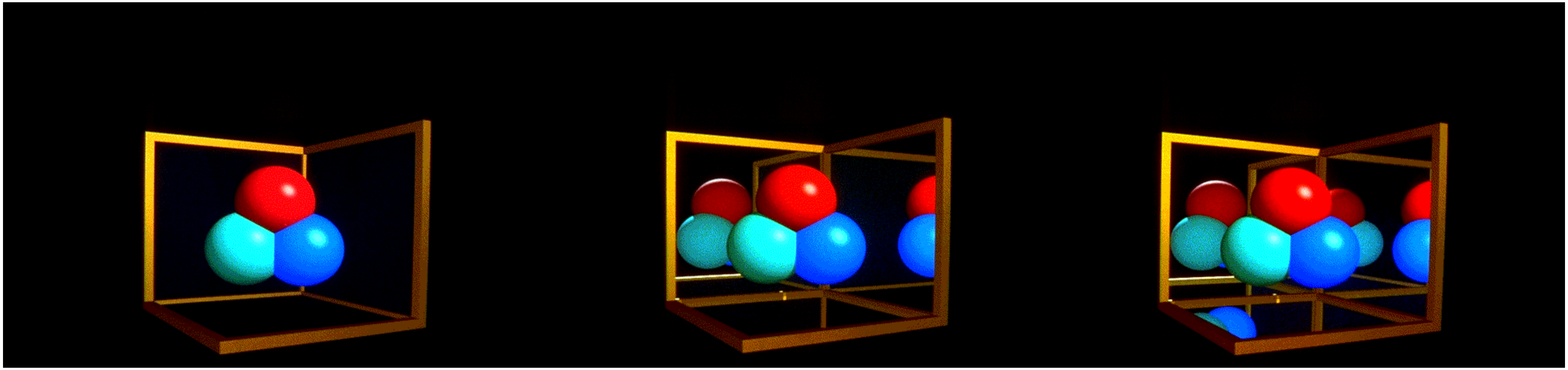
  $\hat{r} = \vec{r}_n + \vec{r}_m$

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



Ray Traced Reflection

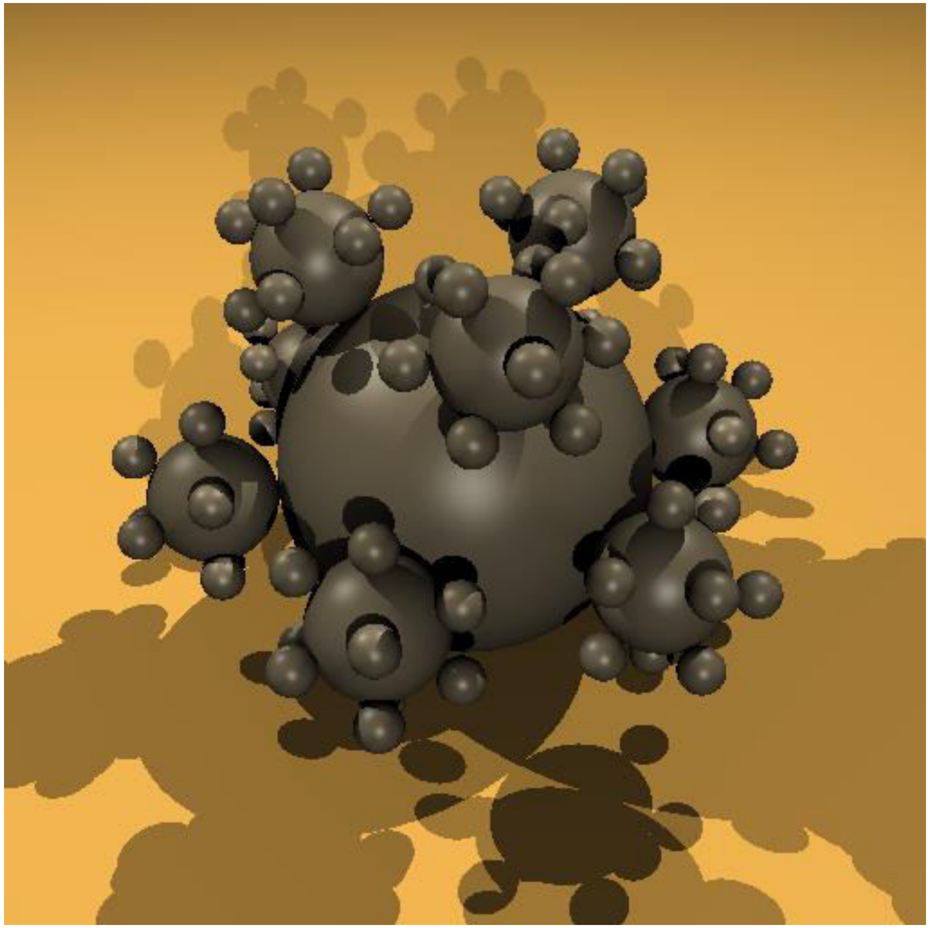
- Avoid looping forever
  - Stop after *n* bounces
  - Stop when contribution to pixel gets too small

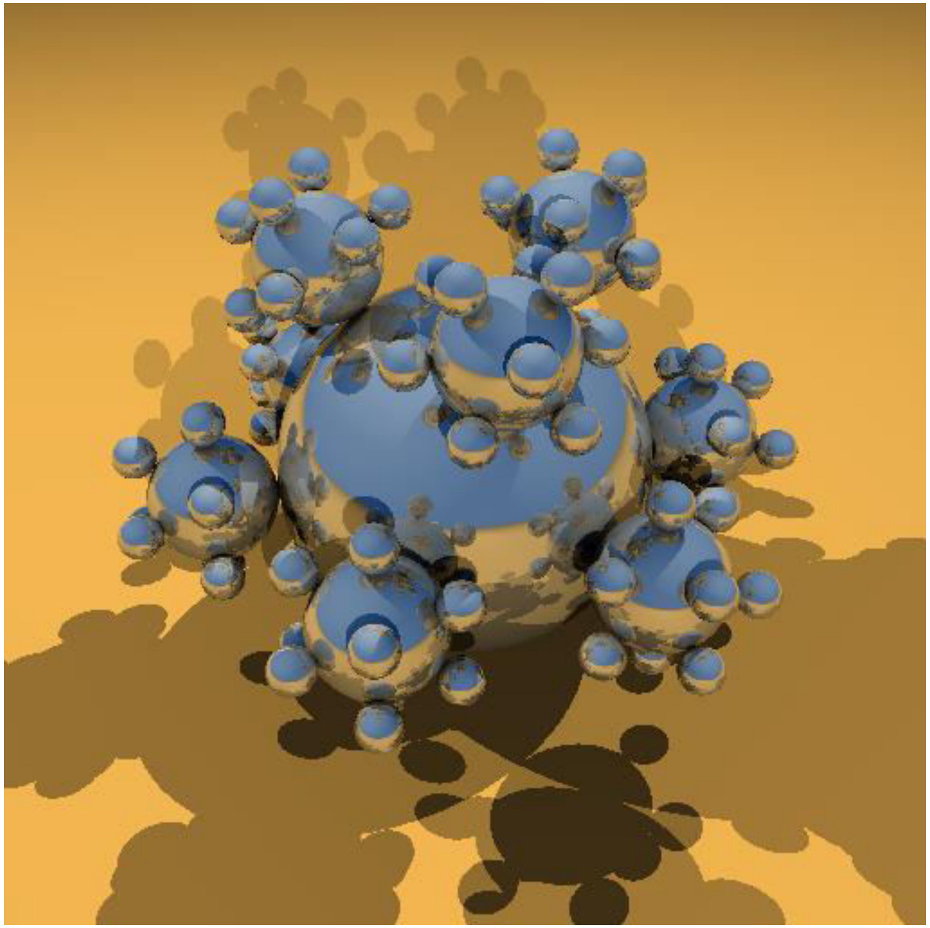


Specular Reflection

- Shiny reflection from rough surface
- Centered around mirror reflection direction
  - But more spread more, depending on roughness
- Easiest for individual light sources

Specular vs. Mirror Reflection

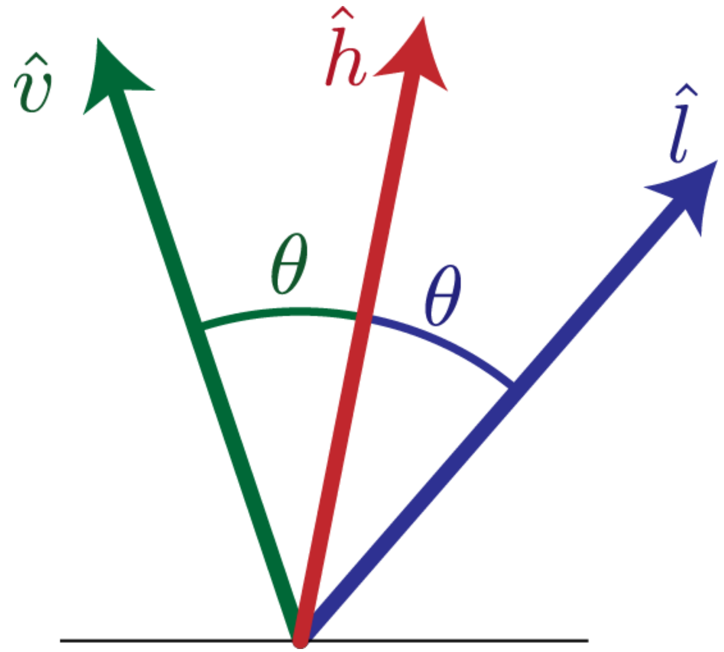


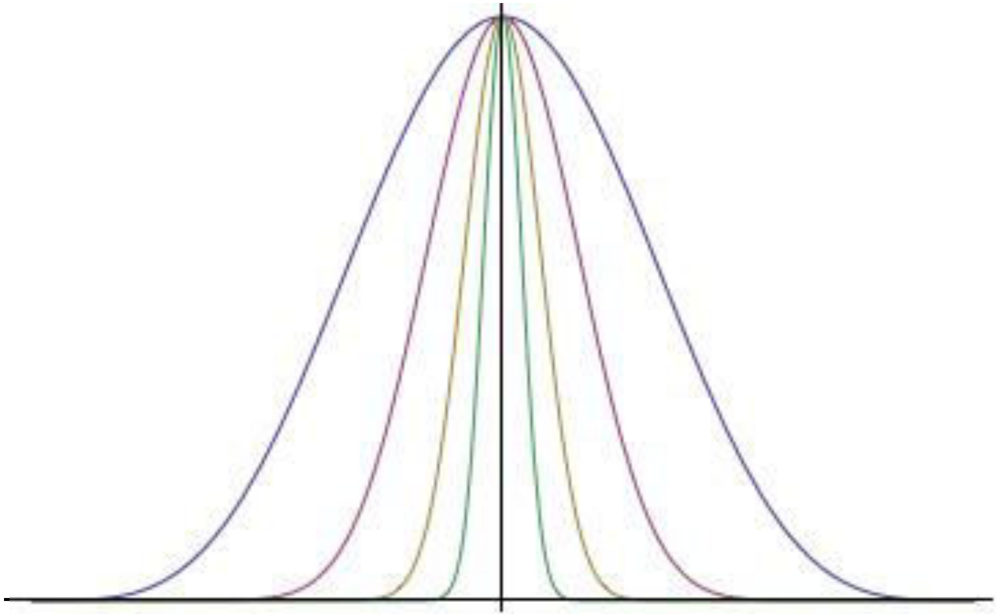


H vector

- Strongest for normal that reflects $\hat{l}$ to $\hat{v}$
- $\hat{h} = \dfrac{\hat{l} + \hat{v}}{|\hat{l} + \hat{v}|}$
- $\hat{n} \cdot \hat{h}$
  - One at center of highlight
  - Zero at 90°
- Control highlight width

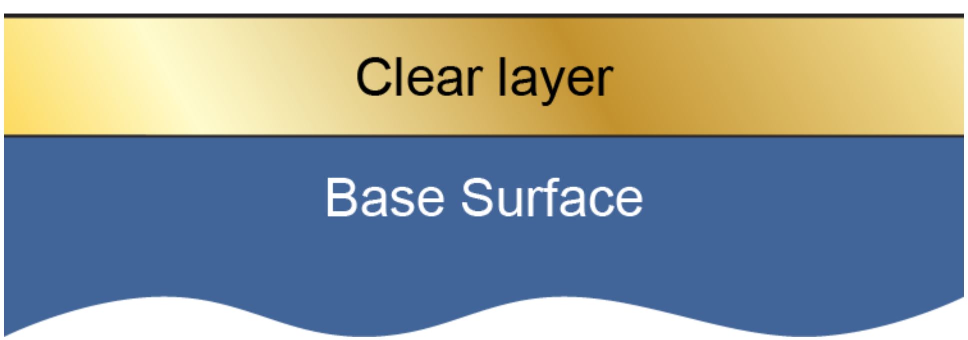
$$(\hat{n} \cdot \hat{h})^e$$

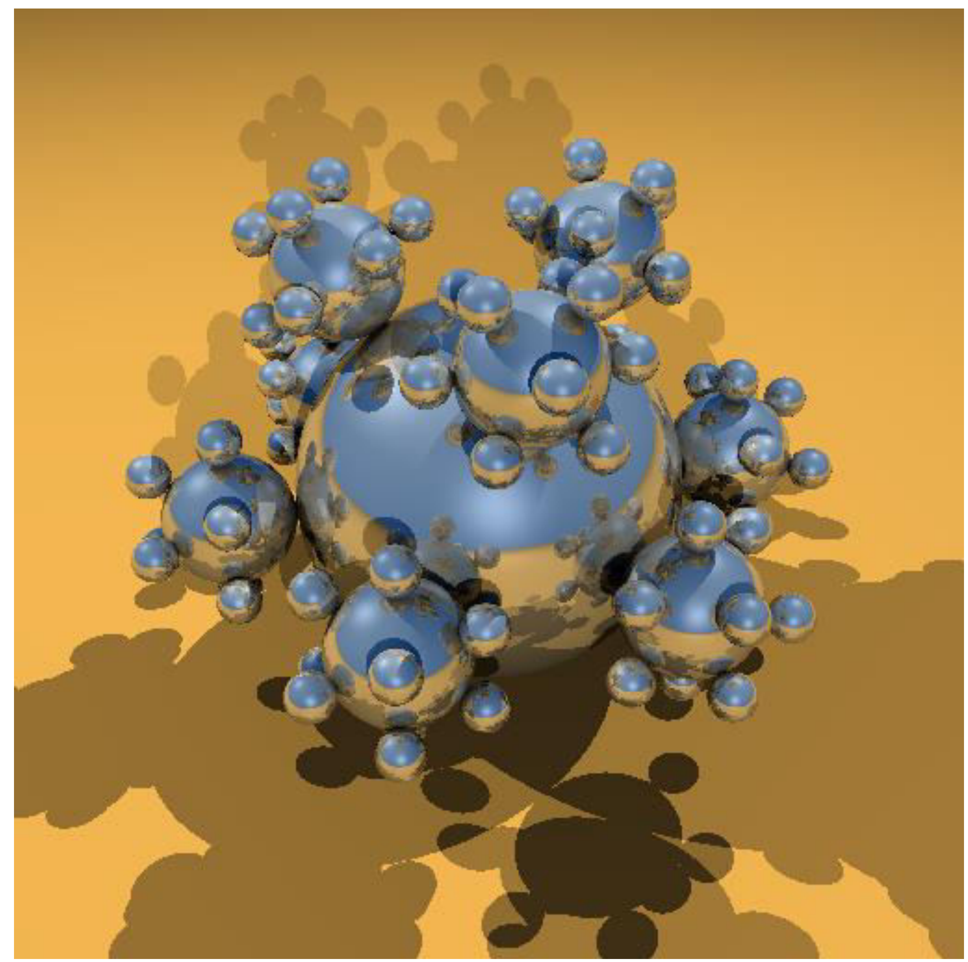




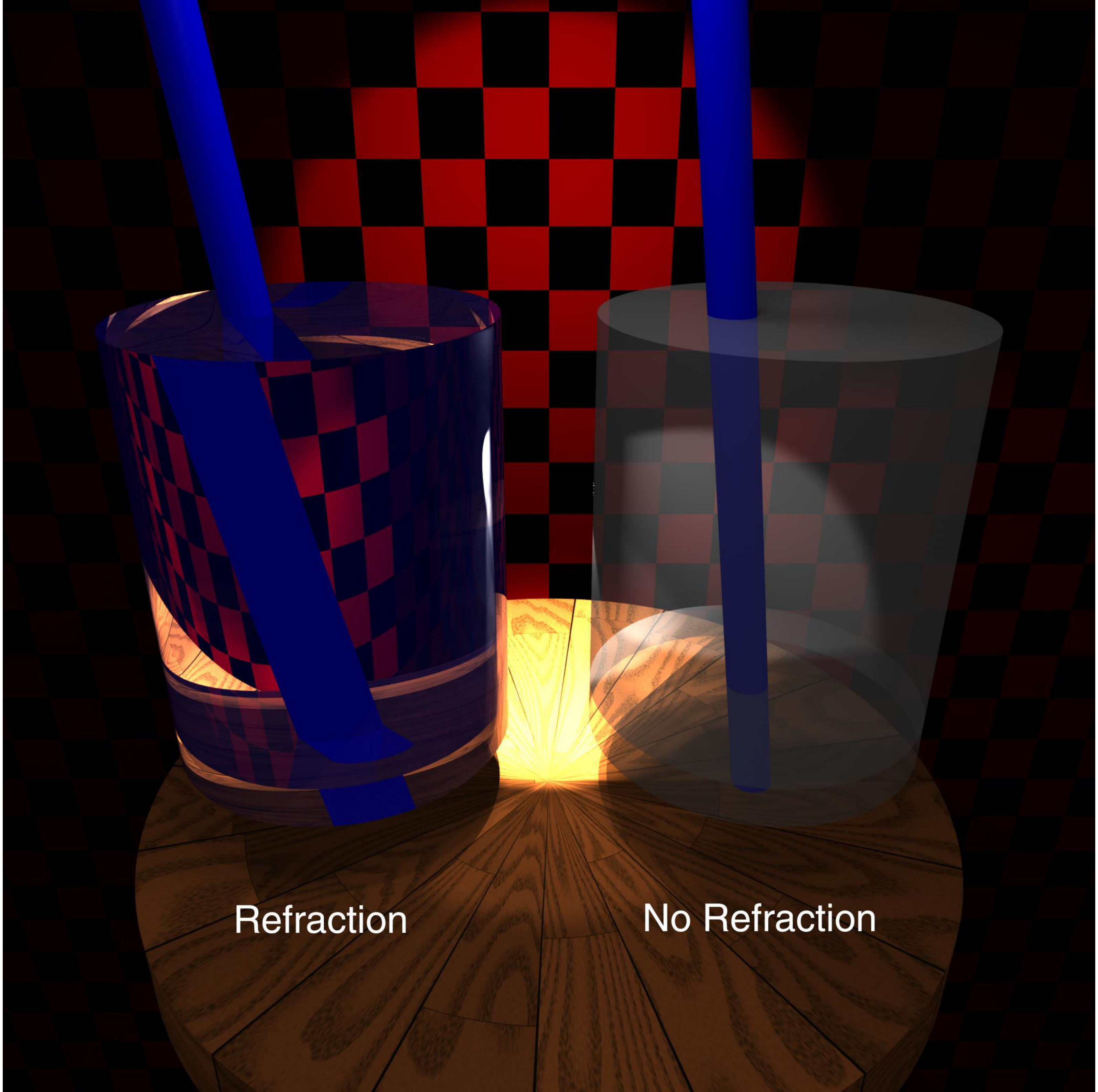
Combined Specular & Mirror

- Many surfaces have both

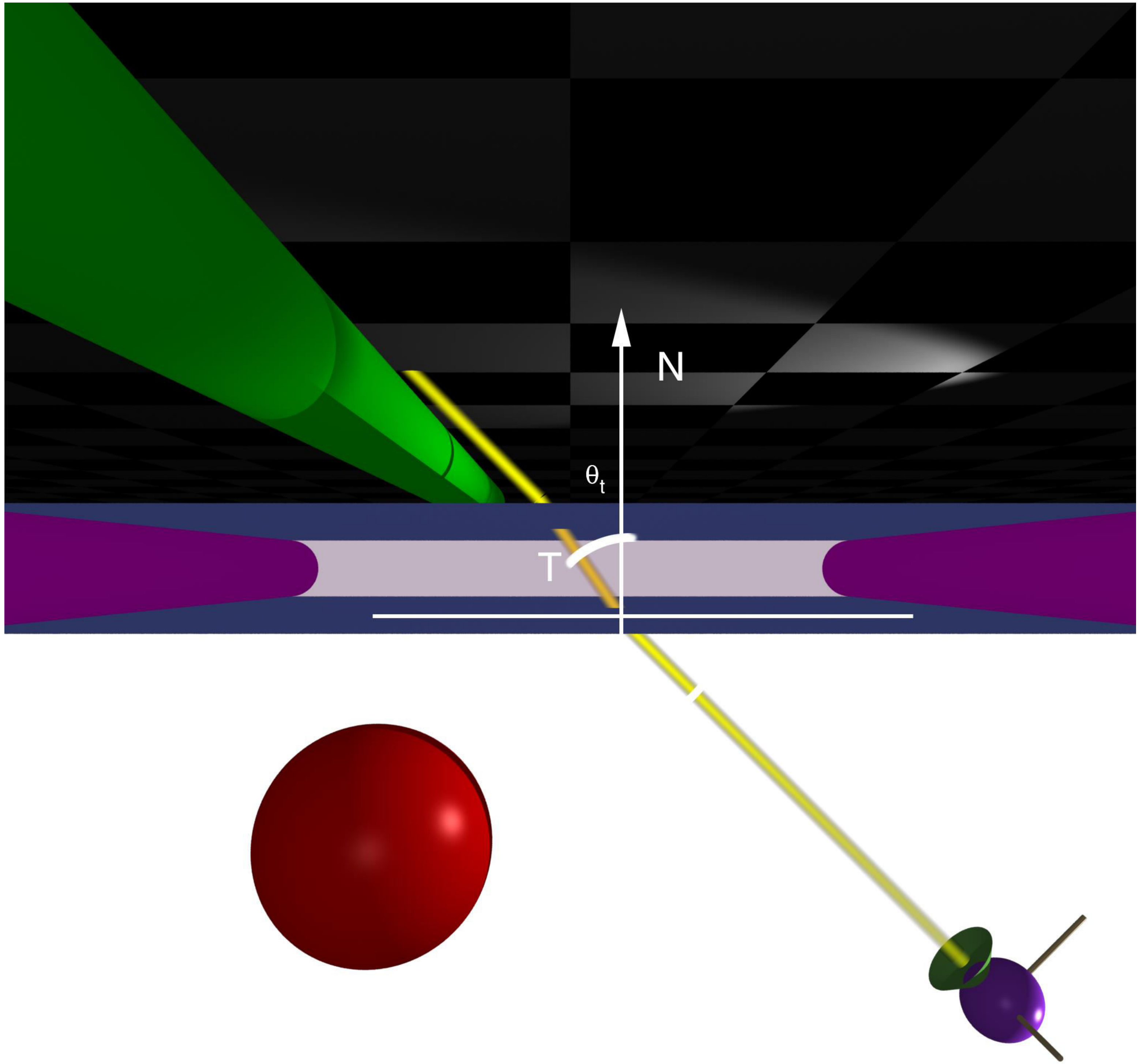




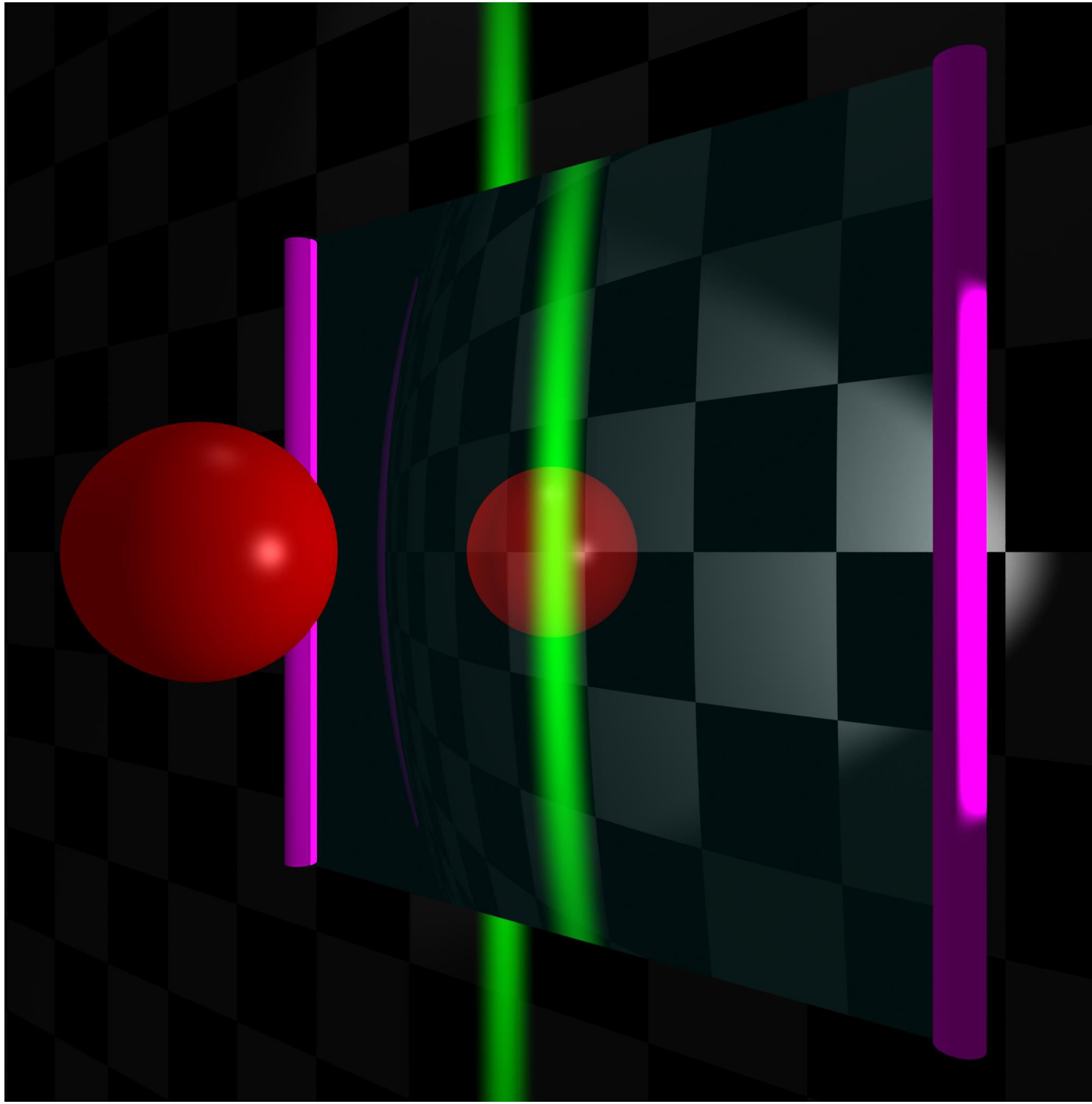
Refraction


Refraction
No Refraction

Top



Front



Calculating Refraction Vector

- Snell's Law

  $n_v \sin\theta_v = n_t \sin\theta_t$

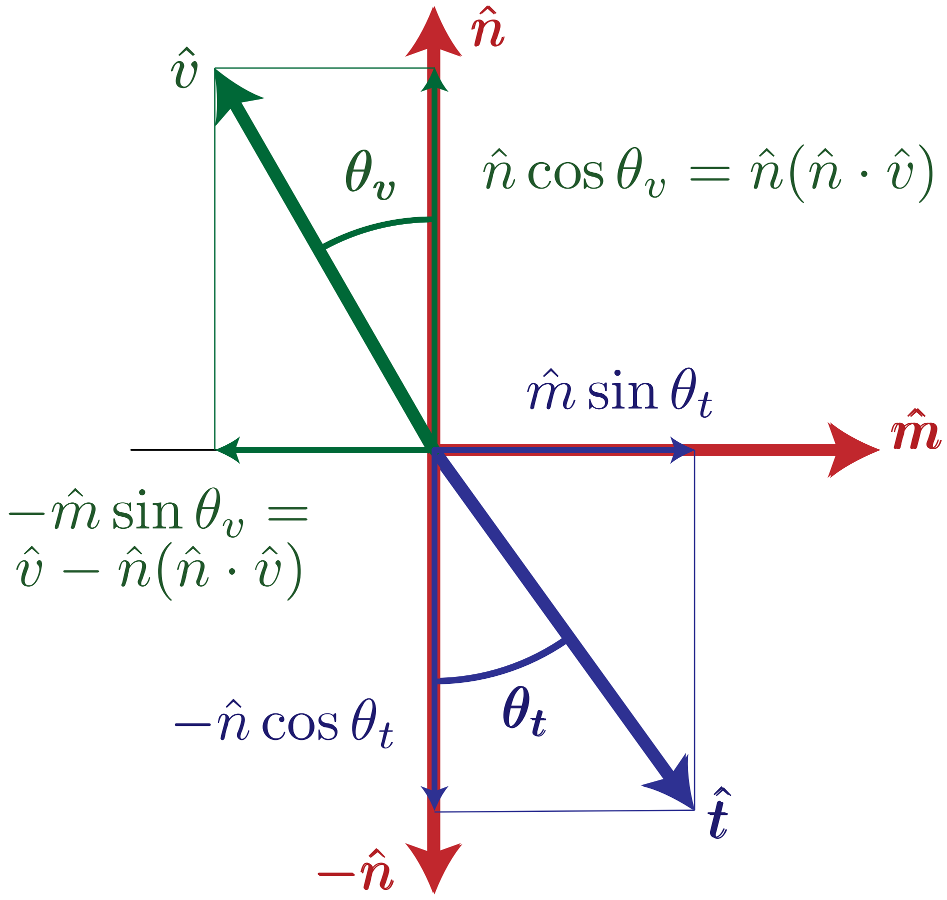
- In terms of θ_t

  $\hat{t} = \hat{m}\sin\theta_t - \hat{n}\cos\theta_t$

- $\hat{m}$ term

  $\hat{m} = (\hat{n}(\hat{n}\cdot\hat{v}) - \hat{v}) / \sin\theta_v$

  $$\begin{aligned}\hat{m}\sin\theta_t &\\ &= (\hat{n}(\hat{n}\cdot\hat{v}) - \hat{v})\sin\theta_t / \sin \\ &= (\hat{n}(\hat{n}\cdot\hat{v}) - \hat{v})\, n_v / n_t\end{aligned}$$



Calculating Refraction Vector

- Snell's Law

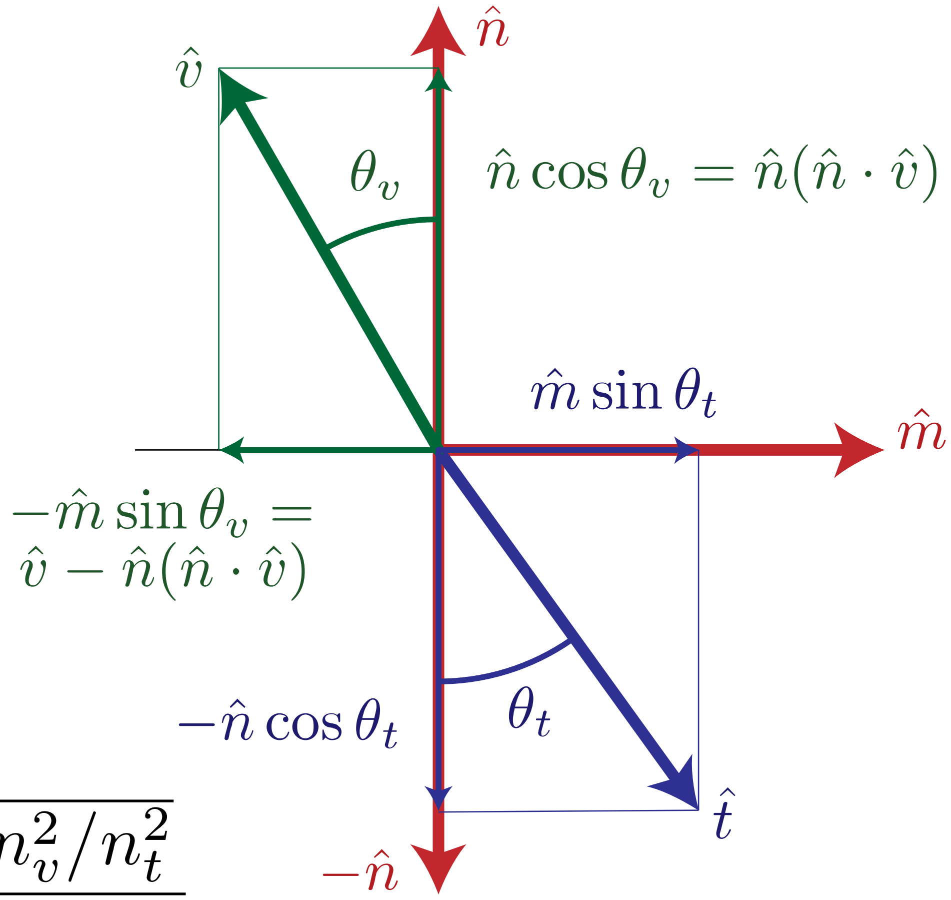
$$n_v \sin\theta_v = n_t \sin\theta_t$$

- In terms of θ_t

$$\hat{t} = \hat{m}\sin\theta_t - \hat{n}\cos\theta_t$$

- $\hat{n}$ term

$$\begin{aligned}
&-\hat{n}\cos\theta_t \\
&\quad = -\hat{n}\sqrt{1-\sin^2\theta_t} \\
&\quad = -\hat{n}\sqrt{1-\sin^2\theta_v\, n_v^2/n_t^2} \\
&\quad = -\hat{n}\sqrt{1-(1-\cos^2\theta_v)\, n_v^2/n_t^2} \\
&\quad = -\hat{n}\sqrt{1-(1-(\hat{n}\cdot\hat{v})^2)\, n_v^2/n_t^2}
\end{aligned}$$



Calculating Refraction Vector

- Snell's Law

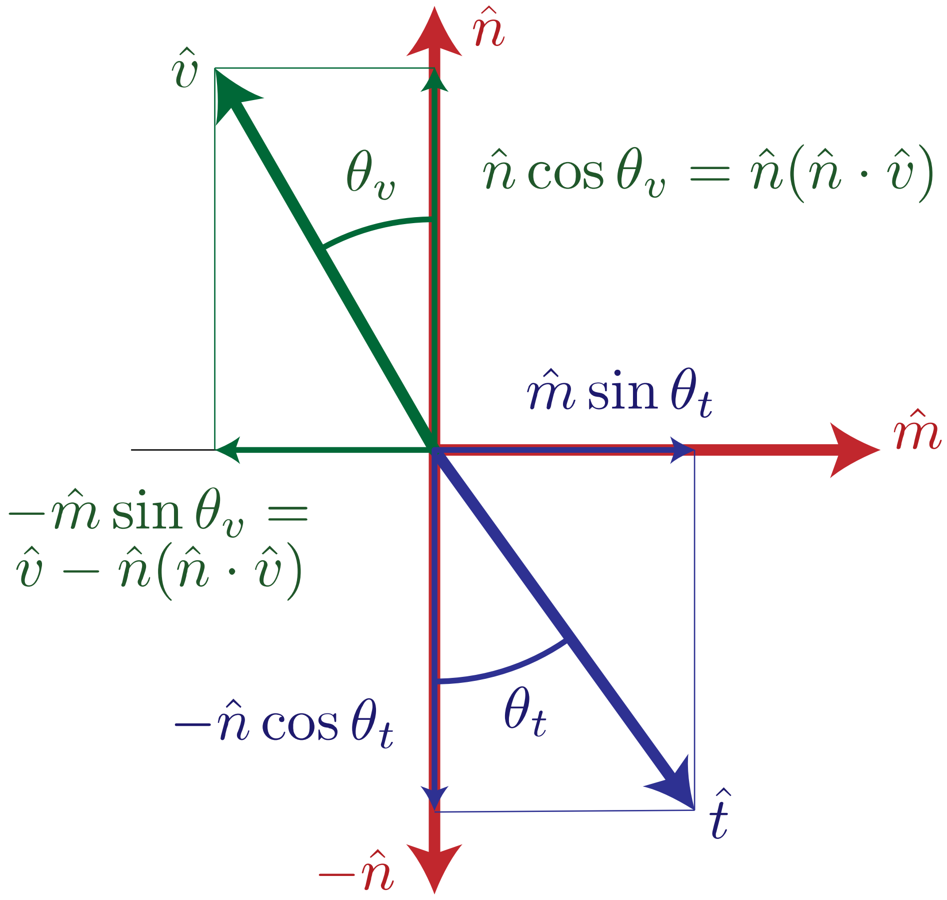
$$n_v \sin\theta_v = n_t \sin\theta_t$$

- In terms of θ_t

$$\hat{t} = \hat{m}\sin\theta_t - \hat{n}\cos\theta_t$$

- In terms of $\hat{n}$ and $\hat{v}$

$$\hat{t} = (\hat{n}(\hat{n}\cdot\hat{v}) - \hat{v})n_v/n_t \\ -\hat{n}\sqrt{1 - \left(1 - (\hat{n}\cdot\hat{v})^2\right) n_v^2/n_t^2}$$



Alpha Blending

- How much makes it through
- α = opacity
  - How much of foreground color 0-1
- 1-α = transparency
  - How much of background color
- Foreground*α + Background*(1-α)

Refraction and Alpha

- Refraction = what direction
- α = how much
  - Often approximate as a constant
  - Better: Use Fresnel

$$F = \frac{1}{2}\left(\frac{n_v\ \hat{n}\cdot\hat{r} + n_t\ \hat{n}\cdot\hat{t}}{n_v\ \hat{n}\cdot\hat{r} - n_t\ \hat{n}\cdot\hat{t}}\right)^2 + \frac{1}{2}\left(\frac{n_v\ \hat{n}\cdot\hat{t} + n_t\ \hat{n}\cdot\hat{r}}{n_v\ \hat{n}\cdot\hat{t} - n_t\ \hat{n}\cdot\hat{r}}\right)^2$$

  - Schlick approximation

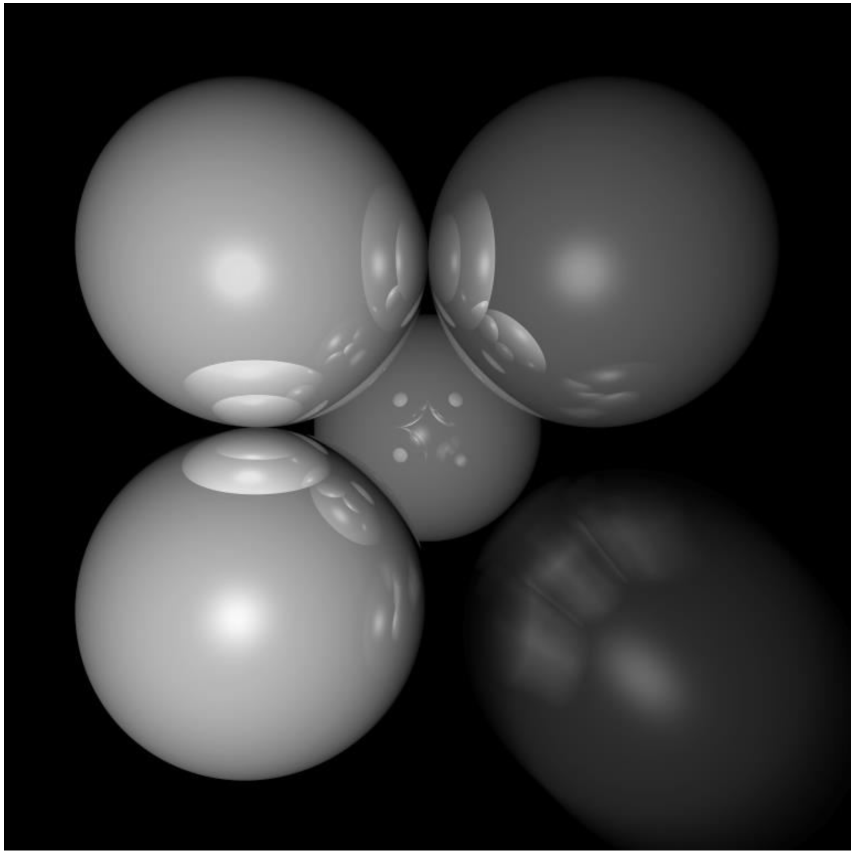
$$F_0 = (n_v - n_t)^2/(n_v + n_t)^2$$
$$F \approx F_0 + (1 - F_0)(1 - \hat{n}\cdot\hat{v})^5$$

Full Ray-Tracing

- For each pixel
  - Compute ray direction
  - Find closest surface
  - For each light
    - Shoot shadow ray
    - If not shadowed, add direct illumination
  - Shoot ray in reflection direction
  - Shoot ray in refraction direction

Motion Blur

- Things move while the shutter is open



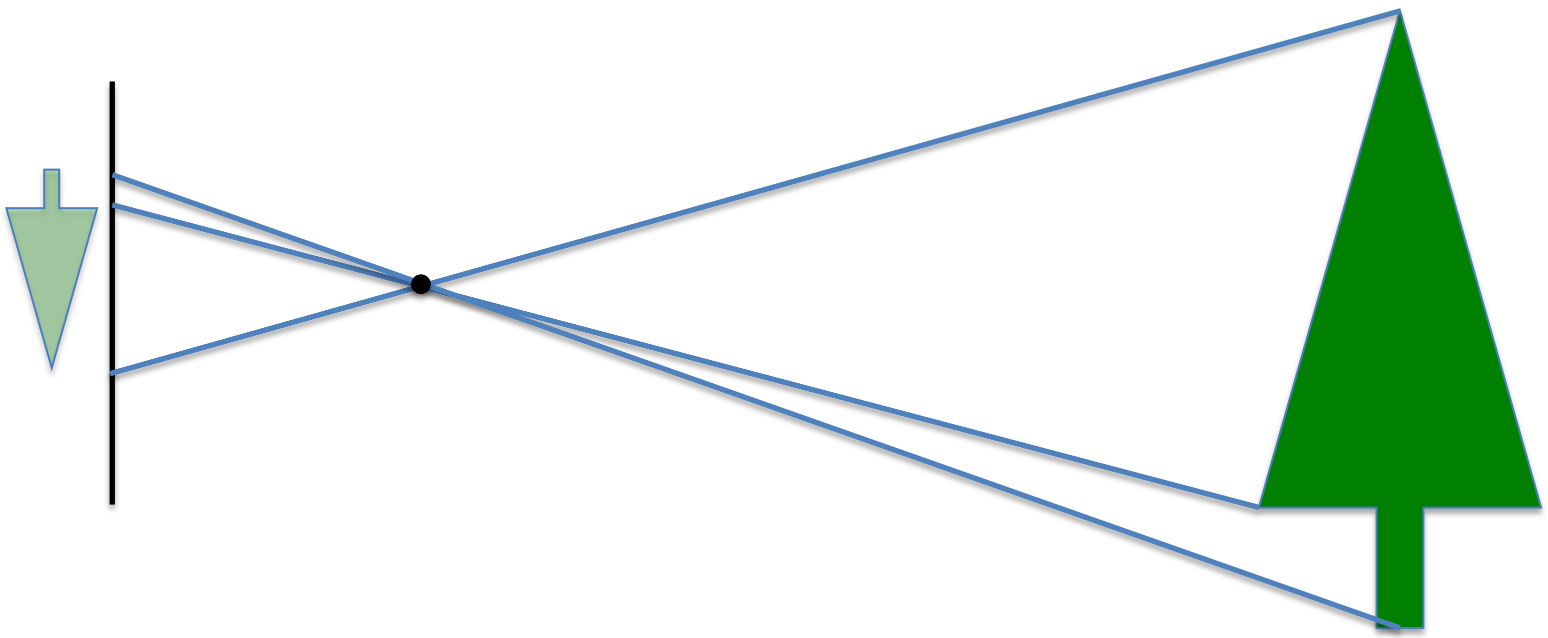
Ray Traced Motion Blur

- Include information on object motion
- Spread multiple rays per pixel across time

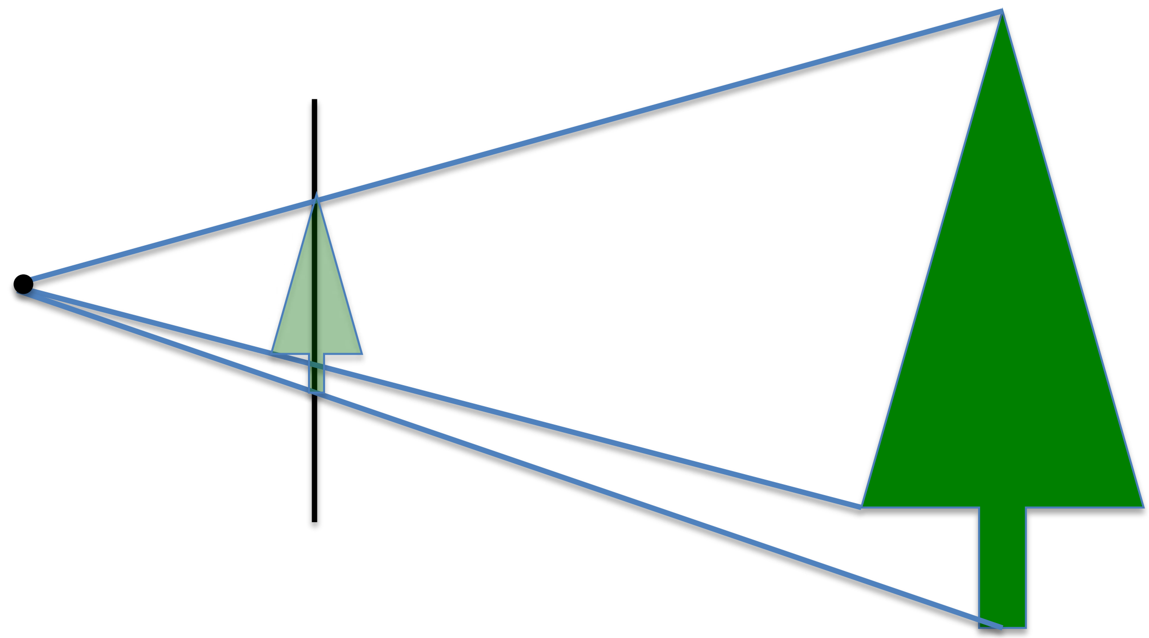
Depth of Field

Soler et al., Fourier Depth of Field, ACM TOG v28n2, April 2009

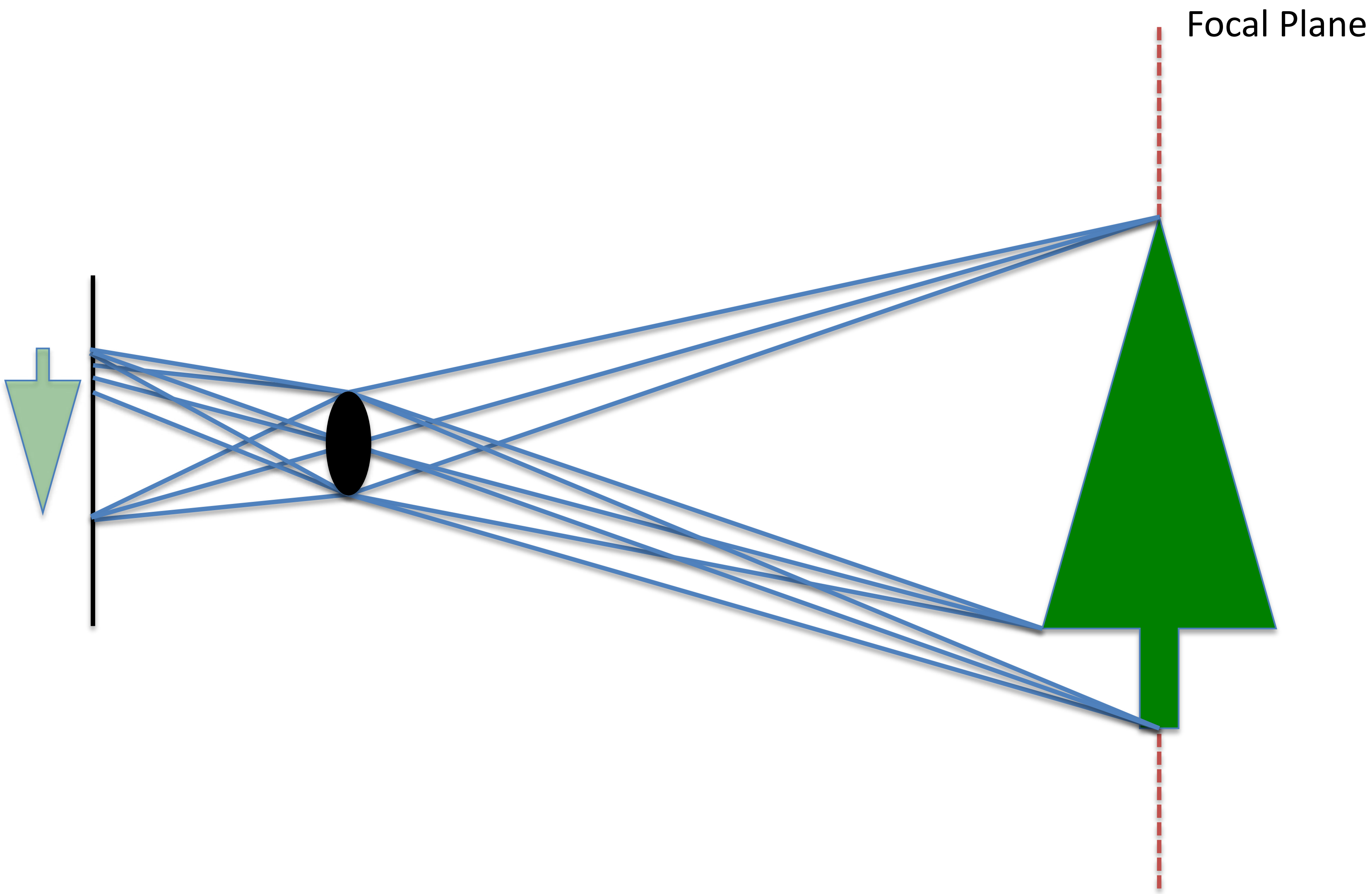
Pinhole Lens



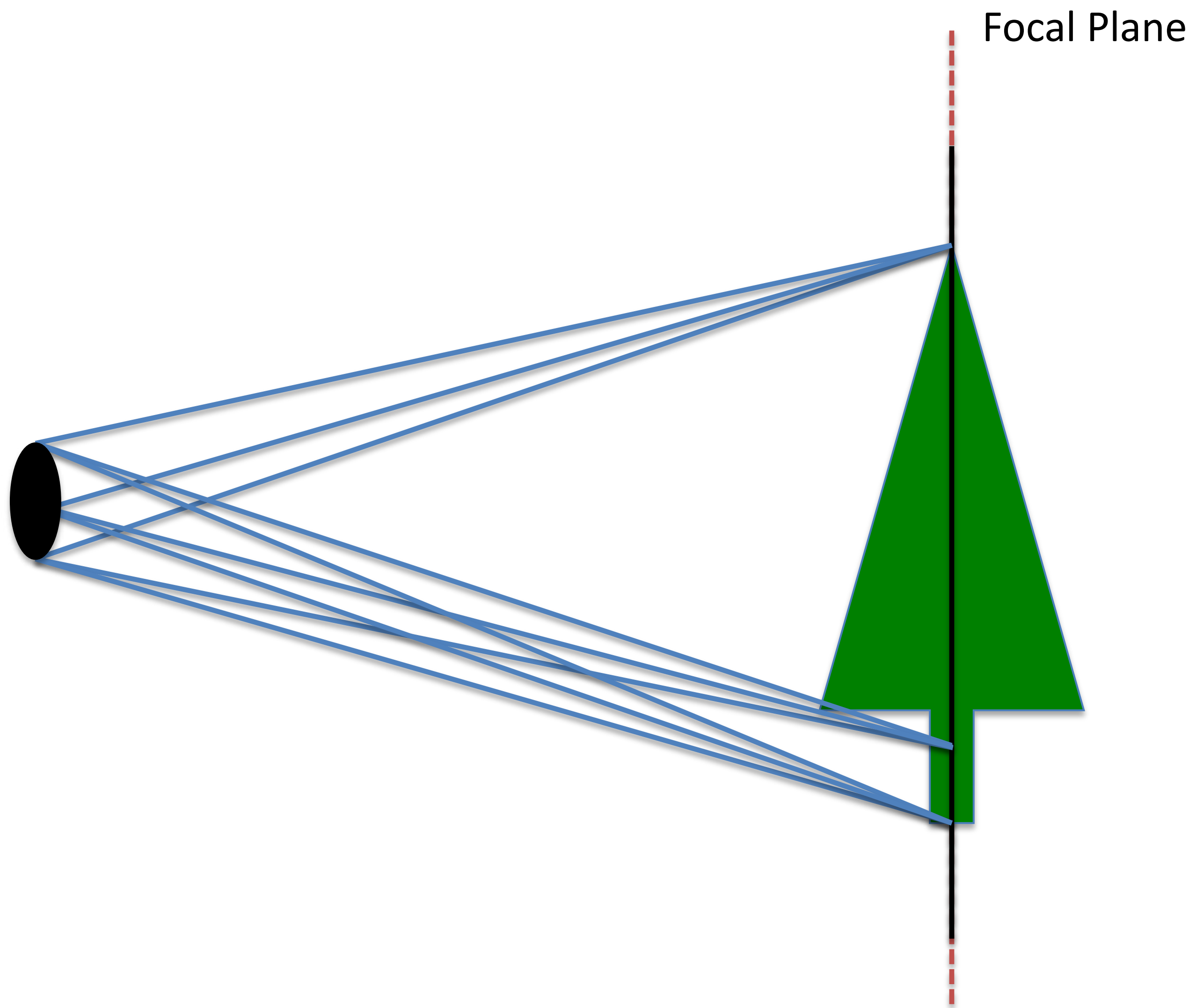
Lens Model



Real Lens



Lens Model



Ray Traced DOF

- Move image plane out to focal plane
- Jitter start position within lens aperture
  - Smaller aperture = closer to pinhole
  - Larger aperture = more DOF blur