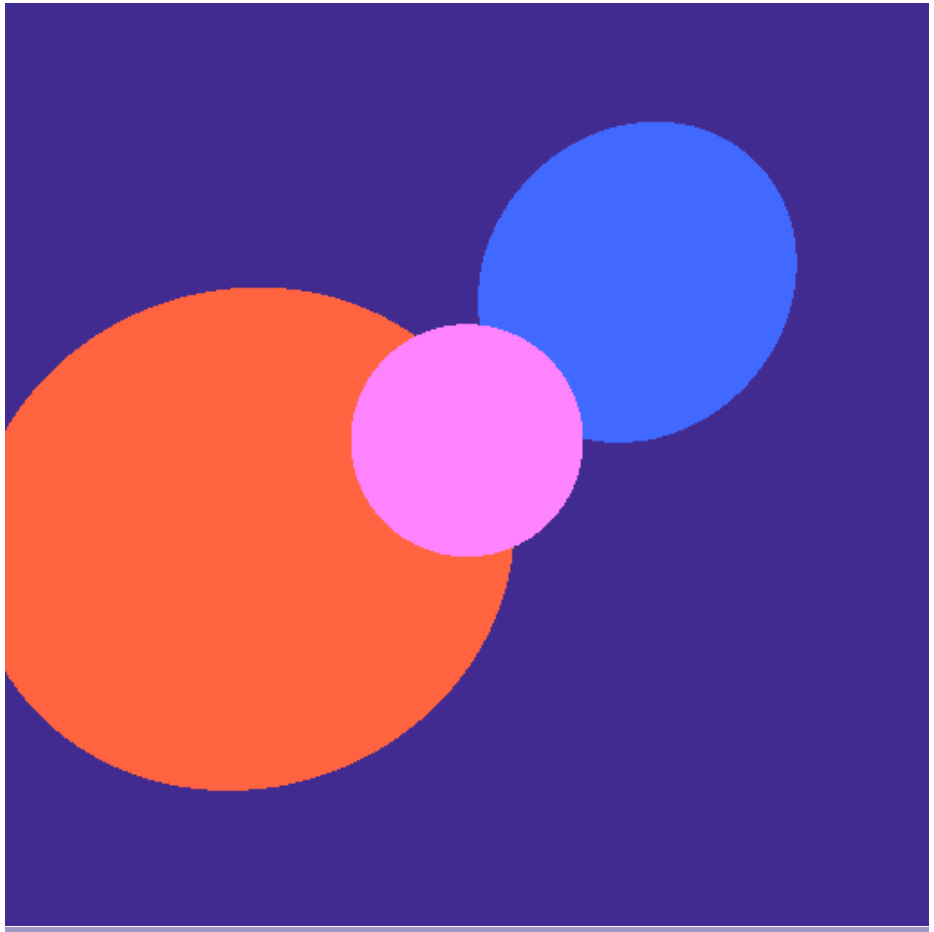
To Shading and Beyond

CS 6965 Fall 2011

Program 1

- Questions on program 1?
- Running simhwrt



Program 1 fix

simhwrt

- Can set scene and image size
  - --no-scene
  - --width 256
  - --height 256
- Can set number of TMs and threads per tm
  - --num-thread-procs 32
  - --num-cores 10
- Gathers statistics for EVERYTHING
- More on simhwrt later

Ray tracing architecture

- The major components in a ray tracer are:
- Camera (Pixels to Rays)
- Objects (Rays to intersection info)
- Materials (Intersection info and light to color)
- Lights
- Background (Rays to Color)

- All together: a Scene

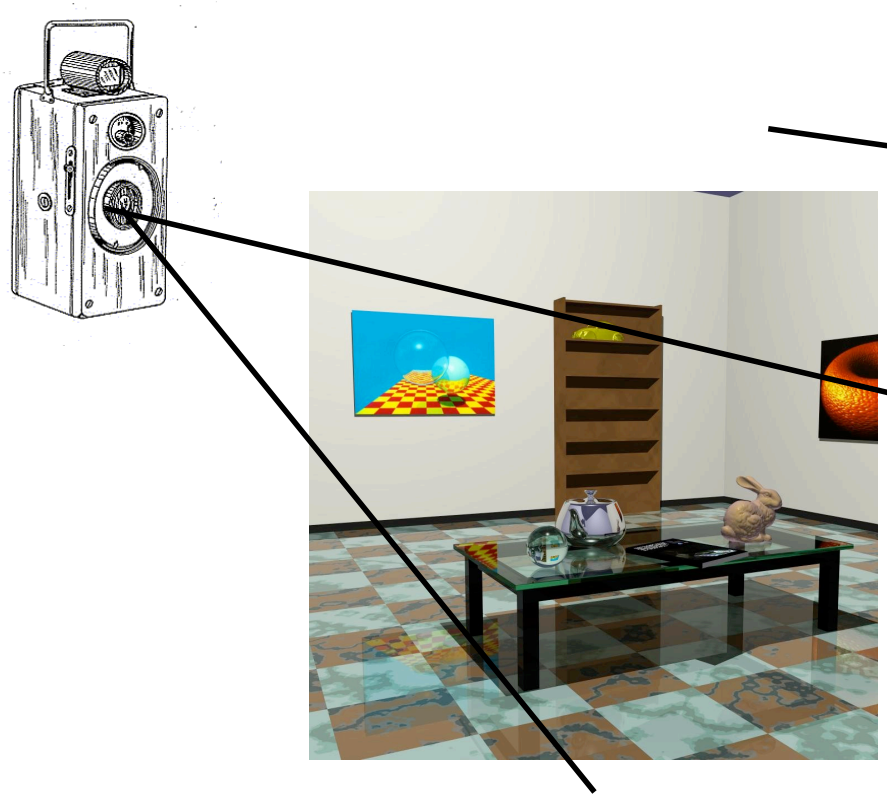
Ray tracing algorithm

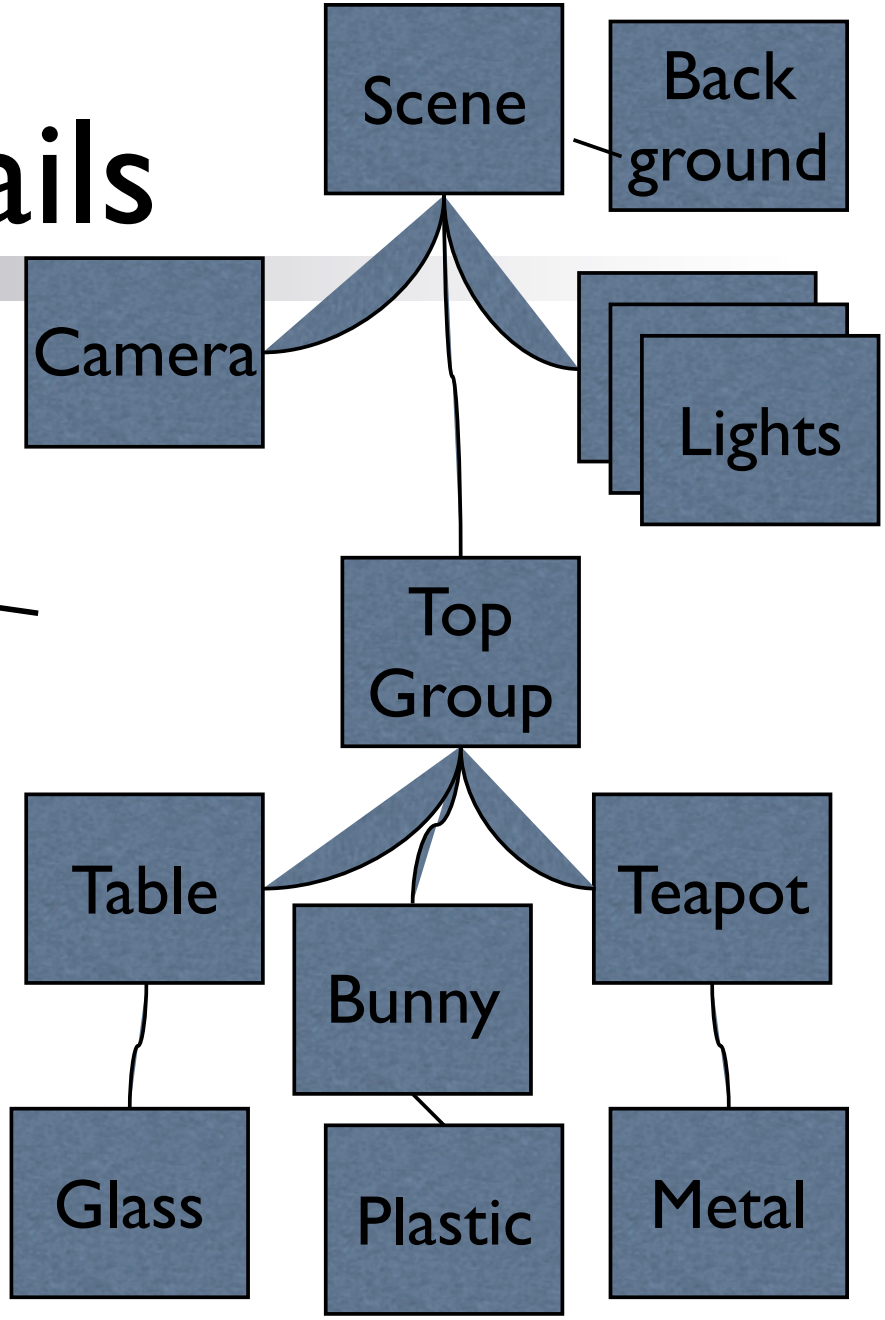
- Create scene (objects, materials, lights, camera, background)
- Preprocess scene
- foreach frame
- foreach pixel
- foreach sample
- generate ray
- intersect ray with objects
- find normal of closest object

Mutually recursive

Details

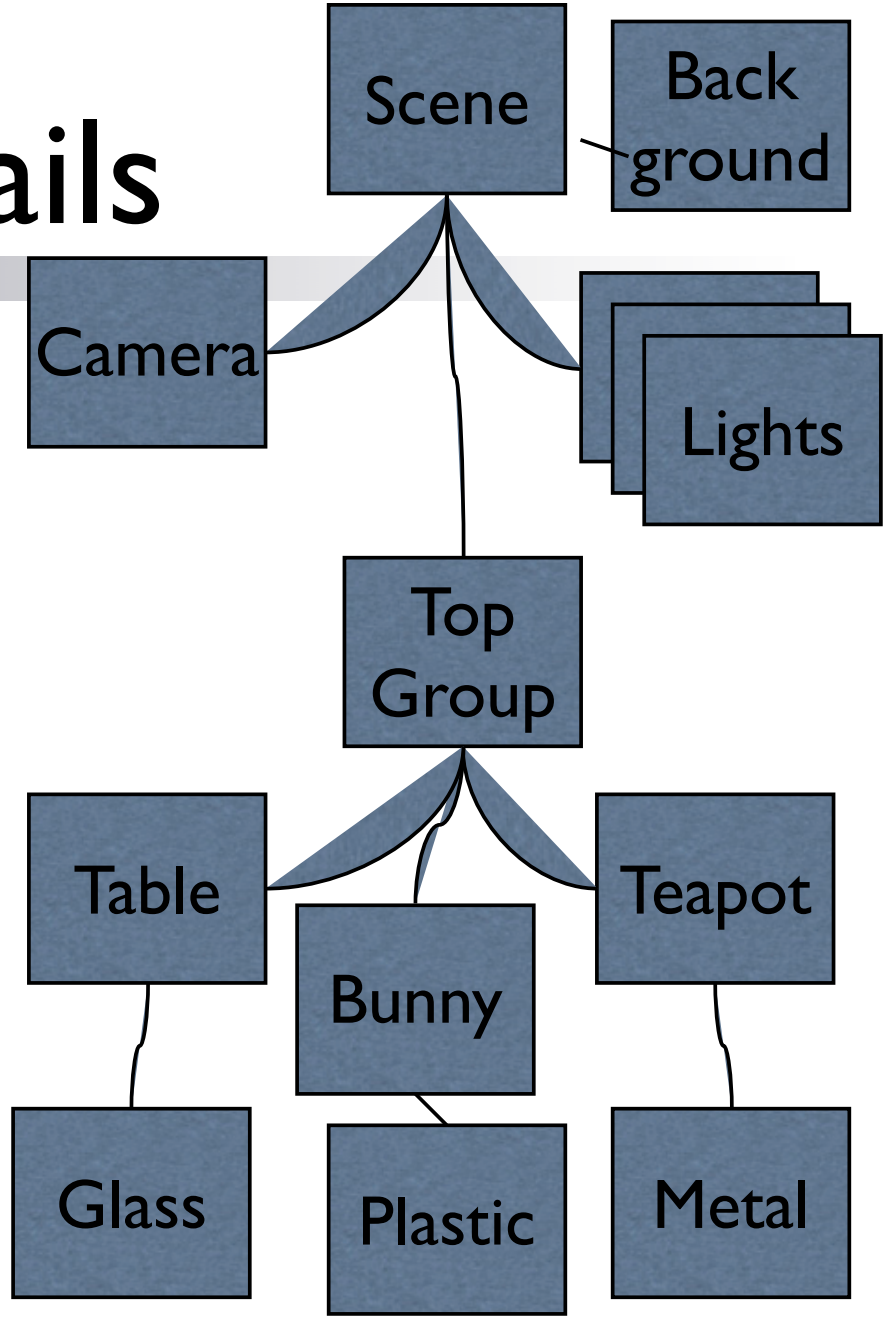
- Create scene





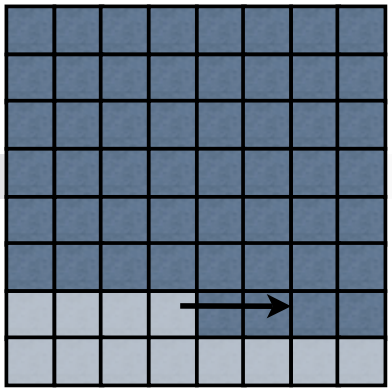
Details

- Create scene
- Preprocess scene

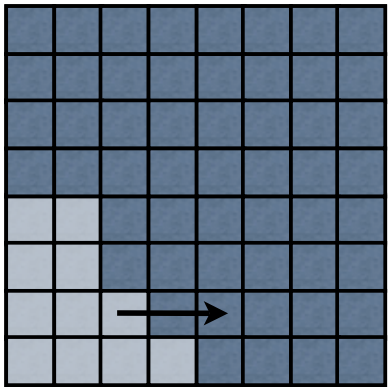


Details

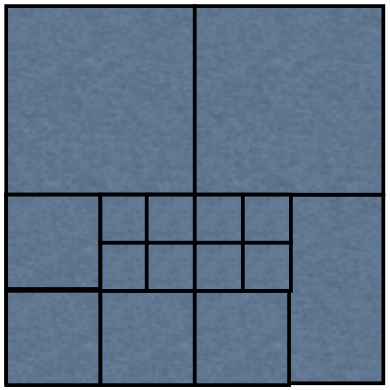
- Create scene
- Preprocess scene
- foreach pixel



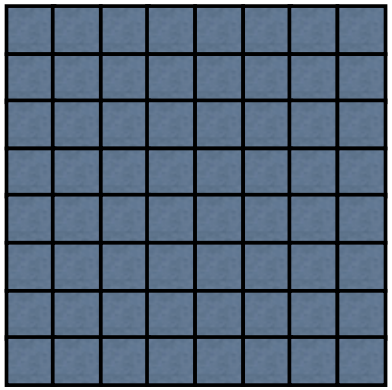
Row-major order



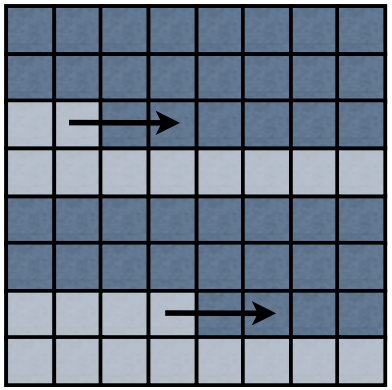
Tiled



Progressive



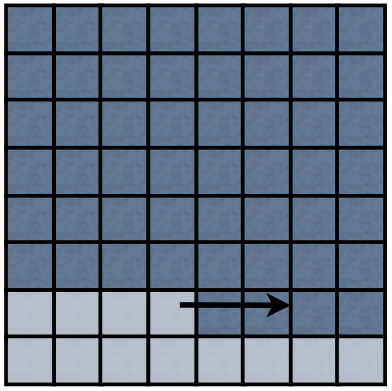
Frameless rendering

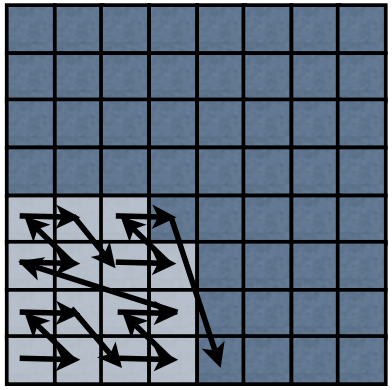


Parallel

How rt.s Does It

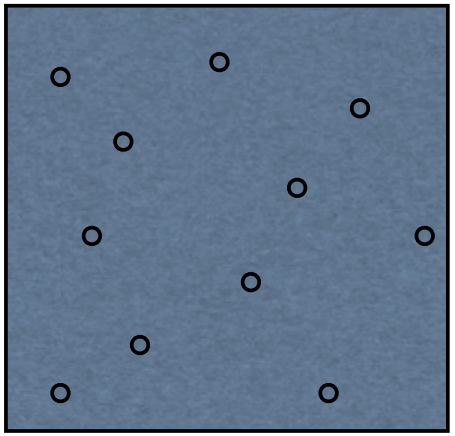
- We did tiled thread assignments at first
- Currently two candidates:
  - Row-major order
  - Space filling curve





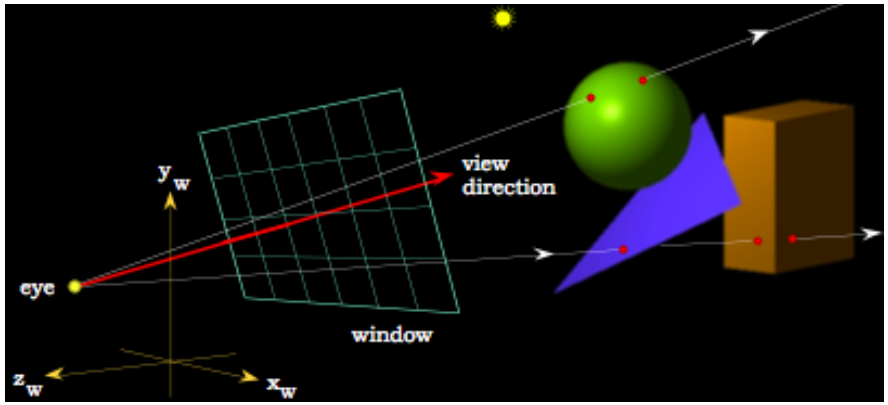
Details

- Create scene
- Preprocess scene
- foreach pixel
- foreach sample



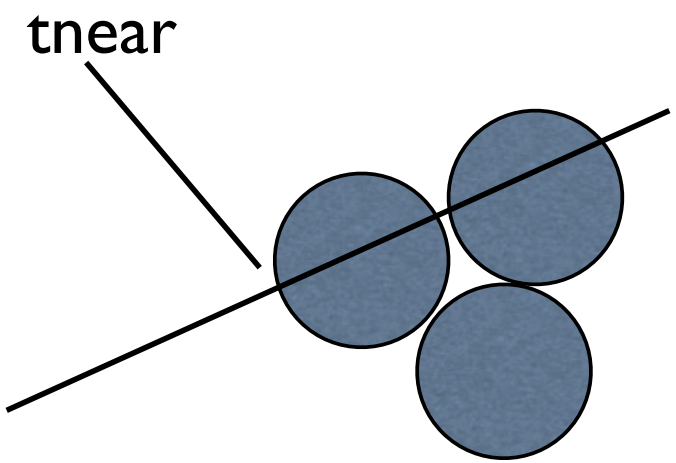
Details

- Create scene
- Preprocess scene
- foreach pixel
- foreach sample
- generate ray



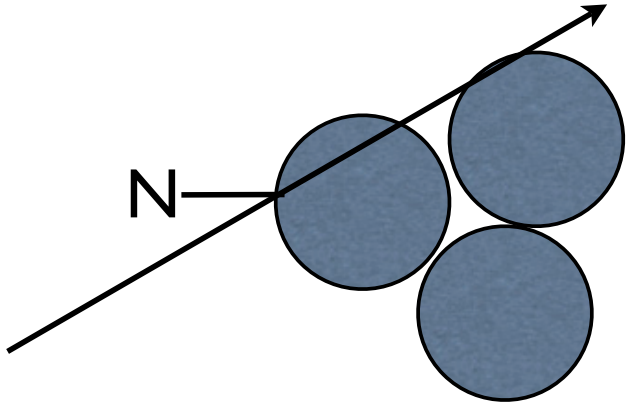
Details

- Create scene
- Preprocess scene
- foreach pixel
- foreach sample
- generate ray
- intersect ray with objects



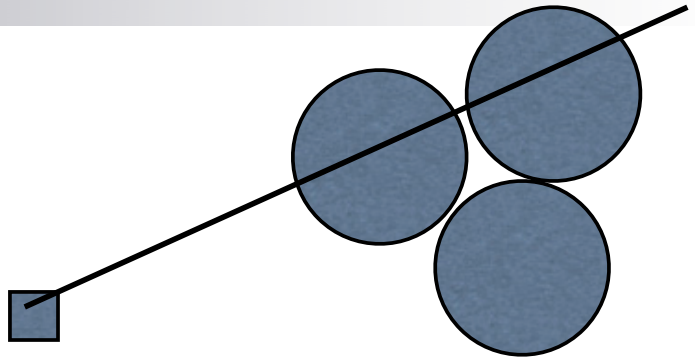
Details

- Create scene
- Preprocess scene
- foreach pixel
- foreach sample
- generate ray
- intersect ray with objects
- find normal of closest object



Details

- Create scene
- Preprocess scene
- foreach pixel
- foreach sample
- generate ray
- intersect ray with objects
- find normal of closest object
- shade intersection point



Background

- What do you do when the ray misses all objects?
  - Constant color
    - Assign color to ray
  - Gradient
    - Map -1 to 1:
    - Map to colors:
  - Star field:
    - Sum up colors:
  - Environment map

$$s = \frac{\left\| \overrightarrow{V_{ray}} \cdot \overrightarrow{V_{up}} \right\|}{\left\| \overrightarrow{V_{ray}} \right\| \left\| \overrightarrow{V_{up}} \right\|}$$

$$C_{down} + \left(\frac{s+1}{2}\right) C_{up}$$

$$\sum_{stars} \left(\overrightarrow{V_{ray}} \cdot \overrightarrow{V_{star}} \right)^p C_{star}, p \simeq 10000$$

Data structures: Light

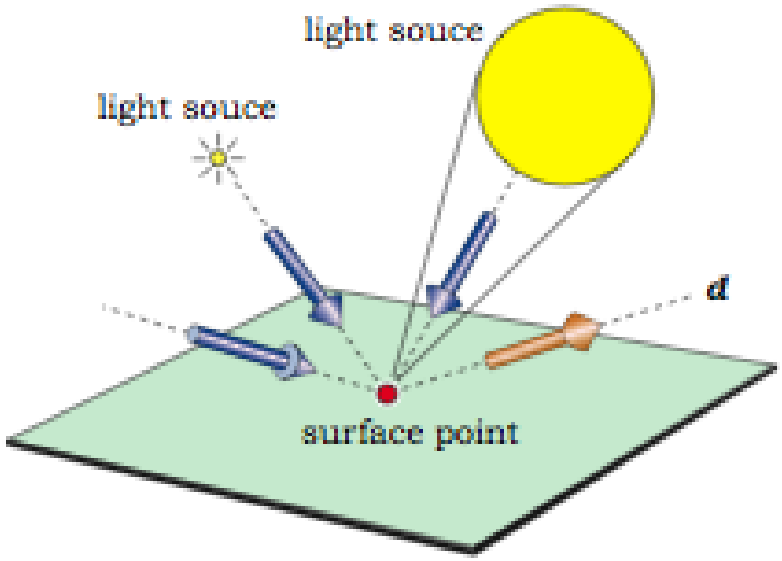
- Important features:
  - Method to compute light incident to a point
  - Returns Color and Direction
- Point light:
  - Position
  - Color
- Directional light:
  - Vector direction
  - Color

Light

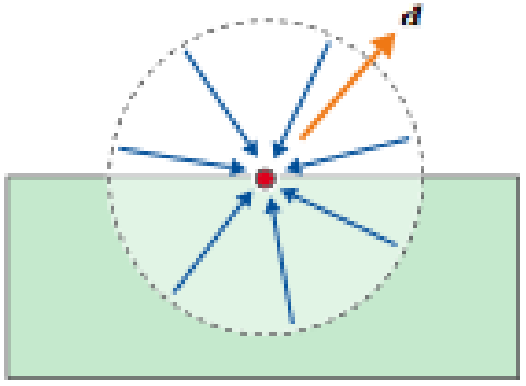
- Stored in global memory
  - A single point light
  - Could treat it as a sphere around the point
- Other lights you should just hard code (for now)
- Program 2 light will probably be hard coded

Shading

- The shading step is the key aspect of ray tracing



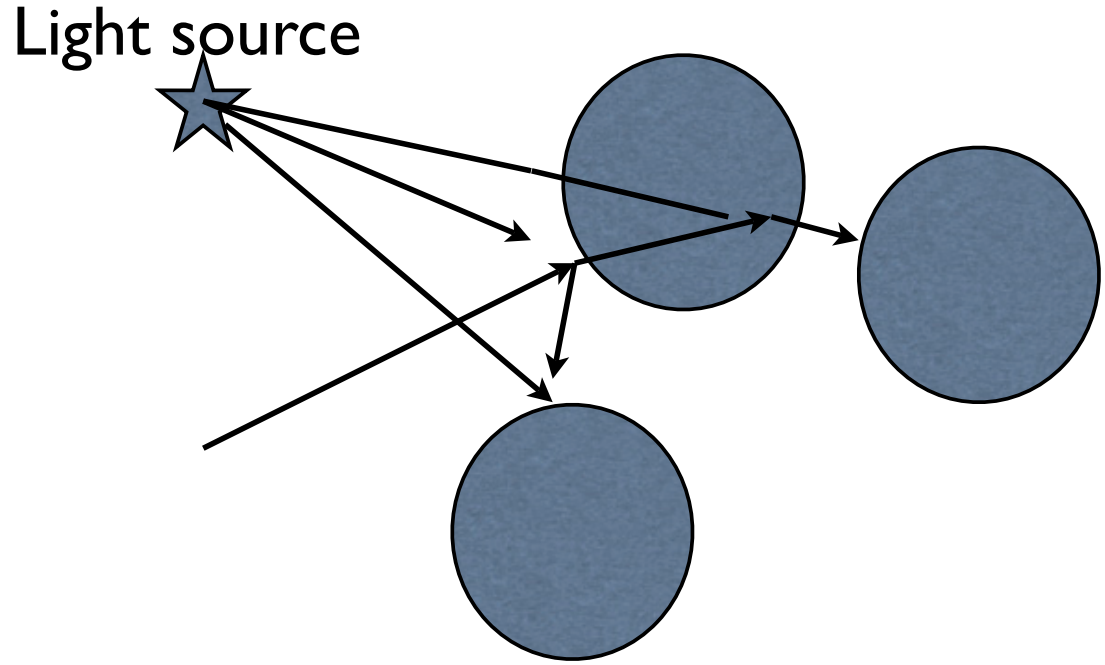
(a)



(b)

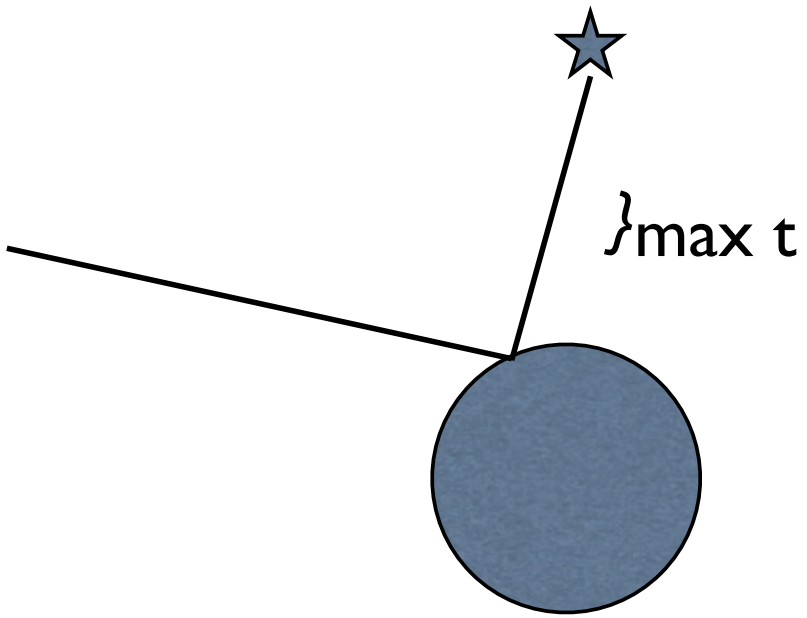
Shading

- Path tracing: consider light from all directions
- Ray tracing: consider the dominant directions:
  - direct (unobstructed from light source)
  - reflection
  - refraction



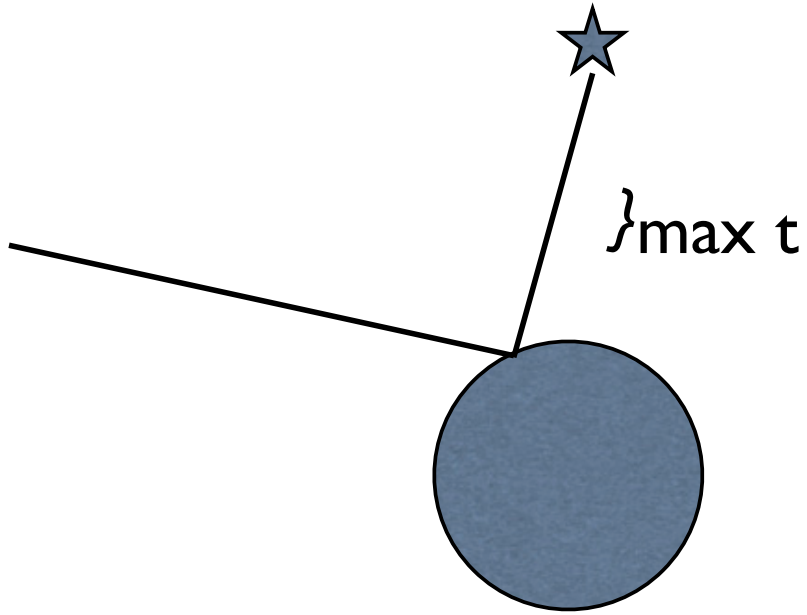
Shadow rays

- Shadows are computed by tracing rays from (to) the light source
- Intersection point: $\vec{P} = \vec{O} + t\vec{V}$
- Origin: $\vec{P}$
- Direction:
- max t: 1.0



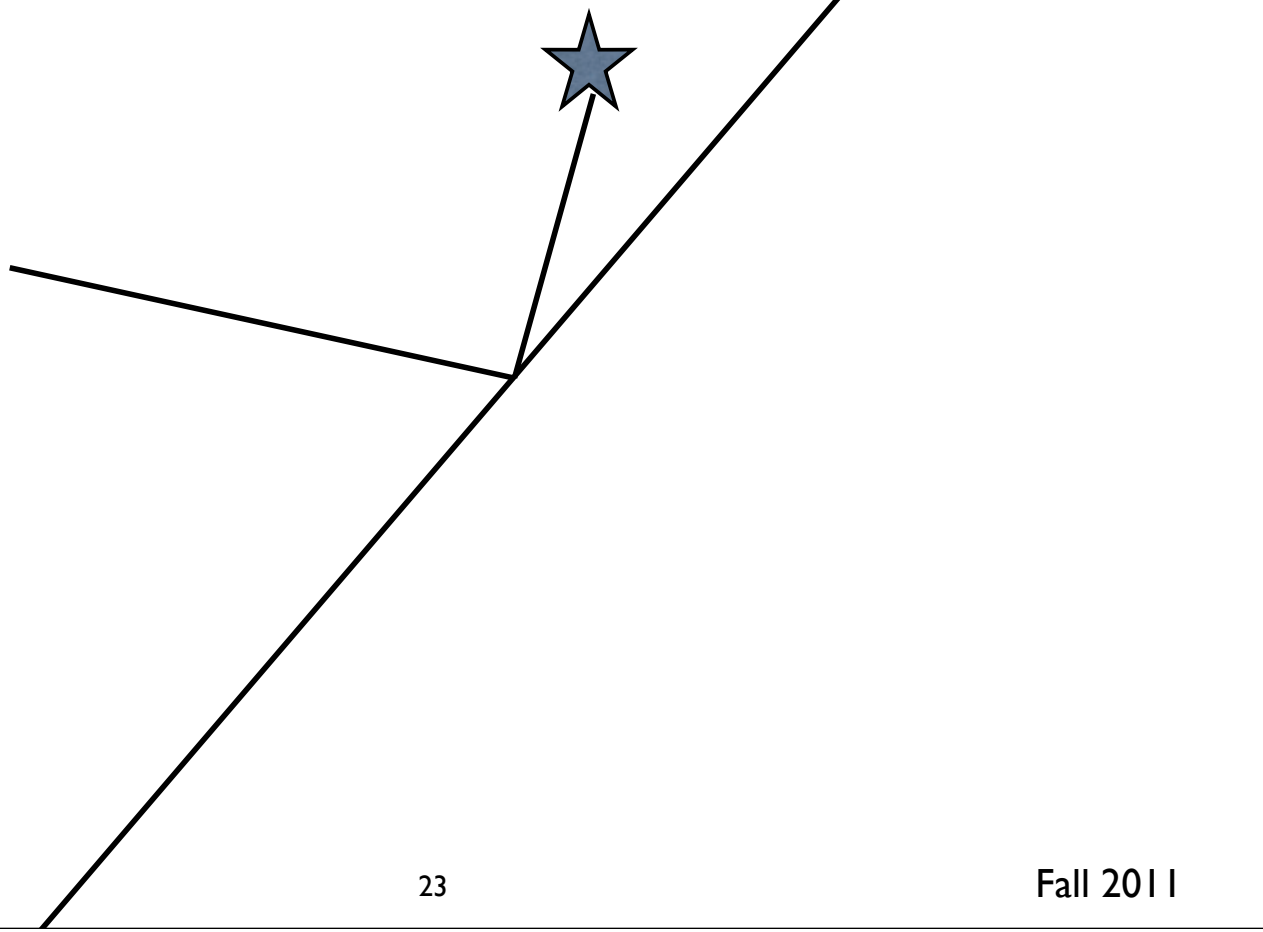
Shadow ray bugs

- Two bugs might show up when you do this:
  - False shadows (considering rays <0 or > 1)
  - Freckles (considering rays == 0)



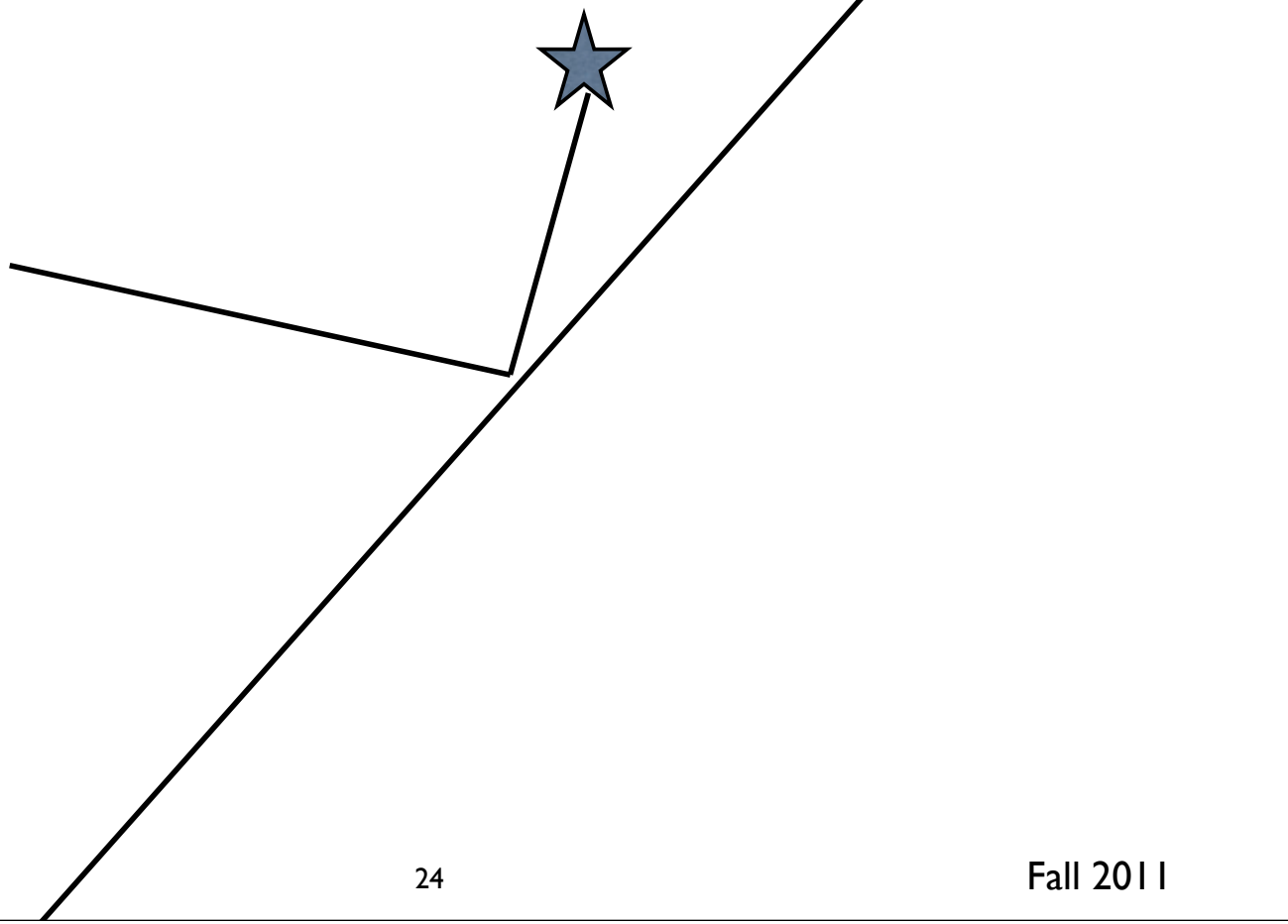
Numerical precision

- Zoomed in: ideal



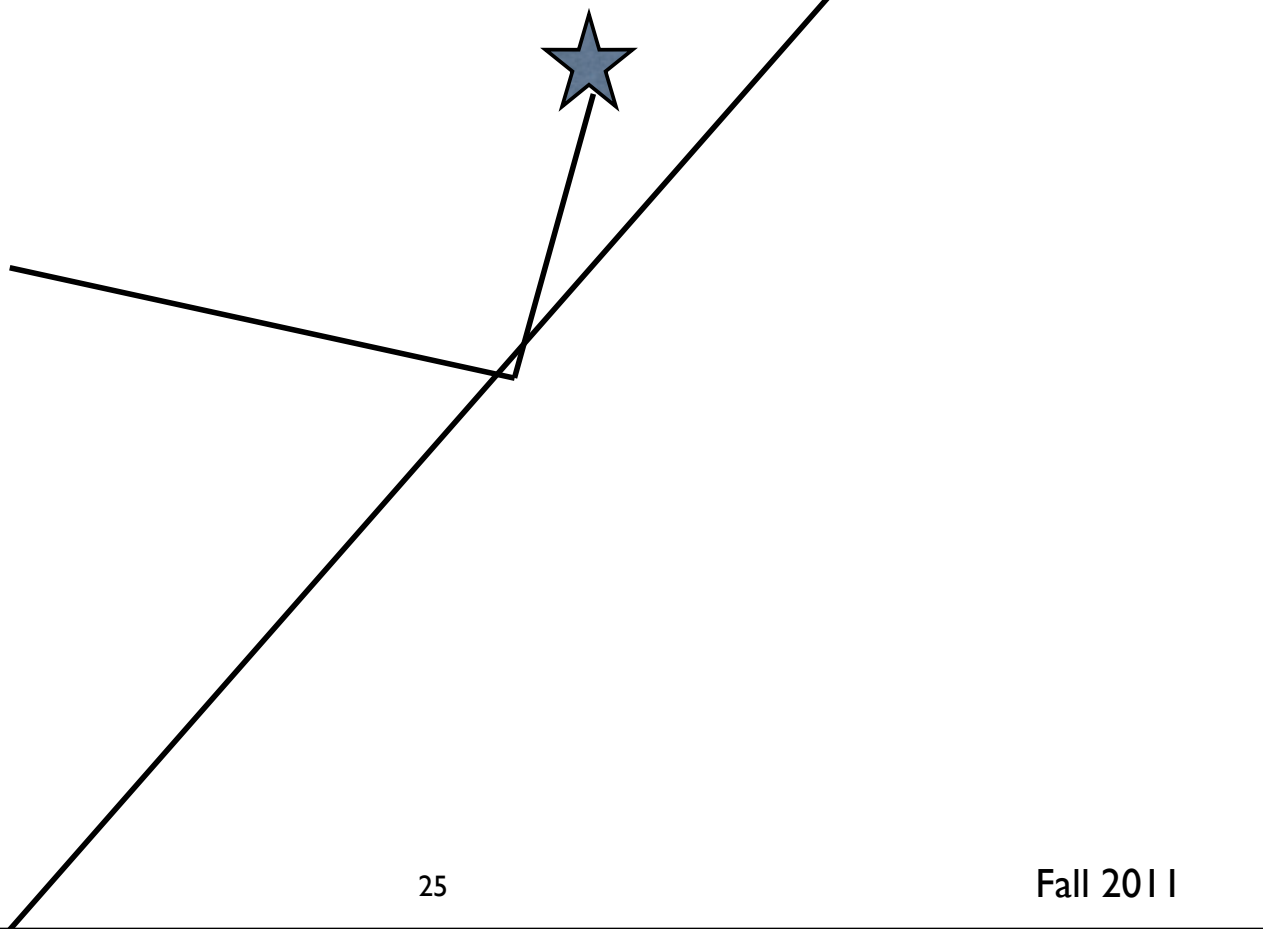
Numerical precision

- Zoomed in $(x1e7 - ish)$: short (numerical roundoff)



Numerical precision

- Zoomed in $(x1e7 - ish)$: short (numerical roundoff)

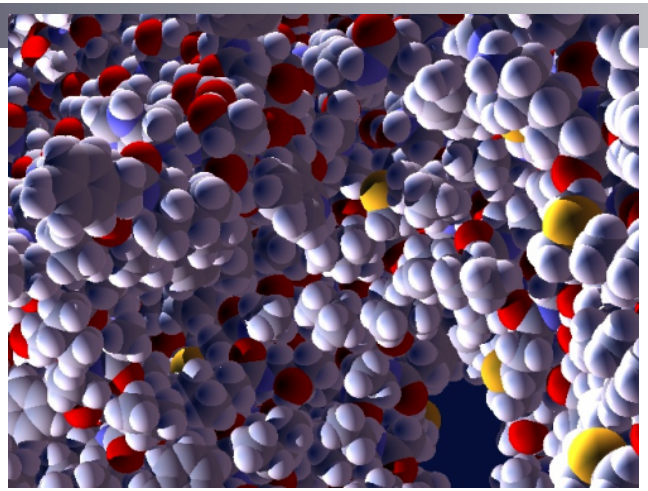


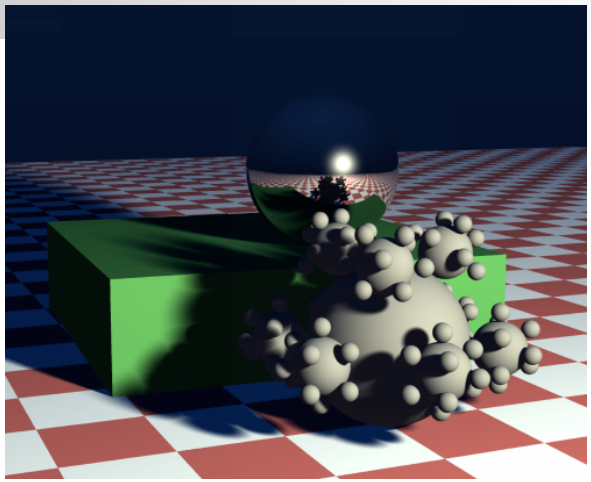
Solutions:

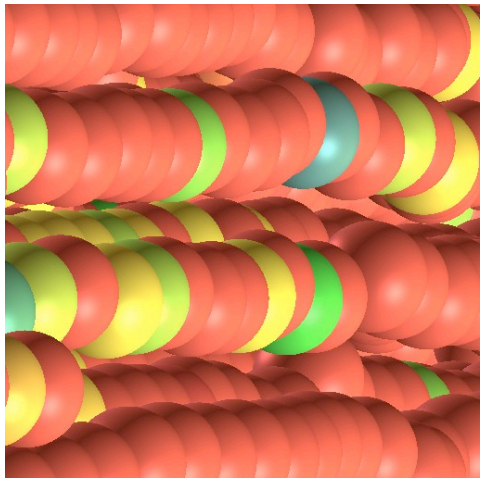
- Only consider intersections where t>small_num
- Offset ray in normal direction: P+=N*small_num
- Offset ray in light source direction:
  - P+=(L-P)*small_num

- small_num = 1.e-6

Shading models

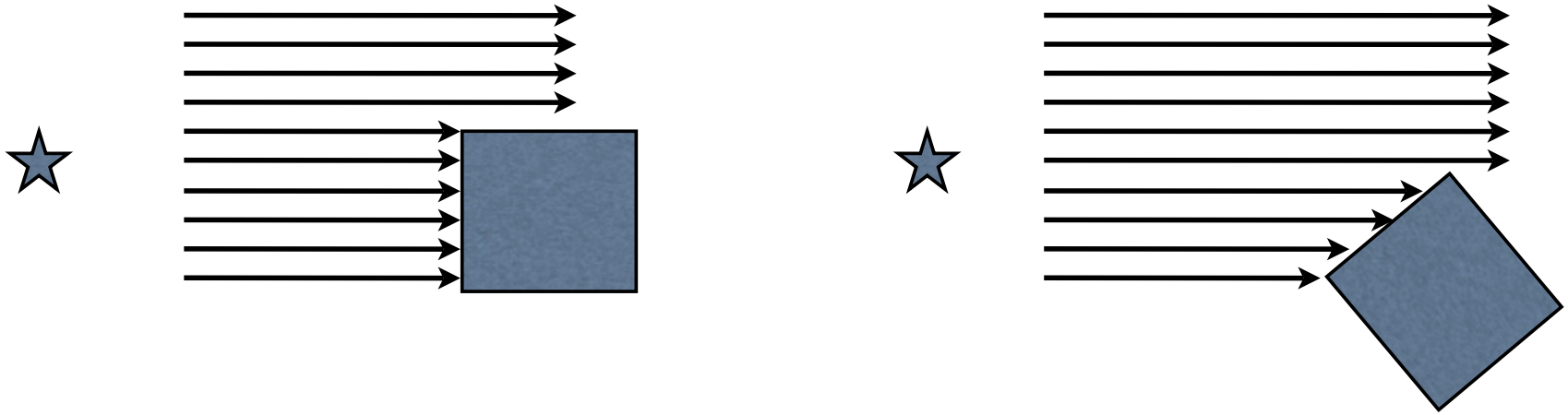






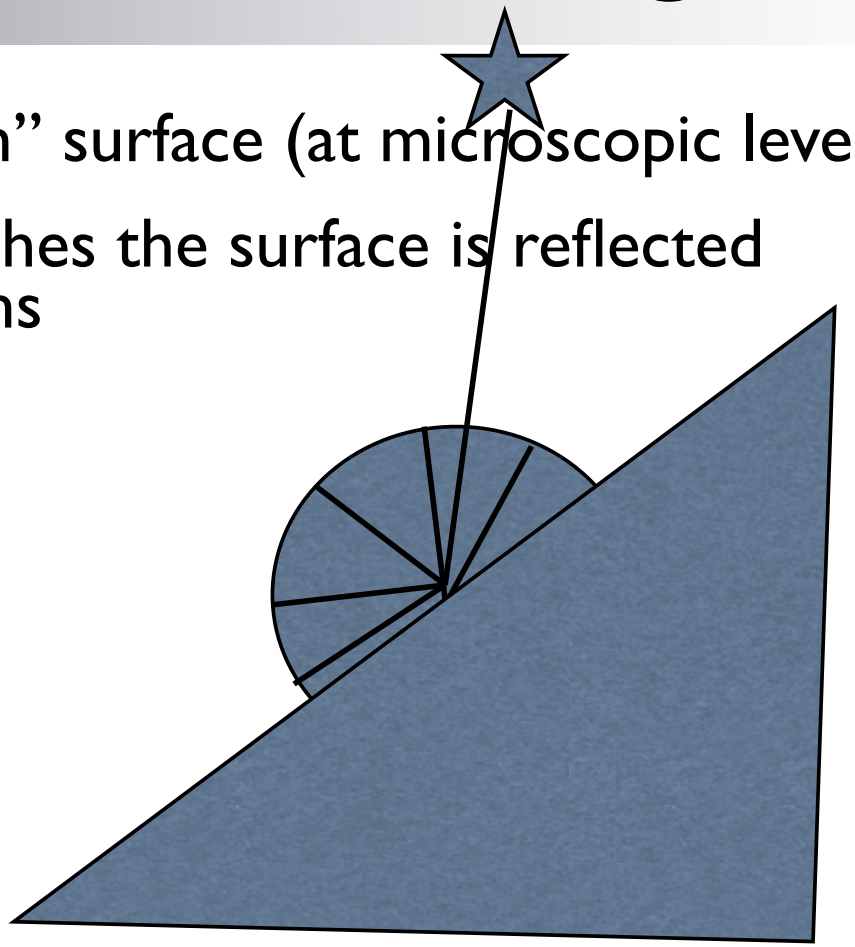
Lambert's cosine law

- Light reaching surface is proportional to projected area: cos θ



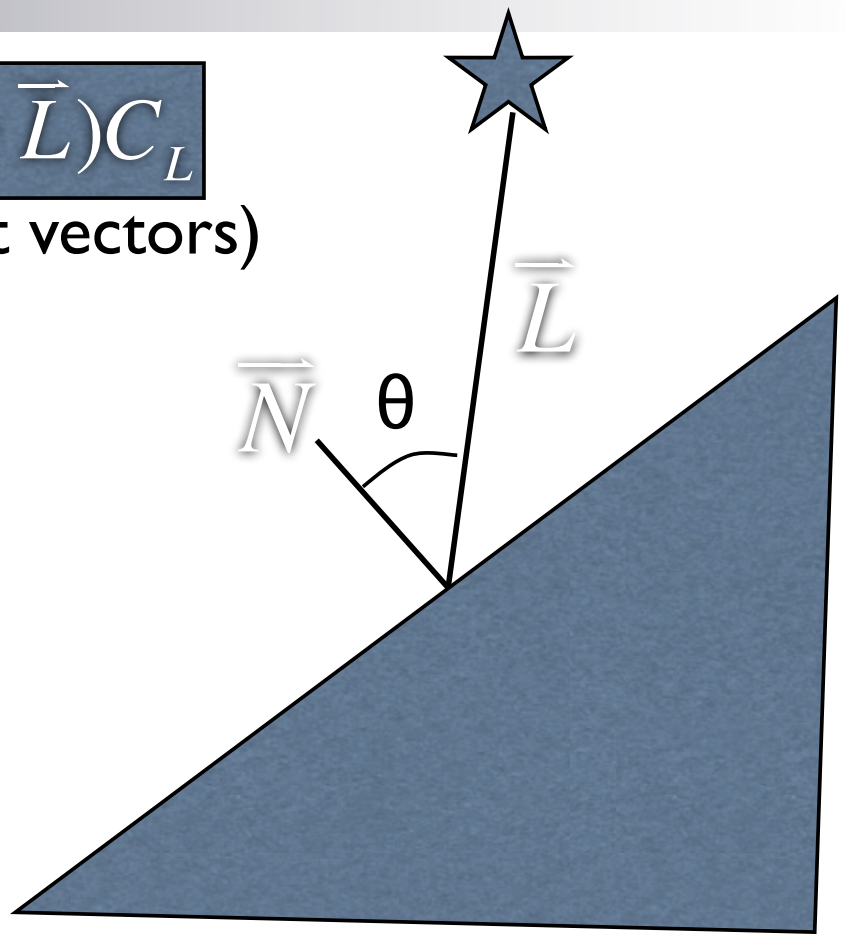
Lambertian shading

- Comes from a "rough" surface (at microscopic level
- Simple: light that reaches the surface is reflected equally in all directions



Lambertian shading

- Color at surface: $(\vec{N} \cdot \vec{L})C_L$
- (where N and L are unit vectors)



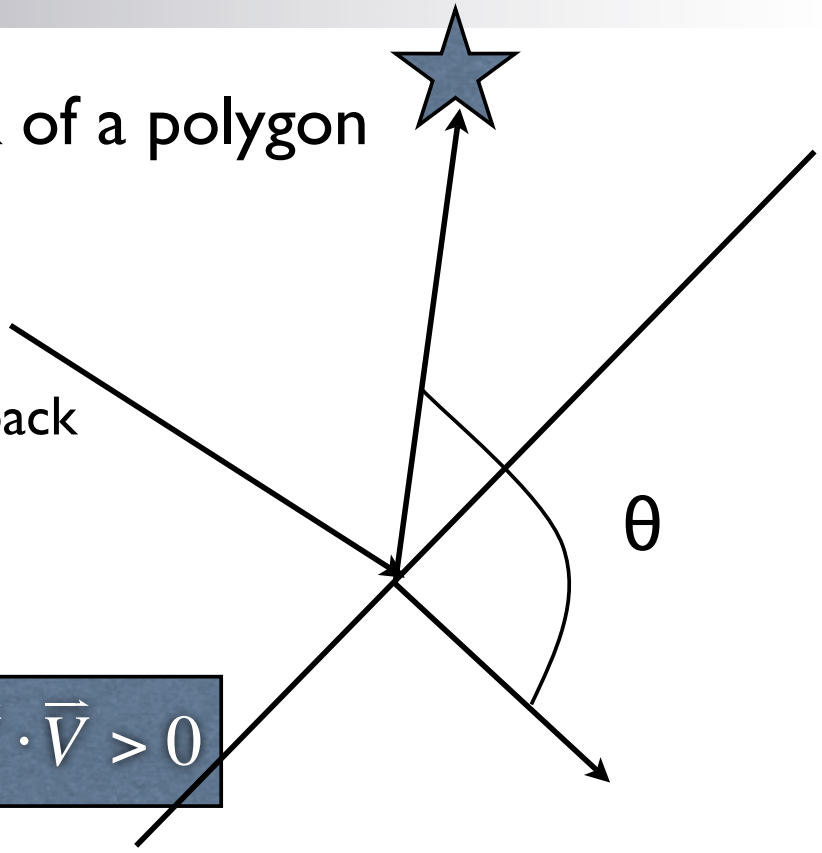
Ambient light

- With this mechanism, the light in a shadowed region is 0 (black)
- To avoid this, use "ambient" lighting

$$C_{ray} = C_{surface}\left[(\vec{N}\cdot\vec{L})C_{Light}K_d + C_{ambient}K_a\right]$$

Two-sided lighting

- What if light hits the back of a polygon
- Options:
  - Black on back
  - Different materials for front/back
  - Two sided lighting:
  - okay: $\left|\vec{N} \cdot \vec{L}\right|$
  - better: Negate $\vec{N}$ if $\vec{N} \cdot \vec{V} > 0$



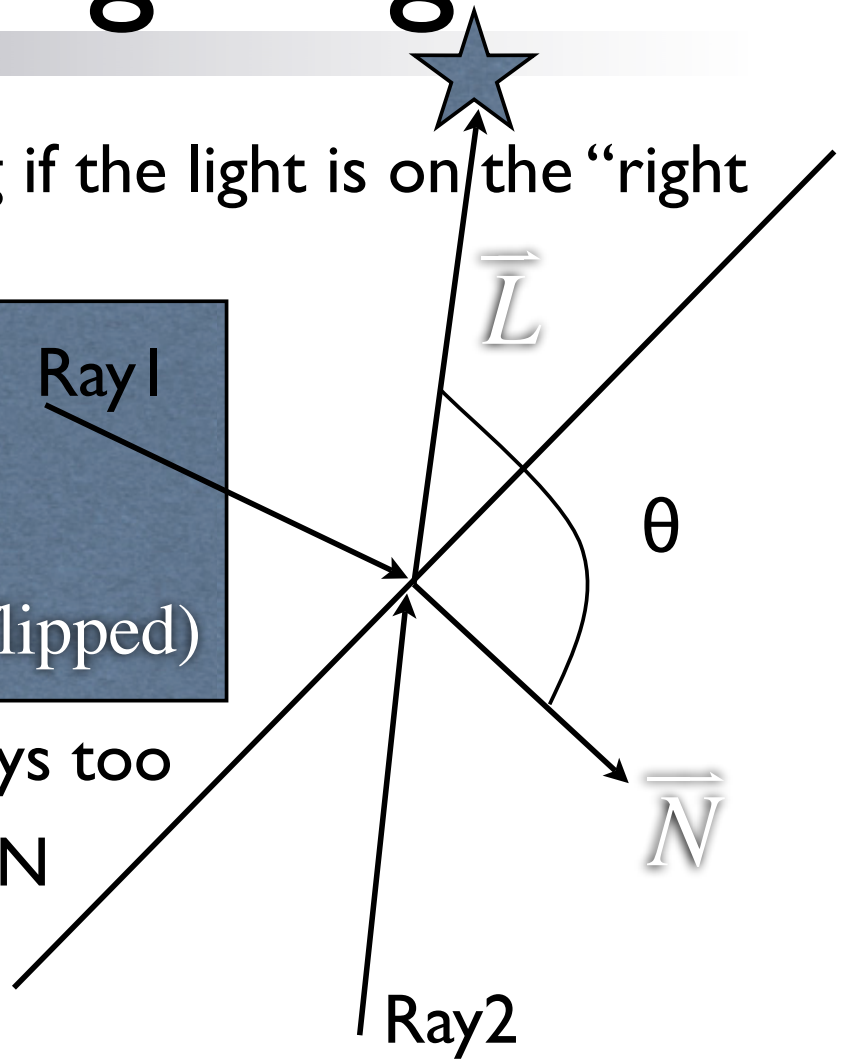
Two-sided lighting

- You might consider checking if the light is on the "right side" of the object

$$\text{sign}(-\vec{V}\cdot\vec{N}) == \text{sign}(\vec{L}\cdot\vec{N})$$

$$\left(\vec{V}\cdot\vec{N}\right)\left(\vec{L}\cdot\vec{N}\right) < 0 : \text{lit}$$

$$\left(\vec{L}\cdot\vec{N}\right) > 0 : \text{lit (when normal flipped)}$$



- Can avoid casting shadow rays too
- Or use absolute value for L ·N

Lambertian Shading

Compute hit position ($\vec{P} = \vec{O} + t\vec{V}$)

Call primitive to get normal ($\vec{N}$) (normalized)

$costheta = \vec{N} \cdot \vec{V}$

$if\,(costheta < 0)$

    normal =-normal

Color light = scene.ambient*Ka

foreach light source

    get C_L and $\vec{L}$

    dist=$\|\vec{L}\|, \overrightarrow{L_n} = \frac{\vec{L}}{\|\vec{L}\|}$

    $cosphi = \vec{N} \cdot \overrightarrow{L_n}$

    if($cosphi > 0$)

        if(!intersect with $0 < t < dist$)

            light += $C_L * (Kd * cosphi)$

result=light*surface color

Planes

- The equation for a plane is: $ax + by + cz + d = 0$
- A plane can be defined with a Vector (the normal to the plane) and a point on the plane:

$$a = N_x; b = N_y; c = N_z$$

$$d = -\vec{N} \bullet \vec{P}$$

- Alternative form of plane equation:

$$\vec{N} \bullet \vec{P} + d = 0$$

Ray-plane intersection

- To find the intersection of a ray with a plane, determine where both equations are satisfied at the same time:

$$\vec{N} \bullet \vec{P} + d = 0 \text{ and } \vec{P} = \vec{O} + t\vec{V}$$

Ray-plane intersection

- To find the intersection of a ray with a plane, determine where both equations are satisfied at the same time:

$$\vec{N} \bullet \vec{P} + d = 0 \text{ and } \vec{P} = \vec{O} + t\vec{V}$$

$$\vec{N} \bullet \left(\vec{O} + t\vec{V} \right) + d = 0$$

Ray-plane intersection

- To find the intersection of a ray with a plane, determine where both equations are satisfied at the same time:

$$\vec{N} \bullet \vec{P} + d = 0 \text{ and } \vec{P} = \vec{O} + t\vec{V}$$

$$\vec{N} \bullet \left(\vec{O} + t\vec{V} \right) + d = 0$$

$$\vec{N} \bullet \vec{O} + t\vec{N} \bullet \vec{V} + d = 0$$

Ray-plane intersection

- To find the intersection of a ray with a plane, determine where both equations are satisfied at the same time:

$$\vec{N} \bullet \vec{P} + d = 0 \text{ and } \vec{P} = \vec{O} + t\vec{V}$$

$$\vec{N} \bullet \left(\vec{O} + t\vec{V} \right) + d = 0$$

$$\vec{N} \bullet \vec{O} + t\vec{N} \bullet \vec{V} + d = 0$$

$$t\vec{N} \bullet \vec{V} = -\left(d + \vec{N} \bullet \vec{O} \right)$$

Ray-plane intersection

- To find the intersection of a ray with a plane, determine where both equations are satisfied at the same time:

$$\vec{N} \bullet \vec{P} + d = 0 \text{ and } \vec{P} = \vec{O} + t\vec{V}$$

$$\vec{N} \bullet \left(\vec{O} + t\vec{V} \right) + d = 0$$

$$\vec{N} \bullet \vec{O} + t\vec{N} \bullet \vec{V} + d = 0$$

$$t\vec{N} \bullet \vec{V} = -\left(d + \vec{N} \bullet \vec{O} \right)$$

$$t = -\frac{\left(d + \vec{N} \bullet \vec{O} \right)}{\vec{N} \bullet \vec{V}}$$

Ray-plane intersection

- If $\vec{N} \bullet \vec{V} = 0$ (or close to it) then the ray is parallel to the plane
- The parameter t defines the point where the ray intersects the plane
- To determine the point of intersection, just plug it back into the ray equation

$$\left(\vec{O} + t\vec{V}\right)$$