ray tracing

image formation

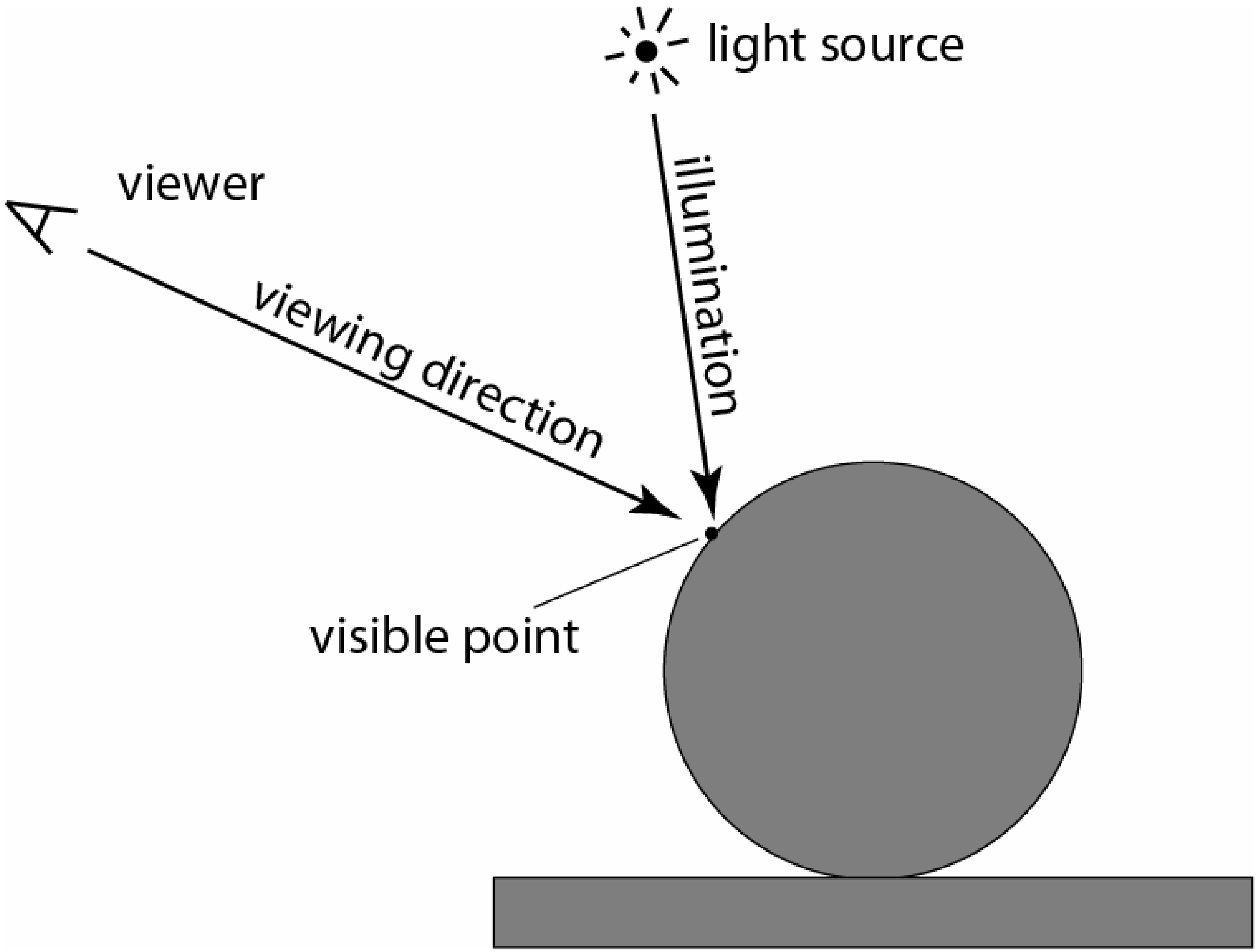
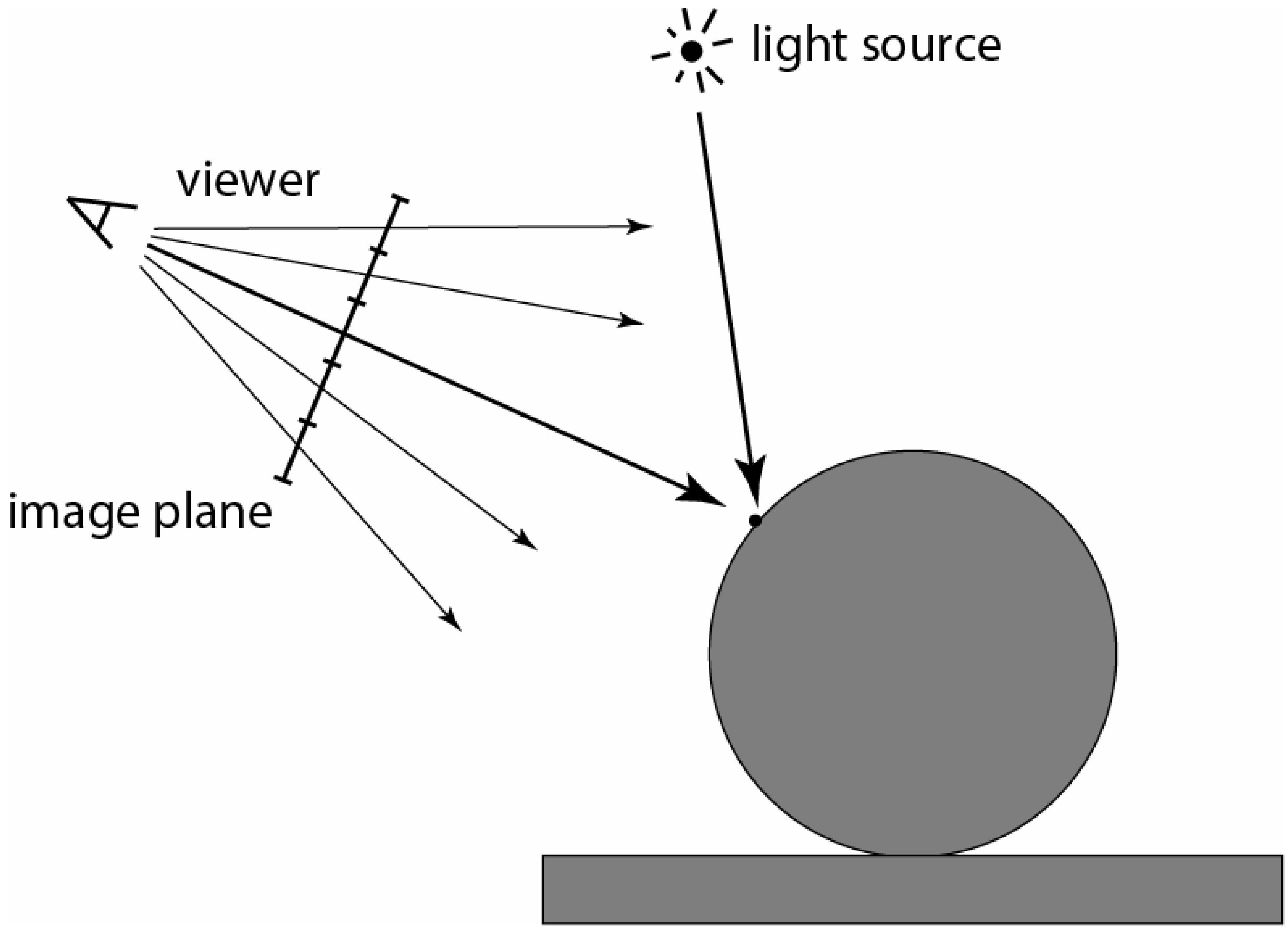


image formation



rendering

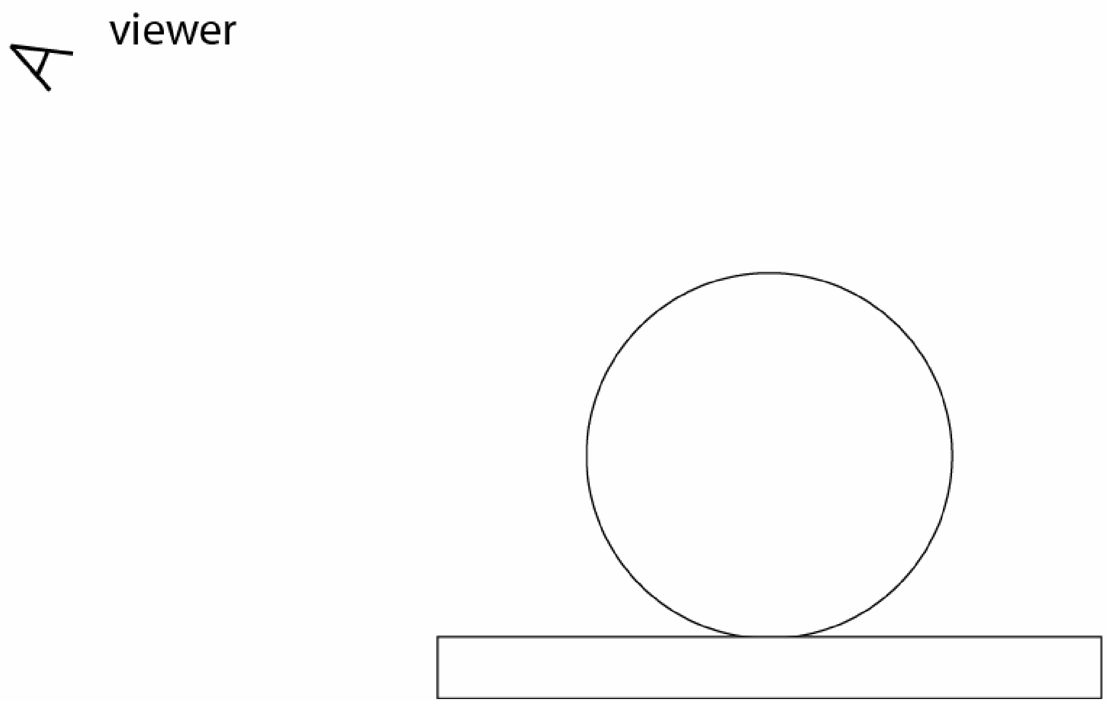
computational simulation of image formation

rendering

- given viewer

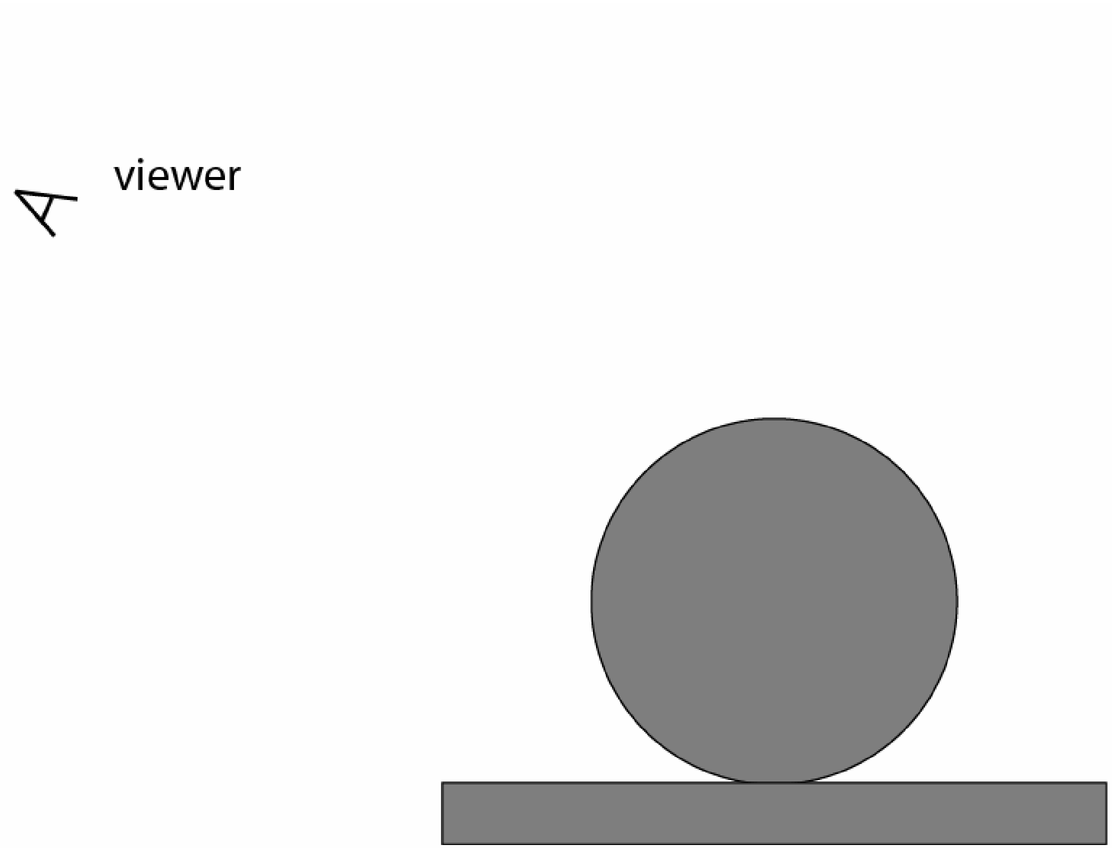
rendering

- given viewer, geometry



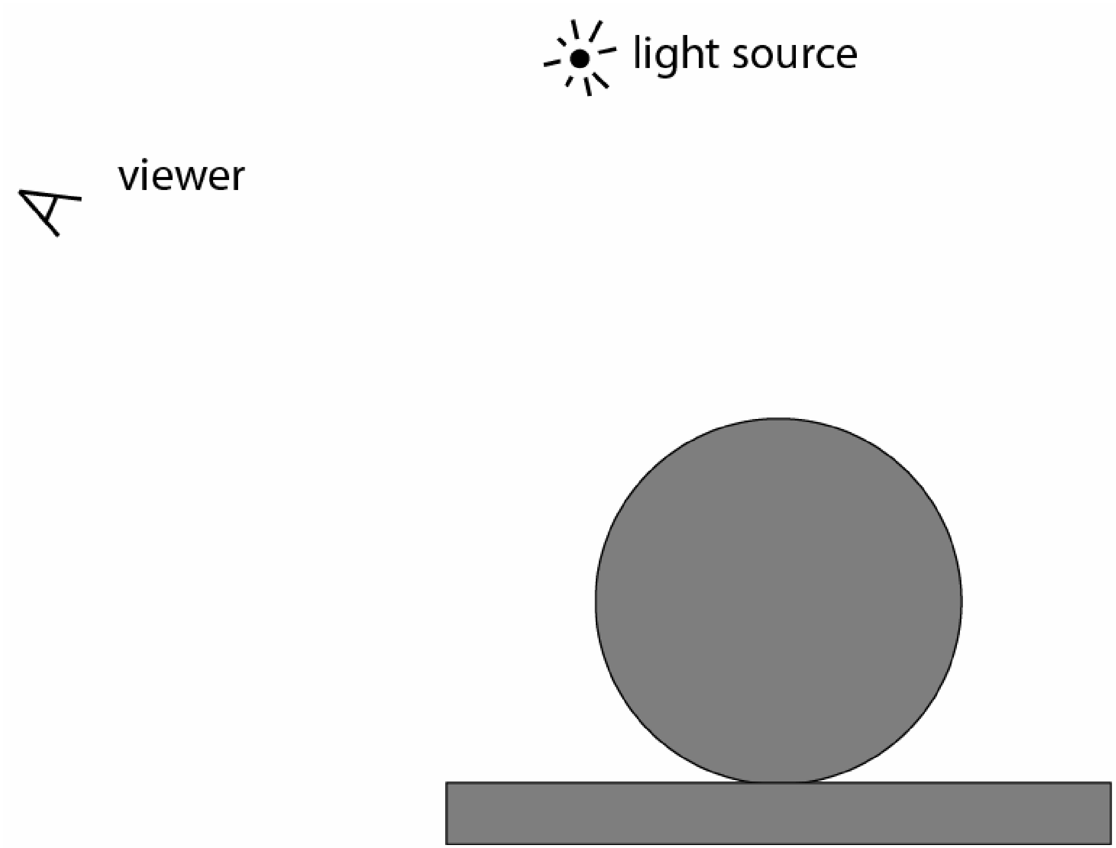
rendering

- given viewer, geometry, materials



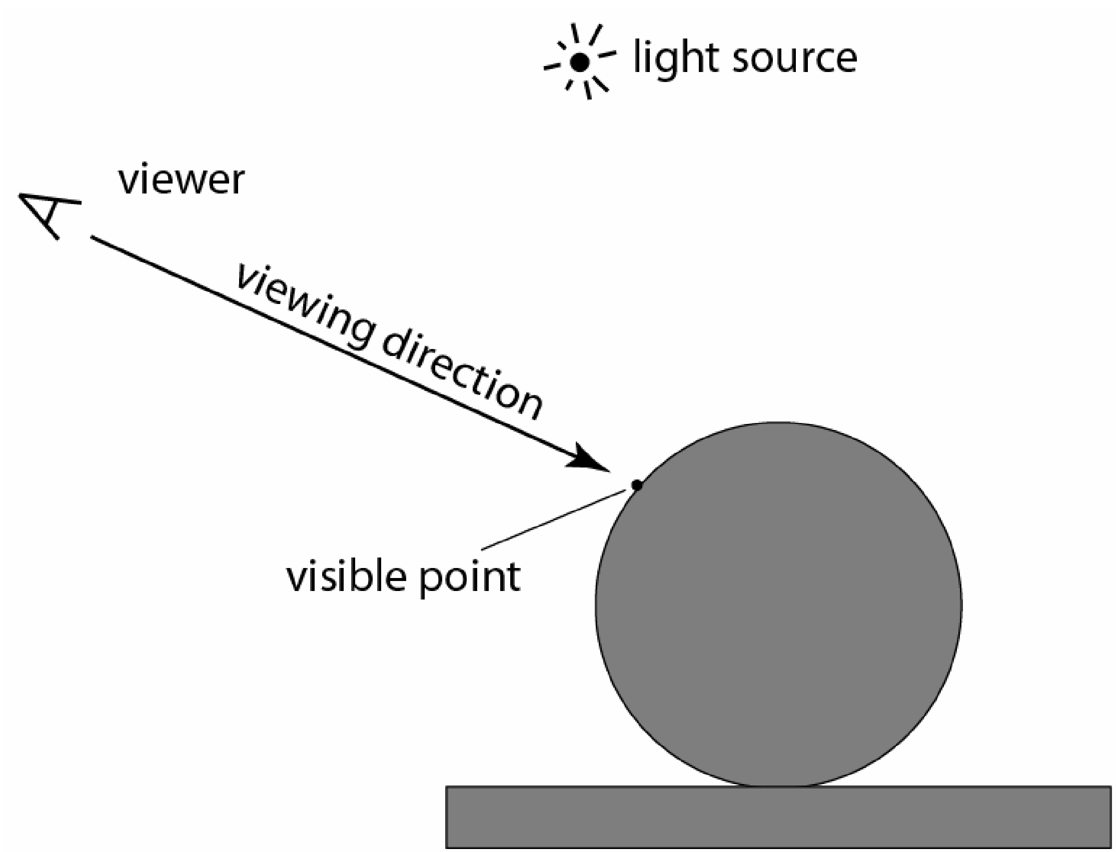
rendering

- given viewer, geometry, materials and lights



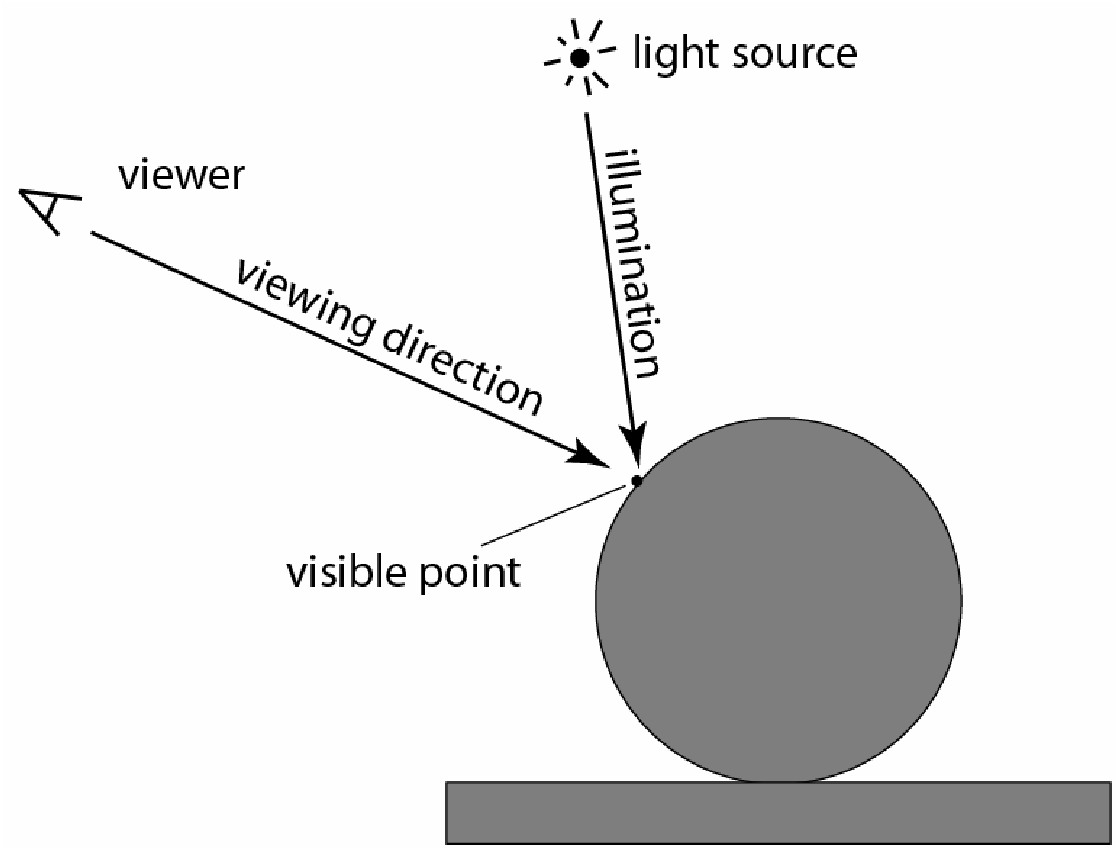
rendering

- given viewer, geometry, materials and lights
- determine visibility



rendering

- given viewer, geometry, materials and lights
- determine visibility and simulate lighting

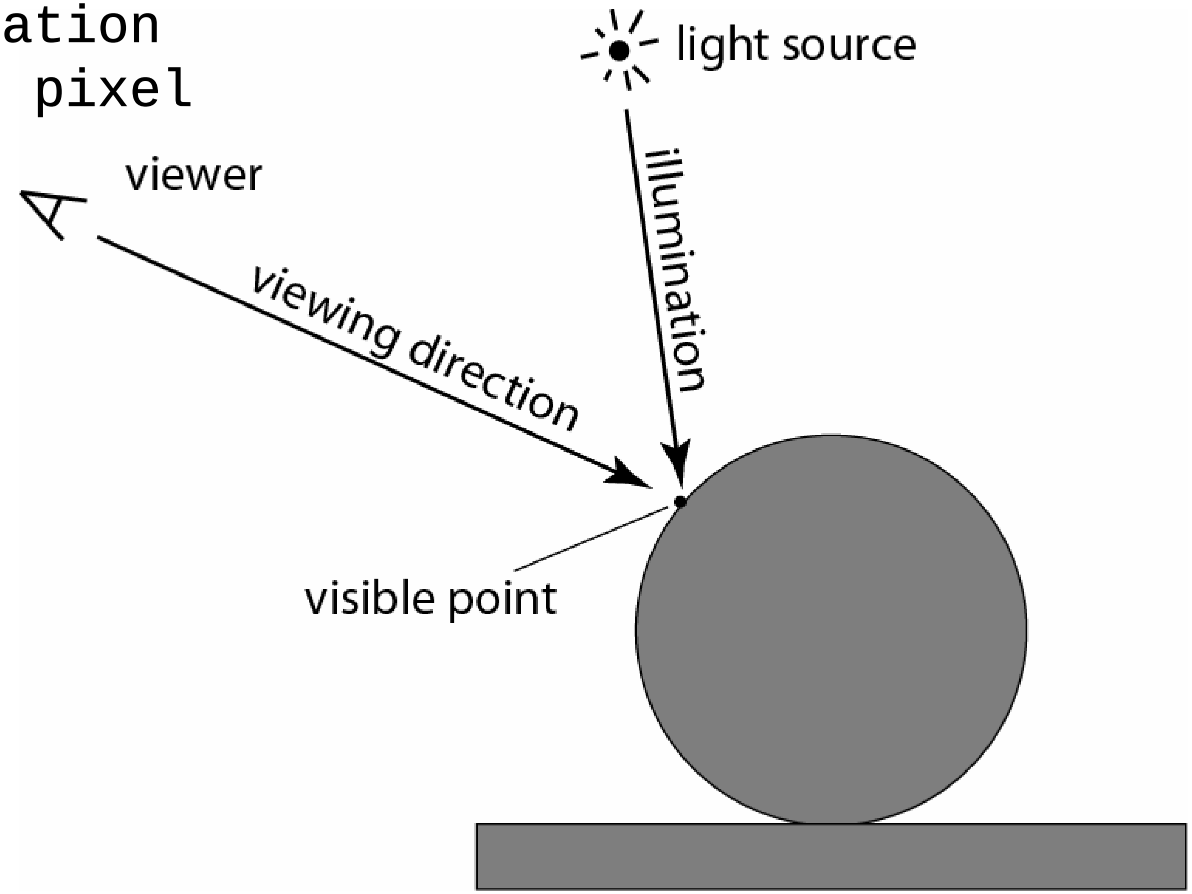


ray tracing

a particular rendering algorithm

ray tracing algorithm

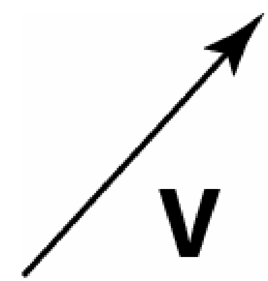
```
for each pixel {
  determine viewing direction
  intersect ray with scene
  compute illumination
  store result in pixel
}
```



vector math review – points and vectors

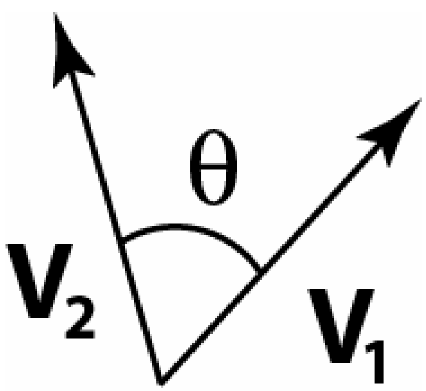
- point
  - location in 3D space
  - **P** = (x,y,z)

- vector
  - direction and magnitude
  - **V** = (u,v,w)



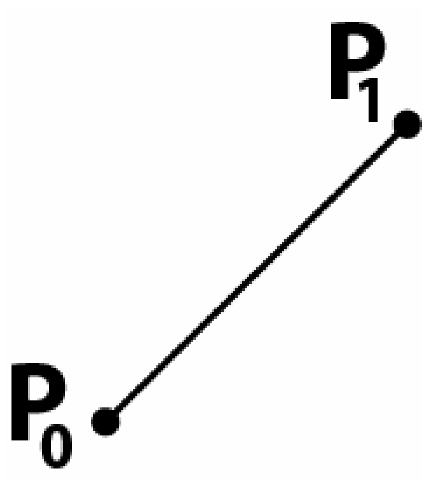
vector math review – vector operations

- dot product
  - $\mathbf{V}_1 \cdot \mathbf{V}_2 = |\mathbf{V}_1||V_2|\cos(\theta)$
- cross product
  - $|\mathbf{V}_1 \times \mathbf{V}_2| = |\mathbf{V}_1||V_2|\sin(\theta)$
  - $\mathbf{V}_1 \times \mathbf{V}_2$ is perpendicular to $\mathbf{V}_1$ and V_2



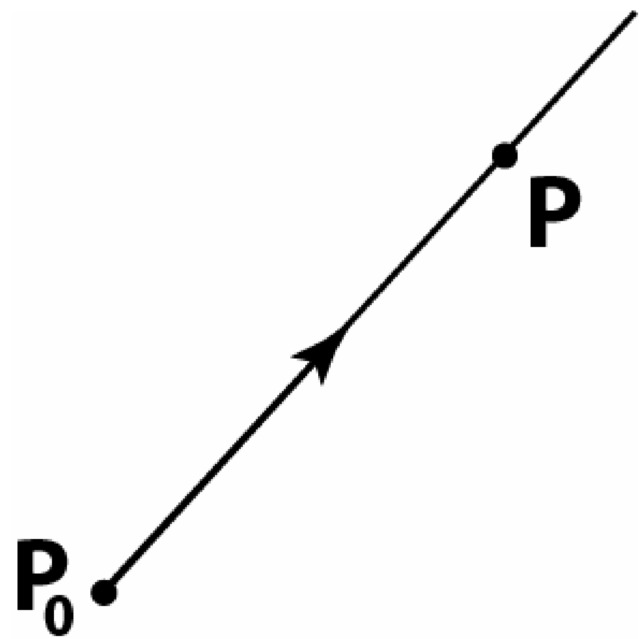
vector math review – segments

- segment
  - set of points (line) between two points
  - $\mathbf{P}(t) = \mathbf{P}_0 + t\,(\mathbf{P}_1 - \mathbf{P}_0)$ with $t \in [0,1]$



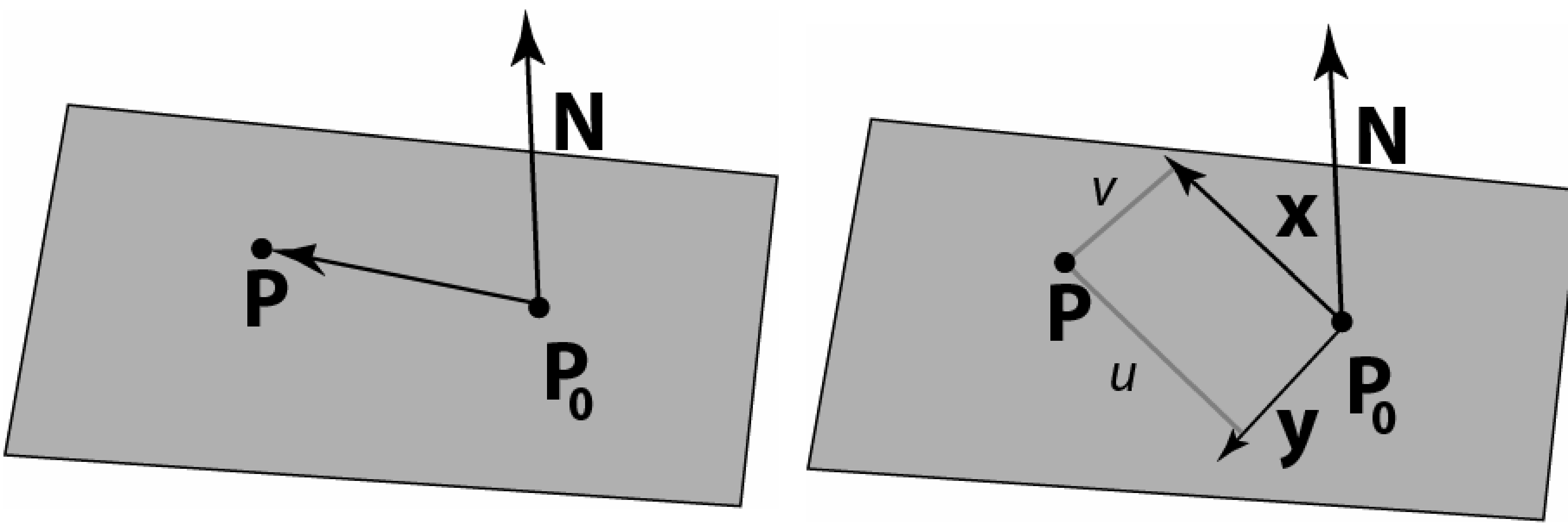
vector math review – rays

- ray
  - infinite line from point in a given direction
  - $\mathbf{P}(t) = \mathbf{P_0} + t\,\mathbf{V}$ with $t \in [0,\infty]$



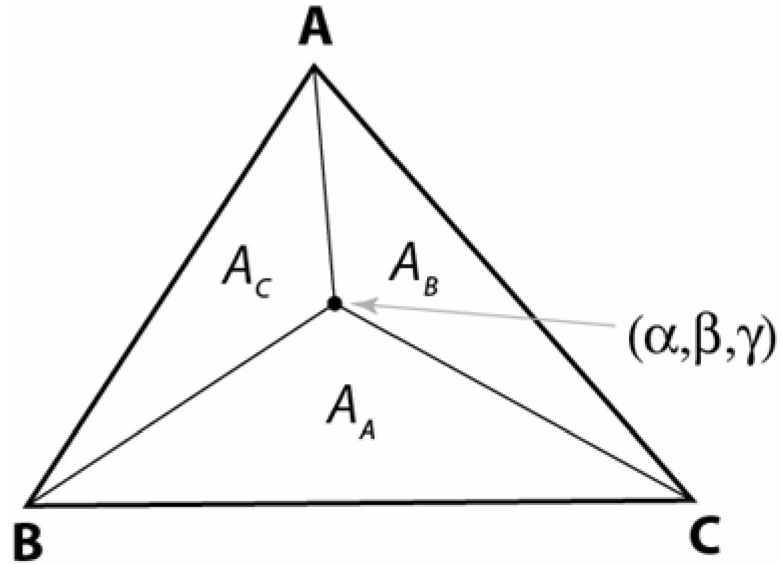
vector math review – planes

- plane
  - $\mathbf{P} \in plane \leftrightarrow \left(\mathbf{P} - \mathbf{P}_0\right) \cdot \mathbf{N} = 0 \leftrightarrow \mathbf{P} \cdot \mathbf{N} + d = 0$
  - $\mathbf{P} = \mathbf{P}_0 + u\hat{\mathbf{x}} + v\hat{\mathbf{y}}$ with $(u, v) \in [-\infty, \infty]^2$
  - $\mathbf{N} = \hat{\mathbf{x}} \times \hat{\mathbf{y}}$



vector math review - triangle

- triangle
  - points in three subspaces defined by triangle edges



- baricentric coordinates

$$\mathbf{P} = \alpha\,\mathbf{A} + \beta\,\mathbf{B} + \gamma\,\mathbf{C}$$

$$\mathbf{P} \in triangle \leftrightarrow \alpha + \beta + \gamma = 1$$

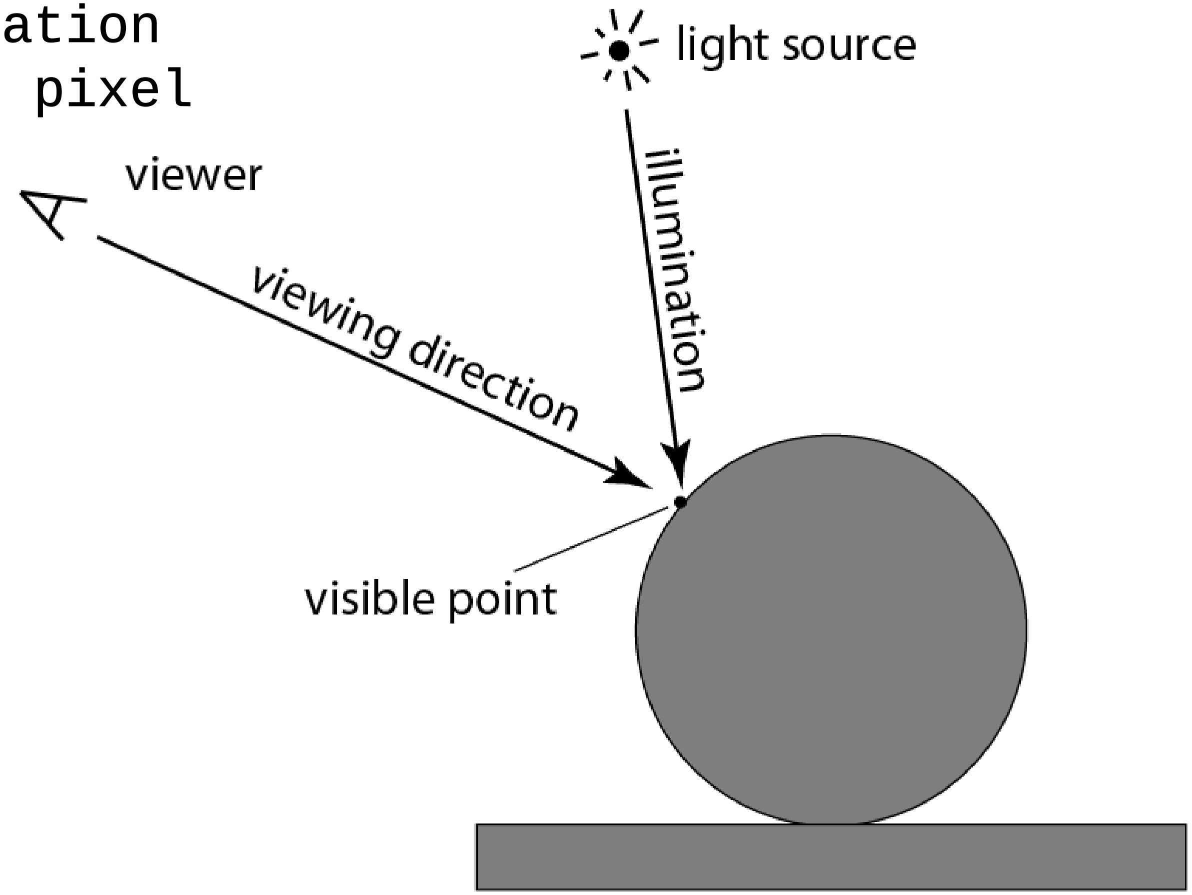
$$\mathbf{P} = \mathbf{C} + \alpha\left(\overrightarrow{\mathbf{CA}}\right) + \beta\left(\overrightarrow{\mathbf{CB}}\right)\ with\ \alpha \geq 0, \beta \geq 0, \alpha + \beta \leq 1$$

  - geometric interpretation: relative area

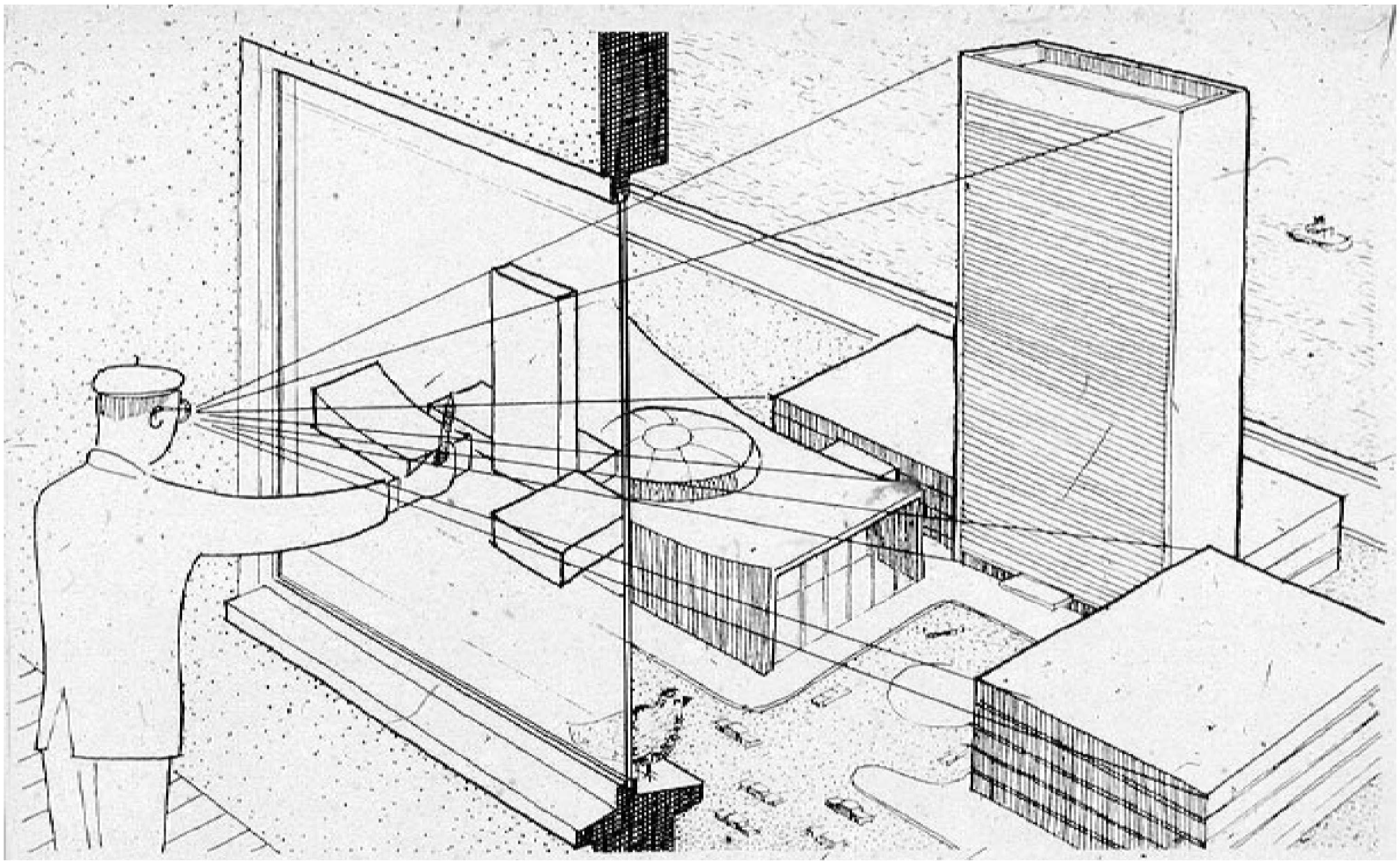
$$\gamma = A_C \,/\, A_{triangle}$$

ray tracing algorithm

```
for each pixel {
  determine viewing direction
  intersect ray with scene
  compute illumination
  store result in pixel
}
```



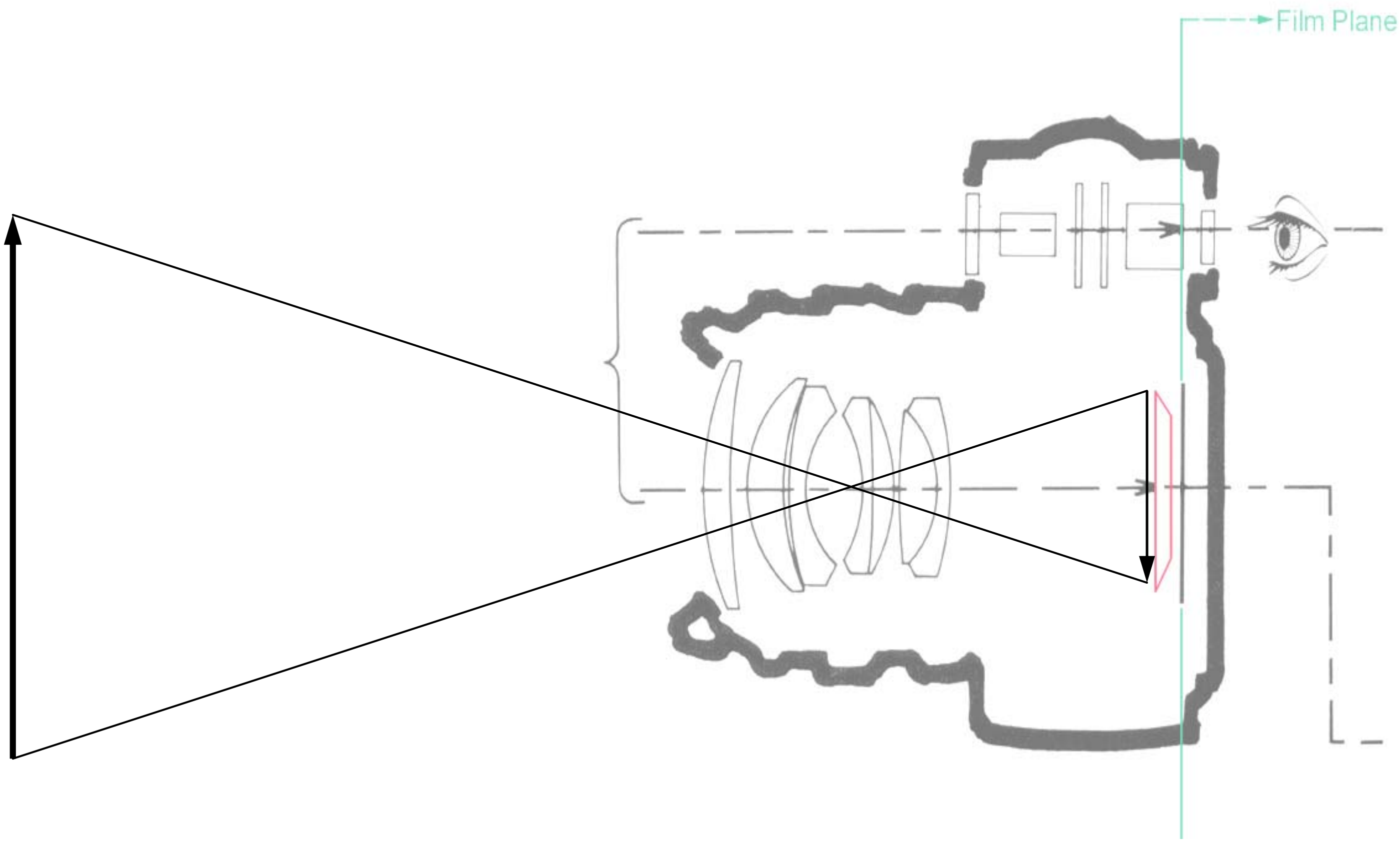
viewer model – window analogy



[Marschner 2004 – original unkwon]

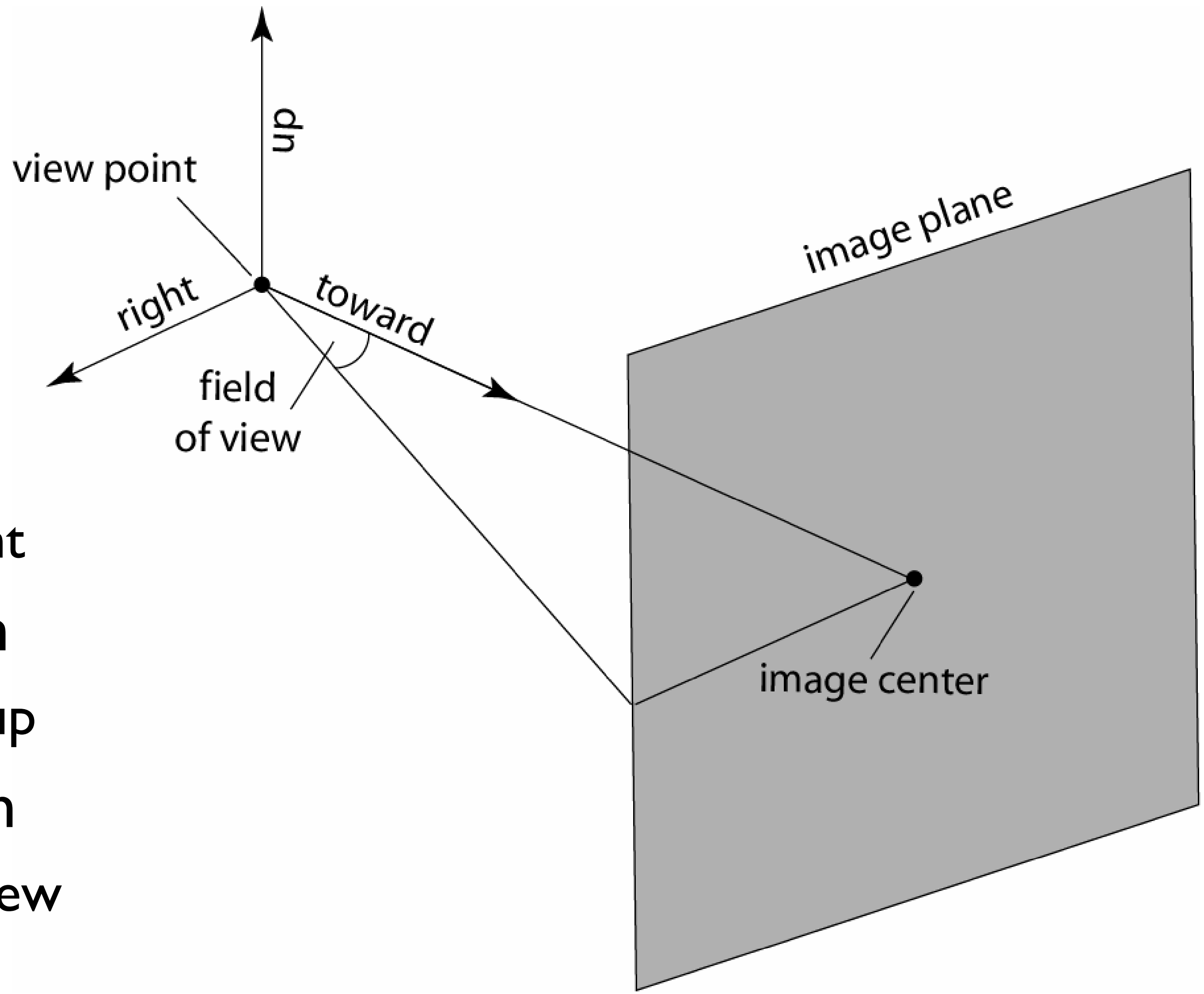
viewer model - photography

- equiv. to pinhole photography



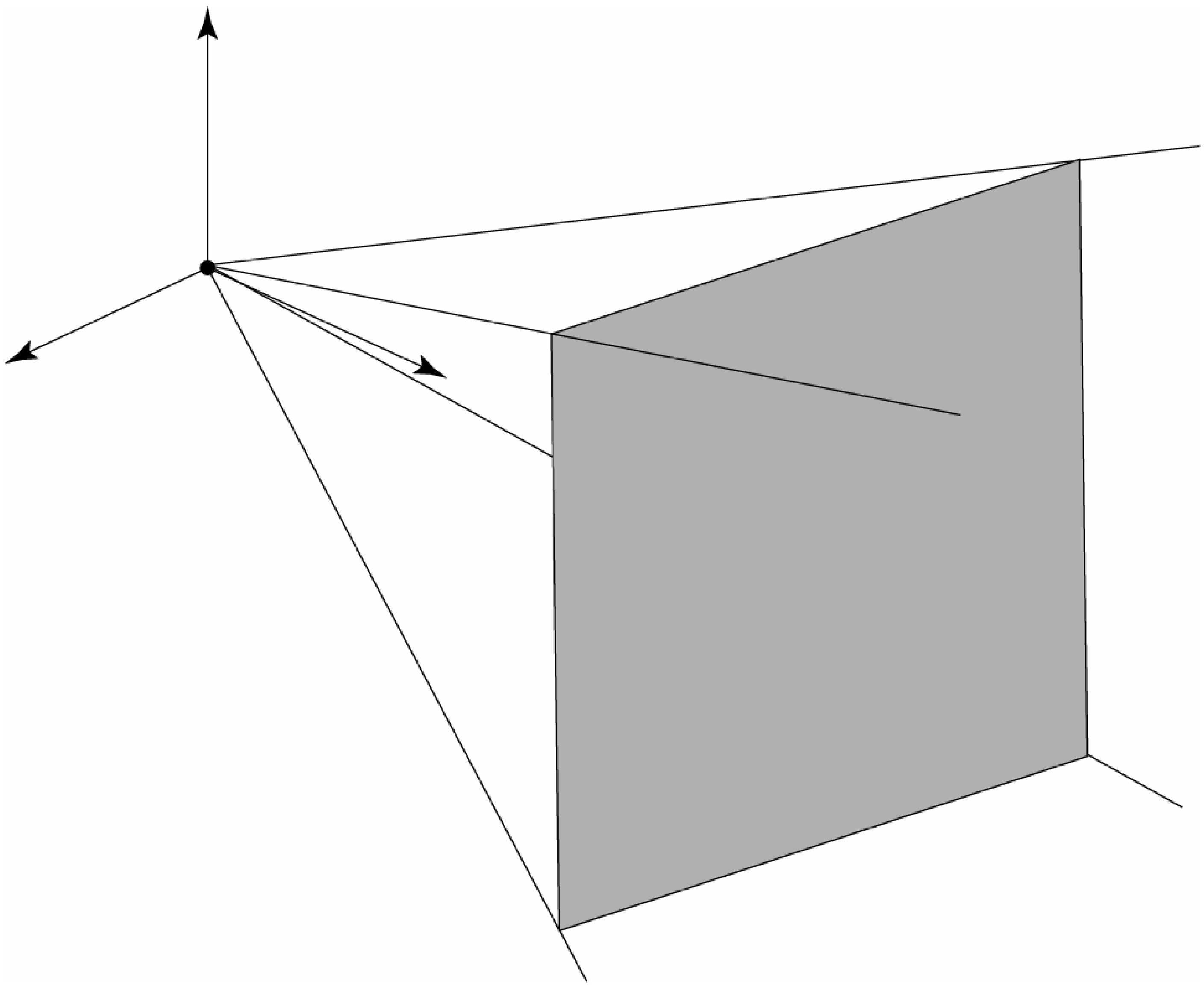
[Marschner 2004 – original unkwon]

viewer model - representation

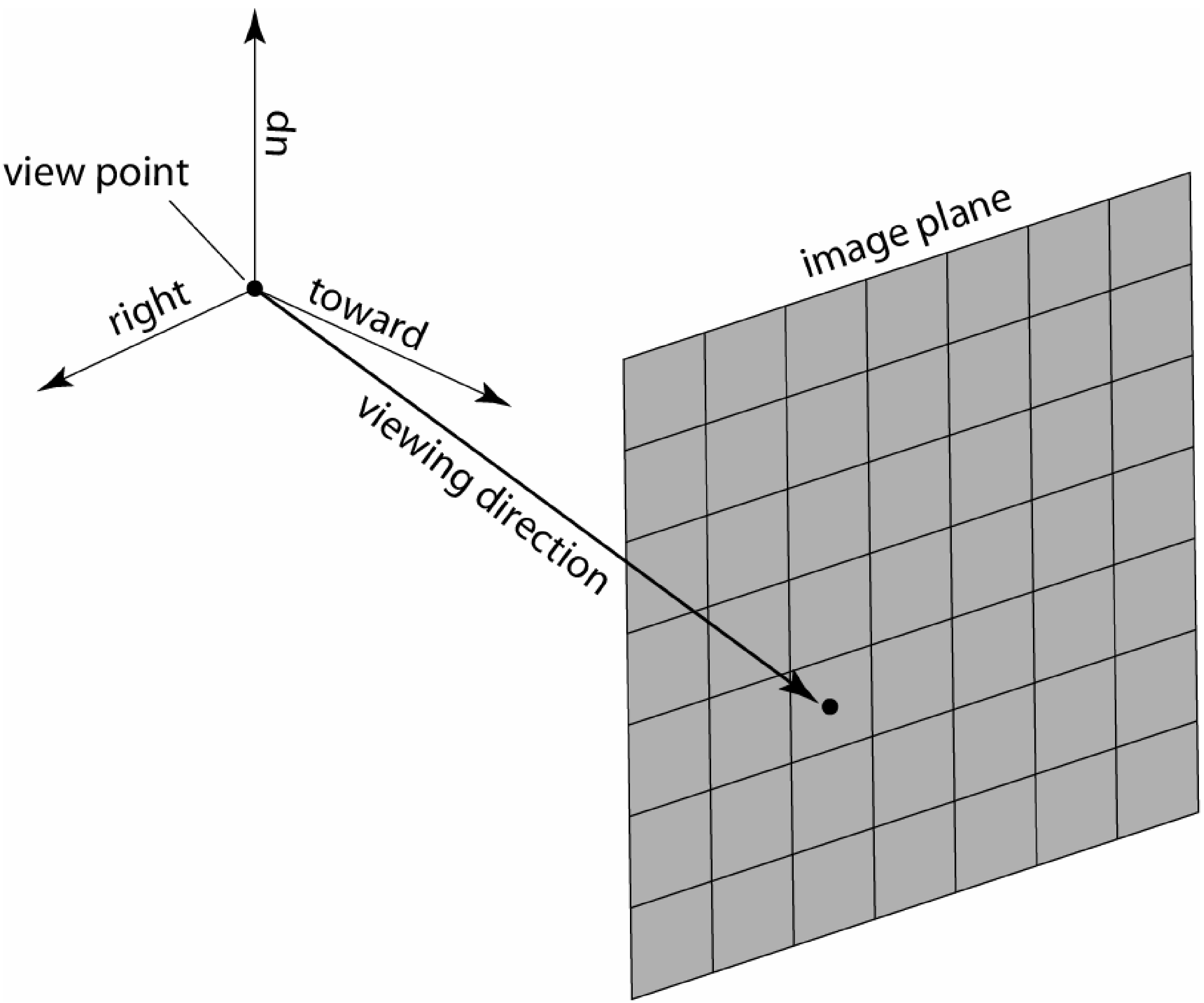


- position
  - view point
- orientation
  - toward, up
- focal length
  - field of view

viewer model – view frustum



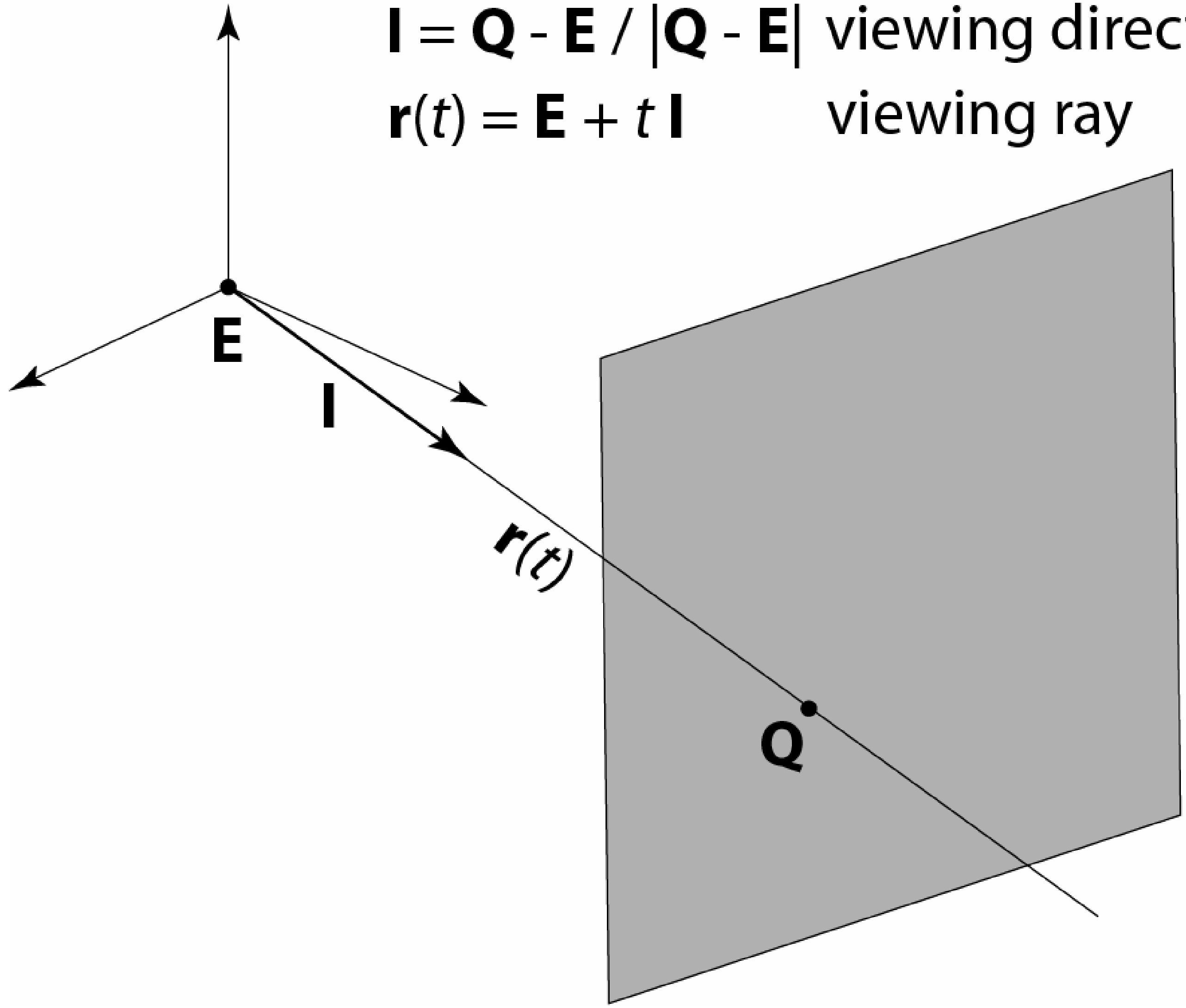
generate viewing rays



generate viewing rays

$\mathbf{l} = \mathbf{Q} - \mathbf{E} / |\mathbf{Q} - \mathbf{E}|$ viewing direction

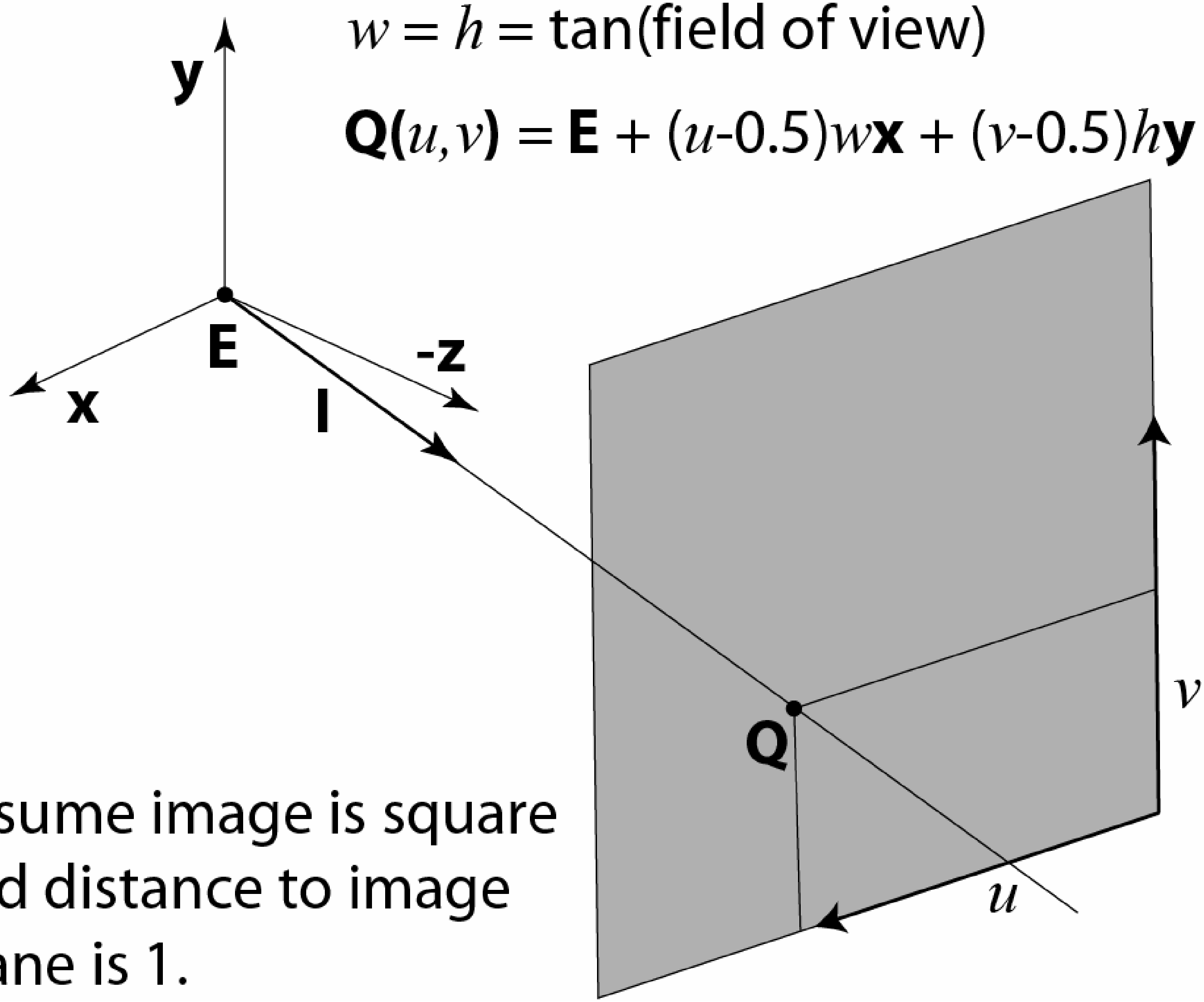
$\mathbf{r}(t) = \mathbf{E} + t\,\mathbf{l}$ viewing ray



generate viewing rays

$w = h = \tan(\text{field of view})$

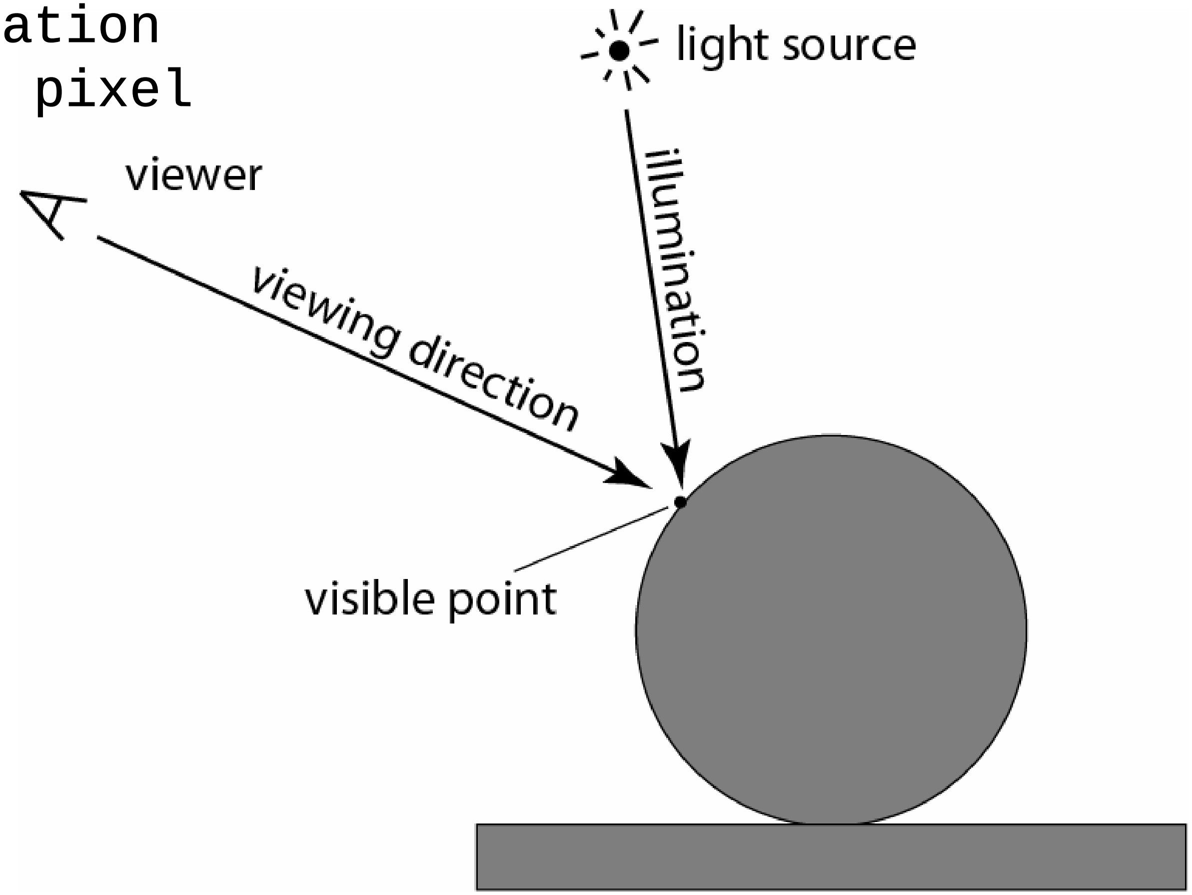
$\mathbf{Q}(u,v) = \mathbf{E} + (u\text{-}0.5)w\mathbf{x} + (v\text{-}0.5)h\mathbf{y} - \mathbf{z}$



Assume image is square and distance to image plane is 1.

ray tracing algorithm

```
for each pixel {
  determine viewing direction
  intersect ray with scene
  compute illumination
  store result in pixel
}
```



geometry model

- simple shapes
  - spheres
  - triangles
  - etc.
- complex shapes
  - later in the course

ray-sphere intersection – algebraic

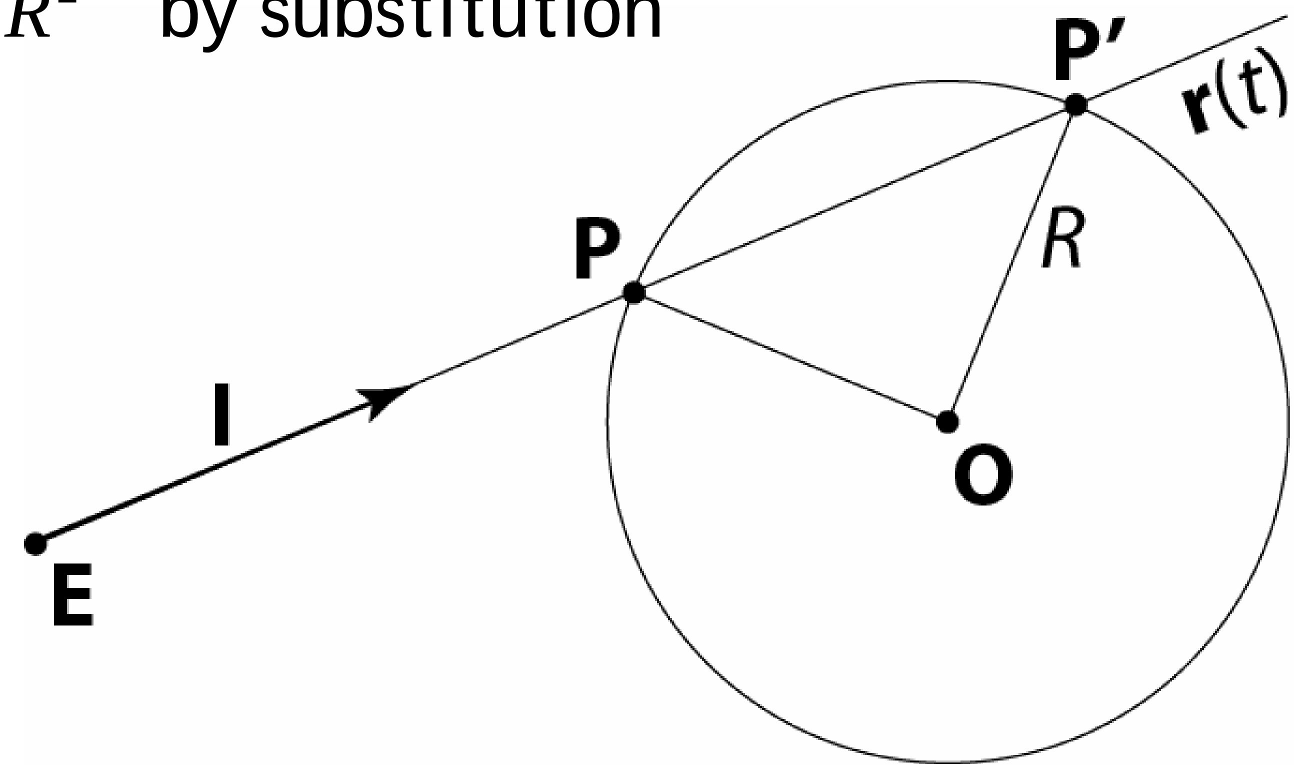
$$\begin{cases} \mathbf{P}(t) = \mathbf{E} + t\mathbf{I} \\ \left|\mathbf{P}(t) - \mathbf{O}\right|^2 = R^2 \end{cases}$$

point on ray

point on sphere

$$\left|\mathbf{E} + t\mathbf{I} - \mathbf{O}\right|^2 = R^2$$

by substitution



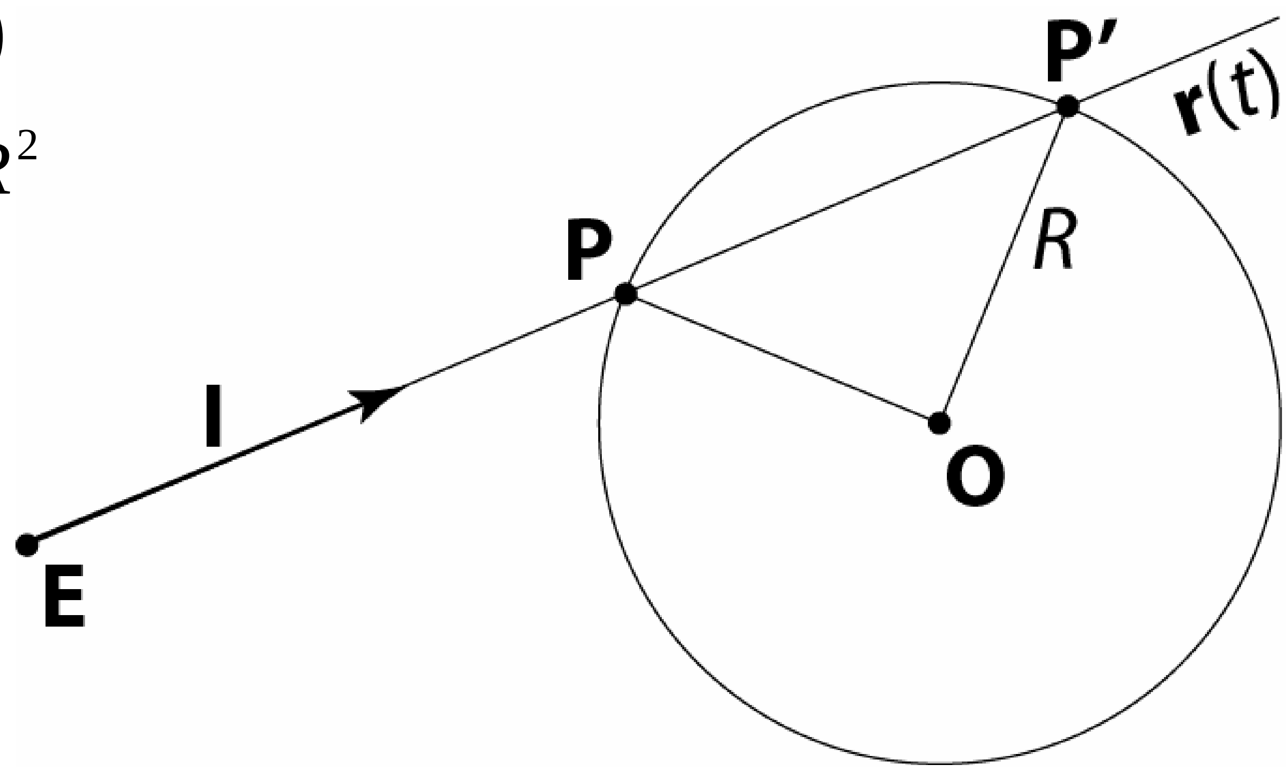
ray-sphere intersection – algebraic

$at^2 + bt + c = 0$ algebraic equation

$a = |\mathbf{I}|^2$

$b = 2\mathbf{I} \cdot (\mathbf{E} - \mathbf{O})$

$c = |\mathbf{E} - \mathbf{O}|^2 - R^2$



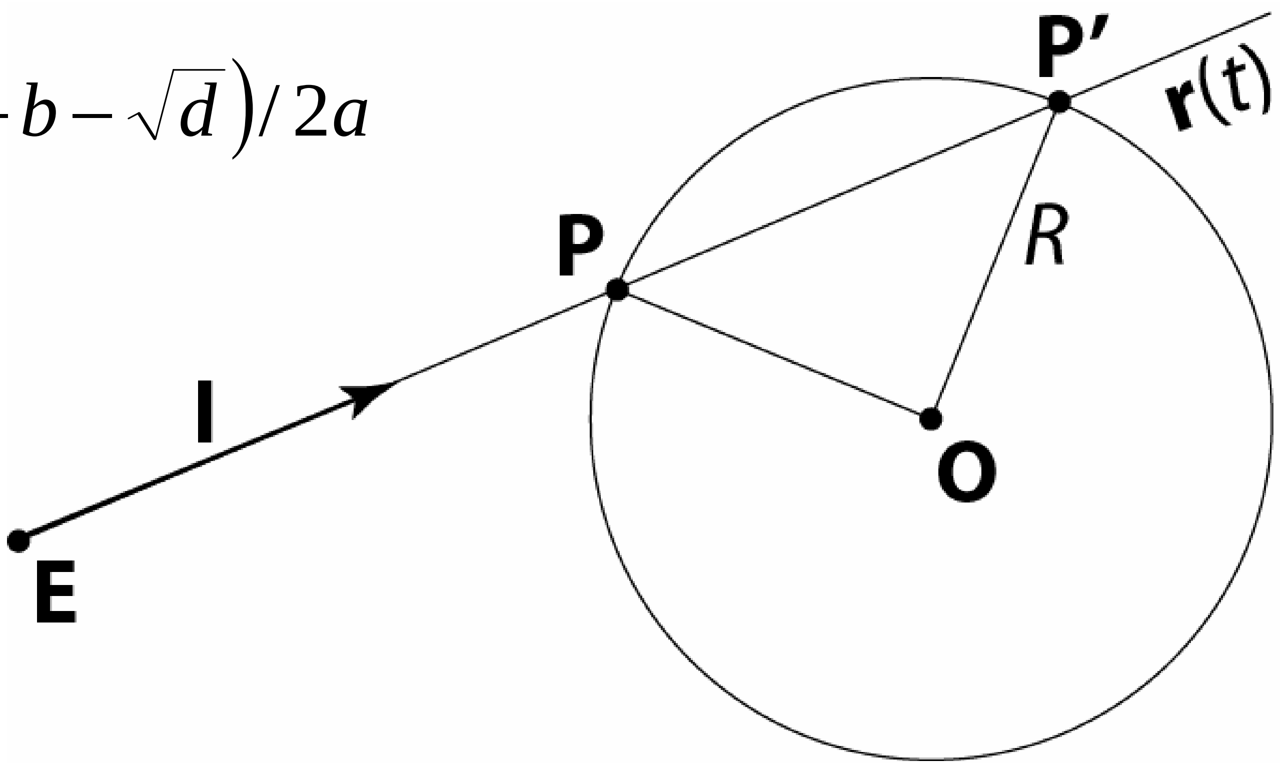
ray-sphere intersection – algebraic

$d = b^2 - 4ac$ determinant

if $d < 0$ no solutions

otherwise

$\mathbf{P}(t) \Leftrightarrow \quad t_- = \left(-b - \sqrt{d}\right)/2a$

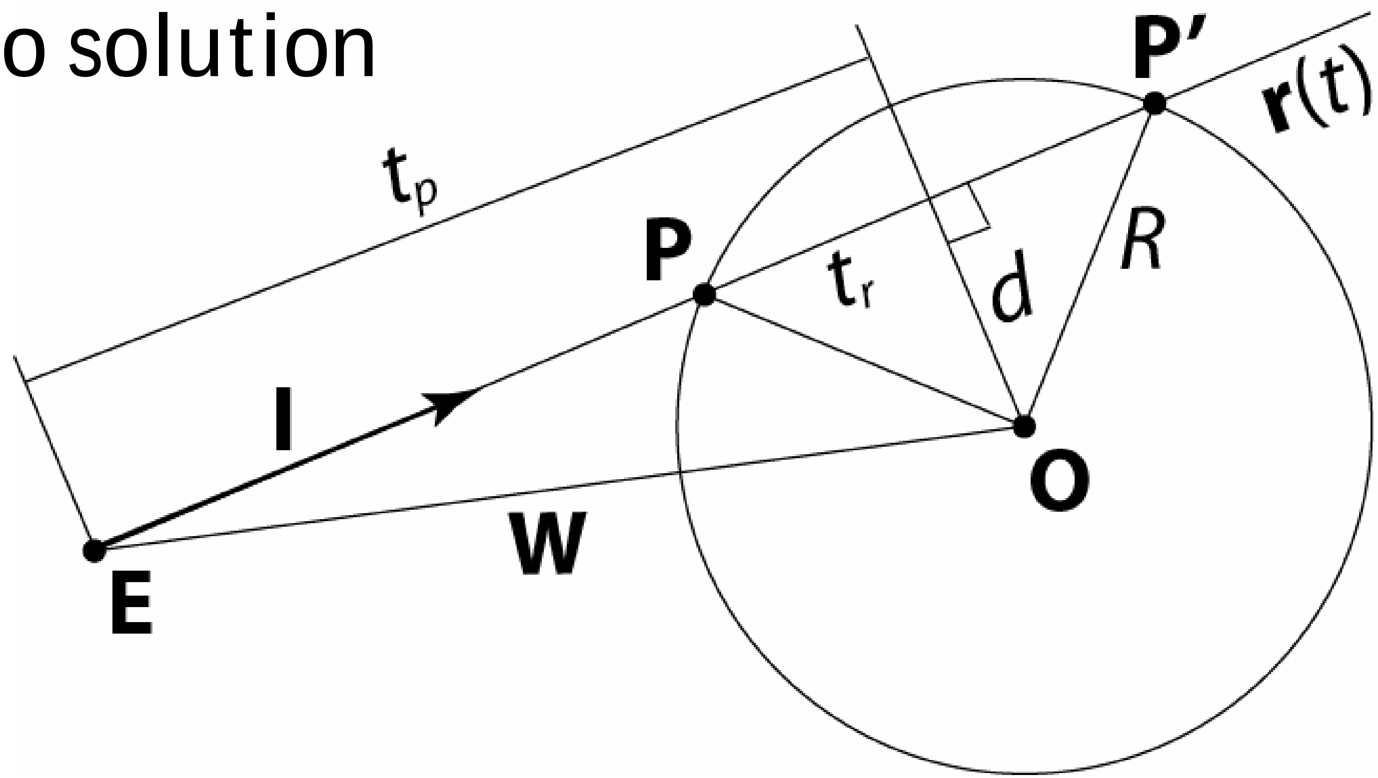


ray-sphere intersection – geometric

$$\mathbf{W} = \mathbf{O} - \mathbf{E}$$

$$t_p = \mathbf{W} \cdot \hat{\mathbf{I}}$$

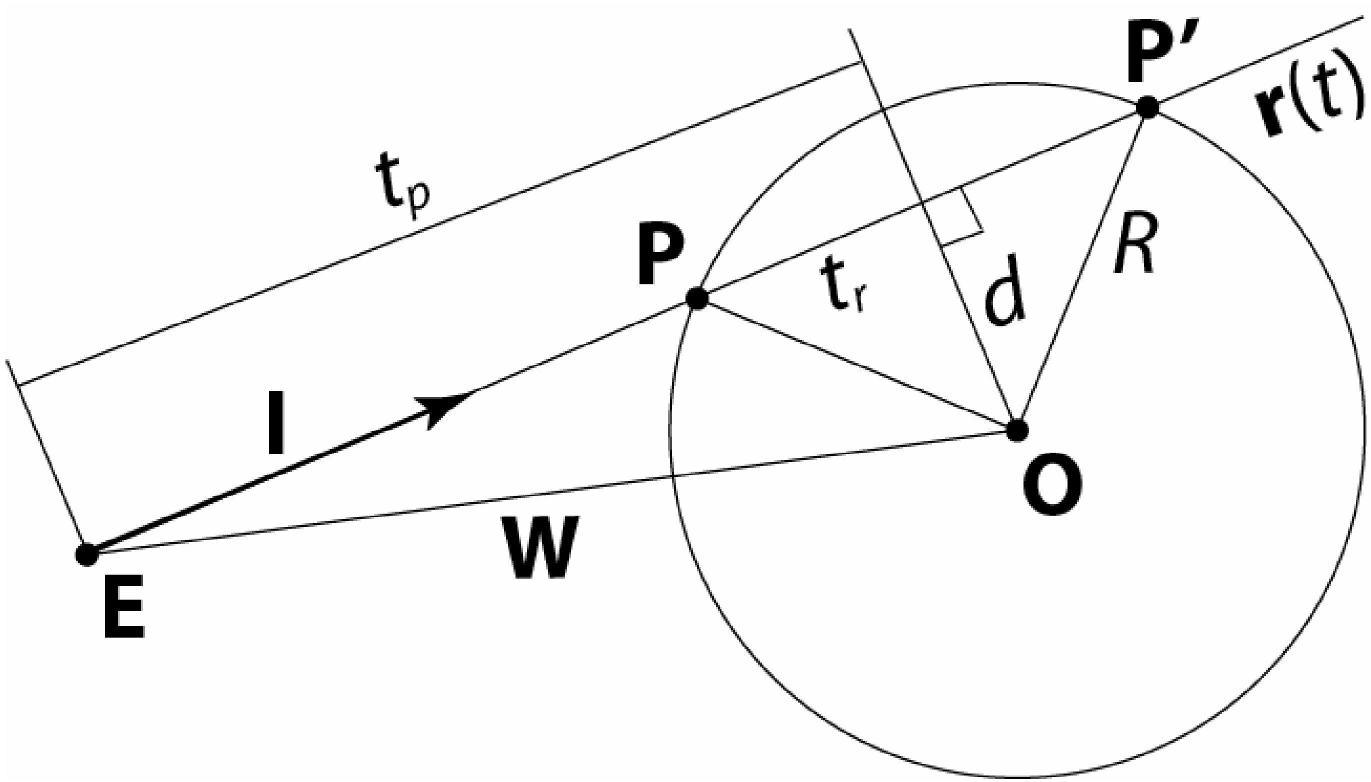
if $t_p < 0$ no solution



ray-sphere intersection – geometric

$$d^2 = \mathbf{W} \cdot \mathbf{W} - t_p^2$$

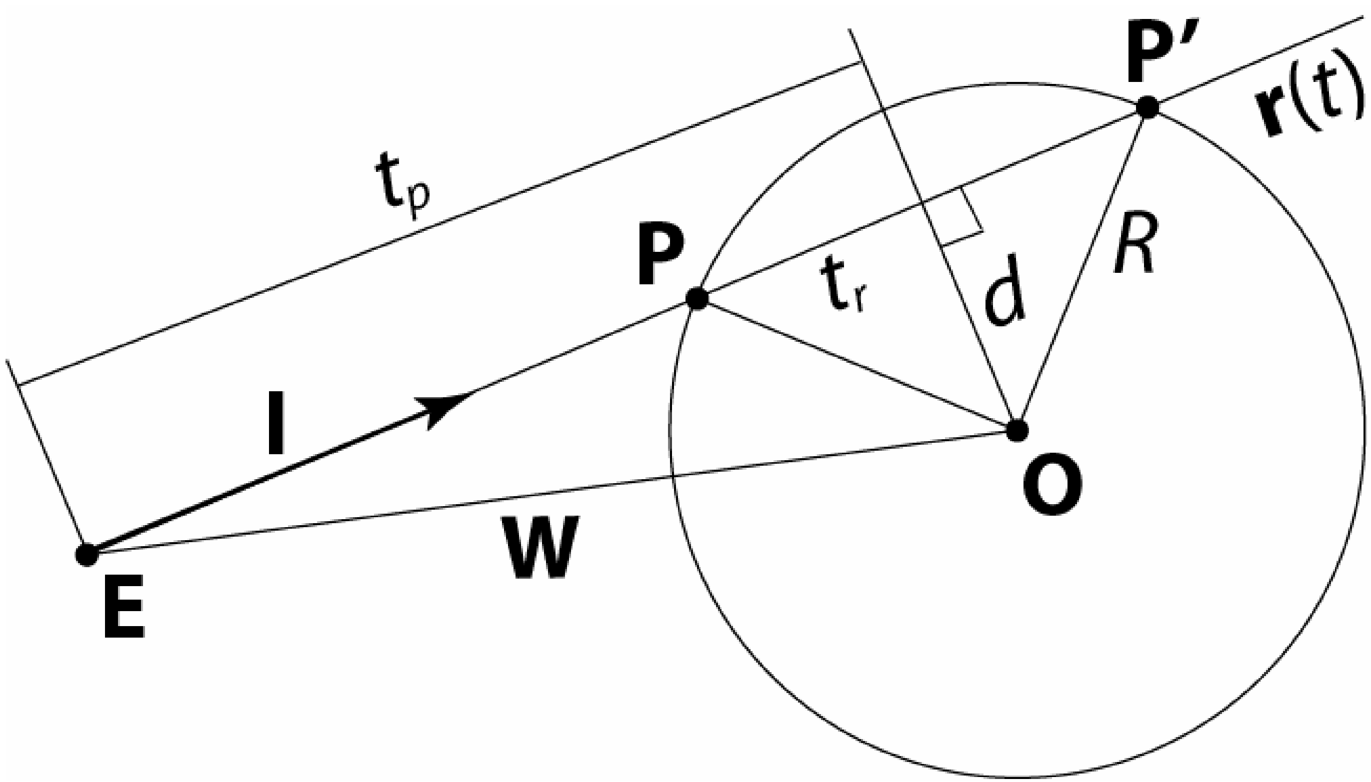
if $d^2 > R^2$ no solution



ray-sphere intersection – geometric

$$t_r = \sqrt{R^2 - d^2}$$

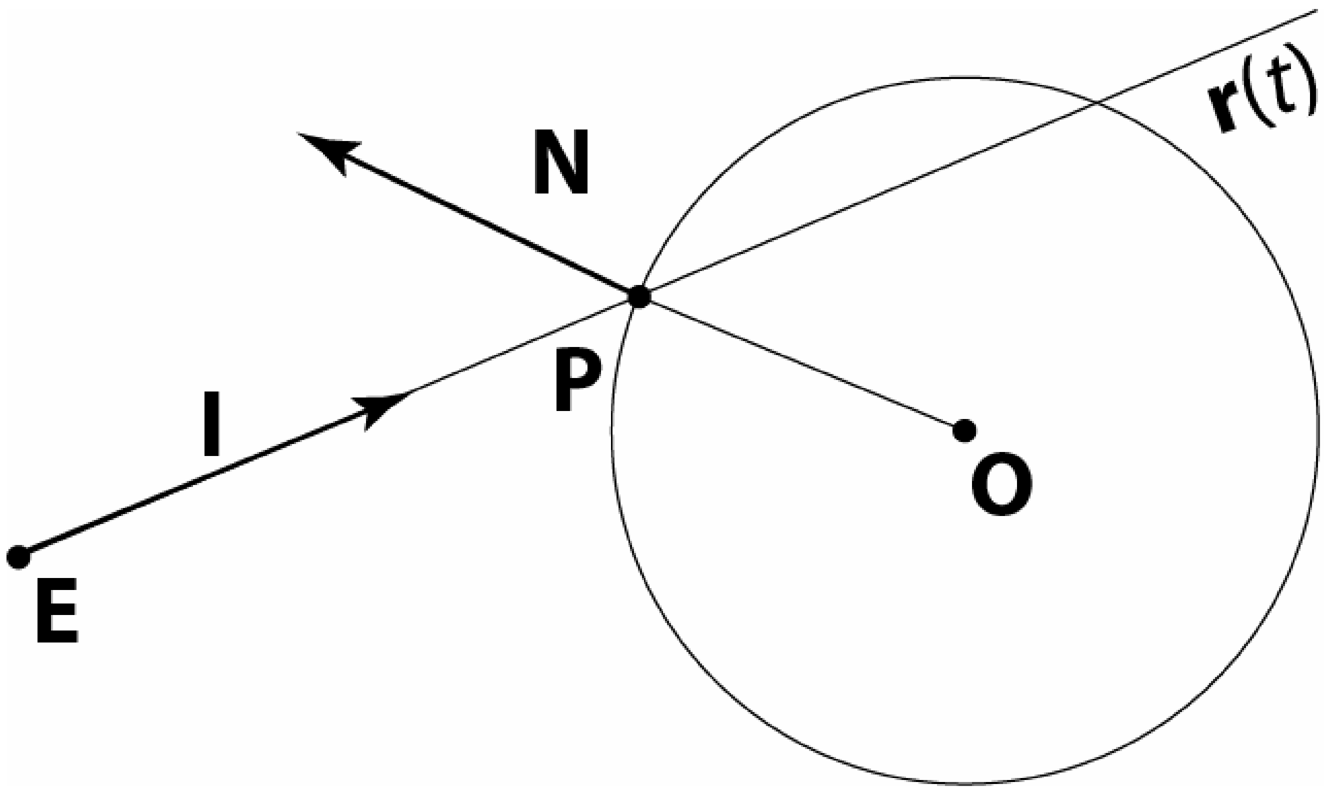
$$\mathbf{P}(t) \Leftrightarrow t_- = t_p - t_r$$



ray-sphere intersection – normal

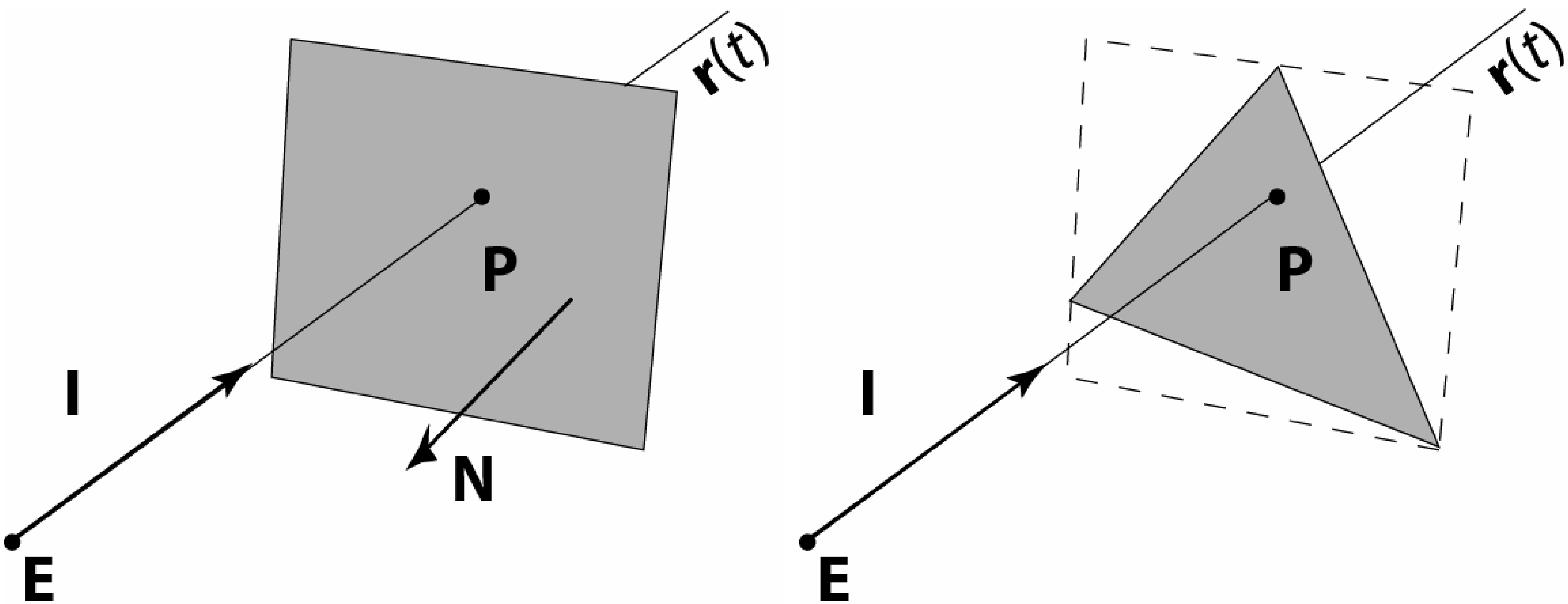
- surface normal need for lighting computation

$$\mathbf{N} = (\mathbf{P} - \mathbf{O}) / |\mathbf{P} - \mathbf{O}|$$



ray-triangle intersection 1

- intersect with a plane
- check if the intersection point is in triangle

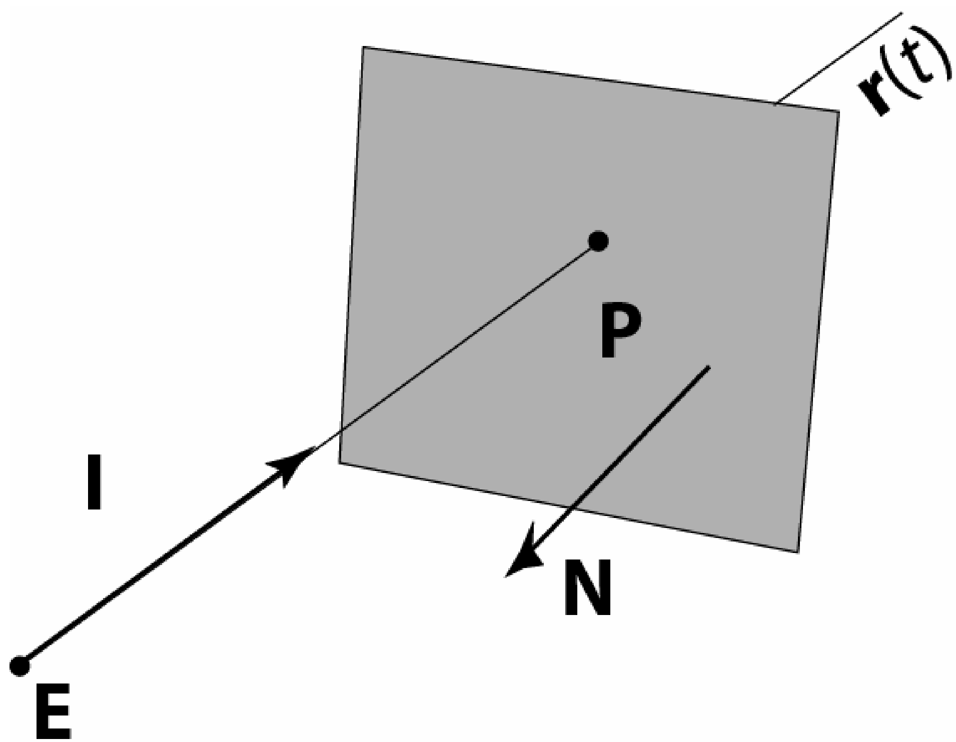


ray-plane intersection

$$\begin{cases} \mathbf{P}(t) = \mathbf{E} + t\mathbf{I} & \text{point on ray} \\ \mathbf{P} \cdot \mathbf{N} + d = 0 & \text{point on plane} \end{cases}$$

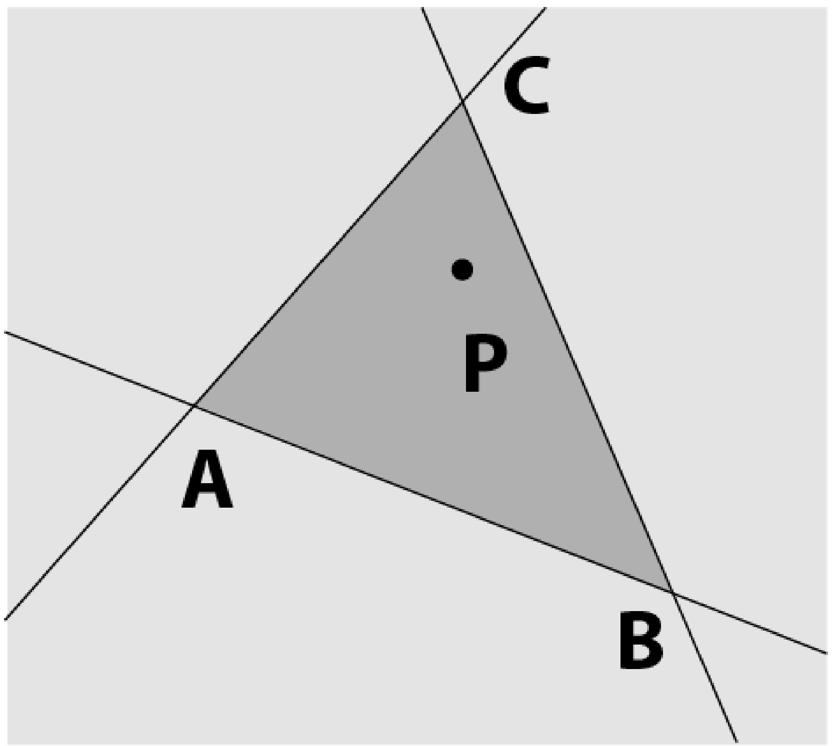
$$(\mathbf{E} + t\mathbf{I}) \cdot \mathbf{N} + d = 0 \quad \text{by substitution}$$

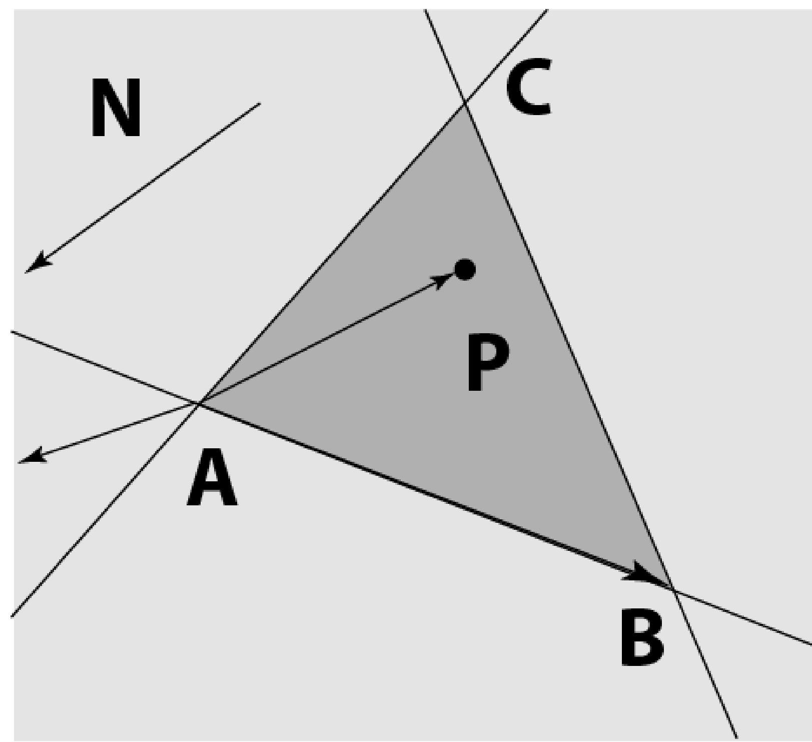
$$\mathbf{P}(t) \Leftrightarrow t = -\frac{\mathbf{E} \cdot \mathbf{N} + d}{\mathbf{I} \cdot \mathbf{N}}$$



ray-triangle intersection 1

- triangle is intersection of 3 half-spaces
- check if P is on right side by comparing “normals”





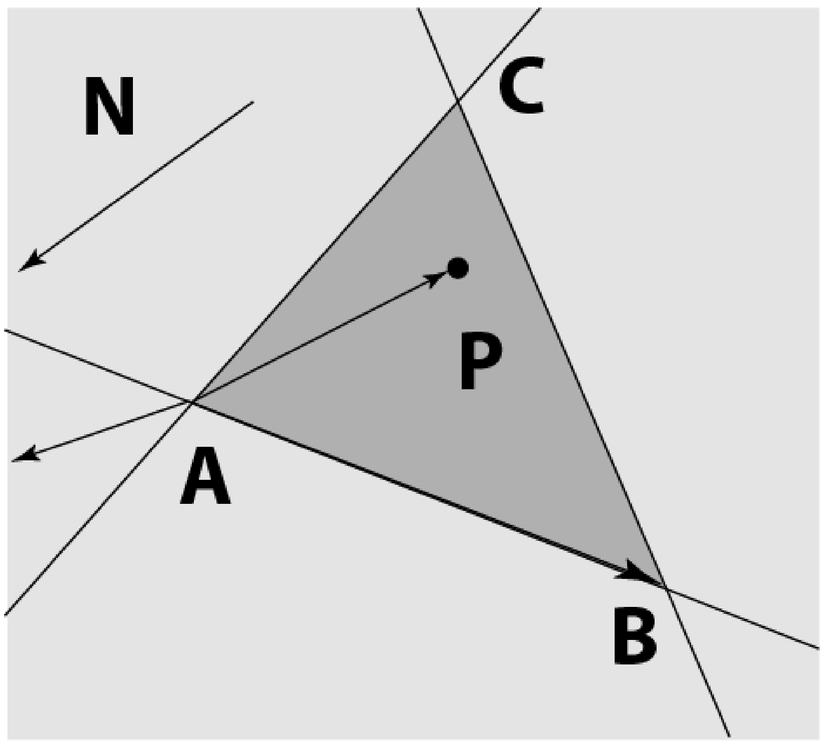
ray-triangle intersection 1

intersects if

$$\left(\mathbf{B}-\mathbf{A}\right)\times\left(\mathbf{P}-\mathbf{A}\right)\cdot\mathbf{N}>0$$

$$\left(\mathbf{C}-\mathbf{B}\right)\times\left(\mathbf{P}-\mathbf{B}\right)\cdot\mathbf{N}>0$$

$$\left(\mathbf{A}-\mathbf{C}\right)\times\left(\mathbf{P}-\mathbf{C}\right)\cdot\mathbf{N}>0$$

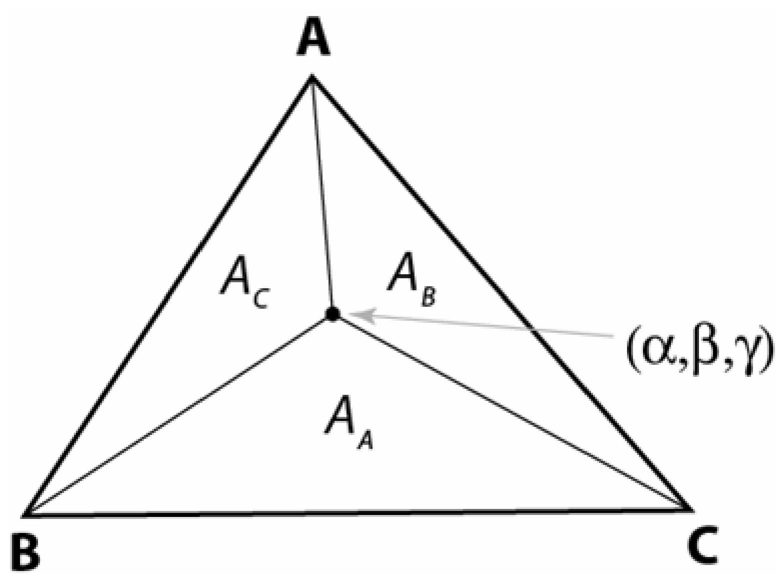


ray-triangle intersection 2

- use baricentric coordinates (useful later)
- algebraic formulation

$$\begin{cases} \mathbf{P}(t) = \mathbf{E} + t\mathbf{I} & \text{point on ray} \\ \mathbf{P}(\alpha,\beta) = \alpha\mathbf{A} + \beta\mathbf{B} + (1-\alpha-\beta)\mathbf{C} & \text{point in the triangle} \end{cases}$$

$$\mathbf{E} + t\mathbf{I} = \alpha\mathbf{A} + \beta\mathbf{B} + (1-\alpha-\beta)\mathbf{C} \qquad \text{by substitution}$$



ray-triangle intersection 2

$$\mathbf{E} + t\mathbf{I} = \alpha(\mathbf{A}-\mathbf{C}) + \beta(\mathbf{B}-\mathbf{C}) + \mathbf{C} \rightarrow$$

$$\alpha(\mathbf{A}-\mathbf{C}) + \beta(\mathbf{B}-\mathbf{C}) - t\mathbf{I} = \mathbf{E}-\mathbf{C} \rightarrow$$

$$\alpha\mathbf{a} + \beta\mathbf{b} - t\mathbf{I} = \mathbf{e} \rightarrow$$

$$\begin{bmatrix} -\mathbf{I} & \mathbf{a} & \mathbf{b} \end{bmatrix} \begin{bmatrix} t \\ \alpha \\ \beta \end{bmatrix} = \mathbf{e}$$

ray-triangle intersection 2

use Cramer's rule

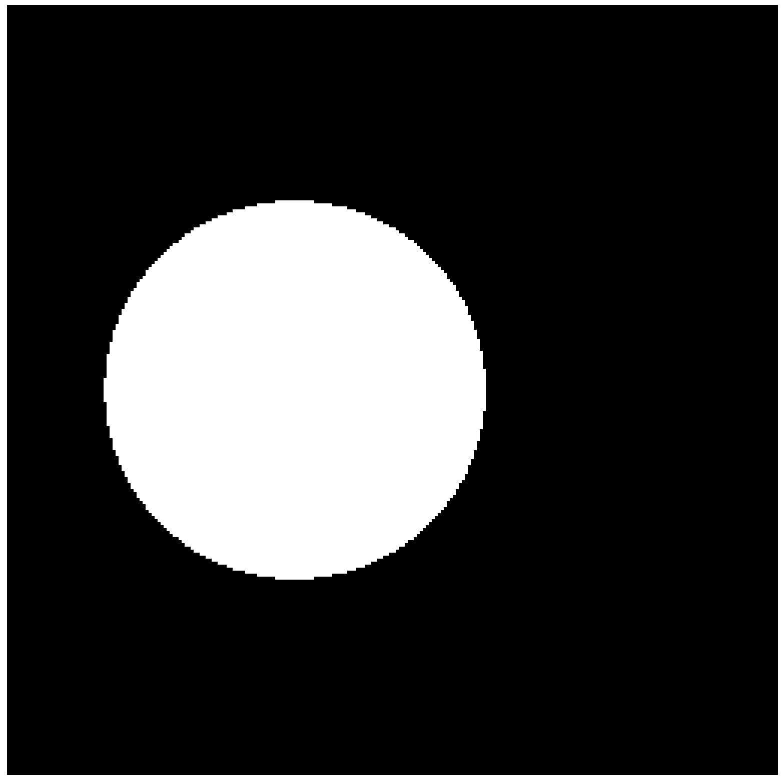
$$t = \frac{|\mathbf{e} \quad \mathbf{a} \quad \mathbf{b}|}{|-\mathbf{I} \quad \mathbf{a} \quad \mathbf{b}|} = \frac{(\mathbf{e} \times \mathbf{a}) \cdot \mathbf{b}}{(\mathbf{I} \times \mathbf{b}) \cdot \mathbf{a}}$$

$$\alpha = \frac{|-\mathbf{I} \quad \mathbf{e} \quad \mathbf{b}|}{|-\mathbf{I} \quad \mathbf{a} \quad \mathbf{b}|} = \frac{(\mathbf{I} \times \mathbf{b}) \cdot \mathbf{e}}{(\mathbf{I} \times \mathbf{b}) \cdot \mathbf{a}}$$

$$\beta = \frac{|-\mathbf{I} \quad \mathbf{a} \quad \mathbf{e}|}{|-\mathbf{I} \quad \mathbf{a} \quad \mathbf{b}|} = \frac{(\mathbf{e} \times \mathbf{a}) \cdot \mathbf{I}}{(\mathbf{I} \times \mathbf{b}) \cdot \mathbf{a}}$$

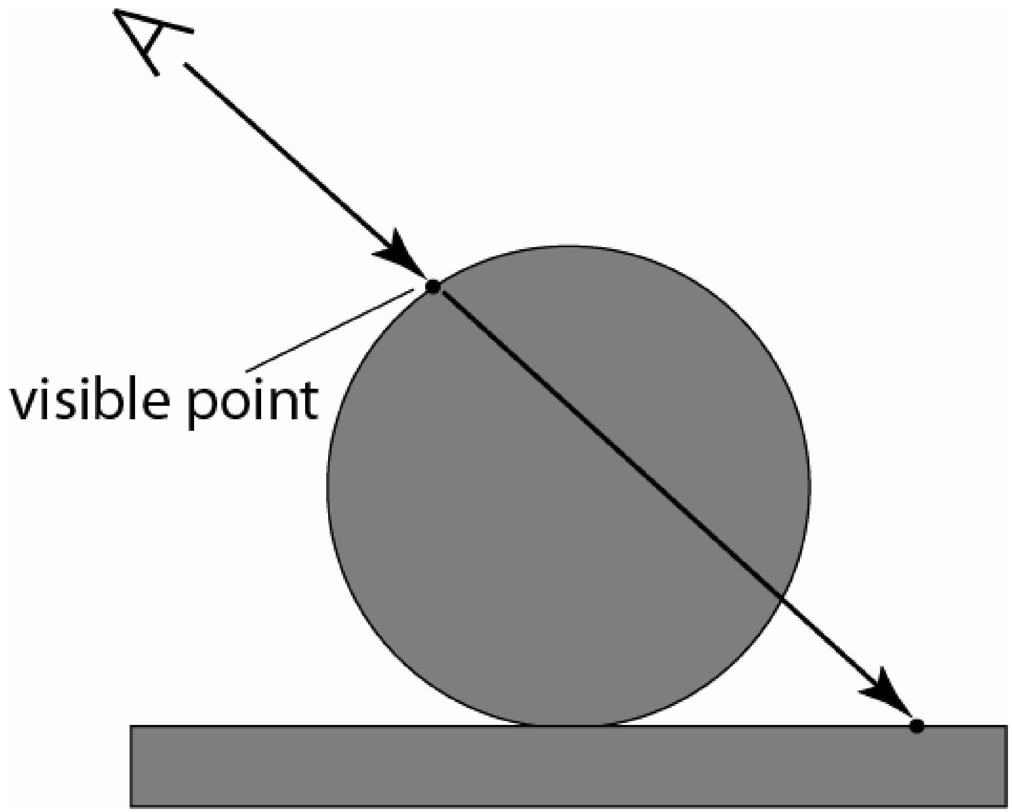
test for $\alpha \geq 0, \beta \geq 0, \alpha + \beta \leq 1$

images so far



intersecting many shapes

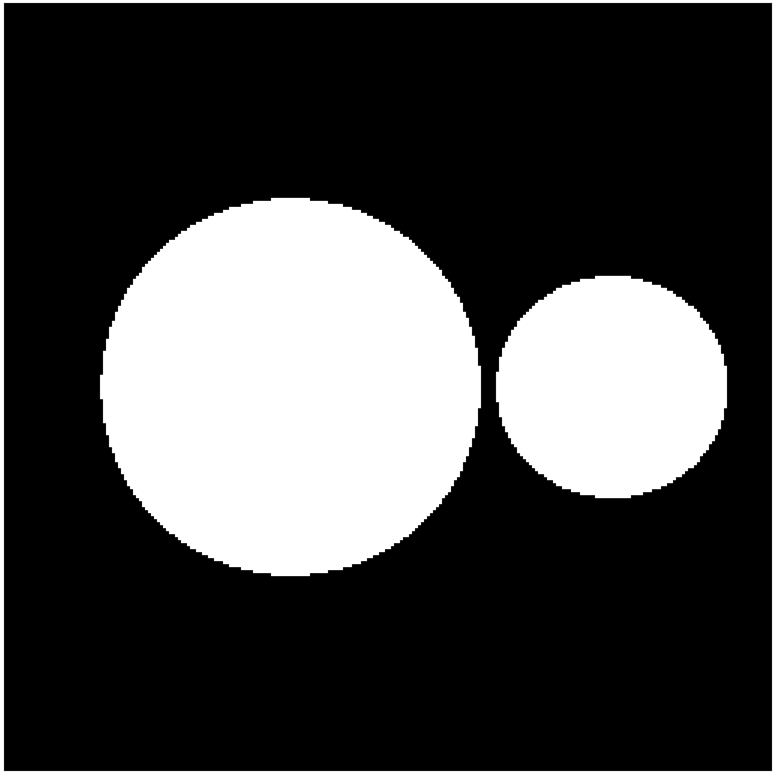
- intersect each primitive
- pick closest intersection



intersecting many shapes - pseudocode

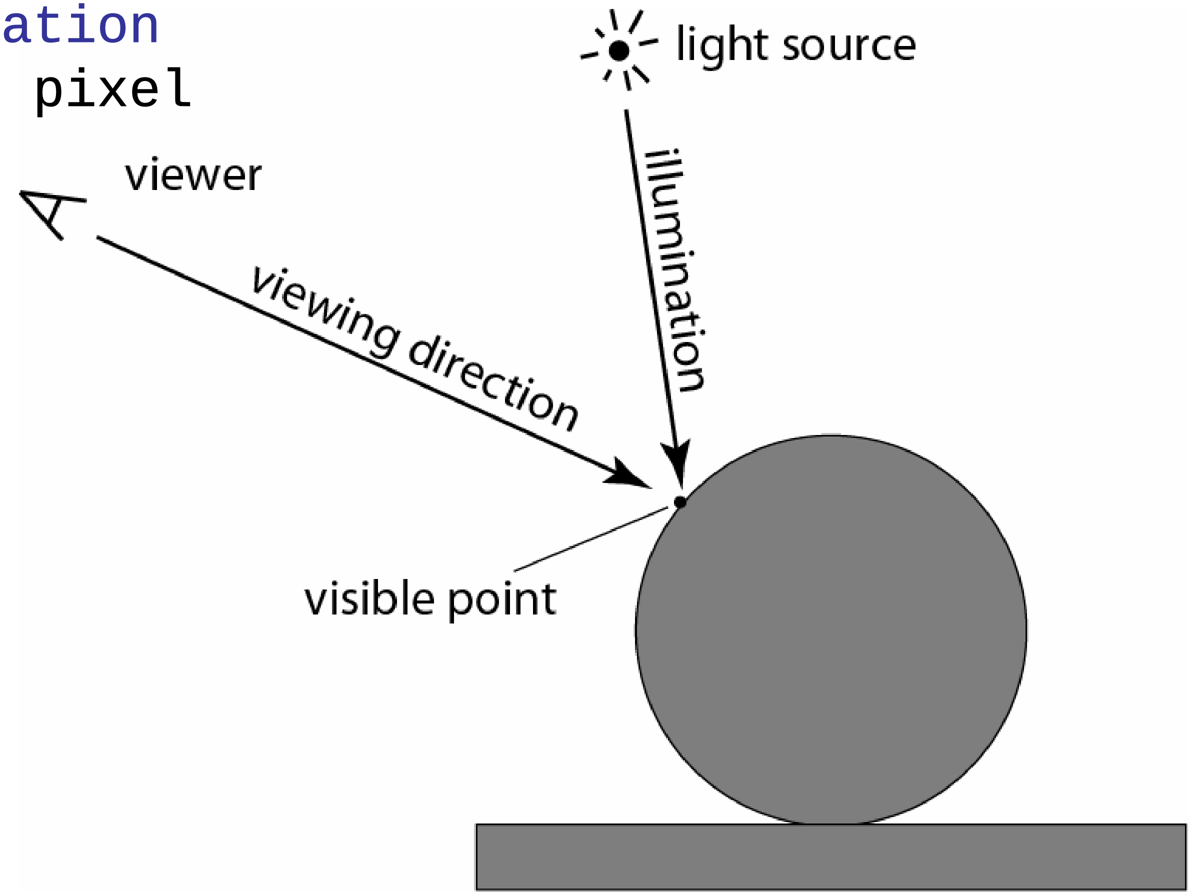
```
minDistance = infinity
hit = false
foreach surface s {
  if(s.intersect(ray,intersection)) {
    if(intersection.distance < minDistance) {
      hit = true;
      minDistance = intersection.distance;
    }
  }
}
```


images so far



ray tracing algorithm

```
for each pixel {
  determine viewing direction
  intersect ray with scene
  compute illumination
  store result in pixel
}
```



shading

variation in observed color across a surface

lighting

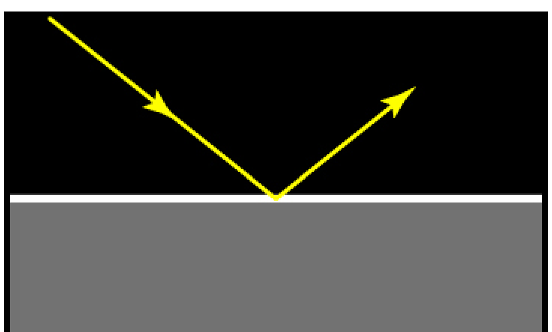
patterns of illumination in the environment

shading

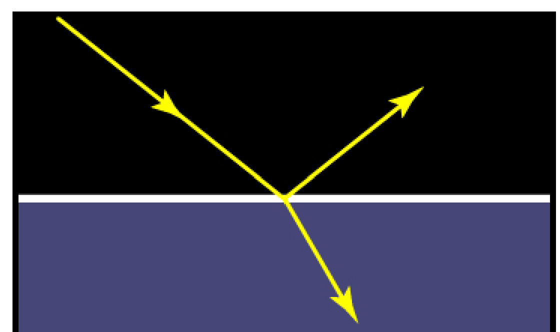
- compute reflected light
- depends on
  - viewer position
  - incoming light, i.e. lighting
  - surface geometry
  - surface material
- more on this later

materials

metals



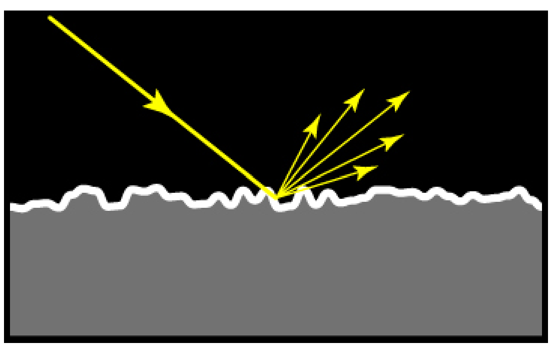
dielectric



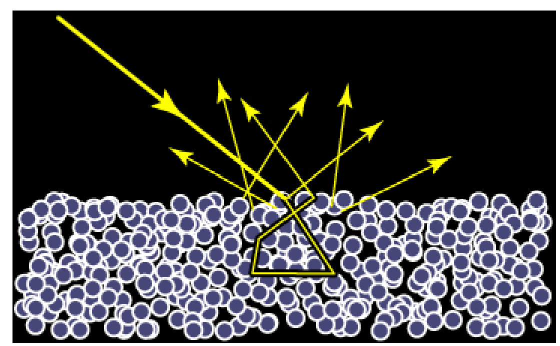
[Marschner 2004]

materials

metals



dielectric



[Marschner 2004]

shading models

- empirical shading models
  - produce believable images
  - simple and efficient
  - only for simple materials
- physically-based shading models
  - can reproduce accurate effects
  - more complex
- will concentrate empirical plastic-like model
  - more on this later in the course

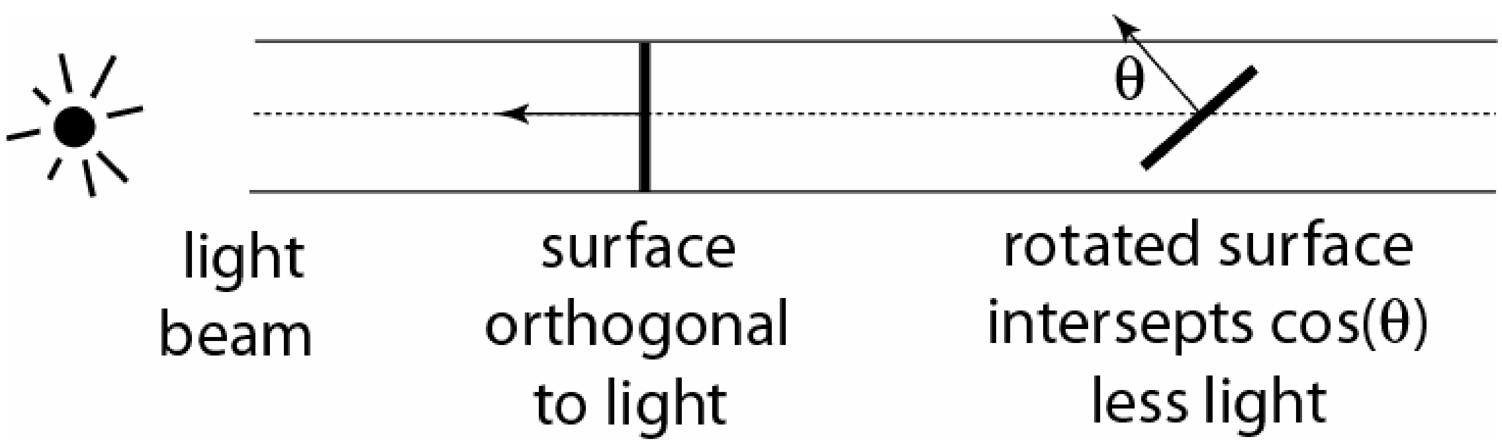
Phong shading model

- shading model = diffuse + specular reflection

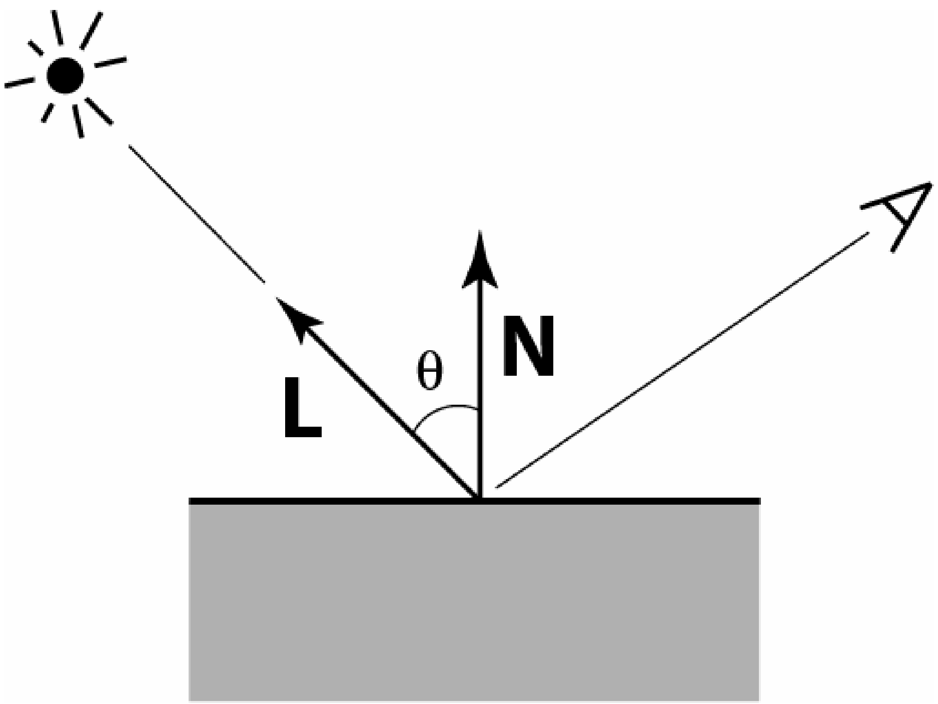
- diffuse reflection
  - light is reflected in every direction equally
  - colored by surface color
- specular reflection
  - light is reflected only around the mirror direction
  - white for plastic-like surfaces (glossy paints)
  - colored for metals (brass, copper, gold)

diffuse reflection

- light reflects equally in all directions
  - i.e. surface looks the same from all view points
  - view-independent
- but incident light depends on angle
  - beam of light is more spread on an oblique surface



lambertian shading model



diffuse coefficient

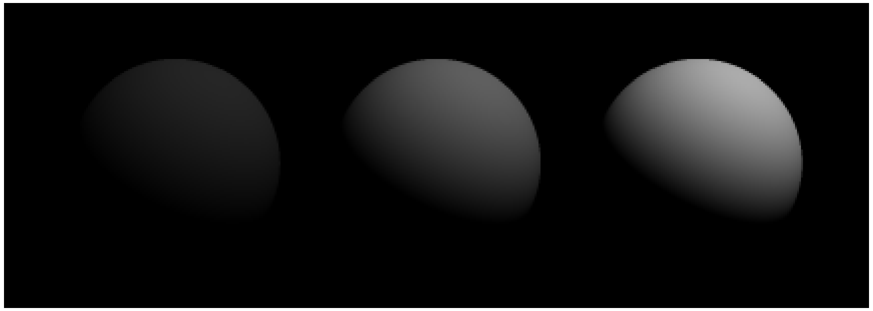
$$C_d = k_d C_l \max(0, \mathbf{N} \cdot \mathbf{L})$$

reflected diffuse illumination

illumination from light

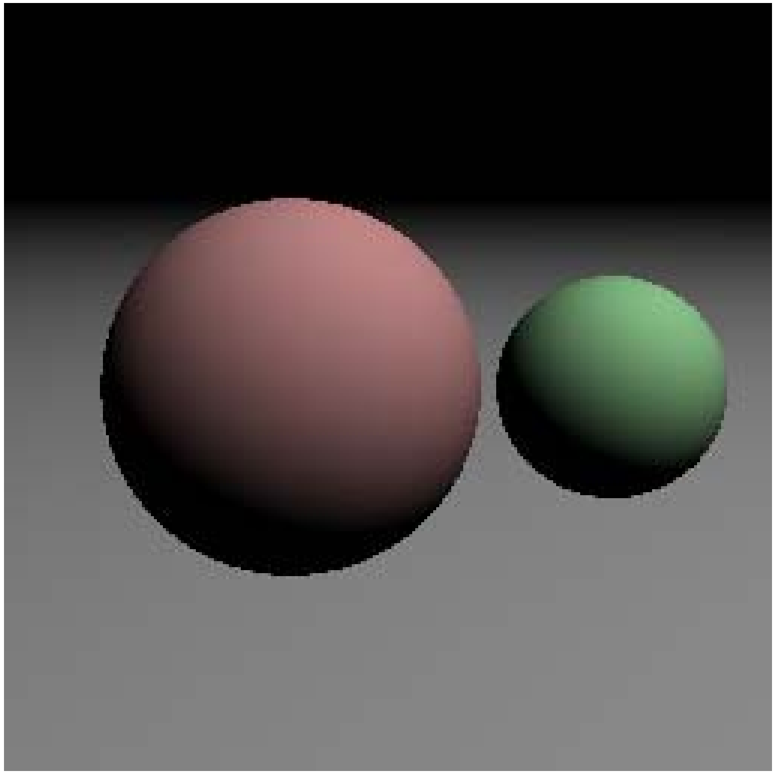
lambertian shading model

- produces matte appearance



k_d

images so far

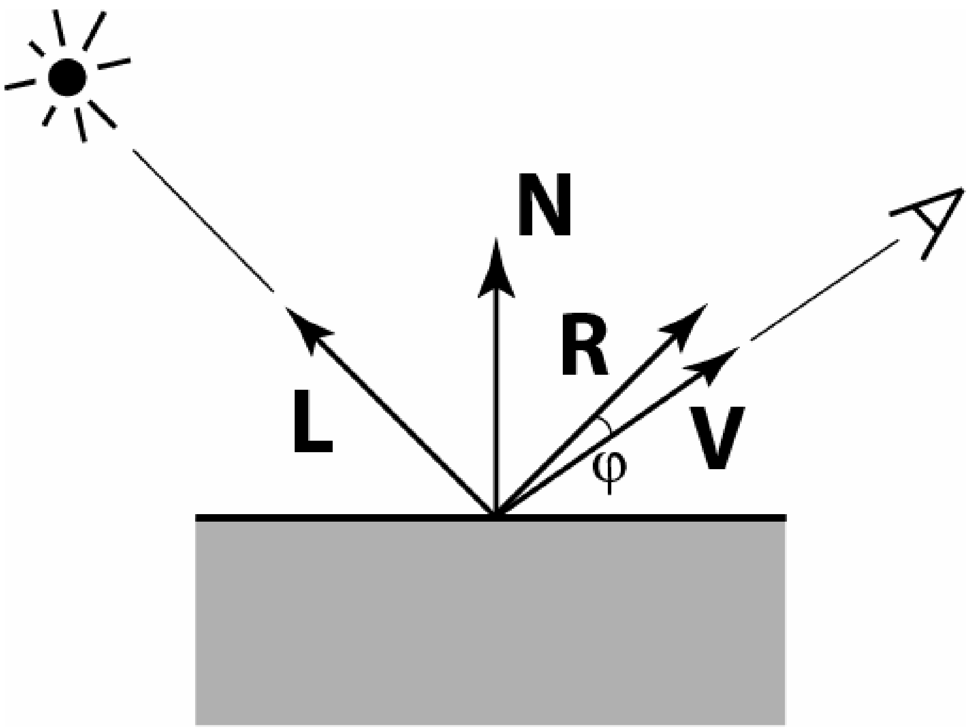


specular shading

- light reflects mostly around mirror direction
  - i.e. surface looks different from view points
  - view-dependent

- Phong specular model
  - empirical, but looks good enough
  - use cosine of mirror and view direction
- Blinn-Phong specular model
  - slightly better than Phong
  - use cosine of bisector and normal direction

Phong specular model



mirror direction

$$\mathbf{R} = -\mathbf{L} + 2(\mathbf{N}\cdot\mathbf{L})\mathbf{N}$$

$$C_s = k_s C_l \max(0, \mathbf{V}\cdot\mathbf{R})^n$$

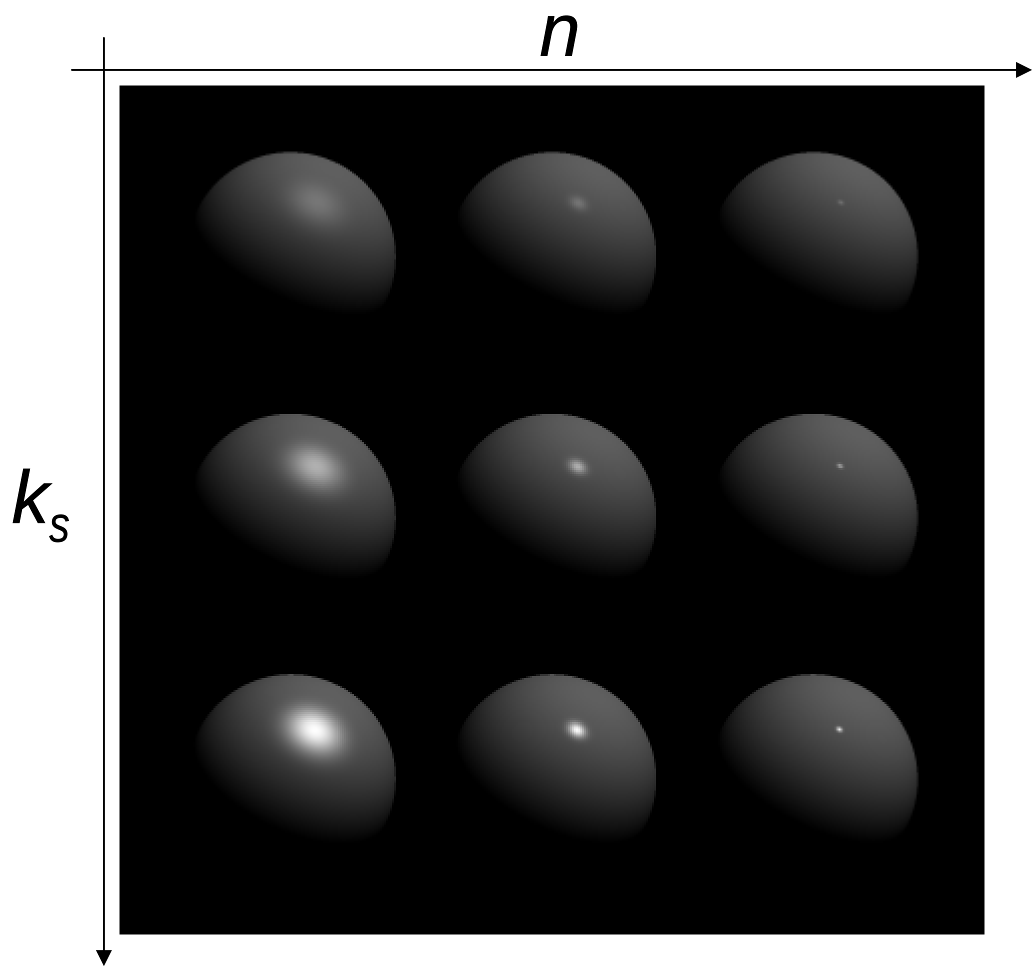
reflected specular illumination

specular coefficient

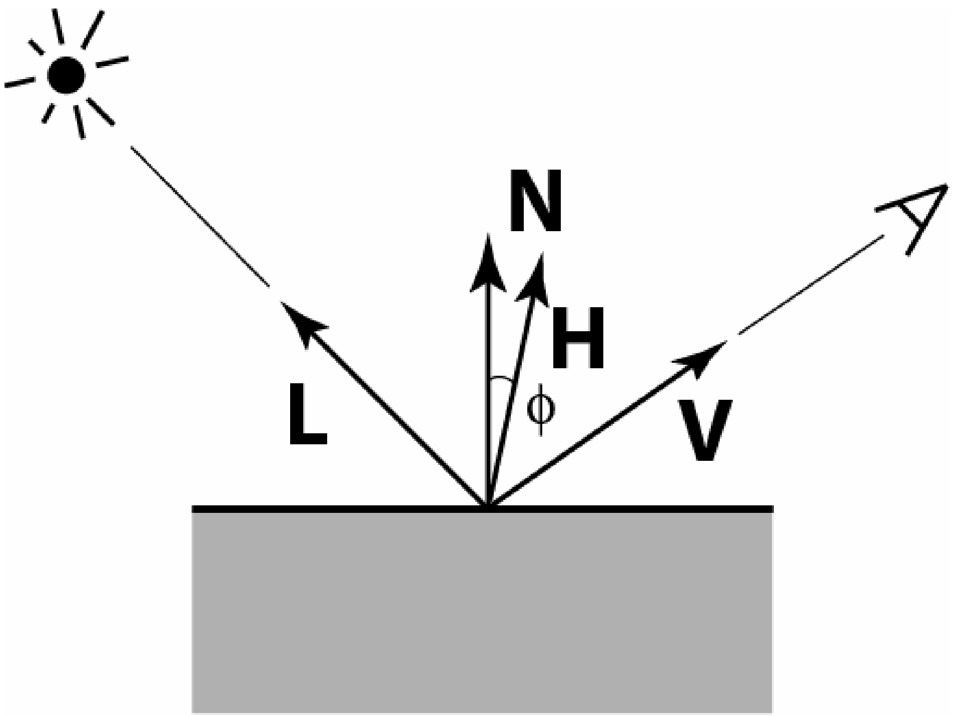
specular exponent

Phong specular model

- produces highlights, shiny appearance



Blinn-Phong variation



bisector direction

$$\mathbf{H} = (\mathbf{L} + \mathbf{V}) / |\mathbf{L} + \mathbf{V}|$$

$$C_s = k_s C_l \max(0, \mathbf{N} \cdot \mathbf{H})^n$$

reflected specular illumination

specular coefficient

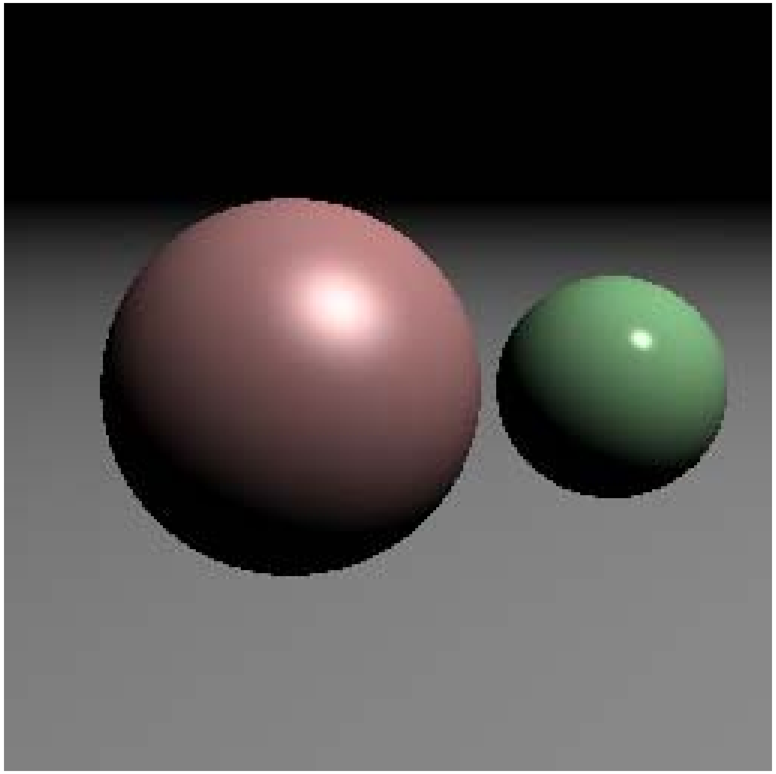
specular exponent

shading model

- putting the pieces together

$$C = C_d + C_s =$$
$$= C_l \left[k_d \max(0, \mathbf{N} \cdot \mathbf{L}) + k_s \max(0, \mathbf{V} \cdot \mathbf{R})^n \right]$$

images so far



lighting

- determines how much light reaches a point
- depends on
  - light geometry
  - light emission
  - scene geometry

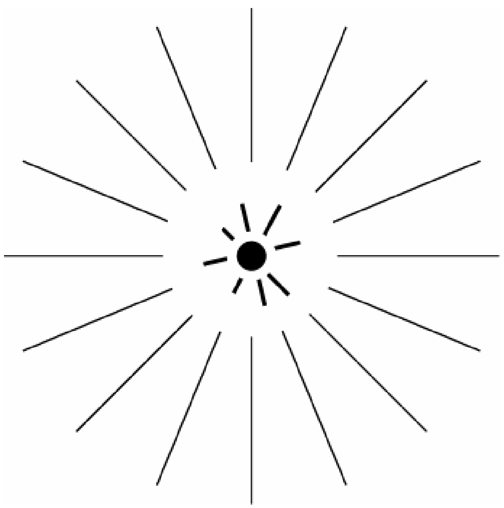
light source models

- describe how light is emitted from light sources

- empirical light source models
  - point, directional, spot
- physically-based light source models
  - area lights, sky model
  - will cover later in the course

point lights

- light emitted equally from point in all directions
  - simulates local lights
  - sometimes r^2 falloff for a bit more realism

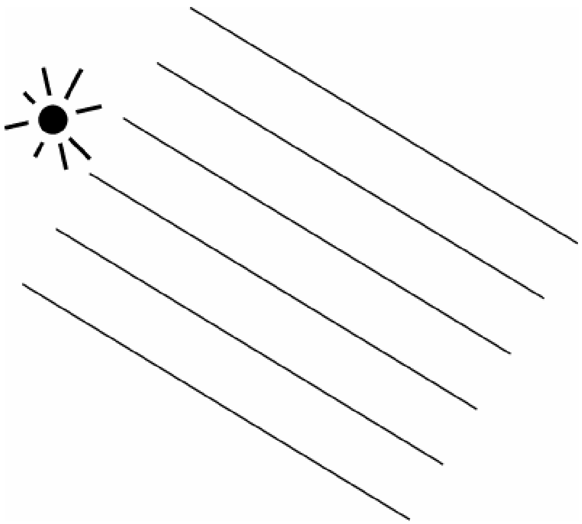


$$\mathbf{L} = \| \mathbf{S} - \mathbf{P} \|$$

$$C_l = C / r^2$$

directional lights

- light emitted from infinity in one direction
  - simulates distant lights, e.g. sun

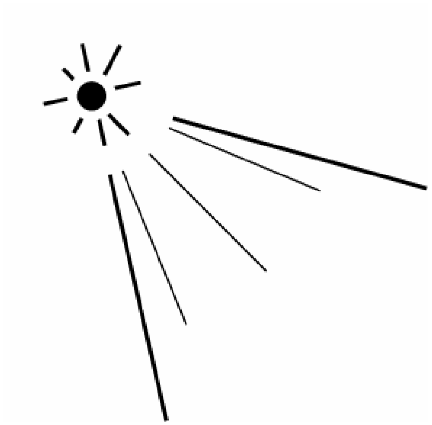


$$\mathbf{L} = \mathbf{D}$$

$$C_l = C$$

spot lights

- same as point light, but only emit in a cone
  - simulate theatrical lights
  - cone falloff model arbitrary



$$\mathbf{L} = \| \mathbf{S} - \mathbf{P} \|$$

$$C_l = C \cdot att / r^2$$

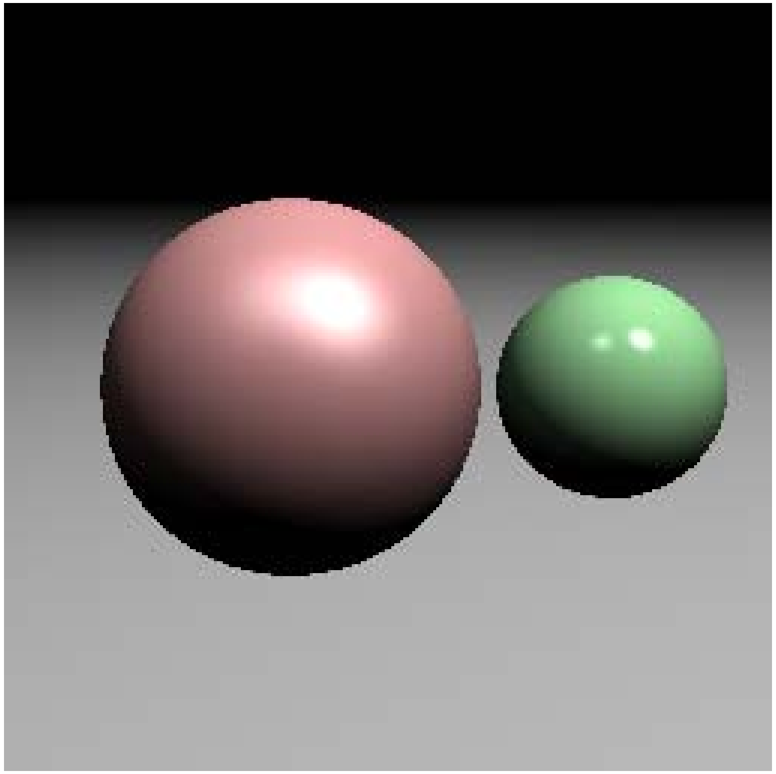
multiple lights

- add contribution for each light

$$C = \sum_i (C_d)_i + (C_s)_i =$$

$$= \sum_i (C_l)_i \left[k_d \max(0, \mathbf{N} \cdot (\mathbf{L})_i) + k_s \max(0, \mathbf{V} \cdot (\mathbf{R})_i)^n \right]$$

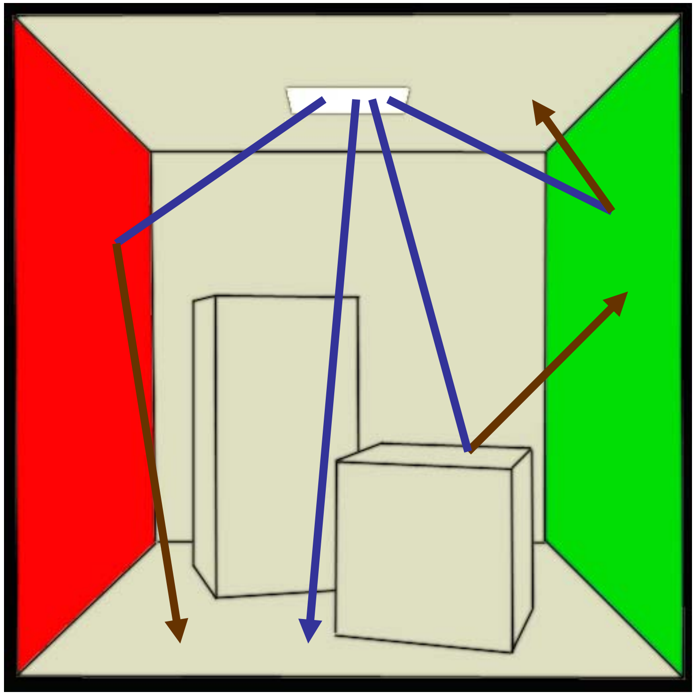
image so far

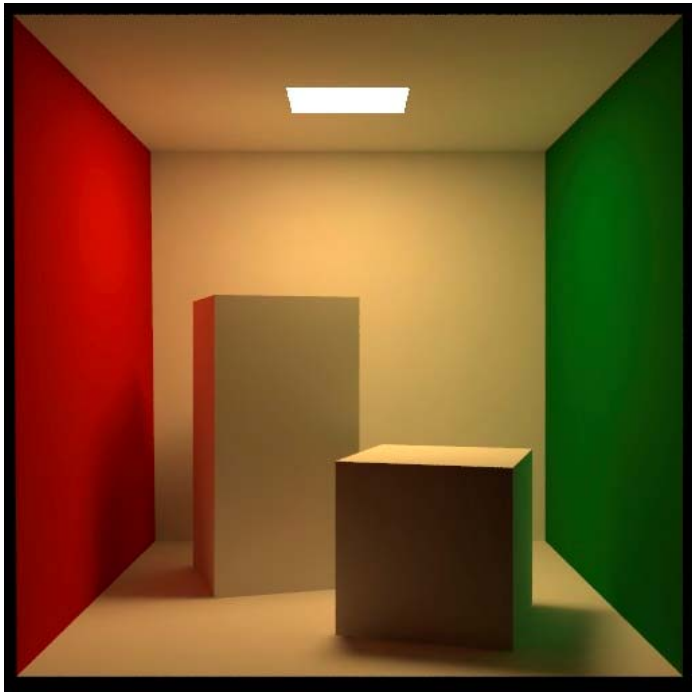


illumination models

- describe how light spreads in the environments
- direct illumination
  - incoming lights comes directly from sources
    - shadows
- indirect illumination
  - incoming lights comes from other objects
    - specular reflections (mirrors), diffuse inter-reflections

illumination models





[PCG]

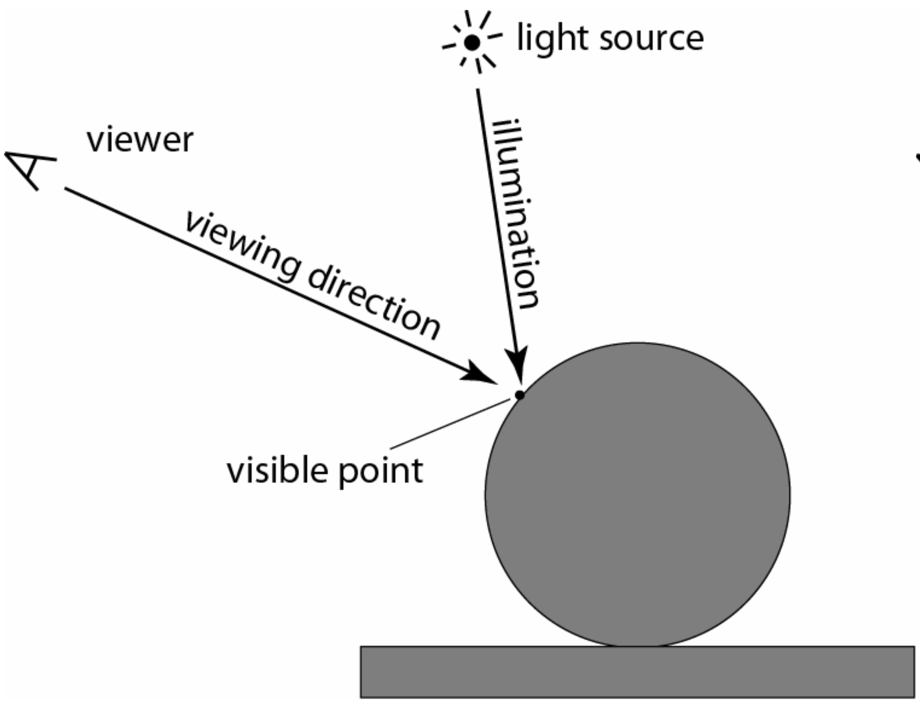
ray tracing lighting model

- captures
  - point/directional/spot light source models
  - sharp shadows
  - sharp reflections/refractions
  - hacked diffuse inter-reflections: ambient term
- more accurate lighting later in the course

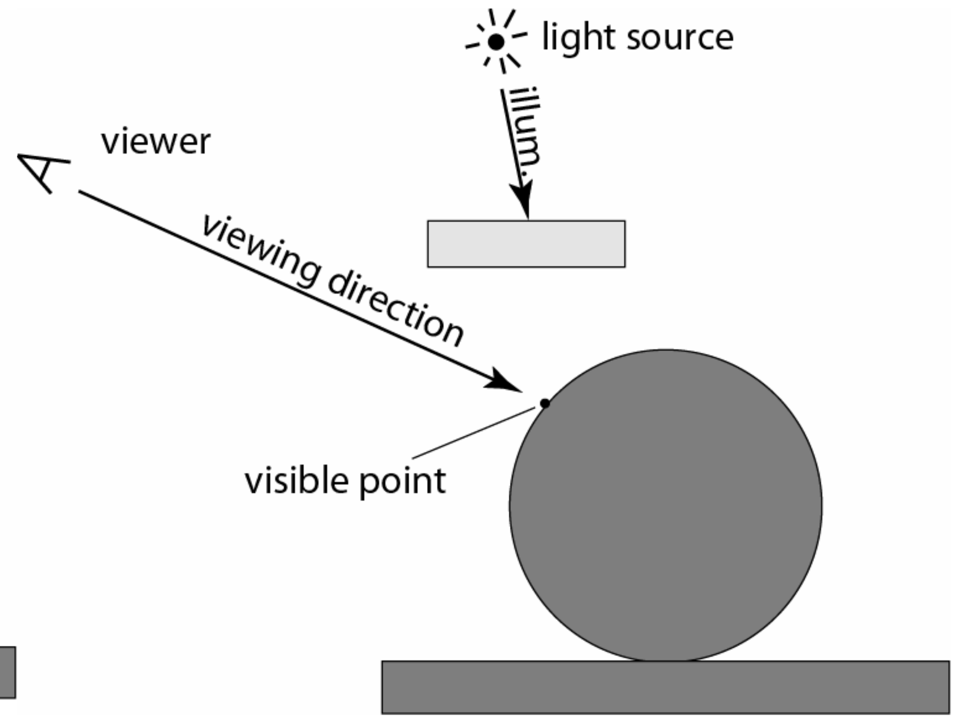
ray traced shadows

- light contributes only if visible at surface point

no shadow

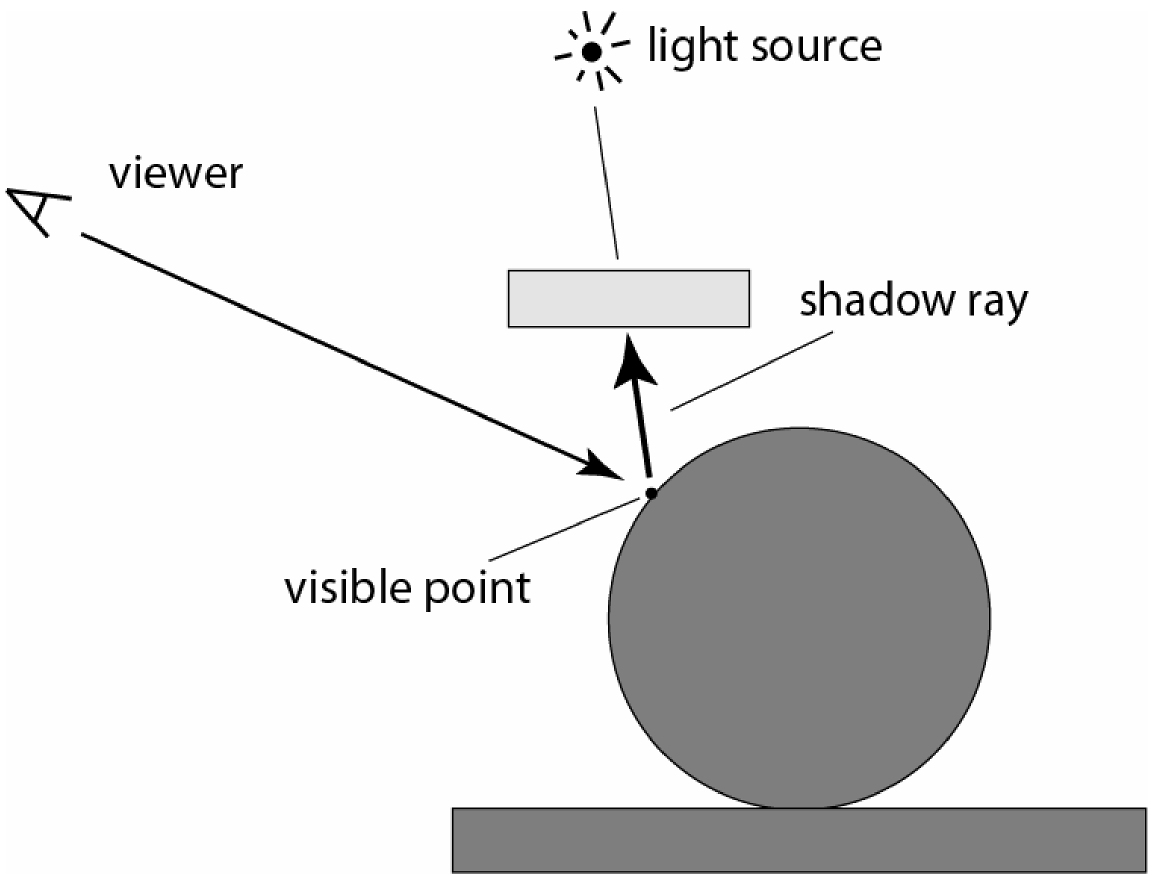


shadow



ray traced shadows

- cast a “shadow-ray” to check if light is visible
- visible if no-hits or if distance > light distance



ray traced shadows

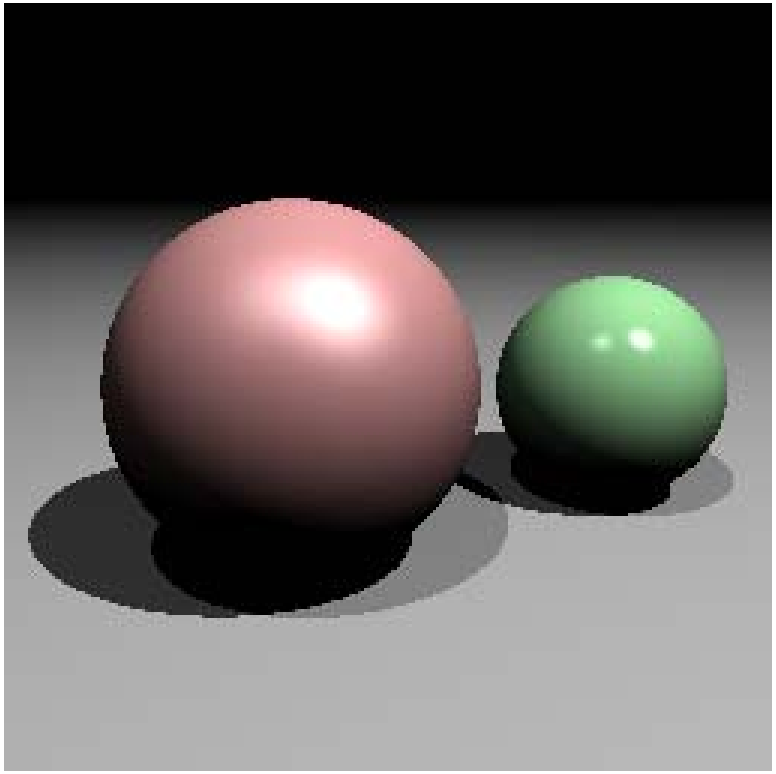
- scale light by a visibility term

$$C = \sum_i (C_d)_i + (C_s)_i =$$

$$= \sum_i (C_l)_i V_i(\mathbf{P}) \left[k_d \max(0, \mathbf{N} \cdot (\mathbf{L})_i) + k_s \max(0, \mathbf{V} \cdot (\mathbf{R})_i)^n \right]$$

light source
visibility {0,1}

images so far



ambient term hack

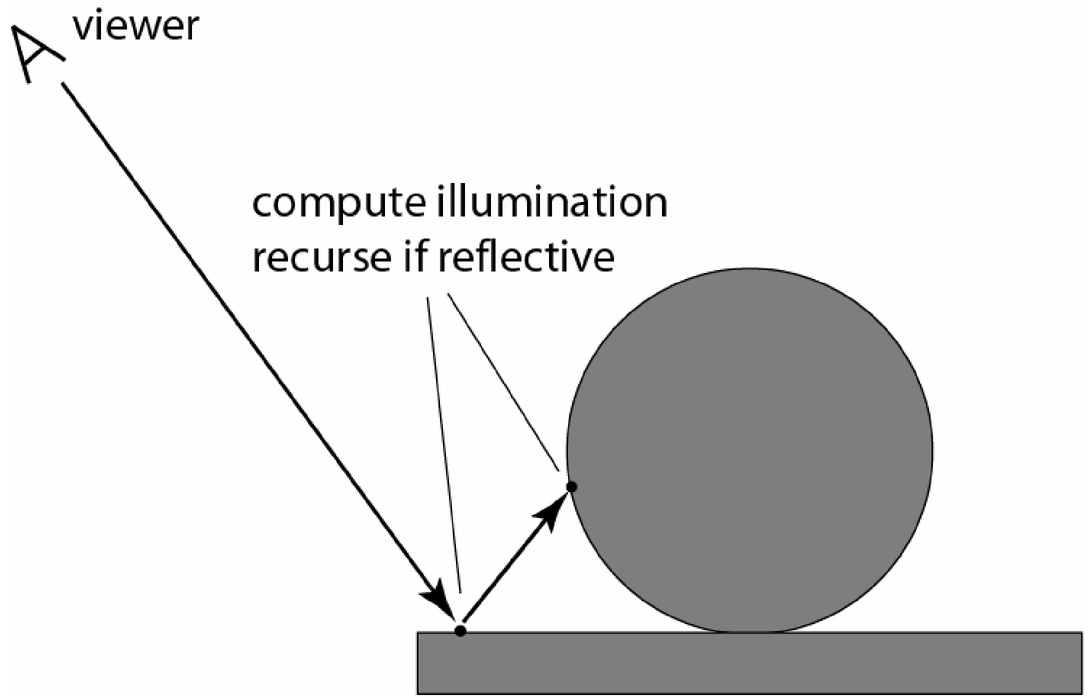
- light bounces even in diffuse environment
  - ceiling are not black
  - shadows are not perfectly black
- very expensive to compute
- approximate (poorly) with a constant term

$$C = C_a + \sum_i \left[(C_d)_i + (C_s)_i \right] =$$

$$= \underbrace{k_a}_{\text{ambient coefficient}} \underbrace{C_{amb}}_{\text{ambient illumination}} + \sum_i \left[(C_d)_i + (C_s)_i \right]$$

ray traced reflections

- perfectly shiny surfaces reflects objects
  - recursively trace a ray if material is reflective
    - along mirror direction, scaled by reflection coeff.
    - mirror direction calculated for Phong



ray traced reflections

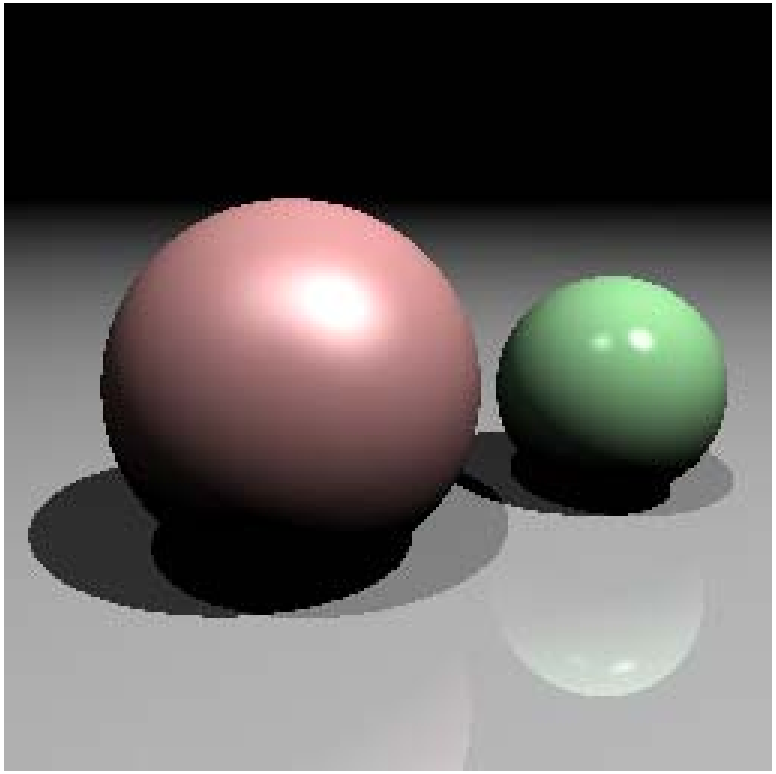
- recursively ray trace, scale by reflection coeff.
- refraction is the same along the transmission dir.

$$C = C_a + \sum_i \left[(C_d)_i + (C_s)_i \right] + C_r + C_t =$$

$$= C_a + \sum_i \left[(C_d)_i + (C_s)_i \right] +$$

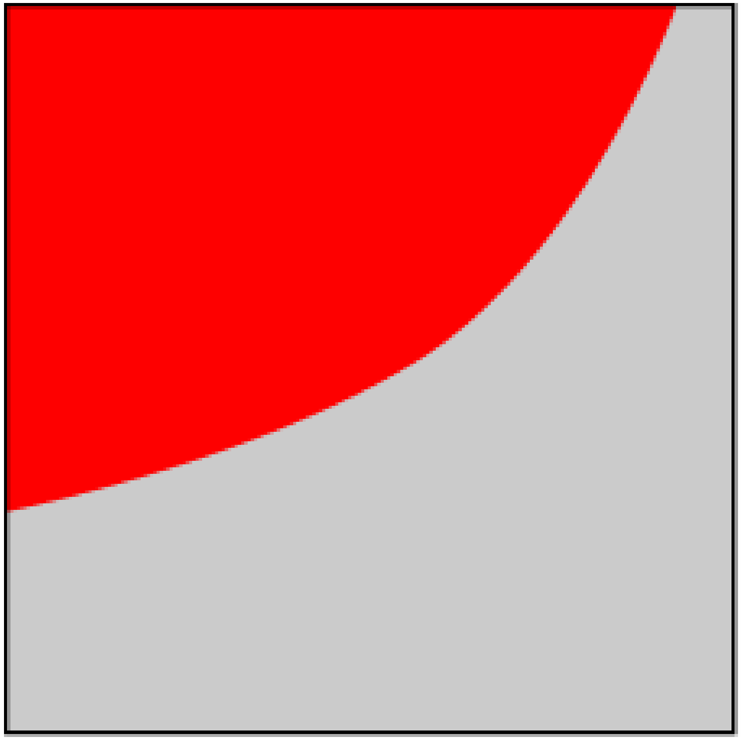
$$+ \underbrace{k_r}_{\text{reflection coefficient}} \,\mathrm{raytrace}(\mathbf{P}, \mathbf{R}) + \underbrace{k_t}_{\text{refraction coefficient}} \,\mathrm{raytrace}(\mathbf{P}, \mathbf{T})$$

images so far



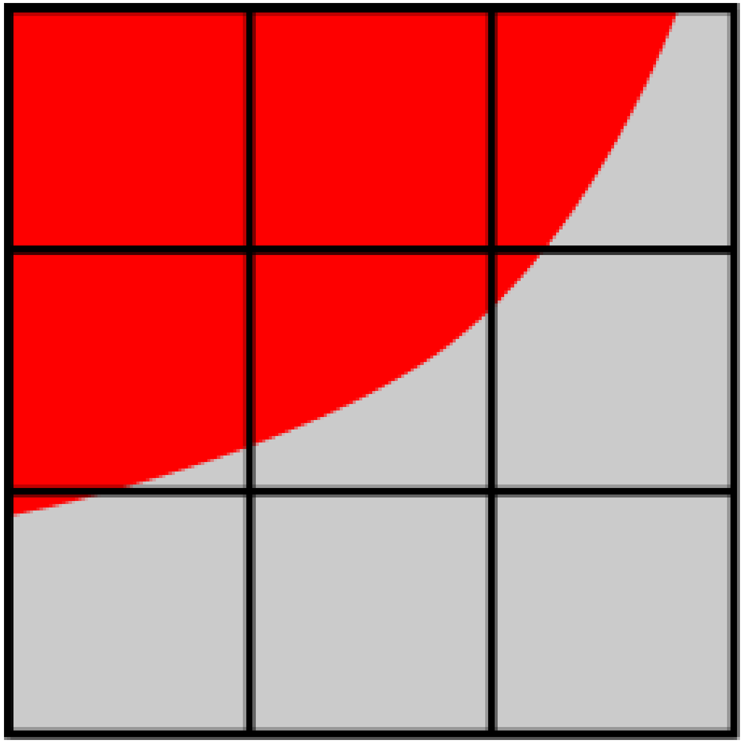
antialiasing - removing jaggies

1 sample/pixel



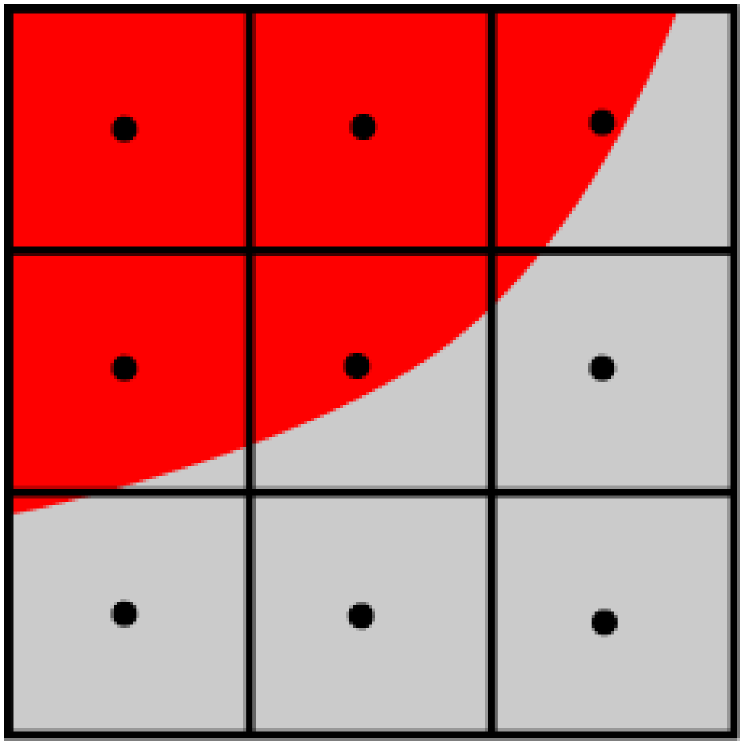
antialiasing - removing jaggies

1 sample/pixel



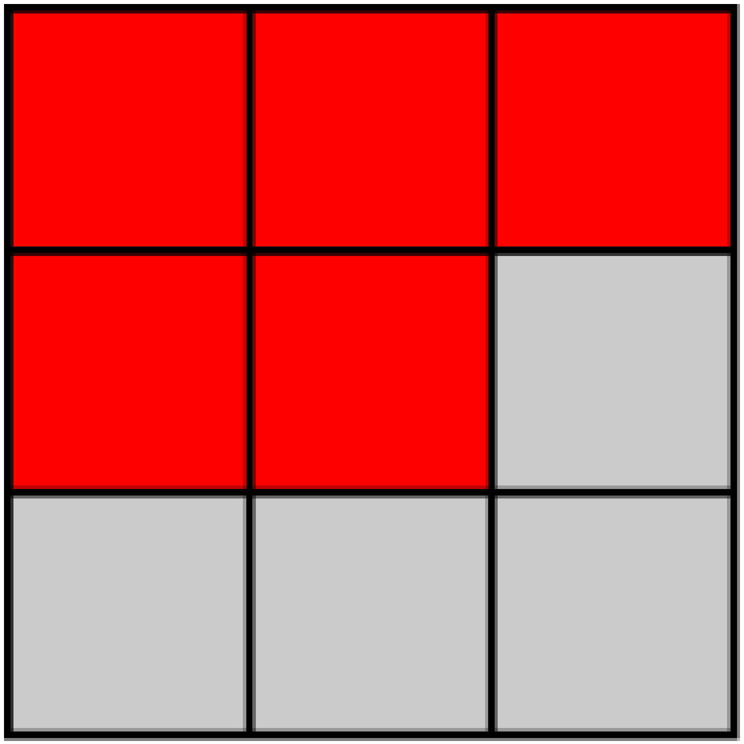
antialiasing - removing jaggies

1 sample/pixel



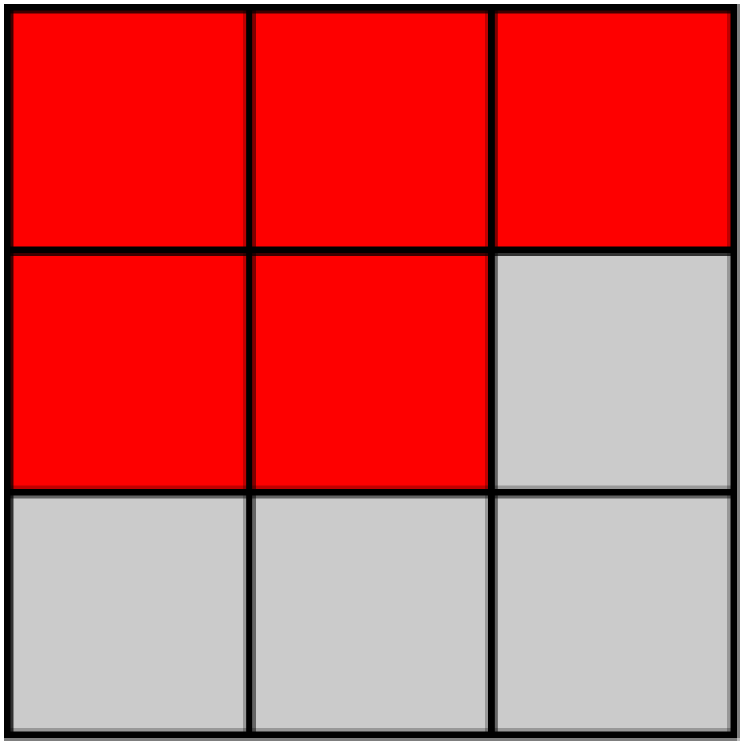
antialiasing - removing jaggies

1 sample/pixel

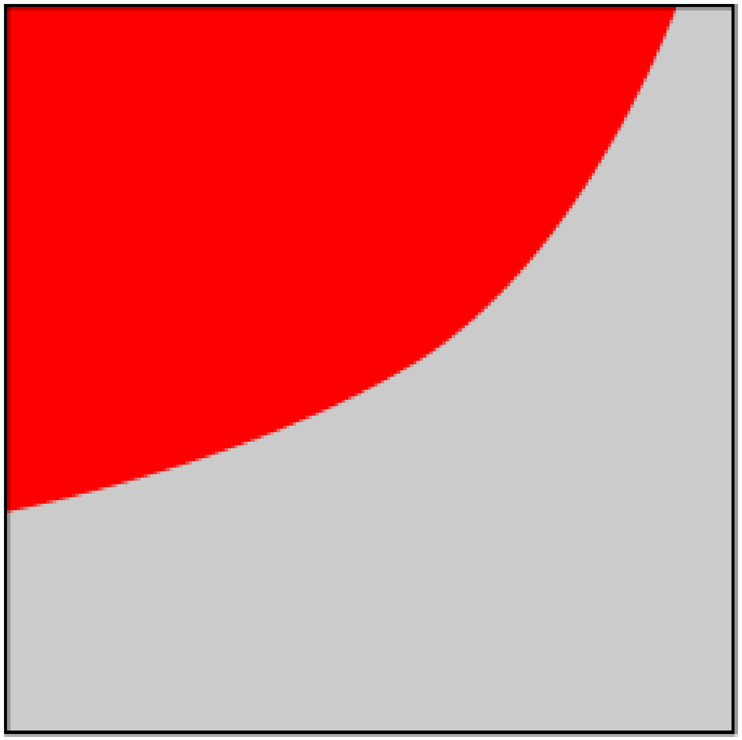


antialiasing - removing jaggies

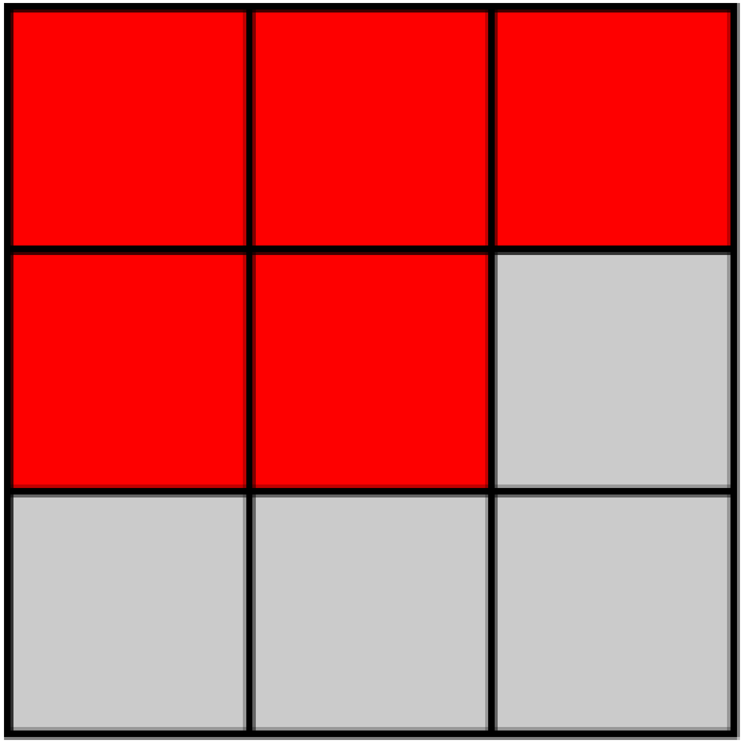
1 sample/pixel

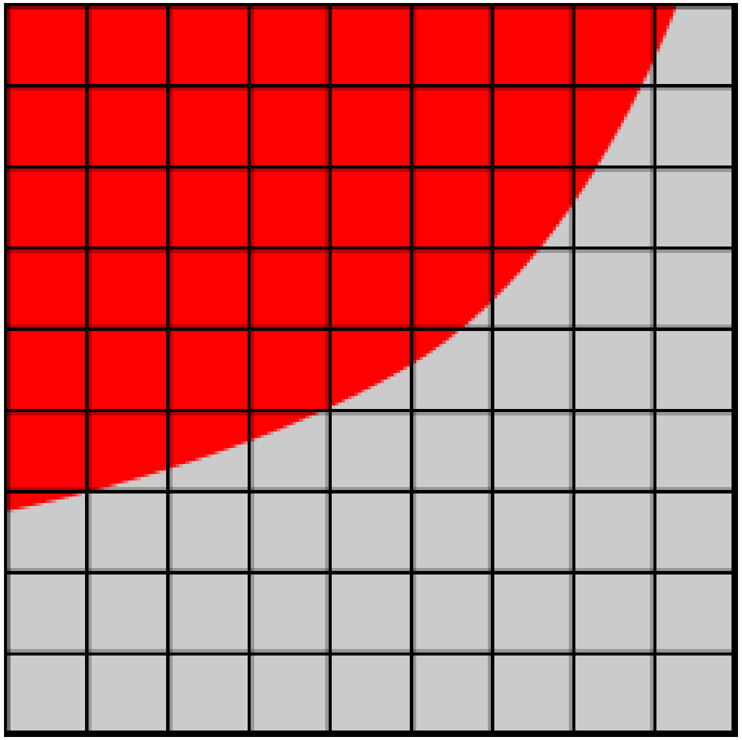


9 samples/pixel



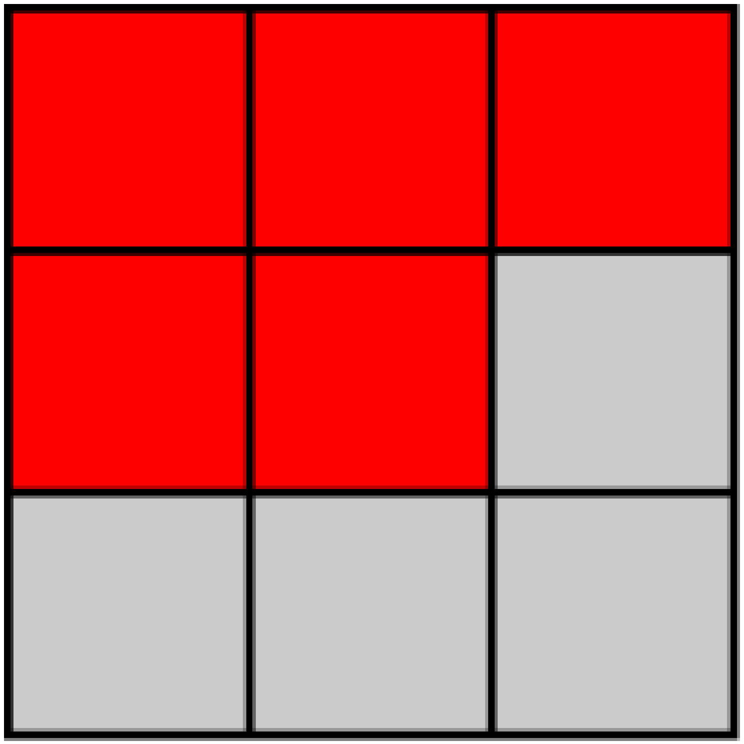
antialiasing - removing jaggies



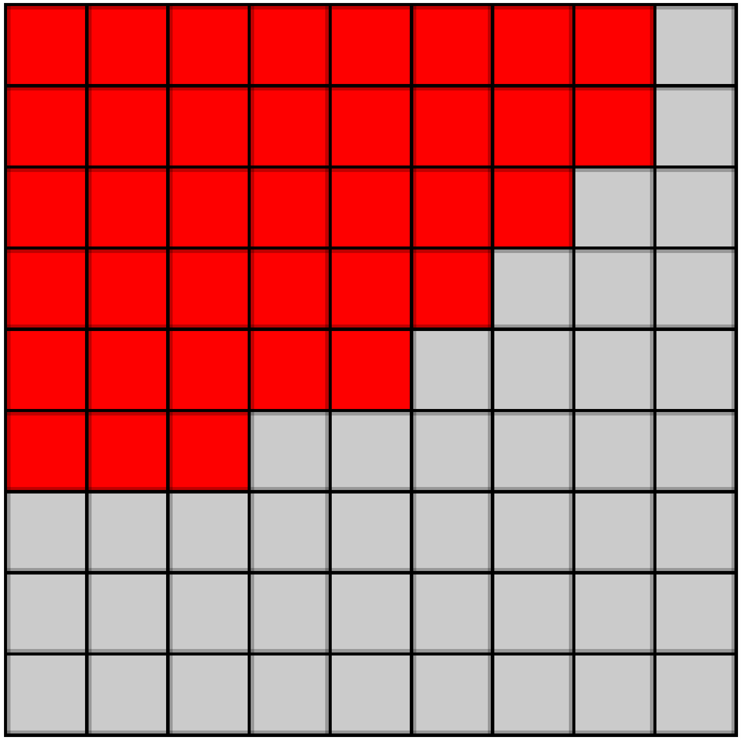


antialiasing - removing jaggies

1 sample/pixel

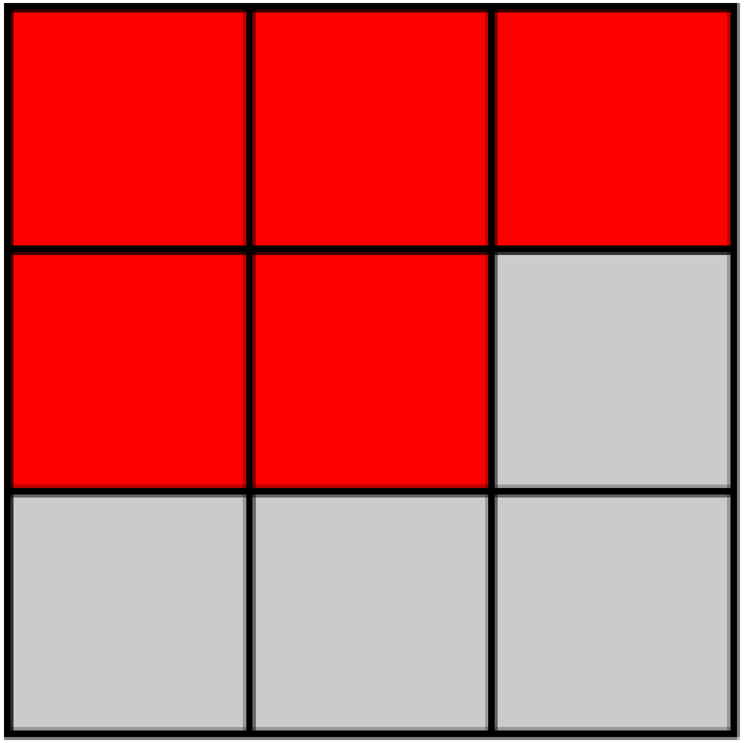


9 samples/pixel

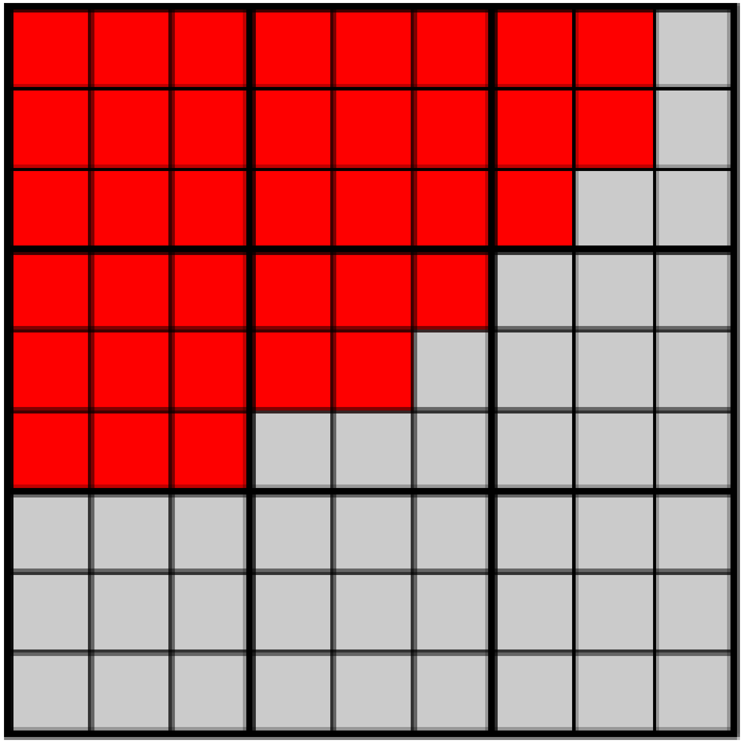


antialiasing - removing jaggies

1 sample/pixel

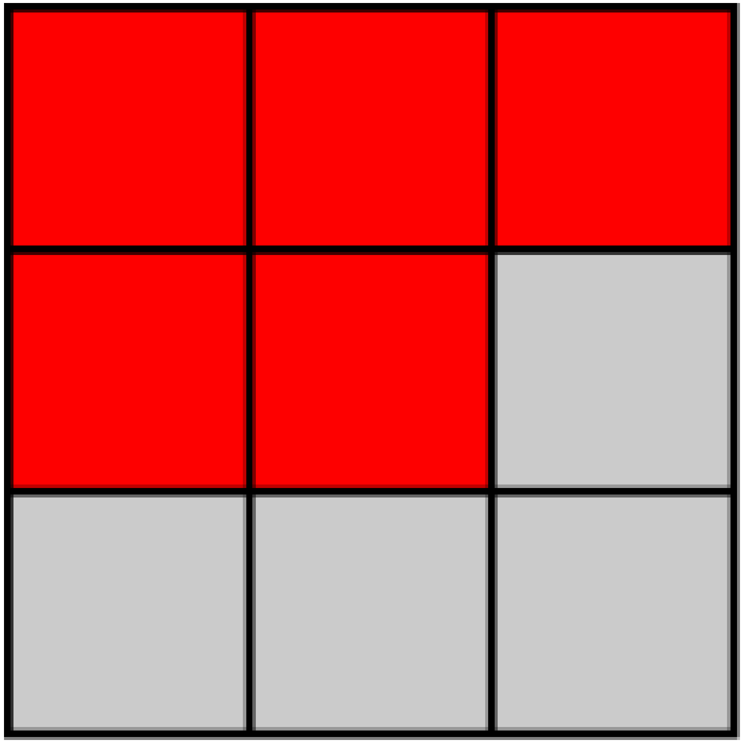


9 samples/pixel

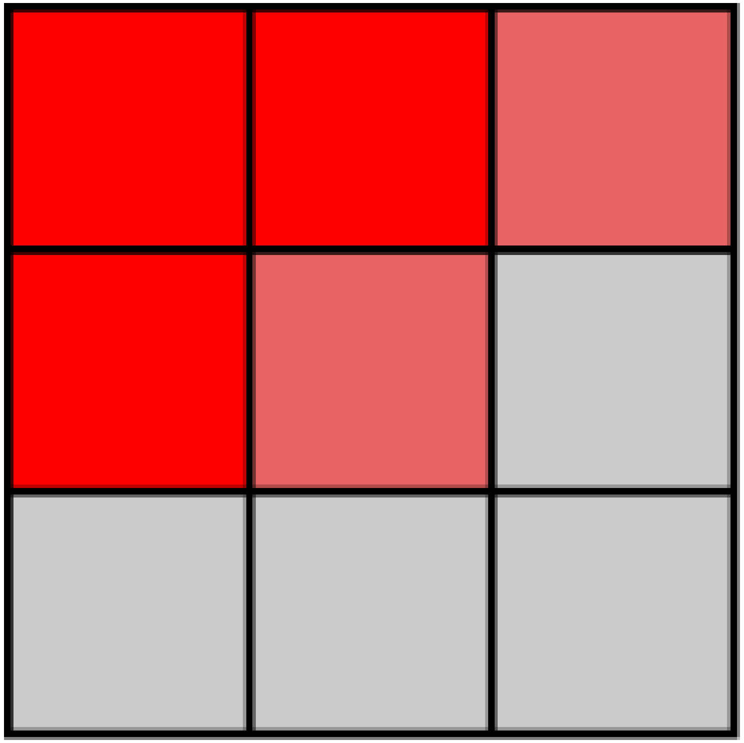


antialiasing - removing jaggies

1 sample/pixel



9 samples/pixel



antialiasing - removing jaggies

- for each pixel
  - take multiple samples
  - compute average
- poor-man antialiasing
  - more on this later in the course

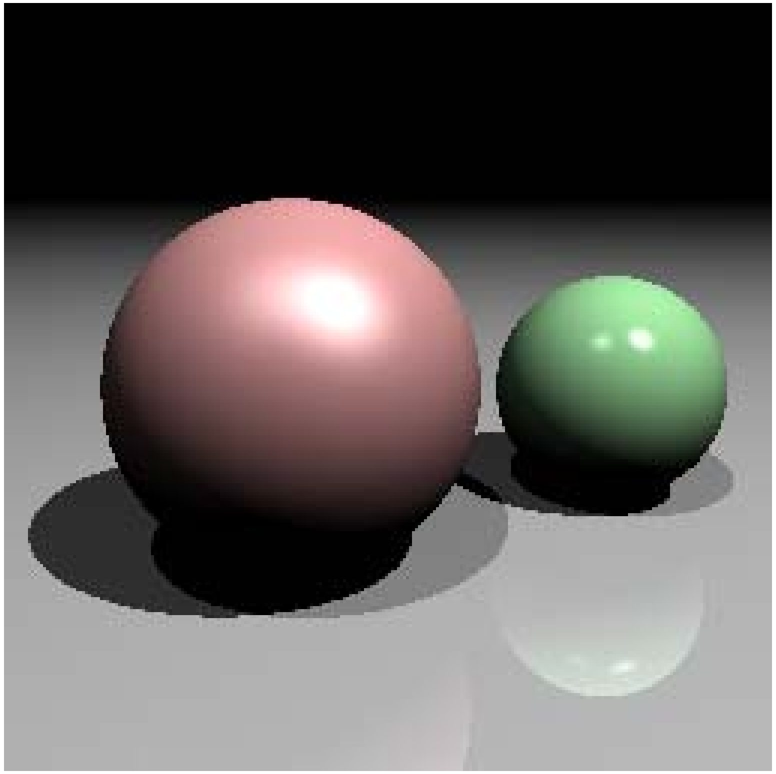
ray tracing pseudocode

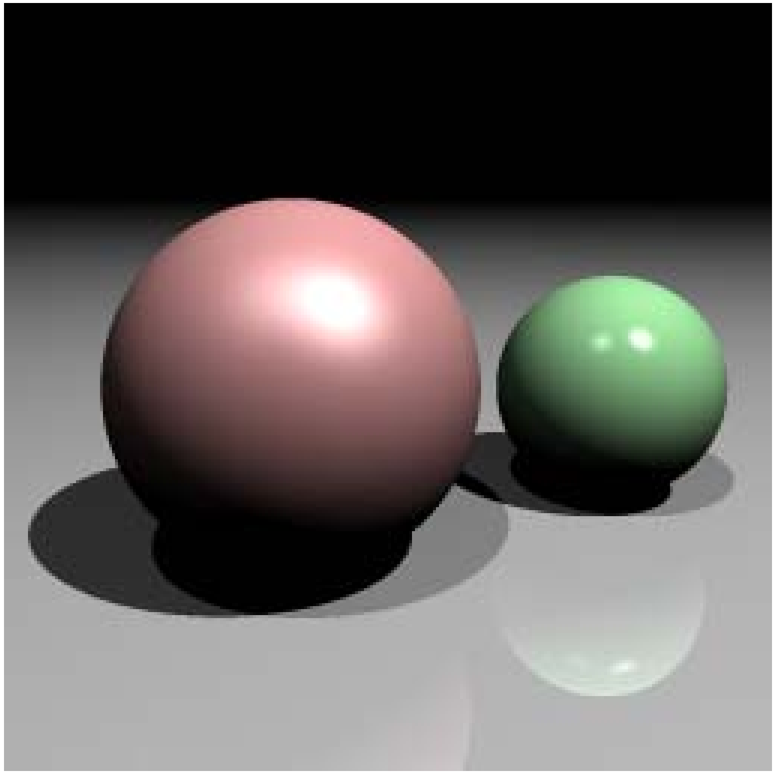
```
for(i = 0; i < imageWidth; i ++) {
  for(j = 0; j < imageHeight; j ++) {
    u = (i + 0.5)/imageWidth;
    v = (j + 0.5)/imageHeight;
    ray = camera.generateRay(u,v);
    c = computeColor(ray);
    image[i][j] = c;
  }
}
```


antialiased ray tracing pseudocode

```
for(i = 0; i < imageWidth; i ++) {
  for(j = 0; j < imageHeight; j ++) {
    color c = 0;
    for(ii = 0; ii < numberOfSamples; ii ++) {
      for(jj = 0; jj < numberofSamples; jj ++) {
        x = (i+(ii+0.5)/numberOfSamples)/imageWidth;
        y = (j+(jj+0.5)/numberofSamples)/imageHeight;
        ray = camera.generateRay(x,y);
        c += computeColor(ray);
      }
    }
    image[i][j] = c / (numberOfSamples^2);
  }
}
```

images so far





ray tracing details

- numerical precision issues come up
  - shadow acne: ray hits the visible point
  - solution: only intersect if distance > epsilon
- make sure you do not recurse infinitely when computing reflections or refractions
  - solution: simply stop after a given number

ray tracing wrap-up

- simple and general algorithm
  - ray cast to evaluate visibility
  - simplified shading model
  - ray cast to evaluate shadows
  - recursive execution for reflections/refractions
- inefficient – linear with number of primitives
  - sub-linear ray cast later in the course
- not photorealistic – too clean
  - missing soft shadows, realistic materials, realistic camera model, diffuse inter-reflection
- base of most general algorithms

