ray tracing

image formation

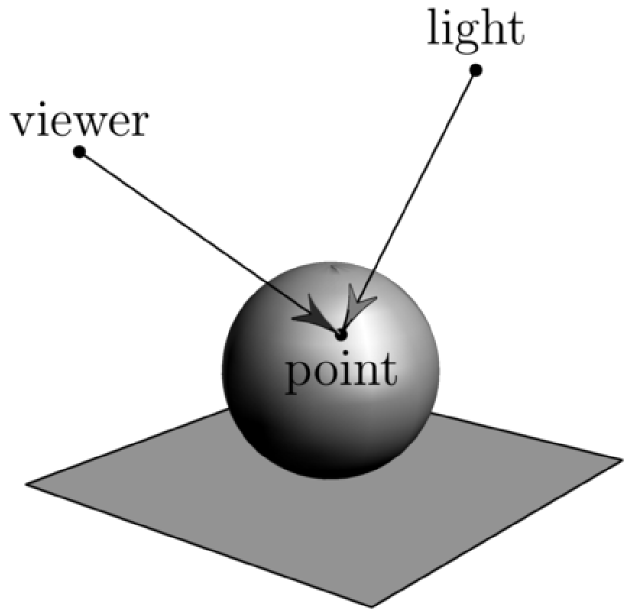
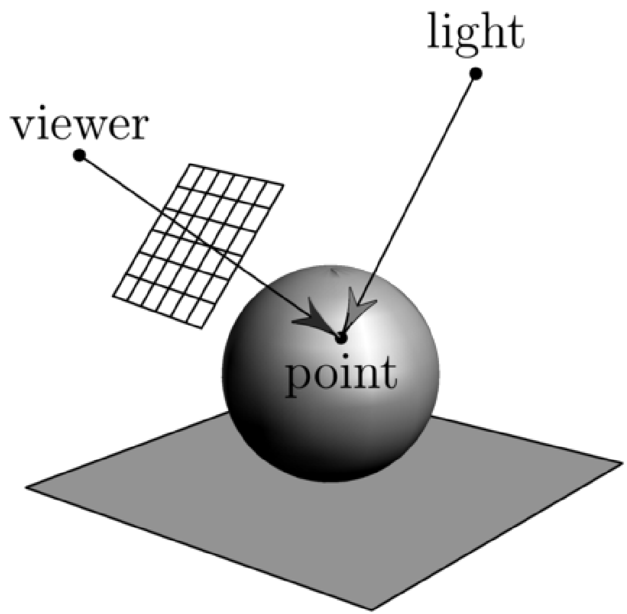


image formation

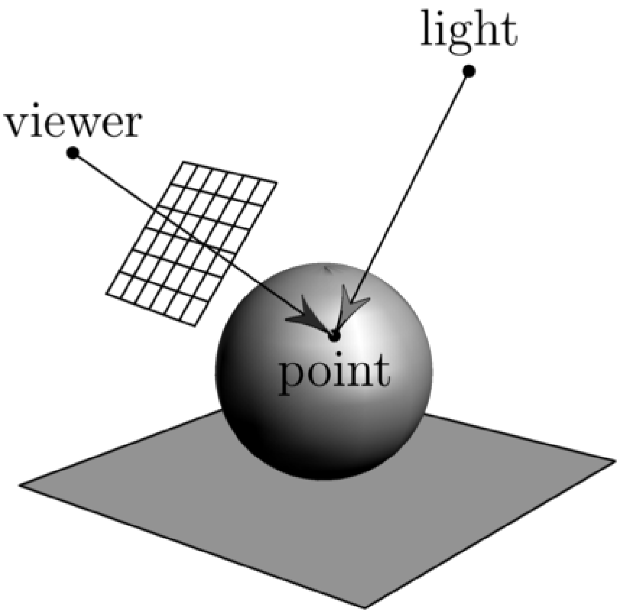


rendering

computational simulation of image formation

rendering

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



raytracing

a specific rendering algorithm

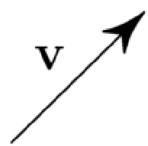
raytracing algorithm

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

vector math review

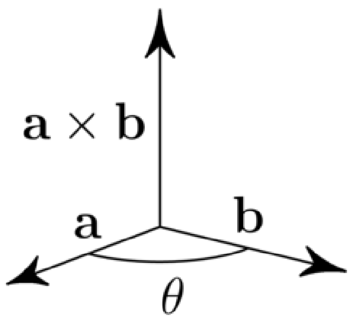
- point: location in 3D space
  - $\mathbf{P} = (P_x, P_y, P_z)$

- vector: direction and magnitude
  - $\mathbf{v} = (v_x, v_y, v_z)$



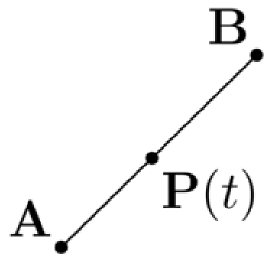
vector math review

- dot product
    - $\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos \theta$
- cross product
    - $|\mathbf{a} \times \mathbf{b}| = |\mathbf{a}||\mathbf{b}| \sin \theta$
    - $\mathbf{a} \times \mathbf{b}$ is orthogonal to $\mathbf{a}$ and $\mathbf{b}$



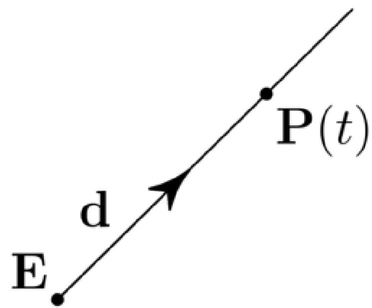
vector math review

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



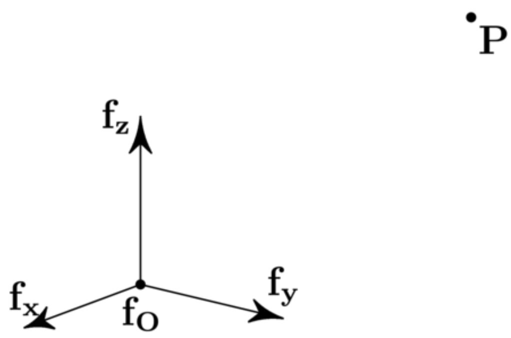
vector math review

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



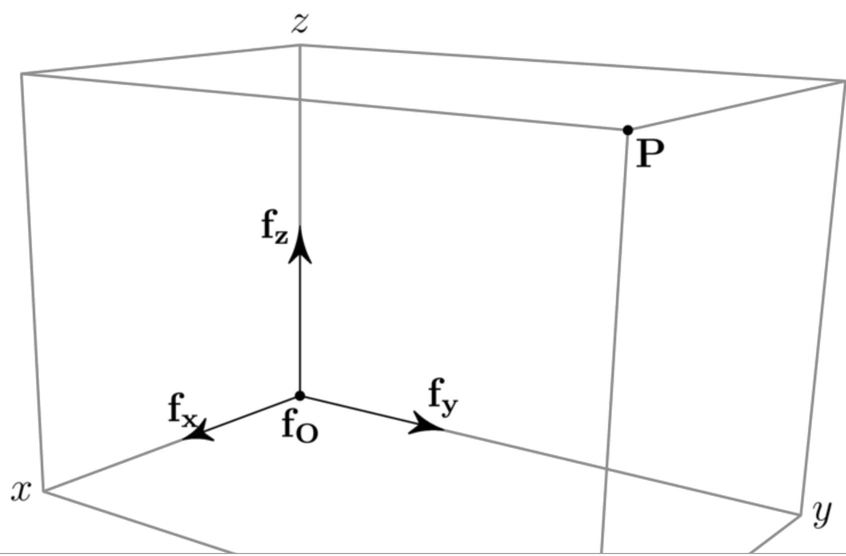
vector math review

- coordinate system aka frame
  - frame $\mathbf{f} = \{\mathbf{f_O}, \mathbf{f_x}, \mathbf{f_y}, \mathbf{f_z}\}$: position and orthonormal axes
  - default (or *world*) frame: origin and three major axes



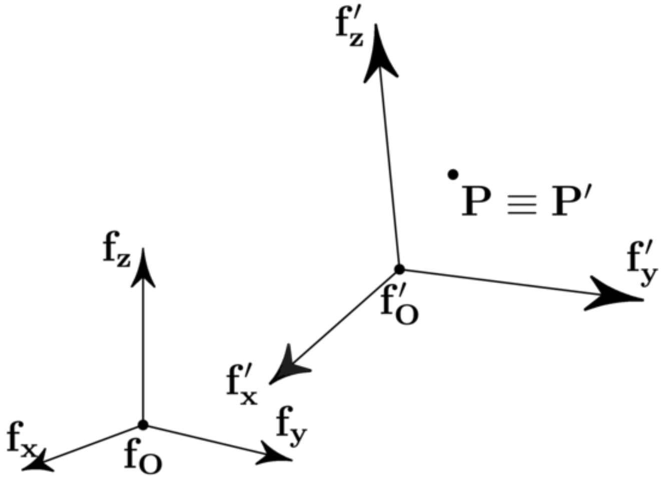
vector math review

- point coords are defined wrt a frame
  - $\mathbf{P} = (P_x, P_y, P_z)$ wrt $\{\mathbf{f_O}, \mathbf{f_x}, \mathbf{f_y}, \mathbf{f_z}\}$ (*world* if not specified)
  - $\mathbf{P} = \big((\mathbf{P} - \mathbf{f_O}) \cdot \mathbf{f_x}, (\mathbf{P} - \mathbf{f_O}) \cdot \mathbf{f_y}, (\mathbf{P} - \mathbf{f_O}) \cdot \mathbf{f_z}\big)$



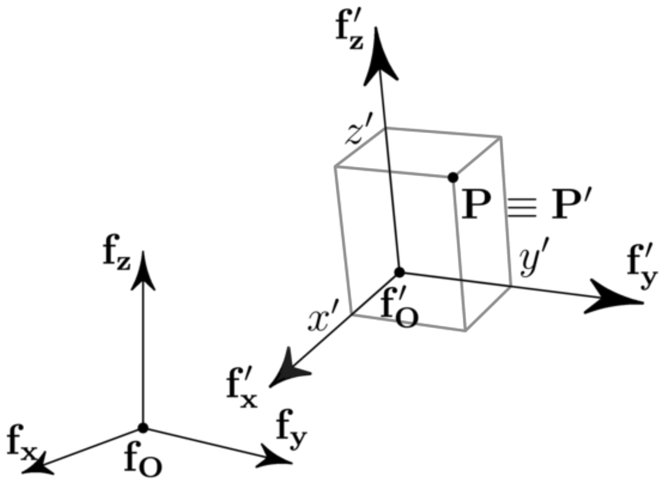
vector math review

- change of coordinate system $\mathbf{f} \to \mathbf{f}'$
    - $\mathbf{P}' = (P'_x, P'_y, P'_z)$ is $\mathbf{P}$ w.r.t $\{\mathbf{f}'_\mathbf{O}, \mathbf{f}'_\mathbf{x}, \mathbf{f}'_\mathbf{y}, \mathbf{f}'_\mathbf{z}\}$
    - $\mathbf{P}' = \Big((\mathbf{P} - \mathbf{f}'_\mathbf{O}) \cdot \mathbf{f}'_\mathbf{x}, (\mathbf{P} - \mathbf{f}'_\mathbf{O}) \cdot \mathbf{f}'_\mathbf{y}, (\mathbf{P} - \mathbf{f}'_\mathbf{O}) \cdot \mathbf{f}'_\mathbf{z}\Big)$



vector math review

- change of coordinate system $\mathbf{f}' \to \mathbf{f}$
  - $\mathbf{P}' = (P'_x, P'_y, P'_z)$ is $\mathbf{P}$ w.r.t $\{\mathbf{f}'_\mathbf{O}, \mathbf{f}'_\mathbf{x}, \mathbf{f}'_\mathbf{y}, \mathbf{f}'_\mathbf{z}\}$
  - $\mathbf{P} = \mathbf{f}'_\mathbf{O} + P'_x\mathbf{f}'_\mathbf{x} + P'_y\mathbf{f}'_\mathbf{y} + P'_z\mathbf{f}'_\mathbf{z}$



vector math review

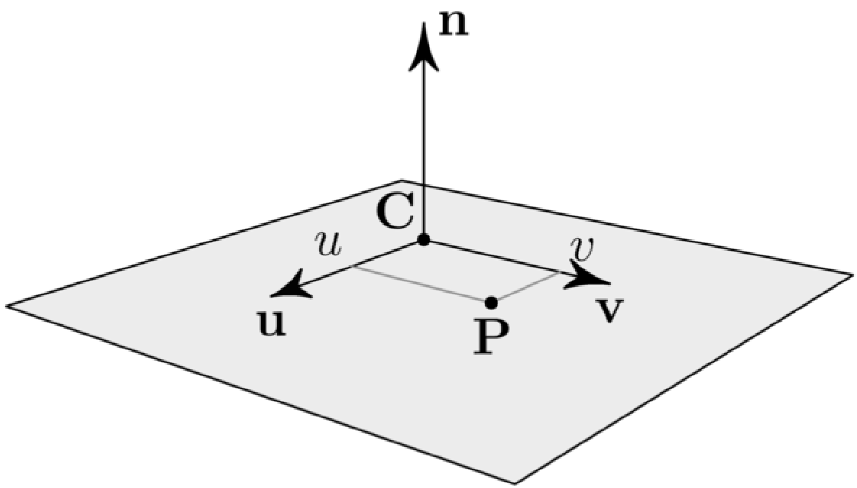
- vector coords are defined wrt a frame
    - to change coord system, ignore origin
    - $\mathbf{v} = v'_x \mathbf{f}'_\mathbf{x} + v'_y \mathbf{f}'_\mathbf{y} + v'_z \mathbf{f}'_\mathbf{z}$
    - $\mathbf{v}' = \left(\mathbf{v} \cdot \mathbf{f}'_\mathbf{x}, \mathbf{v} \cdot \mathbf{f}'_\mathbf{y}, \mathbf{v} \cdot \mathbf{f}'_\mathbf{z}\right)$

vector math review

- construct a frame from two non-orthonormal vectors $\mathbf{z}'$, $\mathbf{y}'$
  - assume that $\mathbf{z}'$ is not parallel to $\mathbf{y}'$
  - $\mathbf{z} = \mathbf{z}' / |\mathbf{z}'|$
  - $\mathbf{x} = \mathbf{y}' \times \mathbf{z} / |\mathbf{y}' \times \mathbf{z}|$
  - $\mathbf{y} = \mathbf{z} \times \mathbf{x}$
- construct a frame from a vector $\mathbf{z}'$
  - pick arbitrary $\mathbf{y}'$ and continue as above

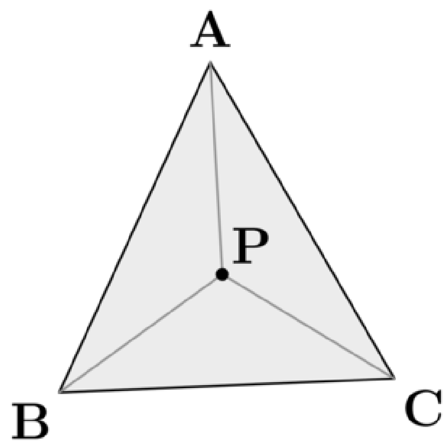
vector math review

- infinite plane
  - $\mathbf{P} \in plane \iff (\mathbf{P} - \mathbf{C}) \cdot \mathbf{n} = 0 \iff \mathbf{P} \cdot \mathbf{n} = d$
  - $\mathbf{P}(u, v) = \mathbf{C} + u \cdot \mathbf{u} + v \cdot \mathbf{v}$ with $(u, v) \in (-\infty, \infty)^2$
  - normal: $\mathbf{n} = \mathbf{u} \times \mathbf{v}$



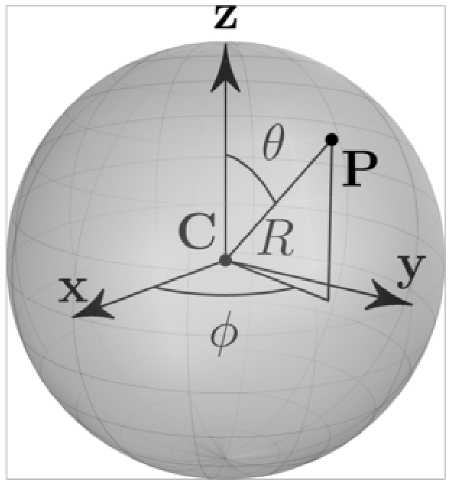
vector math review

- triangle baricentric coordinates
  - $\mathbf{P}(\alpha,\beta,\gamma)=\alpha\mathbf{A}+\beta\mathbf{B}+\gamma\mathbf{C}$ with $\alpha+\beta+\gamma=1$
  - $\mathbf{P}(\alpha,\beta)=\alpha(\mathbf{A}-\mathbf{C})+\beta(\mathbf{B}-\mathbf{C})+\mathbf{C}$
  - $\alpha=area(\mathbf{BCP})/area(\mathbf{ABC})$, ...



vector math review

- sphere
  - $\mathbf{P} \in sphere \iff |\mathbf{P} - \mathbf{C}| = R$
  - $\mathbf{P}(u, v) = \mathbf{C} + R \cdot (\cos\phi\sin\theta, \sin\phi\sin\theta, \cos\theta)$

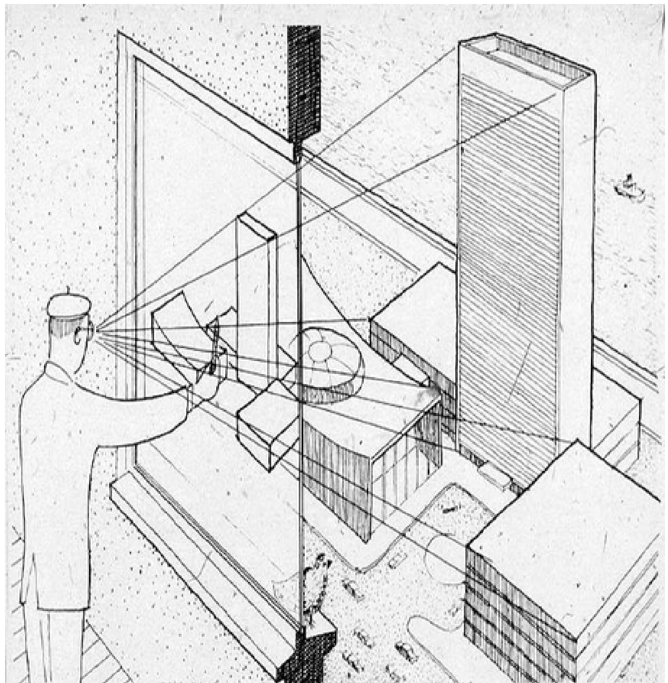


viewing

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

viewer model

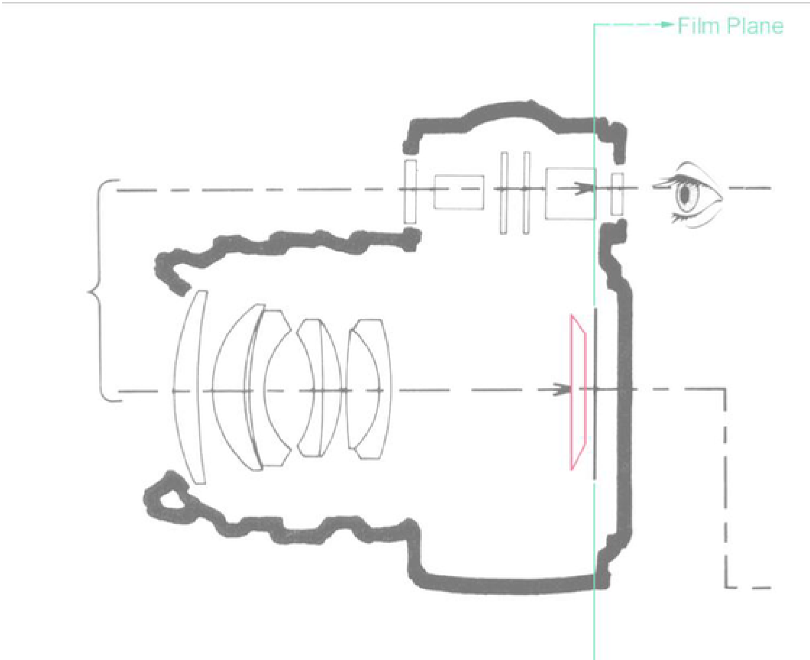
- a painter tracing objects on a canvas in front



[Marschner 2004 – original unknown]

viewer model

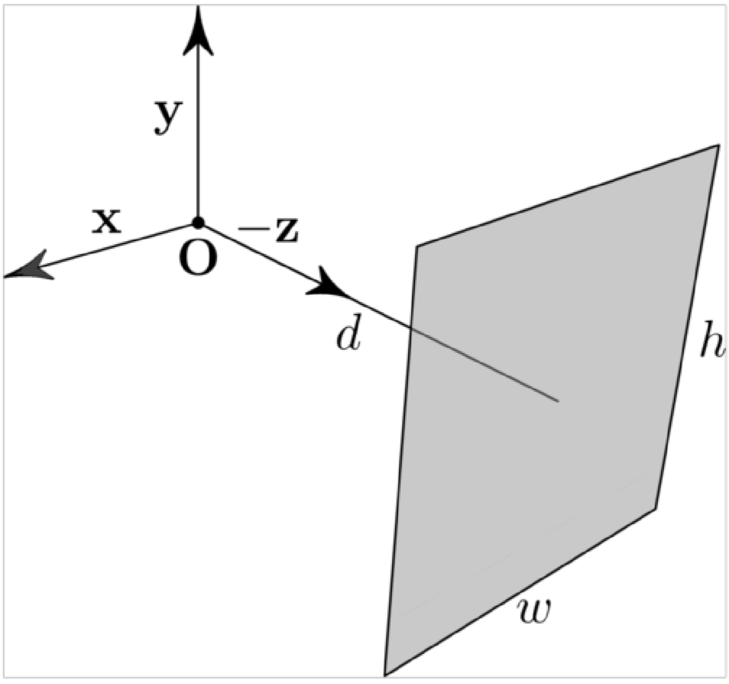
- equivalent to pinhole photography



[Marschner 2004 – original unknown]

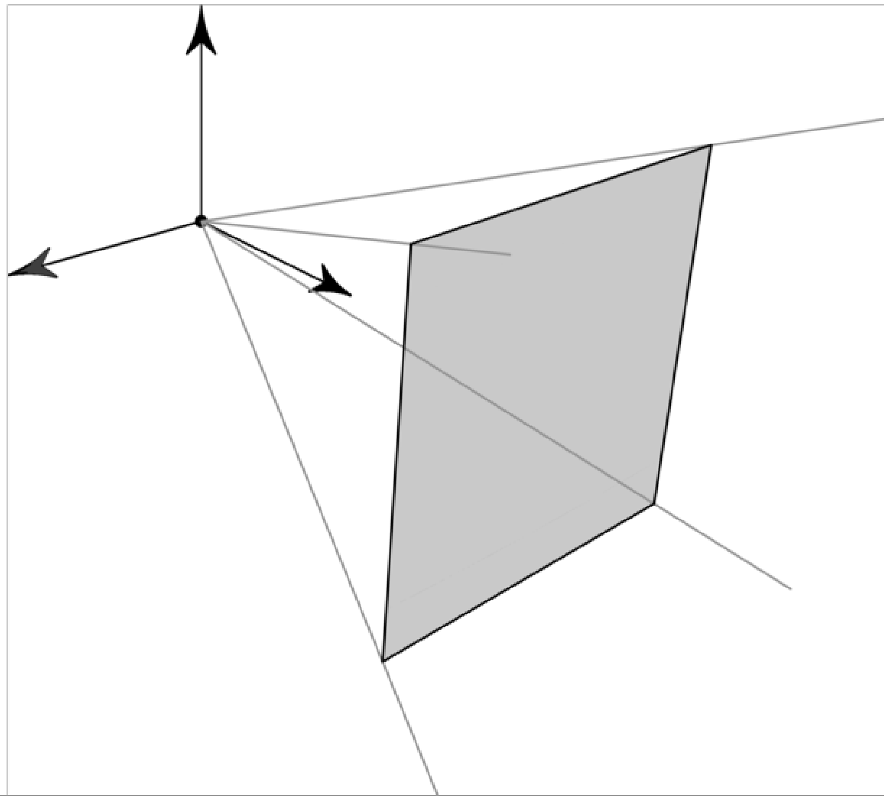
viewer model -- parameters

- camera frame: position $\mathbf{O}$ and orientation $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$
- image plane: distance d and size w, h



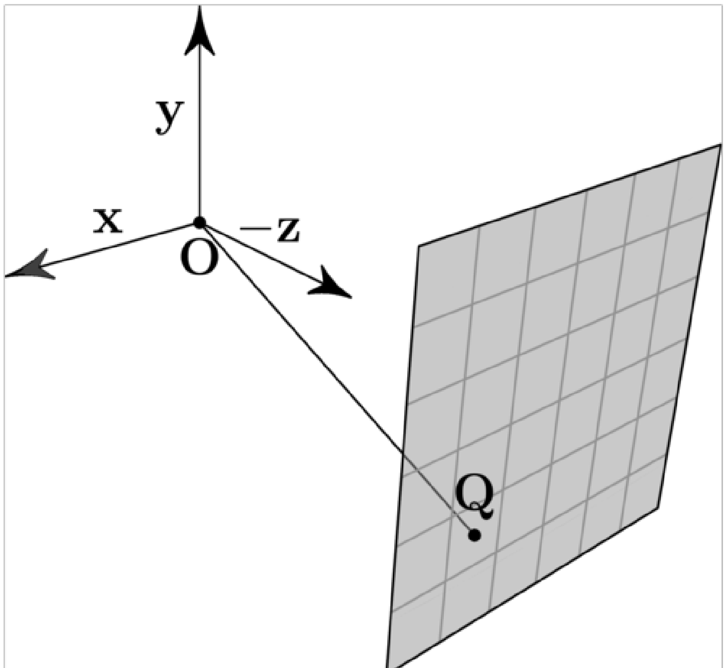
view frustum

- all visible points within a truncated pyramid



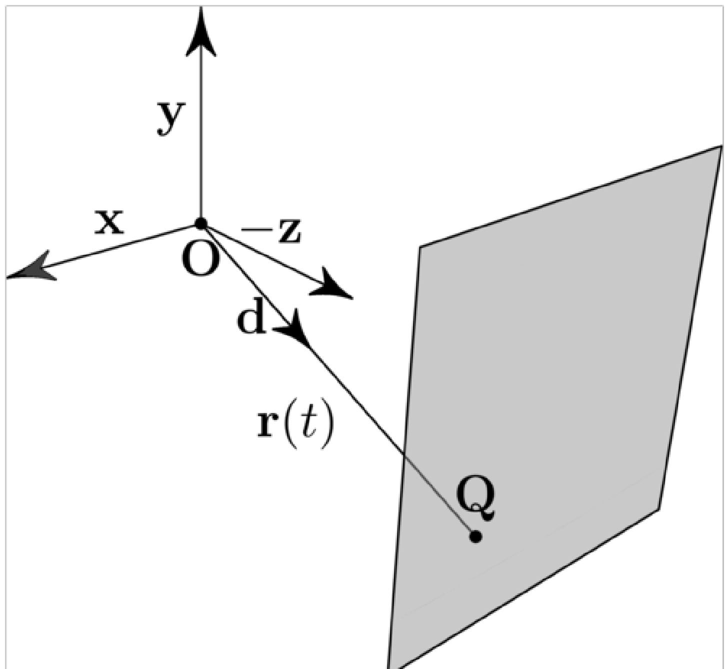
generating view rays

- for each pixel, ray from camera center to the pixel center



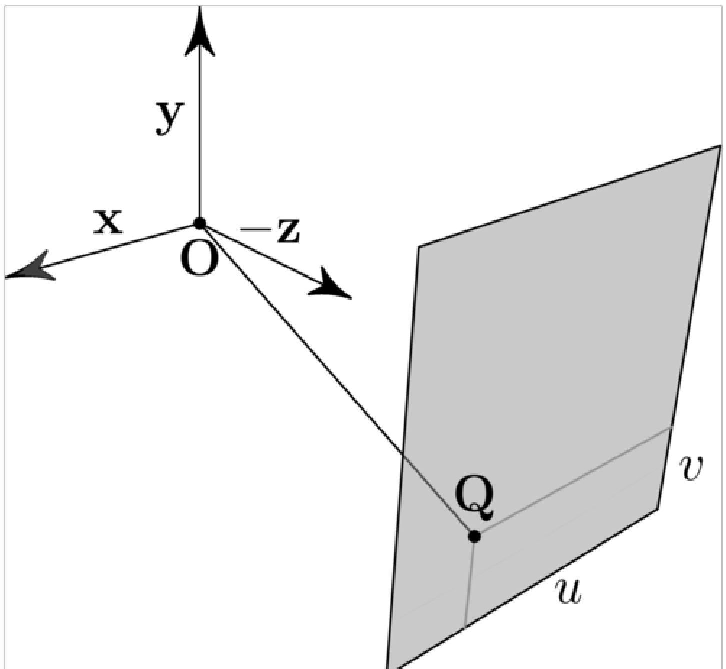
generating view rays

- ray: $\mathbf{r} = \mathbf{O} + t(\mathbf{Q} - \mathbf{O})/|\mathbf{Q} - \mathbf{O}|$
- $\mathbf{Q}$ point on image plane



generating view rays

- $\mathbf{Q}(u, v) = (u - 0.5)w\mathbf{x} + (v - 0.5)h\mathbf{y} - d\mathbf{z}$
- image plane params: $(u, v) \in [0, 1]^2$, origin at bottom



geometry model

- simple shapes
  - spheres, quads, traingles
- complex shapes
  - handled as collections of simple shapes later in the course

ray-shape intersection

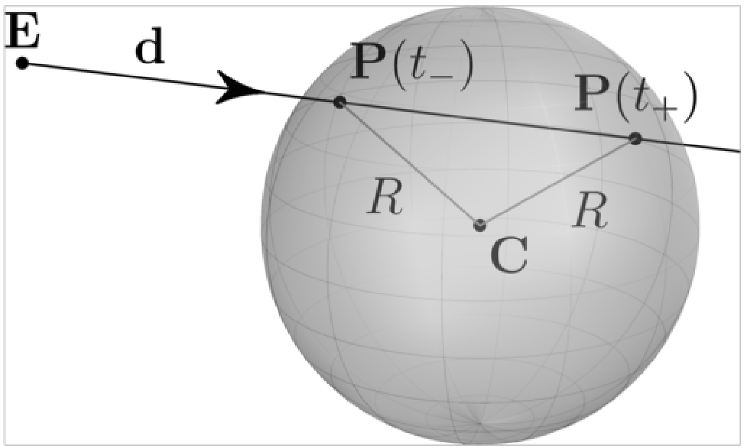
- determine visible surface by finding closest intersection along a ray
- ray $\mathbf{r} : \mathbf{E} + t\mathbf{d}$ with $t \in (t_{min}, t_{max})$
    - keep explicit bounds on t
    - e.g. used in shadows and to improve numerical precision
    - if not specified otherwise: $t_{min} = \epsilon$, $t_{max} = \infty$
    - ϵ mitigate numerical precision issues ("shadow acne")
        - value is scene depedent: start with 10^{-5}

ray-sphere intersection

point on a ray: $\mathbf{P}(t) = \mathbf{E} + t\mathbf{d}$

point on a sphere: $|\mathbf{P}(t) - \mathbf{C}| = R$

by substitution: $|\mathbf{E} + t\mathbf{d} - \mathbf{C}| = R$



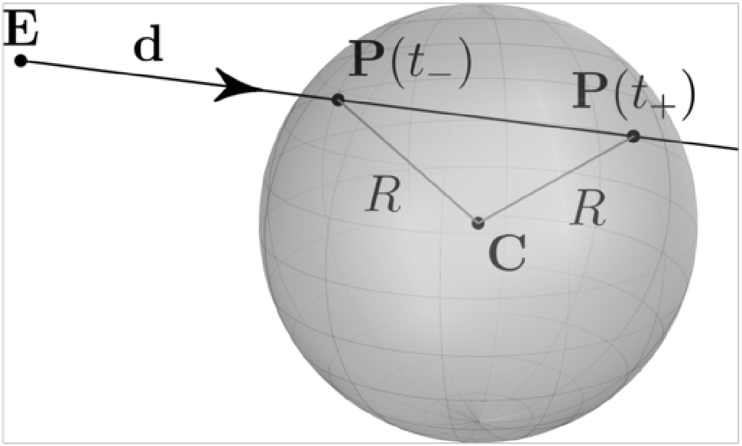
ray-sphere intersection

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

with: $a = |\mathbf{d}|^2$, $b = \mathbf{2d} \cdot (\mathbf{E} - \mathbf{C})$, $c = |\mathbf{E} - \mathbf{C}|^2 - R^2$

determinant: $d = b^2 - 4ac$

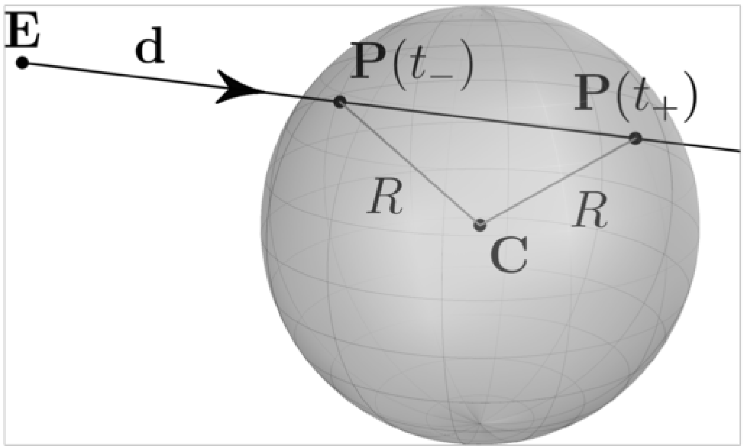
no solution for $d < 0$



ray-sphere intersection

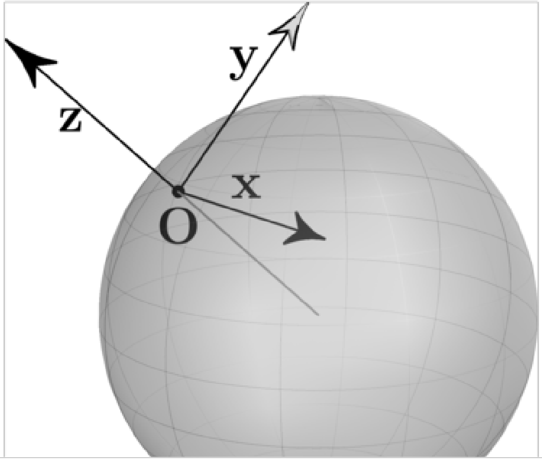
two solutions: $t_{\pm} = \left(-b \pm \sqrt{d}\right) / (2a)$

pick smallest t such that $t \in (t_{min}, t_{max})$



ray-sphere intersection

- shading frame at $\mathbf{P} = \mathbf{f_O}$ with normal $\mathbf{n} = \mathbf{f_z}$ with
  - $\mathbf{P} = \mathbf{E} + t\mathbf{d}$ and $\mathbf{P}^l = (\mathbf{P} - \mathbf{C})/R$
  - $\theta = \arccos P^l_z$ and $\phi = \arctan(P^l_y, P^l_x)$
- $\mathbf{f} = \{\mathbf{P}, \mathbf{x}, \mathbf{y}, \mathbf{P}^l\}$, where $\mathbf{x} = (\sin\phi, \cos\phi, 0)$, $\mathbf{y} = (\cos\theta\cos\phi, \cos\theta\sin\phi, \sin\theta)$

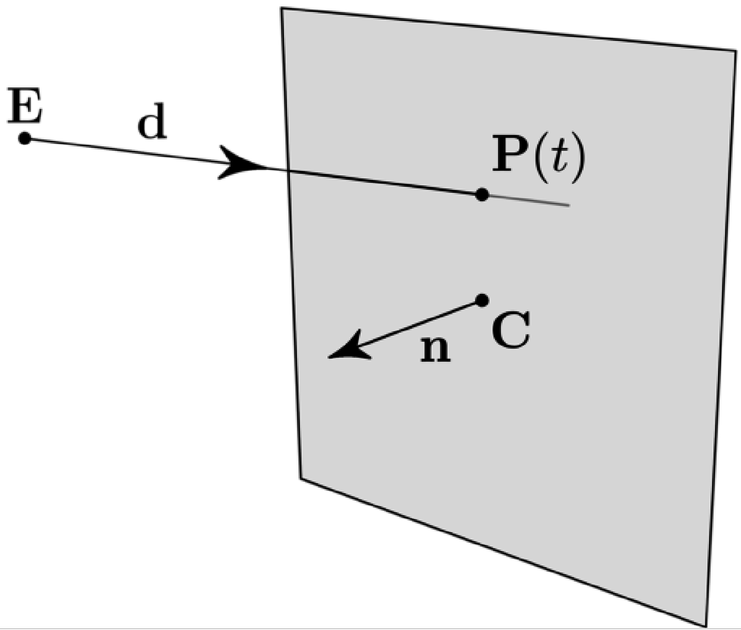


ray-plane intersection

point on a ray: $\mathbf{P}(t) = \mathbf{E} + t\mathbf{d}$

point on a plane: $(\mathbf{P}(t) - \mathbf{C}) \cdot \mathbf{n} = 0$

by substitution: $(\mathbf{E} + t\mathbf{d} - \mathbf{C}) \cdot \mathbf{n} = 0$

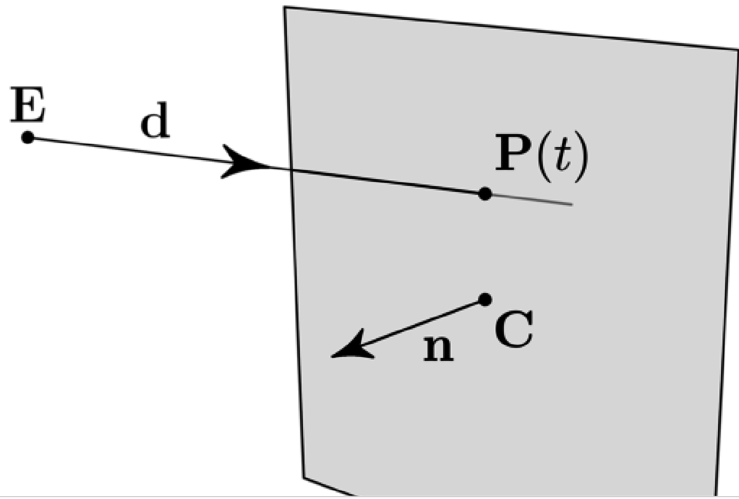


ray-plane intersection

one solution for $\mathbf{d} \cdot \mathbf{n} \neq 0$, no/infinite solutions otherwise

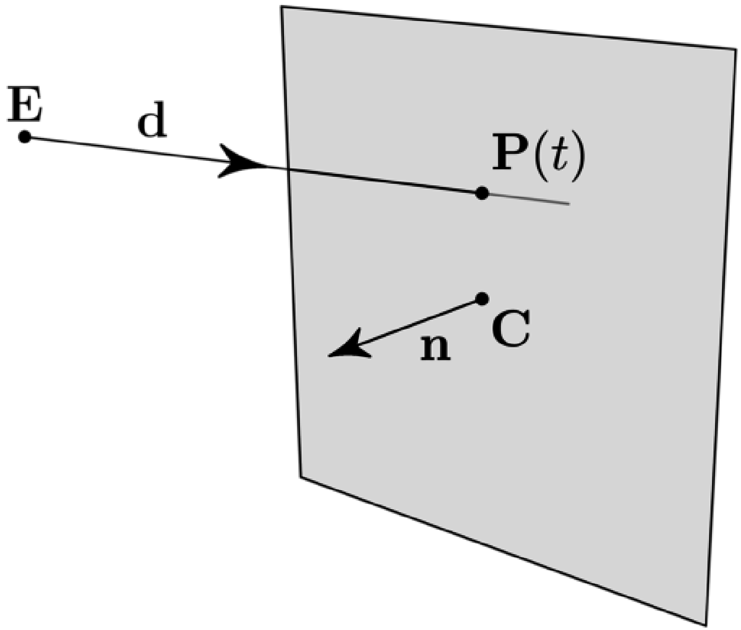
$$t = \frac{(\mathbf{C} - \mathbf{E}) \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}$$

check that $t \in (t_{min}, t_{max})$



ray-plane intersection

- shading frame: $\mathbf{f} = \{\mathbf{e} + t\mathbf{d}, \mathbf{u}, \mathbf{v}, \mathbf{n}\}$

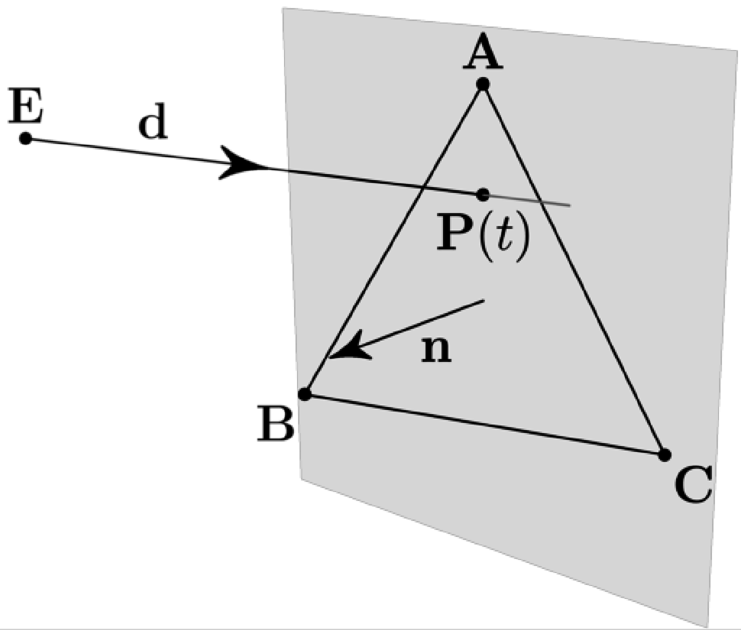


ray-triangle intersection

point on ray: $\mathbf{P}(t) = \mathbf{E} + t\mathbf{d}$

point on triangle: $\mathbf{P}(\alpha, \beta) = \alpha(\mathbf{A} - \mathbf{C}) + \beta(\mathbf{B} - \mathbf{C}) + \mathbf{C}$

by substitution: $\mathbf{E} + t\mathbf{d} = \alpha(\mathbf{A} - \mathbf{C}) + \beta(\mathbf{B} - \mathbf{C}) + \mathbf{C}$



ray-triangle intersection

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

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

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

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

ray-triangle intersection

use Cramer's rule

$$t = \frac{\begin{vmatrix} \mathbf{e} & \mathbf{a} & \mathbf{b} \end{vmatrix}}{\begin{vmatrix} -\mathbf{d} & \mathbf{a} & \mathbf{b} \end{vmatrix}} = \frac{(\mathbf{e} \times \mathbf{a}) \cdot \mathbf{b}}{(\mathbf{d} \times \mathbf{b}) \cdot \mathbf{a}}$$

$$\alpha = \frac{\begin{vmatrix} -\mathbf{d} & \mathbf{e} & \mathbf{b} \end{vmatrix}}{\begin{vmatrix} -\mathbf{d} & \mathbf{a} & \mathbf{b} \end{vmatrix}} = \frac{(\mathbf{d} \times \mathbf{b}) \cdot \mathbf{e}}{(\mathbf{d} \times \mathbf{b}) \cdot \mathbf{a}}$$

$$\beta = \frac{\begin{vmatrix} -\mathbf{d} & \mathbf{a} & \mathbf{e} \end{vmatrix}}{\begin{vmatrix} -\mathbf{d} & \mathbf{a} & \mathbf{b} \end{vmatrix}} = \frac{(\mathbf{e} \times \mathbf{a}) \cdot \mathbf{d}}{(\mathbf{d} \times \mathbf{b}) \cdot \mathbf{a}}$$

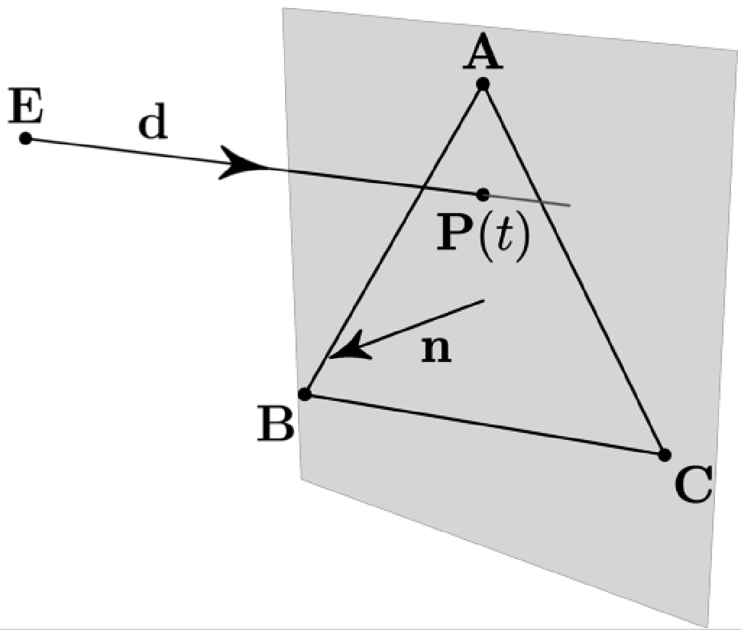
test for

$$t \in (t_{min}, t_{max}), \alpha \geq 0, \beta \geq 0, \alpha + \beta \leq 1$$

ray-triangle intersection

- shading frame: $\mathbf{f} = \{\mathbf{e} + t\mathbf{d}, \mathbf{u}, \mathbf{v}, \mathbf{n}\}$
  - create frame by orthonomalization with

    $\mathbf{z}' = (\mathbf{B} - \mathbf{A}) \times (\mathbf{C} - \mathbf{A}), \mathbf{x}' = (\mathbf{B} - \mathbf{A})$



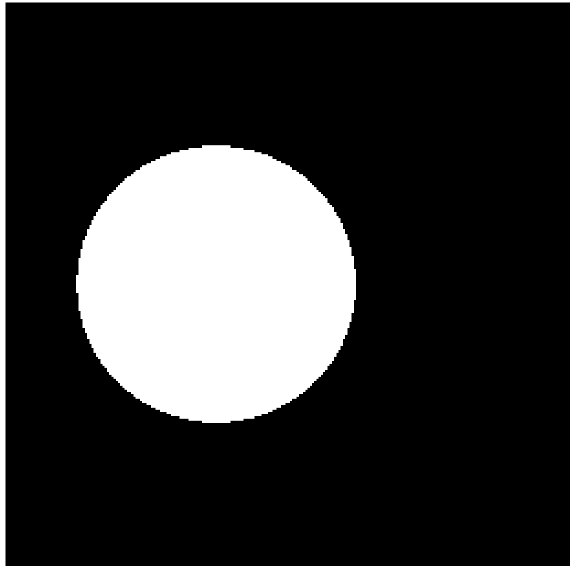
intersection and coord systems

- transform the object
    - simple for triangles, since they transforms to triangles
    - but objects may require more complex intersection tests
- transform the ray
    - much more elegant
    - works on any surface
    - allow for much simpler intersection tests

intersection and coord systems

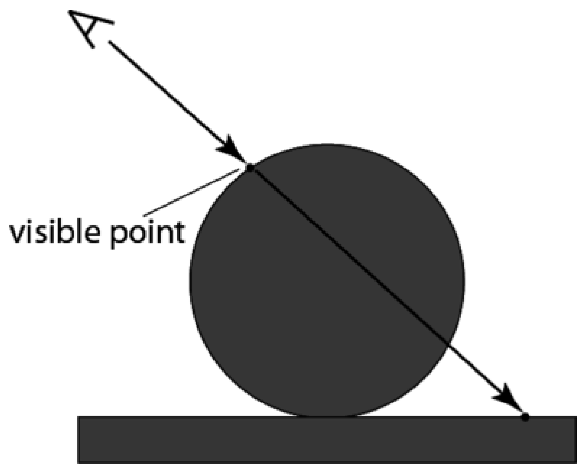
- ray $\mathbf{r} = \{\mathbf{E}, \mathbf{d}\}$ wrt $\mathbf{f}$ (e.g. *world*)
- object o' defined wrt $\mathbf{f}'$ (in turn defined wrt $\mathbf{f}$)
- transform rays $\mathbf{r}' = \{\mathbf{E}', \mathbf{d}'\}$
  - transform origin/direction as point/vector
- intersect object o' with transformed ray $\mathbf{r}'$
  - use standard intersection tests
- transform intersection frame back to $\mathbf{f}$
  - transform origin/axes as point/vectors

image so far



intersecting many shapes

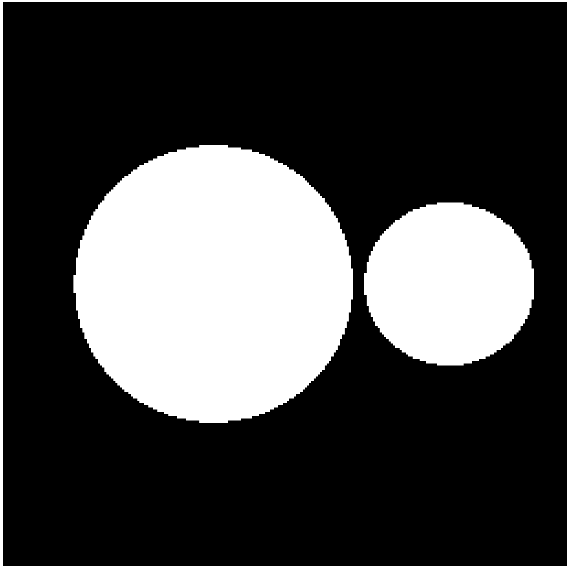
- intersect each primitive
- pick closest intersection
- essentially a line search



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;
    }
  }
}
```

image so far



shading

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

shading

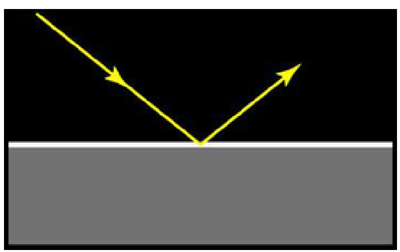
variation in observed color across a surface

shading

- compute reflected light
- depends on:
  - view position
  - incoming light, i.e. lighting
  - surface geometry
  - surface material

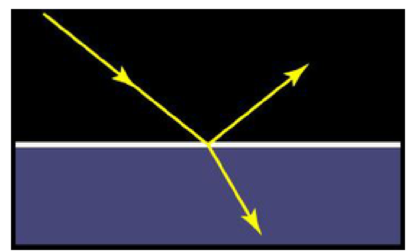
real-world materials

Metals



[Marschner 2004]

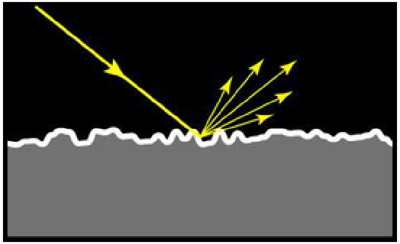
Dielectric



[Marschner 2004]

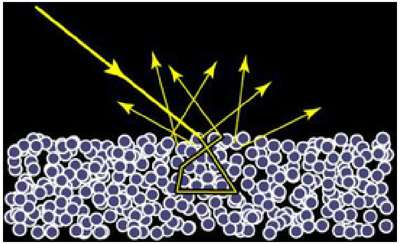
real-world materials

Metals



[Marschner 2004]

Dielectric



[Marschner 2004]

shading models

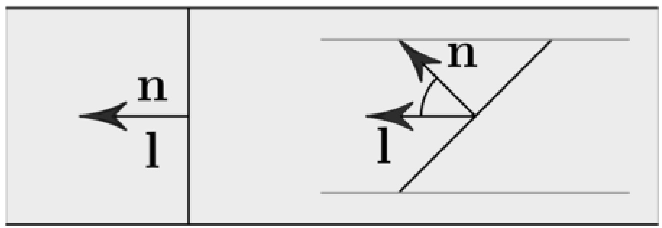
- empirical models
    - produce believable images
    - simple and efficient
    - only for simple materials
- physically-based shading models
    - can reproduce accurate effects
    - more complex and expensive
- will concentrate on empirical models first

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)

incident light

- beam of light is more spread on oblique surfaces
- incident light depends on angle
- light fraction: $f = |\mathbf{n} \cdot \mathbf{l}|$

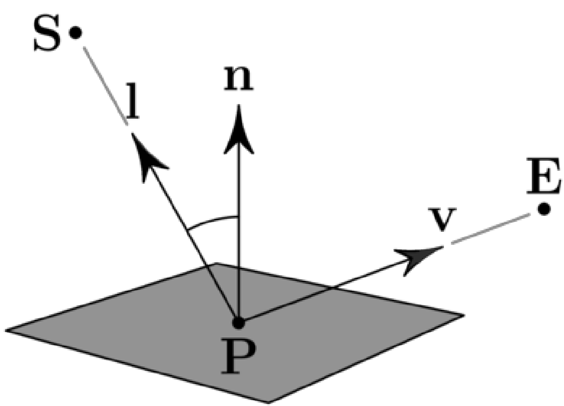


surface reflectance

- surface reflectance is described by the BRDF, *bidirectional surface distribution functions*
- BRDF is simple for simple shading models
- in general, the BRDF is a function of incoming and outgoing angles $\rho(\mathbf{l}, \mathbf{v}; \mathbf{f})$
  - $\mathbf{l}$ is the direction from the point to the light
  - $\mathbf{v}$ is the direction from the point to the viewer
  - $\mathbf{f}$ is the local shading frame that describes surface orientation (normal and tangent)

lambert diffuse model

- simple and efficient diffuse model
- light is scattered uniformly in all directions
- brdf: $\rho_d(\mathbf{l}, \mathbf{v}; \mathbf{f}) = k_d$
- surface color: $C_d = \rho_d(\mathbf{l}, \mathbf{v}; \mathbf{f}) \cdot |\mathbf{n} \cdot \mathbf{l}| = k_d |\mathbf{n} \cdot \mathbf{l}|$

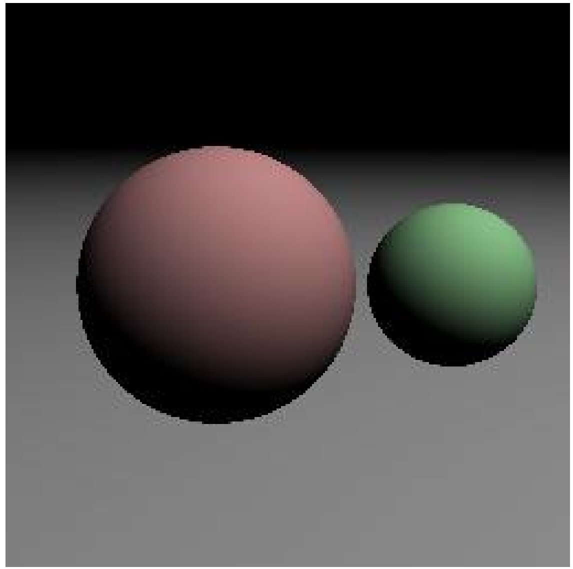


lambert diffuse model

- produce matte appearance

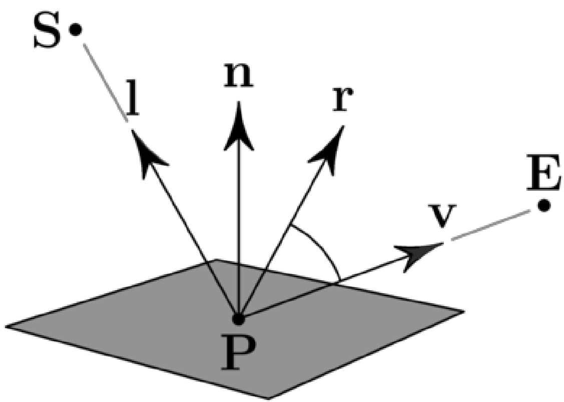
left-to-right: increasing kd

image so far



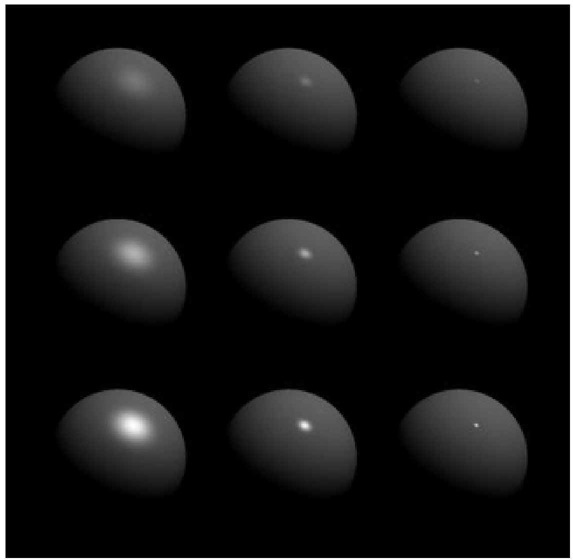
phong specular model

- empirical, used to look good enough
- cosine of mirror $\mathbf{r}$ and view $\mathbf{v}$ direction
- reflected direction: $\mathbf{r} = -\mathbf{l} + 2(\mathbf{n} \cdot \mathbf{l})\mathbf{n}$
- brdf: $\rho_s(\mathbf{l}, \mathbf{v}; \mathbf{f}) = k_s \max(0, \mathbf{v} \cdot \mathbf{r})^n$
- $C_s = \rho_s(\mathbf{l}, \mathbf{v}; \mathbf{f}) \cdot |\mathbf{n} \cdot \mathbf{l}| = k_s \max(0, \mathbf{v} \cdot \mathbf{r})^n \cdot |\mathbf{n} \cdot \mathbf{l}|$



phong specular model

- produces highlight, shiny appearance



left-to-right: increasing n, top-to-bottom: increasing k_s

blinn specular model

- slightly better than Phong
- cosine of bisector $\mathbf{h}$ and normal $\mathbf{n}$
- bisector: $\mathbf{h} = (\mathbf{l} + \mathbf{v})/|\mathbf{l} + \mathbf{v}|$
- brdf: $\rho_s(\mathbf{l}, \mathbf{v}; \mathbf{f}) = k_s \max(0, \mathbf{n} \cdot \mathbf{h})^n$
- $C_s = \rho_s(\mathbf{l}, \mathbf{v}; \mathbf{f}) \cdot |\mathbf{n} \cdot \mathbf{l}| = k_s \max(0, \mathbf{n} \cdot \mathbf{h})^n \cdot |\mathbf{n} \cdot \mathbf{l}|$

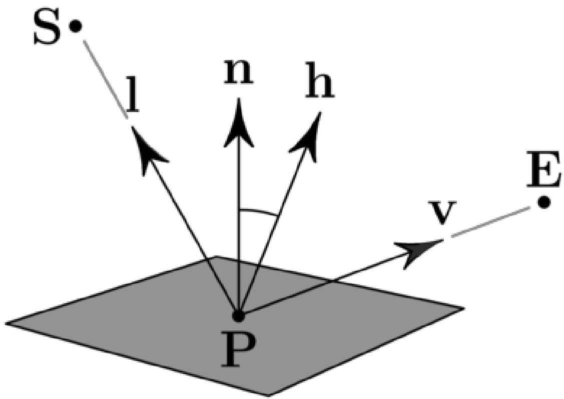
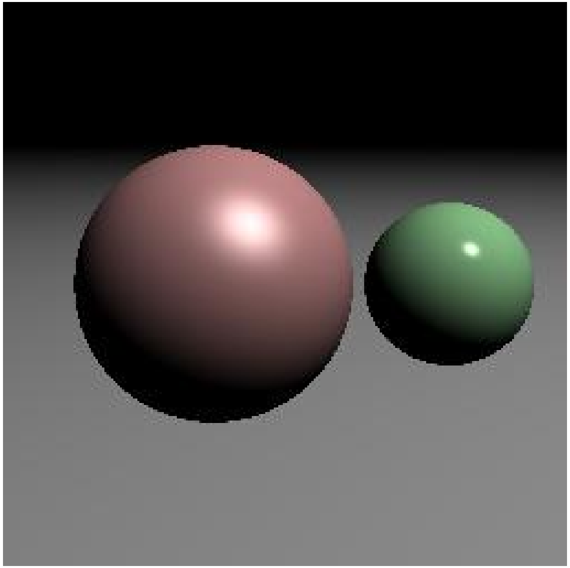


image so far



lighting

patterns of illumination in the environment

lighting

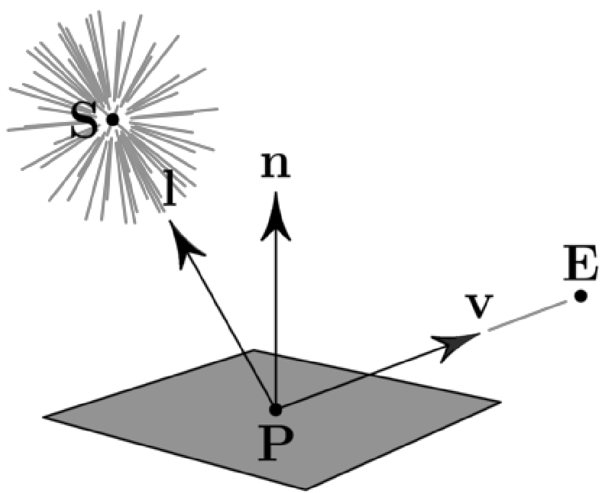
- determines how much light reaches a point
- depends on:
  - light geometry
  - light emission
  - scene geometry

light source models

- describe how light is emitted from light sources
- empirical light source models
    - point, directional, spot
- physically-based light source models
    - area light, sky model

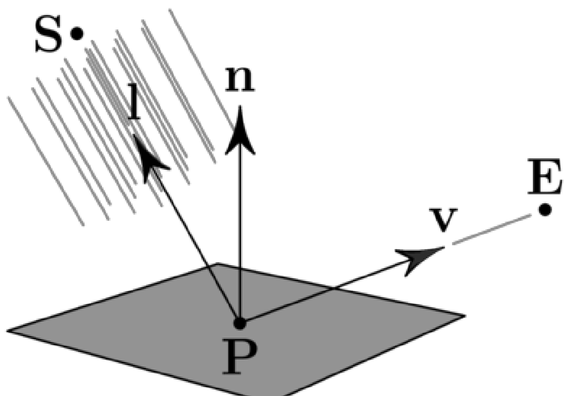
point lights

- light is emitted equally from a point $\mathbf{S}$ in all directions
- simulate local lighting, different at each surface point $\mathbf{P}$
- light direction: $\mathbf{l} = (\mathbf{S} - \mathbf{P})/|\mathbf{S} - \mathbf{P}|$
- light color: $L = k_l/|\mathbf{S} - \mathbf{P}|^2$



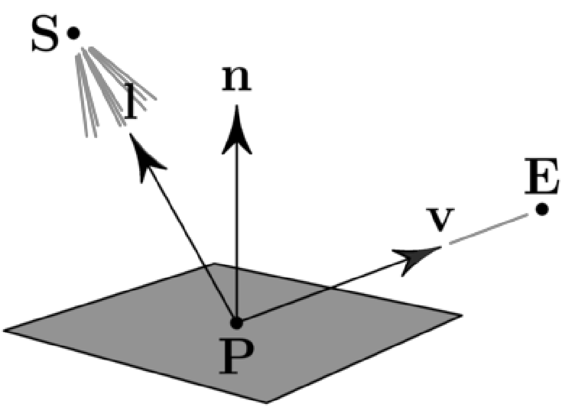
directional lights

- light is emitted from infinity in one direction $\mathbf{d}$
- simulate distant lighting, e.g. sun, same at all surface points $\mathbf{P}$
- light direction: $\mathbf{l} = \mathbf{d}$
- light color: $L = k_l$



spot lights

- same as points lights, but only emits in a cone around $\mathbf{d}$
- simulate theatrical lights
- cone falloff model arbitrary
- light direction: $\mathbf{l} = (\mathbf{S} - \mathbf{P}) / |\mathbf{S} - \mathbf{P}|$
- light color: $L = k_l \cdot attenutation / |\mathbf{S} - \mathbf{P}|^2$



shading model with multiple lights

- add contribution of all lights i for diffuse and specular

$$C = \sum_i L_i \cdot \big(\rho_d(\mathbf{l}_i, \mathbf{v}; \mathbf{f}) + \rho_s(\mathbf{l}_i, \mathbf{v}; \mathbf{f})\big) \cdot |\mathbf{n} \cdot \mathbf{l}_i|$$

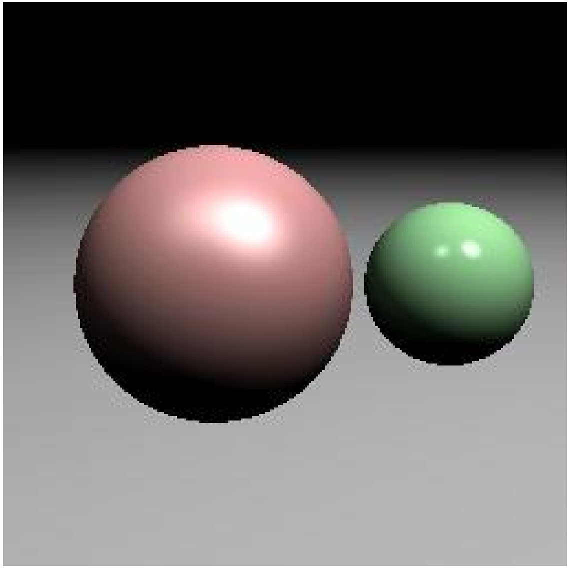
- for Lambert and Phong

$$C = \sum_i L_i \cdot \big(k_d + k_s \max(0, \mathbf{v} \cdot \mathbf{r}_i)^n\big) \cdot |\mathbf{n} \cdot \mathbf{l}_i|$$

- for Lambert and Blinn

$$C = \sum_i L_i \cdot \big(k_d + k_s \max(0, \mathbf{n} \cdot \mathbf{h}_i)^n\big) \cdot |\mathbf{n} \cdot \mathbf{l}_i|$$

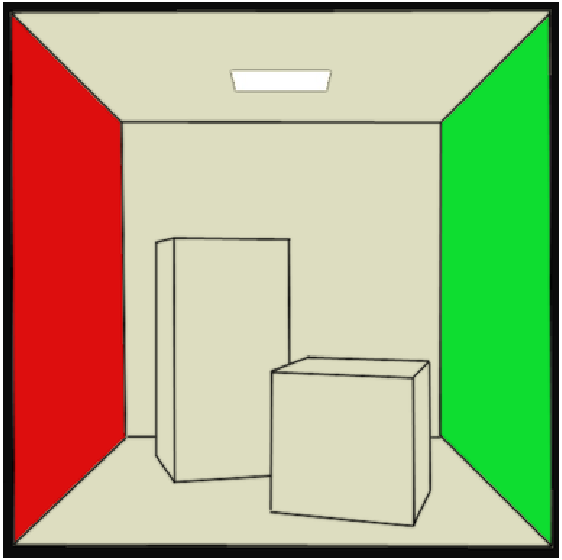
image so far



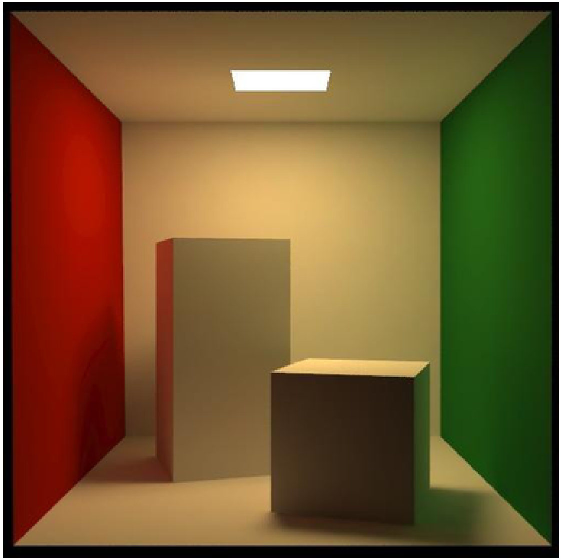
illumination models

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

illumination models



[PCG]



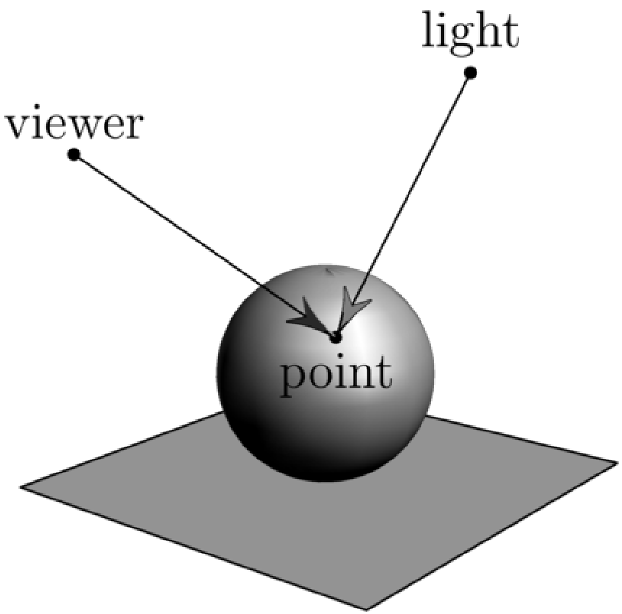
ray tracing lighting model

- point/directional/spot light sources
- sharp shadows
- sharp reflection/refractions
- hacked diffuse inter-reflection: ambient term

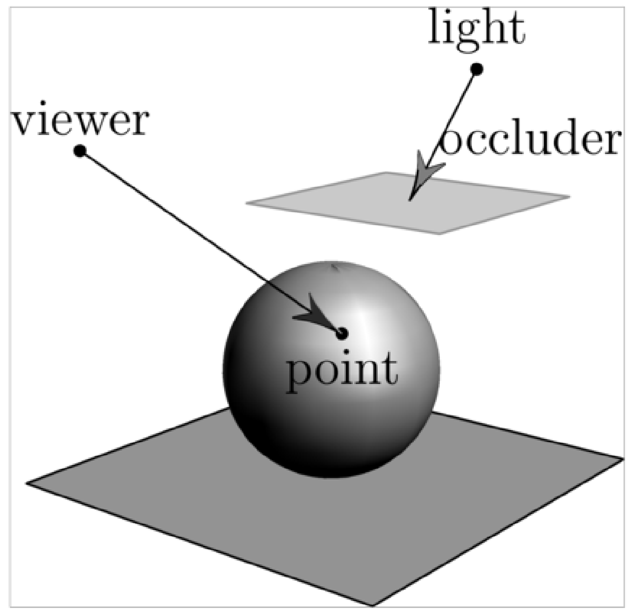
ray traced shadows

- light contributes only if visible at surface point

no shadow

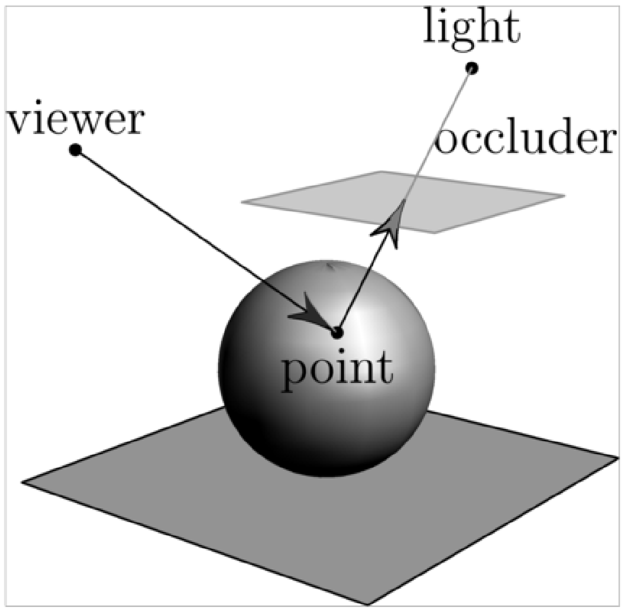


shadow



ray traced shadows

- send a *shadow* ray to check if light is visible
- visible if no hits or if t more than light distance



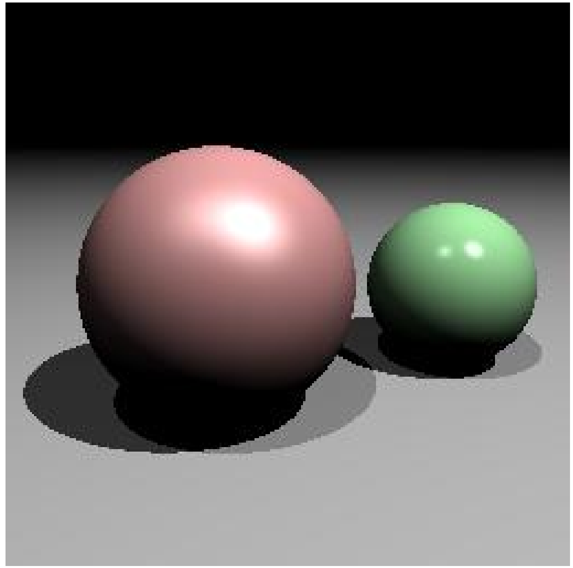
ray traced shadows

- shadow ray $\mathbf{r} = \mathbf{P} + t\mathbf{l}$ with $t \in (t_{min}, t_{max})$
  - spot/point lights at $\mathbf{S}$: $t_{max} = length(\mathbf{S} - \mathbf{P})$
  - directional lights: $t_{max} = \infty$
- scale lighting by visibility term $V_i(\mathbf{P})$ which is 0 or 1

$$C = \sum_i L_i \cdot V_i(\mathbf{P})(\rho_d + \rho_s)|\mathbf{n} \cdot \mathbf{l}_i|$$

- implementation detail: numerical precision
  - shadow acne: ray hits the visible point
  - solution: only intersect if $t > \epsilon$, i.e. $t_{min} = \epsilon$

image so far



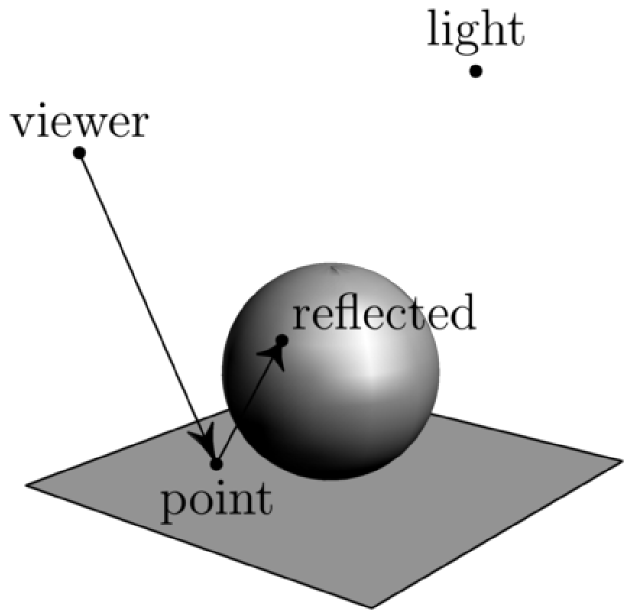
ambient term hack

- 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

$$C = k_d L_a + \sum_i L_i \cdot V_i(\mathbf{P})(\rho_d + \rho_s)|\mathbf{n} \cdot \mathbf{l}_i|$$

ray traced reflections and refractions

- perfectly shiny surfaces reflects objects
- recursively trace a ray if material is reflective or refractive



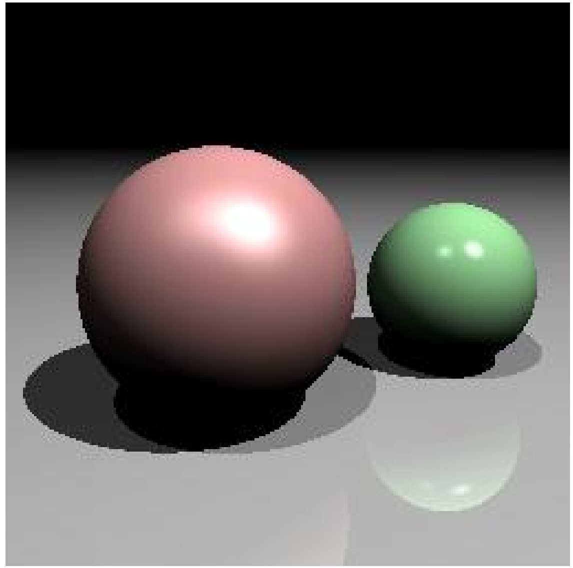
ray traced reflections and refractions

- reflections: along mirror direction $\mathbf{r} = -\mathbf{l} + 2(\mathbf{l} \cdot \mathbf{n})\mathbf{n}$, scaled by k_r
- refractions: along refraction direction scaled by k_t

$$C = k_d L_a + \sum_i L_i \cdot V_i(\mathbf{P})(\rho_d + \rho_s)|\mathbf{n} \cdot \mathbf{l}_i| + \\ + k_r \ \mathrm{raytrace}(\mathbf{P}, \mathbf{r}) + k_t \ \mathrm{raytrace}(\mathbf{P}, \mathbf{t})$$

- implementation detail: recursion
  - avoid hitting visible point: $t_{min} > \epsilon$
  - make sure you do not recurse indefinitely

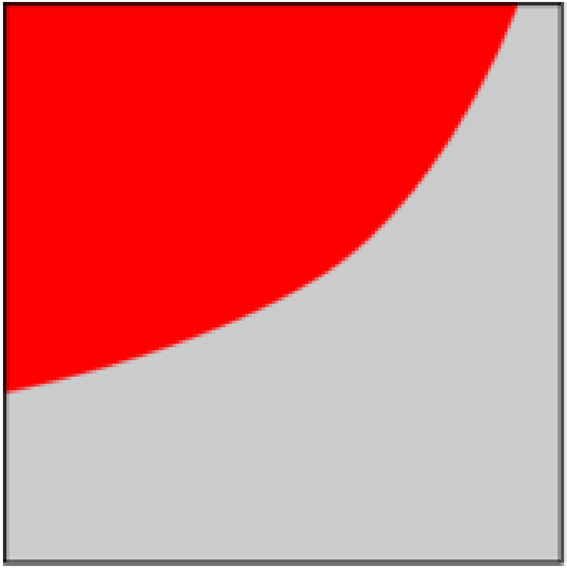
image so far



antialiasing

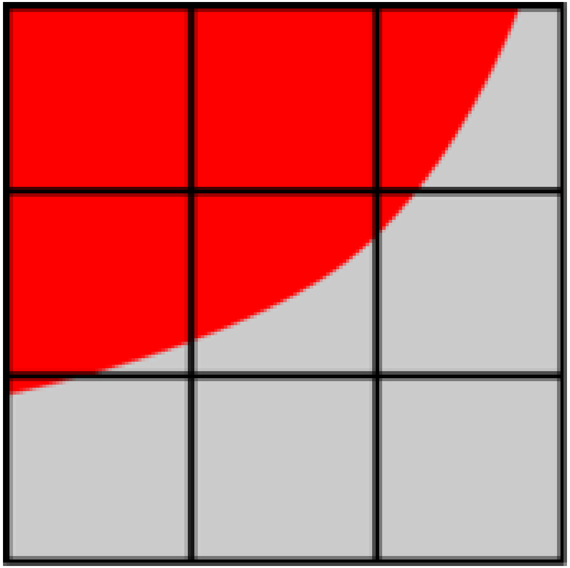
antialiasing: removing jaggies

1 sample/pixel



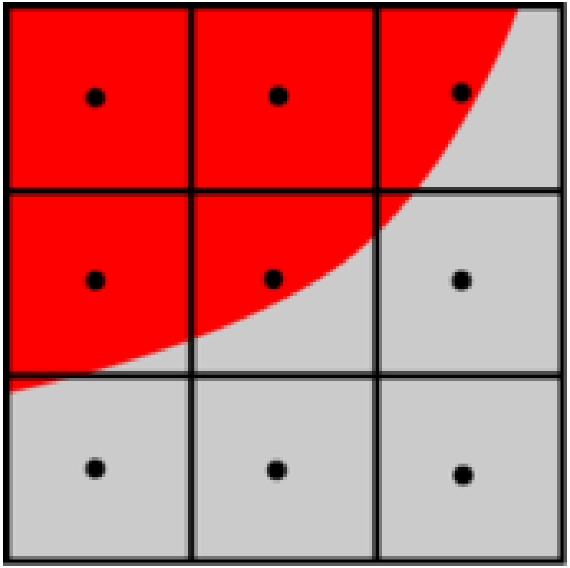
antialiasing: removing jaggies

1 sample/pixel



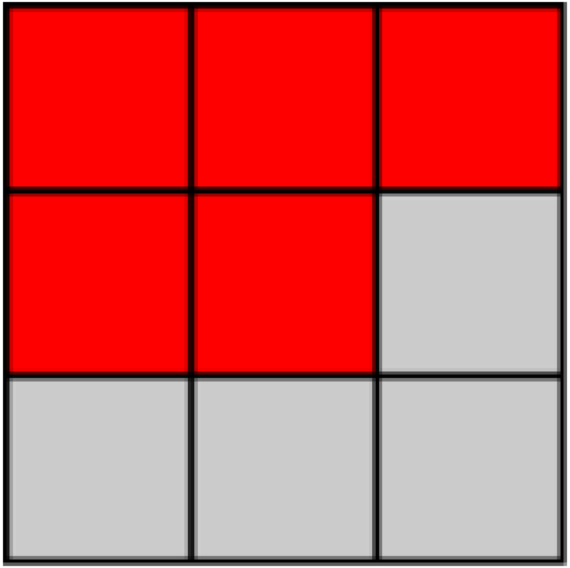
antialiasing: removing jaggies

1 sample/pixel



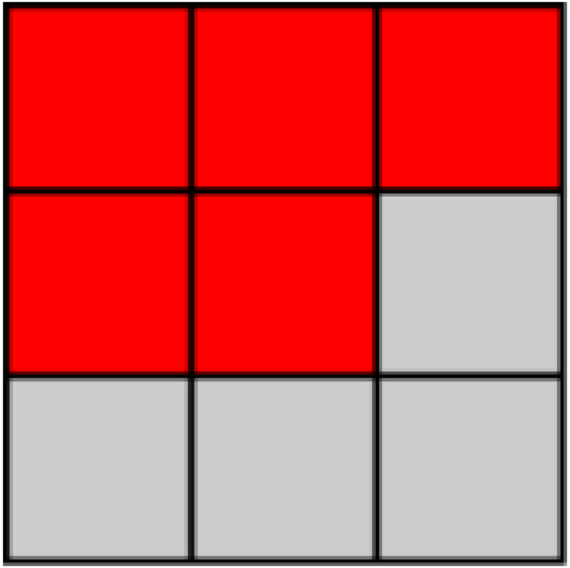
antialiasing: removing jaggies

1 sample/pixel

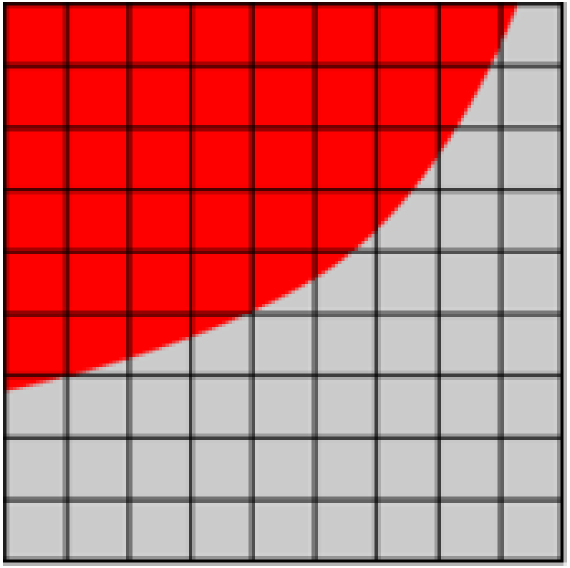


antialiasing: removing jaggies

1 sample/pixel

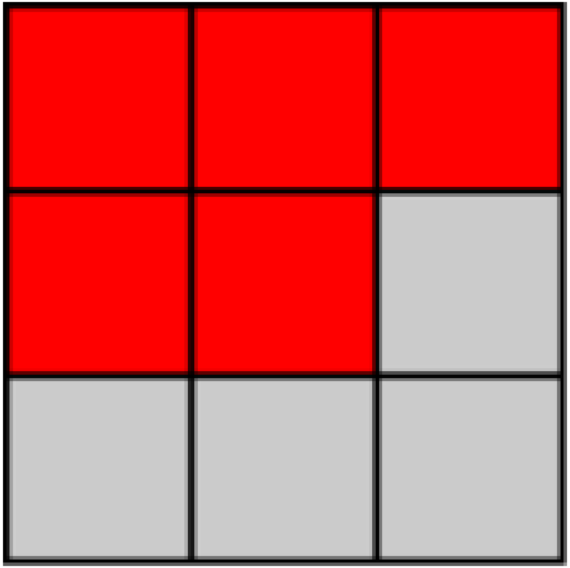


9 sample/pixel

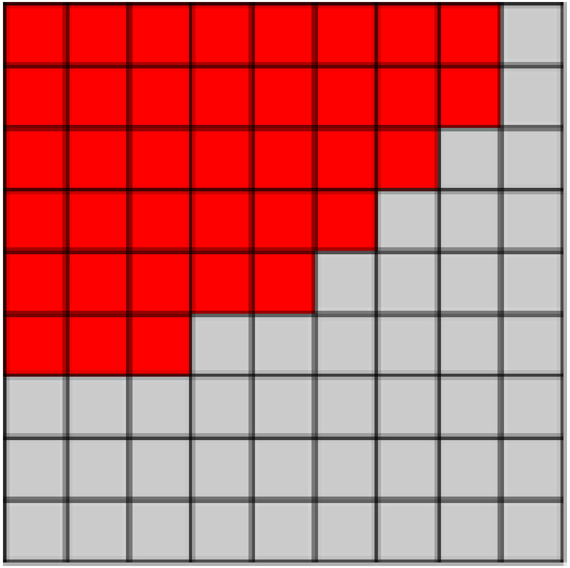


antialiasing: removing jaggies

1 sample/pixel

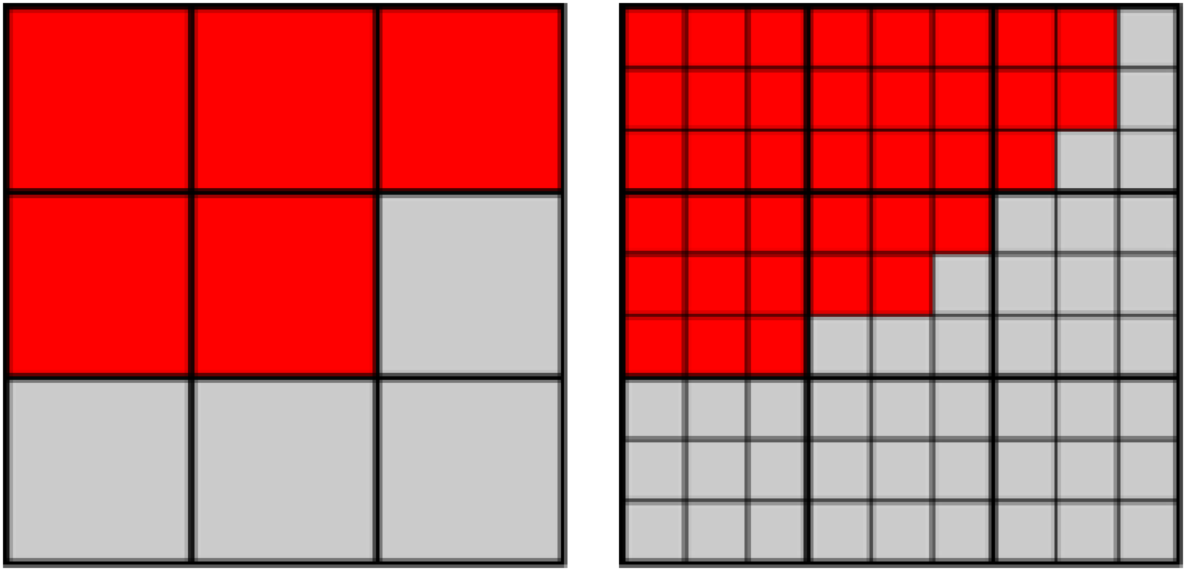


9 sample/pixel



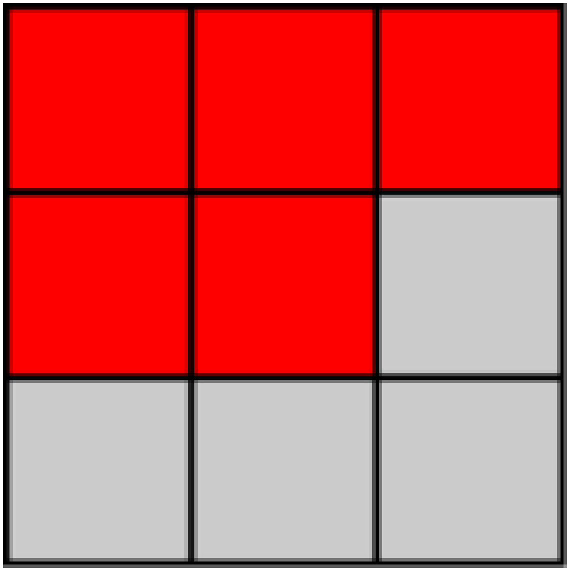
antialiasing: removing jaggies

1 sample/pixel　　9 sample/pixel

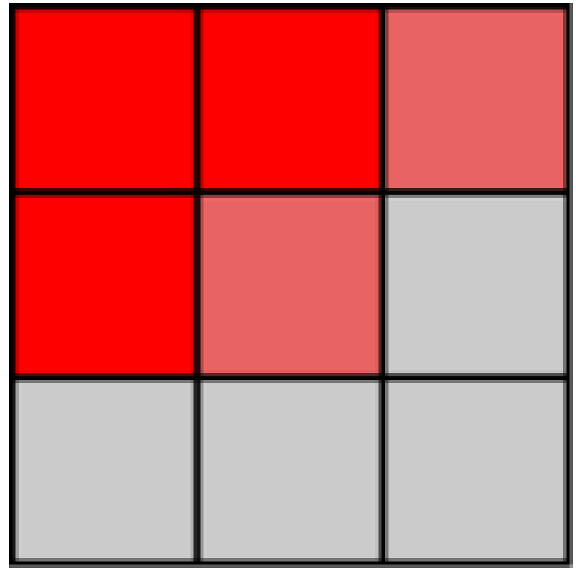


antialiasing: removing jaggies

1 sample/pixel



9 sample/pixel



antialiasing: removing jaggies

poor-man antialiasing:

- for each pixel
  - take multiple samples
  - compute average

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;
  }
}
```

anti-aliased 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 ++) {
        u = (i+(ii+0.5)/numberOfSamples)/imageWidth;
        v = (j+(jj+0.5)/numberofSamples)/imageHeight;
        ray = camera.generateRay(u,v);
        c += computeColor(ray);
      }
    }
    image[i][j] = c / (numberOfSamples^2);
  }
}
```

image so far