Ray Tracing

Image formation

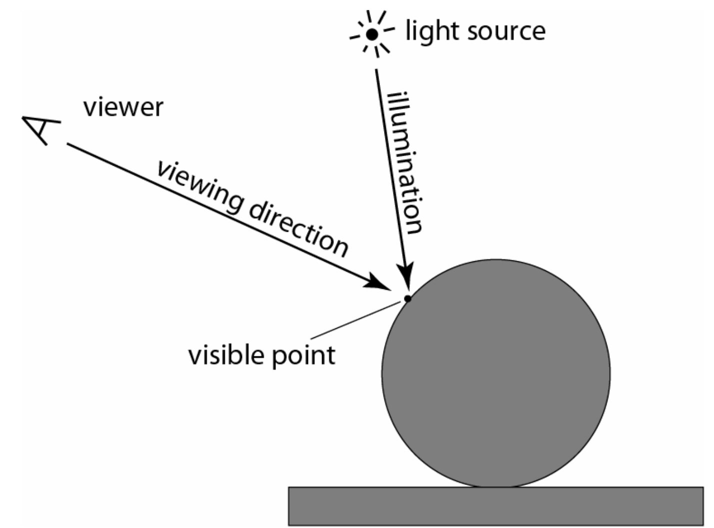
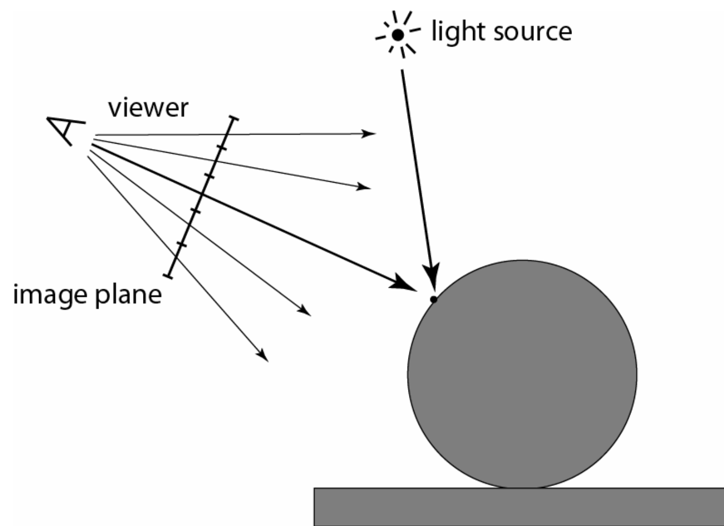


Image formation

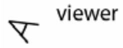


Rendering

Computational simulation of image formation.

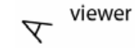
Rendering

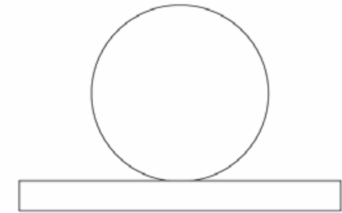
- Given viewer



Rendering

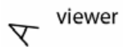
- Given viewer, geometry

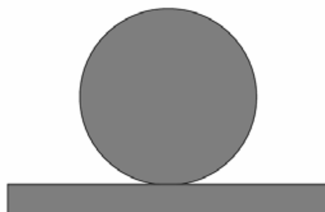




Rendering

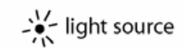
- Given viewer, geometry, materials

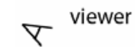


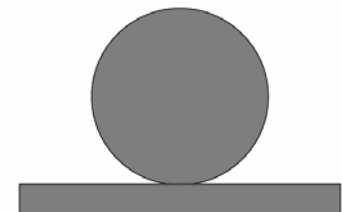


Rendering

- Given viewer, geometry, materials and lights

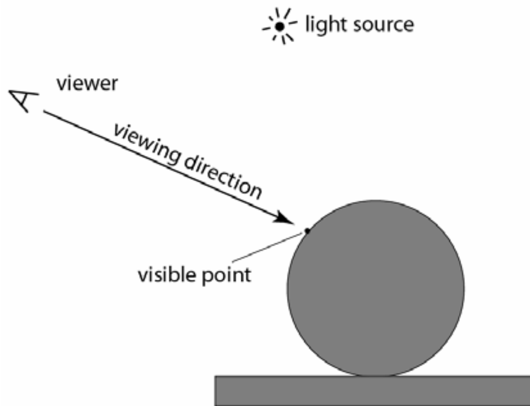






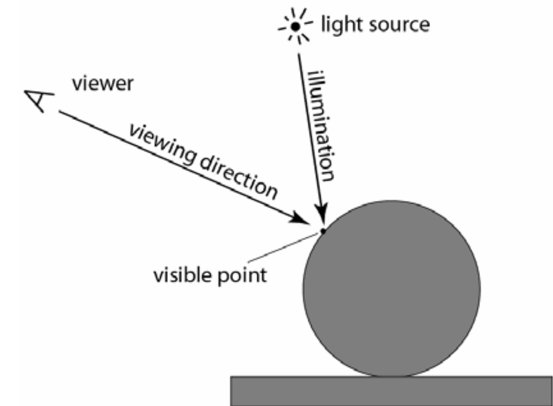
Rendering

- Given viewer, geometry, materials and lights
- Determine visibility



Rendering

- Given viewer, geometry, materials and lights
- Determine visibility and simulate lighting

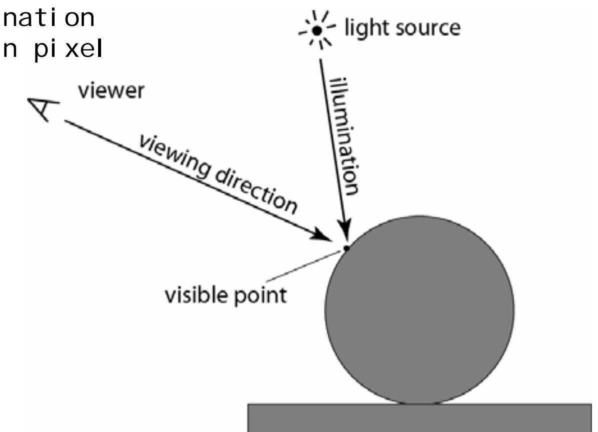


Ray Tracing

A particular rendering algorithm

Ray Tracing Algorithm

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

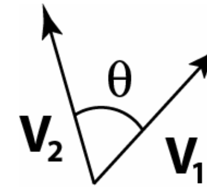


Vector math review – Points and Vectors

- Points
  - location in 3D space
  - P = (x,y,z)
- Vectors
  - direction and magnitude
  - V = (u,v,w)

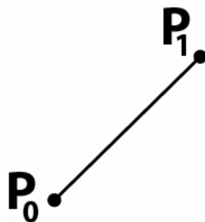
Vector math review – Vector operations

- Dot product
  - $V_1 \cdot V_2 = |V_1||V_2|\cos(\theta)$
- Cross product
  - $|V_1 \times V_2| = |V_1||V_2|\sin(\theta)$
  - $V_1 \times V_2$ is perpendicular to V_1 and V_2



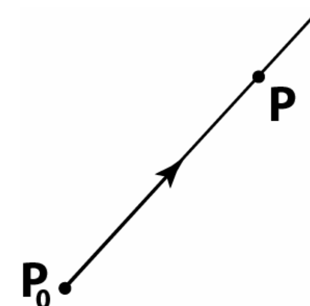
Vector math review – Segment and Rays

- Segment
  - set of points (line) between two points
  - $P(t) = P_0 + t\,(P_1 - P_0)$ with $t \in [0,1]$



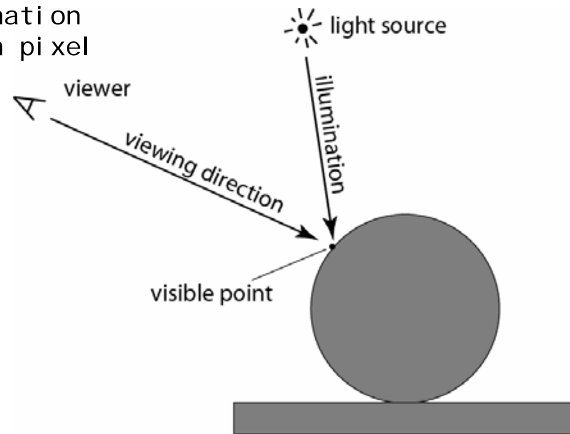
Vector math review – Segment and Rays

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



Ray Tracing Algorithm

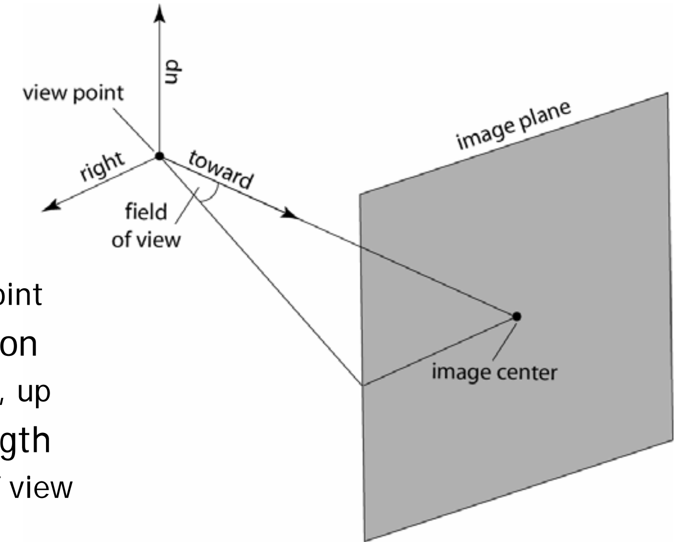
```
for each pixel {
  determine viewing direction
  intersect ray with scene
  compute illumination
  store result in pixel
}
```



© 2005 Fabio Pellacini

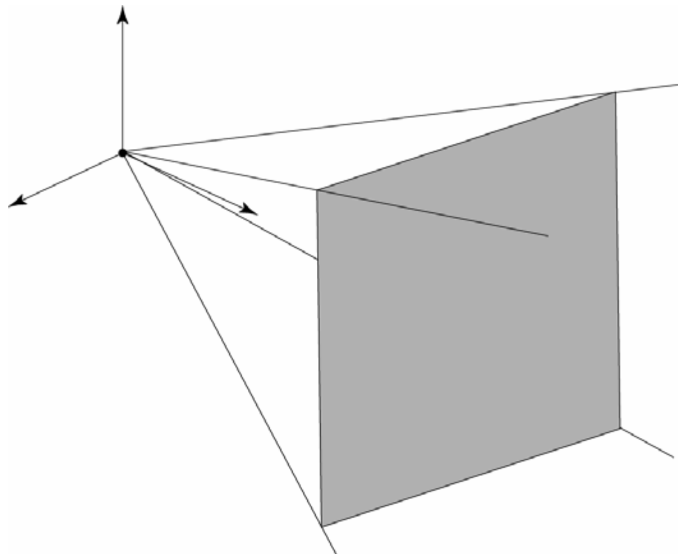
Viewer model - Representation

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



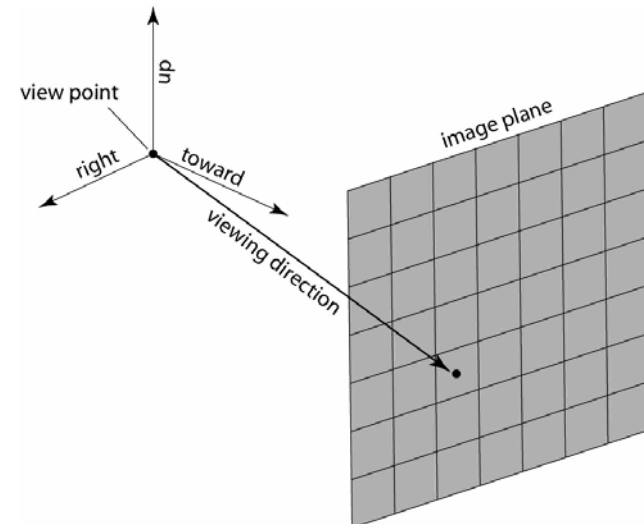
© 2005 Fabio Pellacini

Viewer model - View frustum



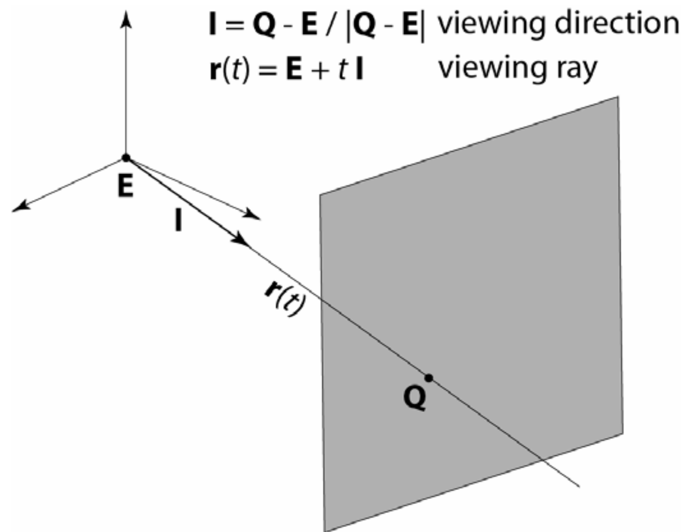
© 2005 Fabio Pellacini

Generate viewing rays

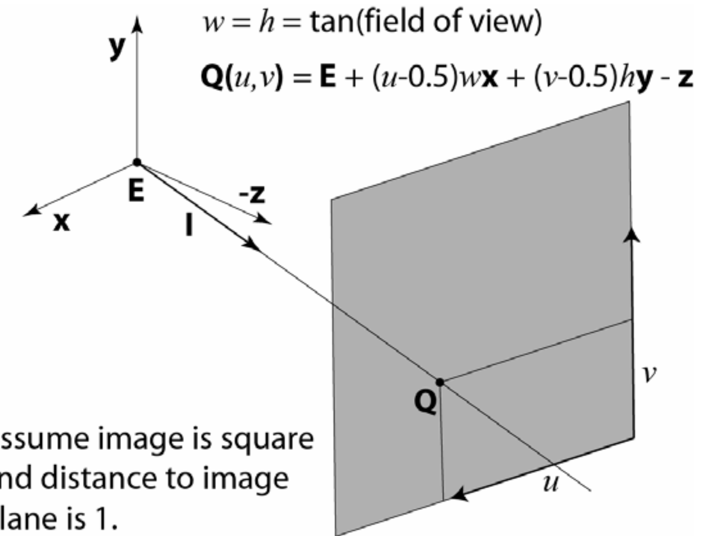


© 2005 Fabio Pellacini

Generate viewing rays

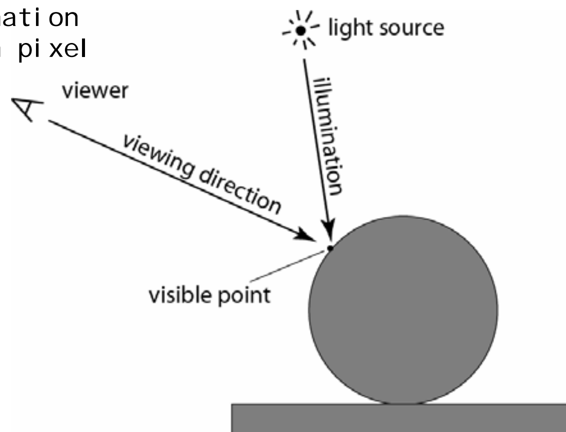


Generate viewing rays



Ray Tracing Algorithm

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



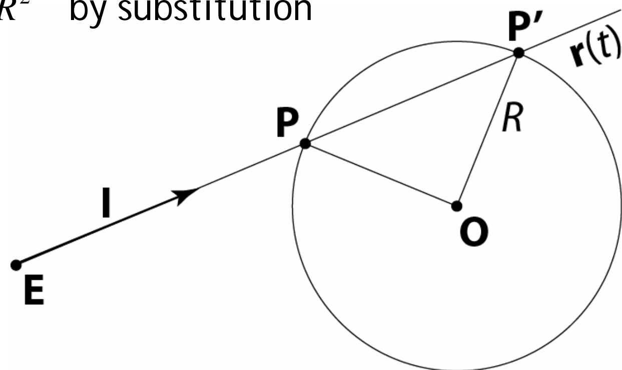
Geometry model

- Simple shapes
  - Spheres
  - Triangles
  - etc.
- Complex shapes
  - Later in the course

Ray-Sphere intersection - Algebraic

$$\begin{cases} \mathbf{P}(t) = \mathbf{E} + t\mathbf{I} & \text{point on ray} \\ |\mathbf{P}(t) - \mathbf{O}|^2 = R^2 & \text{point on sphere} \end{cases}$$

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



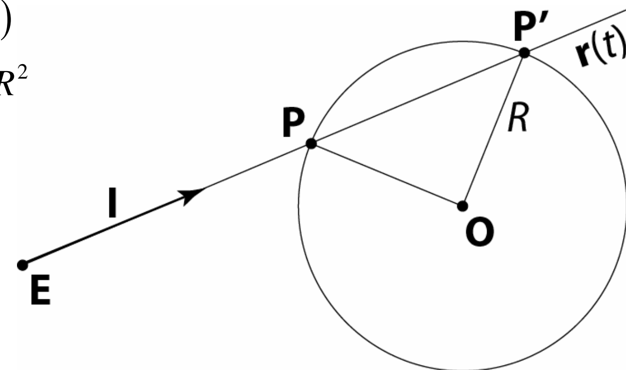
Ray-Sphere intersection - Algebraic

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

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

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

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



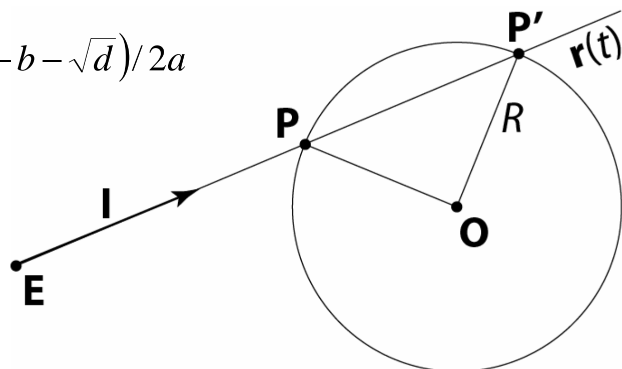
Ray-Sphere intersection - Algebraic

$d = b^2 - 4ac$ determinant

if $d < 0$ no solutions

otherwise

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

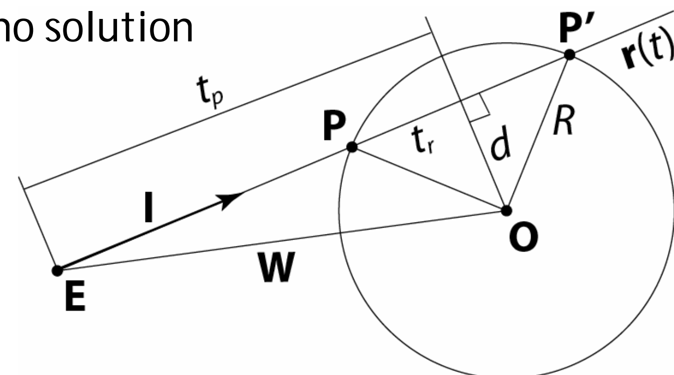


Ray-Sphere intersection - Geometric

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

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

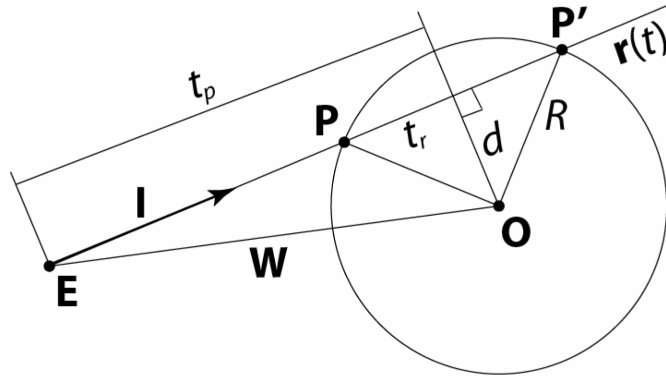
if $t_p < 0$ no solution



Ray-Sphere intersection - Geometric

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

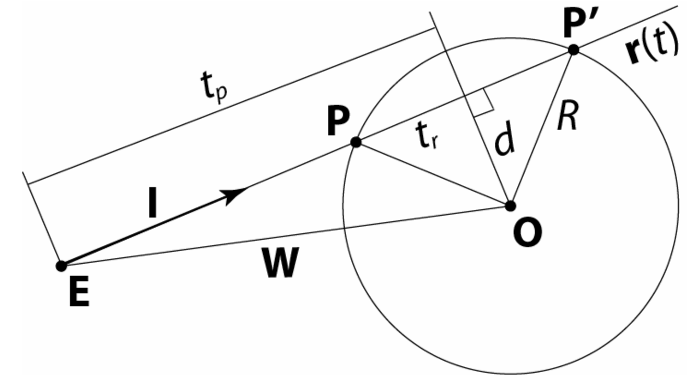
if $d^2 > R^2$ no solution



Ray-Sphere intersection - Geometric

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

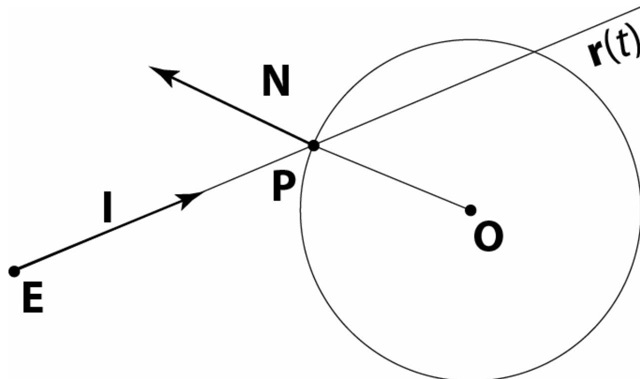
$\mathbf{P}(t) \Leftrightarrow t_- = t_p - t_r$



Ray-Sphere intersection - Normal

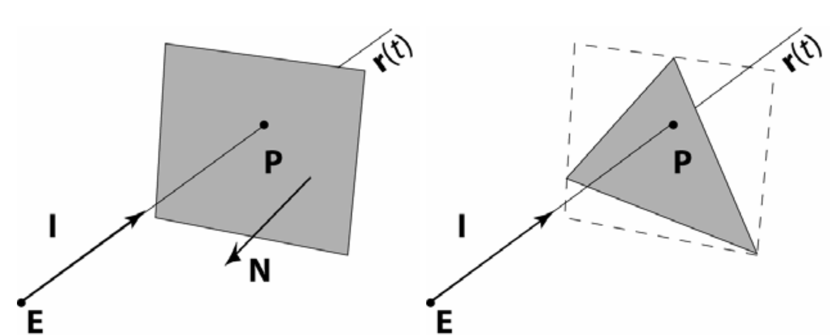
- Surface normal need for lighting computation

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



Ray-Triangle intersection

- Intersect with a plane
- Check if the intersection point is in triangle

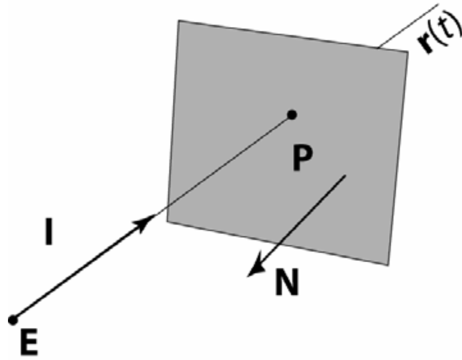


Ray-Plane intersection - Algebraic

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

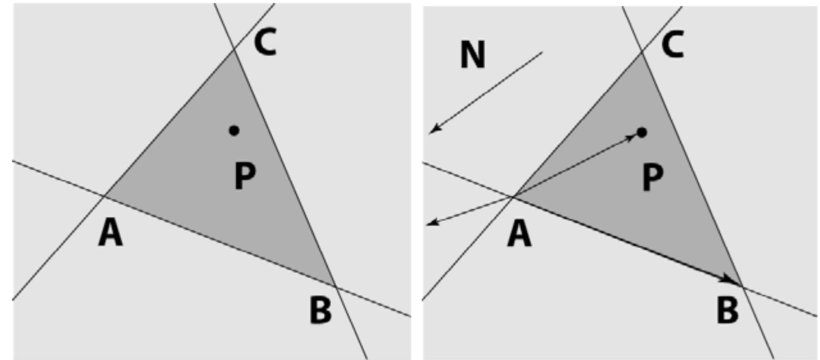
$$(\mathbf{E} + t\mathbf{I})\cdot\mathbf{N} + d = 0 \quad \text{by substitution}$$

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



Ray-Triangle intersection - Inside test

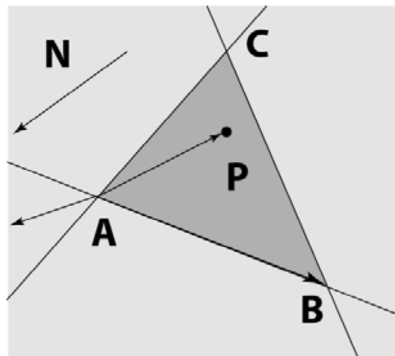
- Triangle is intersection of 3 half-spaces
- Check if P is on right side by comparing "normals"



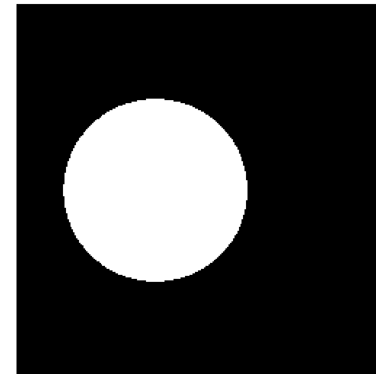
Ray-Triangle intersection - Inside test

intersects if

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

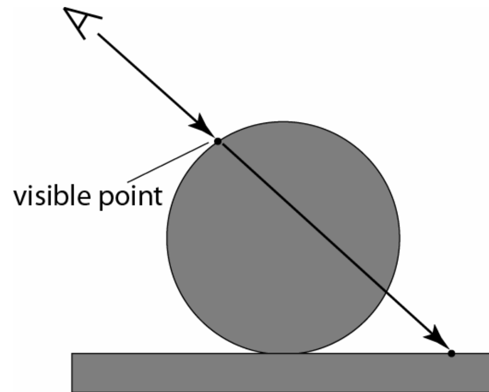


Images so far



Intersecting many shapes

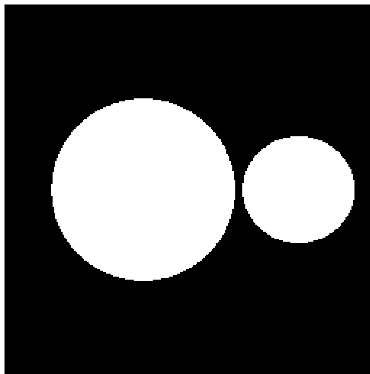
- Intersect each primitive
- Pick closest intersection



Intersecting many shapes - pseudocode

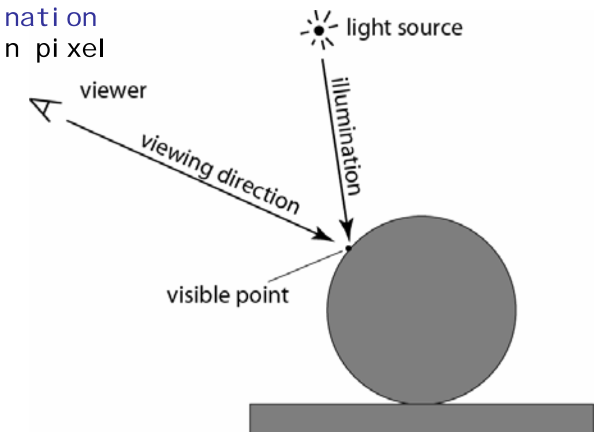
```
minDistance = infinity
hit = false
foreach surface s {
  if(s.intersect(ray,intersection)) {
    if(intersection.distance < minDistance) {
      hit = true;
      minDistance = intersection.distance;
    }
  }
}
```


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

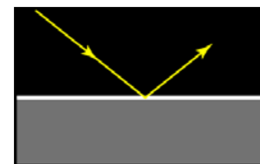
Patterns of illumination in the environment

Shading

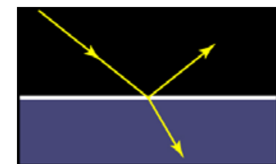
- compute reflected light
- depends on
  - viewer position
  - incoming light, i.e. lighting
  - surface geometry
  - surface material
- more on this later

Materials

metals



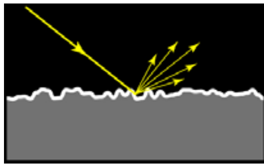
dielectric



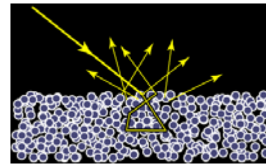
[Marschner 2004]

Materials

metals



dielectric



[Marschner 2004]

Shading models

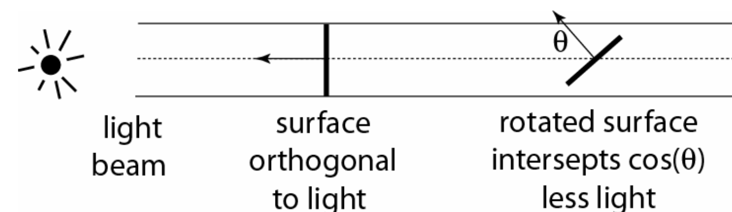
- empirical shading models
  - produce believable images
  - simple and efficient
  - only for simple materials
- physically-based shading models
  - can reproduce accurate effects
  - more complex
- will concentrate empirical plastic-like model
  - more on this later in the course

Phong shading model

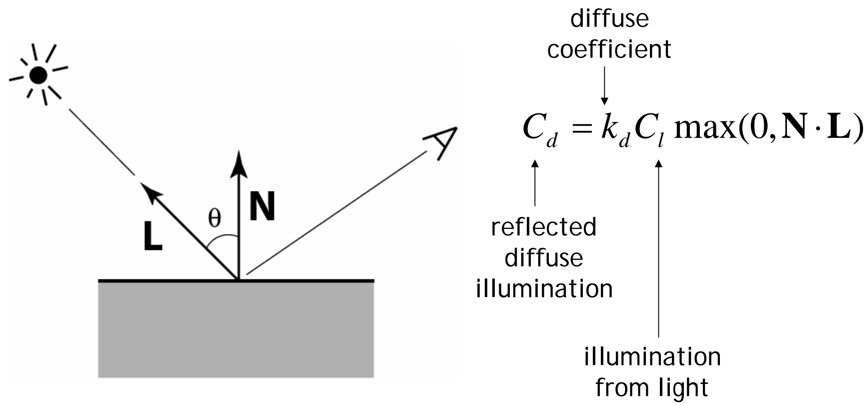
- shading model = diffuse + specular reflection
- diffuse reflection
  - light is reflected in every direction equally
  - colored by surface color
- specular reflection
  - light is reflected only around the mirror direction
  - white for plastic-like surfaces (glossy paints)
  - colored for metals (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

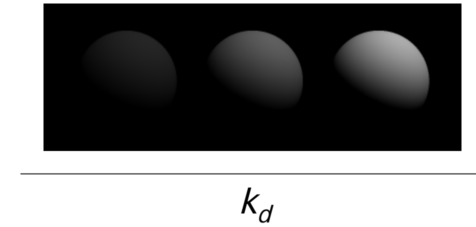


Lambertian shading model

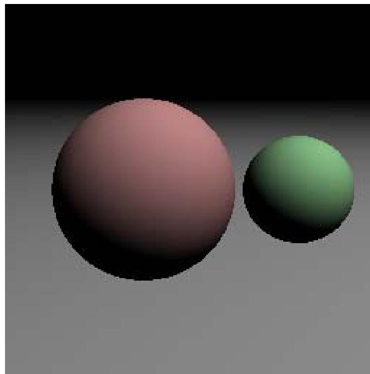


Lambertian shading model

- Produces matte appearance



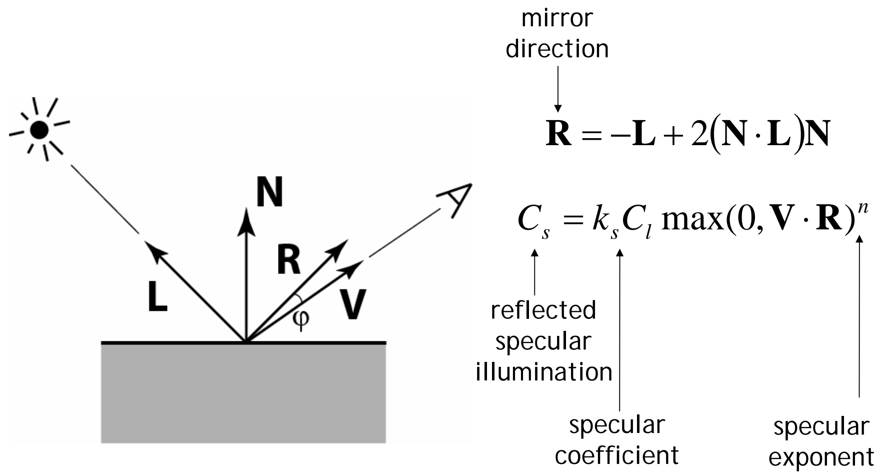
Images so far



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

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

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

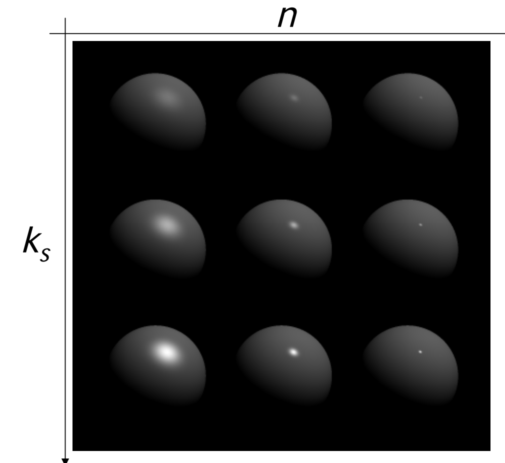
reflected specular illumination

specular coefficient

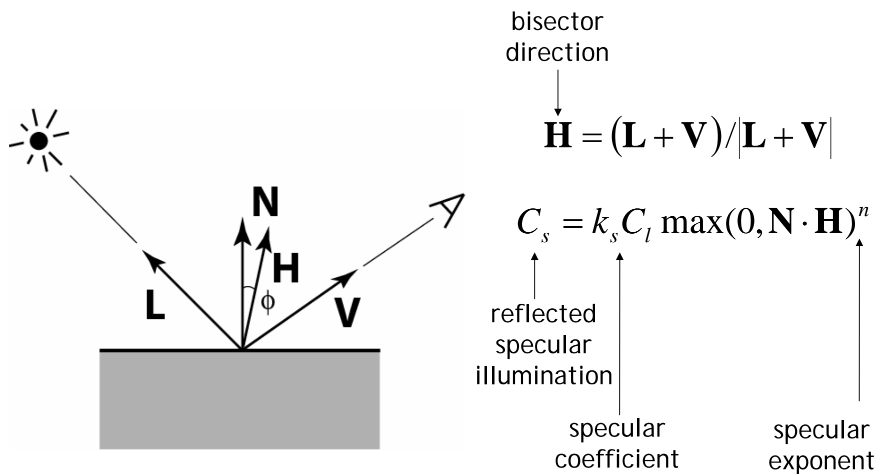
specular exponent

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

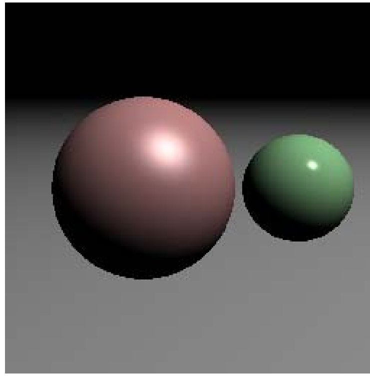
specular exponent

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

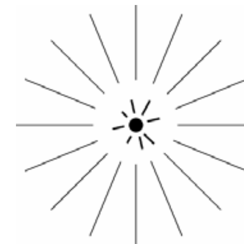
- determines how much light reaches a point
- depends on
  - light geometry
  - light emission
  - scene geometry

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

Light source models - Point lights

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

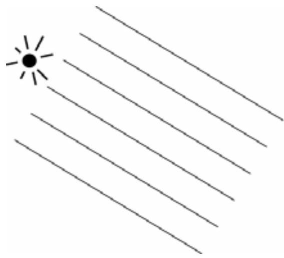


$$\mathbf{L} = \| \mathbf{S} - \mathbf{P} \|$$

$$C_l = C / r^2$$

Light source models - Directional lights

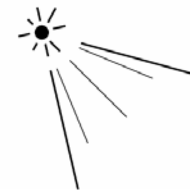
- light emitted from infinity in one direction
  - simulates distant lights, e.g. sun



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

Light source models - Spot lights

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



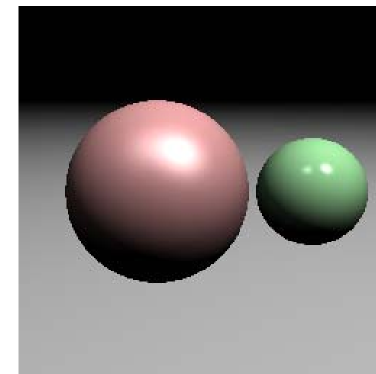
$$\mathbf{L} = \| \mathbf{S} - \mathbf{P} \|$$
$$C_l = C \cdot att / r^2$$

Multiple lights

- add contribution for each light

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

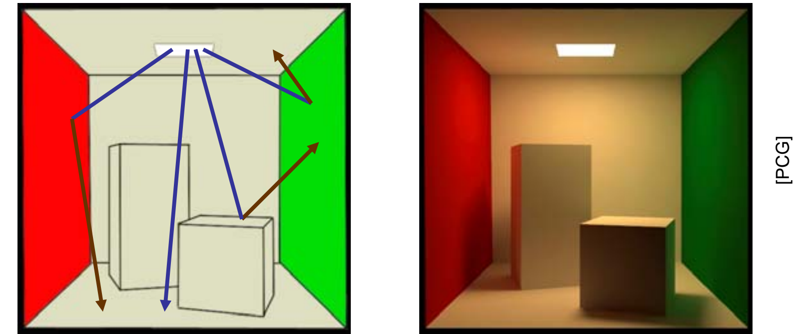
Image so far



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



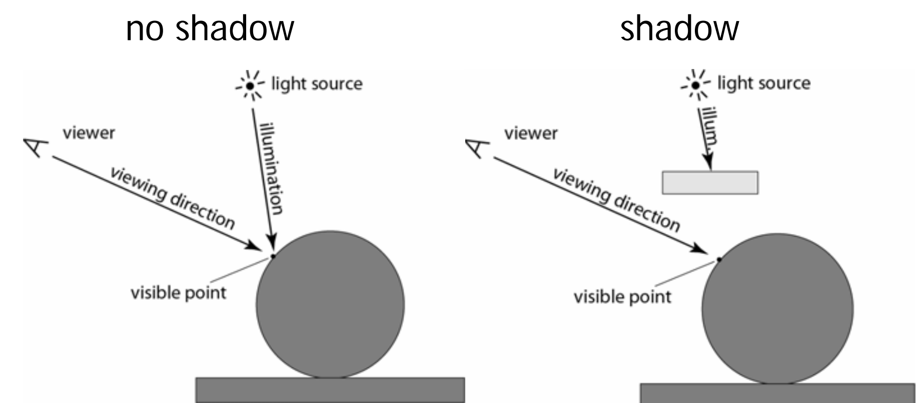
Ray Tracing lighting model

- 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

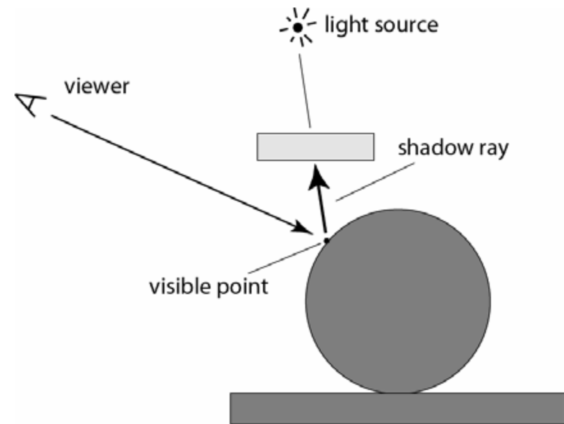
- light contributes only if visible at surface point

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

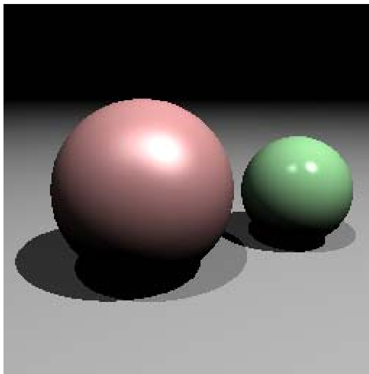
- scale light by a visibility term

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

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

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

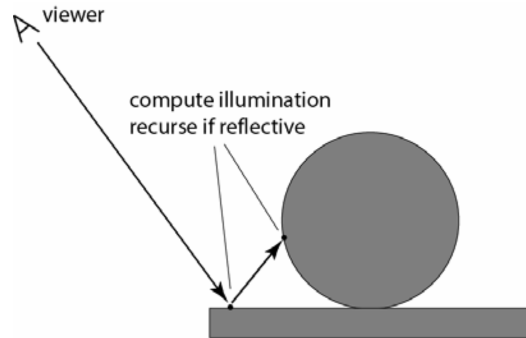
$$C = C_a + \sum_i \left[(C_d)_i + (C_s)_i \right] =$$

$$= k_a C_{amb} + \sum_i \left[(C_d)_i + (C_s)_i \right]$$

ambient coefficient — ambient illumination

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.

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

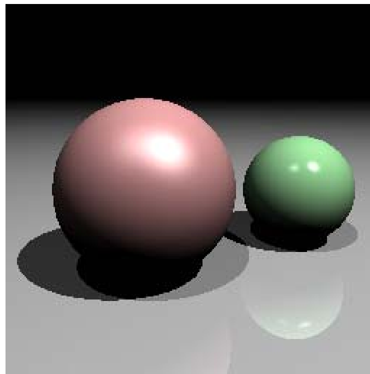
$$= C_a + \sum_i \left[(C_d)_i + (C_s)_i\right] +$$

$$+ k_r \, \text{raytrace}(\mathbf{P}, \mathbf{R}) + k_t \, \text{raytrace}(\mathbf{P}, \mathbf{T})$$

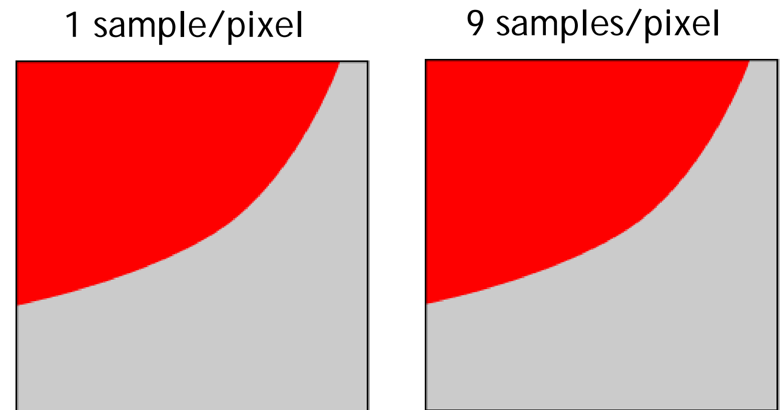
reflection coefficient (k_r)

refraction coefficient (k_t)

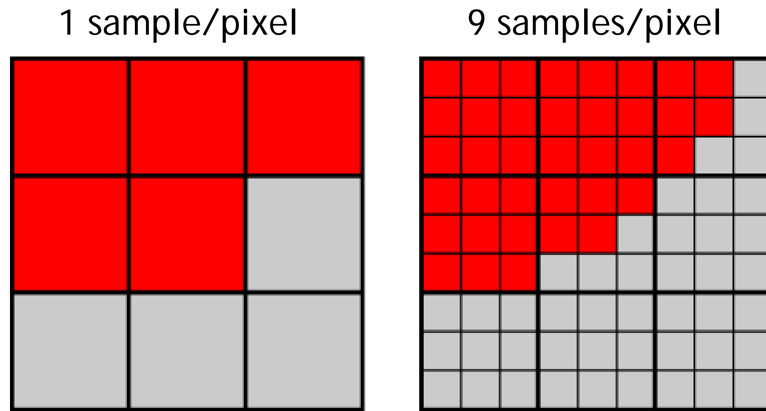
Images so far



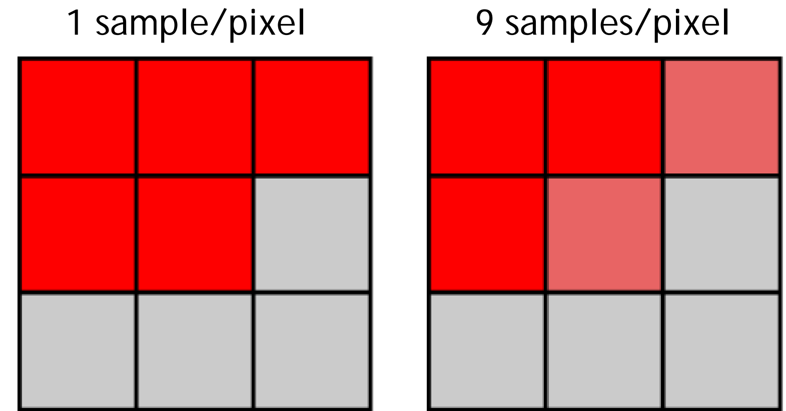
Antialiasing - Removing jaggies



Antialiasing - Removing jaggies



Antialiasing - Removing jaggies



Antialiasing - Removing jaggies

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

Ray Tracing Pseudocode

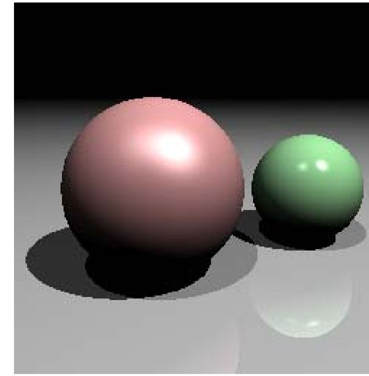
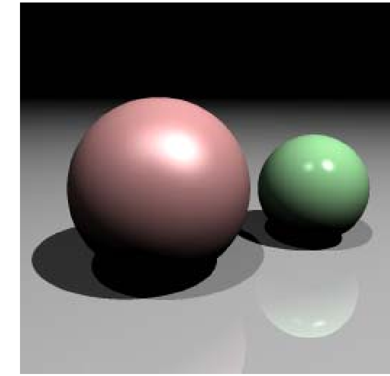
```
for(i = 0; i < imageWidth; i ++) {
  for(j = 0; j < imageHeight; j ++) {
    u = (i + 0.5)/imageWidth;
    v = (j + 0.5)/imageHeight;
    ray = camera.generateRay(u,v);
    c = computeColor(ray);
    image[i][j] = c;
  }
}
```

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);
  }
}
```


© 2005 Fabio Pellacini • 81

Images so far

© 2005 Fabio Pellacini • 82

Ray Tracing details

- Numerical precision issues can 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

© 2005 Fabio Pellacini • 83

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

© 2005 Fabio Pellacini • 84