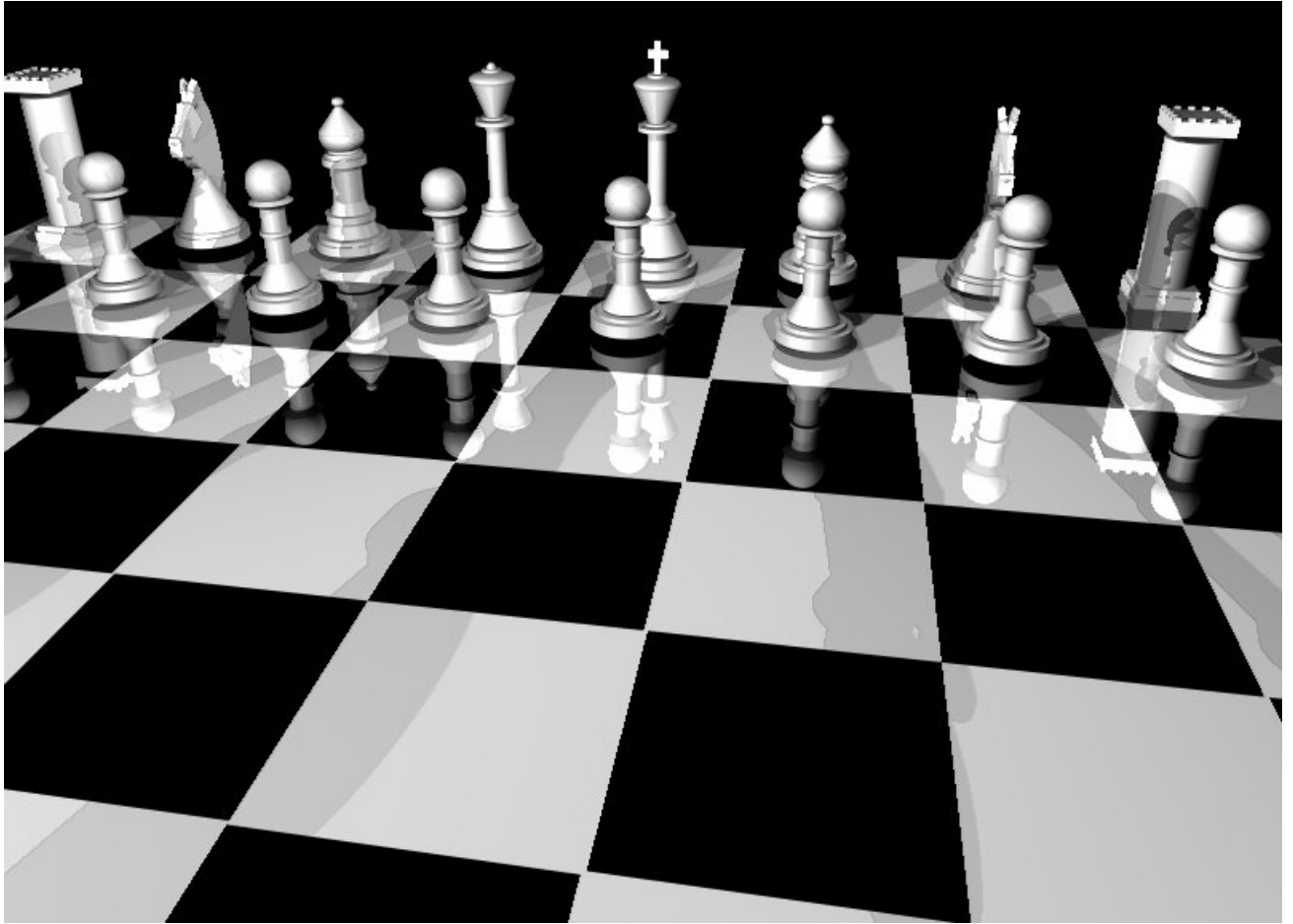


Raytracing Basics



scene rendered with Hari Khalsa's Sc.B. '05 CS123 raytracer

Significantly augmented

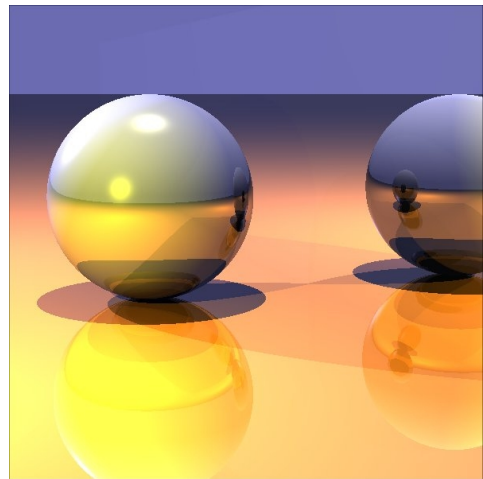
thanks to

Spike and John Alex

Illumination Review (1/3)

Key points

- Global, physically-based illumination models describe energy transport and radiation
 - subject to properties of light and materials
 - subject to geometry of light, objects and viewer
- For each surface, there is a distribution that characterizes its absorption and reflection at each wavelength
 - also some other properties that we ignore for brevity's sake
- All illumination models are, by definition, approximate
 - various rendering



Illumination Review (2/3)

Important illumination models

- Non-global (single-surface element) model

$$I_{\lambda} = \underbrace{I_a \lambda_{ka} O_{d\lambda}}_{\text{ambient}} + \sum_m f_{att} I_{p\lambda} \left[\underbrace{k_d O_{d\lambda} \vec{N} \cdot \vec{L}}_{\text{diffuse}} + \underbrace{k_s O_{s\lambda} (\vec{R} \cdot \vec{V})^n}_{\text{specular}} \right]$$

- ambient (constant) approximates indirect global illumination
- Lambertian diffuse reflection (outgoing energy proportional to cosine of angle between vector to light and surface normal)
- Phong approximation to specular reflection

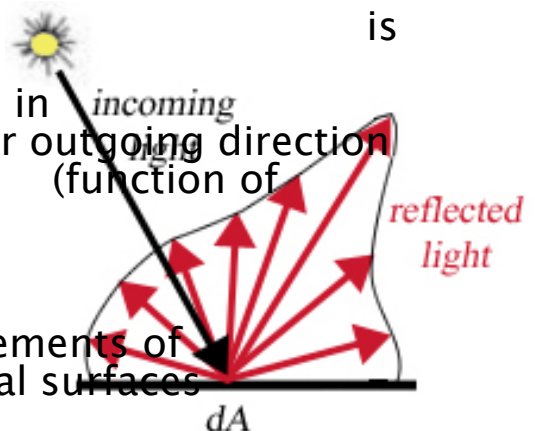
- In general, behavior of light is simulated by bi-directional reflectance distribution functions (BRDFs)

- Lambertian and Phong approximations model simple BRDF
- given incoming light ray, BRDF is used to calculate how much

light will be reflected in a particular outgoing direction (function of

incoming and outgoing angles)

- can be obtained from analytical model or measurements of actual surfaces



Illumination Review (3/3)

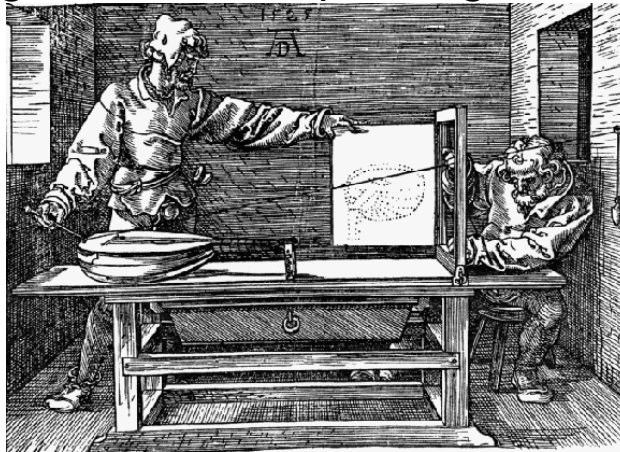
Important illumination models

- Global (multi-surface element) models
 - shadows
 - indirect illumination (from other objects)
 - specular and diffuse reflection, refraction
- Illumination models most commonly used today are tied to particulars of scan-conversion and raytracing
- Z-buffered polygon scan-conversion: non-global illumination
 - highly specialized and parallelized in hardware: fast, but more limited
 - specular reflections (only from planar surfaces)
 - hard shadows
 - shadow maps: render scene from light's point of view to create a map of points that are in shadow
 - shadow volumes: treat shadows cast by objects as polygonal volumes, points inside volume are in shadow
- Raytracing: global illumination
 - targeted shooting of rays (more accurate, but slower)
 - shadows, recursive specular (but not diffuse) reflection, refractive transmission

Rendering with Raytracing (1/2)

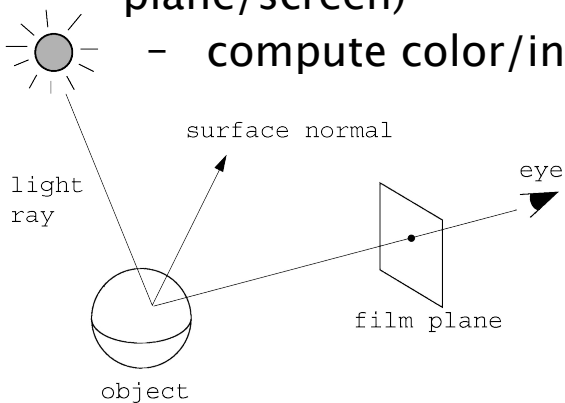
Simple idea: instead of forward mapping infinite number of rays from light source to object to viewer, backmap finite number of rays from viewer through each sample to object to light source (or other object)

- Generalizing from Durer's painting showing perspective projection



- Durer: Put eye at center of projection on wall and record string/plane intersection
- Raytracing: shoot rays from eye through sample point (e.g., a pixel center) of a virtual photo (the image or film plane/screen)

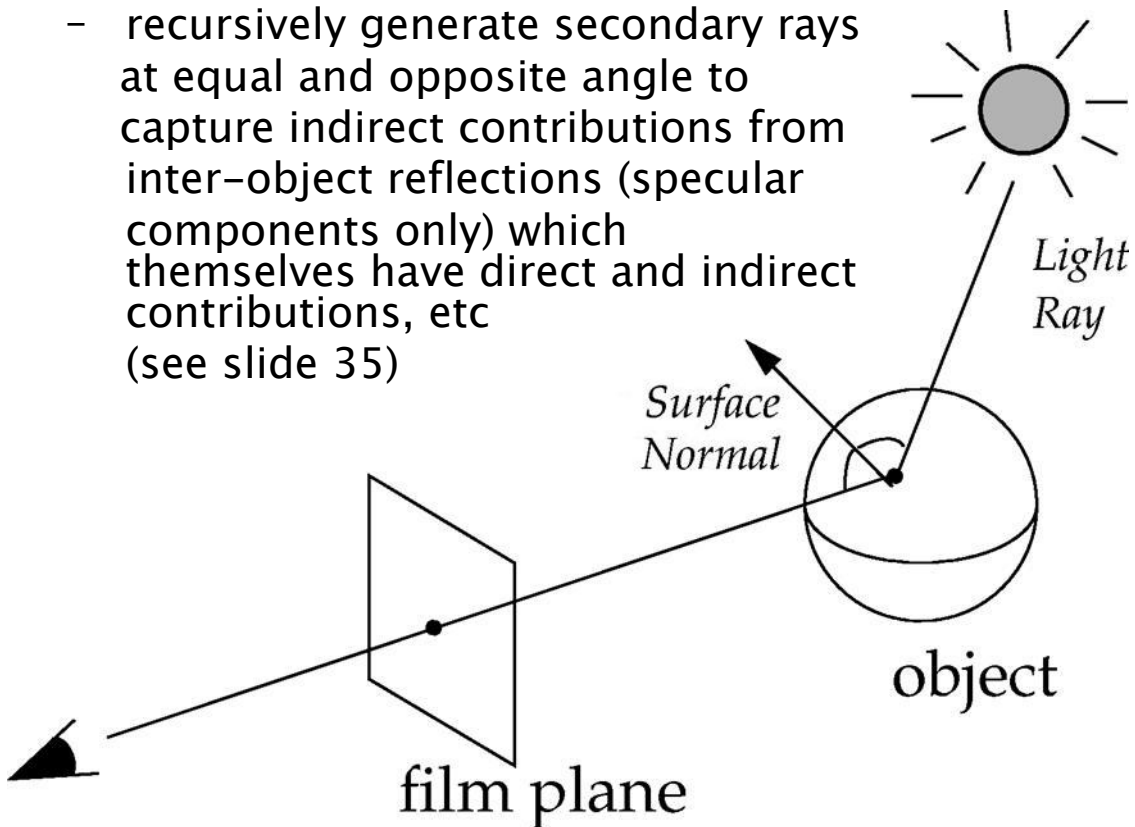
- compute color/intensity at the ray/photo intersection



Rendering with Raytracing (2/2)

Subproblems to solve

- Generate primary ('eye') ray
 - ray goes out from eye through a pixel center (or any other sample point on image plane)
- Find closest object along ray path
 - find first intersection between ray and an object in scene
- Light sample
 - use illumination model to determine direct contribution from light sources
 - recursively generate secondary rays at equal and opposite angle to capture indirect contributions from inter-object reflections (specular components only) which themselves have direct and indirect contributions, etc (see slide 35)

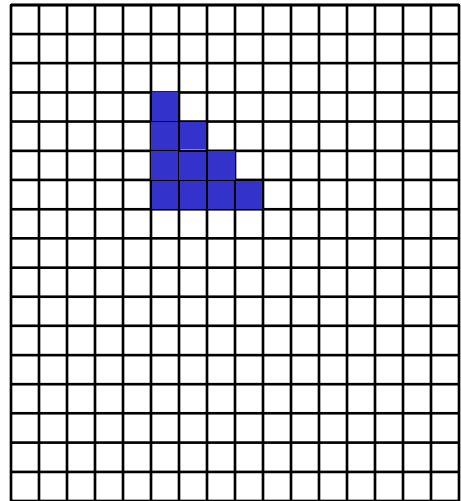
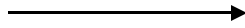
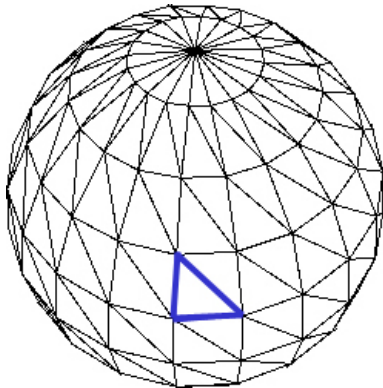


Raytracing

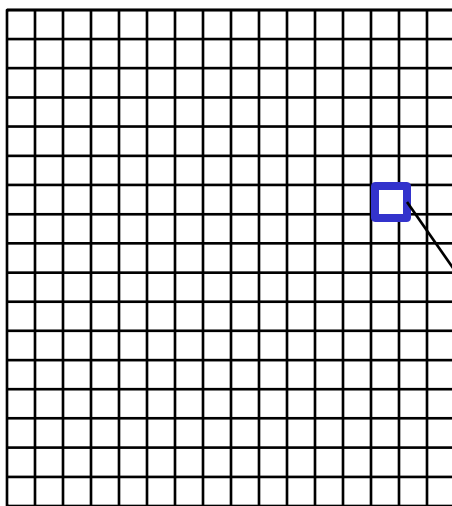
Raytracing versus scan conversion

scan conversion

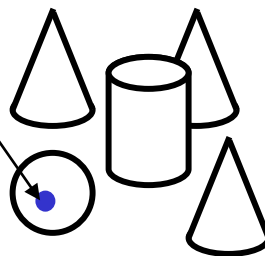
– After meshing and projection for each **triangle** in scene...



raytracing



for each **sample** in pixel image...



- Avoid meshing
- Explicit projection of triangles by working directly with analytical surfaces

Raytracing

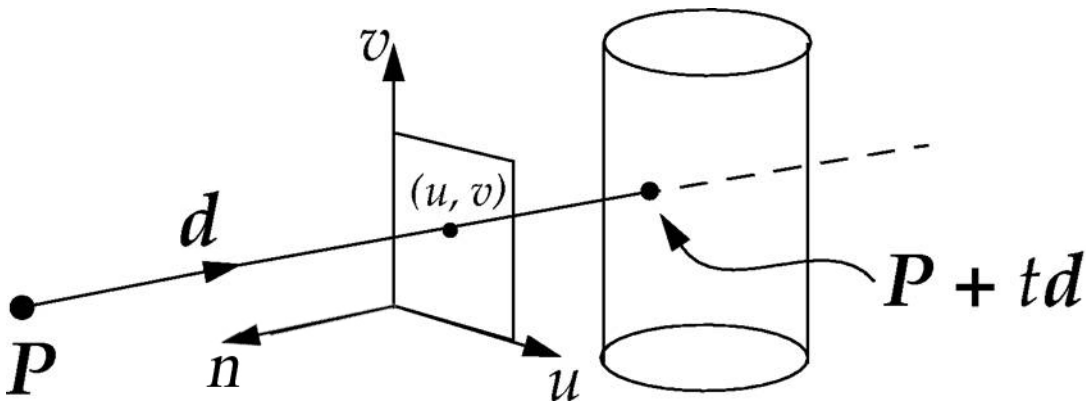
Raytracing versus scan conversion

- How is raytracing different from what you've been doing in your assignments? In Shapes, Sceneview you did:
 - for each object in scene
 - for each triangular facet of object
 - pass vertex geometry, colors to OpenGL, which paints all interior points of triangle in framebuffer
- This is fine if you're just using the simple hardware lighting model, and just using triangles
- We're trying to solve a different problem:
 - for each sample in our image
 - determine which object in scene is hit by ray through that sample;
 - paint that sample the color of the object at that point (accounting for lighting)

Generating Rays (1/4)

Ray origin

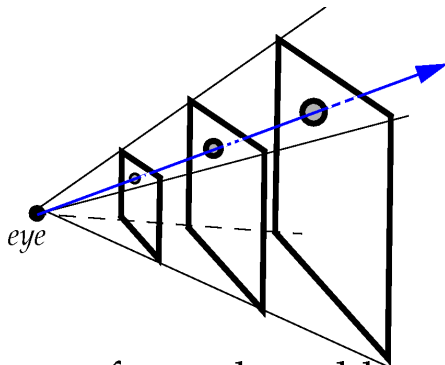
- Let's look at geometry of problem in *un-transformed* world-space (i.e. with perspective view volume)
 - we'll see why later
- Start a ray from an "eye point": P
- Send it out in some direction d from eye toward a point on film plane whose color we want to know
- Points along ray have form $P + td$ where
 - P is ray's base point: the camera's eye
 - d is unit vector direction of ray
 - t is a nonnegative real number
- "Eye point" is center of projection in perspective view volume (view frustum)
- Don't use de-perspectivizing 'unhinging' step; to avoid dealing with inverse of perspective transformation later



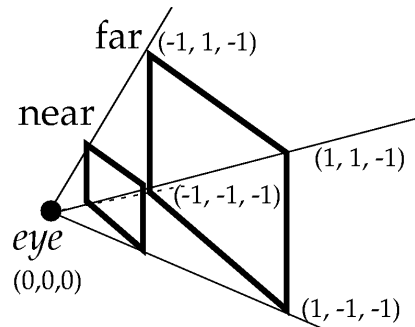
Generating Rays (2/4)

Ray direction

- Start with screen-space points (pixels). Need to find a point in 3D that lies on corresponding ray
 - we'll use ray to intersect with original objects in original, untransformed world coordinate system
- Transform 2D screen-space points into points on camera's film plane located in 3D space
- Any plane orthogonal to look vector is a convenient film plane: constant z in canonical view volume



untransformed world space



canonical view volume

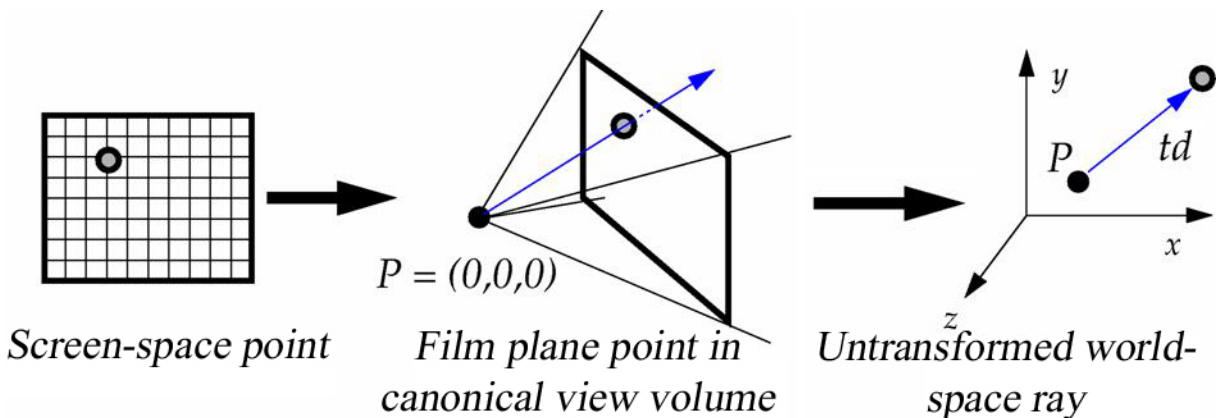
Any plane $z = k$, $-1 \leq k < 0$ can be the film plane

- Choose a plane to be the film plane and then create a function that maps screen-space points onto it
 - what's a convenient plane? Try the far plane :-)
 - to convert, we just have to scale integer screen-space coordinates into real values between -1 and 1

Generating Rays (3/4)

Ray direction(cont.)

- Transform film plane point into world-space point
 - we can make direction vector between eye (at CoP) and this world-space point
 - we need vector to be in world-space in order to intersect with original object in world coordinate system; intersection point is needed for the world-space geometry of the illumination model

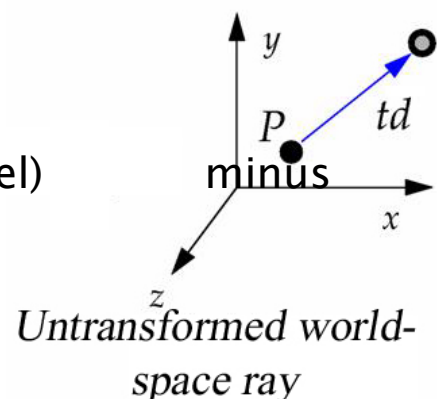


- Normalizing transformation takes world-space points to points in canonical view volume
 - translate to origin, orient with axes, scale x and y to adjust viewing angles, scale z so $z: [-1, 0]$; $x, y: [-1, 1]$
- Apply inverse of normalizing transformation:
Viewing Transformation

Generating Rays (4/4)

Summary

- Start ray at center of projection (eye point)
- Transform 2D integer screen-space point onto 3D film plane
 - use far clip plane as film plane
 - scale points to fit between -1 and 1
 - set z to -1 so points lie on far clip plane
- Transform 3D film plane point (pixel) into untransformed world coordinate system point
 - need to undo normalizing transformation (i.e., viewing transformation)
- Construct direction vector
 - point minus point is a vector
 - world-space point (mapped pixel) minus eye point



Ray-Object Intersection (1/5)

Implicit objects

- If an object is defined implicitly by a function f such that $f(Q) = 0$ IFF Q is a point on surface of object, then ray-object intersection is comparatively easy
 - can define many objects implicitly
 - implicit functions provide potentially infinite resolution
 - tessellating implicit functions is more difficult than using them directly

- For example, a circle of radius R is an implicit object in the plane, and its equation is

$$f(x,y) = x^2 + y^2 - R^2$$

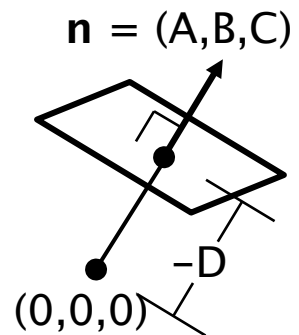
- points where $f(x, y) = 0$ are points on the circle

- An infinite plane is defined by the function:

$$f(x,y,z) = Ax + By + Cz + D$$

- A sphere of radius R in 3-space:

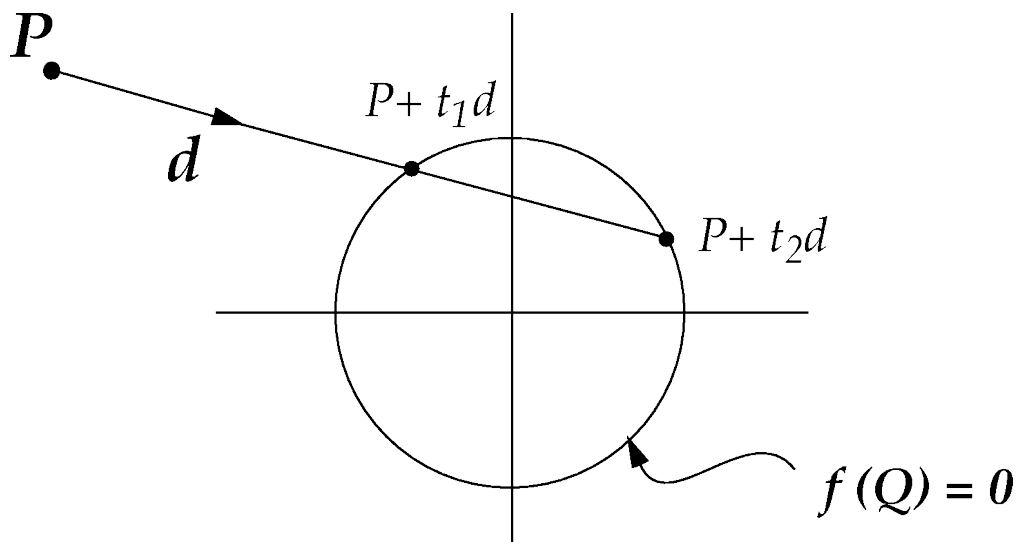
$$f(x,y,z) = x^2 + y^2 + z^2 - R^2$$



Ray–Object Intersection (2/5)

Implicit objects(cont.)

- At what points (if any) does ray intersect object?
- Points on ray have form $P + td$
 - t is any nonnegative real
- A point Q lying on object has property that $f(Q) = 0$
- Combining, we want to know “For which values of t is $f(P + td) = 0$?” (if any)



- We are solving a system of simultaneous equations in x, y (in 2D) or x, y, z (in 3D)

An Explicit Example (1/3)

2D ray-circle intersection example

- Consider the eye-point $P = (-3, 1)$, the direction vector $d = (.8, -.6)$ and the unit circle given by:

$$f(x,y) = x^2 + y^2 - R^2$$

- A typical point of the ray is:

$$Q = P + td = (-3,1) + t(.8,-.6) = (-3 + .8t, 1 - .6t)$$

- Plugging this into the equation of the circle:

$$f(Q) = f(-3 + .8t, 1 - .6t) = (-3 + .8t)^2 + (1 - .6t)^2 - 1$$

- Expanding, we get:

$$9 - 4.8t + .64t^2 + 1 - 1.2t + .36t^2 - 1$$

- Setting this to zero, we get:

$$t^2 - 6t + 9 = 0$$

An Explicit Example (2/3)

2D ray-circle intersection example (cont.)

- Using the quadratic formula:

$$\text{roots} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- We get:

$$t = \frac{6 \pm \sqrt{36 - 36}}{2}, \quad t = 3, 3$$

- Because we have a root of multiplicity 2, ray intersects circle at one point (i.e., it's tangent to the circle)
- We can use discriminant $D = b^2 - 4ac$ to quickly determine if a ray intersects a curve or not
 - if $D < 0$, imaginary roots; no intersection
 - if $D = 0$, double root; ray is tangent
 - if $D > 0$, two real roots; ray intersects circle at two points
- Smallest non-negative real t represents intersection nearest to eye-point

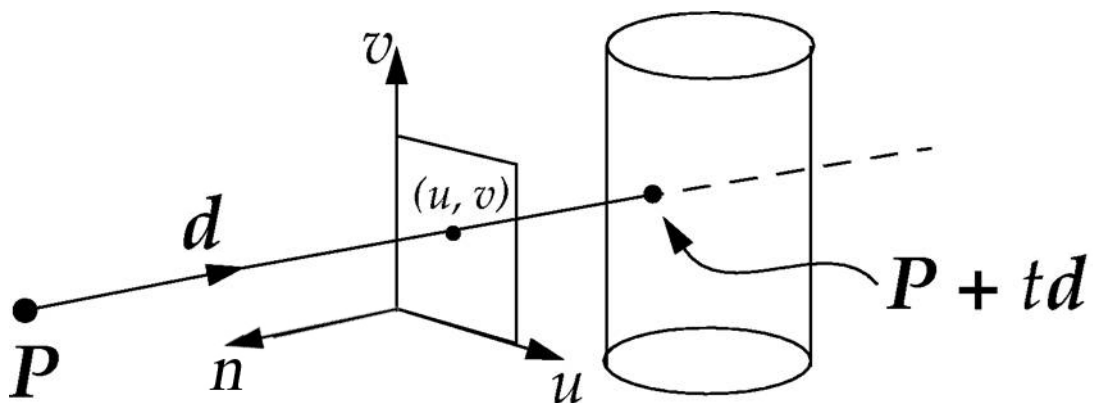
An Explicit Example (3/3)

2D ray-circle intersection example (cont.)

- Generalizing: our approach will be to take an arbitrary implicit surface with equation $f(Q) = 0$, a ray $P + td$, and plug the latter into the former:

$$f(P + td) = 0$$

- This results, after some algebra, in an equation with t as unknown
- We then solve for t , analytically or numerically



Ray–Object Intersection (3/5)

Implicit objects–multiple conditions

- For objects like cylinders, the equation

$$x^2 + z^2 - 1 = 0$$

in 3–space defines an infinite cylinder of unit radius, running along the y–axis

- Usually, it’s more useful to work with finite objects, e.g. such a unit cylinder truncated with the limits

$$y \leq 1$$

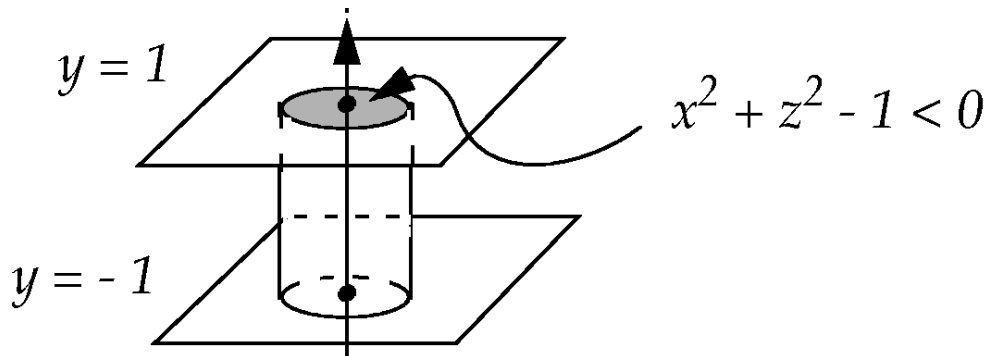
$$y \geq -1$$

- But how do we do the “caps?”
- The cap is the inside of the cylinder at the y extrema of the cylinder

$$x^2 + z^2 - 1 < 0, \quad y = \pm 1$$

Ray-Object Intersection (4/5)

Multiple conditions (cont.)



- We want intersections satisfying the cylinder:

$$x^2 + z^2 - 1 = 0$$

$$-1 \leq y \leq 1$$

or top cap:

$$x^2 + z^2 - 1 \leq 0$$

$$y = 1$$

or bottom cap:

$$x^2 + z^2 - 1 \leq 0$$

$$y = -1$$

Ray-Object Intersection (1/2)

Multiple conditions-cylinder pseudocode

- Solve in a case-by-case approach

```
Ray_inter_finite_cylinder(P,d):
```

```
// Check for intersection with infinite cylinder
```

```
t1,t2 = ray_inter_infinite_cylinder(P,d)
```

```
    compute P + t1*d, P + t2*d
```

```
// If intersection, is it between "end caps"?
```

```
if y > 1 or y < -1 for t1 or t2, toss it
```

```
// Check for intersection with top end cap
```

```
Compute ray_inter_plane(t3, plane y = 1)
```

```
Compute P + t3*d
```

```
// If it intersects, is it within cap circle?
```

```
if  $x^2 + z^2 > 1$ , toss out t3
```

```
// Check intersection with other end cap
```

```
Compute ray_inter_plane(t4, plane y = -1)
```

```
Compute P + t4*d
```

```
// If it intersects, is it within cap circle?
```

```
if  $x^2 + z^2 > 1$ , toss out t4
```

Among all the t's that remain (1-4), select the smallest non-negative one