Ray-Object Intersections

http://xformgamedevelopment.blogspot.com/2012/04/shoot-to-kill.html

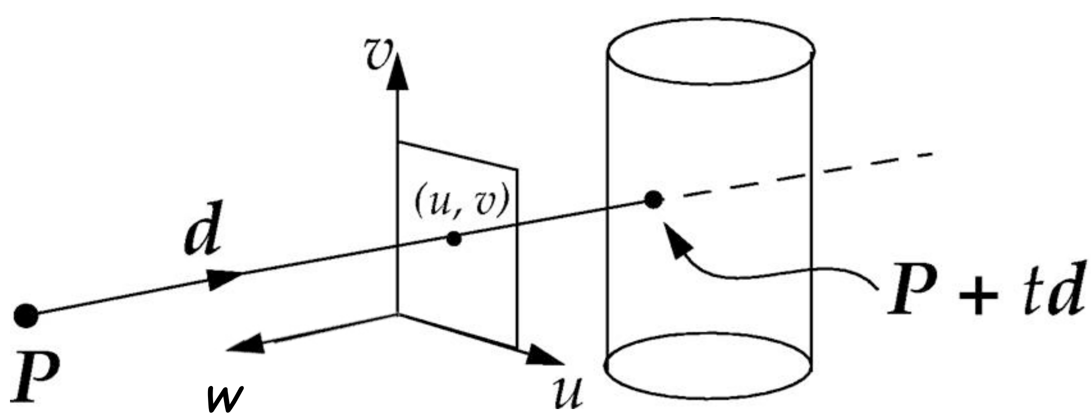
Points on ray have form ***P*** + ***td*,**
with P the location of the shooter, d the gun direction, and *t* any positive real number

Generating Rays (1/4)

Ray origin for a First-Person Picker game

- Let's look at geometry of problem in *world* space
- Start a ray from an "eye point": ***P***
- Send it out in some direction ***d*** from eye toward a point on film plane (a rectangle in the u-v plane in the camera's u,v,w space)

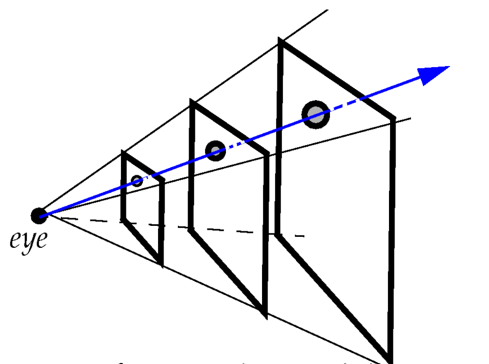
- Points along ray have form ***P*** + ***td*** where
  - ***P*** is ray's base point: the camera's eye
  - ***d*** is unit vector direction of ray
  - *t* is a nonnegative real number



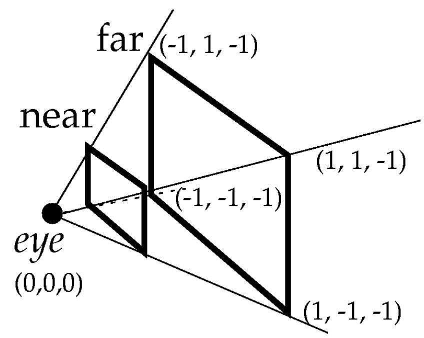
Generating Rays (2/4)

Ray direction

- Start with 2D screen-space point (pixel)
- Must convert 2D screen-space point into a 3D point on film plane in order to create a ray from eye point through film plane
  - Any plane orthogonal to *Look* vector is a convenient film plane
  - Plane z = some constant is orthogonal to *Look* in canonical view volume



untransformed world space



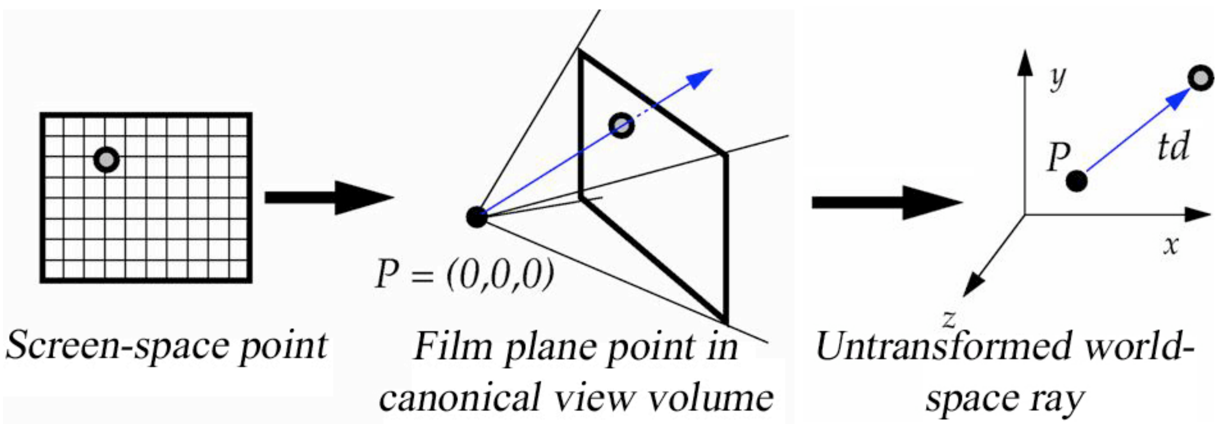
canonical view volume
Any plane $z = k$, $-1 <= k < 0$ can be the film plane

- 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, z = -1 :-)
  - To convert, we 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 and this world-space point
  - we need vector to be in world-space in order to intersect with original object in world coordinate system



- Normalizing transformation takes world-space points to points in canonical view volume
  - translate to origin, orient with axes, scale so z: [-1, 0]; x,y: [-1, 1]

- Apply inverse of normalizing transformation **N**

Generating Rays (4/4)

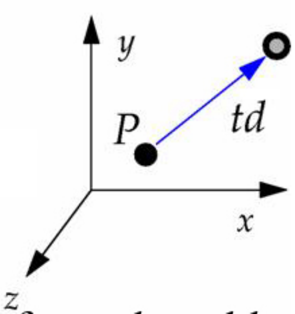
Summary of ray construction

- 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

- Convert 3D film plane point into 3D world coordinate system point
  - need to undo normalizing transformation

- Construct direction vector
  - point minus point is a vector
  - world-space point (mapped pixel) minus eye point



Untransformed world-space ray

Ray-Object Intersection

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

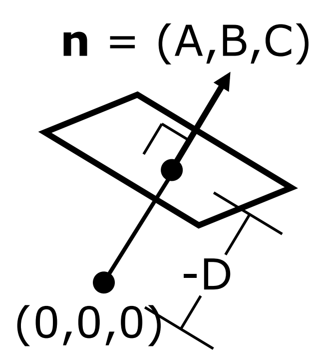
- For example, a circle of radius *R* is an implicit object in the plane, and its equation is

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

  - point (x,y) is on the circle when *f(x, y)* = 0

- An infinite plane is defined by the function:

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

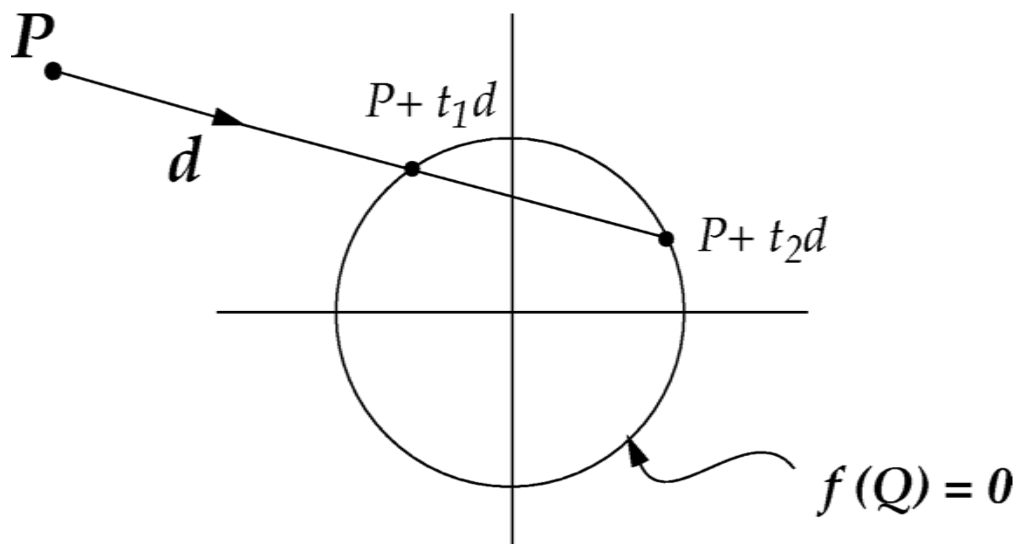


- A sphere of radius *R* in 3-space:

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

Ray and Implicit Object Intersection

- At what points (if any) does ray intersect object?
- Points on ray have form $P + t\mathbf{d}$
  - 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 $f(P + t\mathbf{d}) = 0$?" (if any)



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

A 2D 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:

$$\mathbf{Q} = \boldsymbol{P} + t\boldsymbol{d} = (-3,1) + t(.8,-.6) = (-3 + .8t, 1 - .6t)$$

- Plugging this into the equation of the circle:

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

  - expanding, we get:

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

- Setting $f(\mathbf{Q})$ to zero, we get:

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

A 2D Example (2/3)

2D ray-circle intersection example (cont.)

- Using the quadratic formula:

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

- We get:

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

- 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

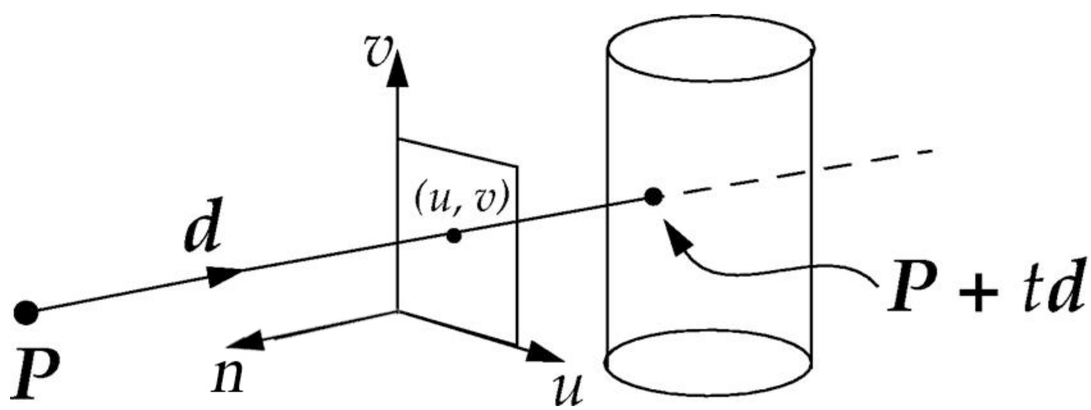
A 2D Example (3/3)

2D ray-circle intersection example (cont.)

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

$$f(P + t\mathbf{d}) = 0$$

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



Ray and Multi-Faceted Object Intersection

- For objects like cylinders, the equation

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

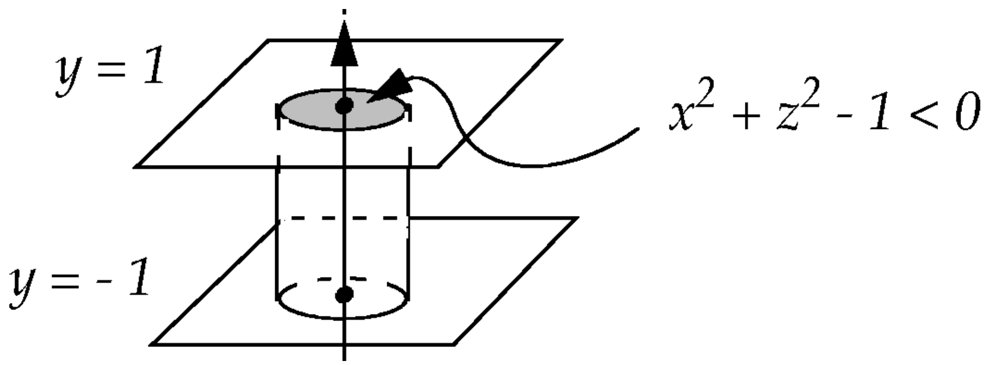
  in 3-space defines an infinite cylinder of unit radius, running along the y-axis
- Usually, it's more useful to work with finite objects, e.g. such a unit cylinder truncated with the limits

$$y \leq 1$$
$$y \geq 1$$

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

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

Intersections: *Multiple Conditions*



- 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:

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

$$y = -1$$

Pseudocode

Multiple conditions-cylinder pseudocode

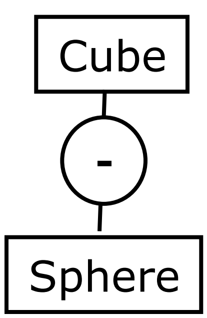
- Solve in a case-by-case approach

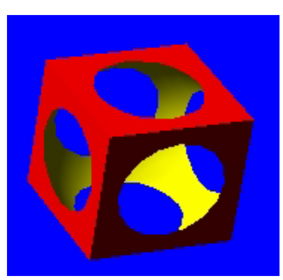
```
Ray_inter_finite_cylinder(P,d):
// Check for intersection with infinite cylinder
t1,t2 = ray_inter_infinite_cylinder(P,d)
              compute P + t1*d, P + t2*d
// If intersection, is it between "end caps"?
if y > 1 or y < -1 for t1 or t2, toss it

// 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 x² + z² > 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 x² + z² > 1, toss out t4

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

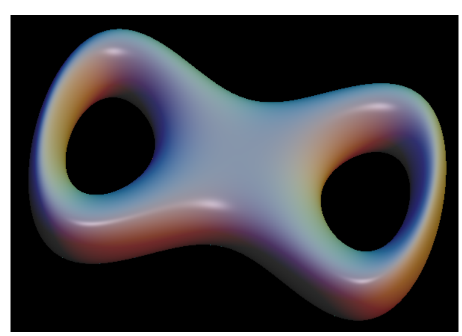
Ray-Object Intersection

Implicit surface strategy summary

- Substitute ray (P + *t*d) into implicit surface equations and solve for *t*
  - surface you see "first" from eye point is at smallest non-negative *t*-value
- For complicated objects (not defined by a single equation), write out a set of equalities and inequalities and then code as cases…
- Latter approach can be generalized cleverly to handle all sorts of complex combinations of objects
  - Constructive Solid Geometry (CSG), where objects are stored as a hierarchy of primitives and 3-D set operations (union, intersection, difference)
  - "blobby objects", which are implicit surfaces defined by sums of implicit equations (F(x,y,z)=0)







F(x,y,z) = ((x^2*(1-x^2)-y^2)^2+0.5*z^2-f*(1+b*(x^2+y^2+z^2)) = 0