CS770/870 Spring 2017
Ray Tracing Implementation

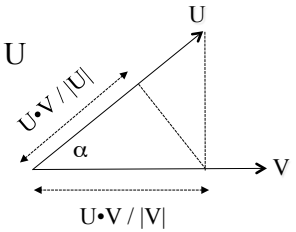
Related material:Angel 6e: Ch 11.3

Ray-Object intersections

Spheres
Plane/Polygon
Box/Slab/Polyhedron
Quadric surfaces
Other implicit/explicit surfaces

Useful Vector Information

- Vector calculus is helpful notation in ray tracing
- *dot* product (also called *inner* product)
  - Let $U = (u_x, u_y, u_z)$ and $V = (v_x, v_y, v_z)$
  - $U \cdot U = a_x * b_x + a_y * b_y + a_z * b_z$
  - $U \cdot V = |U||V| \cos \alpha$, where α is angle between U and V where $|U|$ is the length of U
  - $U \cdot V / |V|$
- *scalar* (orthogonal) projection of U onto V
  - $|U| \cos \alpha = U \cdot V / |V|$
- *scalar* (orthogonal) projection of V onto U
  - $|V| \cos \alpha = U \cdot V / |U|$



Ray-Sphere Definitions

- *Implicit* representation of a sphere of radius, *r*, and center $C = (x_c, y_c, z_c)$:

  $(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = r^2$

- *Parametric* representation of a ray

  $R(t) = R_o + tV \quad \text{for } t > 0$

  where $R_o = (x_o, y_o, z_o)$ is the ray *origin*
  and $V = (x_v, y_v, z_v)$ is the ray *direction* as a *unit* vector
  This is really 3 equations:

  $x(t) = x_o + t\,x_v$
  $y(t) = y_o + t\,y_v$
  $z(t) = z_o + t\,z_v$

Ray/Sphere Intersection Equation

- Start with sphere equation, *with center at origin*

$$x^2 + y^2 + z^2 = r^2$$

- Replace *x, y,* and *z* with *x(t), y(t), z(t)* of ray

$$(x_o + tx_v)^2 + (y_o + ty_v)^2 + (z_o + tz_v)^2 = r^2$$

- Expand the square terms

$$(x_v^2t^2 + 2x_vx_0t + x_0^2) + (y_v^2t^2 + 2y_vy_0t + y_0^2) + (z_v^2t^2 + 2z_vz_0t + z_0^2) = r^2$$

- Collect like terms (powers of t): $at^2 + bt + c = 0$

This simplifies calculations!

$$a = x_v^2 + y_v^2 + z_v^2 = 1 \quad \text{(V is a unit vector)}$$

$$b = 2(x_o - x_c)x_v + 2(y_o - y_c)y_v + 2(z_o - z_c)z_v$$

$$c = (x_o - x_c)^2 + (y_o - y_c)^2 + (z_o - z_c)^2 - r^2$$

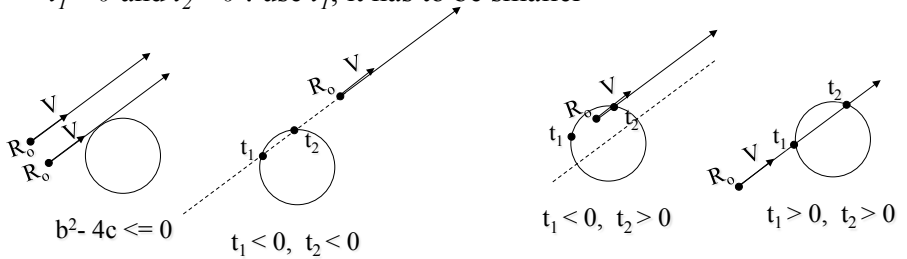
Solve quadratic equation

- Get 0, 1 or 2 solutions:
  - $b^2-4c<0$: 0 intersections
  - $b^2-4c=0$: 1 intersection, ray is tangent to sphere; ignore
  - $t_1<0$ and $t_2<0$: on "wrong" side of ray, ignore
  - $t_1<0$ and $t_2>0$: use t_2
  - $t_1>0$ and $t_2>0$: use t_1; it has to be smaller

Remember the quadratic equation?

$$t_1=\frac{-b-\sqrt{b^2-4c}}{2} \qquad t_2=\frac{-b+\sqrt{b^2-4c}}{2}$$

a=1 in this case



Ray/Arbitrary Sphere Intersection

- Start with sphere equation for center at (x_c, y_c, z_c)

$$(x-x_c)^2+(y-y_c)^2+(z-z_c)^2=r^2$$

- Replace x, y, and z with $x(t)$, $y(t)$, $z(t)$ of ray

$$((x_o+tx_v)-x_c)^2+((y_o+ty_v)-y_c)^2+((z_o+tz_v)-z_c)^2=r^2$$

- Simplify terms by collecting constant expressions

$$(tx_v+x_o-x_c)^2+(ty_v+y_o-y_c)^2+(tz_v+z_o-z_c)^2=r^2$$

$$(tx_v+x_d)^2+(ty_v+y_d)^2+(tz_v+z_d)^2=r^2$$

where $x_d=x_o-x_c$, $y_d=y_o-y_c$ and $z_d=z_o-z_c$

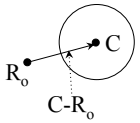
- This is the same equation as sphere at origin with (x_d, y_d, z_d) replacing (x_o, y_o, z_o)

Efficiency Improvement

- Can we find "miss" cases more efficiently
  - First find if R_o is inside or outside the sphere
    - if distance from R_o to C < r, then R_o is inside sphere

$$\text{distance}(C,R_o)=\sqrt{(C-R_o)\bullet(C-R_o)}$$

  - But, no need to compute the square root



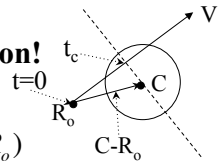
R_o is inside sphere if $(C-R_o)\bullet(C-R_o)-r^2<0$

But, $(C-R_o)\bullet(C-R_o)=(x_c-x_o)^2+(y_c-y_o)^2+(z_c-z_o)^2-r^2=c$

  - So R_o is inside sphere if $c<0$, for the c term of the quadratic equation.
  - If R_o is inside sphere, we know we have exactly 1 intersection and it is t_2

Efficiency Improvement (cont)

- If R_o outside, find t_c = closest point on ray to C
  - Project (C-R_o) onto V: $(C-R_o)\bullet V=t_c$
  - $(C-R_o)\bullet V=-b/2$ **the *b* from quadratic equation!**



- If $t_c>0$, find d, distance from C to ray

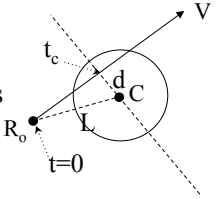
$$L^2=t_c^2+d^2 \quad \text{where } L=\|C-R_0\|=(C-R_o)\bullet(C-R_o)$$

so, $d^2=L^2-t_c^2=(C-R_o)\bullet(C-R_o)-t_c^2$

and $d^2-r^2>0$ means ray misses sphere

or $(C-R_o)\bullet(C-R_o)-t_c^2-r^2>0$ means miss

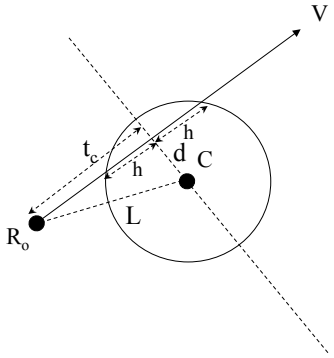
But $c=(C-R_o)\bullet(C-R_o)-r^2$ and $t_c^2=\left(\frac{1}{2}b\right)^2$ **for b and c from quadratic equation :**



so condition is $c-\left(\frac{1}{2}b\right)^2>0 \equiv c-\frac{1}{4}b^2>0 \equiv 4c-b^2>0 \equiv b^2-4c\le 0$

Efficiency Improvement (cont)

- How does this help?
  1. Calculate $c = (C\text{-}R_o)\bullet(C\text{-}R_o)\text{-}r^2$
  2. Calculate $t_c = (C\text{-}R_o)\bullet V$
  3. If $c > 0$ and $t_c < 0$, done; ray is outside and points away
  4. Calculate $h^2 = t_c^2 - c = \frac{1}{4}b^2 - c$
  5. if $h^2 < 0$, done, no int.
  6. Calculate $h = sqrt(h^2)$
  7. if $c > 0$, R_o outside: $t = t_c - h$
     else if $c < 0$, R_o inside: $t = t_c + h$
  8. Use t to get x,y,z
- If C=(0,0,0), even simpler



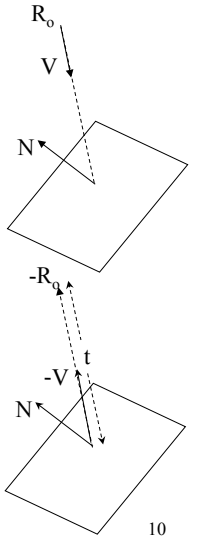
Ray/Plane Intersections

- Ray: $R(t) = R_o + tV$ for $t > 0$
- Plane (implicit)
  $ax + by + cz + d = 0$ where $a^2 + b^2 + c^2 = 1$
  Note: $N = (a, b, c)$ is unit normal to plane
- Substitute ray equations into plane equation

$$a(x_o + tx_v) + b(y_o + ty_v) + c(z_o + tz_v) + d = 0$$

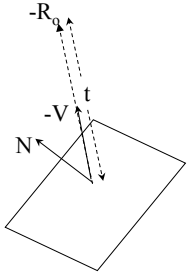
Solve for t

$$t = -\frac{(ax_o + by_o + cz_o + d)}{ax_v + by_v + cz_v} = \frac{N \bullet R_o + d}{N \bullet (-V)}$$



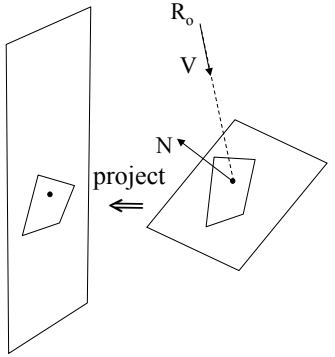
Ray/Plane Intersections Steps

1. Calculate $v_d = N\bullet(-V)$, $|V| == 1$
   $v_d == 0$ means ray is parallel to plane; no int.
   $v_d < 0$ normal in opposite direction from ray;
   if have 1-sided polygons and back-face culling, done: polygon not visible!
2. Calculate $v_o = N\bullet(-R_o) + d$
3. Calculate $t = v_o / v_d$
   $t < 0$: intersection "behind ray", no int.
4. Calculate (x,y,z) from ray equations.



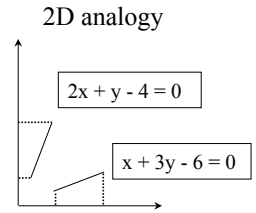
Ray/Polygon Intersection

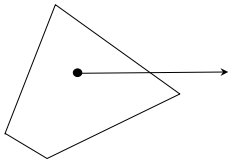
1. Find Ray/Plane intersection
2. Project polygon and point along one of major axes onto 2D plane.
3. Do 2D *point-in-polygon* test to determine if intersection point is inside polygon



Point-in-Polygon Test

1. Project 3D polygon along one coordinate axis to *xy* or *yz* or *xz* plane. Which is best?
   - want to avoid a thin projected polygon
   - to maximize projected area, project along axis with maximum coefficient in plane equation: see 2D analogy on the right
2. 2D point-in-polygon
   - draw horiz (or vert) line from projected intersection point to infinity; $y=y_{int}$
   - intersect $y=y_{int}$ with each edge (pretty simple!)
   - count intersections with edges: odd ⇒ inside
   - be careful with vertex intersections





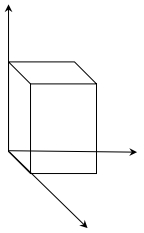
Ray/Box Intersection

- Transform ray back to object space where box is oriented along major axes to origin and even to unit cube
  - Intersect transformed ray with planes:
    - x=0, x=1, y=0, y=1, z=0, z=1
  - Line/plane intersections are trivial; intersection of ray with x=0:

    $x(t) = x_o + tx_v$ // ray equation for x

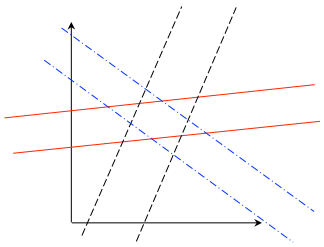
    $t_0 = (0 - x_o) / x_v = -x_o / x_v$ // intersection with x=0
  - The *t* for *x = 1* is just one more addition

    $t_1 = (1 - x_o) / x_v = 1/x_v - x_o / x_v = 1/x_v + t_0$



Ray/Slab Intersection

- Slabs are pairs of parallel planes; they are good for creating tighter bounding volumes
- Just two sides of a box and once get *t* for one slab, the *t* for parallel one is just one addition



Convex Polyhedra

- A *polyhedron* is a 3D object whose faces are planar
- Polyhedra can be defined by:
  - the polygons that make up their faces
  - the space defined by the intersection of the negative sides of all the planes that define the faces
- Polygon representation
  - natural for polygon-based rendering
  - hard to validate *correctness*: are there gaps between polygons? do any polygons intersect? etc.
- Planar representation
  - relatively easy correctness computation
  - hard to render (except with ray tracing)

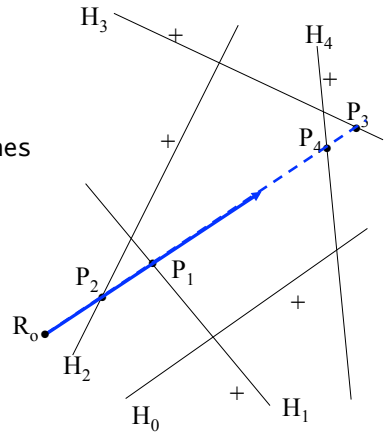
Ray/Polyhedron Intersection

- Given a convex polyhedron defined by intersection of half-planes, find intersection of a ray with polyhedron:

```
for each plane, Hi
   Pi = intersection of ray with Hi
   if Pi on - side of all other planes
      if Ro is on + side of Hi
         return Pi
return null
```

P_0 doesn't exist
P_1 is on – side of other planes and
R_0, is on + side of H_1 Good!
P_2 is on + side of H_1, P_3 is on + side of H_4,
P_4 is on – side of all, but R_0 is on – side of H_4

2D analogy



Ray/Polyhedron Intersection - 2

- Basic algorithm has some weaknesses:
  - If ray starts *inside* the polyhedron, it fails/
    - Easily fixed by first testing R0 against all planes; if it is on the – side of all planes and changing innermost *if* test to:

```
if (Ro is on + side of Hi) or (Ro in polyhedron)
      return Pi
```

  - Needs extra code to recognize intersections with/near vertices, where the "inside" test for the other planes could be wrong due to round-off error
  - Only works for convex polyhedra
- Note that the "correctness" of the polyhedron specification is not a problem for the rendering since a polyhedron with null volume just won't ever appear

Ray v. Quadric Surface

- Any quadric surface can be represented implicitly by

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$$

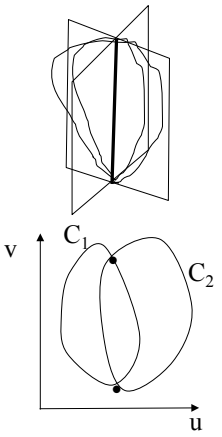
- Substitute the parametric equations for the ray into implicit equation and solve for t using the quadratic formula
- Just like a sphere, but much more algebra!

Ray v. Generic Implicit Surfaces

- Ray intersections with implicit surfaces of degree > 2 or non-polynomial surfaces are normally solved with numerical techniques

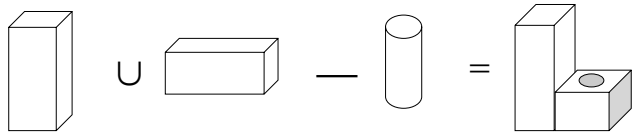
Ray vs. Explicit Surfaces

- Suppose the surface is defined parametrically (in 2 parameters):
  $S(u,v) = [x(u,v), y(u,v), z(u,v)]$
- Can make some progress by representing the ray as the intersection of any two planes that share it:
  $a_1x + b_1y + c_1z + d_1 = 0$
  $a_2x + b_2y + c_2z + d_2 = 0$
- Intersection of planes an surface yields 2 curves in 2D:
  $C_1(u,v) = a_1x(u,v) + b_1y(u,v) + c_1z(u,v) + d_1 = 0$
  $C_2(u,v) = a_2x(u,v) + b_2y(u,v) + c_2z(u,v) + d_2 = 0$
- Use numerical techniques in 2D to solve intersection in (u,v) space



CSG Surfaces

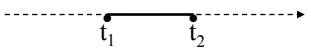
- Constructive Solid Geometry
  - Define objects as union, intersection or set difference between basic shapes, like parallelopiped and cylinders



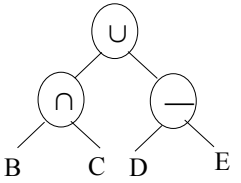
  - Effective representation for analysis
    - validity of representation, many physical properties are easily computed, and much more
  - Not very efficient for rendering purposes, except with ray tracing

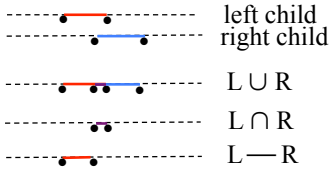
Ray v. CSG

- Given a CSG tree
  Do a postfix tree walk:
  for each leaf
  compute ray/solid intersection in t
  // each primitive has 0 or 2 ints



  for each operator
  combine ray *classification*
  of its children and *simplify*
  by merging





Other Kinds of Surfaces

- Sweep surfaces
  - specialized algorithms exist for some specific kinds of swept objects and paths.
- Procedural objects
  - parametric surfaces can be treated as procedural using subdivision techniques
  - Fractal surfaces
    - some can be handled by subdivision techniques
    - but, bounding volume test can be difficult
  - Particle systems and other true procedural surfaces have very specialized intersections
  - Volume data based on voxels: not very hard
  - Volume data based on octrees: also not hard