Intersections and Containment

- How to tell if two objects intersect, or one is inside another.
- Applications:
  - Culling (if object isn't in visible region, don't render it).
  - Ray tracing (intersect a ray of light with objects).
  - Collision detection (do things intersect?)

Intersections and Containment

- 2D
  - Line and line
  - Line segments (1D containment)
  - Line and Circle
  - Line and other quadratic curves.
  - Point inside convex polygon (sidedness).
  - Point inside circle/ellipse
  - Point inside axial rectangle
  - Point inside non-convex polygon.
    - Divide into convex regions
    - Crossing algorith
  - Polygon intersection is just line intersection or point containment
- 3D
  - Line and Plane
  - Line and Triangle (point inside triangle).
  - Line and sphere (or other polynomial surfaces).
  - Point inside convex polyhedron
- Strategies for Speeding up
  - Enclose with simpler shape (sphere or rectanguloid
  - Multiscale
- Application: Culling

Intersection of line and line

- Solve two equations
  - $y = mx + b$, $y = nx+c$

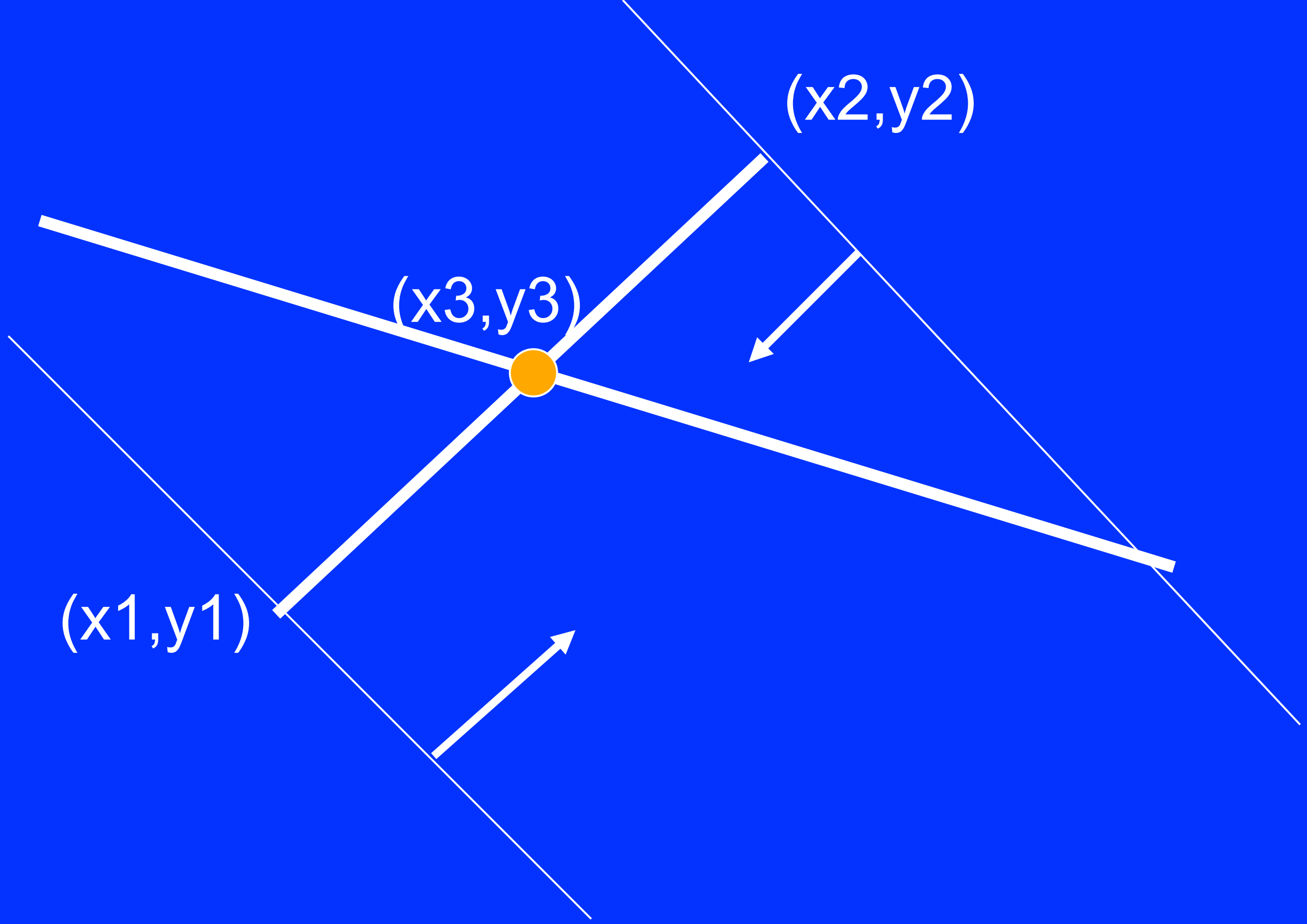
Intersection of line and line segment

- Convert line segment to line
  - Endpoints: (x1,y1) (x2,y2)

    Tangent of line is: (x2-x1, y2-y1).

    Normal is: (y1-y2, x2-x1).

    x(y1-y2) + y(x2-x1) = (y1-y2)x1+(x2-x1)y1.
- Find Intersection point of lines
- Check if this is inside line segment:



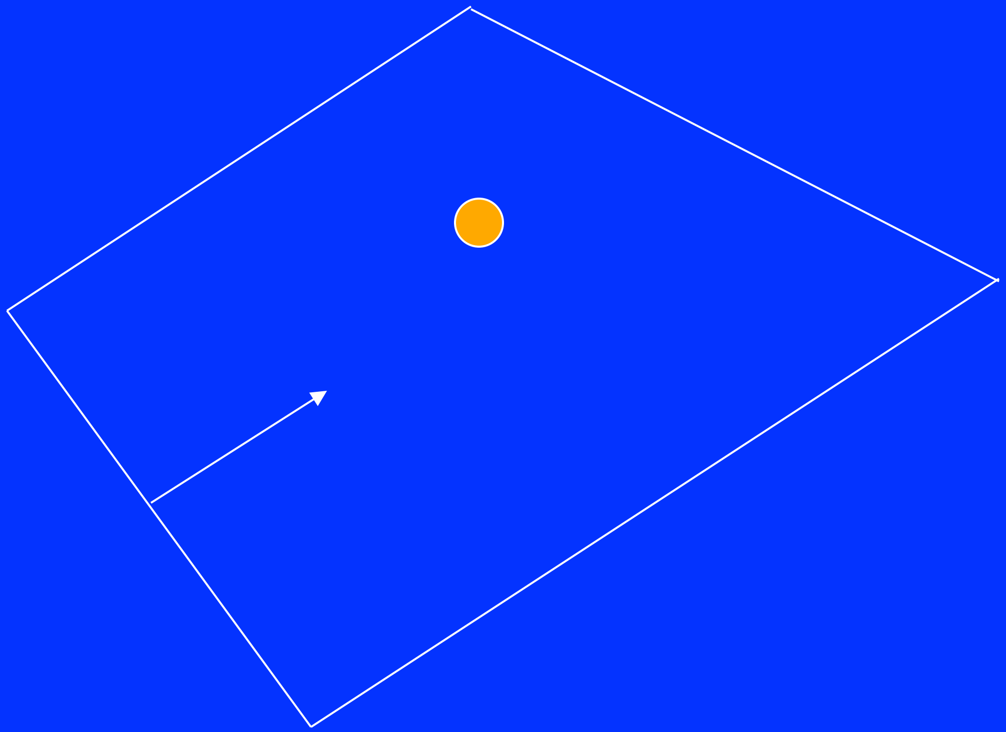
n = (x2-x1, y2-y1). <n, (x3-x1,y3-y1)> > 0?

<-n, (x3-x2, y3-y2)> > 0?

Intersection of line and circle

- Solution to two equations:
  - y = mx + b
  - (x-p)*(x-p) + (y-q)*(y-q) = r*r
- Produces one quadratic equation, which has zero, one or two real solutions.
- Line and other quadratics are just the same.

Point inside convex polygon



- A convex polygon is the intersection of half-planes.
- Point must be on correct side of each line derived from sides.

Eg., ax + by > c

- How do we know if it’s > or <?

If n is the normal to a side, p is a point inside the polygon, and q is a point we are testing, then sign(<n,p>) != sign(<n,q>) means q is not in the polygon.

How can we find p? One way is to average any three vertices.

Point inside Circle

- For circle, point is inside if distance to center is less than radius.

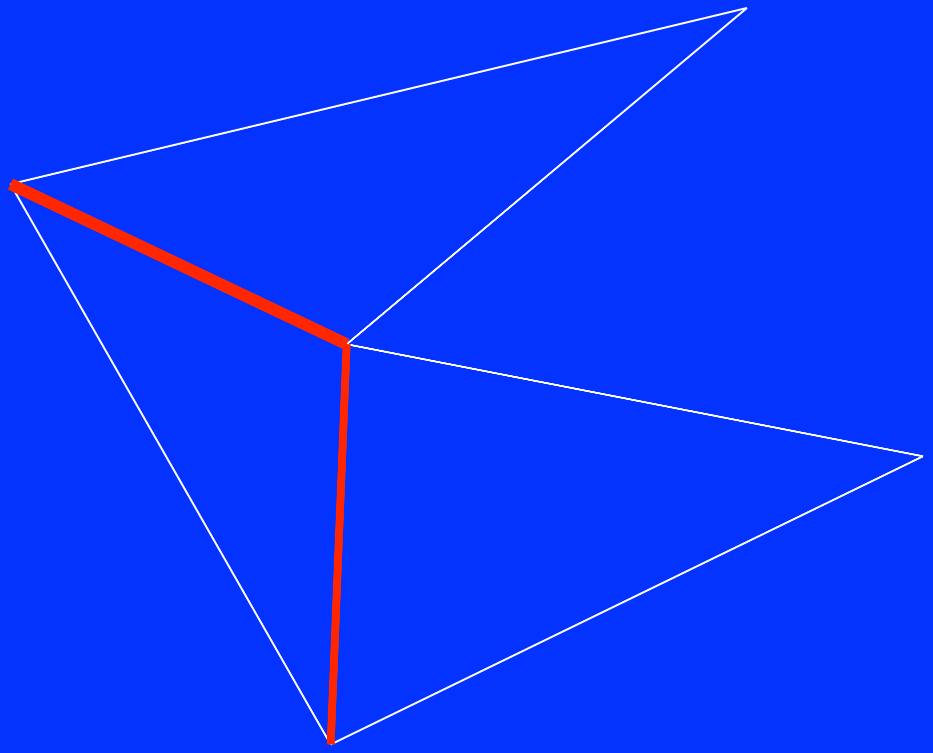
Point inside axial rectangle

- Axial rectangle has sides aligned with x and y axis.
- Described by minx, maxx, miny, maxy.
- (x,y) inside if minx <= x <= maxx,

miny <= y <= maxy.

- Notice that this is the same method used to see if a point is inside a convex polygon. It's simple because the normals to the sides of the rectangle are (1,0) and (0,1).

Point inside non-convex polygon



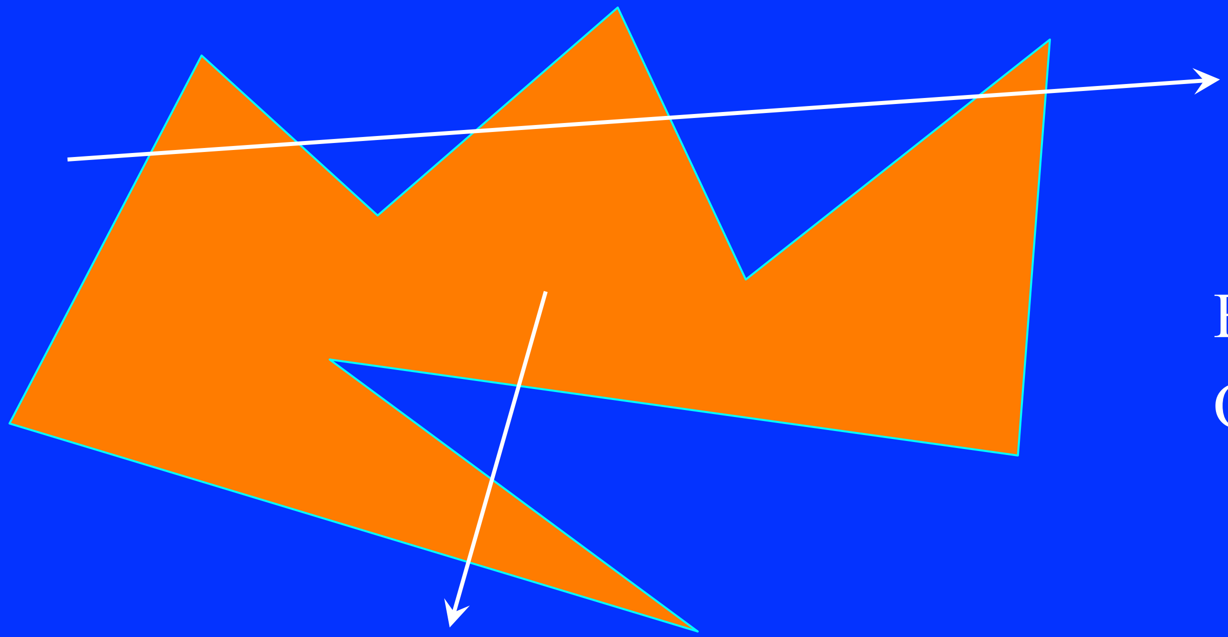
Divide into convex polygons, and see if point is inside any of these.

We can always divide into triangles by taking every three consecutive vertices.

(This may not be the most efficient way, adding the fewest sides).

Point inside non-convex polygon

- Crossing number test

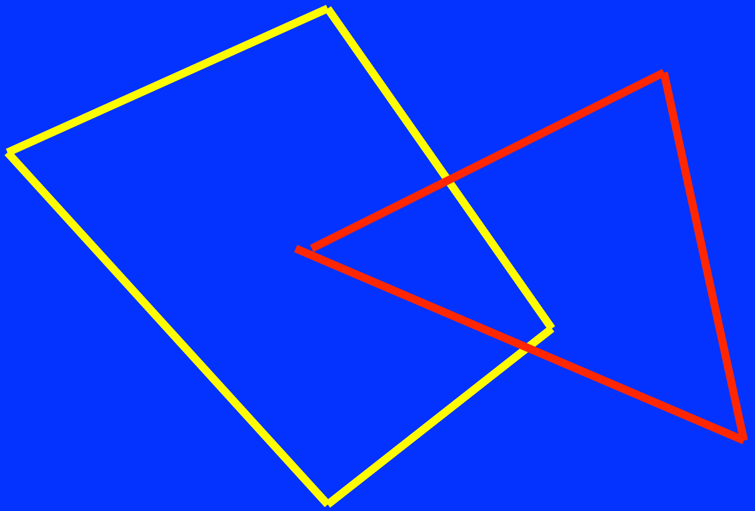


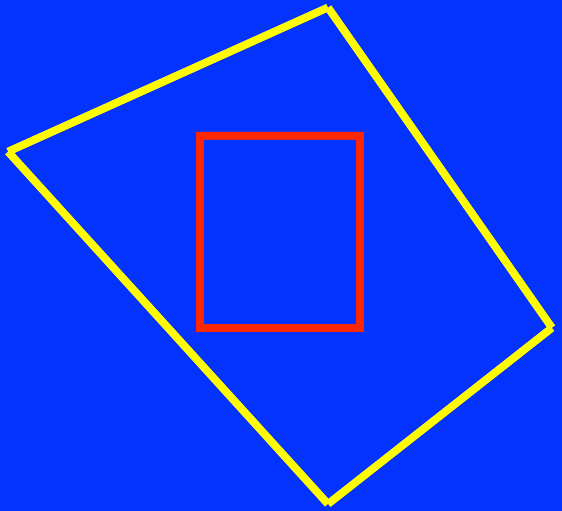
Even: Outside

Odd: Inside

2D Intersections

- Sides intersect
- Or one inside other





3D Line and Plane

- A line is defined by 2 linear eqns, a plane by 1. Solve 3 eqns w/ 3 unknowns.
- Or: $ax + by + cz = d$, &

$(x0,y0,z0)+t(u,v,w) = (x,y,z)$

Substitute for x,y,z in 1st equation and get linear equation in t.

3D Line and Triangle

- Find equation for plane of triangle, (p1, p2, p3).
  - Normal is $n=(p2-p1)\times(p3-p1)$
  - $\langle n,(x,y,z)\rangle = \langle n,p1\rangle$
- Intersect line and plane.
- Intersection point is inside triangle iff it is after orthographic projection to 2d.

3D Line and Polynomial

- Just the same as in 2D.
- Two equations for line, one for polynomial.
- Solve three equations with three unknowns.
- Wind up with a polynomial of one variable, which may have 0, 1 or multiple solutions.

Point inside convex polyhedron

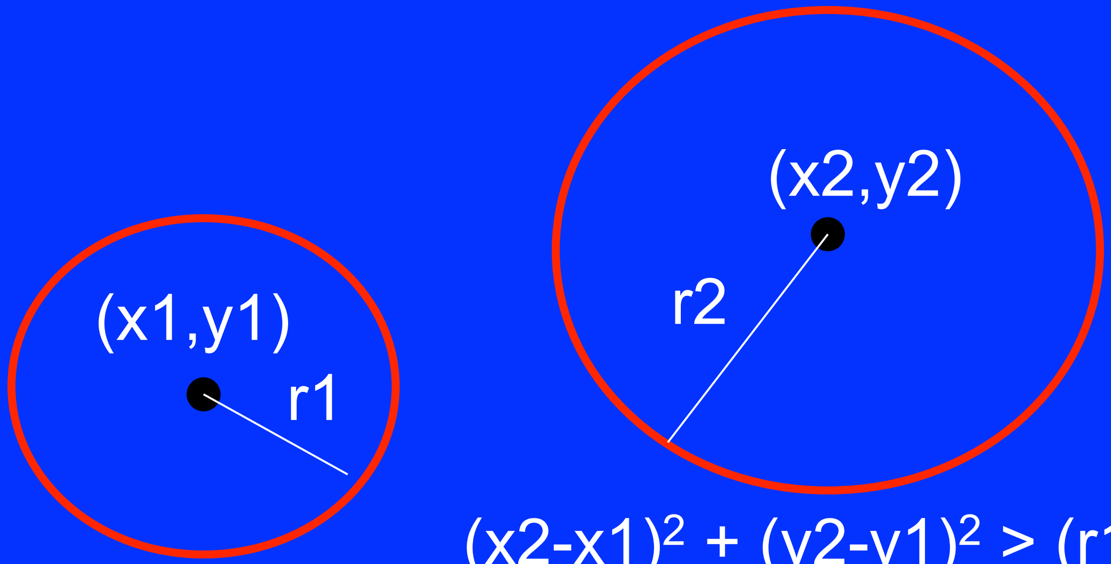
- Same as 2D. Is point on right side of each side of polyhedron?
  - Inner product with the normal to that side should have the right sign.

Collision Strategy

- Brute force test. Fine if few shapes.
- Test by bounding with simpler shape.
  - Only do brute force if necessary.
- Use hierarchy of simpler shapes.
  - Faster for complex scenes.

Circles easier

- Testing whether two circles intersect is much easier.
- So put a circle around each shape (How?)



$$(x2-x1)^2 + (y2-y1)^2 > (r1+r2)^2$$

Circles

- Pros: Very fast.
  - If scene is fixed, circles for scene can be computed once.
  - If object moves, circle can move with it easily.
- Cons: Circle can exaggerate shape.

Axial Rectangles

- A rectangle with sides aligned with the *x* and *y* axes.
  - Easy to generate. Just take min and max x and y values.
  - Easy to check for intersection.
    - Don’t intersect if one max value less than other’s min value.

Hierarchical

- Keep subdividing space into axial rectangles: quadtrees.
  - Bound everything with axial rectangles.
  - Divide space into four squares. Does object share a square with the scene?
  - If yes, recurse.
  - At some point just check.
- Many related, more complicated strategies possible.