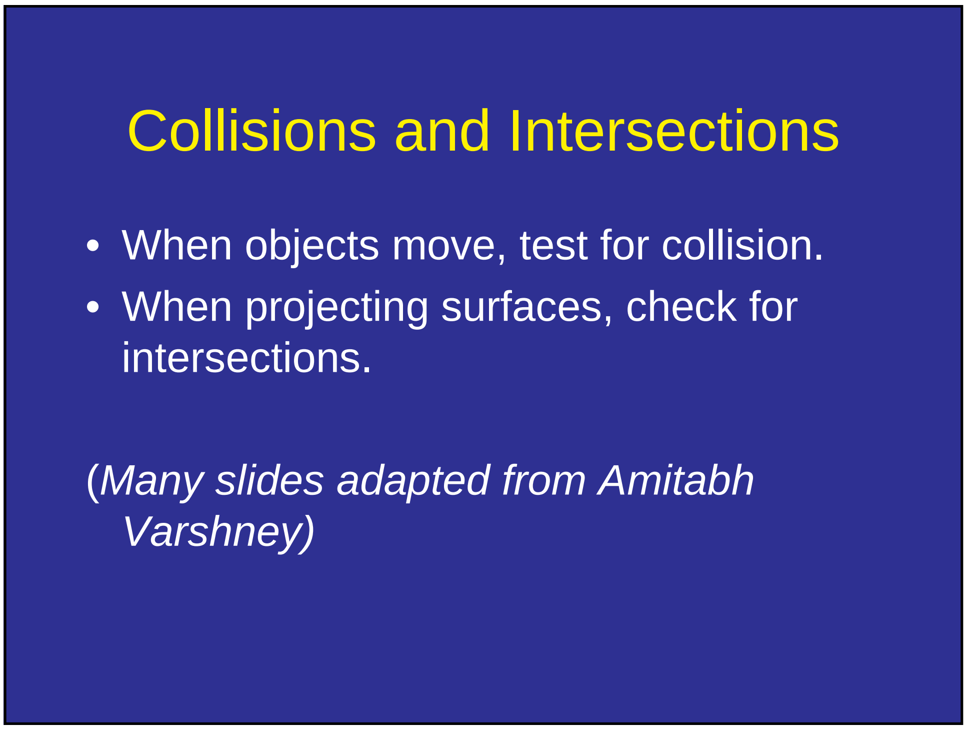
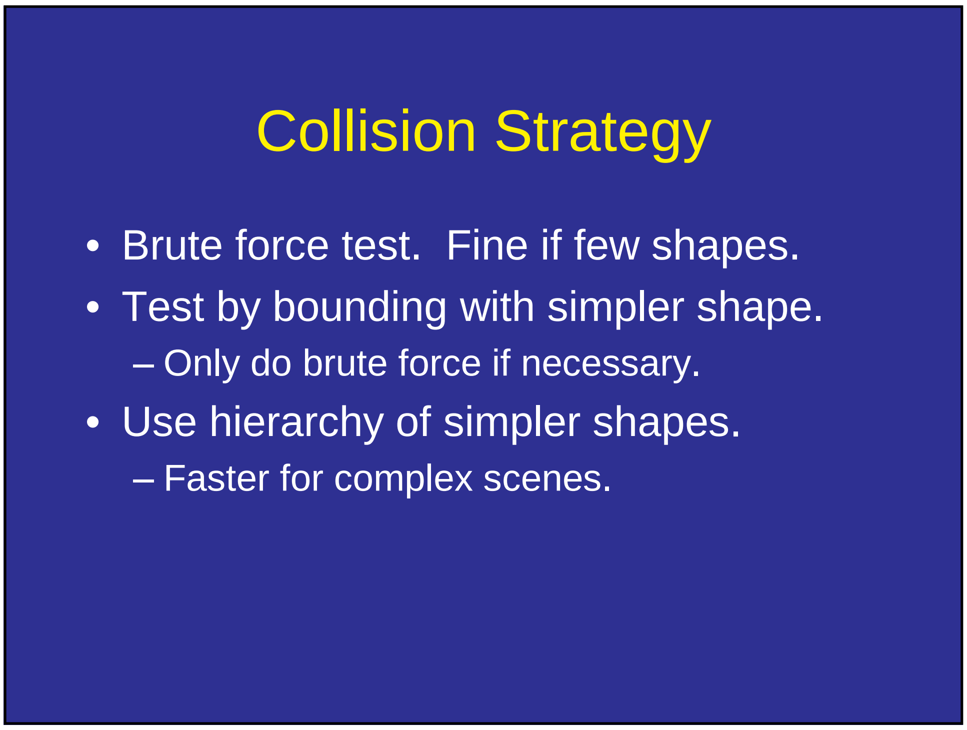


Collisions and Intersections

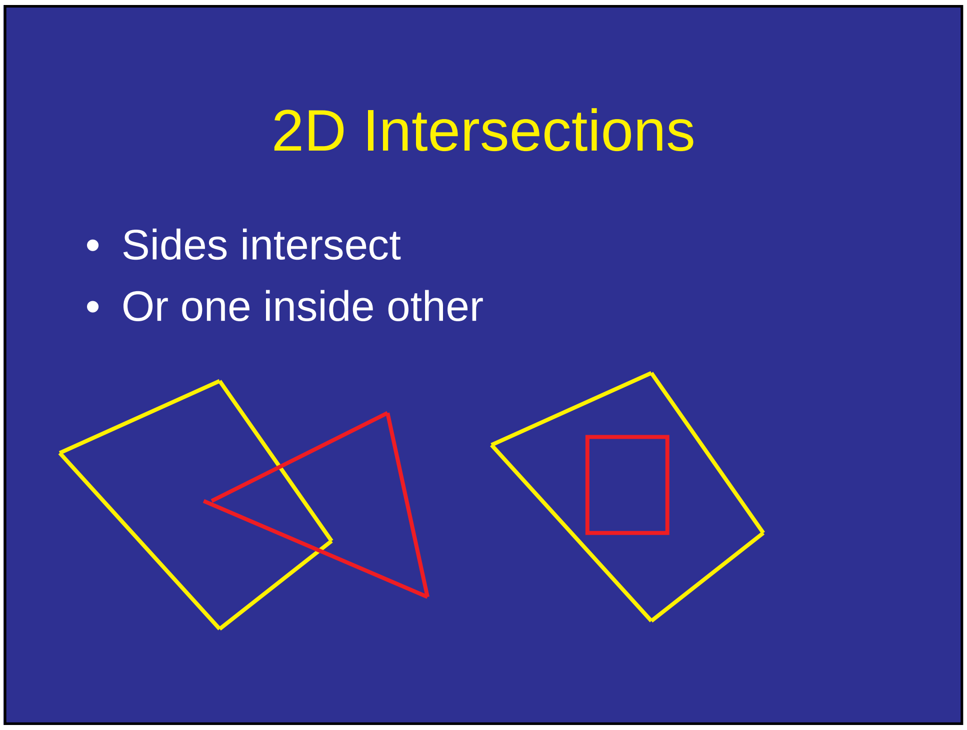
- When objects move, test for collision.
- When projecting surfaces, check for intersections.

(*Many slides adapted from Amitabh Varshney)*



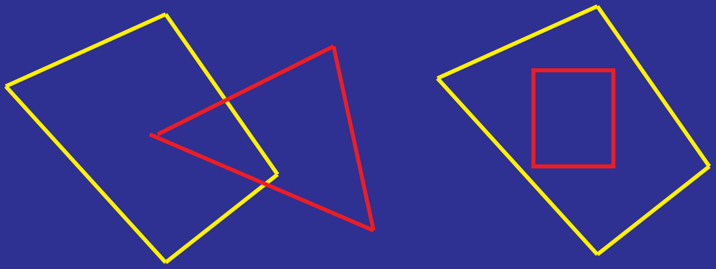
Collision Strategy

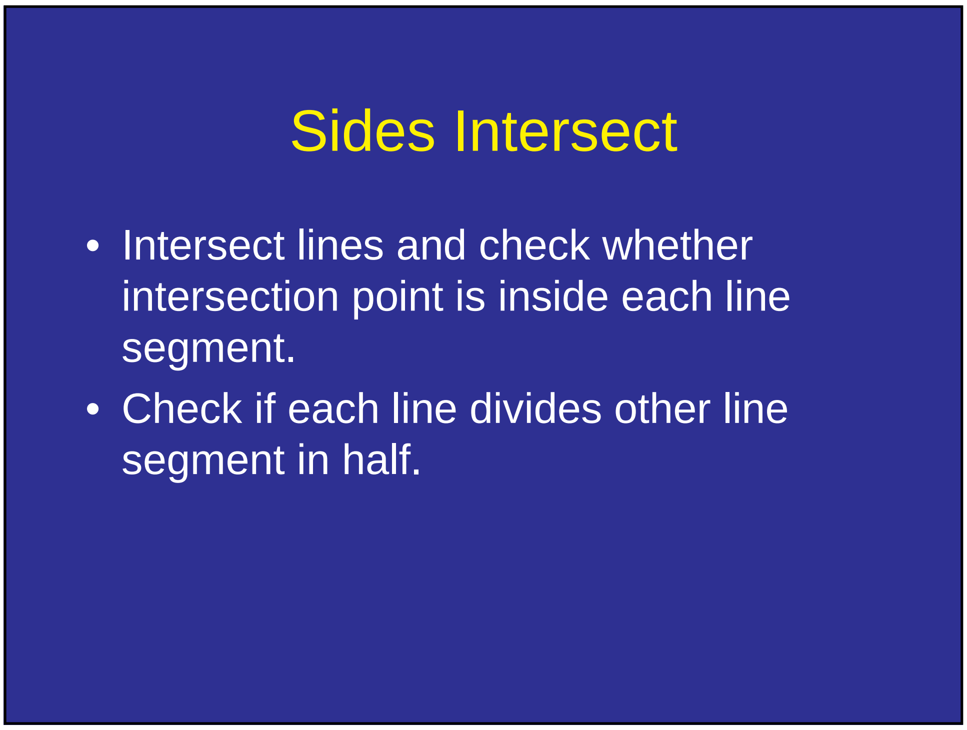
- Brute force test. Fine if few shapes.
- Test by bounding with simpler shape.
  - Only do brute force if necessary.
- Use hierarchy of simpler shapes.
  - Faster for complex scenes.



2D Intersections

- Sides intersect
- Or one inside other





Sides Intersect

- Intersect lines and check whether intersection point is inside each line segment.
- Check if each line divides other line segment in half.

How to tell whether two line segments intersect. First, convert to equations for lines. (x1,y1) (x2,y2) goes to y = (y2-y1)/(x2-x1)x + (y1(x2-x1) – x1(y2-y1). Now suppose we have two lines, y = m1x+b1, y = m2x+b2. Solve for x and y. For example, we have m1x+b1=m2x+b2. x = (b1-b2)/(m2-m1). Now we want to know whether (x,y) is in between (x1,y1) and (x2,y2), and the same for the other line segment. One simple way is to tell whether the sum of the distance from each end point to (x,y) is the same as the distance between the end points.

This is a bit cumbersome. We can do things with less computation by checking whether the line of line segment one divides the endpoints of line segment two, and vice versa. To do this, compute the line the first line segment lies on, and represent it as: ax + by = c, where (a,b) is a unit vector. Then compute (a,b)*(x1,y1) = c1, and (a,b)*(x2,y2) = c2. We should have either c1 <= c <= c2, or c2 <= c <= c1.

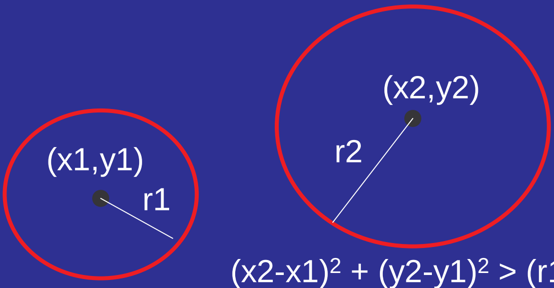
One shape inside another

- Easier for convex shapes.
  - Convex polygon is inside iff all vertices are inside.
  - Vertex is inside iff it is on inside of each side.
- If shape isn’t convex, decompose it into convex shapes.

Recall from before how to tell if a point is inside a convex shape, by judging whether it is on the appropriate side of each bounding line of the shape. We can write a line as (a,b)*(x,y) = c. Then a side of the line is (a,b)*(x,y) >= c (or <=c).

Circles easier

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



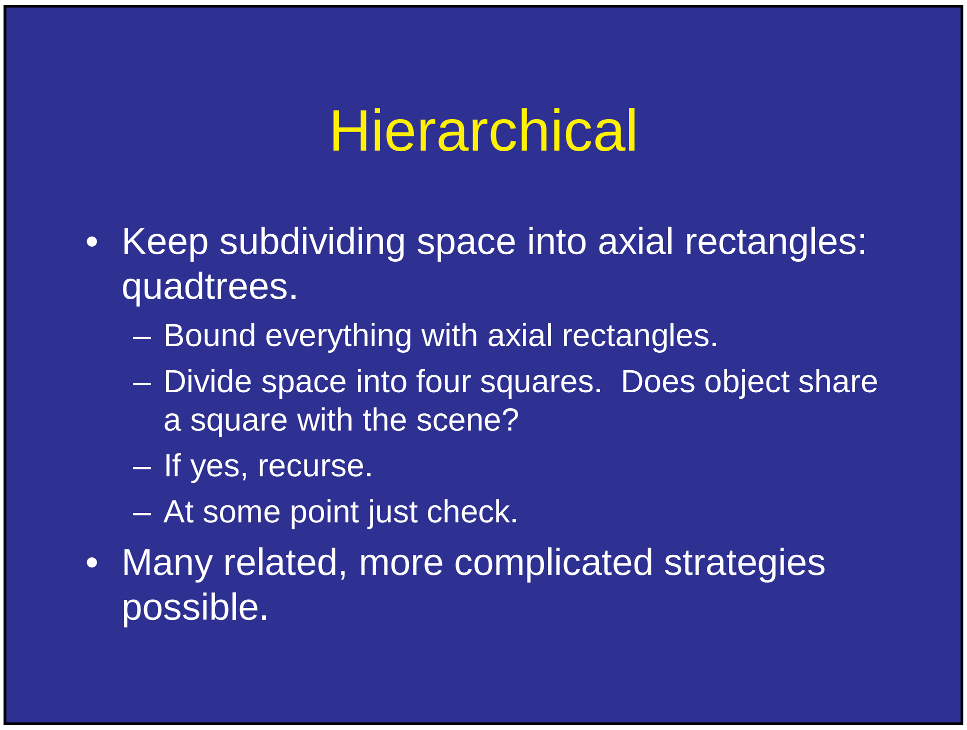
$(x2-x1)^2 + (y2-y1)^2 > (r1+r2)^2$

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.