Collision detection

Multi-Body Problems

- Multi-body problems have many objects that could be colliding
  - Most common case in games is lots of players or agents
  - "Collision" may not mean intersecting, could just mean being close enough to react
- It is essential to avoid considering every pair (n^2 cost)
- Spatial subdivision schemes provide one solution
  - Object interactions are detected in a two step process: which cell am I in, then who else is in my cell?
- Bounding volume schemes provide another, single step, solution
  - Objects supply their own bounds, and overlapping pairs of bounds are found directly

Bounding Volumes?

- Convex-ness is important
- spheres, cylinders, boxes, polyhedra, etc.
- Really you are only going to use spheres, boxes, and polyhedra (...and probably not polyhedra)
- Spheres are mostly used for fast culling
- For boxes and polyhedra, most intersection tests start with point inside-outside tests
  - That's why convexity matters. There is no general inside-outside test for a 3D concave polyhedron.
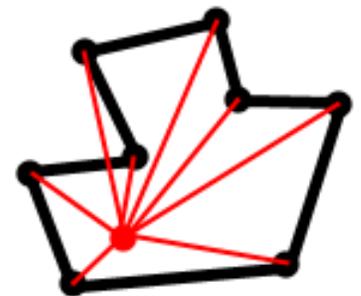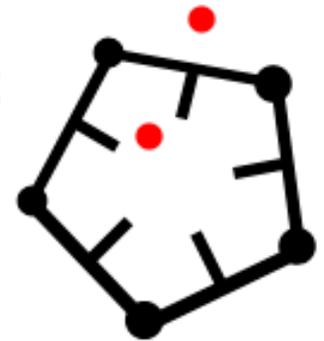
2D Point Inside-Outside Tests

- ## Convex Polygon Test
 - Test point has to be on same side of all ec

- ## Concave Polygon Tests
  - 360 degree angle summation
  - Compute angles between test point and each vertex, inside if they sum to 360
 - Slow, dot product and acos for each angle!

More Concave Polygon Tests

- ## Quadrant Method (see Gamasutra article)
  - Translate poly so test point is origin
  - Walk around polygon edges, find axis crossings
  - +1 for CW crossing, -1 for CCW crossing,
  - Diagonal crossings are +2/-2
  - Keep running total, point inside if final total is +4/-4
- ## Edge Cross Test (see Graphics Gems IV)
  - Take line from test point to a point outside polygon
  - Count polygon edge crossings
  - Even # of crossings, point is outside
  - Odd # of crossings, point is inside
- ## These two are about the same speed
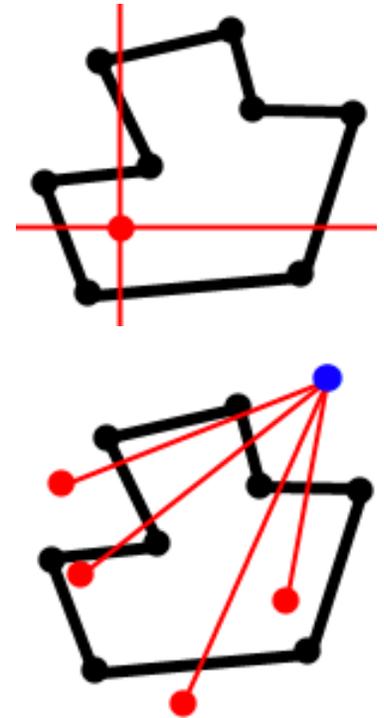
Spheres as Bounding Volumes

- ## Simplest 3D Bounding Volume
  - Center point and radius
- ## Point in/out test:
  - Calculate distance between test point and center point
  - If distance <= radius, point is inside
  - You can save a square root by calculating the squared distance and comparing with the squared radius !!!
  - (this makes things a lot faster)
- It is **ALWAYS** worth it to do a sphere test before any more complicated test. **ALWAYS**. I said **ALWAYS**.
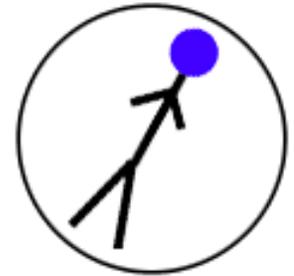
Axis-Aligned Bounding Boxes
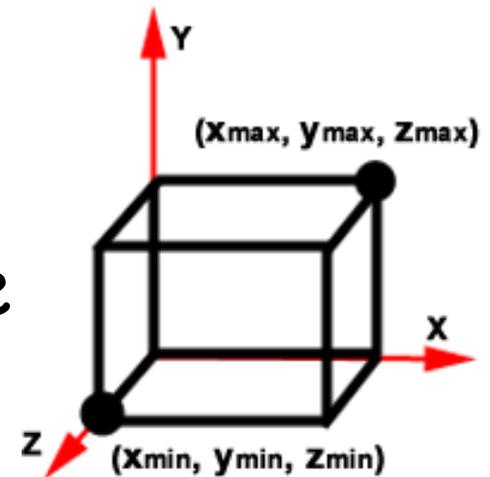
- Specified as two points:

$$(x_{\min}, y_{\min}, z_{\min}), (x_{\max}, y_{\max}, z_{\max})$$

- Normals are easy to calculate
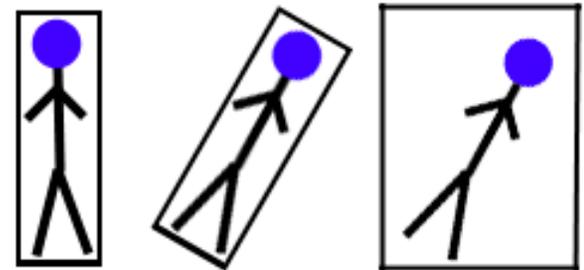- Simple point-inside test:
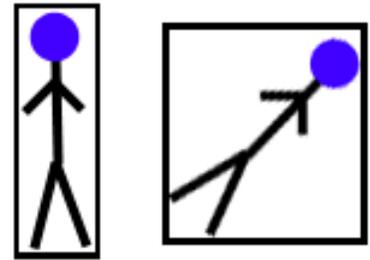
$$x_{\min} \leq x \leq x_{\max}$$

$$y_{\min} \leq y \leq y_{\max}$$

$$z_{\min} \leq z \leq z_{\max}$$

Problems With AABB's

- Not very efficient
- Rotation can be complicated
  - Must rotate all 8 points of box
  - Other option is to rotate model and rebuild AABB, but this is not efficient

Oriented Bounding Boxes

- Center point, 3 normalized axis, 3 edge half-lengths
- Can store as 8 points, sometimes more efficient
  - Can become not-a-box after transformations
- Axis are the 3 face normals
- Better at bounding than spheres and AABB's
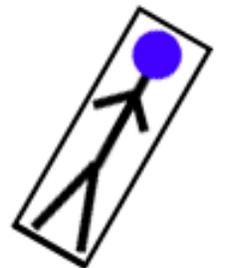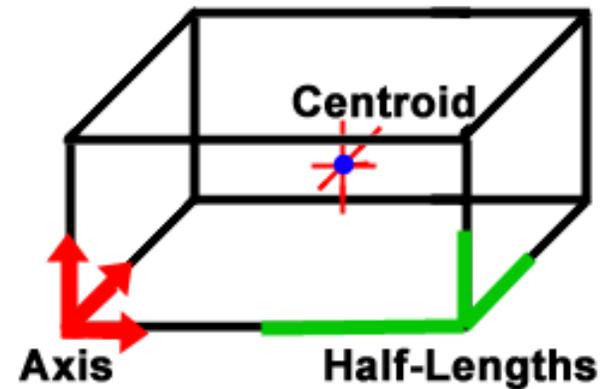
OBB Point Inside/Outside Tests

- ## Plane Equations Test
    - Plug test point into plane equation for all 6 faces
    - If all test results have the same sign, the point is inside (which sign depends on normal orientation, but really doesn't matter)

- ## Smart Plane Equations Test
    - Each pair of opposing faces has same normal, only d changes
    - Test point against d intervals – down to 3 plane tests

$$P' = B_{axis} \bullet P_{test}$$

What if your object isn't a box?
(everything is a box at some scale?)
Bounding Volume Hierarchies

- Basic idea is to build a tree, bottom up, where each level of the tree bounds its children
- Generally used for large numbers of static obstacles and a few moving things
  - Put all the static obstacles into the tree
  - Test for intersection between moving object and tree
- Intersection test looks a lot like testing against an octree or BSP tree
- **The major difference is that a bounding volume hierarchy does not subdivide all of space, only those parts of space that are occupied**
- Also used for subdividing a single object, as we will see later

Heirarchical Bounding Volumes

- Sphere Trees, AABB Trees, OBB Trees
  - Gran Turismo used Sphere Trees
- Trees are built automagically
  - Usually precomputed, fitting is expensive
- Accurate bounding of concave objects
  - Down to polygon level if necessary
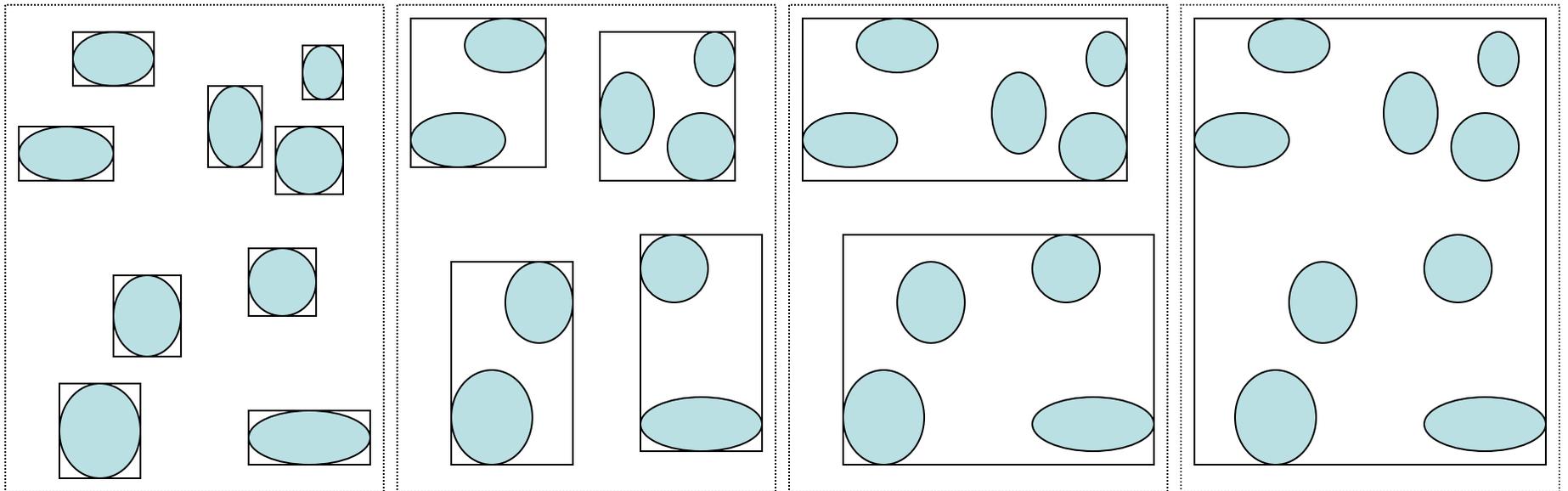 - Still very fast
- See papers on-line
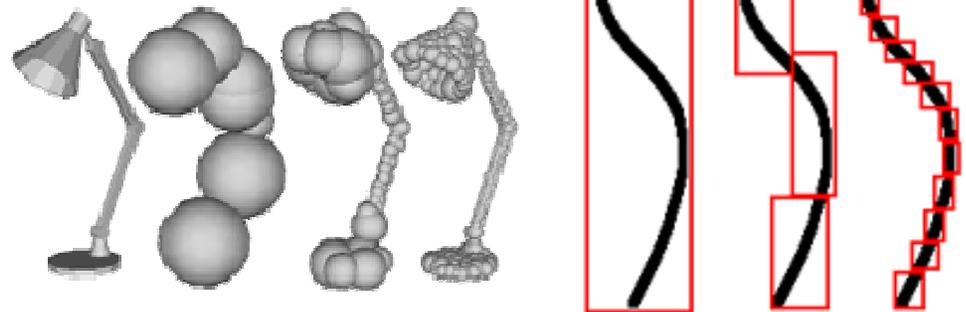
Bounding Tree Example
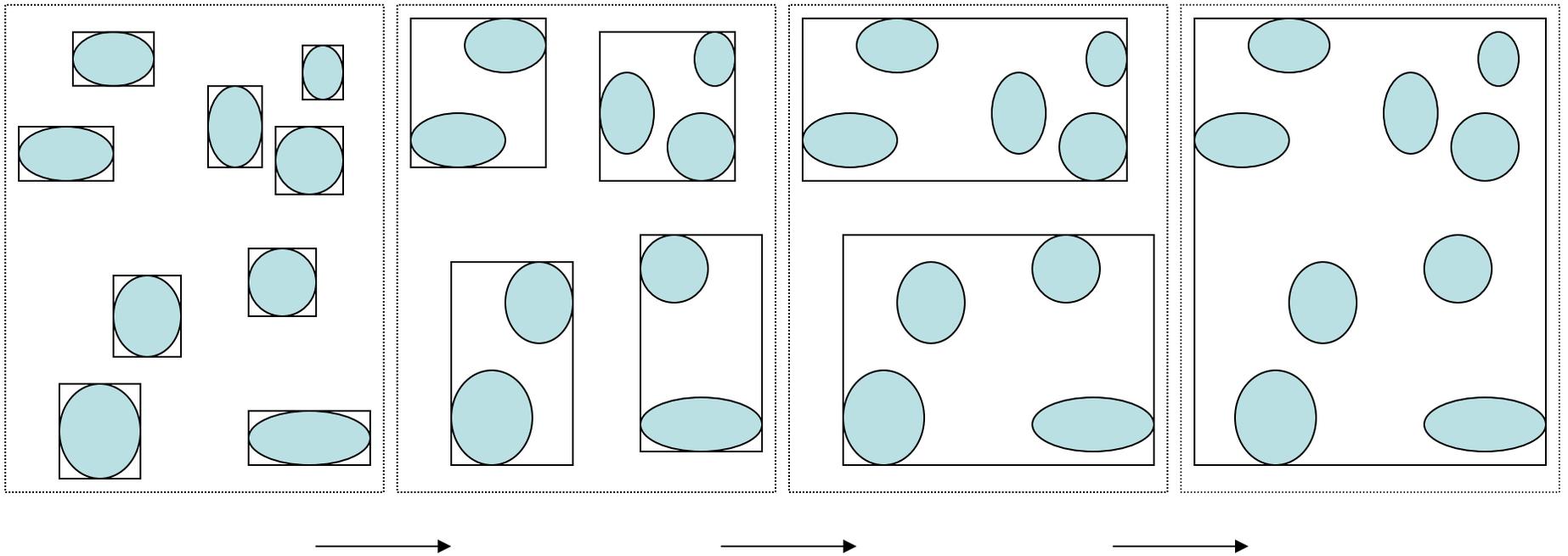
- A Bounding volume hierarchy is an *n*-ary tree in which the bounding volume of each of a nodes *n* children is completely contained in the nodes bounding volume
  - The bounding volumes of two of a nodes children are allowed to overlap, but an object cannot be in more than one of them

Bounding Tree Example

How to choose?

1D Collisions

- It's easy to find all the overlaps between intervals on a line (1D collisions)
  - Sort all the start and endpoints
  - *Sweep* a point the line looking for overlaps
  - Maintain a set of intervals the point is currently inside, initially empty
  - If you reach an new interval start point, report overlap with all the intervals currently in the set, and add the new interval to the set
  - If you reach an interval end point, remove the interval from the set
- A similar algorithm works in 2D for finding overlaps of rectangles, but in 3D, the same algorithm becomes quite expensive
- The solution is to reduce the 3D problem to multiple instances of the 1D or 2D problem - *reduce the dimension*

Dimension Reduction

- Algorithm works with AABBs
- In order for two AABBs to overlap in 3D, they must overlap along every 1D axis
  - This is a special case of the separating plane theorem, more on it later
- Sort the box extents in each dimension
- Find all the 1D overlaps
- Tag pairs that overlap in all dimensions (can be done without overhead on the 1D problems)
- How expensive is this?
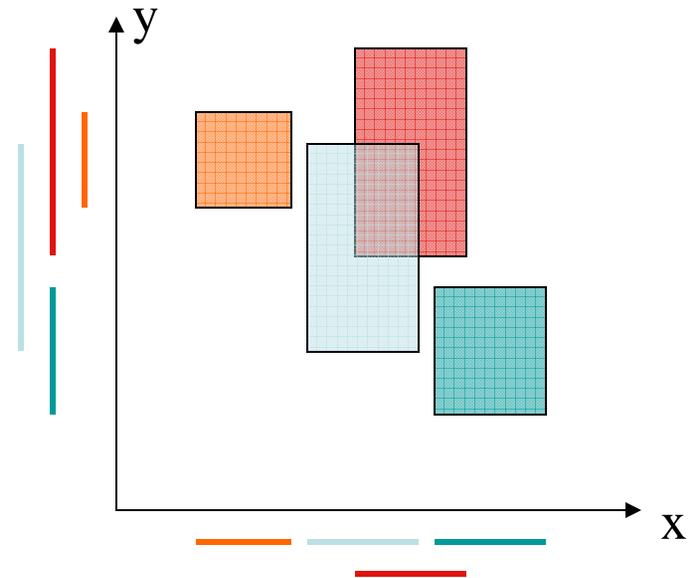
- X overlaps: green-red, red-blue
- Y overlaps: orange-red,orange green, green-red, green-blue
- 2D overlaps: green-red

Bubble SORT!

- Sometime you can claim that the sorted order of the bound extents will not change much between tests
  - If objects don't move far in each frame, and don't frequently change direction
    - This assumption fails for bouncing objects
- Using bubble sort, the cost of sorting depends on the number of swaps in sort order (linear + #swaps)
  - Bubble sort looks at each element in the list, and "bubbles" it down until it is greater than the preceding element
- If the sort order doesn't change much, bubble sort is nearly linear, hence finding overlaps is nearly linear
  - Requires some messy coding and some other data structures to handle big numbers of objects without n^2 storage cost

(not aabb)

- We'll look at some important cases, and some that are the base-case for other schemes
  - Testing spheres against anything, because it's fast and spheres are commonly used bounding volumes
  - Testing two triangles, because meshes are ultimately made up of triangles, as is cloth, subdivision surfaces, …
  - Testing AABBs against each other, and testing OBBs against each other
- Along the way we will learn about *Voronoi regions* and *separating planes*

Sphere-Anything Testing

- The key observation is that collision testing with a sphere can be converted into a closest point problem:

  > **Given a point and an object, find the point on the object that is closest to the given point**

- To test for collision with a sphere, find out if the point closest to the sphere's center is closer than the sphere's radius
  - Small problem if the sphere is inside the object
- There is a different way to solve the closest point problem for each type of geometry
  - We'll look at other spheres, plane, triangles and boxes, and a hack

Sphere-Sphere and Sphere-Plane

- To test two spheres for collision:
  - Compute the square of the Euclidian distance between their centers
  - Compare it to the sum of the radii, squared
  - In math speak: $(c_1-c_2) \cdot (c_1-c_2) < (r_1+r_2)^2$
  - How do you find the closest points?
- To test a sphere against a plane:
  - Find the distance from the sphere to the plane
  - For a plane given by the equation: $n \cdot x + d = 0$, what is the computation?
  - Test the distance against the radius
  - How do you find the closest points?

Sphere-Triangle Testing

- First do a sphere-plane test with the triangle's plane
  - What case can we eliminate?
- Several ways to proceed:
  - Find the closest point on the plane, find the closest point in 2D, then test the distance against the radius of the sphere
  - Test each triangle edge against the sphere
    - Not enough, what else do you have to test?
  - Use a minimization procedure in 3D
    - Requires the idea of *Voronoi regions*

Voronoi Regions

- A Voronoi region is associated with each *feature* of an object: each vertex, edge and face
- Each region is the set of points that are closest to that feature
  - For a triangle, sets are infinite
  - Sets are bounded by planes (in 3D)
  - How are the planes defined?
- To find the closest feature to a given point, find the Voronoi region that contains it
  - Easy to exploit coherence (remember the closest feature from the last frame)

Vert 2

Edge 2

Edge 1

Face

Vert 1

Edge 3
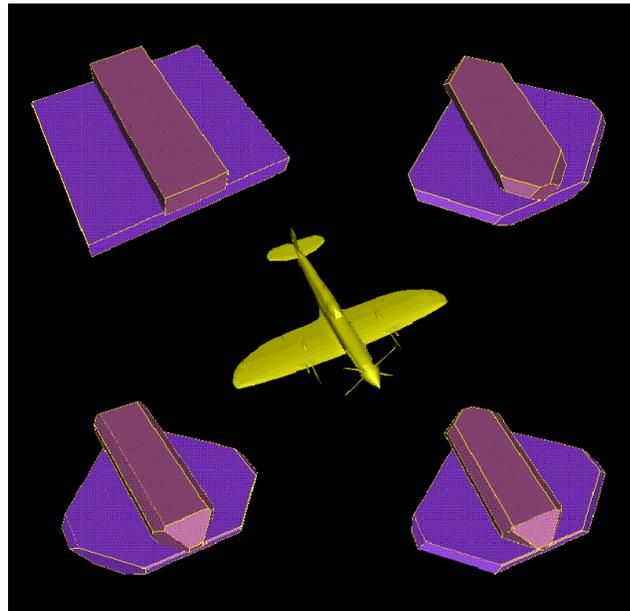
Vert 3
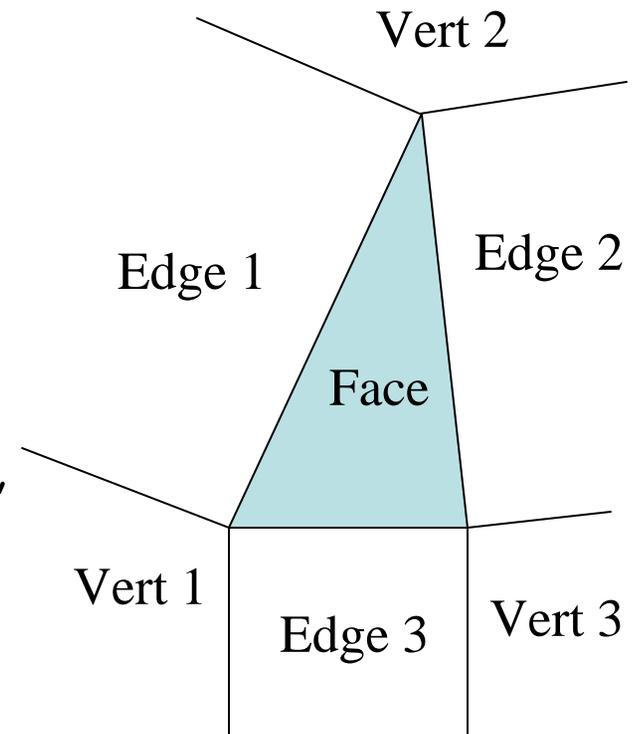
Triangle Voronoi Example
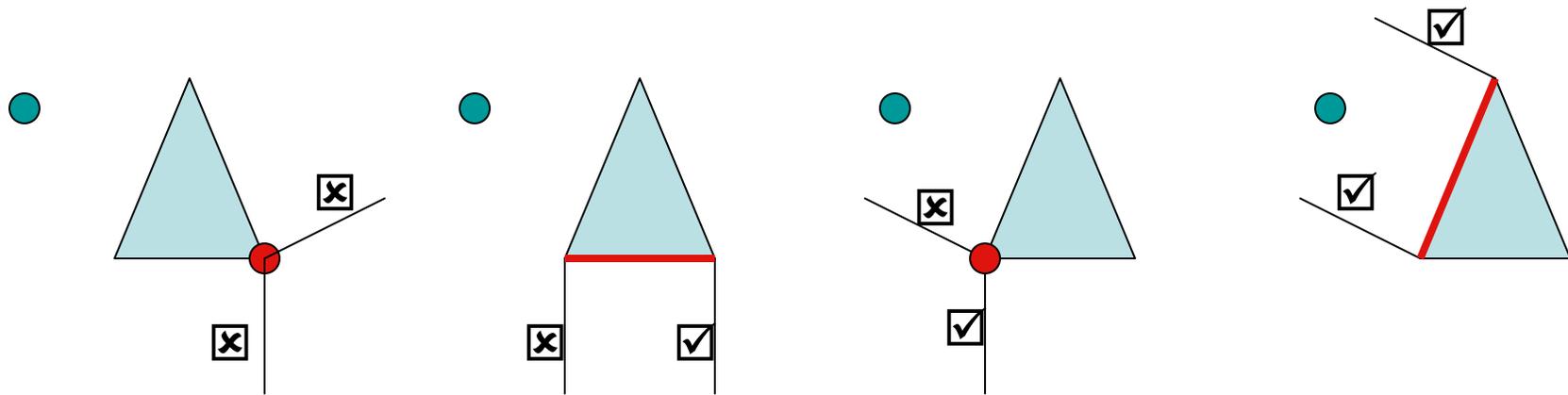
Outside Voronoi
region: planes in error
guide us to next region

Done

Note that regions share planes, so plane tests can be cached.
Also, algorithm cannot infinite loop if careful.

Trade offs in selection of BHV

- Tighter fit ⇨ increased cost of comparison
- Low complexity Polygons ⇨ fast comparison but requires more BV's
- For complex polygons (N-sub-bv) and N-sub-p are lower and C-sub-bv is higher

Heirarchical Bounding Volumes

- Sphere Trees, AABB Trees, OBB Trees
  - Gran Turismo used Sphere Trees
- Trees are built automagically
  - Usually precomputed, fitting is expensive
- Accurate bounding of concave objects
  - Down to polygon level if necessary
  - Still very fast
- See papers on-line