Week 5: Mechanics II
Part 4: Simplifying Collisions

COMP270: Mathematics for 3D Worlds and Simulations

Objectives

- **Introduce** some techniques for optimising collision detection with large numbers of objects

The problem(s)

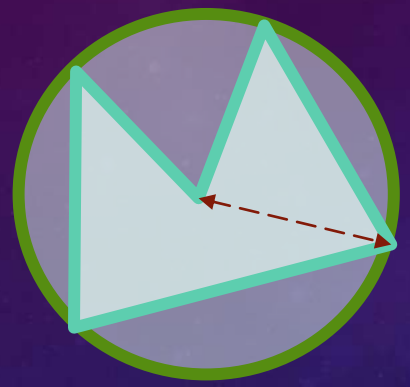
- Not every shape is a point, line, square or a circle
- Intersection tests with arbitrary polygons can be complex
- There may be many (many) objects in a scene
- Collision testing is done in pairs

$O(n^2)$

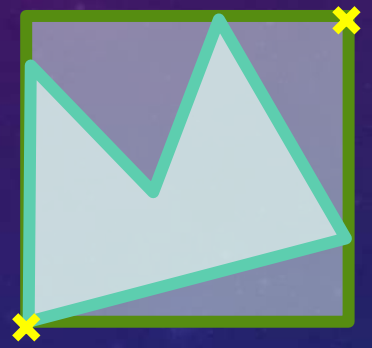
$\Rightarrow$ If there are n objects in a scene, you could have $\frac{n(n-1)}{2}$ complex tests…

The first object could collide with $n-1$ objects, the second with $n-2$ etc.

Strategy 1: bounding shapes



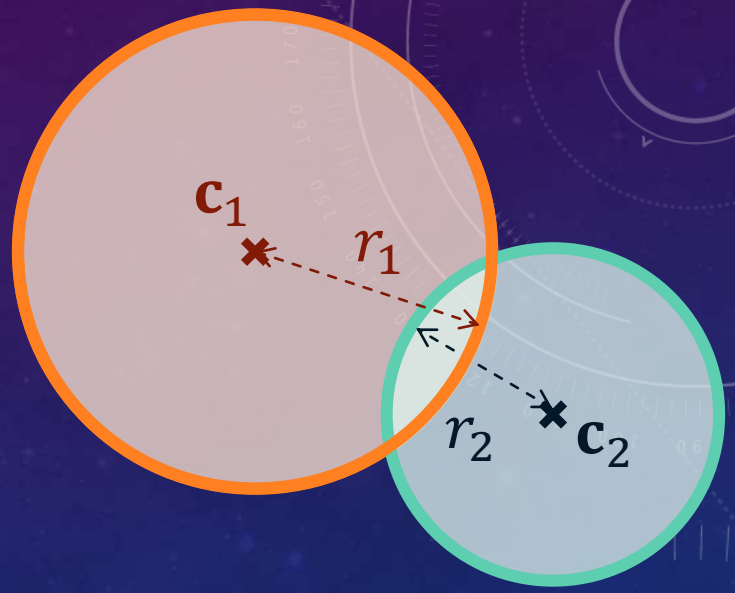
- Radius = max. distance of any vertex from the centre
- Centre = average vertex position



- Min. x = smallest x of any vertex
- Max. x = largest x of any vertex
- Min. y = smallest y of any vertex
- Max. y = largest y of any vertex

Recap: circle-circle intersection

- Collide iff $\|\mathbf{c}_1 - \mathbf{c}_2\| \leq r_1 + r_2$



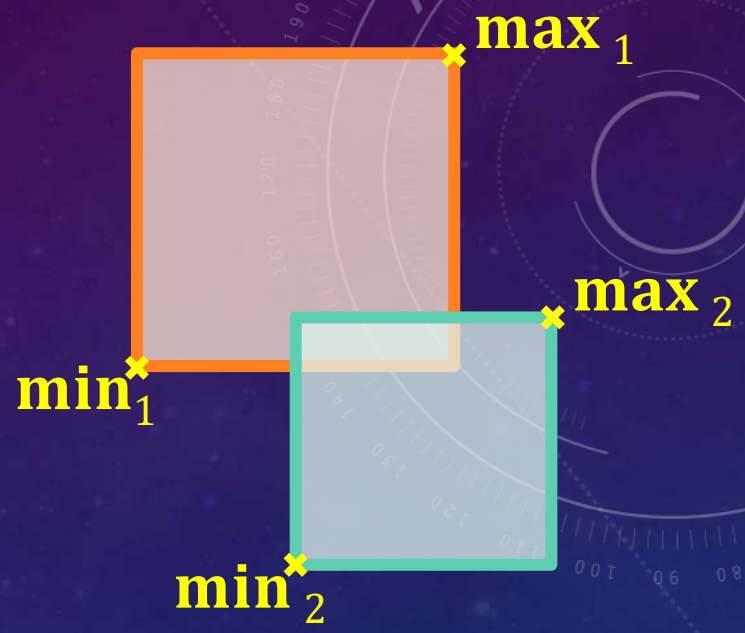

```
bool circlesCollide(Vector2 c1, float r1,
                    Vector2 c2, float r2) {
   return (c2 - c1).magnitude() <= r1 + r2;
}
```

Or avoiding `sqrt()`:

```
Vector2 centreDiff = c2 - c1;
float radSum = r1 + r2;
return centreDiff.dot(centreDiff) <= radSum * radSum;
```

Recap: box-box intersection

- Collide iff the min component of one box is not greater than the max of the other

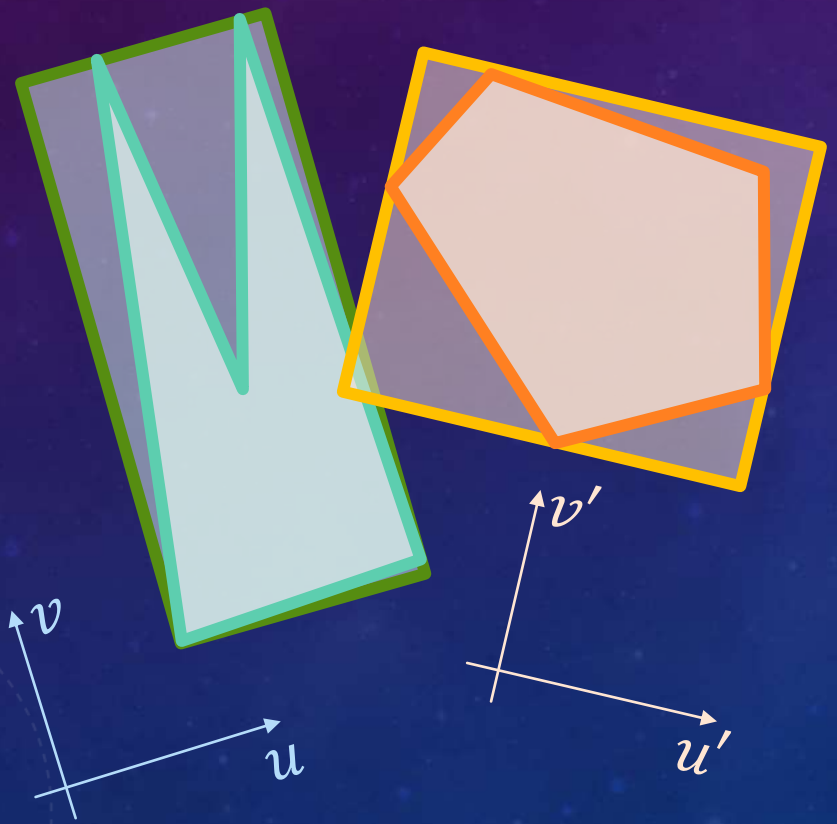



```
bool boxesCollide(Vector2 box1_min, Vector2 box1_max,
                  Vector2 box2_min, Vector2 box2_max) {
    if (box1_min.x >= box2.max.x) return false;
    if (box1_max.x <= box2.min.x) return false;
    if (box1_min.y >= box2.max.y) return false;
    if (box1_max.y <= box2.min.y) return false;
    return true;
}
```

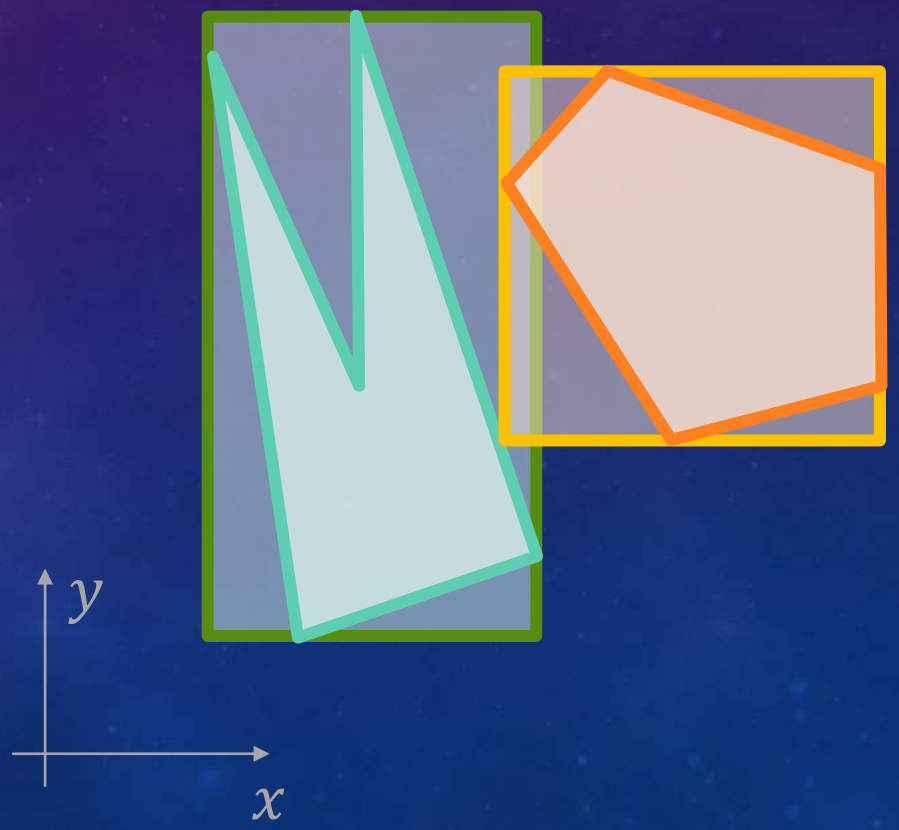
Circle vs. box

Axis aligned bounding boxes (AABBs)

Object orientation



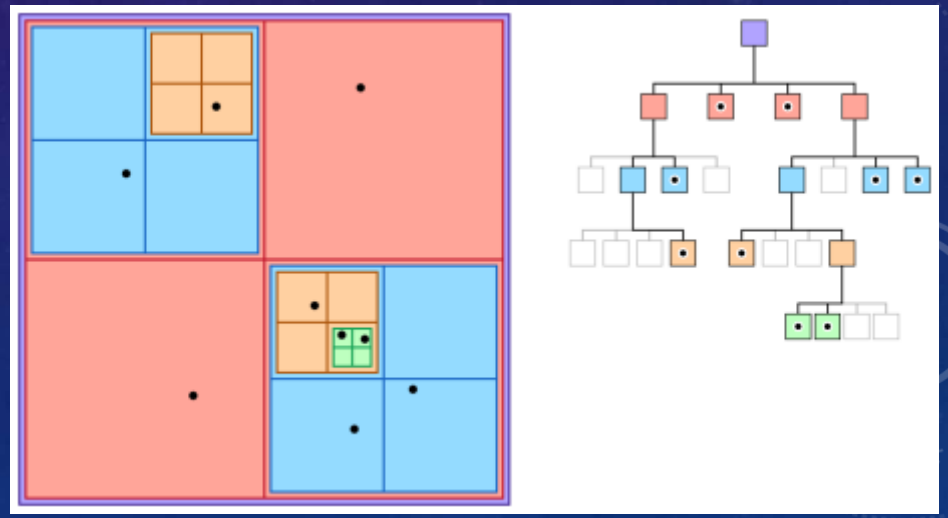
(World) axis aligned



Dealing with many objects

Spatial data structures: quadtree

- A quadtree is a tree data structure in which space is recursively divided into four sections (quadrants)
  - Non-leaf nodes are parent/child quadrants
  - Leaf nodes are the objects (points), so that each “cell” contains only one object
  - Cells under the same parent node are close to each other
  - More details here

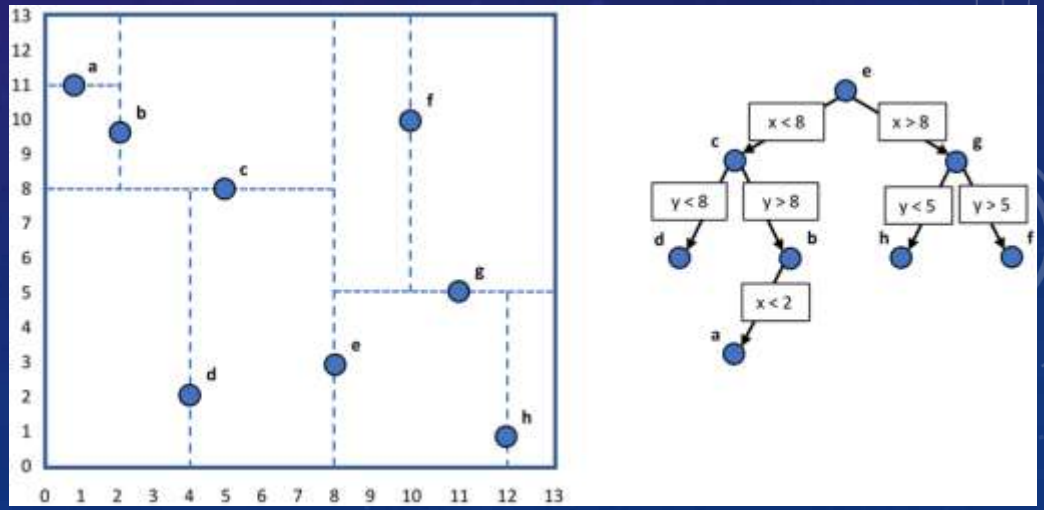


https://developer.apple.com/documentation/gameplaykit/gkquadtree

Spatial data structures: K-d tree

- A K-d tree is a binary data structure for organising points in a *k*-dimensional space
  - All nodes are objects/points
  - Non-leaf nodes represent a division of space into half-spaces by a hyperplane defined by one of the points
  - More details here

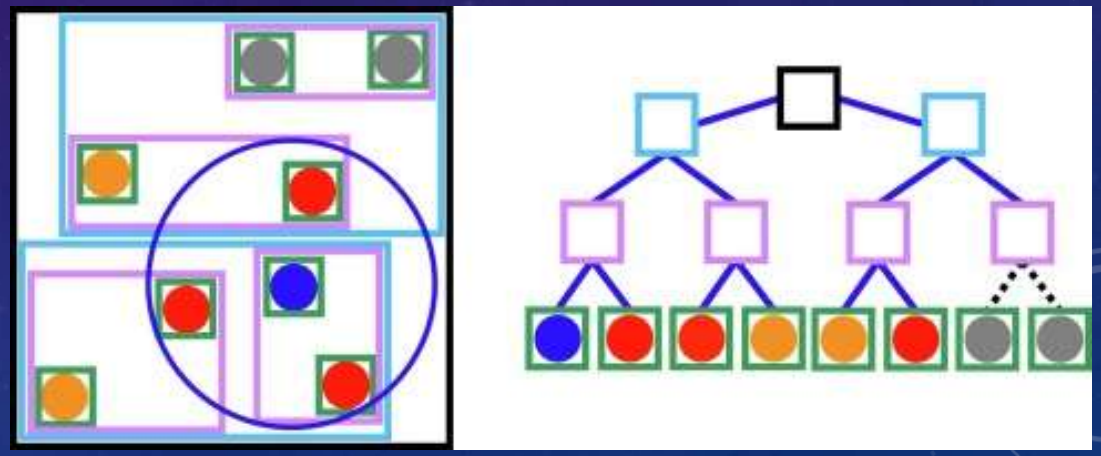
A line in 2D



https://www.researchgate.net/figure/An-example-two-dimensional-k-d-tree-k-2-built-from-nodes-a-through-h-Dividing-planes_fig2_314298746

Spatial data structures: BVH

- A bounding volume hierarchy (BVH) is a tree structure containing a set of geometric objects enclosed in bounding volumes
  - Leaf nodes are the object bounding volumes (AABBs)
  - Groups of nearby BVs are enclosed in larger BVs
  - A single object may have multiple BVs for separate components
  - More details here



https://www.sciencedirect.com/science/article/abs/pii/S0927025619302060X

Summary: simulation tips

- Simplify where necessary:
  - Use constant accelerations
  - Ignore factors that don’t contribute to the overall effect
  - Approximate shapes
- Avoid unnecessary computations:
  - Store objects in a way that makes sense, e.g. spatial subdivision