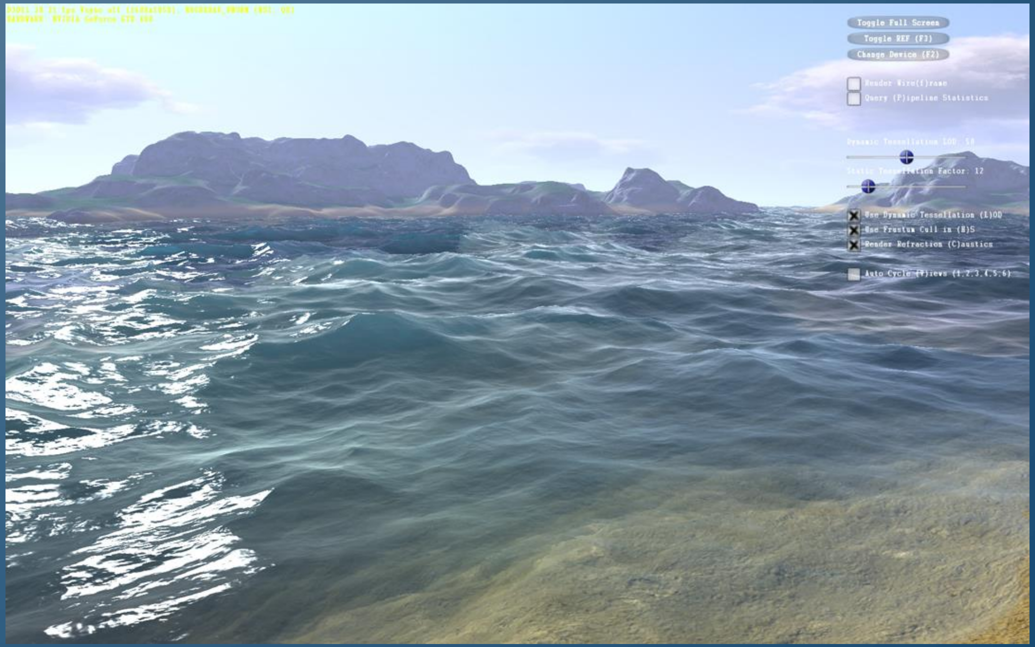
# GPAT – CHAPTER 7 PHYSICS

# PHYSICS OVERVIEW

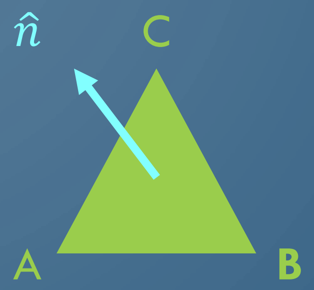
- Physics in games involves these two basic elements:
  - Object-object interaction (Geometry)
    - Collision detection
    - Collision response
  - Mechanics (Calculus)
    - Object movement
- Also can be used to simulate
  - Visual effects such as water dynamics
  - More realistic sound and light effects
  - However, these are often too slow for gaming

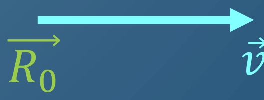


# COLLISION DETECTION

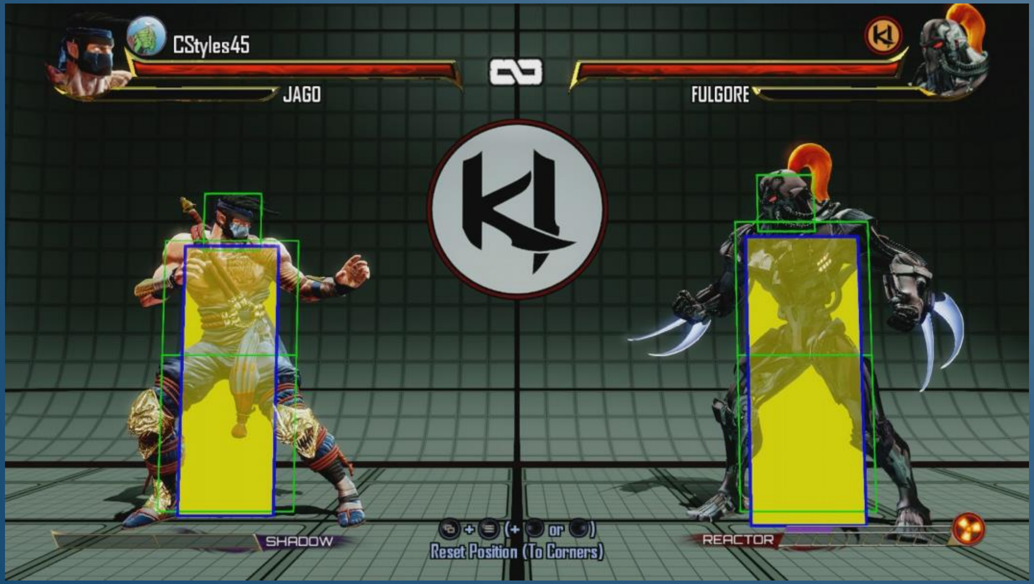
# BASIC GEOMETRY

- Planes
  - Point $\vec{p}$, normal $\hat{n}$
  - $\vec{p} \cdot \hat{n} + d = 0$
  - Easy to derive from triangle
- Rays
  - Point $\overrightarrow{R_0}$, direction $\vec{v}$, parametric value $t \in [0, \infty)$
  - $\overrightarrow{R_0} + \vec{v}t$
- Line segments
  - Same as ray except $t \in [0,1]$ (i.e., two endpoints)
- A **ray cast** involves extending a ray into the scene to determine interaction with objects, e.g., ballistics in FPS games. Often computed by line segments and not actual rays



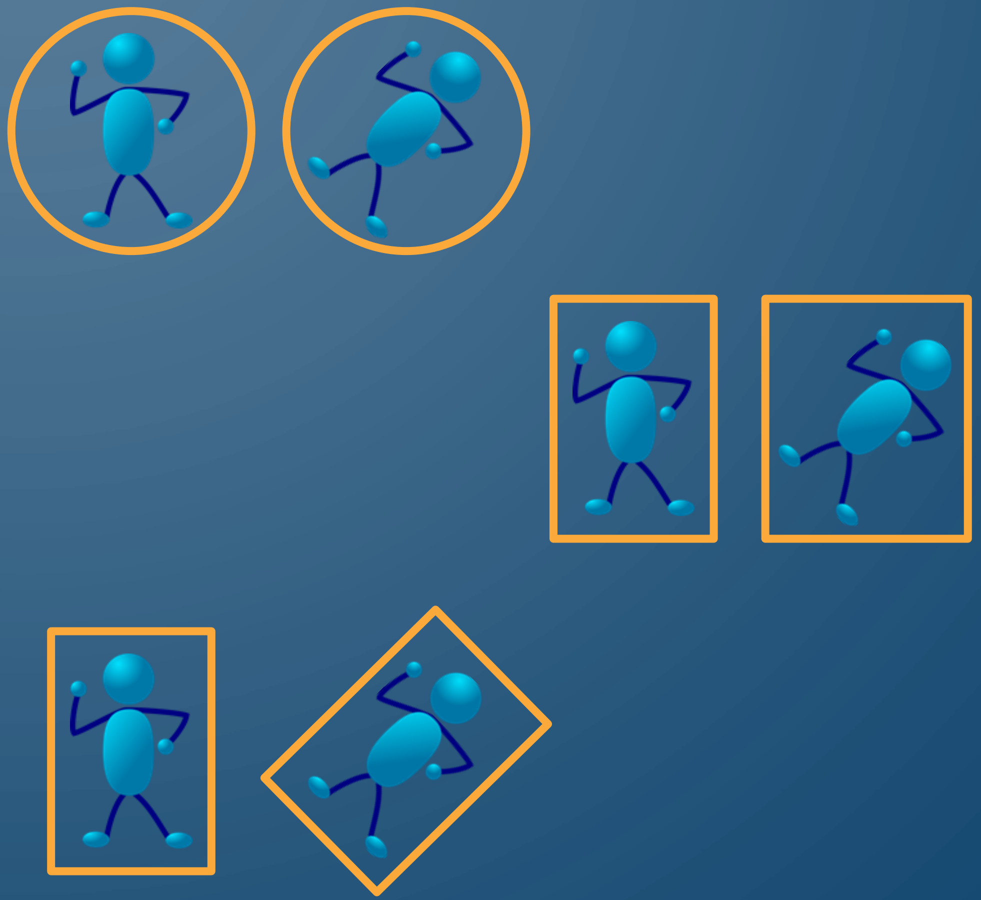

# COLLISION GEOMETRIES



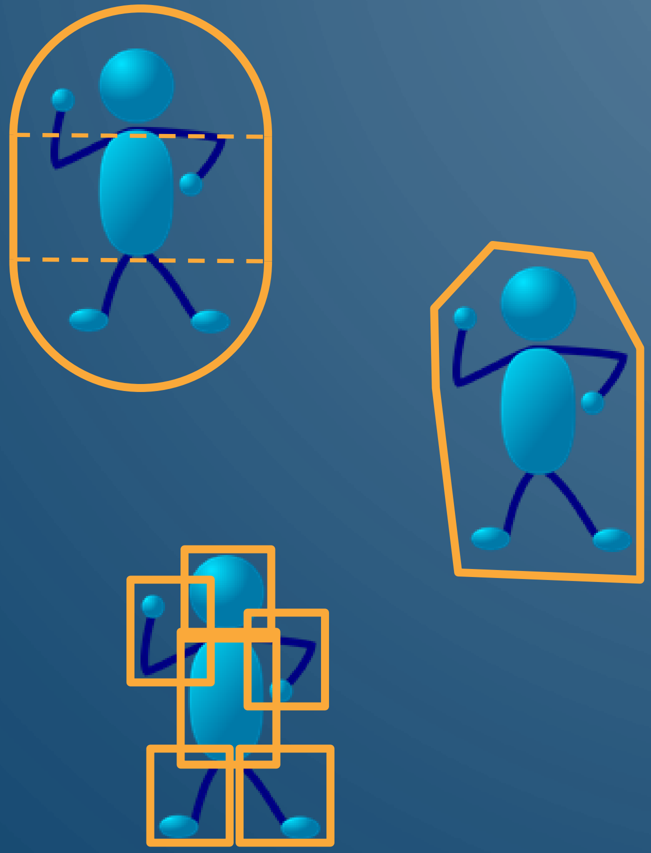

- A separate geometric representation of objects is used instead of the real mesh geometry for collision detection
  - Compare a box (12 triangles) to a complex model for an avatar (>15,000 triangles)
  - Leads to false positives

# COLLISION GEOMETRIES

- Bounding Spheres (BSs)
  - Center
  - Radius
- Axis-Aligned Bounding Boxes (AABBs)
  - Min/Max in each dimension (2 points)
- Oriented Bounding Boxes (OBBs)
  - 8 vertices or 6 planes



# COLLISION GEOMETRIES

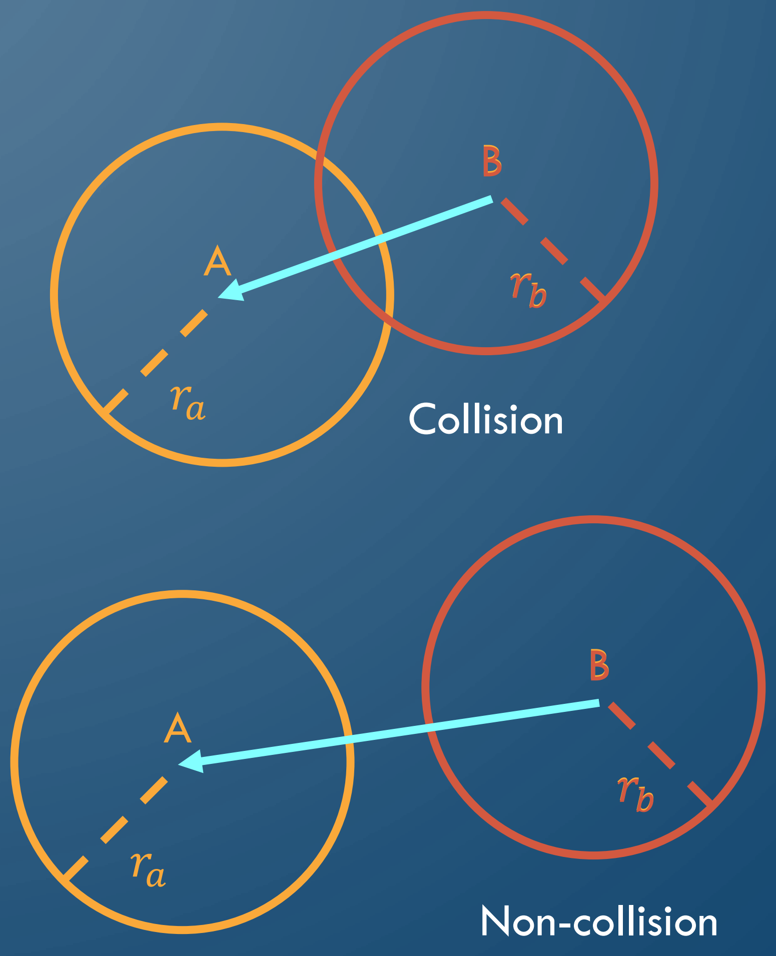


- Capsules
  - 2 points
  - Radius
- Convex Polyhedrons (Convex Hulls)
  - Mesh
- List of Geometries

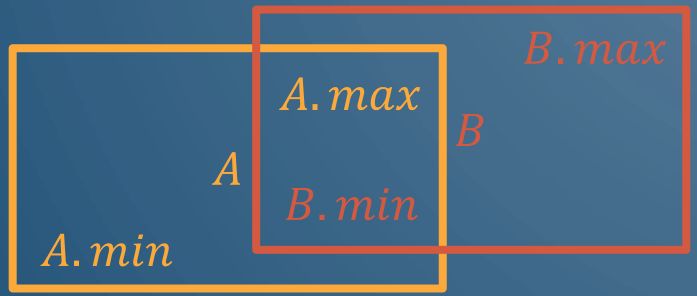
# BS-BS COLLISION DETECTION

- Collision if

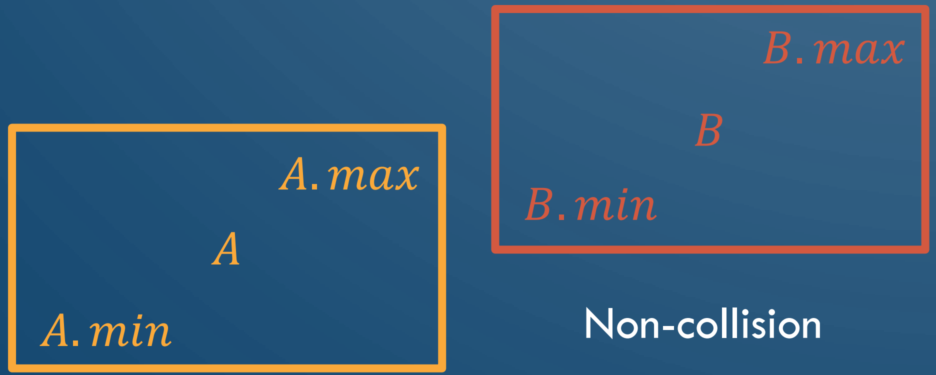
$$\|A - B\|^2 < (r_a + r_b)^2$$



# AABB-AABB COLLISION DETECTION




Collision




Non-collision

- For 2D-boxes A and B:

$$A.max.x > B.min.x \wedge$$
$$B.max.x > A.min.x \wedge$$
$$A.max.y > B.min.y \wedge$$
$$B.max.y > A.min.Y$$

# LINE SEGMENT-PLANE COLLISION DETECTION

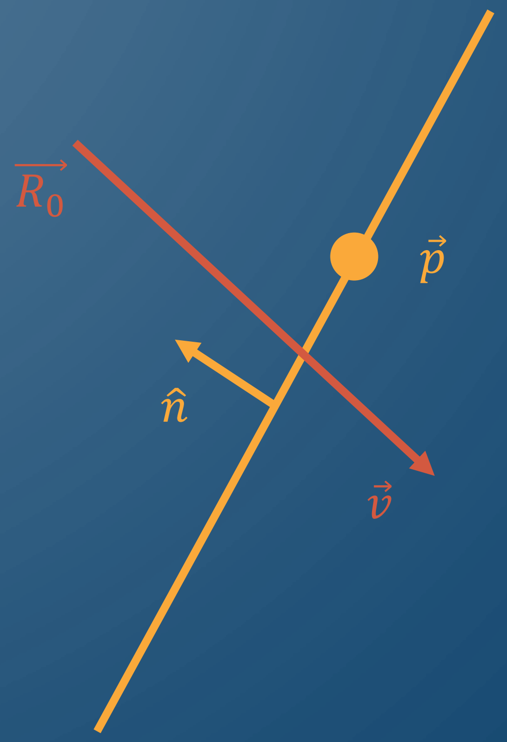
- Given the two equations:

$$\overrightarrow{R_0} + \vec{v}t$$
$$\vec{p} \cdot \hat{n} + d = 0$$

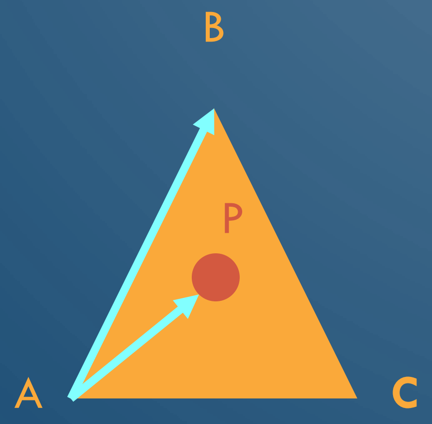
- We solve for their intersection (a point on that plane):

$$\left(\overrightarrow{R_0} + \vec{v}t\right) \cdot \hat{n} + d = 0$$
$$\overrightarrow{R_0} \cdot \hat{n} + (\vec{v} \cdot \hat{n})t + d = 0$$
$$t = \frac{-\left(\overrightarrow{R_0} \cdot \hat{n} + d\right)}{\vec{v} \cdot \hat{n}}$$

- If $t \in [0,1]$ then there is a collision, else non-collision
  - Plug back in if you need the point



# LINE SEGMENT-TRIANGLE COLLISION DETECTION




- First figure out the point hits the plane that the triangle lies in
- Next we determine if that point lies in the triangle
- Key idea is to determine if the point is on the same side of each edge of the triangle:

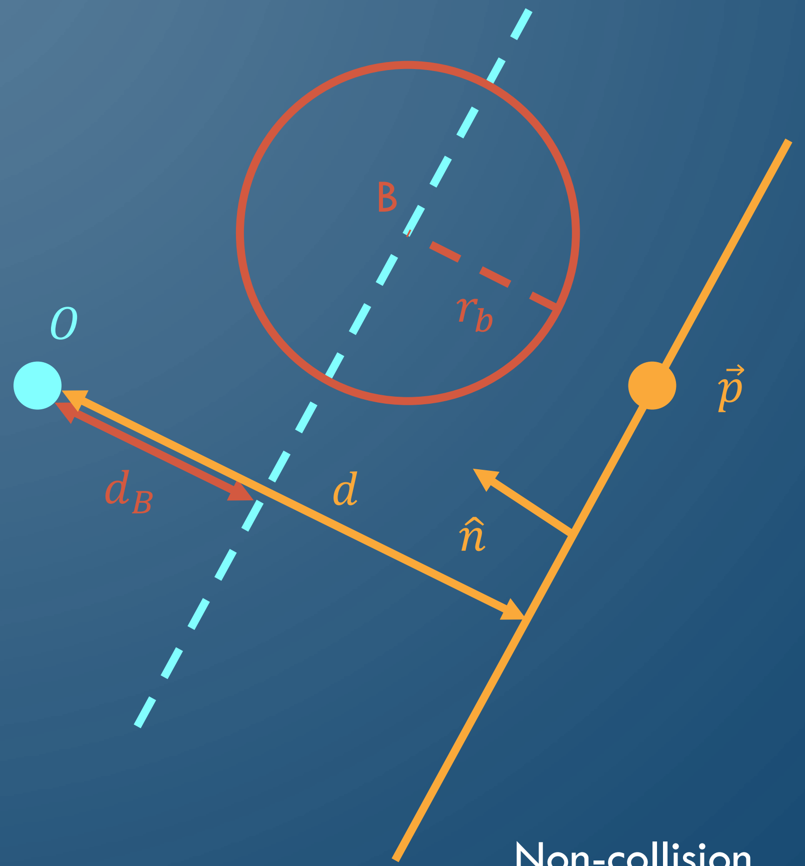
$$\left(\overrightarrow{AB} \times \overrightarrow{AP}\right) \cdot \hat{n} > 0$$

- Can be more efficient with Barycentric coordinates

# BS-PLANE COLLISION DETECTION

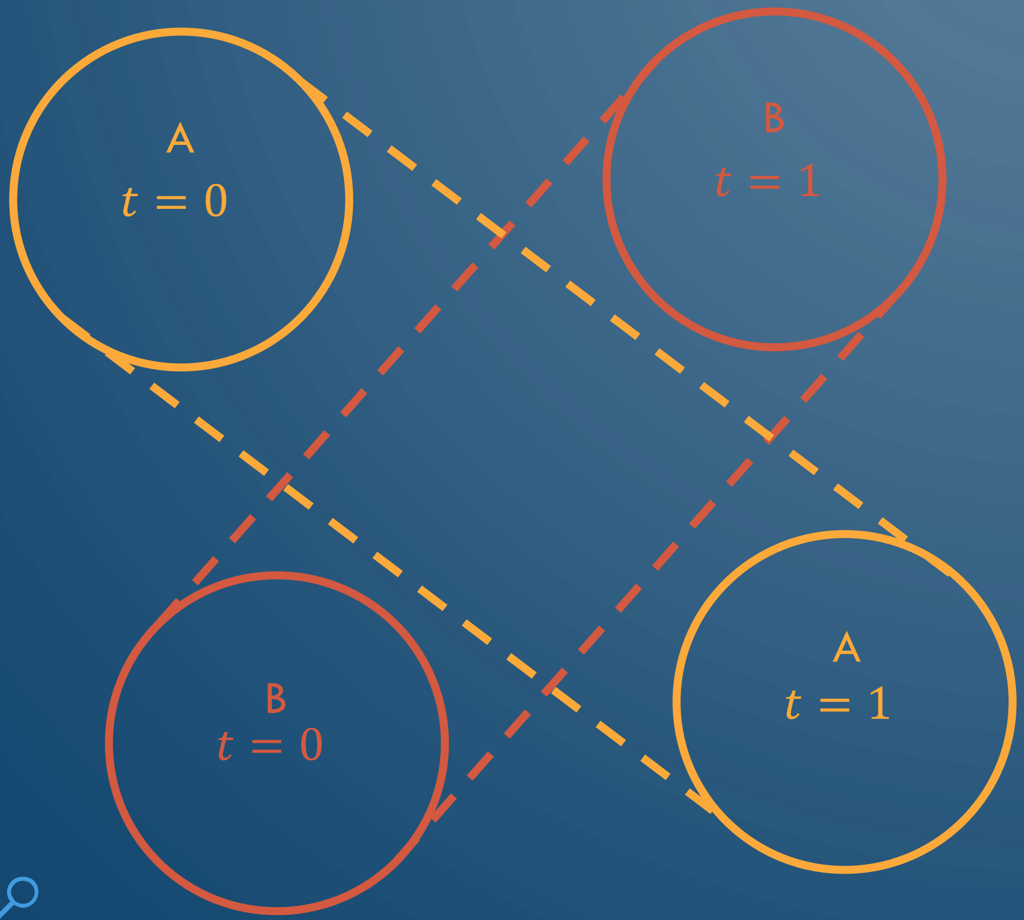
- Essentially, find a hypothetical plane parallel to the first plane and through the center of the sphere
- Compare the difference of the plane's $d$ values to the spheres radius
- Intersection if:

$$d_B = -B \cdot \hat{n}$$
$$|d - d_B| < r_b$$



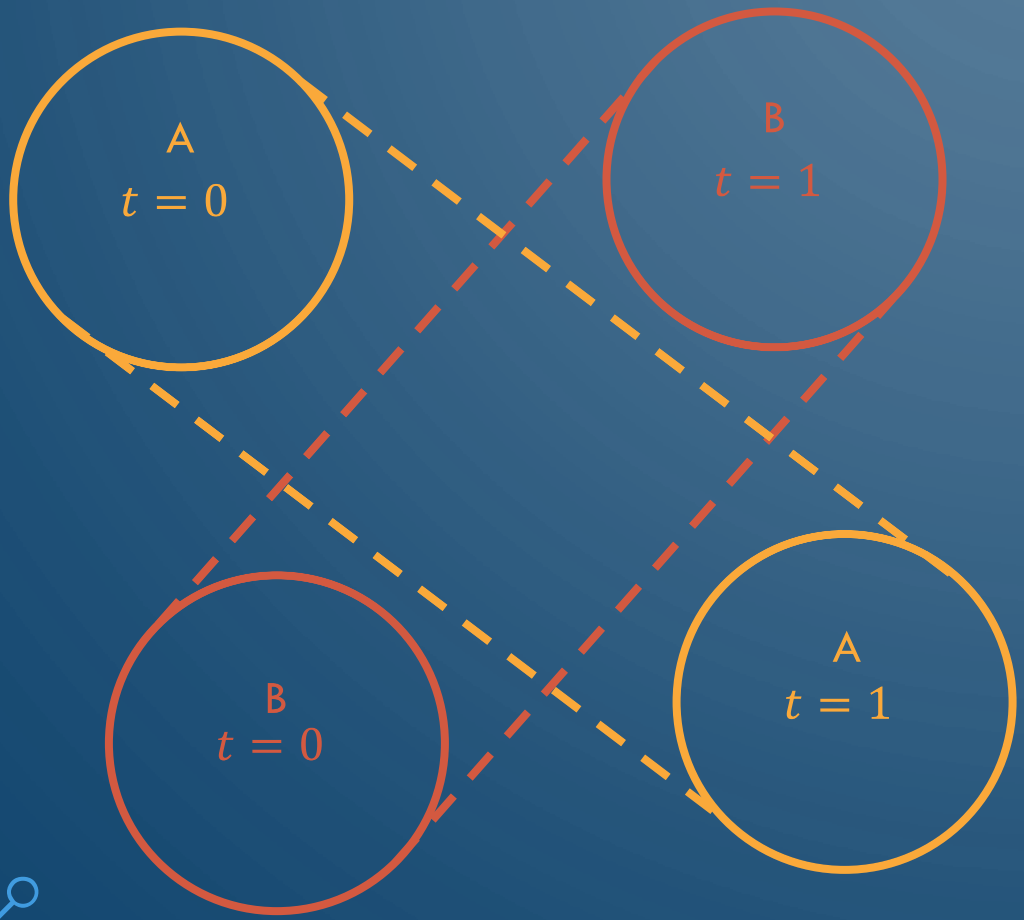

Non-collision

# SWEPT BS COLLISION DETECTION




- Essentially, a version of **continuous collision detection** (or collision detection for capsules)
- Again, construct parametric equations and solve
- Specifically, construct rays for the motion of the object centers and find the point where the distance between the rays is the same as the sum of the radii

# SWEPT BS COLLISION DETECTION



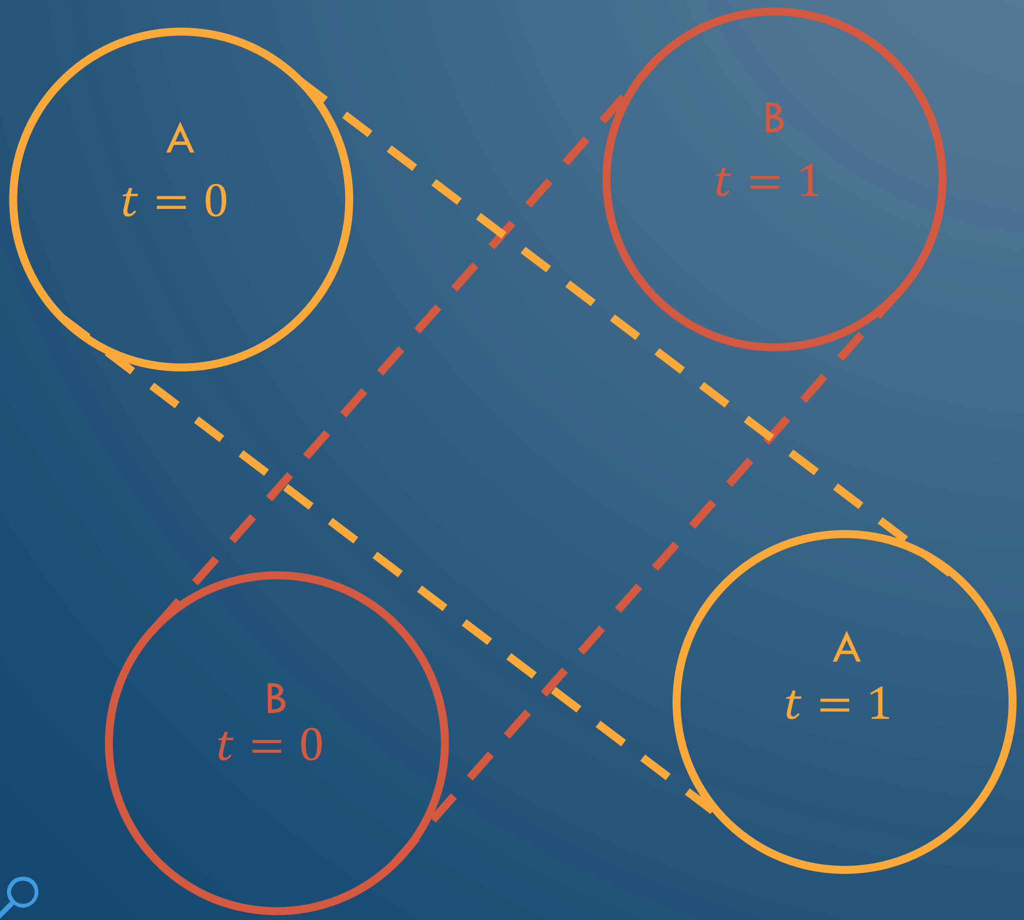

- Center motion:

$$\overrightarrow{A_t} = \overrightarrow{A_0} + \overrightarrow{v_A}t$$
$$\overrightarrow{B_t} = \overrightarrow{B_0} + \overrightarrow{v_B}t$$

- Solve for $t$ in:

$$\left\|\overrightarrow{A_t} - \overrightarrow{B_t}\right\| = r_A + r_B$$
$$\left(\overrightarrow{A_t} - \overrightarrow{B_t}\right) \cdot \left(\overrightarrow{A_t} - \overrightarrow{B_t}\right) = (r_A + r_B)^2$$

# SWEPT BS COLLISION DETECTION



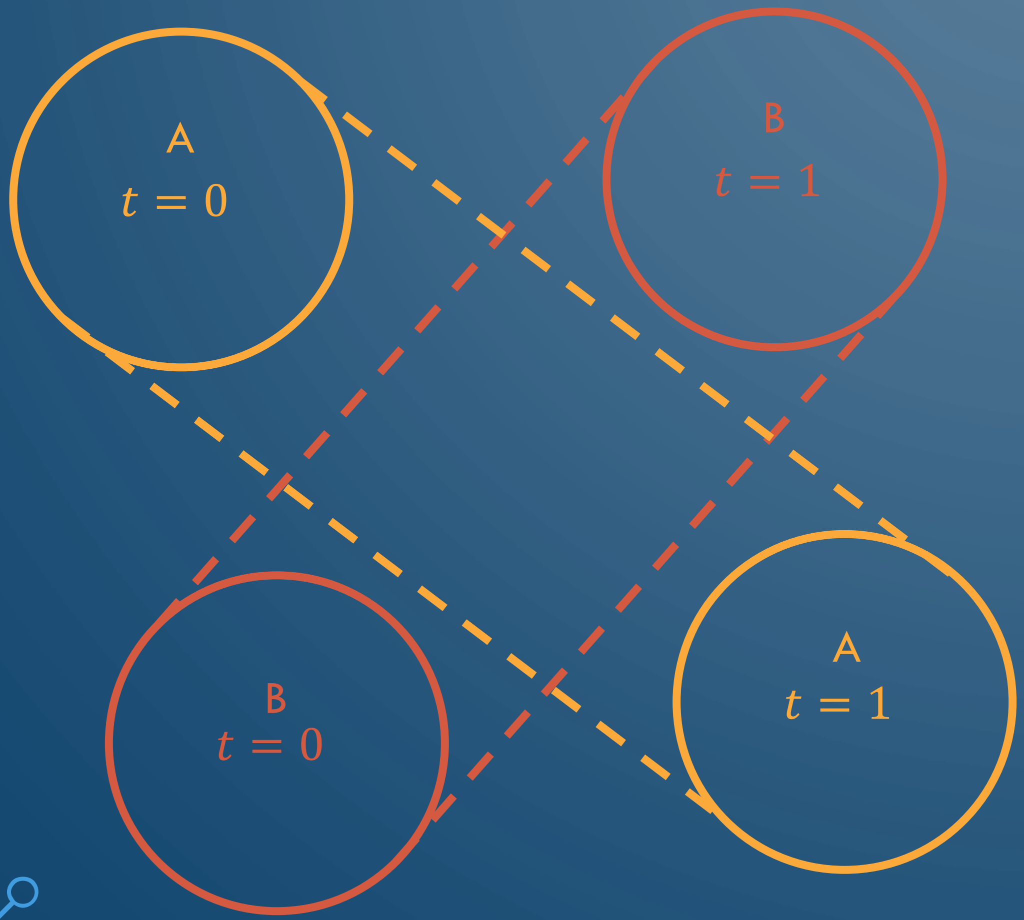

- $\left(\overrightarrow{A_t} - \overrightarrow{B_t}\right) \cdot \left(\overrightarrow{A_t} - \overrightarrow{B_t}\right) = (r_A + r_B)^2$
- Looking at $\overrightarrow{A_t} - \overrightarrow{B_t}$:

$$\overrightarrow{A_0} + \overrightarrow{v_A}t - \overrightarrow{B_0} - \overrightarrow{v_B}t$$
$$\overrightarrow{A_0} - \overrightarrow{B_0} + (\overrightarrow{v_A} - \overrightarrow{v_B})t$$

- Let $\vec{C} = \overrightarrow{A_0} - \overrightarrow{B_0}, \vec{D} = \overrightarrow{v_A} - \overrightarrow{v_B}$ so:

$$\left(\vec{C} + \vec{D}t\right) \cdot \left(\vec{C} + \vec{D}t\right) = (r_A + r_B)^2$$

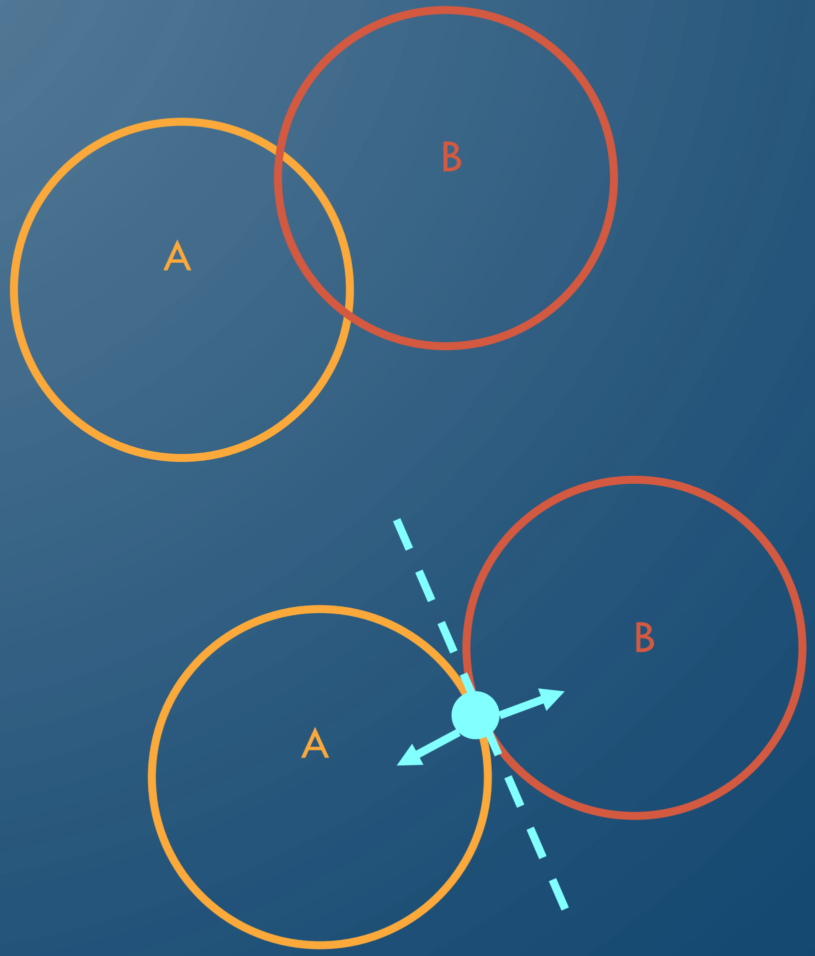
# SWEPT BS COLLISION DETECTION



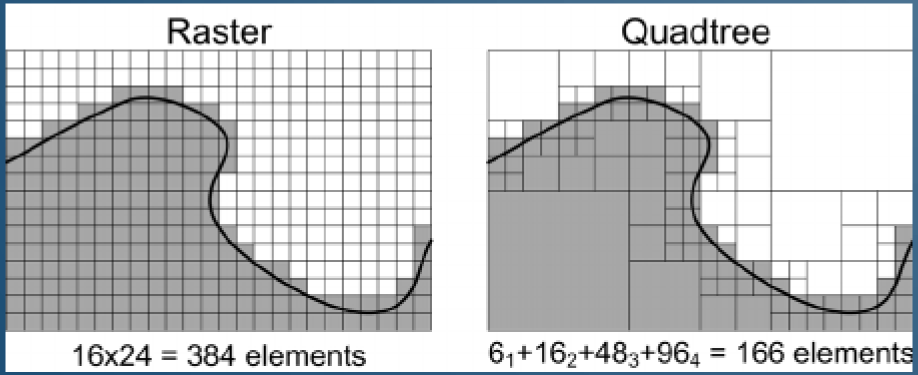

- Expanding the dot product:
  $\vec{C} \cdot \vec{C} + 2(\vec{C} \cdot \vec{D})t + (\vec{D} \cdot \vec{D})t^2 = (r_A + r_B)^2$
- Let $a = \vec{D} \cdot \vec{D}, b = \vec{C} \cdot \vec{D},$
  $c = \vec{C} \cdot \vec{C} - (r_A + r_B)^2$, so:
  $$at^2 + bt + c = 0$$
- Solve with quadratic equation, and if the discriminant is greater than 0 and further $t \in [0,1]$, then there is a collision. Otherwise no collision (or tangent)

# COLLISION RESPONSE

- Simple examples
  - Objects "die"
  - One object loses health
- Complex interactions
  - Need to determine exact point of collision (LERP)
  - Need to determine normals at point of collision
  - Reflect and scale velocities about normals depending on elasticity of collision



# OPTIMIZING COLLISION DETECTION




- Often use a hierarchy of collision geometries (e.g., sphere/box then convex hull)
- Use of spatial trees, e.g., quadtree to limit which objects collision is performed against (~logarithmic time and ~constant number of triangles to collision check)

# PHYSICS-BASED MOVEMENT

# REVIEW OF LINEAR (NEWTONIAN) MECHANICS

- Newton's second law of motion – force is mass by acceleration
$$\vec{F} = m\vec{a}$$
- For position $\vec{x}$:
    - Velocity $\vec{v} = \dot{\vec{x}}$ (first derivative with respect to time)
    - Acceleration $\vec{a} = \dot{\vec{v}} = \ddot{\vec{x}}$ (second derivative with respect to time)
- In games however, we are trying to compute the next time steps position $\vec{x}'$, thus, we need the anti-derivative, i.e., integration
- Further we need to use numerical integration, after all we don't even have a symbolic representation of our motion
- Essentially cannot use variable time step
    - Accuracy is related to the magnitude of the time step
- Begin by calculating the force, e.g., gravity, wind resistance, impulses, etc
- Next compute acceleration
$$\vec{a} = \frac{\vec{F}}{m}$$

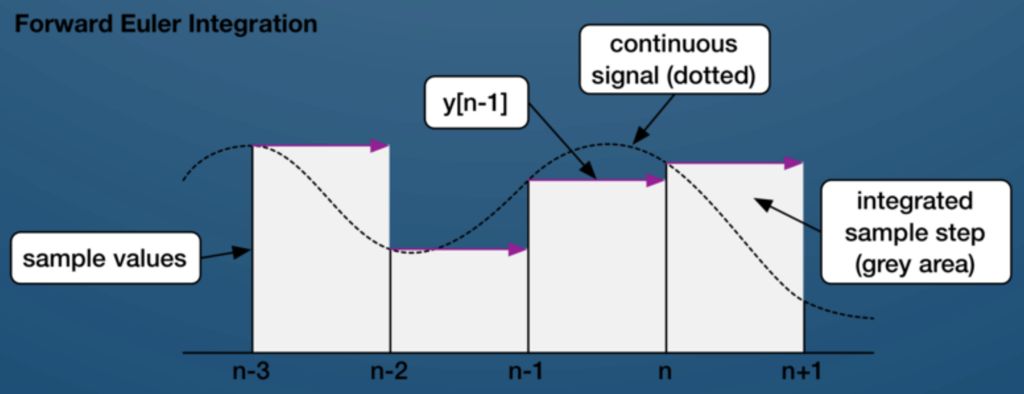
# EULER AND SEMI-IMPLICIT EULER INTEGRATION

- Euler integration – use current velocity to alter position

$$\vec{x}' = \vec{x} + \vec{v}\Delta t$$
$$\vec{v}' = \vec{v} + \vec{a}\Delta t$$

- Semi-implicit Euler integration – use next velocity to alter position

$$\vec{v}' = \vec{v} + \vec{a}\Delta t$$
$$\vec{x}' = \vec{x} + \vec{v}'\Delta t$$



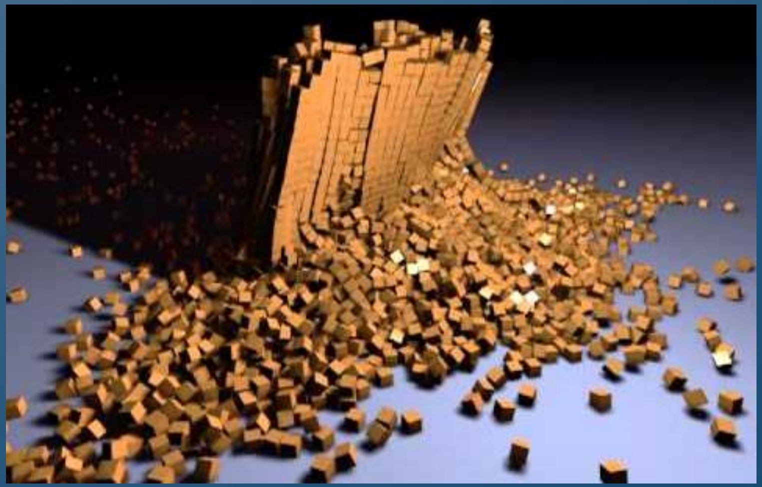
# ADVANCED CONSIDERATIONS

- Can use Velocity Verlet integration (essentially the trapezoid rule)
- Could use Taylor series expansions, e.g., Fourth Order Runge-Kutta
- All about how much error you can handle
- For angular considerations – torque is moment of inertia by angular accelaration

  $$\vec{\tau} = I\vec{\alpha}$$

  then use integration to get angular velocity $\vec{\omega}$ and angle $\vec{q}$
  - Complicated as moment of interia is a matrix

- Typically utilize existing libraries
  - PhysX
  - Bullet
  - etc



# SUMMARY

- In this chapter we delved into the basics elements of emulating physics
  - Collision detection
  - Collision response
  - Physics-based movement through numerical integration