Lecture IV: Collisions

The Story so Far

- Rigid bodies moving in space as forces are applied to them.
  - Gravity, drag, rotation, etc.

- Reaction forces occur when a rigid body comes in contact with another body.

- Handling the event correctly is then two problems:
  - Collision detection
  - Collision resolution

Collisions & Geometry

- We need the actual geometry of the object
  - A point (*e.g.* COM) is not enough anymore.
  - We must know where the objects are in contact to apply the reaction force at that position.

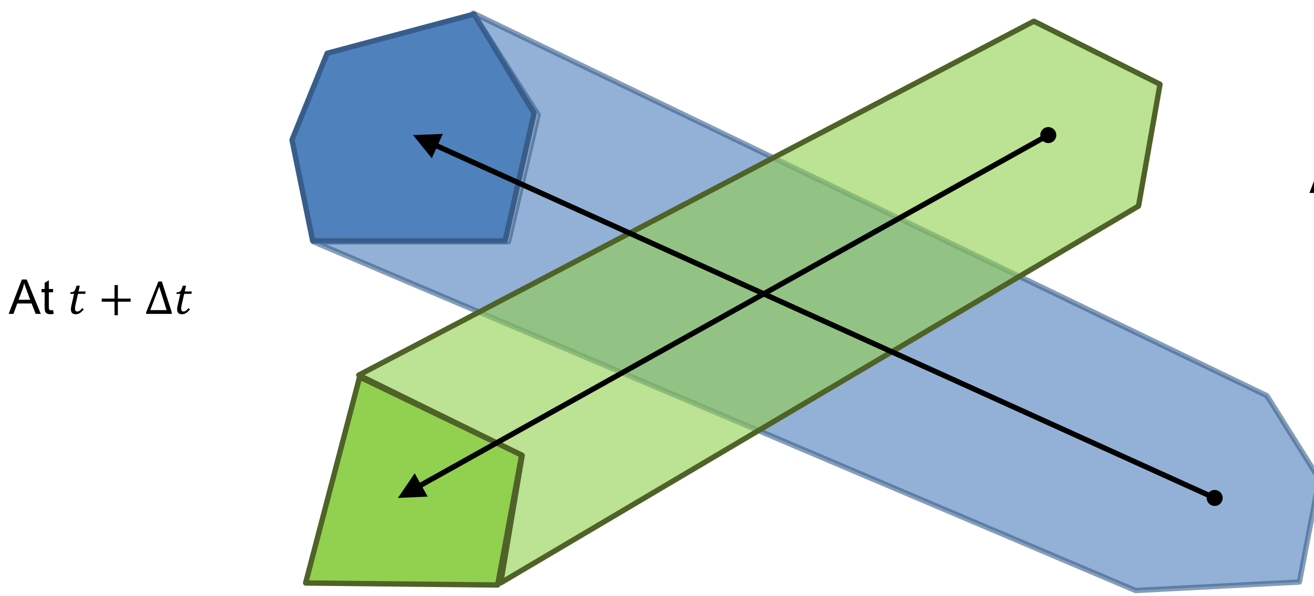
CryEngine 3 (BeamNG)

Collision Detection Algorithms

- To save time and computation, collision detection is done top-down, to rule out non-collisions fast:
  - Broad phase
    - Disregard pairs of objects that cannot collide.
    - ➢ Model and space partitioning.
  - Mid phase
    - Determine potentially-colliding primitives.
    - ➢ movement bounds.
  - Narrow phase
    - determine exact contact between two shapes.
    - Convex object intersection (GJK algorithm)
    - ➢ Triangle-triangle intersections.

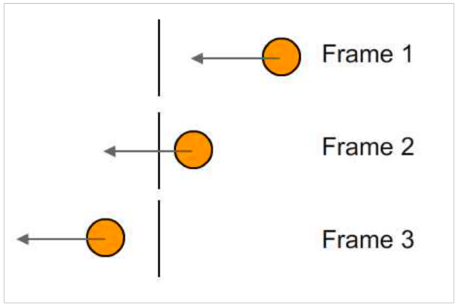
The Time Issue

- Looking at uncorrelated sequences of positions is not enough.
- Our objects are in motion and we need to know when and where they collide.



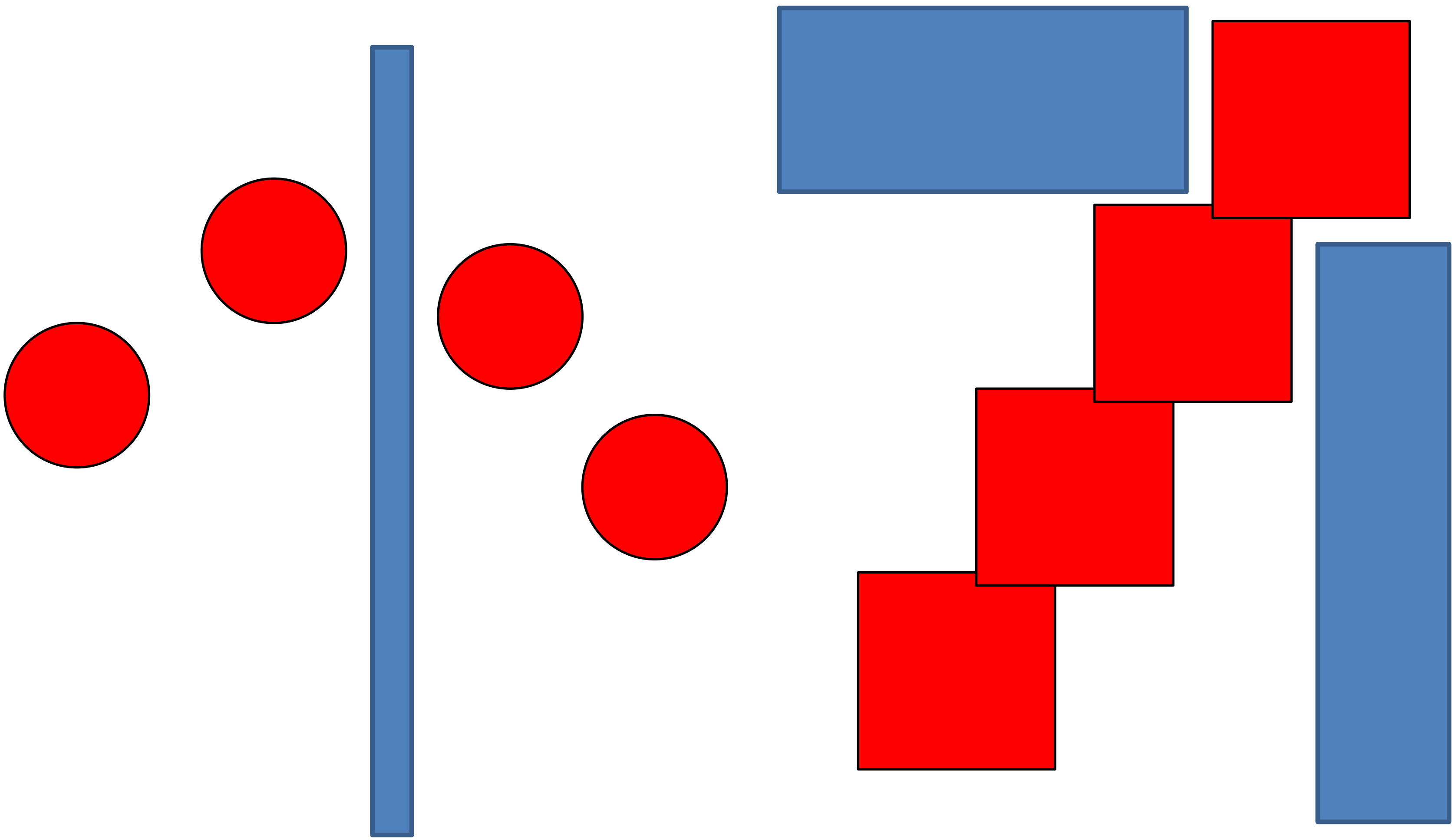
Tunneling

- Collision in-between steps can lead to tunneling.
  - Objects pass through each other
    - Colliding neither at t nor at $t + \Delta t$!
    - …but somewhere in between.
  - Leads to false negatives.



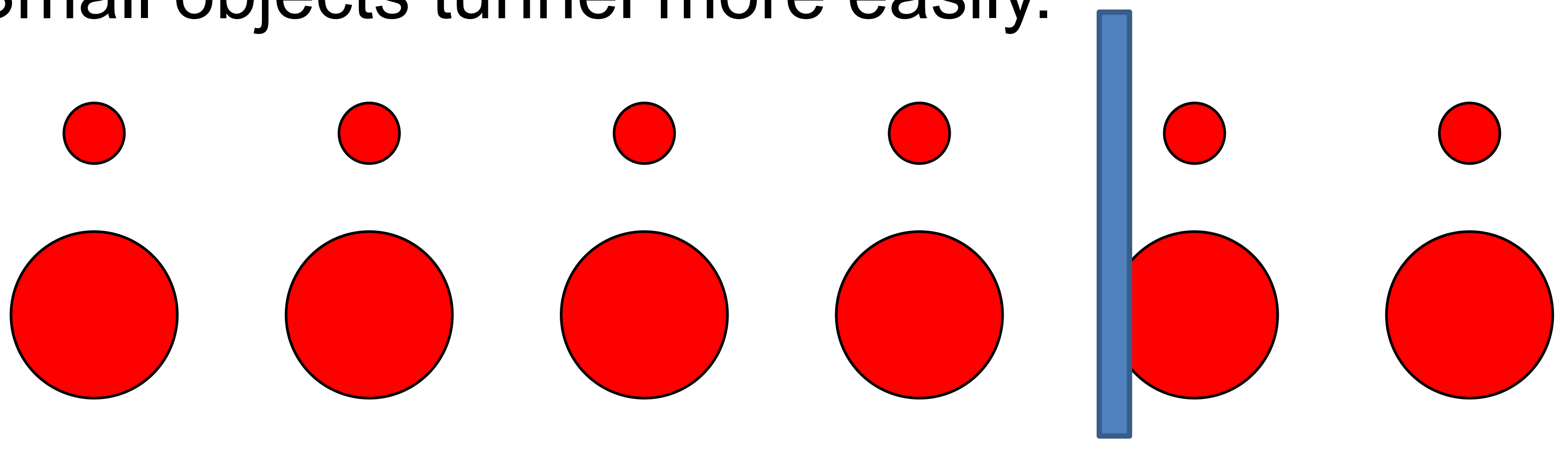
- Tunneling is a serious issue in gameplay.
  - Players getting to places they should not.
  - Projectiles passing through characters and walls.
  - Impossibility for the player to trigger actions on contact events.

Tunneling

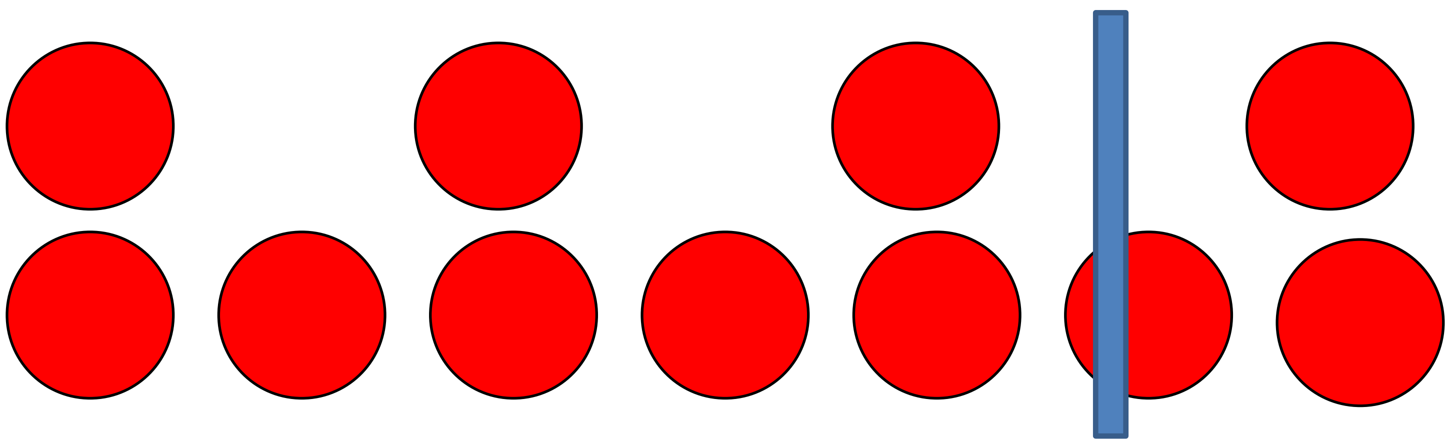


Tunneling

- Small objects tunnel more easily.



- … And fast moving objects.



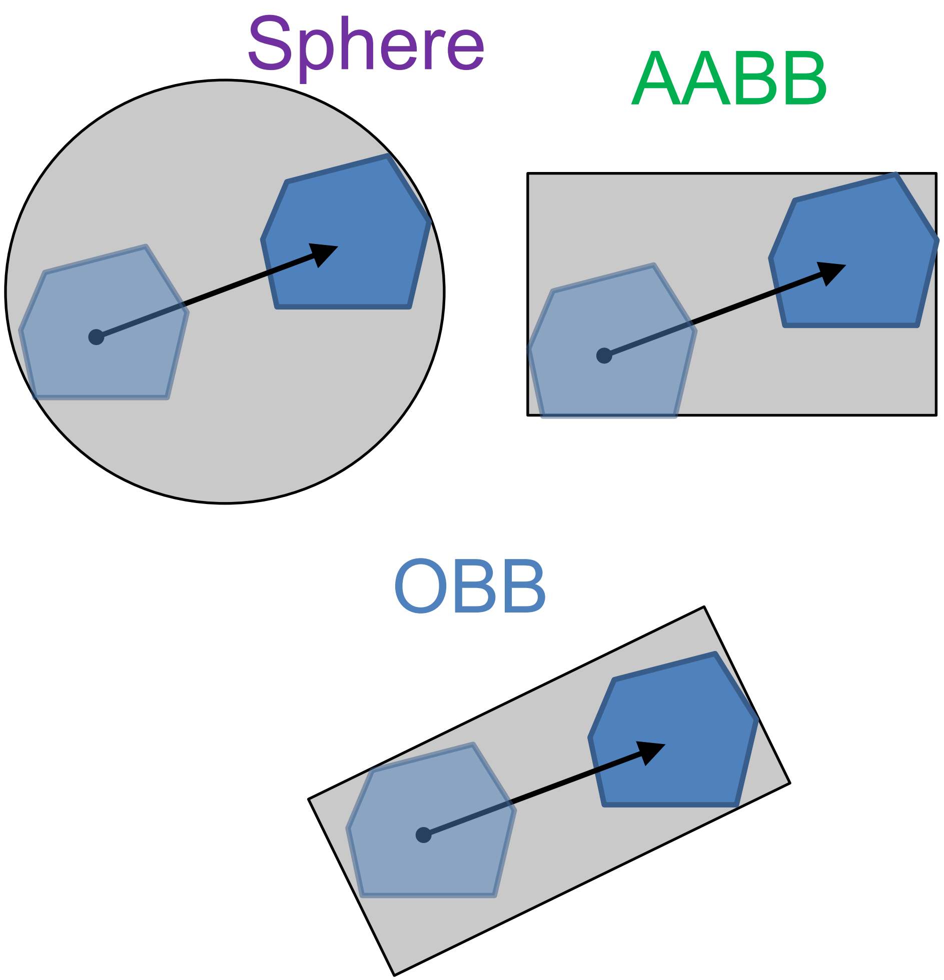
Tunneling

- Possible solutions
  - Minimum size requirement?
    - Fast object still tunnel…
  - Maximum speed limit?
    - Small and fast objects not allowed (*e.g.* bullets...)
  - Smaller time step?
    - Essentially the same as speed limit!

- Another approach is needed!

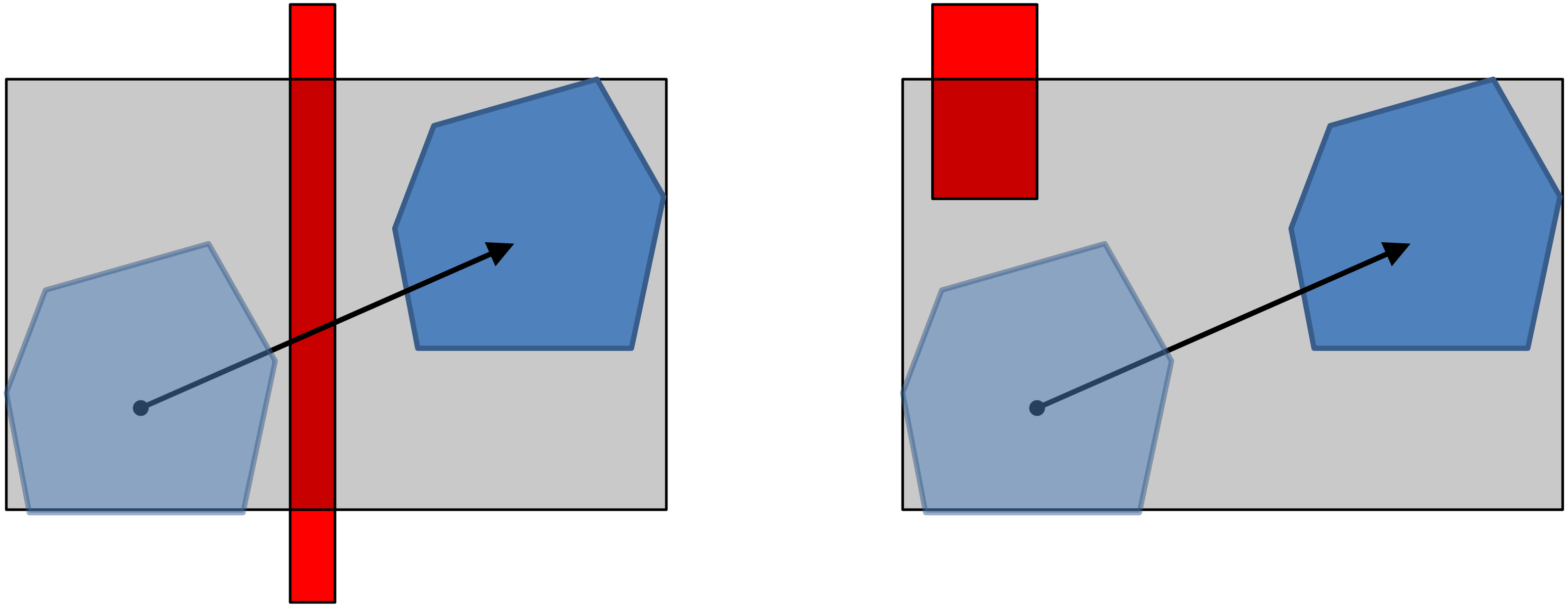
Movement Bounds

- Bounds enclosing the motion of the shape.
- In the time interval Δt, the linear motion of the shape is enclosed.
- Convex bounds are used ➔ movement bounds are also primitive shapes.



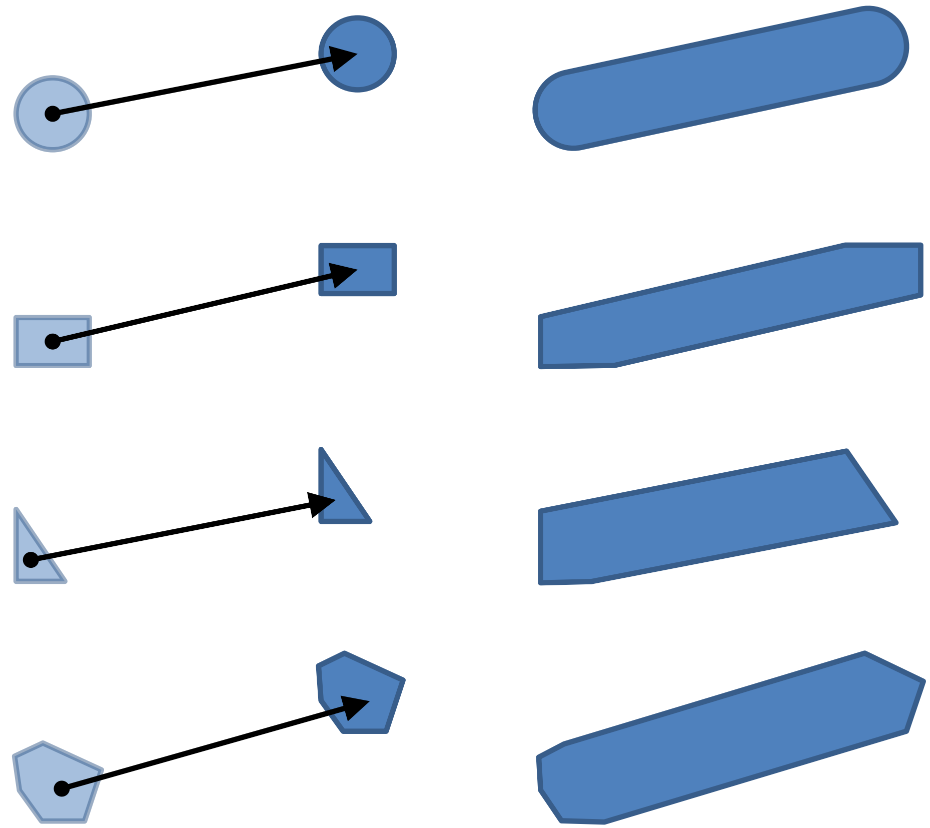
Movement Bounds

- Movement bounds do not collide ➔ there is no collision.
- Movement bounds collide ➔ possible collision.



Swept Bounds

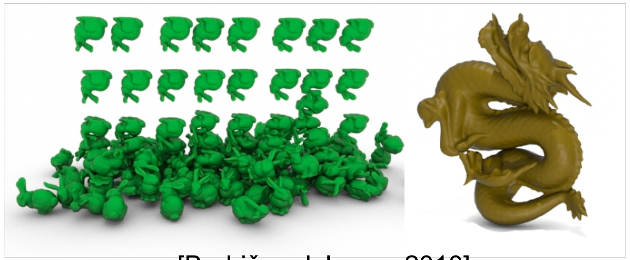
- Primitive-based movement bounds do not have a really good fit.
- We use swept bounds.
  - More accurate & more costly.
- Union of all surfaces (volumes) of a transforming shape
  - We use the affine transformation from t to t $+ \Delta t$.



What's Next?

- Collision detection (supposedly) reported a collision.

- We want to solve it
  - Bounce back the colliding objects?
  - Sticking together?
  - Breaking apart?

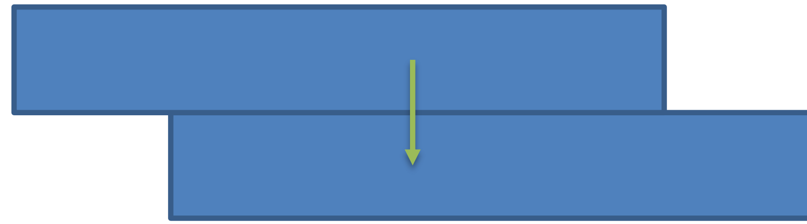


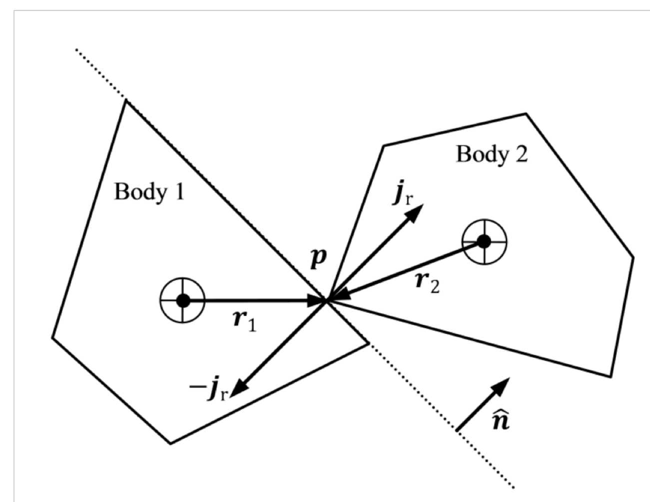
[Barbič and James 2010]

- In which direction and with what magnitude?
  - Momentum, velocity, forces…

Collision Kinematics

- Contact point.
  - point of impact.
  - Might be more than one!
- Contact normal.
  - To both surfaces.
  - Not always well defined (abstractly).
  - Normal to collision plane.
- Contact arms.
  - From COM to point.
- Line of impact: between COMs





Collision Resolution

- We estimated time of collision, contact points and contact normal.

- We still have to correct the position and orientation of the colliding objects

Types of Collisions

- Inelastic collisions
  - energy is not preserved.
    - Objects stop in place, stick together, etc.
  - are easy to implement
    - Backing out or stopping process.

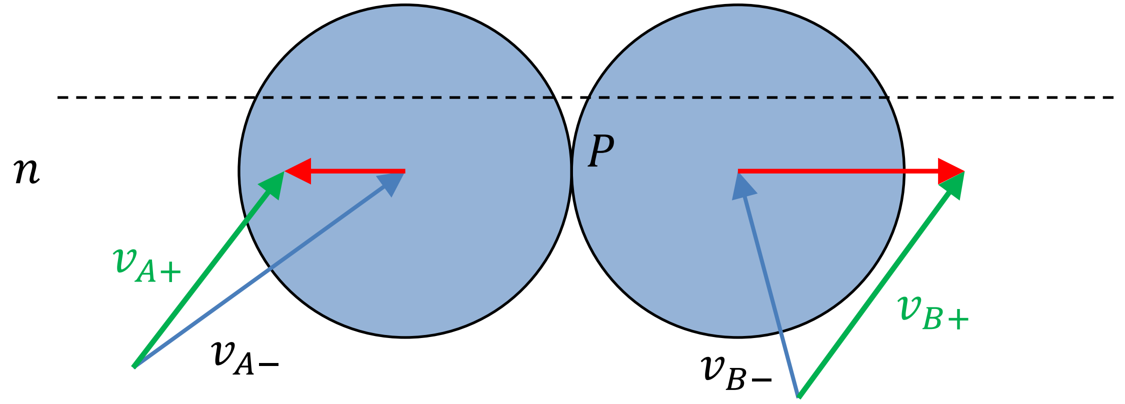
http://physics.about.com/od/energyworkpower/f/InelasticCollision.htm

- Elastic collisions
  - Energy is fully preserved.
    - *e.g.* (ideal) billiard balls.
  - More difficult to calculate.
    - Magnitude of resulting velocities

http://philschatz.com/physics-book/contents/m42183.html

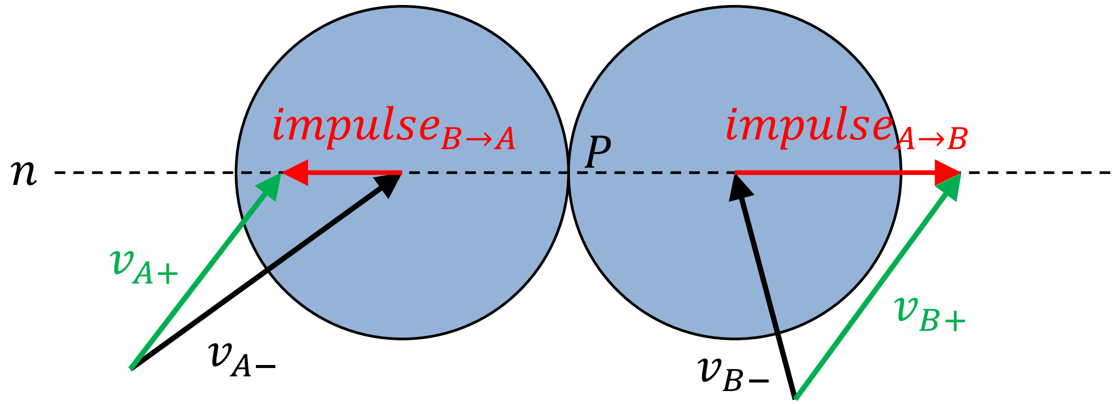
Linear velocity

- Setting:
  - Objects A & B.
  - Masses m_A & m_B.
  - Initial velocities v_{A-} & v_{B-}.
  - Unit collision normal $\hat{n}$, and contact point P.
- $\vec{v}_{A-} - \vec{v}_{B-}$: closing velocity.
- $\vec{v}_{A+} - \vec{v}_{B+}$: separating velocity.



Instant impulses

- We can solve the collision by using an impulse-based technique.
  - At collision time we apply an impulse on each object at P in the direction $\hat{n}$ ($-\hat{n}$ for the other object).
  - 'Pushing' the two objects apart.
  - The impulse magnitude: j. (impulse: $j\hat{n}$)
  - Velocity is then changed accordingly from v_- to v_+.



Reminder: Impulses

- A change in the momentum, or a force delivered in an instant:

$$\vec{F}\Delta t = \Delta\overrightarrow{p} = m(\vec{v}\,(t+\Delta t) - \vec{v}(t))$$
$$\vec{\tau}\Delta t = \Delta\overrightarrow{L} = \mathbf{I}(\overrightarrow{\omega}\,(t+\Delta t) - \overrightarrow{\omega}(t))$$

- Each type of momentum is always conserved:

$$m_A\vec{v}_A(t+\Delta t) + m_B\vec{v}_B(t+\Delta t) = m_A\vec{v}_A(t) + m_B\vec{v}_B(t)$$
$$\mathbf{I}_A\overrightarrow{\omega}_A(t+\Delta t) + \mathbf{I}_B\overrightarrow{\omega}_B(t+\Delta t) = \mathbf{I}_A\overrightarrow{\omega}_A(t) + \mathbf{I}_B\overrightarrow{\omega}_B(t)$$

  - In the same coordinate system to the same fixed point!

Linear velocity

- By the impulse we get:

$$m_A\vec{v}_{A-} + j\hat{n} = m_A\vec{v}_{A+}$$
$$m_B\vec{v}_{B-} - j\hat{n} = m_B\vec{v}_{B+}$$

- And explicitly for the velocities:

$$\vec{v}_{A+} = \vec{v}_{A-} + \frac{j}{m_A}\hat{n}$$
$$\vec{v}_{B+} = \vec{v}_{B-} - \frac{j}{m_B}\hat{n}$$

- 2 equations in 3 variables ➔ missing 1 degree of of freedom!

Coefficient of Restitution

- The coefficient of restitution C_R models elasticity.
- The ratio of speeds after and before collision along the collision normal

$$C_R = -\frac{(\vec{v}_{A+} - \vec{v}_{B+}) \cdot \hat{n}}{(\vec{v}_{A-} - \vec{v}_{B-}) \cdot \hat{n}}$$

  - $C_R = 1$: ideal elastic collision (E_k is conserved)
  - $C_R < 1$: inelastic collision (loss of velocity).
  - $C_R = 0$: the objects stick together.

Velocity Correction

$$\vec{v}_{A+} = \vec{v}_{A-} + \frac{j}{m_A}\hat{n}$$
$$\vec{v}_{B+} = \vec{v}_{B-} - \frac{j}{m_B}\hat{n}$$

- As the velocities before and after collision relate by the coefficient of restitution:

$$C_R = -\frac{(\vec{v}_{A+} - \vec{v}_{B+}) \cdot \hat{n}}{(\vec{v}_{A-} - \vec{v}_{B-}) \cdot \hat{n}}$$

- …we calculate:

$$j = \frac{-(1 + C_R)[(\vec{v}_{A-} - \vec{v}_{B-}) \cdot \hat{n}]}{\left(\frac{1}{m_A} + \frac{1}{m_B}\right)}$$

Joint masses

Velocity Correction

- We can finally calculate the outgoing velocities:

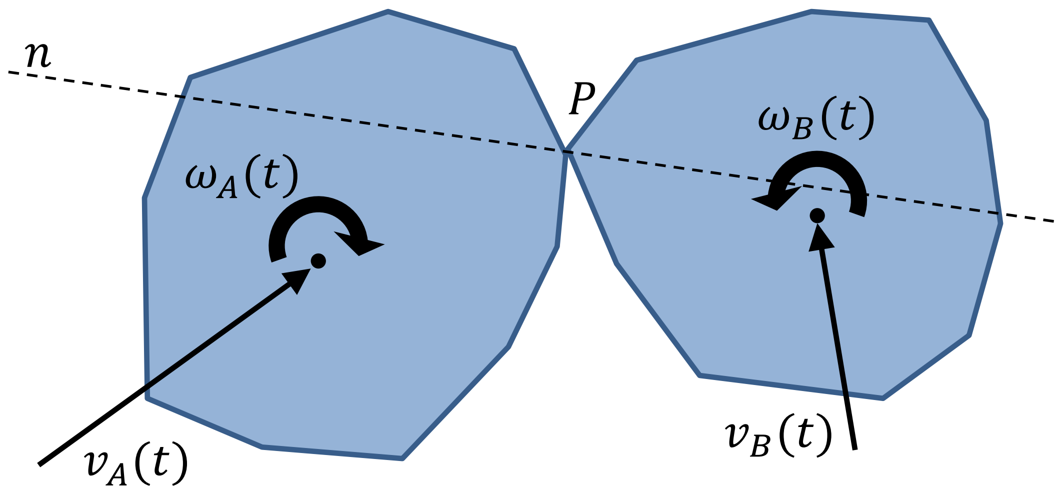
$$\vec{v}_{A+} = \vec{v}_{A-} + \frac{j}{m_A}\hat{n}$$

$$\vec{v}_{B+} = \vec{v}_{B-} - \frac{j}{m_B}\hat{n}$$

- Larger mass difference ⇔ less velocity change.

Angular Velocity

- Point of contact not on line of impact ➔ normal off the center of rotation ➔ the collision also produces a rotation of the two objects.



Angular velocity

- Handling rotational collision similarly to linear collision.
- Impulse factor j is adapted accordingly.
- Rotational velocity contributes to the total closing velocity:

$$\bar{v}_{A-} = \vec{v}_{A-} + \vec{\omega}_{A-} \times \vec{r}_A$$
$$\bar{v}_{B-} = v_{B-} + \vec{\omega}_{B-} \times \vec{r}_B$$

  - $\vec{\omega}$: angular velocities
  - $\vec{r}$: collision arm = (point of contact) – (center of rotation).

Angular velocity

- The coefficient of restitution equation works with the total closing velocity:

$$C_R = -\frac{(\bar{v}_{A+} - \bar{v}_{B+}) \cdot \hat{n}}{(\bar{v}_{A-} - \bar{v}_{B-}) \cdot \hat{n}}$$

- The resulting impulse j will create both angular and linear velocities.

Angular velocity

- By the impulse we get:

$$\mathbf{I}_A \vec{\omega}_{A-} + \vec{r}_A \times (j\hat{n}) = \mathbf{I}_A \vec{\omega}_{A+}$$
$$\mathbf{I}_B \vec{\omega}_{B-} - \vec{r}_B \times (j\hat{n}) = \mathbf{I}_B \vec{\omega}_{B+}$$

- 2 more equations and 2 more variables ($\vec{\omega}_{A+}$ and $\vec{\omega}_{B+}$).
- Inertia tensors: in world coordinates, around each center of rotation.
- And we get:

$$\vec{\omega}_{A+} = \vec{\omega}_{A-} + I_A^{-1}(\vec{r}_A \times (j\hat{n}))$$
$$\vec{\omega}_{B+} = \vec{\omega}_{B-} - I_B^{-1}(\vec{r}_B \times (j\hat{n}))$$

Angular velocity

- The updated factor j:

$$j = \frac{-(1 + C_R)[(\bar{v}_{A-} - \bar{v}_{B-}) \cdot \hat{n}]}{\left(\frac{1}{m_A} + \frac{1}{m_B}\right) + [(\vec{r}_A \times \hat{n})^T I_A^{-1} (\vec{r}_A \times \hat{n}) + (\vec{r}_B \times \hat{n})^T I_B^{-1} (\vec{r}_B \times \hat{n})]}$$

Augmented mass and inertia

Angular velocity

- With this updated factor j, we calculate the separating angular velocities

$$\vec{\omega}_{A+} = \vec{\omega}_{A-} + I_A^{-1}(\vec{r}_A \times (j\hat{n}))$$
$$\vec{\omega}_{B+} = \vec{\omega}_{B-} - I_B^{-1}(\vec{r}_B \times (j\hat{n}))$$

- This factor is also used to calculate the separating linear velocities (same as linear resolution):

$$\vec{v}_{A+} = \vec{v}_{A-} + \frac{j}{m_A}\hat{n}$$

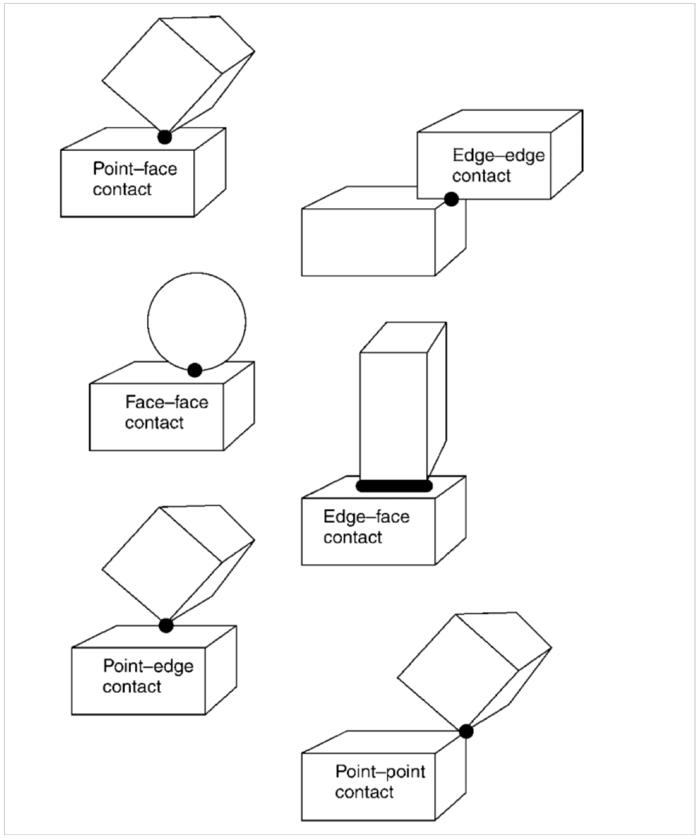
$$\vec{v}_{B+} = \vec{v}_{B-} - \frac{j}{m_B}\hat{n}$$

Practical Considerations

- You need $\mathbf{I}_A^{-1}$, $\mathbf{I}_B^{-1}$ in the world coordinate system, and around each individual COM.
  - Changes with rotation!
- You usually have: $\mathbf{I}'_A$ in the object coordinate system around each individual COM.
  - Preprocess computation.
- Problem: Inverse is expensive.
- Solution:
  - Invert once for object coordinate system $\mathbf{I}_A'^{-1}$.
  - Apply orientation change R: $\mathbf{I}_A^{-1}=R^T\mathbf{I}_A'^{-1}R$.
  - Mind if to use R or R^T according to context!

Types of contact

- Most common (general position):
  - Point-face (PF).
  - Edge-edge (EE).
- Normals:
  - The face in PF.
  - Normal to both edges in EE.
- Note: other cases more difficult.

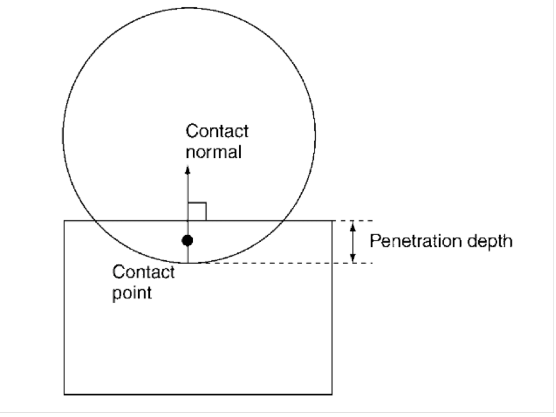


Time of Collision

- Computing the exact time (somewhere between t and $t + \Delta t$) of collision is not always feasible
- We can approximate it by bisection.
- Repeatedly bisecting the time interval and testing, finding a arbitrary short interval $[t_0, t_1]$ for which:
  - The objects do not collide at t_0.
  - The objects collide at t_1.
- Computationally expensive!
  - Usually in games, the frame rate Δt is small enough to not bother.
  - Correction method: interpenetration resolution.

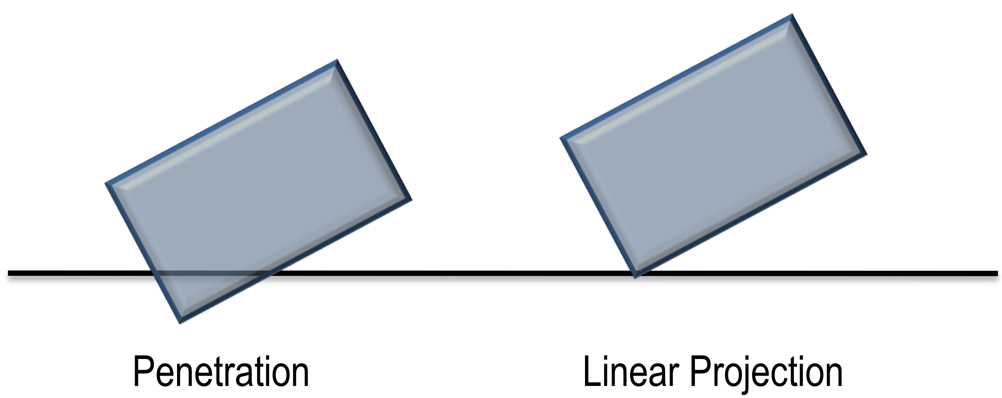
Interpenetration

- Collision happens between t and $t + \Delta t$.
- We run a position update on $t + \Delta t$.
- Objects are now interpenetrating!
- Collision detection algorithms usually provide:
  - Closest point on one of the objects.
  - Contact normal (vector to point).
  - Interpenetration depth.



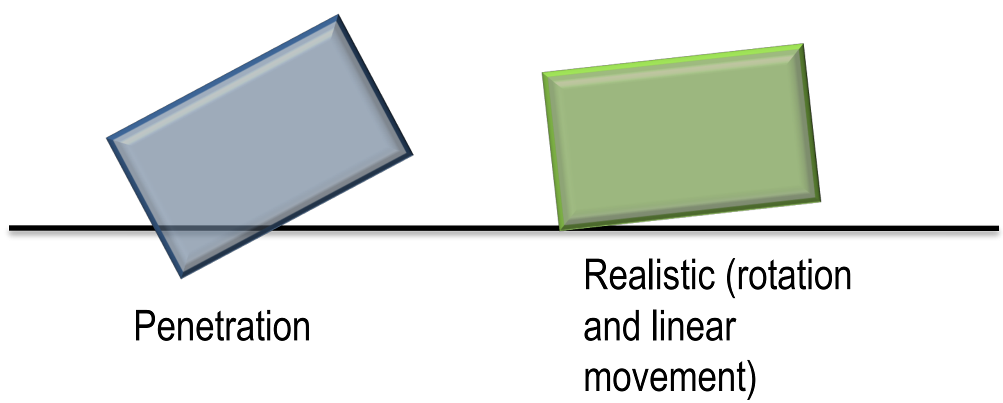
Resolving interpenetration

- Linear Projection
  - Simply “move back”
- Disadvantage: not realistic for rotations.
  - Also “twitched” movement
  - Adding non-existing friction.
- If one object is fixed, move only the other.
- If both mobile: by inverse mass weighting.



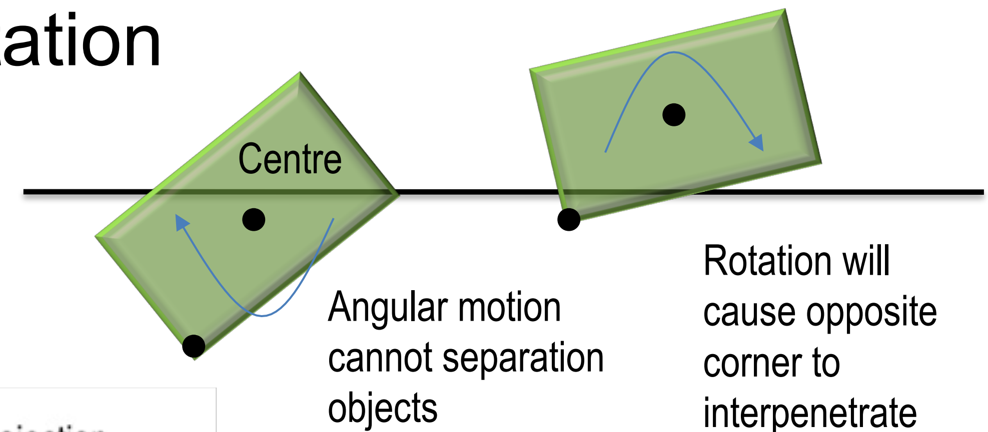
Resolving Interpenetration

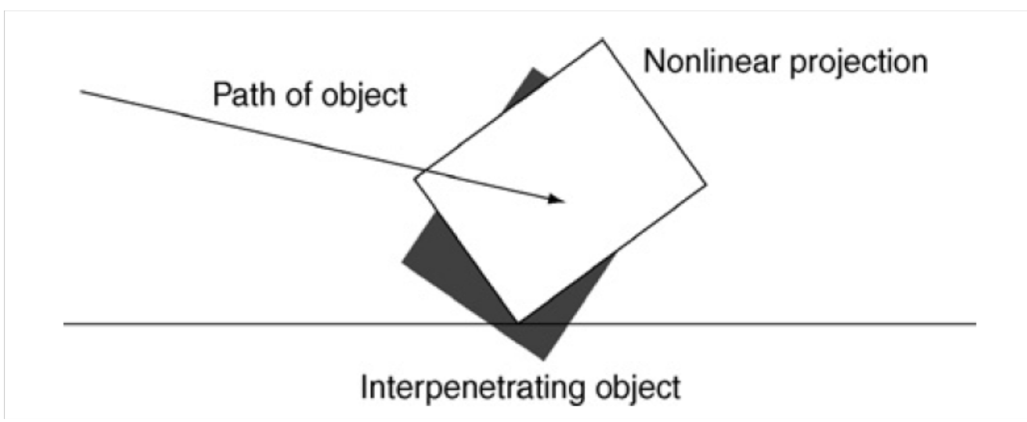
- Non-linear Projection
  - Creating both linear and angular movement until penetration is resolved in the normal direction.
  - But how much of both?
  - Why no just “rollback time”?



Nonlinear Projection

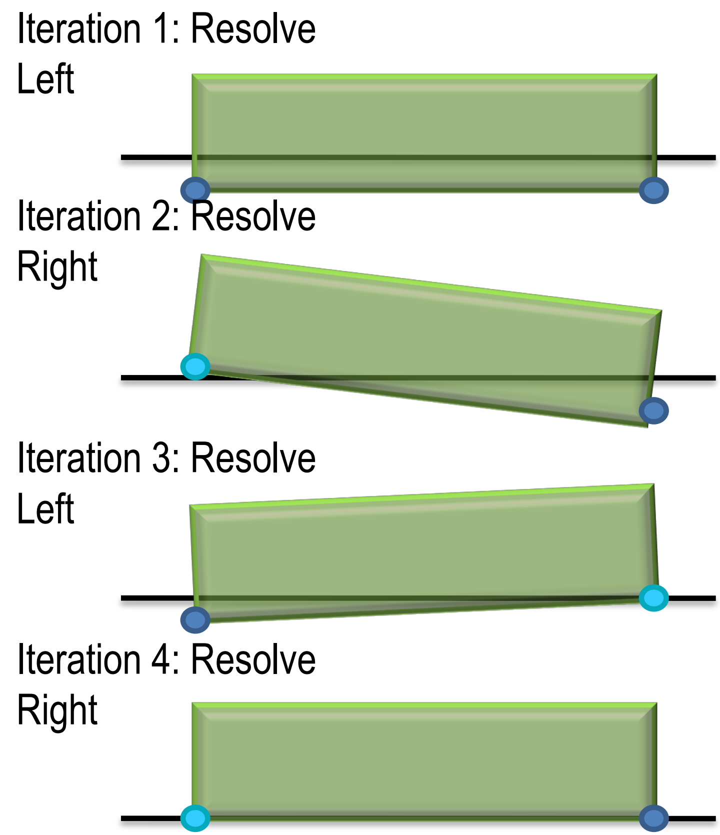
- Compromise: move back on a linear path, and rotate in the process.
  - Until penetration is resolved.
- Problem: excessive rotation





Problem: multiple contacts

- Order is important!
- Approximation:
- Iterate until resolved.
- Always resolve worst.
- Problem: depths keep changing!
  - Update who's worst by applying the velocities from the previous iteration.



Multiple Collisions

- Similar process:
  - Resolve the worst collision
    - Fastest closing velocity.
  - Use resulting separating velocities as closing velocities for the next worst collision.

Collision resolution

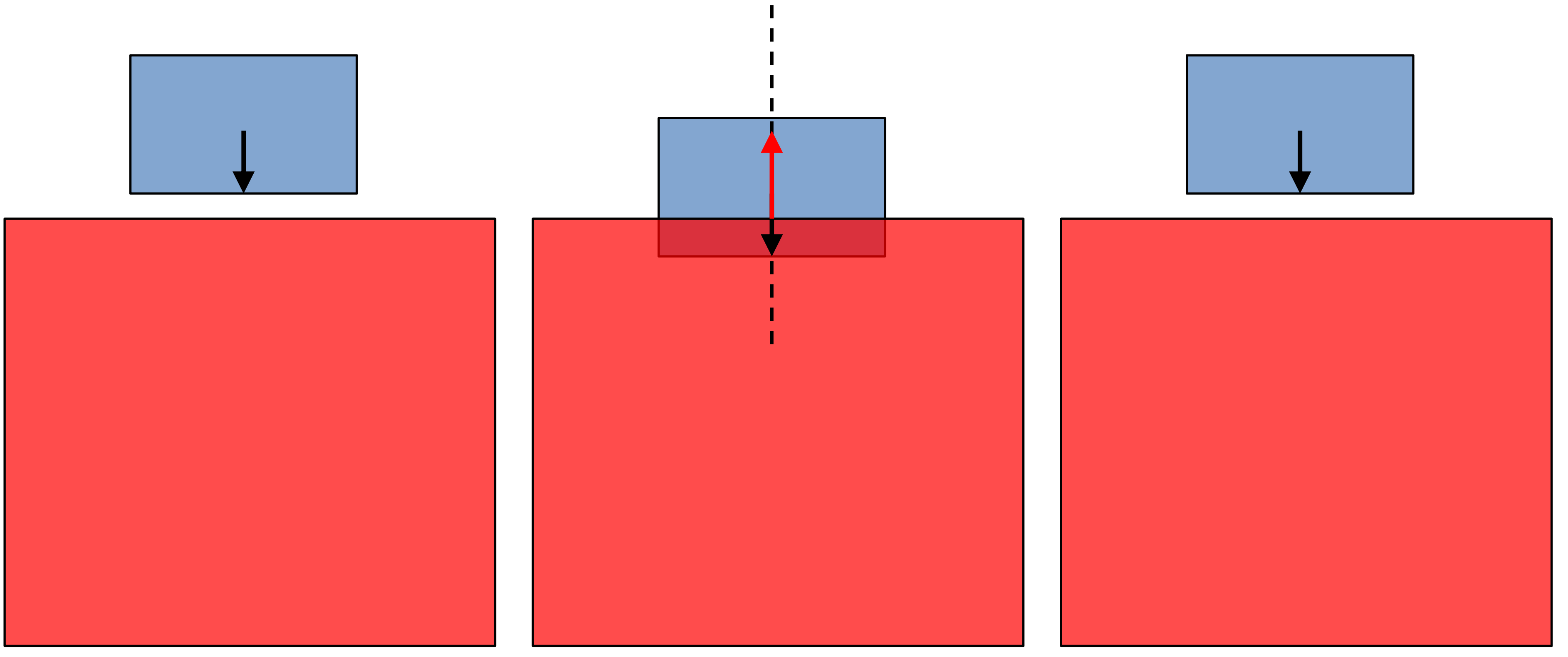
- The full algorithm:
  - Run collision detection to find contact point(s) and contact normal.
  - Resolve interpenetration.
  - Use coefficients of restitution and conservation of momentum to determine the impulses to apply.
  - Calculate linear and angular velocities at these contact points.
  - Solve for velocities using the impulses.
- Part of the greater rigid-body engine loop.

Resting contact

- Our resolution system is theoretically complete.
- Some special cases can be handled more efficiently.
- We can have resting contacts between objects
    - For example, a box colliding with on the floor.
    - the floor theoretically moves down, but is assumed stationary, because of theoretically very large mass.

Resting contact

- In a typical framework, a box sitting on the floor may ‘jitter’ around the surface.



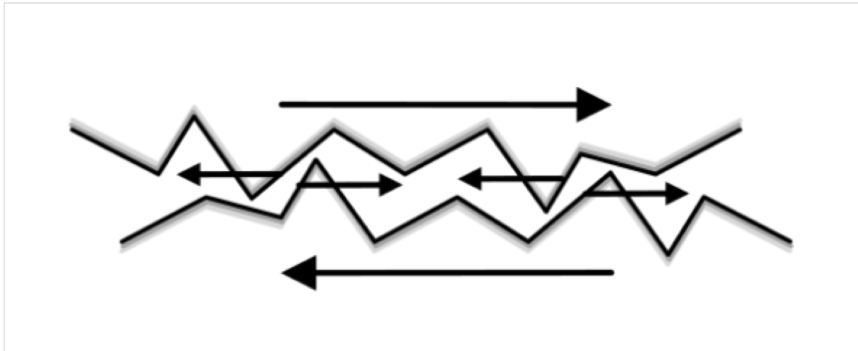
Resting contact

- A resting contact ⇔ relative velocity of the two objects along the normal is 0 (or <tolerance).

- A solution: to 'artificially' reduce C_R when we are in that case.
  - Either: Linearly dependent on the relative velocity,
  - Or: directly set to $C_R = 0$.
  - after resolution the two objects have 0 relative velocity ⇔ the box sticks to the unmoving floor.

Friction

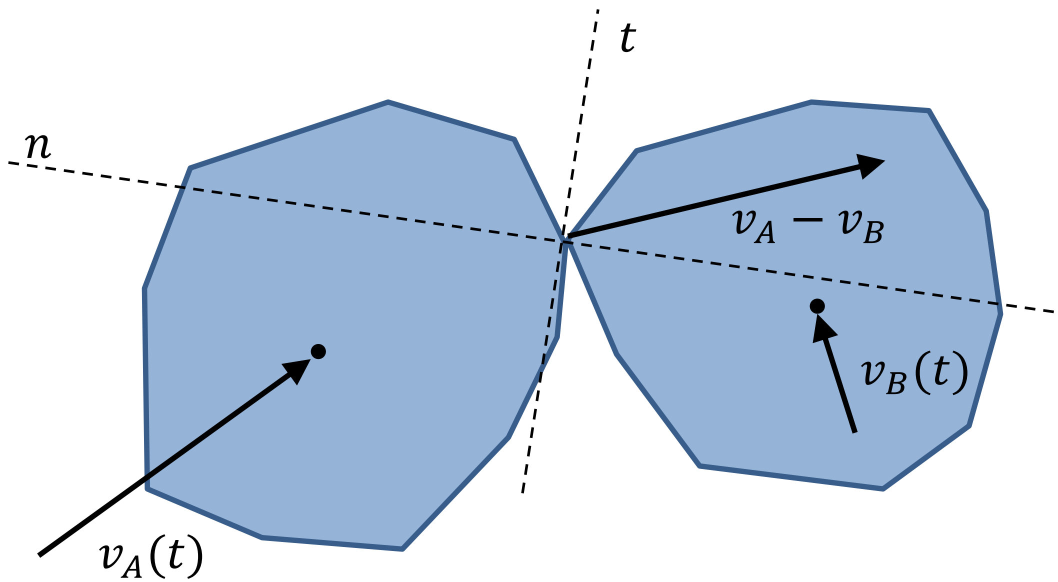
- In practice, there is friction between two objects when in contact.
  - Static friction: relatively stationary.
  - Kinetic friction: moving relatively to each other.
  - Rolling friction: is usually ignored in game physics.
- We can add the friction force in our previous equations using impulses.



Friction

- The friction acts in the tangential plane of the collision normal and resists the movement

$$\vec{t} = \left(\hat{n}\times(\vec{v}_A - \vec{v}_B)\right)\times\hat{n}$$



Kinetic Friction

- The velocity equations become:

$$\vec{v}_{A+} = \vec{v}_{A-} + \frac{j_A(\hat{n} + \mu_k \hat{t})}{m_A}$$

$$\vec{v}_{B+} = \vec{v}_{B-} - \frac{j_B(\hat{n} + \mu_k \hat{t})}{m_B}$$

Note normalization of $\hat{t}$!

$$\vec{\omega}_{A+} = \vec{\omega}_{A-} + I_A^{-1}(\vec{r}_A \times (j(\hat{n} + \mu_k \hat{t})))$$

$$\vec{\omega}_{B+} = \vec{\omega}_{B-} - I_B^{-1}(\vec{r} \times (j(\hat{n} + \mu_k \hat{t})))$$

Static Friction

- For small relative velocity, static friction is used.
- The friction impulses need to be adjusted.
  - When will objects break off?