2D Rigid Body Physics and Collision Detection using Sequential Impulses

TIM STRAUBINGER – CPSC 427 – SPRING 2021

Additional Resources

https://box2d.org/publications/

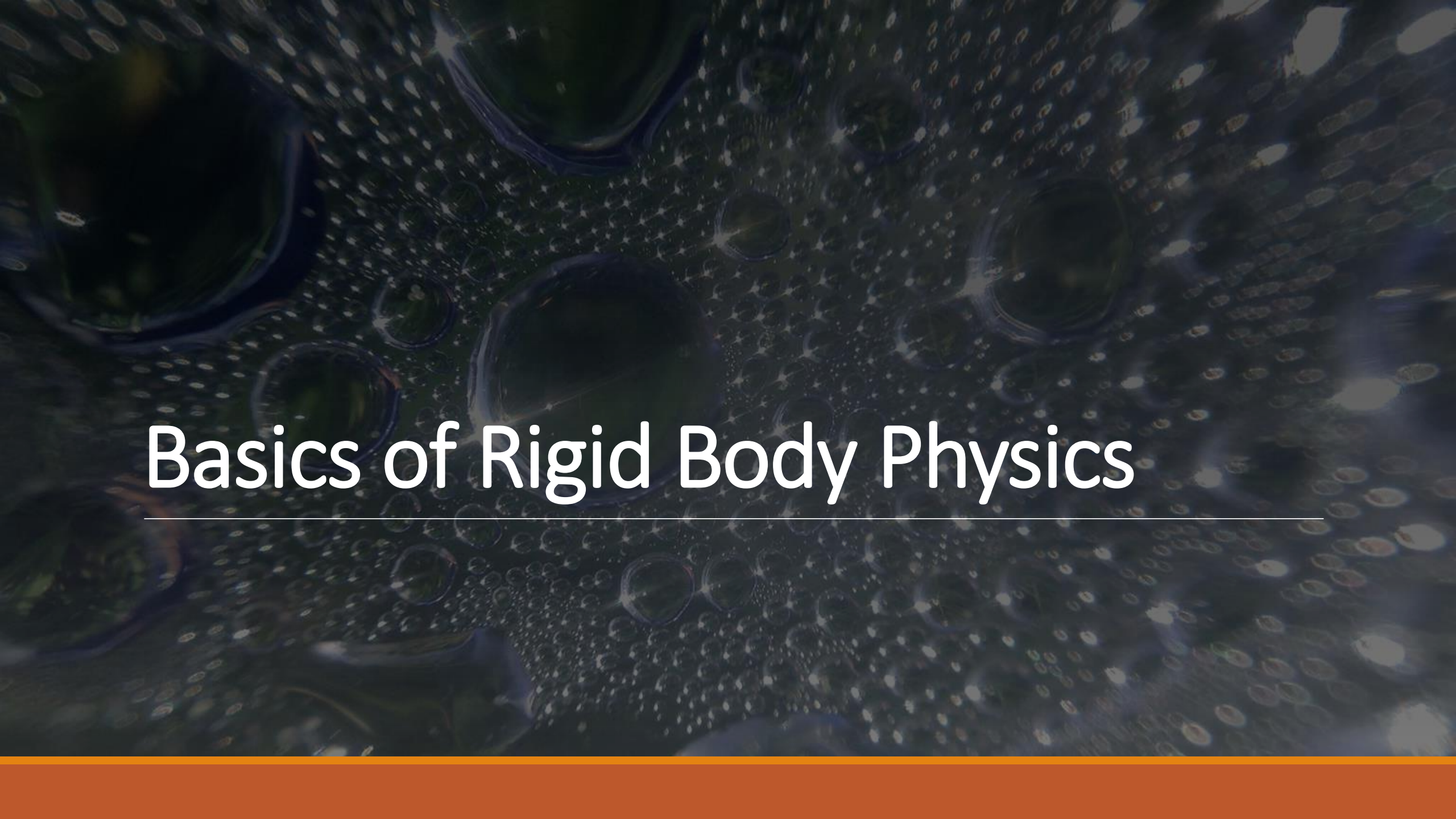
- Materials by Erin Catto, author of Box2D physics engine

https://www.toptal.com/game/video-game-physics-part-i-an-introduction-to-rigid-body-dynamics

- 3-part rigid body dynamics tutorial by Nilson Souto

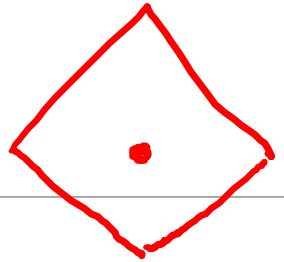
https://en.wikipedia.org/wiki/List_of_moments_of_inertia

- lists moments of inertia for many basic shapes



Basics of Rigid Body Physics

What is a Rigid Body?



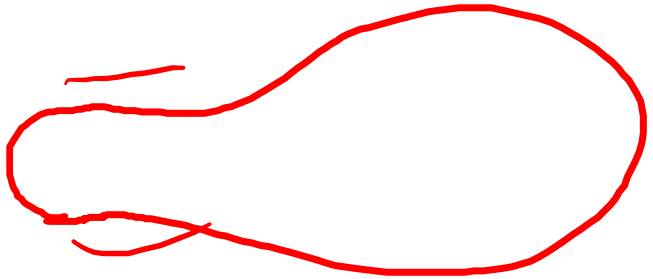
A physical body (with mass, orientation, and shape) that cannot bend

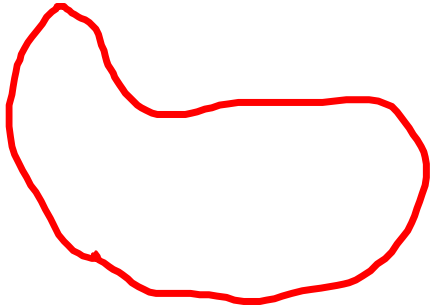
The orientation is easy to describe globally (e.g. position and velocity about center of mass) and it does not depend on the shape

The effects of forces/impulses also do not depend on the shape!

- These depend on the body's **inertia** and **moment of inertia** (if using rotation), which are influenced by the body's mass and distribution of mass

The rigid body's shape can be ignored in all parts of the physics engine **except during collision detection**





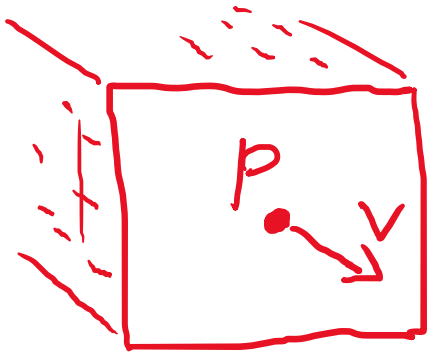
Describing a Rigid Body in 2D

Linear Properties

position (2D vector, e.g. meters)

velocity (2D vector, e.g. meters/second)

mass (scalar, e.g. kg)



Angular Properties

angle (scalar, e.g. radians)

angular velocity (scalar, e.g. radians/second)

moment of inertia (scalar, kg*meters^2)

Moving a Rigid Body

Integrating the equations of motion

$$p_t = p_{t-1} + \Delta t \cdot v_{t-1}$$

$$\theta_t = \theta_{t-1} + \Delta t \cdot \omega_{t-1}$$

What is an Impulse?

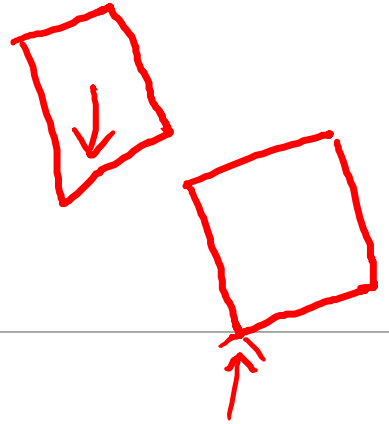
Instantaneous change in momentum (where momentum is mass times velocity)

Usually represented as *J*

Applying an impulse *J* to a light object causes a large change in velocity

Applying the same impulse *J* to a heavy object causes a small change in velocity

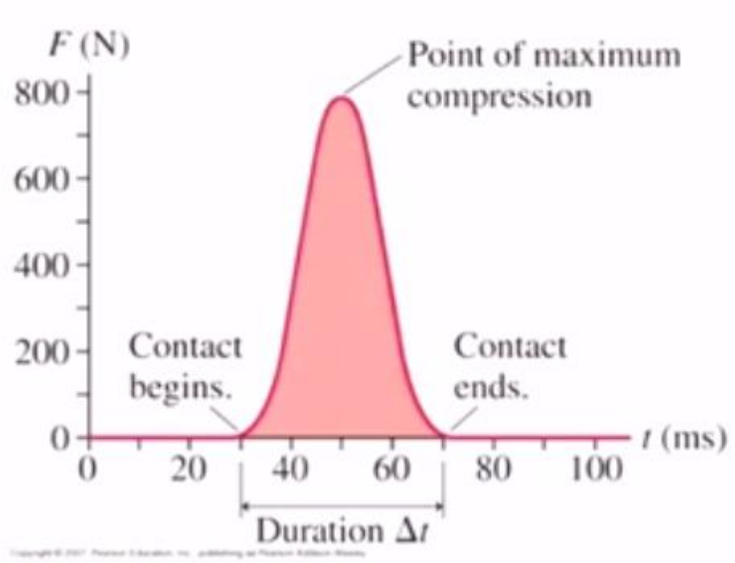
Why use Impulses?



When rigid bodies collide, potentially infinite forces are generated!

- Suppose that two objects are colliding, and 1 Newton of force is needed for a time step of 1 second to keep them separated
- For a time step of 0.1 seconds, we would need 10 Newtons to keep them separated
- For a time step of 0.01 seconds, we would need 100 Newtons
- Etc…

Math for solving constraints is easier compared to forces



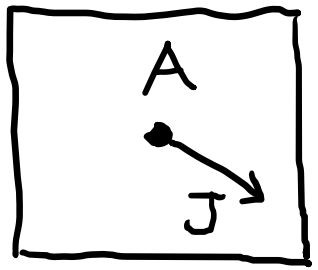
Applying Impulses to a Rigid Body
(at the Center of Mass)

Linear Impulses

instantaneous change in linear momentum

Equivalent to force * time

$$V_{new} = V_{old} + \frac{J_{linear}}{M}$$

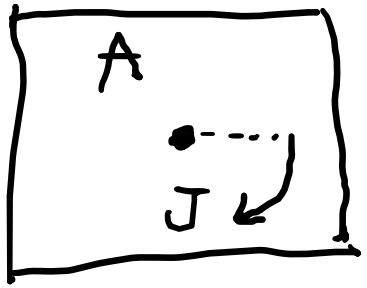


Angular Impulses

Instantaneous change in angular momentum

Equivalent to torque * time

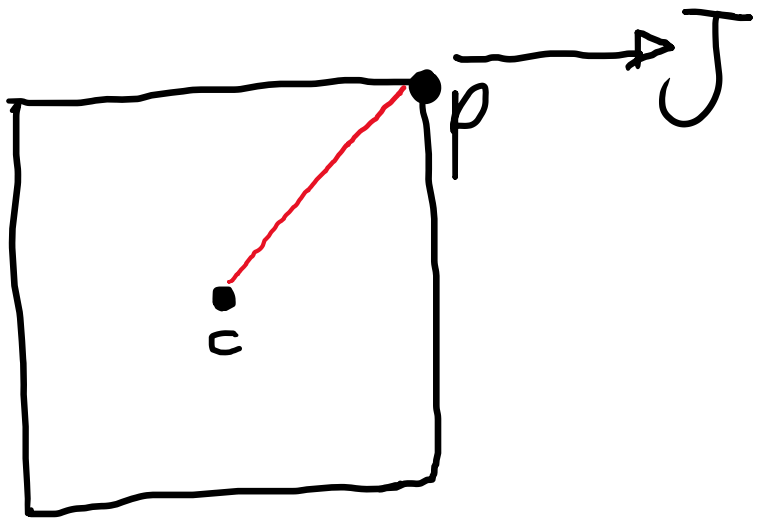
$$\omega_{new} = \omega_{old} + \frac{J_{angular}}{I}$$



Applying Impulses to a Rigid Body
(at any point)

Achieved by splitting impulse into linear and angular terms relative to center of mass

This step can be ignored if you are not using rotation



$$J_{linear} = J$$

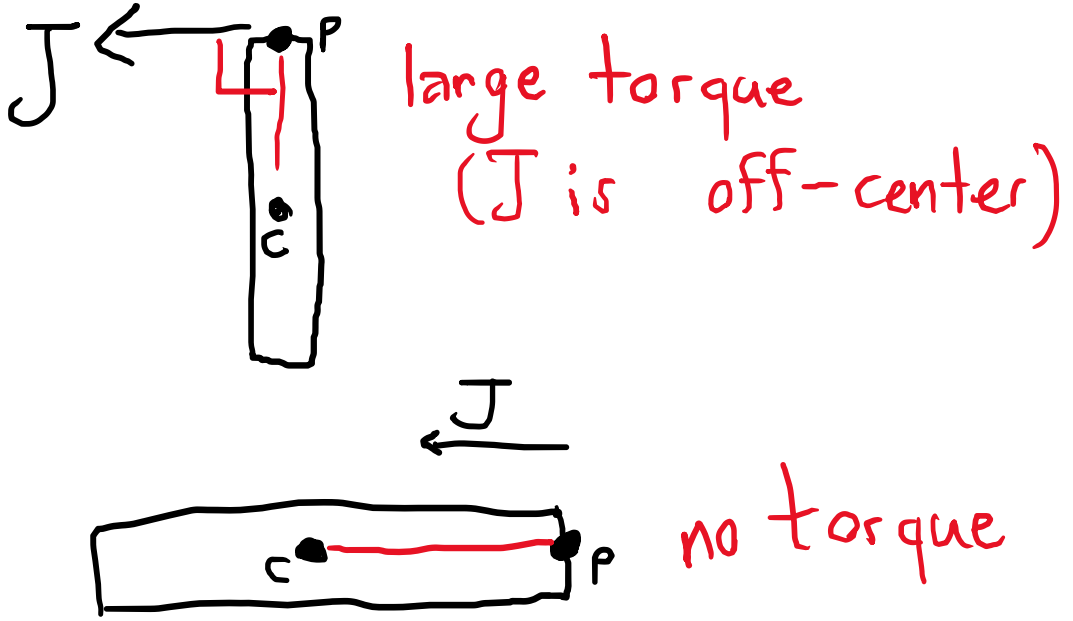
$$J_{angular} = J \times (p - c)$$

vector cross product

Applying Impulses to a Rigid Body
(at any point)

Achieved by splitting impulse into linear and angular terms relative to center of mass

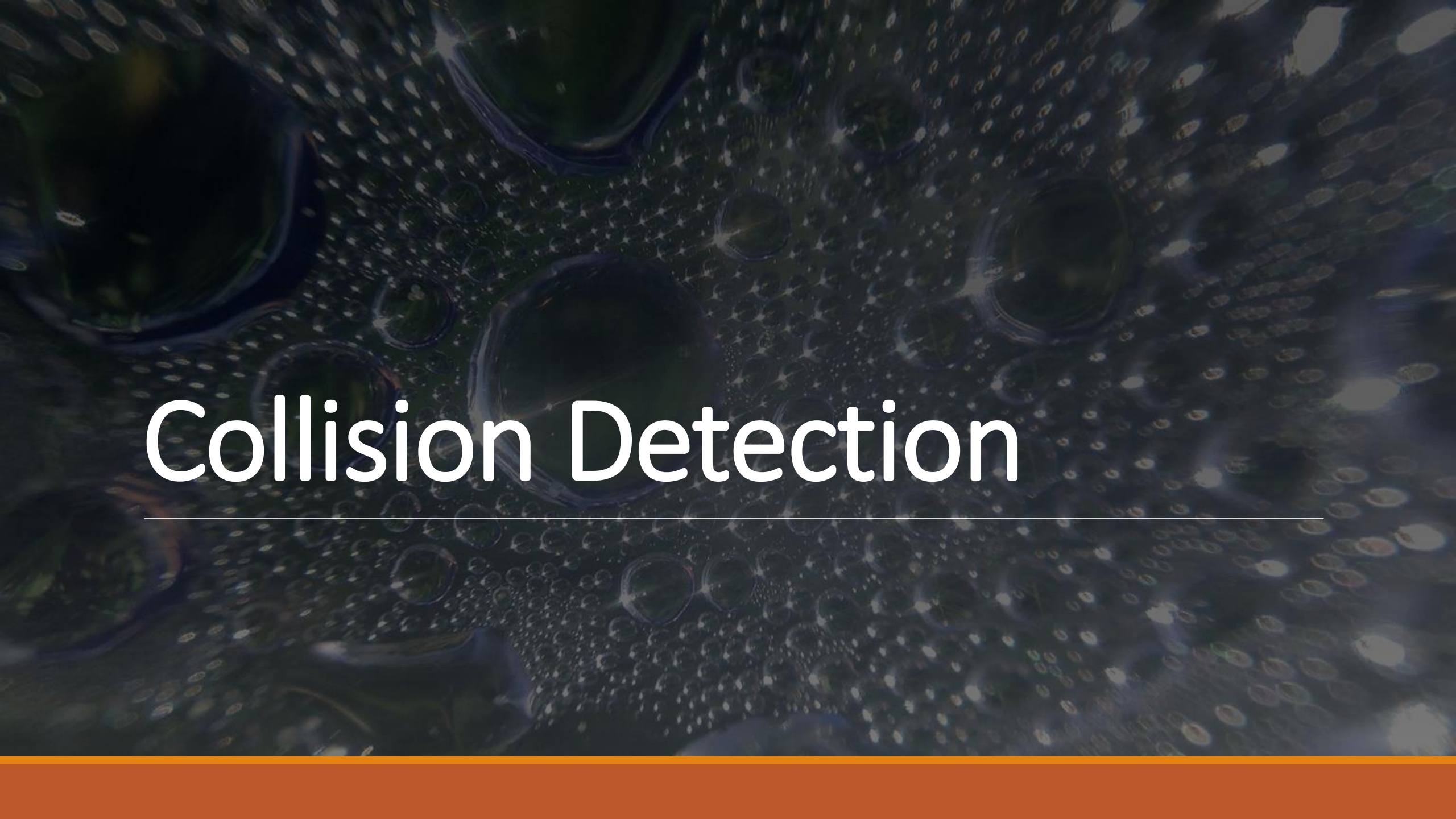
This step can be ignored if you are not using rotation



$$J_{linear} = J$$

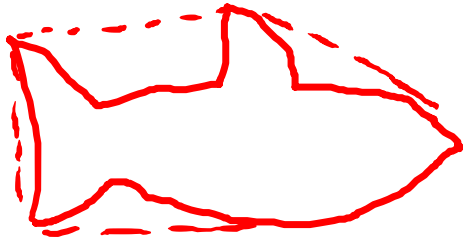
$$J_{angular} = J \times (p - c)$$

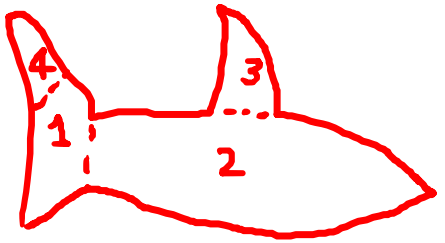
vector cross product



Collision Detection

Convex Shapes



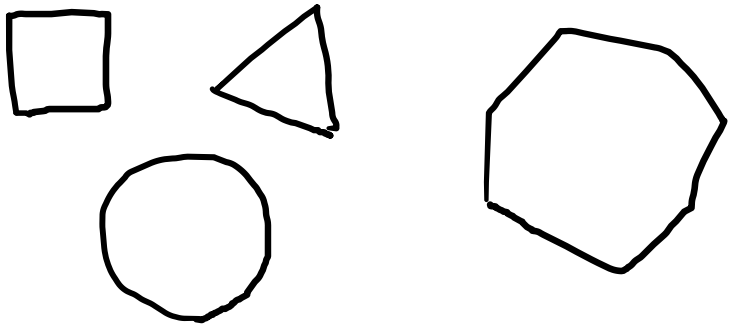


Slightly-formal definition: a shape is convex *if and only if* every straight line segment drawn between any two points inside the shape is also entirely inside the shape.

Less formal definition: a shape is convex *if and only if* you can stretch a rubber band around its perimeter without leaving gaps.

Even less formal definition: a shape is convex *if and only if* you can't drink tea out of it.

Convex



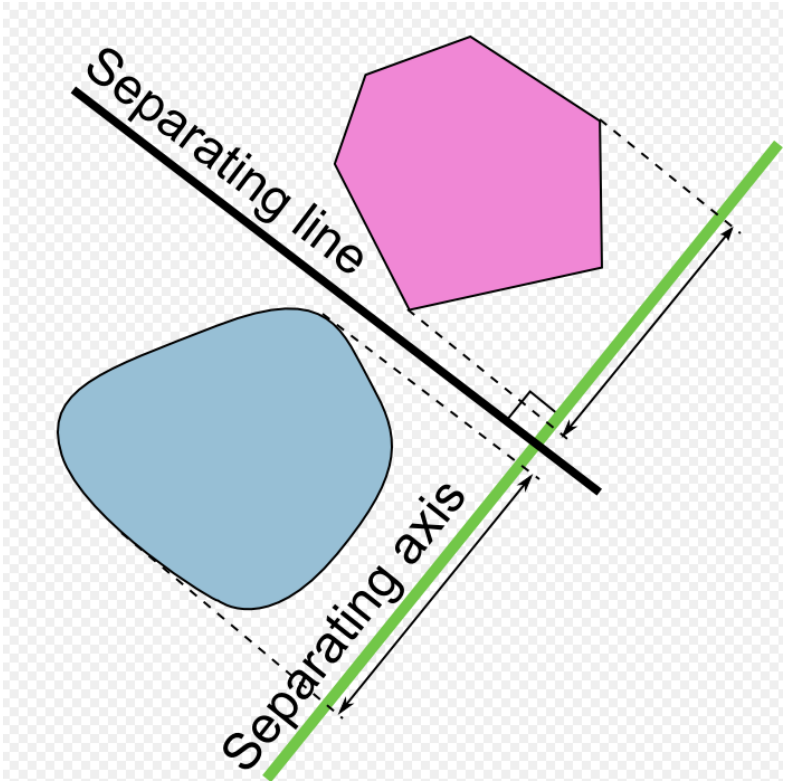
Not Convex

Separating Axis Theorem

Two convex shapes are not colliding *if and only if* there exists a line that separates the two

- In other words, if you can draw a line between two convex shapes without touching either, then the two shapes are not colliding.
- Otherwise, if no such line can be found, the shapes are definitely colliding
- In practice, only a few interesting lines need to be considered (such as edges)

More reading:
https://en.wikipedia.org/wiki/Hyperplane_separation_theorem

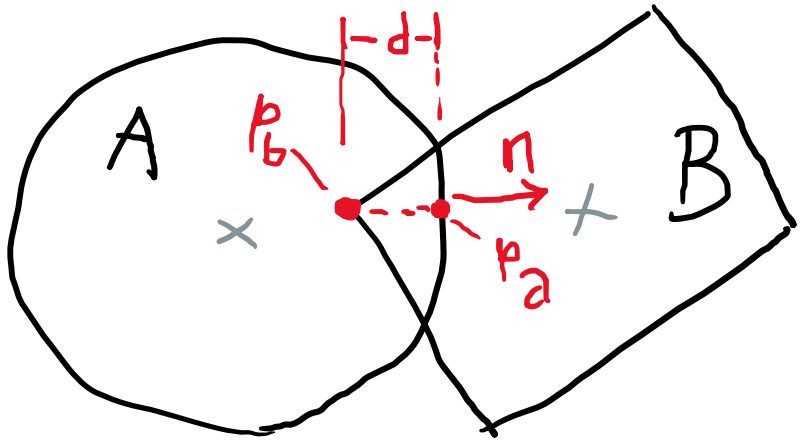


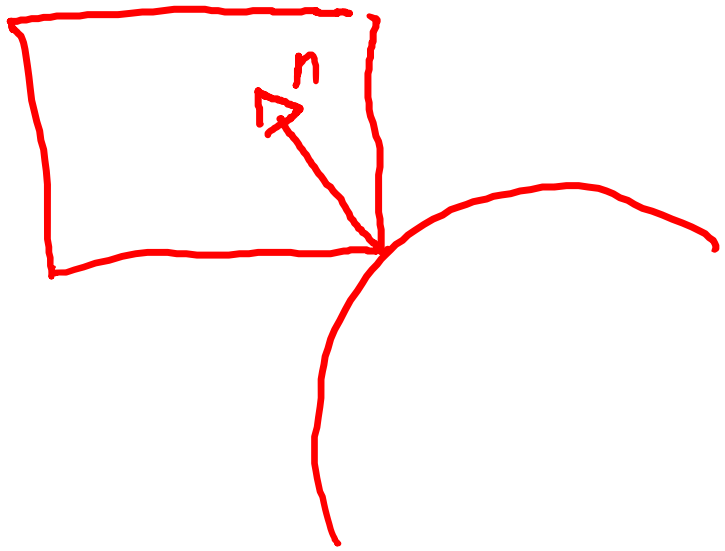
Describing a Collision

A few things are needed to describe a single collision between two rigid bodies:

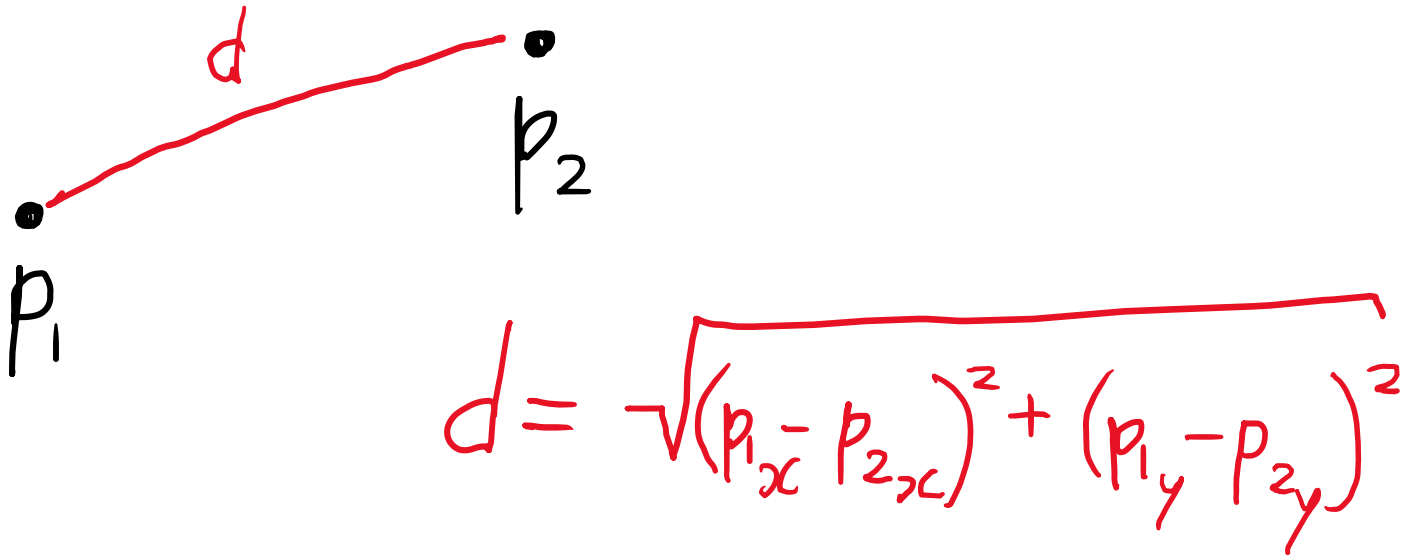
1. The points of collision p_a and p_b both bodies
2. The collision normal n (e.g. the surface normal at the point of collision)
3. The collision depth d (e.g. how deeply the bodies have passed into each other)

The two rigid bodies involved are denoted *A* and *B*



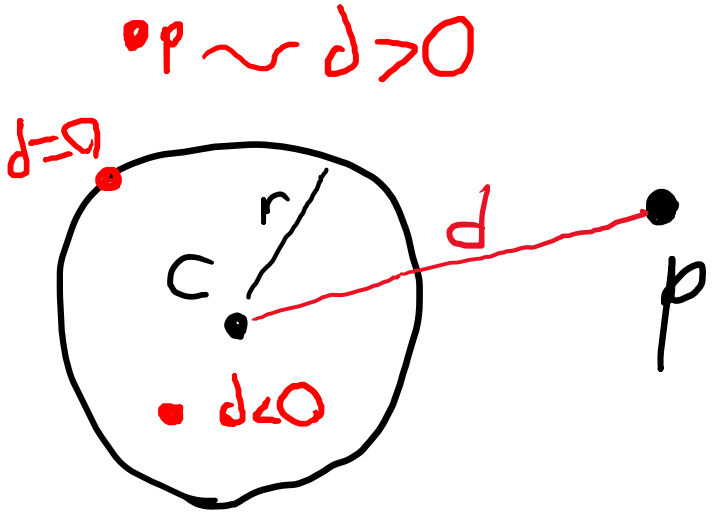


Building Blocks: Point-Point Distance




```
float distancePointToPoint(const vec2& p1, const vec2& p2) {
    return std::hypot(p1.x - p2.x, p1.y - p2.y);
}
```

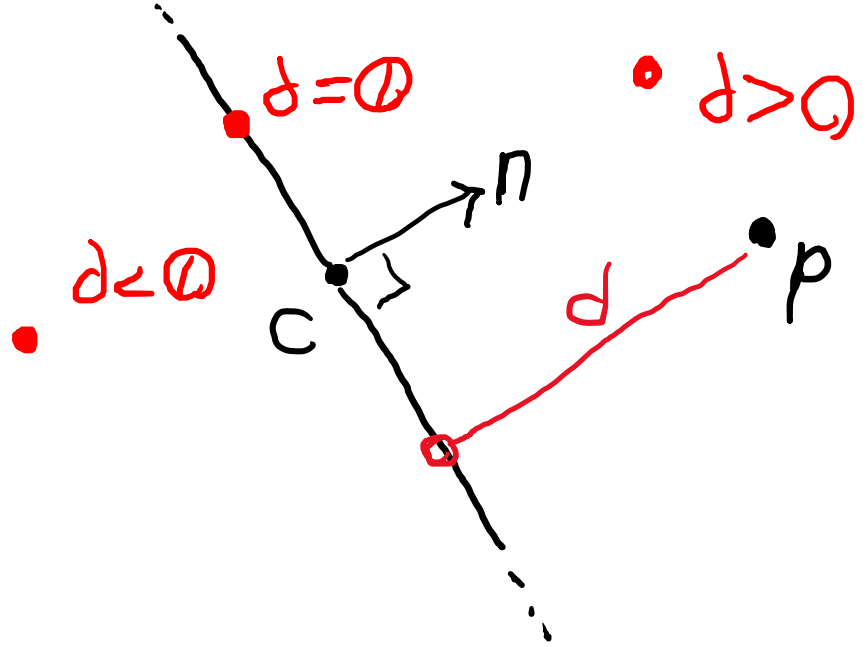
Building Blocks: Point-Circle Signed Distance



$$d = \sqrt{(p_x - c_x)^2 + (p_y - c_y)^2} - r$$

```
float distancePointToCircle(const vec2& p, const vec2& center, float radius) {
    return distancePointToPoint(p, center) - radius;
}
```

Building Blocks: Edge-Line Signed Distance



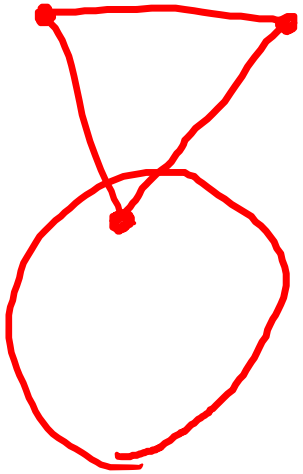
$$d = (p - c) \cdot n$$

$$= (p_x - c_x) * n_{xc} + (p_y - c_y) * n_y$$

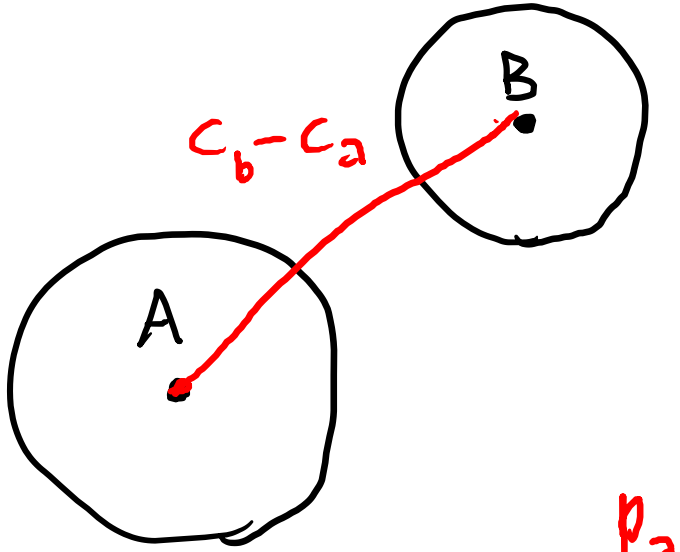
```
float distancePointToLine(const vec2& p, const vec2& pointOnLine, const vec2& normal) {
    return dot(p - pointOnLine) / abs(normal);
}
```

General Approach for Convex Collision Detection

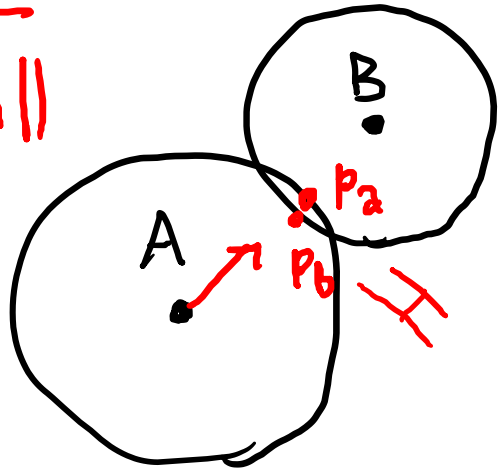
1. Rotate and translate both shapes into the world coordinate frame
2. Loop over all interesting edge-corner pairs between both shapes
    1. If that edge fully separates the two shapes, there is no collision (SAT). Return early.
    2. If the corner is outside the other shape, skip this loop iteration.
    3. Otherwise, find the tentative collision details:
        1. The collision point on one shape is the corner being considered
        2. The other collision point is simply the nearest point to the corner and the edge
        3. The collision normal is the edge normal at its collision point
        4. The collision depth is the distance from the edge
    4. If the collision depth is the biggest yet seen, record these collision details
3. At this point, no separating axis was found and we thus have a collision
4. Return the collision details for the edge-corner pair with the greatest collision depth



Circle-Circle Collision



$$v = \frac{c_b - c_a}{\|c_b - c_a\|}$$

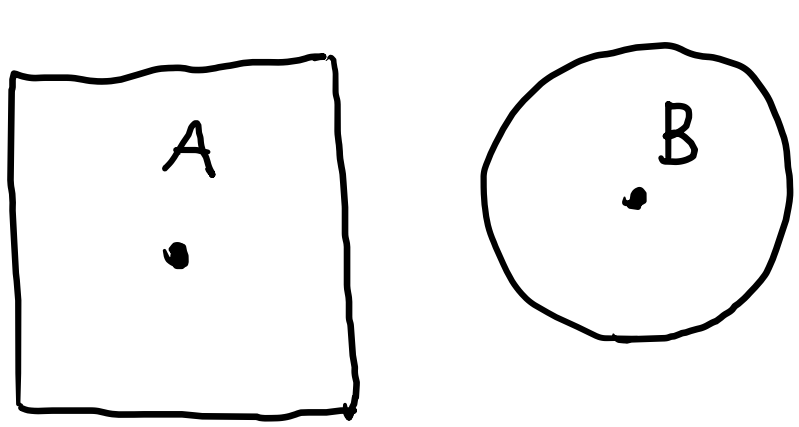


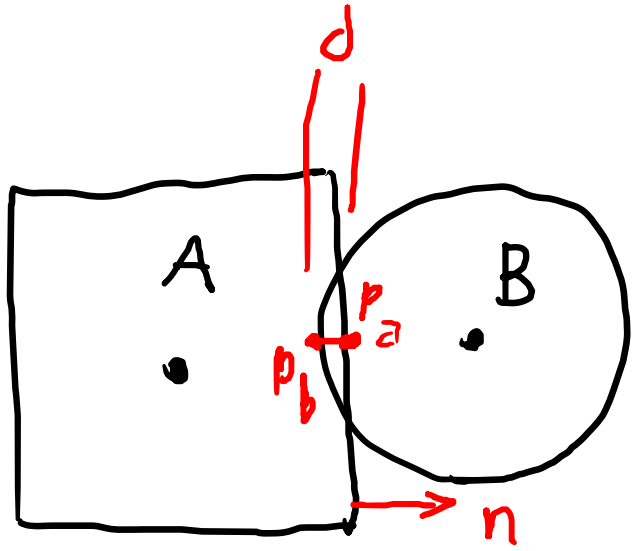
$$p_a = c_a + v * r_a$$

$$p_b = c_b - v * r_b$$

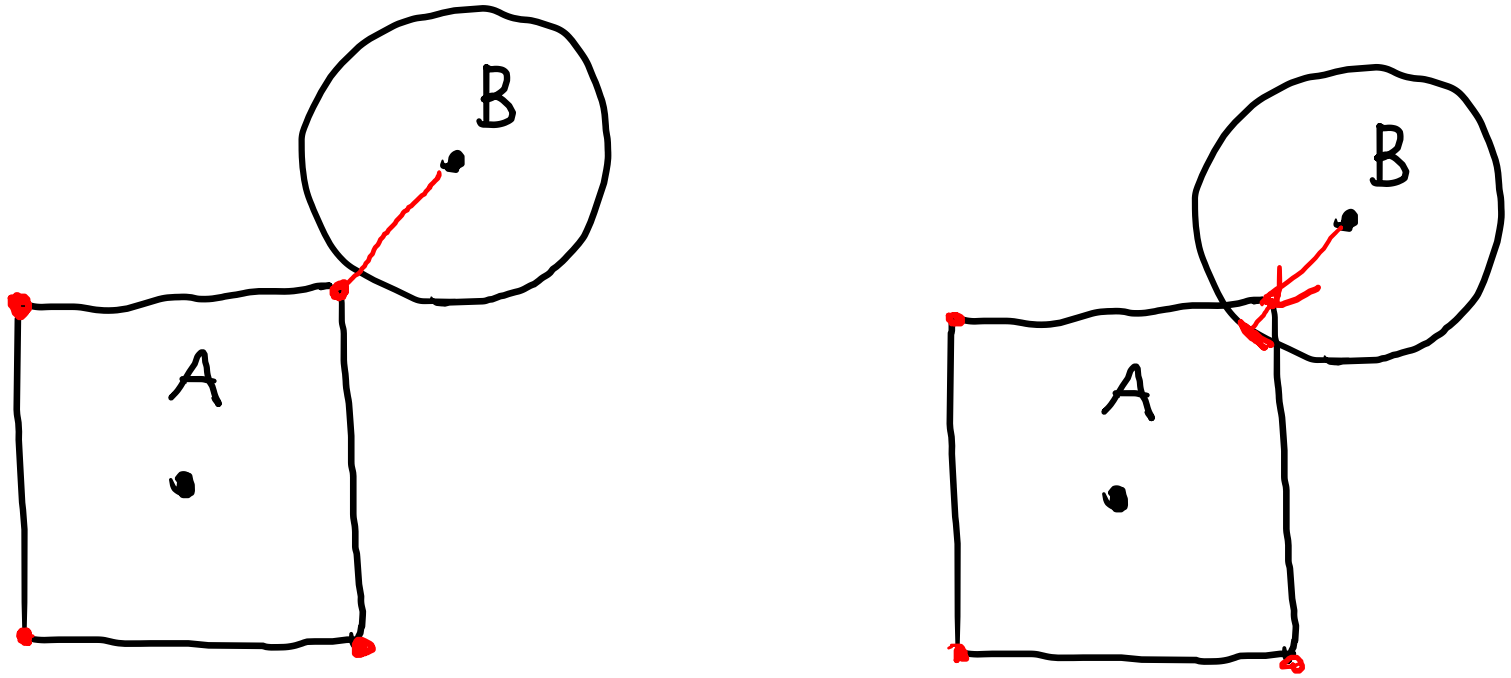
(1) $$d = \|c_b - c_a\| - r_a - r_b$$

Circle-Rectangle Collision: Edge Case

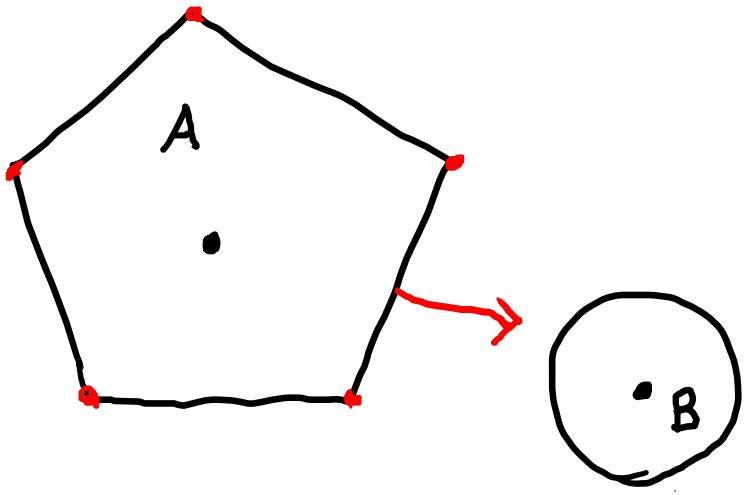


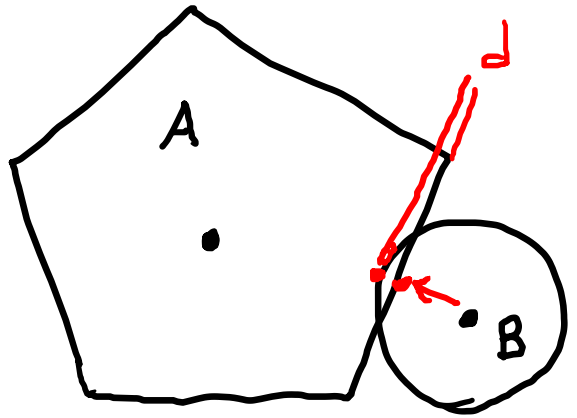


Circle-Rectangle Collision: Corner Case

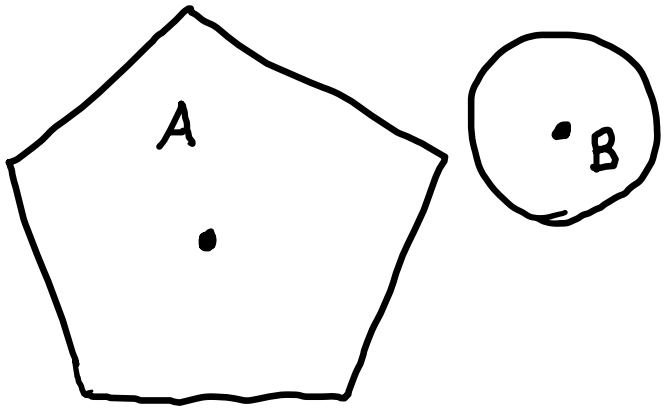


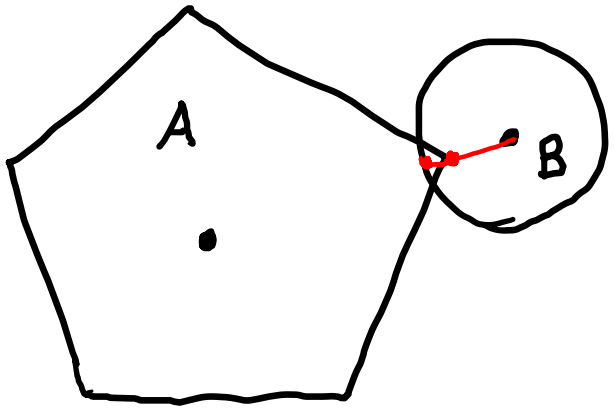
Circle-Polygon Collision: Edge Case



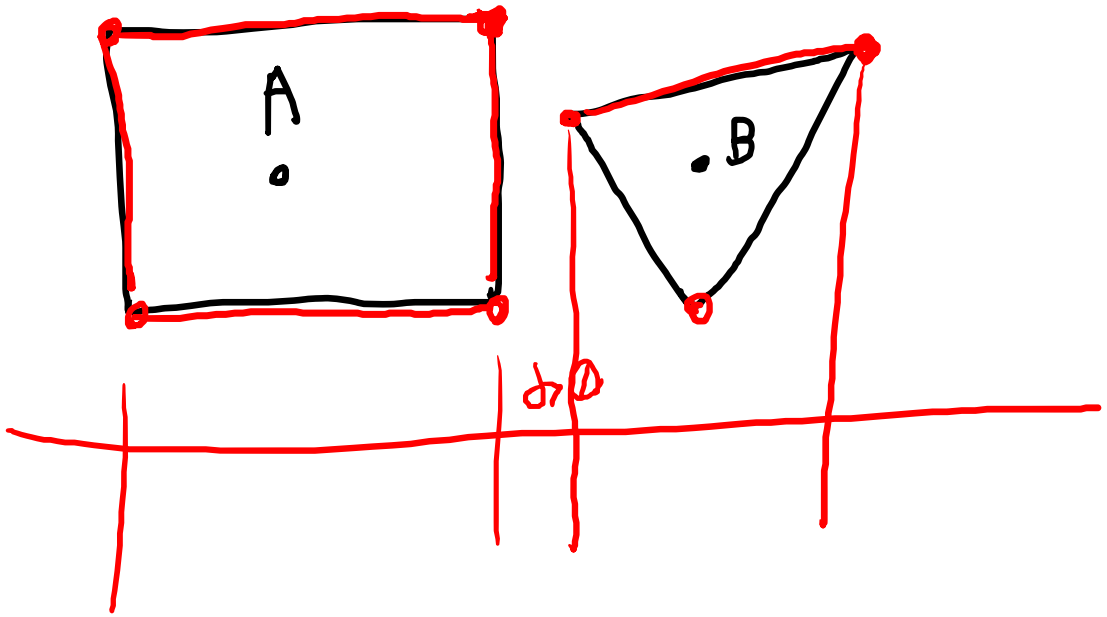


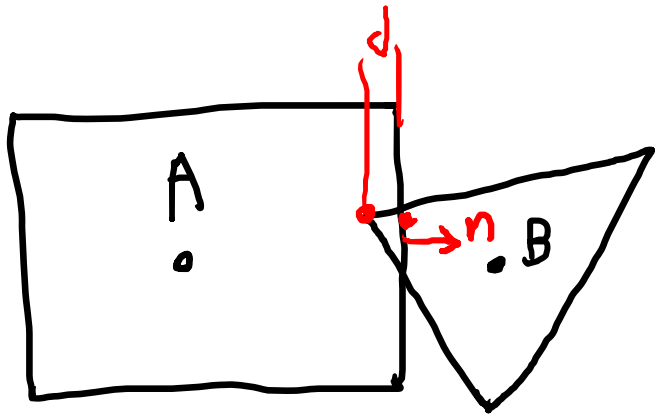
Circle-Polygon Collision: Corner Case



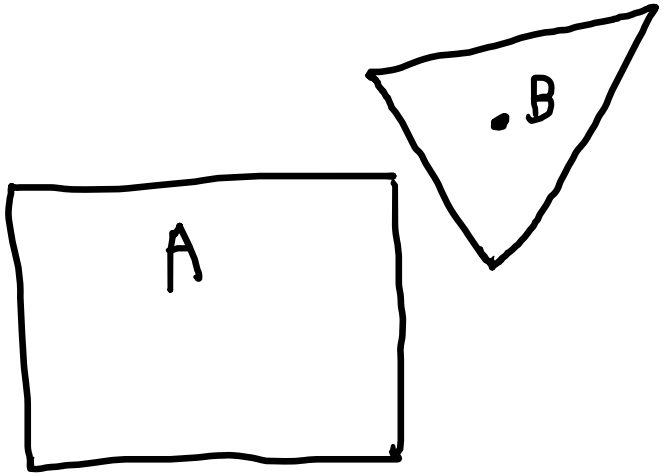


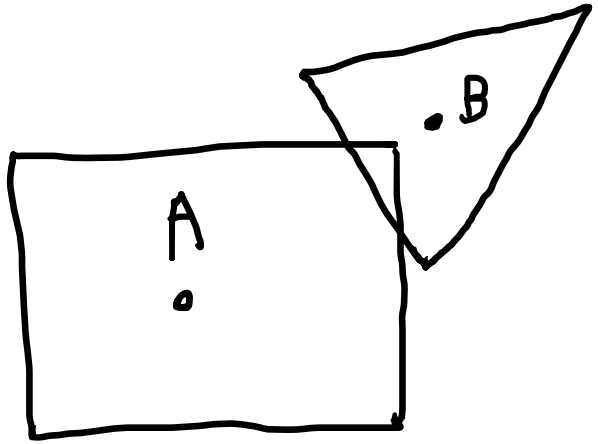
Polygon-Polygon Collision: Edge Case





Polygon-Polygon Collision: Corner Case





Collision Resolution
using Sequential Impulses

Background: Force-Based Solvers

Not used Sequential Impulses

- Find the change in acceleration needed to prevent objects from passing through each other
- Pros:
  - Can bring objects stably to rest
  - Well-studied
- Cons:
  - Serious drift (no way to directly modify positions or velocities, so errors accumulate quickly)
  - Forces become degenerate as time step is made smaller (unintuitive and numerically unstable)

Background: Impulse-Based Solvers

Used in Sequential Impulses

- Find the change in velocity needed to prevent objects from passing through each other
- Pros:
  - Easier to reason about that force-based solvers
  - No infinite forces
  - Can still think in terms of forces (impulse divided by time step results in force)
- Cons:
  - Some drift (but it’s easier to correct for)

C: point of collision

How to solve a (single) collision

1. Find the relative velocity between the two points of collision
   1. If you're ignoring rotation, this is simply the velocity of both bodies
   2. Otherwise, the velocity is given by $v_p = v + (p - c) \times \omega$

$v_{rel} = v_a - v_b$

e: "restitution"

2. Project the relative velocity onto the collision normal (e.g. discard sliding motion)

$v_n = v_{rel} \cdot n$

$v_n \geq 0$

3. We want this velocity to become zero, which is achieved by applying an impulse

$$J = \frac{-(1+e) * v_n}{\frac{1}{\tilde{M}_a} + \frac{1}{\tilde{M}_b}}$$

(to be applied along n)

where

$$\tilde{M}_a = \frac{1}{\left[\frac{p_a - c}{|p_a - c|}\right]^2 * \frac{1}{I_a} + \frac{1}{M_a}}$$

$e = 0$ → inelastic

$e = 1$ → elastic

Background: Global Solvers

Not used in Sequential Impulses

- Given a set of equations describing collisions, solve them all **at the same time**
- Pros:
  - Accurate
  - Many available algorithms for solving systems of equations
- Cons:
  - Bad at handling inequalities (collisions are represented as inequalities ☹)
  - Solver algorithms are very complicated and difficult to implement
  - Resulting systems equations are absurdly complicated

Background: Global Solvers

Solve:

$$0 \leq \begin{bmatrix} {}^{u}\mathbf{J}_n^{(\ell)}\mathbf{u}^{(\ell+1)} + \frac{{}^{u}\mathbf{C}_n^{(\ell)}}{\Delta t} + \frac{\partial {}^{u}\mathbf{C}_n^{(\ell)}}{\partial t} \\ {}^{u}\mathbf{D}^{T}\,{}^{u}\mathbf{J}_f\mathbf{u}^{(\ell+1)} + {}^{u}\mathbf{E}\,{}^{u}\beta \\ {}^{b}\mathbf{D}^{T}\,{}^{b}\mathbf{J}_f\mathbf{u}^{(\ell+1)} + {}^{b}\mathbf{E}\,{}^{b}\beta \\ \mathbf{U}\,{}^{u}\mathbf{p}_n^{(\ell+1)} - {}^{u}\mathbf{E}^{T}\,{}^{u}\alpha \\ {}^{b}\mathbf{p}_{f\max} - {}^{b}\mathbf{E}^{T}\,{}^{b}\alpha \end{bmatrix} \perp \begin{bmatrix} {}^{u}\mathbf{p}^{(\ell+1)} \\ {}^{u}\alpha \\ {}^{b}\alpha \\ {}^{u}\beta \\ {}^{b}\beta \end{bmatrix} \geq 0.$$

where:

$$\widehat{{}^{\kappa}C_{i\sigma}}(\tilde{\mathbf{q}},\tilde{t}) = {}^{\kappa}C_{i\sigma}(\mathbf{q},t) + \frac{\partial {}^{\kappa}C_{i\sigma}}{\partial \mathbf{q}}\Delta\mathbf{q} + \frac{\partial {}^{\kappa}C_{i\sigma}}{\partial t}\Delta t + \frac{1}{2}\left((\Delta\mathbf{q})^{T}\frac{\partial^2 {}^{\kappa}C_{i\sigma}}{\partial \mathbf{q}^2}\Delta\mathbf{q} + 2\frac{\partial^2 {}^{\kappa}C_{i\sigma}}{\partial \mathbf{q}\partial t}\Delta\mathbf{q}\Delta t + \frac{\partial^2 {}^{\kappa}C_{i\sigma}}{\partial t^2}\Delta t^2\right)$$

$${}^{\kappa}\mathbf{J}_{i\sigma} = \frac{\partial({}^{\kappa}\mathbf{C}_{i\sigma})}{\partial \mathbf{q}}\mathbf{H}$$

$${}^{\kappa}\mathbf{k}_{i\sigma}(\mathbf{q},\mathbf{u},t) = \frac{\partial({}^{\kappa}\mathbf{C}_{i\sigma})}{\partial \mathbf{q}}\frac{\partial \mathbf{H}}{\partial t}\mathbf{u} + \frac{\partial^2({}^{\kappa}\mathbf{C}_{i\sigma})}{\partial \mathbf{q}\partial t}\mathbf{H}\mathbf{u} + \frac{\partial^2({}^{\kappa}\mathbf{C}_{i\sigma})}{\partial t^2},$$

Interactive Simulation of Rigid Body Dynamics in Computer Graphics (Bender et al.)

https://www.youtube.com/watch?v=zzy6u1z_l9A

Background: Local Solvers

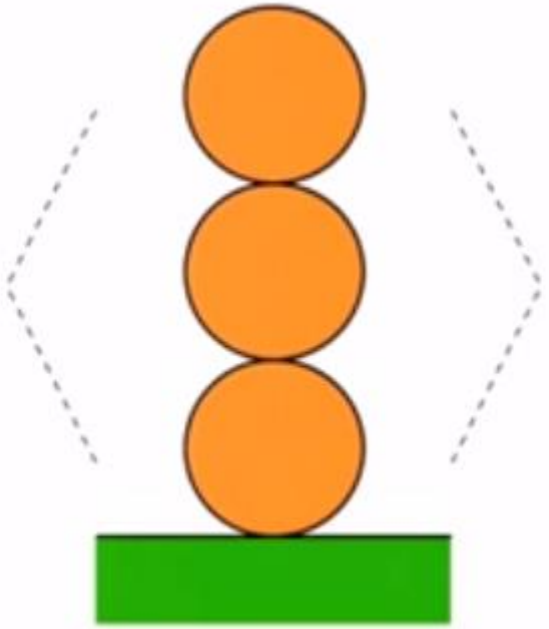
Used in Sequential Impulses

- Given a set of equations describing collisions, solve them **one after the other**
- Pros:
  - Very easy to understand and implement
  - Easily handles inequalities (and thus collisions)
- Cons:
  - Requires several iterations (but so do most global solvers)
  - Convergence trade-offs usually need to be found and tuned manually

Background: Local Solvers

Global

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$



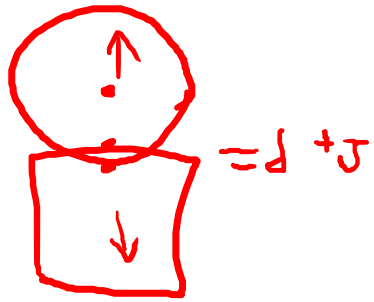
Local (iterative)

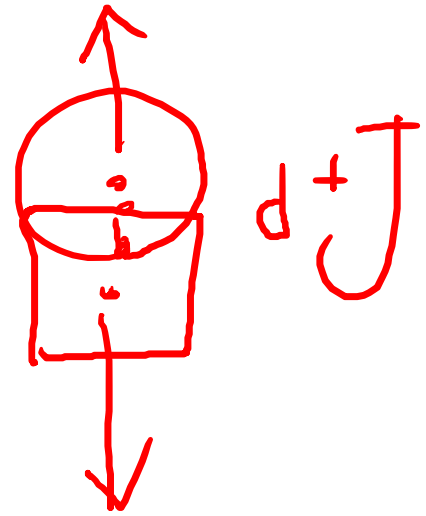
```
for k = 1 to num_iterations
 for i = 1 to 3
  solve  a_ii λ_i = b_i
 end
end
```

How to solve many collisions using Sequential Impulses

The algorithm:

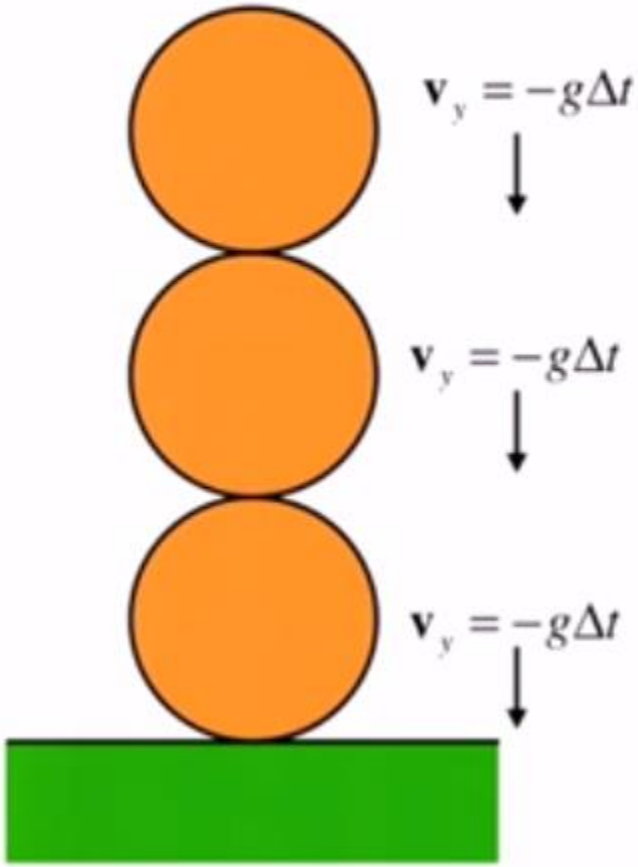
- For each collision, set the accumulated impulse to 0
- For several iterations (e.g. 1 to 50):
  - For every collision:
    - Find the impulse needed to solve the collision (ignoring other collisions)
      - *Slight hack: add a little extra impulse proportional to the collision depth to avoid drift*
    - Add this impulse to the accumulated impulse
    - If the accumulated impulse is less than zero, set it to zero (prevent pulling bodies together)
    - Compute the difference in accumulated impulse from the last iteration, and apply it to both bodies





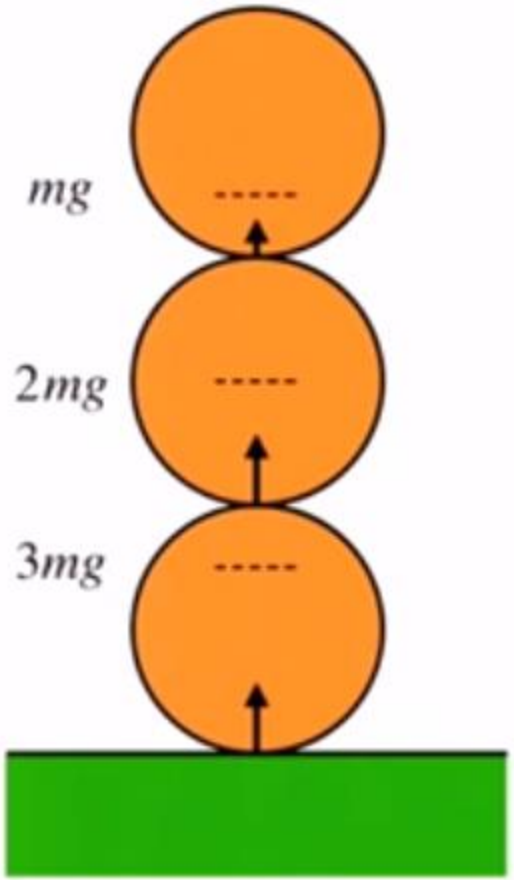
```
oldImpulse = constraint.impulse;
delta = constraint.ComputeImpulse();
constraint.impulse += delta;
constraint.impulse = max(0, constraint.impulse);
delta = constraint.impulse - oldImpulse;
constraint.ApplyImpulse(delta);
```

Sequential Impulses local solver

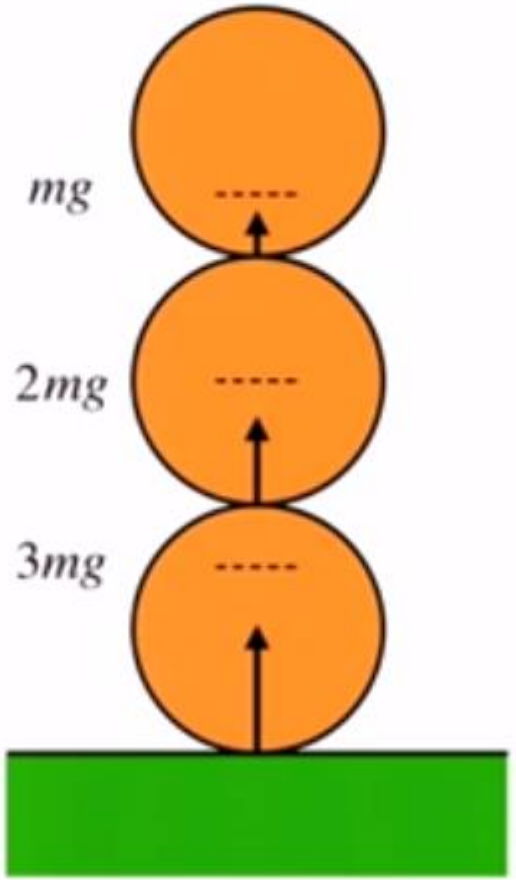


Apply gravity

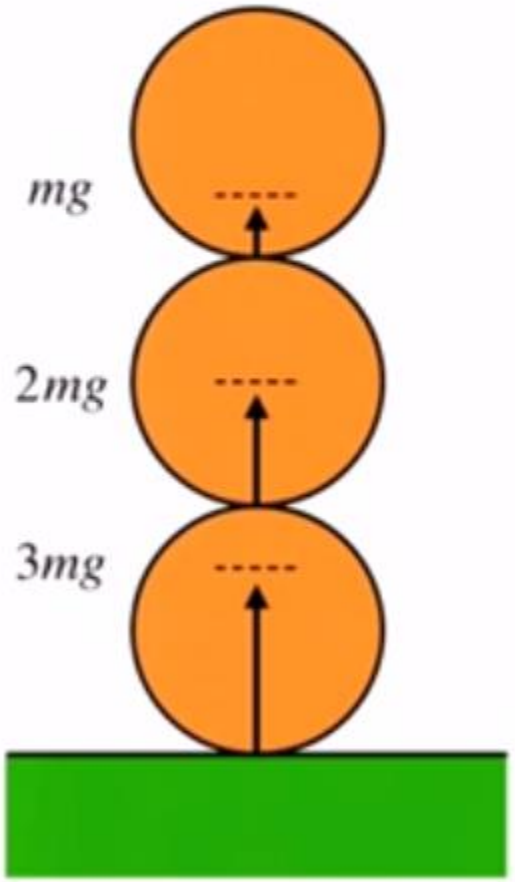
Iteration 1



Iteration 2



Iteration 3




Live Demo


Debugging Tips

Debugging Tips

- Double-check which coordinate frame each point is in.
  - Don’t mix coordinate frames without using the appropriate transformation
- Double-check your units (e.g. kg, meters, meters/second, degrees vs radians)
  - Remember your high-school physics! Dimensional analysis can help rule out many bugs
- Visualize your rigid bodies’ **physical** orientations as used in your physics calculations!
  - These can easily be different from the **visual** orientations if you’re not carefull
- **Visualize your collision points and their depths**
  - This is *extremely* helpful for debugging collision detection

Credits

Some contents in these slides are taken from:

- Erin Catto's GDC 2014 talk: "Understanding Constraints"
  - https://box2d.org/publications/
- Nilson Souto's rigid body dynamics tutorial series:
  - https://www.toptal.com/game/video-game-physics-part-i-an-introduction-to-rigid-body-dynamics