Physics-based Simulation

- simple (independent particles), or complex (robust colliding, stacking, sliding 3D rigid bodies)
- many many simulators!
  - PhysX (Unity, Unreal), Bullet, Open Dynamics Engine, MuJoCo, Havok, Box2D, Chipmunk, OpenSim, RBDL, Simulink (MATLAB), ADAMS, SD/FAST, DART, Vortex, SOFA, Avatar, Project Chrono, Cannon.js, …
  - many course projects, theses, abandon-ware

Resources

- https://processing.org/examples/ see “Simulate”; 2D particle systems
- Non-convex rigid bodies with stacking 3D collision processing and stacking
  http://www.cs.ubc.ca/~rbridson/docs/rigid_bodies.pdf
- Physically-based Modeling, course notes, SIGGRAPH 2001, Baraff & Witkin
  http://www.pixar.com/companyinfo/research/pbm2001/
- Doug James CS 5643 course notes
  http://www.cs.cornell.edu/courses/cs5643/2015sp/
- Rigid Body Dynamics, Chris Hecker http://chrishecker.com/Rigid_Body_Dynamics
- Video game physics tutorial
  https://www.toptal.com/game/video-game-physics-part-i-an-introduction-to-rigid-body-dynamics
- Box2D javascript live demos http://heikobehrens.net/misc/box2d.js/examples/
- Rigid body collisions javascript demo https://www.myphysicslab.com/engine2D/collision-en.html
- Rigid Body Collision Reponse, Michael Manzke, course slides
  https://www.scss.tcd.ie/Michael.Manzke/CS7057/cs7057-1516-09-CollisionResponse-mm.pdf
- Rigid Body Dynamics Algorithms. Roy Featherstone, 2008
- Particle-based Fluid Simulation for Interactive Applications, SCA 2003, PDF
- Stable Fluids, Jos Stam, SIGGRAPH 1999. interactive demo:
  https://29a.ch/2012/12/16/webgl-fluid-simulation

Simulation Basics

- simulation loop
  - Equations of Motion

    sum forces & torques;
    solve for accelerations
  - numerical integration

    update positions, velocities
  - collision detection
  - collision resolution

- basic particle sim

constant dt

$$\boldsymbol{v}(t_{i+1}) = \boldsymbol{v}(t_i) + (\boldsymbol{f}(t_i)/m)dt$$
$$\boldsymbol{p}(t_{i+1}) = \boldsymbol{p}(t_i) + \boldsymbol{v}(t_{i+1})dt$$

$\begin{bmatrix} v_x \\ v_y \end{bmatrix}$, $\begin{bmatrix} p_x \\ p_y \end{bmatrix}$

Physically-based Simulation – movement governed by forces

- particle systems
  - fire, water, smoke, (fluids, cloth, hair)
  - heuristics-based or more principled
- rigid-body simulation
  - blocks, robots, humans
- continuum systems
  - deformable solids
  - fluids, cloth, hair
- group movement
  - flocks, crowds

Ordinary Differential Equations

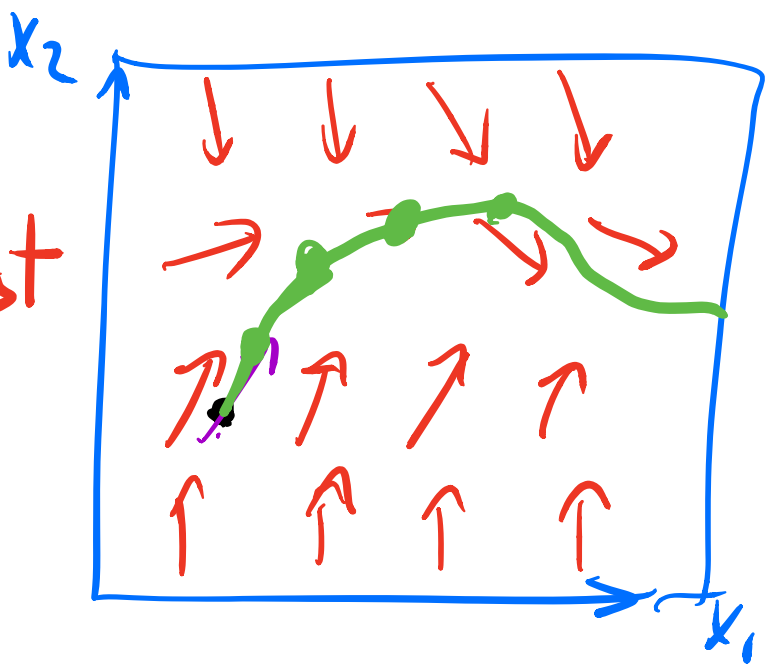
$$\frac{dX(t)}{dt} = f(X(t), t)$$

Given $X_0 = X(t_0)$

Compute $X(t)$ for $t > t_0$

Simulation as a path through state-space, driven by a vector field

$\Delta X = f(x)\,\Delta t$



Newtonian Physics as first-order ODE

- motion of one particle

2nd order ODE

$$\vec{F} = m \frac{d^2\vec{x}}{dt^2}$$

1st order ODE

$$\frac{d}{dt}\begin{bmatrix}\vec{x}\\ \vec{v}\end{bmatrix} = \begin{bmatrix}\vec{v}\\ \Sigma\vec{F}/m\end{bmatrix}$$

- motion of many particles

$$\frac{d}{dt}\begin{bmatrix}x_1\\ v_1\\ x_2\\ v_2\\ \vdots\\ x_n\\ v_n\end{bmatrix} = \begin{bmatrix}v_1\\ F_1/m_1\\ v_2\\ F_2/m_2\\ \vdots\end{bmatrix}$$

Particle Forces

$F = \begin{bmatrix} 0 \\ -mg \\ 0 \end{bmatrix}$

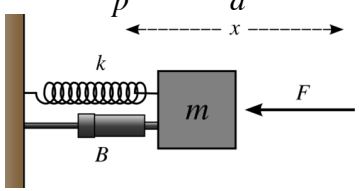
$$\|\boldsymbol{F}_{ij}\| = \frac{G\, m_i\, m_j}{r^2}$$

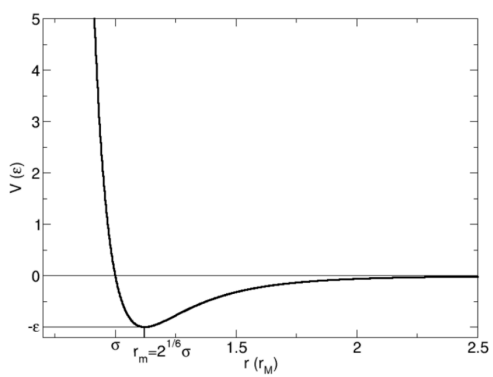
$$f^{(i)} = \begin{pmatrix} 0 \\ 0 \\ -m_i G \end{pmatrix}$$

- gravity
- viscous damping
- springs & dampers
- Lennard-Jones potentials

$$f^{(i)} = -dv^{(i)}$$

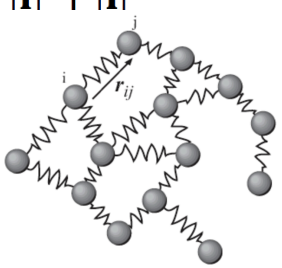
$$F = -k_p x - k_d v$$





$$\mathbf{f}_a = -\left[k_s(|\mathbf{l}| - r) + k_d \frac{\dot{\mathbf{l}} \cdot \mathbf{l}}{|\mathbf{l}|}\right] \frac{\mathbf{l}}{|\mathbf{l}|}, \quad \mathbf{f}_b = -\mathbf{f}_a,$$

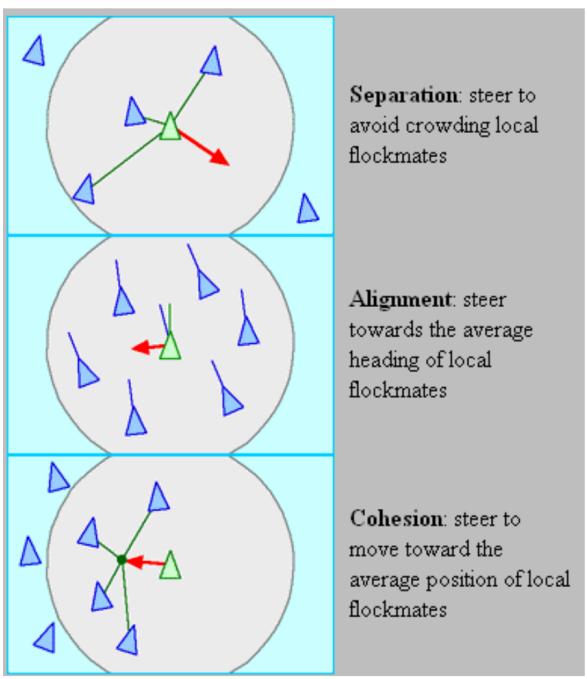
"l" = rest length



Proxy Forces

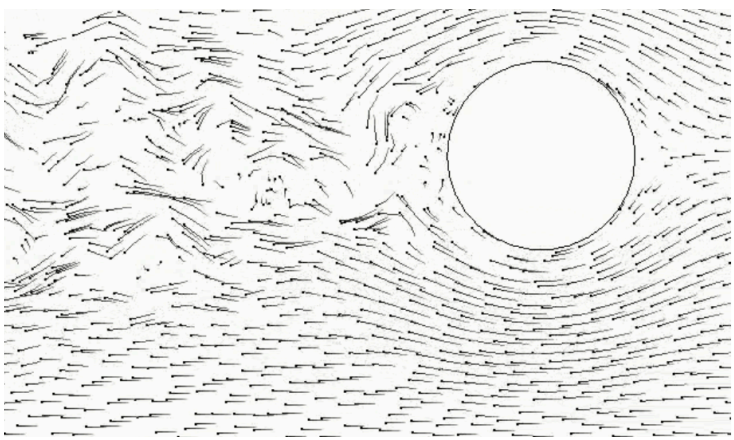
- behavior forces:
  flocking birds, schooling fish, etc.

  ["Boids", Craig Reynolds, SIGGRAPH 1987]



Courtesy of Craig W. Reynolds. Used with permission.

Curl noise for procedural fluid flow
R. Bridson, J. Hourihan, M. Nordenstam,
Proc. SIGGRAPH 2007.



ODE Numerical Integration: Explicit (Forward) Euler

generally fine for basic non-interactive particles.

$$\frac{dX}{dt} = f(X)$$

$$\Delta X = \Delta t\, f(X)$$

$$t_1 = t_0 + h$$

$$\mathbf{X}_1 = \mathbf{X}_0 + h\, f(\mathbf{X}_0, t_0)$$

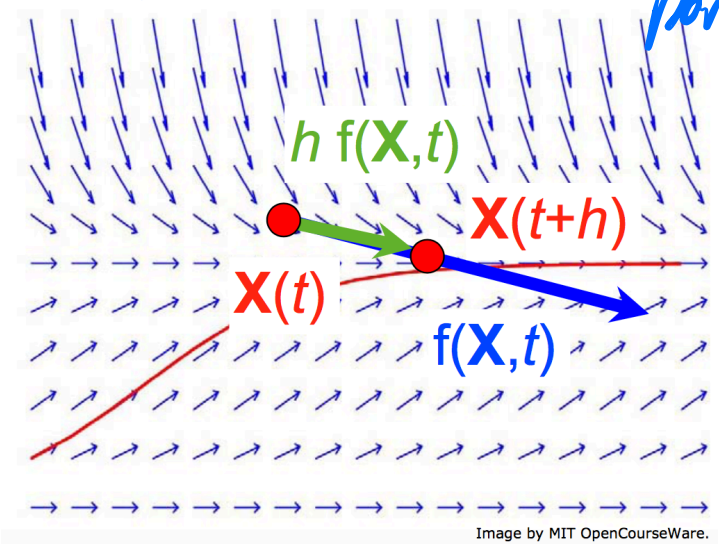


Image by MIT OpenCourseWare.

Images from MIT-OCW:

Wojciech Matusik, and Frédo Durand. *6.837 Computer Graphics.* Fall 2012. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu. License: Creative Commons BY-NC-SA.

Explicit Euler: problems

Solution spirals outwards no matter how small the time step !

(although smaller time steps are still better)

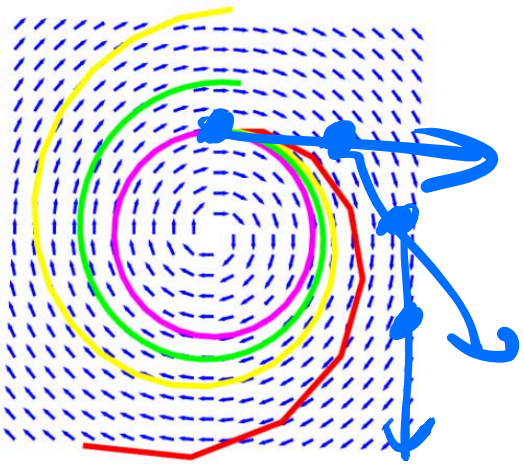
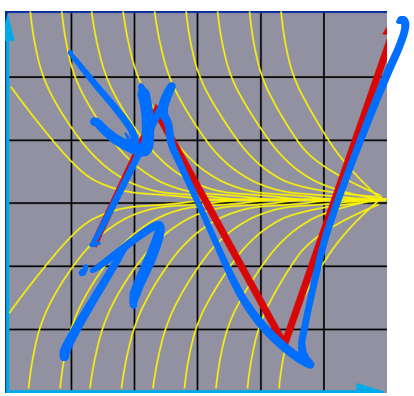


Image by MIT OpenCourseWare.

Or can lead to instabilities:



[Physically Based Animation
Witkin & Baraff, SIGGRAPH course notes]

Midpoint & Trapezoid methods

- Midpoint:
  - ½ Euler step
  - evaluate f_m
  - full step using f_m
- Trapezoid:
  - Euler step (a)
  - evaluate f_1
  - full step using f_1 (b)
  - average (a) and (b)
- Not exactly same result, but same order of accuracy

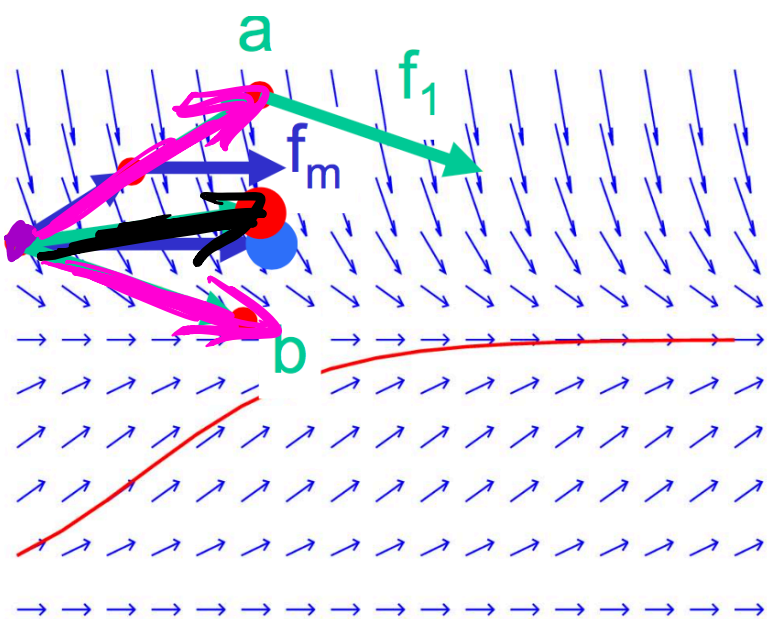
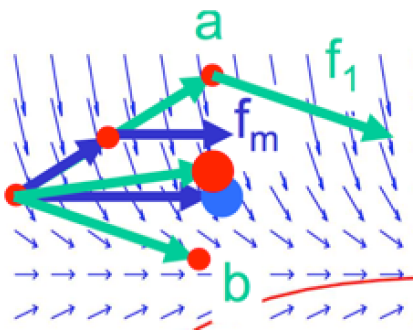


Image by MIT OpenCourseWare.

Second order accurate;
Midpoint method also known as
"second order Runge-Kutta"

[slide in part from MIT-OCW]

Code



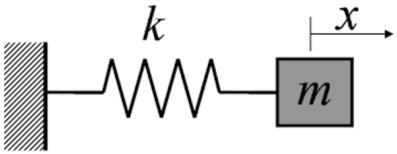
EXPLICIT EULER

```
void takeStep(ParticleSystem* ps, float h)
{
        velocities = ps->getStateVelocities()
        positions = ps->getStatePositions()
        forces = ps->getForces(positions, velocities)
        masses = ps->getMasses()
        accelerations = forces / masses
        newPositions = positions + h*velocities
        newVelocities = velocities  + h*accelerations
        ps->setStatePositions(newPositions)
        ps->setStateVelocities(newVelocities)
}
```

MIDPOINT METHOD

```
void takeStep(ParticleSystem* ps, float h)
{
        velocities = ps->getStateVelocities()
        positions = ps->getStatePositions()
        forces = ps->getForces(positions, velocities)
        masses = ps->getMasses()
        accelerations = forces / masses
        midPositions = positions + 0.5*h*velocities
        midVelocities = velocities  + 0.5*h*accelerations
        midForces = ps->getForces(midPositions, midVelocities)
        midAccelerations = midForces / masses
        newPositions = positions + [illegible]*h*midVelocities
        newVelocities = velocities  + [illegible]*h*midAccelerations
        ps->setStatePositions(newPositions)
        ps->setStateVelocities(newVelocities)
}
```

Implicit Euler

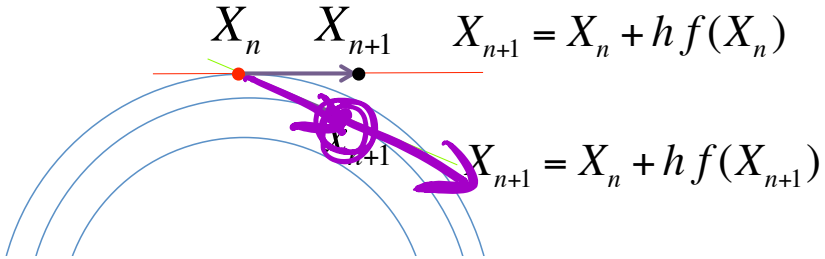


Idea: Use the derivative at the destination
Problem: We don't know the destination yet !!

forward "explicit" Euler

$$x_{n+1} = x_n + h v_n$$

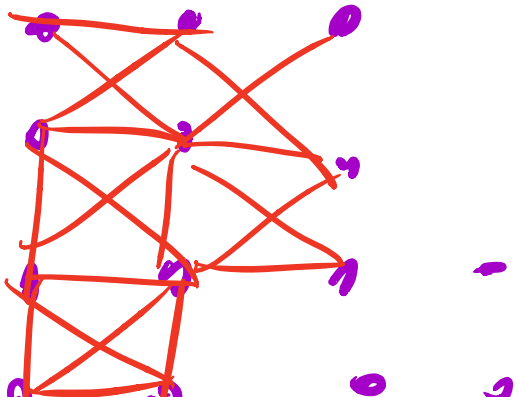
$$v_{n+1} = v_n + h\left(\frac{-k x_n}{m}\right)$$



backward "implicit" Euler

$$x_{n+1} = x_n + h v_{n+1}$$

$$v_{n+1} = v_n + h\left(\frac{-k x_{n+1}}{m}\right)$$

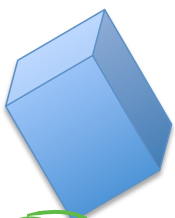


Rigid Body Dynamics

- from particles to rigid bodies…

state: $\vec{x}$ position, $\vec{v}$ velocity $\} \mathbb{R}^6$ in 3D, $\mathbb{R}^4$ in 2D

state: $\in \mathbb{R}^{12}$ $\{ \vec{x}, \vec{v}, q, R$ (rotation matrix 3×3$), \vec{\omega} \}$



Newton's equations of motion

$$\Sigma \vec{F} = m\vec{a}$$

$$\begin{bmatrix} m & & \\ & m & \\ & & m \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} \Sigma F \end{bmatrix}$$

$$M\vec{a} = \Sigma F$$

Newton-Euler equations of motion

$$\begin{bmatrix} m & & & & & \\ & m & & & I & \\ & & m & & & \\ & & & & & \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_z \\ \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = \begin{bmatrix} \Sigma F \\ \\ \\ \Sigma \tau - \vec{\omega} \times I\vec{\omega} \end{bmatrix}$$

Coriolis term

inertia tensor

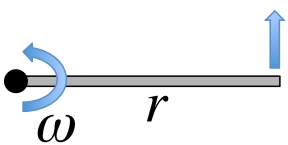
Preliminaries

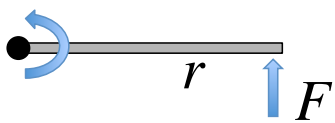
- cross product via a matrix multiply

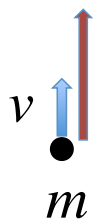
$$\tilde{a} = \begin{pmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{pmatrix}$$

Kinematics of Rotation

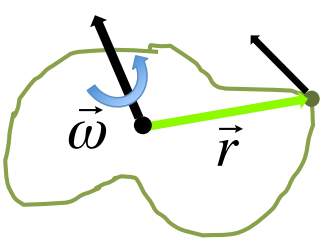
- Intuitively, with scalars:

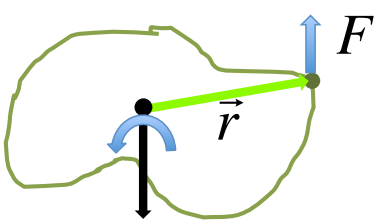


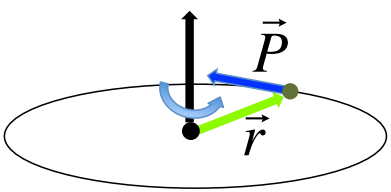




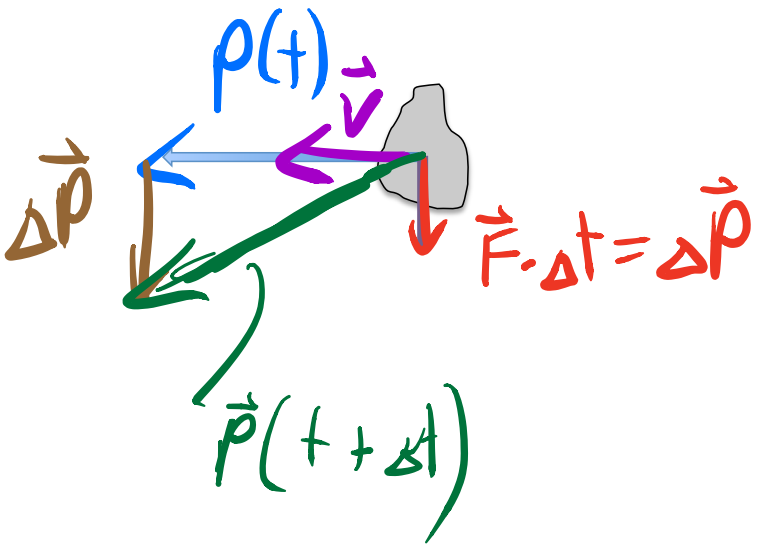
- More generally:







Newton's Law



momentum

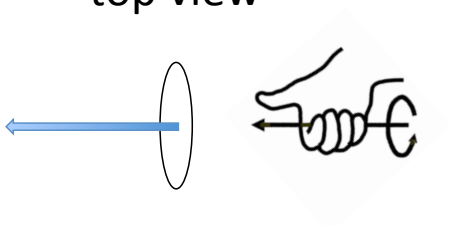
$$\vec{p} = m \cdot \vec{v}$$

$$\frac{d\vec{p}}{dt} = \sum_i F_i$$

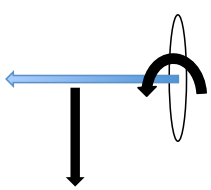
$$\frac{d(m\vec{v})}{dt} = m \cdot a = \Sigma F$$

Euler's Law

top view



side view



Angular Momentum of a Set of Particles

Inertia Tensor

Newton-Euler Equations of Motion

Newton:

$$\Sigma F = \frac{dP}{dt} = \frac{d(m\vec{v})}{dt} =$$

$$= \dot{m}\vec{v}^{\,0} + m\dot{\vec{v}}$$

$$= m\dot{\vec{v}}$$

Euler:

$$\Sigma \tau = \frac{dL}{dt} = \frac{d(I\vec{\omega})}{dt}$$

$L = I\omega$ ← angular momentum.

$$= \dot{I}\vec{\omega} + I\dot{\vec{\omega}}$$

$$= \omega \times I\omega + I\dot{\vec{\omega}}$$

Scalar

$\dot{\vec{\omega}}$ ← angular acceleration.

In 2D: $\Sigma \tau = I\dot{\omega}$ ← Scalar

Updating the Inertia Tensor

Simulation Loop

2D 3D

linear position
linear velocity
angular orientation
angular velocity

for each timestep

$\vec{x}$ $\vec{v}$ θ ω } scalars

$\vec{x}$ $\vec{v}$ R 3x3 → or, more often use quaternion q $\vec{\omega}$

setup

- compute forces & torques

- compute forces & torques.

$I_w = R I_L R^{-1}$

solve eqns of motion

$\Sigma \vec{F} = m\vec{\dot{v}}$

$\Sigma \tau = I\dot{\omega}$

$\Sigma \vec{F} = m\vec{\dot{v}}$

$\Sigma \tau = \omega \times I\omega + I\dot{\omega}$

integrate

$x = x + v\Delta t$
$v = v + \dot{v}\Delta t$
$\theta = \theta + \omega\Delta t$
$\omega = \omega + \dot{\omega}\Delta t$

$x = x + v\Delta t$
$v = v + \dot{v}\Delta t$
$R = R + \dot{R}\Delta t$
$\omega = \omega + \dot{\omega}\Delta t$

3x3

$\dot{R} = \tilde{\omega} R$

Rigid Body Collisions

- Collision Detection
  - broad phase: e.g., AABBs, bounding spheres
  - narrow phase: detailed checks
- Collision Response
  - collision impulses
  - constraint forces: resting, sliding, hinges, ….

Particle-Plane Collisions

https://www.scss.tcd.ie/Michael.Manzke/CS7057/cs7057-1516-09-CollisionResponse-mm.pdf

17

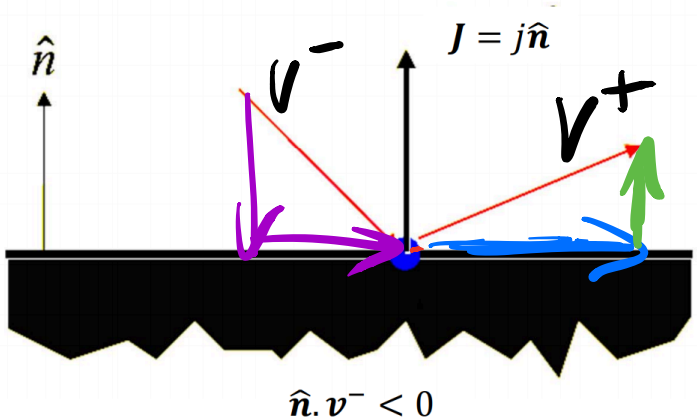
PARTICLE-PLANE: FRICTIONLESS IMPULSE RESPONSE

$\epsilon = 0.9$

Change in velocity caused by applying an Impulse in direction normal to plane, and of magnitude j

$v_{\perp}^{+} = -\epsilon v_{\perp}^{-}$ $\quad v_{\parallel}^{+} = v_{\parallel}^{-}$

Invert and scale perpendicular component of velocity



$$v^+ = \frac{J}{m} + v^-$$

$$J = j\hat{n}$$

$$j = (1+\epsilon)m$$

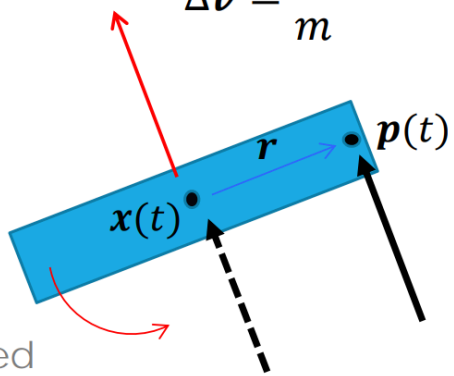
Not this easy for Rigid Bodies

Impulse causes change in Linear and Angular velocity

The effect of an impulse (or for that matter a force) on an object's linear and angular momentum are independent

- Linear component: similar to particles
  - Causes a change in velocity inversely proportional to mass
  - As if force was applied at c.o.m.
- Angular component: impulsive torque
  - Causes change in angular velocity inversely proportional to moment of inertia (determined from inertial tensor)
  - Dependent on position of impulse

$$\Delta \boldsymbol{v} = \frac{\boldsymbol{J}}{m}$$



$$\Delta \boldsymbol{\omega} = \boldsymbol{I}^{-1}(\boldsymbol{r} \times \boldsymbol{J})$$
$$= \boldsymbol{I}^{-1}((\boldsymbol{p} - \boldsymbol{x}) \times \boldsymbol{J})$$

But what is the value of $\boldsymbol{J}$?

RIGID BODY COLLISION

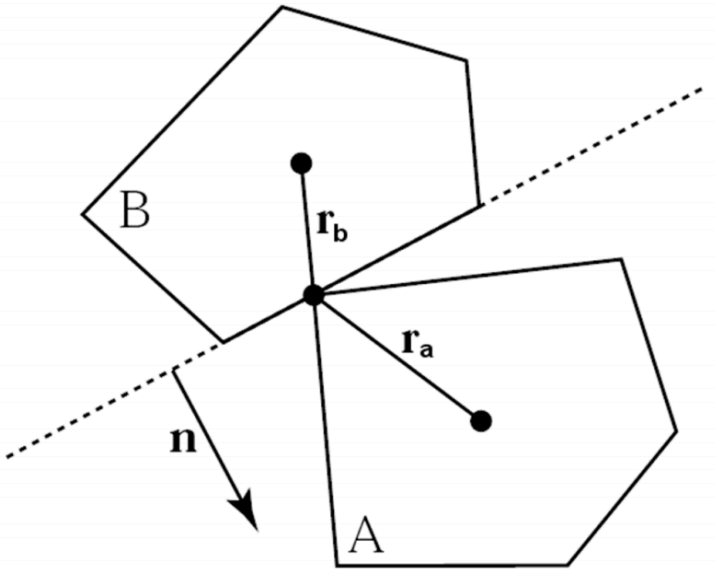
https://www.scss.tcd.ie/Michael.Manzke/CS7057/cs7057-1516-09-CollisionResponse-mm.pdf

$$v_{rel}^{+} = -\epsilon v_{rel}^{-}$$

$$v_{rel} = \hat{n}(\dot{p}_A - \dot{p}_B)$$

$$\begin{aligned}\dot{p}_A &= v_A + \omega_A \times (p_A - x_A) \\ &= v_A + \omega_A \times r_A\end{aligned}$$

$$\begin{aligned}\dot{p}_A &= v_B + \omega_B \times (p_B - x_B) \\ &= v_A + \omega_A \times r_A\end{aligned}$$



Linear velocity component

Angular component: Linear velocity of point p due to its rotation. See previous slide

https://www.scss.tcd.ie/Michael.Manzke/CS7057/cs7057-1516-09-CollisionResponse-mm.pdf

Post-collision velocity should be

$$\dot{\boldsymbol{p}}_A^+ = \boldsymbol{v}_A^+ + \boldsymbol{\omega}_A^+ \times \boldsymbol{r}_A$$

However we need to express this in terms of the previous state of the object (pre-collision values)

$$\boldsymbol{v}_A^+ = \boldsymbol{v}_A^- + \Delta\boldsymbol{v} = \boldsymbol{v}_A^- + \frac{j\hat{\boldsymbol{n}}}{m_A}$$

SEE SLIDE 19

$$\boldsymbol{\omega}_A^+ = \boldsymbol{\omega}_A^- + \Delta\boldsymbol{\omega} = \boldsymbol{\omega}_A^- + \boldsymbol{I}_A^{-1}(\boldsymbol{r}_A \times j\hat{\boldsymbol{n}})$$

$$\dot{\boldsymbol{p}}_A^+ = \boldsymbol{v}_A^- + \frac{j\hat{\boldsymbol{n}}}{m_A} + (\boldsymbol{\omega}_A^- + \boldsymbol{I}_A^{-1}(\boldsymbol{r}_A \times j\hat{\boldsymbol{n}}))\times \boldsymbol{r}_A$$

$$= \boldsymbol{v}_A^- + \boldsymbol{\omega}_A^- \times \boldsymbol{r}_A + j\left(\frac{\hat{\boldsymbol{n}}}{m_A} + \boldsymbol{I}_A^{-1}(\boldsymbol{r}_A \times \hat{\boldsymbol{n}})\right)\times \boldsymbol{r}_A$$

PREVIOUS SLIDE

$$= \dot{\boldsymbol{p}}_A^- + j\left(\frac{\hat{\boldsymbol{n}}}{m_A} + \boldsymbol{I}_A^{-1}(\boldsymbol{r}_A \times \hat{\boldsymbol{n}})\right)\times \boldsymbol{r}_A$$

Similarly (by Newton's 3rd Law: every reaction has equal and opposite reaction):

$$\dot{\boldsymbol{p}}_B^+ = \dot{\boldsymbol{p}}_B^- + j\left(\frac{\hat{\boldsymbol{n}}}{m_A} + \boldsymbol{I}_A^{-1}(\boldsymbol{r}_A \times \hat{\boldsymbol{n}})\right)\times \boldsymbol{r}_A$$

https://www.scss.tcd.ie/Michael.Manzke/CS7057/cs7057-1516-09-CollisionResponse-mm.pdf

Putting this in $v_{rel}^{+} = \hat{\boldsymbol{n}}(\dot{\boldsymbol{p}}_A^{+} - \dot{\boldsymbol{p}}_B^{+})$ *we get*

From previous slide

$$v_{rel}^{+} = \hat{\boldsymbol{n}}\left(\left(\boldsymbol{p}_A^{-} + j\left(\frac{\hat{\boldsymbol{n}}}{m_A} + \boldsymbol{I}_A^{-1}(\boldsymbol{r}_A \times \hat{\boldsymbol{n}})\right) \times \boldsymbol{r}_A\right) - \left(\boldsymbol{p}_B^{-} + j\left(\frac{\hat{\boldsymbol{n}}}{m_B} + \boldsymbol{I}_B^{-1}(\boldsymbol{r}_B \times \hat{\boldsymbol{n}})\right) \times \boldsymbol{r}_B\right)\right)$$

$$= \hat{\boldsymbol{n}}(\dot{\boldsymbol{p}}_A^{-} - \dot{\boldsymbol{p}}_B^{-}) + j\left(\frac{1}{m_A} + \frac{1}{m_B} + \hat{\boldsymbol{n}}\left(\boldsymbol{I}_A^{-1}(\boldsymbol{r}_A \times \hat{\boldsymbol{n}})\right) \times \boldsymbol{r}_A + \hat{\boldsymbol{n}}\left(\boldsymbol{I}_B^{-1}(\boldsymbol{r}_B \times \hat{\boldsymbol{n}})\right) \times \boldsymbol{r}_B\right)$$

$$= v_{rel}^{-} + j\left(\frac{1}{m_A} + \frac{1}{m_B} + \hat{\boldsymbol{n}}\left(\boldsymbol{I}_A^{-1}(\boldsymbol{r}_A \times \hat{\boldsymbol{n}})\right) \times \boldsymbol{r}_A + \hat{\boldsymbol{n}}\left(\boldsymbol{I}_B^{-1}(\boldsymbol{r}_B \times \hat{\boldsymbol{n}})\right) \times \boldsymbol{r}_B\right)$$

And since $v_{rel}^{+} = -\epsilon v_{rel}^{-}$

$$-\epsilon v_{rel}^{-} = v_{rel}^{-} + j\left(\frac{1}{m_A} + \frac{1}{m_B} + \hat{\boldsymbol{n}}\left(\boldsymbol{I}_A^{-1}(\boldsymbol{r}_A \times \hat{\boldsymbol{n}})\right) \times \boldsymbol{r}_A + \hat{\boldsymbol{n}}\left(\boldsymbol{I}_B^{-1}(\boldsymbol{r}_B \times \hat{\boldsymbol{n}})\right) \times \boldsymbol{r}_B\right)$$

giving $$j = \frac{-(1+\epsilon)\, v_{rel}^{-}}{m_A^{-1} + m_B^{-1} + \hat{\boldsymbol{n}}\left(\boldsymbol{I}_A^{-1}(\boldsymbol{r}_A \times \hat{\boldsymbol{n}})\right) \times \boldsymbol{r}_A + \hat{\boldsymbol{n}}\left(\boldsymbol{I}_B^{-1}(\boldsymbol{r}_B \times \hat{\boldsymbol{n}})\right) \times \boldsymbol{r}_B}$$

This gives us the impulse magnitude we were looking for

https://www.scss.tcd.ie/Michael.Manzke/CS7057/cs7057-1516-09-CollisionResponse-mm.pdf

IMPULSE MAGNITUDE EQUATION

WHERE THE VARIABLES COME FROM

$$j = \frac{-(1+\epsilon)\,v_{rel}^{-}}{m_A^{-1} + m_B^{-1} + \hat{n}\left(I_A^{-1}(r_A \times \hat{n})\right) \times r_A + \hat{n}\left(I_B^{-1}(r_B \times \hat{n})\right) \times r_B}$$

$$I_A^{-1} = R_A I_{bodyA}^{-1} R_A^T \quad \text{and} \quad I_B^{-1} = R_B I_{bodyB}^{-1} R_B^T$$

$$r_A = p_A - x_A \quad \text{and} \quad r_B = p_B - x_B$$

$$v_{rel}^{-} = \hat{n}(\dot{p}_A^{-} - \dot{p}_B^{-})$$

$$\dot{p}_A = v_A + \omega_A \times (p_A - x_A) \quad \text{and} \quad \dot{p}_B = v_B + \omega_B \times (p_B - x_B)$$

Constants

Intermediate Variables

Rigid Body State

Contact Model

j: impulse magnitude

ϵ: coefficient of restitution

m_A, m_B: mass of object A and B

$\mathbf{R}_A, \mathbf{R}_B$: orientation of A and B

$\boldsymbol{\omega}_A, \boldsymbol{\omega}_B$: angular velocity of A and B

$\mathbf{x}_A, \mathbf{x}_B$: centre of mass position of A and B

$\mathbf{v}_A, \mathbf{v}_B$: velocity of A and B

v_{rel}^{-}: pre-collision relative velocity of contact points

$\mathbf{I}_A, \mathbf{I}_B$: world space inertial tensor

$\mathbf{I}_{bodyA}, \mathbf{I}_{bodyB}$: object space inertial tensor

$\hat{\mathbf{n}}$: contact plane normal

$\mathbf{p}_A, \mathbf{p}_B$: contact point on A and B

https://www.scss.tcd.ie/Michael.Manzke/CS7057/cs7057-1516-09-CollisionResponse-mm.pdf

APPLYING IMPULSE

Impulse magnitude is given by:

$$j = \frac{-(1+\epsilon)\ v_{rel}^{-}}{m_A^{-1} + m_B^{-1} + \hat{\boldsymbol{n}}\left(\boldsymbol{I}_A^{-1}(\boldsymbol{r}_A \times \hat{\boldsymbol{n}})\right) \times \boldsymbol{r}_A + \hat{\boldsymbol{n}}\left(\boldsymbol{I}_B^{-1}(\boldsymbol{r}_B \times \hat{\boldsymbol{n}})\right) \times \boldsymbol{r}_B}$$

The actual impulse vector is simply $\boldsymbol{J} = j\hat{\boldsymbol{n}}$

This is applied to the objects as follows:

- Change in Linear momentum is directly equal to the impulse:

$$\Delta \boldsymbol{P} = \boldsymbol{J} \quad \Leftrightarrow \quad \Delta \boldsymbol{v} = \boldsymbol{J} m^{-1}$$

- Change in Angular momentum is equal to the imulsive torque ($\boldsymbol{\tau}_{\mathbf{IMPULSE}}$):

$$\Delta \boldsymbol{L} = (\boldsymbol{r} \times \boldsymbol{J}) \quad \Leftrightarrow \quad \Delta \boldsymbol{\omega} = \boldsymbol{I}^{-1}(\boldsymbol{r} \times \boldsymbol{J})$$