CPSC 427
Video Game Programming

Animation

Physics-Based Simulation

- Movement governed by forces
- Simple
  - *Independent particles*
- Complex
  - *Correct collisions, 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 etc…*

Examples

- **Particle systems**
  - ***Fire, water, smoke, pebbles***
- **Rigid-body simulation**
  - ***Blocks, robots, humans***
- **Continuum systems**
  - ***Deformable solids***
  - ***Fluids, cloth, hair***
- **Group movement**
  - ***Flocks, crowds***

Simulation Basics

Simulation loop…

1. ***Equations of Motion***
   - sum forces & torques
   - solve for accelerations: $\vec{F} = ma$
2. ***Numerical integration***
   - update positions, velocities
3. ***Collision detection***
4. ***Collision resolution***

Particles: Newtonian Physics as First-Order ODE

- **Motion of one particle**

Second-order ODE

$$\vec{F} = m\,\frac{\partial^2 x}{\partial t^2}$$

First-order ODE

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

- **Motion of many particles**

$$\frac{\partial}{\partial t}\begin{bmatrix}\overrightarrow{x_1}\\ \overrightarrow{v_1}\\ \overrightarrow{x_2}\\ \overrightarrow{v_2}\\ \vdots\\ \overrightarrow{x_n}\\ \overrightarrow{v_n}\end{bmatrix} = \begin{bmatrix}\overrightarrow{v_1}\\ \overrightarrow{F_1}/m_1\\ \overrightarrow{v_2}\\ \overrightarrow{F_2}/m_2\\ \vdots\\ \overrightarrow{v_n}\\ \overrightarrow{F_n}/m_n\end{bmatrix}$$

Basic Particle Simulation (first try)

Forces only $\vec{f} = ma$

$$d_t = t_{i+1} - t_i$$
$$\vec{v}_{i+1} = \vec{v}(t_i) + (\vec{f}(t_i)/m)d_t$$
$$\vec{p}_{i+1} = \vec{p}(t_i) + \vec{v}(t_{i+1})d_t$$

Basic Particle Simulation (first try)

Forces only $\vec{F} = ma$

$$d_t = t_{i+1} - t_i$$
$$\vec{v}_{i+1} = \vec{v}(t_i) + (\vec{f}(t_i)/m)d_t$$
$$\vec{p}_{i+1} = \vec{p}(t_i) + \vec{v}(t_{i+1})d_t$$

Basic Particle Forces

- **Gravity**

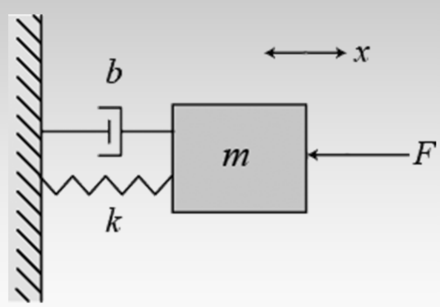
  $$F = \begin{bmatrix} 0 \\ -mg \end{bmatrix}$$

- **Viscous damping**

  $$F^{(i)} = -bv^{(i)}$$

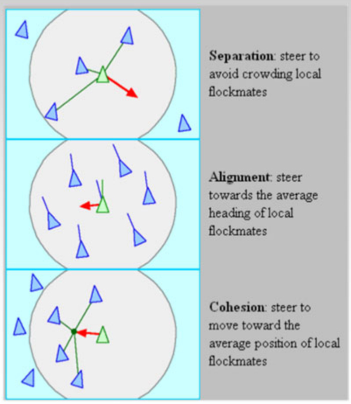
- **Spring & dampers**

  $$F = -kx - bv$$



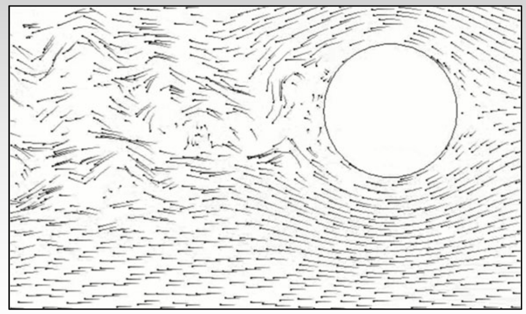
Proxy Forces

- **Behavior forces: flocking birds, schooling fish, etc.**
  ["Boids", Craig Reynolds, SIGGRAPH 1987]



Courtesy of Craig W. Reynolds. Used with permission.

- **Fluids**
  ["Curl Noise for Procedural Fluid Flow" R. Bridson, J. Hourihan, M. Nordenstam, Proc. SIGGRAPH 2007]



Basic Particle Simulation: Small Problem…

Forces only $\vec{F} = ma$

$$d_t = t_{i+1} - t_i$$
$$\vec{v}_{i+1} = \vec{v}(t_i) + (\vec{f}(t_i)/m)d_t$$
$$\vec{p}_{i+1} = \vec{p}(t_i) + \vec{v}(t_{i+1})d_t$$

Equations of motion describe state (equilibrium)

Use: get values at time t_{i+1} from values at time t_i

Newtonian Physics as First-Order ODE

- Motion of one particle

Second-order ODE

$$\vec{F} = m \frac{\partial^2 x}{\partial t^2}$$

First-order ODE

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

- Motion of many particles

$$\frac{\partial}{\partial t}\begin{bmatrix}\overrightarrow{x_1}\\ \overrightarrow{v_1}\\ \overrightarrow{x_2}\\ \overrightarrow{v_2}\\ \vdots\\ \overrightarrow{x_n}\\ \overrightarrow{v_n}\end{bmatrix} = \begin{bmatrix}\overrightarrow{v_1}\\ \overrightarrow{F_1}/m_1\\ \overrightarrow{v_2}\\ \overrightarrow{F_2}/m_2\\ \vdots\\ \overrightarrow{v_n}\\ \overrightarrow{F_n}/m_n\end{bmatrix}$$

Ordinary Differential Equations

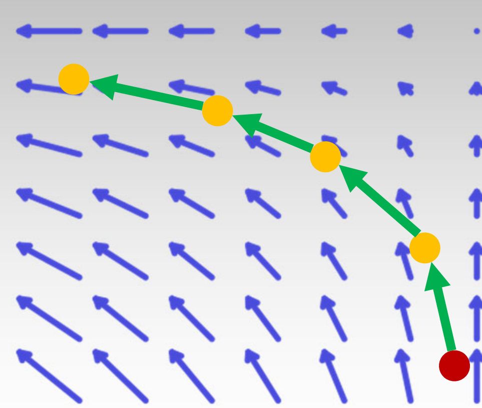
$$\frac{\partial}{\partial t}\vec{X}(t) = f(\vec{X}(t), t)$$

Given that $\vec{X}_0 = \vec{X}(t_0)$

Compute $\vec{X}(t)$ **for** $t > t_0$

$$\Delta\vec{X}(t) = f(\vec{X}(t), t)\Delta t$$

- **Simulation:**
  - ***path through state-space***
  - ***driven by vector field***



ODE Numerical Integration: Explicit (Forward) Euler

$$\frac{\partial}{\partial t}\vec{X}(t) = f(\vec{X}(t), t)$$

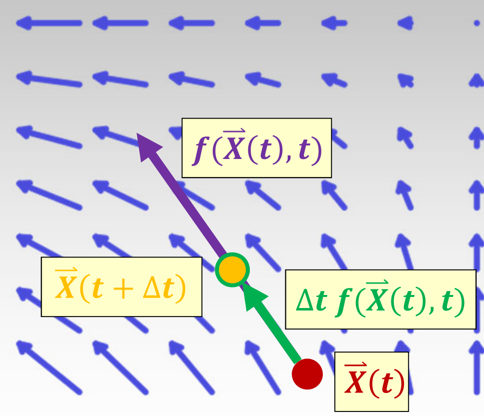
Given that $\vec{X}_0 = \vec{X}(t_0)$

Compute $\vec{X}(t)$ **for** $\mathrm{t} > \mathrm{t}_0$

$$\mathrm{t}_1 = \mathrm{t}_0 + \Delta t$$

$$\Delta\vec{X}(t) = \Delta t\, f(\vec{X}(t), t)$$

$$\vec{X}_1 = \vec{X}_0 + \Delta t\, f(\vec{X}_0, \mathrm{t}_0)$$



ODE Numerical Integration: Explicit (Forward) Euler

$$\frac{\partial}{\partial t}\vec{X}(t) = f(\vec{X}(t), t)$$

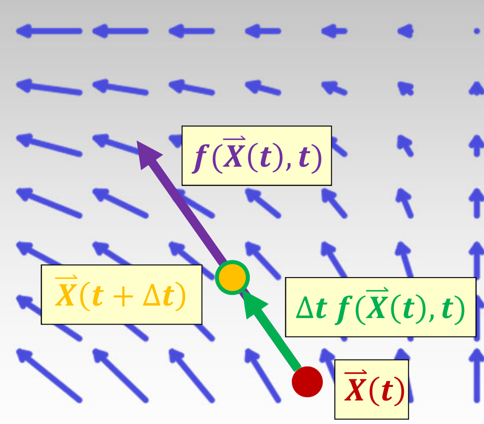
Given that $\vec{X}_0 = \vec{X}(t_0)$

Compute $\vec{X}(\mathrm{t}_i)$ **for** $\mathrm{t}_i > \mathrm{t}_0$

$$\Delta t = \mathrm{t}_i - \mathrm{t}_{i-1}$$

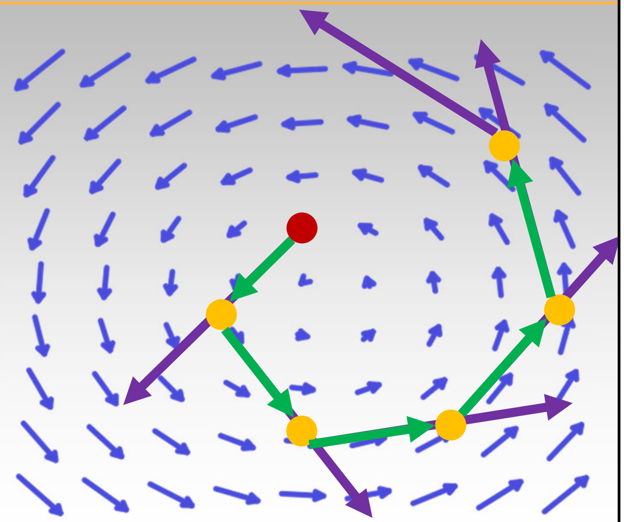
$$\Delta\vec{X}(\mathrm{t}_{i-1}) = \Delta t\, f(\vec{X}(\mathrm{t}_{i-1}), \mathrm{t}_{i-1})$$

$$\vec{X}_i = \vec{X}_{i-1} + \Delta t\, f(\vec{X}_{i-1}, \mathrm{t}_{i-1})$$



Explicit Euler Problems

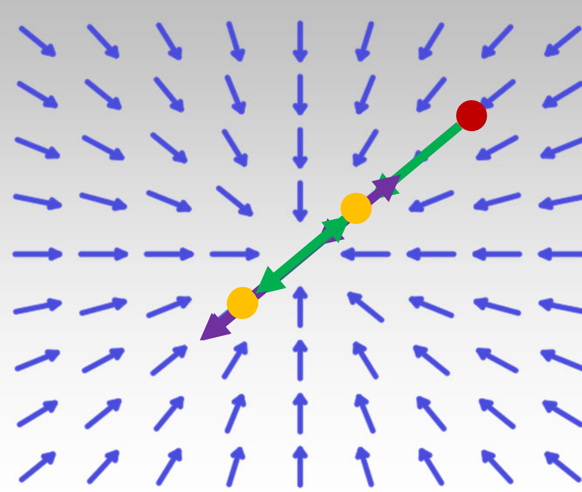
- Solution **spirals** out
  - *Even with small time steps*
  - *Although smaller time steps are still better*



© Alla Sheffer/M. van de Panne

Explicit Euler Problems

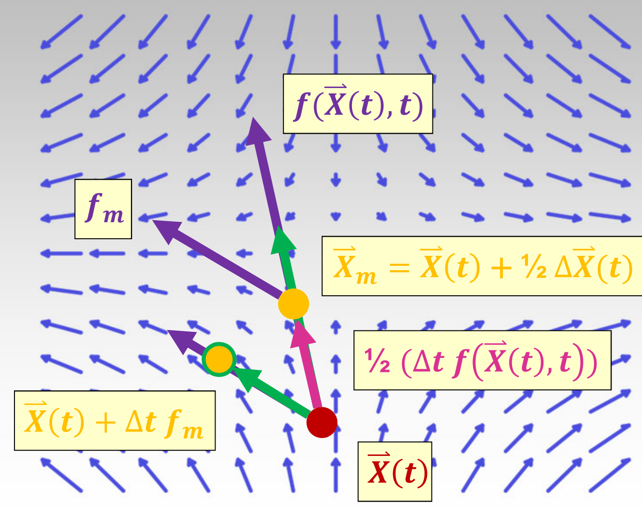
- Can lead to **instabilities**



© Alla Sheffer/M. van de Panne

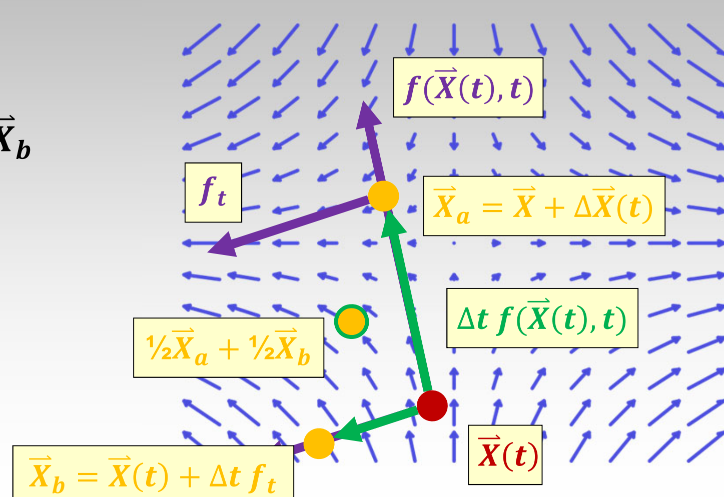
Midpoint Method

1. ½ Euler step
2. evaluate f_m at $\vec{X}_m$
3. full step using f_m



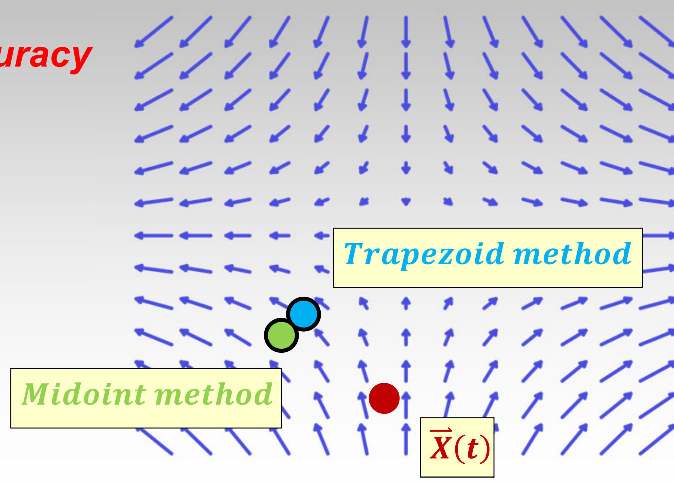
Trapezoid Method

1. full Euler step get $\vec{X}_a$
2. evaluate f_t at $\vec{X}_a$
3. full step using f_t get $\vec{X}_b$
4. average $\vec{X}_a$ and $\vec{X}_b$

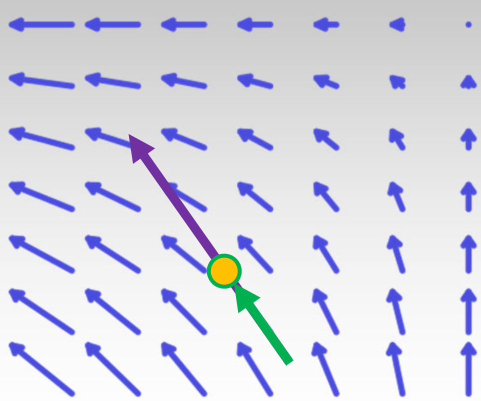


Midpoint & Trapezoid Method

- Not exactly the same
  - *But same order of accuracy*

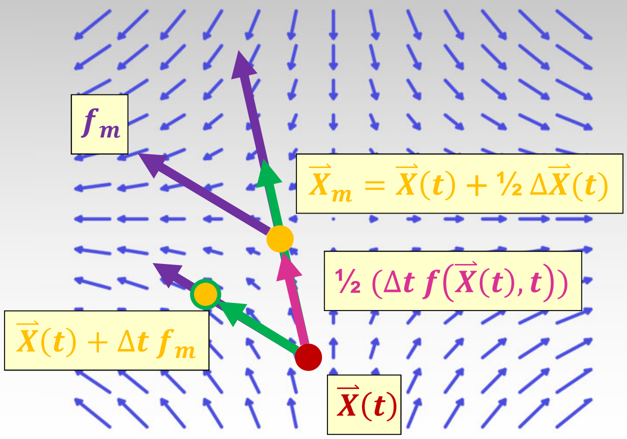


Explicit Euler: Code



```
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)
}
```

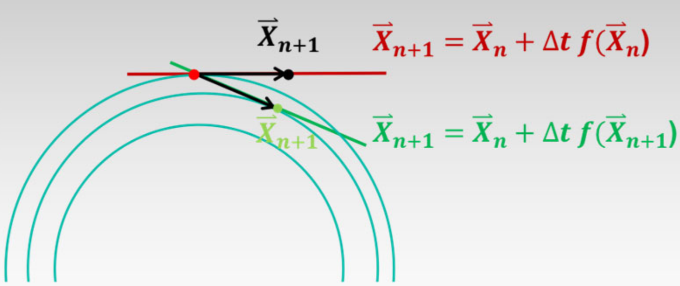
Midpoint Method: Code



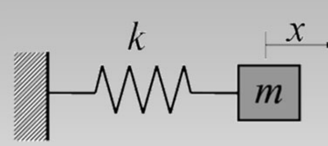

```
void takeStep(ParticleSystem* ps, float h)
{
    velocities = ps->getStateVelocities()
    positions = ps->getStatePositions()
    forces = ps->getForces(positions, velocities)
    masses = ps->getMasses()
    accelerations = forces / masses
    midPositions = positions + 0.5*h*velocities
    midVelocities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


Implicit (Backward) Euler:

- **Use the derivative at the destination**



- **Problem is we don't know the destination yet**



Forward Euler

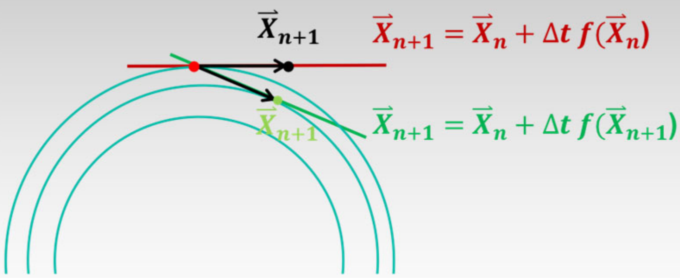
$$x_{n+1} = x_n + h\, v_n$$
$$v_{n+1} = v_n + h\left(\frac{-k\, x_n}{m}\right)$$

Backward Euler

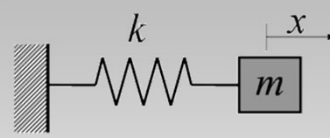
$$x_{n+1} = x_n + h\, v_{n+1}$$
$$v_{n+1} = v_n + h\left(\frac{-k\, x_{n+1}}{m}\right)$$

Implicit (Backward) Euler: Solve a (linear) system

- Use the derivative at the destination



- Problem is we don't know the destination yet



Forward Euler

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

Backward Euler

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

Simulation Basics

Simulation loop…

1. *Equations of Motion*
2. *Numerical integration*
3. *Collision detection*
4. *Collision resolution*

Collisions

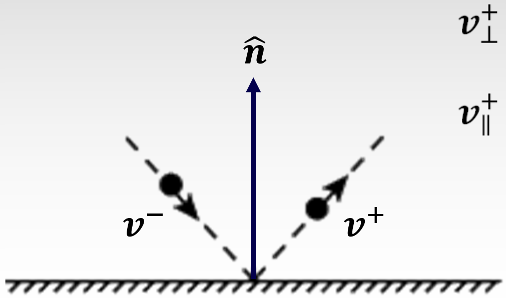
- Collision **detection**
  - *Broad phase: AABBs, bounding spheres*
  - *Narrow phase: detailed checks*
- Collision **response**
  - *Collision impulses*
  - *Constraint forces: resting, sliding, hinges, ….*

Particle-Plane Collisions

- Particle-plane **frictionless impulse response**
  - ***Invert* & *scale* perpendicular velocity component**

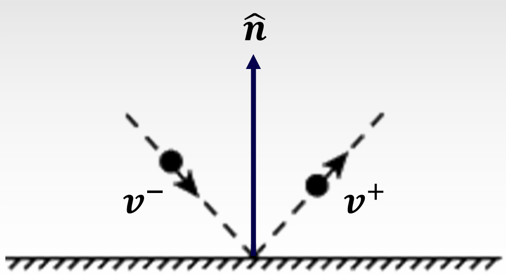
$$v_{\perp}^{+} = -\epsilon v_{\perp}^{-}$$

$$v_{\parallel}^{+} = v_{\parallel}^{-}$$



Particle-Plane Collisions

- **More formally…**
  - ***Apply an impulse of magnitude j***
    - Inversely proportional to mass of particle
  - ***In direction of normal***



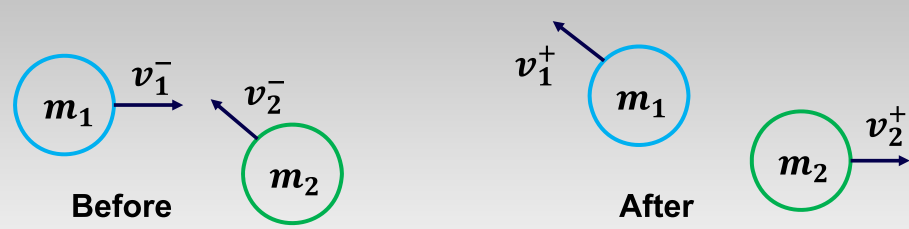
$$j = (1 + \epsilon)m$$

$$\vec{j} = j\,\hat{n}$$

$$v^+ = \frac{\vec{j}}{m} + v^-$$

Particle-Particle Collisions

- **Particle-particle frictionless elastic impulse response**



- **Momentum is preserved**

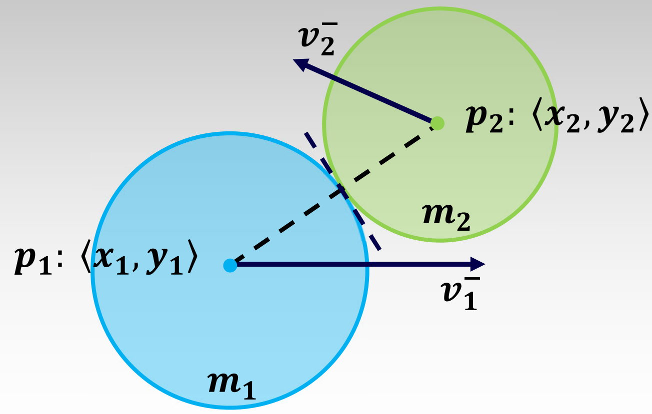
$$m_1 v_1^- + m_2 v_2^- = m_1 v_1^+ + m_2 v_2^+$$

- **Kinetic energy is preserved**

$$½\, m_1 {v_1^-}^2 + ½\, m_2 {v_2^-}^2 = ½\, m_1 {v_1^+}^2 + ½\, m_2 {v_2^+}^2$$

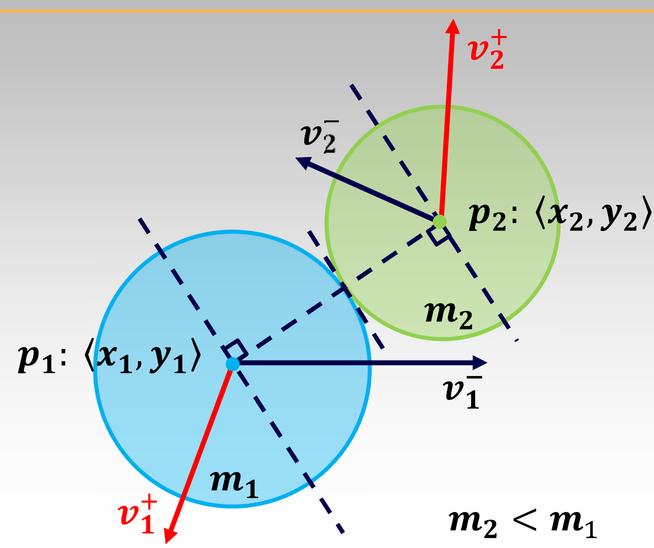
Particle-Particle Collisions (radius >0)

- What we know…
  - *Particle centers*
  - *Initial velocities*
  - *Particle Masses*
- What we can calculate…
  - *Contact normal*
  - *Contact tangent*



Particle-Particle Collisions (radius >0)

- Impulse direction reflected across tangent
- Impulse magnitude proportional to mass of other particle



Particle-Particle Collisions (radius >0)

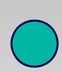
- More formally…

$$v_1^+ = v_1^- - \frac{2m_2}{m_1 + m_2} \frac{\langle v_1^- - v_2^- \rangle \cdot \langle p_1 - p_2 \rangle}{\|p_1 - p_2\|^2} \langle p_1 - p_2 \rangle$$

$$v_2^+ = v_2^- - \frac{2m_1}{m_1 + m_2} \frac{\langle v_2^- - v_1^- \rangle \cdot \langle p_2 - p_1 \rangle}{\|p_2 - p_1\|^2} \langle p_2 - p_1 \rangle$$

Rigid Body Dynamics

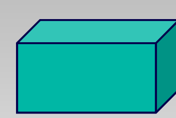
- From particles to rigid bodies…



Particle

$$state = \begin{cases} \vec{x}\ position \\ \vec{v}\ velocity \end{cases}$$

$\mathbb{R}^4$ in 2D
$\mathbb{R}^6$ in 3D



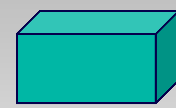
Rigid body

$$state = \begin{cases} \vec{x}\ position \\ \vec{v}\ velocity \\ q, R\ rotation\ matrix\ 3x3 \\ \vec{w}\ angular\ velocity \end{cases}$$

$\mathbb{R}^{12}$ in 3D

Rigid Body Dynamics

- From particles to rigid bodies…



Newton's equations of motion

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

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

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

Newton-Euler equations of motion

$$\begin{bmatrix} m & & & \\ & m & & \\ & & m & \\ & & & \mathbf{I} \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_z \\ w_x \\ w_y \\ w_z \end{bmatrix} = \begin{bmatrix} \Sigma\vec{F} \\ \Sigma\vec{\tau} - \vec{w} \times \mathbf{I}\vec{w} \end{bmatrix}$$

Inertia tensor

$\Sigma\vec{\tau} - \vec{w} \times \mathbf{I}\vec{w}$

Resources

- 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
- Interactive simulation of rigid body dynamics in computer graphics, CGF 2014 http://onlinelibrary.wiley.com/doi/10.1111/cgf.12272/abstract
- A Mathematical Introduction to Robotic Manipulation (textbook) http://www.cds.caltech.edu/~murray/books/MLS/pdf/mls94-complete.pdf
- Particle-based Fluid Simulation for Interactive Applications, SCA 2003, PDF

© Alla Sheffer/M. van de Panne