Simulation & Animation

5. Physically-based Animation

Reinhold Preiner
SS 2025

Physically-based animation

Goal: compute the motion of objects based on the underlying laws of physics.

Motivation:

1. **Realistic environments**
   A realistic virtual environment requires not only accurate rendering, but also realistic physically-based animations
2. **Interactive environments**
   Arbitrary interactions require dynamic real-time simulations (we cannot precompute all possible outcomes)
3. **More productive animation pipeline**
   Artists only have to specify high-level characteristics (mass, forces, initial conditions)

Overview

The key components of a physically-based animation system are:

1. **ODE* Solvers**
2. **Particle dynamics**
3. **Rigid-body dynamics**
4. **Collision detection and response**

***Ordinary Differential Equation**

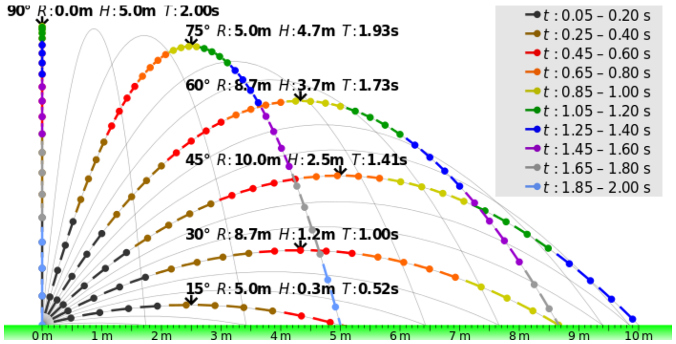
1. Ordinary Differential Equations

Position Functions

- So far: position functions defining the motion of objects along smooth parametric curves (polynomials, splines)

- Simple physics-based example: parabolic motion of ballistic body

$$y = y_0 + x \tan\theta - \frac{gx^2}{2(v\cos\theta)^2}$$



- In many physical systems, an object's path is influenced based on complex interactions with the environment
→ cannot be expressed by an analytic position function x(t).

- However, we can define the **change of position x'(t)** depending on these interactions.
→ also depends on the **current position x** itself

Ordinary Differential Equations

General form:

$$\frac{d}{dt}x(t) = f(x,t) \qquad \textbf{OR} \qquad \dot{x}(t) = f(x,t)$$

Where:

- $\boldsymbol{x}$ *is the state of a system, typically a vector that changes over time*
- $\boldsymbol{f}$ *is a known function that we can evaluate*

In ***an initial value problem*** we are given the state of the system at beginning: $\mathbf{x}(t_0) = \mathbf{x}_0$

Equations of Motion

If x(t) represents the position of an object, then we define:

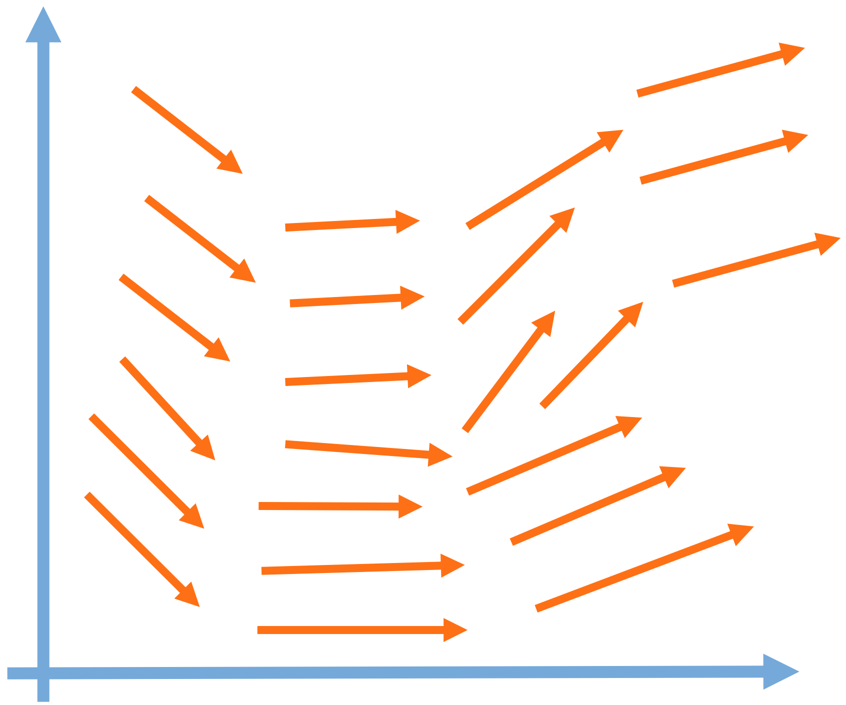
Velocity: $$\vec{v}(t) = \frac{d\vec{x}(t)}{dt}$$ 1st order ODE

Acceleration: $$\vec{a}(t) = \frac{d\vec{v}(t)}{dt} = \frac{d^2\vec{x}(t)}{dt^2}$$ 2nd order ODE

Our discussion here regards 1st order ODEs. We will see later how we can handle the 2nd order equation for the acceleration.

Ordinary Differential Equations

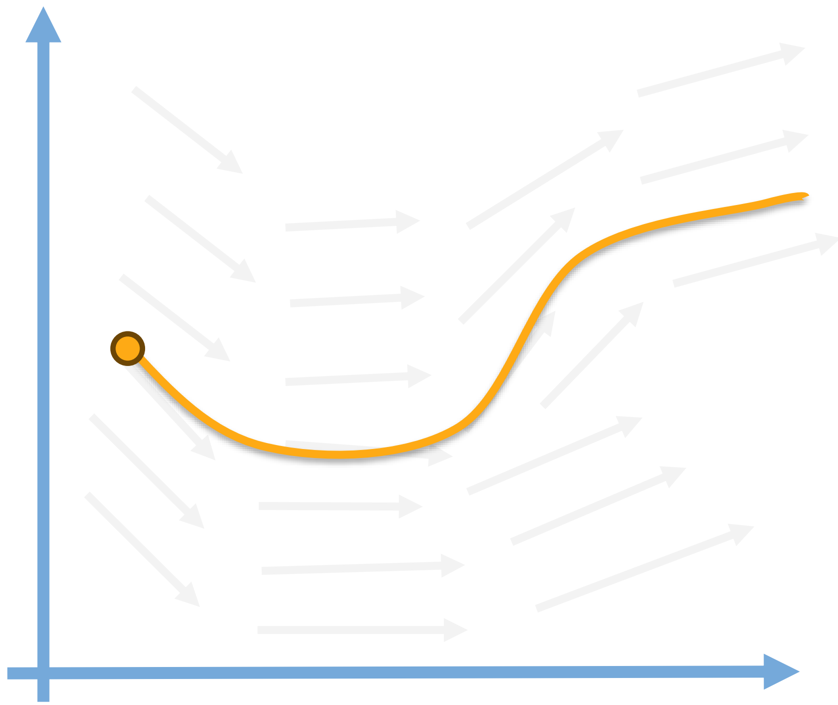
$f(x,t)$ defines a vector field corresponding to the velocity of a moving point **p** at every possible position **x** and time **t**.



$$\boxed{\frac{d}{dt}x(t) = f(x,t)}$$

Ordinary Differential Equations

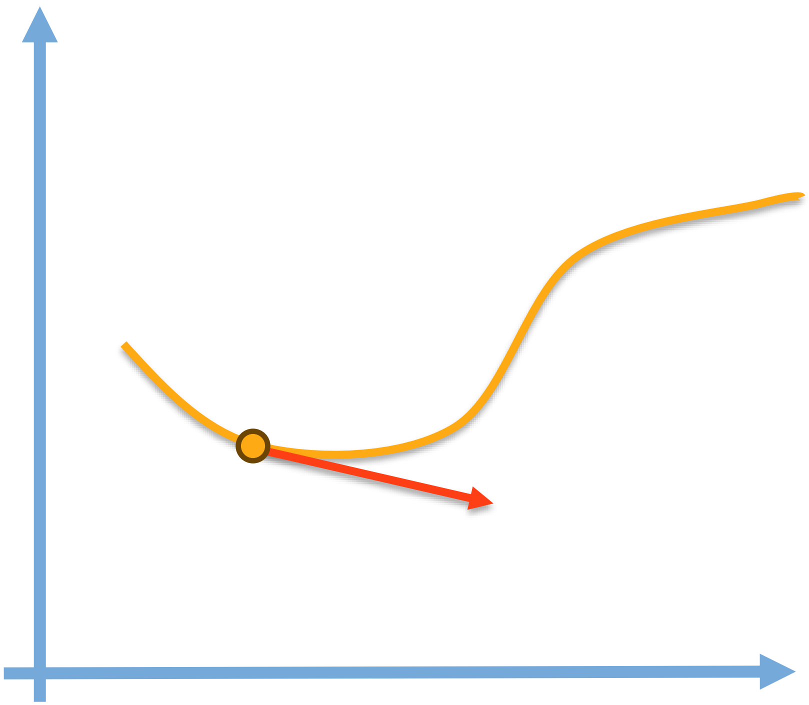
Starting at a point x_o, integrating $x(t)$ sweeps out a curve that describes the motion of a point **p** in the plane.



$$\boxed{\frac{d}{dt}x(t) = f(x,t)}$$

Ordinary Differential Equations

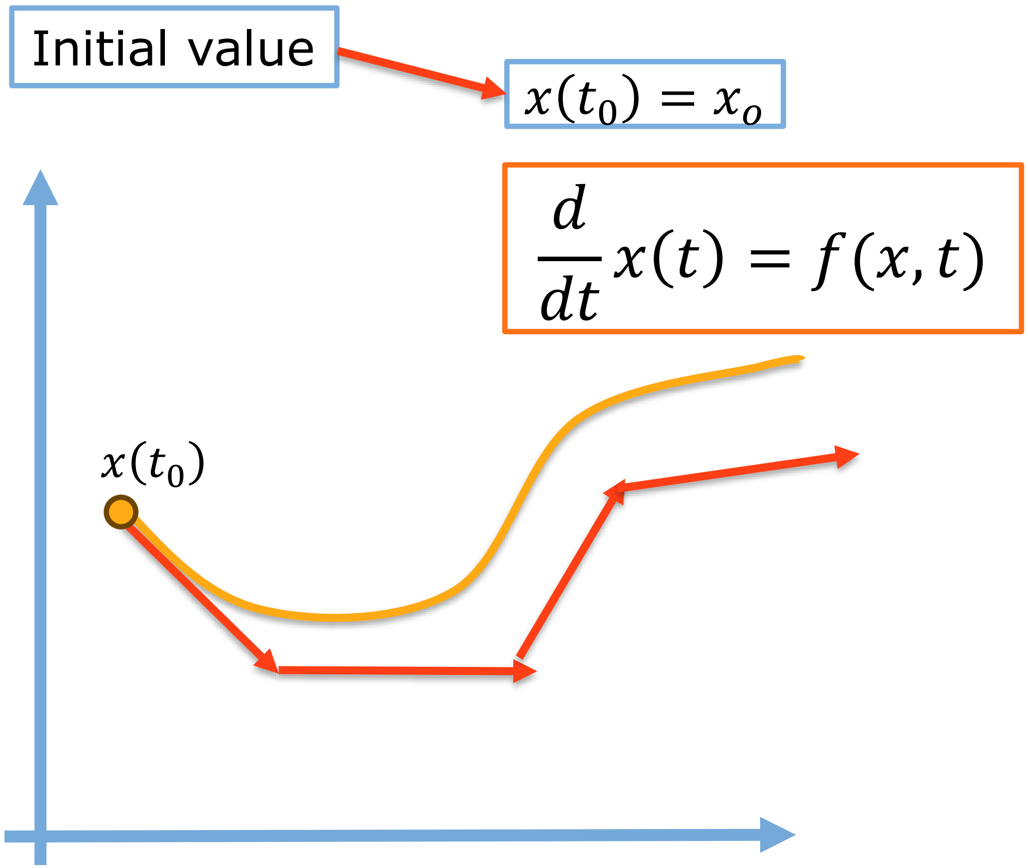
For a single position **x** and time **t,** f(x,t) defines the velocity of of point **p** at that time, which is **tangent** to this curve.



$$\frac{d}{dt}x(t) = f(x,t)$$

Initial value problem

Given a starting point, follow the trajectory by doing multiple **steps**.



Taylor Series

Assuming x is smooth, we can express its value at the end of the step as an infinite sum:

$$\mathbf{x}(t_0 + h) = \mathbf{x}(t_0) + h\dot{\mathbf{x}}(t_0) + \frac{h^2}{2!}\ddot{\mathbf{x}}(t_0) + \frac{h^3}{3!}\dddot{\mathbf{x}}(t_0) + \ldots + \frac{h^n}{n!}\frac{\partial^n \mathbf{x}}{\partial t^n} + \ldots$$

Taylor Series

Assuming x is smooth, we can express its value at the end of the step as an infinite sum:

$$\mathbf{x}(t_0 + h) = \mathbf{x}(t_0) + h\dot{\mathbf{x}}(t_0) + \cancel{\frac{h^2}{2!}\ddot{\mathbf{x}}(t_0) + \frac{h^{3\ldots}}{3!}\mathbf{x}(t_0) + \ldots + \frac{h^n}{n!}\frac{\partial^n \mathbf{x}}{\partial t^n}} + \ldots$$

$$\mathbf{x}(t_0 + h) = \mathbf{x}(t_0) + h\dot{\mathbf{x}}(t_0)$$

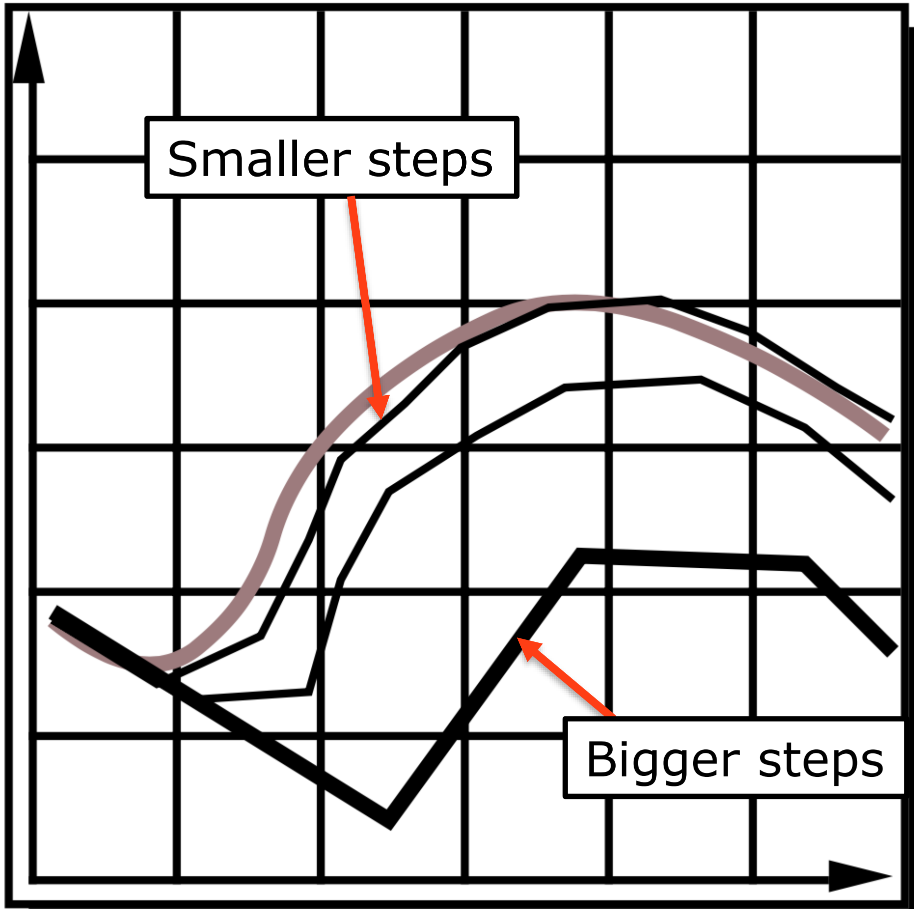
If we **truncate** the Taylor series by assuming that all derivatives except the first one are zero, we get **Euler's method**.

Euler's Method

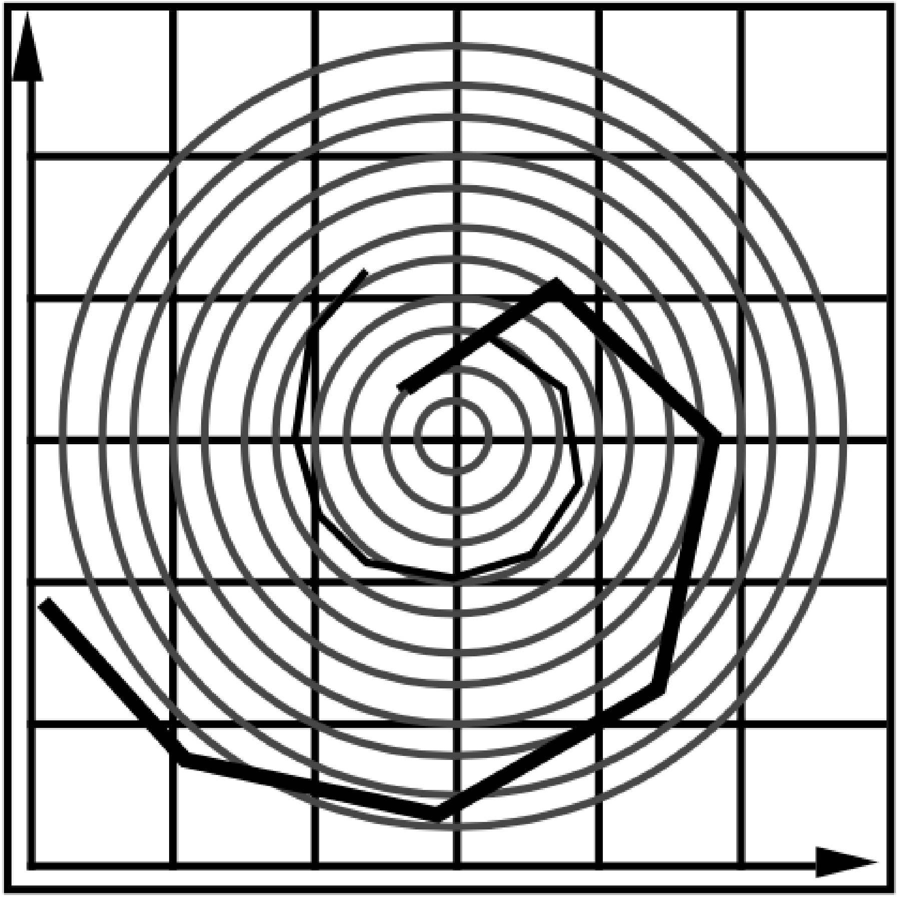
$$\mathbf{x}(t_0 + h) = \mathbf{x}(t_0) + h\dot{\mathbf{x}}(t_0)$$

- Simplest numerical solution method
- Discrete time steps
- Bigger steps, bigger errors

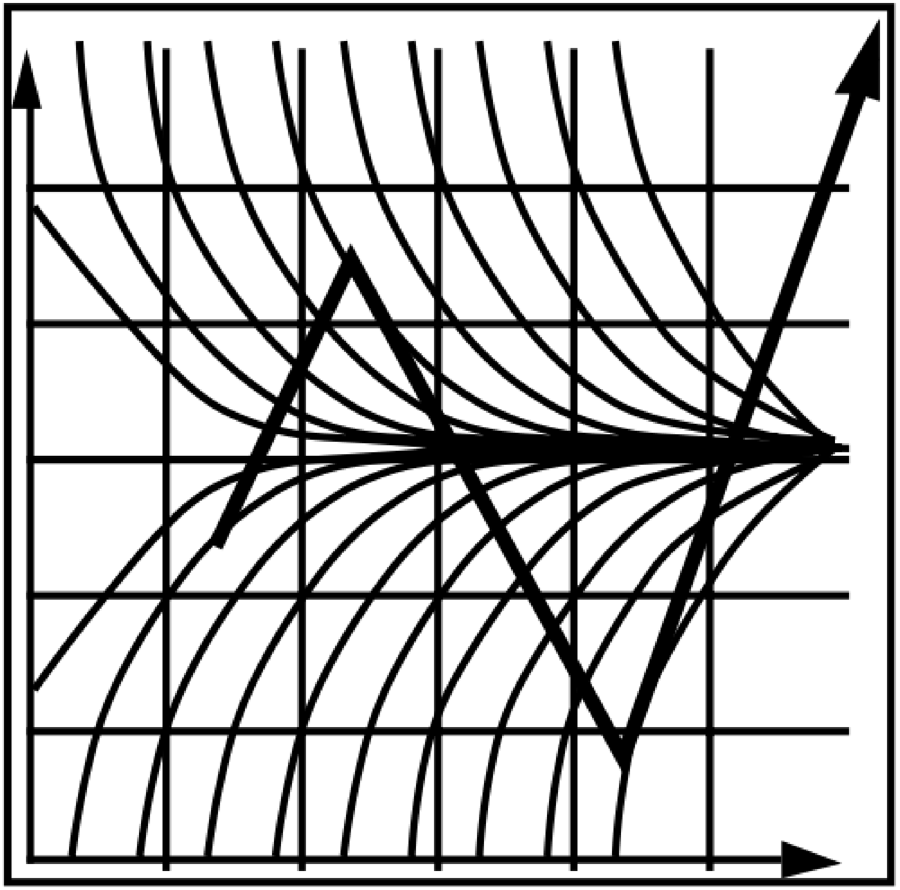
Only correct when x is linear, otherwise error is introduced!



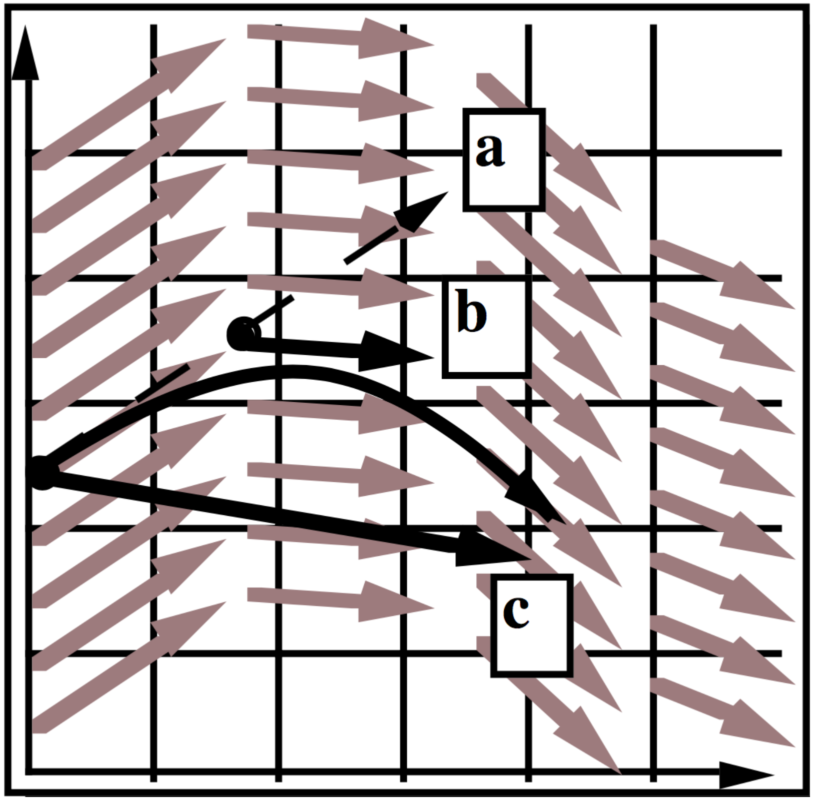
Problems: Inaccuracy



Problems: Instability



Midpoint Method



a. Evaluate f at the initial point:

$$f(\mathbf{x}_0)$$

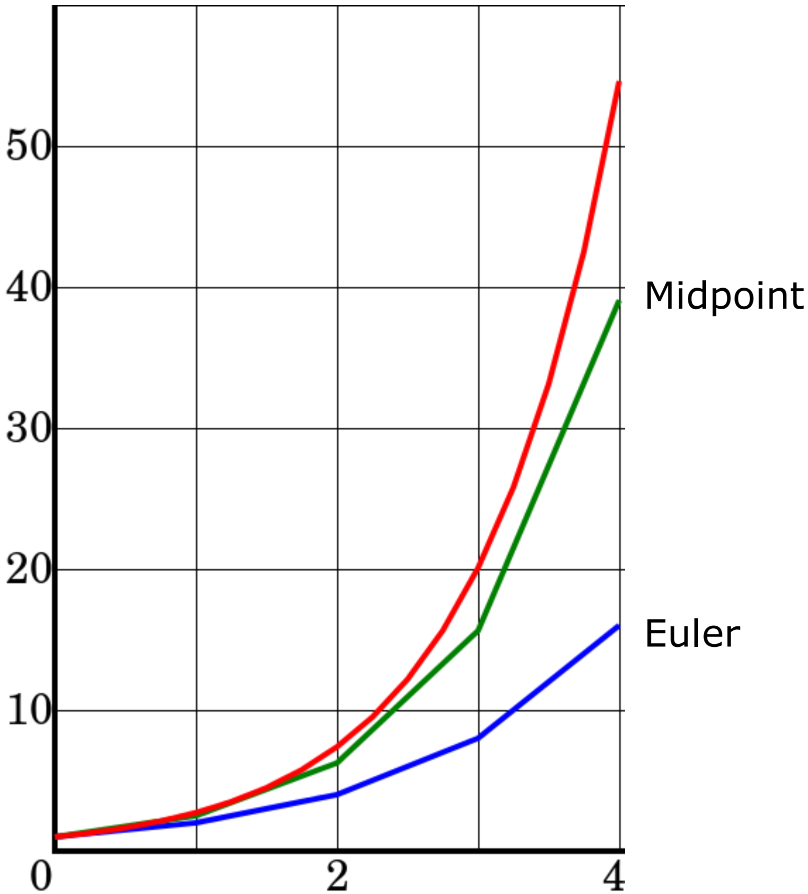
b. Evaluate f at the midpoint:

$$f\left(\mathbf{x}_0 + \frac{h}{2} f(\mathbf{x}_0)\right)$$

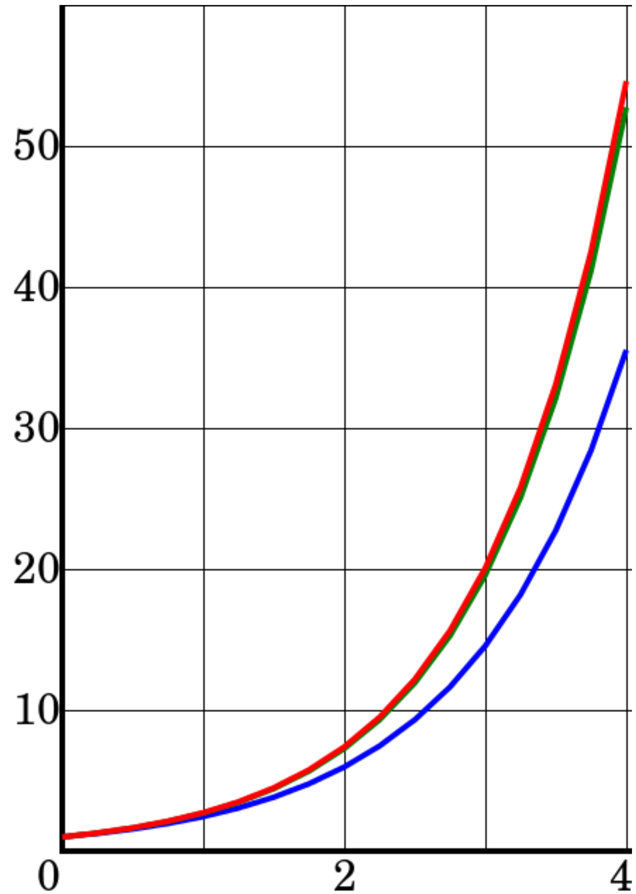
c. Take a step using the midpoint value:

$$\mathbf{x}(t_0 + h) = \mathbf{x}(t_0) + h\left(f\left(\mathbf{x}_0 + \frac{h}{2} f(\mathbf{x}_0)\right)\right).$$

Euler vs. Midpoint



$h = 1.0$



$h = 0.25$

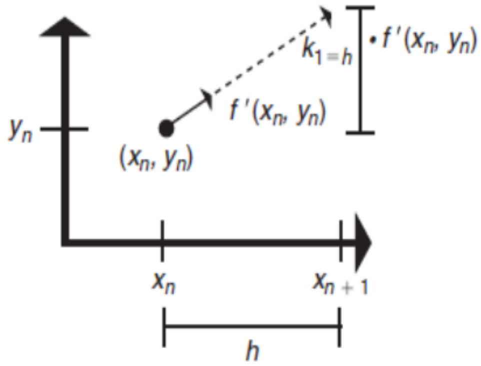
Higher-order methods

- Euler's method performs one function evaluation
- Midpoint performs two evaluations
- 4th-order **Runge-Kutta** performs four function evaluations:

$$
\begin{aligned}
k_1 &= hf(\mathbf{x}_0, t_0) \\
k_2 &= hf(\mathbf{x}_0 + \frac{k_1}{2}, t_0 + \frac{h}{2}) \\
k_3 &= hf(\mathbf{x}_0 + \frac{k_2}{2}, t_0 + \frac{h}{2}) \\
k_4 &= hf(\mathbf{x}_0 + k_3, t_0 + h) \\
\mathbf{x}(t_0 + h) &= \mathbf{x}_0 + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4.
\end{aligned}
$$

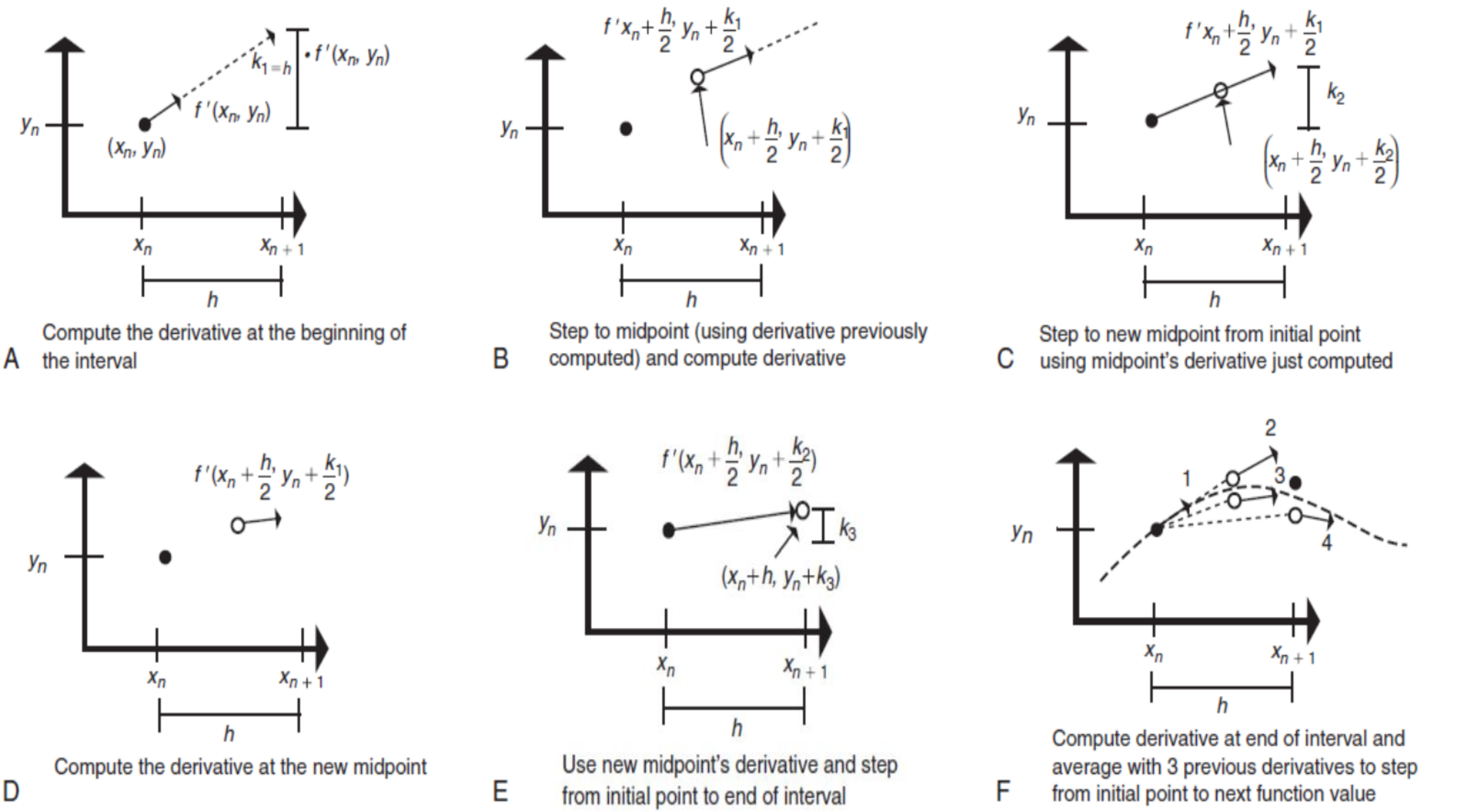
Higher accuracy, but also higher complexity

4th-order RK steps



A Compute the derivative at the beginning of the interval

4th-order RK steps



Adaptive step size

- Large step sizes: better performance but lower accuracy
- Small step sizes: Better accuracy but lower performance

Ideally we want the largest possible step size that does not introduce an unreasonable amount of error.

Determining such a good step size can be a problem, no matter which underlying ODE solver we are using.

Adaptive methods vary the step size over the course of solving an ODE: smaller steps are used in non-linear (curvy) segments.

Adaptive Euler method

- Compute two estimates for $x(t_0 + h)$
- For the first estimate x_a, use one step of size h
- For the 2nd estimate x_b, use two steps of size $h/2$
- The current step size h is adjusted based on the error value:

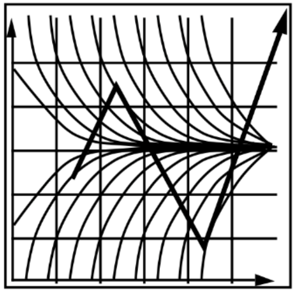
$$e = |x_a - x_b|$$

For linear segments, e will be close to zero, and larger step sizes can be used.

Step Sizes in Interactive Simulations

- Interactive simulations, such as video games, render the world at a specific frame rate
  - Typically locked to the monitor refresh rate (vsync)
  - But can often be lower, due to limited performance.
- A naïve approach is to perform the physics simulation at the same rate as the visual refresh
  - The step size of the simulation is the interval between two subsequent frames displayed on the screen.

Problem: Sudden drops in the framerate can result in unstable physics simulation!!!!



Step sizes in interactive simulations

- The update rate of the rendering and physics simulation should be decoupled
- Video games often sample the input and update the physics at a fixed rate, which is higher than the display refresh rate.

General Solver Interface

In a C-like language, an ODE solver will typically have this interface:

```
typedef void (*dydt_func)(double t, double y[], double ydot[]);

void    ode(double y0[], double yend[], int len, double t0,
            double t1, dydt_func dydt);
```

Abstracts the underlying implementation, could be Euler's method, midpoint or RK.

General Solver Interface

In a C-like language, an ODE solver will typically have this interface:

```
typedef void (*dydt_func)(double t, double y[], double ydot[]);

void    ode(double y0[], double yend[], int len, double t0,
            double t1, dydt_func dydt);
```

Input: Initial state at time t0

Output: End state at time t1

Abstracts the underlying implementation, could be Euler's method, midpoint or RK.

General Solver Interface

In a C-like language, an ODE solver will typically have this interface:

```
typedef void (*dydt_func)(double t, double y[], double ydot[]);

void    ode(double y0[], double yend[], int len, double t0,
            double t1, dydt_func dydt);
```

Helper function to compute the derivatives of the state
(function pointer in ANSI C terms)

Abstracts the underlying implementation, could be Euler's method, midpoint or RK.

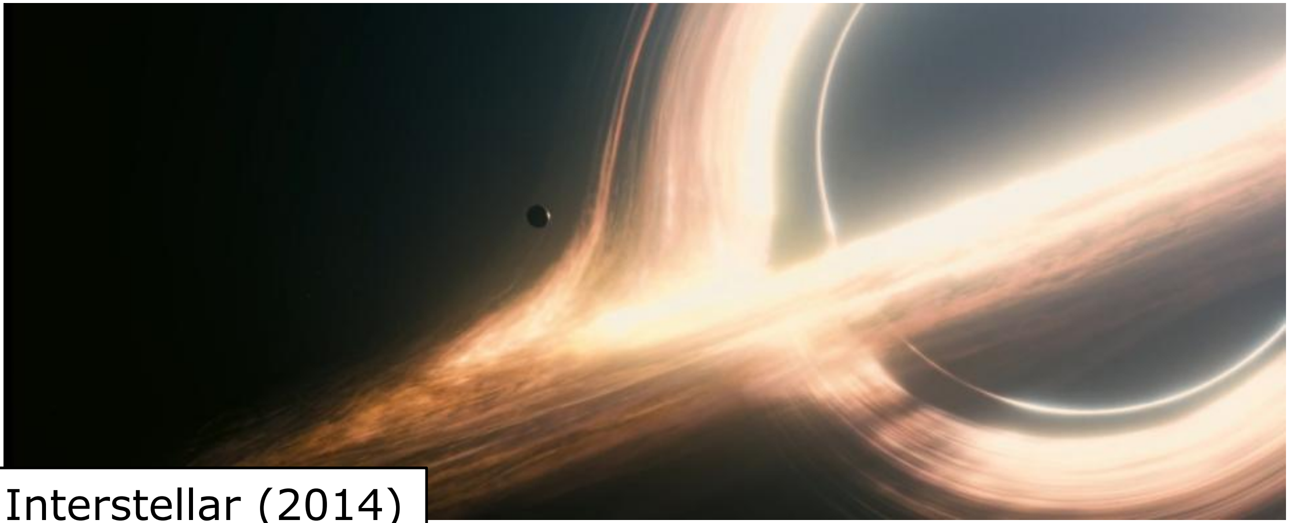
2. Particle Dynamics

Newtonian Physics

- **First law:** an object either remains at rest or continues to move at a constant velocity
- **Second law:** the sum of the forces F on an object is equal to its mass m multiplied by the acceleration a of the object: **F = m*a**
  (accurate for a **particle of mass**, integration is required for arbitrary mass distributions/solid objects)
- **Third law:** When one body exerts a force on a second body, the second body simultaneously exerts a force equal in magnitude and opposite in direction on the first body.

Newtonian Physics

- Empirical laws, based on observation
- Not accurate for objects that move very fast or have a very large mass
- Good enough for most simulations
  (unless the simulation involves black holes…)



Interstellar (2014)

Newtonian Particle

- The motion of a **particle of mass** is governed by this differential equation:

$$\frac{d^2x(t)}{dt^2} = \frac{F(x,t)}{m}$$

  where the total force $F(x,t)$ can change depending on the position of the particle and the time.

- This equation has a second-order derivative and differs from the equations that we have seen in the previous slides about ODEs.

Phase Space

- To handle the second-order ODE, we convert it to a first-order one by introducing extra variables

$$\left.\begin{aligned}\frac{dx(t)}{dt} &= v \\ \frac{dv(t)}{dt} &= F/m\end{aligned}\right\}$$

Coupled first-order ODEs
(our solvers work for these!)

Phase Space

We concatenate the 3D position and 3D velocity vectors to make a new 6D vector that denotes the state of the particle in ***phase-space***.

$$\begin{bmatrix} x(t) \\ v(t) \end{bmatrix}$$

State in 6D phase space

$$\frac{d}{dt}\begin{bmatrix} x(t) \\ v(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ F/m \end{bmatrix}$$

A standard 1st order ODE

Particle Data Structure

$x(t)$
$v(t)$
F
m

State of the particle in 6-dimensional phase space

Total accumulated force (gravity, wind - can vary with time and position)

Mass of the particle (can vary with time)

Particle Dynamics

Given the state of a particle at time t0, the ODE solver will compute the state at time t1.

$x(t)$
$v(t)$
F
m

State of the particle provided as input (y0)

Required for derivative evaluation

```
void    ode(double y0[], double yend[], int len, double t0,
            double t1, dydt_func dydt);
```

For a single particle, len = 6

Particle Systems

Given the state of N particles at time t0, the ODE solver will compute the state at time t1.

$x_0(t)$	$x_1(t)$	...	$x_\mathrm{N}(t)$
$v_0(t)$	$v_1(t)$	...	$v_\mathrm{N}(t)$
F_0	F_1	...	F_N
m_0	m_1	...	m_N

State of the particle provided as input (y0)

Required for derivative evaluation

```
void    ode(double y0[], double yend[], int len, double t0,
                double t1, dydt_func dydt);
```

For **N** particles, **len = 6N**

Derivative Evaluation

1. Zero forces
   - Loop over all particles and zero the accumulators
2. Accumulate forces
   - For each particle sum all forces
3. Construct the derivative vector
   - For each particle copy velocity **v** and **F/m** into the derivative vector

Forces

- **Constant:** the thrust of a rocket engine
- **Position dependent:** gravity, magnetic fields, other force fields
- **Velocity dependent:** drag
- **n-ary:** particles connected with springs

The forces should be recomputed and accumulated on each solver step.

Viscous Drag

Viscous force opposing the direction of motion of an object in a medium, with strength proportional to the speed (velocity magnitude):

$$F_{drag} = -K_{drag}\, v$$

where K_{drag} depends on density of the medium (zero for vacuum, higher for viscous fluids)

Gravity

The magnitude of the gravitational force from earth on an object with mass m at height r is given by:

$$F_{earth} = mg, \qquad g = -G\frac{m_{earth}}{r^2}$$

If the height of the object does not vary a lot during the motion, then g can be considered a constant.

Special Cases – Free Falling Particle

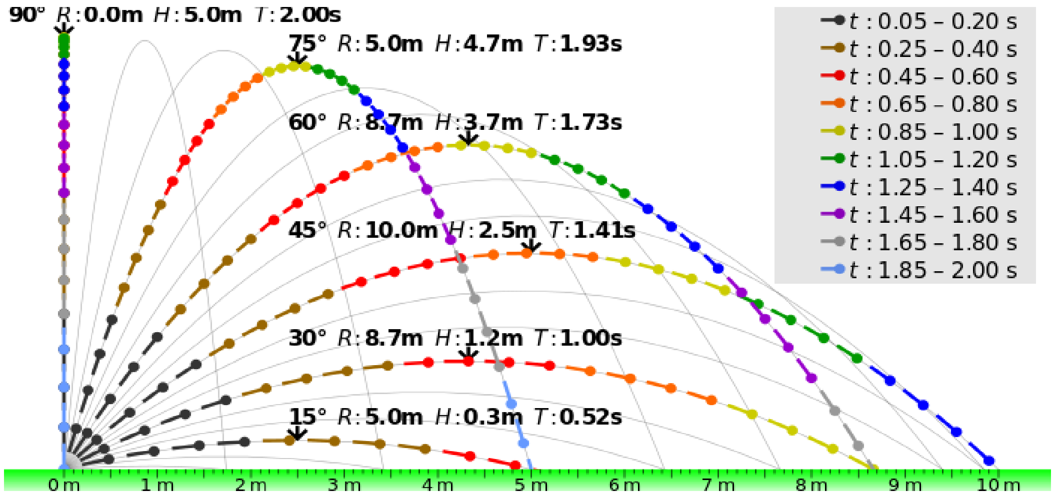
- If we assume only a constant gravitational force F=g*m acts on the particle, then the position x(t) is given by

$$x(t) = x(t_o) + v_o t + 0.5 g t^2$$

Where g is the acceleration from the gravitational force, assumed here constant.

Closed-form solution, but not 100% accurate, as the gravitational force depends on the height of the particle.
For accurate results the ODE should be solved.

Special Cases – Ballistics Trajectories



Closed-form formula:

$$y = y_0 + x \tan\theta - \frac{gx^2}{2(v\cos\theta)^2}.$$

Trajectories of projectiles launched at different elevation angles at the same initial speed in a **vacuum (no drag)** and **uniform downward gravity field.**

In the general case the gravitational field is not constant, various additional time-varying forces act upon the projectile and the distribution of mass in the projectile is not symmetric.
For accurate results in such cases, we need to properly solve the ODE.

Spring Forces

- Applied in n-ary “mass-spring” systems
- Guided by Hooke’s Law:

$$F_{spring} = -k\left(\left\|x_i - x_j\right\| - r_{ij}\right)\frac{x_i - x_j}{\left\|x_i - x_j\right\|}$$

Restoring Force vector of a spring

- Total potential energy:
  - x_i …… Position of the i-th mass vertex
  - r_{ij} …… Rest length of spring
  - k …… Stiffness factor

Velocity Verlet Integration

- Specific flavour of class Verlet methods
- 2nd-order approximation of the Taylor series
- Store acceleration a(t) in the state vector

- Current state: $\mathrm{x(t), v(t), a(t)}$

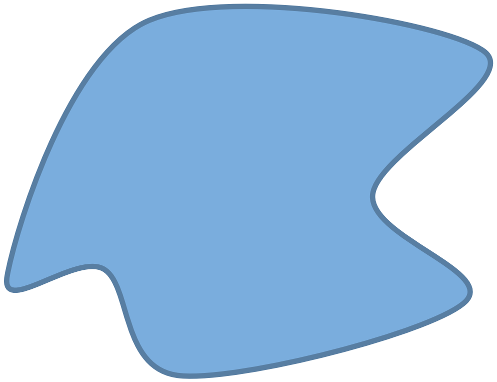
- 1. $\vec{x}(t+\Delta t) = \vec{x}(t) + \vec{v}(t)\,\Delta t + \frac{1}{2}\,\vec{a}(t)\,\Delta t^2$
- 2. Compute $\vec{a}(t+\Delta t)$ from forces at $\vec{x}(t+\Delta t)$
- 3. $\vec{v}(t+\Delta t) = \vec{v}(t) + \frac{1}{2}\left(\vec{a}(t) + \vec{a}(t+\Delta t)\right)\Delta t$

- New state: $\mathrm{x(t{+}\Delta t), v(t{+}\Delta t), a(t{+}\Delta t)}$

3. Rigid Body Dynamics

Rigid Bodies

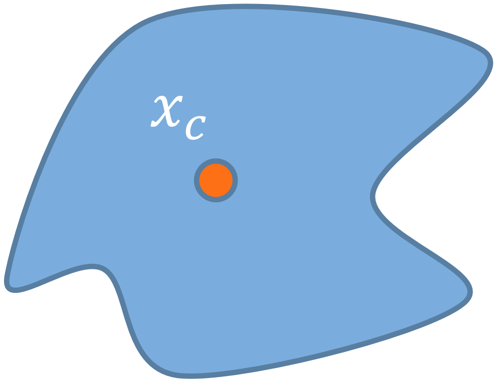
In the general case, an object (body of mass) has a non-uniform mass distribution.



In our physically-based simulations we will assume objects are **rigid**: they can be only rotated and translated – cannot be deformed.

Center of mass

If our object has a mass distribution with density $\rho(x)$ within a solid Q, then we define as center of mass the point x_c that satisfies the following equation:



$$\frac{1}{M} \int_{x \text{ in } Q} \rho(x)\,(x - x_c)\,dV = 0$$

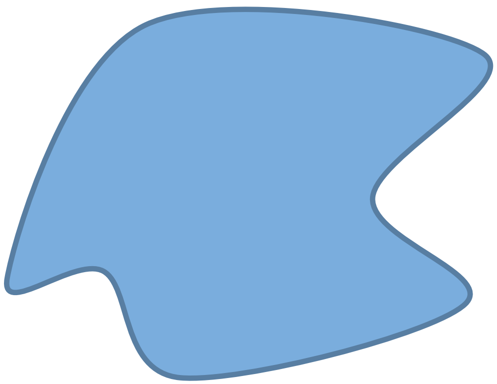
$$\Downarrow$$

$$x_c = \frac{1}{M} \int_{x \text{ in } Q} \rho(x)\,x\,dV$$

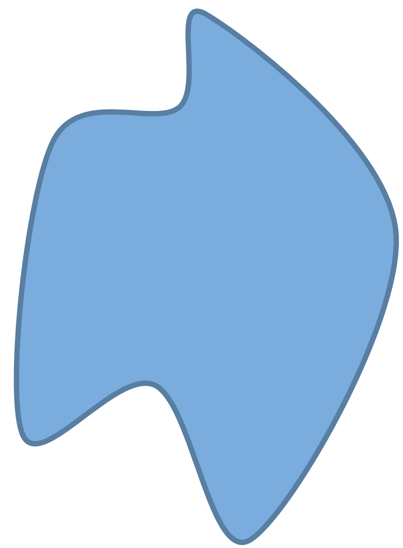
When we refer to the position **x(t)** of a rigid object, we will refer to the position (coordinates in world space) of its **center of mass**.

Rigid Bodies

A rigid body, aside from **position**, also has ***orientation***.



$\boldsymbol{q_1(t)}$



$\boldsymbol{q_2(t)}$

The orientation of an object is represented by a ***quaternion*** $\boldsymbol{q(t)}$*.*
While other representations are possible, quaternions are preferable. *→ see L2*

Animation state

Particle:

$$\begin{bmatrix} x(t) \\ v(t) \end{bmatrix}$$

Rigid body:

$$\begin{bmatrix} x(t) \\ q(t) \\ ? \\ ? \end{bmatrix}$$

Animation state

Particle:

$$\begin{bmatrix} x(t) \\ v(t) \end{bmatrix}$$

Rigid body:

$$\begin{bmatrix} x(t) \\ q(t) \\ P(t) \\ L(t) \end{bmatrix}$$

Quaternion rotation

Linear Momentum

Angular Momentum

Linear Momentum

Formula: $P(t) = m\,v(t)$

Derivative: $\frac{d}{dt}P(t) = F(t)$ (from Newton's second law)

If a closed system is not affected by external forces, its total linear momentum cannot change.

Example:
A heavy truck moving rapidly has a large momentum, and it takes a large or prolonged force to get the truck up to this speed, and would take a similarly large or prolonged force to bring it to a stop.

Torque

Just as a linear force pushes or pulls objects, torque can be thought of as a force twisting/spinning objects.

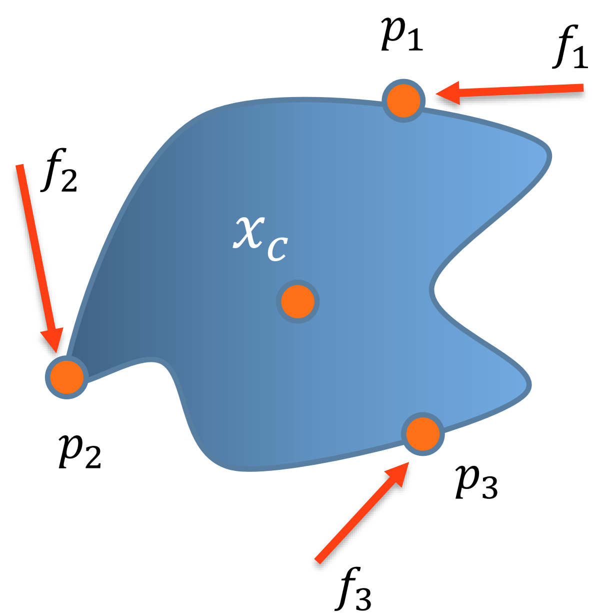
Net torque formula:

$$\tau(t) = \sum (p_i - x_c(t)) \times f_i$$

Remark 1:
Larger force magnitude results in larger torque

Remark 2:
It's easier to spin objects when applying the force at a larger distance to the center of mass (larger lever)

Remark 3:
It's easier to spin objects when applying the force orthogonally



Torque will result in a rotational motion, so we need to define the speed of rotation…

Angular Velocity

In 2D, the **scalar** rate of change of angular position of a rotating body.

Formula: $\omega(t) = \frac{d\varphi(t)}{dt}$ (in 2 dimensions)

In 3D, we use a **vector** $\vec{\omega}(t)$, encoding both the (unit) rotation axis $\vec{u}$ and the speed of the spin (rate of change of angular position $\varphi(t)$ in the plane defined by $\vec{u}$). *See axis-angle representation → L2*

Formula: $\vec{\omega}(t) = \frac{d\varphi(t)}{dt}\vec{u}$ (in 3 dimensions)

Angular Momentum

For simple **particles of mass** (ball attached to a string, satellite orbiting the earth):

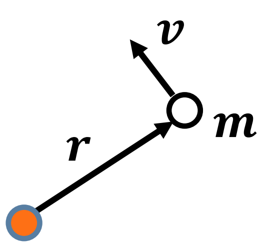
Formula:

$$\vec{L}(t) = \vec{r} \times \vec{P}(t)$$

radius vec × linear momentum

$$= \vec{r} \times \vec{v}(t)\, m$$

linear velocity in t



Angular Momentum

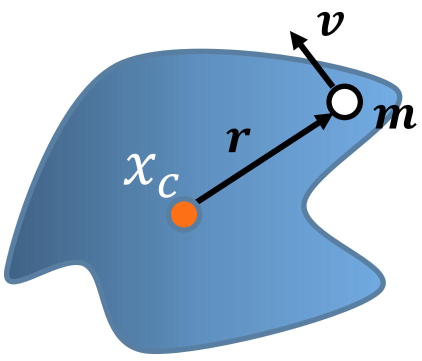
For simple **particles of mass** (ball attached to a string, satellite orbiting the earth):

Formula: $\vec{L}(t) = \vec{r} \times \vec{P}(t)$ ← radius vec × linear momentum!

$$= \underline{\vec{r}} \times \vec{v}(t)\, \underline{m}$$

← linear velocity in t

For general rigid bodies, we need to consider a non-singular volume and non-uniform mass distribution …



Angular Momentum

For simple **particles of mass** (ball attached to a string, satellite orbiting the earth):

$$\vec{L}(t) = \vec{r} \times \vec{v}(t)\, m$$

Relationship
Linear ↔ Angular Velocity

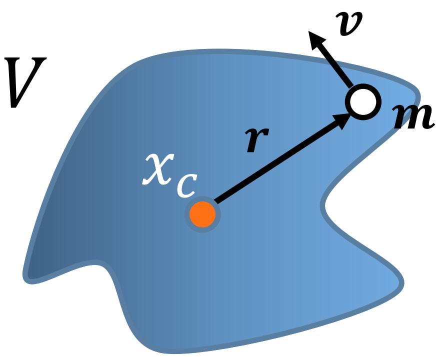
$$\vec{v} = \vec{\omega} \times \vec{r}$$

For **general rigid objects:**

Infinitesimal mass → density

$$\vec{L}(t) = \int_{x \in V} \vec{r}(x) \times \left(\vec{\omega}(t) \times \vec{r}(x)\right) \rho(x)\, dV$$

$$= I(t)\, \vec{\omega}(t)$$



Inertia tensor, encodes the mass distribution of the object.

Inertia Tensor

$$I(t) = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}$$

The inertia tensor is the only thing we need to describe how an object with an arbitrary distribution of mass responds to forces!

Diagonal terms:

$$I_{xx} = \int_V \rho(x,y,z)\,(y^2 + z^2)dV$$

Non-diagonal terms:

$$I_{xy} = -\int_V \rho(x,y,z)\,x\,y\,dV$$

Note: Integration variables x,y,z defined relative to the rotation center.
(Center of mass for free moving objects. Hinge point for objects rotating around a fixed hinge.)

Inertia Tensor

$$I(t) = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}$$

For objects with **uniform mass distribution** and total mass $\boldsymbol{M}$:

Diagonal terms:

$$I_{xx} = M \int_V (y^2 + z^2) dV$$

Non-diagonal terms:

$$I_{xy} = -M \int_V x\, y\, dV$$

Inertia Tensor

- In general, $I(t)$ depends on the body rotation at time t.
- Has to be updated whenever its orientation changes.
- For rigid bodies we can simply **precompute** the integrals of its elements (MC sampling or discretization) in object space:

  → **body-space** tensor I_{body}

- Gives current tensor $I(t)$ by applying current rotation:

$$I(t) = R(t)\, I_{body}\, R(t)^T$$

where $R(t)$ is the current rotation matrix (can be derived from $\mathrm{q}(t)$)

Angular Momentum

For **general rigid objects:**

Formula: $\vec{L}(t) = I(t)\,\vec{\omega}(t) \quad \longleftrightarrow \quad \vec{\omega}(t) = I^{-1}(t)\,\vec{L}(t)$

Derivative: $\frac{d}{dt}\vec{L}(t) = \vec{\tau}(t)$

Analogous to linear momentum, but for rotational motion: a heavy object that rotates fast requires large prolonged force to get it up to this speed, and an equally large force in order to stop it.

Rigid-body Motion ODE

$$\frac{d}{dt}\begin{bmatrix} x(t) \\ q(t) \\ P(t) \\ L(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ 0.5\,\omega(t)q(t) \\ F(t) \\ \tau(t) \end{bmatrix}$$

$\omega(t)$ as quaternion $(0, \omega(t))$

Proof in the lecture notes

Sum of all forces

Total torque

Rigid-body Representation

$x(t)$
$q(t)$
$v(t)$
$I(t)$
$\omega(t)$
$F(t)$
$\tau(t)$
M

```
void  ode(double y0[], double yend[], int len, double t0,
          double t1, dydt_func dydt);
```

Animation state y0 at t0

Derivative computation

ODE Solver

Animation state at t1

x N times for simulations with N objects

Quaternions vs. Rotation Matrices

- Instead of quaternions, a 3x3 rotation matrix can be used to represent orientation
  - 9 vs. 4 variables to represent the 3DoF of rotation
  - The numerical ODE solver introduces drift
  - Less variables → less drift
    (quaternions are more robust to numerical errors)
  - Drift in the case of a rotation matrix will result in a non-orthogonal matrix that will cause a skewing effect
  - Drift in the case of a quaternion will result in a quaternion that is not unit length

Solution: Normalize quaternion after every solver step to obtain unit quaternion again.

Angular Momentum in 2D

- Object extent only in xy-plane
- Perpendicular axis theorem: $I_{zz} = I_{xx} + I_{yy}$

$$\rightarrow \ \boldsymbol{I} = \begin{bmatrix} I_{xx} & I_{xy} & 0 \\ I_{yx} & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} = \begin{bmatrix} I_{xx} & I_{xy} & 0 \\ I_{yx} & I_{yy} & 0 \\ 0 & 0 & I_{xx} + I_{yy} \end{bmatrix}$$

- Axis of rotation is the z-axis: $\vec{\omega}(t) = \left(0, 0, \omega_{xy}(t)\right)$

angular velocity in the xy-plane

- Angular momentum $\vec{L}(t) = \boldsymbol{I}\,\vec{\omega}(t) = (0, 0, I_{zz}\omega_{xy}(t))$
- simplifies to scalar product $L_{xy}(t) = I_{zz}\,\omega_{xy}(t)$

Single scalar, all we need to store for a 2D object

4. Collision Detection and Response

Collision detection types

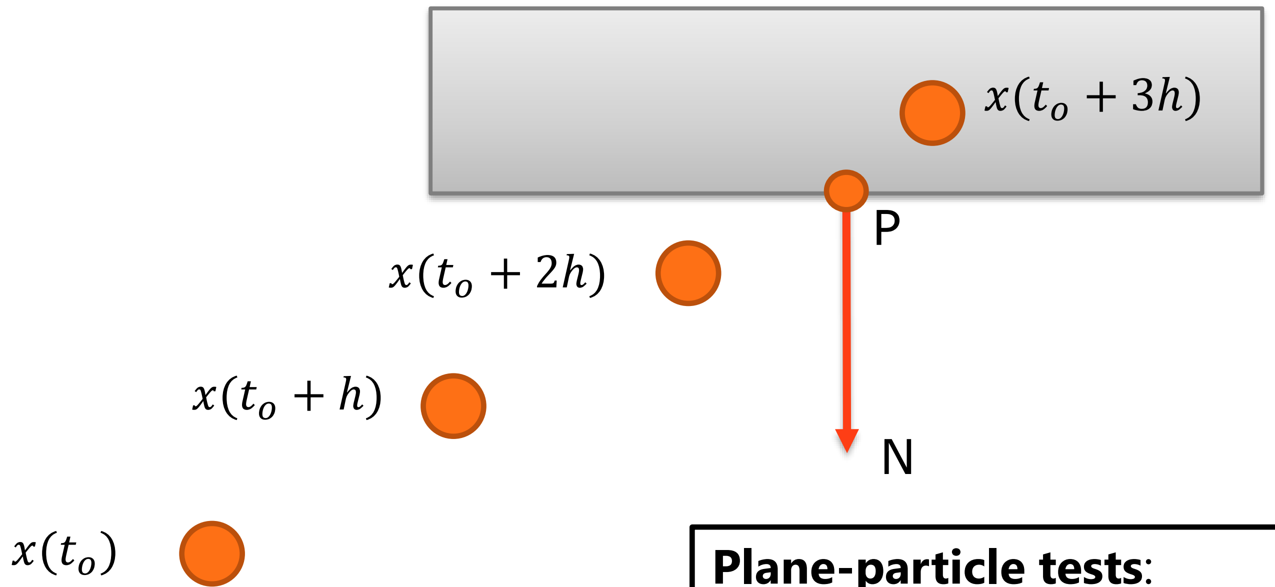
- **Discrete** (a posteriori)
  The simulation proceeds in steps. After each step, a list of colliding objects are detected, and their position is "fixed"
  → collisions are detected *after* the collision event.

- **Continuous** (a priori)
  The collision detection method is able to predict very precisely the time and place a collision happens and the physical bodies never actually interpenetrate
  → collisions are detected *before* the collision event.

Discrete

Particle collisions

Avoid interpenetrations between solid/rigid objects at collision.

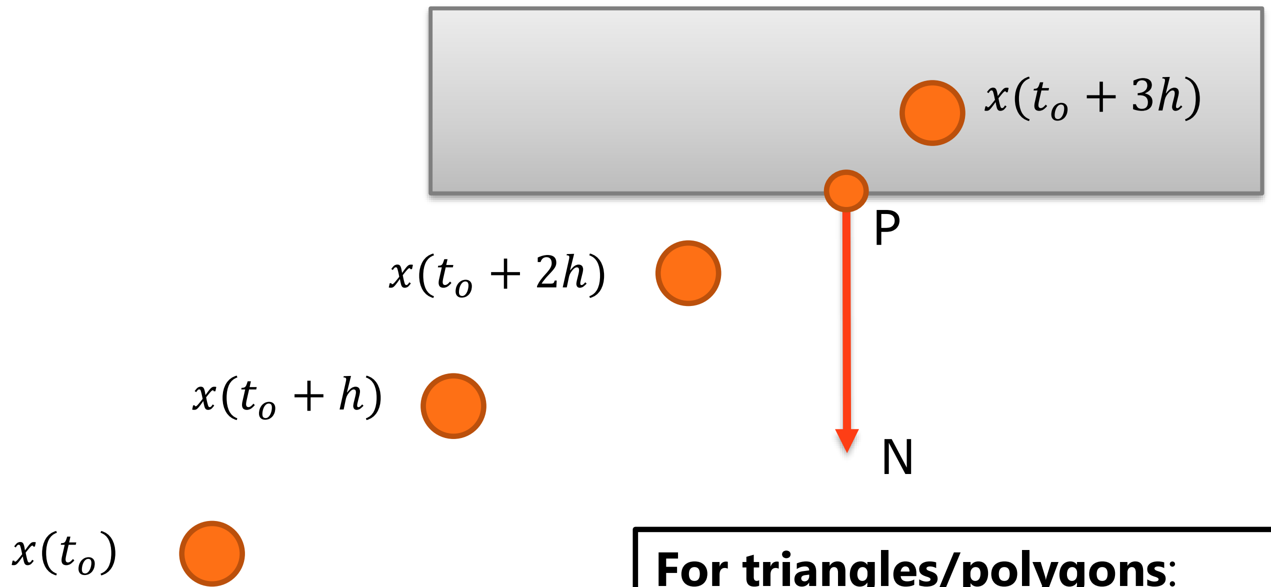


Plane-particle tests:
$(x - P) \cdot N > 0 \;\rightarrow in\ front\ of\ the\ plane$
$(x - P) \cdot N < 0 \;\rightarrow behind\ the\ plane$
$(x - P) \cdot N < \varepsilon \rightarrow very\ close, heading\ in$

Discrete

Particle collisions

Avoid interpenetrations between solid/rigid objects at collision.

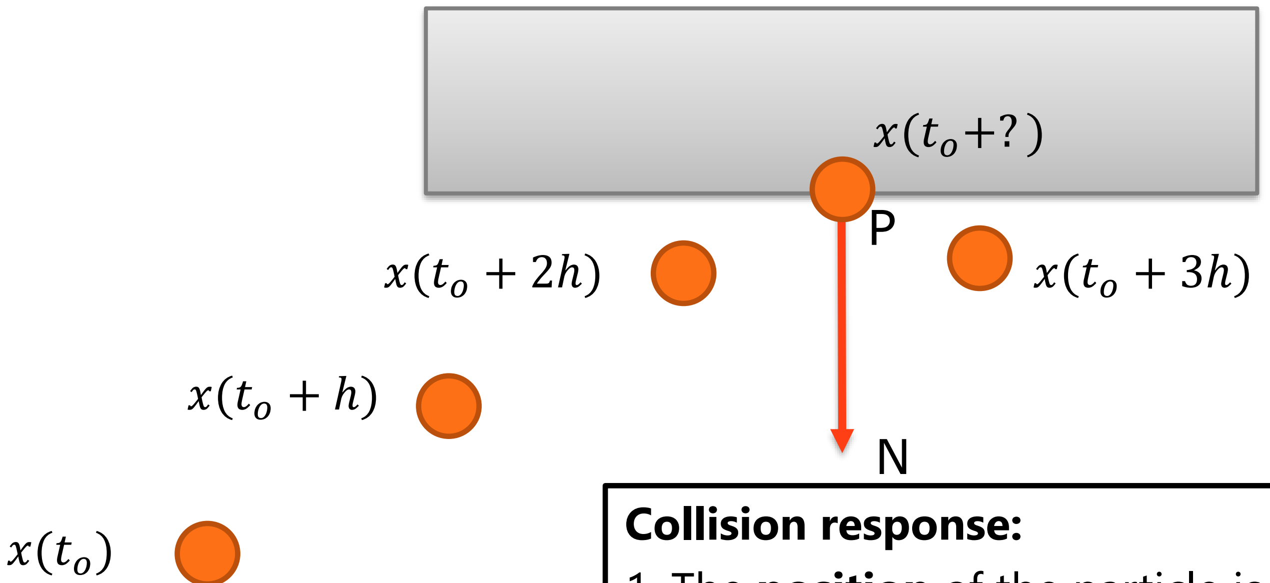


For triangles/polygons:
If the particle crosses the plane of the polygon, compute the ray-plane intersection and test if the intersection point lies within the polygon.

Discrete

Particle collisions

Avoid interpenetrations between solid/rigid objects at collision.



Collision response:

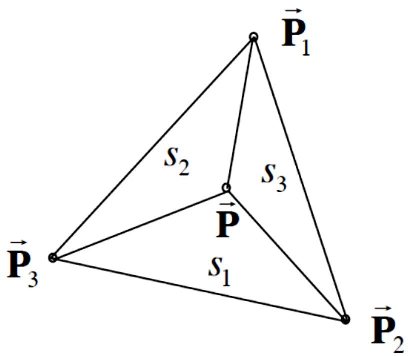
1. The **position** of the particle is moved to a non-penetrating position
2. The **velocity** of the object is adjusted
3. The correct position is computed based on the new position, velocity and residual time

Continuous

Particle collisions

For linear trajectories, perform a ray-object intersection.

Ray-triangle intersection:



Barycentric coordinates

$$\vec{P} = s_1\vec{P}_1 + s_2\vec{P}_2 + s_3\vec{P}_3$$

Inside triangle criteria

$$s_1 = \mathrm{area}(\triangle \mathbf{P}\mathbf{P_2}\mathbf{P_3})/\mathrm{area}(\triangle \mathbf{P_1}\mathbf{P_2}\mathbf{P_3})$$
$$s_2 = \mathrm{area}(\triangle \mathbf{P_1}\mathbf{P}\mathbf{P_3})/\mathrm{area}(\triangle \mathbf{P_1}\mathbf{P_2}\mathbf{P_3})$$
$$s_3 = \mathrm{area}(\triangle \mathbf{P_1}\mathbf{P_2}\mathbf{P})/\mathrm{area}(\triangle \mathbf{P_1}\mathbf{P_2}\mathbf{P_3})$$

$$0 \le s_1 \le 1$$
$$0 \le s_2 \le 1$$
$$0 \le s_3 \le 1$$
$$s_1 + s_2 + s_3 = 1$$

Equation for the intersection point:

$$O + tD = (1 - s_1 - s_2)P_3 + s_1 P_1 + s_2 P_2$$

$O = ray\ origin$
$D = ray\ direction$

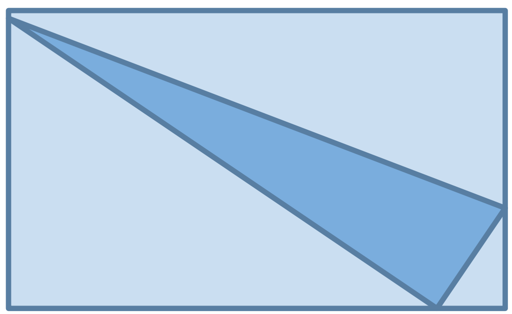
N-body collision

- Trivial approach: test all possible pairs
  - For N objects this will result in $\boldsymbol{O}(\boldsymbol{N}^2)$ tests!
  - **Prohibitive cost** for real-time applications with high number of objects.

↓

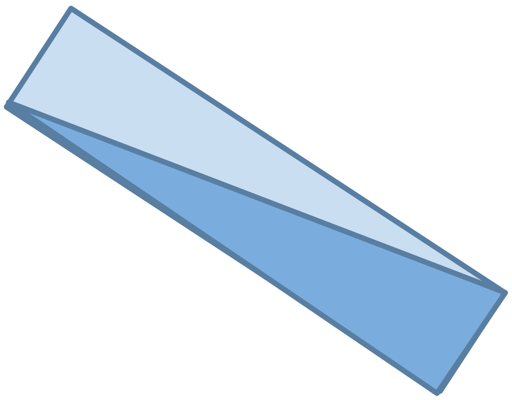
- Prune as-fast-as-possible collision tests of object-pairs that are not inter-penetrating.
- Only spend time for accurate collision tests on object pairs that are potentially penetrating.

Bounding boxes



Axis Aligned BB

1. Fast to compute
2. Very fast tests against points, other AABB, and polygons
3. Less tight bounds
   → BB test will rule out less non-intersecting objects

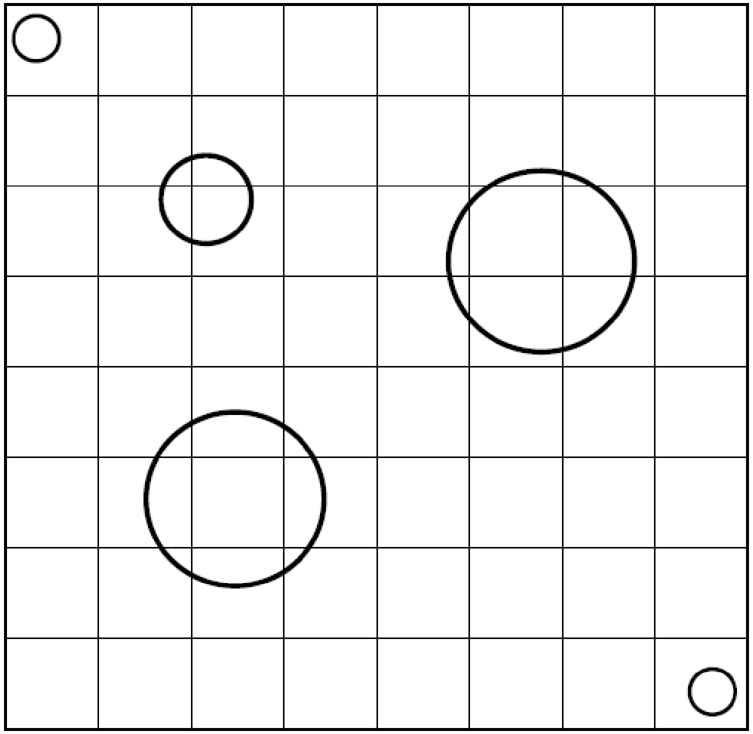


Object-Oriented BB

1. Tight Bounds
2. Slower collision tests
3. Slower build times

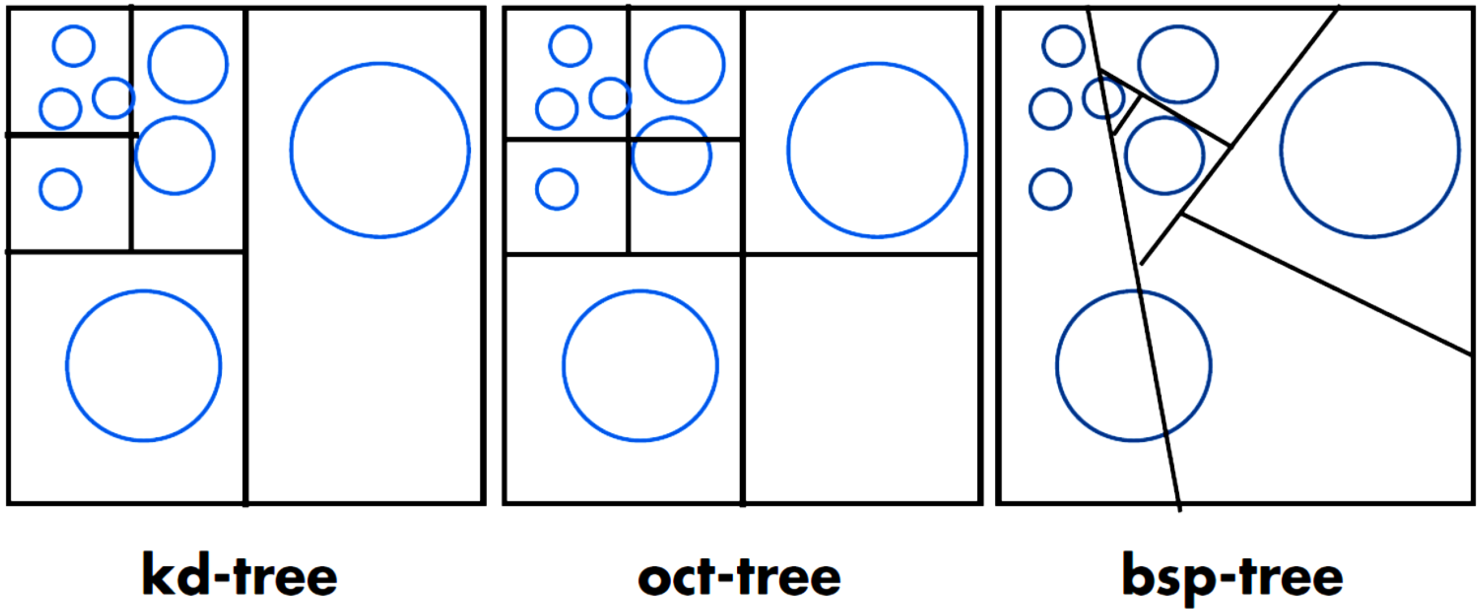
Uniform Grids

- Subdivide the space in uniform cells
- Only test objects in the same cells



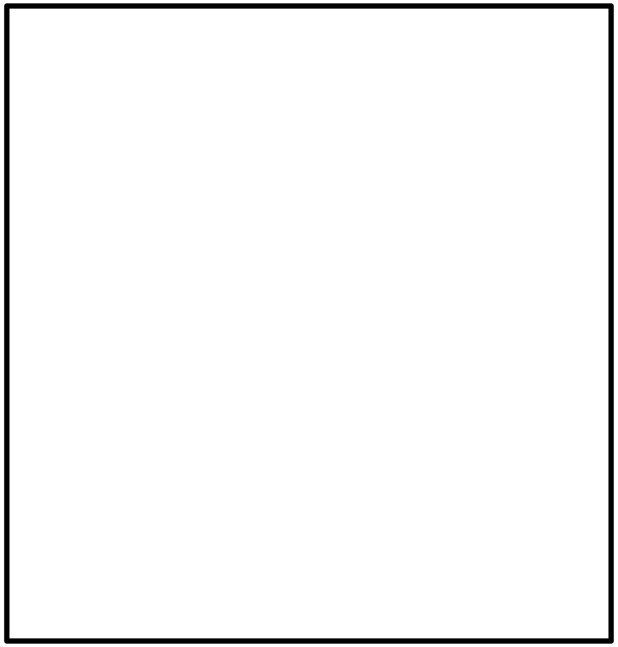
Problems:
1. Does not work very well when the distribution of objects is non-uniform
2. What is the optimal cell size?

Spatial Hierarchies

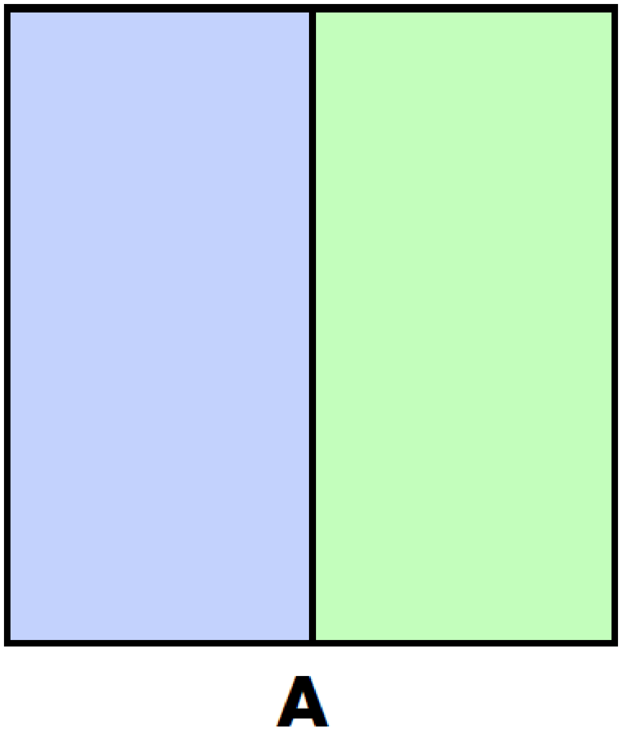


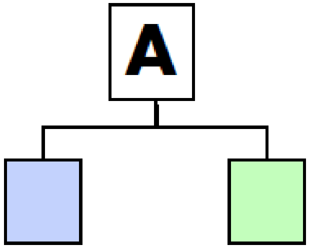
1. The space is subdivided in convex cells.
2. Only perform collision tests with objects in the same convex-cell.

Spatial Hierarchies



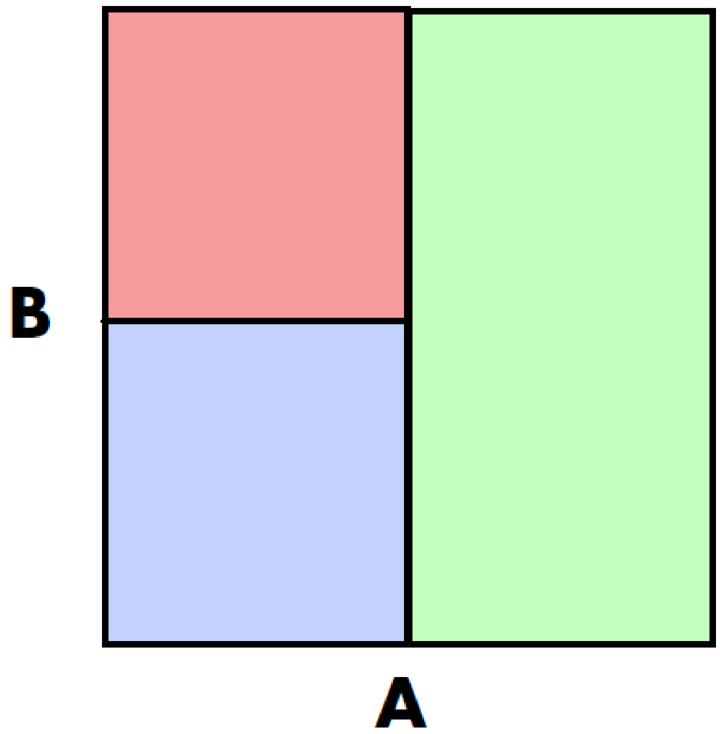
Spatial Hierarchies

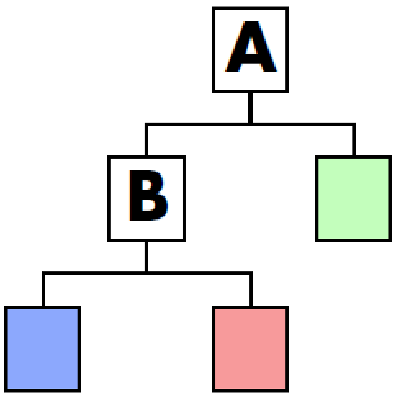




Letters correspond to planes (A)

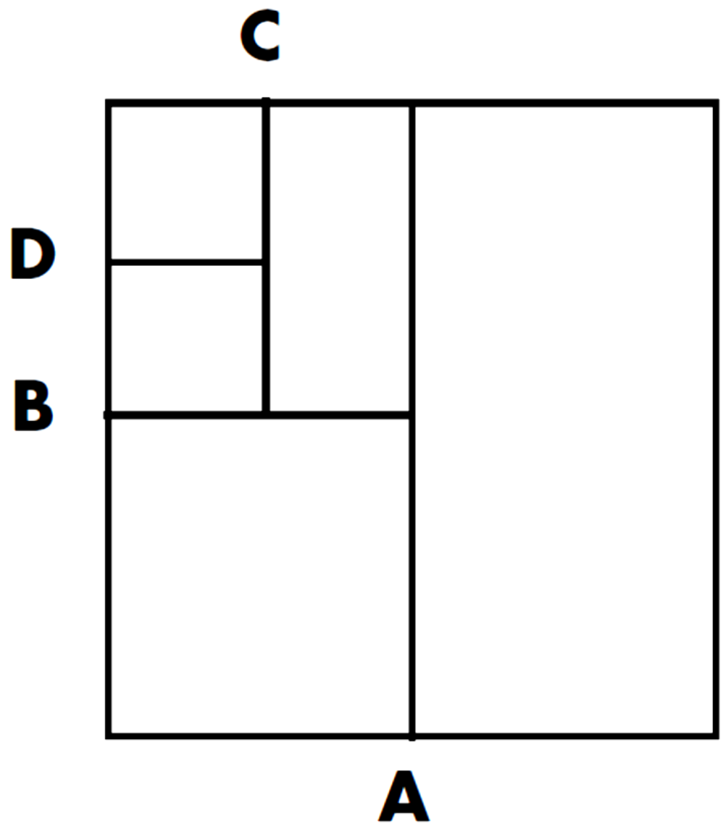
Spatial Hierarchies

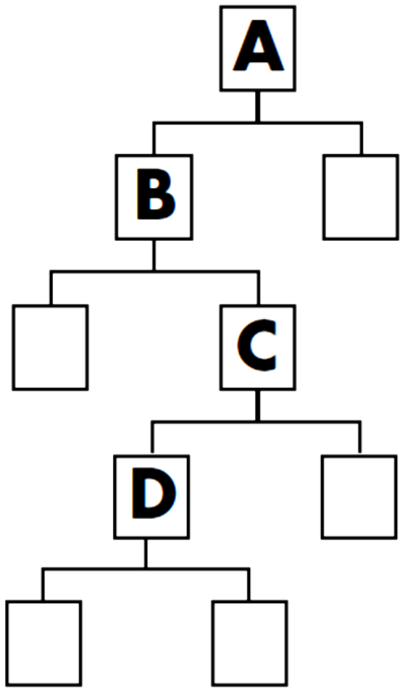




Letters correspond to planes (A,B)

Spatial Hierarchies

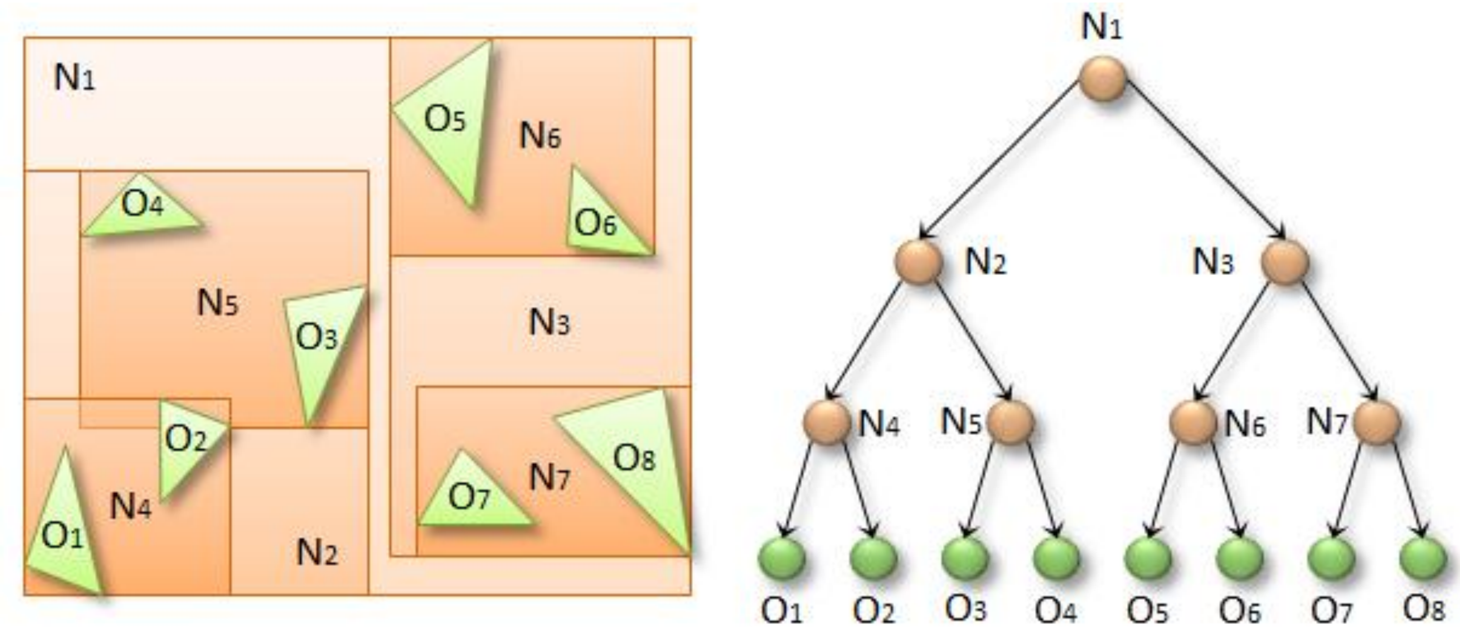




Letters correspond to planes (A,B,C,D)

Bounding Volume Hierarchies

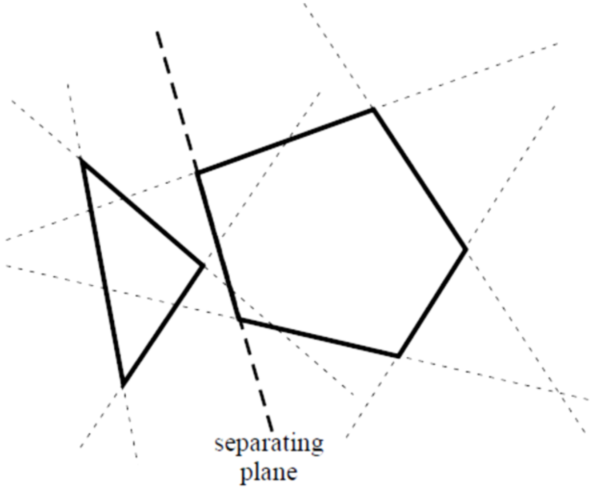
- Bottom-up hierarchical grouping of primitives
  Note: bounding volumes might overlap!
- Fast build times, more suitable for dynamic objects



Object-object collisions

- Convert both objects to BVHs or other spatial hierarchies
- Test the convex subspaces for collisions
- **Separating-Axis Theorem (SAT):**

  Two arbitrary convex regions do not interpenetrate if a separating axis (or plane in 3D) exists:



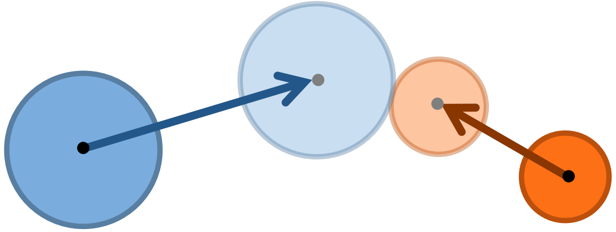
In convex triangle-meshes, finding the separating axis/plane minimizes the amount of analytical triangle-triangle intersection tests!

Collision Response

- Objects that collide should respond according to

  Newton's third law of motion:

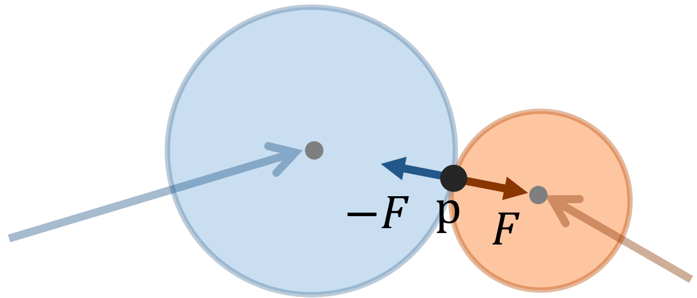
  When one body exerts a force on a second body, the second body simultaneously exerts a force equal in magnitude and opposite in direction on the first body



- Requires updating the linear and angular momentum of both objects. (Analogously: update linear and angular velocities)

Momentum Update

- Momentums are updated based on exerted (opposing) forces acting **at the contact point** $\mathbf{p}$.



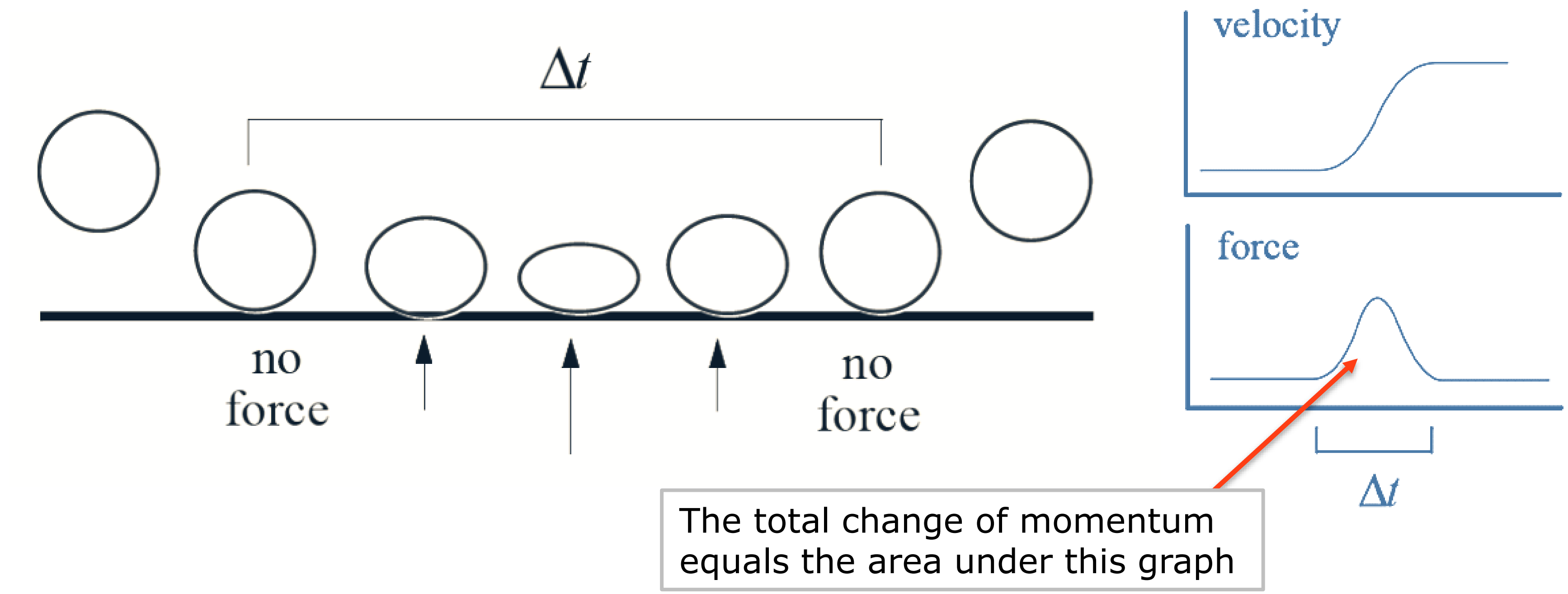
- In fricitonless bodies, forces act only in direction orthogonal to the contact surface (surface normal direction).
- Change in linear momentum equals force F times duration d of exertion.

$$\Delta P = \int_0^d F(t)\,dt$$

- Rigid Bodies: How large is F? … and d ?

Collision Process

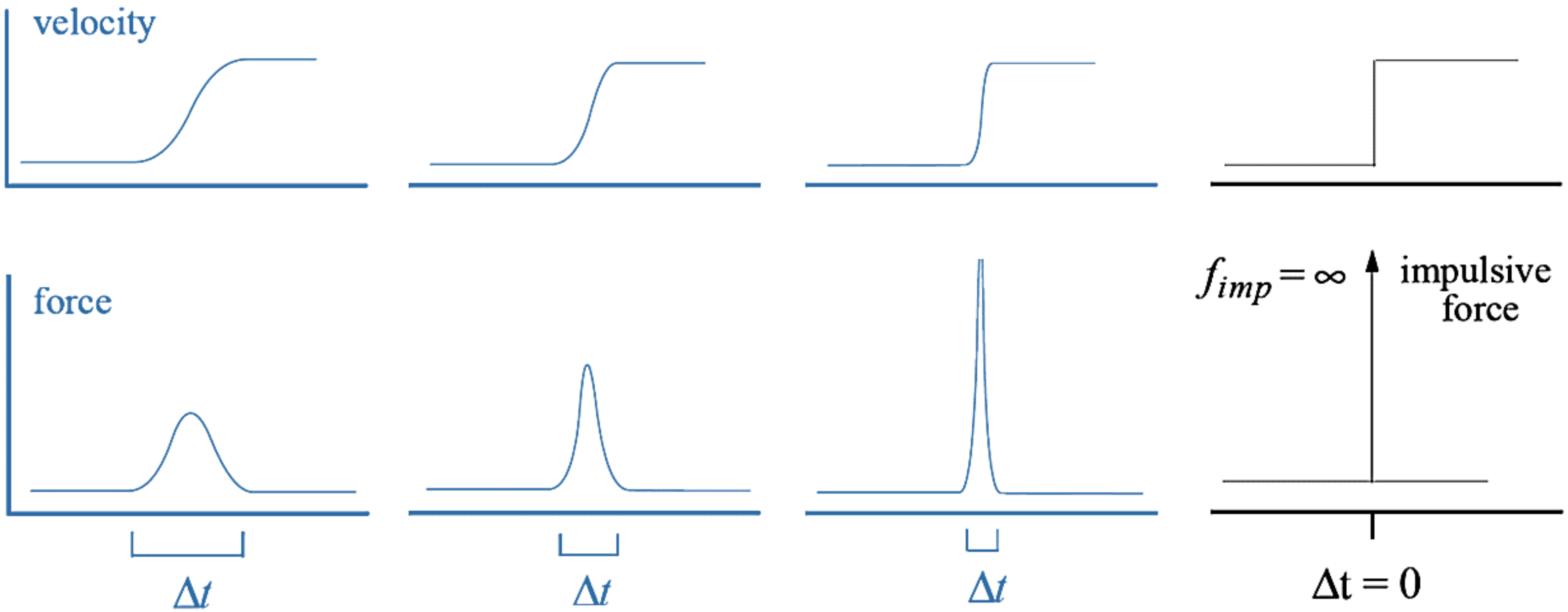
- General (non-singular) collision and bounce process:



- **In practice** we will assume instantaneous rigid collisions (no deformations) → $\Delta t = 0$

Collision Process

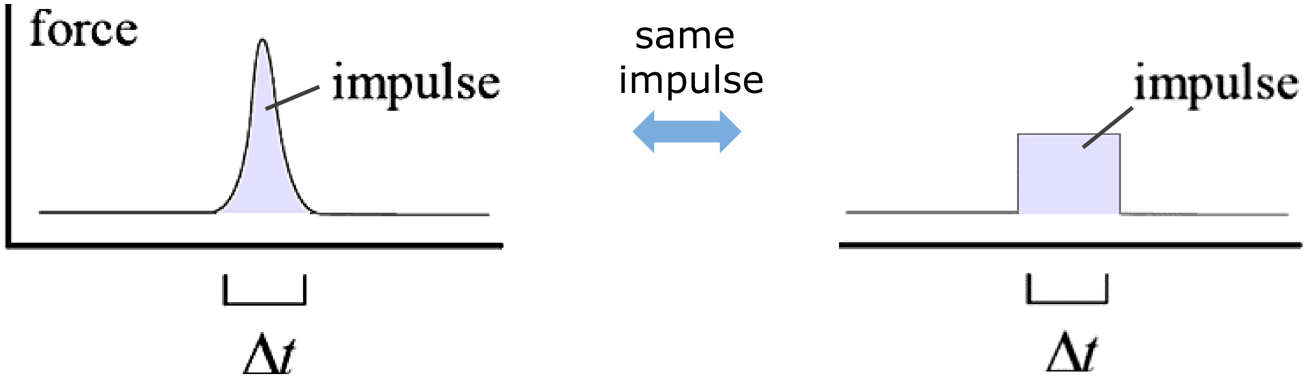
- Vanishing the contact duration/deformation time: $\Delta t \to 0$



- Instantaneous collisions ($\Delta t = 0$) would require an infinite force to produce the same change in momentum ΔP.
- In practice we will instead work with a finite **impulse**.

Impulse (J)

- Defines the integral force F over a certain time interval Δt:



- Impulse $\mathrm{J} = \mathrm{F} \cdot \Delta \mathrm{t} = \Delta \mathrm{P}$ → defines change in momentum:
  - linear: $\mathrm{dP} = \mathrm{F\,dt} = \mathrm{J}$
  - angular: $\mathrm{dL} = \tau\,\mathrm{dt} = (\mathrm{p} - \mathrm{c}) \times \mathrm{F\,dt} = (\mathrm{p} - \mathrm{c}) \times \mathrm{J}$
- If one colliding body experiences an impulse J, the other experiences an impulse –J.

Impulse at collision point

- On frictionless colliding bodies, impulse acts only along the contact surface normal direction $\hat{n}$.

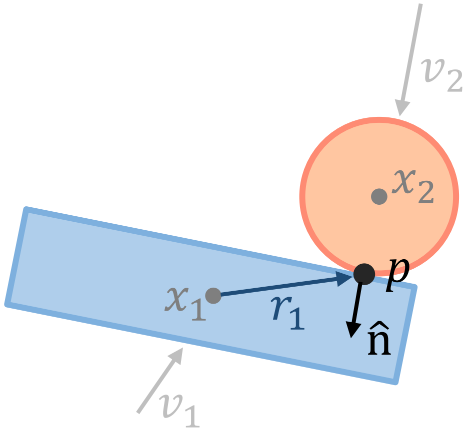
$J = j\,\hat{n}$

- The first body experiences an impulse at **collision point $\boldsymbol{p}$** based on its **relative velocity at $\boldsymbol{p}$** along $\hat{n}$:

$$j = \frac{-(1+\varepsilon)\ (\dot{p}_1 - \dot{p}_2)\cdot\hat{n}}{m_1^{-1} + m_2^{-1} + \left[\left(I_1^{-1}(r_1\times\hat{n})\right)\times r_1 + \left(I_2^{-1}(r_2\times\hat{n})\right)\times r_2\right]\cdot\hat{n}}$$

m_i, v_i	mass and linear velocity of body i
$r_i = p - x_i$	radius vector of p in body i
$\dot{p}_i = v_i + \omega_i \times r_i$	linear velocity of p in body i
$\varepsilon \in [0,1]$	restitution coefficient (bounciness) ε=1: perfect elastic collision ε<1: loss of kinetic energy

at time of contact

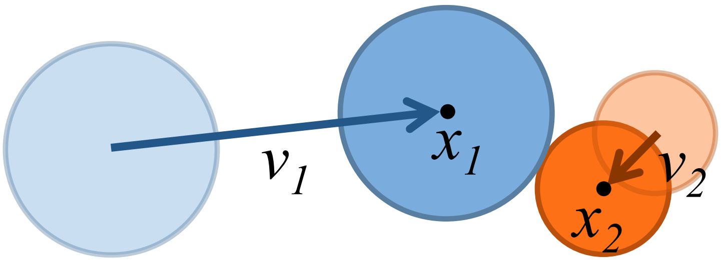


By convention, $\hat{n}$ points towards body 1.

Example: Elastic sphere collision

$$j = \frac{-(1+\varepsilon)\ (\dot{p}_1 - \dot{p}_2)\cdot \hat{n}}{m_1^{-1} + m_2^{-1} + \left[\left(I_1^{-1}(r_1 \times \hat{n})\right) \times r_1 + \left(I_2^{-1}(r_2 \times \hat{n})\right) \times r_2\right] \cdot \hat{n}}$$

- Perfect elastic collision: ε=1

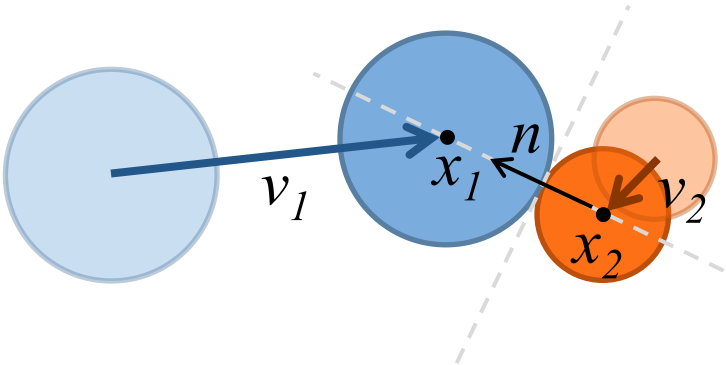


Example: Elastic sphere collision

$$j = \frac{-2\,(\dot{p}_1 - \dot{p}_2)\cdot\hat{n}}{m_1^{-1} + m_2^{-1} + \left[\left(I_1^{-1}\cancel{(r_1\times\hat{n})}\right)\times r_1 + \left(I_2^{-1}\cancel{(r_2\times\hat{n})}\right)\times r_2\right]\cdot\hat{n}}$$

- Perfect elastic collision: ε=1
- Contact point on spheres:

$$\hat{n} = \frac{x_1 - x_2}{\|x_1 - x_2\|} \quad || \quad r_i$$

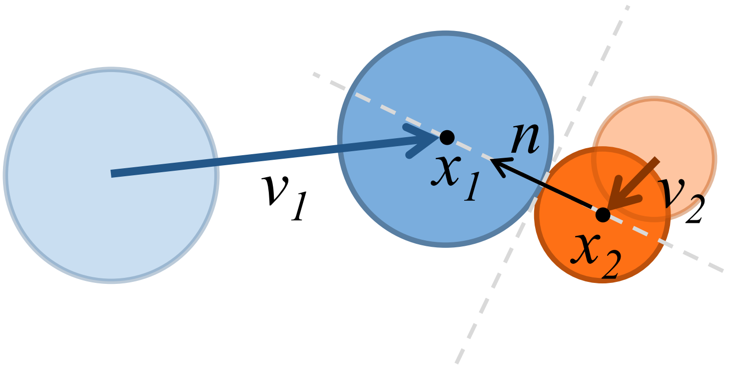


Example: Elastic sphere collision

$$j = \frac{-2\,(\dot{p}_1 - \dot{p}_2)\cdot \hat{n}}{m_1^{-1} + m_2^{-1}}$$

- Perfect elastic collision: ε=1
- Contact point on spheres:

$$\hat{n} = \frac{x_1 - x_2}{\|x_1 - x_2\|} \quad || \; r_i$$



Example: Elastic sphere collision

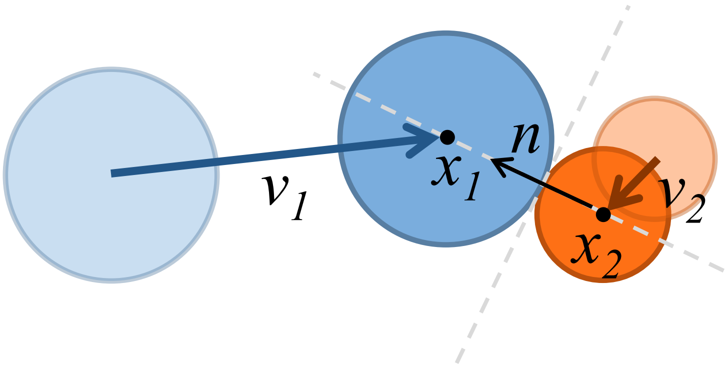
$$j = \frac{-2\,(v_1 - v_2)\cdot \hat{n}}{m_1^{-1} + m_2^{-1}}$$

- Perfect elastic collision: ε=1
- Contact point on spheres:

  $\hat{n} = \frac{x_1 - x_2}{\|x_1 - x_2\|} \;||\; r_i$
- Assume no spin: $\dot{p}_i = v_i$
- Velocity changes:

$$dv_1 = \frac{dP_1}{m_1} = +\boldsymbol{J}/m_1 = +\boldsymbol{j}\,\hat{n}/m_1$$

$$dv_2 = \frac{dP_2}{m_2} = -\boldsymbol{J}/m_2 = -\boldsymbol{j}\,\hat{n}/m_2$$



Example: Elastic sphere collision

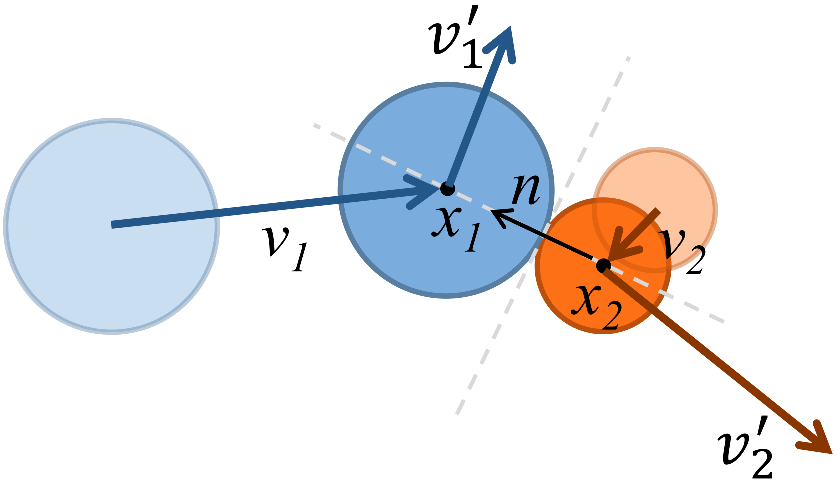
$$j = \frac{-2\,(v_1 - v_2)\cdot \hat{n}}{m_1^{-1} + m_2^{-1}}$$

- Perfect elastic collision: $\varepsilon=1$
- Contact point on spheres:

  $\hat{n} = \frac{x_1 - x_2}{\|x_1 - x_2\|} \;||\; r_i$
- Assume no spin: $\dot{p}_i = v_i$
- New Velocities

  $$v_1' = v_1 - \frac{2m_2}{m_1+m_2}(v_1 - v_2)\cdot n * n$$

  $$v_2' = v_2 + \frac{2m_1}{m_1+m_2}(v_1 - v_2)\cdot n * n$$



Many real-world collisions are inelastic, i.e., some kinetic energy is transformed to deformation energy or heat.

Lecture Notes

- Additional details, examples and code samples in lecture nodes → TeachCenter:
  - **Ordinary differential equations**
    L6_ODE_basics.pdf
  - **Rigid body dynamics (Siggraph course notes)**
    L6_rigid_bodies.pdf

Acknowledgements:

Andrew Witkin and David Baraff,
Physically Based Modeling: Principles and Practice

Conclusion

- Overview of the principles behind physically-based animation.
  - Particles
  - Rigid bodies
- First order ODEs and solvers
  - Euler’s method, midpoint, etc…
- Equations of motion and related physical quantities
  - Inertia tensor, linear momentum, angular momentum
- Collision detection and response
  - Impulse, elastic collision