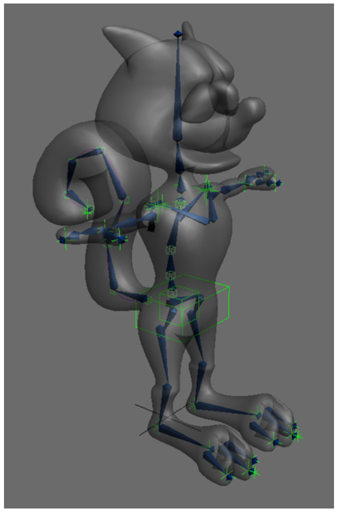
CSC 433/533 Computer Graphics

Alon Efrat
Credit: Joshua Levine

Animation 2

Character Animation

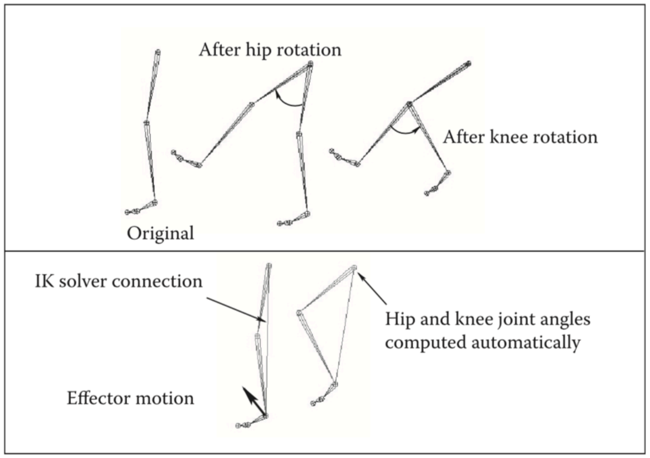
Rigged character



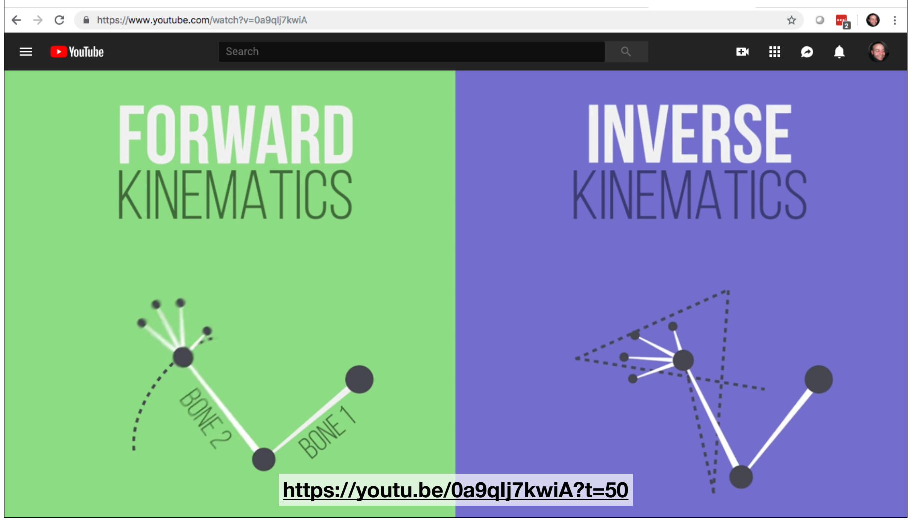
[CIS 565 staff]

- Surface is deformed by a set of *bones*
- Bones are in turn controlled by a smaller set of *controls*
- The controls are useful, intuitive DOFs for an animator to use

Forward vs. Inverse Kinematics

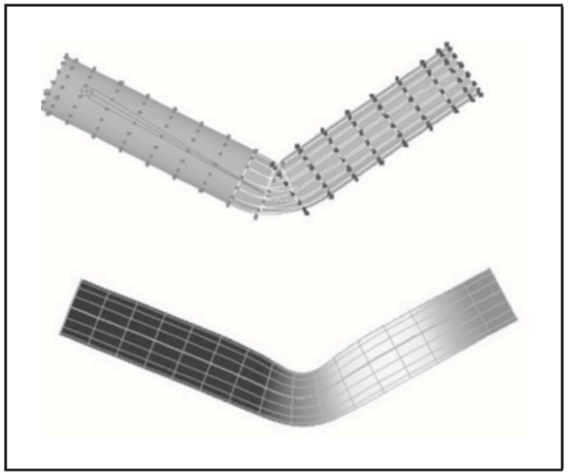


Inverse Kinematics Solves for all Intermediate Constraints



Skinning

- After solving for the skeleton, one still needs to update and deform the surface



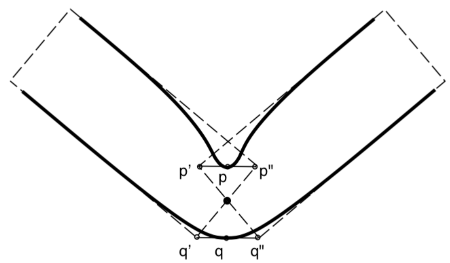
Mesh skinning math: setup

- Surface has control points $\mathbf{p}_i$
  - Triangle vertices, spline control points, subdiv base vertices
- Each bone has a transformation matrix M_j
  - Normally a rigid motion
- Every point–bone pair has a weight w_{ij}
  - In practice only nonzero for small # of nearby bones
  - The weights are provided by the user

Mesh skinning math

- Deformed position of a point is a weighted sum
  - of the positions determined by each bone's transform alone
  - weighted by that vertex's weight for that bone

$$\mathbf{p}'_i = \sum_j w_{ij} M_j \mathbf{p}_i$$



[Lewis et al. SIGGRAPH 2000]

Skinning Mesh Animations

Doug L. James
Christopher D. Twigg

Carnegie Mellon University

http://graphics.cs.cmu.edu/projects/sma/, 2005

https://youtu.be/_QZN2IC0vOo

Physics-Based Animation

Animation vs. Simulation

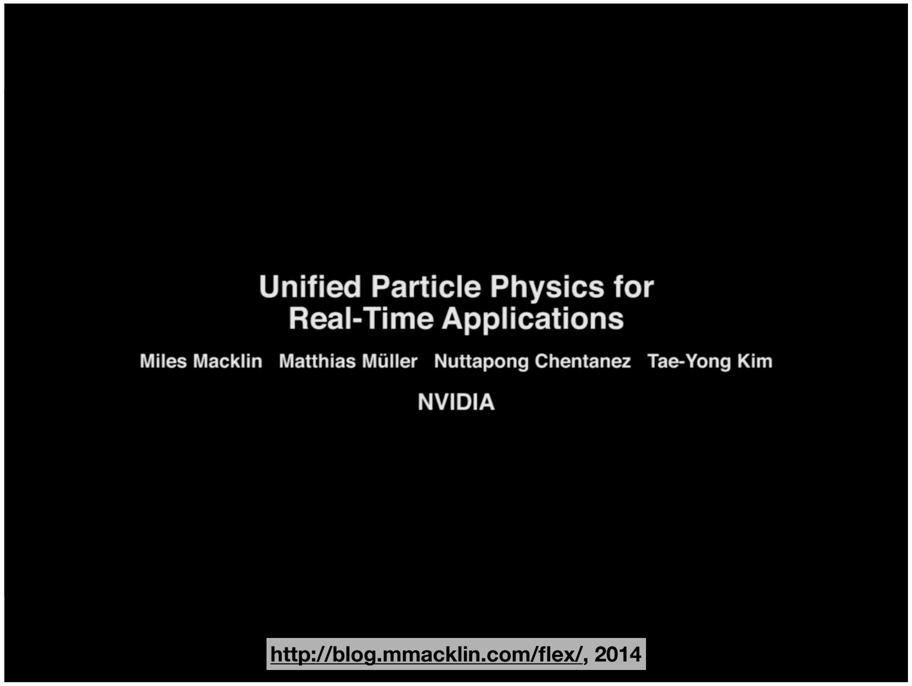
- Animation methods use scripted actions to make objects change
- Simulation: simulate physical laws by associating physical properties to objects
- Solve for physics to achieve (predict) realistic effects

Using Particle Systems

- Idea: Represent the physics on the simplest possible entity: particles
- Used for effects like smoke, fire, water, sparks, and more
- Plenty of other approaches, this is just one family

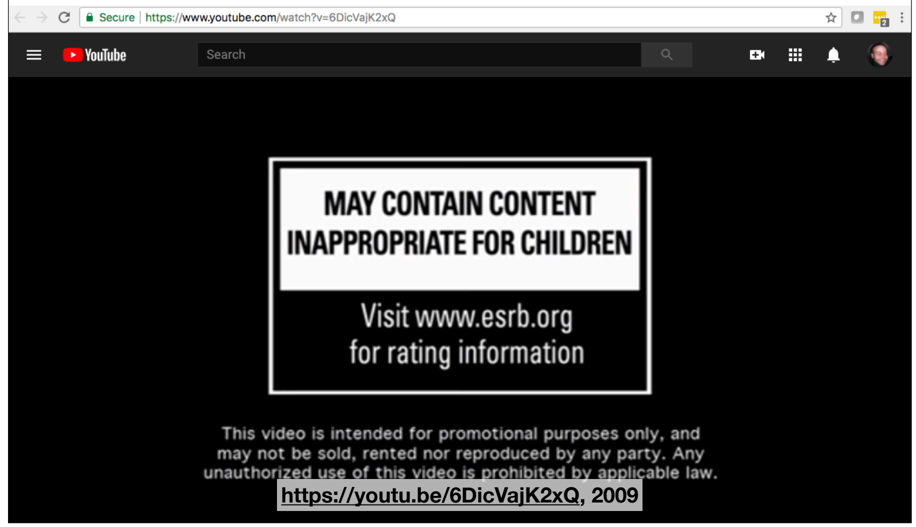
, 1988

http://physbam.stanford.edu/~fedkiw/, 2008



http://blog.mmacklin.com/flex/, 2014

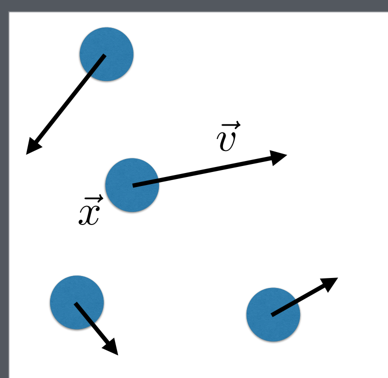
Used in Games Physics



https://youtu.be/6DicVajK2xQ, 2009

Particle System Setup

```
class Particle {
  Vector3 position;
  Vector3 velocity;
};
```



Moving Particles

Position is a function of time

- i.e., $\vec{x} \equiv \vec{x}(t)$
- Note that $\vec{v}(t) \equiv \frac{\partial \vec{x}}{\partial t}$

Use a function to control the particle's velocity

- $\vec{v}(t) = f(\vec{x}(t))$

This is an Ordinary Differential Equation (ODE)

Solve this ODE at every frame

- i.e., solve for $\vec{x}(t_0), \vec{x}(t_1), \vec{x}(t_2), \ldots$
- Then we can draw each of these positions to the screen

A Simple Example

Let $\vec{v}$ be constant

- e.g., $\vec{v} = f(\vec{x}) = (0, 0, 1)^\top$

Then we can solve for the position at any time:

- $\vec{x}(t) = \vec{x}(0) + t\vec{v}$

Not always so easy

- $f(\vec{x})$ can be anything!
- Might be unknown until runtime (e.g., user interaction)
- Often times, not solved exactly

Moving Particles, Revisited

Now, acceleration is in the mix

- $\vec{a}(t) \equiv \dfrac{\partial \vec{v}}{\partial t} \equiv \dfrac{\partial^2 \vec{x}}{\partial t^2}$

Use a function to control the particle's acceleration

- $\vec{a}(t) = f(\vec{x}(t))$

This is a Second Order ODE

Solve this ODE at every frame, same as before

- Can sometimes be reduced to a first order ODE
- Calculate position and velocity together

Physically-based Motion

Acceleration based on Newton's laws

- $\vec{f}(t) = m\vec{a}(t)$ …or, equivalently… $\vec{a}(t) = \vec{f}(t)/m$
- i.e., force is mass times acceleration

Forces are known beforehand

- e.g., gravity, springs, others….
- Multiple forces sum together
- These often depend on the position, i.e., $\vec{f}(t) \equiv \vec{f}(\vec{x}(t))$
- Sometimes velocity, too

If we know the values of the forces, we can solve for particle's state

Unary Forces

Constant

- Gravity

Position/Time-Dependent

- Force fields, e.g. wind

Velocity-Dependent

- Drag

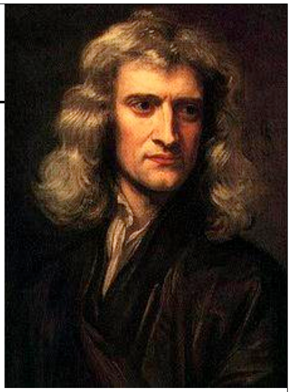
Ordinary Differential Equations

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

- Given a function $f(\mathbf{X},t)$ compute $\mathbf{X}(t)$
- Typically, *initial value problems*:
  - Given values $\mathbf{X}(t_0)=\mathbf{X}_0$
  - Find values $\mathbf{X}(t)$ for $t > t_0$

- We can use lots of standard tools

Newtonian Mechanics



This image is in the public domain.
Source: Wikimedia Commons.

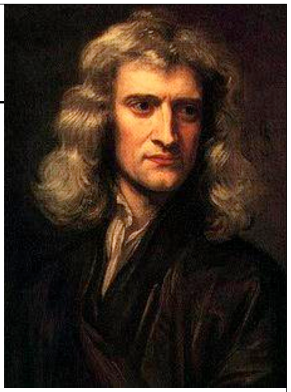
- Point mass: 2nd order ODE

$$\vec{F} = m\vec{a} \quad \text{or} \quad \vec{\boldsymbol{F}} = m\frac{d^2\vec{\boldsymbol{x}}}{dt^2}$$

- Position $\boldsymbol{x}$ and force $\boldsymbol{F}$ are vector quantities
  - We know $\boldsymbol{F}$ and m, want to solve for $\boldsymbol{x}$

- You have all seen this a million times before

Reduction to 1st Order



This image is in the public domain.
Source: Wikimedia Commons.

- Point mass: 2nd order ODE

$$\vec{F} = m\vec{a} \quad \text{or} \quad \vec{\boldsymbol{F}} = m\frac{d^2\vec{\boldsymbol{x}}}{dt^2}$$

- Corresponds to system of first order ODEs

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

2 unknowns (**x**, **v**)
instead of just **x**

Reduction to 1st Order

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

2 variables (**x**, **v**)
instead of just one

- Why reduce?

Reduction to 1st Order

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

2 variables (**x**, **v**)
instead of just one

- Why reduce?
  - Numerical solvers grow more complicated with increasing order, can just write one 1st order solver and use it
  - Note that this doesn't mean it would always be easy :-)

Notation

- Let's stack the pair (**x**, **v**) into a bigger state vector **X**

$$\boldsymbol{X} = \begin{pmatrix} \vec{\boldsymbol{x}} \\ \vec{\boldsymbol{v}} \end{pmatrix}$$

For a particle in 3D, state vector **X** has 6 numbers

$$\frac{d}{dt}\boldsymbol{X} = f(\boldsymbol{X}, t) = \begin{pmatrix} \vec{\boldsymbol{v}} \\ \vec{\boldsymbol{F}}(x, v)/m \end{pmatrix}$$

Now, Many Particles

- We have N point masses
  - Let's just stack all **x**s and **v**s in a big vector of length 6N

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{v}_1 \\ \vdots \\ \boldsymbol{x}_N \\ \boldsymbol{v}_N \end{pmatrix} \qquad f(\boldsymbol{X}, t) = \begin{pmatrix} \boldsymbol{v}_1 \\ \boldsymbol{F}^1(\boldsymbol{X}, t) \\ \vdots \\ \boldsymbol{v}_N \\ \boldsymbol{F}^N(\boldsymbol{X}, t) \end{pmatrix}$$

Now, Many Particles

- We have N point masses
  - Let's just stack all **x**s and **v**s in a big vector of length 6N
  - $\mathbf{F}^i$ denotes the force on particle *i*
    - When particles don't interact, $\mathbf{F}^i$ only depends on $\mathbf{x}_i$ and $\mathbf{v}_i$.

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{v}_1 \\ \vdots \\ \boldsymbol{x}_N \\ \boldsymbol{v}_N \end{pmatrix} \qquad f(\boldsymbol{X}, t) = \begin{pmatrix} \boldsymbol{v}_1 \\ \boldsymbol{F}^1(\boldsymbol{X}, t) \\ \vdots \\ \boldsymbol{v}_N \\ \boldsymbol{F}^N(\boldsymbol{X}, t) \end{pmatrix}$$

***f* gives d/dt X, remember!**

Path through a Vector Field

- $X(t)$: path in multidimensional <u>phase space</u>

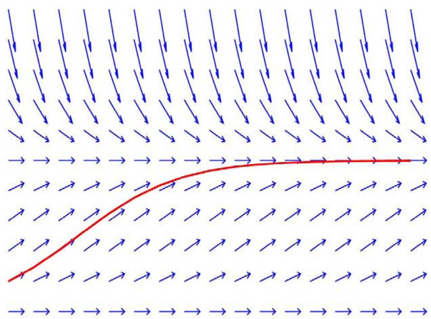


Image by MIT OpenCourseWare.

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X} = f(\boldsymbol{X}, t)$$

"When we are at state **X** at time *t*, where will **X** be after an infinitely small time interval d*t* ?"

Path through a Vector Field

- $X(t)$: path in multidimensional <u>phase space</u>

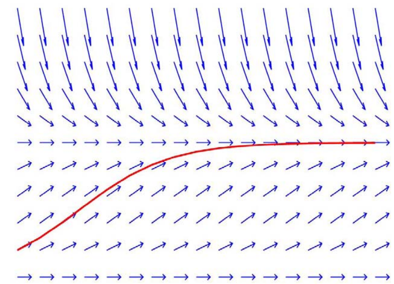


Image by MIT OpenCourseWare.

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X} = f(\boldsymbol{X}, t)$$

"When we are at state **X** at time *t*, where will **X** be after an infinitely small time interval d*t* ?"

- f=d/dt $\boldsymbol{X}$ is a vector that sits at each point in phase space, pointing the direction.

Integration Algorithm 1

Calculating Particle State from Forces: First attempt

- Use forces to update velocity: $\vec{v}(t+h) = \vec{v}(t) + \frac{h}{m}\vec{f}(t)$
- Use old velocity to update position: $\vec{x}(t+h) = \vec{x}(t) + h\vec{v}(t)$

Issues

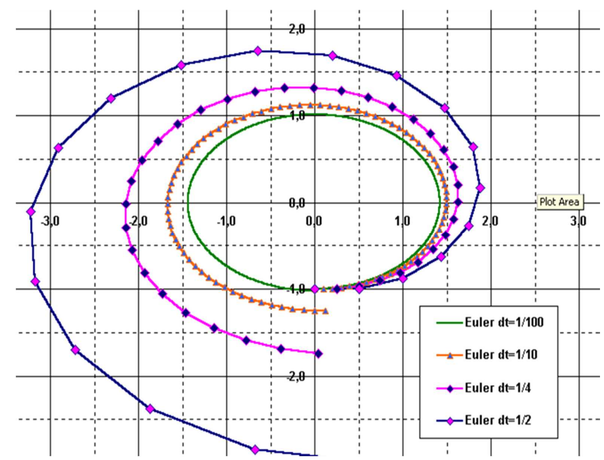
- Unstable in certain cases!
- Reducing time step can help, but this becomes computationally expensive
- Error is $O(h^2)$ per step (and accumulates!). Error is $O(h)$ globally.

This technique is called Forward (Explicit) Euler Integration

Example: circle

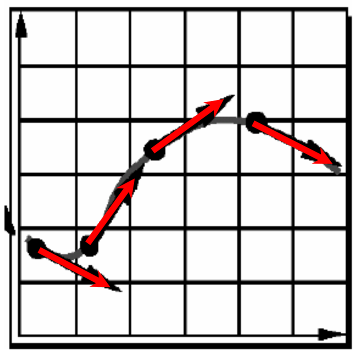
Comparison Euler, Step Sizes

Euler quality is proportional to d*t*



Intuitive Solution: Take Steps

- Current state **X**
- Examine f(**X**,t) at (or near) current state
- Take a step to new value of **X**



$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X} = f(\boldsymbol{X},t)$$

$$\Rightarrow \text{“}\mathrm{d}\boldsymbol{X} = \mathrm{d}t\, f(\boldsymbol{X},t)\text{”}$$

f = d/dt ***X*** is a vector that sits at each point in phase space, pointing the direction.

Euler's Method

- Simplest and most intuitive
- Pick a **step size** *h*
- Given $\mathbf{X}_0=\mathbf{X}(t_0)$, take step:

$$t_1 = t_0 + h$$
$$\mathbf{X}_1 = \mathbf{X}_0 + h\, f(\mathbf{X}_0, t_0)$$

- Piecewise-linear approximation to the path
- **Basically, just replace d*t* by a small but finite number**

Euler, Visually

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X} = f(\boldsymbol{X},t)$$

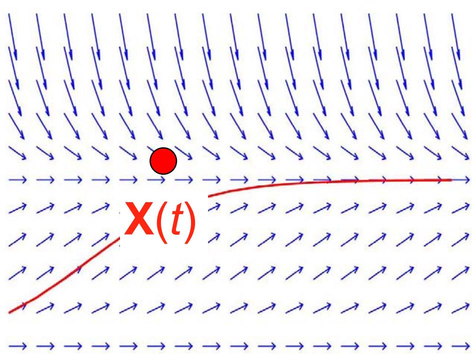


Image by MIT OpenCourseWare.

Euler, Visually

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X} = f(\boldsymbol{X},t)$$

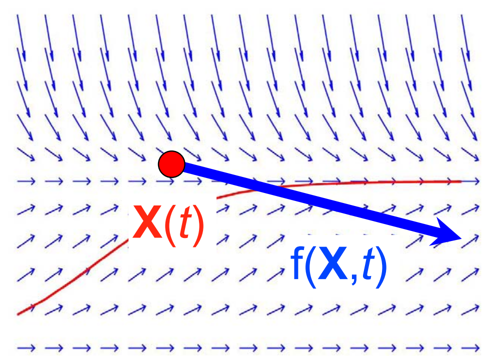


Image by MIT OpenCourseWare.

Euler, Visually

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X} = f(\boldsymbol{X}, t)$$

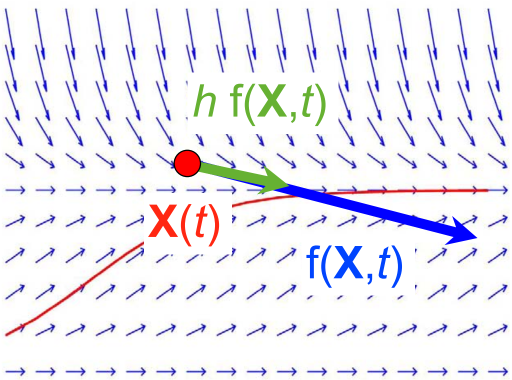


Image by MIT OpenCourseWare.

Euler, Visually

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X} = f(\boldsymbol{X}, t)$$

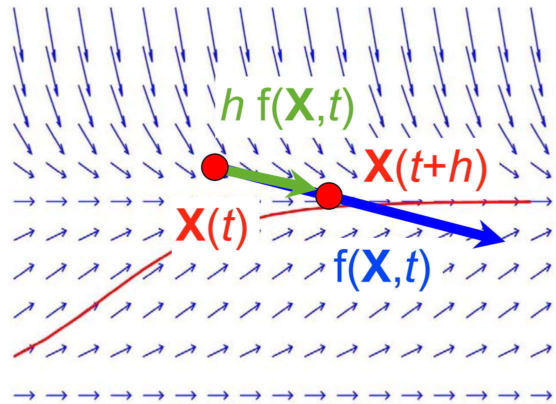


Image by MIT OpenCourseWare.

Integration Algorithm 2

Another attempt

- Update velocity with forces at next time step: $\vec{v}(t+h) = \vec{v}(t) + \frac{h}{m}\vec{f}(t+h)$
- Use new velocity to update position: $\vec{x}(t+h) = \vec{x}(t) + h\vec{v}(t+h)$

Benefits

- Unconditionally stable if the system is linear!

Issues

- Solving for $\vec{f}(t+h)$ is often expensive
- Can introduce artificial viscous damping
- Error is still $O(h^2)$ per step

This technique is called Backward (Implicit) Euler Integration

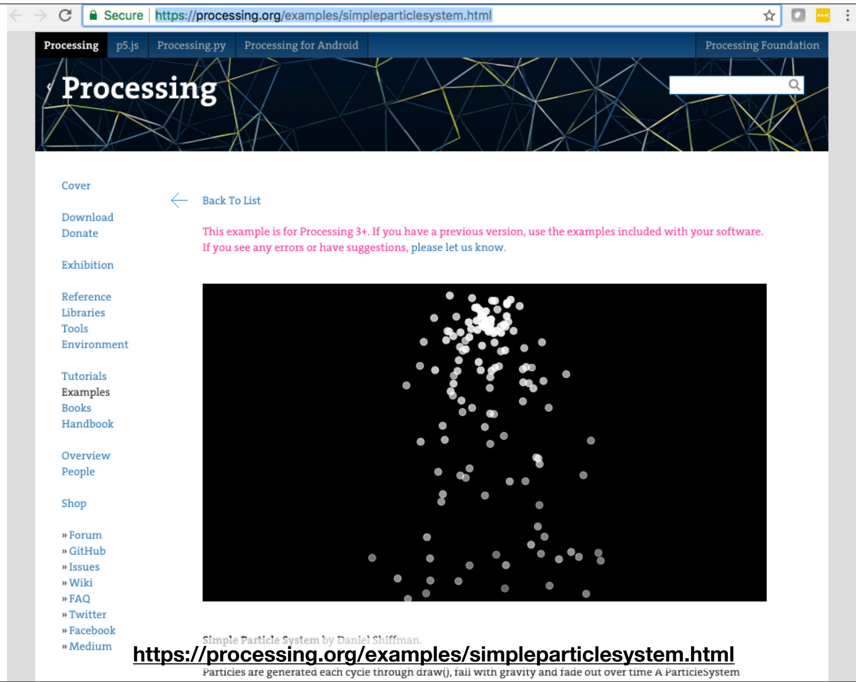
Another Simple Example: Sprinkler

```
list<Particle> PL;
spread = 0.1; //how random the velocity is

 //add k particles to the list
  for (int i=0; i<k; i++) {
    Particle p;
    p->position = Vec3(0,0,0);
    p->velocity = Vec3(0,0,1) + spread*Vec3(rand(), rand(), rand());
    PL->add(p);
  }

for (each time step) {
  for (each particle p in PL) {
    p->position += p->velocity*dt; //dt: time step
    p->velocity -= g*dt; //g: gravitation constant
  }
}
```

https://webglfundamentals.org/webgl/lessons/webgl-qna-how-to-process-particle-positions.html



https://processing.org/examples/simpleparticlesystem.html

Binary, *n*-ary Forces

Much more interesting behaviors to be had from particles that interact

Simplest: binary forces, e.g. springs

$$\vec{f}_i(\vec{x}_i, \vec{x}_j) = -k_s(\|\vec{x}_i - \vec{x}_j\| - r_{ij})\frac{\vec{x}_i - \vec{x}_j}{\|\vec{x}_i - \vec{x}_j\|}$$

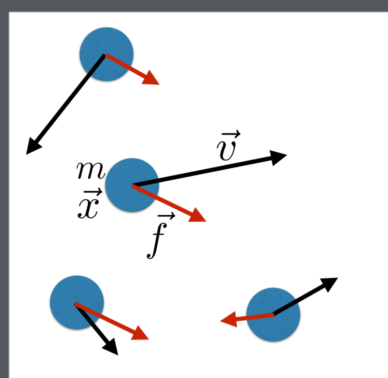
Nice example project with mass-spring systems:

- https://vimeo.com/73188339

More sophisticated models for deformable things use forces relating 3 or more particles

Particle System Setup, Revisited

```
class Particle {
  float mass;
  Vector3 position;
  Vector3 velocity;
  Vector3 force;
};
```



Basic Algorithm

1) Clear forces from previous calculations

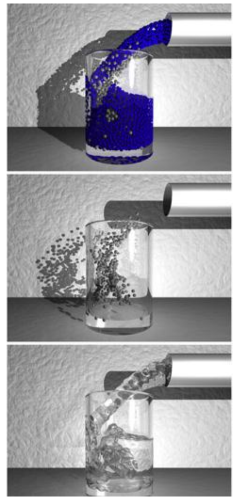
2) Calculate/accumulate forces for each particle

3) Solve for particle's state (position, velocity) for the next time step h

Generalizations

Müller et al. 2005

- It's not all hacks:
  Smoothed Particle Hydrodynamics (SPH)
  - A family of "real" particle-based fluid simulation techniques.
  - Fluid flow is described by the Navier-Stokes Equations, a nonlinear partial differential equation (PDE)
    - SPH discretizes the fluid as small packets (particles!), and evaluates pressures and forces based on them.



https://youtu.be/Qe9qSLYK5q4, 1991

https://dl.acm.org/citation.cfm?id=357320