上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Game Design & Development

Game Physics

Outline

- Motivation of Game Physics
- Particle systems & ODE
- Hair modeling and rendering

Physically Based Simulation in Games

Half Life 2

Max Payne 2

Fuel

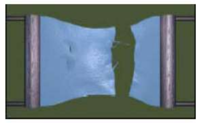
Black

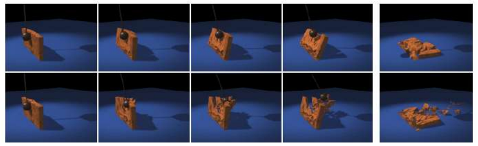
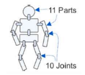
Introduction to Game Physics

- Goal: Simulate the motion of objects that obey physical laws
- Traditional Game Physics
  —Collisions for Game Physics
  —Particle system
  —Rigid body dynamics
  —Flexible body dynamics

Advanced Topics in Game Physics

- Fluid Dynamics
- Car Dynamics
- Rag-doll Physics
- Fracture Mechanics



Funny Examples

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Particle Systems & ODE

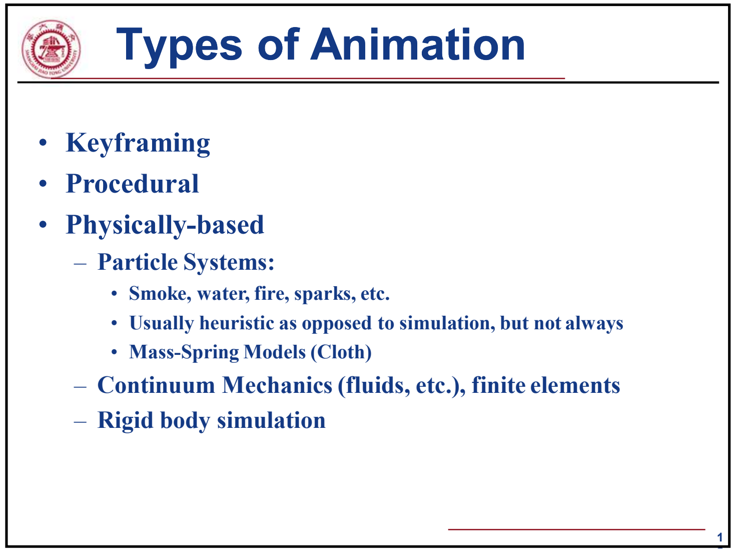
Numeric Integration

Particle Systems

- Single particles are very simple
- Large groups can produce interesting effects
- Supplement basic ballistic rules
  - Collisions
  - Interactions
  - Force fields
  - Springs
  - Others...

Karl Sims, SIGGRAPH 1990

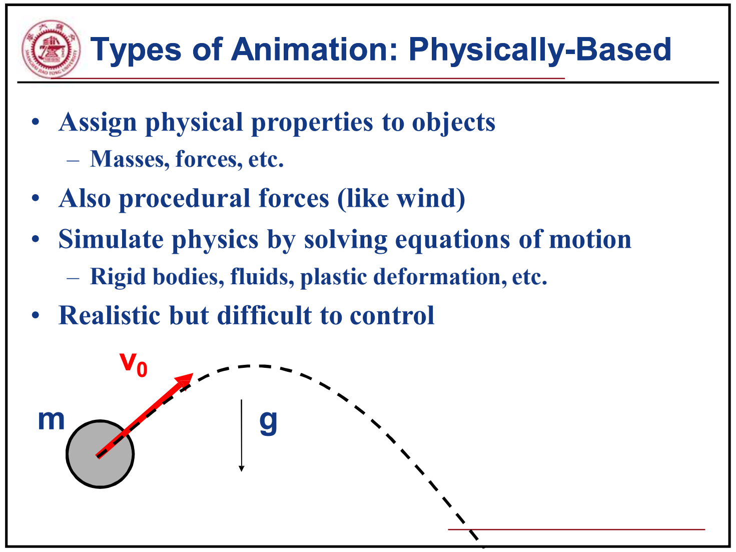
Feldman, Klingner, O'Brien, SIGGRAPH 2005



Types of Animation

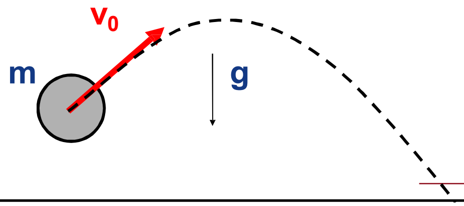
- Keyframing
- Procedural
- Physically-based
  - Particle Systems:
    - Smoke, water, fire, sparks, etc.
    - Usually heuristic as opposed to simulation, but not always
    - Mass-Spring Models (Cloth)
  - Continuum Mechanics (fluids, etc.), finite elements
  - Rigid body simulation

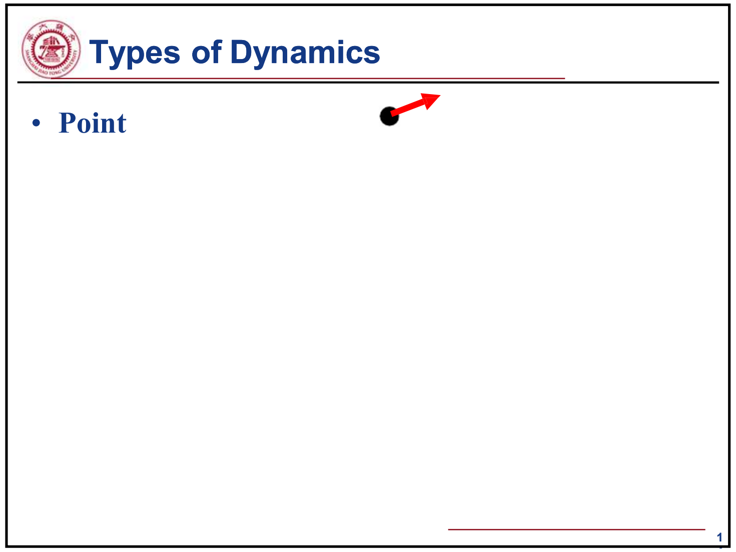
1



Types of Animation: Physically-Based

- Assign physical properties to objects
  - Masses, forces, etc.
- Also procedural forces (like wind)
- Simulate physics by solving equations of motion
  - Rigid bodies, fluids, plastic deformation, etc.
- Realistic but difficult to control

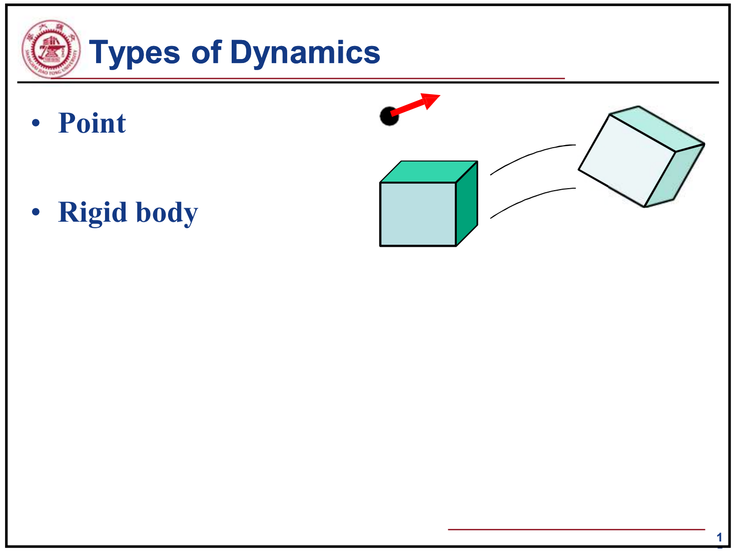




Types of Dynamics

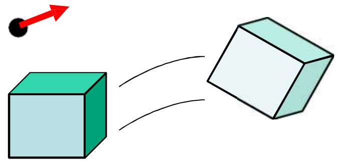
- Point

1



Types of Dynamics

- Point
- Rigid body

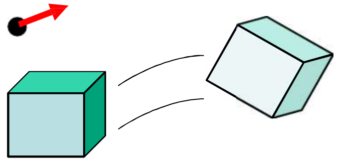


1

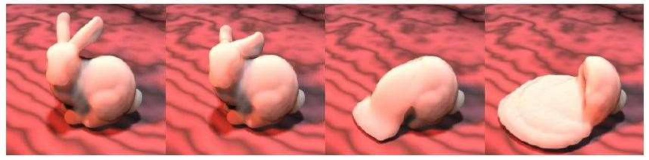
Types of Dynamics

- Point

- Rigid body



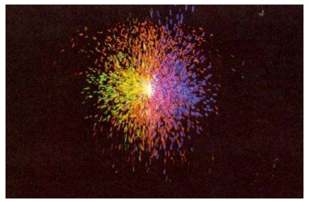
- Deformable body
  (include clothes, fluids, smoke,

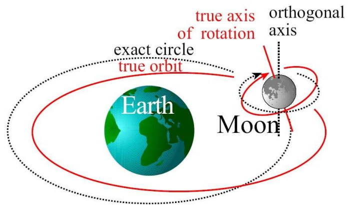


1

Particles: Point Dynamics

- Lots of points!
- Particles systems
  - Borderline between procedural and physically- based





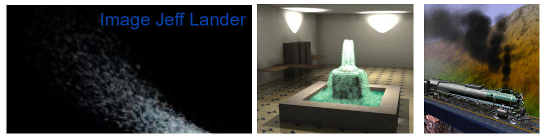
1

Particle Systems Overview

- **Emitters** generate tons of "particles"
  - Sprinkler, waterfall, chimney, gun muzzle, exhaust pipe, etc.

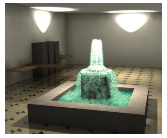
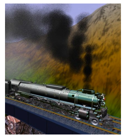
Particle Systems Overview

- **Emitters** generate tons of "particles"
- Describe the external **forces** with a force field
  - E.g., gravity, wind



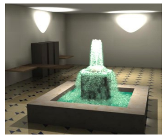
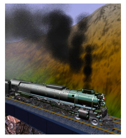
Particle Systems Overview

- **Emitters** generate tons of "particles"
- Describe the external **forces** with a force field
- **Integrate** the laws of mechanics (ODEs)
  - Makes the particles move

Particle Systems Overview

- **Emitters** generate tons of "particles"
- Describe the external **forces** with a force field
- **Integrate** the laws of mechanics (ODEs)
- In the simplest case, each particle is **independent**

Generalizations

- More advanced versions of behavior
  - flocks, crowds
- Forces between particles
  - Not independent any more

Generalizations

- Mass-spring and deformable surface dynamics
  - surface represented as a set of points
  - forces between neighbors keep the surface coherent

Image Witkin & Baraff

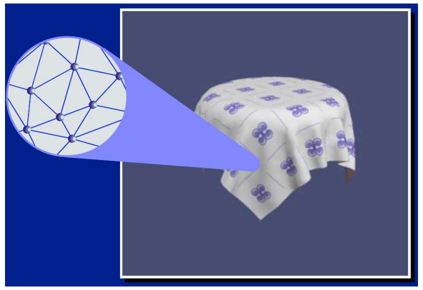
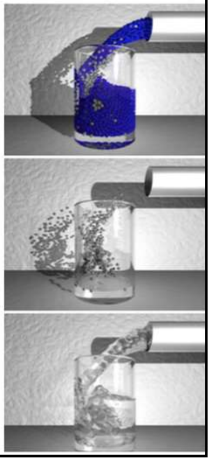


Image Michael Kass

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.



Jos Stam

Simple particle system: sprinkler

```
PL: linked list of particle = empty;
spread=0.1;//how random the initial velocity is
colorSpread=0.1; //how random the colors are
For each time step
    Generate k particles
        p=new particle();
        p->position=(0,0,0);
        p->velocity=(0,0,1)+spread*(rnd(), rnd(), rnd());
        p.color=(0,0,1)+colorSpread*(rnd(), rnd(),rnd());
        PL->add(p);
    For each particle p in PL
        p->position+=p->velocity*dt; //dt: time step
        p->velocity-=g*dt; //g: gravitation constant
        glColor(p.color);
        glVertex(p.position);
```

Path forward

- Basic particle systems are simple hacks
- Extend to physical simulations, e.g. clothes
- For this, we need to understand numerical integration

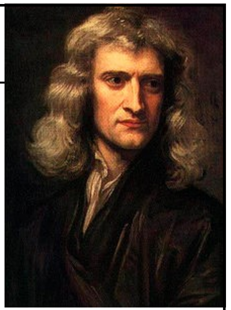
Integrating ODEs

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

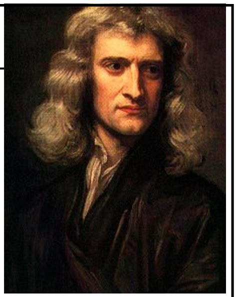


- Point mass: 2nd order ODE

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

- Position ***x*** and force ***F*** are vector quantities
  - We know ***F*** and *m*, want to solve for ***x***
- You've all seen this a million times before

Reduction to 1st Order



- Point mass: 2nd order ODE

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

- 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

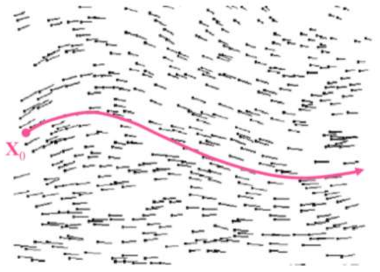
- We have N point masses
  - Let's just stack all **x**s and **v**s in a big vector of length 6N
  - **F**^{i} denotes the force on particle *i*
    - When particles don't interact, **F**^{i} only depends on **x**$_i$ and **v**$_i$.

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

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

Path through a Vector Field

- *X(t)*: path in multidimensional phase space

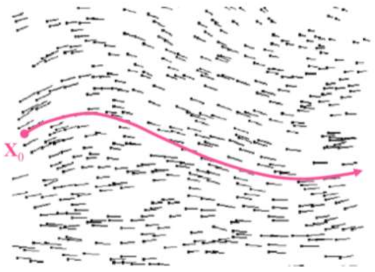


$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X} = f(\boldsymbol{X}, t)$$

"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

- $\boldsymbol{X(t)}$*:* path in multidimensional phase space



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

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

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

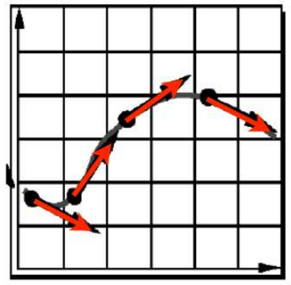
Intuitive Solution: Take Steps

Integrating ODEs !

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

Euler's Method

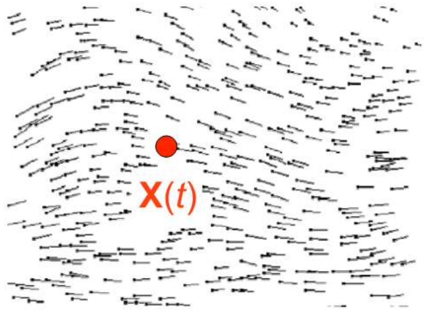
- Simplest and most intuitive
- Pick a **step size** *h*
- Given $\mathbf{X}_0 = \mathbf{X}(t_0)$, take step:

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

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

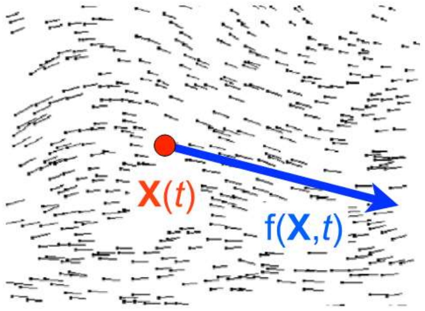
Euler, Visually

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



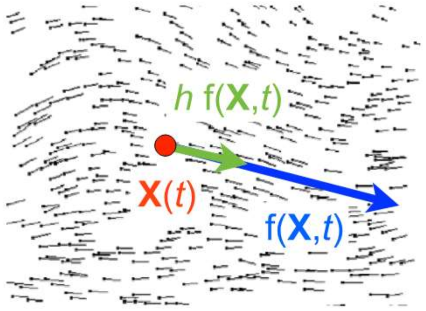
Euler, Visually

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



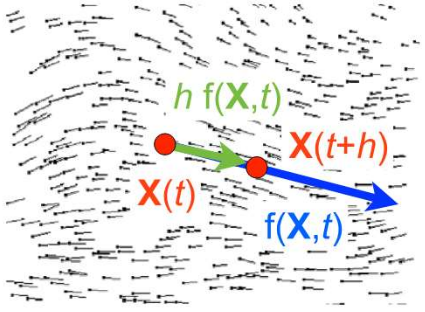
Euler, Visually

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



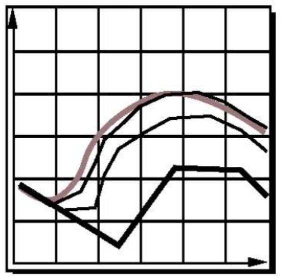
Euler, Visually

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



Effect of Step Size

- Step size controls accuracy
- Smaller steps more closely follow curve
  - May need to take many small steps per frame
  - Properties of $f(\mathbf{X}, t)$ determine this (more later)



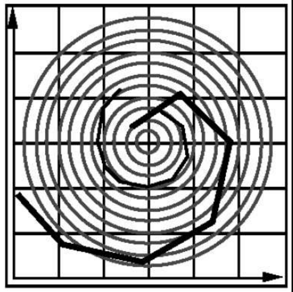
Euler's method: Inaccurate

- Moves along tangent; can leave solution curve, e.g.:

$$f(\mathbf{X},t)=\begin{pmatrix}-y\\x\end{pmatrix}$$

- Exact solution is circle:

$$\mathbf{X}(t)=\begin{pmatrix}r\cos(t+k)\\r\sin(t+k)\end{pmatrix}$$



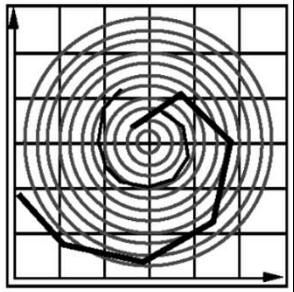
Euler's method: Inaccurate

- Moves along tangent; can leave solution curve, e.g.:

$$f(\mathbf{X},t)=\begin{pmatrix}-y\\x\end{pmatrix}$$

- Exact solution is circle:

$$\mathbf{X}(t)=\begin{pmatrix}r\cos(t+k)\\r\sin(t+k)\end{pmatrix}$$



- Euler spirals outward no matter how small *h* is
  - will just diverge more slowly

Euler's method: Not Always Stable

- "Test equation" $f(x,t)=-kx$

Euler's method: Not Always Stable

- "Test equation" $f(x,t)=-kx$

- Exact solution is a decaying exponential:

$$x(t)=x_0e^{-kt}$$

Euler's method: Not Always Stable

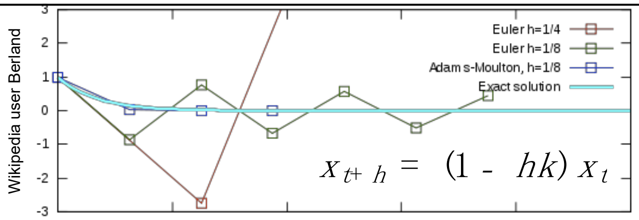
- "Test equation" $f(x,t) = -kx$
- Exact solution is a decaying exponential:

$$x(t) = x_0 e^{-kt}$$

- Let's apply Euler's method:

$$\begin{aligned} x_{t+h} &= x_t + h\,f(x_t,t) \\ &= x_t - hkx_t \\ &= \boxed{(1-hk)\,x_t} \end{aligned}$$

Euler's method: Not Always Stable



- Limited step size!
  - When $0 < (1-hk) < 1 \Leftrightarrow h < 1/k$ things are fine, the solution decays
  - When $-1 < (1-hk) < 0 \Leftrightarrow 1/k < h < 2/k$ we get oscillation
  - When $(1-hk) < -1 \Leftrightarrow h > 2/k$ things explode!

Euler's method: Not Always Stable

If *k* is big,
h must be small!

- Limited step size!
  - When $0 < (1-hk) < 1 \Leftrightarrow h < 1/k$ things are fine, the solution decays
  - When $-1 < (1-hk) < 0 \Leftrightarrow 1/k < h < 2/k$ we get oscillation
  - When $(1-hk) < -1 \Leftrightarrow h > 2/k$ things explode!

Analysis: Taylor series

- Expand exact solution **X**(*t*)

$$\mathbf{X}(t_0+h) = \mathbf{X}(t_0) + h\left(\tfrac{d}{dt}\mathbf{X}(t)\right)\Big|_{t_0} + \tfrac{h^2}{2!}\left(\tfrac{d^2}{dt^2}\mathbf{X}(t)\right)\Big|_{t_0} + \tfrac{h^3}{3!}(\cdots) + \cdots$$

- Euler's method approximates:

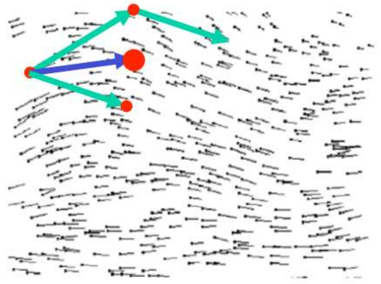
$$\mathbf{X}(t_0+h) = \mathbf{X}_0 + h\,f(\mathbf{X}_0,t_0) \quad \ldots + O(h^2)\text{ error}$$

$$h \to h/2 \;\Rightarrow\; error \to error/4 \text{ per step} \times \text{twice as many steps} \to error/2$$

- First-order method: Accuracy varies with *h*
- To get 100x better accuracy need 100x more steps

Can we do better?

- Problem: *f* varies along our Euler step
- Idea 1: look at *f* at the arrival of the step and compensate for variation

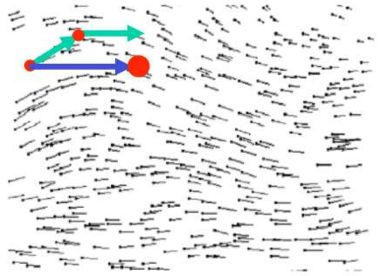


2nd Order Methods

- Let $\boxed{\begin{array}{l} f_0 = f(\mathbf{X}_0, t_0) \\ f_1 = f(\mathbf{X}_0 + h f_0, t_0 + h) \end{array}}$
- Then $\boxed{\mathbf{X}(t_0 + h) = \mathbf{X}_0 + \frac{h}{2}(f_0 + f_1)} + O(h^3)$
- This is the *trapezoid method*
- Note! What we mean by "2nd order" is that the error goes down with h^2, not h – the equation is still 1st order!

Can we do better?

- Problem: *f* has varied along our Euler step
- Idea 2: look at *f* after a smaller step, use that value for a full step from initial position



2nd Order Methods cont'd

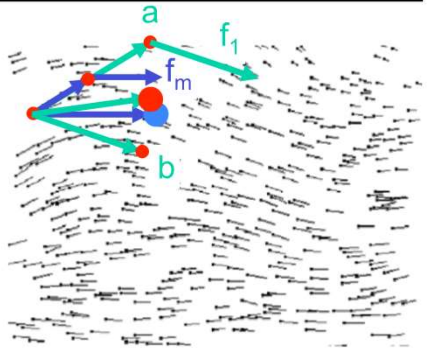
- This translates to...

$$\boxed{\begin{array}{l} f_0 = f(\mathbf{X}_0, t_0) \\ f_m = f(\mathbf{X}_0 + \frac{h}{2} f_0, t_0 + \frac{h}{2}) \end{array}}$$

- and we get $\boxed{\mathbf{X}(t_0 + h) = \mathbf{X}_0 + h\, f_m} + O(h^3)$
- This is the *midpoint method*
  - Analysis omitted again, but it's not very complicated

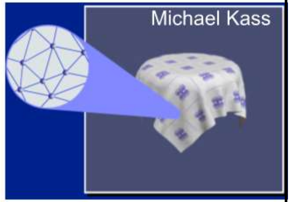
Comparison

- 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



Mass-Spring Modeling

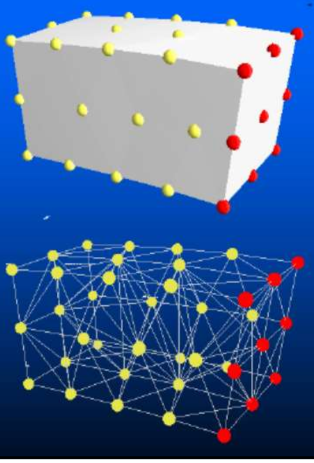
- Beyond pointlike objects: strings, cloth, hair, etc.
- Interaction between particles
  - Create a network of spring forces that link pairs of particles



- First, slightly hacky version of cloth simulation
- Then, some motivation/intuition for *implicit integration*

Mass-spring systems

- Simple extension of particle systems
- Virtual "springs" impart forces to connected particles



Examples of Deformable Objects

- 1d: Ropes, hair
- 2d: Cloth, clothing
- 3d: Fat, tires, organs

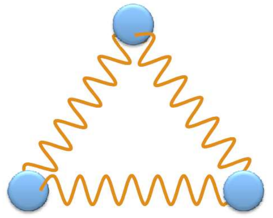
PhysX by NVIDIA SIGGRAPH2008

Dimensionality

- Every real object is 3d
- Approximated object with lower dimentional models if possible
- Dimension reduction substantially saves simulation time

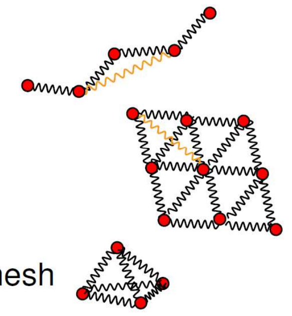
PhysX by NVIDIA — SIGGRAPH2008

Mass Spring Systems



Mass Spring Meshes

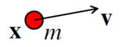
- Rope: chain
  - Additional springs for bending and torsional resistance needed
- Cloth: triangle mesh
  - Additional springs for bending restistance needed
- Soft body: tetrahedral mesh



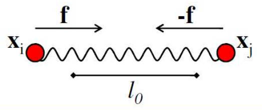
PhysX by NVIDIA — SIGGRAPH2008

Mass Spring Physics

- Mass point: mass m, position $\mathbf{x}$, velocity $\mathbf{v}$



- Springs:



$$\mathbf{f} = \frac{\mathbf{x}_j - \mathbf{x}_i}{|\mathbf{x}_j - \mathbf{x}_i|}\left[k_s(|\mathbf{x}_j - \mathbf{x}_i| - l_o) + k_d(\mathbf{v}_j - \mathbf{v}_i)\cdot\frac{\mathbf{x}_j - \mathbf{x}_i}{|\mathbf{x}_j - \mathbf{x}_i|}\right]$$

- Scalars k_s, k_d, stretching, damping coefficients

PhysX by NVIDIA — SIGGRAPH2008

Verlet Integration

Taylor expansion of $r_i(t)$

$$r_i(t_0+\Delta t) = r_i(t_0) + v_i(t_0)\Delta t + \frac{1}{2}a_i(t_0)\Delta t^2 + \dots$$

$$r_i(t_0 - \Delta t) = r_i(t_0) - v_i(t_0)\Delta t + \frac{1}{2}a_i(t_0)\Delta t^2 + \dots$$

$$a = t_0 \qquad x = t_0 + \Delta t \qquad x - a = t_0 + \Delta t - t_0 = \Delta t$$

$$a = t_0 \qquad x = t_0 - \Delta t \qquad x - a = t_0 - \Delta t - t_0 = -\Delta t$$

Taylor expansion of $r_i(t)$

$$r_i(t_0+\Delta t) = r_i(t_0) + v_i(t_0)\Delta t + \frac{1}{2}a_i(t_0)\Delta t^2 + \dots$$

$$+\left[\, r_i(t_0-\Delta t) = r_i(t_0) - v_i(t_0)\Delta t + \frac{1}{2}a_i(t_0)\Delta t^2 + \dots \right]$$

$$\downarrow$$

$$r_i(t_0-\Delta t) + r_i(t_0+\Delta t) = 2r_i(t_0) - v_i(t_0)\Delta t + v_i(t_0)\Delta t + a_i(t_0)\Delta t^2 + \dots$$

Taylor expansion of $r_i(t)$

$$r_i(t_0+\Delta t) = r_i(t_0) + v_i(t_0)\Delta t + \frac{1}{2}a_i(t_0)\Delta t^2 + \dots$$

$$+\left[\, r_i(t_0-\Delta t) = r_i(t_0) - v_i(t_0)\Delta t + \frac{1}{2}a_i(t_0)\Delta t^2 + \dots \right]$$

$$\downarrow$$

$$r_i(t_0-\Delta t) + r_i(t_0+\Delta t) = 2r_i(t_0) \cancel{- v_i(t_0)\Delta t + v_i(t_0)\Delta t} + a_i(t_0)\Delta t^2 + \dots$$

$$\downarrow$$

$$r_i(t_0+\Delta t) = 2r_i(t_0) - r_i(t_0-\Delta t) + a_i(t_0)\Delta t^2 + \dots$$

Taylor expansion of $r_i(t)$

$$r_i(t_0+\Delta t) = r_i(t_0) + v_i(t_0)\Delta t + \frac{1}{2}a_i(t_0)\Delta t^2 + \dots$$

$$+\left[\, r_i(t_0-\Delta t) = r_i(t_0) - v_i(t_0)\Delta t + \frac{1}{2}a_i(t_0)\Delta t^2 + \dots \right]$$

$$\downarrow$$

$$r_i(t_0-\Delta t) + r_i(t_0+\Delta t) = 2r_i(t_0) \cancel{- v_i(t_0)\Delta t + v_i(t_0)\Delta t} + a_i(t_0)\Delta t^2 + \dots$$

$$\downarrow$$

$$\boxed{r_i(t_0+\Delta t) = 2\underbrace{r_i(t_0)}_{\text{Positions at } t_0} - \underbrace{r_i(t_0-\Delta t)}_{\text{Positions at } t_0-\Delta t} + \underbrace{a_i(t_0)}_{\text{Accelerations at } t_0}\Delta t^2 + \dots}$$

Verlet central difference method

$$r_i(t_0+\Delta t) = \underbrace{2r_i(t_0)}_{\text{Positions at } t_0} - \underbrace{r_i(t_0-\Delta t)}_{\text{Positions at } t_0-\Delta t} + \underbrace{a_i(t_0)}_{\text{Accelerations at } t_0}\Delta t^2 + \ldots$$

How to obtain accelerations? $f_i = ma_i$ $a_i = f_i / m$

Need forces on atoms!

Implicit methods

Explicit Euler: $Y_{new} = Y_0 + hf(Y_0)$

Implicit Euler: $Y_{new} = Y_0 + hf(Y_{new})$

Solving for Y_{new} such that f, at time $t_0 + h$, points directly back at Y_0

Implicit methods

Our goal is to solve for $\mathbf{Y}_{new}$ such that

$$\mathbf{Y}_{new} = \mathbf{Y}_0 + hf(\mathbf{Y}_{new})$$

Approximating $f(\mathbf{Y}_{new})$ by linearizing $f(\mathbf{Y})$

$f(\mathbf{Y}_{new}) = f(\mathbf{Y}_0) + \Delta\mathbf{Y} f'(\mathbf{Y}_0)$, where $\Delta\mathbf{Y} = \mathbf{Y}_{new} - \mathbf{Y}_0$

$$\mathbf{Y}_{new} = \mathbf{Y}_0 + \cancel{h}f(\mathbf{Y}_0) + h\Delta\mathbf{Y} f'(\mathbf{Y}_0)$$

$$\Delta\mathbf{Y} = \left(\frac{1}{h}\mathbf{I} - f'(\mathbf{Y}_0)\right)^{-1} f(\mathbf{Y}_0)$$

$f(\mathbf{Y}, t) = \dot{\mathbf{Y}}(t)$

$f(\mathbf{Y}, t)' = \frac{\partial f}{\partial \mathbf{Y}}$

Implicit vs. explicit

correct solution: $x(h) = e^{-hk}$

explicit Euler: $x(h) = 1 - hk$

implicit Euler: $x(h) = \frac{1}{1 + hk}$

$\dot{x}(h) = -kx(h)$
$x(0) = 1$