Mass-Spring Systems

A Basic Tool for Modeling Deformable Objects

Part 2

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

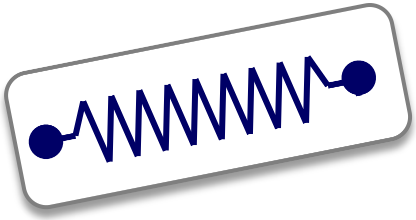
Disney Research

Computer Graphics Laboratory ETH Zurich

Outline

- Numerical time integration (*continued*)
  - Explicit vs. Implicit Euler
  - Solving nonlinear systems: Newton's method
  - The semi-implicit Euler method
  - Solving sparse linear systems
  - Integration Schemes for higher order ODEs
- Towards practical mass spring systems
- Constraints

Mass-Spring Systems



Mass-Spring Systems

- **Forces**: elastic springs, point damping, gravity
- **Dynamics**: equations of motion, system of 2nd order ODEs
- **Temporal discretization**: solve 1st order ODE numerically

Numerical Time Integration

- **Explicit methods**: Euler, Heun, Midpoint, Runge-Kutta 4
- **Criteria**: accuracy, stability
- **Model problem**: test equation

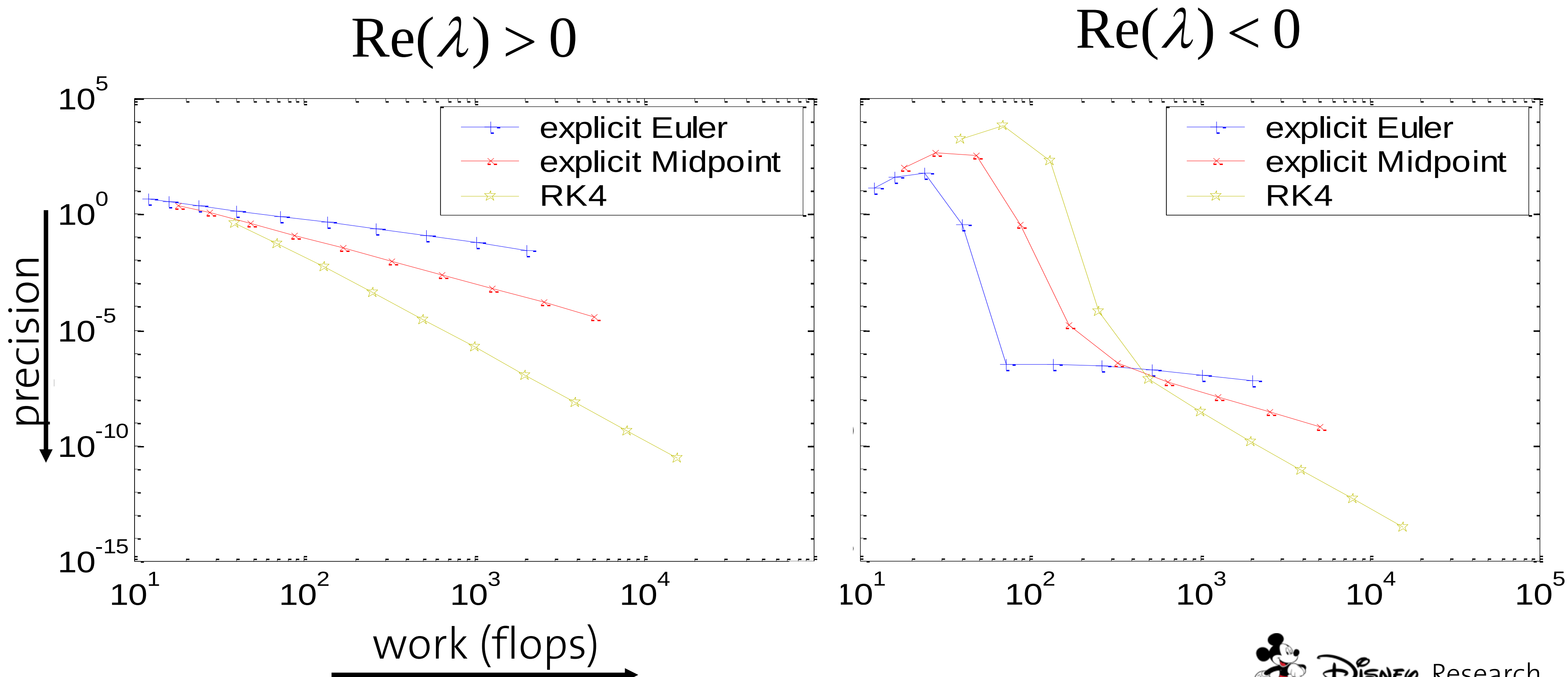
Explicit Methods

Test equation

$$y'(t) = \lambda y(t), \quad \lambda \in \mathbf{C}$$

Analytical solution

$$y(t) = e^{\lambda t} y_0$$



Disney Research

Stiff Problems

Observations from test equation: explicit methods

- require very small time steps for stable integration
- are inefficient since step size is determined by stability, not accuracy requirements

Problems with this characteristic are termed *stiff*

Don't use explicit methods for stiff problems,
use *implicit* methods.

Disney Research

Implicit Euler

- Explicit Euler: step with slope at t_n

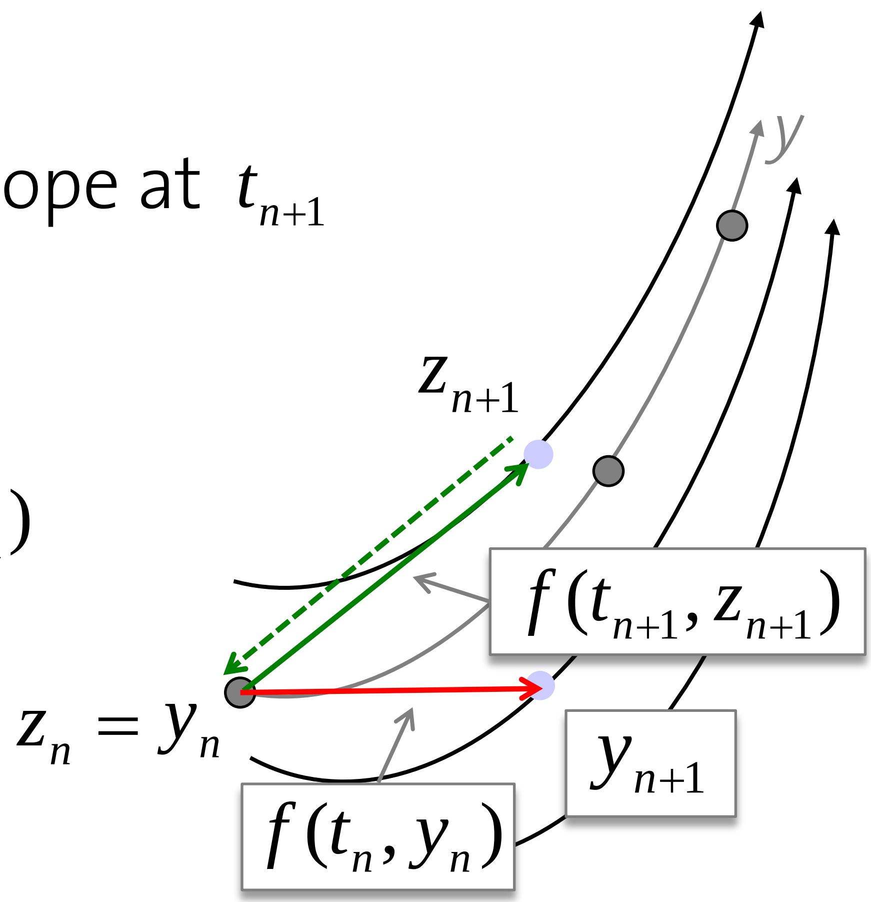
$$y_{n+1} = y_n + hf(t_n, y_n)$$

- Implicit Euler: step with slope at t_{n+1}

$$z_{n+1} = z_n + hf(t_{n+1}, z_{n+1})$$

- A.k.a. *Backward* Euler

$$z_n = z_{n+1} - hf(t_{n+1}, z_{n+1})$$



Numerical Example

Harmonic Oscillator

Explicit vs. Implicit Euler

What about stability?

- Euler step for test equation $y' = \lambda y, \quad \lambda < 0$
  - Explicit Euler $\quad y_{n+1} = y_n + h\lambda y_n \quad = y_n(1 + h\lambda)$
  - Implicit Euler $\quad y_{n+1} = y_n + h\lambda y_{n+1} = y_n(1 - h\lambda)^{-1}$
- Stability conditions for Euler
  - Explicit $\quad y_n = (1 + h\lambda)^n y_0 \quad < \infty \quad \Leftrightarrow \quad |1 + h\lambda| \quad < 1$
  - Implicit $\quad y_n = (1 - h\lambda)^{-n} y_0 \quad < \infty \quad \Leftrightarrow \quad |1 - h\lambda|^{-1} < 1$

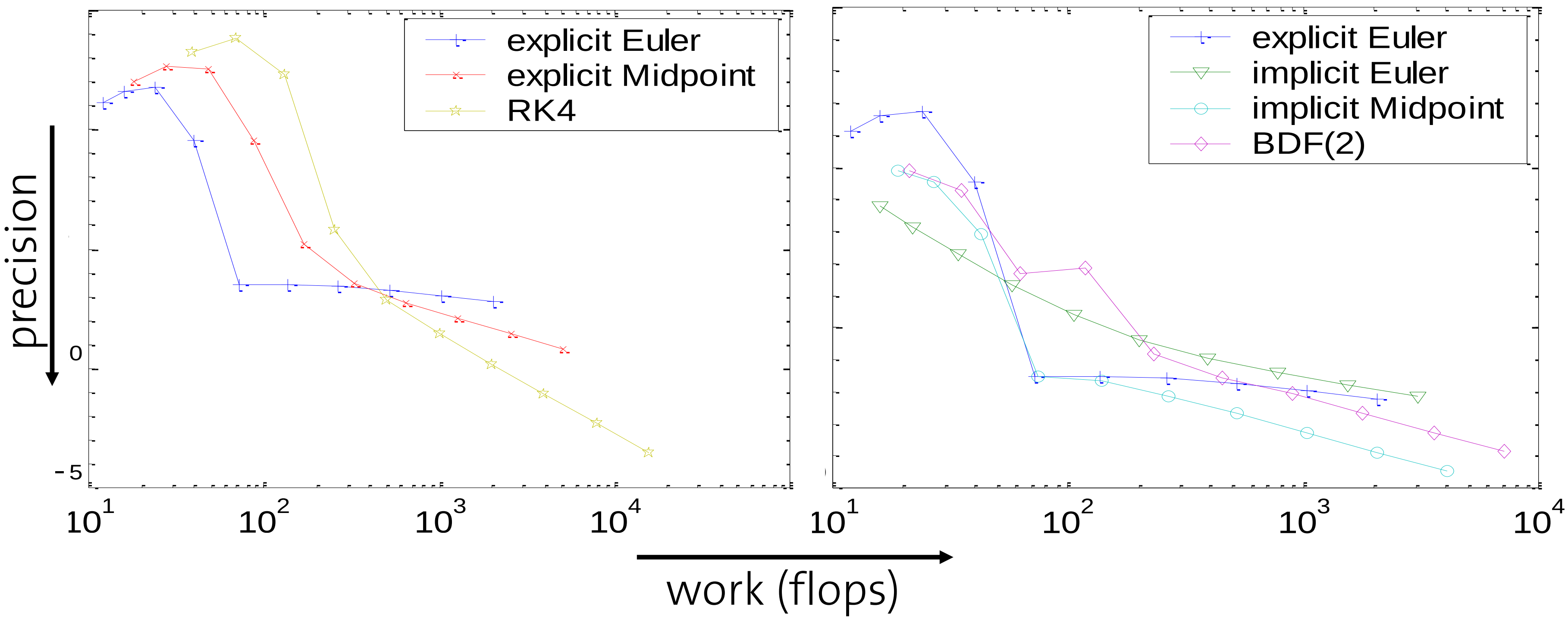
$\Rightarrow$ **Implicit Euler is stable for all $h > 0$!**

Comparing Integration Schemes

Test equation
$$y'(t) = \lambda y(t), \quad \lambda \in \mathbf{C}$$

Analytical solution
$$y(t) = e^{\lambda t} y_0$$

- $\mathrm{Re}(\lambda) < 0$ (*damped harmonic oscillator*)



Explicit vs. Implicit Euler

Explicit Euler step

$$y_{n+1} = y_n + hf(t_n, y_n)$$

Implicit Euler step

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$$

Why are these methods called like this?

- **Explicit**: all quantities are known (*given explicitly*)
- **Implicit**: y_{n+1} is unknown (*given implicitly*)

$\rightarrow$ solve (nonlinear) equation(s)!

Solving Nonlinear Equations

- Implicit Euler step $y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$

  $\rightarrow$ solve nonlinear system of equations

- Why nonlinear?

1D Spring Force $$\boxed{F = -k(l - L)}$$

3D Spring Force $$\boxed{\mathbf{F}_i = -k(|\mathbf{x}_i - \mathbf{x}_j| - L)\frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|}}$$

$$|\mathbf{x}| = \sqrt{x^2 + y^2 + z^2}$$

$\rightarrow$ linear material law, but nonlinear geometry

How do we solve nonlinear equations?

Outline

- Numerical time integration (*continued*)
  - Explicit vs. Implicit Euler
  - Solving nonlinear systems: Newton’s method
  - The semi-implicit Euler method
  - Solving sparse linear systems
  - Integration Schemes for higher order ODEs
- Towards practical mass spring systems
- Constraints

Newton's Method

Rewrite

$$0 = y_{n+1} - y_n - hf(t_{n+1}, y_{n+1}) =: g(y_{n+1})$$

Solve

$$g(y_{n+1}) = 0$$

Make initial guess

$$\tilde{y}_{n+1} = y_n$$

Define

$$\Delta y = y_{n+1} - \tilde{y}_{n+1}$$

Taylor expansion

$$g(\tilde{y}_{n+1} + \Delta y) = g(\tilde{y}_{n+1}) + g'(\tilde{y}_{n+1}) \cdot \Delta y + O(\Delta y^2) = 0$$

Linearize

$$g'(\tilde{y}_{n+1}) \cdot \Delta y = -g(\tilde{y}_{n+1})$$

Solve for Δy → **Update** → **Compute error**

$$g'(\tilde{y}_{n+1}) \cdot \Delta y = -g(\tilde{y}_{n+1}) \qquad \tilde{y}_{n+1} = \tilde{y}_{n+1} + \Delta y \qquad r = g(\tilde{y}_{n+1})$$

Repeat until $|r|$ **small enough**

Newton's Method Visually

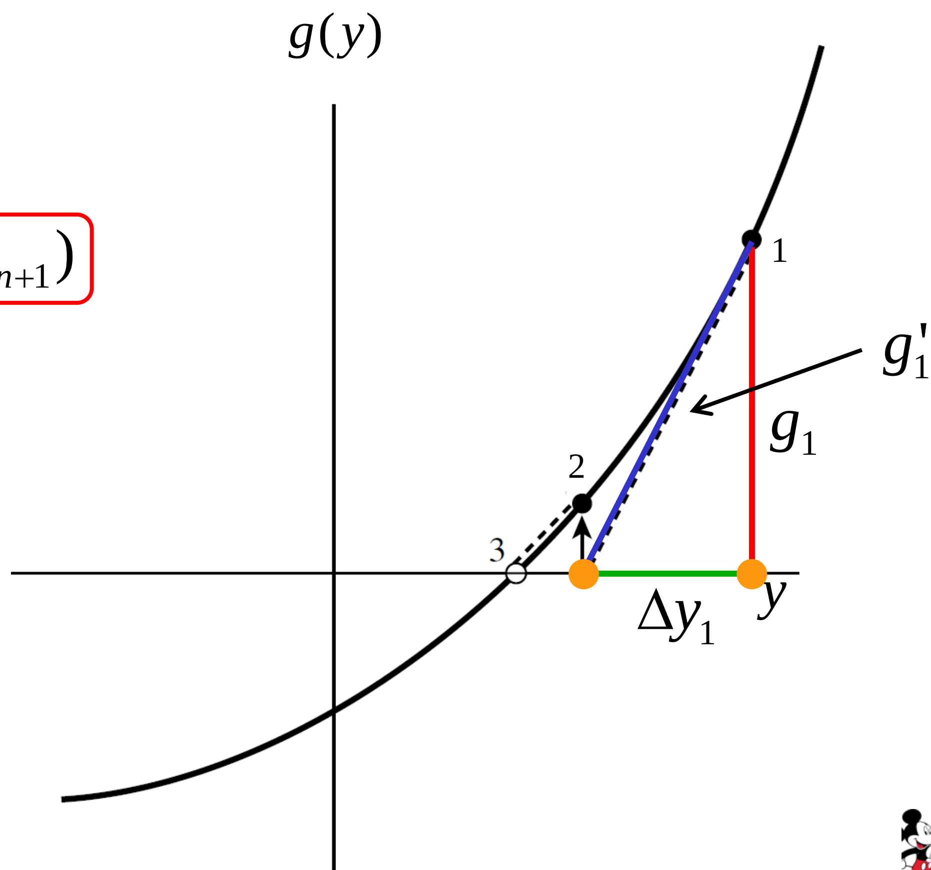
Solve for $g(y) = 0$

Solve for Δy

$$\boxed{g'(\tilde{y}_{n+1})} \cdot \boxed{\Delta y} = -\boxed{g(\tilde{y}_{n+1})}$$

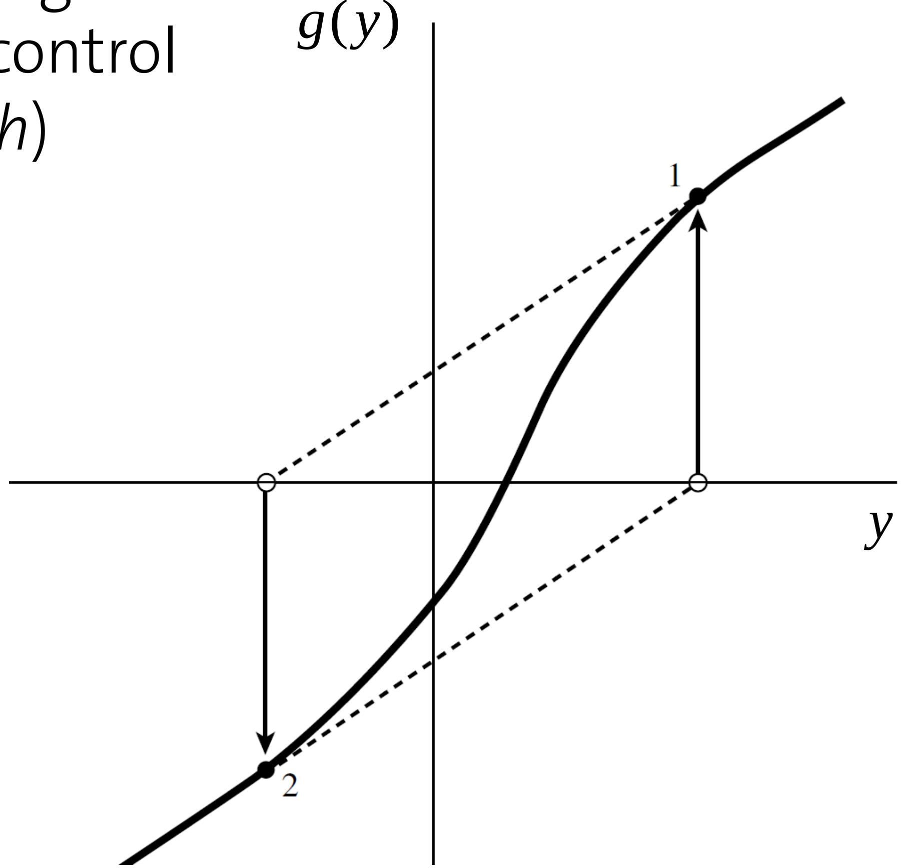
Update

$$\tilde{y}_{n+1} = \tilde{y}_{n+1} + \Delta y$$



Newton's Method: Problems

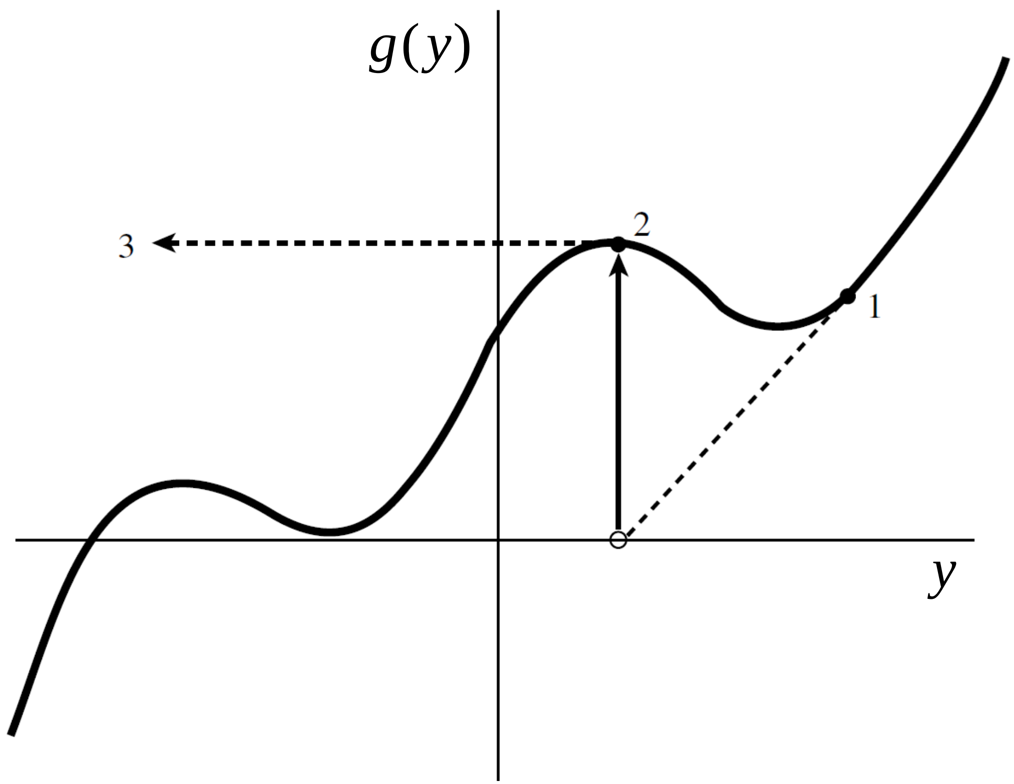
Problem: step too large
Solution: step-size control
(*line search*)



Disney Research

Newton's Method: Problems

Problem: g has local extrema or saddle points
Solution: problem-dependent (*more difficult*)



Disney Research

Newton’s Method

Powerful, but has to be used with care

– Iterative method, requires solution of linear system in each iteration $\leftarrow$ *expensive*

– Robust implementation needs advanced strategies (*line search*) $\leftarrow$ *expensive*

Can we sidestep nonlinearities?

Outline – Part 1

- Recap of last lecture
- Numerical time integration (*continued*)
  - Explicit vs. implicit Euler revisited
  - Solving nonlinear systems: Newton's method
  - The semi-implicit Euler method
  - Solving sparse linear systems
  - Integration Schemes for higher order ODEs

Disney Research

Semi-Implicit Euler

Implicit Euler

$$\mathbf{v}(t_i + h) = \mathbf{v}(t_i) + h \cdot \mathbf{M}^{-1}\mathbf{F}(t_i + h)$$

$$\mathbf{x}(t_i + h) = \mathbf{x}(t_i) + h \cdot \mathbf{v}(t_i + h)$$

Forces

$$\mathbf{F}(t_i + h) = \mathbf{F}\big(\mathbf{x}(t_i + h), \mathbf{v}(t_i + h)\big)$$

Semi-implicit: linearize forces at current state

$$\mathbf{F}(t_i + h) \approx \mathbf{F}(t_i) + \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{t=t_i} \cdot \big(\mathbf{x}(t_i + h)_{\,i} - \mathbf{x}(t_i)\big) + \left.\frac{\partial \mathbf{F}}{\partial \mathbf{v}}\right|_{t=t_i} \cdot \big(\mathbf{v}(t_i + h) - \mathbf{v}(t_i)\big) \quad + \left.\frac{\partial \mathbf{F}}{\partial t}\right|_{t=t_i} \cdot h$$

$\frac{\partial \mathbf{F}}{\partial \mathbf{x}}, \frac{\partial \mathbf{F}}{\partial \mathbf{v}}$ are (*3nx3n*) Jacobian matrices
(*derivatives of a vector w.r.t. a vector*)

Semi-Implicit Euler

For mass-spring systems

- $\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$ is given by the spring force
- $\frac{\partial \mathbf{F}}{\partial \mathbf{v}}$ is given by the damping force
- $\frac{\partial \mathbf{F}}{\partial t}$ is typically zero (no time-dependent forces)

For exercise

- **Task 1**: derive Jacobians for a 1-DoF mass-spring system

Semi-Implicit Euler

- Substitute $\mathbf{F}(t_i + h)$ with linearized expression
- Substitute $\mathbf{x}(t_i + h) = \mathbf{x}(t_i) + h \cdot \mathbf{v}(t_i + h)$ in linearized force

$$\mathbf{v}(t_i + h) = \mathbf{v}(t_i) + h\mathbf{M}^{-1}\left(\mathbf{F}(t_i) + h\frac{\partial \mathbf{F}}{\partial \mathbf{x}} \cdot \mathbf{v}(t_i + h) + \frac{\partial \mathbf{F}}{\partial \mathbf{v}} \cdot [\mathbf{v}(t_i + h) - \mathbf{v}(t_i)]\right)$$

Linear system to solve (multiplied by $\mathbf{M}$)

$$\underbrace{\left(\mathbf{M} - h\frac{\partial \mathbf{F}}{\partial \mathbf{v}} - h^2\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right)}_{\mathbf{A}}\mathbf{v}(t_i + h) = \underbrace{\left(\mathbf{M} - h\frac{\partial \mathbf{F}}{\partial \mathbf{v}}\right) \cdot \mathbf{v}(t_i) + h\mathbf{F}(t_i)}_{\mathbf{b}}$$

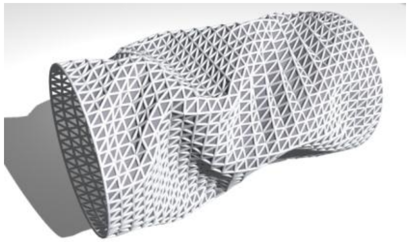
Matrix Structure

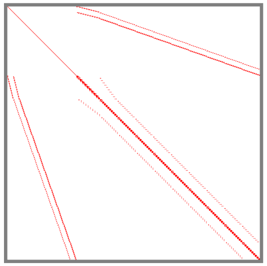
The matrix $\mathbf{A}$ *has a special structure.* $\mathbf{A}$ *is*

- ***blocked***: (3x3)-blocks relate 3D forces to 3D positions
- ***sparse***: block $\mathbf{A}_{\mathrm{ij}}$ is only nonzero if $\mathbf{i} = \mathbf{j}$ or if there is a spring between nodes $\mathbf{i}$ and $\mathbf{j}$ (-> *most entries are zero*)
- ***symmetric***: force Jacobian is related to Hessian of Energy

$$\boxed{\frac{\partial^2 E}{\partial \mathbf{x}_i \partial \mathbf{x}_j} = \frac{\partial^2 E}{\partial \mathbf{x}_j \partial \mathbf{x}_i}}$$

#v:1448
#f: 2880





Disney Research

Outline

- Numerical time integration (*continued*)
  - Explicit vs. Implicit Euler
  - Solving nonlinear systems: Newton's method
  - The semi-implicit Euler method
  - Solving sparse linear systems
  - Integration Schemes for higher order ODEs
- Towards practical mass spring systems
- Constraints

Solving Sparse Linear Systems

Solve $\mathbf{Av} = \mathbf{b}$ with *direct* or *iterative* methods.

Direct methods

- Factorize $\mathbf{A} = \mathbf{L}\mathbf{L}^t$, with $\mathbf{L}$ lower triangular
- Solve $\mathbf{L}\mathbf{L}^t\mathbf{v} = \mathbf{b}$ via two triangular solves,

$$\mathbf{Lu} = \mathbf{b}, \qquad \mathbf{L}^t\mathbf{v} = \mathbf{u}$$

Cholesky decomposition

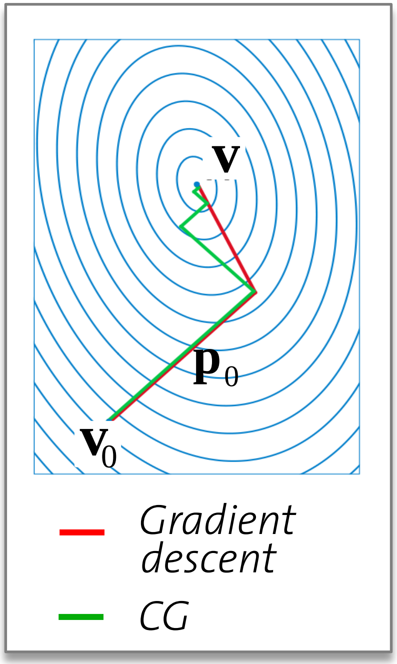
+ Robust, accuracy to numerical precision
+ Factorization can be reused for different $\mathbf{b}$
– Memory requirements $\mathbf{L}$ (limits problem size)

Solving Sparse Linear Systems

Solve $\mathbf{Av} = \mathbf{b}$ with *direct* or *iterative* methods.

Iterative methods

- Jacobi & Gauss-Seidel
  - solve system *row by row*
  - simple but slow convergence
- Conjugate Gradients
  - minimize $\frac{1}{2}\mathbf{v}^t\mathbf{Av} - \mathbf{v}^t\mathbf{b}$ via sequence of conjugate corrections $\mathbf{p}_k$, where $\mathbf{p}_i^t\mathbf{Ap}_j = 0$
  - Fast convergence, low memory requirements



Disney Research

Example: Baraff & Witkin '98

[BW98]: cloth animation with semi-Implicit Euler (SIE)

- Cast time integration as a linear problem, $\mathbf{A} \cdot \mathbf{v}(t_i + h) = \mathbf{b}$
- Solve linear system with conjugate gradients

- A step of SIE is far more expensive than explicit Euler
- Better stability of SIE allows (*very*) large time steps, $h \approx 0.01s$ vs. $h \approx 0.0001s$ $\rightarrow$ much higher performance!

Example: Baraff & Witkin '98

- Limitations
  - *Unconditional* stability only for truly linear problems
  - '*Semi*'-implicit variant leads to severe numerical dissipation (*video*)

#triangles: 8844
h=0.01-0.02s

Numerical Dissipation

Example: Baraff & Witkin ‘98

- Limitations
  - *Unconditional* stability only for truly linear problems
  - ‘*Semi*’-implicit variant leads to severe numerical dissipation (*video*)
- Alternatives
  - Solve full nonlinear equations
  - Use more accurate implicit schemes $\rightarrow$ *more expensive (in most cases)!*

#triangles: 8844
h=0.01-0.02s

Outline – Part 1

- Recap of last lecture
- Numerical time integration (*continued*)
  - Explicit vs. implicit Euler revisited
  - Solving nonlinear systems: Newton's method
  - The semi-implicit Euler method
  - Solving sparse linear systems
  - Integration Schemes for higher order ODEs

Higher-Order Numerical Integration

Methods for
1st order ODEs

$$\mathbf{M}\ddot{\mathbf{x}} = \mathbf{F}^{\text{int}} + \mathbf{F}^{\text{ext}}$$

$$\dot{\mathbf{x}} = \mathbf{v} \qquad \mathbf{M}\dot{\mathbf{v}} = \mathbf{F}^{\text{int}} + \mathbf{F}^{\text{ext}}$$

1st order ODE solver

- Euler, Heun, Midpoint, Runge-Kutta schemes
- Implicit Euler

Methods for
higher order ODEs

$$\mathbf{M}\ddot{\mathbf{x}} = \mathbf{F}^{\text{int}} + \mathbf{F}^{\text{ext}}$$

Higher order ODE solver

- Verlet, Leapfrog, Symplectic Euler
- Newmark

Verlet Integration

Combine forward and backward expansions of $\mathbf{x}(t)$

$$\mathbf{x}(t+h) = \mathbf{x}(t) \boxed{+h\mathbf{x}'(t)} + \frac{h^2}{2}\mathbf{a}(t) \boxed{+\frac{h^3}{6}\mathbf{x}'''(t)} + O(h^4)$$

$+$

$$\mathbf{x}(t-h) = \mathbf{x}(t) \boxed{-h\mathbf{x}'(t)} + \frac{h^2}{2}\mathbf{a}(t) \boxed{-\frac{h^3}{6}\mathbf{x}'''(t)} + O(h^4)$$

$=$

Verlet scheme

$$\mathbf{x}(t+h) = 2\mathbf{x}(t) - \mathbf{x}(t-h) + h^2\mathbf{a}(t) + O(h^4)$$

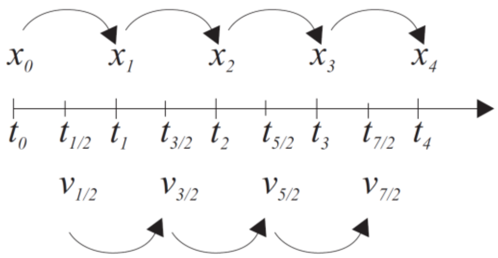
+ 2nd order accurate with only one force evaluation
– *Two-step* method problematic for discontinuities
– Velocities have to be approximated *a posteriori*

Leapfrog

- Compute velocities on *staggered* grid

Leapfrog

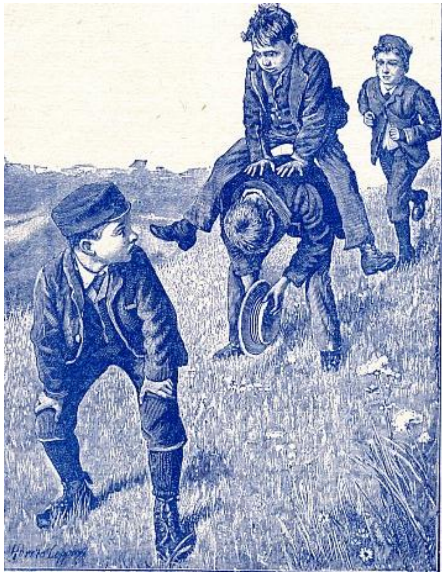
$$\begin{aligned}
\mathbf{v}(t+h/2) &= \mathbf{v}(t-h/2)+h\cdot\mathbf{a}(t)\\
\mathbf{x}(t+h) &= \mathbf{x}(t)+h\cdot\mathbf{v}(t+h/2)
\end{aligned}$$



\+ 2^{nd} order accurate and only one force evaluation

\+ One-step method

– Only applicable if *a(t)* does not depend on velocity $\rightarrow$ *no damping*!



Symplectic Euler

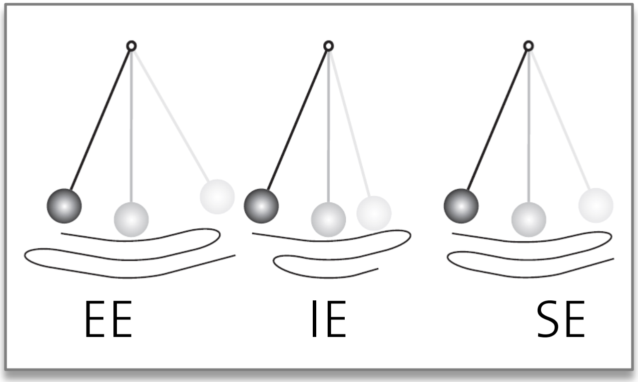
Symplectic Euler (SE)

$$
\begin{aligned}
\mathbf{v}(t+h) &= \mathbf{v}(t) + h \cdot \mathbf{a}(t) \\
\mathbf{x}(t+h) &= \mathbf{x}(t) + h \cdot \mathbf{v}(t+h)
\end{aligned}
$$

Leapfrog

$$
\begin{aligned}
\mathbf{v}(t+h/2) &= \mathbf{v}(t-h/2) + h \cdot \mathbf{a}(t) \\
\mathbf{x}(t+h) &= \mathbf{x}(t) + h \cdot \mathbf{v}(t+h/2)
\end{aligned}
$$

- Only 1st order accurate, but
- Good stability for oscillatory motion (*unlike EE*)
- Good conservation properties (*unlike IE*)
  - Exactly conserves momentum
  - *Nearly* conserves energy over long run times

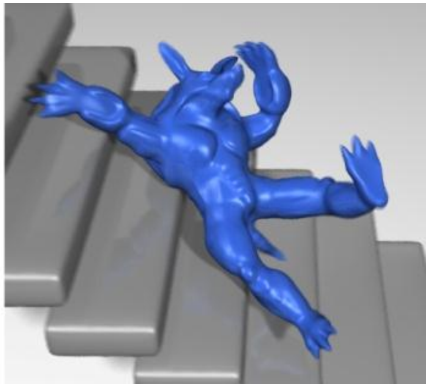


Outline – Part 2

- Towards practical Mass-spring simulations
  - Collision handling
  - Setting up springs for solids and cloth
  - Limitations of mass-spring systems
- Constraints
  - Motivation and definitions
  - Soft constraints and the penalty method
  - Hard constraints
  - Examples

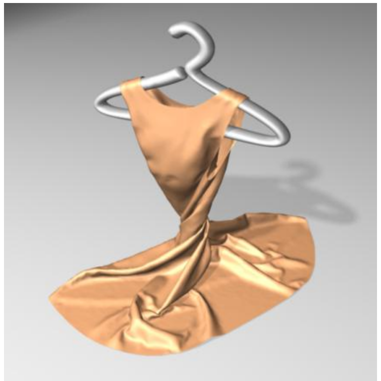
Where We Are…

- Basic mass-spring system
  - Elastic spring forces, point damping
  - Explicit and implicit time stepping



What else do we need to simulate cloth and solids?

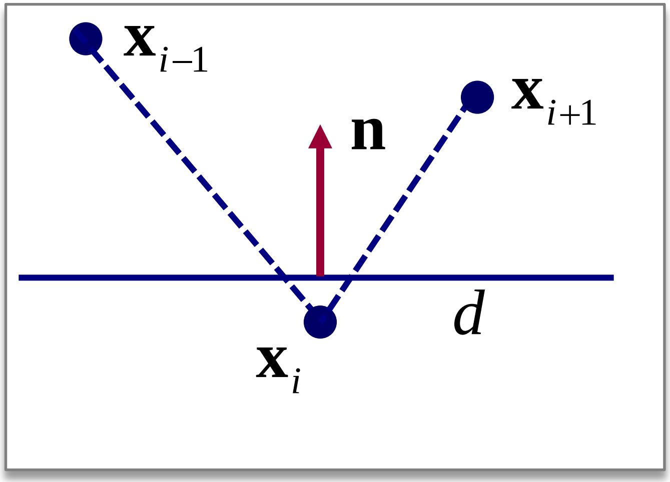
  - Handle collisions
  - Set up springs
  - …



Simple Collision Handling

Details follow in class on
Collision Detection and Response (November 21st)

For exercise: *Handle collisions between particles and a plane.*

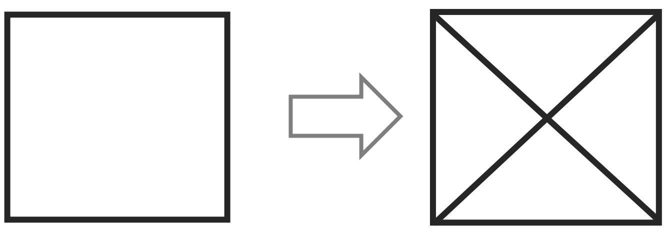


1. Step forward in time
2. Find particles that are below plane
3. For each, set up spring force
   - proportional to penetration depth d
   - in direction of normal $\mathbf{n}$
4. Add penalty force in next time step

Setting Up Springs: Solids

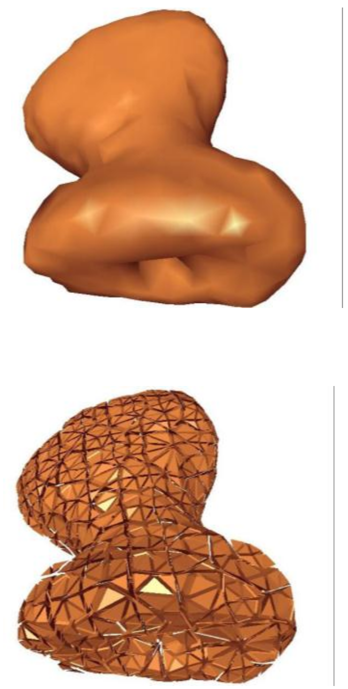
Model provided as tetrahedral mesh?

- *Yes*: springs along edges
- *No*: find *stable* topology (*shearing*)



Problems

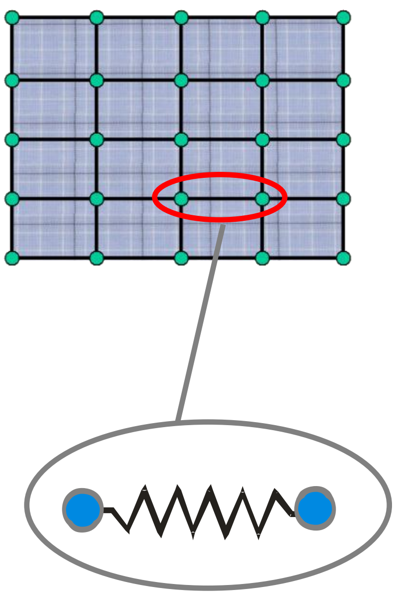
- No control over material properties
- No explicit volume preservation



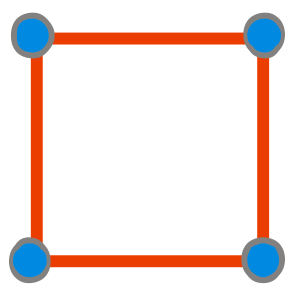
1349 mass points
4562 tets
6888 springs

Setting Up Springs: Cloth

What types of springs are required?

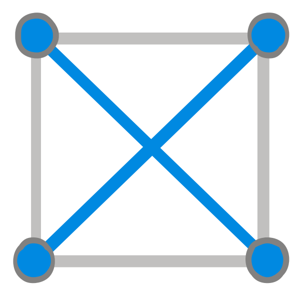


Structural Springs



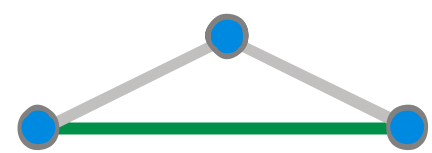
Stretching

Diagonal Springs



Shearing

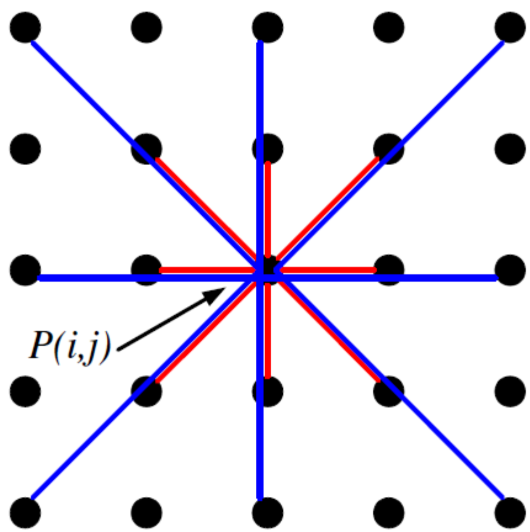
Interleaved Springs

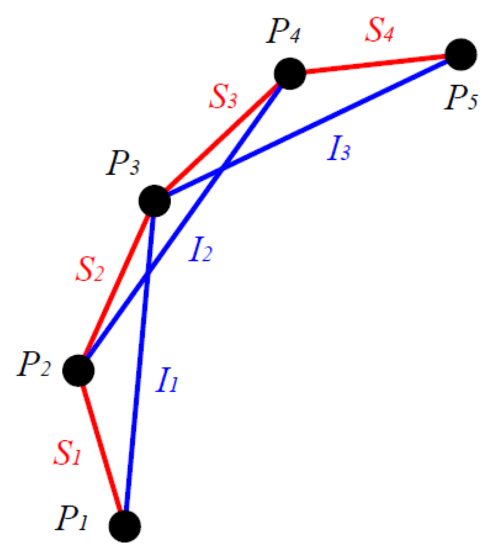


Bending

Setting Up Springs: Cloth

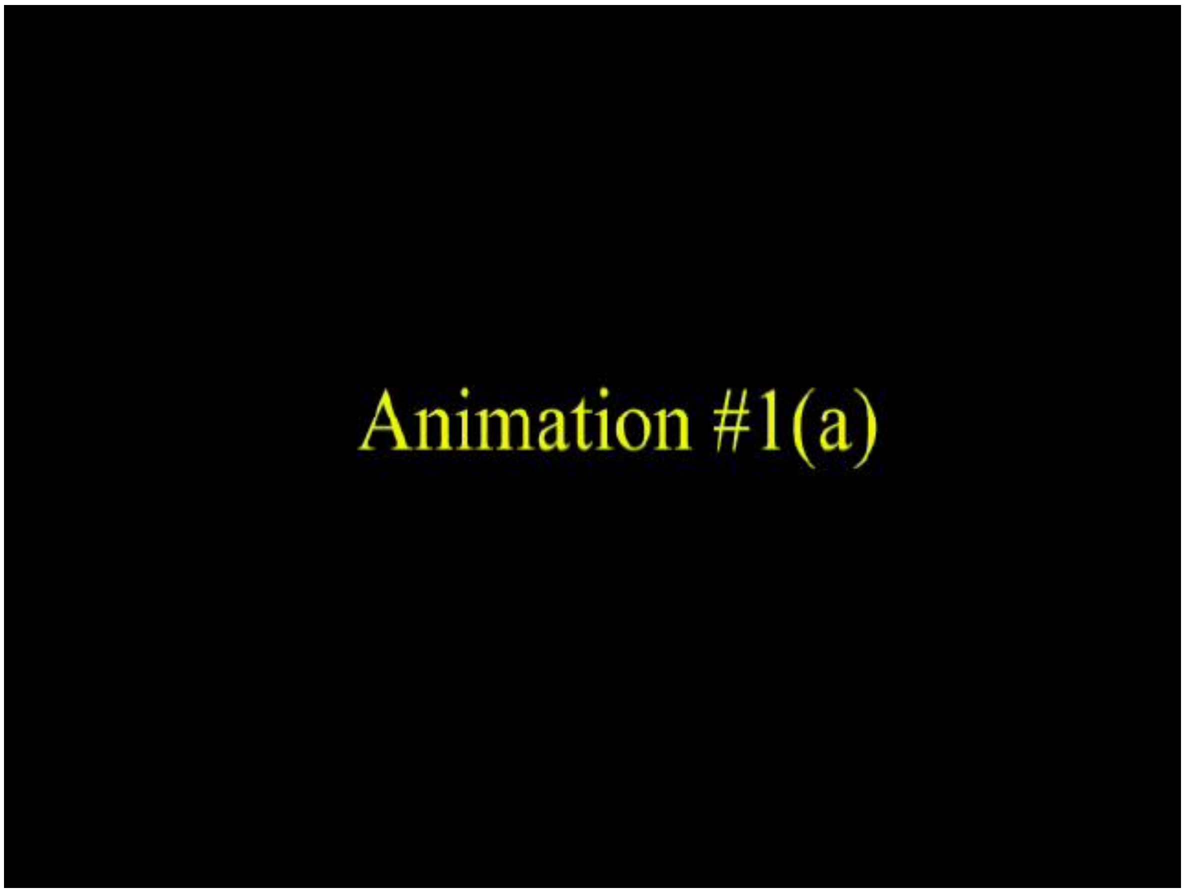
Red: stretch and shear springs
Blue: bending springs





Disney Research

Choi & Ko ‘02

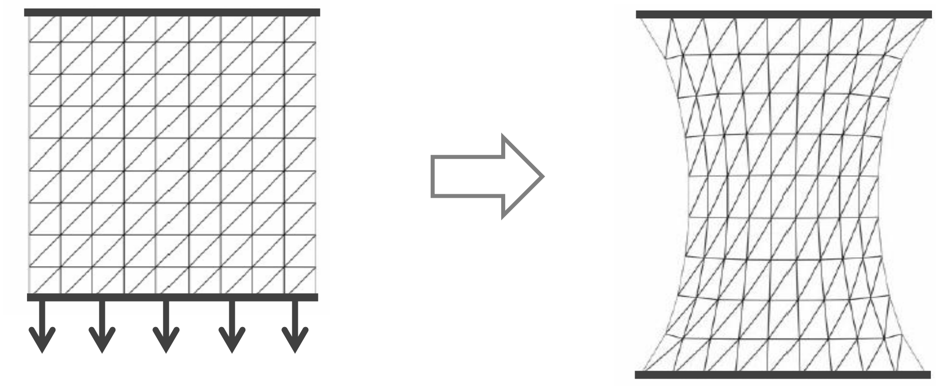


Choi & Ko ‘02

Problems

Springs couple different deformation modes

- Bending and shear springs respond to stretch
- Leads to uncontrollable transverse contraction

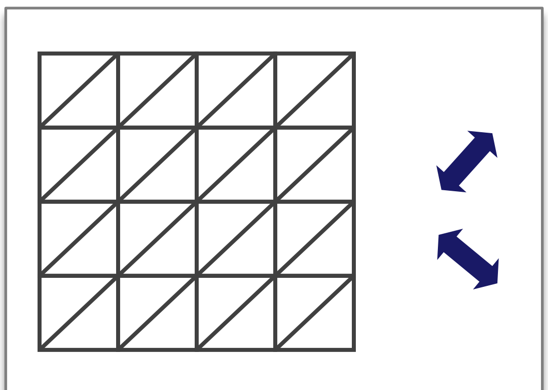


Disney Research

Problems

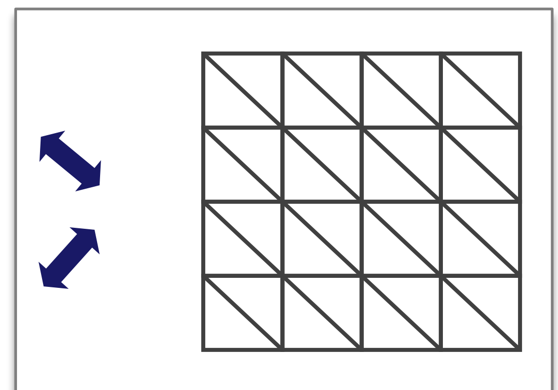
Material behavior depends strongly on topology.

$\rightarrow$ *direction-dependent behavior!*



high resistance

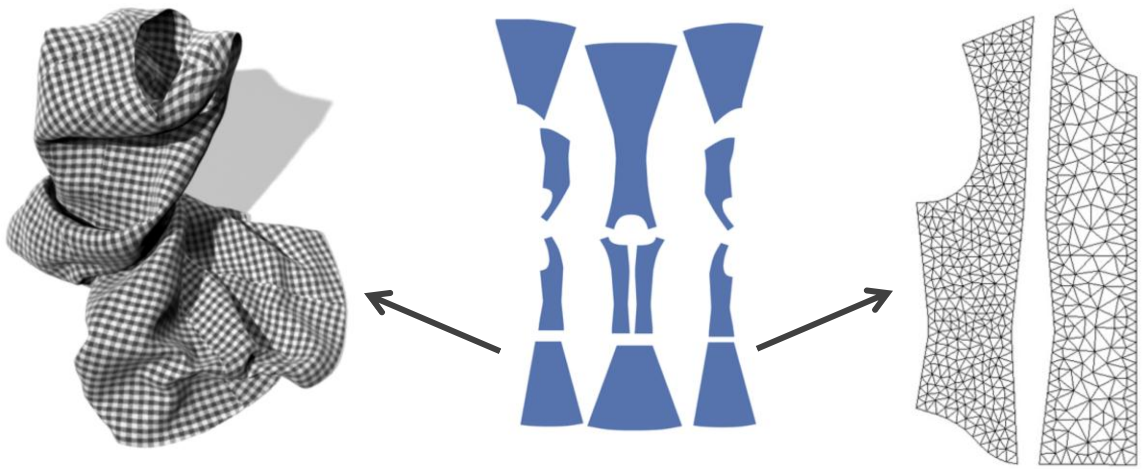
low resistance



Problems

Curved boundaries require triangle meshes.

How to distinguish between stretch and shear?



Mass-Spring Systems

Limitations

- Strong dependence on discretization (*topology*)
- Limited control over material behavior
- No (explicit) volume preservation (*for solids*)

Verdict

Simple and efficient, but not very accurate

Alternatives:

- Constraints (*next*)
- Continuum-mechanics with Finite Elements (*November 7th*)

Outline – Part 2

- Towards practical Mass-spring simulations
  - Collision handling
  - Setting up springs for solids and cloth
  - Limitations of mass-spring systems
- Constraints
  - Motivation and definitions
  - Soft constraints and the penalty method
  - Hard constraints
  - Examples

Constraints: Motivation

- So far: motion of particles defined by forces
  - Elastic springs, damping, gravity, …
- Motion is unrestricted (*just apply enough force*)
- Want to restrict motion sometimes → *Condition*
  - Bead on a wire (constant length)
  - Incompressible deformations (constant volume)
- **Constraints** can be used
  - to strictly enforce conditions (*unlike elastic forces*)
  - to derive general forces (*unlike springs*)

Constraints: Definition

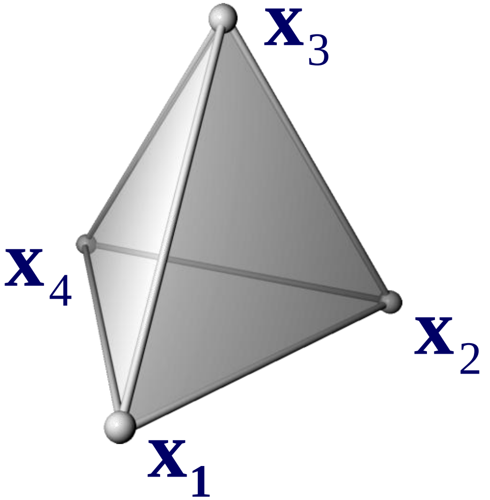
A constraint $C(\mathbf{x}_1,...,\mathbf{x}_n)$ is

- a scalar-valued function of one or several arguments
- an implicit expression for a relation that must hold between its arguments

Convention: a constraint is satisfied if $C = 0$

Examples of *n*-ary constraints

- constant position $C_1(\mathbf{x}_1)$
- constant length $C_2(\mathbf{x}_1, \mathbf{x}_2)$
- constant area $C_3(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$
- constant volume $C_4(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$



Outline – Part 2

- Towards practical Mass-spring simulations
  - Collision handling
  - Setting up springs for solids and cloth
  - Limitations of mass-spring systems
- Constraints
  - Motivation and definitions
  - Soft constraints and the penalty method
  - Hard constraints
  - Examples

Penalty Forces

How can we enforce constraints?

- Define potential energy for constraint

$$E_C(\mathbf{x}_1,\ldots,\mathbf{x}_n) = \tfrac{1}{2}kC(\mathbf{x}_1,\ldots,\mathbf{x}_n)^2$$

k is stiffness coefficient

$\rightarrow E_C = 0$ when constraint met, $E_C > 0$ otherwise

- Constraint Force is negative gradient of potential energy

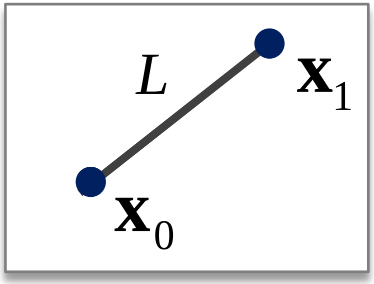
$$\mathbf{F}_j = -\frac{\partial E_C}{\partial \mathbf{x}_j} = -kC(\mathbf{x}_1,\ldots,\mathbf{x}_n)\frac{\partial C(\mathbf{x}_1,\ldots,\mathbf{x}_n)}{\partial \mathbf{x}_j}$$

⇨ **Penalty** force, pulls system towards $C = 0$

Example: Distance Preservation

- Preserve distance between two points

$$C(\mathbf{x}_0, \mathbf{x}_1) = |\mathbf{x}_1 - \mathbf{x}_0| - L$$



- Constraint force on $\mathbf{x}_1$ follows as

$$\mathbf{F}_1^C(\mathbf{x}_0, \mathbf{x}_1) = -kC(\mathbf{x}_0, \mathbf{x}_1)\frac{\partial C(\mathbf{x}_0, \mathbf{x}_1)}{\partial \mathbf{x}_1}$$

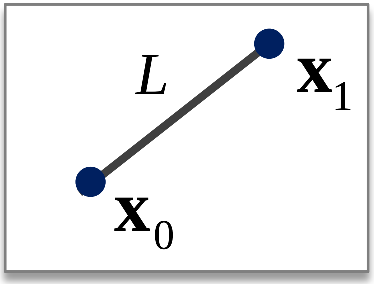
$$= -k\left(|\mathbf{x}_1 - \mathbf{x}_0| - L\right)\frac{\mathbf{x}_1 - \mathbf{x}_0}{|\mathbf{x}_1 - \mathbf{x}_0|}$$

= Spring Force!

Disney Research

Simulation with Penalty Forces

- Simulate $\mathbf{x}_0$ and $\mathbf{x}_1$ using Newton's law (*no spring force!*)



- Enforce distance constraint with penalty forces

$$\mathbf{F}_i^C(\mathbf{x}_0,\mathbf{x}_1) = -kC(\mathbf{x}_0,\mathbf{x}_1)\frac{\partial C(\mathbf{x}_0,\mathbf{x}_1)}{\partial \mathbf{x}_i}$$

- Step forward in time (*explicit Euler*)

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{M}^{-1}\left(\mathbf{F}^C(\mathbf{x}) + \mathbf{F}^{ext}\right)$$

More Constraints

- Recipe for constraint forces
  - Define constraint via geometric condition
  - Set up energy and compute gradient (*chain rule*)

- Preserve area A of triangle $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2)$

$$C(\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_2}) = \tfrac{1}{2}\left|(\mathbf{x_1} - \mathbf{x_0}) \times (\mathbf{x_2} - \mathbf{x_0})\right| - A$$

- Preserve volume V of tetrahedron $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$

$$C(\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}) = \tfrac{1}{6}(\mathbf{x}_1 - \mathbf{x_0}) \cdot \left[(\mathbf{x}_2 - \mathbf{x_0}) \times (\mathbf{x}_3 - \mathbf{x_0})\right] - V$$

Constraint Forces

Constraint forces are

- a powerful mechanism to enforce various conditions
- generic: express condition, forces from standard scheme
- *n*-ary forces as opposed to binary springs

However, constraints can be

- computationally expensive (implicit integration?)
- redundant or conflicting

- penalty forces model ***soft*** constraints, not sufficient for strict enforcement (***hard*** constraints)

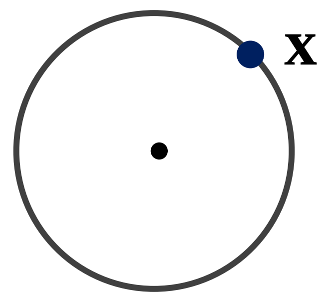
Outline – Part 2

- Towards practical Mass-spring simulations
  - Collision handling
  - Setting up springs for solids and cloth
  - Limitations of mass-spring systems
- Constraints
  - Motivation and definitions
  - Soft constraints and the penalty method
  - Hard constraints
  - Examples

Hard Constraints

How can we model a hard constraint?

- Consider 2D particle constrained to move on unit circle
- Define *legal* kinematics
  - position $C(\mathbf{x}) = 0$
  - velocity $\dot{C}(\mathbf{x}) = 0$
  - acceleration $\ddot{C}(\mathbf{x}) = 0$



$$C(\mathbf{x}) = \tfrac{1}{2}(\mathbf{x} \cdot \mathbf{x} - 1)$$

Idea: start with legal position and velocity, ensure that acceleration remains legal (constraint forces!)

⇨ *constraint stays satisfied (if ODE is solved exactly)*

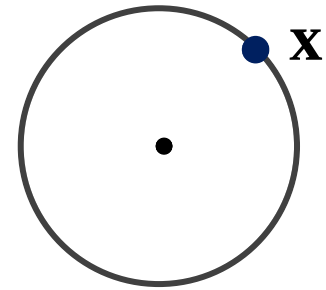
Computing Constraint Forces

Legal acceleration (1)

$$\ddot{C}(\mathbf{x}) = \ddot{\mathbf{x}} \cdot \mathbf{x} + \dot{\mathbf{x}} \cdot \dot{\mathbf{x}} = 0$$

Newton's Law

$$\ddot{\mathbf{x}} = \frac{\mathbf{F} + \mathbf{F}^C}{m} \quad (2)$$



$$C(\mathbf{x}) = \tfrac{1}{2}(\mathbf{x} \cdot \mathbf{x} - 1)$$

- Use (2) in (1)

$$\mathbf{F}^C \cdot \mathbf{x} = -\mathbf{F} \cdot \mathbf{x} - m\dot{\mathbf{x}} \cdot \dot{\mathbf{x}} \quad (3)$$

- Require constraint force to act only in gradient direction

$$\mathbf{F}^C = \lambda \frac{\partial C}{\partial \mathbf{x}} = \lambda \mathbf{x} \quad (4)$$

- Use (4) in (3)

$$\lambda = -\frac{\mathbf{F} \cdot \mathbf{x} + m\dot{\mathbf{x}} \cdot \dot{\mathbf{x}}}{\mathbf{x} \cdot \mathbf{x}}$$

⇨ *hard constraint force!*

Simulation with Hard Constraints

Use hard constraints with, e.g., explicit Euler

1. Solve (LES) for constraint force magnitudes λ_n
2. Compute constraint forces $\mathbf{F}_n^C = \frac{\partial C(\mathbf{x}_n)}{\partial \mathbf{x}}^t \lambda_n$
3. Step forward in time $\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{M}^{-1}\left(\mathbf{F}_n^C + \mathbf{F}^{ext}\right)$

Note: *using implicit integration with constraints is (much) more complicated!*

Hard Constraints

Hard constraint forces

- add just enough force to maintain constraint (*exactly*)
- require no high stiffness (*numerically pleasing*)

However,

- Constraints drift (error in ODE solve), needs correction
- General formulation (many constraints) more involved
- Requires solution of systems of equations

A. Witkin and D. Baraff. *Physically-Based Modeling: Constrained Dynamics*. In Siggraph '01 Course Notes.

References & Further Reading

Textbooks (check with library)

- W. Press, S. Teukolsky, W. Vetterling, and B. Flannery: *Numerical Recipes. The Art of Scientific Computing*. 3rd edition, Cambridge University Press, 2007.

Articles (available from lecture website)

- D. Baraff and A. Witkin. *Large Steps in Cloth Simulation*. In Siggraph '98.
- A. Witkin and D. Baraff. *Physically-Based Modeling: Constrained Dynamics*. In Siggraph '01 Course Notes.