

50.017 Graphics and Visualization

Picture: A. Selle et al.

# Particle Systems and ODE Solvers II, Mass-Spring Modeling

Sai-Kit Yeung  
SUTD ISTD

With slides from Wojciech Matusik, Jaakko  
Lehtinen and others

# ODEs and Numerical Integration

---

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

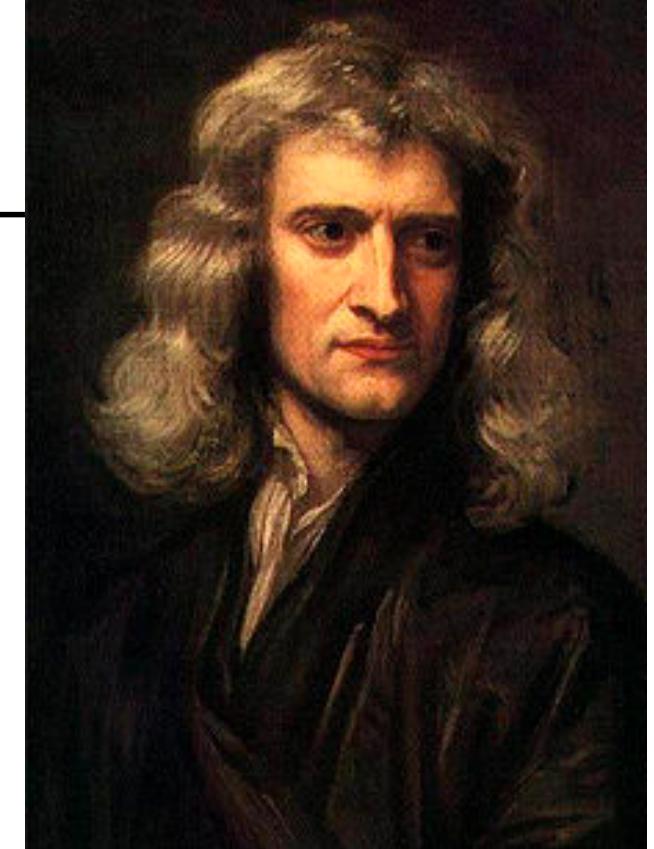
- Given a function  $f(\mathbf{X}, t)$  compute  $\mathbf{X}(t)$
- Typically, initial value problems:
  - Given values  $\mathbf{X}(t_0) = \mathbf{X}_0$
  - Find values  $\mathbf{X}(t)$  for  $t > t_0$
- We can use lots of standard tools

# Reduction to 1st Order

---

- Point mass: 2<sup>nd</sup> 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

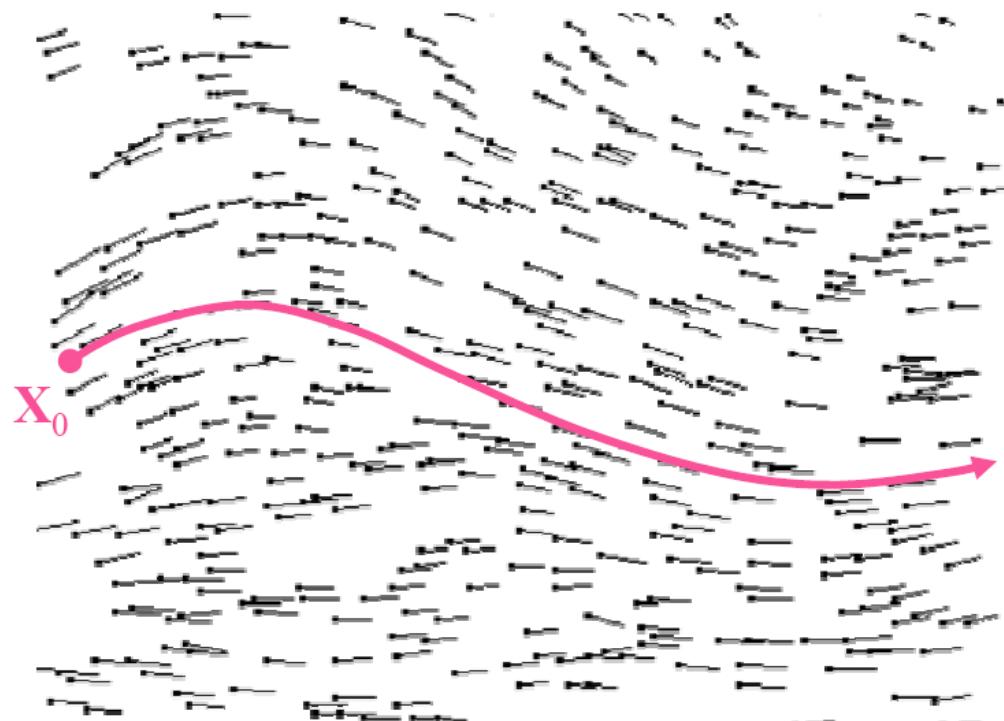
$$\begin{cases} \frac{d}{dt}\vec{x} = \vec{v} \\ \frac{d}{dt}\vec{v} = \vec{F}/m \end{cases}$$

2 unknowns (x, v)  
instead of just x

# ODE: Path Through a Vector Field

---

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



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

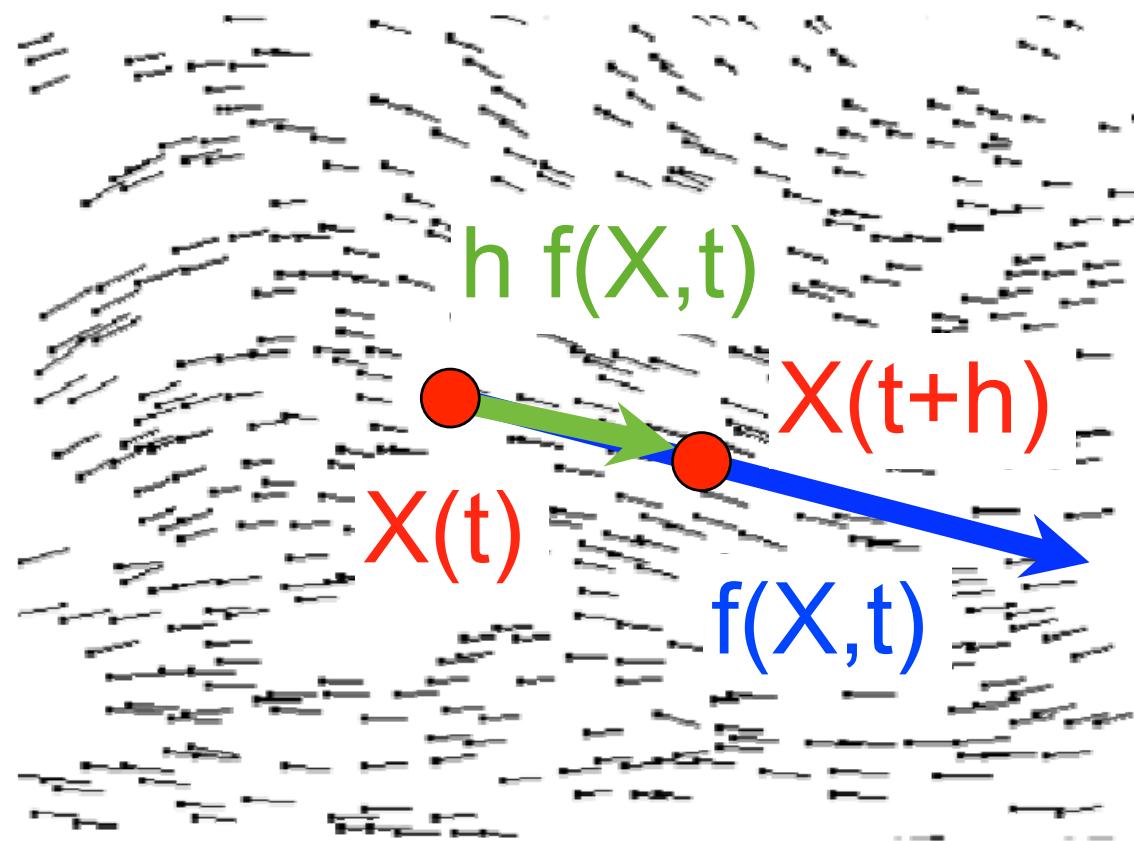
“When we are at state  $\mathbf{X}$  at time  $t$ , where will  $\mathbf{X}$  be after an infinitely small time interval  $dt$  ?”

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

# Euler, Visually

---

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



What is the inherent assumption on the velocity?

# 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:

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

# 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:

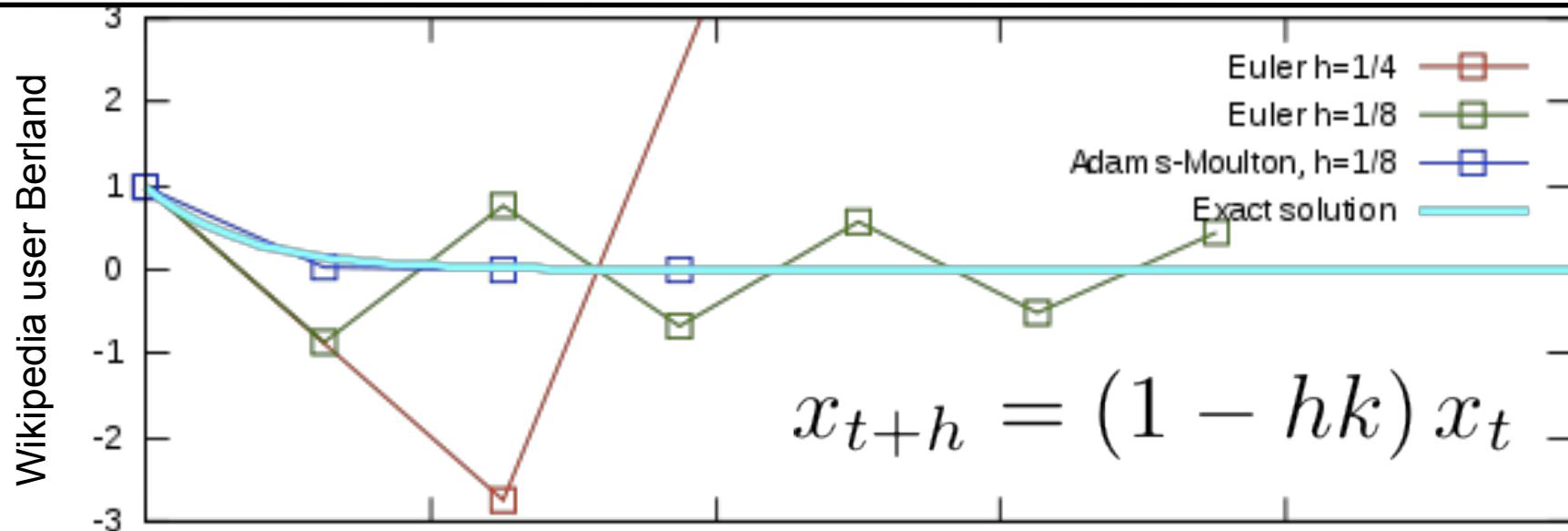
$$x_{t+h} = x_t + h f(x_t, t)$$

$$= x_t - hkx_t$$

$$= \boxed{(1 - hk) x_t}$$

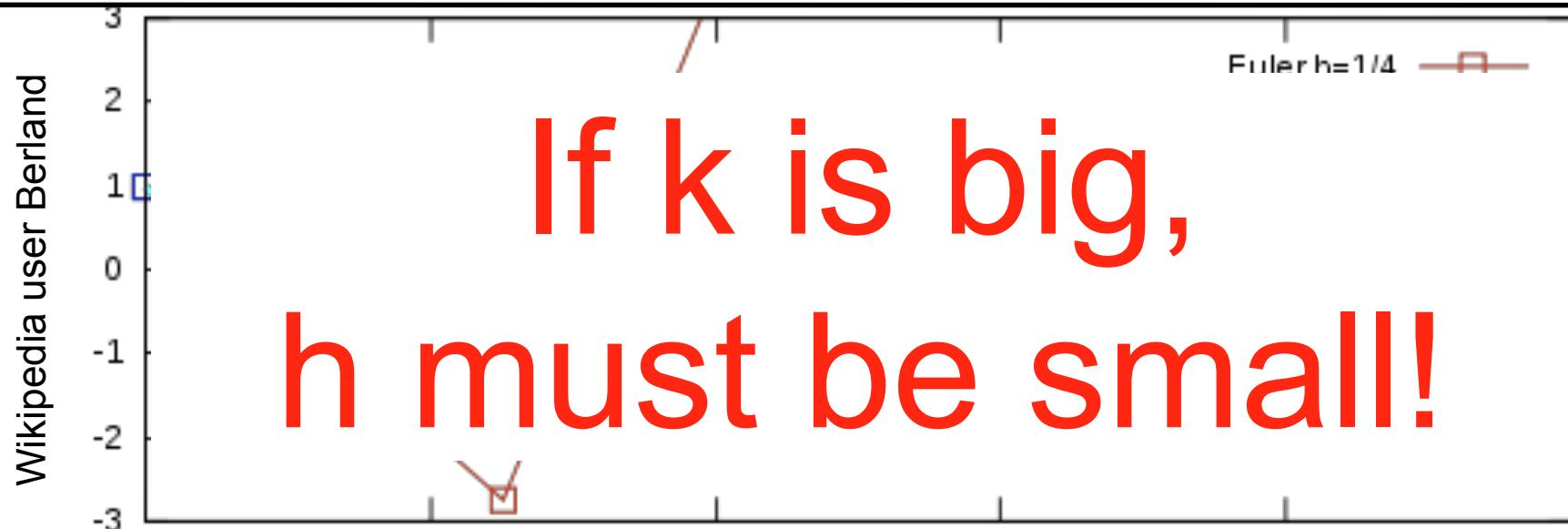
Look familiar?

# Euler's Method: Not Always Stable



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

# Euler's Method: Not Always Stable



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

# Analysis: Taylor Series

---

- Expand exact solution  $\mathbf{X}(t)$

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

- Euler's method approximates:

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

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

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

# Analysis: Taylor Series Questions?

---

- Expand exact solution  $\mathbf{X}(t)$

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

- Euler's method approximates:

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

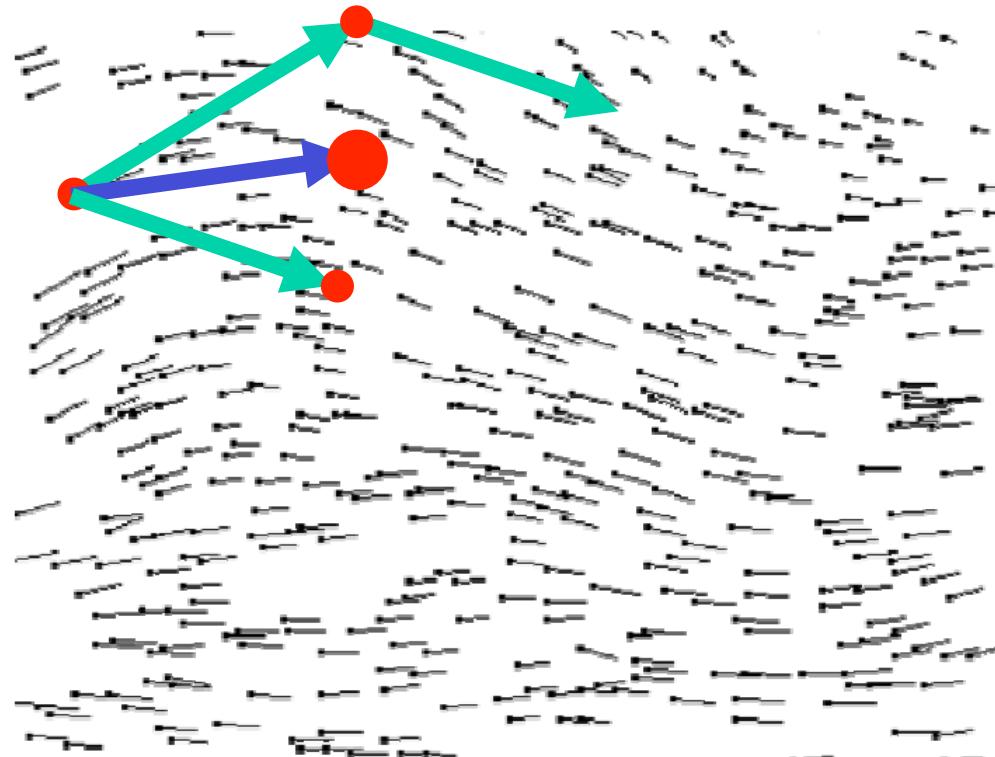
$h \rightarrow h/2 \Rightarrow \text{error} \rightarrow \text{error}/4 \text{ per step} \times \text{twice as many steps}$   
 $\rightarrow \text{error}/2$

- 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

---

- This translates to...

$$\boxed{\begin{aligned}f_0 &= f(\mathbf{X}_0, t_0) \\f_1 &= f(\mathbf{X}_0 + h f_0, t_0 + h)\end{aligned}}$$

- and we get

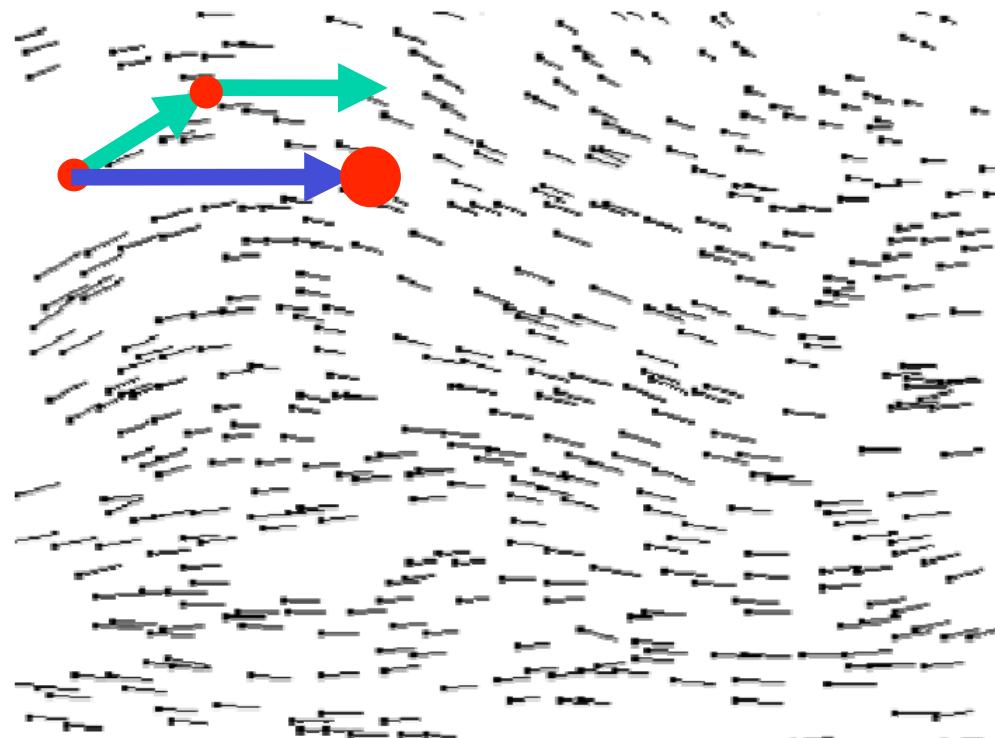
$$\boxed{\mathbf{X}(t_0 + h) = \mathbf{X}_0 + \frac{h}{2}(f_0 + f_1)} + O(h^3)$$

- This is the trapezoid method
  - Analysis omitted
- Note: What we mean by “2<sup>nd</sup> order” is that the error goes down with  $h^2$ , not  $h$  – the equation is still 1<sup>st</sup> 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...

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

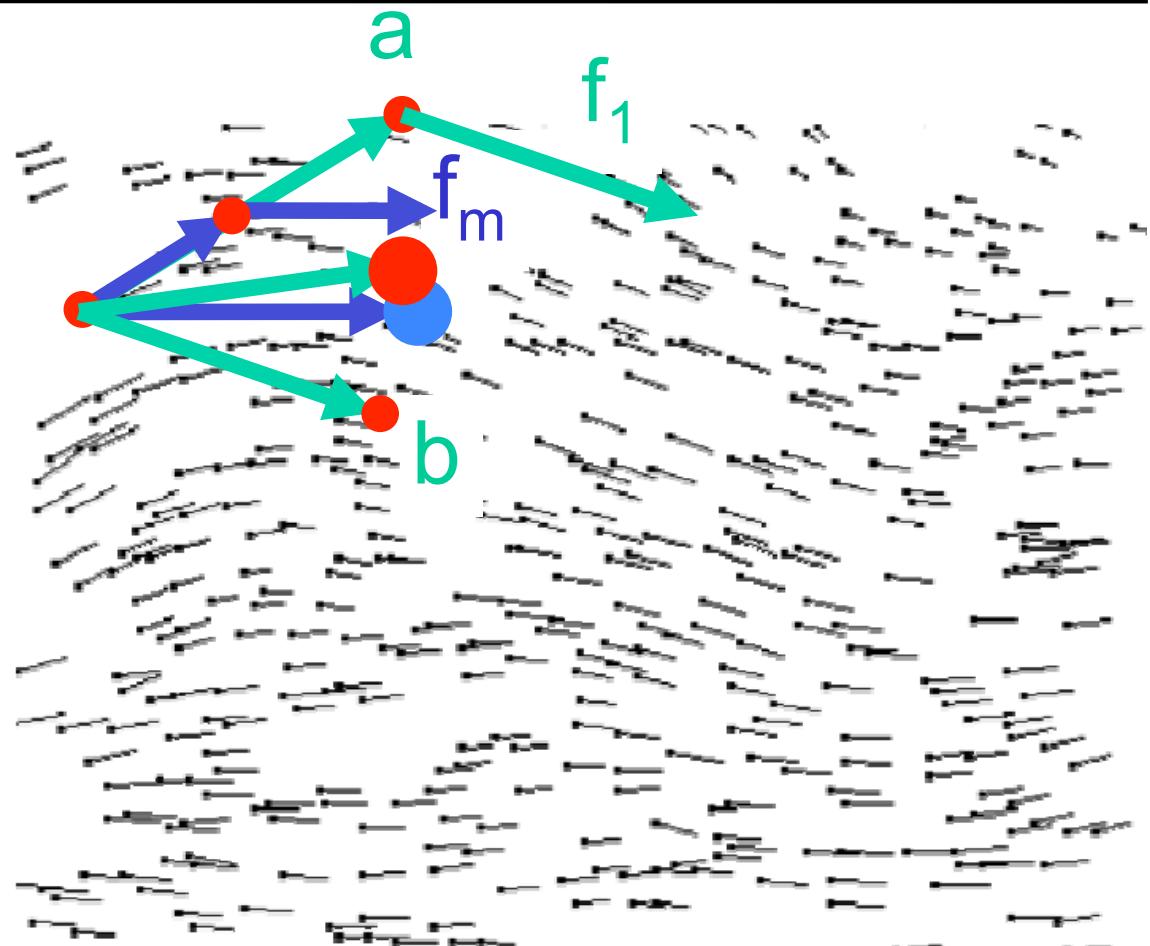
- and we get  $\boxed{\mathbf{X}(t_0 + h) = \mathbf{X}_0 + h f_m} + O(h^3)$

- This is the midpoint method
  - Analysis omitted again,  
but it's not very complicated, see [http://en.wikipedia.org/  
wiki/Midpoint\\_method](http://en.wikipedia.org/wiki/Midpoint_method).

# Comparison

---

- Midpoint:
  - $\frac{1}{2}$  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



# Can We Do Even Better?

---

- You bet!
- You can implement Runge-Kutta for assignment 3
- Again, see  
Witkin, Baraff, Kass: Physically-based Modeling  
Course Notes, SIGGRAPH 2001
- See eg  
<http://www.youtube.com/watch?v=HbE3L5CIIdQg>

# Can We Do Even Better? Questions?

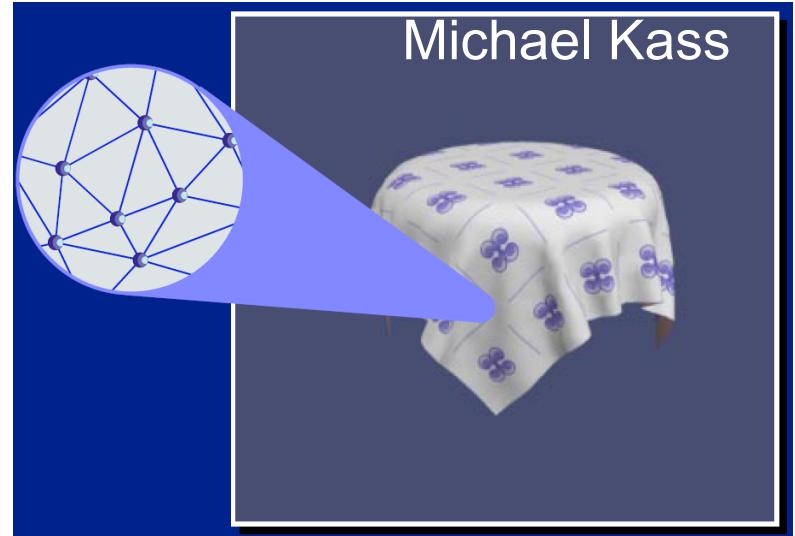
---

- You bet!
- You can implement Runge-Kutta for assignment 3
- Again, see  
Witkin, Baraff, Kass: Physically-based Modeling  
Course Notes, SIGGRAPH 2001
- See eg  
<http://www.youtube.com/watch?v=HbE3L5CIIdQg>

# Mass-Spring Modeling

---

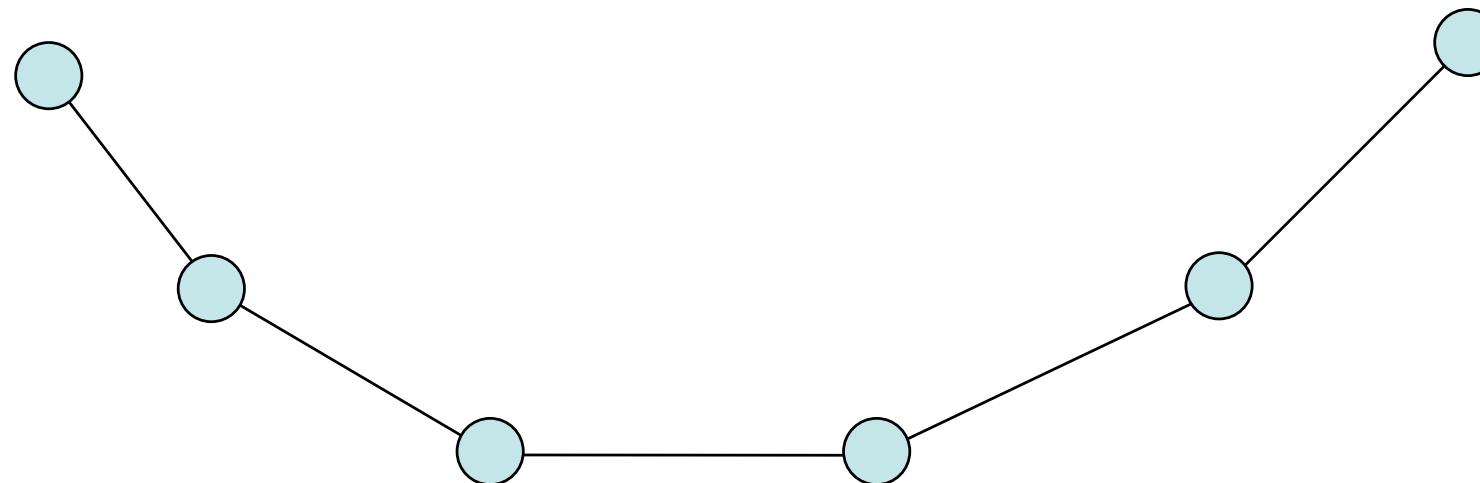
- Beyond point-like objects:  
strings, cloth, hair, etc.
- Interaction between particles
  - Create a network of spring forces that link pairs of particles



# How Would You Simulate a String?

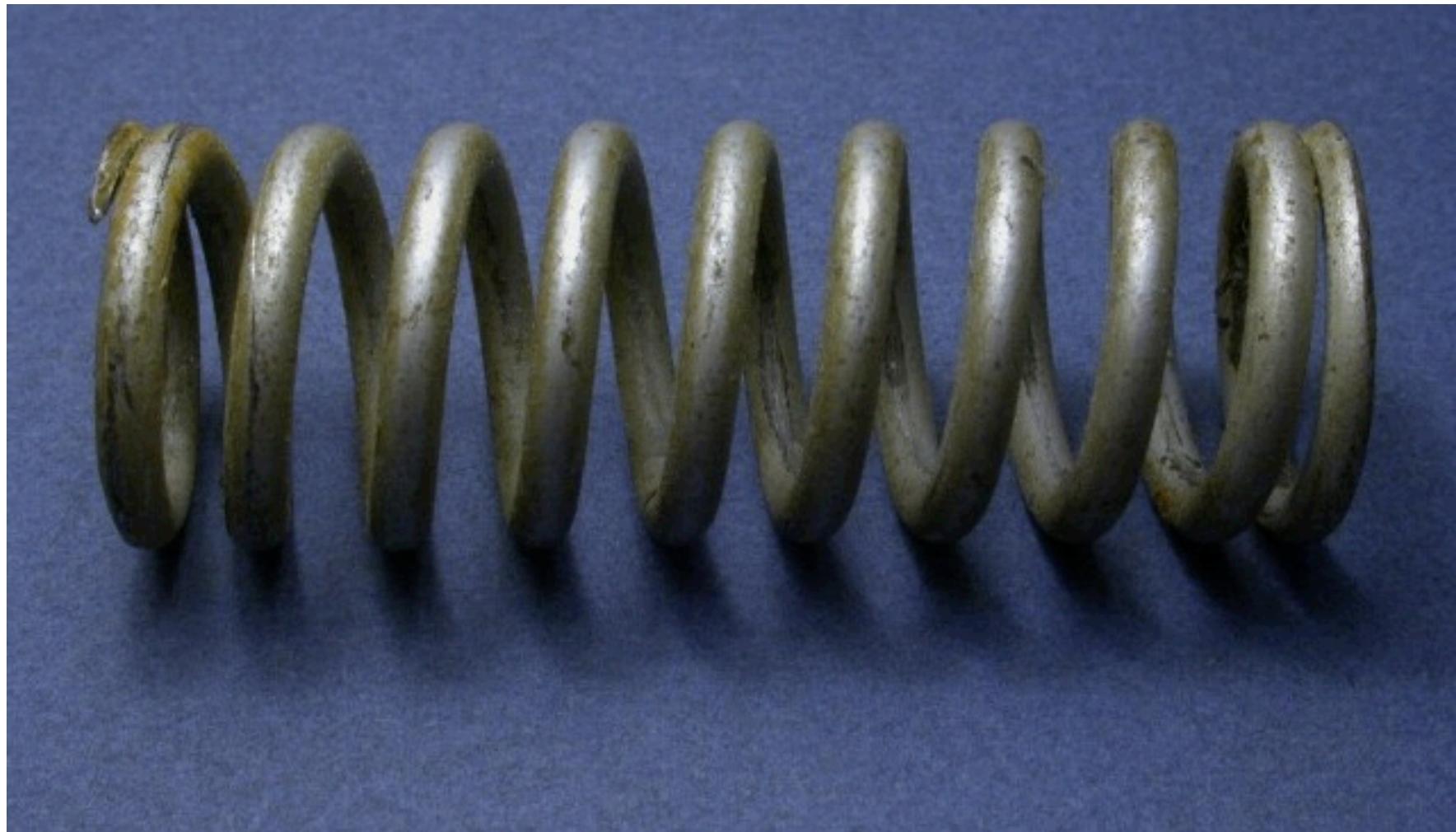
---

- Each particle is linked to two particles (except ends)
- Come up with forces that try to keep the distance between particles constant



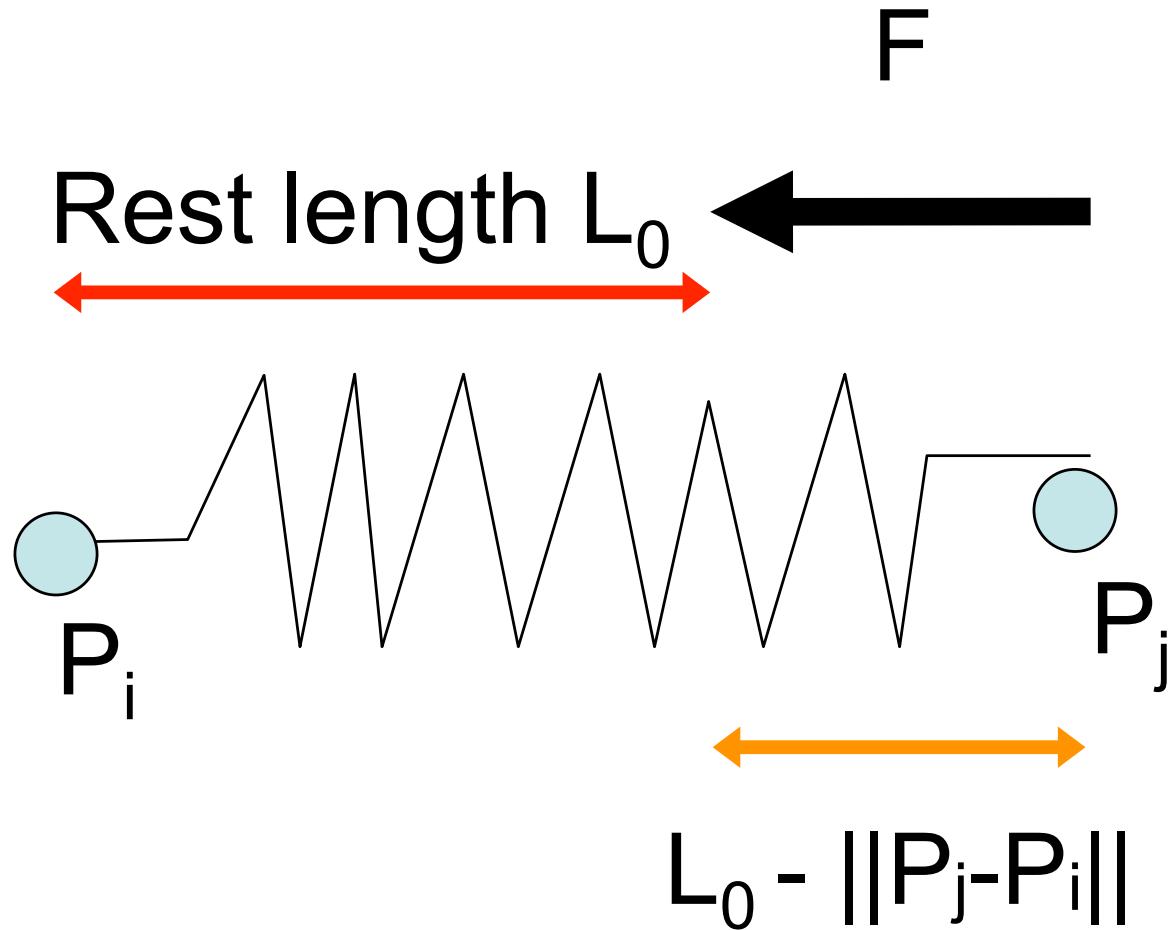
# Springs

---



# Spring Force – Hooke's Law

---



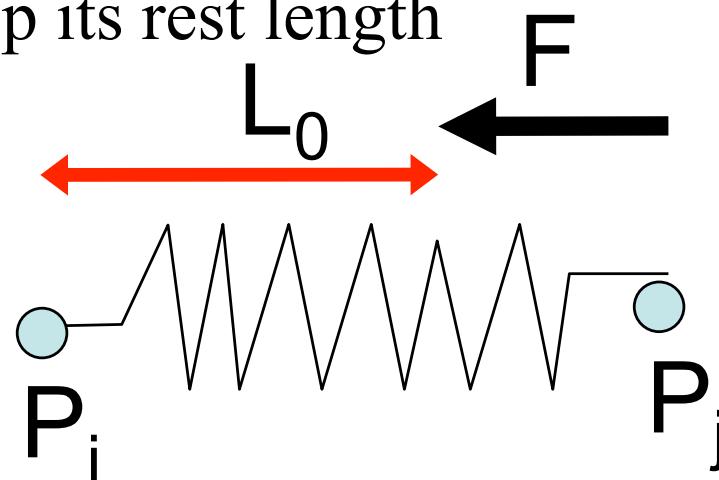
# Spring Force – Hooke's Law

---

- Force in the direction of the spring and proportional to difference with rest length  $L_0$ .

$$F(P_i, P_j) = K(L_0 - \|\vec{P_i P_j}\|) \frac{\vec{P_i P_j}}{\|\vec{P_i P_j}\|}$$

- $K$  is the stiffness of the spring
  - When  $K$  gets bigger, the spring really wants to keep its rest length



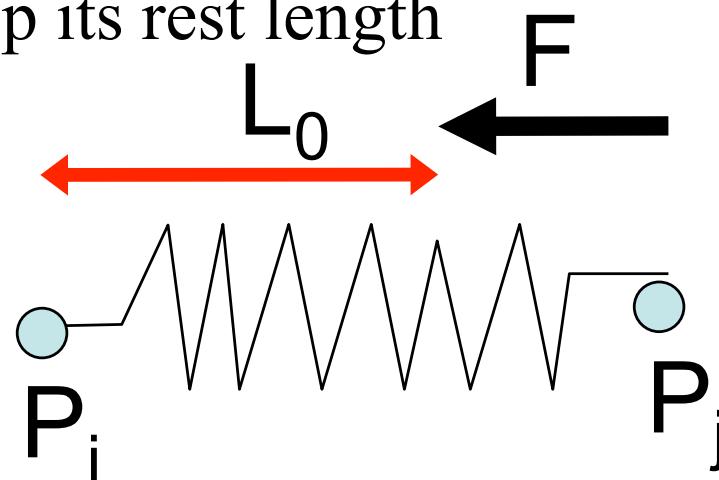
# Spring Force – Hooke's Law

---

- Force in the direction of the spring and proportional to difference with rest length  $L_0$ .

$$F(P_i, P_j) = K(L_0 - \|\vec{P_i} \vec{P_j}\|) \frac{\vec{P_i} \vec{P_j}}{\|\vec{P_i} \vec{P_j}\|}$$

- $K$  is the stiffness of the spring
  - When  $K$  gets bigger, the spring really wants to keep its rest length

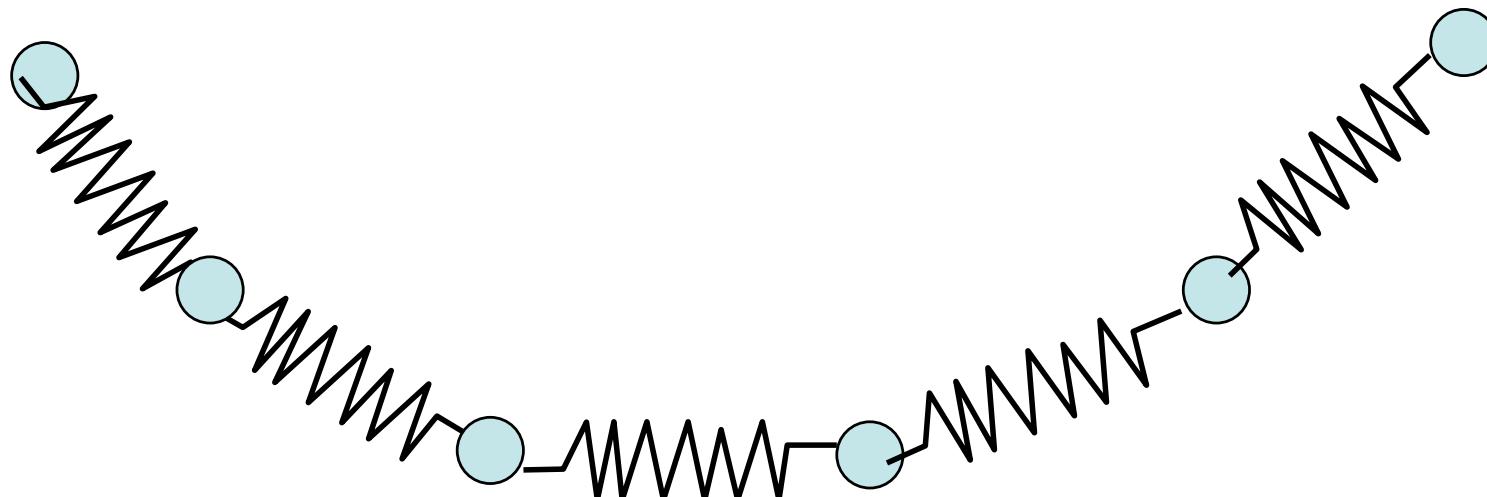


This is the force on  $P_j$ .  
Remember Newton:  $P_i$  experiences force of equal magnitude but opposite direction.

# How Would You Simulate a String?

---

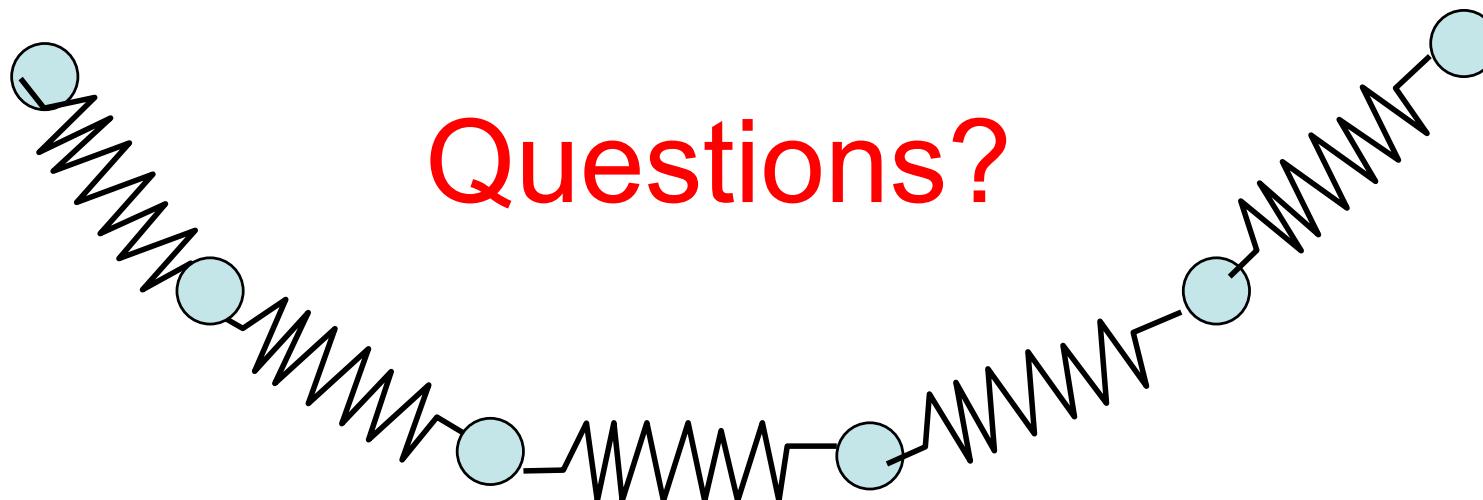
- Springs link the particles
- Springs try to keep their rest lengths and preserve the length of the string
- Not exactly preserved though, and we get oscillation
  - Rubber band approximation



# How Would You Simulate a String?

---

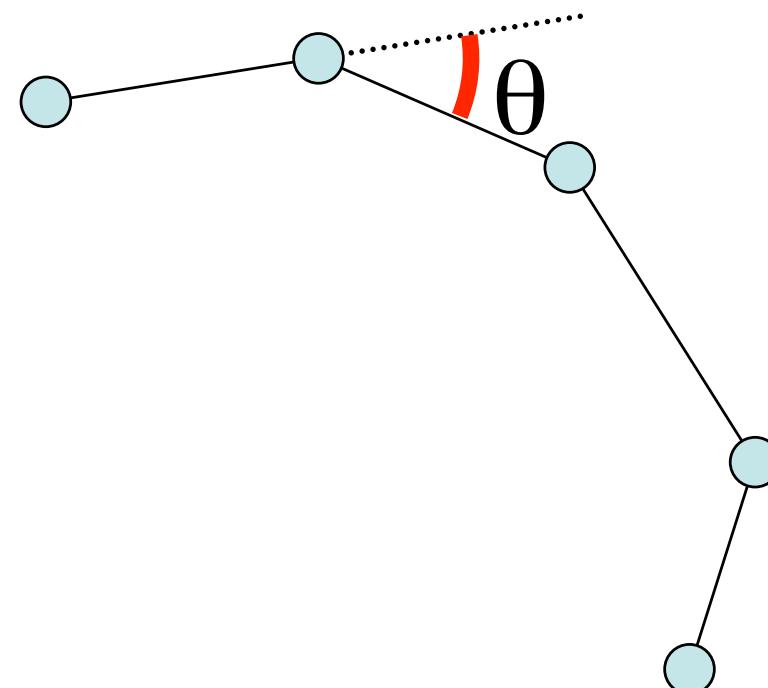
- Springs link the particles
- Springs try to keep their rest lengths and preserve the length of the string
- Not exactly preserved though, and we get oscillation
  - Rubber band approximation



# Hair

---

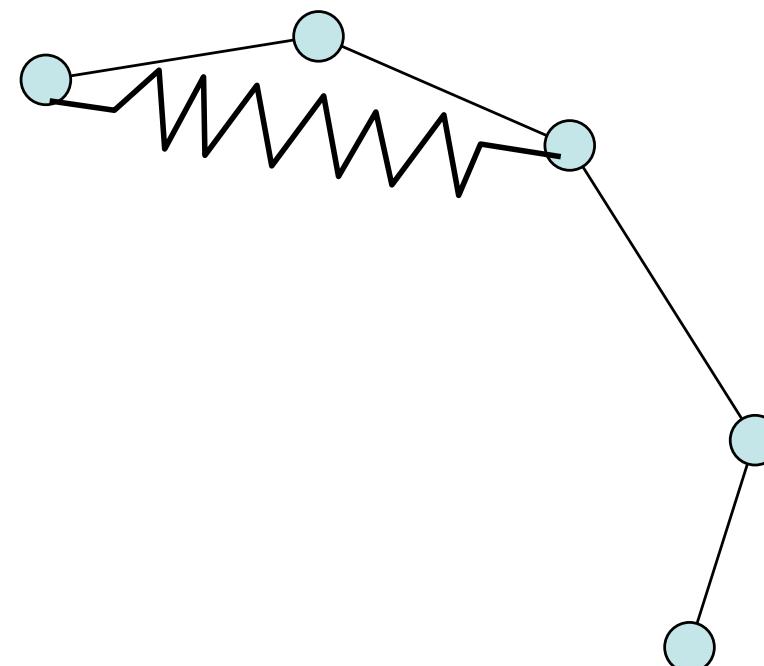
- Linear set of particles
- Length-preserving structural springs like before
- Deformation forces proportional to the angle between segments
- External forces



# Hair - Alternative Structural Forces

---

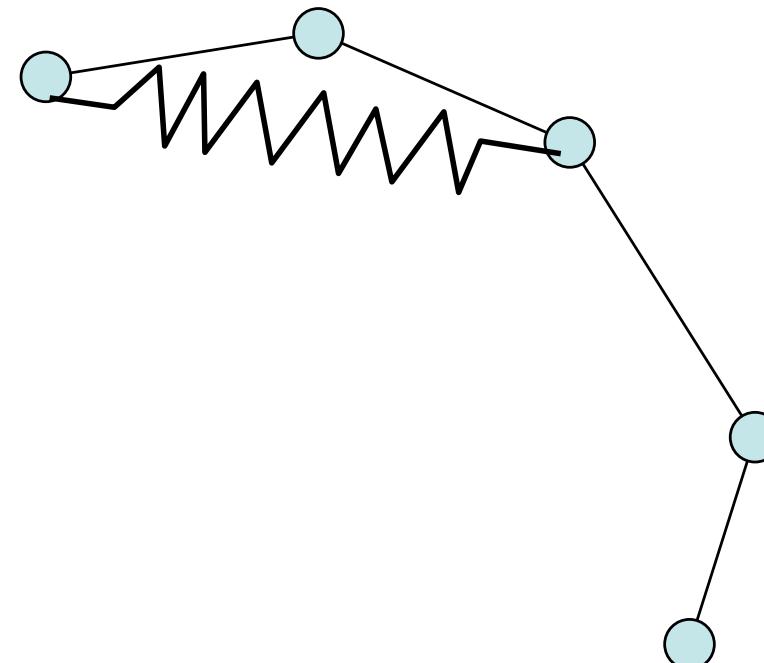
- Springs between mass  $n$  &  $n+2$  with rest length  $2L_0$ 
  - Wants to keep particles aligned



# Hair - Alternative Structural Forces

---

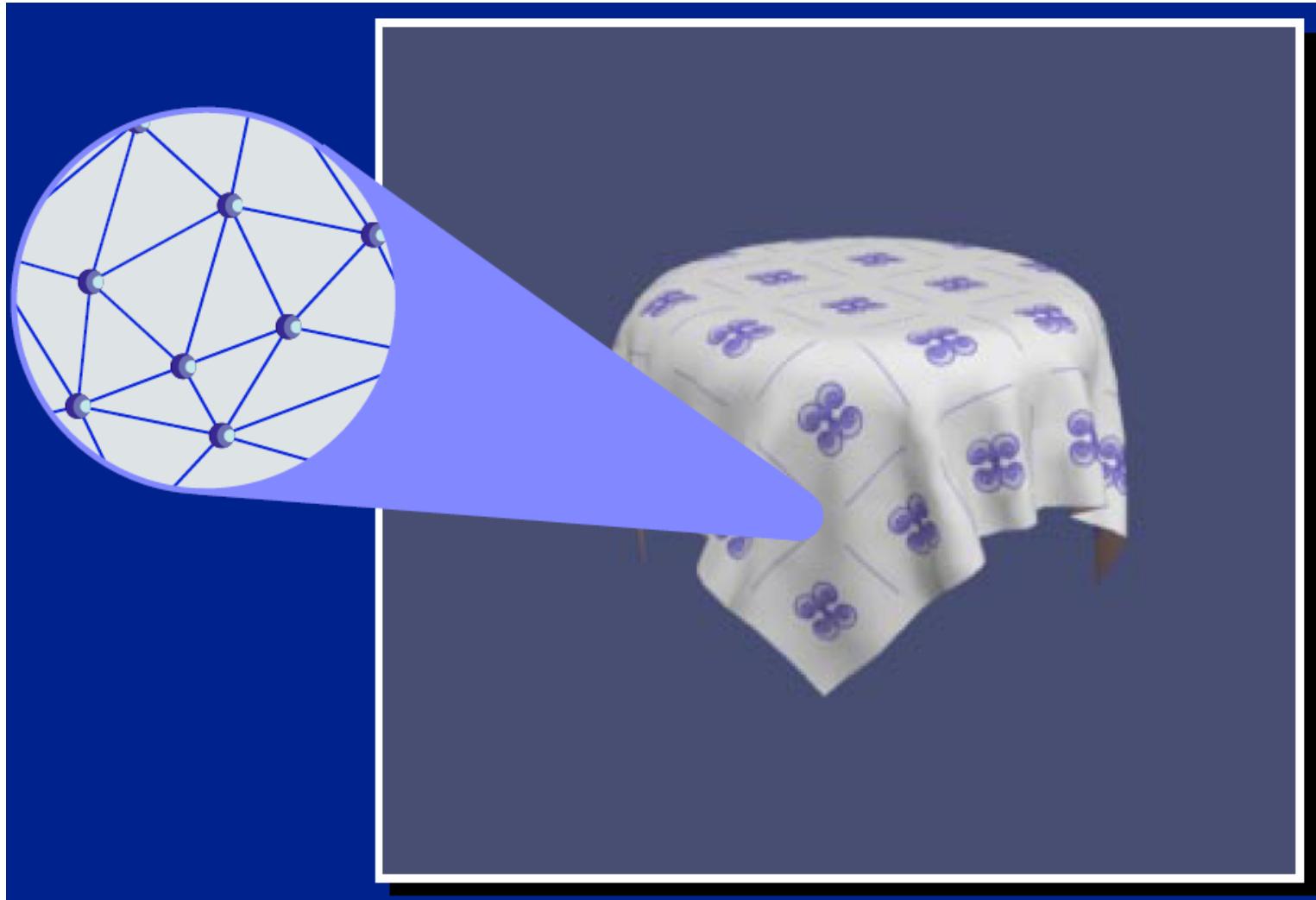
- Springs between mass  $n$  &  $n+2$  with rest length  $2L_0$ 
  - Wants to keep particles aligned



Questions?

# Mass-Spring Cloth

---

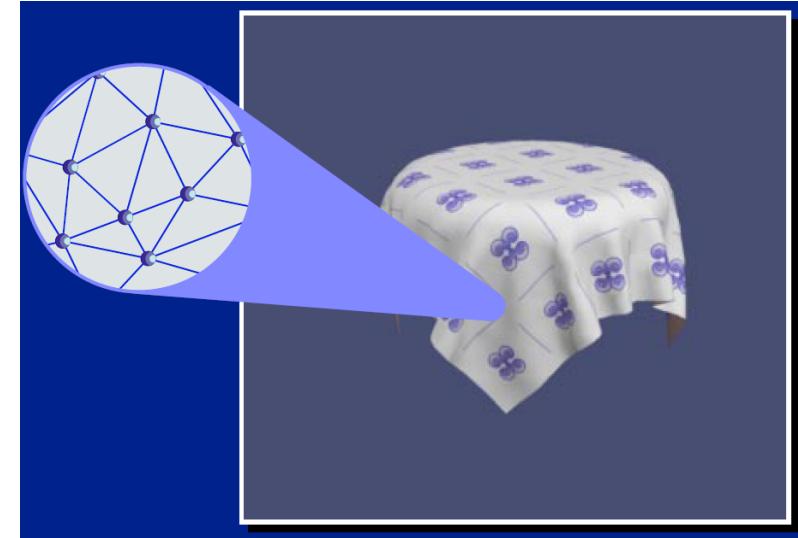


Michael Kass

# Cloth – Three Types of Forces

---

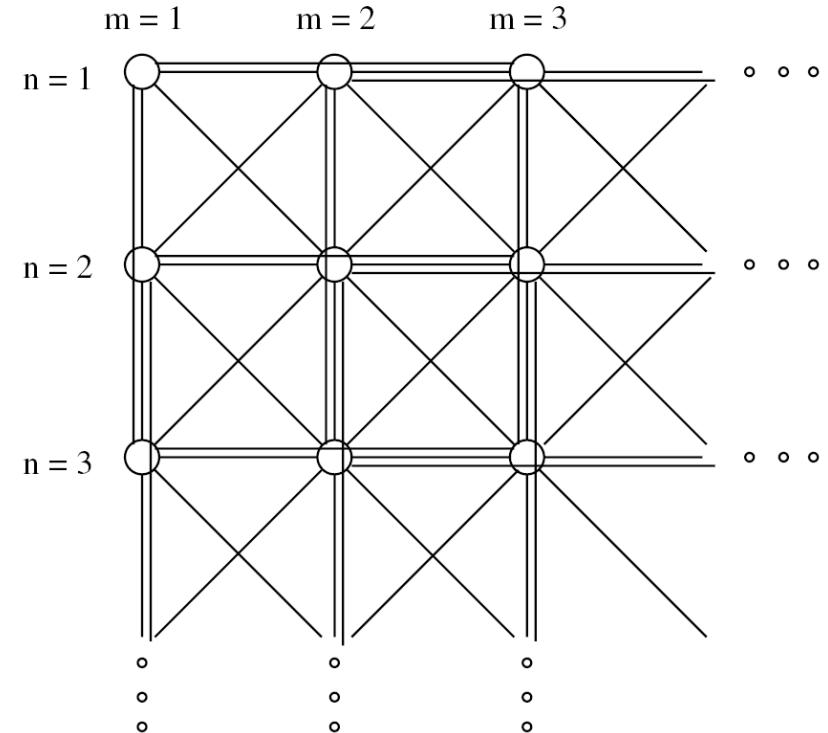
- Structural forces
  - Try to enforce invariant properties of the system
    - E.g. force the distance between two particles to be constant
  - Ideally, these should be constraints, not forces
- Internal deformation forces
  - E.g. a string deforms, a spring board tries to remain flat
- External forces
  - Gravity, etc.



# Springs for Cloth

---

- Network of masses and springs
- Structural springs:
  - link  $(i, j)$  and  $(i+1, j)$ ;  
and  $(i, j)$  and  $(i, j+1)$
- Deformation:
  - Shear springs
    - $(i, j)$  and  $(i+1, j+1)$
  - Flexion springs (prevent bending)
    - $(i, j)$  and  $(i+2, j)$ ;  
 $(i, j)$  and  $(i, j+2)$
- See Provot's Graphics Interface '95 paper for details



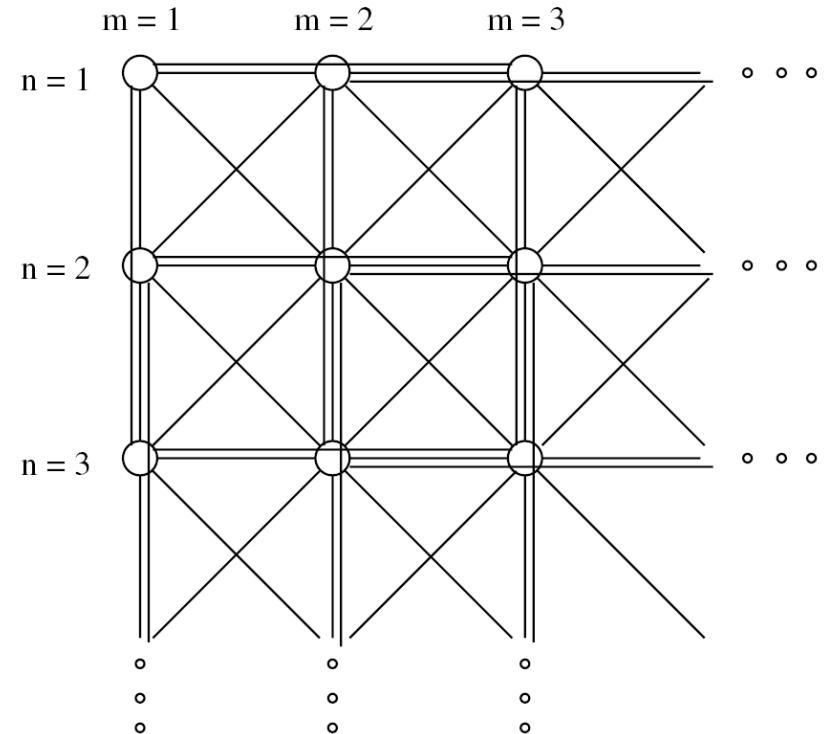
Provot 95

<http://graphics.stanford.edu/courses/cs468-02-winter/Papers/Rigidcloth.pdf>

# External Forces

---

- Gravity G
- Friction
- Wind, etc.

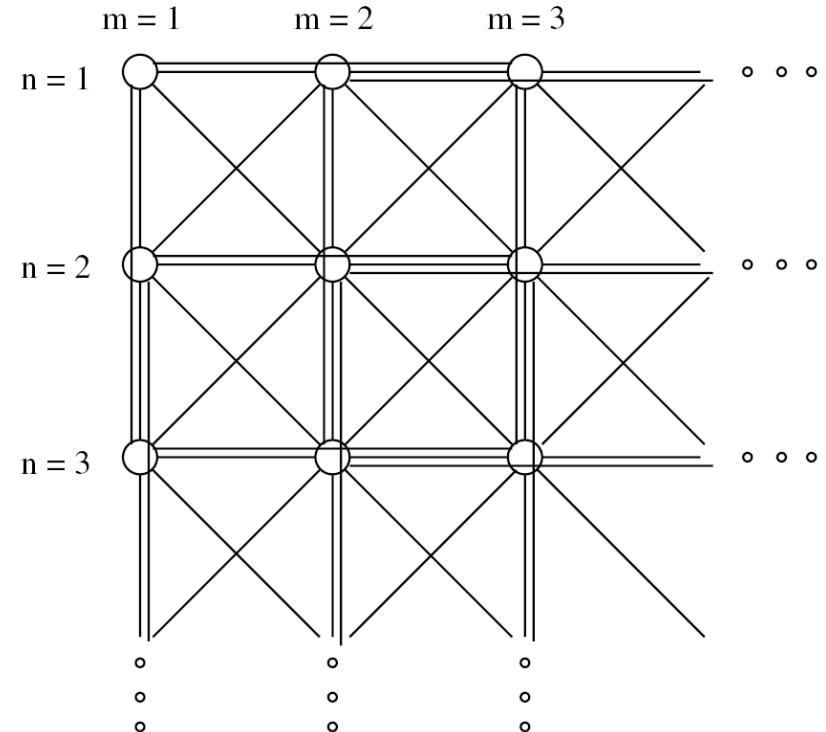


Provot 95

# Cloth Simulation

---

- Then, the all trick is to set the stiffness of all springs to get realistic motion!
- Remember that forces depend on other particles (coupled system)
- But it is sparse (only near neighbors)
  - This is in contrast to e.g. the N-body problem.

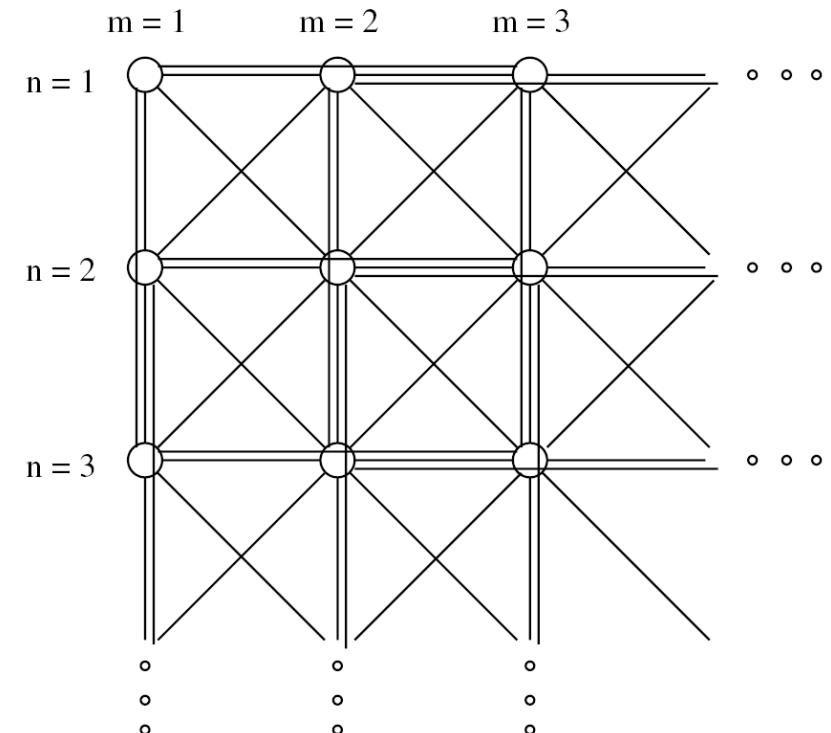


Provot 95

# Forces: Structural vs. Deformation

---

- Structural forces are here just to enforce a constraint
- Ideally, the constraint would be enforced strictly
  - at least a lot more than we can afford
- We'll see that this is the root of a lot of problems
- In contrast, deformation forces actually correspond to physical forces



Provot 95

# Practical Issues

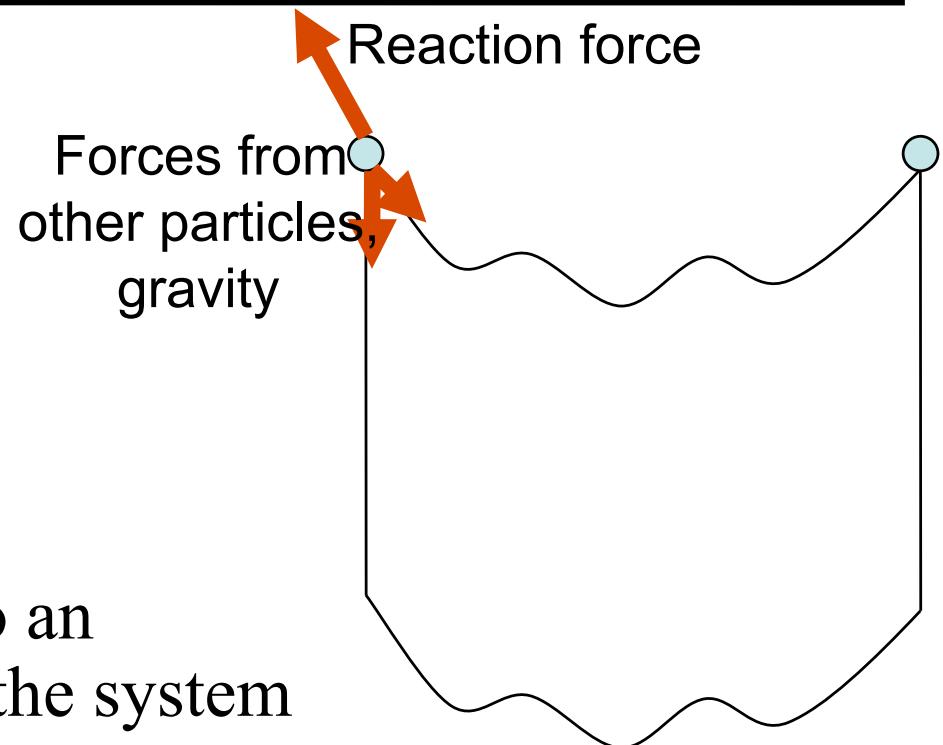
---

- Contact forces
- Discretization problem
- Stiffness

# Contact Forces

---

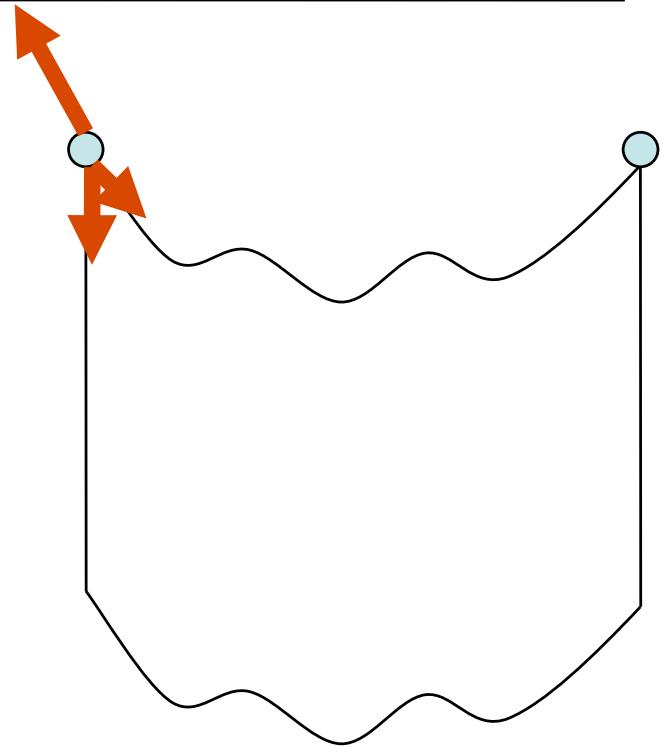
- Hanging curtain:
  - 2 contact points stay fixed
- What does it mean?
  - Sum of the forces is zero
- How so?
  - Because those points undergo an external force that balances the system
- What is the force at the contact?
  - Depends on all other forces in the system
  - Gravity, wind, etc.



# Contact Forces

---

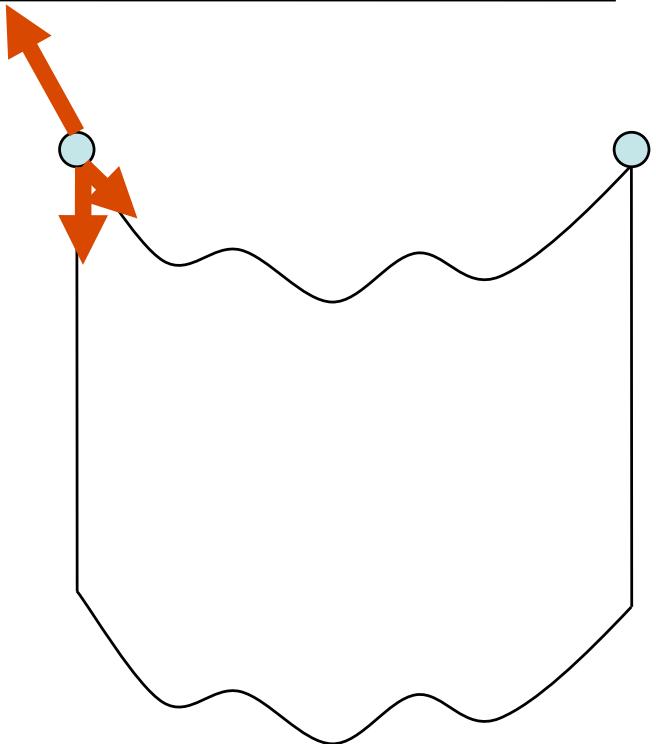
- How can we compute the external contact force?
  - Inverse dynamics!
  - Sum all other forces applied to point
  - Take negative
- Do we really need to compute this force?
  - Not really, just ignore the other forces applied to this point!



# Contact Forces

---

- How can we compute the external contact force?
  - Inverse dynamics!
  - Sum all other forces applied to point
  - Take negative
- Do we really need to compute this force?
  - Not really, just ignore the other forces applied to this point!

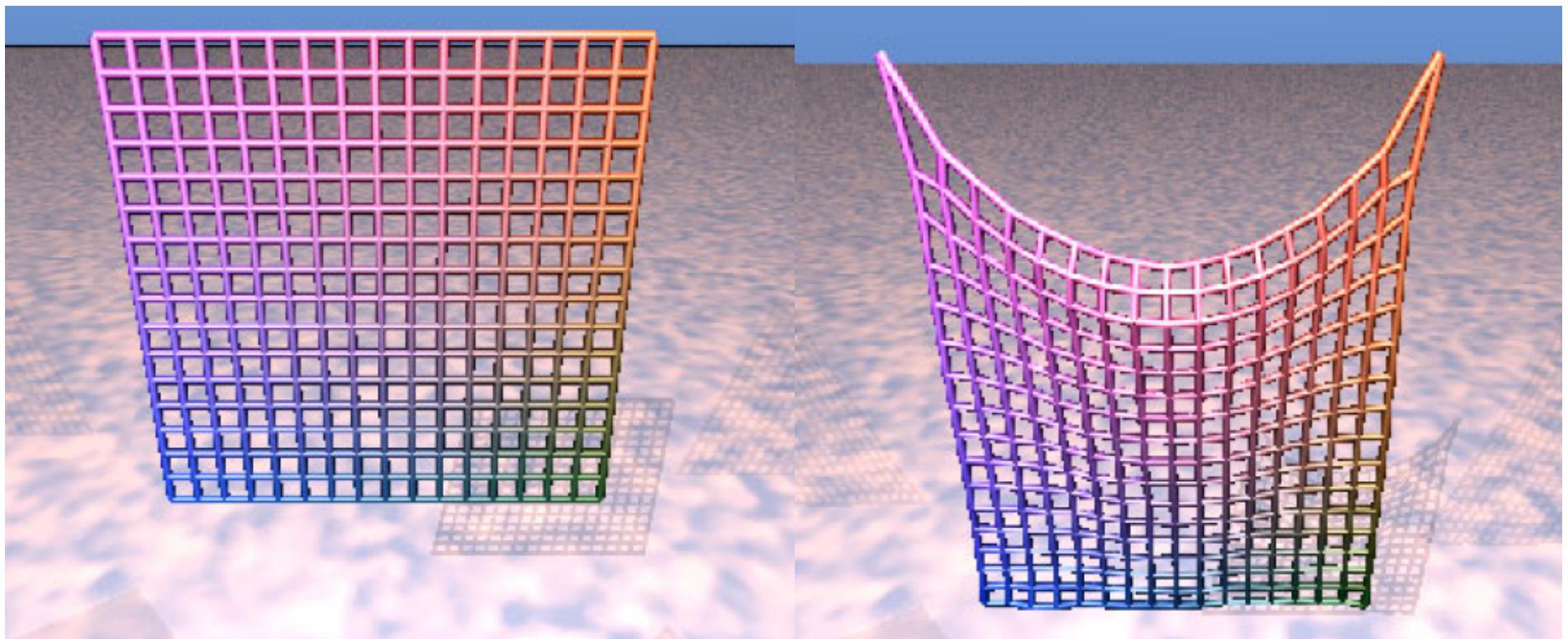


Questions?

# Example

---

- Excessive rubbery deformation:  
the strings are not stiff enough



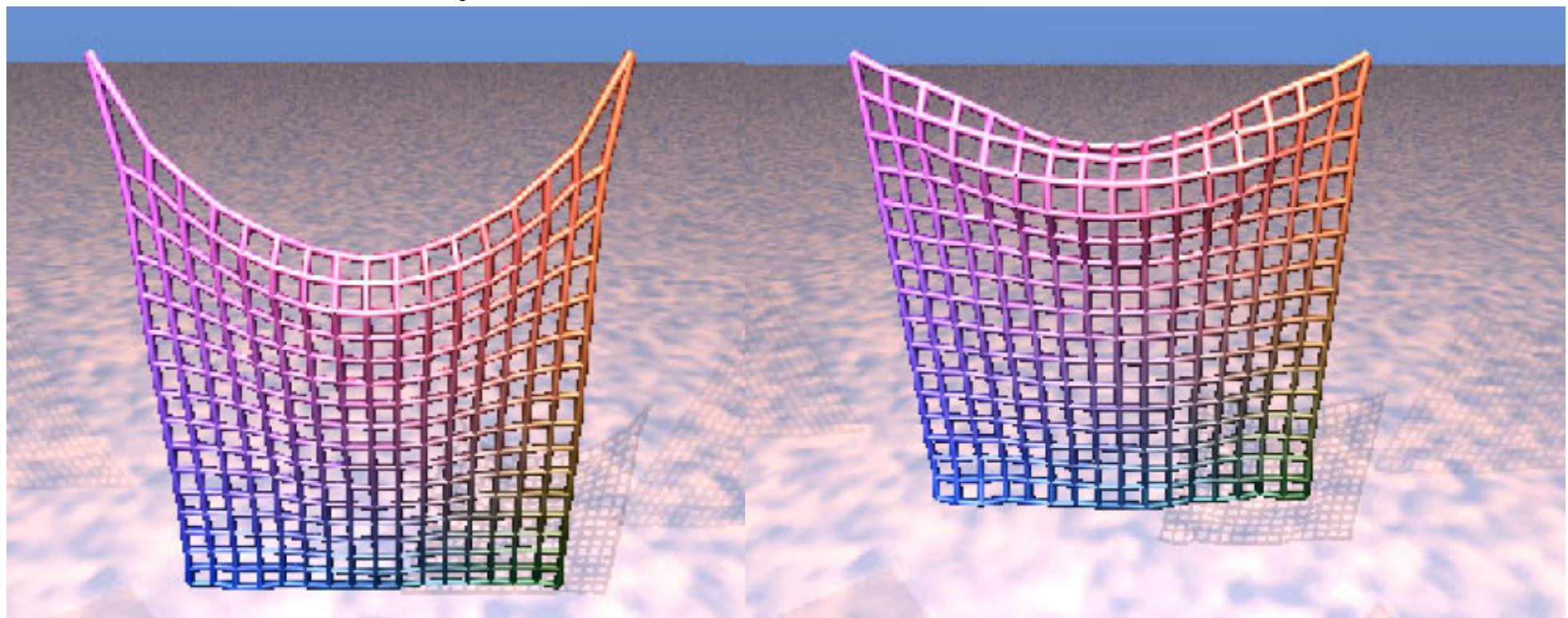
Initial position

After 200 iterations

# One Solution

---

- Constrain length to increase by less than 10%
  - A little hacky



Simple mass-spring system

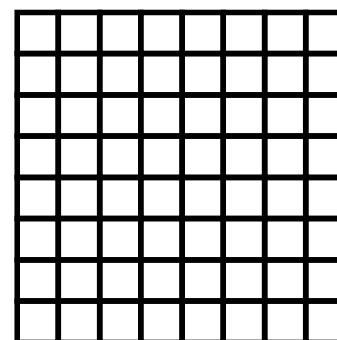
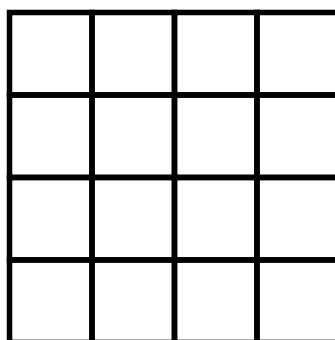
Improved solution  
(see Provot Graphics Interface 1995)

[http://citeseer.ist.psu.edu/  
provot96deformation.html](http://citeseer.ist.psu.edu/provot96deformation.html)

# The Discretization Problem

---

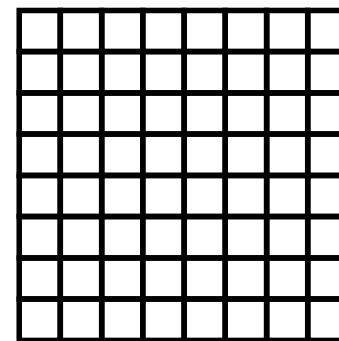
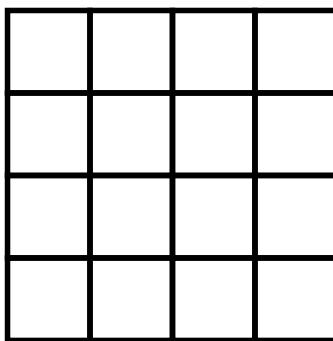
- What happens if we discretize our cloth more finely?
- Do we get the same behavior?
- Usually not! It takes a lot of effort to design a scheme that is mostly oblivious to the discretization.



# The Discretization Problem

---

- What happens if we discretize our cloth more finely?
- Do we get the same behavior?
- Usually not! It takes a lot of effort to design a scheme that is mostly oblivious to the discretization.



Questions?

# The Stiffness Issue

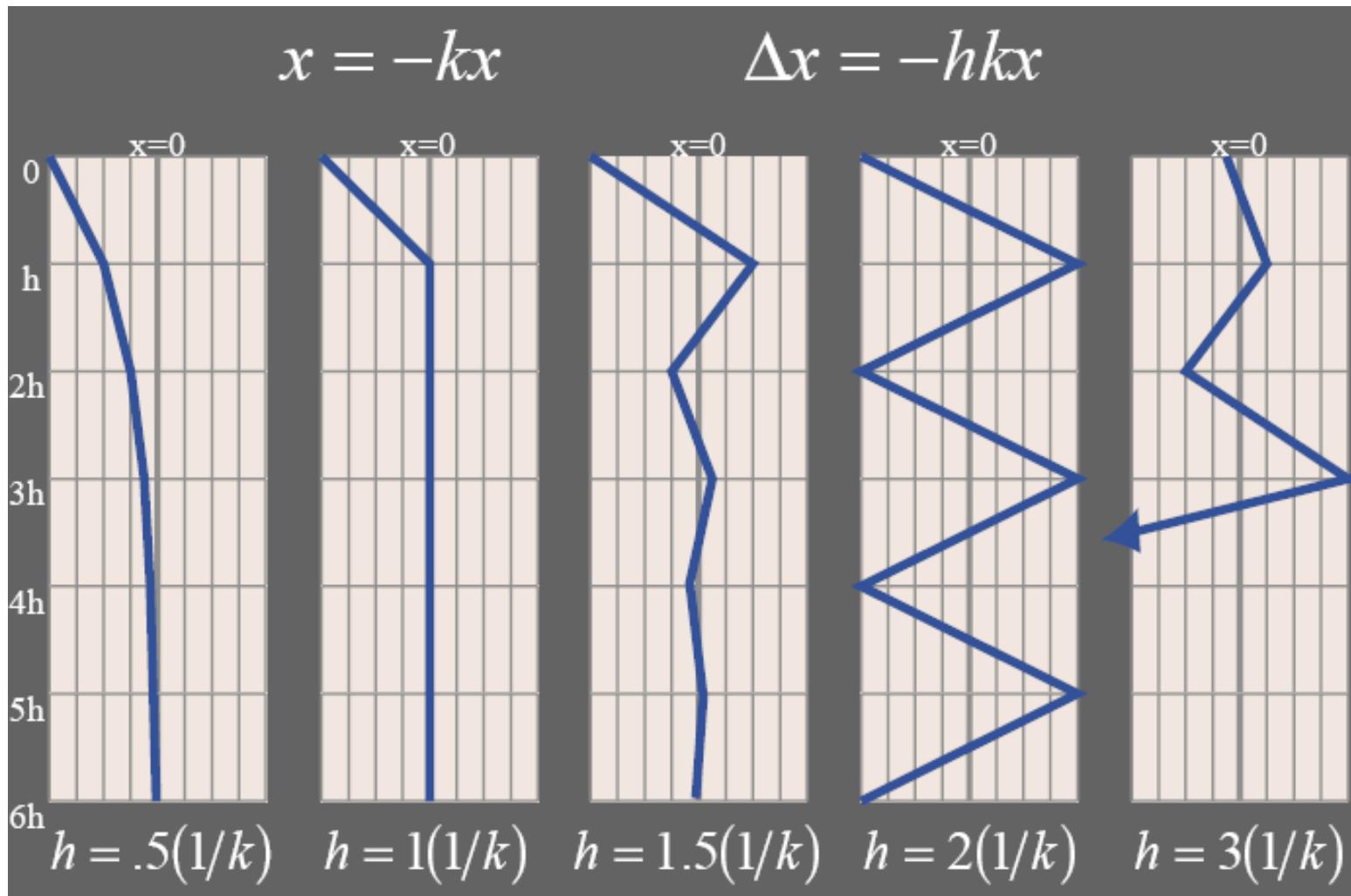
---

- We use springs while we really mean constraint
  - Spring should be super stiff, which requires tiny  $\Delta t$
  - Remember  $x' = -kx$  system and Euler speed limit!
    - The story extends to  $N$  particles and springs (unfortunately)
- Many numerical solutions
  - Reduce  $\Delta t$  (well, not a great solution)
  - Actually use constraints
  - Implicit integration scheme

# Euler Has a Speed Limit!

---

- $h > 1/k$ : oscillate.  $h > 2/k$ : explode!

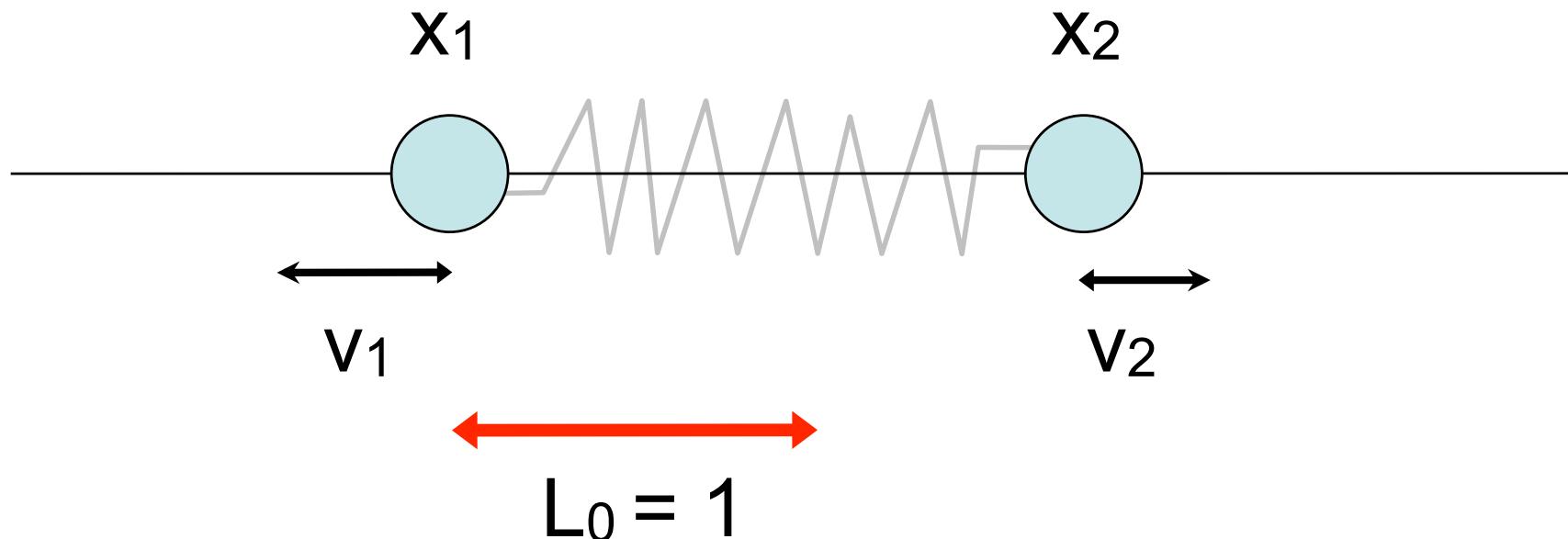


From the SIGGRAPH PBM notes

# Why Stiff Springs Are Difficult

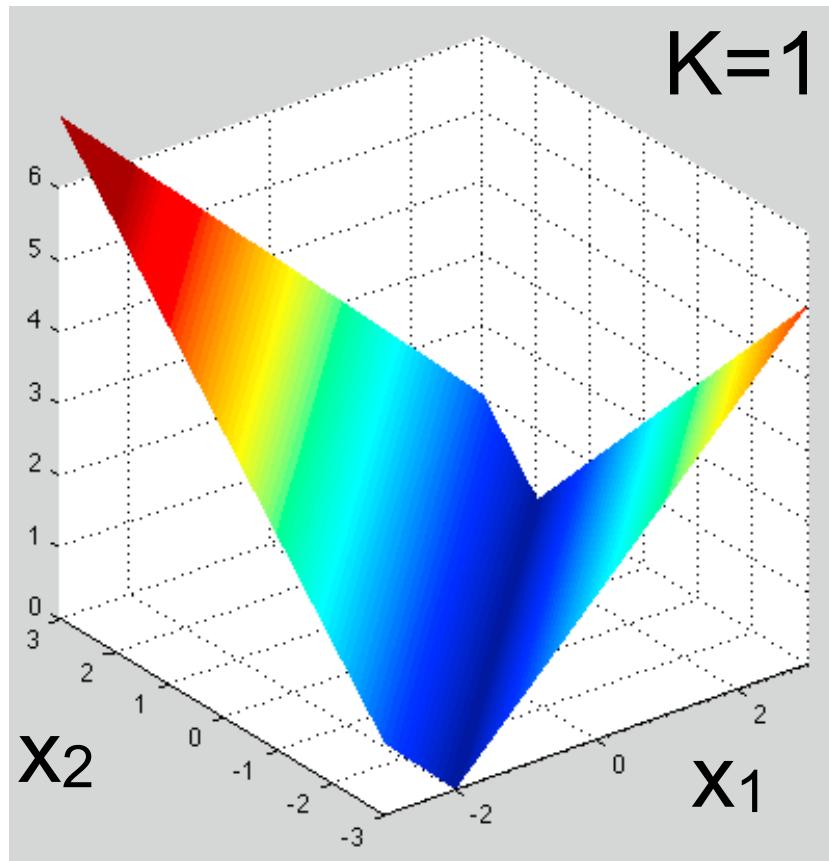
---

- 1D example, with two particles constrained to move along the x axis only, rest length  $L_0 = 1$
- Phase space is 4D:  $(x_1, v_1, x_2, v_2)$ 
  - Although spring force only depends on  $x_1$ ,  $x_2$  and  $L_0$ .

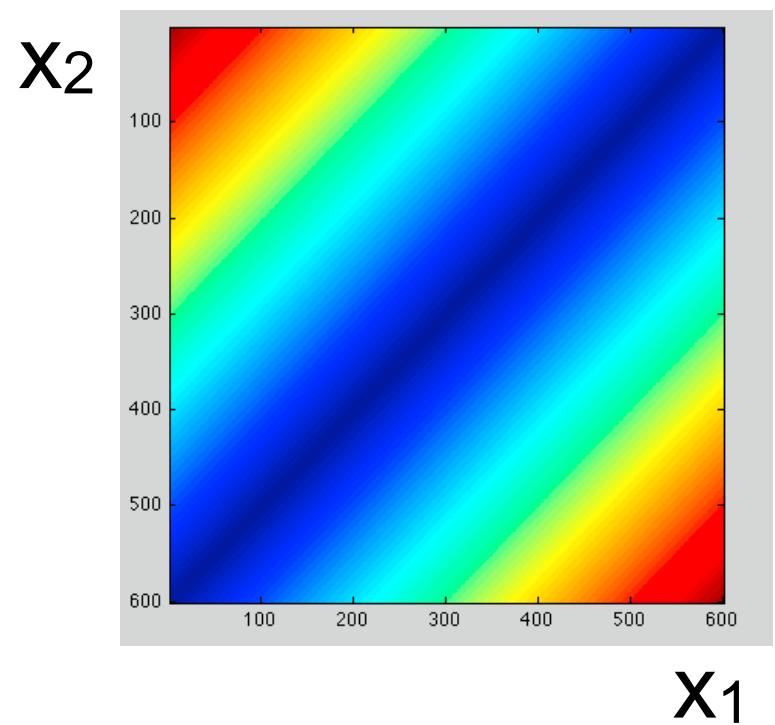


# Why Stiff Springs Are Difficult

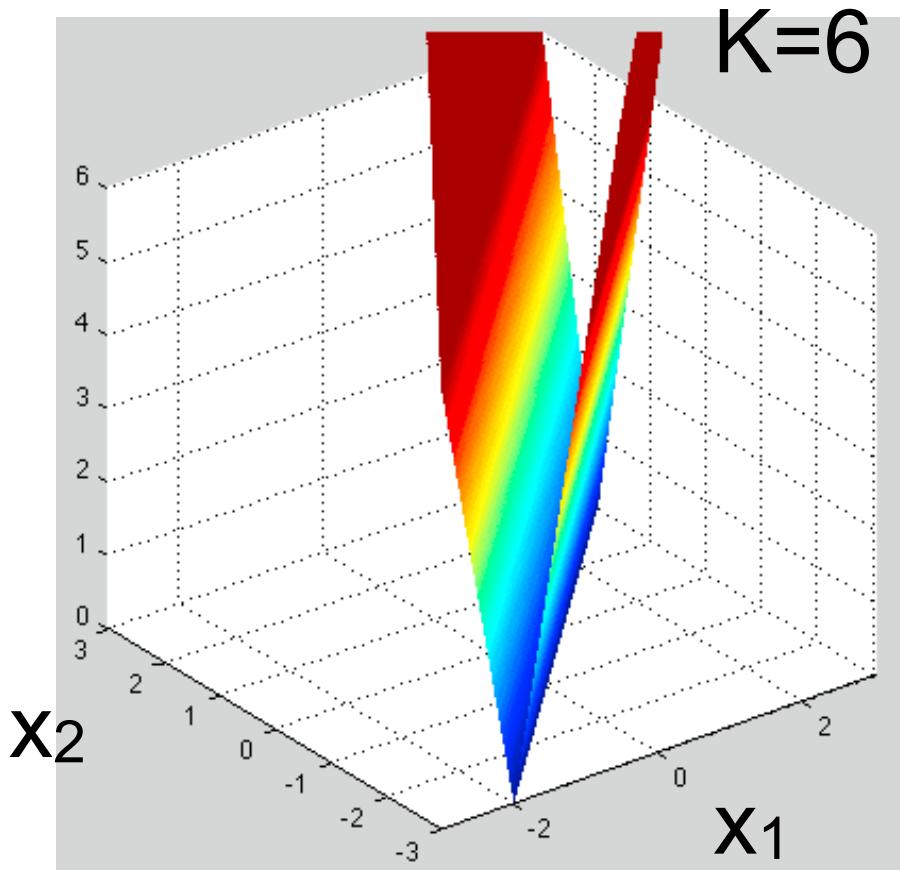
---



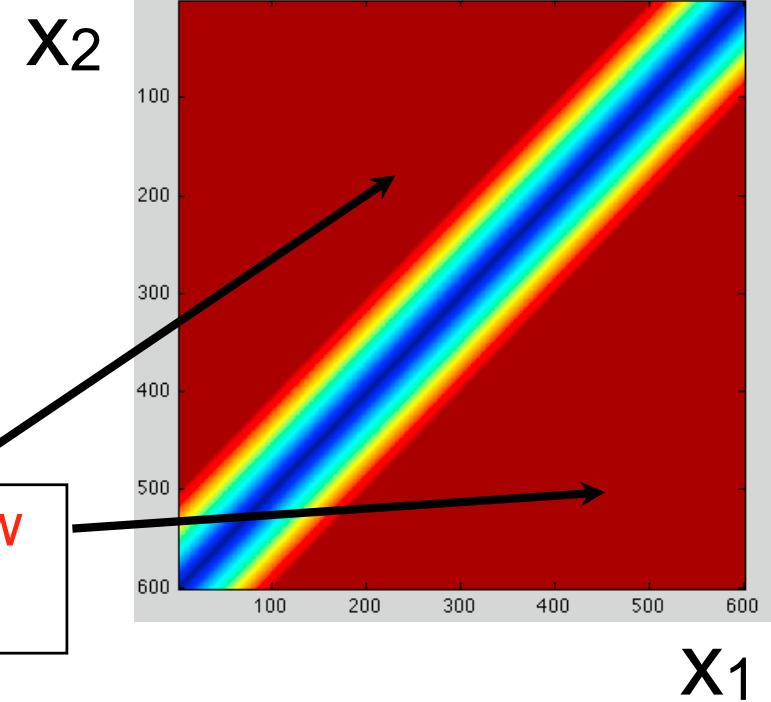
height=magnitude  
of spring force



# Why Stiff Springs Are Difficult

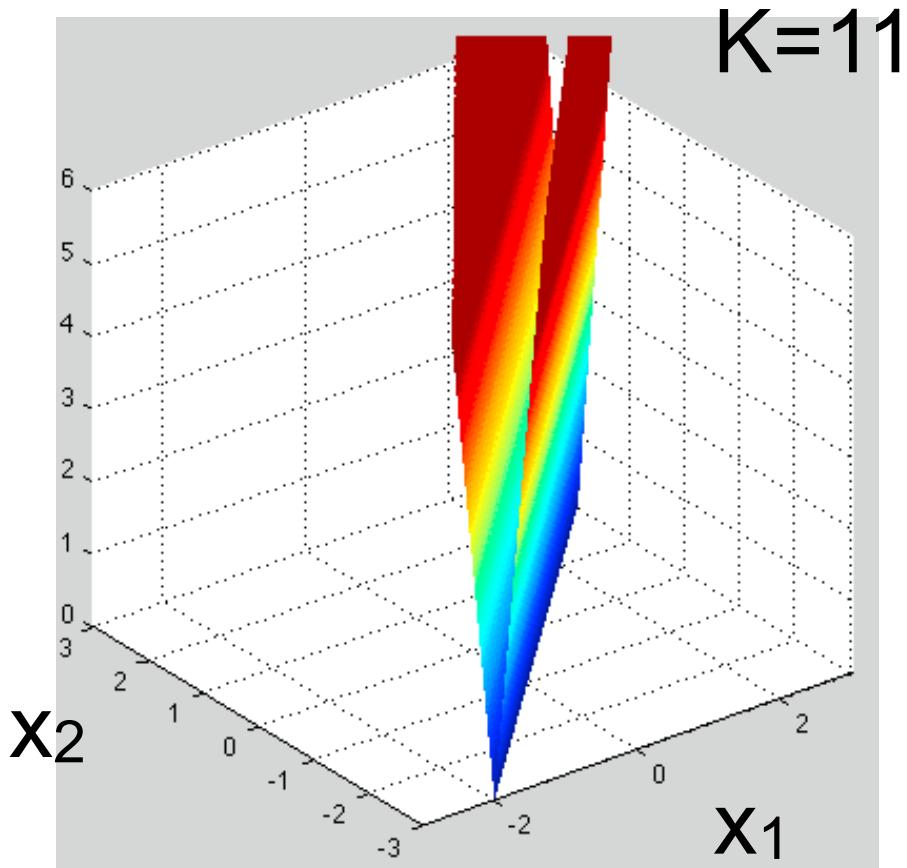


height=magnitude  
of spring force



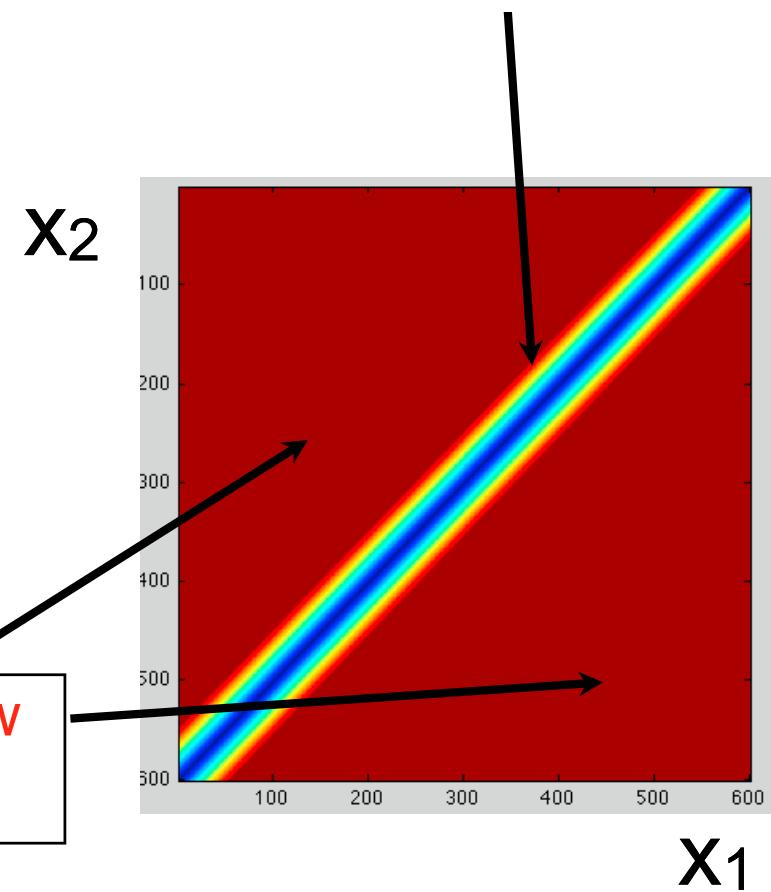
Forces grow  
really big!

# Why Stiff Springs Are Difficult

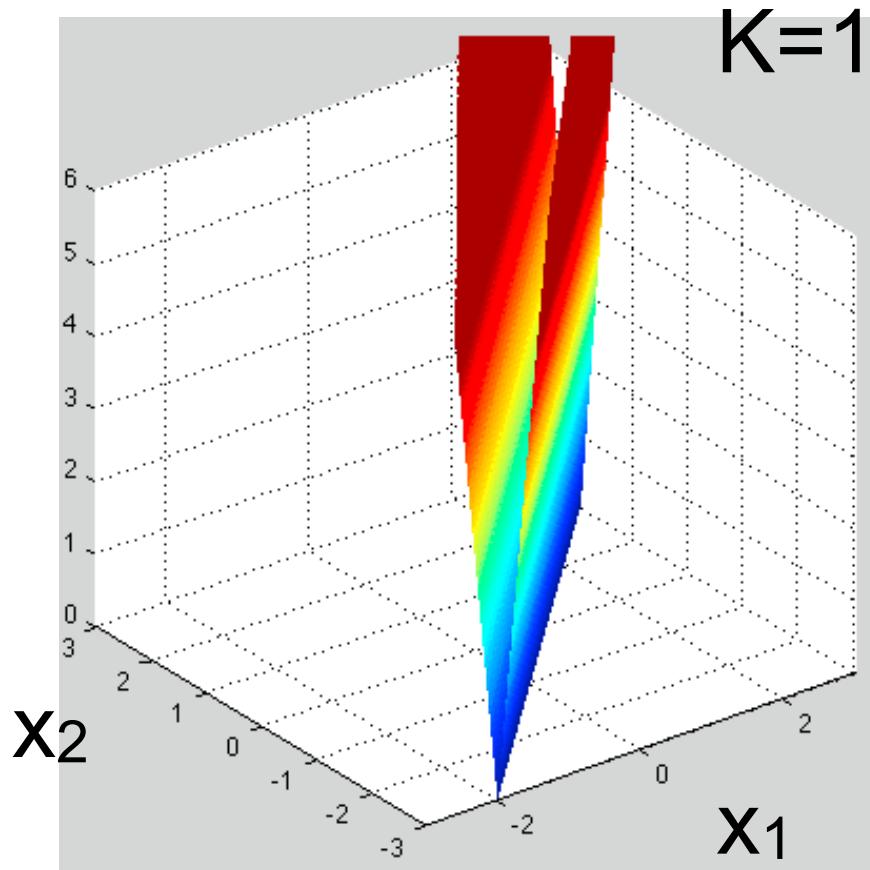


Forces grow  
really big!

The “admissible region”  
shrinks towards the line  
 $x_1 - x_2 = 1$  as  $K$  grows

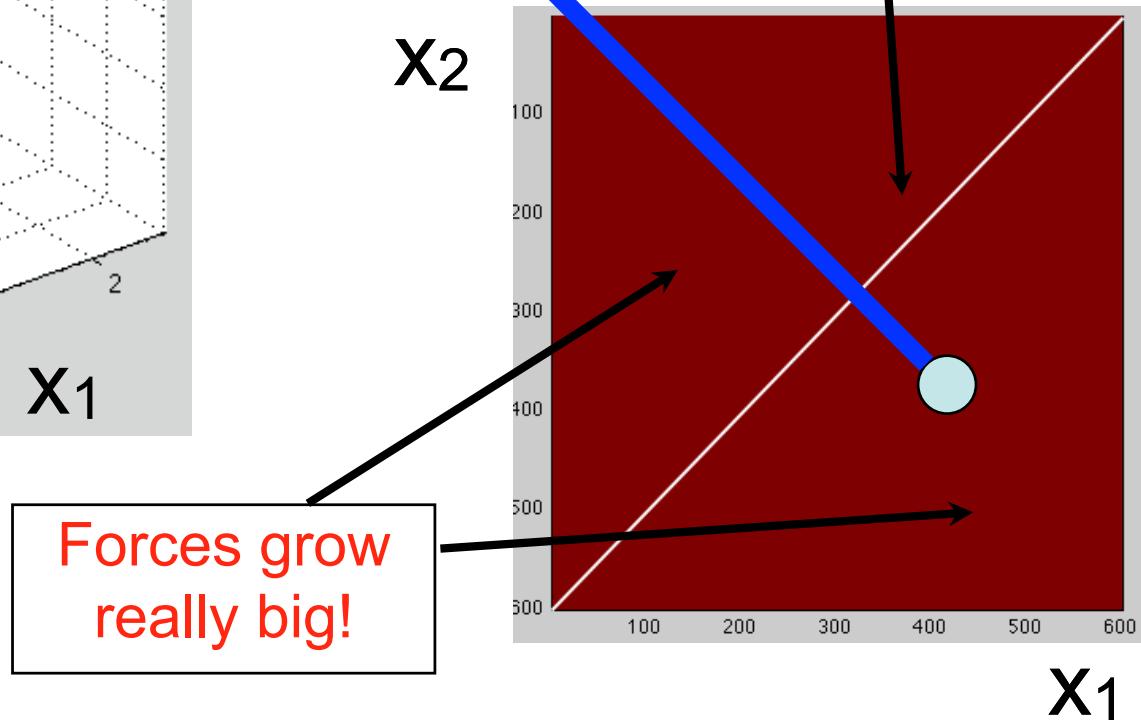


# Why Stiff Springs Are Difficult



*off to the moon!*

The “admissible region”  
shrinks towards the line  
 $x_1-x_2=1$  as  $K$  grows



Forces grow  
really big!

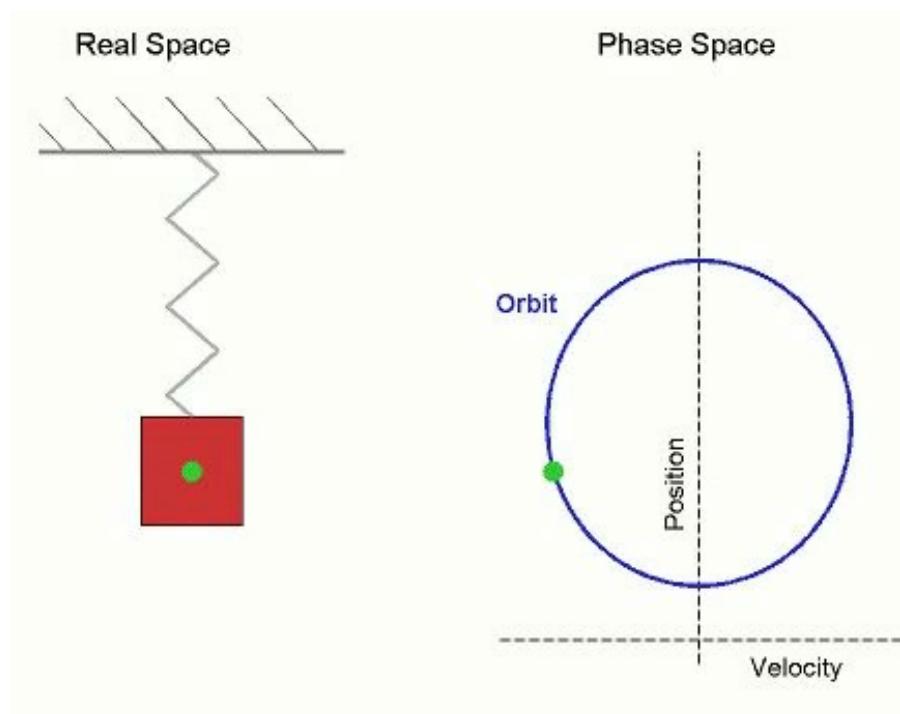
$x_1$

# Mass on a Spring, Phase Space

---

- State of system (phase) : velocity & position
  - similar to our  $X=(x\ v)$  to get 1st order

Wikipedia user Mazemaster

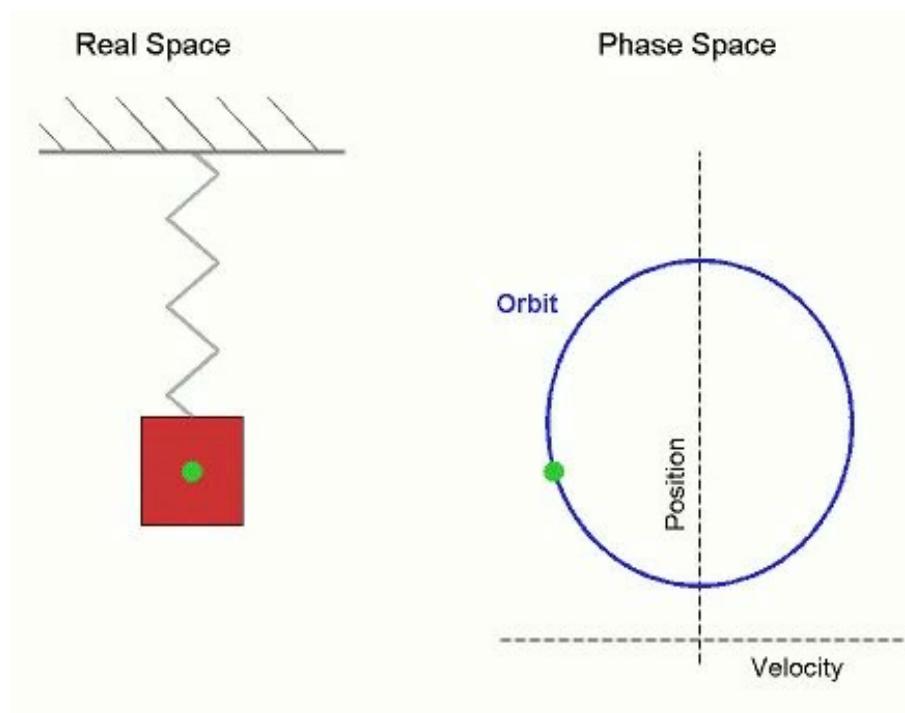


# Mass on a Spring, Phase Space

---

- Guess how well Euler will do...  
always diverge

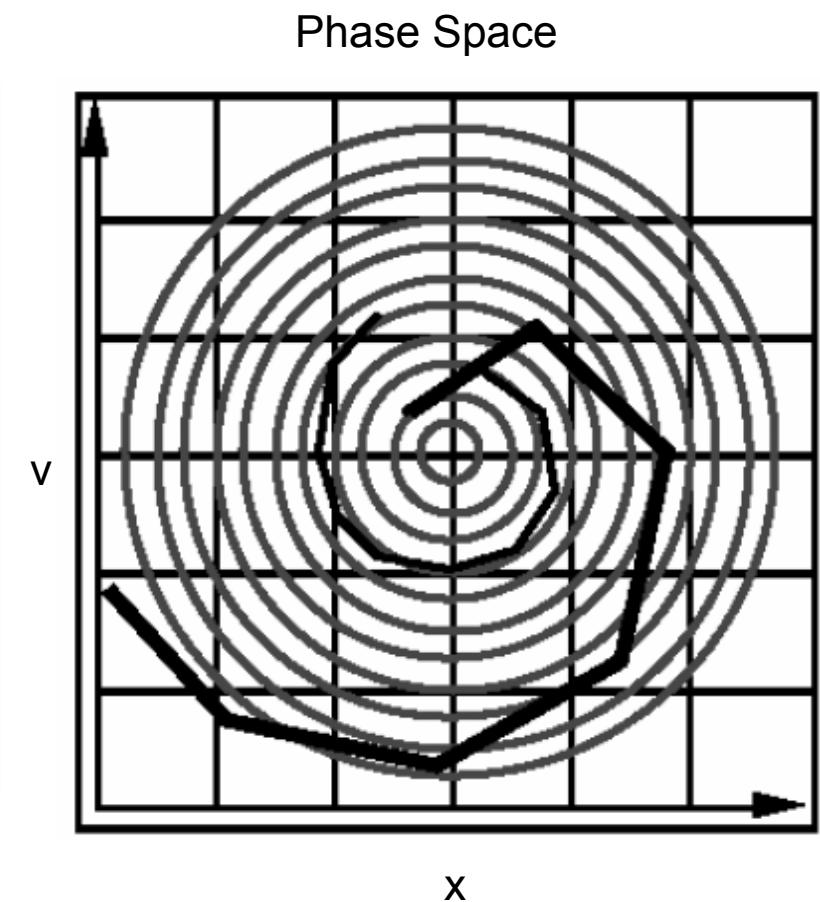
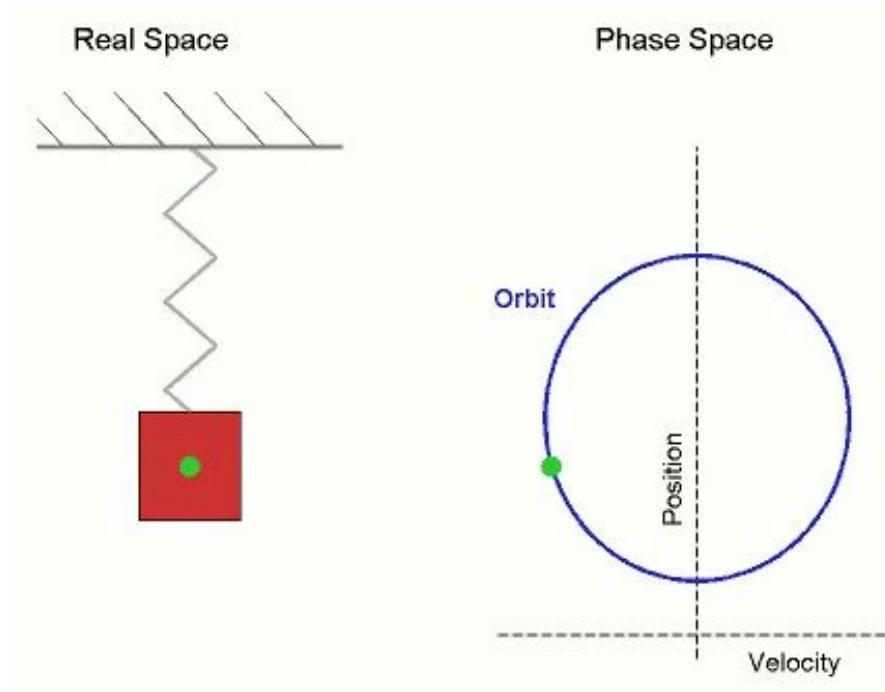
Wikipedia user Mazemaster



# Mass on a Spring, Phase Space

- Guess how well Euler will do...  
always diverge

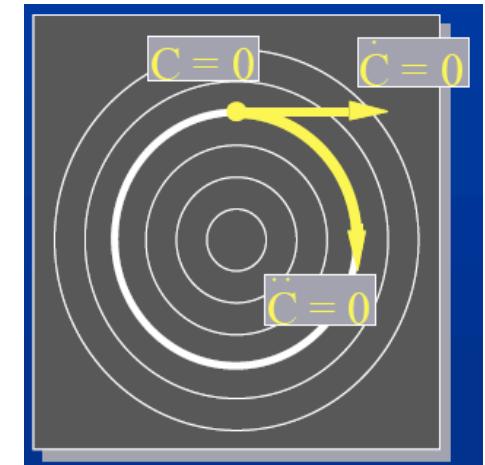
Wikipedia user Mazemaster



# Constrained Dynamics (Advanced)

---

- In our mass-spring cloth, we have “encouraged” length preservation using springs that want to have a given length
- Constrained dynamic simulation: force it to be constant!
- How it works
  - Start with constraint equation
    - E.g.,  $(x_2 - x_1) - 1 = 0$  in the previous 1D example
  - Derive extra forces that will exactly enforce constraint
    - This means projecting the external forces (like gravity) onto the “subspace” of phase space where constraints are satisfied
    - Fancy name for this: “Lagrange multipliers”
  - See the SIGGRAPH 2001 Course Notes for more details



# Questions?

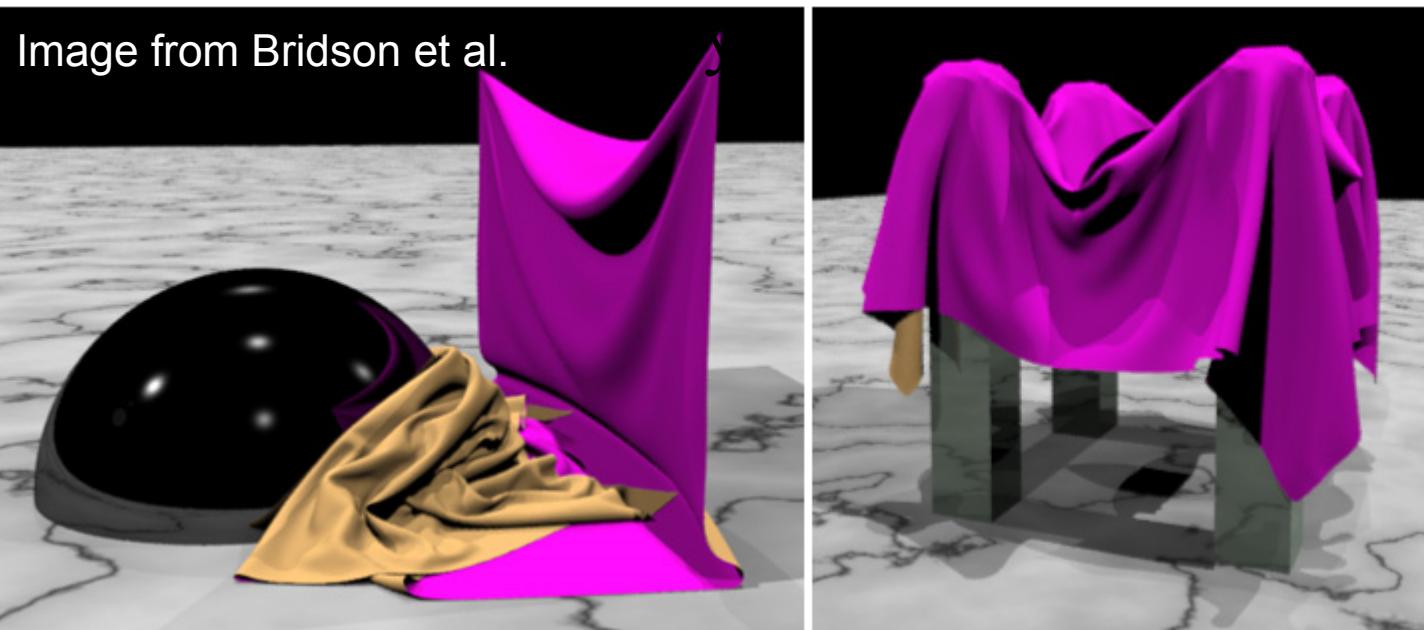
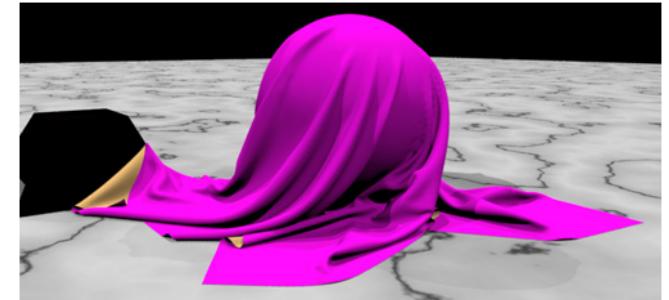
---

- Further reading
  - Stiff systems: [http://en.wikipedia.org/wiki/Stiff\\_equation](http://en.wikipedia.org/wiki/Stiff_equation)
  - Explicit vs. implicit solvers: [http://en.wikipedia.org/wiki/Explicit\\_and\\_implicit\\_methods](http://en.wikipedia.org/wiki/Explicit_and_implicit_methods)
  - Again, consult the SIGGRAPH 2001 course notes!
    - <http://www.pixar.com/companyinfo/research/pbm2001/>

# The Collision Problem

---

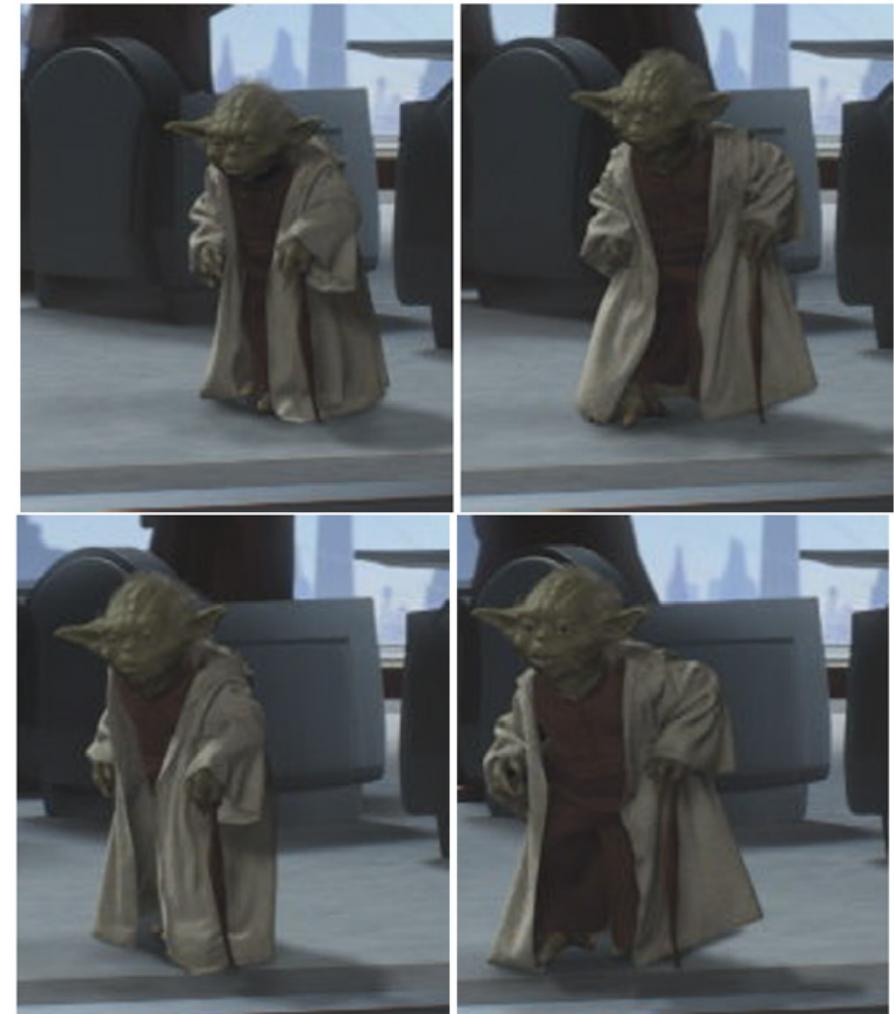
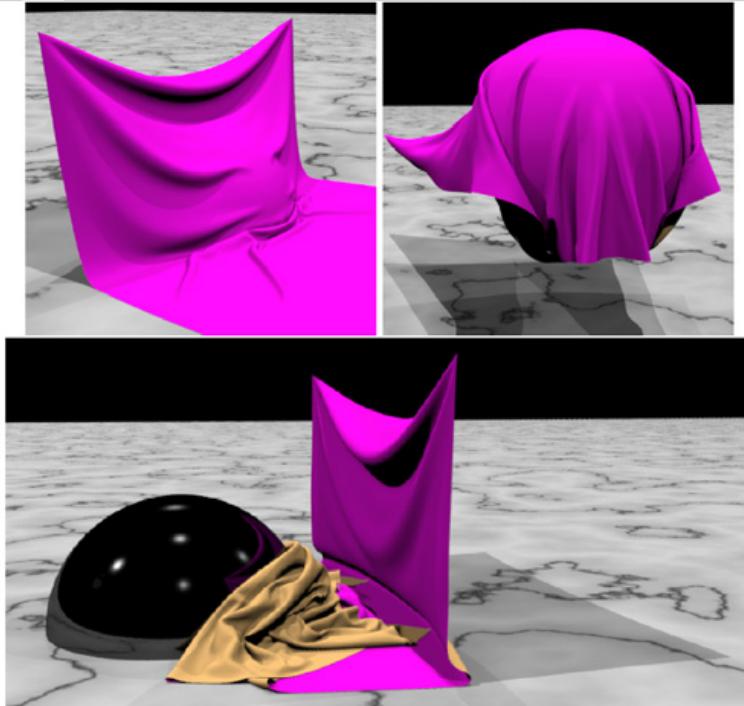
- More advanced topic
- A cloth has many points of contact
- Requires
  - Efficient collision detection
  - Efficient numerical



# Collisions

[Robert Bridson, Ronald Fedkiw & John Anderson](#)  
[Robust Treatment of Collisions, Contact](#)  
[and Friction for Cloth Animation](#)  
[SIGGRAPH 2002](#)

- Cloth has many points of contact
- Need efficient collision detection and stable treatment



# Cool Cloth/Hair Demos

---

- Robert Bridson, Ronald Fedkiw & John Anderson: Robust Treatment of Collisions, Contact and Friction for Cloth Animation SIGGRAPH 2002
- Selle. A, Su, J., Irving, G. and Fedkiw, R., "Robust High-Resolution Cloth Using Parallelism, History-Based Collisions, and Accurate Friction," IEEE TVCG 15, 339-350 (2009).
- Selle, A., Lentine, M. and Fedkiw, R., "A Mass Spring Model for Hair Simulation", SIGGRAPH 2008, ACM TOG 27, 64.1-64.11 (2008).

# Cool Cloth/Hair Demos Questions?

---



- [Selle, A., Su, J., Irving, G. and Fedkiw, R., "Robust High-Resolution Cloth Using Parallelism, History-Based Collisions, and Accurate Friction," IEEE TVCG 15, 339-350 \(2009\).](#)

# Implementation Notes

---

- It pays off to abstract (as usual)
  - It's easy to design your “Particle System” and “Time Stepper” to be unaware of each other
- Basic idea
  - “Particle system” and “Time Stepper” communicate via floating-point vectors  $X$  and a function that computes  $f(X,t)$ 
    - “Time Stepper” does not need to know anything else!

# Implementation Notes

---

- Basic idea
  - “Particle System” tells “Time Stepper” how many dimensions ( $N$ ) the phase space has
  - “Particle System” has a function to write its state to an  $N$ -vector of floating point numbers (and read state from it)
  - “Particle System” has a function that evaluates  $f(X,t)$ , given a state vector  $X$  and time  $t$
  - “Time Stepper” takes a “Particle System” as input and advances its state

# Particle System Class

---

```
class ParticleSystem
{
    virtual int getDimension()
    virtual setDimension(int n)
    virtual float* getStatePositions()
    virtual setStatePositions(float* positions)
    virtual float* getStateVelocities()
    virtual setStateVelocities(float* velocities)
    virtual float* getForces(float* positions, float* velocities)
        virtual setMasses(float* masses)
        virtual float* getMasses()

    float* m_currentState
}
```

# Time Stepper Class

---

```
class TimeStepper
{
    virtual takeStep(ParticleSystem* ps, float h)
}
```

# Forward Euler Implementation

---

```
class ForwardEuler : TimeStepper
{
    void takeStep(ParticleSystem* ps, float h)
    {
        velocities = ps->getStateVelocities()
        positions = ps->getStatePositions()
        forces = ps->getForces(positions, velocities)
        masses = ps->getMasses()
        accelerations = forces / masses
        newPositions = positions + h*velocities
        newVelocities = velocities + h*accelerations
        ps->setStatePositions(newPositions)
        ps->setStateVelocities(newVelocities)
    }
}
```

# Mid-Point Implementation

---

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

        ps->setStatePositions(newPositions)
        ps->setStateVelocities(newVelocities)
    }
}
```

# Particle System Simulation

---

```
ps = new MassSpringSystem(particleCount, masses, springs, externalForces)
stepper = new ForwardEuler()
time = 0
while time < 1000
    stepper->takeStep(ps, 0.0001)
    time = time + 0.0001
    // render
```

# Particle System Simulation

---

```
ps = new MassSpringSystem(particleCount, masses, springs, externalForces)
stepper = new MidPoint()
time = 0
while time < 1000
    stepper->takeStep(ps, 0.0001)
    time = time + 0.0001
    // render
```

# Questions?

---

# That's All for Today!

---

Bungie / ign.com