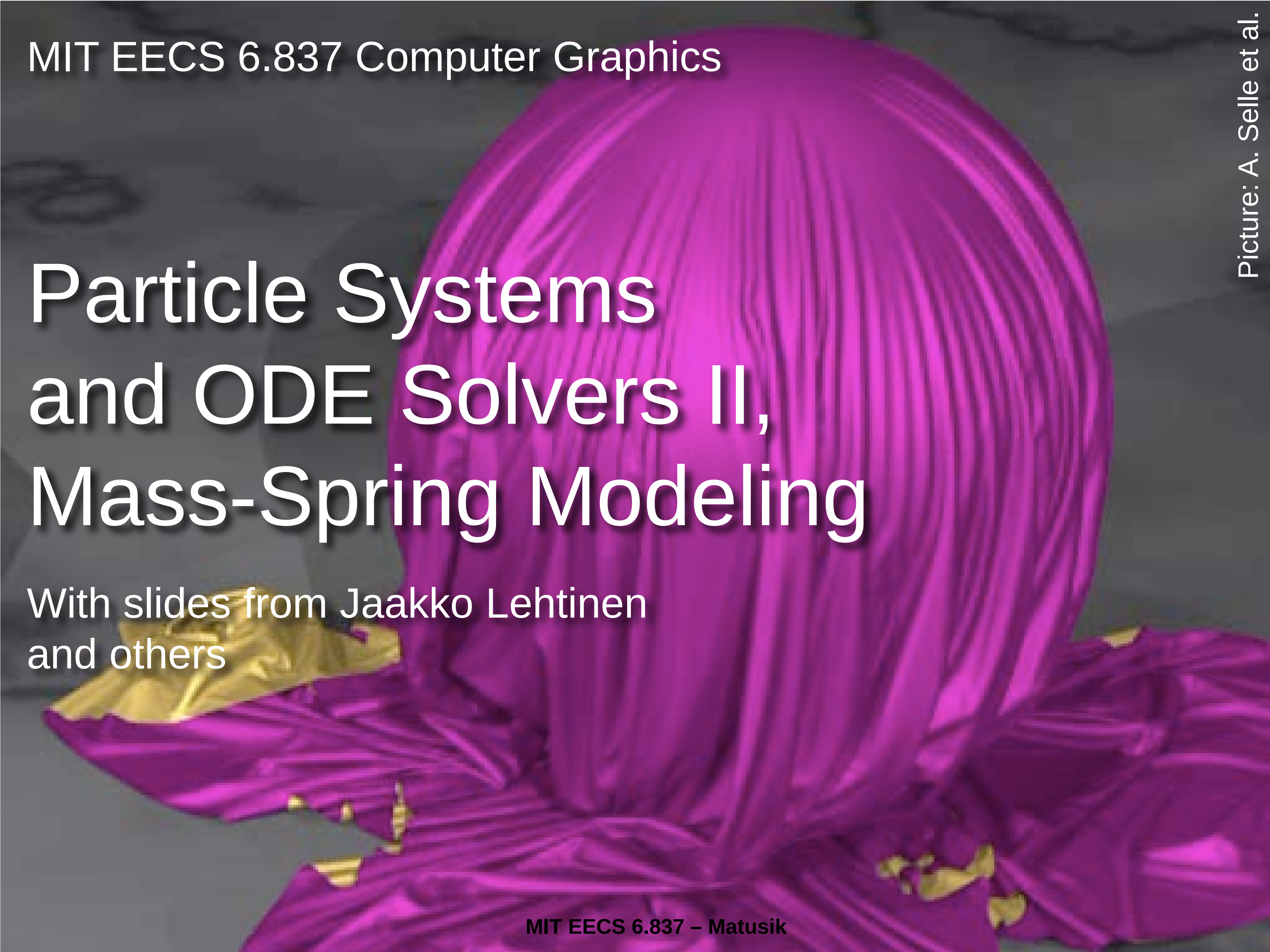
Particle Systems and ODE Solvers II, Mass-Spring Modeling

With slides from Jaakko Lehtinen and others



Picture: A. Selle et al.

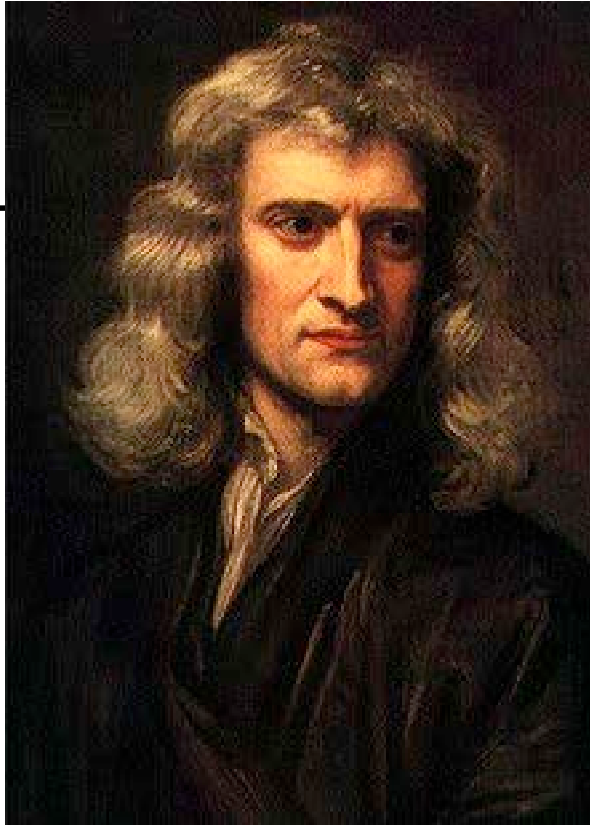
ODEs and Numerical Integration

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

- 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: 2nd order ODE

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

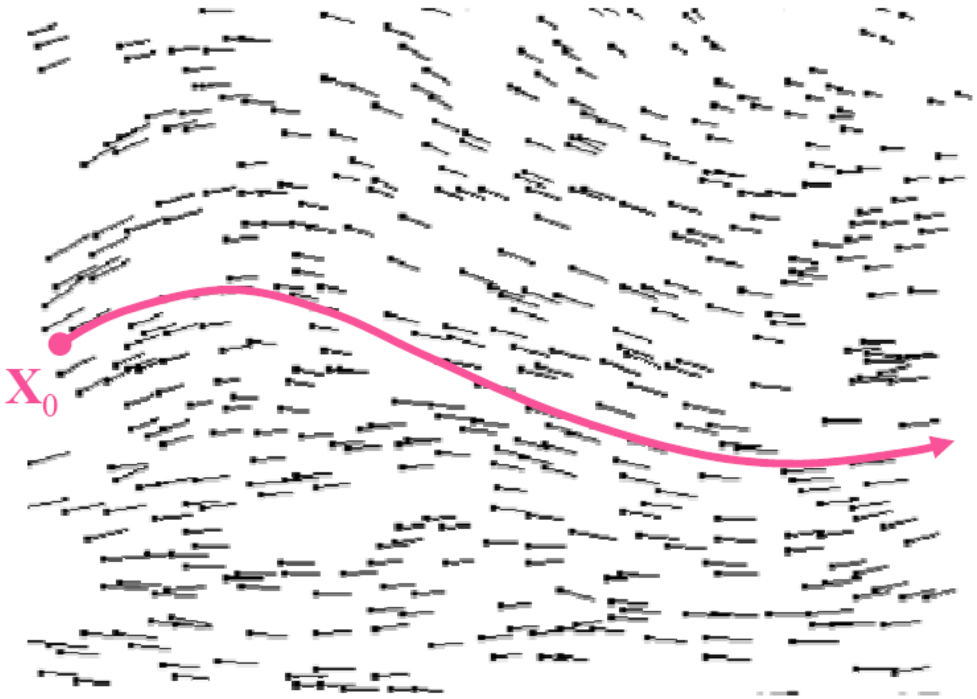
- Corresponds to system of first order ODEs

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

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

ODE: Path Through a Vector Field

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



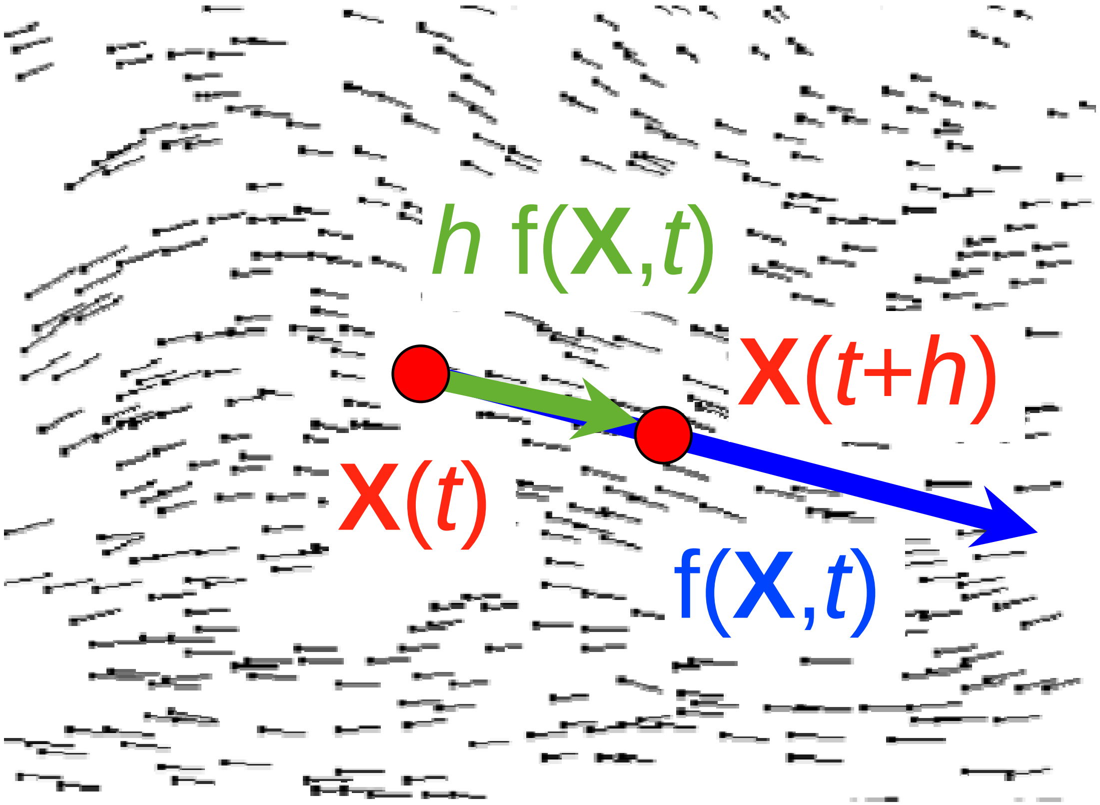
$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X} = f(\boldsymbol{X}, t)$$

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

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

Euler, Visually

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



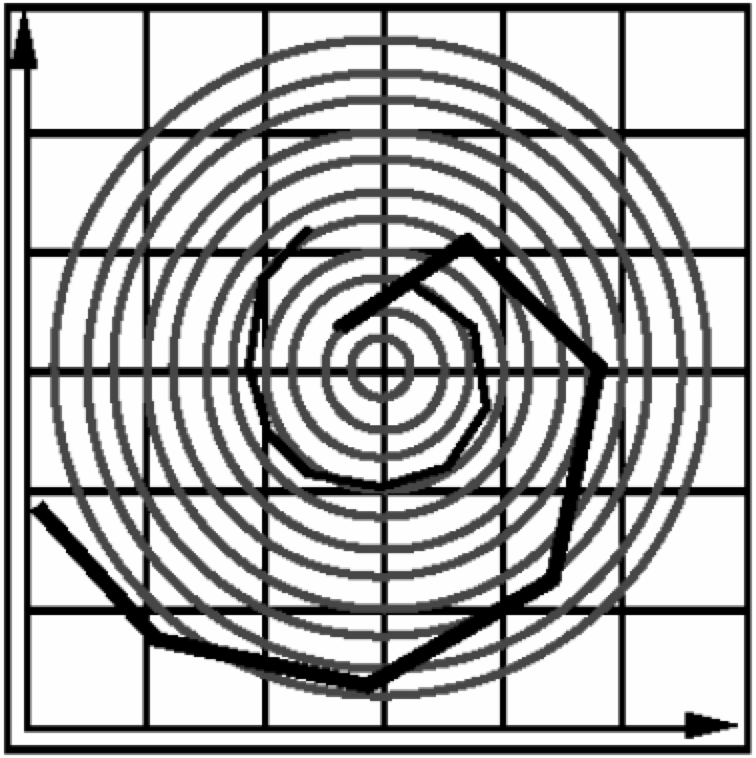
Euler's Method: Inaccurate

- 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

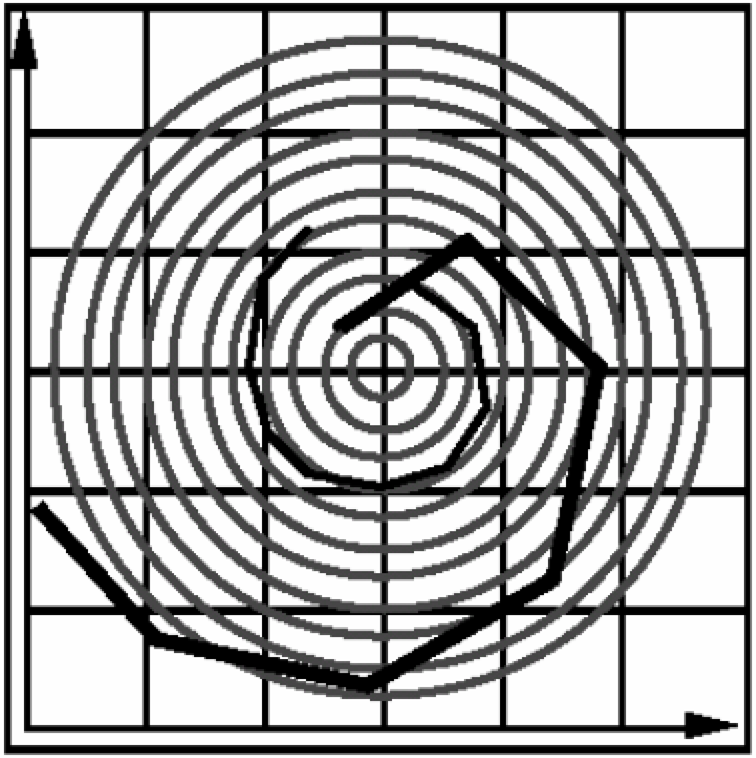
Euler's Method: Inaccurate

- 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

Questions?

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” $f(x,t) = -kx$

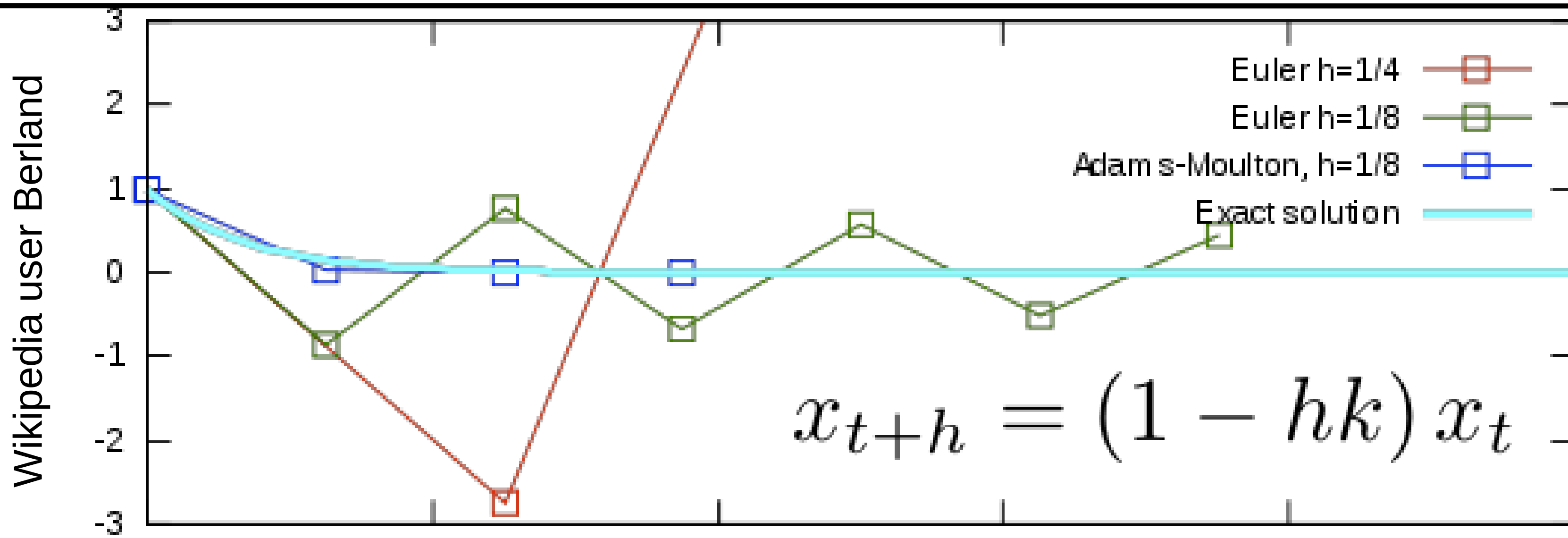
- Exact solution is a decaying exponential:

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

- Let’s apply Euler’s method:

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

Euler's Method: Not Always Stable



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

Analysis: Taylor Series

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

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

- Euler's method approximates:

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

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

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

Analysis: Taylor Series Questions?

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

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

- Euler's method approximates:

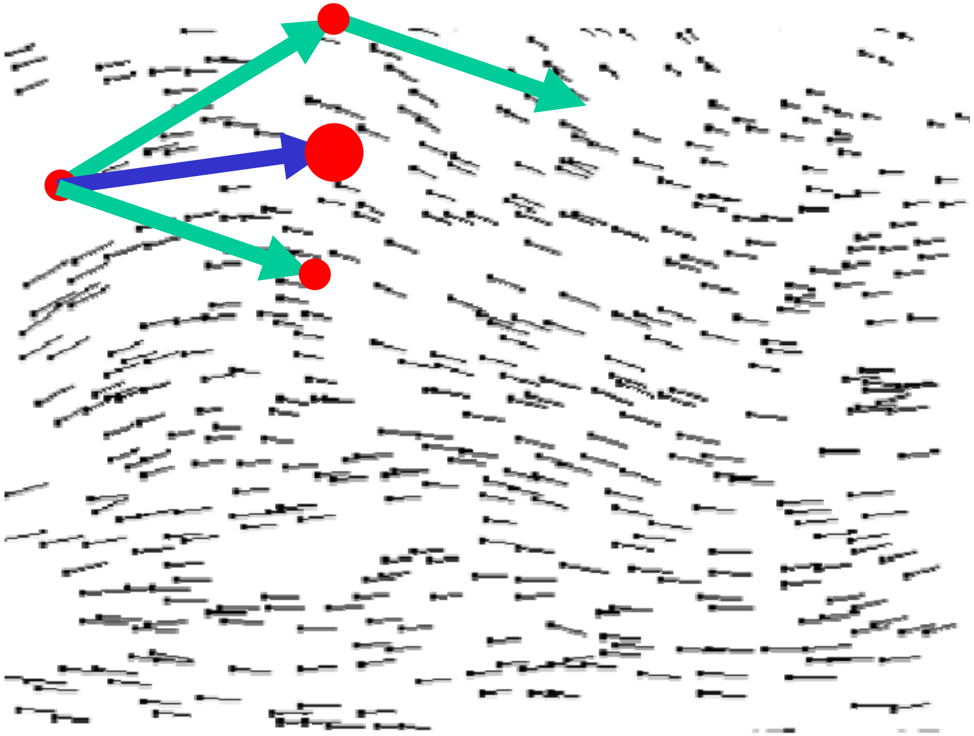
$$\mathbf{X}(t_0+h) = \mathbf{X}_0 + h\, f(\mathbf{X}_0, t_0) \qquad \ldots + O(h^2)\,\text{error}$$

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

- 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...

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

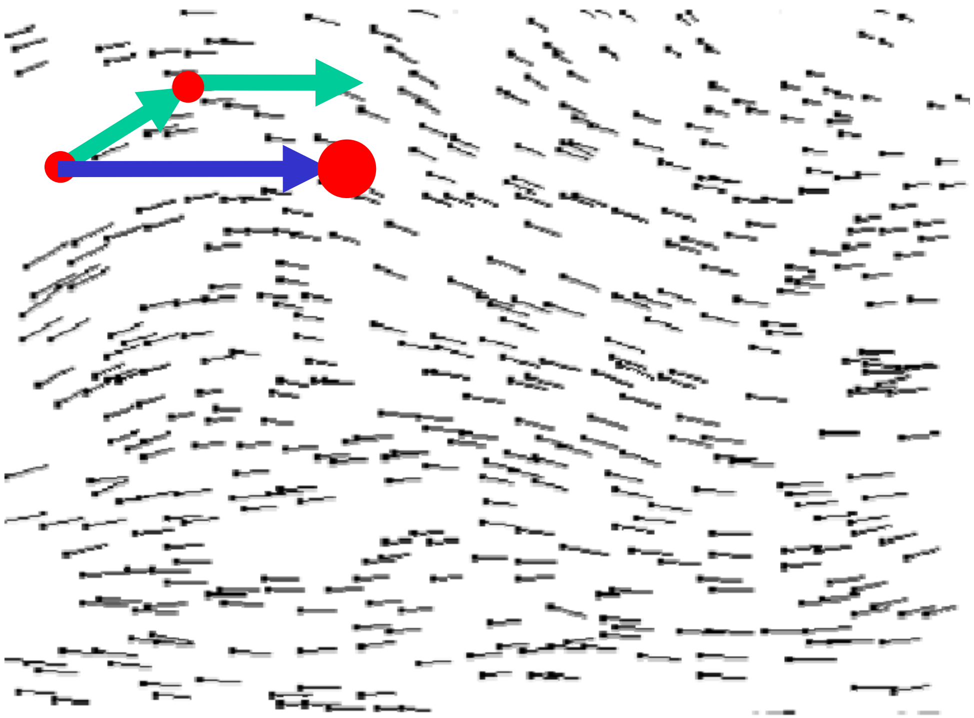
- and we get

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

- This is the *trapezoid method*
  – Analysis omitted (see 6.839)
- 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...

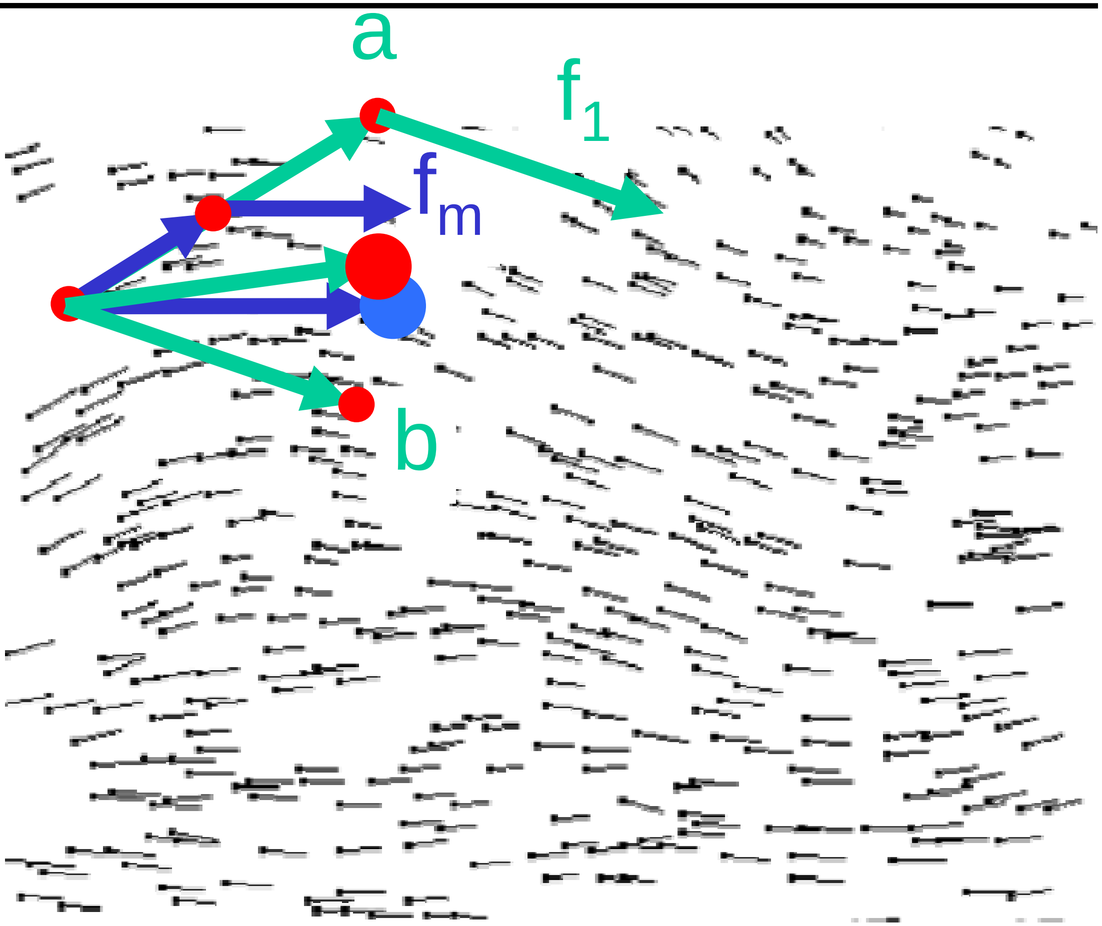
$$\boxed{\begin{aligned} f_0 &= f(\mathbf{X}_0, t_0) \\ f_m &= f(\mathbf{X}_0 + \tfrac{h}{2} f_0, t_0 + \tfrac{h}{2}) \end{aligned}}$$

- 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 here.

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



Can We Do Even Better?

- You bet!
- You will 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=HbE3L5CIdQg

Can We Do Even Better? Questions?

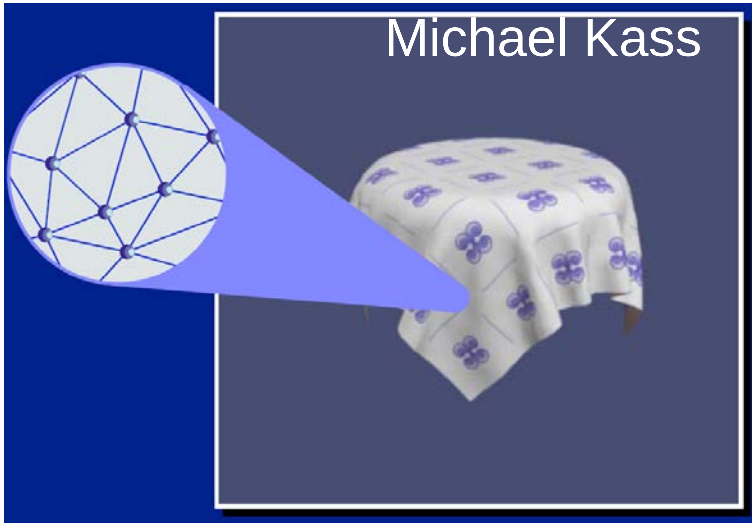
- You bet!
- You will implement Runge-Kutta for assignment 3

- Again, see Witkin, Baraff, Kass: Physically-based Modeling Course Notes, SIGGRAPH 2001

- See eg http://www.youtube.com/watch?v=HbE3L5CIdQg

Mass-Spring Modeling

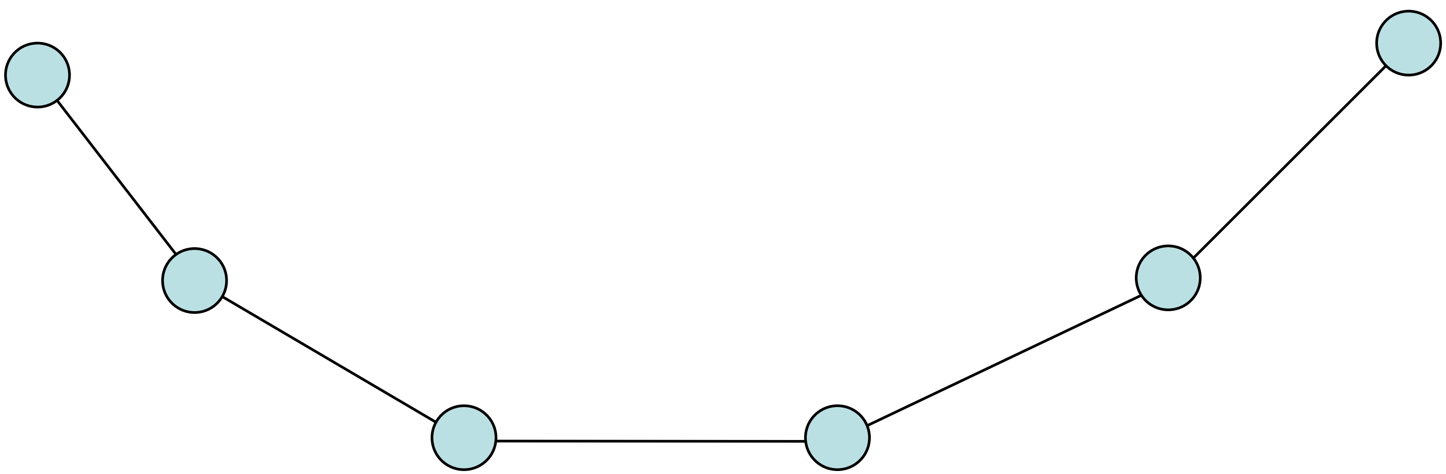
- Beyond pointlike objects: strings, cloth, hair, etc.
- Interaction between particles
  - Create a network of spring forces that link pairs of particles



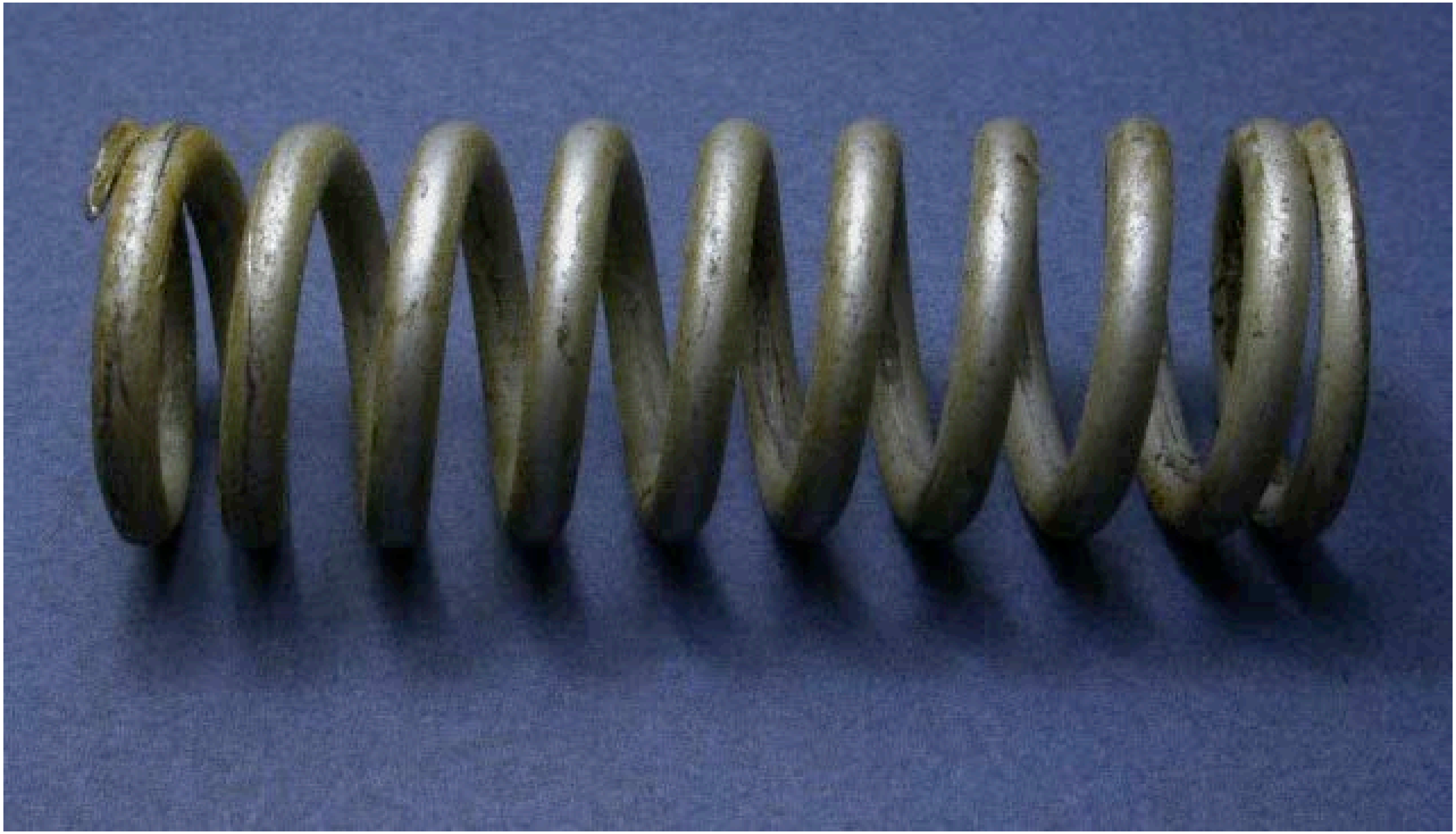
- First, slightly hacky version of cloth simulation
- Then, some motivation/intuition for *implicit integration* (NEXT LECTURE)

How Would You Simulate a String?

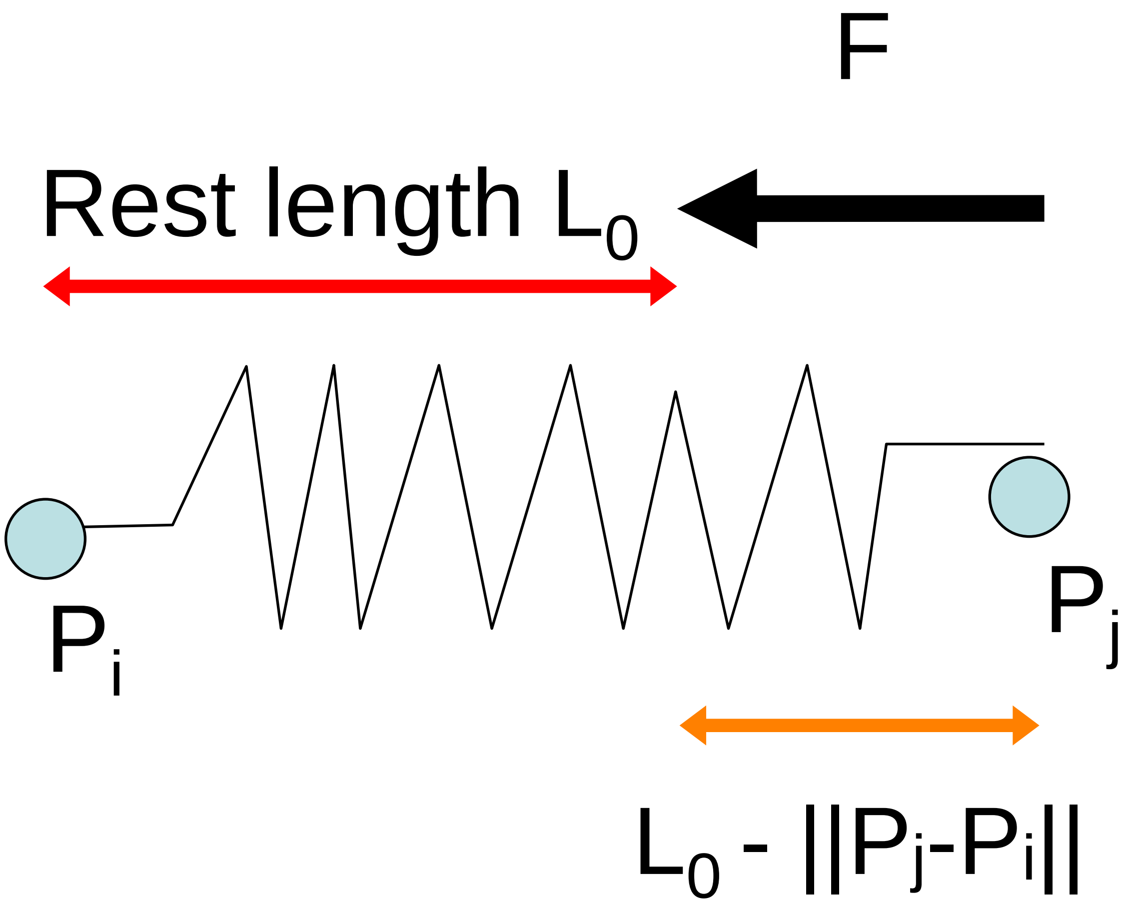
- Each particle is linked to two particles (except ends)
- Come up with forces that try to keep the distance between particles constant



Springs



Spring Force – Hooke's Law

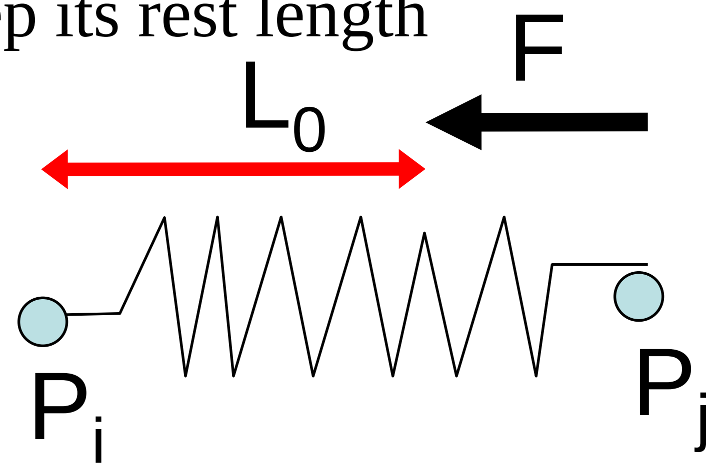


Spring Force – Hooke's Law

- 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

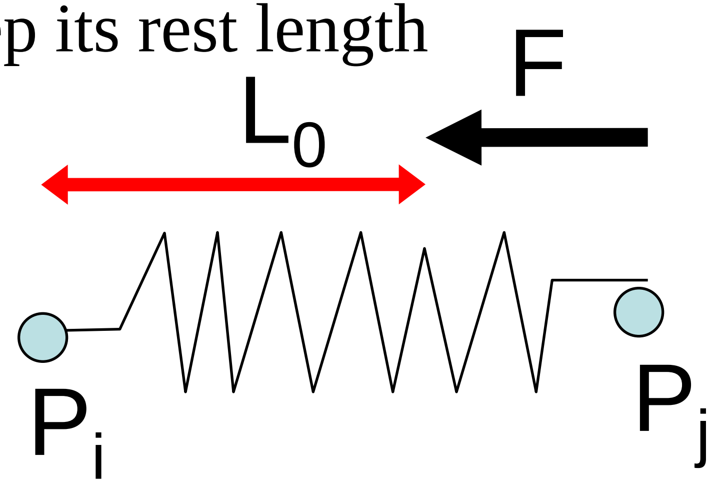


Spring Force – Hooke's Law

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

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

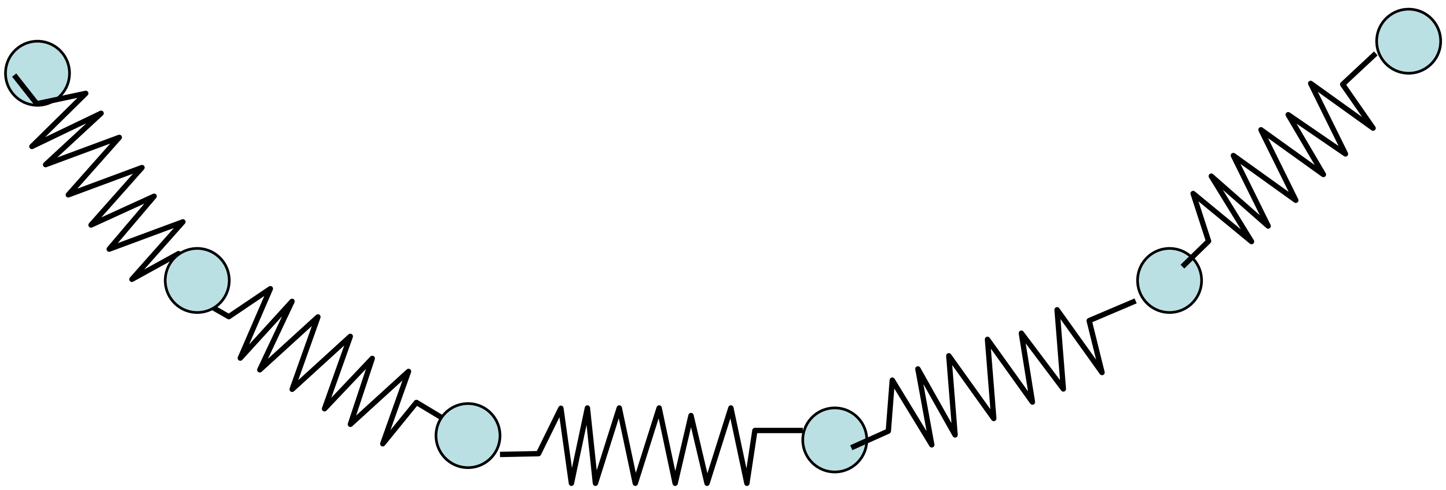
- K is the stiffness of the spring
  - When K gets bigger, the spring *really* wants to keep its rest length



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

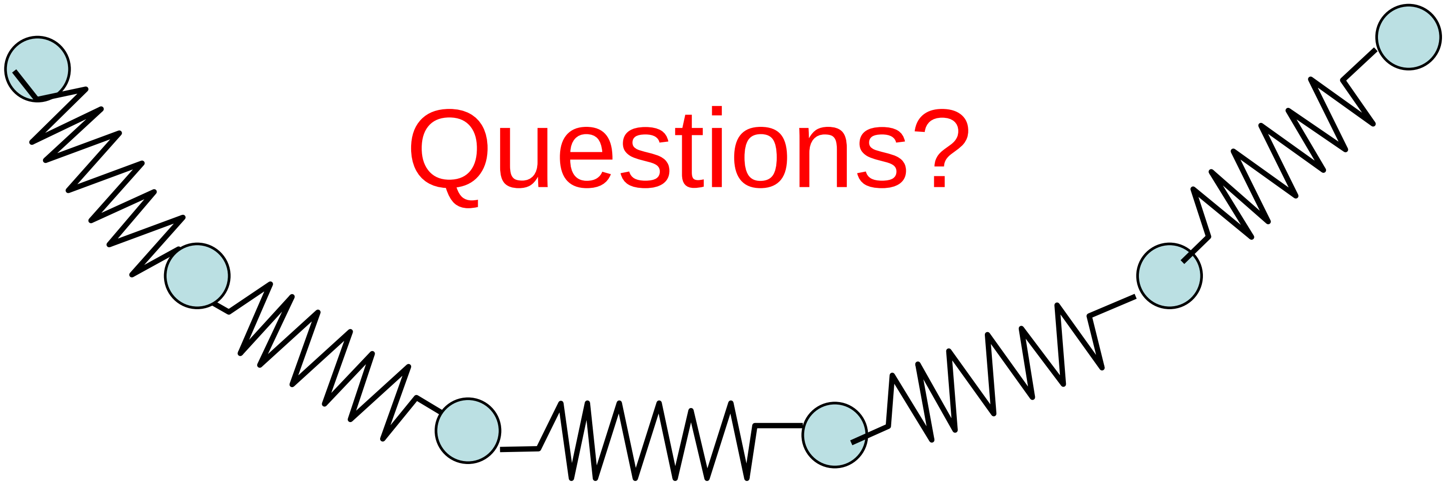
How Would You Simulate a String?

- 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?

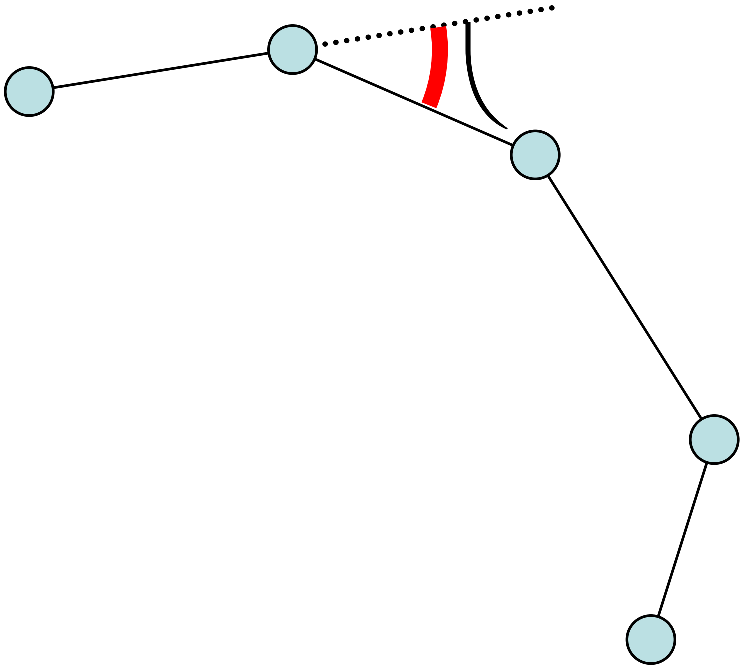
- Springs link the particles
- Springs try to keep their rest lengths and preserve the length of the string
- Not exactly preserved though, and we get oscillation
  - Rubber band approximation



Questions?

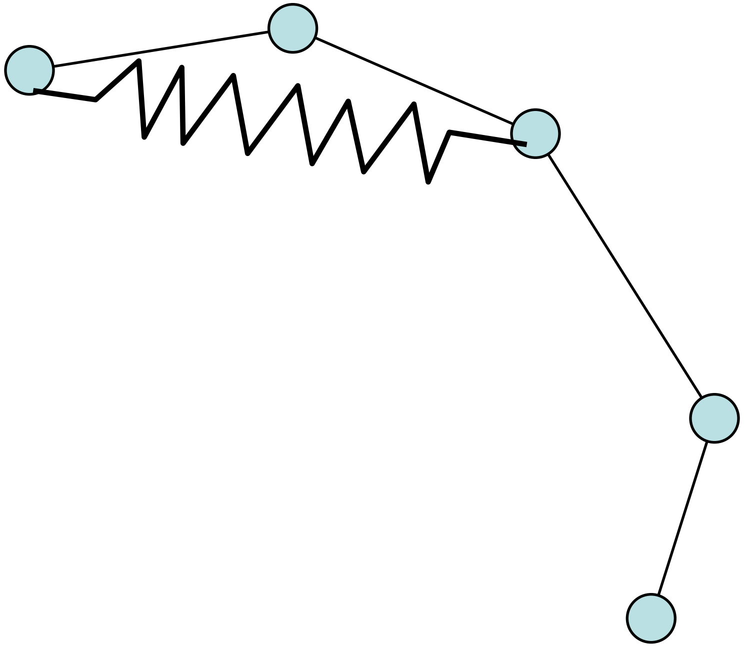
Hair

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



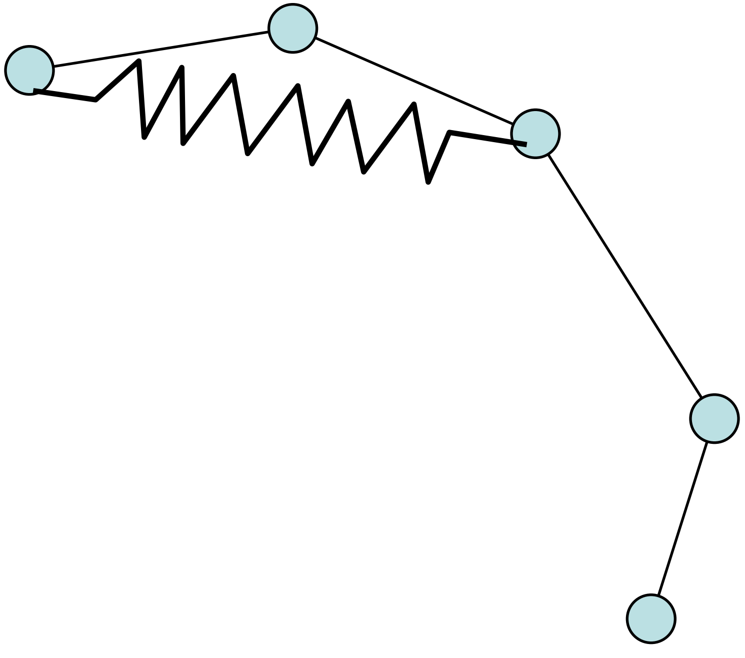
Hair - Alternative Structural Forces

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



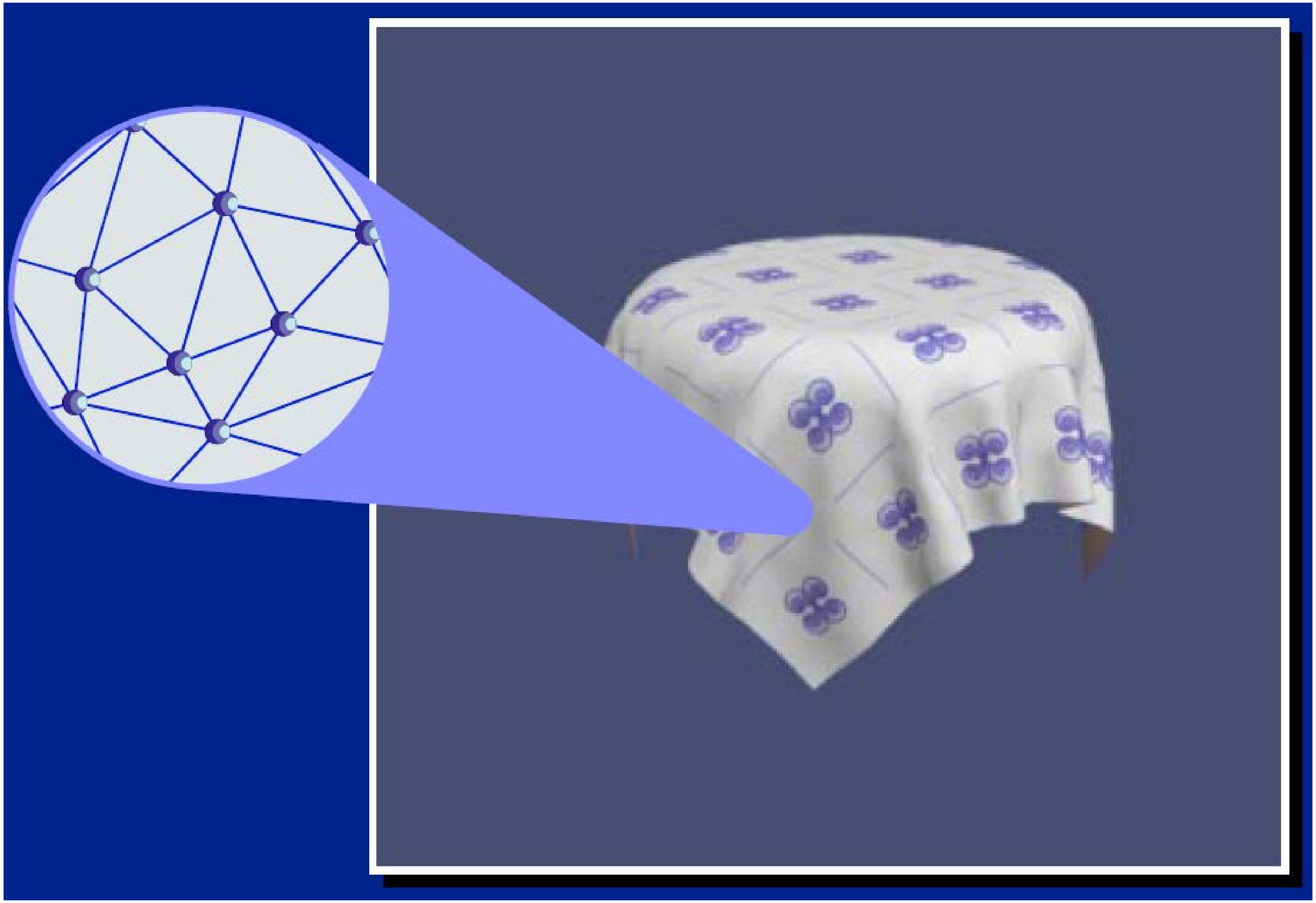
Hair - Alternative Structural Forces

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



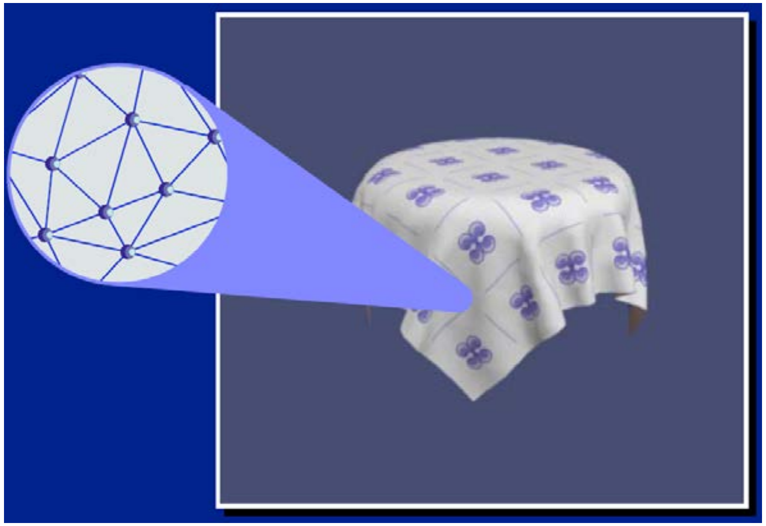
Questions?

Mass-Spring Cloth



Michael Kass

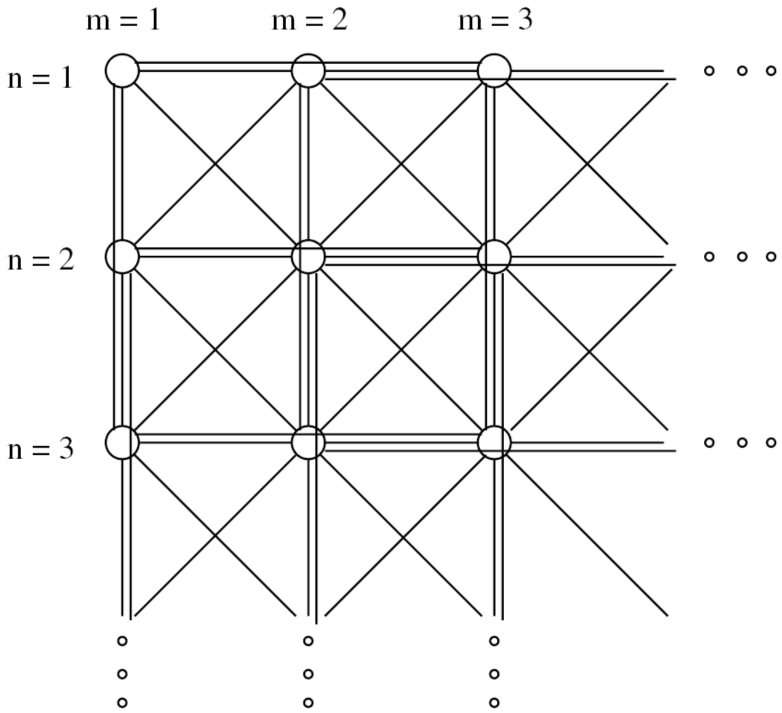
Cloth – Three Types of Forces



- **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

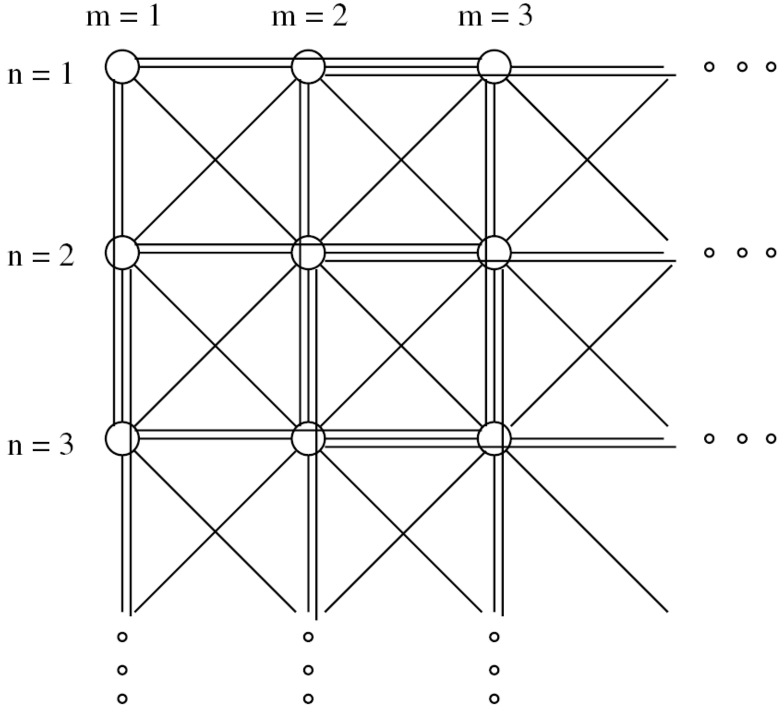
- Network of masses and springs
- **Structural** springs:
  - link (i j) and (i+1, j); and (i, j) and (i, j +1)
- **Deformation:**
  - Shear springs
    - (i j) and (i+1, j+1)
  - Flexion springs
    - (i,j) and (i+2,j); (i,j) and (i,j+2)
- See [Provot's Graphics Interface '95 paper] for details



Provot 95

External Forces

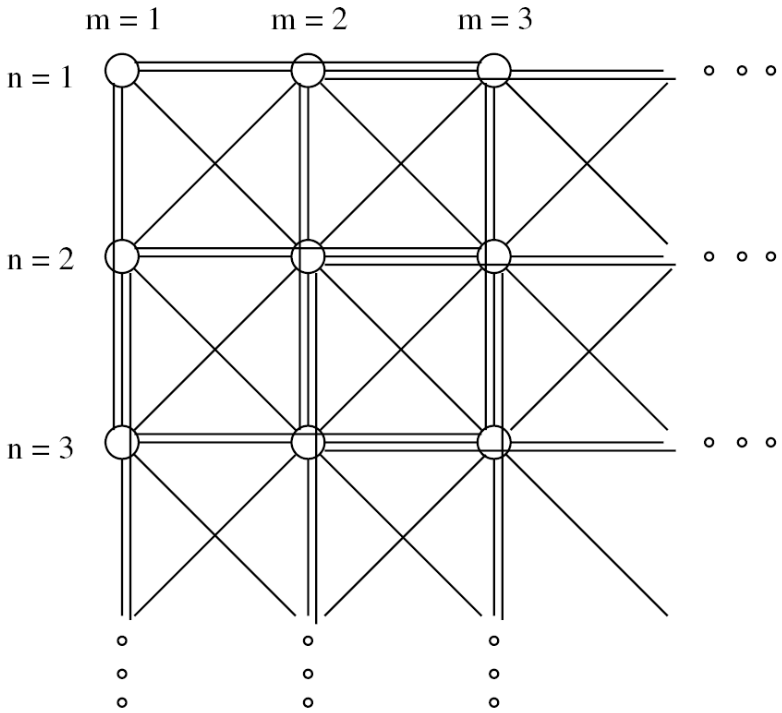
- Gravity G
- Friction
- Wind, etc.



Provot 95

Cloth Simulation

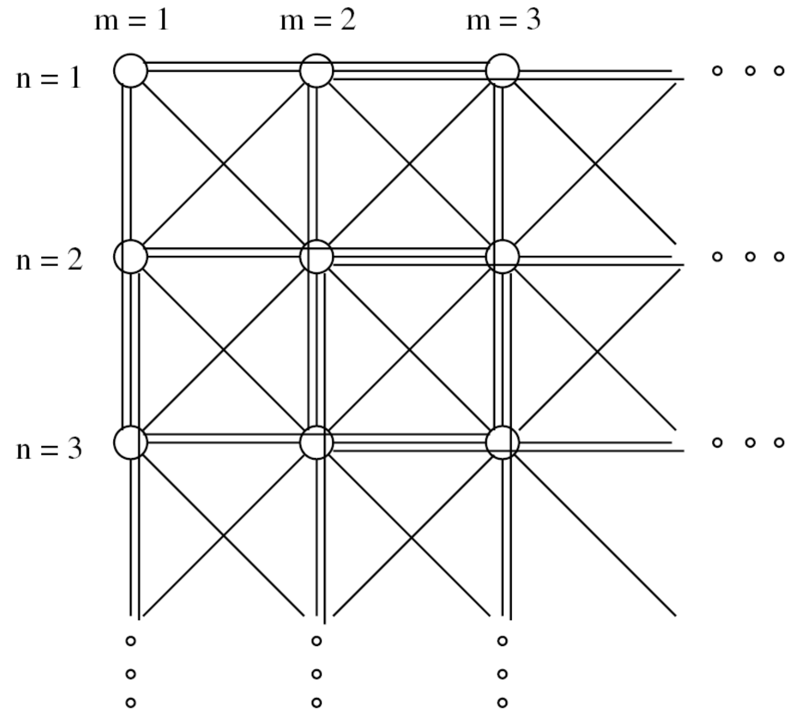
- Then, the all trick is to set the stiffness of all springs to get realistic motion!
- Remember that forces depend on other particles (coupled system)
- But it is *sparse* (only near neighbors)
  - This is in contrast to e.g. the N-body problem.



Provot 95

Forces: Structural vs. Deformation

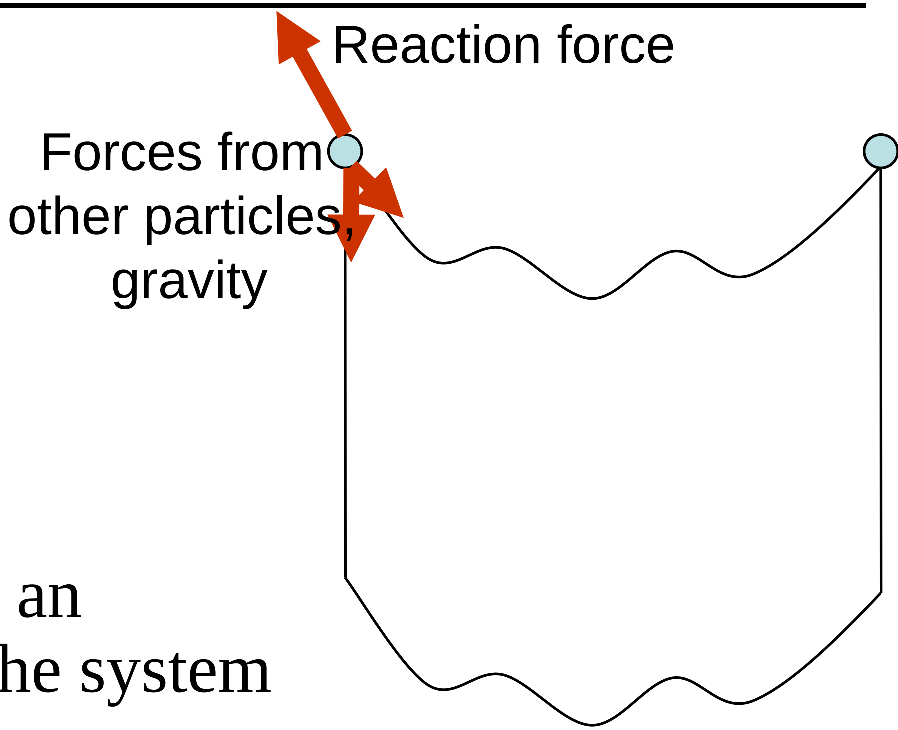
- Structural forces are here just to enforce a constraint
- Ideally, the constraint would be enforced strictly
  - at least a lot more than we can afford
- We'll see that this is the root of a lot of problems
- In contrast, deformation forces actually correspond to physical forces



Provot 95

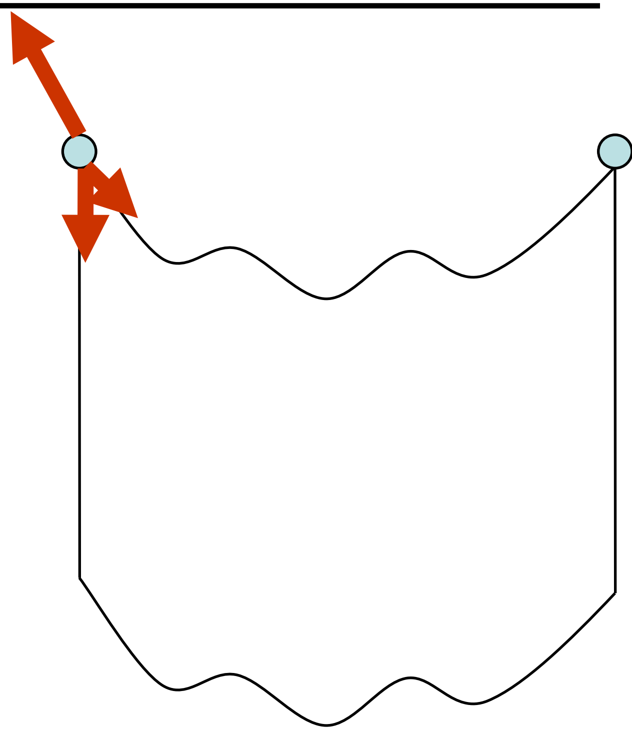
Contact Forces

- Hanging curtain:
  - 2 contact points stay fixed
- What does it mean?
  - Sum of the forces is zero
- How so?
  - Because those point 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.



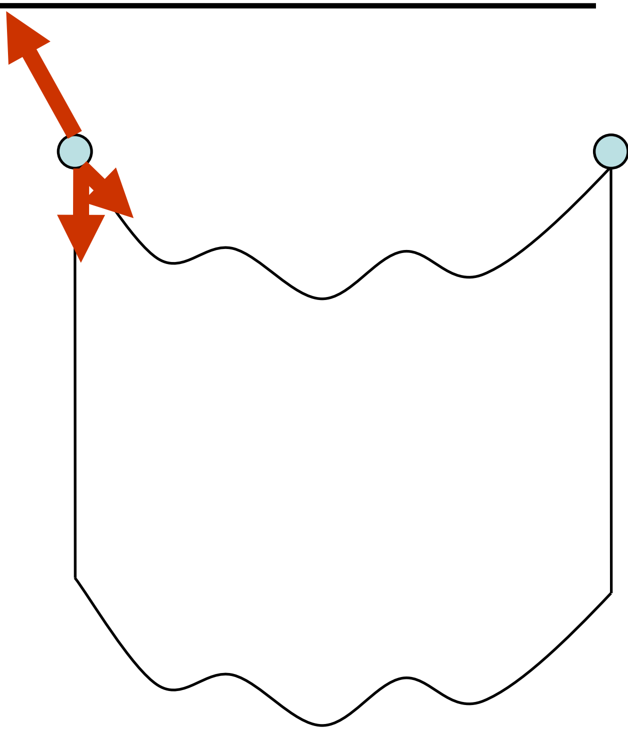
Contact Forces

- 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

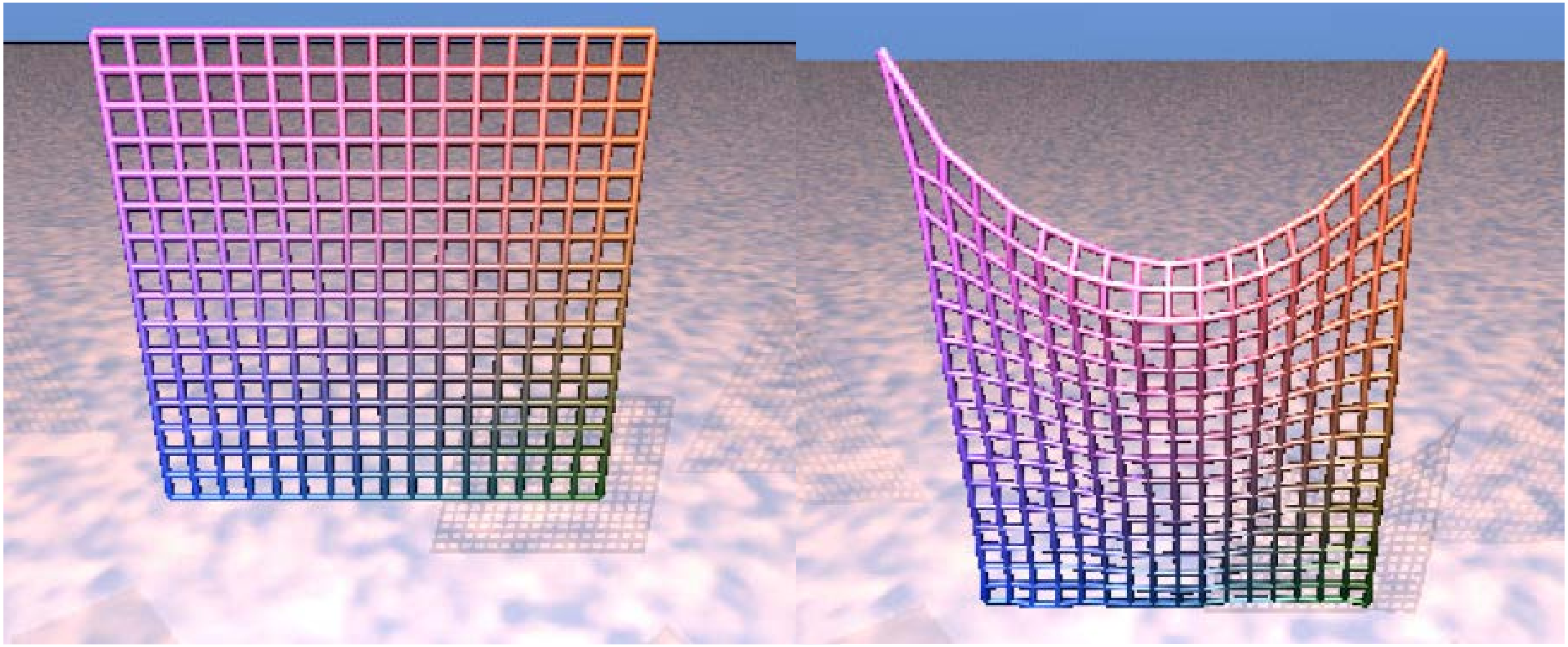
- How can we compute the external contact force?
  - Inverse dynamics!
  - Sum all other forces applied to point
  - Take negative
- Do we really need to compute this force?
  - Not really, just ignore the other forces applied to this point!



Questions?

Example

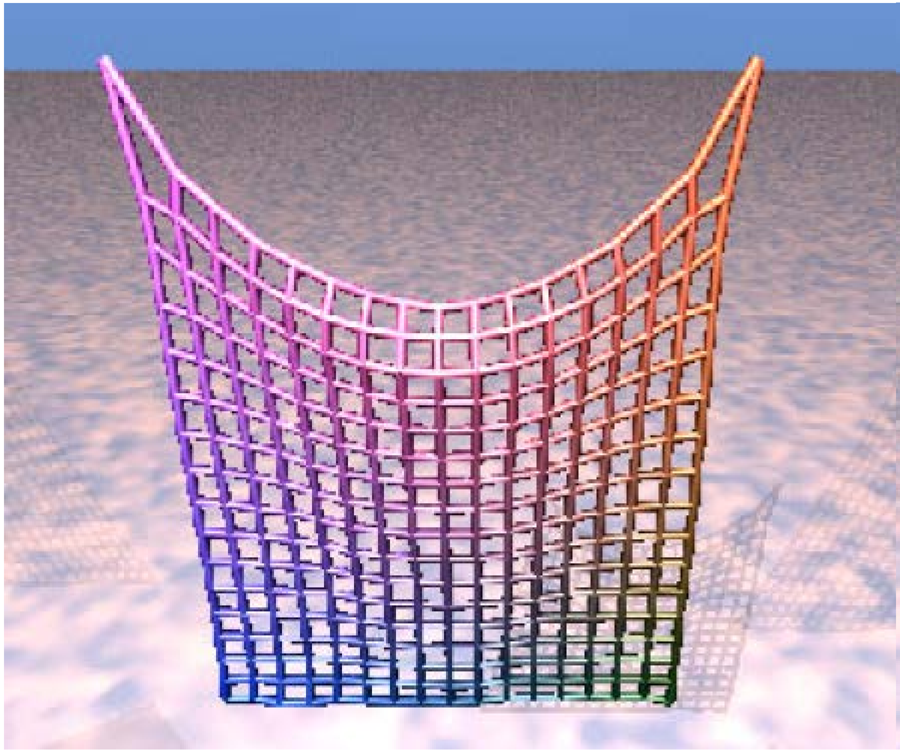
- Excessive rubbery deformation:
  the strings are not stiff enough



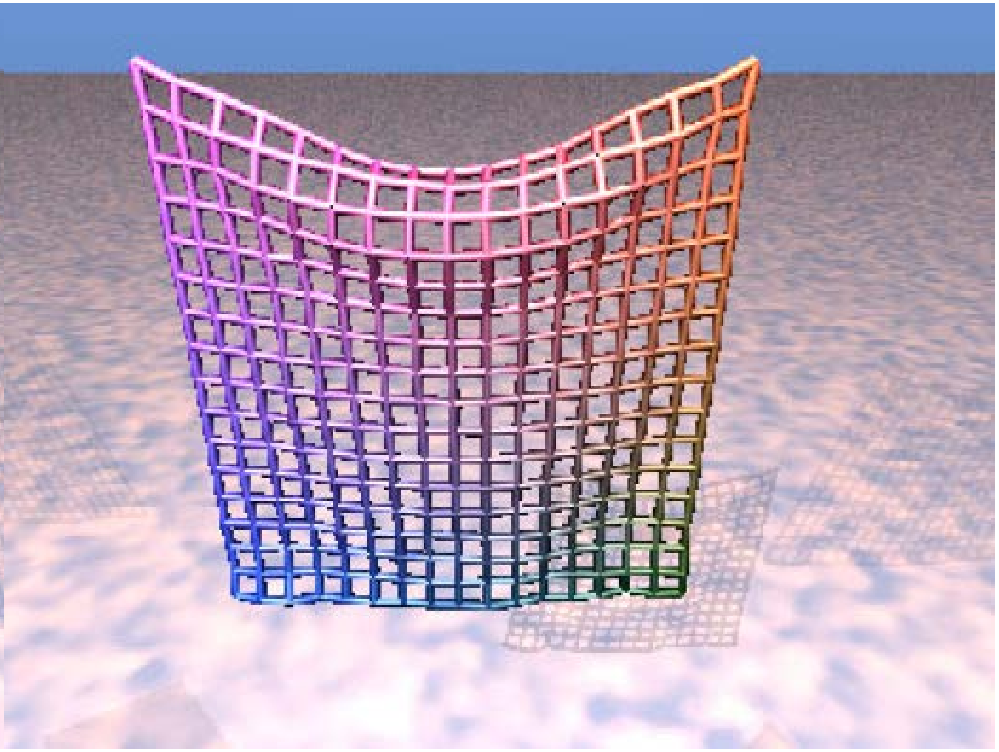
Initial position After 200 iterations

One Solution

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



Simple mass-spring system

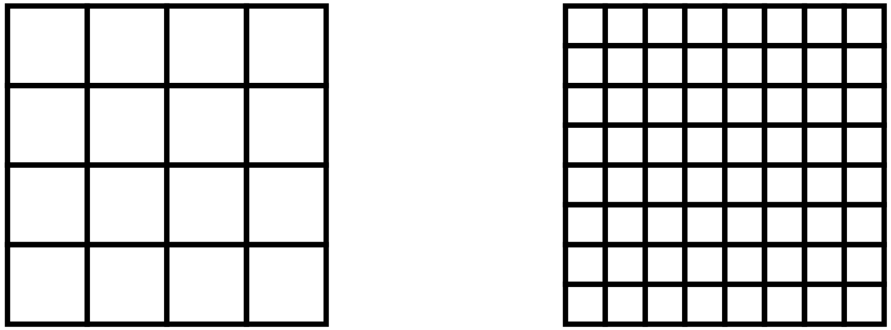


Improved solution
(see Provot Graphics Interface 1995)

http://citeseer.ist.psu.edu/provot96deformation.html

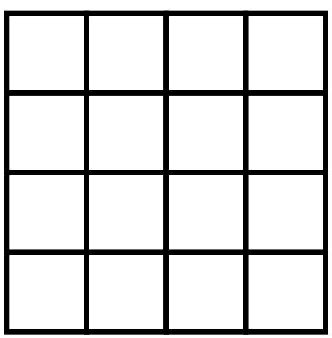
The Discretization Problem

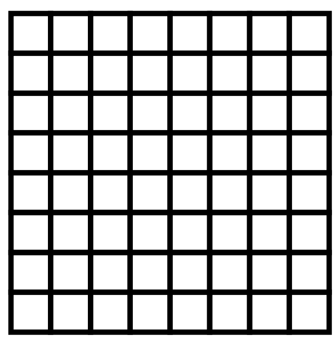
- What happens if we discretize our cloth more finely?
- Do we get the same behavior?
- Usually not! It takes a lot of effort to design a scheme that is mostly oblivious to the discretization.



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?

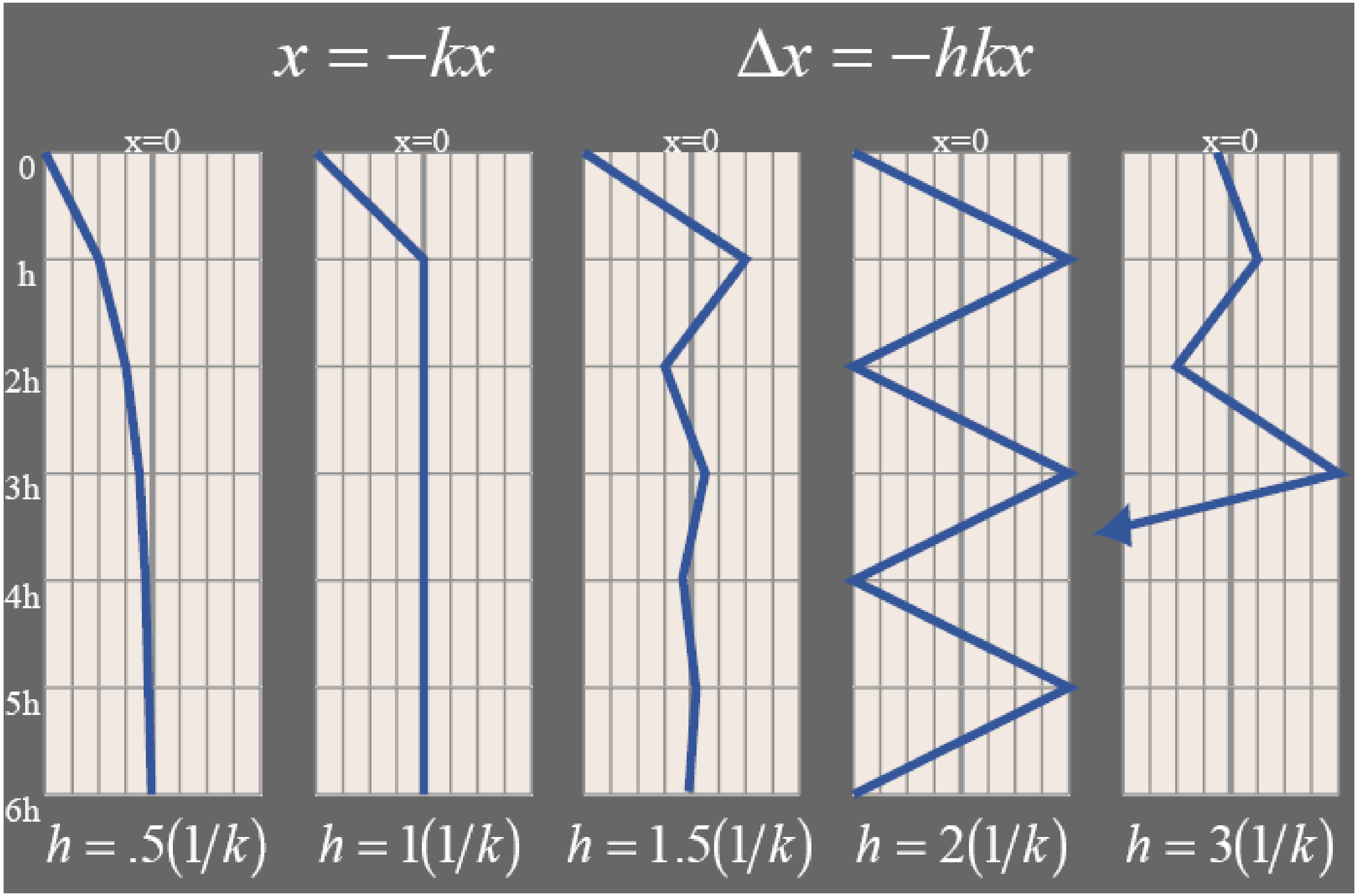
The Stiffness Issue

- We use springs while we really mean constraint
  - Spring should be super stiff, which requires tiny Δt
  - Remember $x' = -kx$ system and Euler speed limit!
    - The story extends to N particles and springs (unfortunately)

- Many numerical solutions
  - Reduce Δt (well, not a great solution)
  - Actually use constraints (see 6.839)
  - Implicit integration scheme (more next Thursday)

Euler Has a Speed Limit!

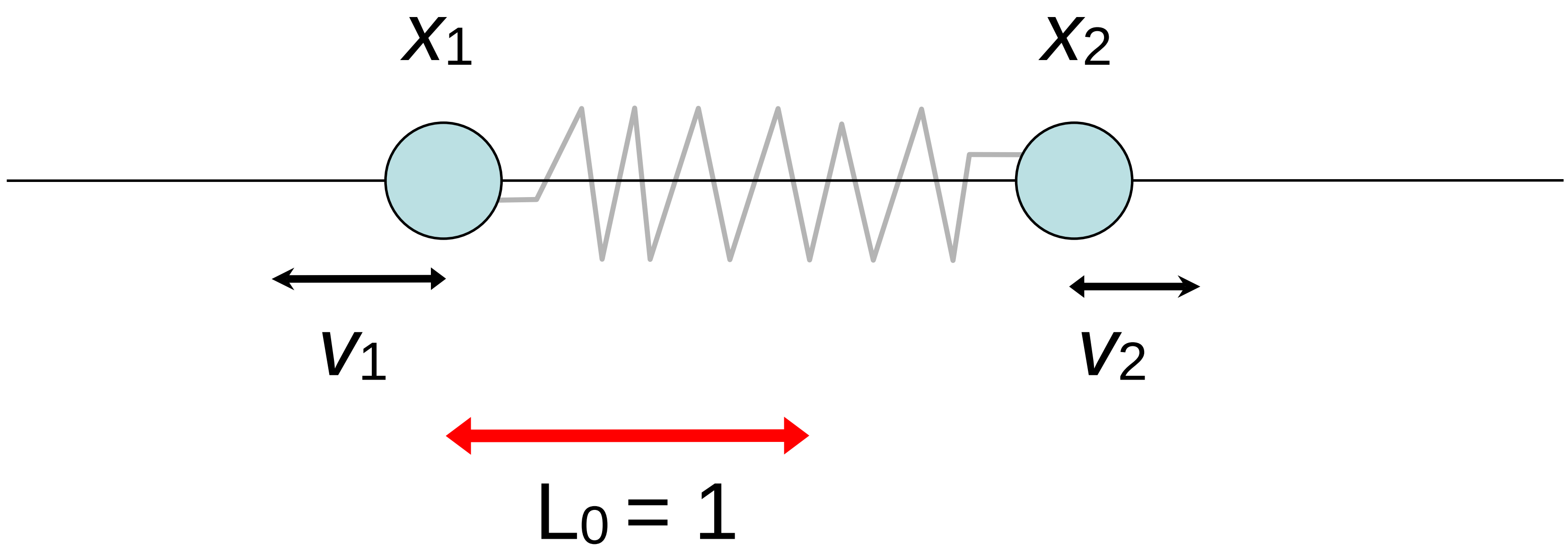
- h > 1/k: oscillate. h > 2/k: explode!



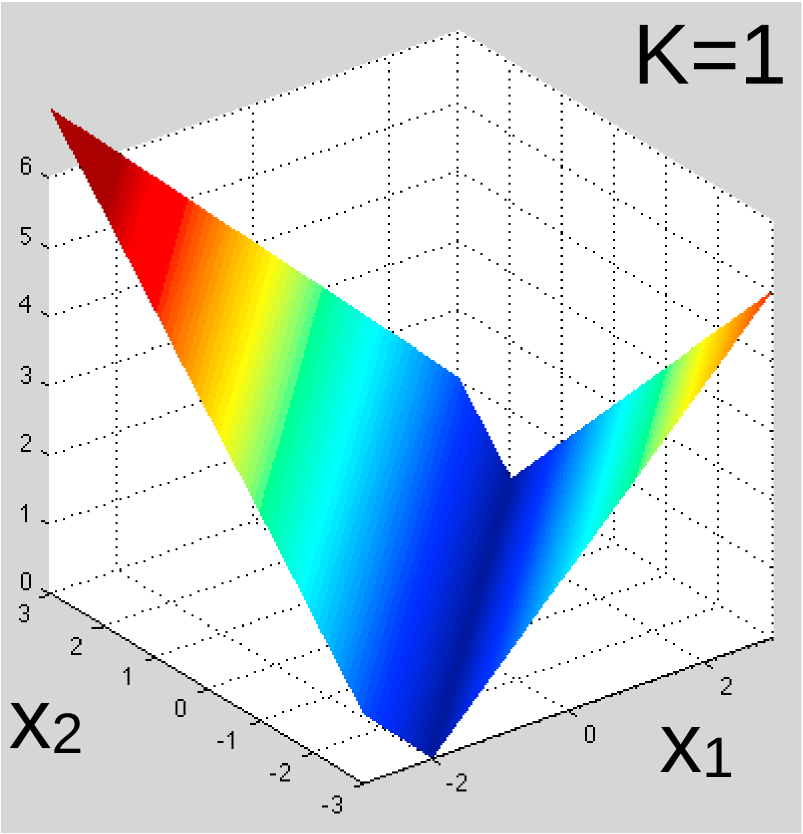
From the SIGGRAPH PBM notes

Why Stiff Springs Are Difficult

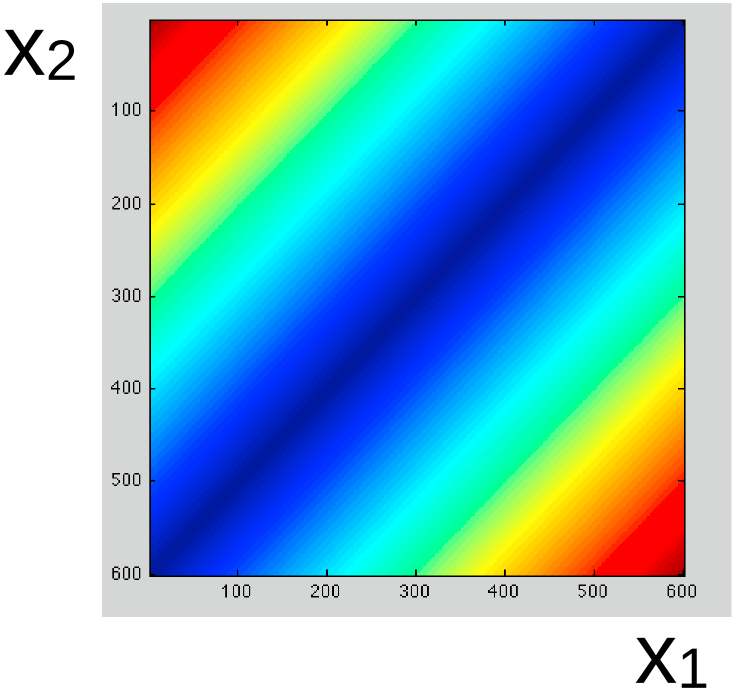
- 1D example, with two particles constrained to move along the x axis only, rest length $L_0 = 1$
- Phase space is 4D: (x_1, v_1, x_2, v_2)
  - But spring force only depends on x_1, x_2 and L_0.



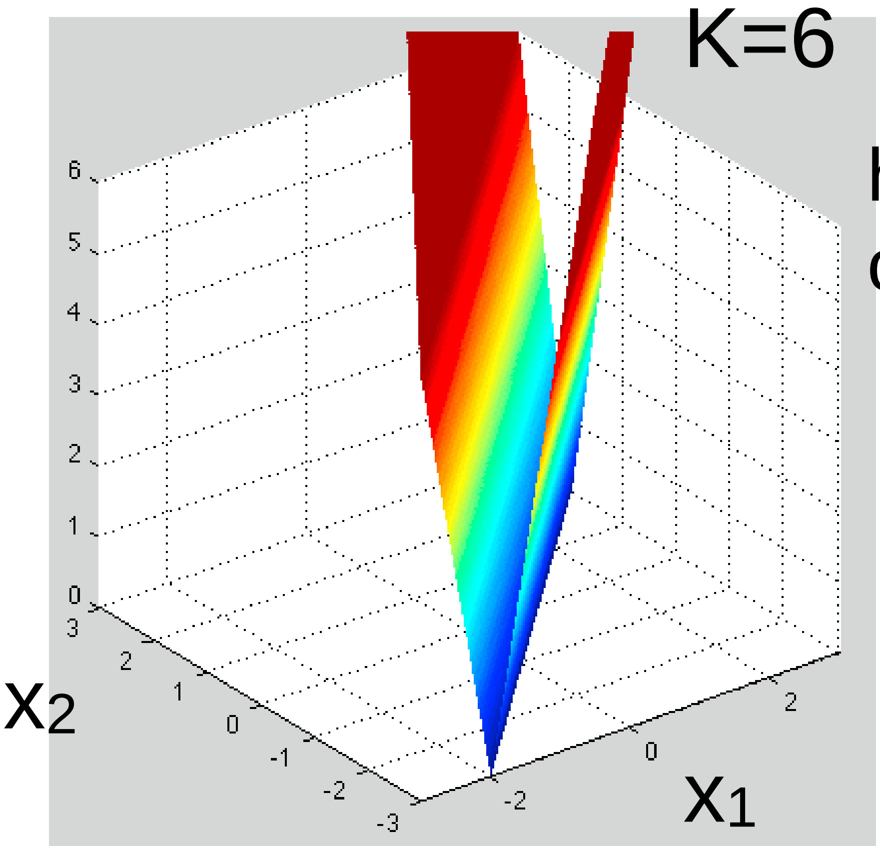
Why Stiff Springs Are Difficult



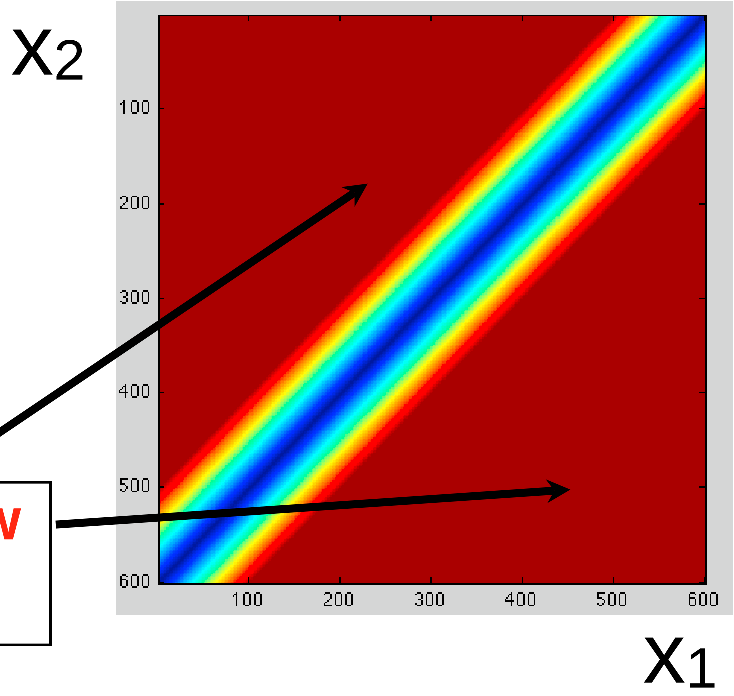
height=magnitude of spring force



Why Stiff Springs Are Difficult

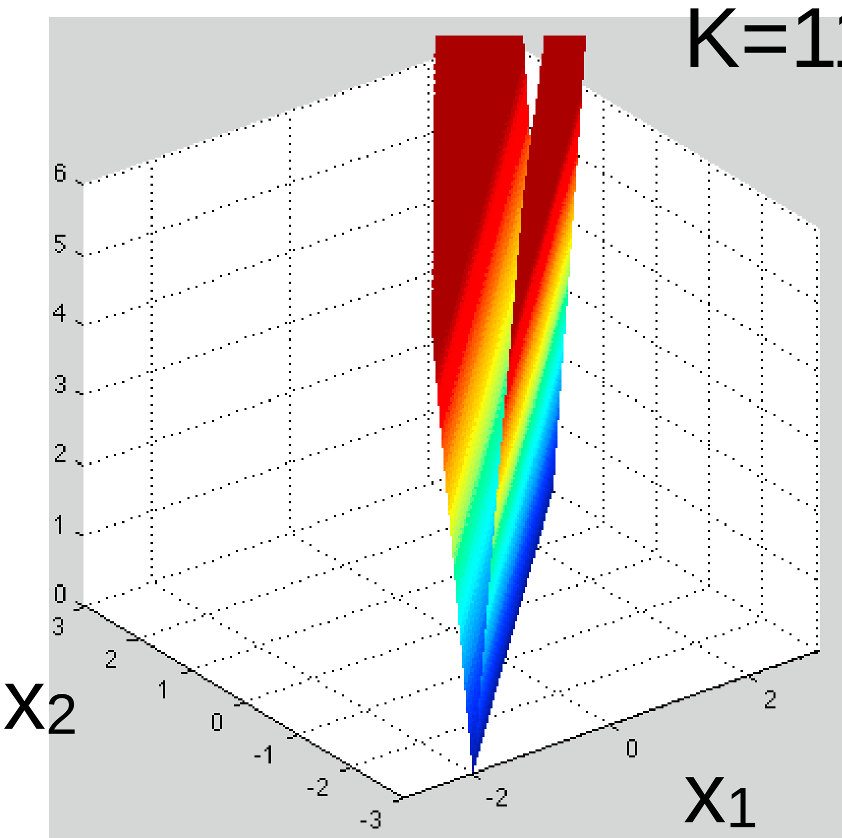


height=magnitude
of spring force

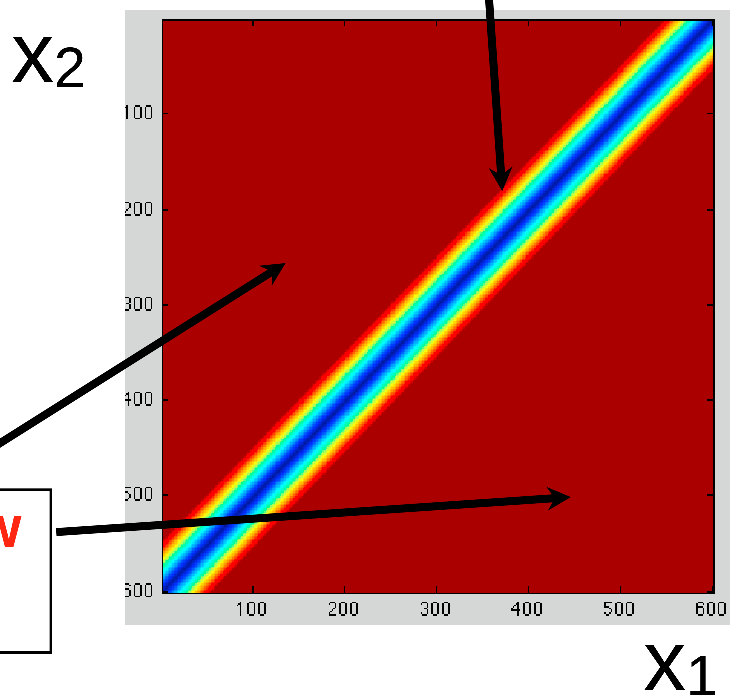


Forces grow really big!

Why Stiff Springs Are Difficult

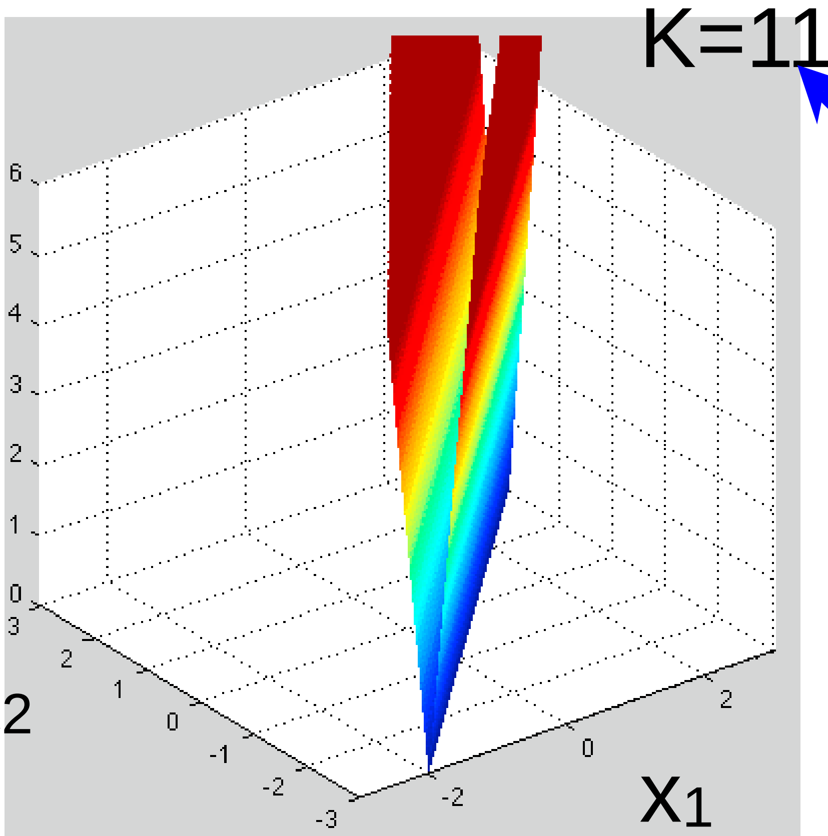


The "admissible region" shrinks towards the line $x_1 - x_2 = 1$ as K grows



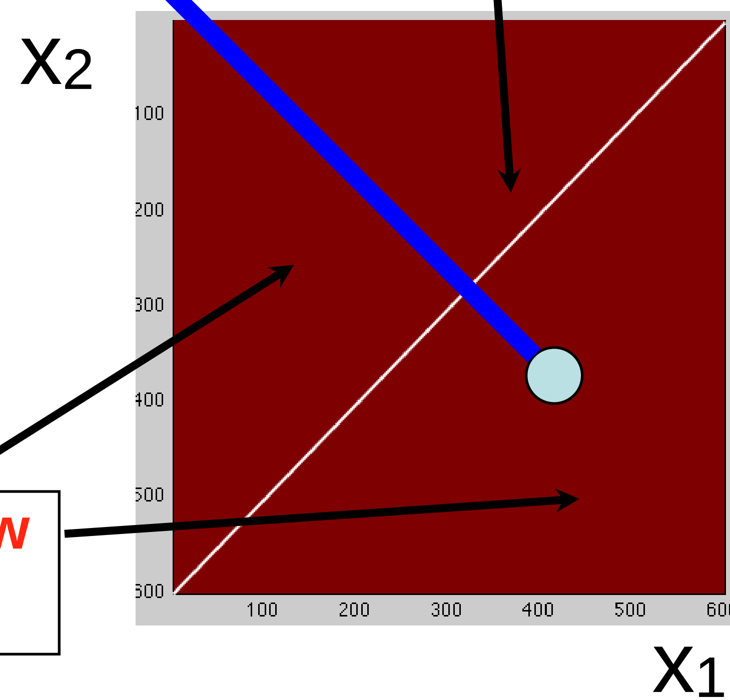
Forces grow really big!

Why Stiff Springs Are Difficult



off to the moon!

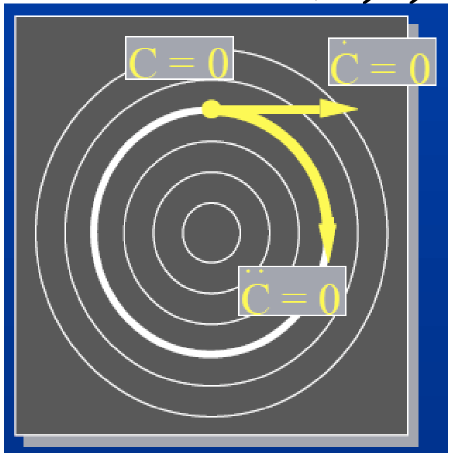
The “admissible region” shrinks towards the line x_1-x_2=1 as K grows



Forces grow really big!

Constrained Dynamics

- In our mass-spring cloth, we have “encouraged” length preservation using springs that want to have a given length (unfortunately, they can refuse offer ;-))
- Constrained dynamic simulation: force it to be constant!



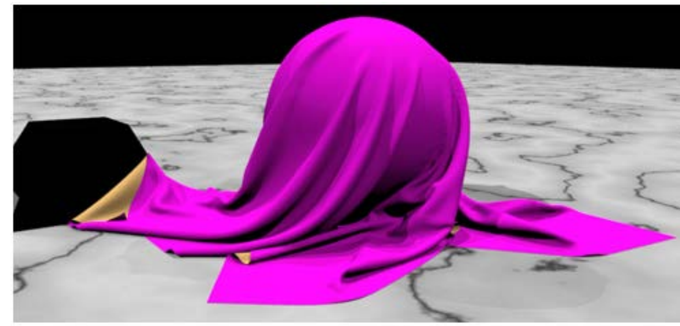
- How it works – more in 6.839
  - Start with constraint equation
    - E.g., $(x_2-x_1)-1 = 0$ in the previous 1D example
  - Derive extra forces that will exactly enforce constraint
    - This means *projecting* the external forces (like gravity) onto the “subspace” of phase space where constraints are satisfied
    - Fancy name for this: “Lagrange multipliers”
  - Again, see the SIGGRAPH 2001 Course Notes

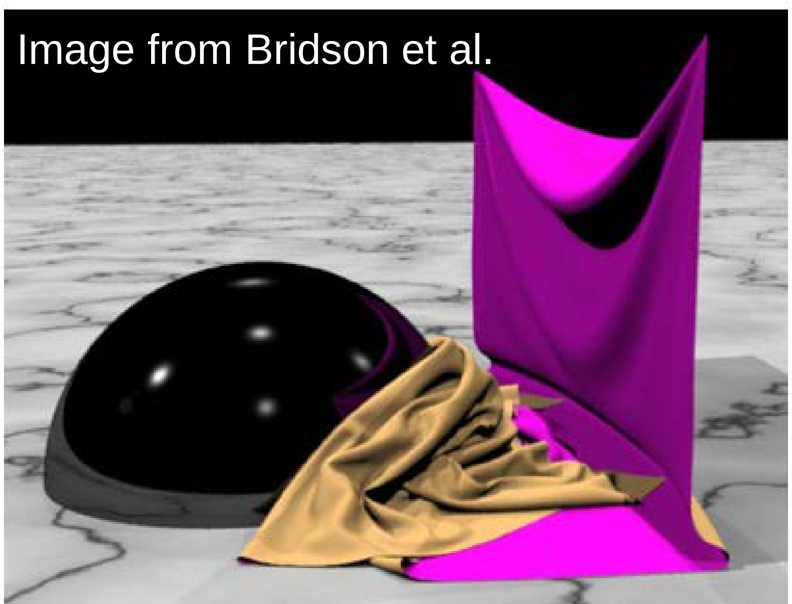
Questions?

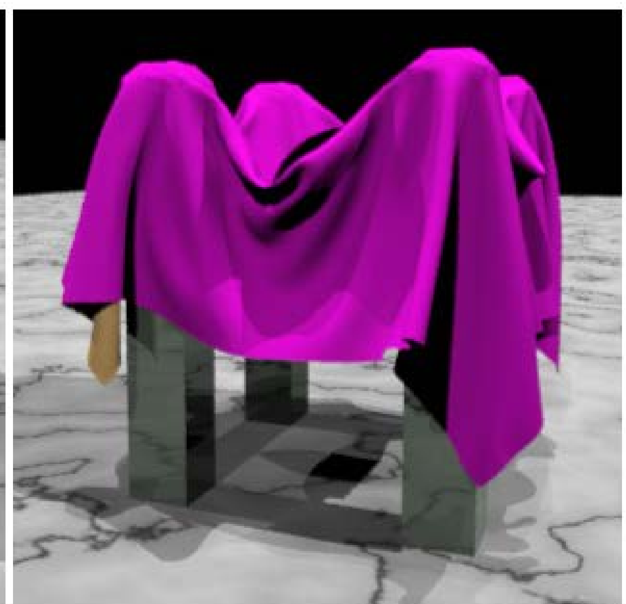
- Further reading
  - Stiff systems
  - Explicit vs. implicit solvers
  - Again, consult the 2001 course notes!

The Collision Problem

- A cloth has many points of contact
- Requires
  - Efficient collision detection
  - Efficient numerical treatment (stability)



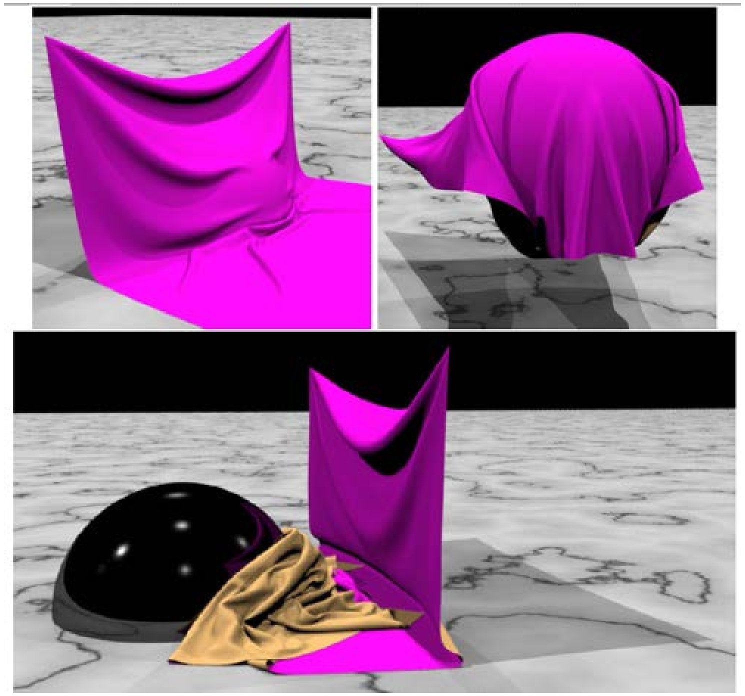


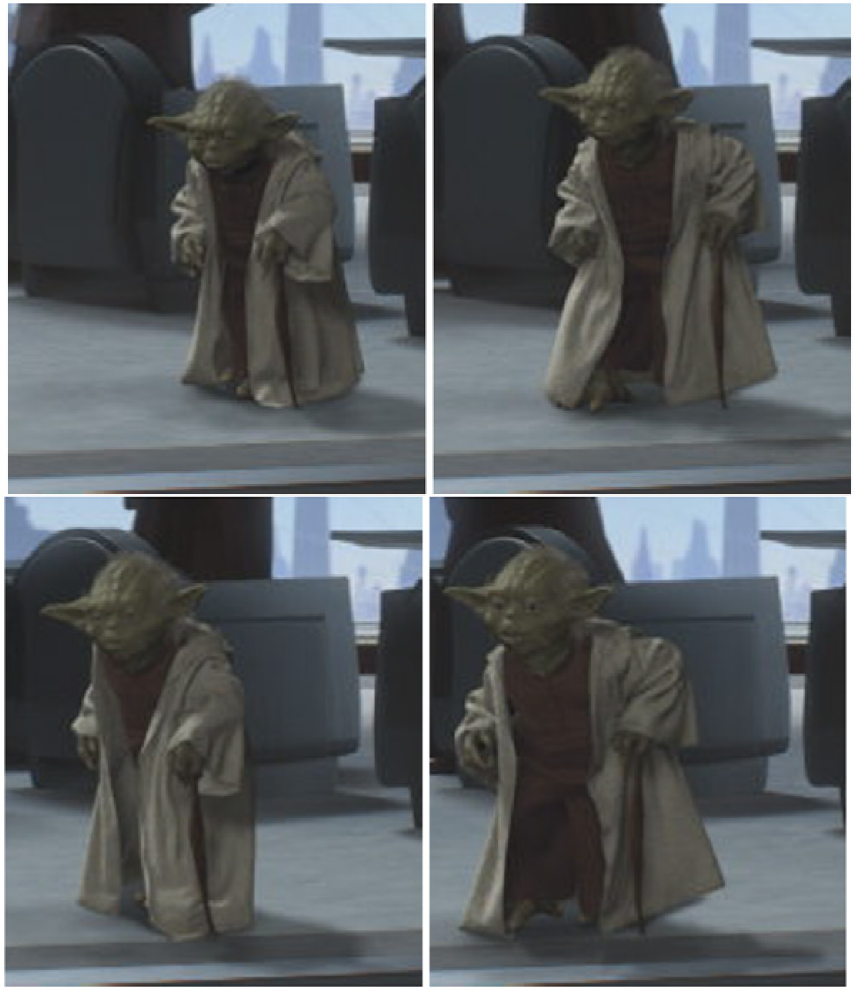


Collisions

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

- 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).

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 + 0.5*h*midVelocities
        newVelocities = velocities  + 0.5*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