

# GAMES103: Intro to Physics-Based Animation

## Physics-Based Cloth Simulation

Huamin Wang

Nov 2021

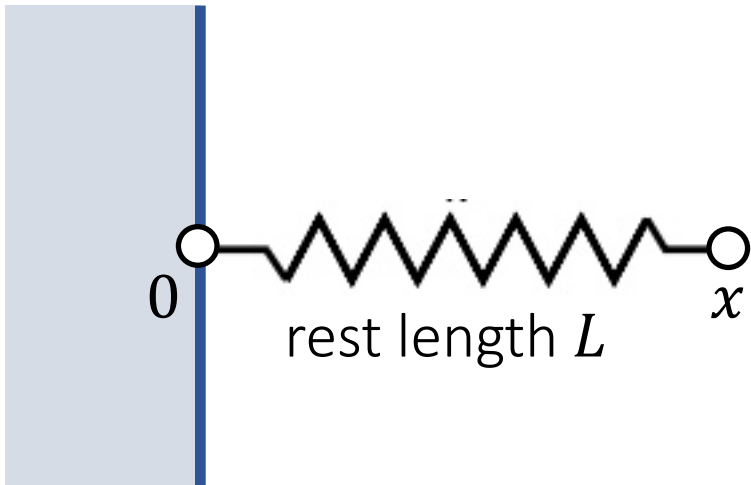
# Topics for the Day

- A Mass-Spring System
  - Explicit Integration
  - Implicit Integration
- Bending and Locking Issues
- Co-rotational FEM

# A Mass Spring System

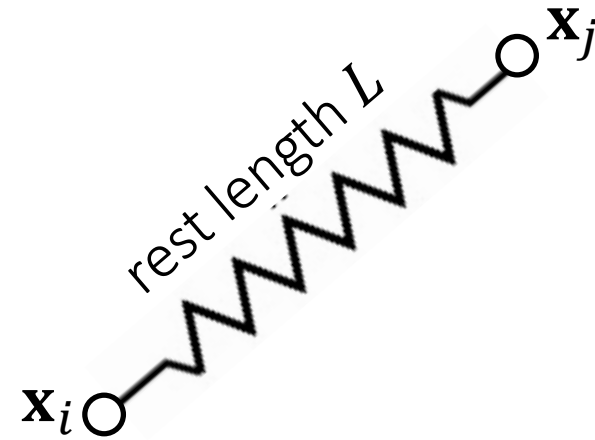
# An Ideal Spring

An ideal spring satisfies Hooke's law: the spring force tries to restore the rest length.



$$E(x) = \frac{1}{2}k(x - L)^2$$

$$f(x) = -\frac{dE}{dx} = -\underbrace{k}_{\text{spring stiffness}}(x - L)$$



$$\mathbf{f}_i = -\mathbf{f}_j$$

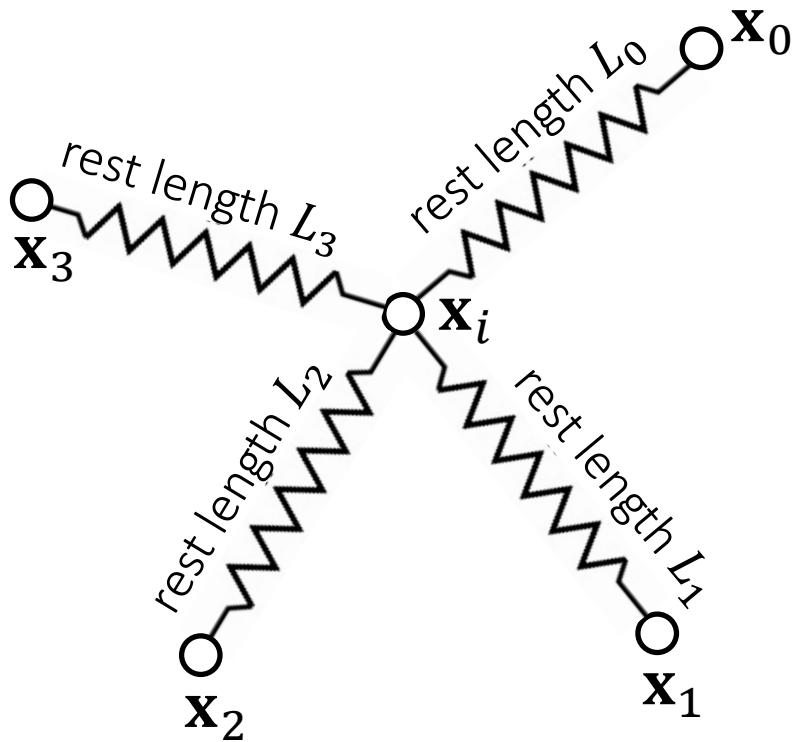
$$E(\mathbf{x}) = \frac{1}{2}k(\|\mathbf{x}_i - \mathbf{x}_j\| - L)^2$$

$$\mathbf{f}_i(\mathbf{x}) = -\nabla_i E = -k(\|\mathbf{x}_i - \mathbf{x}_j\| - L) \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}$$

$$\mathbf{f}_j(\mathbf{x}) = -\nabla_j E = -k(\|\mathbf{x}_j - \mathbf{x}_i\| - L) \frac{\mathbf{x}_j - \mathbf{x}_i}{\|\mathbf{x}_j - \mathbf{x}_i\|}$$

# Multiple Springs

When there are many springs, the energies and the forces can be simply summed up.



$$E = \sum_{j=0}^3 E_j = \sum_{j=0}^3 \left( \frac{1}{2} k (\|\mathbf{x}_i - \mathbf{x}_j\| - L)^2 \right)$$

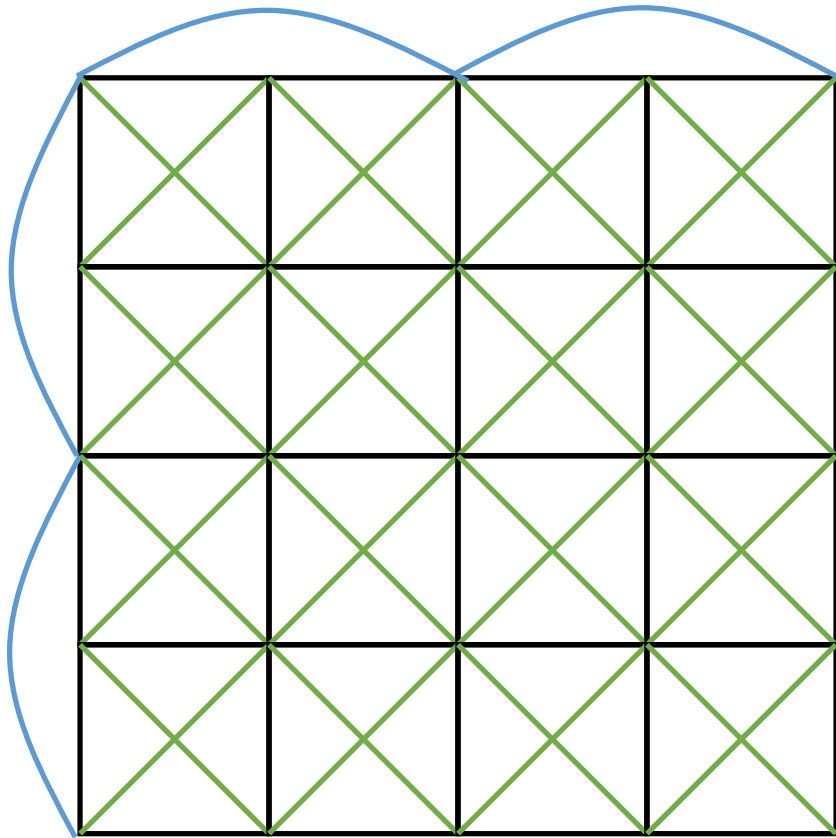
$$\mathbf{f}_i = -\nabla_i E = \sum_{j=0}^3 \left( -k (\|\mathbf{x}_i - \mathbf{x}_j\| - L) \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|} \right)$$

# Structured Spring Networks

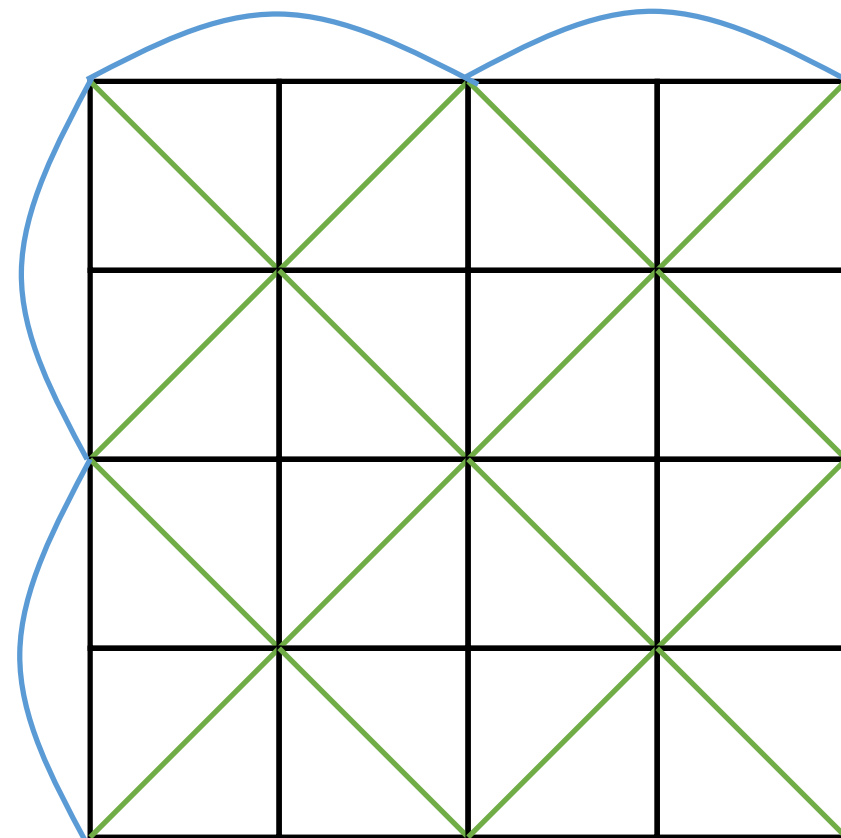
— Horizontal and vertical

— Diagonal

— Bending



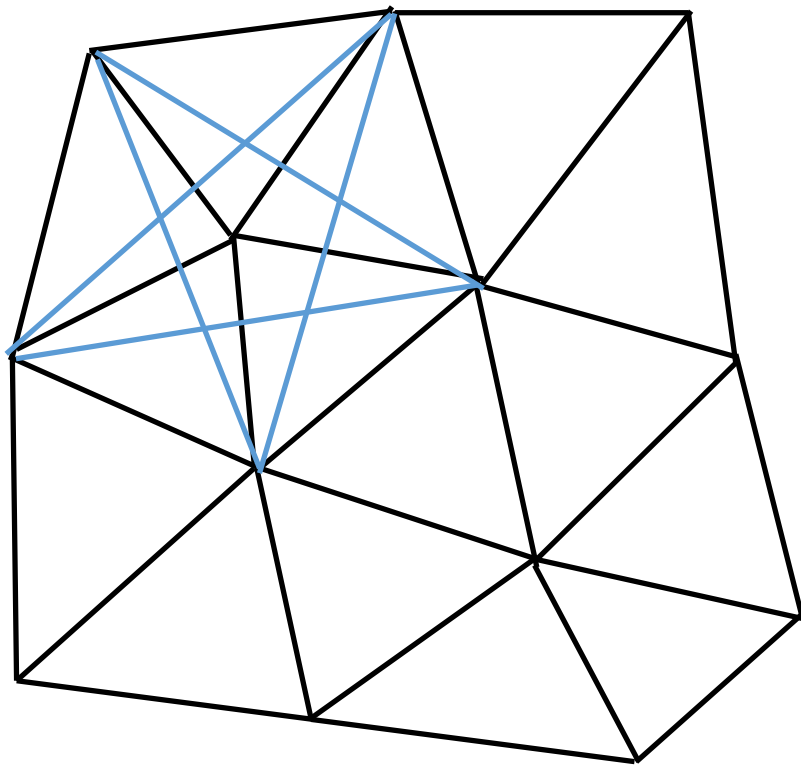
A structured network



A simplified network

# Unstructured Spring Networks

We can also turn an unstructured triangle mesh into a spring network for simulation.

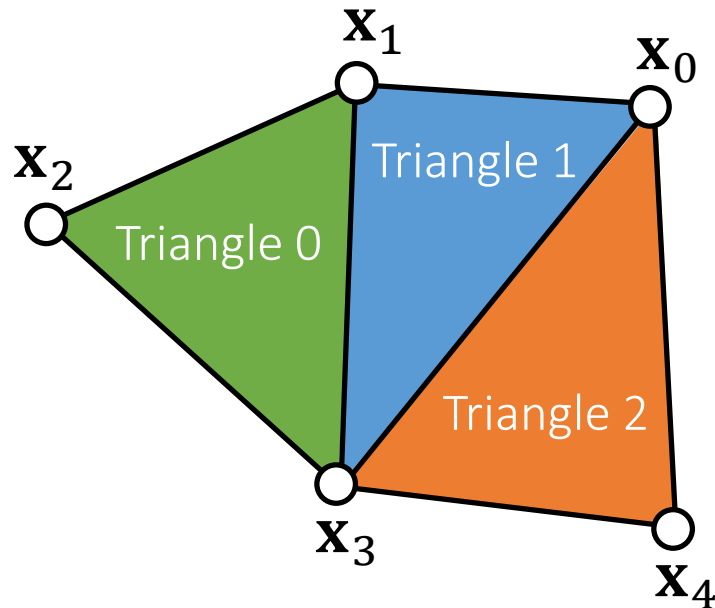


— Edges

— Bending (every neighboring triangle pair)

# Triangle Mesh Representation

The basic representation of a triangle mesh uses vertex and triangle lists.



Vertex list:  $\{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$  (3D vectors)

Triangle list:  $\{1, 2, 3, 0, 1, 3, 0, 3, 4\}$  (index triples)

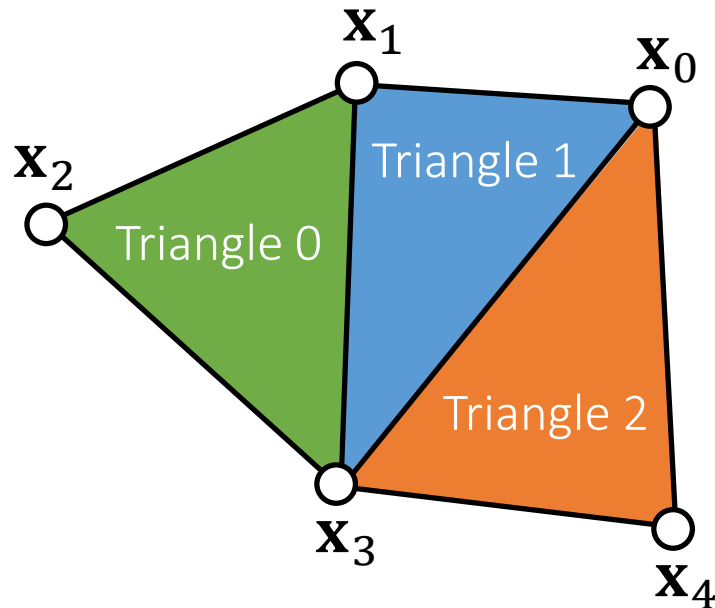
Each triangle has three edges. But there are repeated ones.



# Topological Construction

The key to topological construction is to sort triangle edge triples.

Each triple contains: edge vertex index 0, edge vertex index 1 and triangle index (index  $0 < \text{index}$ ).



Triple list:  $\{\{1, 2, 0\}, \{2, 3, 0\}, \{1, 3, 0\},$   
 $\{0, 1, 1\}, \{1, 3, 1\}, \{0, 3, 1\},$   
 $\{0, 3, 2\}, \{3, 4, 2\}, \{0, 4, 2\}\}$

Sorting

Sorted triple list:  $\{\{0, 1, 1\}, \{0, 3, 1\}, \{0, 3, 2\}, \{0, 4, 2\},$   
 $\{1, 2, 0\}, \{1, 3, 0\}, \{1, 3, 1\}, \{2, 3, 0\}, \{3, 4, 2\}\}$

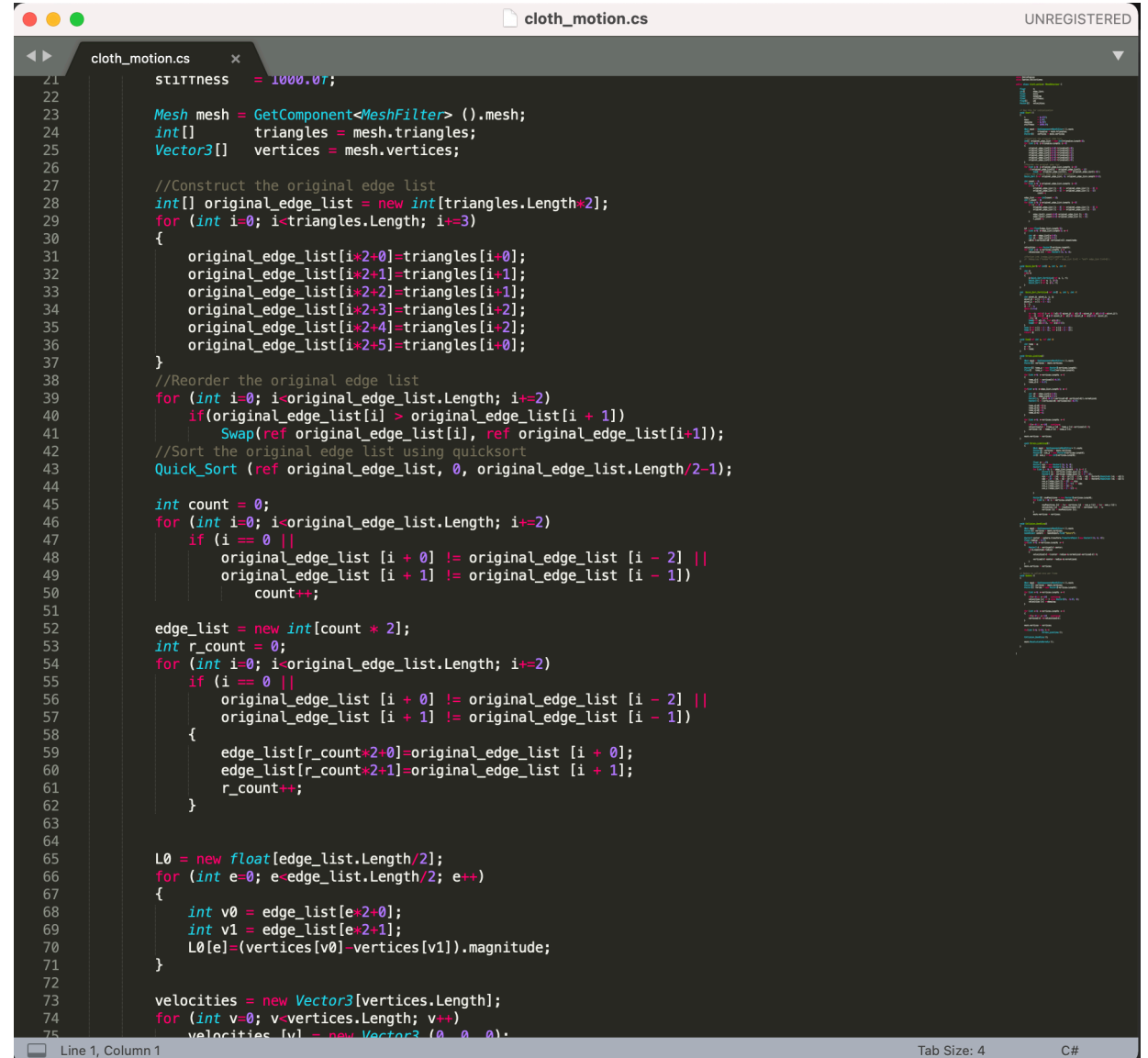
Removal

Edge list:  $\{\{0, 1\}, \{0, 3\}, \{0, 4\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{3, 4\}\}$

Neighboring triangle list:  $\{\{1, 2\}, \{0, 1\}\}$  (for bending)

# After-Class Reading

See lab assignment demo code for details.



```
21 stillness = 1000.0f;
22
23 Mesh mesh = GetComponent<MeshFilter>().mesh;
24 int[] triangles = mesh.triangles;
25 Vector3[] vertices = mesh.vertices;
26
27 //Construct the original edge list
28 int[] original_edge_list = new int[triangles.Length*2];
29 for (int i=0; i<triangles.Length; i+=3)
30 {
31     original_edge_list[i*2+0]=triangles[i+0];
32     original_edge_list[i*2+1]=triangles[i+1];
33     original_edge_list[i*2+2]=triangles[i+1];
34     original_edge_list[i*2+3]=triangles[i+2];
35     original_edge_list[i*2+4]=triangles[i+2];
36     original_edge_list[i*2+5]=triangles[i+0];
37 }
38 //Reorder the original edge list
39 for (int i=0; i<original_edge_list.Length; i+=2)
40     if(original_edge_list[i] > original_edge_list[i + 1])
41         Swap(ref original_edge_list[i], ref original_edge_list[i+1]);
42 //Sort the original edge list using quicksort
43 Quick_Sort (ref original_edge_list, 0, original_edge_list.Length/2-1);
44
45 int count = 0;
46 for (int i=0; i<original_edge_list.Length; i+=2)
47     if (i == 0 ||
48         original_edge_list [i + 0] != original_edge_list [i - 2] ||
49         original_edge_list [i + 1] != original_edge_list [i - 1])
50         count++;
51
52 edge_list = new int[count * 2];
53 int r_count = 0;
54 for (int i=0; i<original_edge_list.Length; i+=2)
55     if (i == 0 ||
56         original_edge_list [i + 0] != original_edge_list [i - 2] ||
57         original_edge_list [i + 1] != original_edge_list [i - 1])
58     {
59         edge_list[r_count*2+0]=original_edge_list [i + 0];
60         edge_list[r_count*2+1]=original_edge_list [i + 1];
61         r_count++;
62     }
63
64 L0 = new float[edge_list.Length/2];
65 for (int e=0; e<edge_list.Length/2; e++)
66 {
67     int v0 = edge_list[e*2+0];
68     int v1 = edge_list[e*2+1];
69     L0[e]=(vertices[v0]-vertices[v1]).magnitude;
70 }
71
72 velocities = new Vector3[vertices.Length];
73 for (int v=0; v<vertices.Length; v++)
74     velocities [v] = new Vector3 (0, 0, 0);
```

# Explicit Integration of A Mass-Spring System

## Compute Spring Forces

For every edge  $e$

$i \leftarrow E[e][0]$

$j \leftarrow E[e][1]$

$L_e \leftarrow L[e]$

$\mathbf{f} \leftarrow -k(\|\mathbf{x}_i - \mathbf{x}_j\| - L_e) \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}$

$\mathbf{f}_i \leftarrow \mathbf{f}_i + \mathbf{f}$

$\mathbf{f}_j \leftarrow \mathbf{f}_j - \mathbf{f}$

$E$ : Edge list       $L$ : Edge length list

## A Particle System

For every vertex

$\mathbf{f}_i \leftarrow \text{Force}(\mathbf{x}_i, \mathbf{v}_i)$

$\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t m_i^{-1} \mathbf{f}_i$

$\mathbf{x}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$

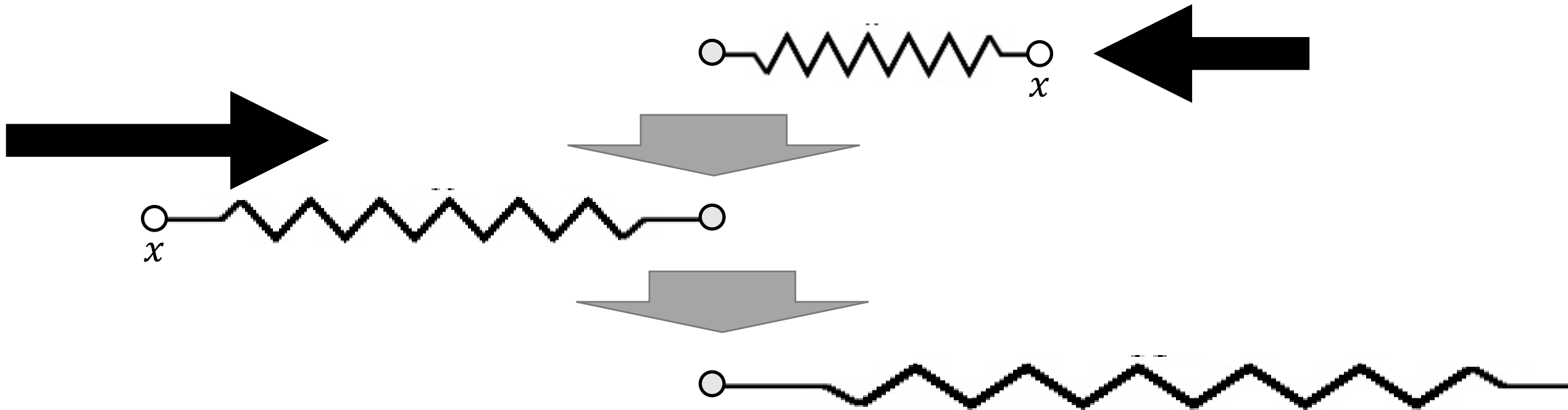
Done

$M_i$ : Mass of vertex  $i$

# Explicit Integration of A Mass-Spring System

Explicit integration suffers from numerical instability caused by overshooting, when the stiffness  $k$  and/or the time step  $\Delta t$  is too large.

A naive solution is to use a small  $\Delta t$ . But that slows down the simulation.



# Implicit Integration

Implicit integration is a better solution to numerical instability. The idea is to integrate both  $\mathbf{x}$  and  $\mathbf{v}$  implicitly.

$$\begin{cases} \mathbf{v}^{[1]} = \mathbf{v}^{[0]} + \Delta t \mathbf{M}^{-1} \mathbf{f}^{[1]} \\ \mathbf{x}^{[1]} = \mathbf{x}^{[0]} + \Delta t \mathbf{v}^{[1]} \end{cases}$$

or

$$\begin{cases} \mathbf{x}^{[1]} = \mathbf{x}^{[0]} + \Delta t \mathbf{v}^{[0]} + \Delta t^2 \mathbf{M}^{-1} \mathbf{f}^{[1]} \\ \mathbf{v}^{[1]} = (\mathbf{x}^{[1]} - \mathbf{x}^{[0]}) / \Delta t \end{cases}$$

Assuming that  $\mathbf{f}$  is *holonomic*, i.e., depending on  $\mathbf{x}$  only, our question is how to solve:

$$\mathbf{x}^{[1]} = \mathbf{x}^{[0]} + \Delta t \mathbf{v}^{[0]} + \Delta t^2 \mathbf{M}^{-1} \mathbf{f}(\mathbf{x}^{[1]})$$

# Implicit Integration

These two are equivalent:

$$\mathbf{x}^{[1]} = \mathbf{x}^{[0]} + \Delta t \mathbf{v}^{[0]} + \Delta t^2 \mathbf{M}^{-1} \mathbf{f}(\mathbf{x}^{[1]})$$

=

$$\|\mathbf{x}\|_{\mathbf{M}}^2 = \mathbf{x}^T \mathbf{M} \mathbf{x}$$

$$\mathbf{x}^{[1]} = \operatorname{argmin} F(\mathbf{x}) \quad \text{for} \quad F(\mathbf{x}) = \frac{1}{2\Delta t^2} \|\mathbf{x} - \mathbf{x}^{[0]} - \Delta t \mathbf{v}^{[0]}\|_{\mathbf{M}}^2 + E(\mathbf{x})$$

This is because:

$$\nabla F(\mathbf{x}^{[1]}) = \frac{1}{\Delta t^2} \mathbf{M}(\mathbf{x}^{[1]} - \mathbf{x}^{[0]} - \Delta t \mathbf{v}^{[0]}) - \mathbf{f}(\mathbf{x}^{[1]}) = \mathbf{0}$$

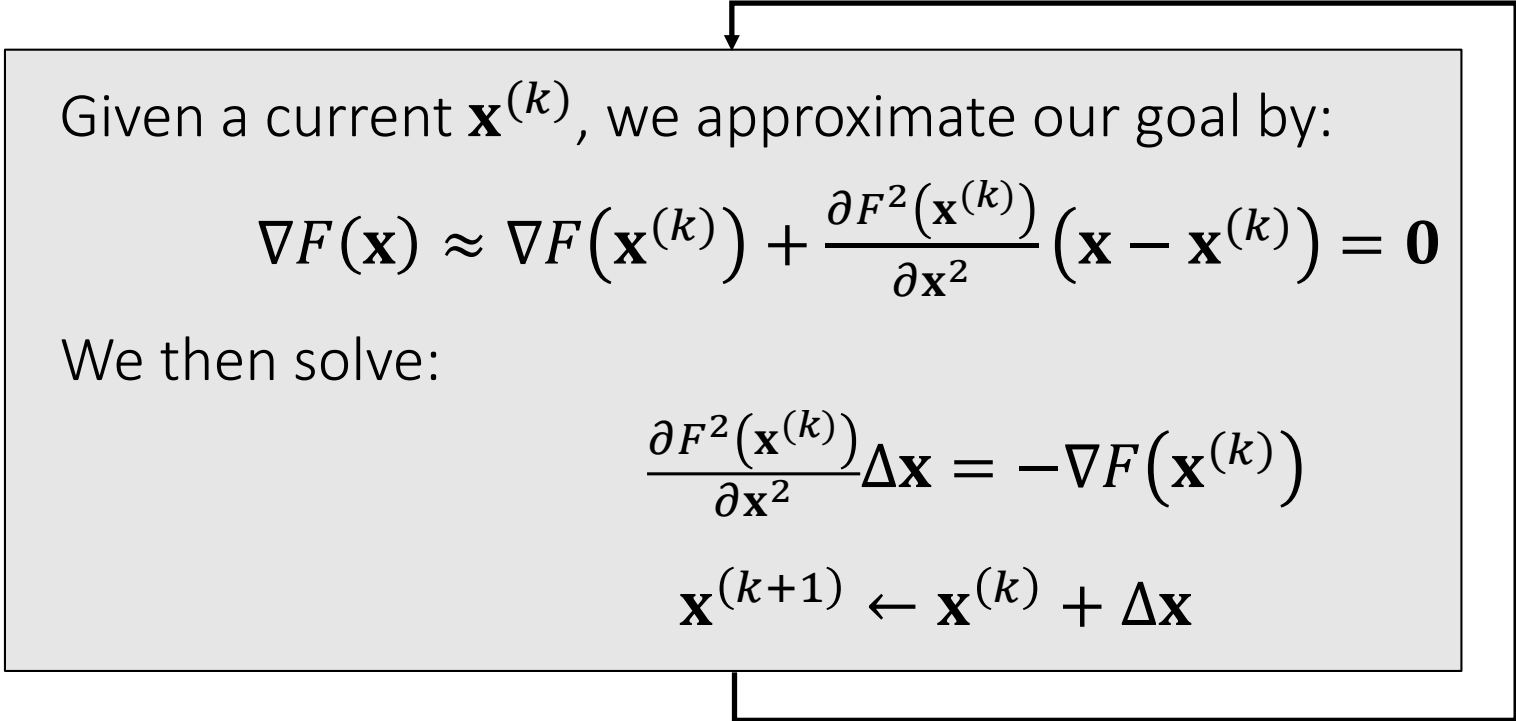


$$\mathbf{x}^{[1]} - \mathbf{x}^{[0]} - \Delta t \mathbf{v}^{[0]} - \Delta t^2 \mathbf{M}^{-1} \mathbf{f}(\mathbf{x}^{[1]}) = \mathbf{0}$$

Note that this is applicable to every system, not just a mass-spring system.

# Newton-Raphson Method

Now we know implicit integration is an optimization problem:  $\mathbf{x}^{[1]} = \operatorname{argmin} F(\mathbf{x})$ , but how to solve it?



Given a current  $\mathbf{x}^{(k)}$ , we approximate our goal by:

The text is enclosed in a light gray box with a black border. An arrow points from the top of the box to the text 'Now we know implicit integration is an optimization problem: ...'. Another arrow points from the bottom of the box to the next iteration step.

$$\nabla F(\mathbf{x}) \approx \nabla F(\mathbf{x}^{(k)}) + \frac{\partial F^2(\mathbf{x}^{(k)})}{\partial \mathbf{x}^2} (\mathbf{x} - \mathbf{x}^{(k)}) = \mathbf{0}$$

We then solve:

$$\frac{\partial F^2(\mathbf{x}^{(k)})}{\partial \mathbf{x}^2} \Delta \mathbf{x} = -\nabla F(\mathbf{x}^{(k)})$$

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \Delta \mathbf{x}$$

# Simulation by Newton's Method

Specifically to simulation, we have:

$$F(\mathbf{x}) = \frac{1}{2\Delta t^2} \|\mathbf{x} - \mathbf{x}^{[0]} - \Delta t \mathbf{v}^{[0]}\|_{\mathbf{M}}^2 + E(\mathbf{x})$$

$$\nabla F(\mathbf{x}^{(k)}) = \frac{1}{\Delta t^2} \mathbf{M}(\mathbf{x}^{(k)} - \mathbf{x}^{[0]} - \Delta t \mathbf{v}^{[0]}) - \mathbf{f}(\mathbf{x}^{(k)})$$

$$\frac{\partial^2 F(\mathbf{x}^{(k)})}{\partial \mathbf{x}^2} = \frac{1}{\Delta t^2} \mathbf{M} + \mathbf{H}(\mathbf{x}^{(k)})$$

Initialize  $\mathbf{x}^{(0)}$ , often as  $\mathbf{x}^{[0]}$  or  $\mathbf{x}^{[0]} + \Delta t \mathbf{v}^{[0]}$

For  $k = 0 \dots K$

$$\text{Solve } \left( \frac{1}{\Delta t^2} \mathbf{M} + \mathbf{H}(\mathbf{x}^{(k)}) \right) \Delta \mathbf{x} = - \frac{1}{\Delta t^2} \mathbf{M}(\mathbf{x}^{(k)} - \mathbf{x}^{[0]} - \Delta t \mathbf{v}^{[0]}) + \mathbf{f}(\mathbf{x}^{(k)})$$

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \Delta \mathbf{x}$$

If  $\|\Delta \mathbf{x}\|$  is small then break

$$\mathbf{x}^{[1]} \leftarrow \mathbf{x}^{(k+1)}$$

$$\mathbf{v}^{[1]} \leftarrow (\mathbf{x}^{[1]} - \mathbf{x}^{[0]}) / \Delta t$$



# Spring Hessian

According to Lecture 2, Page 48,

$$\mathbf{H}(\mathbf{x}) = \sum_{e=\{i,j\}} \begin{bmatrix} \frac{\partial^2 E}{\partial \mathbf{x}_i^2} & \frac{\partial^2 E}{\partial \mathbf{x}_i \partial \mathbf{x}_j} \\ \frac{\partial^2 E}{\partial \mathbf{x}_i \partial \mathbf{x}_j} & \frac{\partial^2 E}{\partial \mathbf{x}_j^2} \end{bmatrix} = \sum_{e=\{i,j\}} \begin{bmatrix} \mathbf{H}_e & -\mathbf{H}_e \\ -\mathbf{H}_e & \mathbf{H}_e \end{bmatrix}$$

$$\mathbf{H}_e = \underbrace{k \frac{\mathbf{x}_{ij} \mathbf{x}_{ij}^T}{\|\mathbf{x}_{ij}\|^2}}_{\text{s.p.d.}} + k \underbrace{\left(1 - \frac{L}{\|\mathbf{x}_{ij}\|}\right)}_{\text{negative if } \|\mathbf{x}_{ij}\| < L_e} \underbrace{\left(\mathbf{I} - \frac{\mathbf{x}_{ij} \mathbf{x}_{ij}^T}{\|\mathbf{x}_{ij}\|^2}\right)}_{\text{s.p.d.}}$$

This is because for any  $\mathbf{x}_{ij}$ ,  $\mathbf{v} \neq \mathbf{0}$ ,

$$\mathbf{v}^T \frac{\mathbf{x}_{ij} \mathbf{x}_{ij}^T}{\|\mathbf{x}_{ij}\|^2} \mathbf{v} = \left\| \frac{\mathbf{x}_{ij}^T \mathbf{v}}{\|\mathbf{x}_{ij}\|} \right\|^2 > 0$$

$$\mathbf{v}^T \left( \mathbf{I} - \frac{\mathbf{x}_{ij} \mathbf{x}_{ij}^T}{\|\mathbf{x}_{ij}\|^2} \right) \mathbf{v} = \frac{\|\mathbf{x}_{ij}\|^2 \|\mathbf{v}\|^2 - \|\mathbf{x}_{ij}^T \mathbf{v}\|^2}{\|\mathbf{x}_{ij}\|^2} > 0$$

# Spring Hessian

When a spring is stretched,  $\mathbf{H}_e$  is s.p.d.; but when it's compressed,  $\mathbf{H}_e$  may not be s.p.d.

As a result,  $\mathbf{H}(\mathbf{x})$  may not be s.p.d. (Lecture 2, Page 36).

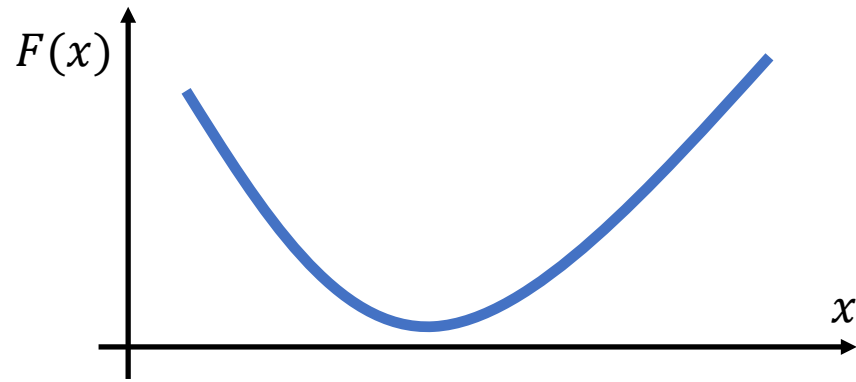
$\mathbf{A}$  may not be s.p.d. either.

$$\mathbf{A} = \frac{1}{\Delta t^2} \mathbf{M} + \mathbf{H}(\mathbf{x}) = \underbrace{\frac{1}{\Delta t^2} \mathbf{M}}_{\text{s.p.d.}} + \sum_{e=\{i,j\}} \left[ \begin{array}{ccc} \ddots & \vdots & \vdots & \ddots \\ & \mathbf{H}_e & -\mathbf{H}_e & \\ & -\mathbf{H}_e & \mathbf{H}_e & \\ & \vdots & \vdots & \ddots \end{array} \right]$$

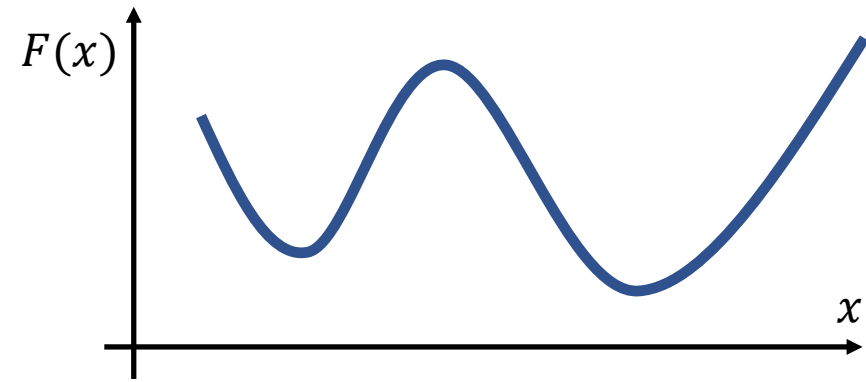
May not be s.p.d.

# Positive Definiteness of Hessian

Consider a real function  $F(x) \in \mathbf{R}$ ,



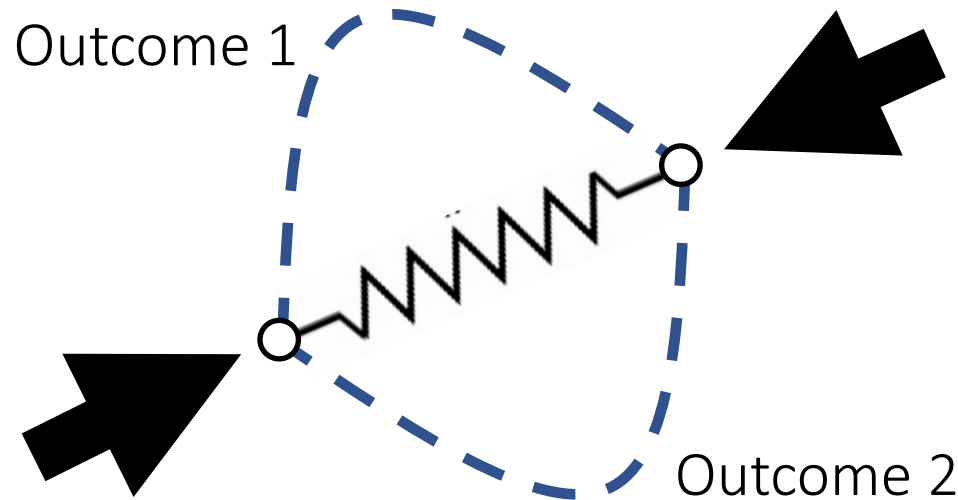
If  $F''(x) > 0$  for any  $x$ ,  $F'(x)$  must be monotonic and  $F(x)$  must have only one minimal solution.



Otherwise, it's possible to have multiple local minima.

# Positive Definiteness of Hessian

When a spring is compressed, the spring Hessian may not be positive definite. This means there can be multiple local minima (outcomes).



Note: This issue occurs only in 2D and 3D. In 1D,  $E(x) = \frac{1}{2}k(x - L)^2$  and  $E''(x) = k > 0$ .

# Enforcement of Positive Definiteness

- Nevertheless, some linear solvers can fail to work if the matrix  $\mathbf{A}$  in  $\mathbf{A}\Delta\mathbf{x} = \mathbf{b}$  is not positive definite.
- One solution is to simply drop the ending term, when  $\|\mathbf{x}_{ij}\| < L_e$ :

$$\mathbf{H}_e = k \frac{\mathbf{x}_{ij}\mathbf{x}_{ij}^T}{\|\mathbf{x}_{ij}\|^2} + k \left( 1 - \frac{L}{\|\mathbf{x}_{ij}\|} \right) \left( \mathbf{I} - \frac{\mathbf{x}_{ij}\mathbf{x}_{ij}^T}{\|\mathbf{x}_{ij}\|^2} \right)$$

- Other solutions exist. For example,
  - Choi and Ko. 2002. Stable But Responsive Cloth. TOG (SIGGRAPH)

# The Jacobi Method

We can use the Jacobi method to solve  $\mathbf{A}\Delta\mathbf{x} = \mathbf{b}$ .

The Jacobi Method

$\Delta\mathbf{x} \leftarrow \mathbf{0}$

For  $k = 0 \dots K$

$\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\Delta\mathbf{x}$

    If  $\|\mathbf{r}\| < \varepsilon$       break

$\Delta\mathbf{x} \leftarrow \Delta\mathbf{x} + \alpha\mathbf{D}^{-1}\mathbf{r}$

← - - - Residual error

← - - - Convergence condition  $\varepsilon$

← - - - Update by  $\mathbf{D}$ , the diagonal of  $\mathbf{A}$

The vanilla Jacobi method ( $\alpha = 1$ ) has a tight convergence requirement on  $\mathbf{A}$ , i.e., being diagonal dominant.

The use of  $\alpha$  allows the method to converge even when  $\mathbf{A}$  is positive definite only.

# Linear Solvers – An Incomplete Summary

- Direct Solvers (LU, LDLT, Cholesky, ...)

Intel MKL PARDISO

- One shot, expensive but worthy if you need exact solutions.
- Little restriction on **A**
- Mostly suitable on CPUs

- Iterative Solvers

- Expensive to solve exactly, but controllable
- Convergence restriction on **A**, typically positive definiteness
- Suitable on both CPUs and GPUs
- Easy to implement
- Accelerable: Chebyshev, Nesterov, Conjugate Gradient...

# The Jacobi Method with Chebyshev Acceleration

We can use the accelerated Jacobi method to solve  $\mathbf{A}\Delta\mathbf{x} = \mathbf{b}$ .

## The Accelerated Jacobi Method

$\Delta\mathbf{x} \leftarrow \mathbf{0}$

$\text{last\_}\Delta\mathbf{x} \leftarrow \mathbf{0}$

For  $k = 0 \dots K$

$\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\Delta\mathbf{x}$

If  $\|\mathbf{r}\| < \varepsilon$       break

If  $k = 0$        $\omega = 1$

Else If  $k = 1$        $\omega = 2/(2 - \rho^2)$

Else       $\omega = 4/(4 - \rho^2\omega)$

$\text{old\_}\Delta\mathbf{x} \leftarrow \Delta\mathbf{x}$

$\Delta\mathbf{x} \leftarrow \Delta\mathbf{x} + \alpha\mathbf{D}^{-1}\mathbf{r}$

$\Delta\mathbf{x} \leftarrow \omega\Delta\mathbf{x} + (1 - \omega)\text{last\_}\Delta\mathbf{x}$

$\text{last\_}\Delta\mathbf{x} \leftarrow \text{old\_}\Delta\mathbf{x}$

$\rho$  ( $\rho < 1$ ) is the estimated spectral radius of the iterative matrix.



# After-Class Reading

Baraff and Witkin. 1998. Large Step in Cloth Simulation. SIGGRAPH.

One of the first papers using implicit integration.

The paper proposes to use only one Newton iteration, i.e., solving only one linear system. This practice is fast, but can fail to converge.

## Large Steps in Cloth Simulation

David Baraff Andrew Witkin

Robotics Institute  
Carnegie Mellon University

### Abstract

The bottle-neck in most cloth simulation systems is that time steps must be small to avoid numerical instability. This paper describes a cloth simulation system that can stably take large time steps. The simulation system couples a new technique for enforcing constraints on individual cloth particles with an implicit integration method. The simulator models cloth as a triangular mesh, with internal cloth forces derived using a simple continuum formulation that supports modeling operations such as local anisotropic stretch or compression; a unified treatment of damping forces is included as well. The implicit integration method generates a large, unbanded sparse linear system at each time step which is solved using a modified conjugate gradient method that simultaneously enforces particles' constraints. The constraints are always maintained exactly, independent of the number of conjugate gradient iterations, which is typically small. The resulting simulation system is significantly faster than previous accounts of cloth simulation systems in the literature.

**Keywords**—Cloth, simulation, constraints, implicit integration, physically-based modeling.

### 1 Introduction

Physically-based cloth animation has been a problem of interest to the graphics community for more than a decade. Early work by Terzopoulos *et al.* [17] and Terzopoulos and Fleischer [15, 16] on deformable models correctly characterized cloth simulation as a problem in deformable surfaces, and applied techniques from the mechanical engineering and finite element communities to the problem. Since then, other research groups (notably Carignan *et al.* [4] and Volino *et al.* [20, 21]; Breen *et al.* [3]; and Eberhardt *et al.* [5]) have taken up the challenge of cloth.

Although specific details vary (underlying representations, numerical solution methods, collision detection and constraint methods, etc.), there is a deep commonality amongst all the approaches: physically-based cloth simulation is formulated as a time-varying partial differential equation which, after discretization, is numerically solved as an ordinary differential equation

$$\ddot{\mathbf{x}} = \mathbf{M}^{-1} \left( -\frac{\partial E}{\partial \mathbf{x}} + \mathbf{F} \right). \quad (1)$$

In this equation the vector  $\mathbf{x}$  and diagonal matrix  $\mathbf{M}$  represent the geometric state and mass distribution of the cloth,  $E$ —a scalar func-

tion of  $\mathbf{x}$ —yields the cloth's internal energy, and  $\mathbf{F}$  (a function of  $\mathbf{x}$  and  $\dot{\mathbf{x}}$ ) describes other forces (air-drag, contact and constraint forces, internal damping, etc.) acting on the cloth.

In this paper, we describe a cloth simulation system that is much faster than previously reported simulation systems. Our system's faster performance begins with the choice of an *implicit* numerical integration method to solve equation (1). The reader should note that the use of implicit integration methods in cloth simulation is far from novel: initial work by Terzopoulos *et al.* [15, 16, 17] applied such methods to the problem.<sup>1</sup> Since this time though, research on cloth simulation has generally relied on *explicit* numerical integration (such as Euler's method or Runge-Kutta methods) to advance the simulation, or, in the case of energy minimization, analogous methods such as steepest-descent [3, 10].

This is unfortunate. Cloth strongly resists stretching motions while being comparatively permissive in allowing bending or shearing motions. This results in a "stiff" underlying differential equation of motion [12]. Explicit methods are ill-suited to solving stiff equations because they require many small steps to stably advance the simulation forward in time.<sup>2</sup> In practice, the computational cost of an explicit method greatly limits the realizable resolution of the cloth. For some applications, the required spatial resolution—that is, the dimension  $n$  of the state vector  $\mathbf{x}$ —can be quite low: a resolution of only a few hundred particles (or nodal points, depending on your formulation/terminology) can be sufficient when it comes to modeling flags or tablecloths. To animate clothing, which is our main concern, requires much higher spatial resolution to adequately represent realistic (or even semi-realistic) wrinkling and folding configurations.

In this paper, we demonstrate that implicit methods for cloth overcome the performance limits inherent in explicit simulation methods. We describe a simulation system that uses a triangular mesh for cloth surfaces, eliminating topological restrictions of rectangular meshes, and a simple but versatile formulation of the internal cloth energy forces. (Unlike previous metric-tensor-based formulations [15, 16, 17, 4] which model some deformation energies as quartic functions of positions, we model deformation energies only as quadratic functions with suitably large scaling. Quadratic energy models mesh well with implicit integration's numerical properties.) We also introduce a simple, unified treatment of damping forces, a subject which has been largely ignored thus far. A key step in our simulation process is the solution of an  $O(n) \times O(n)$  sparse linear system, which arises from the implicit integration method. In this respect, our implementation differs greatly from the implementation by Terzopoulos *et al.* [15, 17], which for large simulations

<sup>1</sup>Additional use of implicit methods in animation and dynamics work includes Kass and Miller [8], Terzopoulos and Qin [18], and Tu [19].

<sup>2</sup>Even worse, the number of time steps per frame tends to increase along with the problem size, for an explicit method. Cloth simulations of size  $n$ —meaning  $\mathbf{x} \in \mathbb{R}^{O(n)}$ —generally require  $O(n)$  explicit steps per unit simulated time. Because the cost of an explicit step is also  $O(n)$  (setting aside complications such as collision detection for now) explicit methods for cloth require time  $O(n^2)$ —or worse.

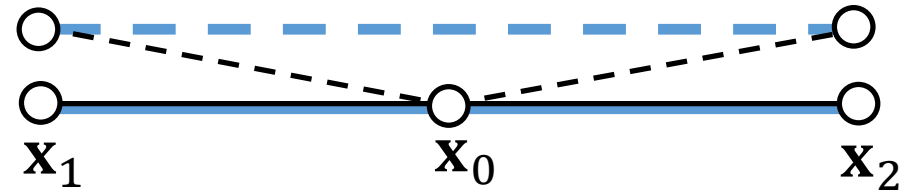
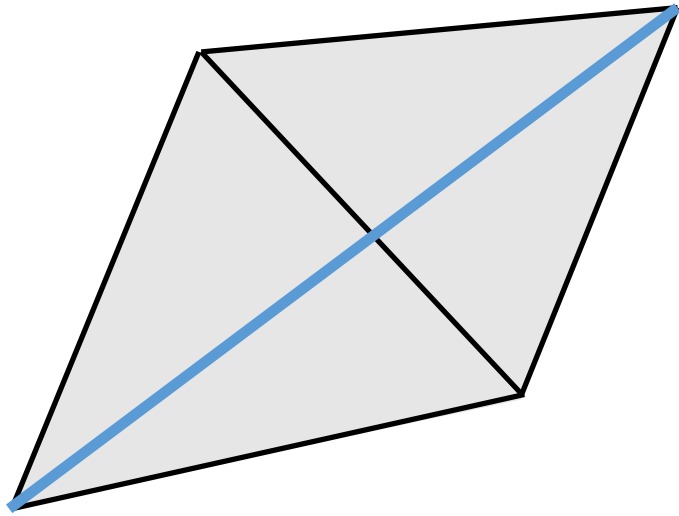
Author affiliation (September 1998): David Baraff, Andrew Witkin, Pixar Animation Studios, 1001 West Cutting Blvd., Richmond, CA 94804. Email: deb@pixar.com, aw@pixar.com.

This is an electronic reprint. Permission is granted to copy part or all of this paper for noncommercial use provided that the title and this copyright notice appear. This electronic reprint is ©1998 by CMU. The original printed paper is ©1998 by the ACM.

# Bending and Locking Issues

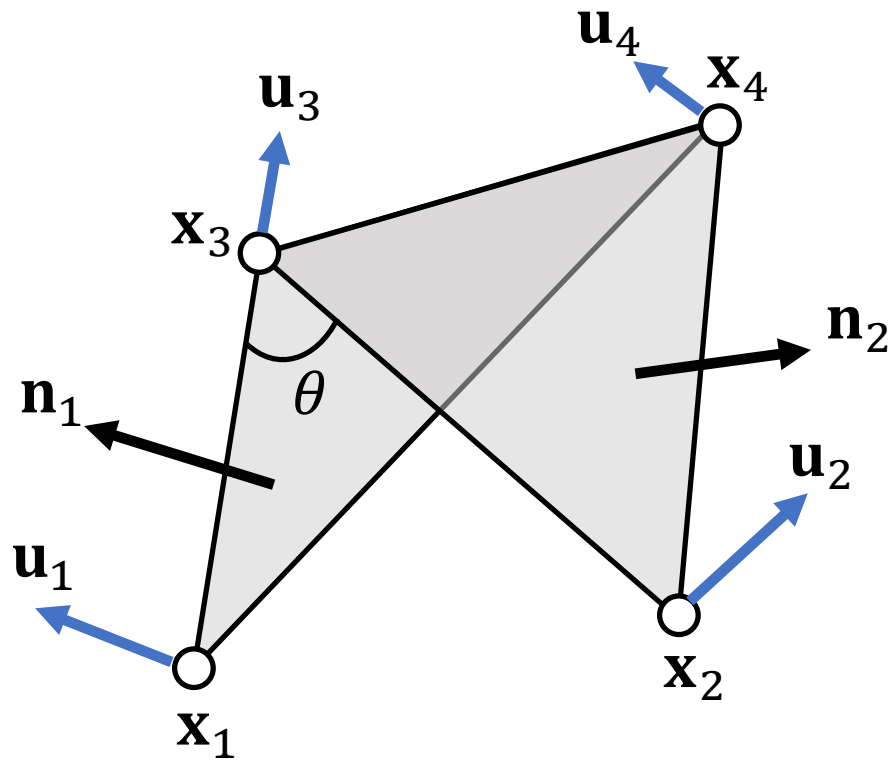
# The Bending Spring Issue

A **bending** spring offers little resistance when cloth is nearly planar, since its length barely changes.



# A Dihedral Angle Model

A dihedral angle model defines bending forces as a function of  $\theta$ :  $\mathbf{f}_i = f(\theta)\mathbf{u}_i$ .



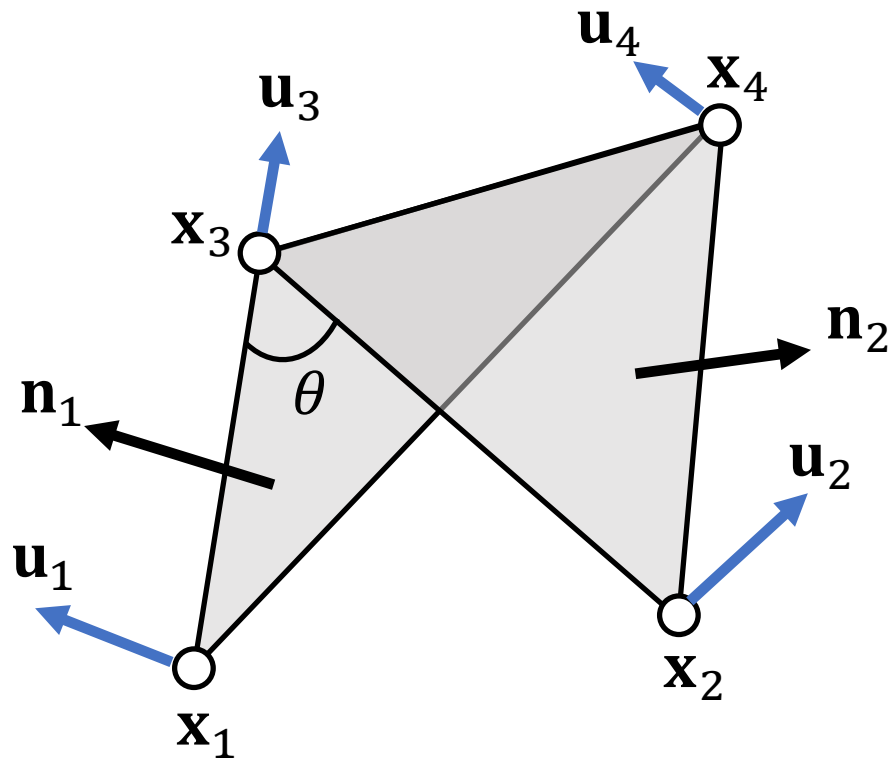
First,  $\mathbf{u}_1$  and  $\mathbf{u}_2$  should be in the normal directions  $\mathbf{n}_1$  and  $\mathbf{n}_2$ .

Second, bending doesn't stretch the edge, so  $\mathbf{u}_4 - \mathbf{u}_3$  should be orthogonal to the edge, i.e., in the span of  $\mathbf{n}_1$  and  $\mathbf{n}_2$ .

Finally,  $\mathbf{u}_1 + \mathbf{u}_2 + \mathbf{u}_3 + \mathbf{u}_4 = 0$ , which means  $\mathbf{u}_3$  and  $\mathbf{u}_4$  are in the span of  $\mathbf{n}_1$  and  $\mathbf{n}_2$ .

# A Dihedral Angle Model

A dihedral angle model defines bending forces as a function of  $\theta$ :  $\mathbf{f}_i = f(\theta)\mathbf{u}_i$ .



Conclusion:

$$\mathbf{u}_1 = \|\mathbf{E}\| \frac{\mathbf{N}_1}{\|\mathbf{N}_1\|^2}$$

$$\mathbf{u}_2 = \|\mathbf{E}\| \frac{\mathbf{N}_2}{\|\mathbf{N}_2\|^2}$$

$$\mathbf{u}_3 = \frac{(\mathbf{x}_1 - \mathbf{x}_4) \cdot \mathbf{E}}{\|\mathbf{E}\|} \frac{\mathbf{N}_1}{\|\mathbf{N}_1\|^2} + \frac{(\mathbf{x}_2 - \mathbf{x}_4) \cdot \mathbf{E}}{\|\mathbf{E}\|} \frac{\mathbf{N}_2}{\|\mathbf{N}_2\|^2}$$

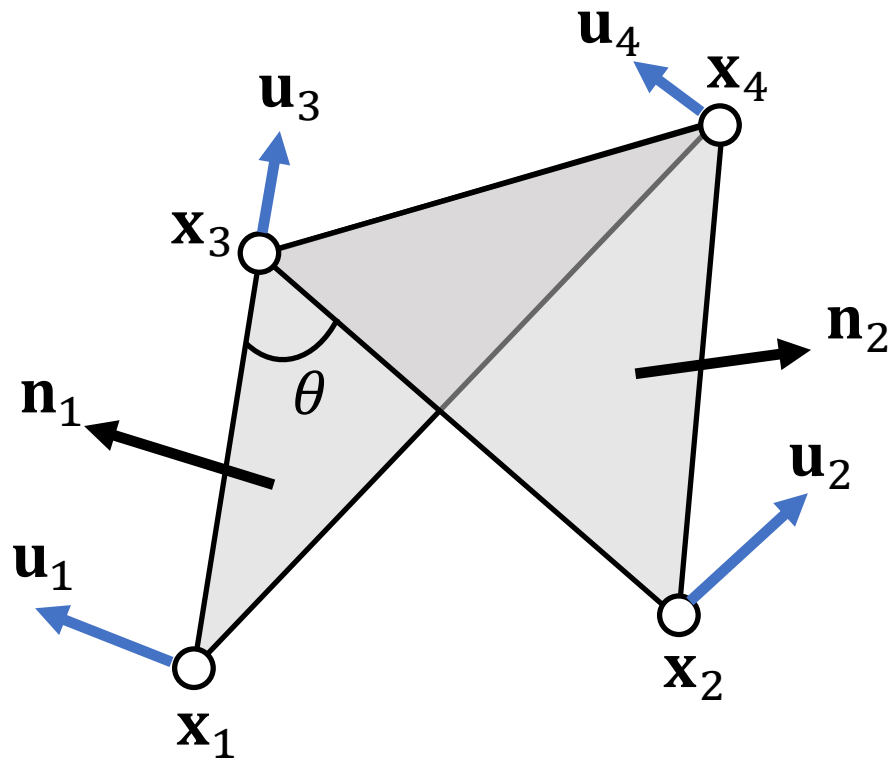
$$\mathbf{u}_4 = -\frac{(\mathbf{x}_1 - \mathbf{x}_3) \cdot \mathbf{E}}{\|\mathbf{E}\|} \frac{\mathbf{N}_1}{\|\mathbf{N}_1\|^2} - \frac{(\mathbf{x}_2 - \mathbf{x}_3) \cdot \mathbf{E}}{\|\mathbf{E}\|} \frac{\mathbf{N}_2}{\|\mathbf{N}_2\|^2}$$

$$\mathbf{N}_1 = (\mathbf{x}_1 - \mathbf{x}_3) \times (\mathbf{x}_1 - \mathbf{x}_4) \quad \mathbf{N}_2 = (\mathbf{x}_2 - \mathbf{x}_4) \times (\mathbf{x}_2 - \mathbf{x}_3)$$

$$\mathbf{E} = \mathbf{x}_4 - \mathbf{x}_3$$

# A Dihedral Angle Model

A dihedral angle model defines bending forces as a function of  $\theta$ :  $\mathbf{f}_i = f(\theta)\mathbf{u}_i$ .



Planar case:

$$\mathbf{f}_i = k \frac{\|\mathbf{E}\|^2}{\|\mathbf{N}_1\| + \|\mathbf{N}_2\|} \sin\left(\frac{\pi - \theta}{2}\right) \mathbf{u}_i$$

Non-planar case:

$$\mathbf{f}_i = k \frac{\|\mathbf{E}\|^2}{\|\mathbf{N}_1\| + \|\mathbf{N}_2\|} \left( \sin\left(\frac{\pi - \theta}{2}\right) - \sin\left(\frac{\pi - \theta_0}{2}\right) \right) \mathbf{u}_i$$

# After-Class Reading

Bridson et al. 2003. *Simulation of Clothing with Folds and Wrinkles*. SCA.

Explicit integration.  
Derivative is difficult to compute.

## Large Steps in Cloth Simulation

David Baraff Andrew Witkin

Robotics Institute  
Carnegie Mellon University

### Abstract

The bottle-neck in most cloth simulation systems is that time steps must be small to avoid numerical instability. This paper describes a cloth simulation system that can stably take large time steps. The simulation system couples a new technique for enforcing constraints on individual cloth particles with an implicit integration method. The simulator models cloth as a triangular mesh, with internal cloth forces derived using a simple continuum formulation that supports modeling operations such as local anisotropic stretch or compression; a unified treatment of damping forces is included as well. The implicit integration method generates a large, unbanded sparse linear system at each time step which is solved using a modified conjugate gradient method that simultaneously enforces particles' constraints. The constraints are always maintained exactly, independent of the number of conjugate gradient iterations, which is typically small. The resulting simulation system is significantly faster than previous accounts of cloth simulation systems in the literature.

**Keywords**—Cloth, simulation, constraints, implicit integration, physically-based modeling.

### 1 Introduction

Physically-based cloth animation has been a problem of interest to the graphics community for more than a decade. Early work by Terzopoulos *et al.* [17] and Terzopoulos and Fleischer [15, 16] on deformable models correctly characterized cloth simulation as a problem in deformable surfaces, and applied techniques from the mechanical engineering and finite element communities to the problem. Since then, other research groups (notably Carignan *et al.* [4] and Volino *et al.* [20, 21]; Breen *et al.* [3]; and Eberhardt *et al.* [5]) have taken up the challenge of cloth.

Although specific details vary (underlying representations, numerical solution methods, collision detection and constraint methods, etc.), there is a deep commonality amongst all the approaches: physically-based cloth simulation is formulated as a time-varying partial differential equation which, after discretization, is numerically solved as an ordinary differential equation

$$\ddot{\mathbf{x}} = \mathbf{M}^{-1} \left( -\frac{\partial E}{\partial \mathbf{x}} + \mathbf{F} \right). \quad (1)$$

In this equation the vector  $\mathbf{x}$  and diagonal matrix  $\mathbf{M}$  represent the geometric state and mass distribution of the cloth,  $E$ —a scalar func-

tion of  $\mathbf{x}$ —yields the cloth's internal energy, and  $\mathbf{F}$  (a function of  $\mathbf{x}$  and  $\dot{\mathbf{x}}$ ) describes other forces (air-drag, contact and constraint forces, internal damping, etc.) acting on the cloth.

In this paper, we describe a cloth simulation system that is much faster than previously reported simulation systems. Our system's faster performance begins with the choice of an *implicit* numerical integration method to solve equation (1). The reader should note that the use of implicit integration methods in cloth simulation is far from novel: initial work by Terzopoulos *et al.* [15, 16, 17] applied such methods to the problem.<sup>1</sup> Since this time though, research on cloth simulation has generally relied on *explicit* numerical integration (such as Euler's method or Runge-Kutta methods) to advance the simulation, or, in the case of energy minimization, analogous methods such as steepest-descent [3, 10].

This is unfortunate. Cloth strongly resists stretching motions while being comparatively permissive in allowing bending or shearing motions. This results in a "stiff" underlying differential equation of motion [12]. Explicit methods are ill-suited to solving stiff equations because they require many small steps to stably advance the simulation forward in time.<sup>2</sup> In practice, the computational cost of an explicit method greatly limits the realizable resolution of the cloth. For some applications, the required spatial resolution—that is, the dimension  $n$  of the state vector  $\mathbf{x}$ —can be quite low: a resolution of only a few hundred particles (or nodal points, depending on your formulation/terminology) can be sufficient when it comes to modeling flags or tablecloths. To animate clothing, which is our main concern, requires much higher spatial resolution to adequately represent realistic (or even semi-realistic) wrinkling and folding configurations.

In this paper, we demonstrate that implicit methods for cloth overcome the performance limits inherent in explicit simulation methods. We describe a simulation system that uses a triangular mesh for cloth surfaces, eliminating topological restrictions of rectangular meshes, and a simple but versatile formulation of the internal cloth energy forces. (Unlike previous metric-tensor-based formulations [15, 16, 17, 4] which model some deformation energies as quartic functions of positions, we model deformation energies only as quadratic functions with suitably large scaling. Quadratic energy models mesh well with implicit integration's numerical properties.) We also introduce a simple, unified treatment of damping forces, a subject which has been largely ignored thus far. A key step in our simulation process is the solution of an  $O(n) \times O(n)$  sparse linear system, which arises from the implicit integration method. In this respect, our implementation differs greatly from the implementation by Terzopoulos *et al.* [15, 17], which for large simulations

<sup>1</sup>Additional use of implicit methods in animation and dynamics work includes Kass and Miller [8], Terzopoulos and Qin [18], and Tu [19].

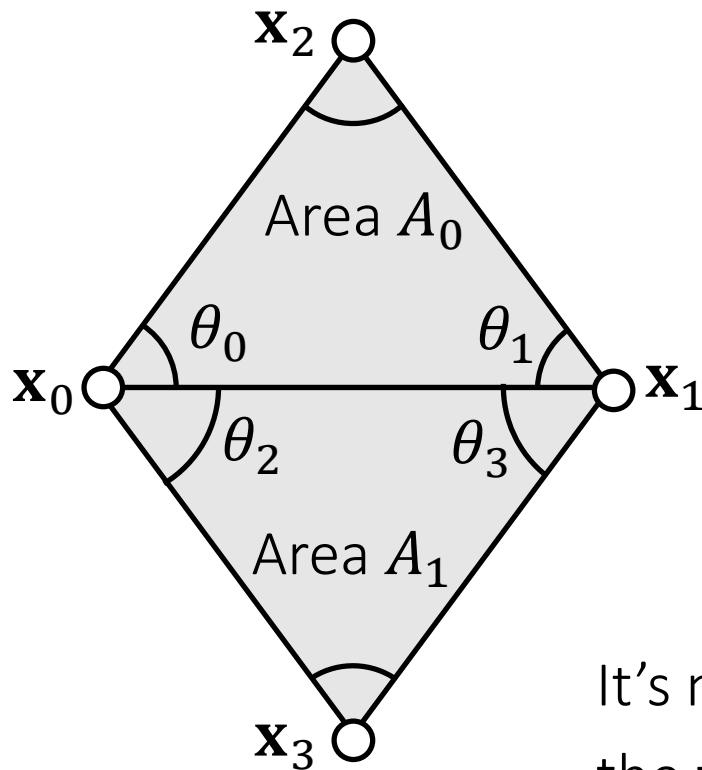
<sup>2</sup>Even worse, the number of time steps per frame tends to increase along with the problem size, for an explicit method. Cloth simulations of size  $n$ —meaning  $\mathbf{x} \in \mathbb{R}^{O(n)}$ —generally require  $O(n)$  explicit steps per unit simulated time. Because the cost of an explicit step is also  $O(n)$  (setting aside complications such as collision detection for now) explicit methods for cloth require time  $O(n^2)$ —or worse.

Author affiliation (September 1998): David Baraff, Andrew Witkin, Pixar Animation Studios, 1001 West Cutting Blvd., Richmond, CA 94804. Email: deb@pixar.com, aw@pixar.com.

This is an electronic reprint. Permission is granted to copy part or all of this paper for noncommercial use provided that the title and this copyright notice appear. This electronic reprint is ©1998 by CMU. The original printed paper is ©1998 by the ACM.

# A Quadratic Bending Model

A quadratic bending model has two assumptions: 1) planar case; 2) little stretching.



$$E(\mathbf{x}) = \frac{1}{2} [\mathbf{x}_0 \quad \mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3] \mathbf{Q} \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{bmatrix}$$

$$\mathbf{Q} = \frac{3}{A_0 + A_1} \mathbf{q} \mathbf{q}^T$$

$$\mathbf{q} = \begin{bmatrix} (\cot\theta_1 + \cot\theta_3)\mathbf{1} \\ (\cot\theta_0 + \cot\theta_2)\mathbf{1} \\ (-\cot\theta_0 - \cot\theta_1)\mathbf{1} \\ (-\cot\theta_2 - \cot\theta_3)\mathbf{1} \end{bmatrix}$$

It's not hard to see that:  $E(\mathbf{x}) = \frac{3(\mathbf{q}^T \mathbf{x})^2}{2(A_0 + A_1)}$ . Also,  $E(\mathbf{x}) = 0$  when the triangles are flat.



# Pros and Cons of The Quadratic Bending Model

- Easy to implement:

$$\mathbf{f}(\mathbf{x}) = -\nabla E(\mathbf{x}) = -\mathbf{Q} \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{bmatrix} \quad \mathbf{H}(\mathbf{x}) = \frac{\partial^2 E(\mathbf{x})}{\partial \mathbf{x}^2} = \mathbf{Q}$$

- Compatible with implicit integration.
- No longer valid if cloth stretches much.
- Not suitable if the rest configuration is not planar.
  - Cubic shell model.
  - Projective dynamics model.
  - Details skipped here.

# After-Class Reading

Bergou et al. 2006. *A Quadratic Bending Model for Inextensible Surfaces*. SCA.

Eurographics Symposium on Geometry Processing (2006)  
Konrad Polthier, Alla Sheffer (Editors)

## A Quadratic Bending Model for Inextensible Surfaces

Miklós Bergou Columbia University    Max Wardetzky Freie Universität Berlin    David Harmon Columbia University    Denis Zorin New York University    Eitan Grinspun Columbia University

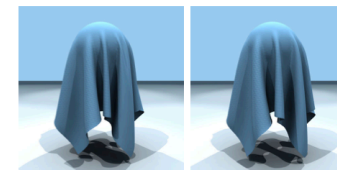
### Abstract

*Relating the intrinsic Laplacian to the mean curvature normal, we arrive at a model for bending of inextensible surfaces. Due to its constant Hessian, our isometric bending model reduces cloth simulation times up to three-fold.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

### 1. Introduction

Computation of curvature-based energies and their derivatives is a costly component of many physical simulation and geometric modeling applications. Typically the energy density is expressed in terms of elementary symmetric functions of the principal curvatures of the mesh [Cia00, YB02, CDD<sup>+</sup>04, BS05, TW06, GGRZ06]). In general the resulting expressions are nonlinear in the positions of mesh vertices, and the attendant numerics involve costly evaluations of energy gradients and Hessians. Our contribution is to consider the class of isometric surface deformations, arriving at an expression for bending energy which is quadratic in positions. Such quasi-isometric deformations are typical, e.g., for inextensible plates and shells where membrane (stretching) stiffness is greater than bending stiffness by four or more orders of magnitude, hence we focus on cloth simulation as a primary application area.



**Figure 1:** Final rest state of a cloth draped over a sphere, for (left) the proposed isometric bending model and (right) the widely-adopted nonlinear hinge model.

**Continuous setting.** Consider the bending energy of a deformable surface  $S$ :

$$E_b(S) = \frac{1}{2} \int_S H^2 dA, \quad (1)$$

where  $H$  is mean curvature and  $dA$  is the differential area.  $E_b(S)$  is closely related to the Willmore energy of a surface, and the Canham-Helfrich energy of thin bilipid membranes. Note the invariance of  $E_b(S)$  under (i) rigid motions and (ii) uniform scaling of the surface: (i) is required for conservation of linear and angular momenta (Nöther's theorem); (ii) affects the characteristic size of folds and wrinkles.

We may rewrite (1) by the following argument. If  $\mathbf{x} : S \rightarrow \mathbb{R}^3$  denotes the embedding of the surface, the mean curvature normal  $\mathbf{H}$  of  $S$  can be written as the Laplace-Beltrami,  $\Delta$ , induced by the Riemannian metric of  $S$ , applied to the embedding of the surface:  $\mathbf{H} = \Delta \mathbf{x}$ . Thus we write (1) as

$$E_b(S) = \frac{1}{2} \int_S \langle \Delta \mathbf{x}, \Delta \mathbf{x} \rangle_{\mathbb{R}^3} dA, \quad (2)$$

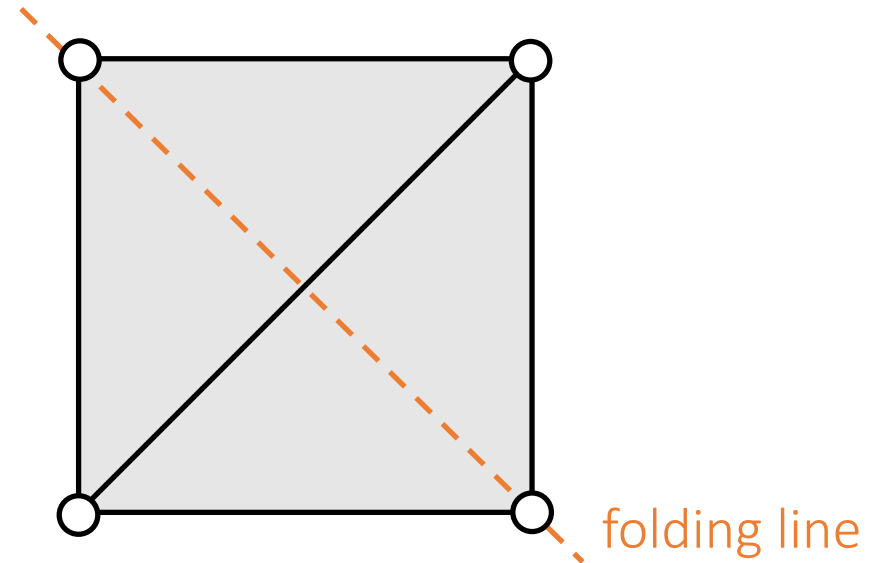
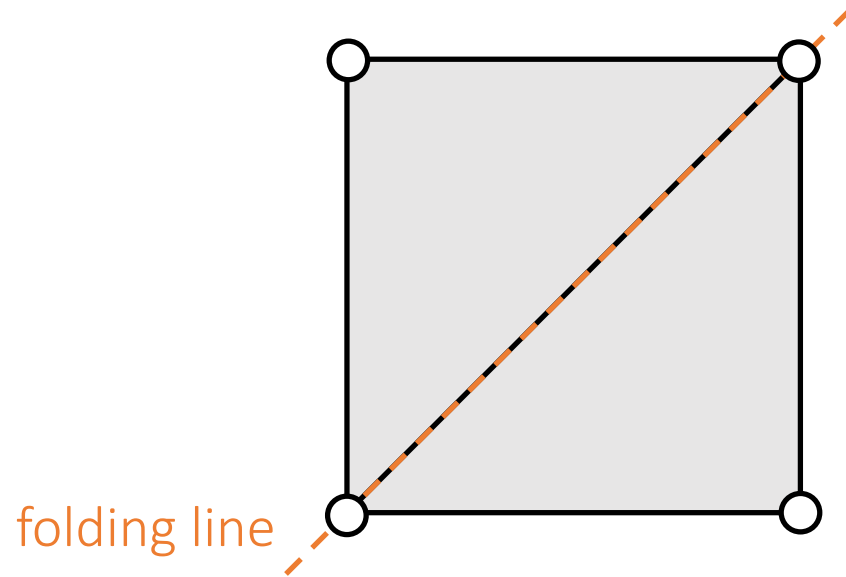
where  $\langle \cdot, \cdot \rangle_{\mathbb{R}^3}$  denotes the standard inner product of  $\mathbb{R}^3$ .

**Central observation.** The Laplace-Beltrami,  $\Delta$ , remains unchanged under isometric deformations of the surface—therefore, for inextensible surfaces,  $E_b(S)$  is *quadratic* in positions. Equation (2) together with the assumption of isometric deformation is henceforth called the *isometric bending model* (IBM). Our contribution is to present an analogous *discrete* IBM that is quadratic in positions. Its linear gradient and constant Hessian present an economic model for computing bending forces and their derivatives, enabling fast time-integration of cloth dynamics.

# The Locking Issue

So far we talked about the mass-spring model and other bending models, assuming *cloth planar deformation and cloth bending deformation are independent*.

Is it true? Think about a zero bending case. Can a simulator fold cloth freely?



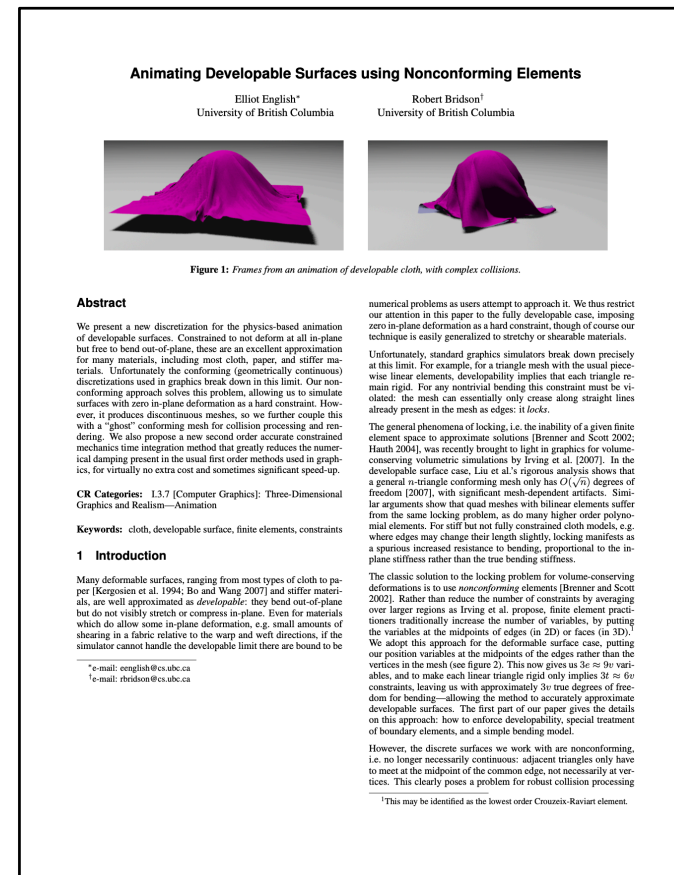
# The Locking Issue

The fundamental reason is due to a short of degrees of freedoms (DoFs).

For a manifold mesh, Euler's formula tells:  $\#edges = 3\#vertices - 3 - \#boundary\_edges$ .

So if edges are all hard constraints, the DoFs are only:  $3 + \#boundary\_edges$ .

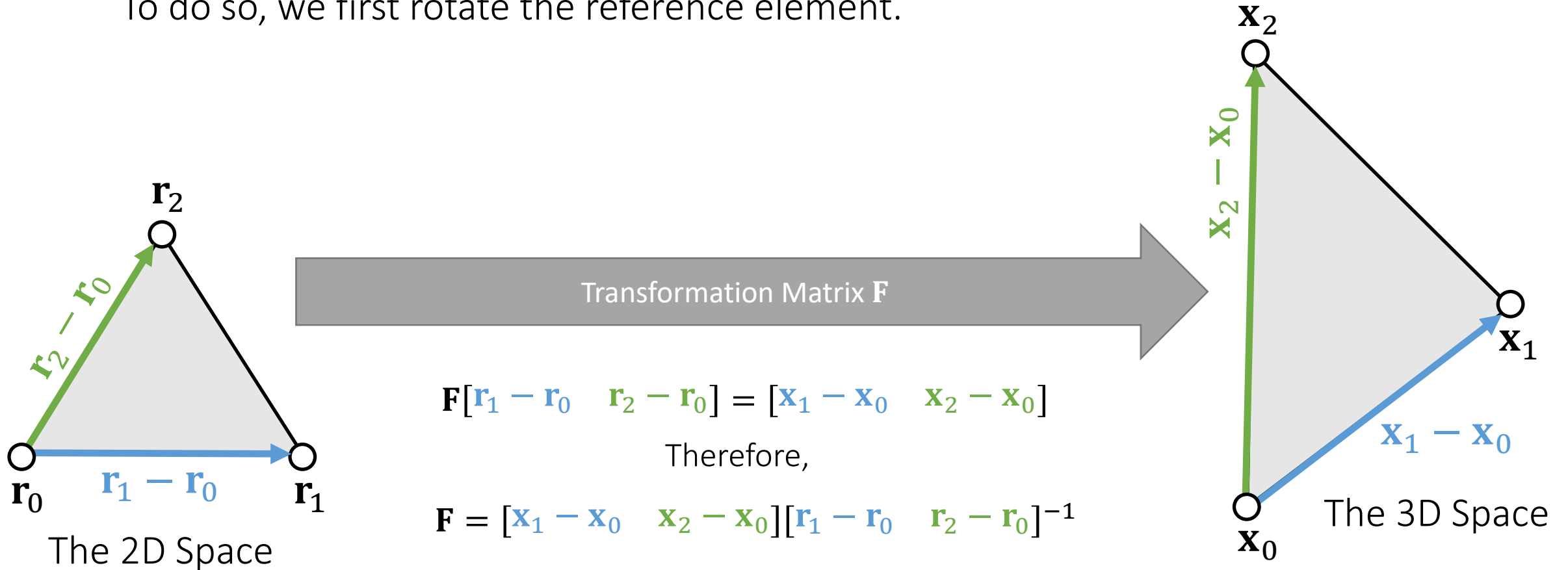
English and Bridson. 2008. *Animating Developable Surfaces Using Nonconforming Elements*. SIGGRAPH.  
(optional)



# Co-Rotational FEM

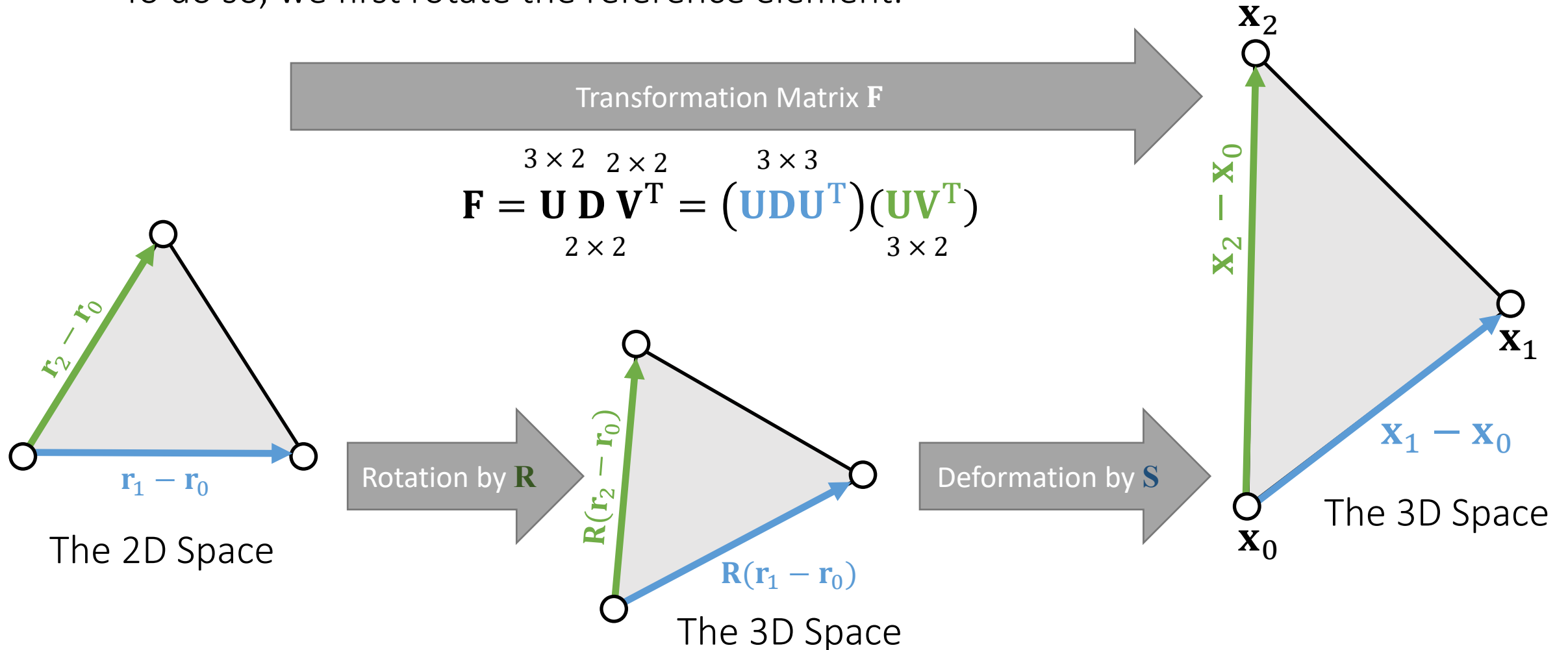
# Co-Rotational FEM

The basic idea is to define a quadratic energy based on the rotated reference element. To do so, we first rotate the reference element.



# Co-Rotational FEM

The basic idea is to define a quadratic energy based on the rotated reference element. To do so, we first rotate the reference element.



# Co-Rotational FEM

We can then define the quadratic energy as:

$$E(\mathbf{x}) = \frac{1}{2} \|\mathbf{x}_1 - \mathbf{x}_0 - \mathbf{R}(\mathbf{r}_1 - \mathbf{r}_0)\|^2 + \frac{1}{2} \|\mathbf{x}_2 - \mathbf{x}_0 - \mathbf{R}(\mathbf{r}_2 - \mathbf{r}_0)\|^2$$

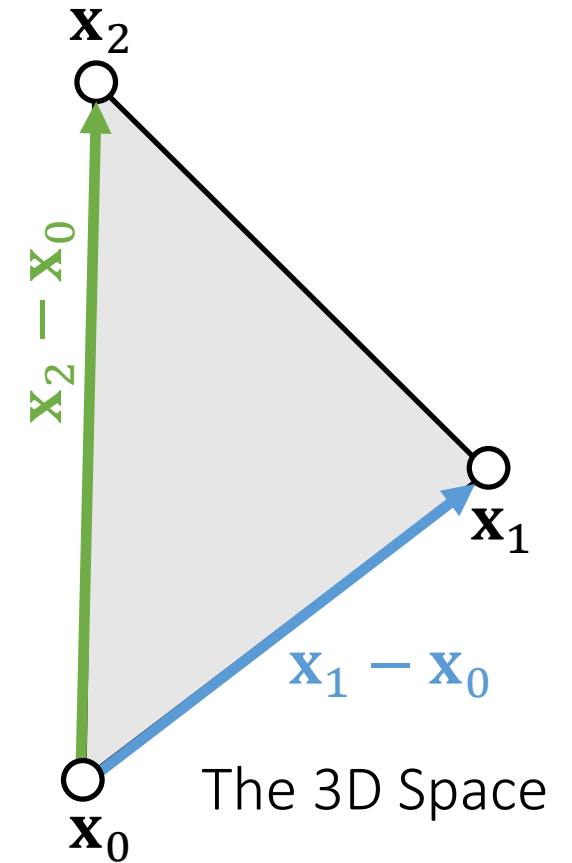
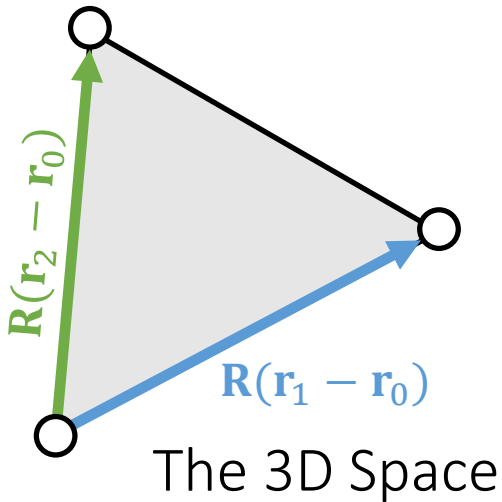
Assuming that  $\mathbf{R}$  is constant,

$$\mathbf{f}_1 = -\nabla_1 E(\mathbf{x}) = -\mathbf{x}_1 + \mathbf{x}_0 + \mathbf{R}(\mathbf{r}_1 - \mathbf{r}_0)$$

$$\mathbf{f}_2 = -\nabla_2 E(\mathbf{x}) = -\mathbf{x}_2 + \mathbf{x}_0 + \mathbf{R}(\mathbf{r}_2 - \mathbf{r}_0)$$

$$\mathbf{f}_0 = -\nabla_0 E(\mathbf{x}) = -\mathbf{f}_2 - \mathbf{f}_0$$

$$\mathbf{H} = \frac{\partial^2 E(\mathbf{x})}{\partial \mathbf{x}^2} = \begin{bmatrix} 2\mathbf{I} & -\mathbf{I} & -\mathbf{I} \\ -\mathbf{I} & \mathbf{I} & \\ -\mathbf{I} & & \mathbf{I} \end{bmatrix}$$





# After-Class Reading

Muller et al. 2005.

*Meshless Deformations Based on Shape Matching.*  
TOG (SIGGRAPH).

## Meshless Deformations Based on Shape Matching

Matthias Müller  
NovodeX/AGEIA & ETH Zürich

Bruno Heidelberger  
ETH Zürich

Matthias Teschner  
University of Freiburg

Markus Gross  
ETH Zürich

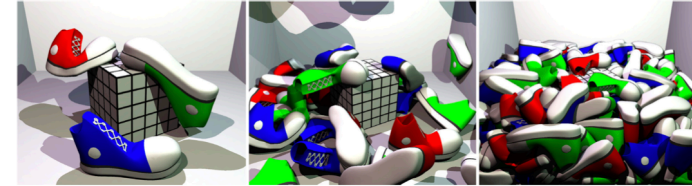


Figure 1: The presented technique is stable under all circumstances and allows to simulate hundreds of deformable objects in real-time.

### Abstract

We present a new approach for simulating deformable objects. The underlying model is geometrically motivated. It handles point-based objects and does not need connectivity information. The approach does not require any pre-processing, is simple to compute, and provides unconditionally stable dynamic simulations.

The main idea of our deformable model is to replace energies by geometric constraints and forces by distances of current positions to goal positions. These goal positions are determined via a generalized shape matching of an undeformed rest state with the current deformed state of the point cloud. Since points are always drawn towards well-defined locations, the overshooting problem of explicit integration schemes is eliminated. The versatility of the approach in terms of object representations that can be handled, the efficiency in terms of memory and computational complexity, and the unconditional stability of the dynamic simulation make the approach particularly interesting for games.

**CR Categories:** 1.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically Based Modeling; 1.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation and Virtual Reality

**Keywords:** deformable modeling, geometric deformation, shape matching, real-time simulation

### 1 Introduction

Since Terzopoulos' pioneering work on simulating deformable objects in the context of computer graphics [Terzopoulos et al. 1987],

many deformable models have been proposed. In general, these approaches focus on an accurate material representation, on stability aspects of the dynamic simulation and on versatility in terms of advanced object characteristics that can be handled, e. g. plastic deformation or fracturing.

Despite the long history of deformable modeling in computer graphics, research results have rarely been applied in computer games. Nowadays, deformable cloth models with simple geometries can be found in a few games, but in general, games are dominated by rigid bodies. Although rigid bodies can be linked with joints to represent articulated structures, there exist no practical solution which allows to simulate elastically deformable three-dimensional objects in a stable and efficient way. There are several reasons that prevent current deformable models from being used in interactive applications.

**Efficiency.** Existing deformable models based on complex material laws in conjunction with stable, implicit integration schemes are computationally expensive. Such approaches do not allow for interactive simulations of objects with a reasonable geometrical complexity. Further, these approaches might require a specific object representation and the algorithms can be hard to implement and debug. In contrast, interactive applications such as games constitute hard constraints on the computational efficiency of a deformable modeling approach. The approach is only allowed to use a small fraction of the available computing resources. Further, specific volumetric representations of deformable objects are often not available since the geometries are typically represented by surfaces only.

**Stability.** In interactive applications, the simulation of deformable objects needs to remain stable under all circumstances. While sophisticated approaches allow for stable numerical integration of velocities and positions, additional error sources such as degenerated geometries, physically incorrect states, or problematic situations with large object interpenetrations are not addressed by many approaches. A first contribution to this research area has been presented in [Irving et al. 2004], where large deformations and the inversion of elements in FE approaches can be handled in a robust way. However, this approach is not intended to be used in interactive applications.

# A Summary For the Day

- A mass-spring system
  - Planar springs against stretching/compression - replaceable by co-rotational FEM
  - Bending springs - replaceable by dihedral or quadratic bending
  - Regardless of the models, as long as we have  $E(\mathbf{x})$ , we can calculate force  $\mathbf{f}(\mathbf{x}) = -\nabla E(\mathbf{x})$  and Hessian  $\mathbf{H}(\mathbf{x}) = \partial^2 E(\mathbf{x}) / \partial \mathbf{x}^2$ . Forces and Hessians are stackable.
- Two integration approaches
  - Explicit integration, just need force. Instability
  - Implicit integration, as a nonlinear optimization problem
  - One way is to use Newton's method, which solves a linear system in every iteration:
$$\left( \frac{1}{\Delta t^2} \mathbf{M} + \mathbf{H}(\mathbf{x}^{(k)}) \right) \Delta \mathbf{x} = -\frac{1}{\Delta t^2} \mathbf{M}(\mathbf{x}^{(k)} - \mathbf{x}^{[0]} - \Delta t \mathbf{v}^{[0]}) + \mathbf{f}(\mathbf{x}^{(k)})$$
  - There are a variety of linear solvers (beyond the scope of this class).
  - Some simulators choose to solve only one Newton iteration, i.e., one linear system per time step.

Real-world fabrics are very complicated.

Models are making approximations only.

