GAMES103: Intro to Physics-Based Animation

Physics-Based Cloth Simulation

Huamin Wang

Nov 2021

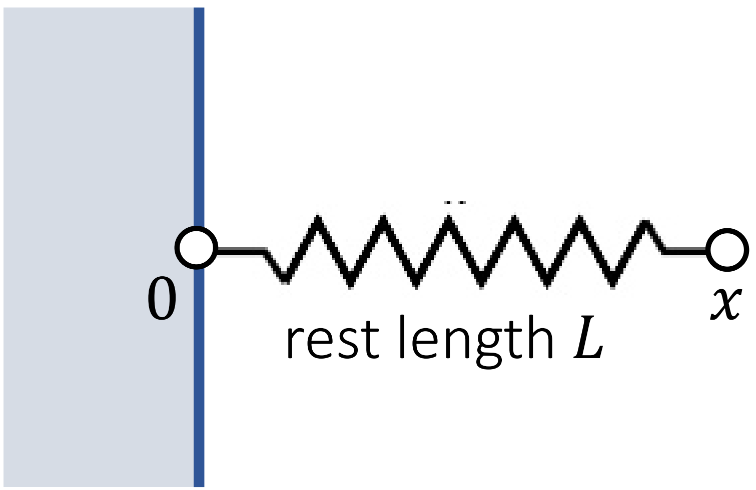
Topics for the Day

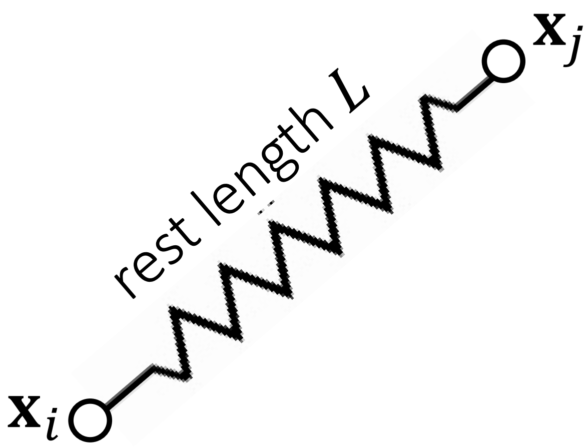
- A Mass-Spring System
  - Explicit Integration
  - Implicit Integration
- Bending and Locking Issues
- A Co-Rotational Method

A Mass Spring System

An Ideal Spring

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





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

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

$$f(x) = -\frac{dE}{dx} = -k(x - L)$$

spring stiffness

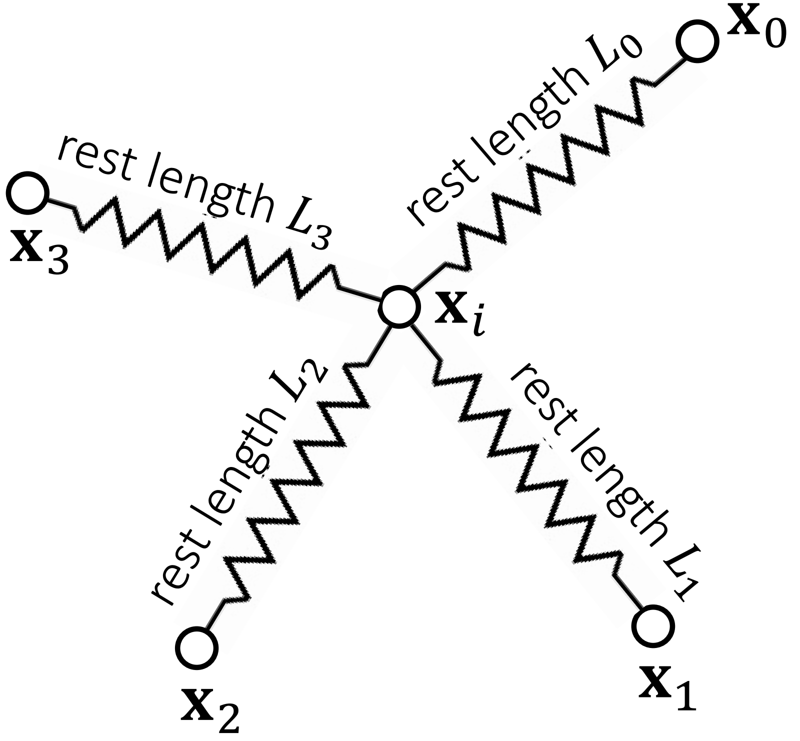
$$E(\mathbf{x}) = \frac{1}{2}k\left(\|\mathbf{x}_i - \mathbf{x}_j\| - L\right)^2$$

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

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

Multiple Springs

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

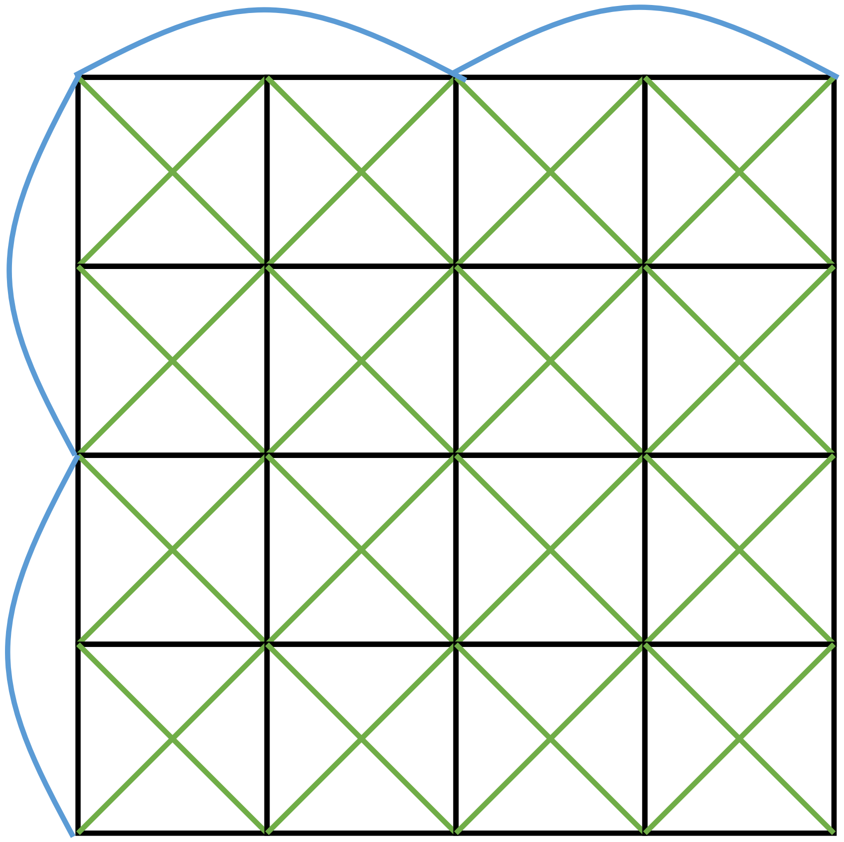


$$E = \sum_{e=0}^{3} E_e = \sum_{e=0}^{3} \left(\tfrac{1}{2} k (\|\mathbf{x}_i - \mathbf{x}_e\| - L_e)^2\right)$$

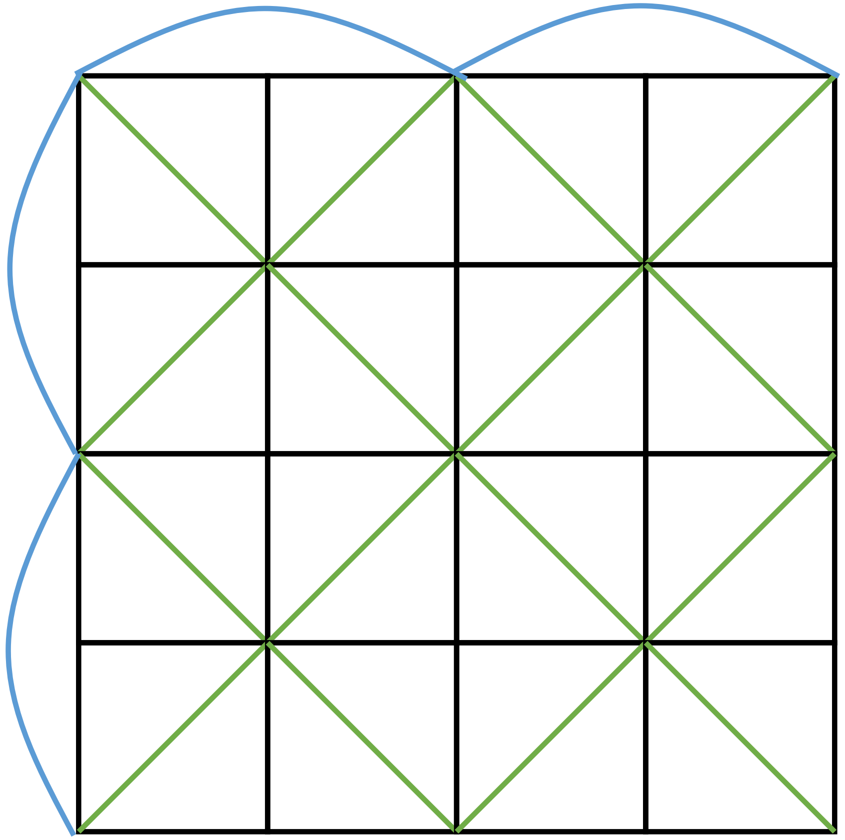
$$\mathbf{f}_i = -\nabla_i E = \sum_{e=0}^{3} \left(-k(\|\mathbf{x}_i - \mathbf{x}_e\| - L_e) \frac{\mathbf{x}_i - \mathbf{x}_e}{\|\mathbf{x}_i - \mathbf{x}_e\|}\right)$$

Structured Spring Networks

Horizontal and vertical — Diagonal — Bending



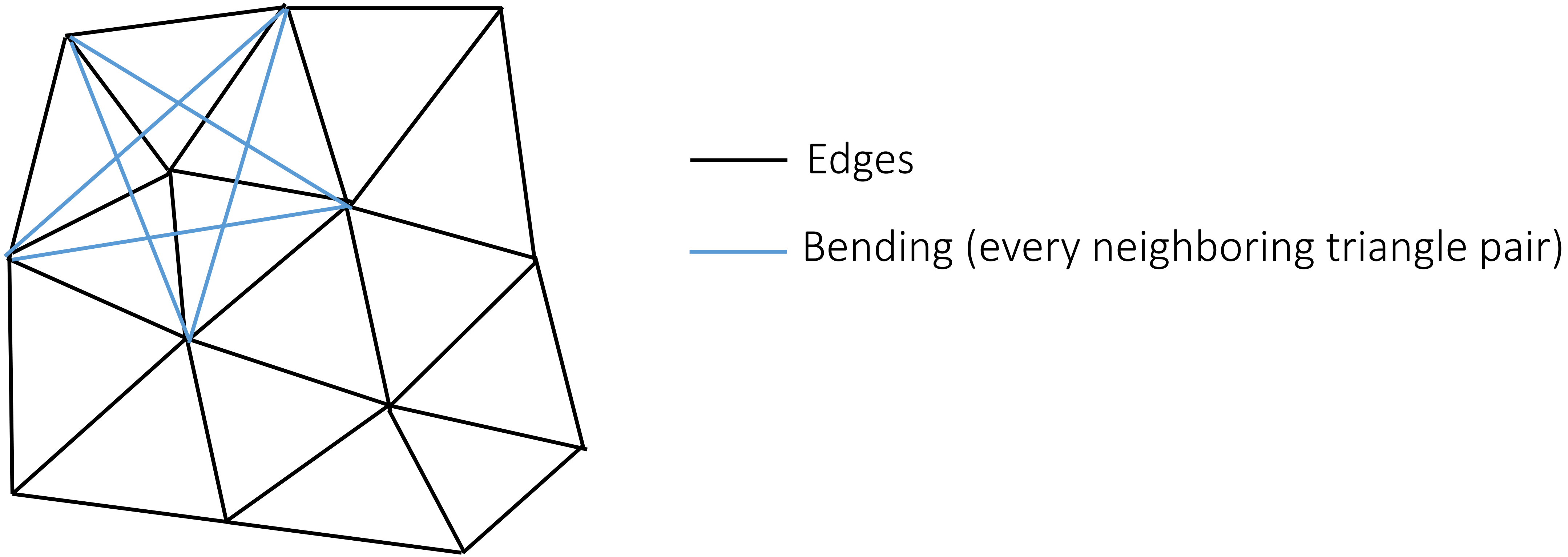
A structured network



A simplified network

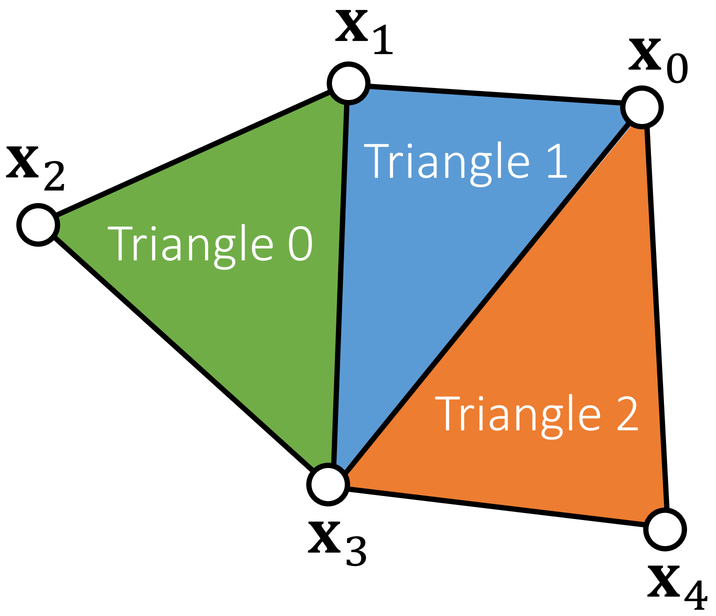
Unstructured Spring Networks

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



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)

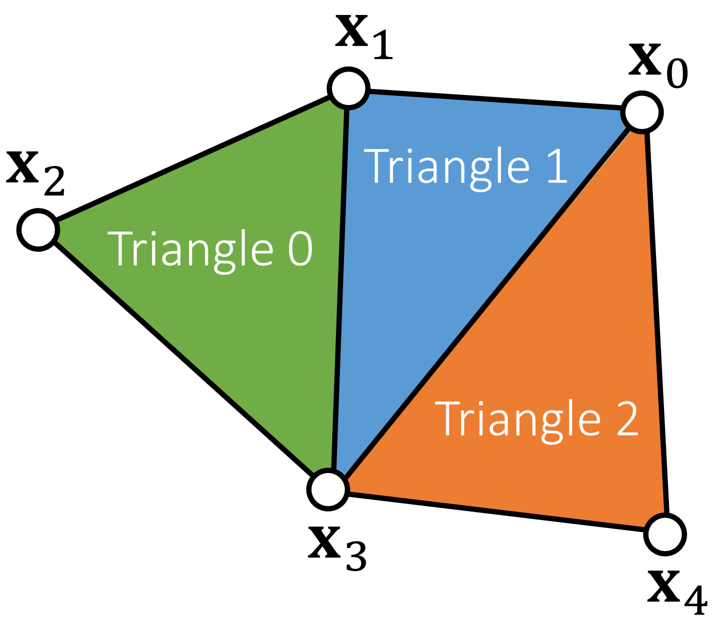
Triangle list: {1, 2, 3, 0, 1, 3, 0, 3, 4} (index triples)

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<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.

cloth_motion.cs

```
        stiffness   = 1000.0f;

        Mesh mesh = GetComponent<MeshFilter> ().mesh;
        int[]       triangles = mesh.triangles;
        Vector3[]   vertices = mesh.vertices;

        //Construct the original edge list
        int[] original_edge_list = new int[triangles.Length*2];
        for (int i=0; i<triangles.Length; i+=3)
        {
            original_edge_list[i*2+0]=triangles[i+0];
            original_edge_list[i*2+1]=triangles[i+1];
            original_edge_list[i*2+2]=triangles[i+1];
            original_edge_list[i*2+3]=triangles[i+2];
            original_edge_list[i*2+4]=triangles[i+2];
            original_edge_list[i*2+5]=triangles[i+0];
        }
        //Reorder the original edge list
        for (int i=0; i<original_edge_list.Length; i+=2)
            if(original_edge_list[i] > original_edge_list[i + 1])
                Swap(ref original_edge_list[i], ref original_edge_list[i+1]);
        //Sort the original edge list using quicksort
        Quick_Sort (ref original_edge_list, 0, original_edge_list.Length/2-1);

        int count = 0;
        for (int i=0; i<original_edge_list.Length; i+=2)
            if (i == 0 ||
                original_edge_list [i + 0] != original_edge_list [i - 2] ||
                original_edge_list [i + 1] != original_edge_list [i - 1])
                    count++;

        edge_list = new int[count * 2];
        int r_count = 0;
        for (int i=0; i<original_edge_list.Length; i+=2)
            if (i == 0 ||
                original_edge_list [i + 0] != original_edge_list [i - 2] ||
                original_edge_list [i + 1] != original_edge_list [i - 1])
            {
                edge_list[r_count*2+0]=original_edge_list [i + 0];
                edge_list[r_count*2+1]=original_edge_list [i + 1];
                r_count++;
            }


        L0 = new float[edge_list.Length/2];
        for (int e=0; e<edge_list.Length/2; e++)
        {
            int v0 = edge_list[e*2+0];
            int v1 = edge_list[e*2+1];
            L0[e]=(vertices[v0]-vertices[v1]).magnitude;
        }

        velocities = new Vector3[vertices.Length];
        for (int v=0; v<vertices.Length; v++)
            velocities [v] = new Vector3 (0, 0, 0);
```

Explicit Integration of A Mass-Spring System

Compute Spring Forces

For every edge e

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

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

$$L_e \longleftarrow L[e]$$

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

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

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

A Particle System

For every vertex

$$\mathbf{f}_i \longleftarrow \mathrm{Force}(\mathbf{x}_i, \mathbf{v}_i)$$

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

$$\mathbf{x}_i \longleftarrow \mathbf{x}_i + \Delta t\, \mathbf{v}_i$$

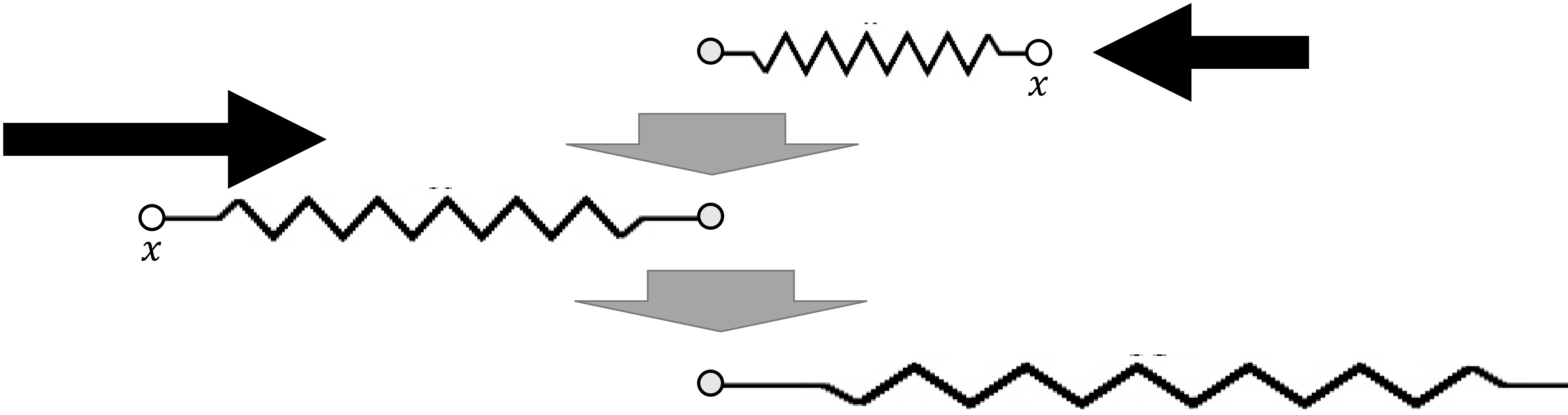
Done

E: Edge list L: Edge length list m_i: Mass of vertex *i*

Explicit Integration of A Mass-Spring System

Explicit integration suffers from numerical instability caused by <u>overshooting</u>, when the stiffness k and/or the time step Δt is too large.

A naive solution is to use a small Δt . 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:

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

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

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

This is because:

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

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

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

Newton-Raphson Method

The Newton-Raphson method, commonly known as Newton's method, solves the optimization problem: $x^{[1]} = \mathbf{argmin}\ F(x)$. ($F(x)$ is Lipschitz continuous.)

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

$$0 = F'(x) \approx F'\left(x^{(k)}\right) + F''\left(x^{(k)}\right)\left(x - x^{(k)}\right)$$

Newton's Method

Initialize $x^{(0)}$

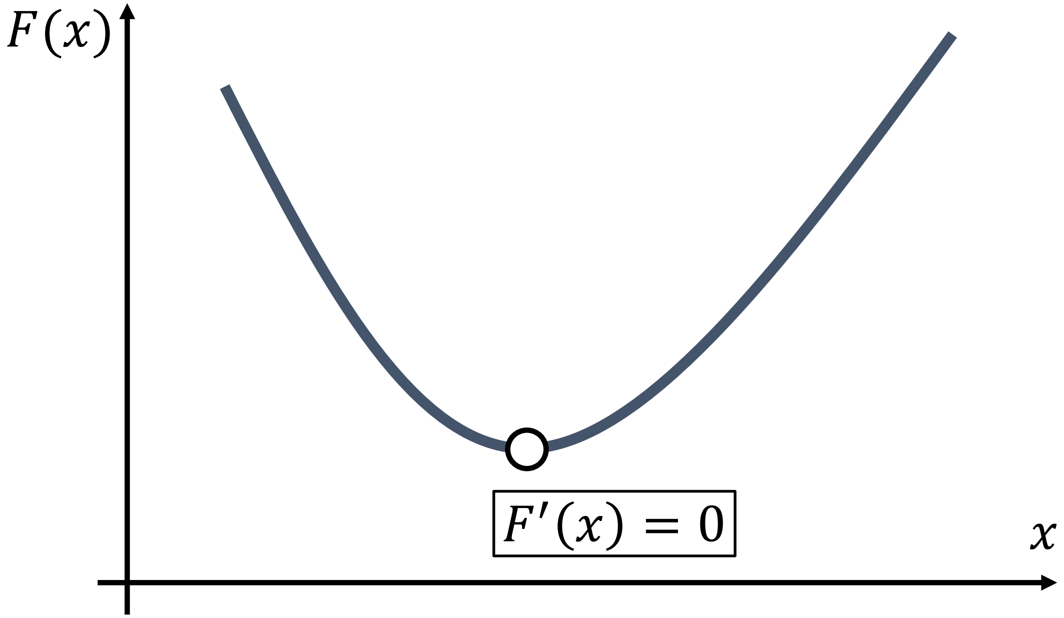
For $k = 0 \ldots K$

$\quad \Delta x \leftarrow -\left(F''\left(x^{(k)}\right)\right)^{-1} F'\left(x^{(k)}\right)$

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

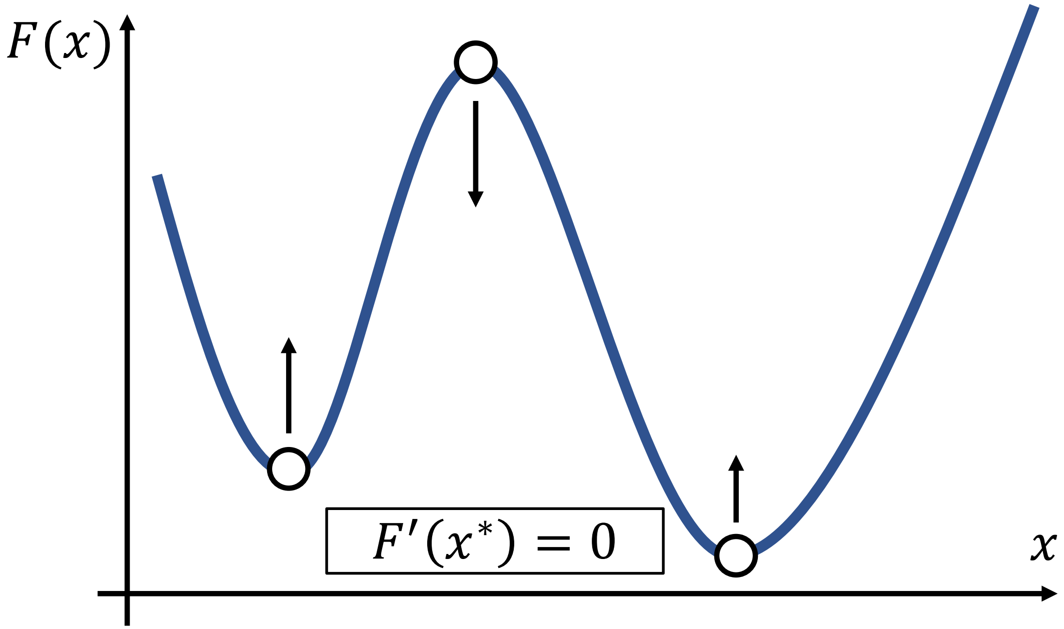
$\quad$ If $|\Delta x|$ is small then break

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



Newton-Raphson Method

Newton's method finds an extremum, but it can be a minimum or maximum.



- At a minimum x^*, $F''(x^*) > 0$.
- At a maximum x^*, $F''(x^*) < 0$.
- If $F''(x) > 0$ is everywhere, $F(x)$ has no maximum. => $F(x)$ has only one minimum.

Newton-Raphson Method

Now we can apply Newton's method to: $\mathbf{x}^{[1]} = \mathrm{argmin}\, F(\mathbf{x})$.

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

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

Newton's Method

Initialize $x^{(0)}$

For $k = 0 \dots K$

$$\Delta\mathbf{x} \leftarrow -\left(\frac{\partial F^2\left(\mathbf{x}^{(k)}\right)}{\partial \mathbf{x}^2}\right)^{-1} \nabla F\left(\mathbf{x}^{(k)}\right)$$

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

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

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

Simulation by Newton's Method

Specifically to simulation, we have: $\boxed{F(\mathbf{x}) = \frac{1}{2\Delta t^2}\left\|\mathbf{x} - \mathbf{x}^{[0]} - \Delta t\mathbf{v}^{[0]}\right\|_{\mathbf{M}}^2 + E(\mathbf{x})}$

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

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

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

For $k = 0 \dots K$

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

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

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

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

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

Spring Hessian

According to Lecture 2, Page 48,

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

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

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

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

Spring Hessian

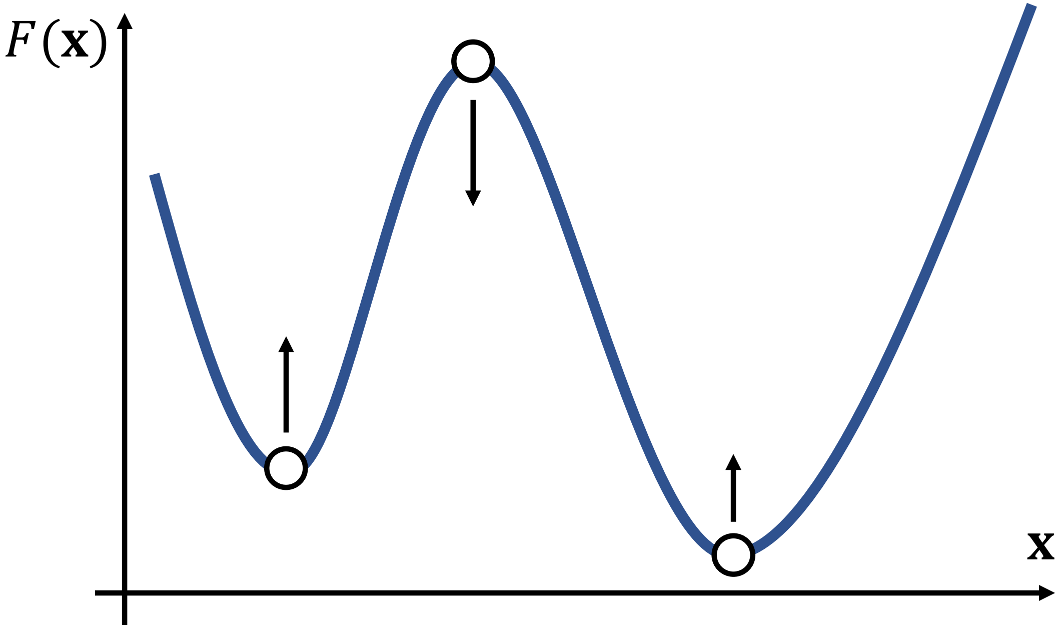
When a spring is stretched, $\mathbf{H}_e$ is s.p.d.; but when it's compressed, $\mathbf{H}_e$ may not be s.p.d.

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.

$$\mathbf{A} = \frac{1}{\Delta t^2}\mathbf{M} + \mathbf{H}(\mathbf{x}) = \underbrace{\boxed{\frac{1}{\Delta t^2}\mathbf{M}}}_{\text{s.p.d.}} + \sum_{e=\{i,j\}} \underbrace{\boxed{\begin{bmatrix} \ddots & \vdots & \vdots & ⋰ \\ & \mathbf{H}_e & -\mathbf{H}_e & \\ & -\mathbf{H}_e & \mathbf{H}_e & \\ ⋰ & & & \ddots \end{bmatrix}}}_{\text{May not be s.p.d.}}$$

Positive Definiteness of Hessian

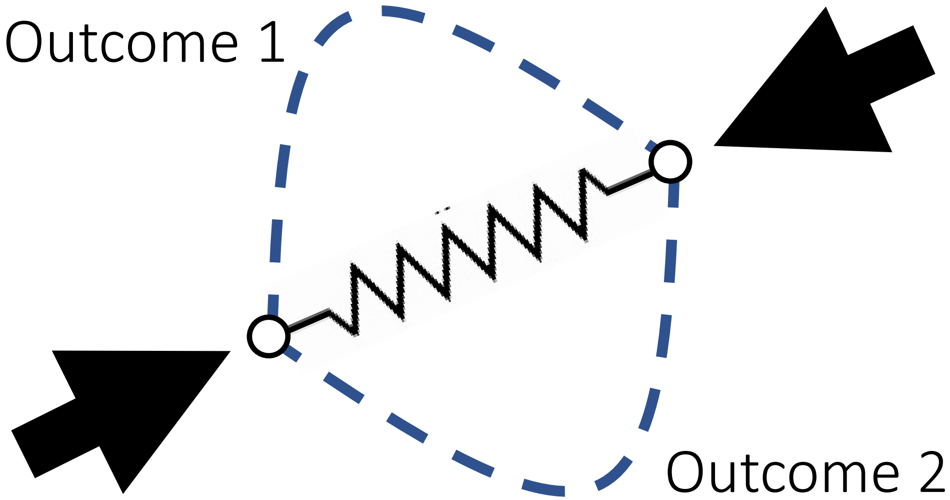


- As before, if $\partial^2 F/\partial \mathbf{x}^2$ is positive definite everywhere, $F(\mathbf{x})$ has no maximum but only one minimum.

- This is a sufficient but not necessary condition.

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 $\|\mathbf{x}_{ij}\| < L_e$:

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

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

The Jacobi Method

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

The Jacobi Method

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

For $k = 0 \dots K$

$\quad \mathbf{r} \longleftarrow \mathbf{b} - \mathbf{A}\Delta\mathbf{x}$ ← - - - Residual error

$\quad$ If $\|\mathbf{r}\| < \varepsilon$ break ← - - - Convergence condition ε

$\quad \Delta\mathbf{x} \longleftarrow \Delta\mathbf{x} + \alpha\mathbf{D}^{-1}\mathbf{r}$ ← - - - 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 α allows the method to converget 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 $\mathbf{A}$
  - Mostly suitable on CPUs
- Iterative Solvers
  - Expensive to solve exactly, but controllable
  - Convergence restriction on $\mathbf{A}$, 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

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

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

For $k = 0 \ldots K$

- $\mathbf{r} \longleftarrow \mathbf{b} - \mathbf{A}\Delta\mathbf{x}$
- If $\|\mathbf{r}\| < \varepsilon$ break
- If $k = 0$ $\quad \omega = 1$
- Else If $k = 1$ $\quad \omega = 2/(2 - \rho^2)$
- Else $\quad \omega = 4/(4 - \rho^2\omega)$
- $\text{old\_}\Delta\mathbf{x} \longleftarrow \Delta\mathbf{x}$
- $\Delta\mathbf{x} \longleftarrow \Delta\mathbf{x} + \alpha\mathbf{D}^{-1}\mathbf{r}$
- $\Delta\mathbf{x} \longleftarrow \omega\Delta\mathbf{x} + (1 - \omega)\text{last\_}\Delta\mathbf{x}$
- $\text{last\_}\Delta\mathbf{x} \longleftarrow \text{old\_}\Delta\mathbf{x}$

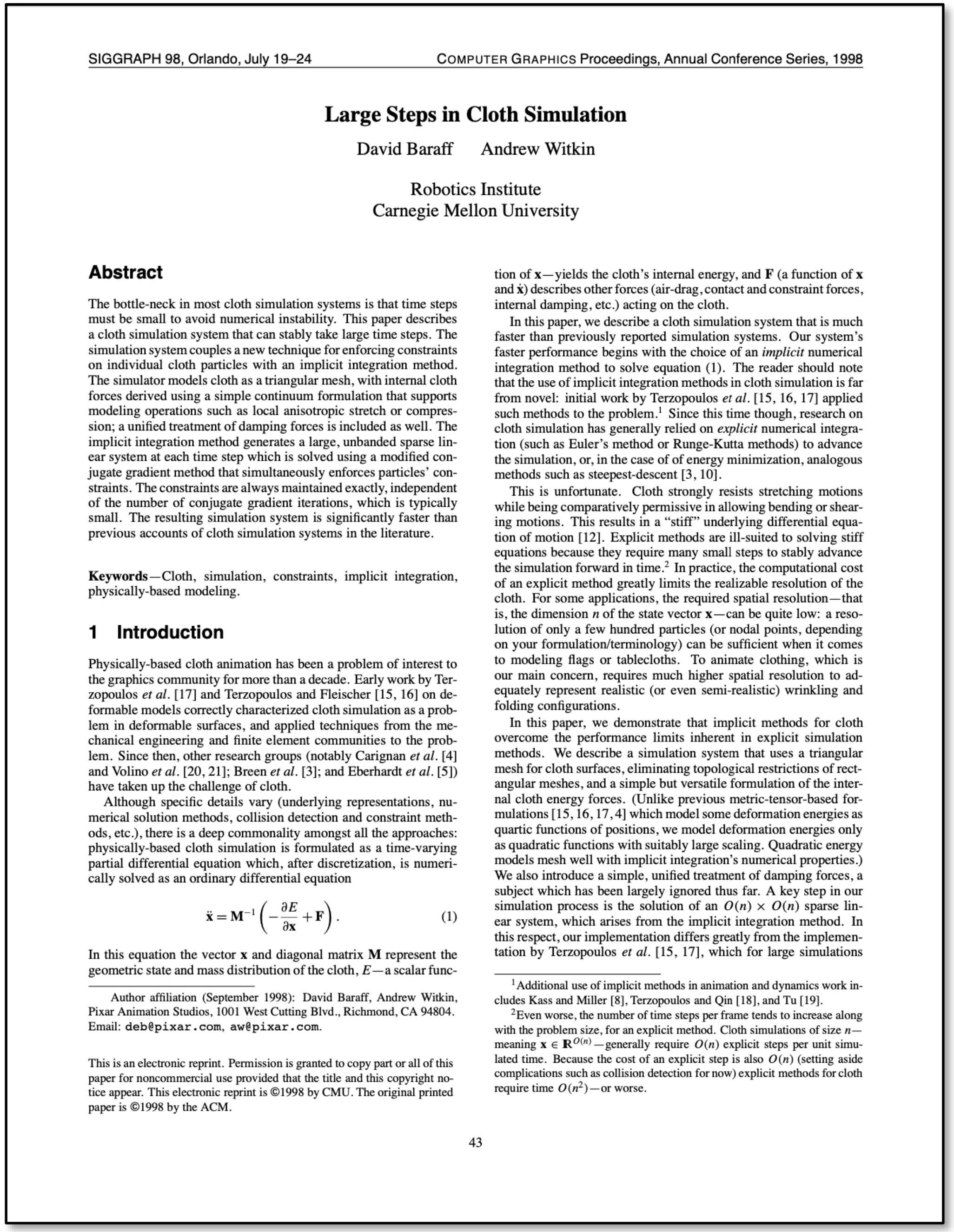
ρ $(\rho < 1)$ is the estimated spectral radius of the iterative matrix.

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

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

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 function 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.[1] 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 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.[2] 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

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.

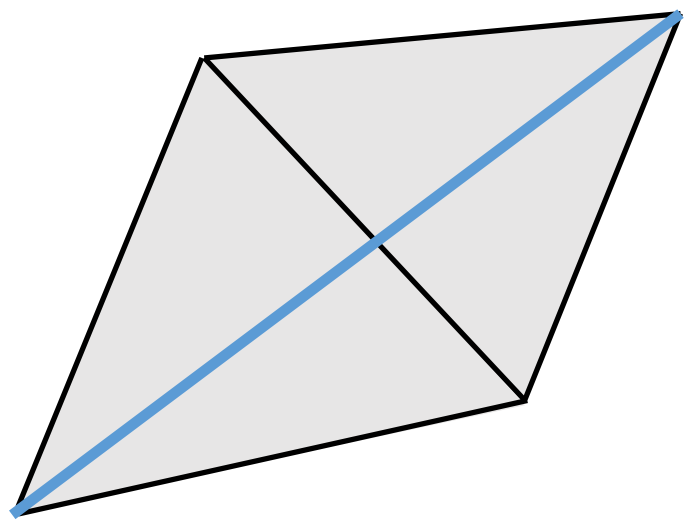
[1] Additional use of implicit methods in animation and dynamics work includes Kass and Miller [8], Terzopoulos and Qin [18], and Tu [19].

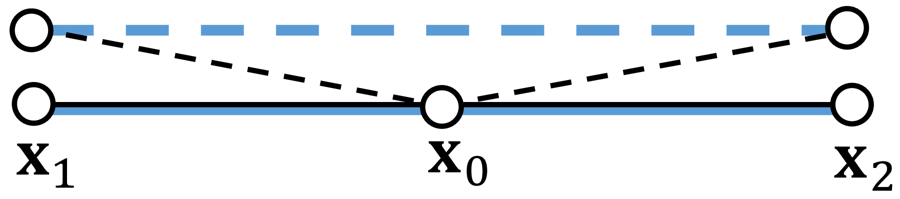
[2] Even worse, the number of time steps per frame tends to increase along with the problem size, for an explicit method. Cloth simulations of size n—meaning $\mathbf{x} \in \mathbb{R}^{O(n)}$—generally require $O(n)$ explicit steps per unit simulated time. Because the cost of an explicit step is also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

Bending and Locking Issues

The Bending Spring Issue

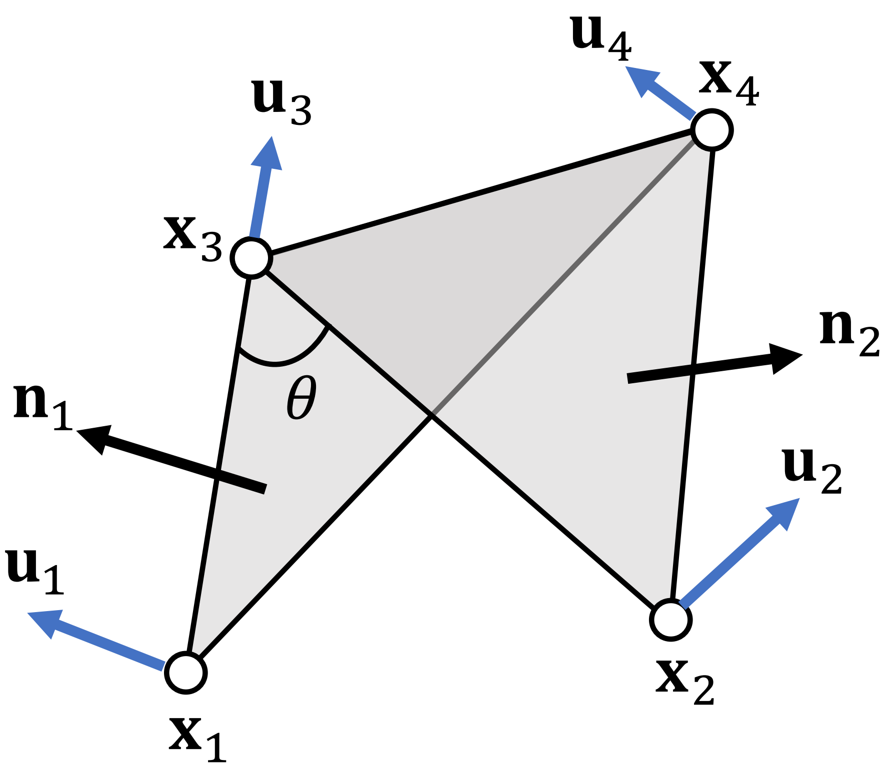
A bending spring offers little resistance when cloth is nearly planar, since its length barely changes.





A Dihedral Angle Model

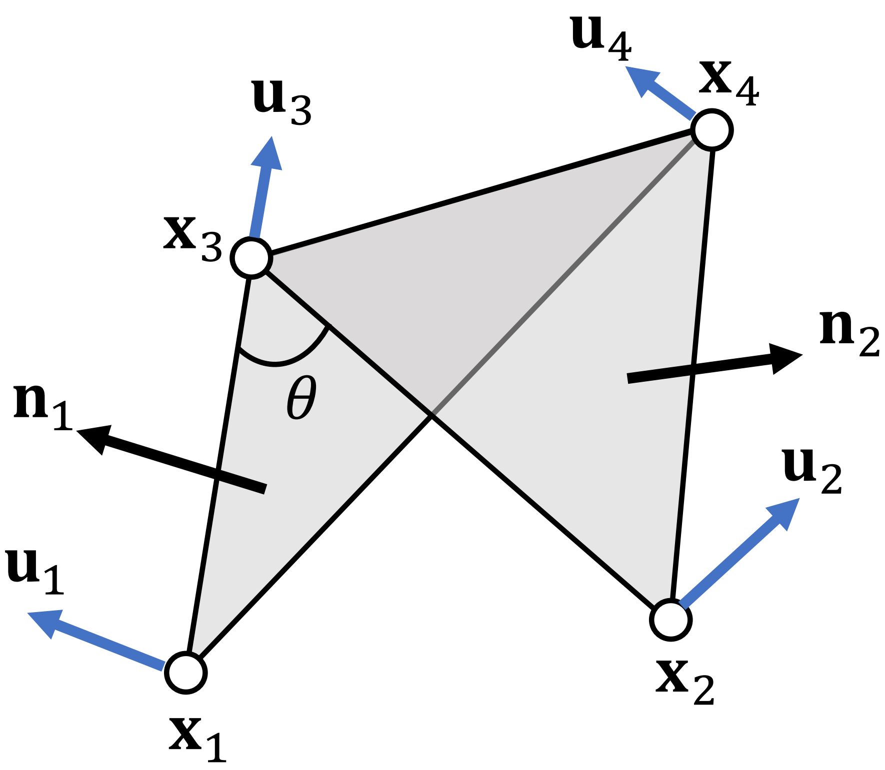
A dihedral angle model defines bending forces as a function of θ: $\mathbf{f}_i = f(\theta)\mathbf{u}_i$.



- 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, $\mathbf{u}_1 + \mathbf{u}_2 + \mathbf{u}_3 + \mathbf{u}_4 = \mathbf{0}$, which means $\mathbf{u}_3$ and $\mathbf{u}_3$ 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 θ: $\mathbf{f}_i = f(\theta)\mathbf{u}_i$.



Conclusion:

$$\mathbf{u}_1 = \|\mathbf{E}\| \frac{\mathbf{N}_1}{\|\mathbf{N}_1\|^2} \qquad \mathbf{u}_2 = \|\mathbf{E}\| \frac{\mathbf{N}_2}{\|\mathbf{N}_2\|^2}$$

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

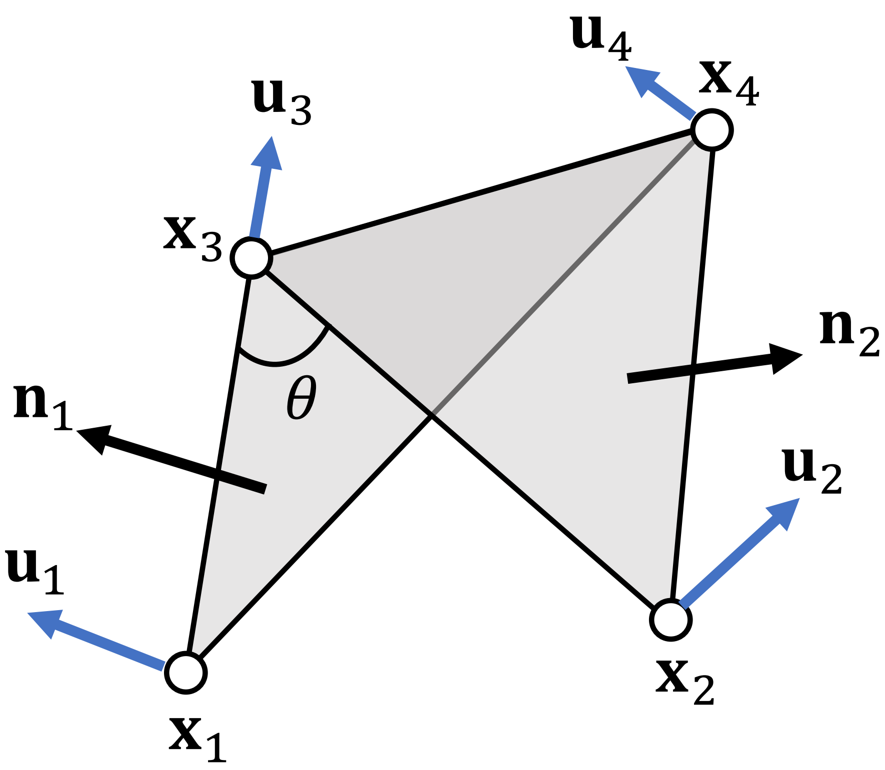
$$\mathbf{u}_4 = -\frac{(\mathbf{x}_1 - \mathbf{x}_3)\cdot\mathbf{E}}{\|\mathbf{E}\|}\frac{\mathbf{N}_1}{\|\mathbf{N}_1\|^2} - \frac{(\mathbf{x}_2 - \mathbf{x}_3)\cdot\mathbf{E}}{\|\mathbf{E}\|}\frac{\mathbf{N}_2}{\|\mathbf{N}_2\|^2}$$

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

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

A Dihedral Angle Model

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



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.

Eurographics/SIGGRAPH Symposium on Computer Animation (2003)
D. Breen, M. Lin (Editors)

Simulation of Clothing with Folds and Wrinkles

R. Bridson,[1] S. Marino[2] and R. Fedkiw[3]

[1] Stanford University, rbridson@stanford.edu
[2] Industrial Light + Magic, smarino@ilm.com
[3] Stanford University, Industrial Light + Magic, fedkiw@cs.stanford.edu

Abstract

Clothing is a fundamental part of a character's persona, a key storytelling tool used to convey an intended impression to the audience. Draping, folding, wrinkling, stretching, etc. all convey meaning, and thus each is carefully controlled when filming live actors. When making films with computer simulated cloth, these subtle but important elements must be captured. In this paper we present several methods essential to matching the behavior and look of clothing worn by digital stand-ins to their real world counterparts. Novel contributions include a mixed explicit/implicit time integration scheme, a physically correct bending model with (potentially) nonzero rest angles for pre-shaping wrinkles, an interface forecasting technique that promotes the development of detail in contact regions, a post-processing method for treating cloth-character collisions that preserves folds and wrinkles, and a dynamic constraint mechanism that helps to control large scale folding. The common goal of all these techniques is to produce a cloth simulation with many folds and wrinkles improving the realism.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Animation, I.3.5 [Computer Graphics]: Physically based modeling

1. Introduction

Clothing is an integral part of both live action and computer generated characters. The relationship between tailoring, body composition and material selection are equally important for either medium as these qualities make each garment unique to every individual, real or fictional. Clothing folds, wrinkles and stretches to conform to its wearer, it sticks to itself and other pieces of clothing, snags, etc. The appearance of a piece of clothing comes primarily from response to these conditions, and thus it is essential for clothing created using computer graphics to model them.

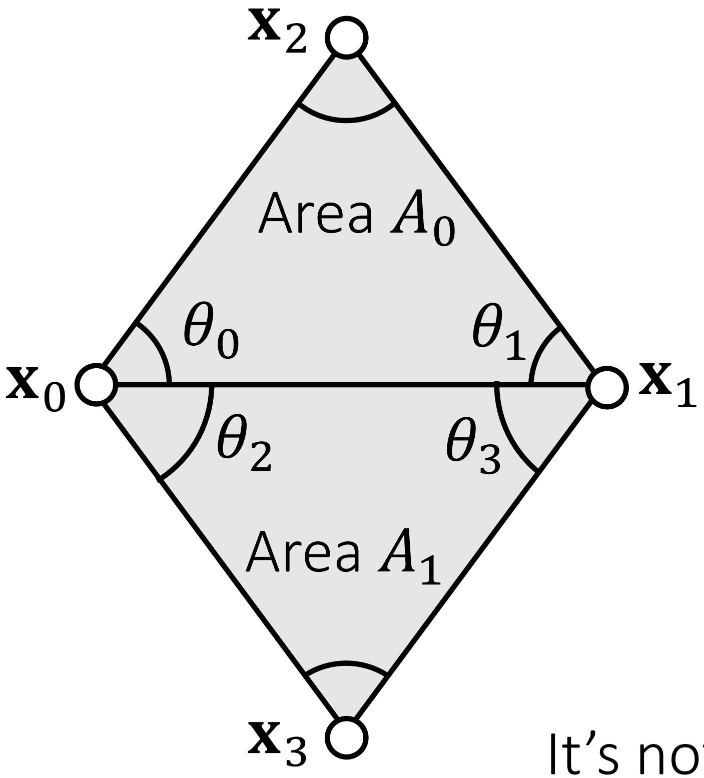
A judicious choice of cloth model is imperative both to obtain the desired look and feel of the cloth, and to obtain simulation results in a reasonable amount of time. We present a mixed explicit/implicit time integration scheme that combines the flexibility of explicit methods for handling nonlinearities (such as the biphasic nature of cloth) with the speed of implicit methods. One of the keys to obtaining high levels of visual detail, i.e. dynamic folds and wrinkles, is to have a good model for bending. One of our novel contributions is a derivation of the *only* physically correct family of bending forces that act between pairs of triangles. Furthermore, our model is capable of retaining and restoring artist-sculpted features of a garment by using nonzero rest angles.

Once equipped with a reasonable dynamic model of our character's clothing, the cloth must interact with its environment. This means collision detection and modeling for both self-collision and cloth-object or cloth-character collision. There are a number of techniques for treating self-collision, either untying the cloth[60,61,6] or stopping all collisions before they happen[47,30,9]. All of these papers illustrated that it is important to have a robust self-collision strategy in order to model wrinkling and folding. For cloth-object collision, we use a level set approach modeling the (possibly deforming) objects with a signed distance function defined on a grid[41]. A number of authors have used implicit surfaces for volumetric collisions, and we illustrate that this approach flattens out the cloth—removing visual detail—by moving it to a smooth surface (the zero isocontour). We propose a new dynamic interface forecasting technique and a new post-processing technique which help to alleviate this difficulty.

At this point, one can carry out physically plausible cloth-ing simulation, but this is nevertheless insufficient to model compelling real world clothing. Not only do many of the

A Quadratic Bending Model

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



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

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

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

$\mathbf{I}$ is 3-by-3 identity.

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

Pros and Cons of The Quadratic Bending Model

- Easy to implement:

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

- 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+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.

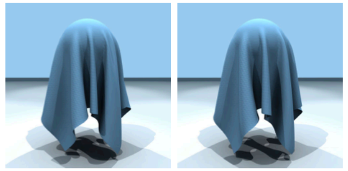


Figure 1: *Final rest state of a cloth draped over a sphere, for* (left) *the proposed isometric bending model and* (right) *the widely-adopted nonlinear hinge model.*

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

$$E_b(S) = \frac{1}{2}\int_S H^2 \mathrm{d}A\,, \qquad (1)$$

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

$$E_b(S) = \frac{1}{2}\int_S \langle \Delta\mathbf{x}, \Delta\mathbf{x}\rangle_{\mathbb{R}^3} \mathrm{d}A, \qquad (2)$$

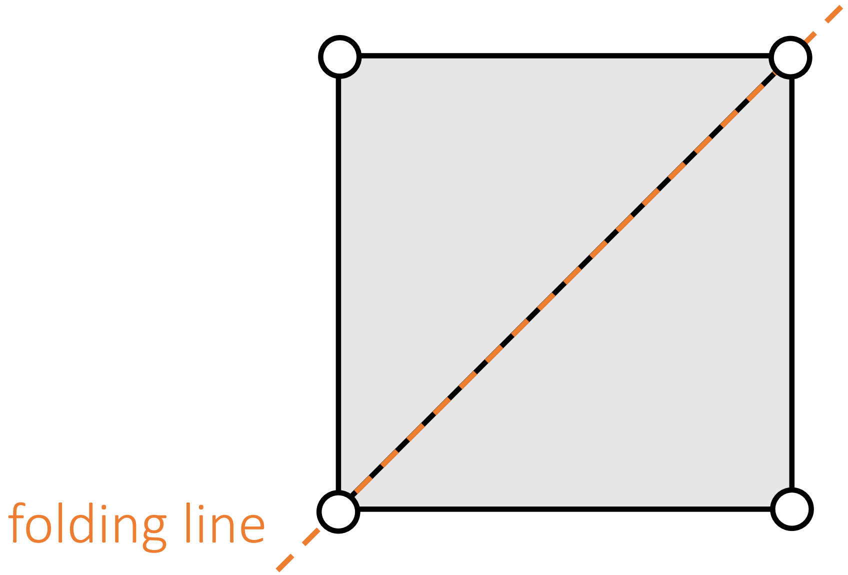
where $\langle\cdot,\cdot\rangle_{\mathbb{R}^3}$ denotes the standard inner product of $\mathbb{R}^3$.

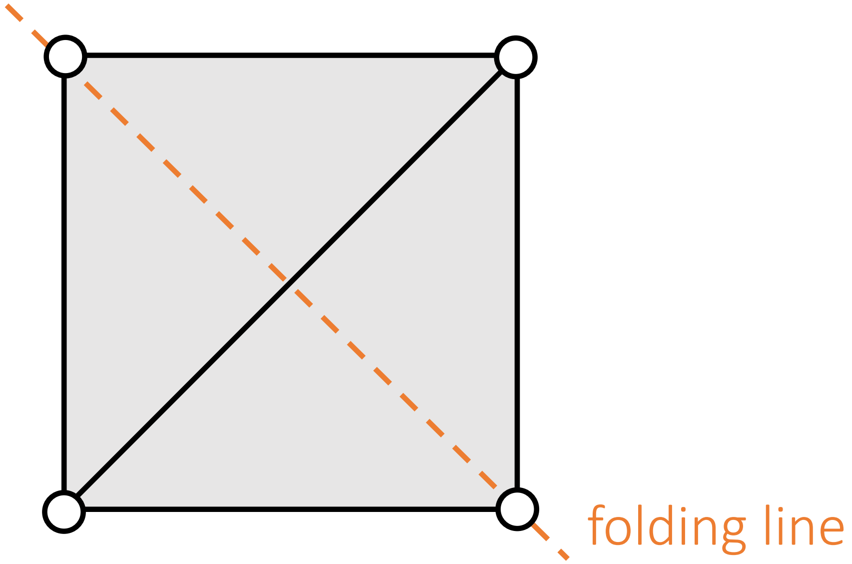
Central observation. The Laplace-Beltrami, Δ, remains unchanged under isometric deformations of the surface—therefore, for inextensible surfaces, $E_b(S)$ is *quadratic* in positions. Equation (2) together with the assumption of isometric deformation is henceforth called the *isometric bending model* (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?





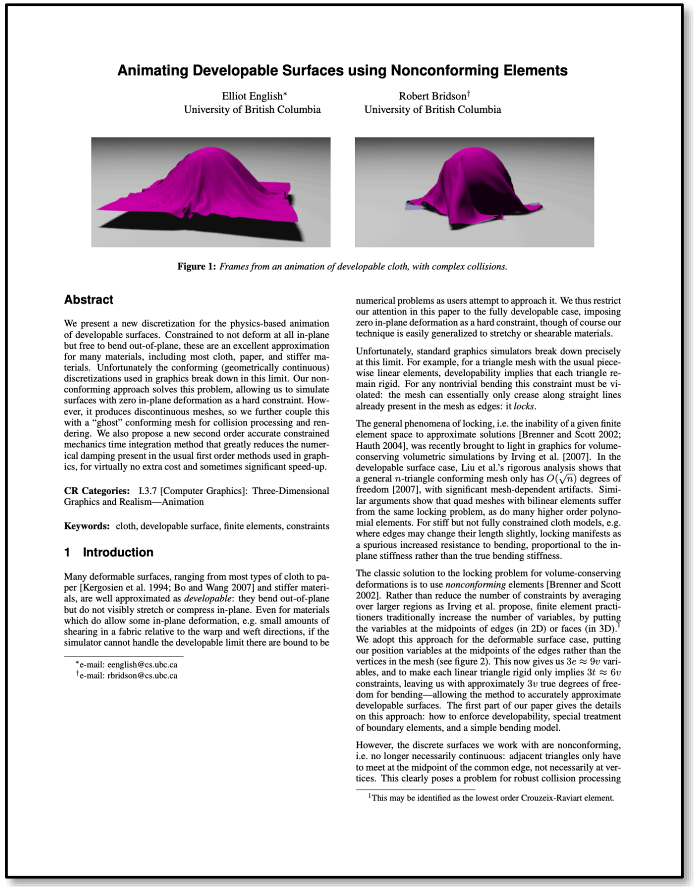
The Locking Issue

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

For a manifold mesh, Euler's formula says: #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)



Animating Developable Surfaces using Nonconforming Elements

Elliot English[*]
University of British Columbia

Robert Bridson[†]
University of British Columbia

Figure 1: *Frames from an animation of developable cloth, with complex collisions.*

Abstract

We present a new discretization for the physics-based animation of developable surfaces. Constrained to not deform at all in-plane but free to bend out-of-plane, these are an excellent approximation for many materials, including most cloth, paper, and stiffer materials. Unfortunately the conforming (geometrically continuous) discretizations used in graphics break down in this limit. Our nonconforming approach solves this problem, allowing us to simulate surfaces with zero in-plane deformation as a hard constraint. However, it produces discontinuous meshes, so we further couple this with a "ghost" conforming mesh for collision processing and rendering. We also propose a new second order accurate constrained mechanics time integration method that greatly reduces the numerical damping present in the usual first order methods used in graphics, for virtually no extra cost and sometimes significant speed-up.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: cloth, developable surface, finite elements, constraints

1 Introduction

Many deformable surfaces, ranging from most types of cloth to paper [Kergosien et al. 1994; Bo and Wang 2007] and stiffer materials, are well approximated as *developable*: they bend out-of-plane but do not visibly stretch or compress in-plane. Even for materials which do allow some in-plane deformation, e.g. small amounts of shearing in a fabric relative to the warp and weft directions, if the simulator cannot handle the developable limit there are bound to be numerical problems as users attempt to approach it. We thus restrict our attention in this paper to the fully developable case, imposing zero in-plane deformation as a hard constraint, though of course our technique is easily generalized to stretchy or shearable materials.

Unfortunately, standard graphics simulators break down precisely at this limit. For example, for a triangle mesh with the usual piecewise linear elements, developability implies that each triangle remain rigid. For any nontrivial bending this constraint must be violated: the mesh can essentially only crease along straight lines already present in the mesh as edges: it *locks*.

The general phenomena of locking, i.e. the inability of a given finite element space to approximate solutions [Brenner and Scott 2002; Hauth 2004], was recently brought to light in graphics for volume-conserving volumetric simulations by Irving et al. [2007]. In the developable surface case, Liu et al.'s rigorous analysis shows that a general n-triangle conforming mesh only has $O(\sqrt{n})$ degrees of freedom [2007], with significant mesh-dependent artifacts. Similar arguments show that quad meshes with bilinear elements suffer from the same locking problem, as do many higher order polynomial elements. For stiff but not fully constrained cloth models, e.g. where edges may change their length slightly, locking manifests as a spurious increased resistance to bending, proportional to the in-plane stiffness rather than the true bending stiffness.

The classic solution to the locking problem for volume-conserving deformations is to use *nonconforming* elements [Brenner and Scott 2002]. Rather than reduce the number of constraints by averaging over larger regions as Irving et al. propose, finite element practitioners traditionally increase the number of variables, by putting the variables at the midpoints of edges (in 2D) or faces (in 3D).[1] We adopt this approach for the deformable surface case, putting our position variables at the midpoints of the edges rather than the vertices in the mesh (see figure 2). This now gives us $3e \approx 9v$ variables, and to make each linear triangle rigid only implies $3t \approx 6v$ constraints, leaving us with approximately $3v$ true degrees of freedom for bending—allowing the method to accurately approximate developable surfaces. The first part of our paper gives the details on this approach: how to enforce developability, special treatment of boundary elements, and a simple bending model.

However, the discrete surfaces we work with are nonconforming, i.e. no longer necessarily continuous: adjacent triangles only have to meet at the midpoint of the common edge, not necessarily at vertices. This clearly poses a problem for robust collision processing

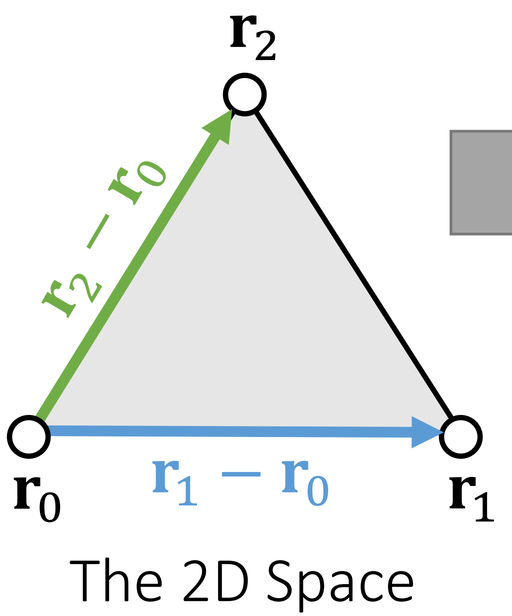
*e-mail: eenglish@cs.ubc.ca
†e-mail: rbridson@cs.ubc.ca

[1] This may be identified as the lowest order Crouzeix-Raviart element.

A Co-Rotational Method

A Co-Rotational Method

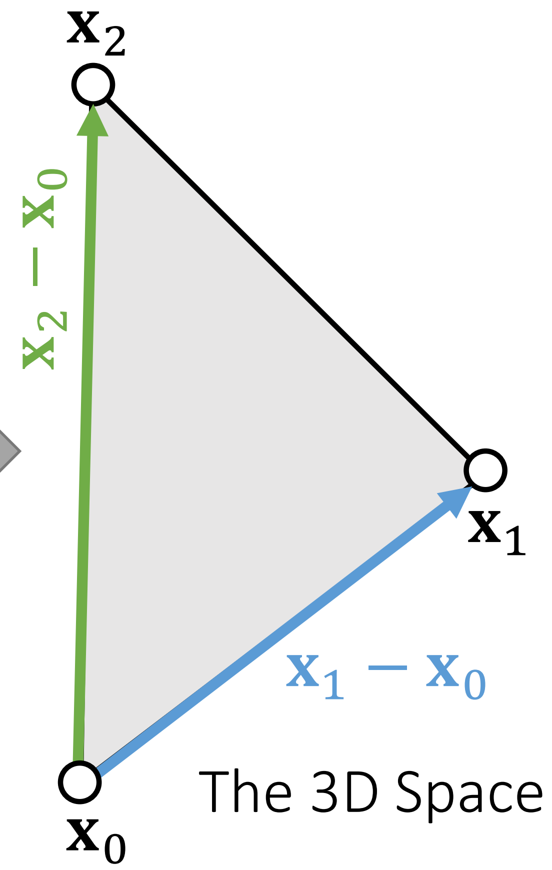
The basic idea is to define a quadratic energy based on the rotated reference element. To do so, we first rotate the reference element.



$$\mathbf{F}[\mathbf{r}_1 - \mathbf{r}_0 \quad \mathbf{r}_2 - \mathbf{r}_0] = [\mathbf{x}_1 - \mathbf{x}_0 \quad \mathbf{x}_2 - \mathbf{x}_0]$$

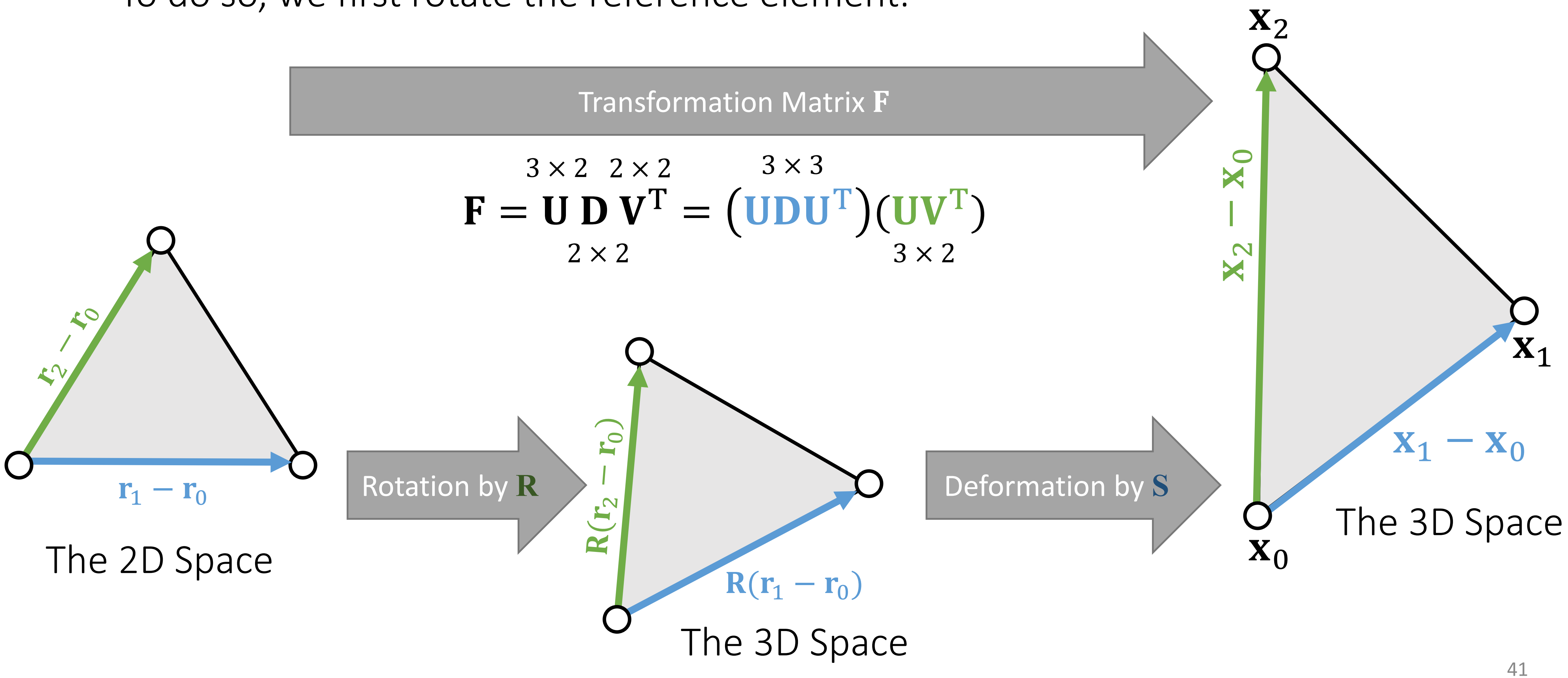
Therefore,

$$\mathbf{F} = [\mathbf{x}_1 - \mathbf{x}_0 \quad \mathbf{x}_2 - \mathbf{x}_0][\mathbf{r}_1 - \mathbf{r}_0 \quad \mathbf{r}_2 - \mathbf{r}_0]^{-1}$$



A Co-Rotational Method

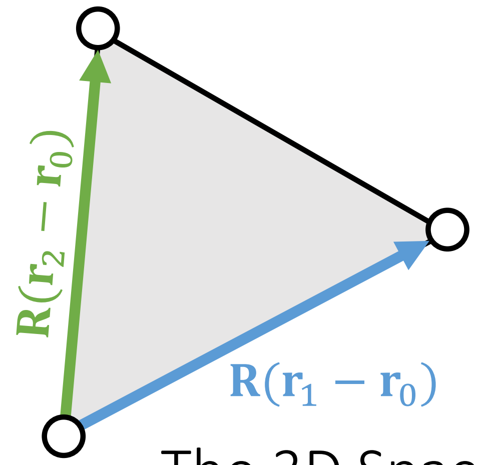
The basic idea is to define a quadratic energy based on the rotated reference element. To do so, we first rotate the reference element.



A Co-Rotational Method

We can then define the quadratic energy as:

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



The 3D Space

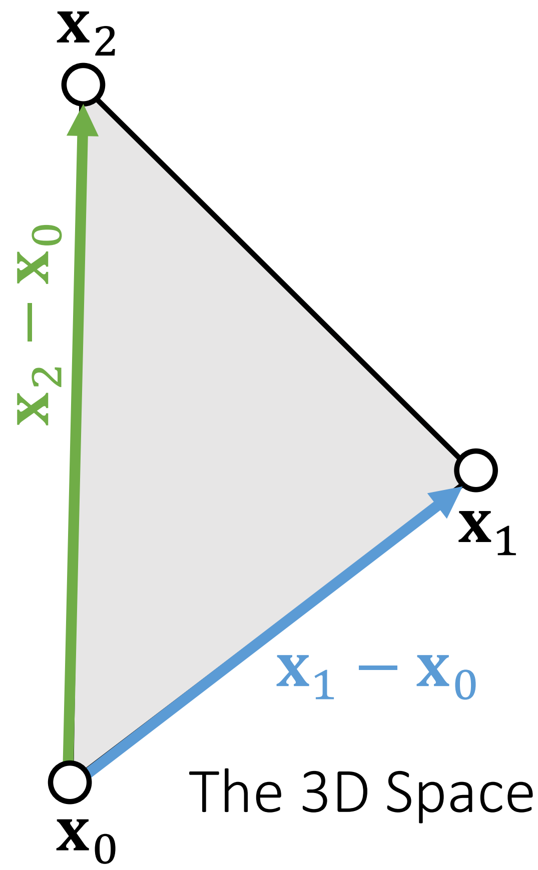
Assuming that **R** is constant,

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

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

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

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



The 3D Space

A Summary For the Day

- A mass-spring system
  - Planar springs against stretching/compression - replaceable by co-rotational model
  - 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 $\mathbf{H}(\mathbf{x}) = \partial E^2(\mathbf{x})/\partial \mathbf{x}^2$. 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:

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

  - 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.