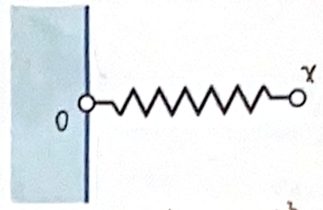
- Mass Spring System

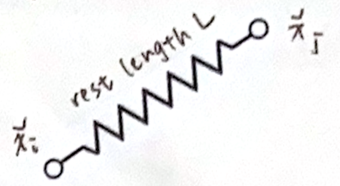
An ideal spring satisfies Hooke's law: the spring force tries to restore the rest length.



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

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

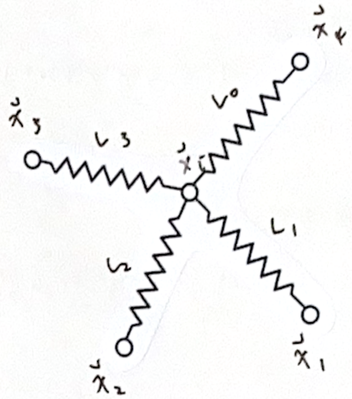
spring stiffness



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

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

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



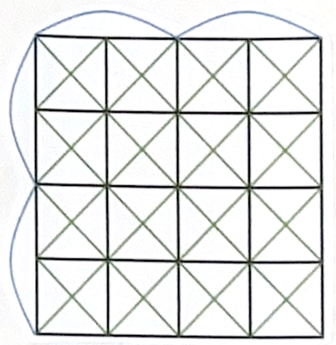
When there are many springs, the energies and the forces can be simply summed up.

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

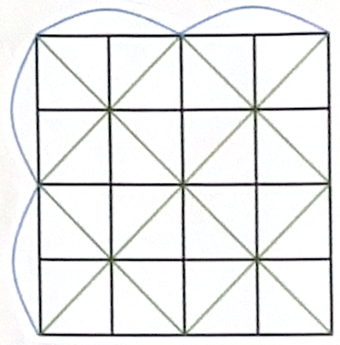
$$\vec{f} = -\nabla_i E = \sum_{e=0}^{3} \left(-k\|\vec{x}_i - \vec{x}_e\| - L_e\right)\frac{\vec{x}_i - \vec{x}_e}{\|\vec{x}_i - \vec{x}_e\|}$$

→ structured spring networks

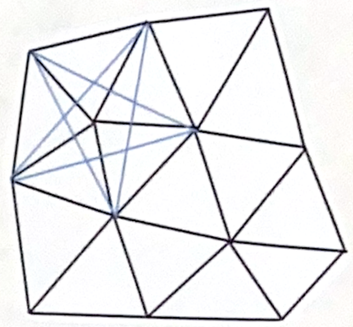
— Edge — Bending — Diagonal



structured network



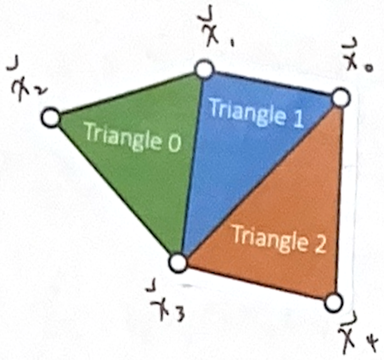
simplified network



untructured △ mesh

★ Triangle Mesh Representation

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



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

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

The key to topological construction is to sort triangle edge triples. Each triple contains:

1. edge vertex index 0
2. edge vertex index 1 (index 0 < index 1)
3. triangle index

⇒ Triple list: $\{\underbrace{\{1,2,0\},\{2,3,0\},\{1,3,0\}}_{\Delta 0}, \underbrace{\{0,1,1\},\{1,3,1\},\{0,3,1\}}_{\Delta 1}, \underbrace{\{0,3,2\},\{3,4,2\},\{0,4,2\}}_{\Delta 2}\}$

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: $E = \{\{0,1\},\{0,3\},\{0,4\},\{1,2\},\{1,3\},\{2,3\},\{3,4\}\}$

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

★ Explicit Integration of A Mass-Spring System

Compute Spring Force

For every edge(spring) e

$i = E[e][0]$

$j = E[e][1]$

$L_e = L[e]$ → edge length list

$\vec{f} = -K(\|\vec{x}_i - \vec{x}_j\| - L_e)\dfrac{\vec{x}_i - \vec{x}_j}{\|\vec{x}_i - \vec{x}_j\|}$

$\vec{f}_i = \vec{f}_i + \vec{f}$

$\vec{f}_j = \vec{f}_j - \vec{f}$

A Particle System

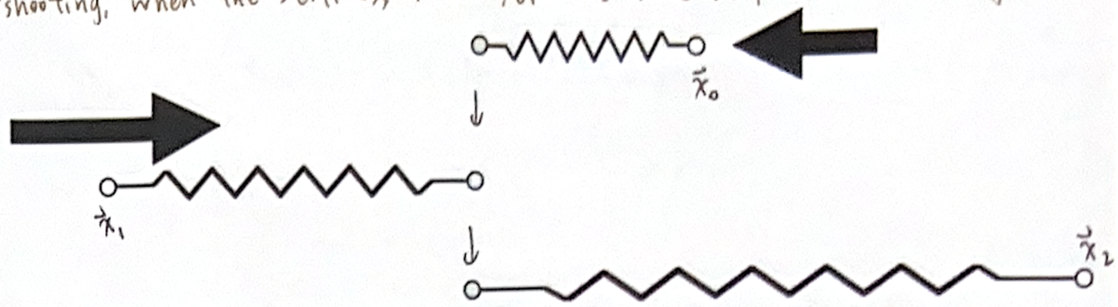
For every vertex

$\vec{f}_i = \#$

$\vec{v}_{n+1,i} = \vec{v}_{n,i} + \Delta t\, m_i^{-1}\vec{f}_i$ (mass of vertex i)

$\vec{x}_{n+1,i} = \vec{x}_{n,i} + \Delta t\, \vec{v}_{n+1,i}$

However, explicit integration suffers from numerical instability caused by overshooting, when the stiffness k and/or the time step Δt is too large.



A native solution is to use a small Δt, but that slows down the simulation.

★ Implicit Integration

$$\begin{cases}\vec{v}_{n+1} = \vec{v}_n + \Delta t\,\vec{M}^{-1}\vec{f}_n \\ \vec{x}_{n+1} = \vec{x}_n + \Delta t\,\vec{v}_{n+1}\end{cases} \rightarrow \begin{cases}\vec{x}_{n+1} = \vec{x}_n + \Delta t\,\vec{v}_n + \Delta t^2\vec{M}^{-1}\vec{f}_n \\ \vec{v}_{n+1} = (\vec{x}_{n+1} - \vec{x}_n)/\Delta t\end{cases}$$

Assume that $\vec{f}$ is holonomic, which is depending on $\vec{x}$ only, then the equation would be:

$$\vec{x}_{n+1} = \vec{x}_n + \Delta t\,\vec{v}_n + \Delta t^2\vec{M}^{-1}\vec{f}(\vec{x}_{n+1})$$

By optimization theory, solving the equation is equivalent to

$$\vec{x}_{n+1} = \operatorname{argmin}(F(\vec{x})), \text{ where } F(\vec{x}) = \frac{1}{2\Delta t^2}\|\vec{x} - \vec{x}_n - \Delta t\vec{v}_n\|_{\vec{M}}^2 + E(\vec{x})$$

(Because $\nabla F(\vec{x}_{n+1}) = \frac{1}{\Delta t^2}\vec{M}(\vec{x}_{n+1} - \vec{x}_n - \Delta t\,\vec{v}_n) - \vec{f}(\vec{x}_{n+1}) = 0$

$\rightarrow \vec{x}_{n+1} - \vec{x}_n - \Delta t\,\vec{v}_n - \Delta t^2\vec{M}^{-1}\vec{f}(\vec{x}_{n+1}) = 0$)

Use Newton's method to solve $\vec{x}_{n+1} = \operatorname{argmin}(F(\vec{x}))$. By giving current $\vec{x}_n$, we approximate our goal by $0 = \nabla F(\vec{x}) \approx \nabla F(\vec{x}_n) + \frac{\partial F^2(\vec{x}_n)}{\partial \vec{x}^2}(\vec{x} - \vec{x}_n)$

Newton's Method

Initialize $\vec{x}_0$

For k = 0 ... K

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

$\vec{x}^{k+1} = \vec{x}^k + \Delta\vec{x}$

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

$\vec{x}_1 = \vec{x}^{k+1}$

在一個 Δt 內需逐步進行迭代直到收斂

$\vec{x}_0$ 上標表時間步之間迭代的過程

下標表時間點

$\#\|\vec{x}\|_M^2 = \vec{x}^T\vec{M}\vec{x}$

- Simulation by Newton's Method

Now we have

$$\begin{cases} F(\vec{x}) = \frac{1}{2\Delta t^2}\|\vec{x}-\vec{x}_n-\Delta t\vec{v}_n\|_M^2 + E(\vec{x}) \\ \nabla F(\vec{x}) = \frac{1}{\Delta t^2}M(\vec{x}-\vec{x}_n-\Delta t\vec{v}_n) - \nabla\vec{f}(\vec{x}) \\ \frac{\partial^2 F(\vec{x})}{\partial \vec{x}^2} = \frac{M}{\Delta t^2} + \vec{H}(\vec{x}) \end{cases}$$

Hessian

The process of using $\vec{x}_0$ to iterate $\vec{x}_1$, only 1 step!

Initialize $\vec{x}_{0,0}$, often as the position $\vec{x}_0$ or $\vec{x}_0+\Delta t\vec{v}_0$ as initial guess

For $k=0\cdots K$

Solve $\left(\frac{M}{\Delta t^2}+\vec{H}(\vec{x}^k)\right)\Delta\vec{x} = -\frac{1}{\Delta t^2}M(\vec{x}^k-\vec{x}_0-\Delta t\vec{v}_0)+\vec{f}(\vec{x}^k)$

$\vec{x}^{k+1} = \vec{x}^k + \Delta\vec{x}$

if $\|\Delta\vec{x}\|$ is small then break

$\vec{x}_1 = \vec{x}^{k+1}$

$\vec{v}_1 = (\vec{x}_1-\vec{x}_0)/\Delta t$

The spring Hessian is

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

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

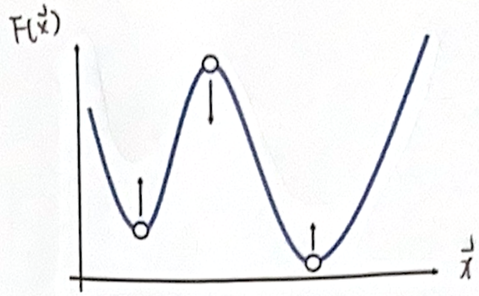
- $\vec{x}_{ij} = \vec{x}_i - \vec{x}_j$
- s.p.d. = symmetric positive definiteness

$\frac{\vec{x}_{ij}\vec{x}_{ij}^T}{\|\vec{x}_{ij}\|^2}$: s.p.d

$k\left(1-\frac{L}{\|\vec{x}_{ij}\|}\right)$: negative if $\|\vec{x}_{ij}\| < L_e$

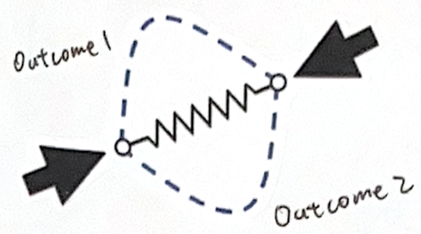
$\left(\vec{I}-\frac{\vec{x}_{ij}\vec{x}_{ij}^T}{\|\vec{x}_{ij}\|^2}\right)$: s.p.d

When a spring is stretched, $\vec{H}_e$ is s.p.d., but when it's compressed, $\vec{H}_e$ may not be s.p.d. As a result, $\vec{H}(\vec{x})$ may not be s.p.d. So as $\frac{\partial^2 F(\vec{x})}{\partial \vec{x}^2}$.



If $\frac{\partial^2 F}{\partial \vec{x}^2}$ is positive definite everywhere, $F(\vec{x})$ has no maximum but only one minimum.

▲ This is a sufficient but not necessary condition.



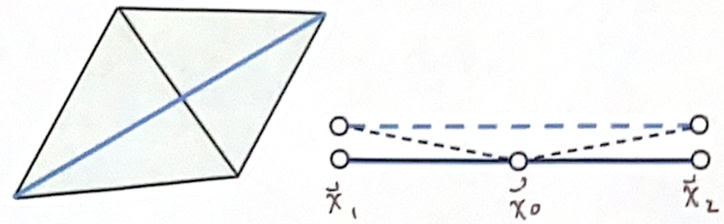
When a spring is compressed, the spring Hessian may not be positive definite. This means there can be multiple local minima.

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

- Bending issue

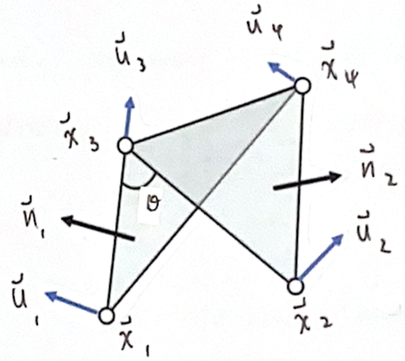
A bending spring offers little resistance when cloth is nearly planar, since its length barely changes.

$\vec{x}_1, \vec{x}_2$ 距離變化的力量不大



1. Dihedral Angle Model:

This model defines bending forces as function of θ: $\vec{f}_i = f(\theta)\vec{u}_i$



1. $\vec{u}_1 \parallel \vec{n}_1$, $\vec{u}_2 \parallel \vec{n}_2$
2. bending doesn't stretch the edge, so $\vec{u}_4-\vec{u}_3$ should be orthogonal to the edge-span of $\vec{n}_1$ and $\vec{n}_2$
3. $\vec{n}_1+\vec{n}_2+\vec{n}_3+\vec{n}_4=0$ (same meaning)

To sum up, by $\vec{N}_1 = (\vec{x}_1-\vec{x}_3)\times(\vec{x}_1-\vec{x}_4)$, $\vec{E}=\vec{x}_4-\vec{x}_3$,
$\vec{N}_2 = (\vec{x}_2-\vec{x}_4)\times(\vec{x}_2-\vec{x}_3)$

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

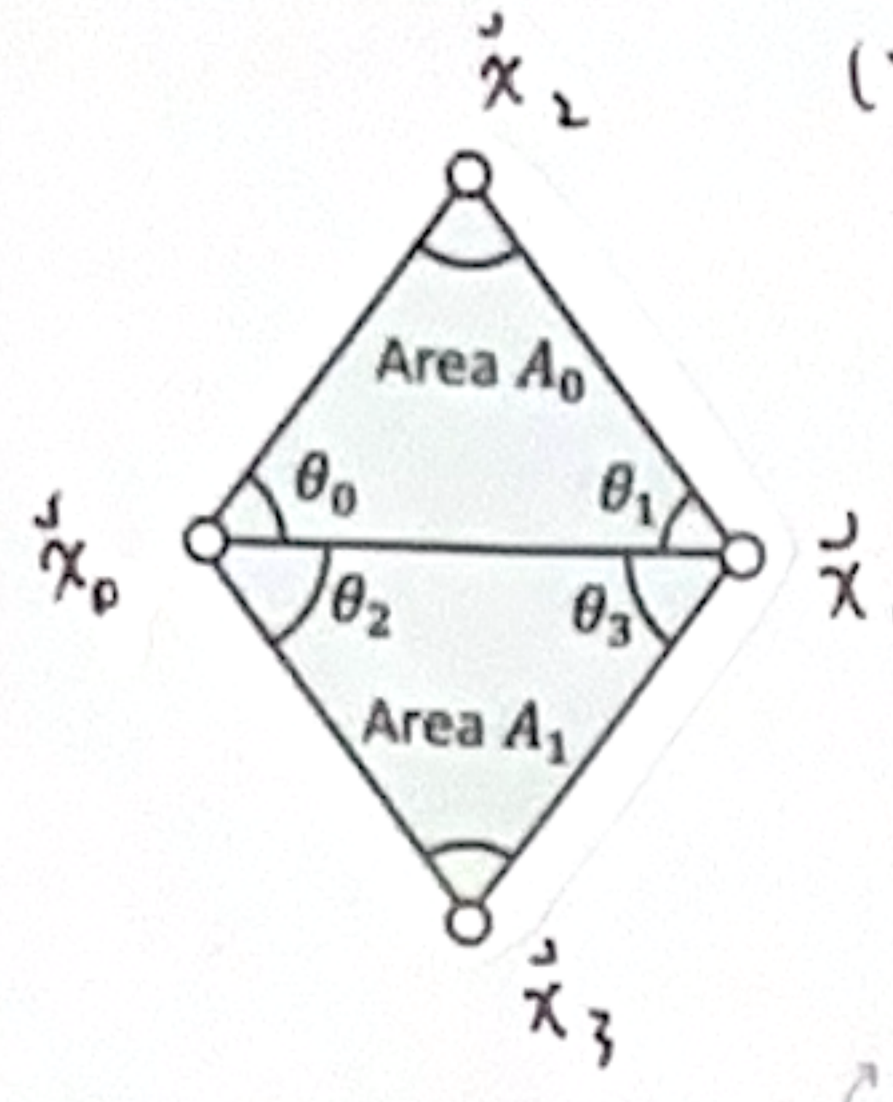
$$\vec{u}_4 = -\frac{(\vec{x}_1-\vec{x}_3)\cdot\vec{E}}{\|\vec{E}\|}\frac{\vec{N}_1}{\|\vec{N}_1\|^2} - \frac{(\vec{x}_2-\vec{x}_3)\cdot\vec{E}}{\|\vec{E}\|}\frac{\vec{N}_2}{\|\vec{N}_2\|^2}$$

→ For Non-planar case: $\vec{f}_i = k\frac{\|\vec{E}\|^2}{\|\vec{N}_1\|+\|\vec{N}_2\|}\left(\sin\left(\frac{\pi-\theta}{2}\right)-\sin\left(\frac{\pi-\theta_0}{2}\right)\right)\vec{u}_i$ (planar case: $\theta_0=0$)

2. Quadratic Bending Model:

Assumption: (1) planar case (when static)

(2) little stretching



Define energy function: $E(\vec{x}) = \frac{1}{2}[\vec{x}_0\ \vec{x}_1\ \vec{x}_2\ \vec{x}_3]\vec{Q}\begin{bmatrix}\vec{x}_0\\\vec{x}_1\\\vec{x}_2\\\vec{x}_3\end{bmatrix}^{12\times1}$,

(Not by physics, by the defination of 曲率 in math.)

where $\vec{Q}^{12\times12} = \frac{3}{A_0+A_1}\vec{q}\vec{q}^T$ and $\vec{q} = \begin{bmatrix}(\cot\theta_1+\cot\theta_3)I\\(\cot\theta_0+\cot\theta_2)I\\(-\cot\theta_0-\cot\theta_1)I\\(-\cot\theta_2-\cot\theta_3)I\end{bmatrix} \in R^{12\times3}$

(Laplace → 兩△之間的曲率)

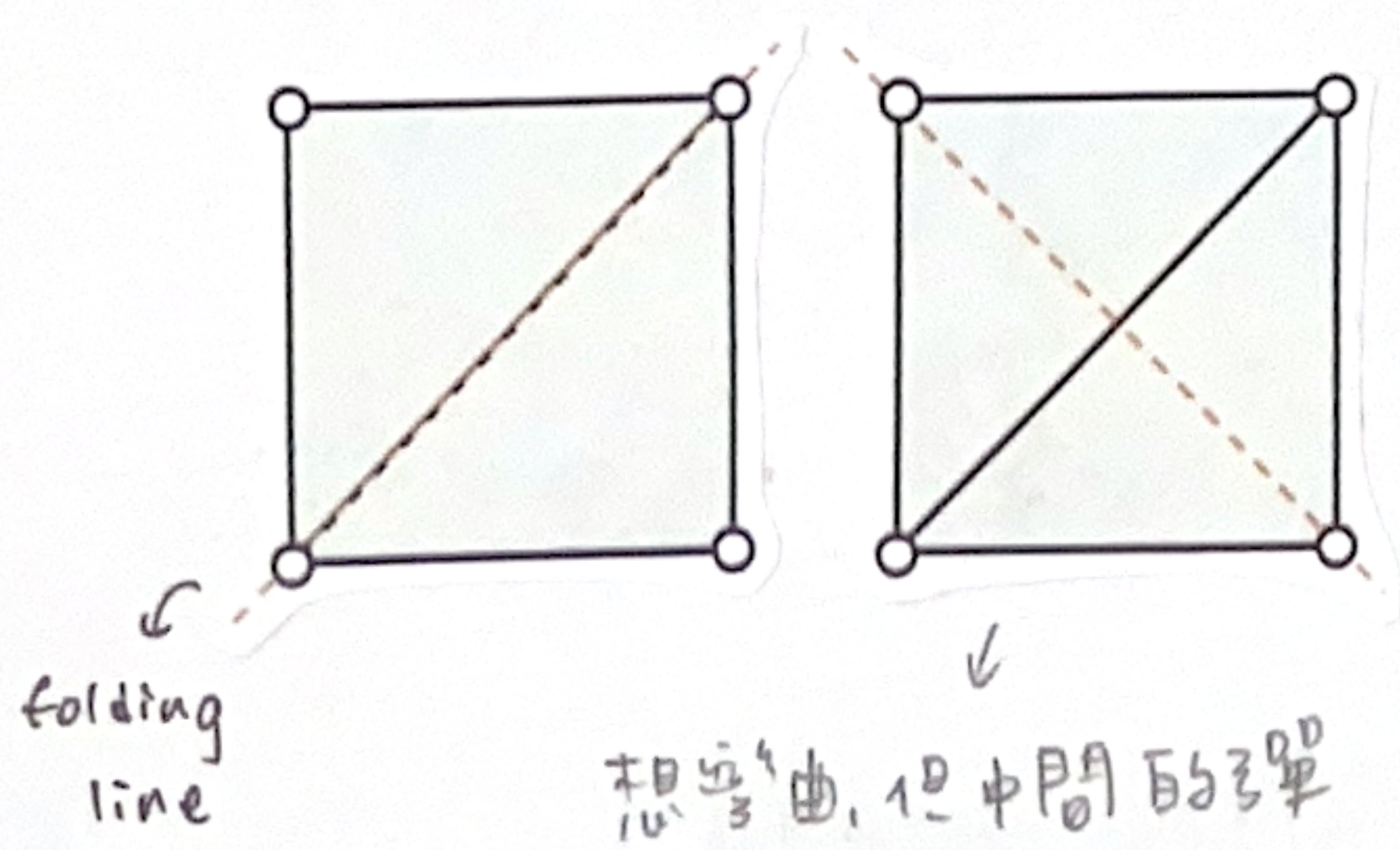
→ $E(\vec{x}) = \frac{3\|\vec{q}^T\vec{x}\|^2}{2(A_0+A_1)}$. $E(\vec{x})=0$ when the triangles are flat.

Short summary:

	Dihedral Angle Model	Quadratic Bending Model
pro	According to force Explicit integration	According to energy, so it is easy to implement: $\vec{f}(\vec{x}) = -\nabla E(\vec{x}) = -\vec{Q}\begin{bmatrix}\vec{x}_0\\\vec{x}_1\\\vec{x}_2\\\vec{x}_3\end{bmatrix}$ $\vec{H}(\vec{x}) = \frac{\partial^2 E(\vec{x})}{\partial \vec{x}^2} = \vec{Q}$ Compatible with implicit integration
con	Hard to derivation	No longer valid if cloth stretchs much Not suitable if the rest configuration is not planar (理論上彈性係數大，和彎曲相關)

- Locking issue: 彎曲會被彈簧edge上的彈性干擾
  ex. 彈簧很stiff, mesh resolution low → 布料會彎不下去，紙很硬，但可彎曲

Think about a zero bending case, can a simulator fold cloth freely?



Reason: short of degree of freedoms (DoFs)

For manifold mesh ($\triangle$ mesh): #edges = 3#vertices − 3 − #boundary_edges

If the edges are all hard constraints, the DoFs are only: 3 + #boundary_edges (3#vertices − #edges)

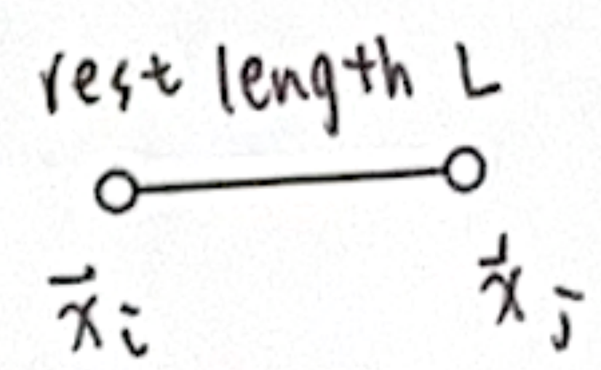
- Position Based Dynamics

In order to solve the stiffness issue, which means the real-world fabric resist strongly to stretching once they stretch beyond certain limits.
（在一定程度的拉伸後，就拉不動了）

But, increacing the stiffness can cause problems, (1) explicit become unstable, and (2) implicit become ill-conditioned. we want to achieve high stiffness with low computational cost.

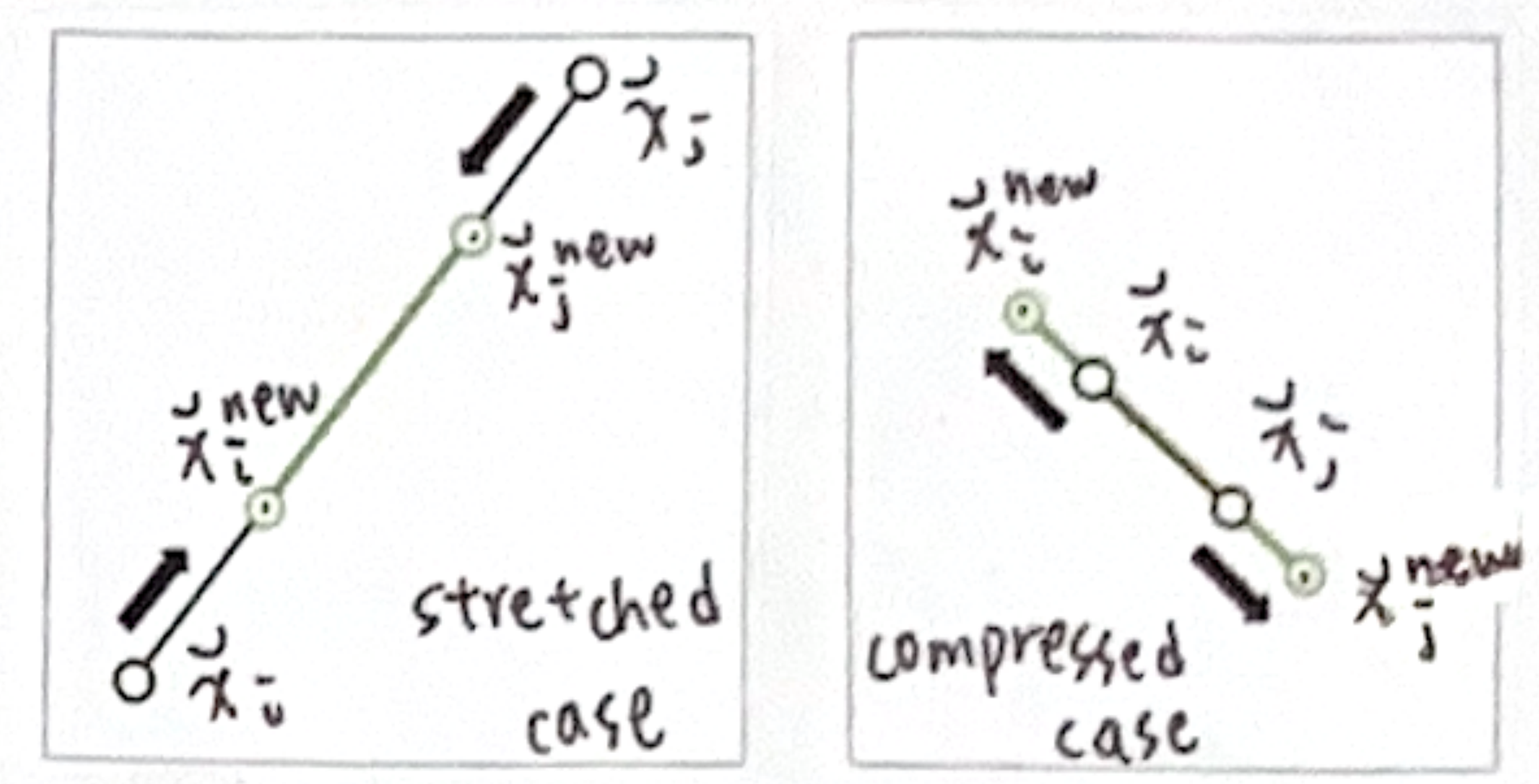
★ A Single Spring

If a spring is infinitely stiff, we can treat the length as a constraint and define a projection function.



Constraint:

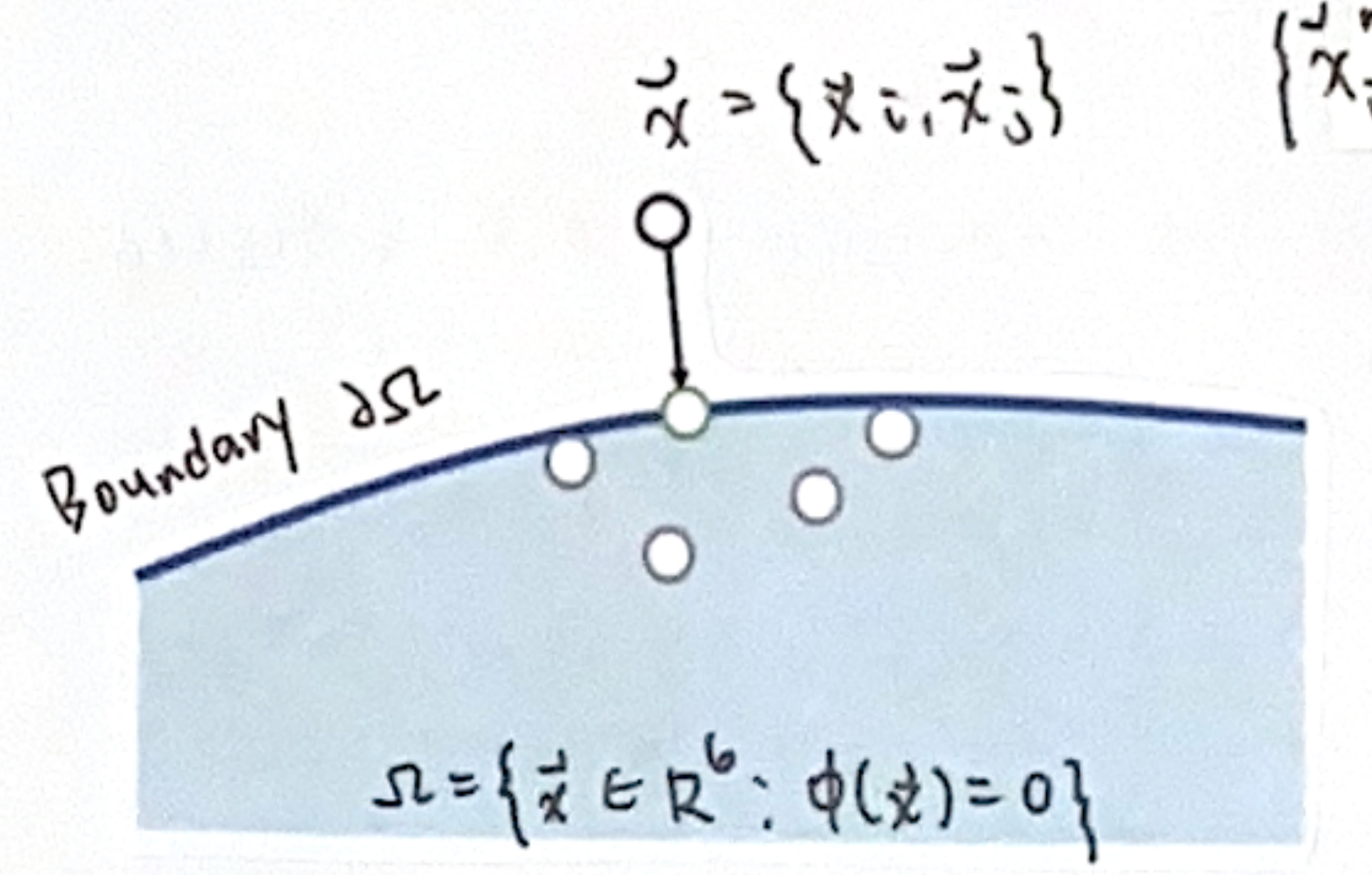
$\phi(\vec{x}) = \|\vec{x}_i - \vec{x}_j\| - L = 0$



Projection function:

$\{\vec{x}_i^{new}, \vec{x}_j^{new}\} = \text{argmin}\frac{1}{2}\{m_i\|\vec{x}_i^{new} - \vec{x}_i\|^2 + m_j\|\vec{x}_j^{new} - \vec{x}_j\|^2\}$,

such that $\phi(\vec{x}) = 0$



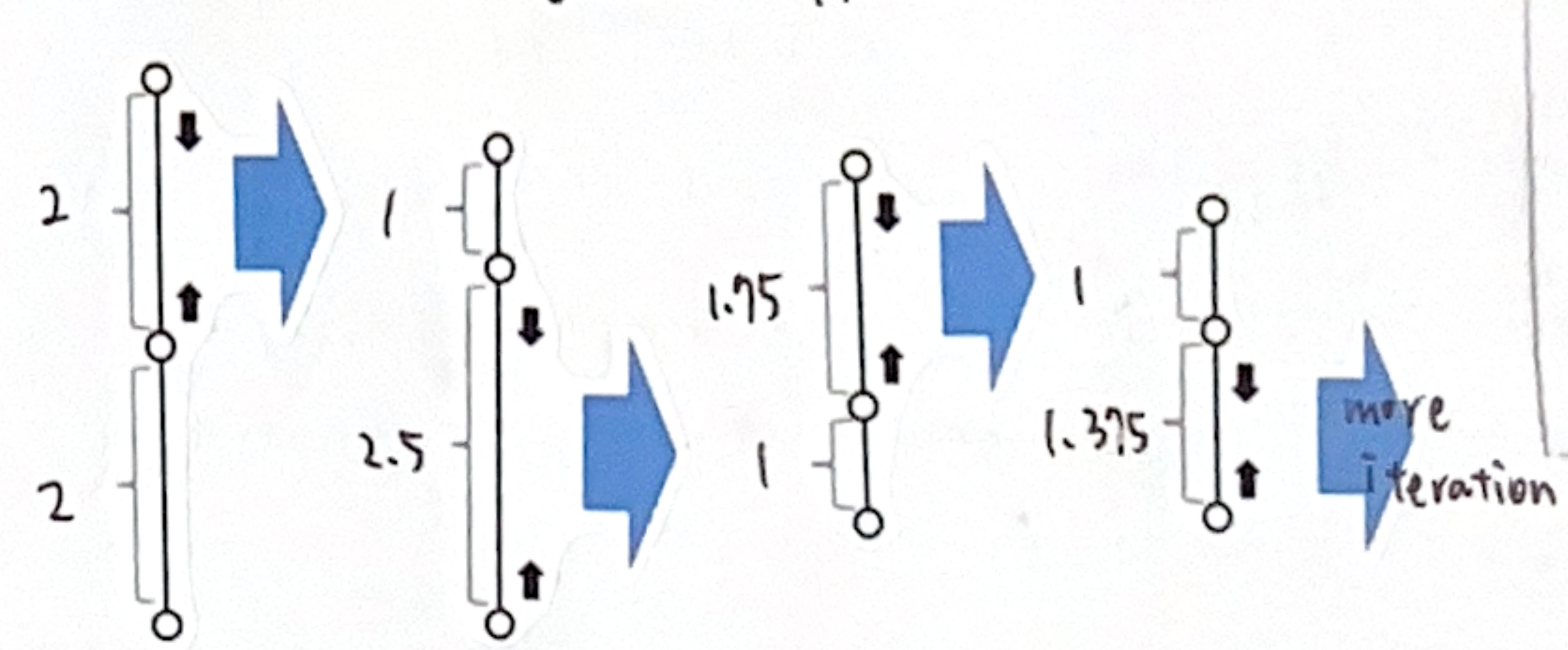
$\vec{x}_i^{new} = \vec{x}_i - \frac{m_j}{m_i+m_j}(\|\vec{x}_i - \vec{x}_j\| - L)\frac{\vec{x}_i - \vec{x}_j}{\|\vec{x}_i - \vec{x}_j\|}$

$\vec{x}_j^{new} = \vec{x}_j + \frac{m_i}{m_i+m_j}(\|\vec{x}_i - \vec{x}_j\| - L)\frac{\vec{x}_i - \vec{x}_j}{\|\vec{x}_i - \vec{x}_j\|}$

★ 質心位置不變

★ Multiple Springs

1. Gauss-Seidel Approach



↑ project each spring sequentially in a certain order

Projection (by Gauss-Seidel)
for $k = 0 \ldots K$
  for every edge $e = \{i, j\}$
    $\vec{x}_i^{new} = \vec{x}_i - \frac{1}{2}(\|\vec{x}_i - \vec{x}_j\| - L_e)\frac{\vec{x}_i - \vec{x}_j}{\|\vec{x}_i - \vec{x}_j\|}$
    $\vec{x}_j^{new} = \vec{x}_j + \frac{1}{2}(\|\vec{x}_i - \vec{x}_j\| - L_e)\frac{\vec{x}_i - \vec{x}_j}{\|\vec{x}_i - \vec{x}_j\|}$

▲ Not ensure the satisfication of every constrain (string).

▲ Order may cause bias and effect convergence behavior

2. Jacobi Approach

To avoid bias, this method projects all the edges simultaneously and then linearly blend the results.

分別移動點，再將其點所受到的移動量依據加權平均

Projection (by Jacobi)

For $k = 0 \dots K$

  for every vertex i

    $\vec{x}_i^{new} = 0$ # initialize $\vec{x}_i^{new}$

    $n_i = 0$ # count

  for every edge $e = \{i, j\}$

$$\vec{x}_i^{new} = \vec{x}_i^{new} + \vec{x}_i - \frac{1}{2}\left(\|\vec{x}_i - \vec{x}_j\| - L_e\right)\frac{\vec{x}_i - \vec{x}_j}{\|\vec{x}_i - \vec{x}_j\|}$$

$$\vec{x}_j^{new} = \vec{x}_j^{new} + \vec{x}_j + \frac{1}{2}\left(\|\vec{x}_i - \vec{x}_j\| - L_e\right)\frac{\vec{x}_i - \vec{x}_j}{\|\vec{x}_i - \vec{x}_j\|}$$

    n_i ++

    n_j ++

  for every vertex i

$$\vec{x}_i = \left(\vec{x}_i^{new} + \alpha\vec{x}_i\right)/(n_i + \alpha)$$

- lower convergence rate
- more iteration used, the better the constraints are enforced.

★ Position Based Dynamics (PBD)

Based on projection function:

- The stiffness behavior, i.e., how tightly constraints are enforced, is subject to non-physical factor.
  - # of iteration
  - mesh resolution
- The velocity update following projection is important to dynamic effects.
- By defining the projection function, PBD can be simplified apply to other constraint such like triangle constraint, volume constraint, and colligion constraint.
- Pros: parrallelable on GPU (PhyX), easy to implement, fast in low resolution, generic that can handle other coupling and constraints, including fluid
- cons: not physically correct, low performance in high resolutions.

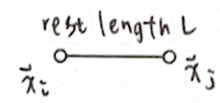
A PBD Simulator

```
// DO simulation, update x and v
v = ...
x = ...
// PBD
x_new = projection(x)
v_new = v + (x_new - x)/Δt
```

★ Strain Limiting

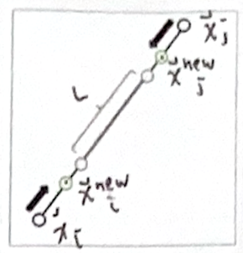
Strain limiting aims at using the projection function for correction only.

- PBD的改進版本
- 1x1保證模擬穩定

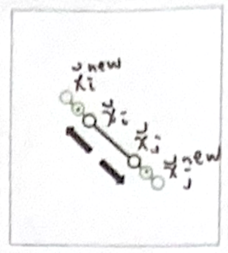


Constraint:

$$\sigma^{min} \le \frac{1}{L}\|\vec{x}_i - \vec{x}_j\| \le \sigma^{max}$$



stretched case



compressed case

Projection (Strain Limit)

$$\sigma = \frac{1}{L}\|\vec{x}_i - \vec{x}_j\|$$

$$\sigma_0 = \min(\max(\sigma, \sigma^{min}), \sigma^{max})$$

$$\vec{x}_i^{new} = \vec{x}_i - \frac{m_j}{m_i + m_j}\left(\|\vec{x}_i - \vec{x}_j\| - \sigma_0 L\right)\frac{\vec{x}_i - \vec{x}_j}{\|\vec{x}_i - \vec{x}_j\|}$$

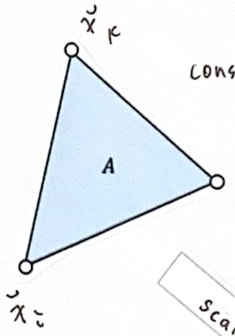
$$\vec{x}_j^{new} = \vec{x}_j - \frac{m_i}{m_i + m_j}\left(\|\vec{x}_i - \vec{x}_j\| - \sigma_0 L\right)\frac{\vec{x}_i - \vec{x}_j}{\|\vec{x}_i - \vec{x}_j\|}$$

- PBD: $\sigma_0 = 1$
- No limit: $\sigma^{min} = 0$, $\sigma^{max} = \infty$

★ Triangle Area Limit

Projection function:

$$\{\vec{x}_i^{new}, \vec{x}_j^{new}, \vec{x}_k^{new}\} = \operatorname{argmin}\frac{1}{2}\left\{m_i\|\vec{x}_i^{new} - \vec{x}_i\|^2 + m_j\|\vec{x}_j^{new} - \vec{x}_j\|^2 + m_k\|\vec{x}_k^{new} - \vec{x}_k\|^2\right\}$$

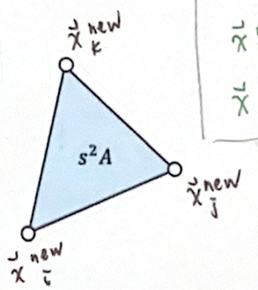


constraint:

$$A^{min} \le A \le A^{max}$$

scaling by S

$$S = \sqrt{\frac{\min(\max(A, A^{min}), A^{max})}{A}}$$



Projection (Δ Area Limit)

$$A = \frac{1}{2}\|(\vec{x}_j - \vec{x}_i) \times (\vec{x}_k - \vec{x}_i)\|$$

$$S = \sqrt{\min(\max(A, A^{min}), A^{max})/A}$$

$$\vec{c} = \frac{1}{m_i + m_j + m_k}(m_i\vec{x}_i + m_j\vec{x}_j + m_k\vec{x}_k)$$

$$\vec{x}_i^{new} = \vec{c} + s(\vec{x}_i - \vec{c})$$

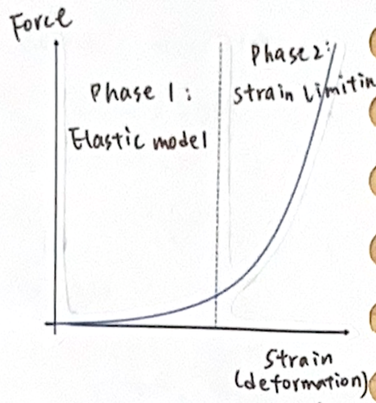
$$\vec{x}_j^{new} = \vec{c} + s(\vec{x}_j - \vec{c})$$

$$\vec{x}_k^{new} = \vec{c} + s(\vec{x}_k - \vec{c})$$

Short Summary:

1. Strain limiting (保證沒有太形變) is widely used in physics-based simulation, typically for avoiding instability and artifacts due to large deformation.

ex. 史萊姆：有太形變則不需使用 string limit
肌肉：沒有 〃 需 〃



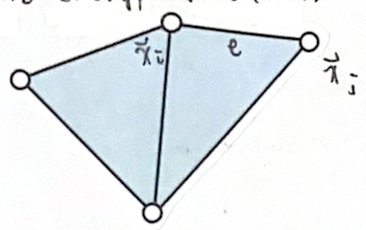
2. Strain limiting is useful for nonlinear effects, in a biphasic way.
3. Strain limiting also helps address the locking issue.

- Projective Dynamic (利用 $\vec{x}^{new}$ 來定義能量而非改變頂點位置 → 以得到常數的 $\vec{H}$)

Instead of blending projections in a Jacobi or Gauss-Seidel fashion as in PBD, projective dynamics uses projection to define a quadratic energy.

Quadratic energy function:

$$E(\vec{x}) = \sum_{e=\{i,j\}} \frac{k}{2} \| (\vec{x}_i - \vec{x}_j) - (\vec{x}_i^{new} - \vec{x}_j^{new}) \|^2$$



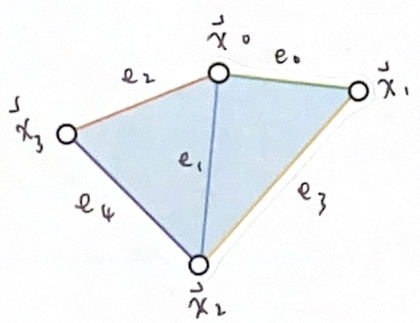
where $\{\vec{x}_i^{new}, \vec{x}_j^{new}\} = projection(\vec{x}_i, \vec{x}_j)$ for every edge e

$$= \sum_{e=\{i,j\}} \frac{k}{2} (\|\vec{x}_i - \vec{x}_j\| - L_e)^2 \rightarrow \text{力學能量公式}$$

$$\rightarrow \vec{f}_i = -\nabla_i E(\vec{x}) = -\sum_{e:i \in e} k(\vec{x}_i - \vec{x}_j) - (\vec{x}_i^{new} - \vec{x}_j^{new})$$

$$= -k \sum_{e:i \in e} (\|\vec{x}_i - \vec{x}_j\| - L_e) \frac{\vec{x}_i - \vec{x}_j}{\|\vec{x}_i - \vec{x}_j\|} \rightarrow \text{彈簧拉力!}$$

For example:



$$E(\vec{x}) = \frac{1}{2}\|(\vec{x}_0 - \vec{x}_1) - (\vec{x}_0^{new} - \vec{x}_1^{new})\|^2 + \frac{1}{2}\|(\vec{x}_0 - \vec{x}_2) - (\vec{x}_0^{new} - \vec{x}_2^{new})\|^2 + \frac{1}{2}\|(\vec{x}_0 - \vec{x}_3) - (\vec{x}_0^{new} - \vec{x}_3^{new})\|^2 + \frac{1}{2}\|(\vec{x}_1 - \vec{x}_2) - (\vec{x}_1^{new} - \vec{x}_2^{new})\|^2 + \frac{1}{2}\|(\vec{x}_2 - \vec{x}_3) - (\vec{x}_2^{new} - \vec{x}_3^{new})\|^2$$

for $\vec{x}_0$

$$\rightarrow f(\vec{x}_0) = (\vec{x}_0^{new} - \vec{x}_1^{new}) - (\vec{x}_0 - \vec{x}_1) + (\vec{x}_0^{new} - \vec{x}_2^{new}) - (\vec{x}_0 - \vec{x}_2) + (\vec{x}_0^{new} - \vec{x}_3^{new}) - (\vec{x}_0 - \vec{x}_3)$$

$$\rightarrow \vec{H} = \begin{bmatrix} 3I & -I & -I & -I \\ -I & 2I & -I & \\ -I & -I & 3I & -I \\ -I & & -I & 2I \end{bmatrix}$$

1. 斜對角數字表示 e 點連接幾個邊
2. -I 代表有連線的兩點
3. 常數矩陣!

06

- Simulation by Projective Dynamics

According to implicit integration and Newton's method, a projective dynamics simulor looks as follows, with matrix $\vec{A} = \frac{\vec{M}}{\Delta t^2} + \vec{H}$. (ref. C2-2)

constant matrix

Initialize $\vec{x}_0$, often as $\vec{x}_0$ or $\vec{x}_0 + \Delta t \vec{v}_0$.

for k = 0 ... K

recalculate projection

Solve $\left(\frac{\vec{M}}{\Delta t^2} + \vec{H}\right) \Delta \vec{x} = -\frac{\vec{M}}{\Delta t^2}(\vec{x}^k - \vec{x}_0 - \Delta t \vec{v}_0) + \vec{f}(\vec{x}^k)$

$\vec{x}^{k+1} = \vec{x}^k + \Delta \vec{x}$

if $\|\Delta \vec{x}\|$ is small than break

$\vec{x}_1 = \vec{x}^{k+1}$

$\vec{v}_1 = (\vec{x}_1 - \vec{x}_0)/\Delta t$

NOTICE!! There are a lot of methods do not calculate $\vec{H}$ exactly. The performance depends on how well the Hessian is approximated.

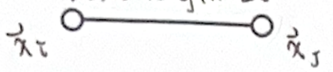
① truncating the Hessian to make it S.p.d.
② a diagonal approximation (HW2)
③ constant matrix by dynamic projection.

Pros: has physical meaning, fast on CPUs with direct solver, fast convergence in the first few iterations

Cons: slow on GPUs, slow convergence over time, can't easily handle constraint changes.

- Constrained Dynamics

To deal with the problem that the constraints/forces are very stiff.



$$E(\vec{x}) = \sum_e \frac{1}{2} k (\|\vec{x}_i - \vec{x}_j\| - L_e)^2 = \frac{1}{2} \underbrace{\vec{\phi}^T(\vec{x})}_{\in R^e} \vec{C}^{-1} \vec{\phi}(\vec{x}), \quad \vec{C}^{-1} = \begin{bmatrix} \frac{1}{k} & & \\ & \frac{1}{k} & \\ & & \ddots \end{bmatrix} \in R^{e \times e}$$

constraint:

$$\phi(\vec{x}) = \|\vec{x}_i - \vec{x}_j\| - L_e$$

$$f(\vec{x}) = -\nabla E = -\left(\frac{\partial E}{\partial \vec{\phi}} \frac{\partial \vec{\phi}}{\partial \vec{x}}\right)^T = -\underbrace{\vec{J}^T}_{\text{Jacobian: } \frac{\partial \vec{\phi}}{\partial \vec{x}} \in R^{e \times 3N}} \vec{C}^{-1} \vec{\phi} = \vec{J}^T \underbrace{\vec{\lambda}}_{-\vec{C}^{-1}\vec{\phi} \in R^e}$$

dual variable (Lagrangian multipliers)

By momentum conservation:

$$\vec{M} \vec{v}^{new} - \Delta t \vec{J}^T \vec{\lambda}^{new} = \vec{M} \vec{v}, \quad \vec{J}^T \vec{\lambda}^{new} = \vec{f}(\vec{x})$$

By $\vec{\lambda} = -\vec{C}^{-1}\vec{\phi}$ at new state:

$$\vec{C} \vec{\lambda}^{new} = -\vec{\phi}^{new} \approx -\vec{\phi} - \vec{J}(\vec{x}^{new} - \vec{x}) \approx -\vec{\phi} - \Delta t \vec{J} \vec{v}^{new}$$

Taylor expansion

$$\rightarrow \begin{bmatrix} \vec{M} & -\Delta t \vec{J}^T \\ \Delta t \vec{J} & \vec{C} \end{bmatrix} \begin{bmatrix} \vec{v}^{new} \\ \vec{\lambda}^{new} \end{bmatrix} = \begin{bmatrix} \vec{M}\vec{v} \\ -\vec{\phi} \end{bmatrix}$$

Now we have a system with two sets of variables: the primal variable $\vec{x}$ (or $\vec{v} = \dot{\vec{x}}$) and the dual variable $\vec{\lambda}$.

method 1: solve two variables by direct solver.

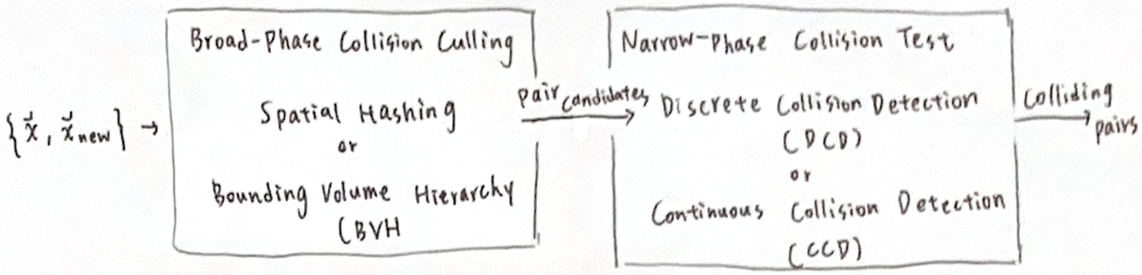
method 2: reduce two equations into one and solve λ^{new} first.

- For very stiff object (also rigid body):
  stiffness $\rightarrow \infty$, means that $\vec{C} \rightarrow 0$.

- example: ragdoll animation

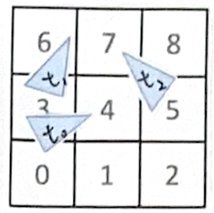
- Collision Detection

* Pipeline



* Spatial Partitioning

This method divides the space by a grid scheme and store objects into grid cell.



cell 6: t_1 cell 7: t_2 cell 8: t_2

cell 3: t_0, t_1 (pair candidate) cell 4: t_0, t_2 (pair candidate) cell 5: t_2

cell 0: t_0 cell 1: cell 2:

- Instead of allocating memories to cells, we can build an object-cell list and then sort them. This avoids memories wasted in empty cells.

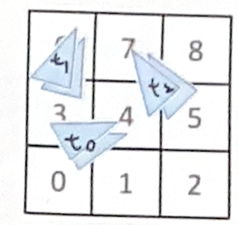
$\{0, t_0\}; \{3, t_0\}; \{4, t_0\}; \{3, t_1\}; \{6, t_1\}; \{4, t_2\}; \{5, t_2\}; \{7, t_2\}; \{8, t_2\};$

↓ sort by cell ID

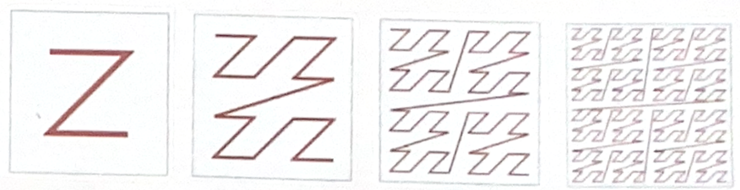
$\{0, t_0\}; \underbrace{\{3, t_0\}; \{3, t_1\}}_{\text{pair candidate}}; \underbrace{\{4, t_0\}; \{4, t_2\}}_{\text{pair candidate}}; \{5, t_2\}; \{6, t_1\}; \{7, t_2\}; \{8, t_2\};$

- If we need to consider moving objects, we just expand the object region.

* Using the grid order to define cell ID is not optimal, since it cannot be easily extended and it is lack of locality. → Morton code use a Z-pattern instead!.

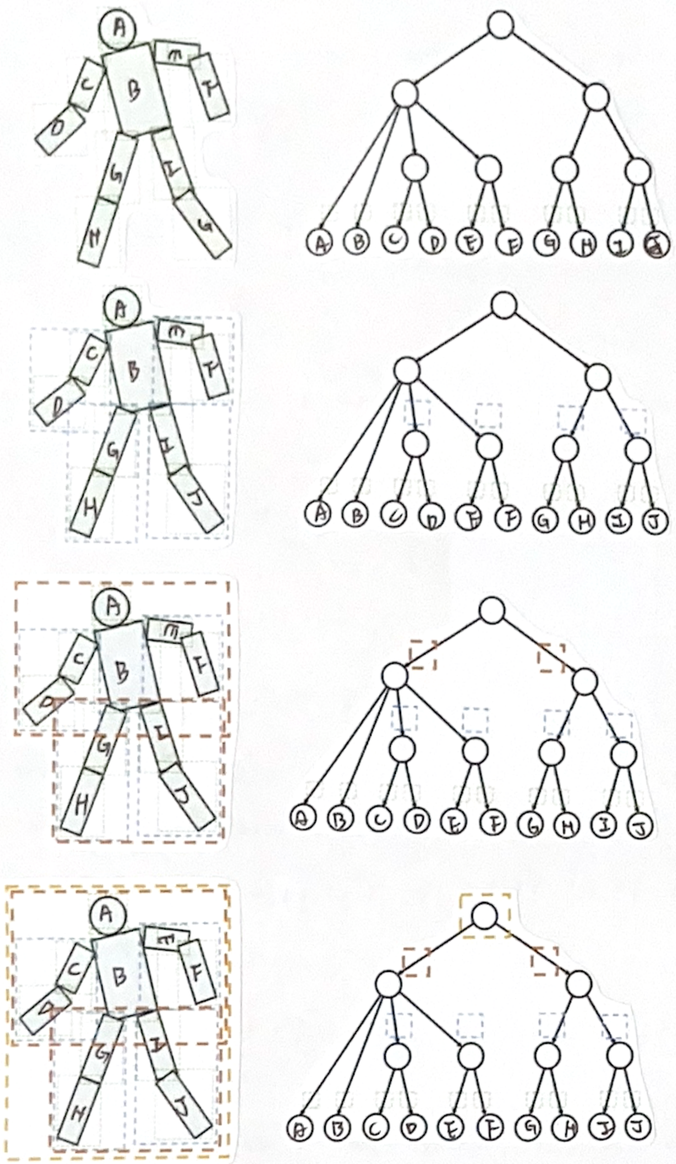


← Grid order
Morton code order

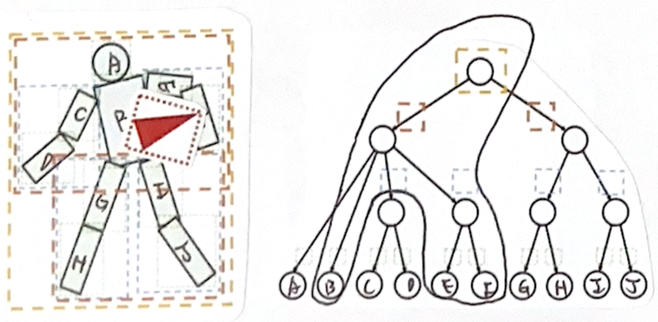


★ Bounding Volume Hierarchy

This method builds on geometric/topological proximity of objects.

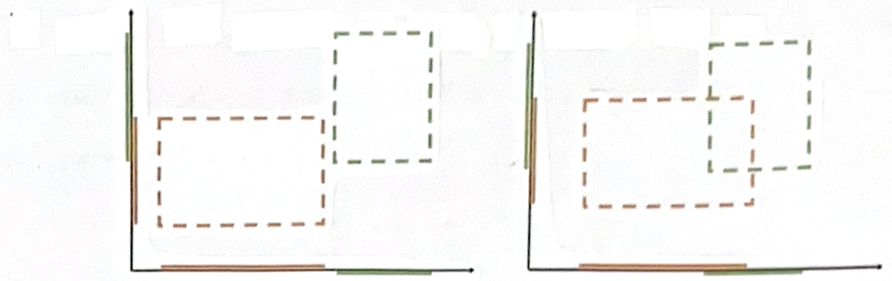


To find elements potentially in collision with an object, we just traverse the tree.



- Axis-aligned bounding box (AABB) is the most popular bounding volume. Besides that, there are also spheres and oriented bounding box (OBB).

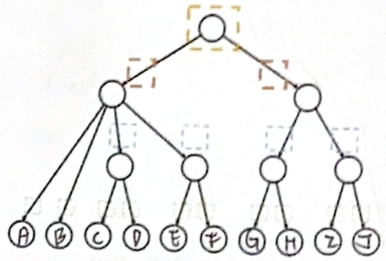
Two AABBs intersect if and only if they intersect in every axis.



To process self collision by BVH, we define two procedures.

```
Process_Node(A) {
   for every A's child : B
      Process_Node(B)
   for every A's children pair <B,C>
      if B and C intersect
         Process_Pair(B,C)
}
```

```
Process_Pair(B,C) {
  for every B's child : B'
    for every C's child : C'
      if B' and C' intersect
         Process_Pair(B',C')
}
```



Short summary:

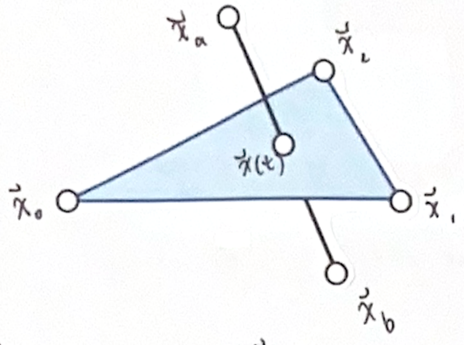
Spatial Hashing: easy to implement, GPU friendly, need to recompute after updating objects

BHV : more involved (比較難), not GPU friendly (because of tree structure), to update BHV just update bounding volumes.

★ Discrete Collision Detection (DCD)

This method tests if any intersection exists in each state at discrete time instant: $\vec{x}_0, \vec{x}_1, \dots$

- edge-triangle intersection test:



Find edge-plane intersection time, solve

$$((1-t)\vec{x}_a + t\vec{x}_b - \vec{x}_0) \cdot \vec{x}_{10} \times \vec{x}_{20} = 0$$

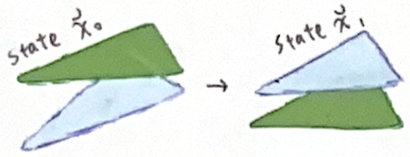
$$\rightarrow \quad t = \frac{\vec{x}_{0a} \cdot \vec{x}_{10} \times \vec{x}_{20}}{\vec{x}_{ba} \cdot \vec{x}_{10} \times \vec{x}_{20}}$$

if $t \in [0,1]$

$\vec{x}(t) = (1-t)\vec{x}_a + t\vec{x}_b$

test if $\vec{x}(t)$ is inside the Δ

if yes → intersection!!

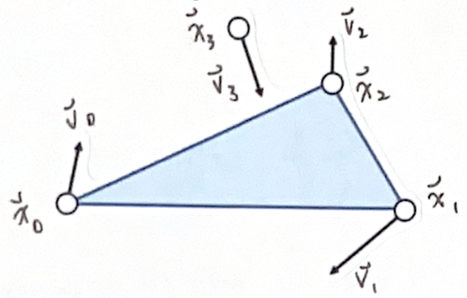


▲ DCD is simple and robust, but it suffers from the tunneling problem: object penetrating through each other without being detected.

★ Continuous Collision Detection (CCD)

This method tests if any intersection exists two state $\vec{x}_0$ and $\vec{x}_1$.

- vertex-triangle intersection test:



Co-planar test, solve $\vec{x}_{30}(t) \cdot \vec{x}_{10}(t) \times \vec{x}_{20}(t) = 0$

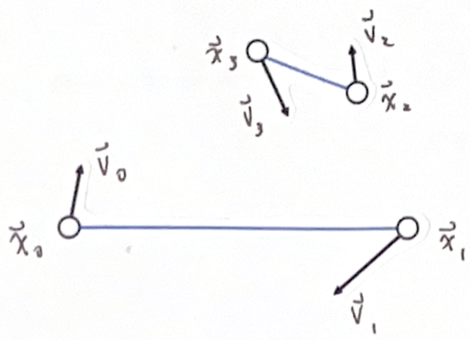
$\rightarrow \quad at^3 + bt^2 + ct + d = 0$ 三次方解根 數值error大! 可用二分法.

if $t \in [0,1]$

test if $\vec{x}_3(t)$ is now inside of $\Delta 012$

if yes → intersection!!

- edge-edge intersection test:



Co-planar test, solve $\vec{x}_{30}(t) \cdot \vec{x}_{10}(t) \times \vec{x}_{20}(t) = 0$

$\rightarrow \quad at^3 + bt^2 + ct + d = 0$

if $t \in [0,1]$

test if the two edges $\overline{23}$ and $\overline{01}$ intersect

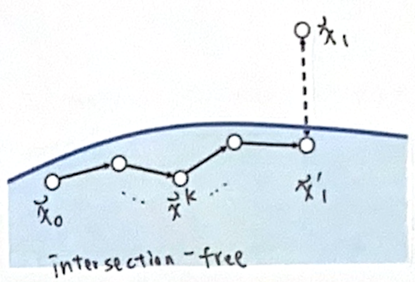
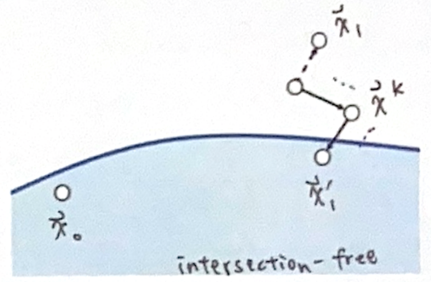
If yes → intersection!!

▲ Issues with CCD

1. Floating-point errors, especially due to root finding of a cubic equation.
   Solution: a. buffering epsilons, but that causes false positives.
   b. gaming GPUs often use single floating-point precision.
2. Computational costs: more expensive than DCD.
   (actually some argue that broad-phase collision culling is the bottleneck.)
3. Difficulty in implementation.

- Collision Responce

Given the calculated next state $\vec{x}_1$, we want to update it into $\vec{x}'_1$, such that the path from $\vec{x}_0$ to $\vec{x}'_1$ is intersection-free.

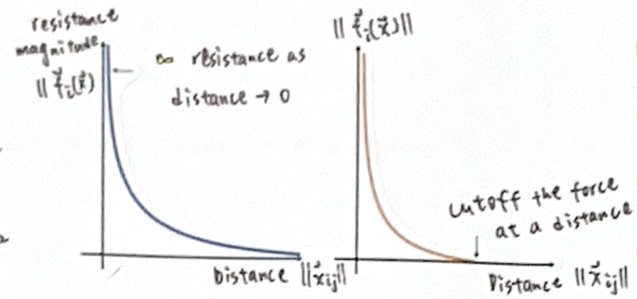
	• Interior Point Methods	• Impact Zone Optimization
configuration		
Pros	Always succeed	Fast! a. Close to solution b. Only vertices in collision (impact zone). c. Can take large step sizes.
Cons	Slow... a. far from solution b. all of the vertices. c. cautiously by small step sizes.	May not succeed
implementation	The problem can be formulated as $\vec{x}'_1 = \arg\min_{\vec{x}} \left(\frac{1}{2}\|\vec{x} - \vec{x}_1\|^2 - \rho \sum \log(\|\vec{x}_{ij}\|) \right)$ By gradient descent: $\vec{x}^0 = \vec{x}_0$ for $k = 0 \dots K$ $\vec{x}^{k+1} = \vec{x}^k + \alpha(\vec{x}_1 - \vec{x}^k + \rho \sum \frac{\vec{x}_{ij}}{\|\vec{x}_{ij}\|^2})$ $\vec{x}'_1 = \vec{x}^{k+1}$	→ adjust α to ensure no collision happens on the way. (do collision test to find α). → log-barrier repulsion (RB 4-2)

▲ Log-Barrier Interior Point Method (RB4-2)

For simplicity, the log-barrier repulsion between two vertices:

$$E(\vec{x}) = -\rho \log \|\vec{x}_{ij}\|$$

$$\rightarrow \begin{cases} \vec{f}_i(\vec{x}) = -\nabla_i E = \rho \dfrac{\vec{x}_{ij}}{\|\vec{x}_{ij}\|^2} \\ \vec{f}_j(\vec{x}) = -\nabla_j E = -\rho \dfrac{\vec{x}_{ij}}{\|\vec{x}_{ij}\|^2} \end{cases}$$



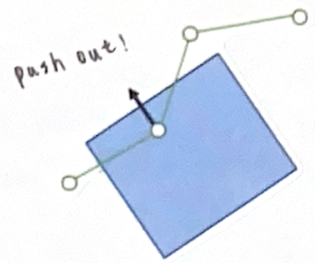
▲ The rigid impact zone method simply freezes vertices in collision from moving in their pre-collision state. It's simple and safe, but has noticable artifact...

• Untangling Cloth

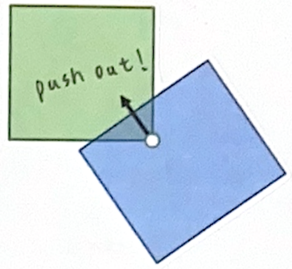
★ Intersection Elimination 相交解除

In the method, we try to deal with the collision problem in every time step, but we don't require the simulation always intersection-free. Such method is useful in past collision handling farlures and intense user interation.

• Cloth-volume intersection

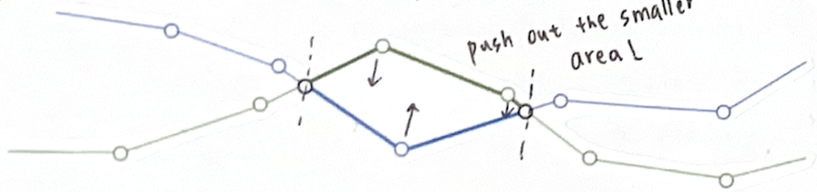


• Volume-volume intersection



• cloth-cloth intersection

Since we don't have a clear definition of inside and outside, It is complicated to deal with this problem.



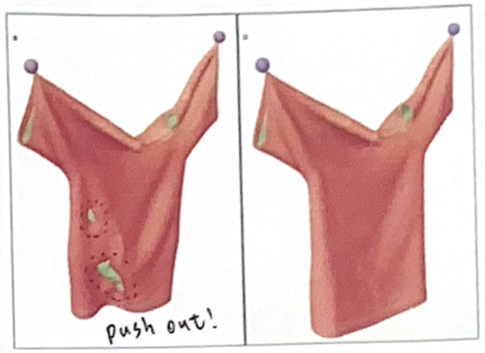
a. flood-fill method to decide which region is in intersection

b. reducing intersection contour

Short summary of collision:

- Collision handling involves two steps: collision detection / collision responce
- Collision detection contains two phases: broad-phase culling / narrow-phase test
- Collision detection test has two types: discrete / continuous
- Collision responce has two types: discrete (allow intersections to stay and hope to remove them in long turn) / continuous
  - must update the state to become collision free state

a.



b.