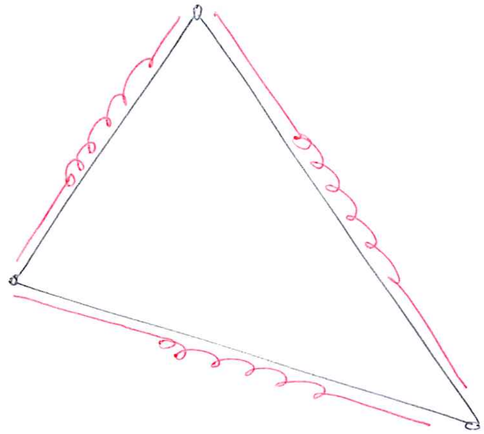
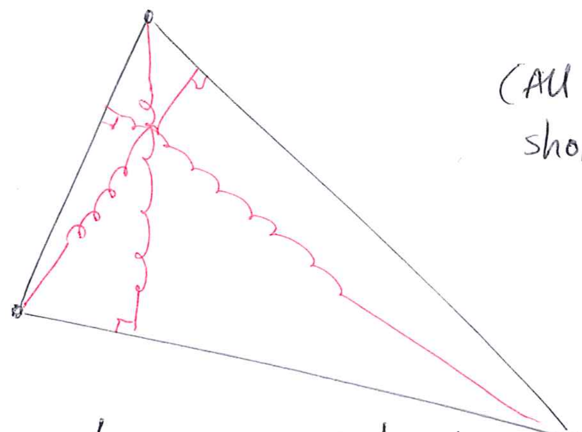
Review:

Types of springs employed in constructing cloth models

Edge Springs:

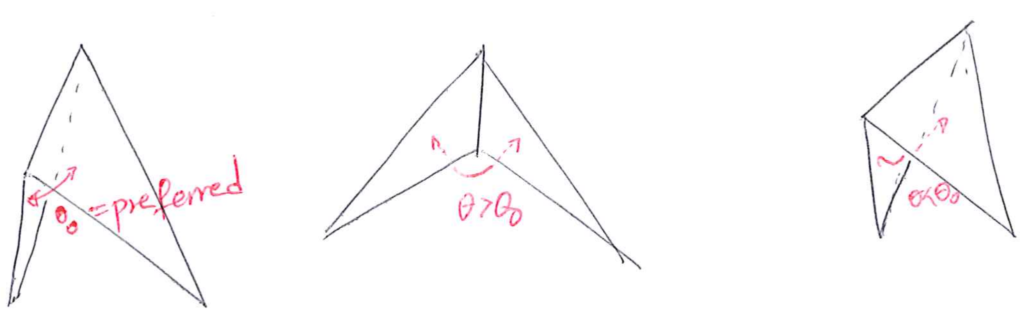


Altitude Springs:



(All three, or just shortest one)

So far, none of the spring types we introduced does anything to discourage "bending" (or impart a "preferred" angle between two triangles)



Approaches:

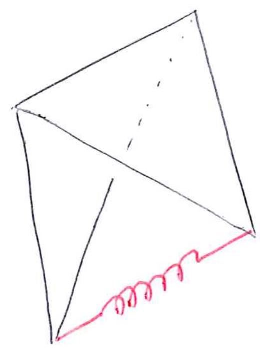
→ Some sort of "angle spring" i.e. something like "$f = -c(\theta - \theta_0)$"

(Compare with $f = -k(l - l_0)$)

⇒ Reasonable in principle, but

- Unclear where such forces should be applied (on vertices? triangles? edges?)
  Any ad-hoc choices for this could easily risk violating conservation of linear or angular momentum (appearing as "ghost" forces that accelerate or rotate an object).
- See Monday's talk for more details.

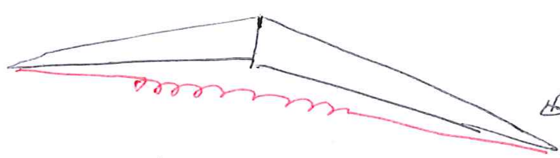
→ Approximate with "cross-connect" springs



\+ Restlength can be easily adjusted to mimic a "preferred" bend angle

\- Not purely an "angle" spring; it interferes with in-plane stretch resistance

e.g. (Near flat pair of triangles

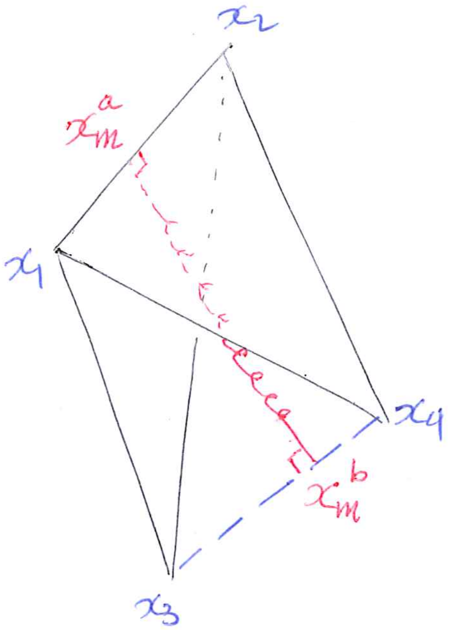


The "bend" spring affects the perceived in-plane response of the triangles to stretching.

<u>Still</u> these springs are very popular due to their simplicity.

<u>Implementation</u>: Instance <u>1</u> spring per adjacent triangle pair (or 1 spring for each mesh edge neighboring 2 triangles).

→ Another approach: "Modified" Bending Springs



Place a spring along the <u>shortest segment</u> connecting lines $\overline{x_1x_2}$ and $\overline{x_3x_4}$ (this spring has to be perpendicular to both line segments).

⇒ Less invasive on overall perceived (in-plane) stiffness.

⇒ Need to compute $\underline{\underline{2}}$ interpolation ratios:

λ_1 : For x_m^a w.r.t. $(x_1 \,\&\, x_2)$

λ_2 : For x_m^b w.r.t. $(x_3 \,\&\, x_4)$

⇒ Implementation of forces: similar to altitude springs, i.e.

- Interpolate velocities e $v_m^a = (1-\lambda_1) v_1 + \lambda_1 v_2$ for damping
- Distribute forces, e.g. $f_3 = (1-\lambda_2) f_m^b$

$$f_4 = \lambda_2 f_m^b$$

Two additional implementation details:

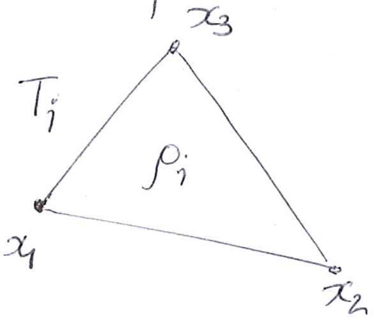
⇒ <u>How to specify mass of particles?</u>

1. Use constant mass for all particles $m_i = \dfrac{m_{total}}{\#\text{particles}} = \text{const}$

→ Simple, and CG may actually converge faster!

→ Inaccurate when there's a variety of triangle sizes

2. Compute mass on a triangle-by-triangle basis;

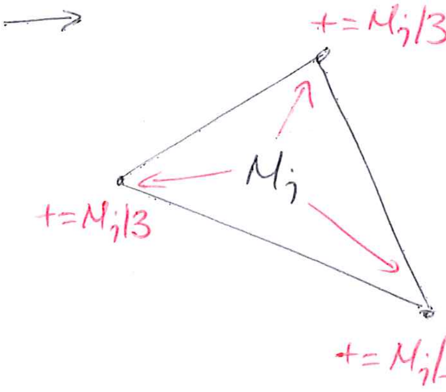


→ Compute Mass of Triangle (M_i) →

$M_i = \rho_i \cdot \text{Area}(\widehat{x_1 x_2 x_3})$

ρ_i = density
(mass per unit area)

Distribute 1/3 to each vertex



foreach (Triangle $T_\ell = \widehat{x_i x_j x_k}$)

$\text{Area}_\ell \leftarrow \frac{1}{2} \| (\vec{x}_i - \vec{x}_j) \times (\vec{x}_i - \vec{x}_k) \|_2$

$M_\ell = \rho_\ell \cdot \text{Area}_\ell$

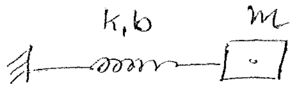
$m_i \mathrel{+}= M_\ell/3;\ m_j \mathrel{+}= M_\ell/3;\ m_k \mathrel{+}= M_\ell/3$

⇒ <u>How to specify damping parameters?</u>

Damping force $f^d = -b n n^T (v_i - v_j)$

A constant b for all springs will look odd, if triangles of many different sizes are present

We can gather some intuition from the 1D case:



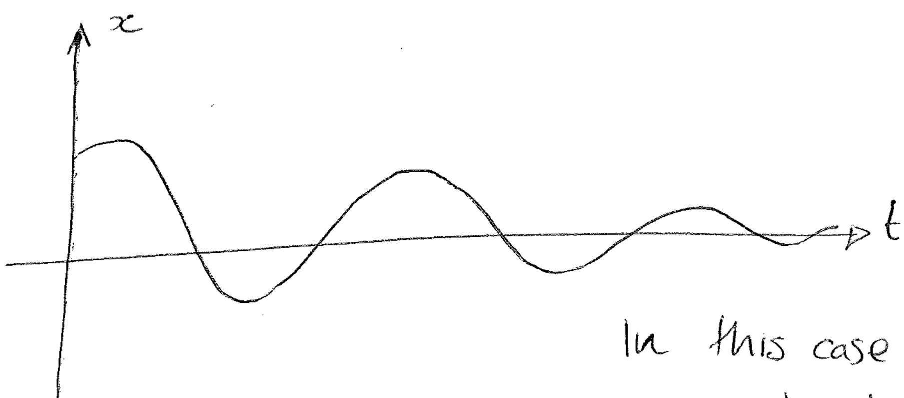
$$f = m x''(t) = -k x(t) - b v(t)$$

$$\Rightarrow \boxed{m x'' + b x' + k x = 0}$$

This O.D.E. can have 3 "types" of solutions:

→ If $b^2 < 4mk$ the solution is <u>oscillatory</u>

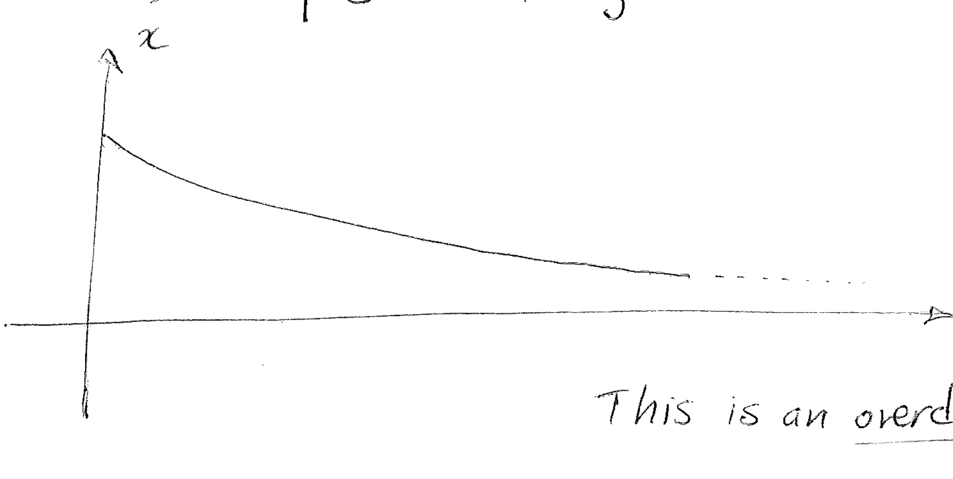
$$x(t) = e^{-|c_1|t} \left\{ c_2 \cos(\omega t) + c_3 \sin(\omega t) \right\}$$



In this case the spring is called <u>underdamped</u>

→ If $b^2 > 4mk$ the solution decays slowly, w/o oscillations:

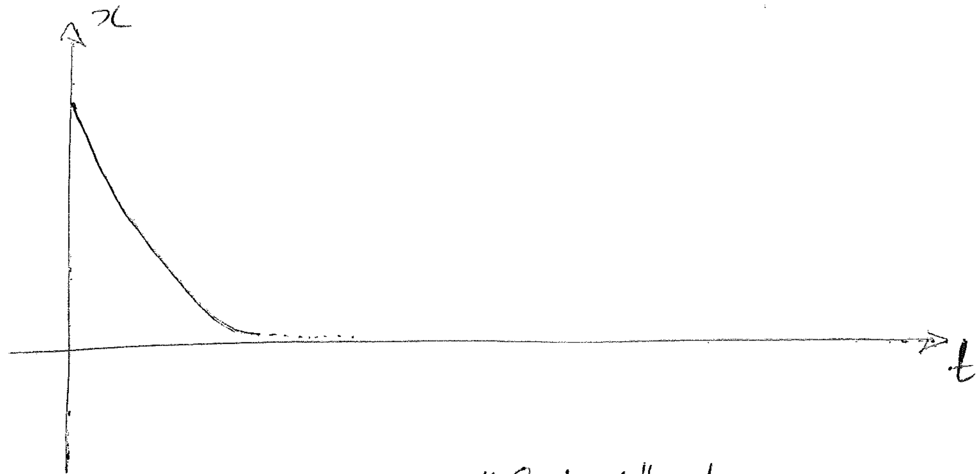
$$x(t) = c_1 e^{-|c_2|t} + c_3 e^{-|c_4|t}$$



This is an <u>overdamped</u> spring.

→ If $b^2 = 4mk$ this is called critical damping

$$x(t) = (c_1 t + c_2)e^{-|c_3|t}$$



⇒ This is the "fastest" decay we can force, without making the motion oscillatory.

⇒ Idea: Define "Critical" $b_0 = 2\sqrt{mk}$

and specify chosen b as $b = \gamma \cdot b_0$

$$\gamma = \begin{cases} 1 & \text{yields critical damping} \\ <1 & \Rightarrow \text{underdamped (oscillatory)} \\ >1 & \Rightarrow \text{overdamped (slow decay)}. \end{cases}$$

For our 3D spring, the ratio k/l_0 plays the role of the "spring constant", so the critical value b_0 is

↳ Young's modulus

$$b_0 = 2\sqrt{\frac{mk}{l_0}}$$

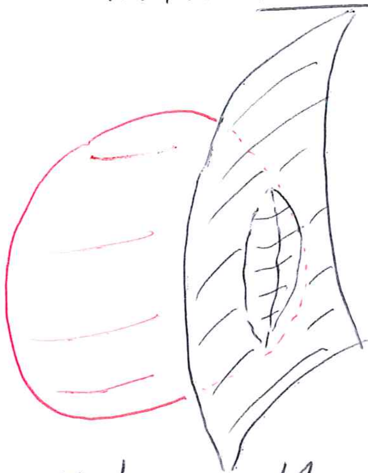
(Note: In practice, we would also have $m \propto l_0$, which is why it may not be so bad to specify b as a "constant")

⇒ Ultimately, specify just the value of γ, across the cloth model.

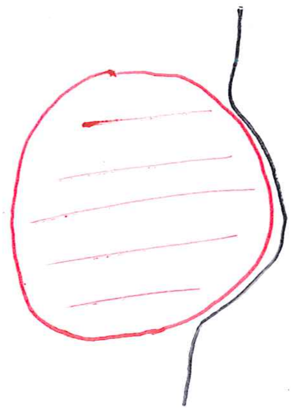
Collisions

Various types, approaches, methods, etc...

⇒ Collision with kinematic objects:

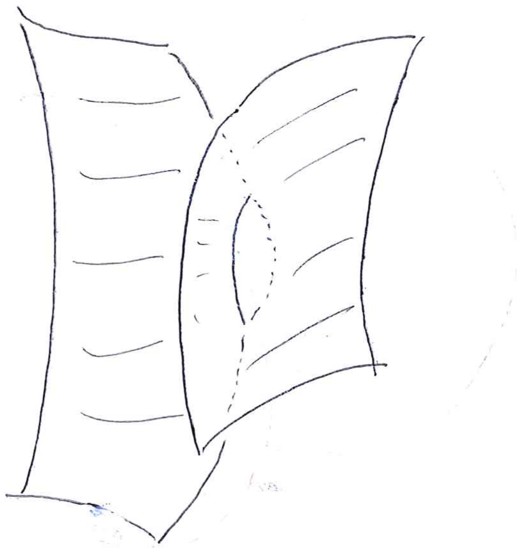


(3D) Cloth collides with kinematic sphere

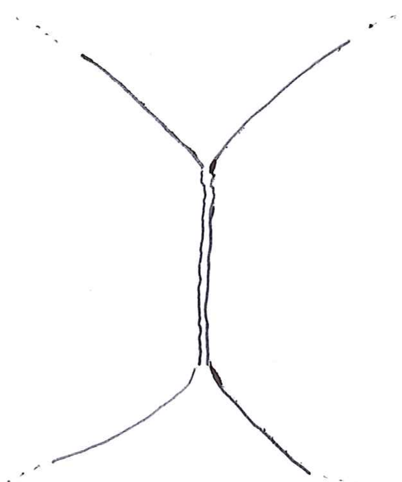


(2D Analogue)

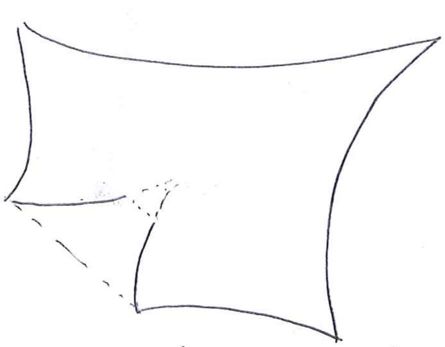
⇒ Self collision



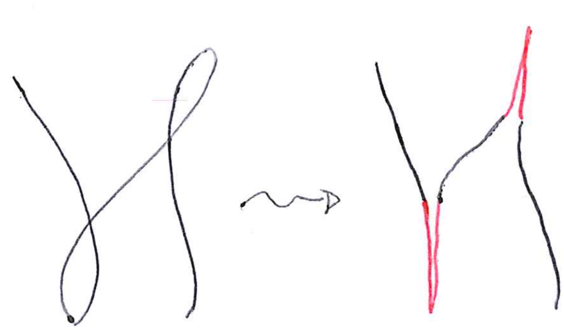
(3D) Two pieces of cloth collide



(2D Analogue)

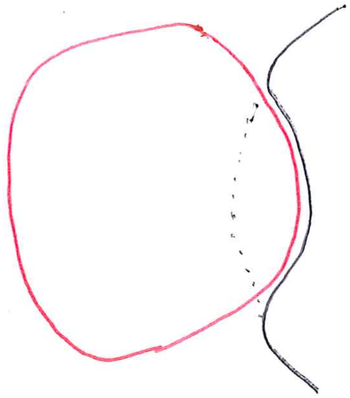


One cloth colliding with itself

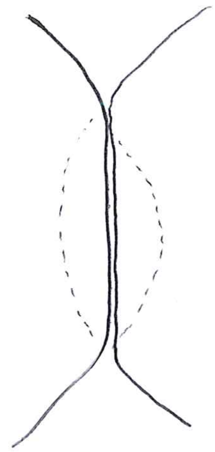


Self-collisions

We will illustrate these aspects in 2D



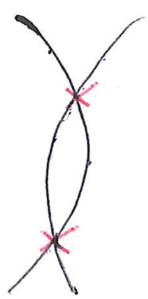
Object collision



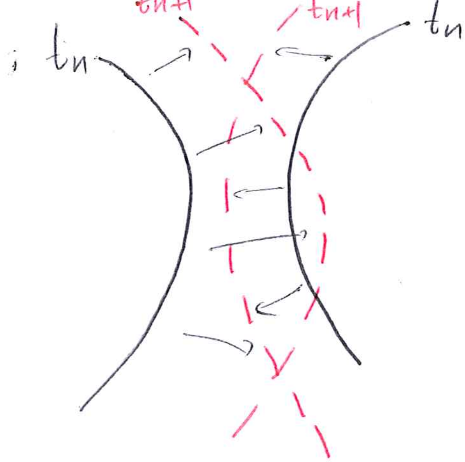
Self collision

Another classification:

⇒ Static collision "Do objects collide now?"



⇒ Dynamic collision (or moving collision)
"At their current trajectories, will objects collide between $t = t_{start}$ & $t = t_{end}$?"



The "typical" stages of collision handling are:

⇒ I. Collision <u>detection</u>

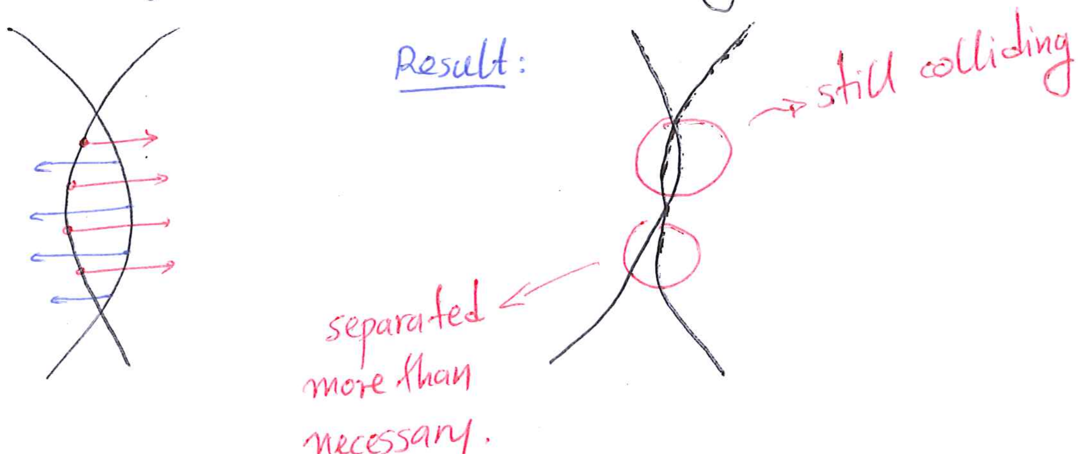
⇒ Determine if a collision is happening <u>now</u>, or if it is imminent (within the next interval dt)

⇒ II. Collision <u>response</u>

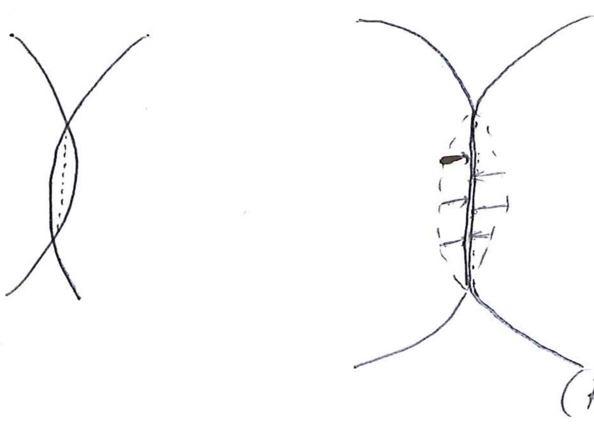
⇒ Try to fix or lessen the collision

↳ General approaches:

1. Penalty methods: Apply repulsive forces that try to push objects "out" of a colliding state



2. Impulse/projection methods: "Instantaneously" fix the collision



<u>Pros</u>: Tight contact
"May" generate collision-free state
<u>Cons</u>: Hard to make <u>always</u> work
(Repulsion is more forgiving).

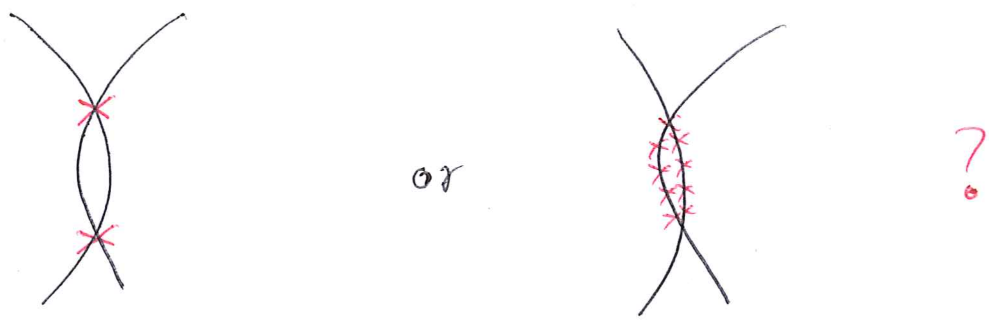
3. Do nothing during simulation, fix as a post-process (w/ some projection method)

Pros : Cheapest

Cons : Unrealistic physics (volume loss, absence of friction, sliding, etc).

Collision DETECTION, in detail:

?? WHAT is colliding?



Continuous vs. discrete detection / handling