Basic Position Based Dynamics

Matthias Müller

www.MatthiasMueller.info



Outline

- Basic Idea
- The solver
- Constraint examples
  - Stretching, bending
  - Volume conservation
  - Fluids
- Solver accelerations



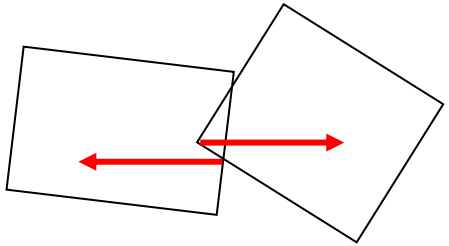
Simulation Requirements in Games

- Cheap to compute
  - 30-60 fps of which physics only gets a small fraction
- Low memory consumption
  - Consoles, fit into graphics (local) memory
- Stable even in non-realistic settings
  - Characters at 100 km/h, 180 degree turns
- Easy to tune and control

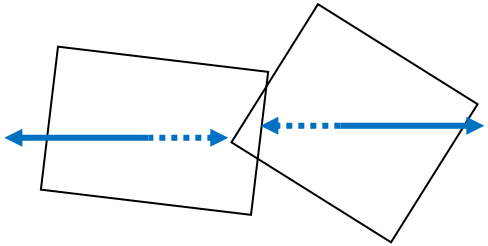


Basic Idea

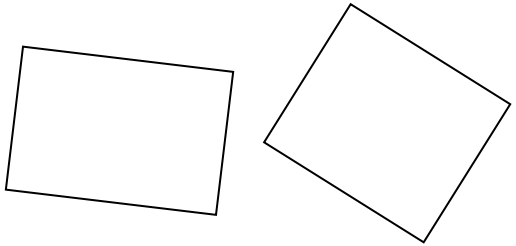
Force Based Update



penetration
causes forces



forces
change velocities

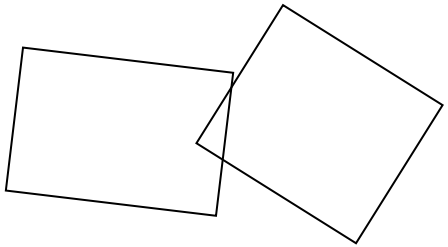


velocities
change positions

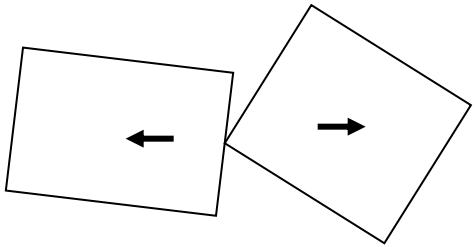
- Reaction lag
- Small spring stiffness → squashy system
- Large spring stiffness → stiff system, overshooting

NVIDIA

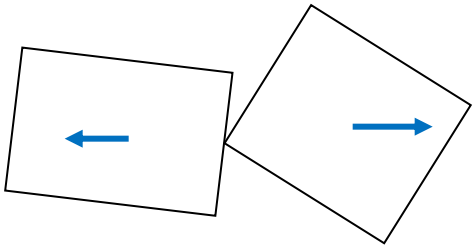
Position Based Update



penetration detection only



move objects so that they do not penetrate



update velocities!

- Controlled position change
- Only as much as needed → no overshooting
- Velocity update needed to get 2nd order system!

NVIDIA

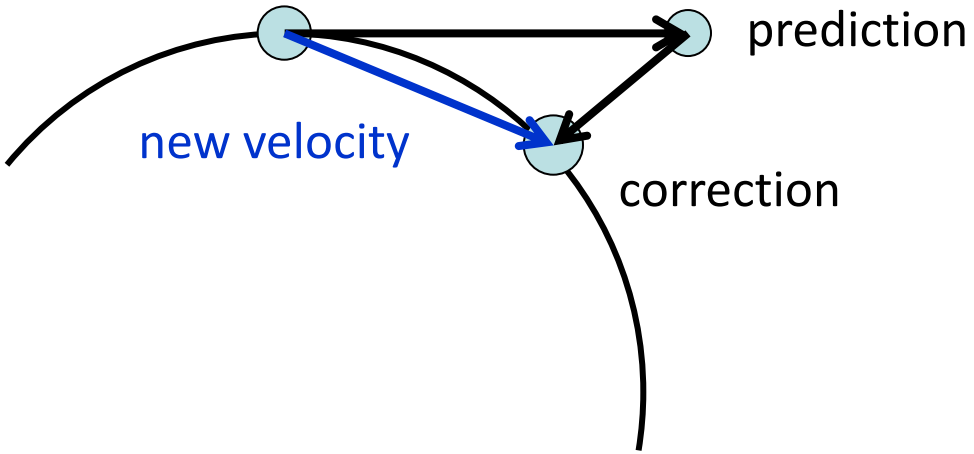
Position Based Integration

init $\mathbf{x}_0, \mathbf{v}_0$		$\mathbf{x}_n, \mathbf{v}_n, \mathbf{p}, \mathbf{u} \in \mathbb{R}^{3N}$
loop		
$\mathbf{p}$	$\leftarrow \mathbf{x}_n + \Delta t \cdot \mathbf{v}_n$	prediction
$\mathbf{x}_{n+1}$	$\leftarrow$ modify $\mathbf{p}$	position correction
$\mathbf{u}$	$\leftarrow (\mathbf{x}_{n+1} - \mathbf{x}_n)/\Delta t$	velocity update
$\mathbf{v}_{n+1}$	$\leftarrow$ modify $\mathbf{u}$	velocity correction
end loop		

NVIDIA

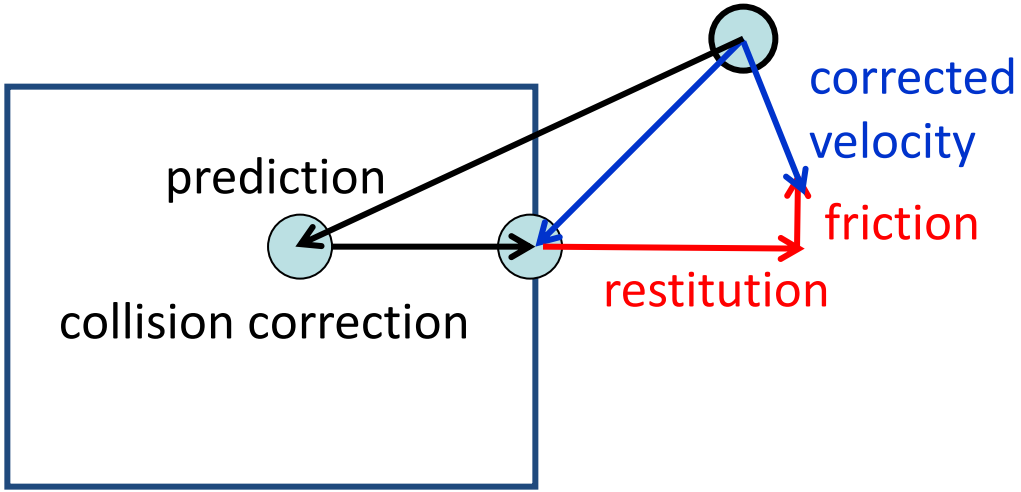
Position Correction

- Example: Particle on circle

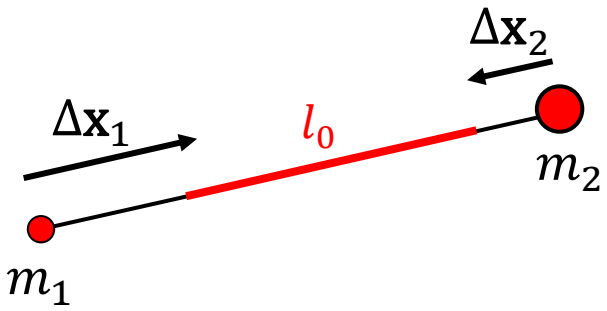


Velocity Correction

- External forces: $\mathbf{v}_{n+1} = \mathbf{u} + \Delta t \, \frac{\mathbf{g}}{m}$
- Internal damping
- Friction
- Restitution



Distance Constraint



$$\Delta\mathbf{x}_1 = -\frac{w_1}{w_1 + w_2}(|\mathbf{x}_1 - \mathbf{x}_2| - l_0)\frac{\mathbf{x}_1 - \mathbf{x}_2}{|\mathbf{x}_1 - \mathbf{x}_2|}$$

$$\Delta\mathbf{x}_2 = +\frac{w_2}{w_1 + w_2}(|\mathbf{x}_1 - \mathbf{x}_2| - l_0)\frac{\mathbf{x}_1 - \mathbf{x}_2}{|\mathbf{x}_1 - \mathbf{x}_2|}$$

$$w_i = \frac{1}{m_i}$$

- Conservation of momentum
- Stiffness: scale corrections by $k \in [0,1]$
  - Easy to tune
  - Effect dependent on time step size and iteration count
  - Often constant in games

NVIDIA

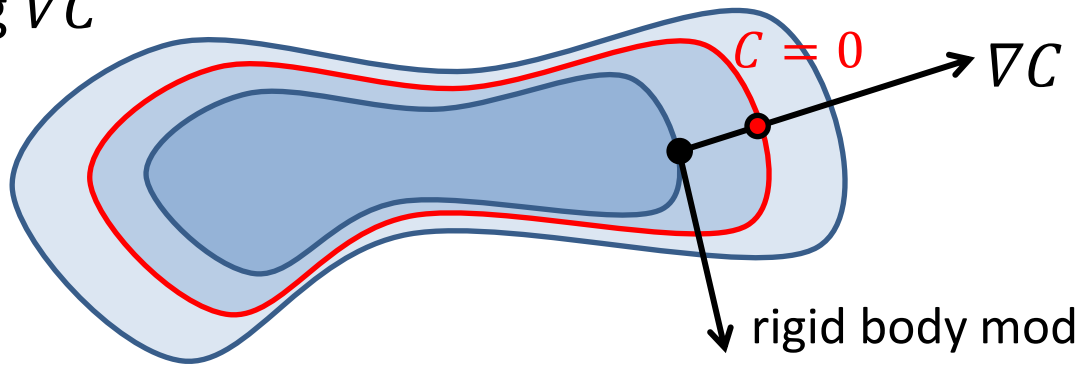
General Internal Constraint

- Define constraint via scalar function:

$$C_{stretch}(\mathbf{x}_1, \mathbf{x}_2) = |\mathbf{x}_1 - \mathbf{x}_2| - l_0$$

$$C_{volume}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) = [(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)] \cdot (\mathbf{x}_4 - \mathbf{x}_1) - 6v_0$$

- Find configuration for which $C = 0$
- Search along ∇C



NVIDIA

Constraint Projection

$$C(\mathbf{x} + \Delta\mathbf{x}) = 0$$

- Linearization (equal for distance constraint)

$$C(\mathbf{x} + \Delta\mathbf{x}) \approx C(\mathbf{x}) + \nabla C(\mathbf{x})^T \Delta\mathbf{x} = 0$$

- Correction vectors

$$\Delta\mathbf{x} = \lambda \nabla C(\mathbf{x})$$

$$\lambda = -\frac{C(\mathbf{x})}{\nabla C(\mathbf{x})^T \nabla C(\mathbf{x})}$$

$$\Delta\mathbf{x} = \lambda \, \mathrm{M}^{-1} \nabla C(\mathbf{x})$$

$$\lambda = -\frac{C(\mathbf{x})}{\nabla C(\boldsymbol{x})^T \mathrm{M}^{-1} \nabla C(\mathbf{x})}$$

$$\mathrm{M} = diag(m_1, m_2, .., m_n)$$



The Solver

Constraint Solver

- Gauss-Seidel
  - Iterate through all constraints and apply projection
  - Perform multiple iterations
  - Simple to implement
  - Atomic operations required for parallelization
- Modified Jacobi
  - Process all constraints in parallel
  - Accumulate corrections
  - After each iteration, average corrections [Bridson et al., 2002]
- Both known for slow convergence



Global Solver

[Goldenthal et al., 2007]

- Constraint vector

$$\mathrm{C}(\mathbf{x}) = \begin{bmatrix} C_1(\mathbf{x}) \\ \cdots \\ C_M(\mathbf{x}) \end{bmatrix} \qquad \nabla\mathrm{C}(\mathbf{x}) = \begin{bmatrix} \nabla C_1(\mathbf{x})^T \\ \cdots \\ \nabla C_M(\mathbf{x})^T \end{bmatrix} \qquad \boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \cdots \\ \lambda_M \end{bmatrix}$$

$$\Delta\mathbf{x} = \mathrm{M}^{-1}\nabla C(\mathbf{x})\,\lambda \qquad \lambda = -\frac{C(\mathbf{x})}{\nabla C(\boldsymbol{x})^T \mathrm{M}^{-1}\,\nabla C(\mathbf{x})}$$

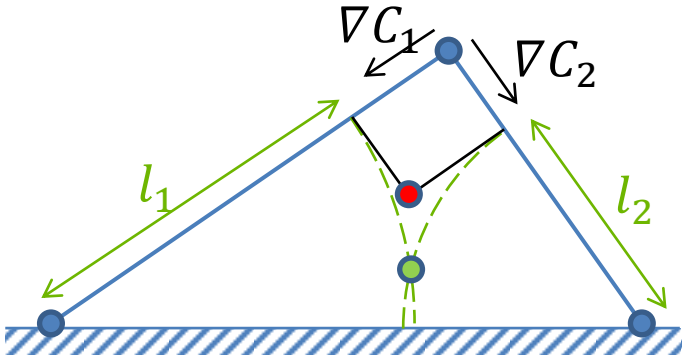
$$\Delta\mathbf{x} = \mathrm{M}^{-1}\nabla\mathrm{C}(\mathbf{x})^T\boldsymbol{\lambda}$$

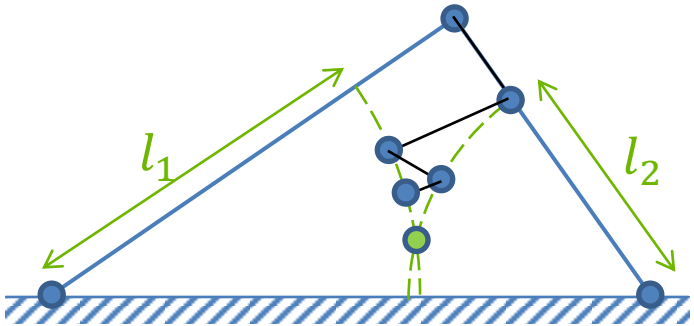
$$[\nabla C(\mathbf{x})\mathrm{M}^{-1}\nabla\mathrm{C}(\mathbf{x})^T]\,\boldsymbol{\lambda} = -\mathrm{C}(\mathbf{x})$$

NVIDIA

Global vs. Gauss-Seidel

- Gradients fixed
- Linear solution ≠ true solution
- Multiple Newton steps necessary
- Current gradients at each constraint projection
- Solver converges to the true solution





NVIDIA

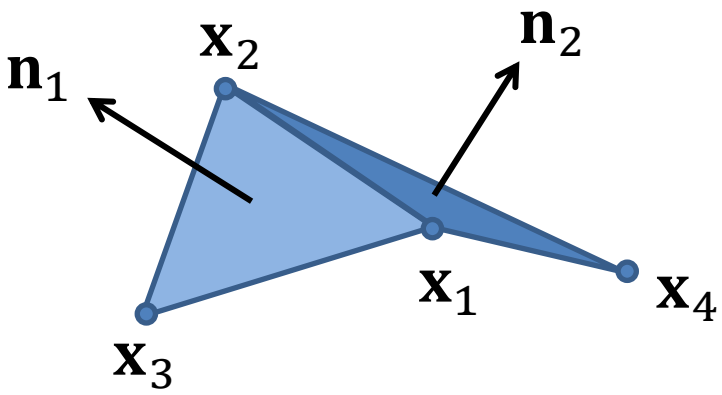
Gauss-Seidel Doesn't Suck!

- Cons:
  - Slow convergence
  - Speed up with specialized methods (hierarchical, LRA, shape matching, ..)
- Amazing list of pros:
  - Can handle unilateral constraints ($C \geq 0$)!
    - Fluids: non-sucking boundary conditions [Chentanez at al., 2012]
    - Rigid bodies: LCP [Tonge et al., 2012]
    - Deformable objects: LRA [Kim et al., 2012]
  - Works on non-linear problem directly
  - GS + PBD: garbage in, simulation out (almost ☺)
  - Fine grained interleaved solver trivial
  - Easy to implement and parallelize



Constraint Examples

Bending

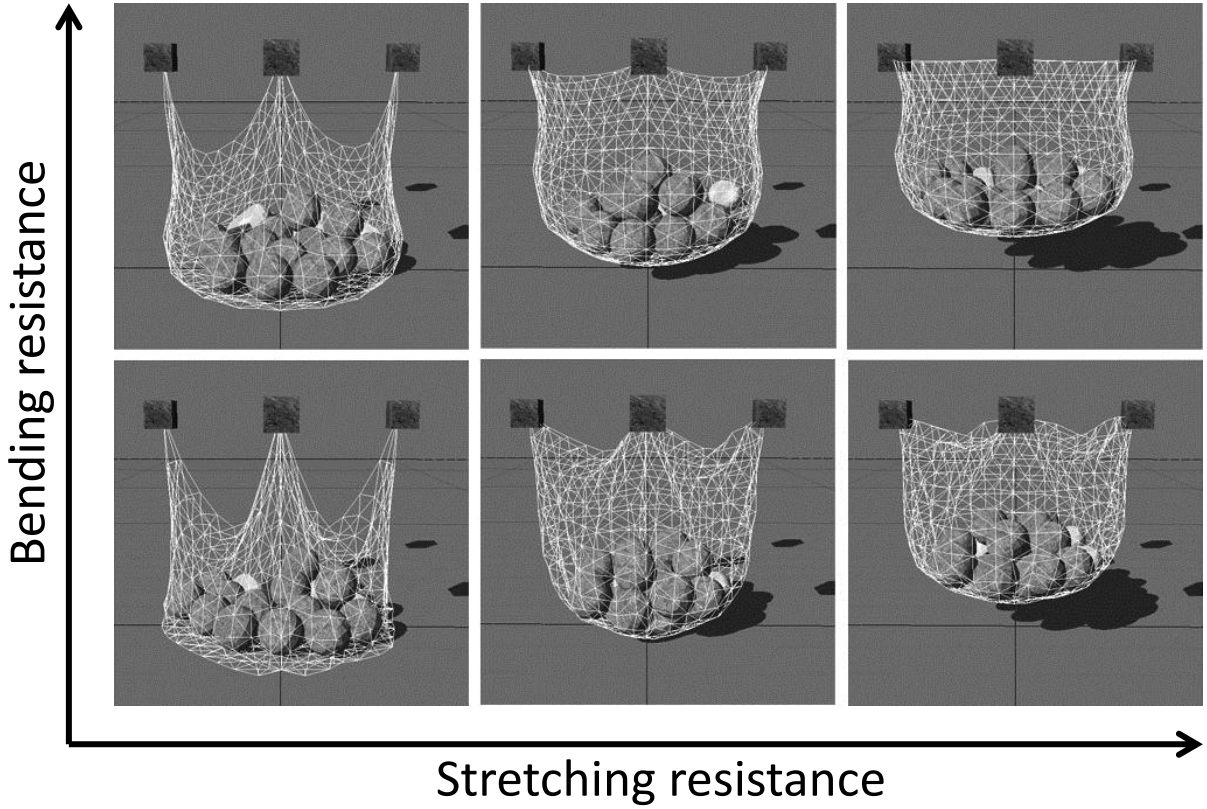


$$C_{bending}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x_3}, \mathbf{x_4}) = acos\left(\frac{(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)}{|(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)|} \cdot \frac{(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_4 - \mathbf{x}_1)}{|(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_4 - \mathbf{x}_1)|}\right) - \varphi_0$$

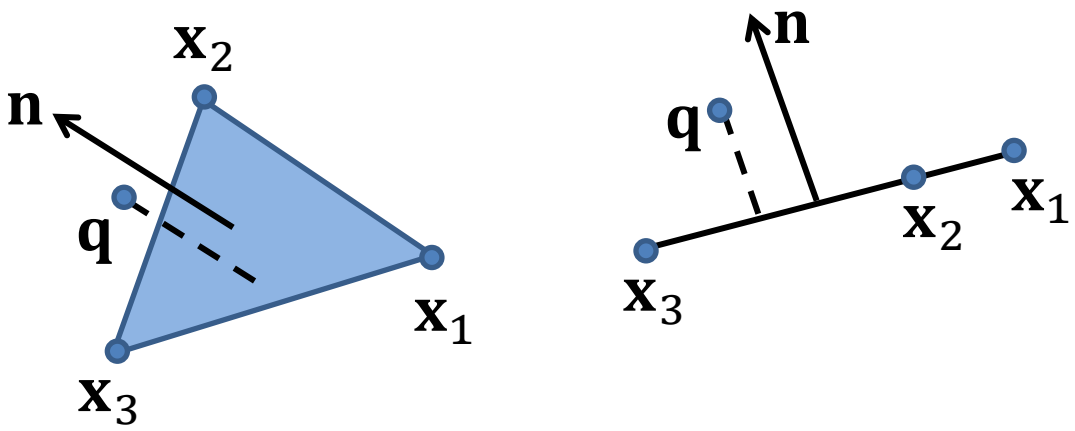
- More expensive than constraint $C_{stretch}(\mathbf{x}_3, \mathbf{x}_4)$
- But: Orthogonal to stretching

NVIDIA

Stretching – Bending Independence



Triangle Collision



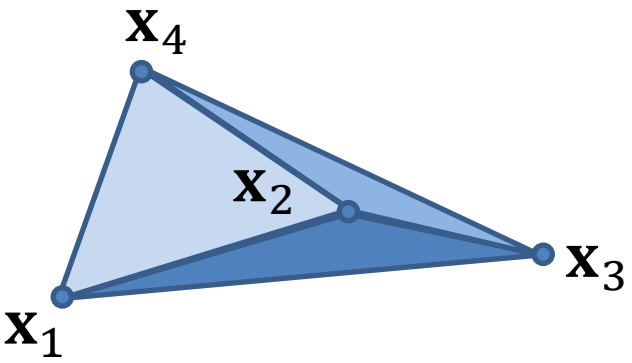
$$C_{coll}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x_3}, \mathbf{x_4}) = \quad (\mathbf{q} - \mathbf{x}_1) \cdot \frac{(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)}{|(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)|} - h$$

NVIDIA

Cloth Example



Tetra Volume



$$C_{air}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) = det[\mathbf{x}_2 - \mathbf{x}_1, \mathbf{x}_3 - \mathbf{x}_1, \mathbf{x}_4 - \mathbf{x}_1] - 6V_0$$

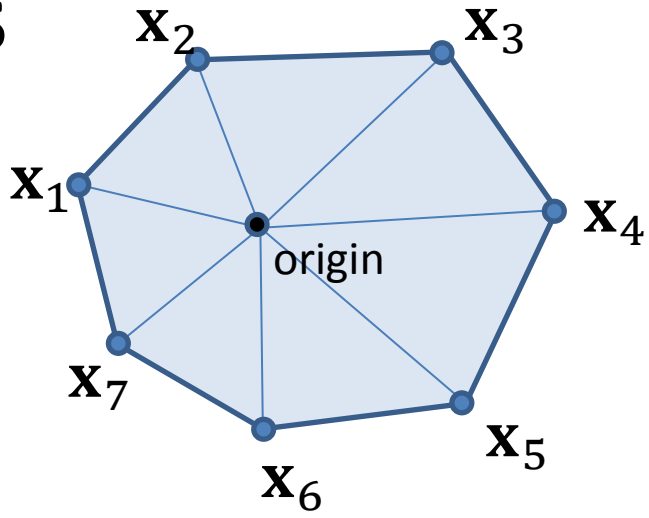
NVIDIA

Soft Body Example



NVIDIA

Global Volume - Balloons



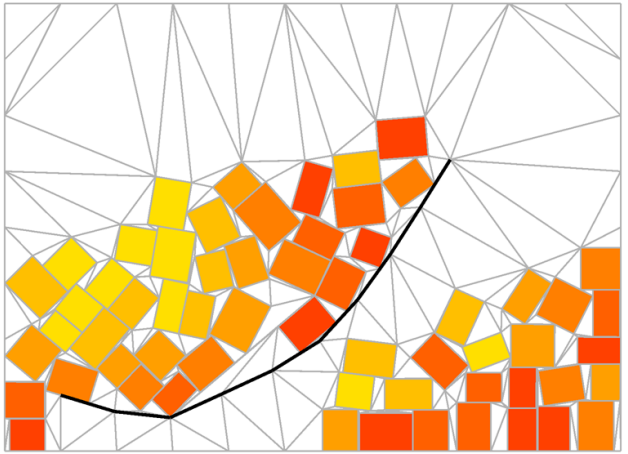
$$C_{balloon}(\mathbf{x}_1, \ldots, \mathbf{x}_N) =$$

$$\frac{1}{6}\left(\sum_{i=1}^{n_{triangles}} \left(\mathbf{x}_{t_1^i} \times \mathbf{x}_{t_2^i}\right) \cdot \mathbf{x}_{t_3^i}\right) - k_{pressure} V_0$$



NVIDIA

Air Meshes

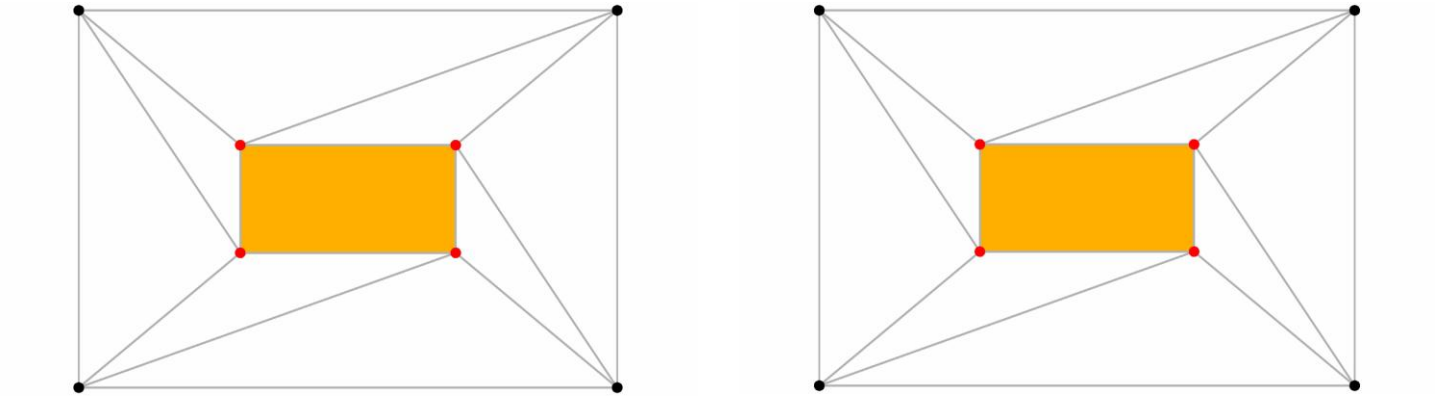


- Triangulate air
- Prevent volume from inverting
- Add one unilateral constraint per cell:

$$C_{air}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) = |(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)| \geq 0$$

NVIDIA

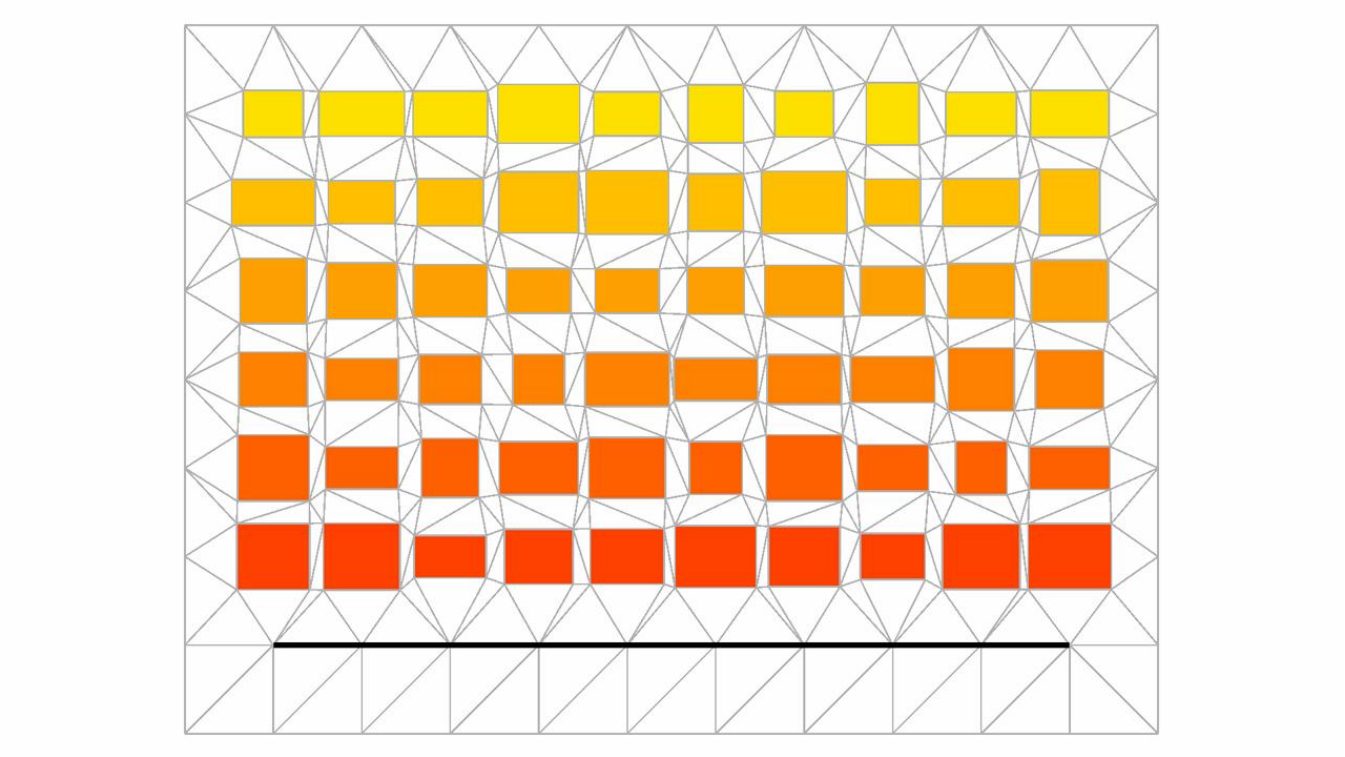
Locking



- Elements can invert without collisions
- Solution: Mesh optimization (edge flips)

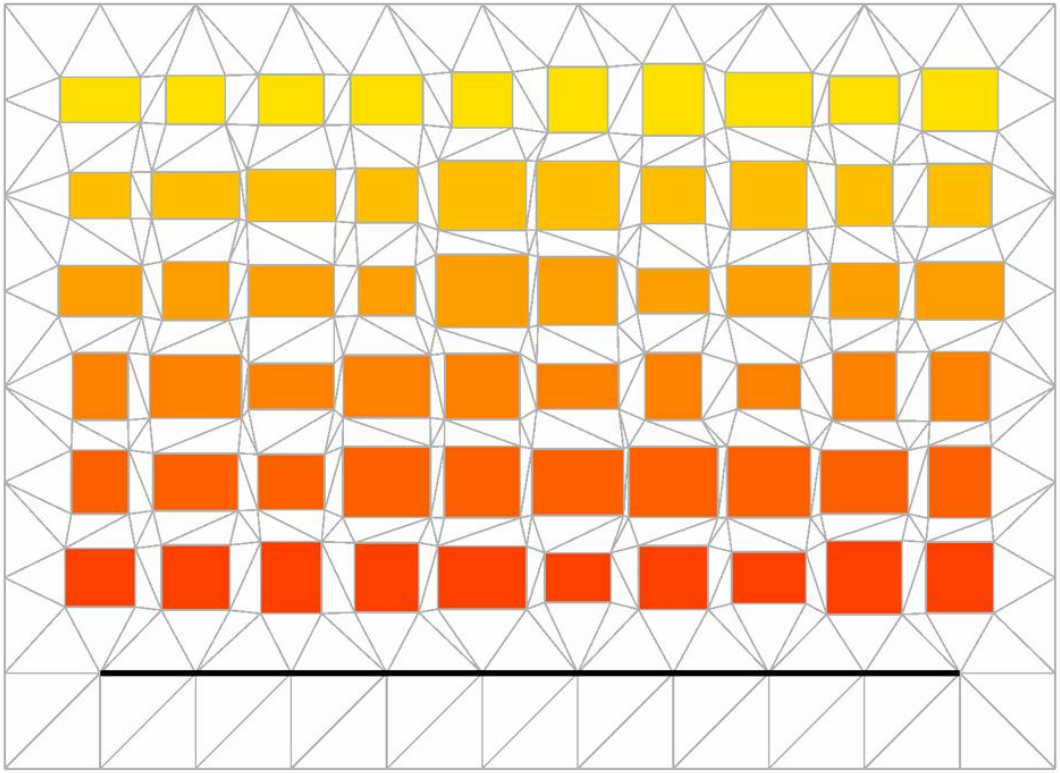
NVIDIA.

2D Boxes



NVIDIA

Boxes Recovery

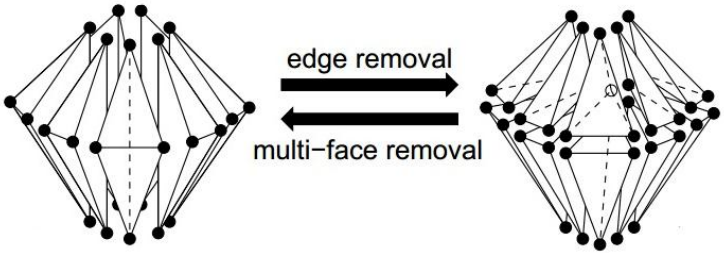


3D Air Meshes

- Per tetra unilateral constraint:

$$C_{air}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) = det[\mathbf{x}_2 - \mathbf{x}_1, \mathbf{x}_3 - \mathbf{x}_1, \mathbf{x}_4 - \mathbf{x}_1] \geq 0$$

- Mesh optimization more expensive!



3D Air Meshes

- Two cases that work well without optimization





- Multi-layered clothing
- Tissue collision
- No large relative translations / rotations

NVIDIA.

Multi-Layered Clothing



NVIDIA

Untangling



Tissue Collision Handling



Position Based Fluids

[Macklin et al. 2013]

- Particle based
- Pair-wise lower distance constraints
  $\rightarrow$ granular behavior
- Move particles in local neighborhood such that density = rest density
- Density constraint

$$C(\mathbf{x}_1, .., \mathbf{x}_n) = \rho_{SPH}(\mathbf{x}_1, .., \mathbf{x}_n) - \rho_0$$

NVIDIA.

Position Based Fluids

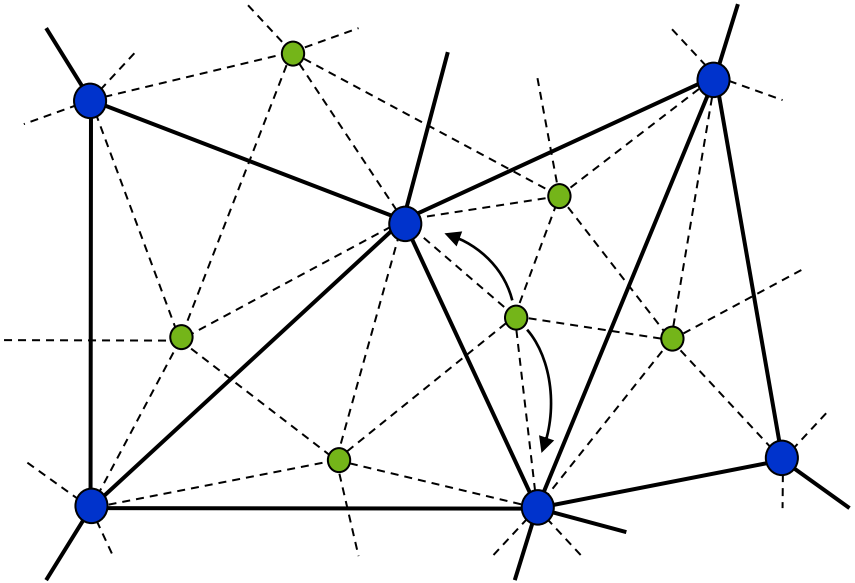


NVIDIA

Solver Accelerations

Hierarchical PBD

- Create hierarchical mesh

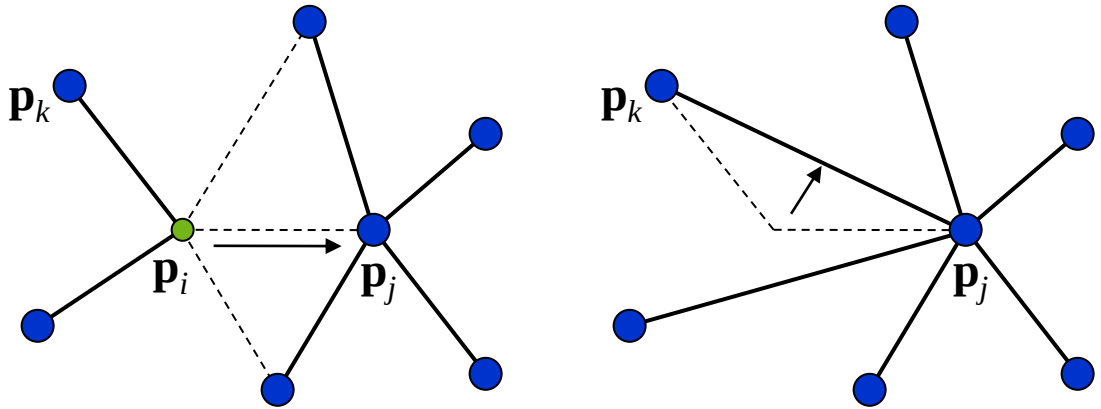


- Next coarser mesh:
  - Subset of vertices
  - Each fine vertex is connected to at least k coarse vertices

NVIDIA.

Hierarchical Constraints

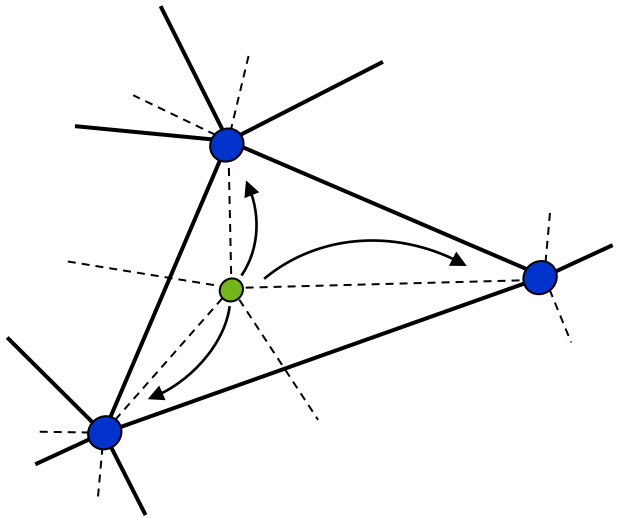
- Constraints on coarse meshes



- Unilateral, upper bounds!

NVIDIA

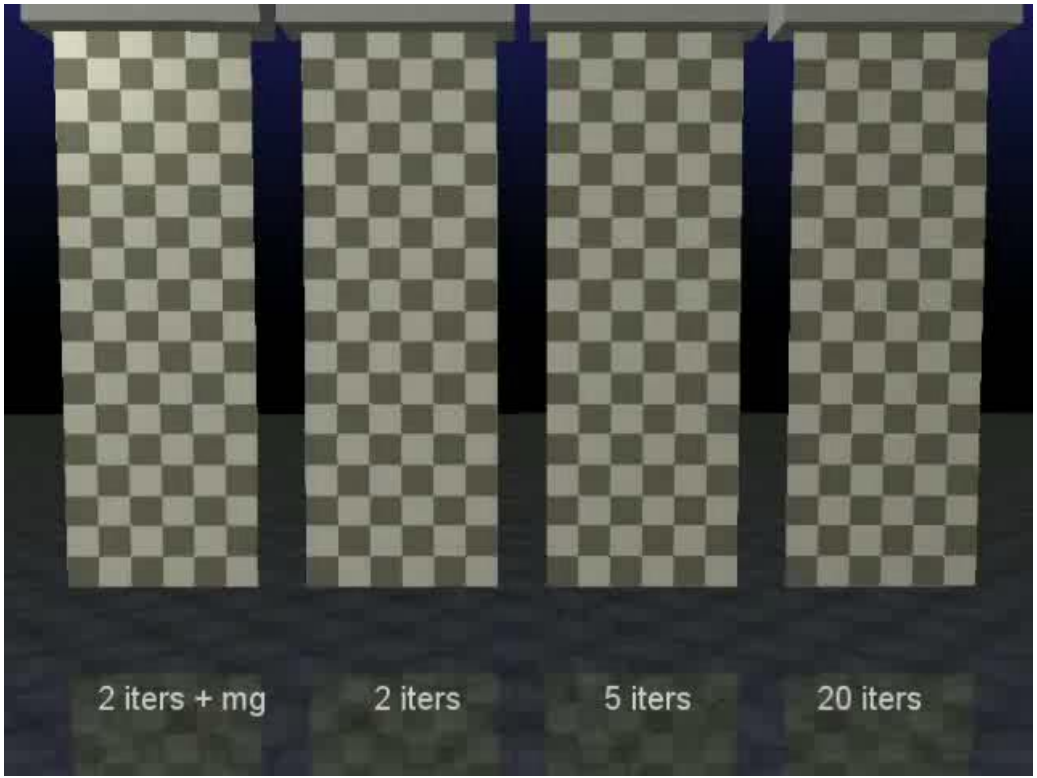
Hierarchical Solver



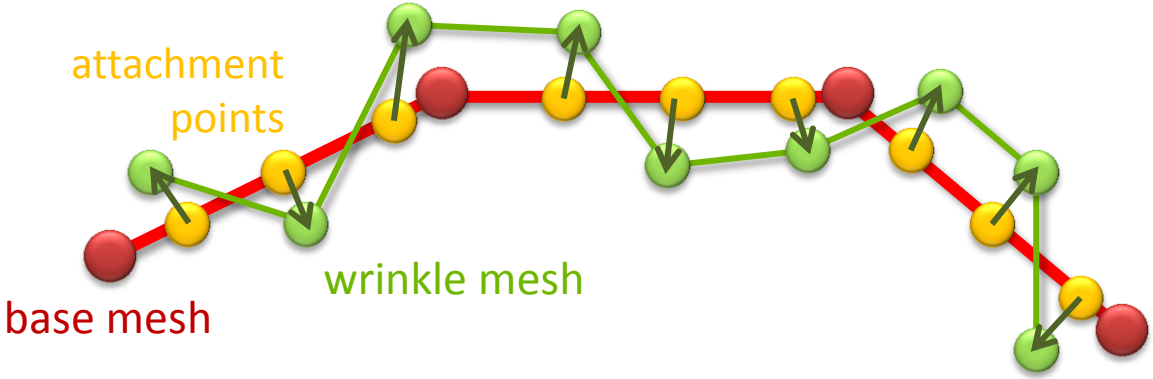
- Solve coarse → fine
- Interpolate displacements from next coarser level

NVIDIA.

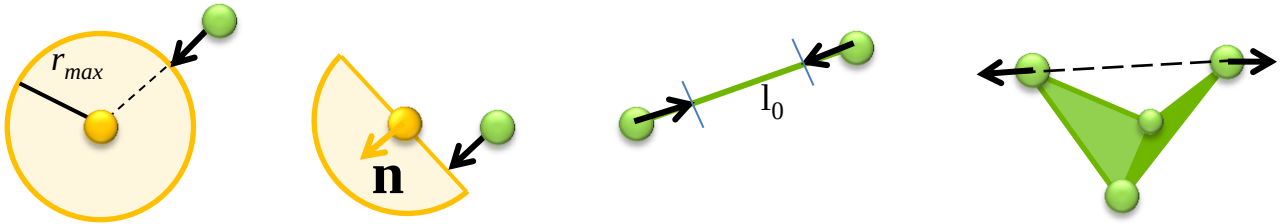
Hierarchical PBD



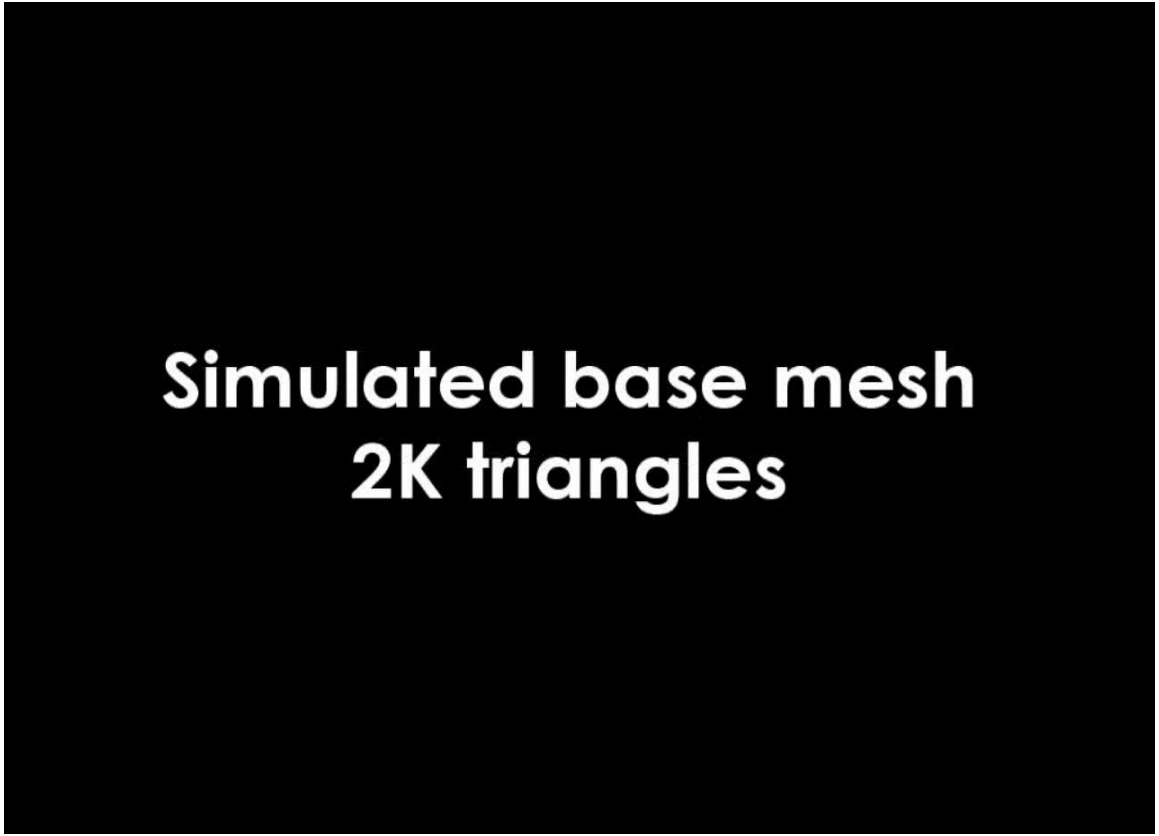
Wrinkle Meshes



- 4 constraint types, geometric projection

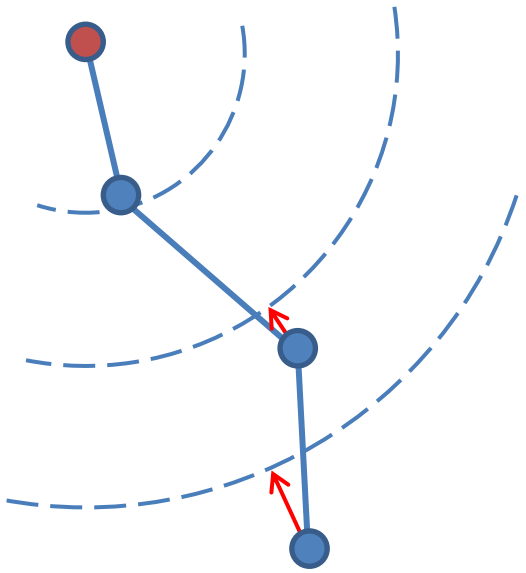


Wrinkle Meshes



Long Range Attachments (LRA)

- Very often cloth is attached (curtain, flags, clothing)
- Upper distance constraint to closest attachment point
- Only radial stretch resistance



Long Range Attachments (LRA)

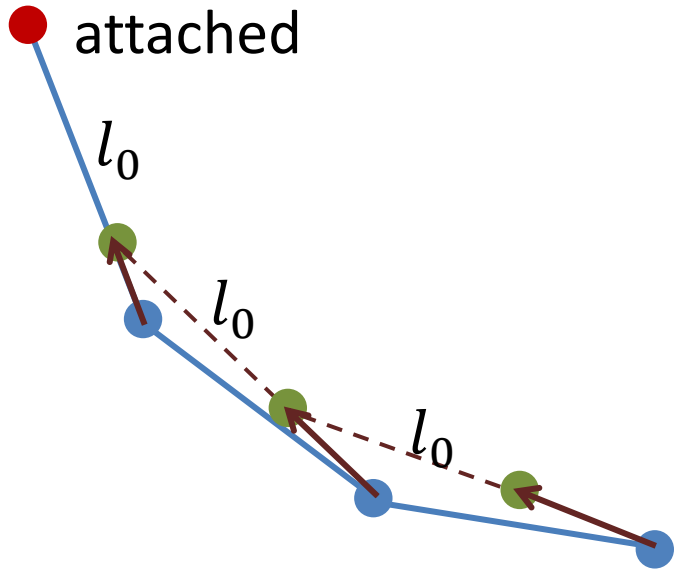


[Kim et al., 2012], 90k particles

NVIDIA

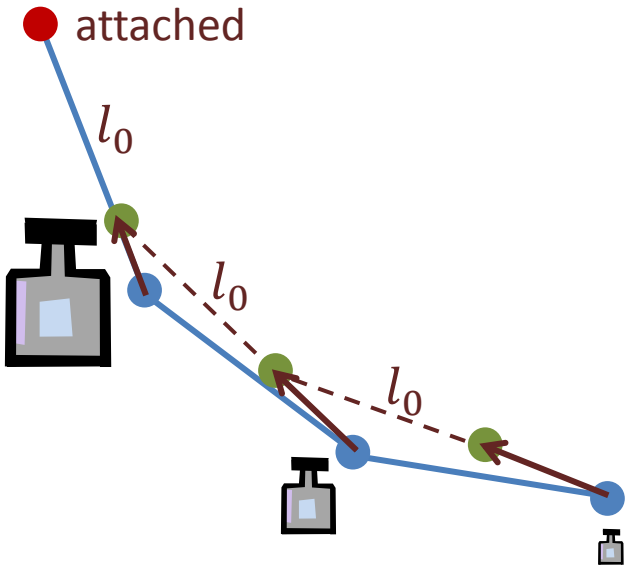
Follow The Leader (FTL)

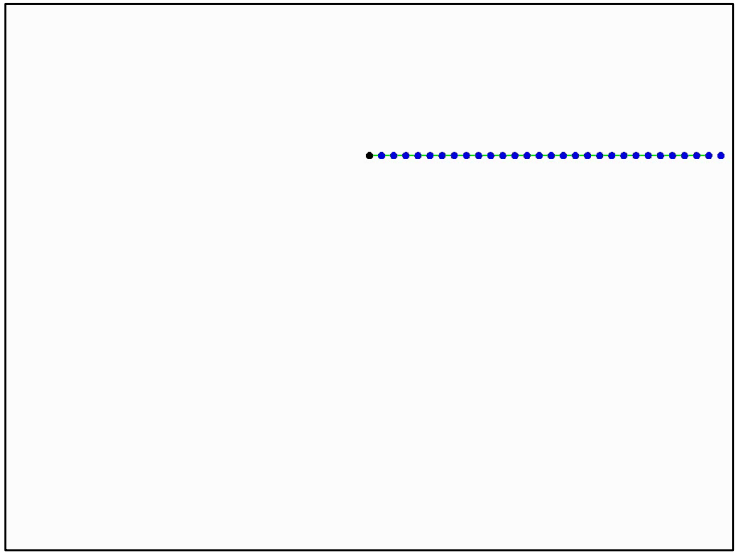
- From top to bottom
- Only move lower particle
- All constraints satisfied!



Follow The Leader (FTL)

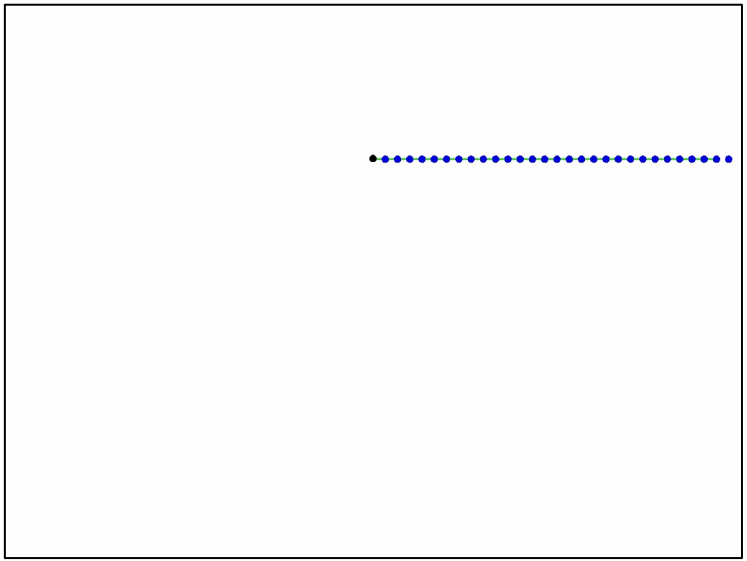
- Momentum not conserved!





Dynamic Follow The Leader (DFTL)

- Update positions one-sided
- Update velocities symmetrically



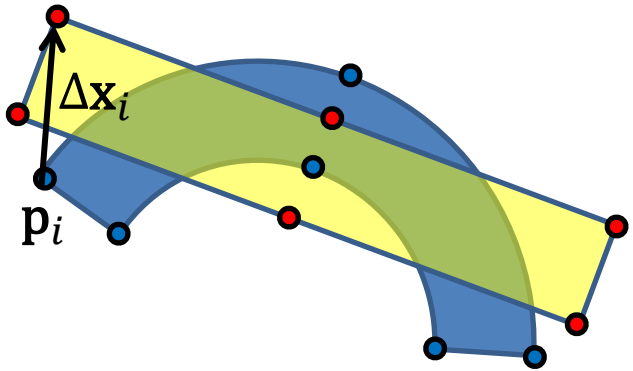
Fur Demo



Left Mouse – Blow
Right Mouse – Rotate Camera
3 Lights
8xMSAA
Supersampling
Shadow Opacity
Shadow Ambient
Per Hair Shadow Variation
Hair Opacity
Hair Base Thickness
Hair Tip Thickness
Hair Diffuse Color Scale
Hair Specular Color Scale
Hair Specularity
Fluffiness
Wind X
Wind Y
Wind Z
Blower Speed
Blower Radius
Animate (A)
Simulate (S)
Slim Windy (1)
Normal Windy (2)
Fluffy Windy (3)
Slim No Wind (4)
Normal No Wind (5)
Fluffy No Wind (6)

Rigid Objects

- Optimally match un-deformed with deformed shape
- Only allow translation and rotation



- Global correction, no propagation needed
- No mesh needed! → See Jan's part

NVIDIA

Questions?