Physics in Games

Matthias Müller

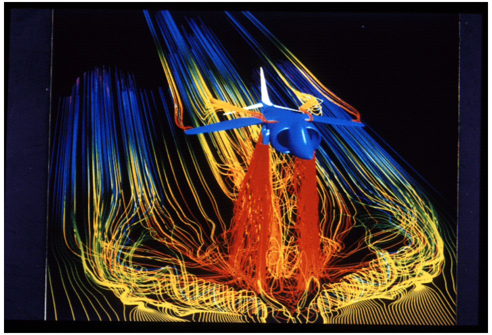
www.MatthiasMueller.info

NVIDIA

Outline

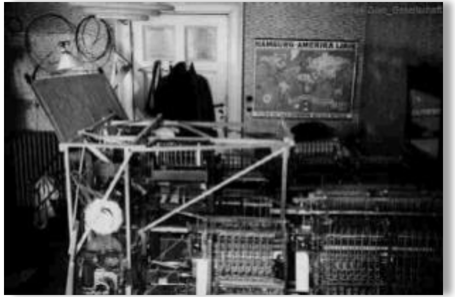
- Comparison
  - Physical simulations in engineering
  - Offline physics in graphics (mostly movies)
  - Interactive physics
  - Real time physics in games
- Position Based Dynamics
  - Algorithm
  - Examples: cloth, rigid bodies, fluids, unified solver
- Q & A

Simulations in Engineering



- Complement real experiments
- Extreme conditions, spatial scale, time scale
- Accuracy most important factor
- Low accuracy: Useless result!
- One central gigantic computer

Evolution of Compute Power

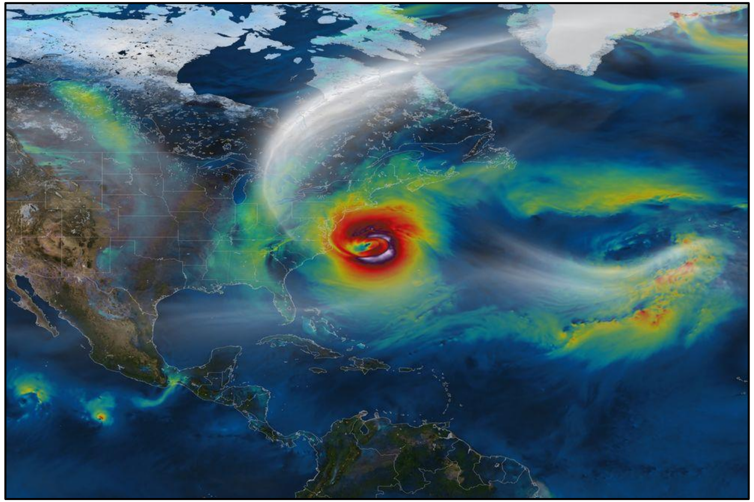


Zuse's Z1 (1938)
0.2 ops

Titan (currently number 2)
using 18,000 nvidia GPUs

~27,000,000,000,000,000 flops!

Simulation of Hurricane Sandy

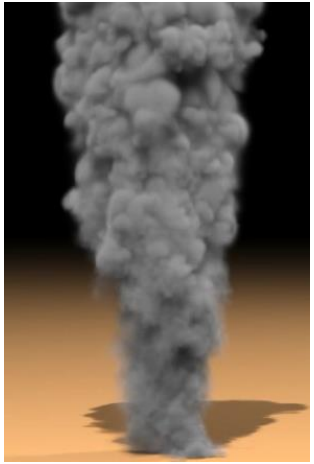


- National Center for Supercomputing Applications
- 9120 x 9216 x 48 cells (500 m)
- 13,680 nodes and 437,760 cores on Titan
- Sustained rate of 285 teraflops

Physics in Graphics

Re-inventing the Wheel?

- Since late 80's [Terzopoulos et al. 87, 88]
- Rediscoveries
  - Semi-Lagrangian advection, co-rotational FEM, X introduced Y to graphics (SPH, MPM, FLIP, …)
- Goals of physics in graphics
  - Imitation of physical phenomena / effects
  - Plausible behavior (cheating possible)
  - Trade accuracy for speed, stability, simplicity
  - Control (by director / game developer)
- New goals require new methods!



Offline Methods

[Emmerich, Movie 2012]

- Main application: Movies
- >> 1 sec of computation for 1 sec of simulation allows:
  - High resolution (fluid grid, FEM mesh, time steps)
  - Re-runs and adaptive time steps
  - Time consuming shading

Interactive Physics

- Between offline and game physics
- Virtual surgery, virtual reality, demos
- All available compute power
- > 15 fps
- No adaptive time steps
- Robust
  - No re-runs
  - Unforeseeable situations

Water Demo (GTC 2012)

- First time real-time Eulerian water sim + ray tracing

2 x GTX 680

Multi-grid
[Chentanez et al., 2011]

OptiX

Dragon

- Eulerian fluid simulation + combustion model + volumetric rendering

Physics in Games

Game Requirements

- Cheap to compute
  - 30-60 fps of which physics only gets a small fraction
- Low memory consumption
  - Consoles, fit into graphics (local) memory
- Stable in extreme settings
  - 180 degree turns in one time step
- High level of control

- Challenge
  - Meet all these constraints
  - Get to offline results as close as possible

Speedup Tricks

- Reduce simulation resolution
  - Simple: Use same algorithms
  - Interesting details disappear

- Reduce dimension (e.g. 3d → 2d)
- Use different resolution for physics and appearance
- Simulate only in active regions (sleeping)
- Camera dependent level of detail (LOD)
- Invent new simulation methods!

- Use nvidia GPUs and CUDA! ☺

NVIDIA

Game Physics Methods

Animation

- Pros:
  - Can be and still is used for almost everything (3d movie playback)
  - Full control
  - What artists are used to do
- Cons
  - Time consuming manual work
  - Hard to handle complex phenomena
  - Repeating behavior

Particle Physics

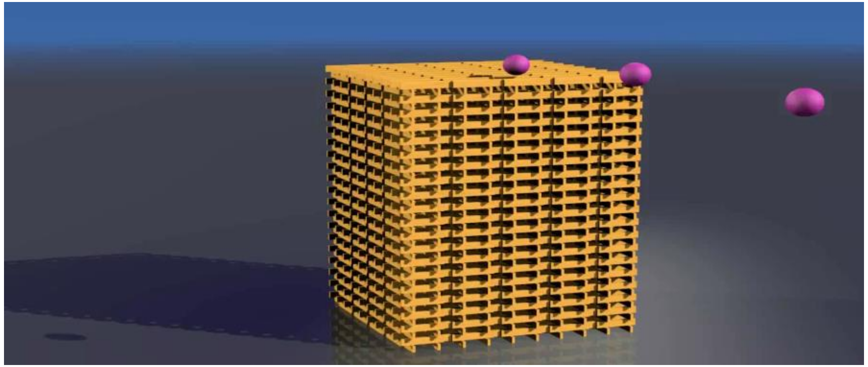
- Simplest and very popular form of physics effect
  - droplets, smoke, fire, debris [Reeves, 1983]
- Effects physics vs. game play physics
  - does not influence game play, no path blocking
- Most expensive part:
  - collision detection with large environments
  - particle-particle interaction (often not needed)
  - Advection by incompressible velocity field (fluid solver)

NVIDIA


PhysX
by NVIDIA

Rigid Bodies

- Game physics engines = rigid body engines
- Challenges
  - Stability (stacking)
  - Speed (solver and collision detection)
  - Continuous collision detection (fast moving objects)
- Rarely in-house
- Middleware popular (*PhysX, havok, bullet*)



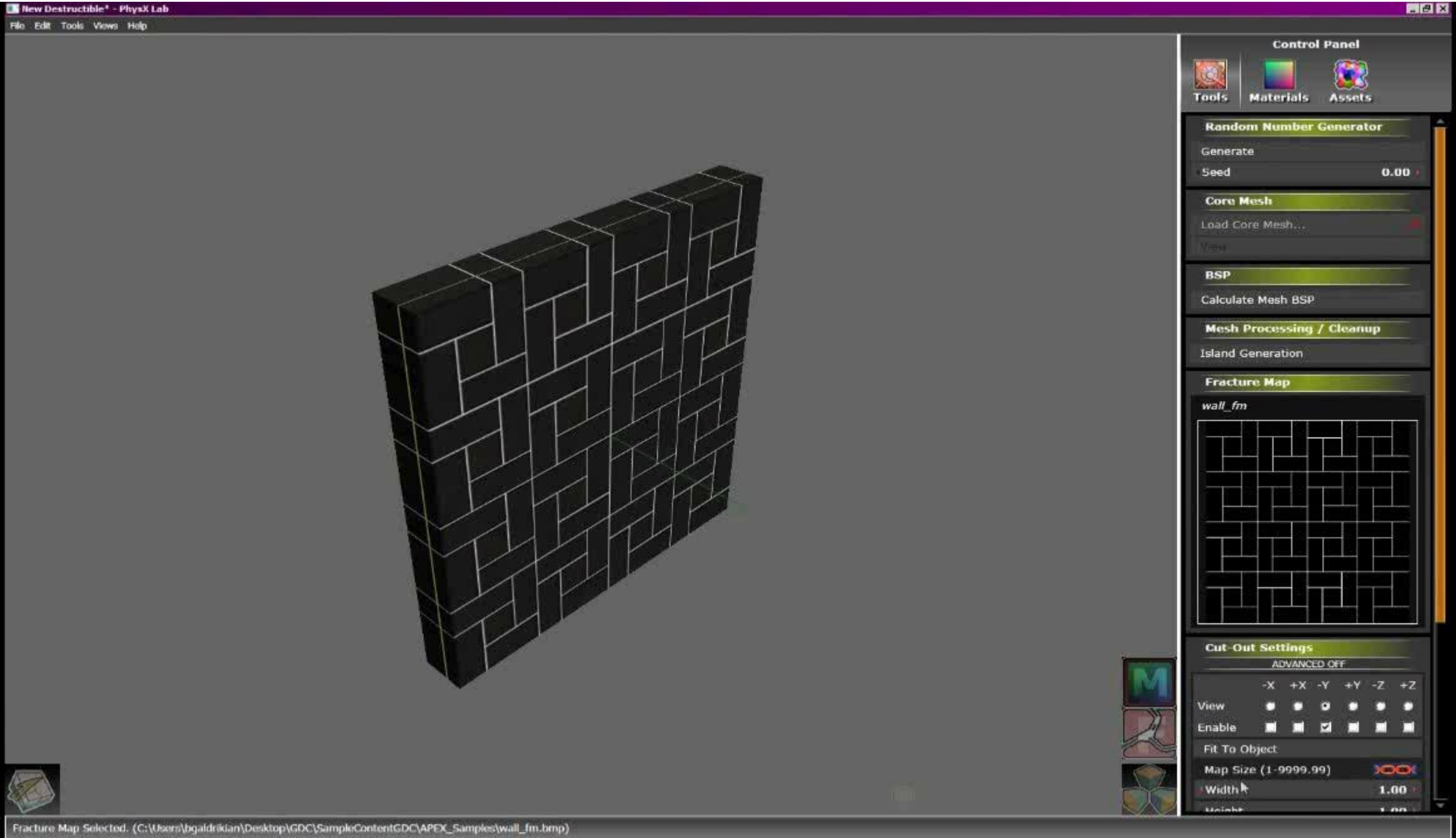
inthevif with blender & bullet

Destruction

- Traditional: static fracture
- Artists pre-fracture models
- Models are replaced by parts
  when collision forces exceed a threshold

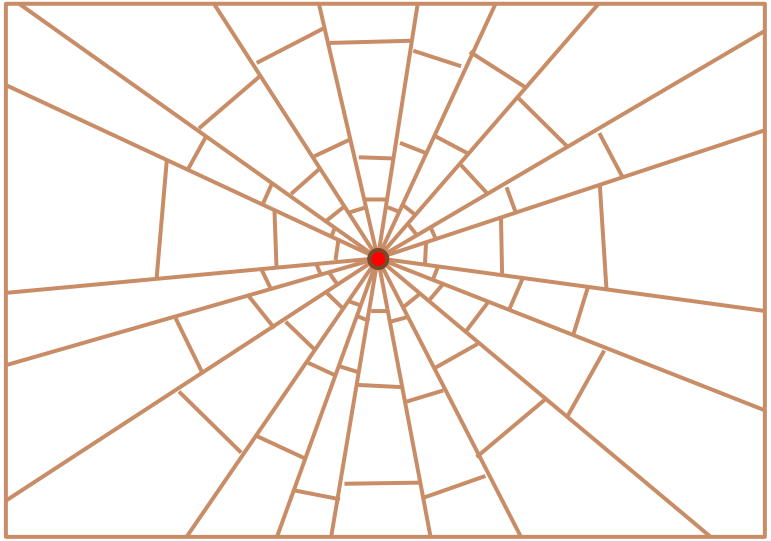
- Pro:
  - High level of control
- Cons:
  - Tedious manual work
  - Independent of impact location

PhysX Destruction Tool



Pattern Based Fracture

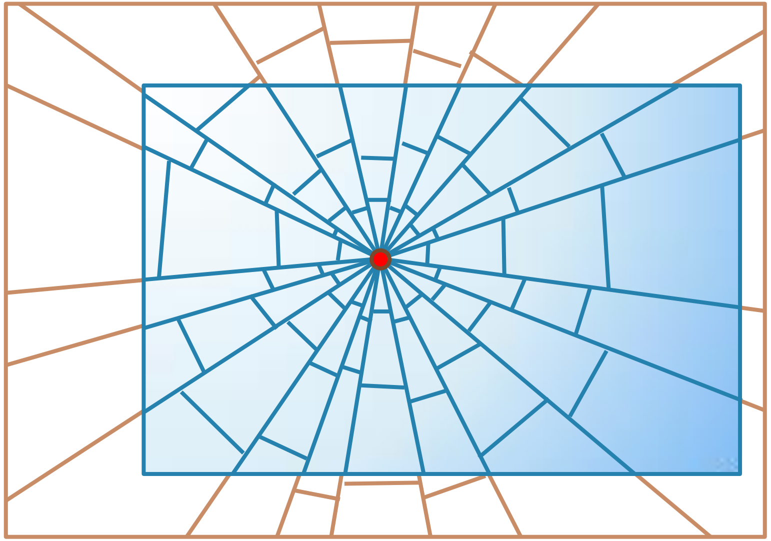
[Müller et al., 2013]



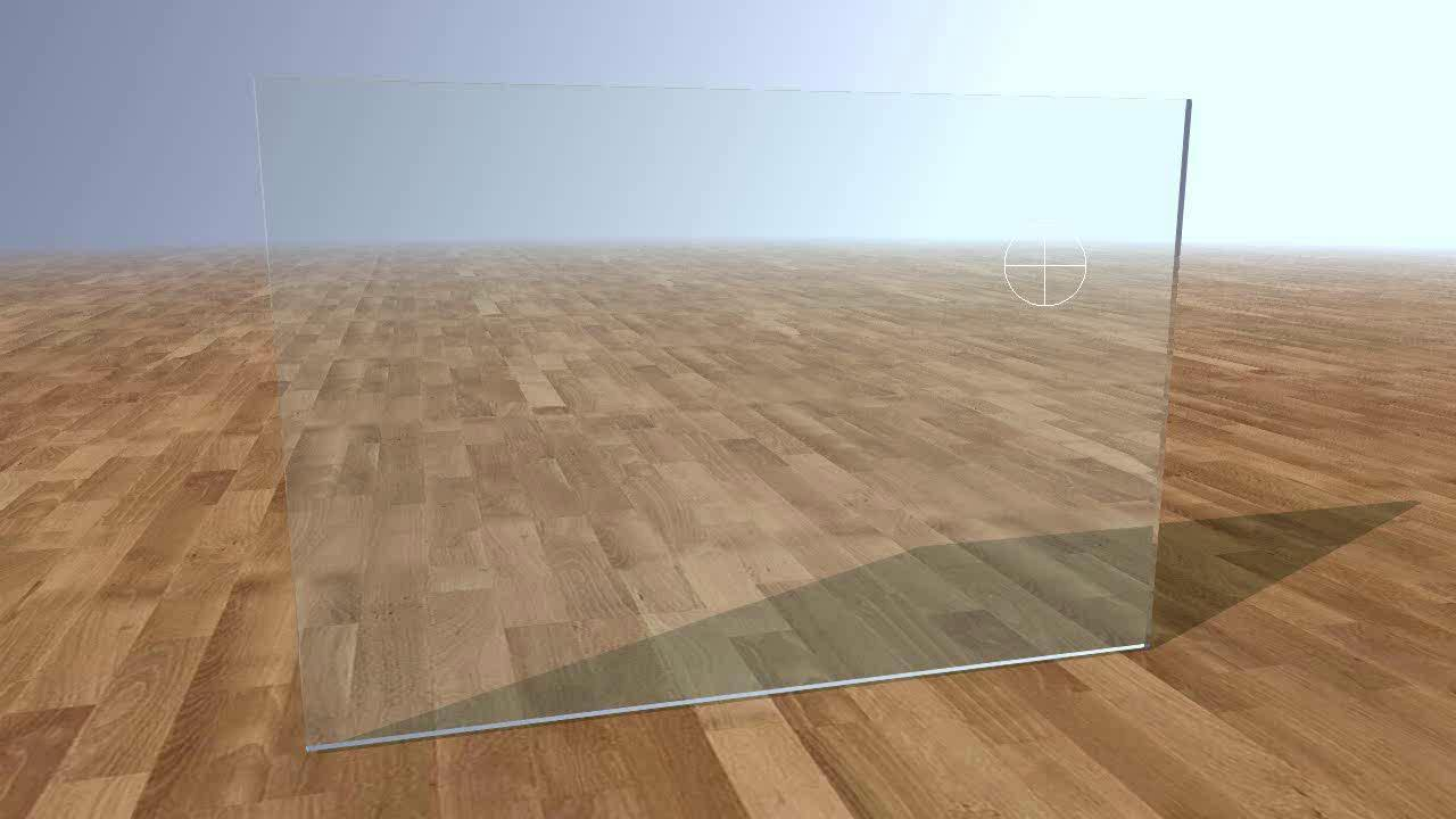
- Pre-designed fracture pattern

Pattern Based Fracture

[Müller et al., 2013]

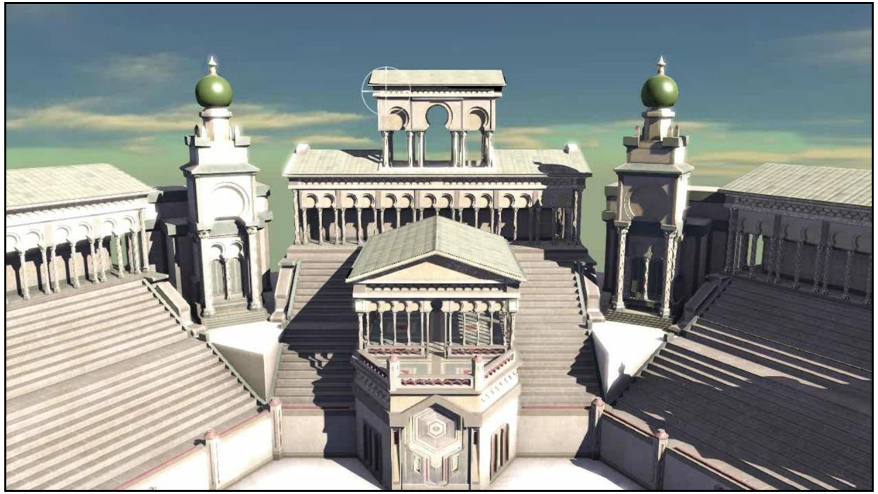


- Pre-designed fracture pattern
- Align pattern with impact location at runtime
- Use pattern as stencil



Arena Destruction (SG 2013 real time live)

- 500k faces at start
- GPU1: rigid body simulation
- GPU2: smoke, rendering
- CPU: dynamic fracturing



Deformable Objects

- 1d: Ropes, hair
- 2d: Cloth, clothing
- 3d: Fat guys, tires

Existing Methods

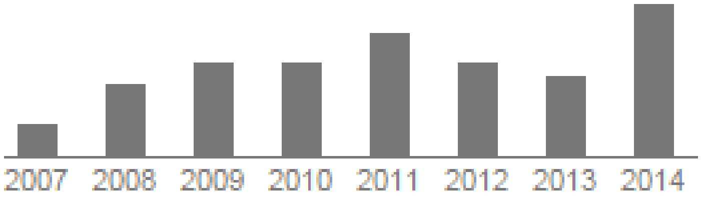
- Force based
- Mass-Spring Systems / FEM
- Explicit integration unstable
- Implicit integration
  - Expensive
  - Large time steps for real time simulation needed
  - Numerical damping

Position Based Dynamics

[Müller et al., 2006]

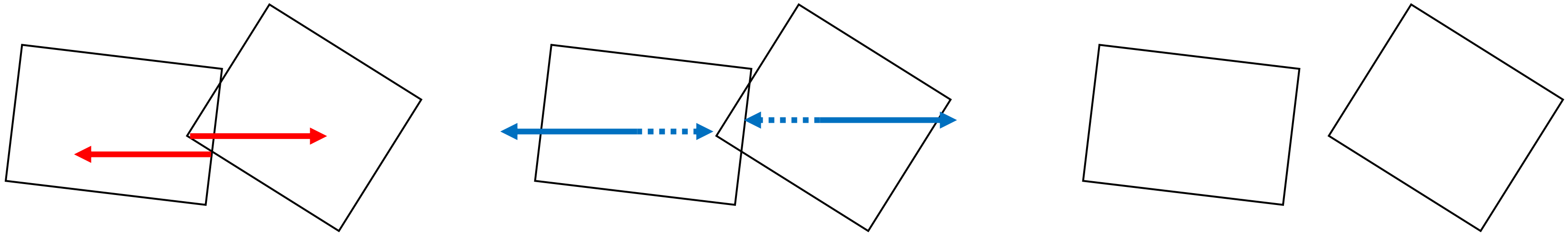
Position Based Dynamics

[Müller et al., 2006]



[google scholar]

Force Based Update



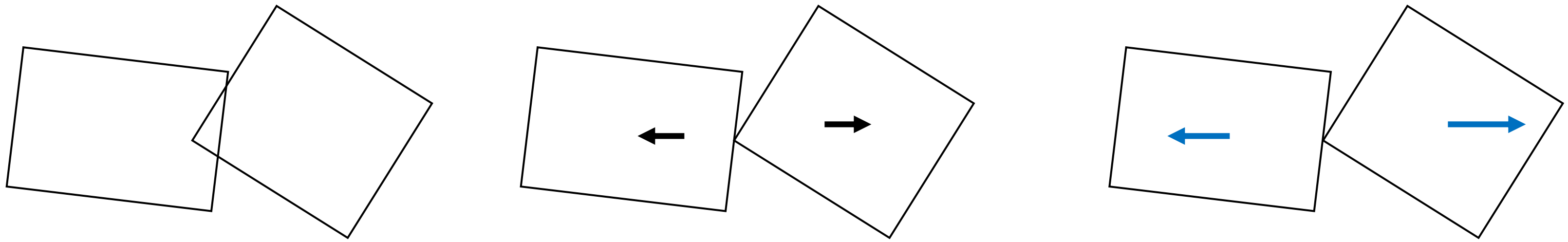
penetration causes forces

forces change velocities

velocities change positions

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

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!

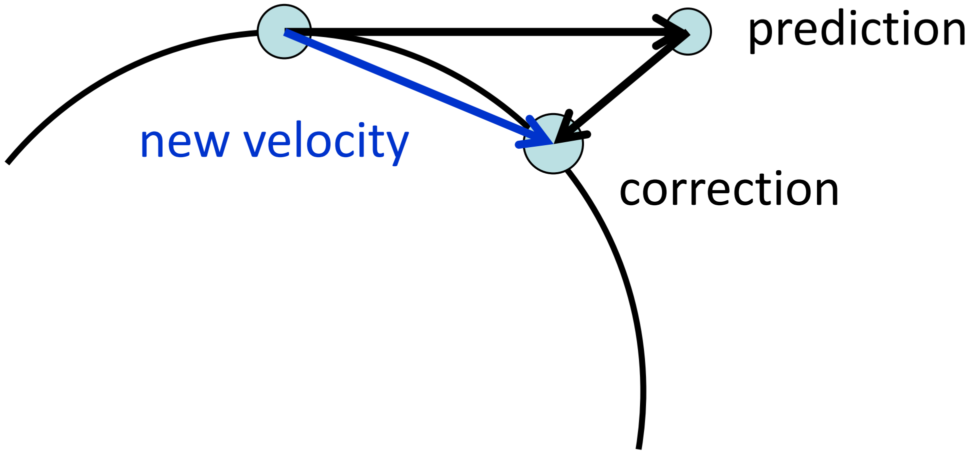
Position Based Integration

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

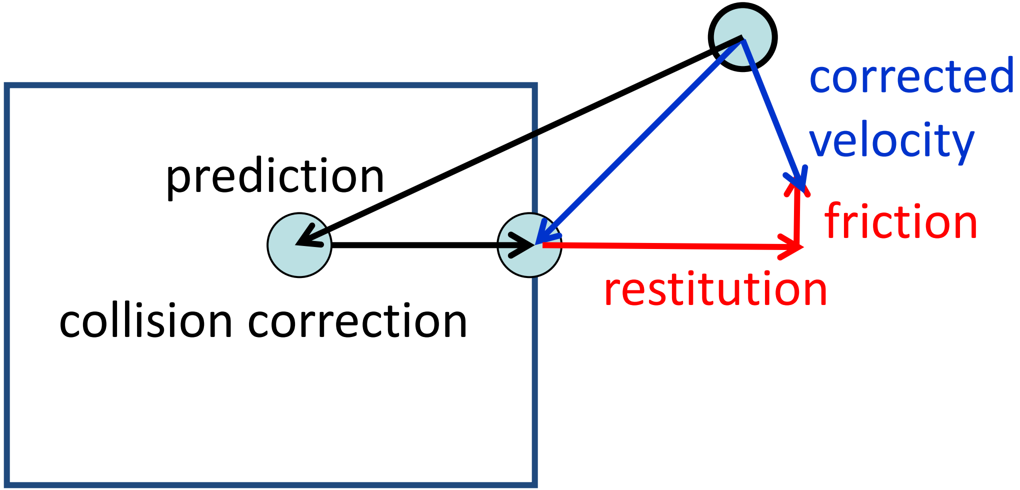
Position Correction

- Example: Particle on circle

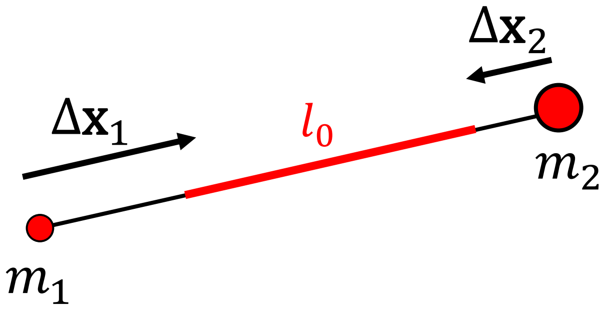


Velocity Correction

- External forces: $\mathbf{v}_{n+1} = \mathbf{u} + \Delta t \dfrac{\mathbf{g}}{m}$
- 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

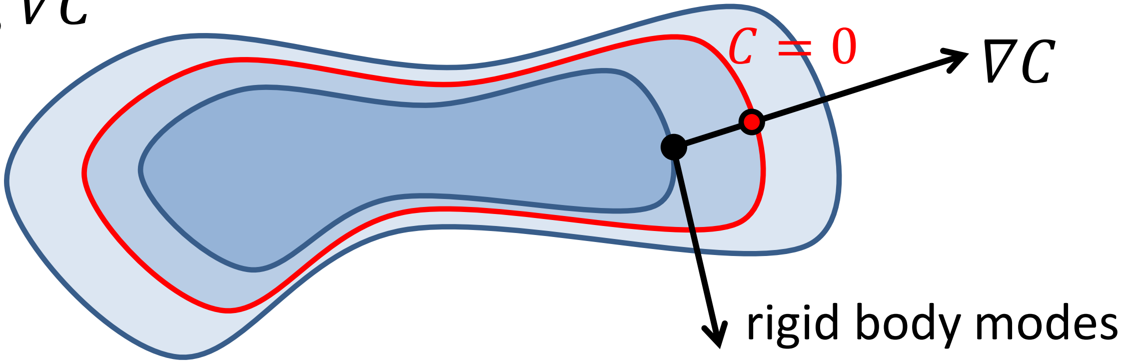
General Internal Constraint

- Define constraint via scalar function:

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

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

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



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

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

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

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

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

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

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

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

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

⇩

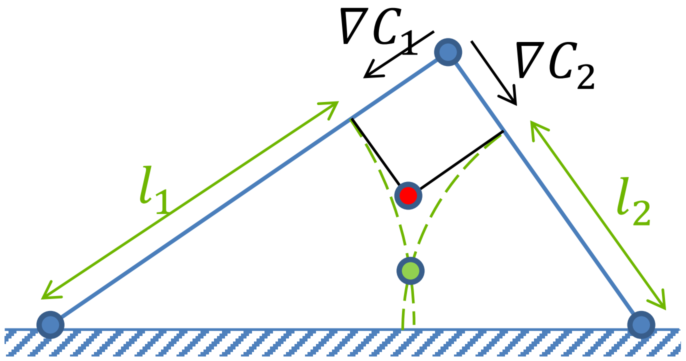
$$\Delta\mathbf{x} = \mathrm{M}^{-1}\nabla\mathrm{C}(\mathbf{x})^T\boldsymbol{\lambda}$$

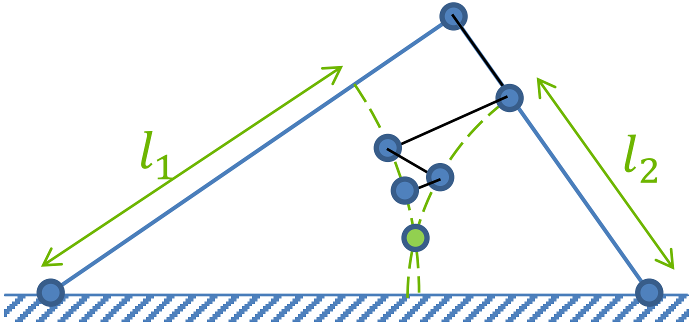
$$[\nabla C(\mathbf{x})\mathrm{M}^{-1}\nabla\mathrm{C}(\mathbf{x})^T]\,\boldsymbol{\lambda} = -\mathrm{C}(\mathbf{x})$$

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

Other Speedup Tricks

- Use as smoother in a multi-grid method
- Long range distance constraints (LRA)
- Shape matching
- Hierarchy of meshes

Amazing Gauss-Seidel!

- Can handle unilateral (inequality) constraints (LCPs, QPs)!
  - Fluids: separating boundary conditions [Chentanez at al., 2012]
  - Rigid bodies: LCP solver [Tonge et al., 2012]
  - Deformable objects: Long range attachments [Kim et al., 2012]
- Works on non-linear problem directly
- Handles under and over-constrained problems
- GS + PBD: garbage in, simulation out (almost ☺)
- Fine grained interleaved solver trivial
- Easy to implement and parallelize

Analysis of PBD

Correction = Acceleration

- Predicted position

$$\mathbf{p} = \mathbf{x}_n + \Delta t \mathbf{v}_n = \mathbf{x}_n + \Delta t \frac{(\mathbf{x}_n - \mathbf{x}_{n-1})}{\Delta t} = 2\mathbf{x}_n - \mathbf{x}_{n-1}$$

- Projection

$$\mathbf{x}_{n+1} = \mathbf{p} + \Delta \mathbf{x}$$

$$\Delta \mathbf{x} = \mathbf{x}_{n+1} - 2\mathbf{x}_n + \mathbf{x}_{n-1}$$

Implicit Euler

$$\mathrm{M}\frac{\mathbf{x}_{n+1} - 2\mathbf{x}_n + \mathbf{x}_{n-1}}{\Delta t^2} = \mathbf{f}(\mathbf{x}_{n+1})$$

$$\mathrm{M}\Delta\mathbf{x} = \Delta t^2\, \mathbf{f}(\mathbf{x}_{n+1})$$

Formulation as an optimization problem for $\Delta\mathrm{x}$:

$$\min\left(\underbrace{\frac{1}{2}\Delta\mathbf{x}^T\mathrm{M}\Delta\mathbf{x}}_{\text{inertia term}} + \underbrace{\Delta t^2 E(\mathbf{x}_{n+1})}_{\text{energy term}}\right)$$

Stiffness → Infinity

$$\min\left(\frac{1}{2}\Delta\mathbf{x}^T \mathrm{M}\Delta\mathbf{x} \;+\; \Delta t^2 \frac{1}{2} k C^2(\mathbf{x}_{n+1})\right) \qquad // \; E(\mathbf{x}) = \frac{1}{2} k C^2(\mathbf{x})$$

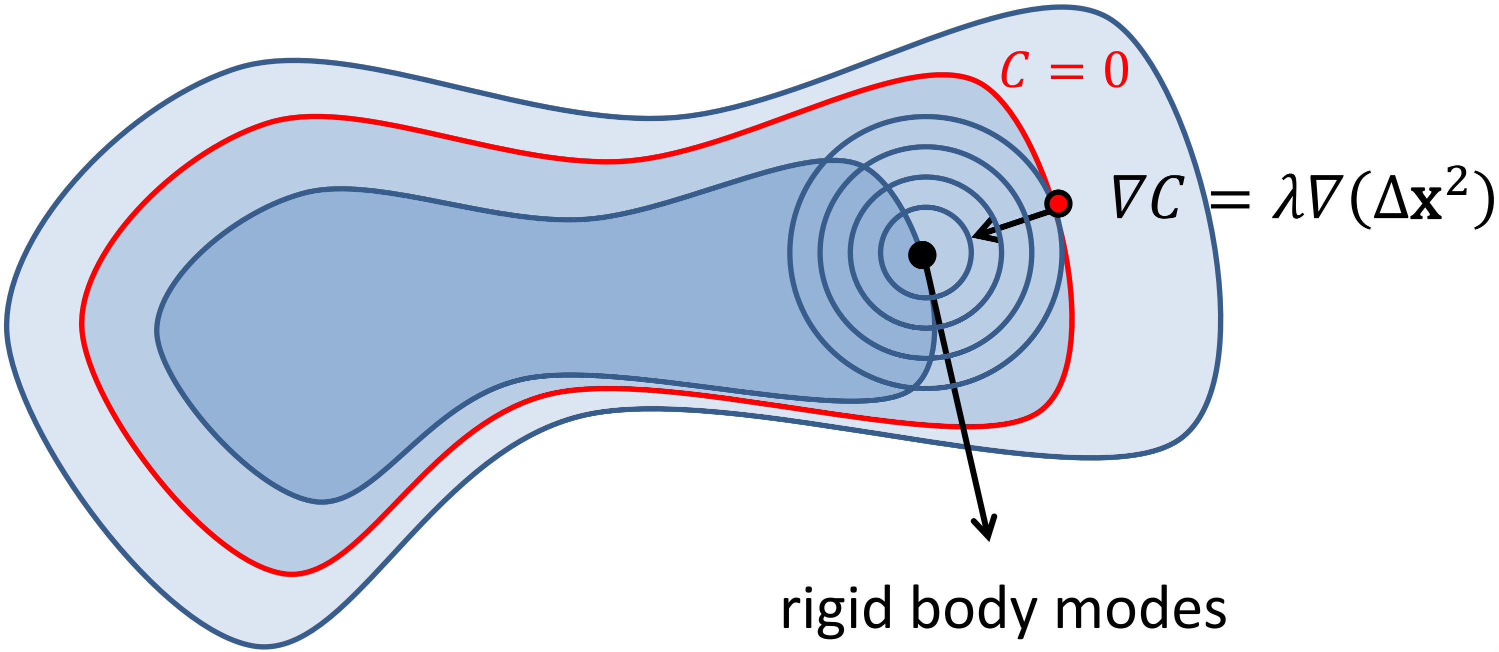
Now let $k \to \infty$

$$\min\left(\frac{1}{2}\Delta\mathrm{x}^T \mathrm{M}\Delta\mathrm{x}\right) \text{ subject to } C(x_{n+1}) = 0$$

- $C(\mathbf{x}_{n+1}) = 0$
- $\mathrm{M}\Delta\mathbf{x} = \lambda \nabla C(\mathbf{x}_{n+1})$

$\Rightarrow$ $\Delta\mathbf{x} = \lambda\, \mathrm{M}^{-1} \nabla C(\mathbf{x}_{n+1})$ PBD

Two Interpretations



Constraint Solver

- PBD solves a non-linear optimization problem

$$\min\left(\frac{1}{2}\Delta\mathbf{x}^T \mathrm{M}\Delta\mathbf{x}\right) \text{ subject to } C_i(\mathbf{x}_{n+1}) = 0, \quad i \in [1,..,m]$$

by solving a sequence of QPs:

$$\min\left(\frac{1}{2}\Delta\mathbf{x}^T \mathrm{M}\Delta\mathbf{x}\right) \text{ subject to } C_i(\mathbf{x}_{n+1}) = 0$$

Clothing Demo

Nurien

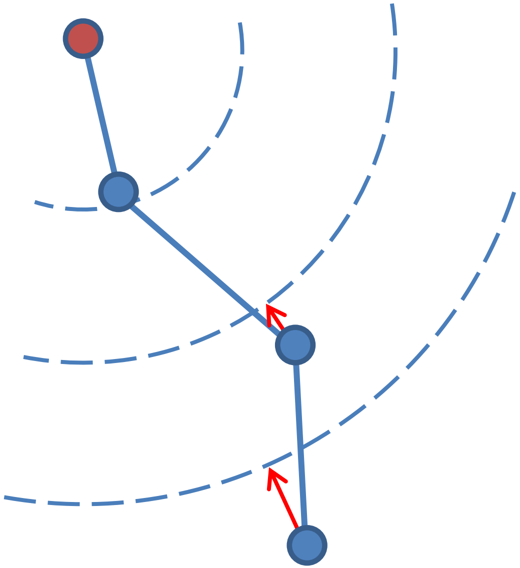
Cloth

- Slow error propagation $\rightarrow$ stretchy cloth
- Low resolution: no detailed wrinkles

- Solutions
  - Use hierarchy of meshes (complicated)
  - Has been an open problem for us
  - Found an embarrassingly simple solution

Long Range Attachments (LRA)

- Upper distance constraint to closest attachment point
- Unilateral: project only if distance too big

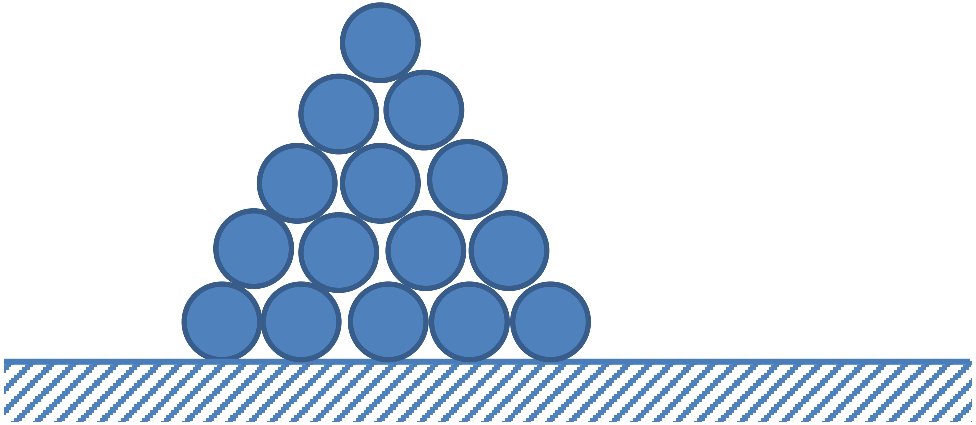


[Kim et al., 2012], 90k particles

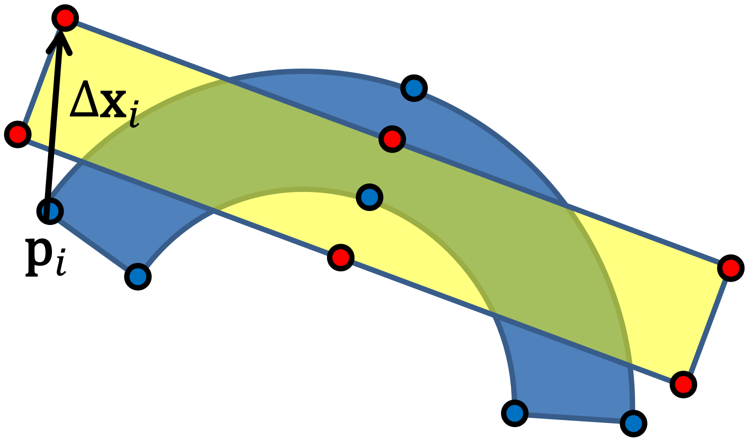
630k particles, 100k hairs, 27.29 fps
Left Mouse – Blow
Right Mouse – Rotate Camera
3 Lights
8xMSAA
Supersampling
Shadow Opacity
Shadow Ambient
Per Hair Shadow Variation
Hair Opacity
Hair Base Thickness
Hair Tip Thickness
Hair Diffuse Color Scale
Hair Specular Color Scale
Hair Specularity
Fluffiness
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)

Challenge

- Similar idea for compression?
- Long range distance constraint to the ground?



Rigid Objects



- Optimally match un-deformed with deformed shape
- Only allow translation and rotation
- Global correction, no propagation needed
- No mesh needed!

NVIDIA

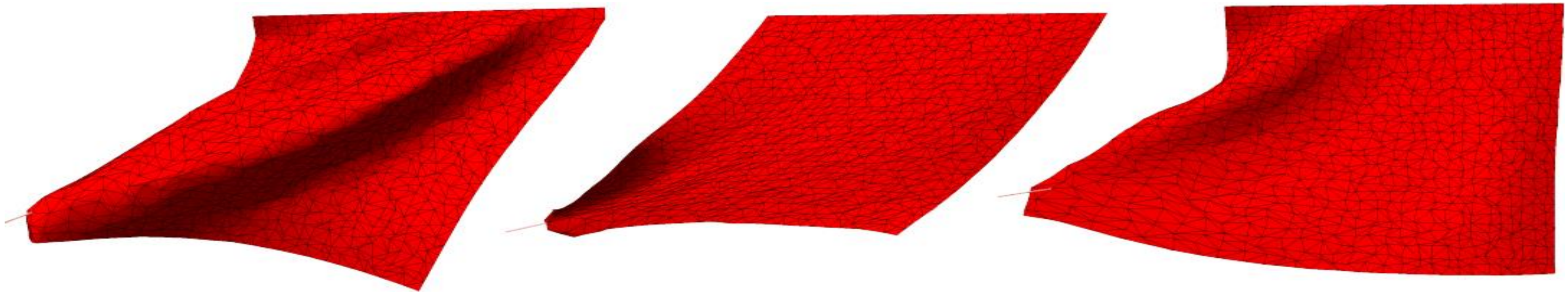
Position Based Fluids

[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

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

Mesh Independent Deformations



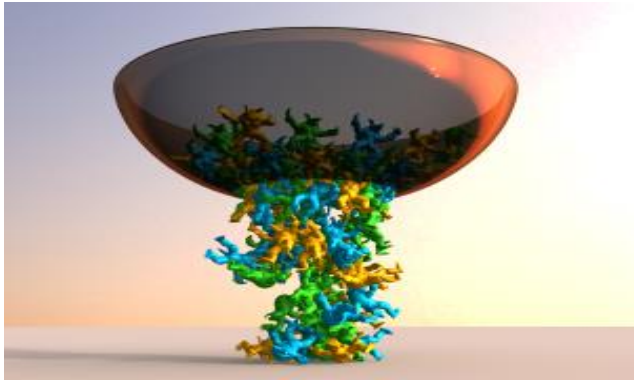
[Müller et al, 2014]

- For each triangle:

$$C(\mathbf{x}_1, .., \mathbf{x}_3) = \mathbf{G}_{ij}(\mathbf{x}_1, .., \mathbf{x}_3)$$

$$\mathbf{G} = \mathbf{F}^{\mathrm{T}}\mathbf{F} - \mathbf{I}$$

FEM



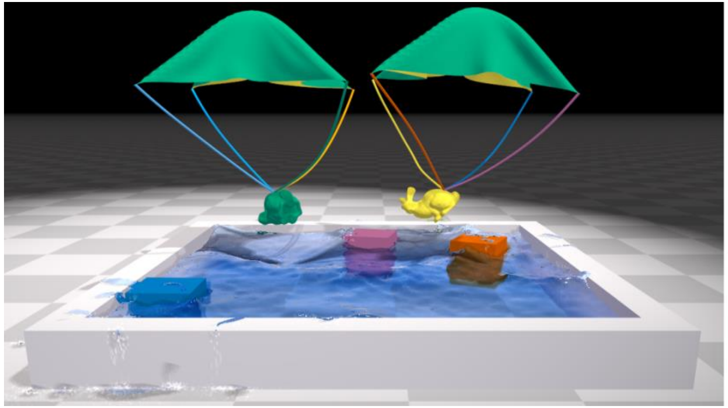
[Bender et al, 2014]

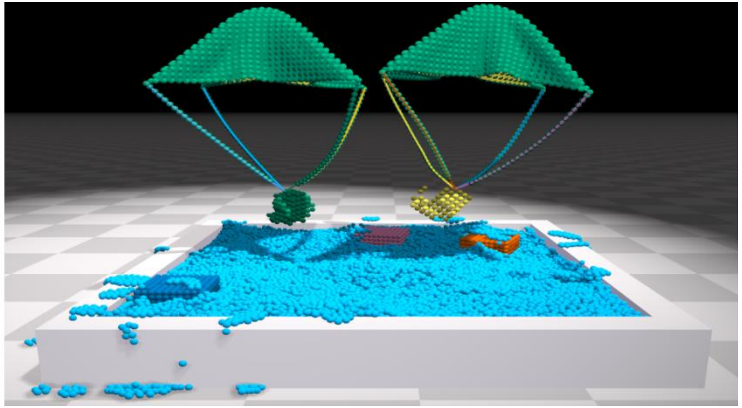
- For each tetrahedron:

$$C(\mathbf{x}_1, .., \mathbf{x}_4) = E_{FEM}(\mathbf{x}_1, .., \mathbf{x}_4)$$

Unified Solver

[Macklin et al., 2014]





- Putting it all together
- Plus
  - Static friction
  - Stiff stacks via mass modifications
  - Two-way fluid – solid coupling

Acknowledgements

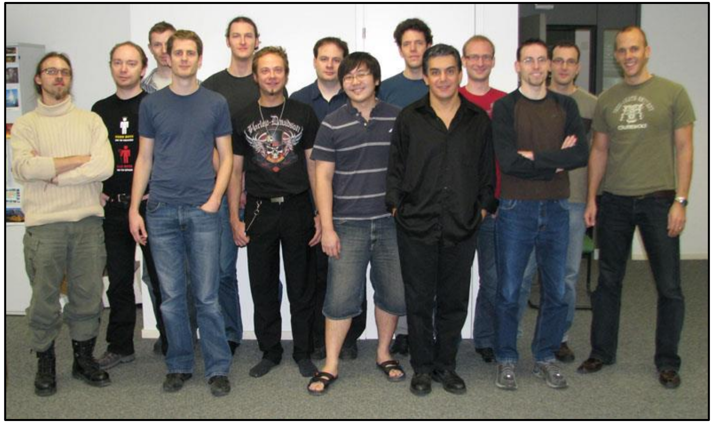
- PhysX Research Group

Nuttapong Chentanez

Tae-Yong Kim

Miles Macklin

- PhysX Group



Thanks!

Questions?