Position-Based Dynamics

Analysis and Implementation

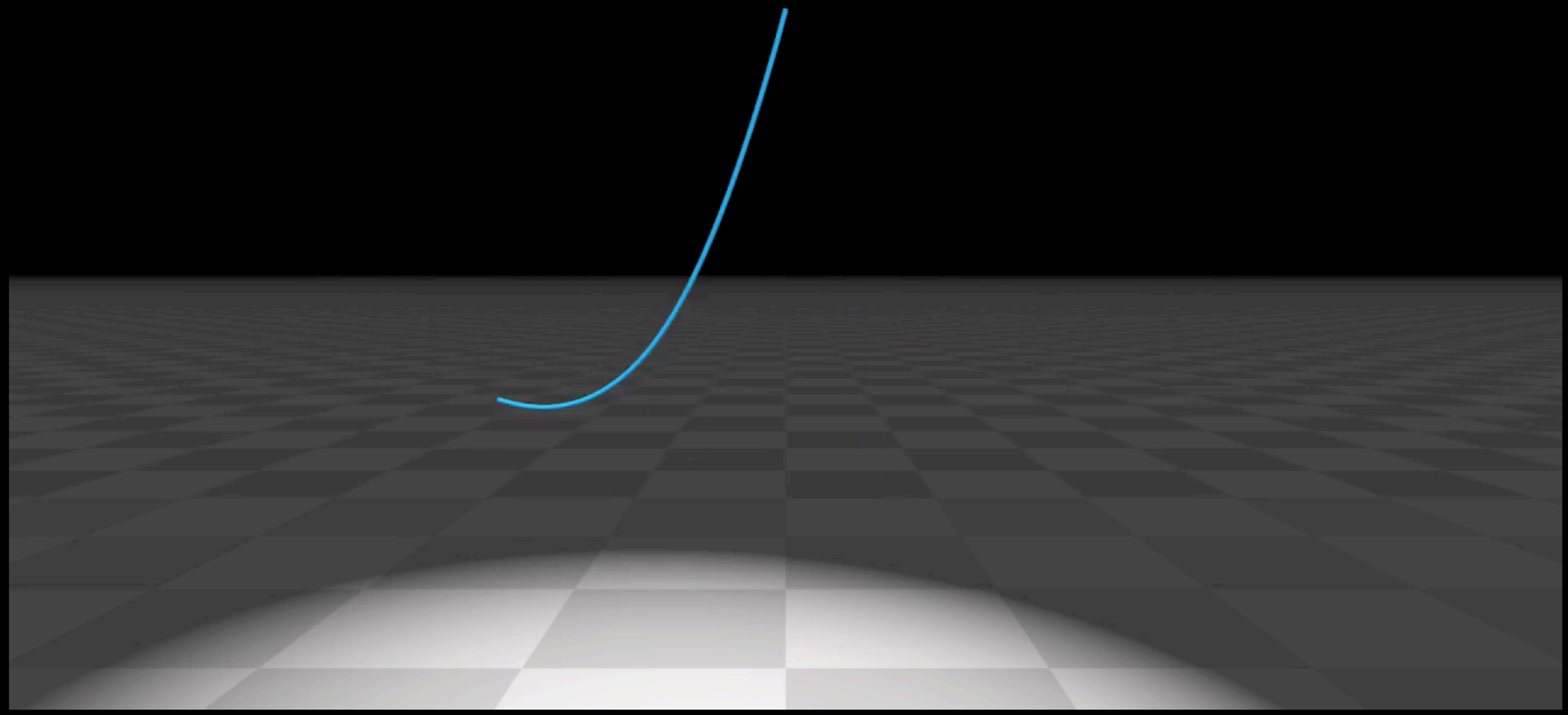
Miles Macklin

NVIDIA

Analysis

Position-Based Dynamics

- Very stable
- Highly damped
- Example



Verlet Integration

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

- Symplectic (long term energy conservation)
- So why the damping?
- Where are the constraint forces?

Implicit Euler Integration

$$\mathbf{v}^{n+1} = \mathbf{v}_n + \Delta t \mathbf{M}^{-1} \mathbf{f}(\mathbf{x}^{n+1})$$

$$\mathbf{x}^{n+1} = \mathbf{x}_n + \Delta t \mathbf{v}^{n+1}$$

- Forces evaluated at end of the time-step
- Equivalent to:

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

- Very stable
- Highly damped

Variational Implicit Euler

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

- Can be seen as the first order optimality conditions for the following minimization:

$$\min_{\mathbf{x}^{n+1}} \quad \frac{1}{2}(\mathbf{x}^{n+1} - \tilde{\mathbf{x}})^T \mathbf{M}(\mathbf{x}^{n+1} - \tilde{\mathbf{x}}) + \Delta t^2 E(\mathbf{x}^{n+1})$$

- Predicted (inertial) position: $\tilde{\mathbf{x}} = 2\mathbf{x}^n - \mathbf{x}^{n-1} + \mathbf{M}^{-1}\mathbf{f}_{ext}$
  $= \mathbf{x}^n + \Delta t \mathbf{v}^n + \mathbf{M}^{-1}\mathbf{f}_{ext}$
- See [Goldenthal 07], [Liu 13]

Variational Implicit Euler

$$\min_{\mathbf{x}^{n+1}} \quad \frac{1}{2}(\mathbf{x}^{n+1} - \tilde{\mathbf{x}})^T \mathbf{M}(\mathbf{x}^{n+1} - \tilde{\mathbf{x}}) + \Delta t E(\mathbf{x}^{n+1})$$

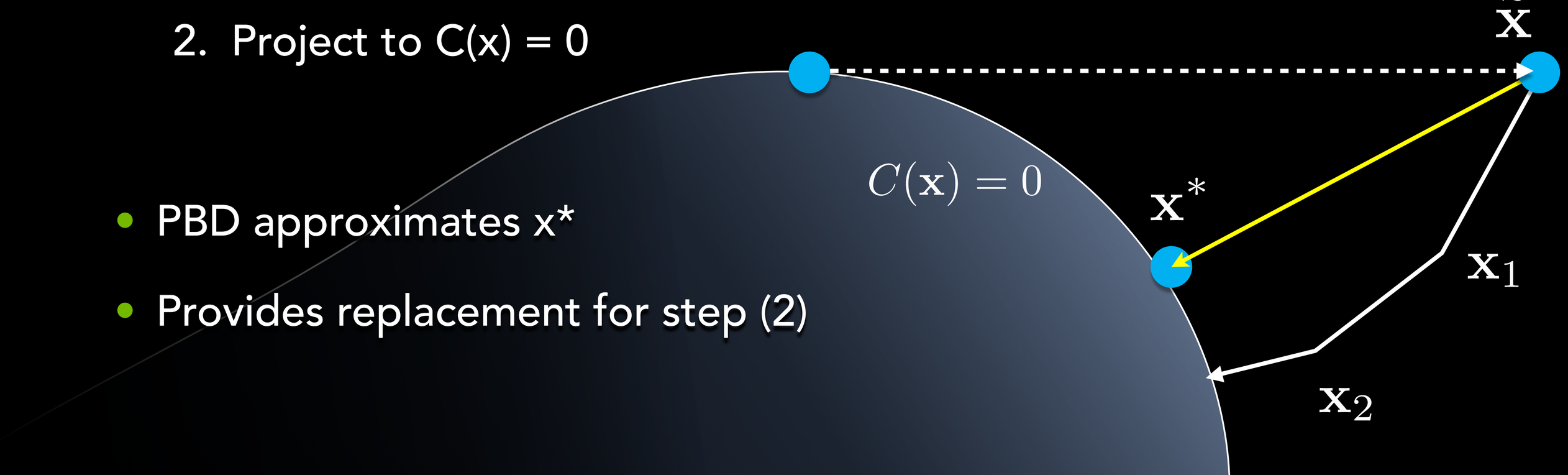
- As $E \to \infty$ we obtain the following constrained minimization:

$$\begin{aligned} \min_{\mathbf{x}^{n+1}} \quad & \frac{1}{2}(\mathbf{x}^{n+1} - \tilde{\mathbf{x}})^T \mathbf{M}(\mathbf{x}^{n+1} - \tilde{\mathbf{x}}) \\ \text{s.t.} \quad & C_i(\mathbf{x}^{n+1}) = 0 \end{aligned}$$

- Searching for the point closest to the predicted (inertial) position that lies on the constraint manifold

PBD and Implicit Euler

- Minimization form gives the following algorithm for implicit Euler:
  1. Predict new position $\tilde{\mathbf{x}}$
  2. Project to C(x) = 0



- PBD approximates x*
- Provides replacement for step (2)

Constraint Projection

- Newton's method requires global linearization
- PBD uses local linearization (no linear solves)
- PBD uses first derivatives (no Hessians)
- Iterative projection
  - Simple handling of inequalities
  - Robust for over constrained systems

Putting it all together

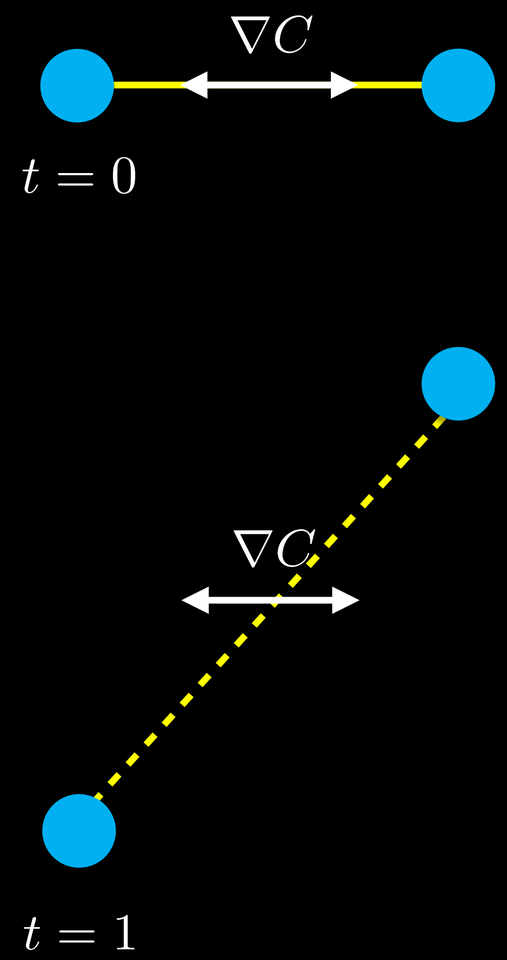
$$\mathbf{M}(\mathbf{x}^{n+1} - 2\mathbf{x}^{n} + \mathbf{x}^{n-1}) = \Delta t^2 \left[\mathbf{f}_{ext}(\mathbf{x}^{n}) + \mathbf{f}_{con}(\mathbf{x}^{n+1})\right]$$

- Approximate mixed explicit, implicit scheme
- Stiff constraints handled implicitly (stable, damped)
- PBD assumes infinite stiffness
- Addressed by Projective Dynamics [Bouaziz et al. 14]

Other Close Relatives

- SHAKE / RATTLE [Ryckaert et al 77]
  - Constraint gradients fixed at start of time-step
- Strain Limiting [Provot 95]
  - No velocity update
- Nucleus [Stam 09]
  - Velocity formulation:

$$C(\mathbf{x}^n + \Delta t \mathbf{v}^n + \Delta t \Delta \mathbf{v}) = 0$$



Second Order Implicit Euler

- First order backward Euler (BDF1)

$$\mathbf{v}^{n+1} = \mathbf{v}_n + \Delta t \mathbf{M}^{-1} \mathbf{f}(\mathbf{x}^{n+1})$$
$$\mathbf{x}^{n+1} = \mathbf{x}_n + \Delta t \mathbf{v}^{n+1}$$

- Second order backward Euler (BDF2)

$$\mathbf{v}^{n+1} = \frac{4}{3}\mathbf{v}^n - \frac{1}{3}\mathbf{v}^{n-1} + \frac{2}{3}\Delta t \mathbf{M}^{-1} \mathbf{f}(\mathbf{x}^{n+1})$$
$$\mathbf{x}^{n+1} = \frac{4}{3}\mathbf{x}^n - \frac{1}{3}\mathbf{x}^{n-1} + \frac{2}{3}\Delta t \mathbf{v}^{n+1}$$

Second Order PBD

- First order prediction:

$$\tilde{\mathbf{x}} = \mathbf{x}^n + \Delta t \mathbf{v}^n + \Delta t^2 \mathbf{M}^{-1} \mathbf{f}_{ext}$$

- First order velocity update:

$$\mathbf{v}^{n+1} = \frac{1}{\Delta t} \left[\mathbf{x}^{n+1} - \mathbf{x}^n \right]$$

- Second order prediction:

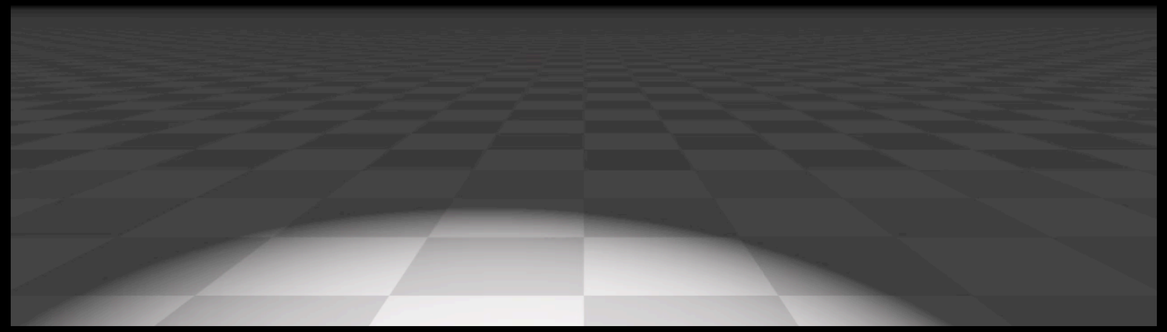
$$\begin{aligned} \tilde{\mathbf{x}} = & \frac{4}{3}\mathbf{x}^n - \frac{1}{3}\mathbf{x}^{n-1} + \frac{8}{9}\Delta t \mathbf{v}^n \\ & - \frac{2}{9}\Delta t \mathbf{v}^{n-1} + \frac{4}{9}\Delta t^2 \mathbf{M}^{-1} \mathbf{f}_{ext} \end{aligned}$$

- Second order velocity update:

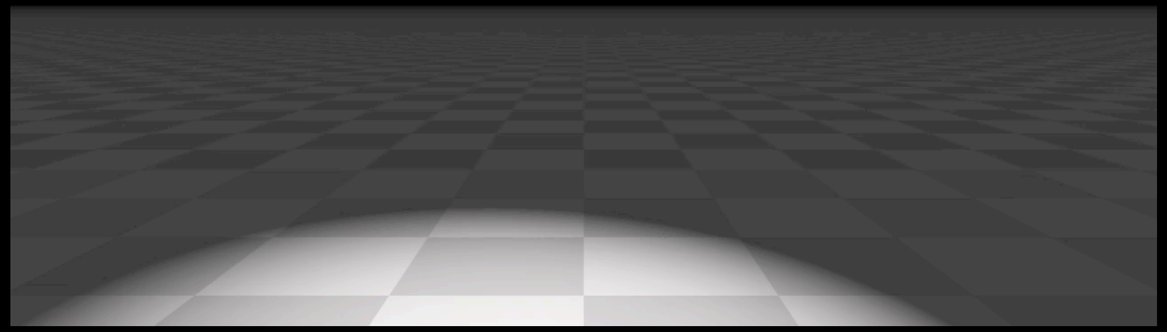
$$\mathbf{v}^{n+1} = \frac{1}{\Delta t} \left[\frac{3}{2}\mathbf{x}^{n+1} - 2\mathbf{x}^n + \frac{1}{2}\mathbf{x}^{n-1} \right].$$

- See [English 08]

Second Order PBD

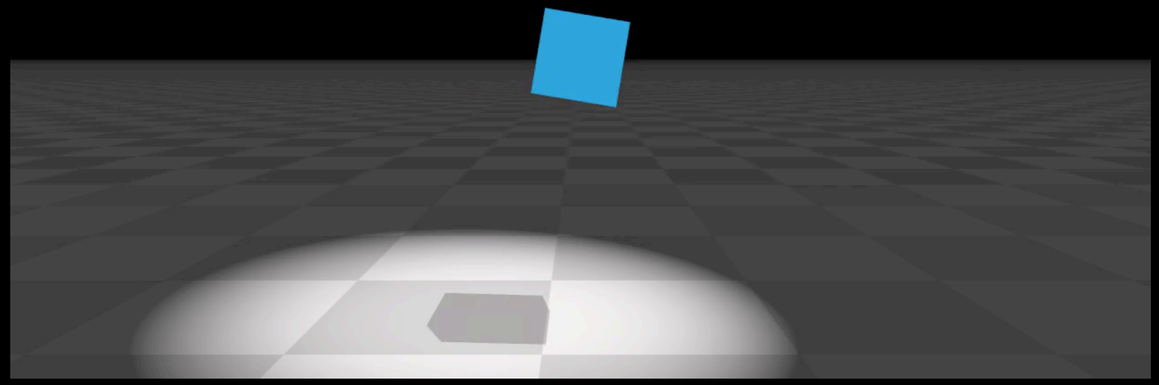


First Order

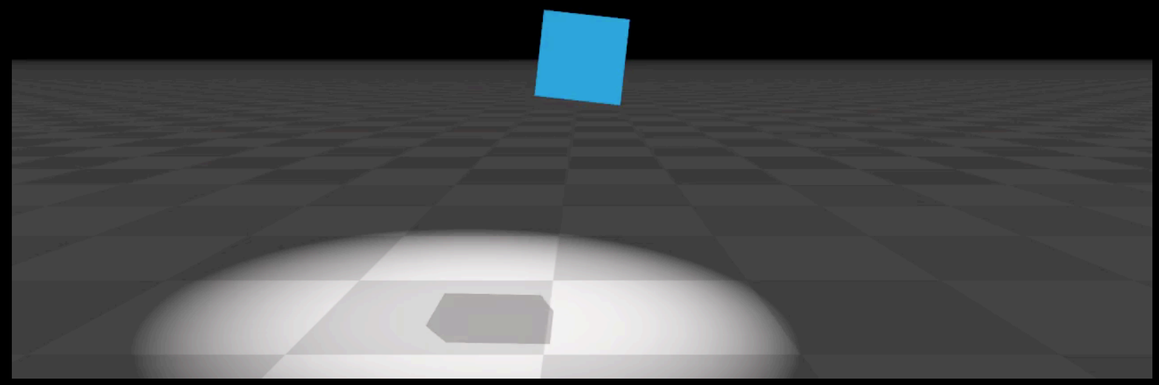


Second Order

Second Order PBD

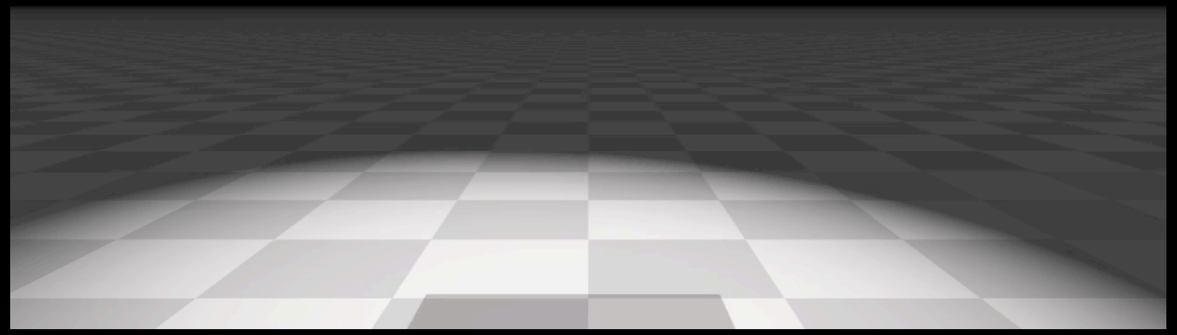


First Order

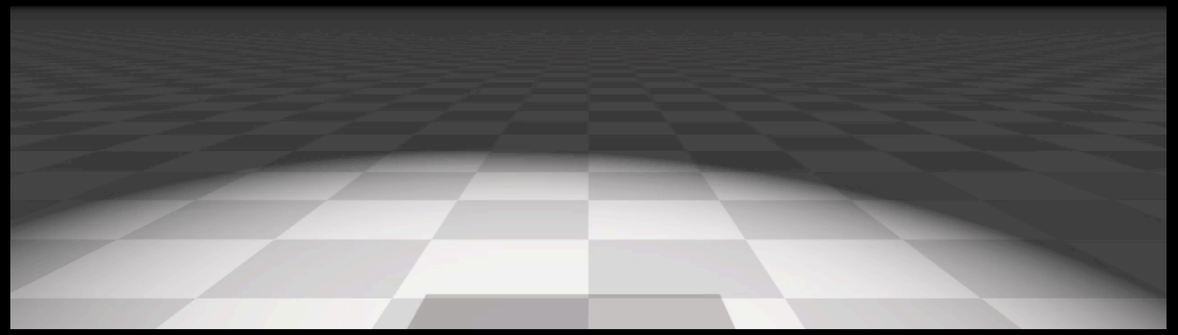


Second Order

Second Order PBD



First Order



Second Order

Second Order PBD

- Significantly less damping
- Positions stay closer to constraint manifold
- Requires fewer constraint iterations!
- Non-smooth events (contact) need special handling

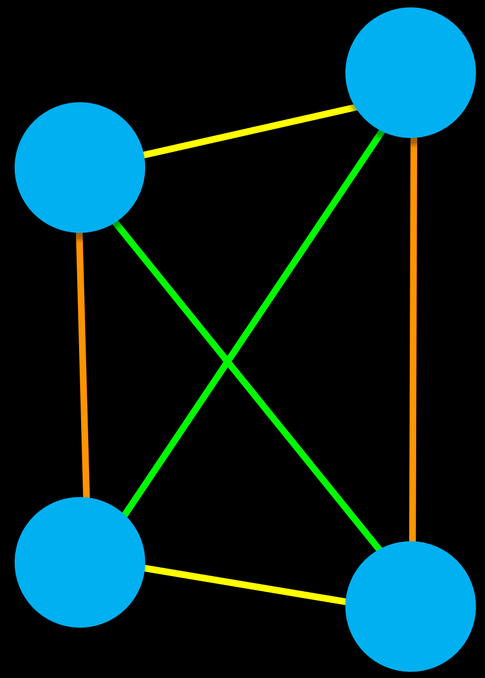
Implementation

Parallel PBD

- Gauss-Seidel inherently serial
- Parallel options:
  - Graph Coloring methods
  - Jacobi methods
  - Hybrid methods

Graph Coloring Methods

- Break constraint graph into independent sets
- Solve the constraints in a set in parallel
- “Batched” Gauss-Seidel
- Requires synchronization between each set
- Size of sets decreases -> poor utilisation



3 Color Graph

Jacobi Methods

- Process each constraint or particle in parallel
- Sum up contributions on each particle

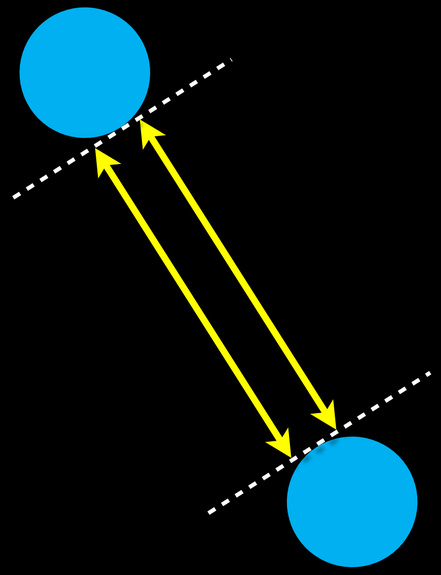
Particle-centric approach (gather)	Constraint-centric approach (scatter)

`foreach particle (in parallel)`
`{`
`  foreach constraint`
`  {`
`  calculate constraint error`
`  update delta`
`  }`
`}`
	`foreach constraint (in parallel)`
	`{`
	`  calculate constraint error`
	`  foreach particle`
	`  {`
	`  update delta (atomically)`
	`  }`
	`}`

Jacobi Methods

- Problem: system matrix can be indefinite, Jacobi will not converge, e.g.: for redundant constraints (cf. figure)
- Regularized Jacobi iteration via averaging [Bridson et al. 02]
- Sum all constraint deltas together and divide by constraint count for that particle

$$\mathbf{x_i} \leftarrow \mathbf{x_i} + \frac{1}{n_i} \sum_{n_i} \lambda_j \nabla C_j$$



- Successive-over relaxation by user parameter omega [0,2]:

$$\mathbf{x_i} \leftarrow \mathbf{x_i} + \frac{\omega}{n_i} \sum_{n_i} \lambda_j \nabla C_j$$

Comparison

Method	Advantages	Disadvantages
Batched Gauss-Seidel	• Good Convergence • Very Robust	• Graph Coloring • Synchronization
Jacobi	• Trivial Parallelism	• Slow Convergence • Less Robust

Hybrid Methods

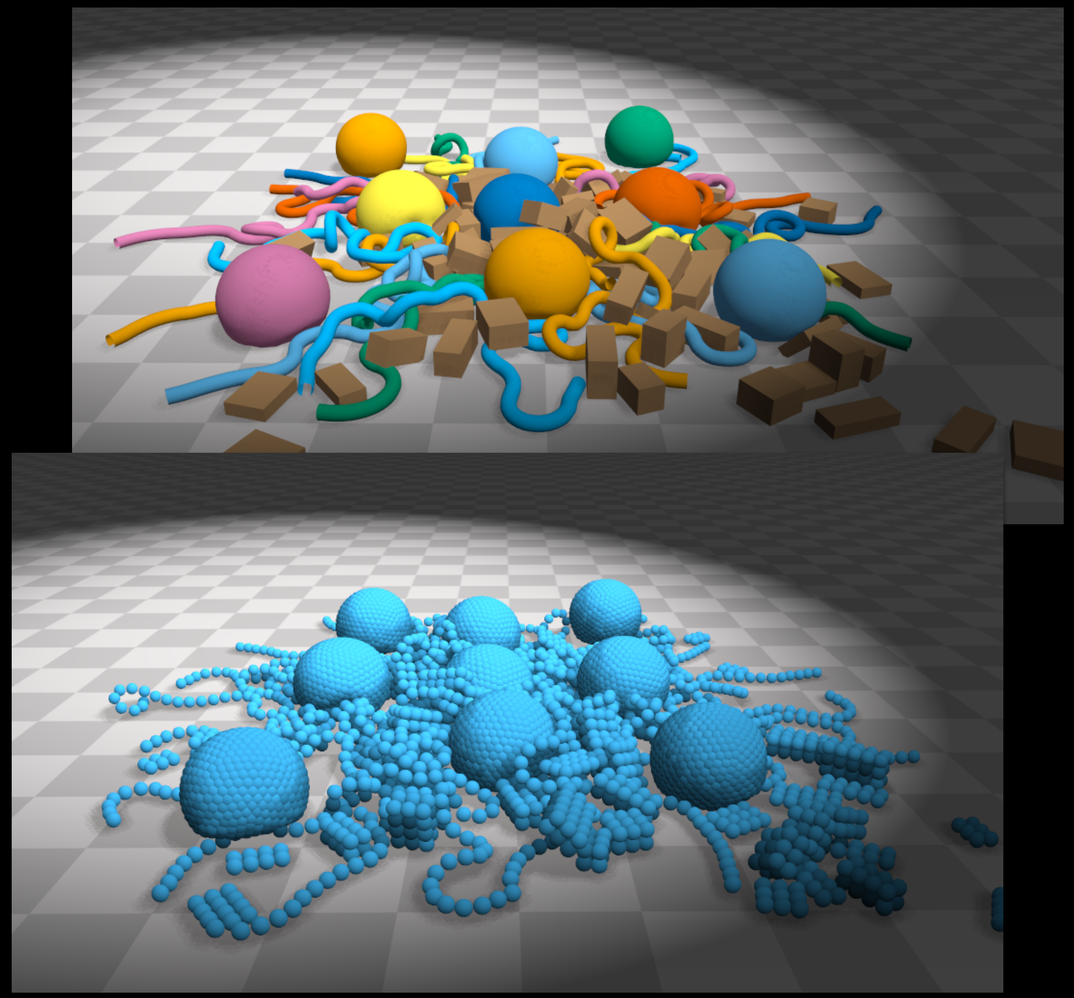
- Best of both worlds
- Perform graph-coloring
- Upper limit on number of colors
- Process everything else with Jacobi
- [Fratarcangeli & Pellacini 15] this week!

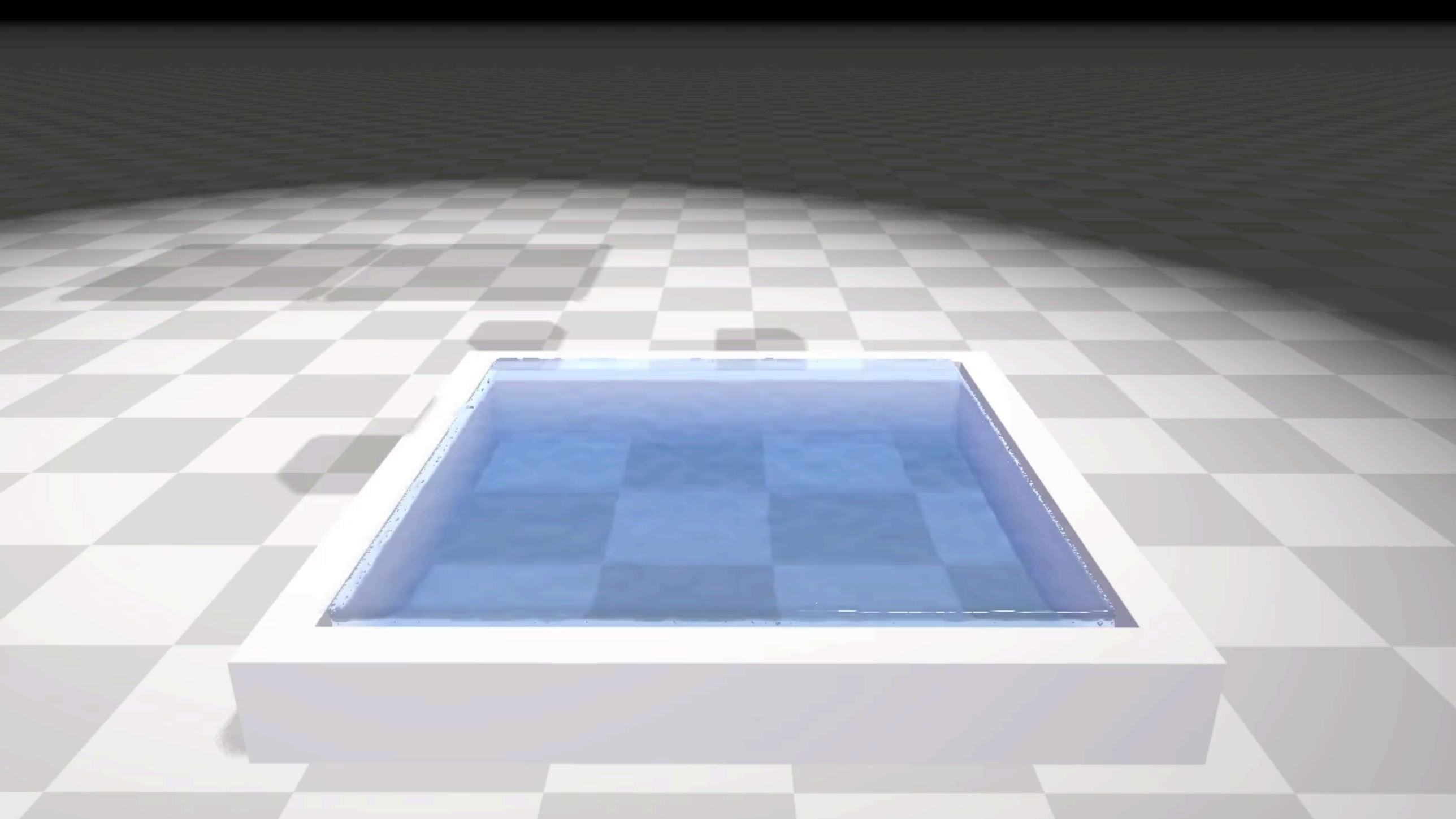
Solver Framework

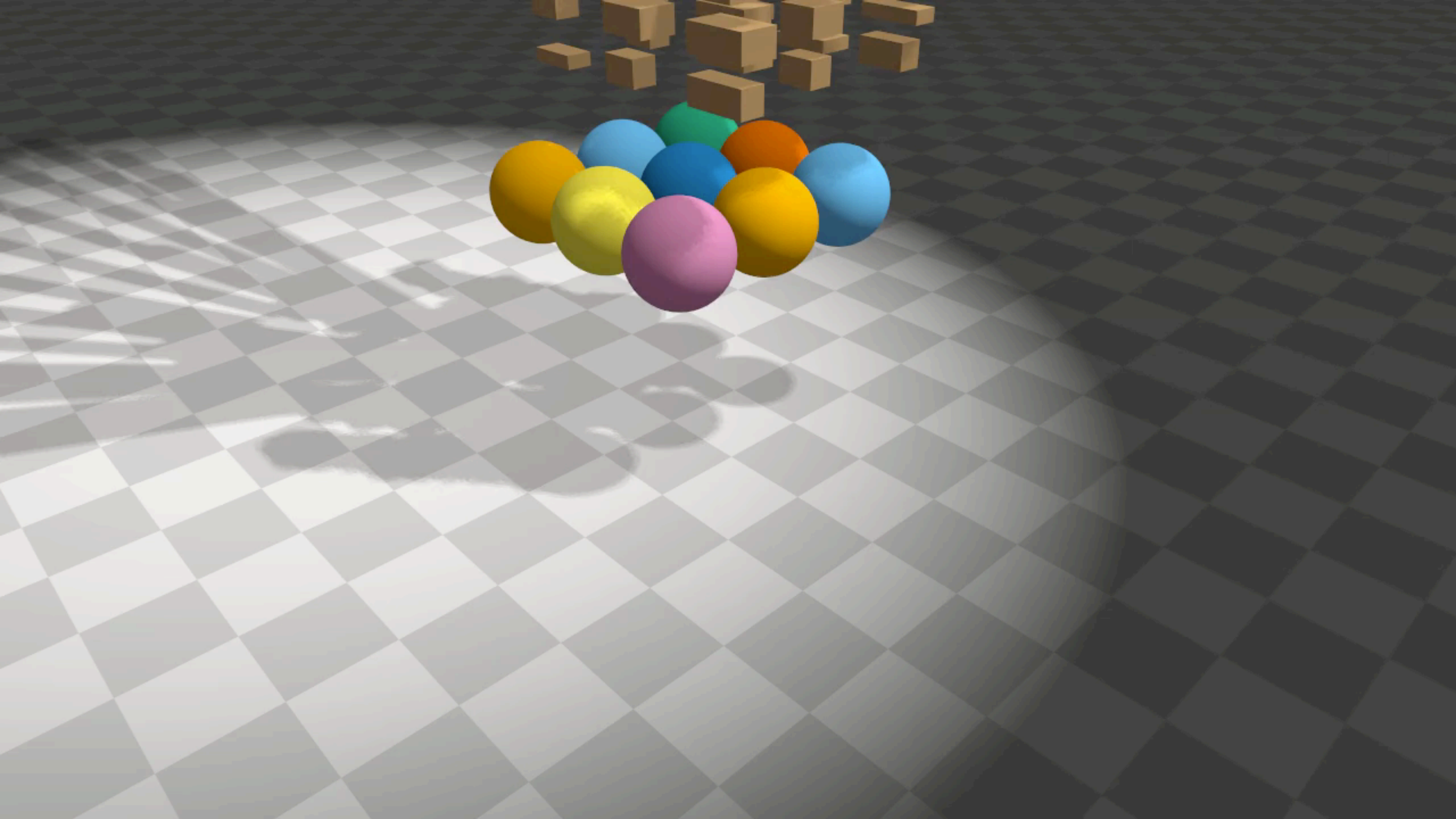
Unified Solver

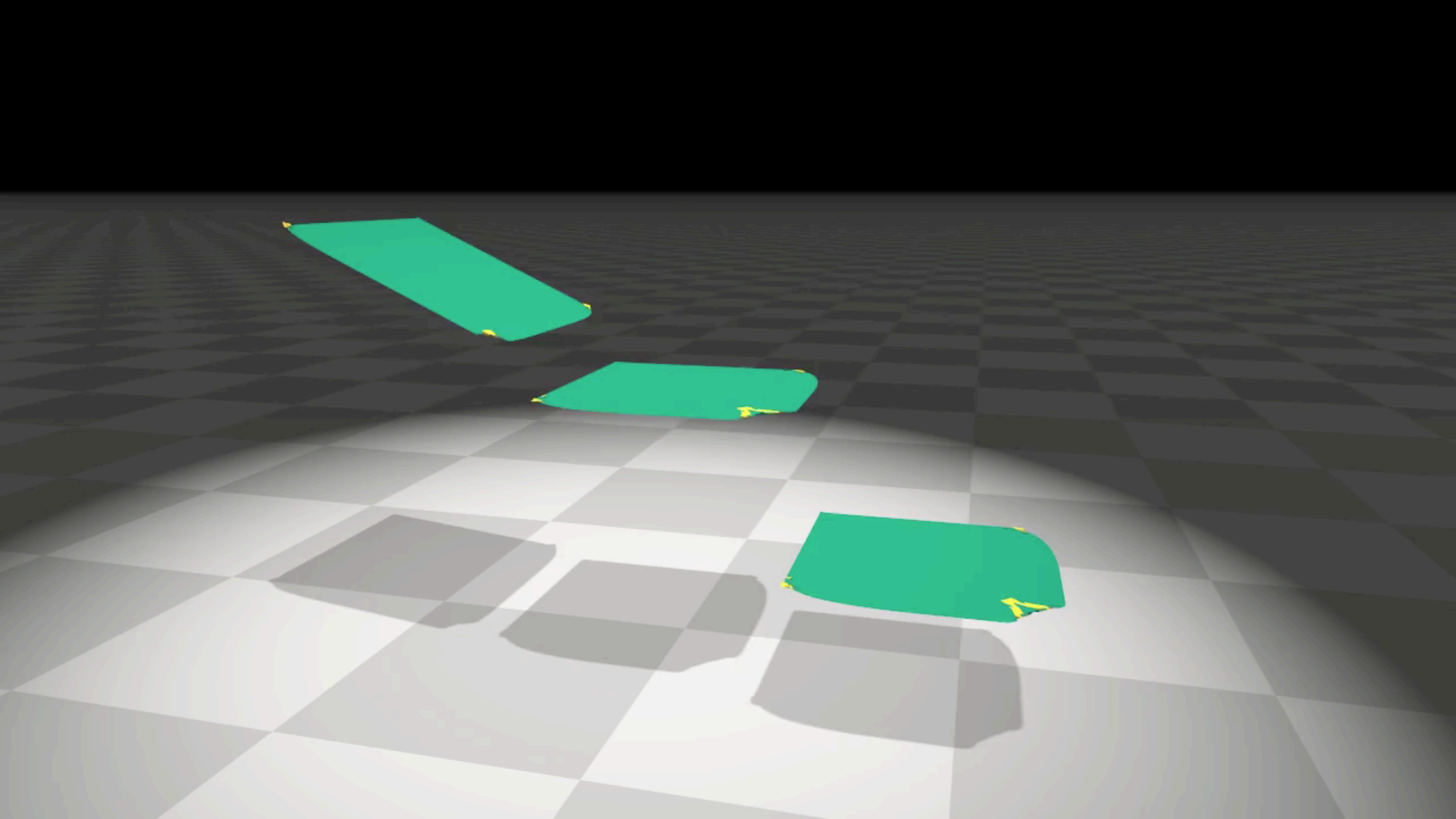
Everything is a set of particles connected by constraints

- Simplifies collision detection
- Two-way interaction of all object types:
  - Cloth
  - Deformables
  - Fluids
  - Rigid Bodies
- Fits well on the GPU









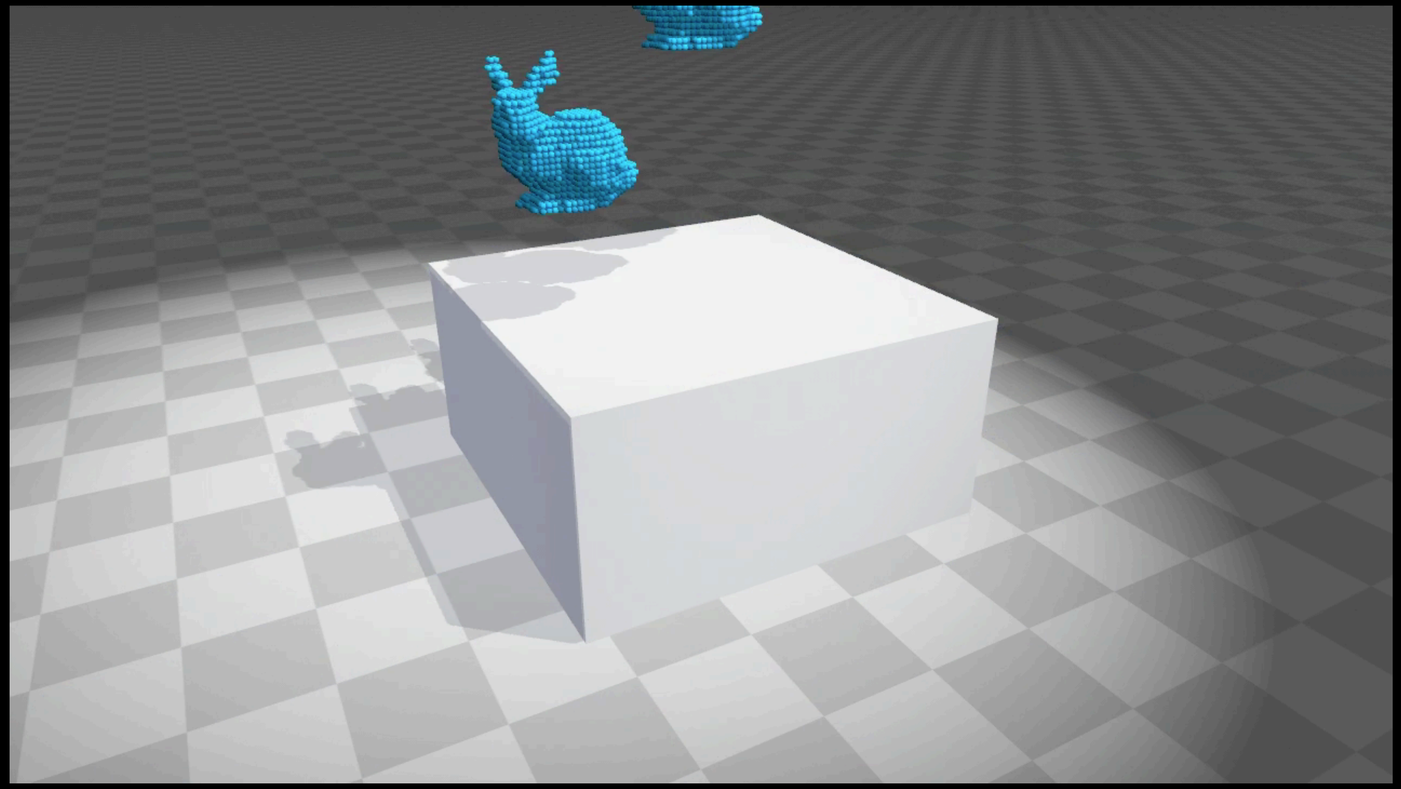
Particles

```
struct Particle
{
 float pos[3];
 float vel[3];
 float invMass;
 int phase;
};
```

- Velocity stored explicitly
- Phase-ID used to control collision filtering
- Global radius
- SOA layout

Constraints

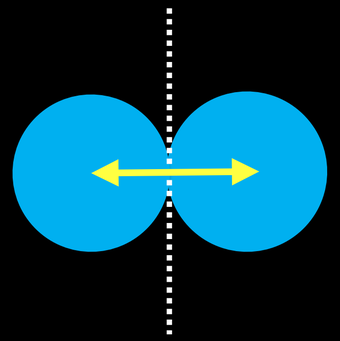
- Constraint types:
  - Distance (clothing)
  - Shape (rigids, plastics)
  - Density (fluids)
  - Volume (inflatables)
  - Contact (non-penetration)
- Combine constraints
  - Melting, phase-changes
  - Stiff cloth, bent metal



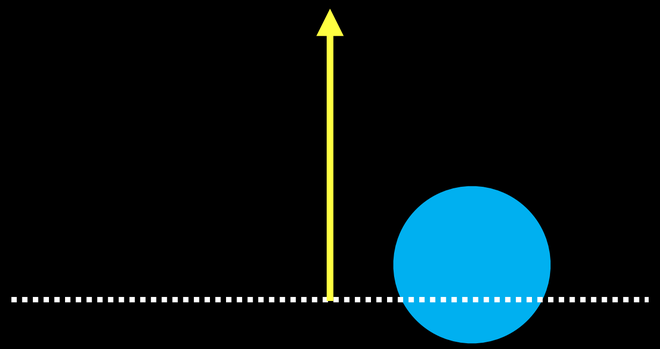
Contact and Friction

Collision Detection Between Particles

- All dynamics represented as particles
- Kinematic objects represented as meshes
- Two types of collision detection:
  - Particle-Particle
  - Particle-Mesh



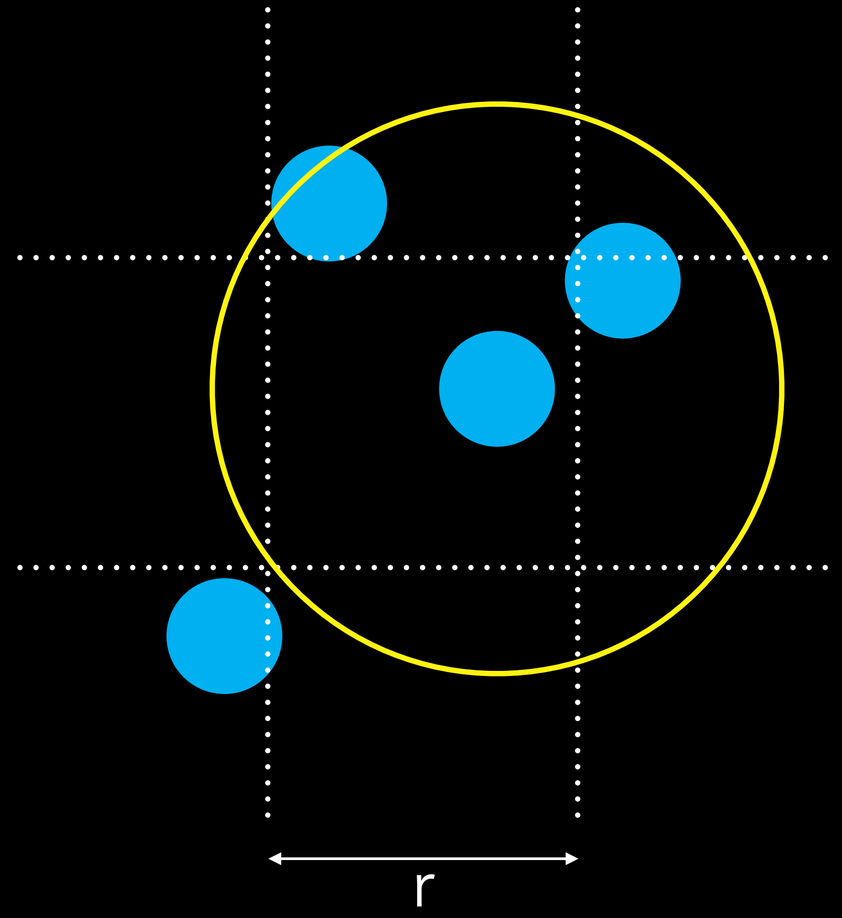
$$C_{contact} = |\mathbf{x}_i - \mathbf{x}_j| - 2r \geq 0$$



$$C_{contact} = \mathbf{n} \cdot \mathbf{x} - r \geq 0$$

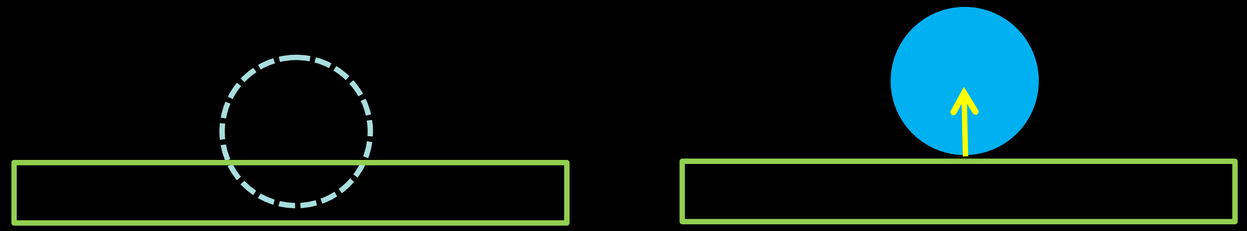
Collision Detection Between Particles

- Particle-Particle
  - Tiled uniform grid
  - Fixed maximum radius
  - Built using cub::DeviceRadixSort
  - Re-order particle data according to cell index to improve memory locality
  - CUDA Particles Sample [Green 07]



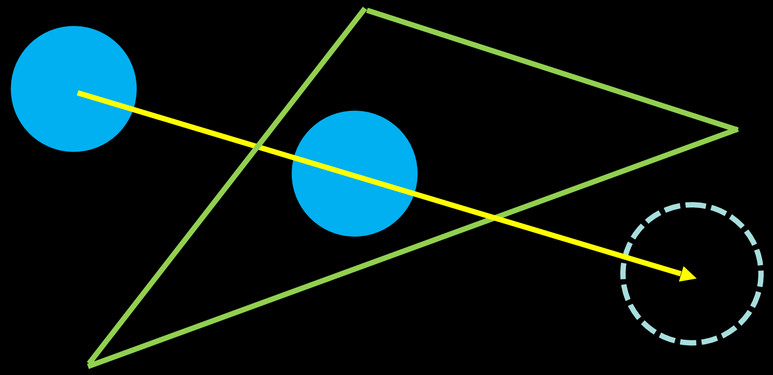
Collision Detection Against Shapes

- Particle-Convex
  - 2D hash-grid
  - Built on GPU



Convex Collision (MTD)

- Particle-Triangle Mesh
  - 3D hash-grid
  - Rasterized in CUDA
  - Lollipop test (CCD)



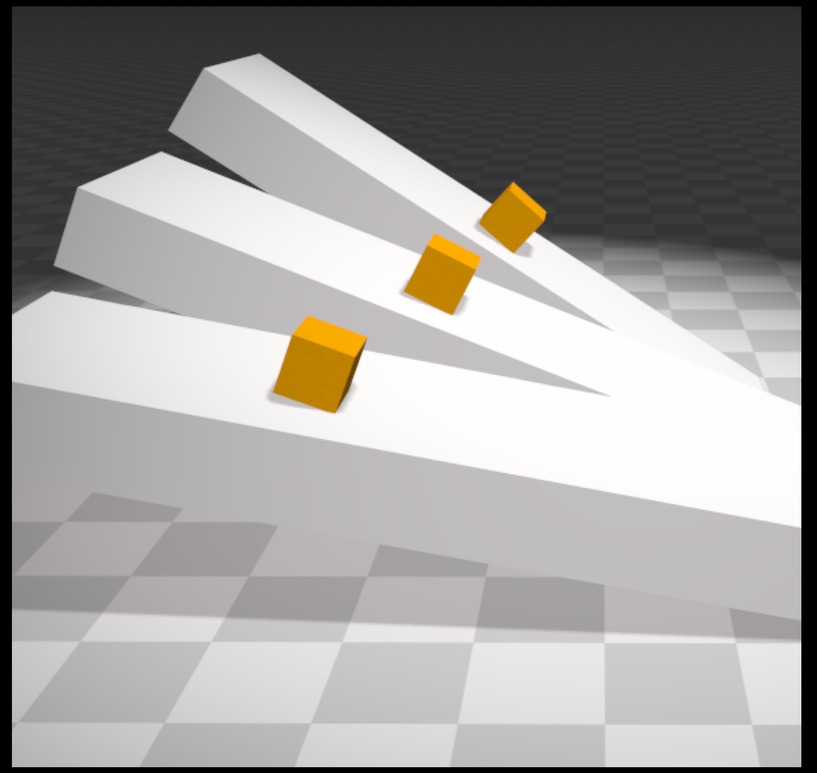
Triangle Collision (TOI)

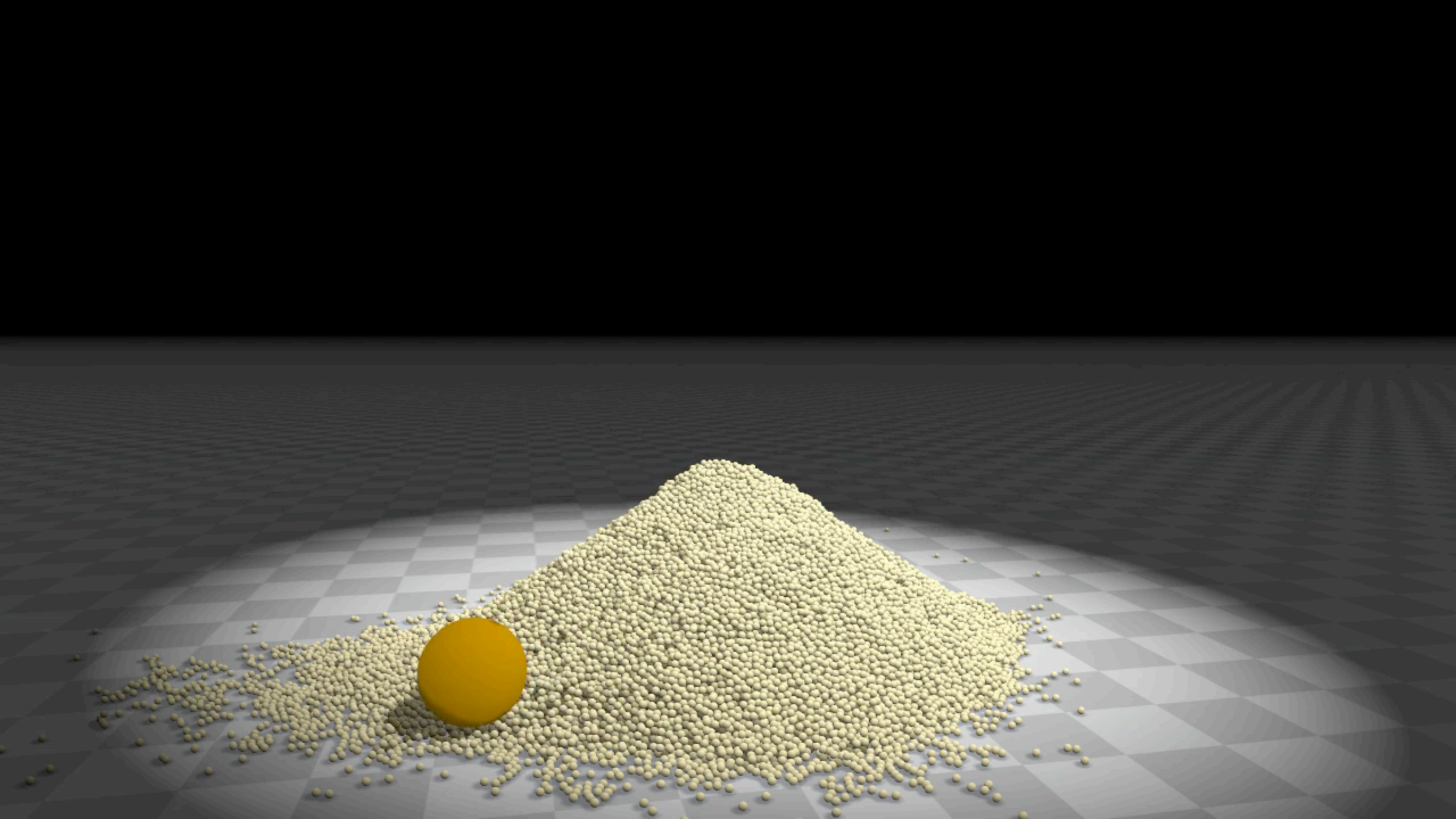
Friction

- Friction in PBD traditionally applied using a velocity filter
- Replace with a position-level frictional constraint

$$C_{friction} = |(\mathbf{x} - \mathbf{x}_0) \perp \mathbf{n}|$$

- Approximate Coulomb friction using penetration depth to limit constraint lambda
- Generates convincing particle piling

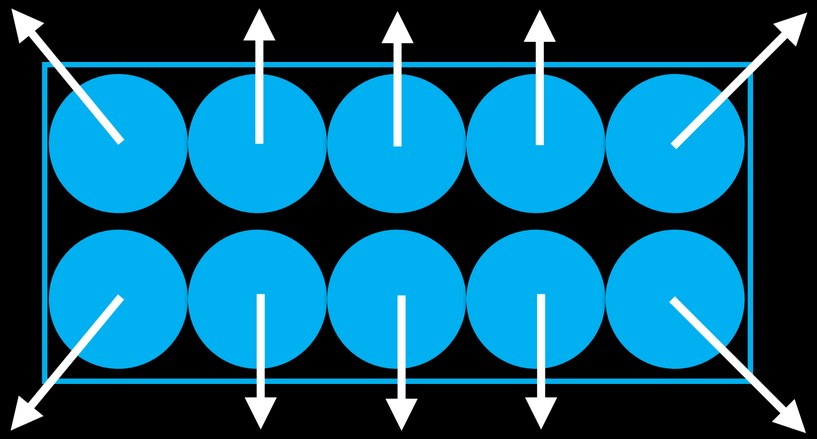


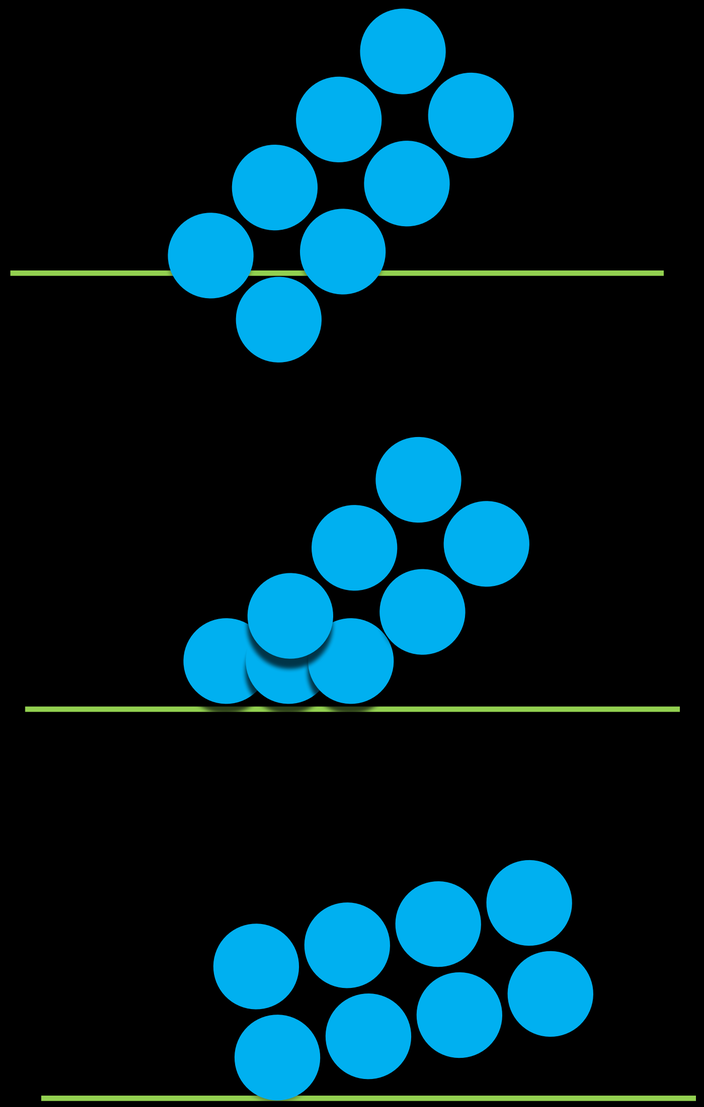


Rigid Bodies

Rigid Bodies

- Convert mesh->SDF
- Place particles in interior
- Add shape-matching constraint
- Store SDF dist + gradient on particles

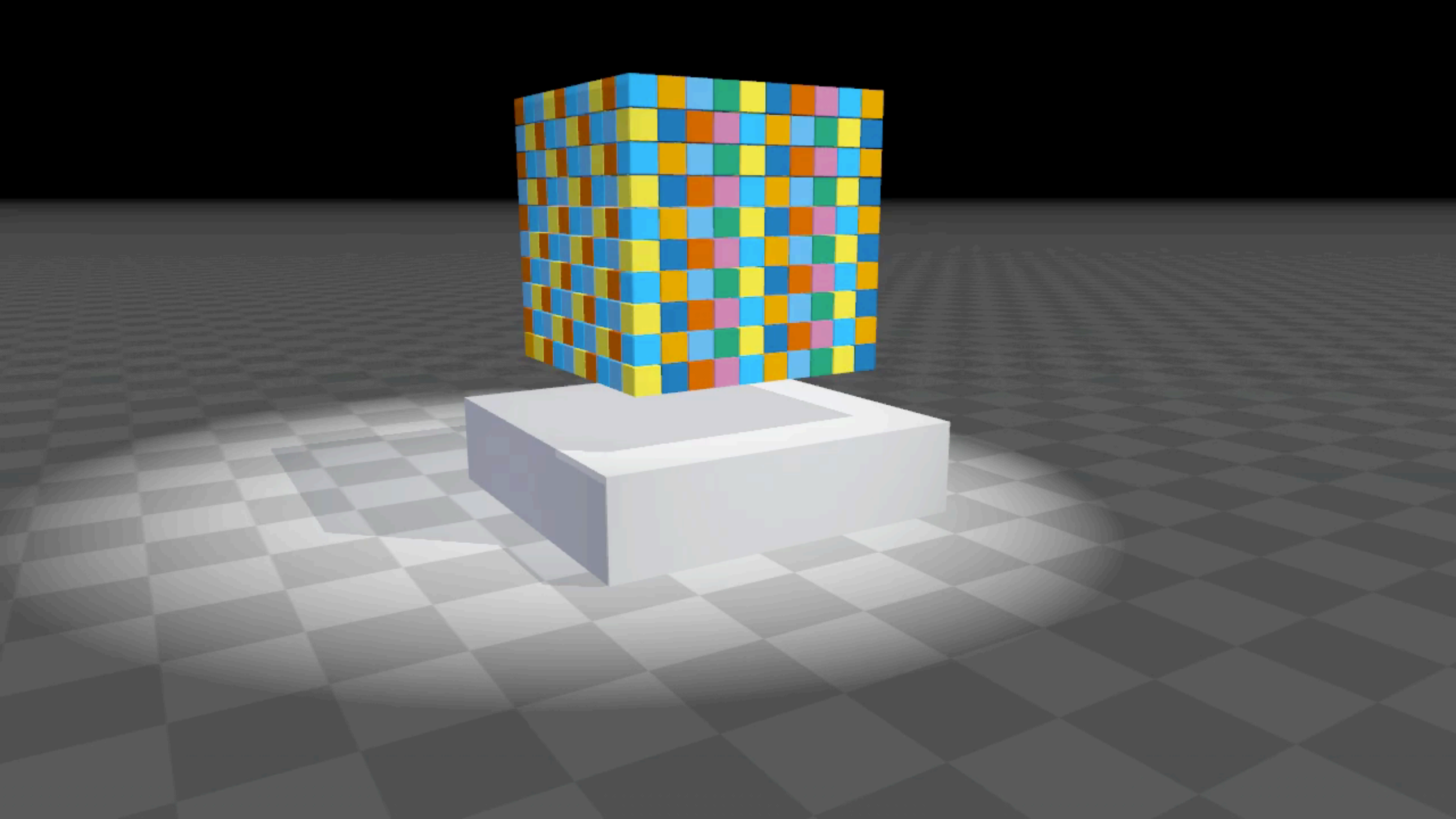




Rest Configuration

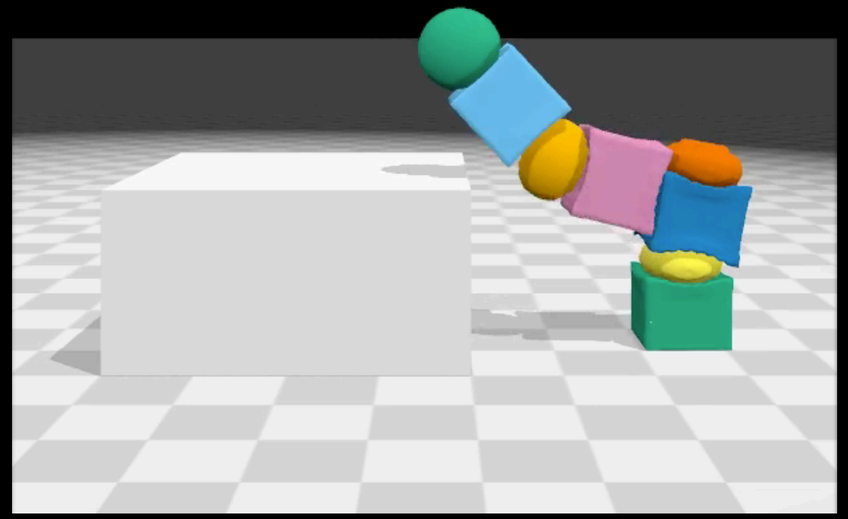
Deformed State

Best Rigid Rotation/ Translation



Plastic Deformation

- Detect when deformation exceeds a threshold
- Simply change rest-configuration of particles
- Adjust visual mesh (linear skinning)

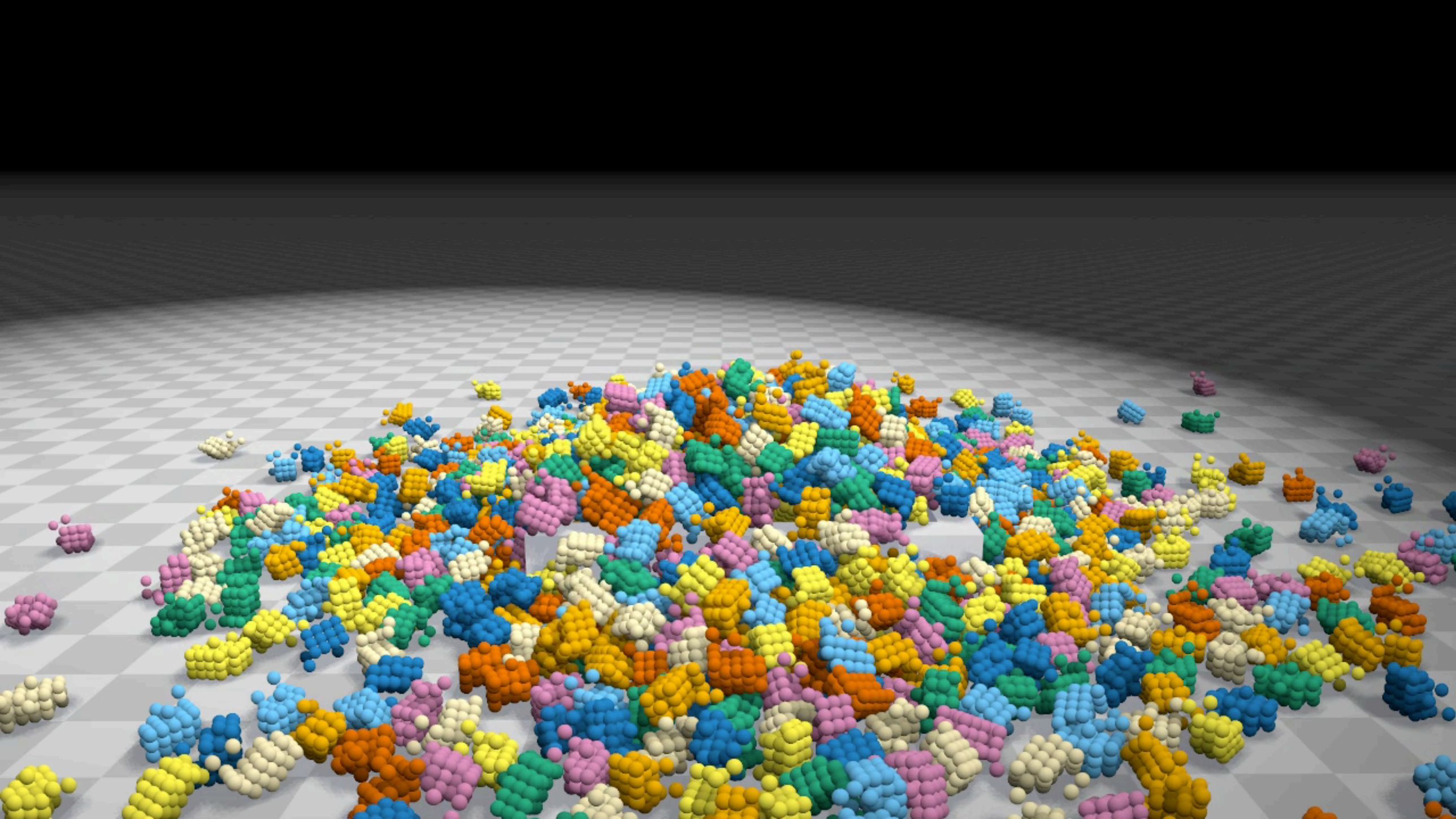


Shape matching on the GPU

- Shape matching requires computing centre of mass and the moment matrix for particles:

$$\mathbf{c} = \sum_i m_i \mathbf{x_i} / \sum_i m_i \qquad \mathbf{A} = \sum_i m_i (\mathbf{x_i} - \mathbf{c})(\bar{\mathbf{x}}_\mathbf{i} - \bar{\mathbf{c}})^\mathbf{T}$$

- Large summations, not immediately parallel friendly
- Optimized using two parallel cub::BlockReduce calls
- O(N) -> O(logN) (18ms -> 0.6ms)
- 1 block per-rigid shape (64 threads, heuristic, irregular workload problem)
- Polar decomposition still single threaded

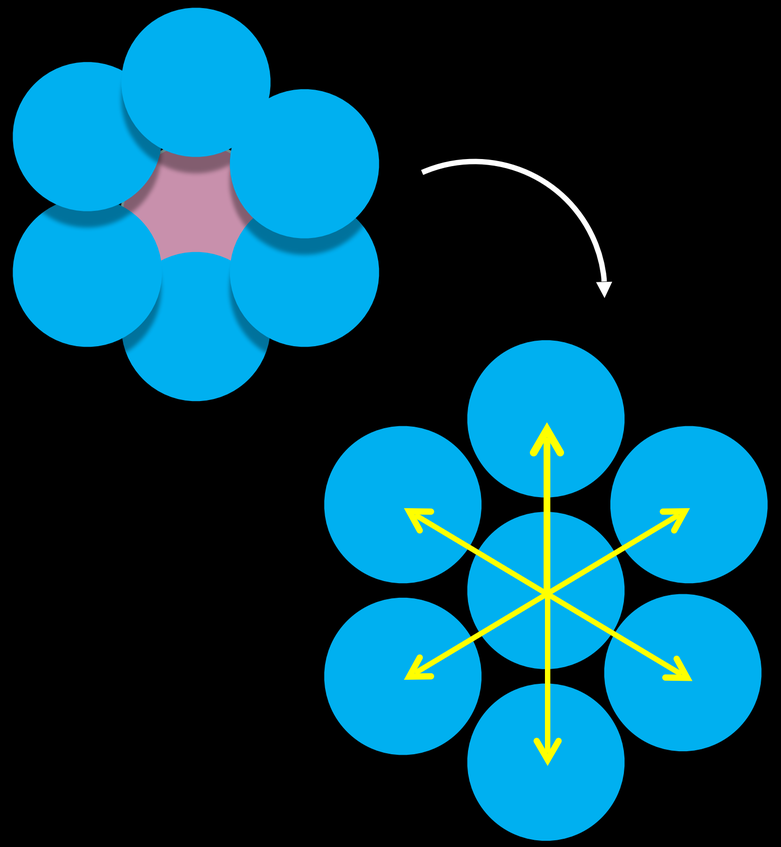


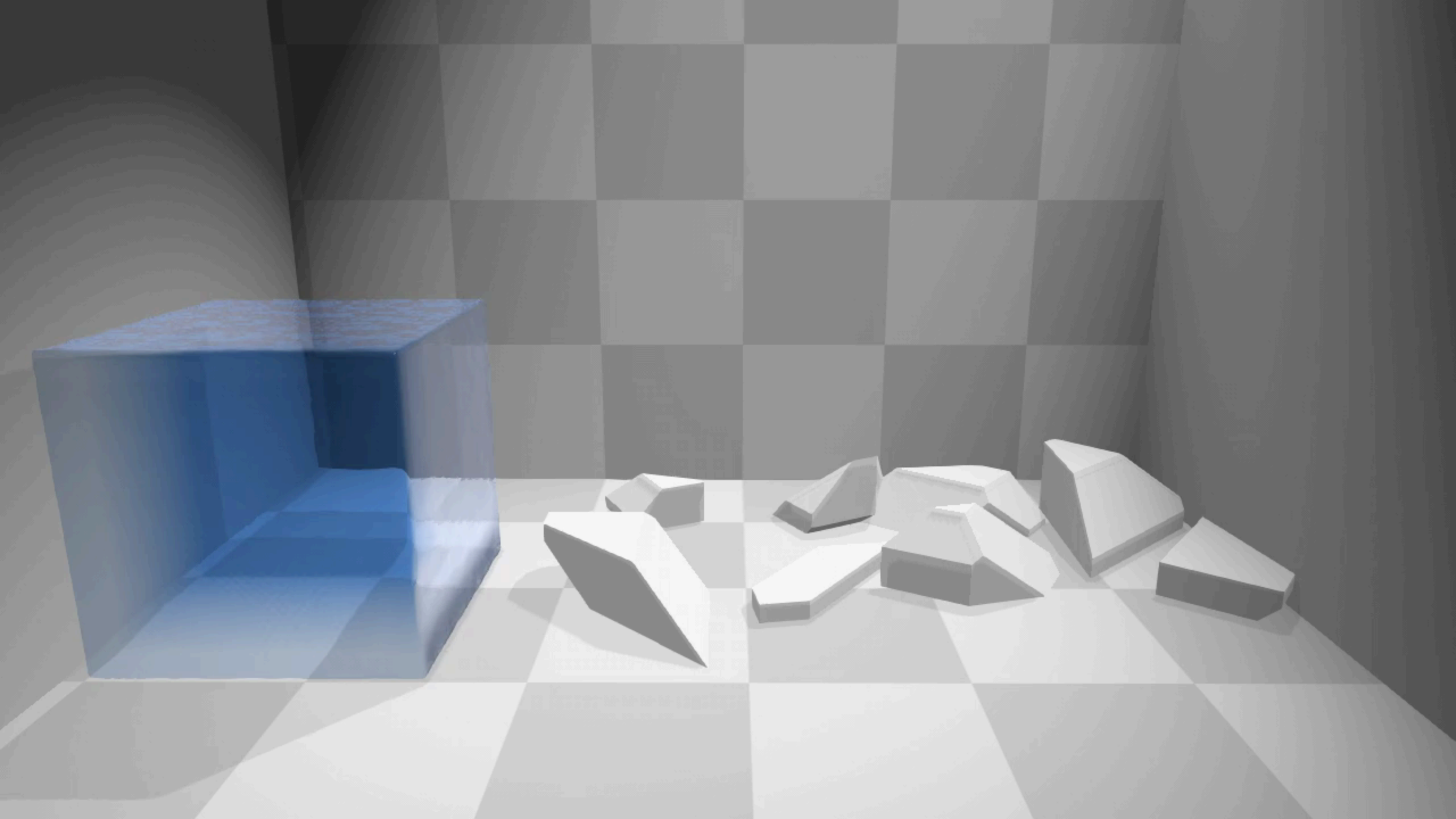
Fluids

Density Constraint

$$C_{density} = \frac{\rho_i}{\rho_0} - 1 \leq 0$$

- Density via SPH kernels
- Unilateral constraint
- Cohesion from [Akinci13]

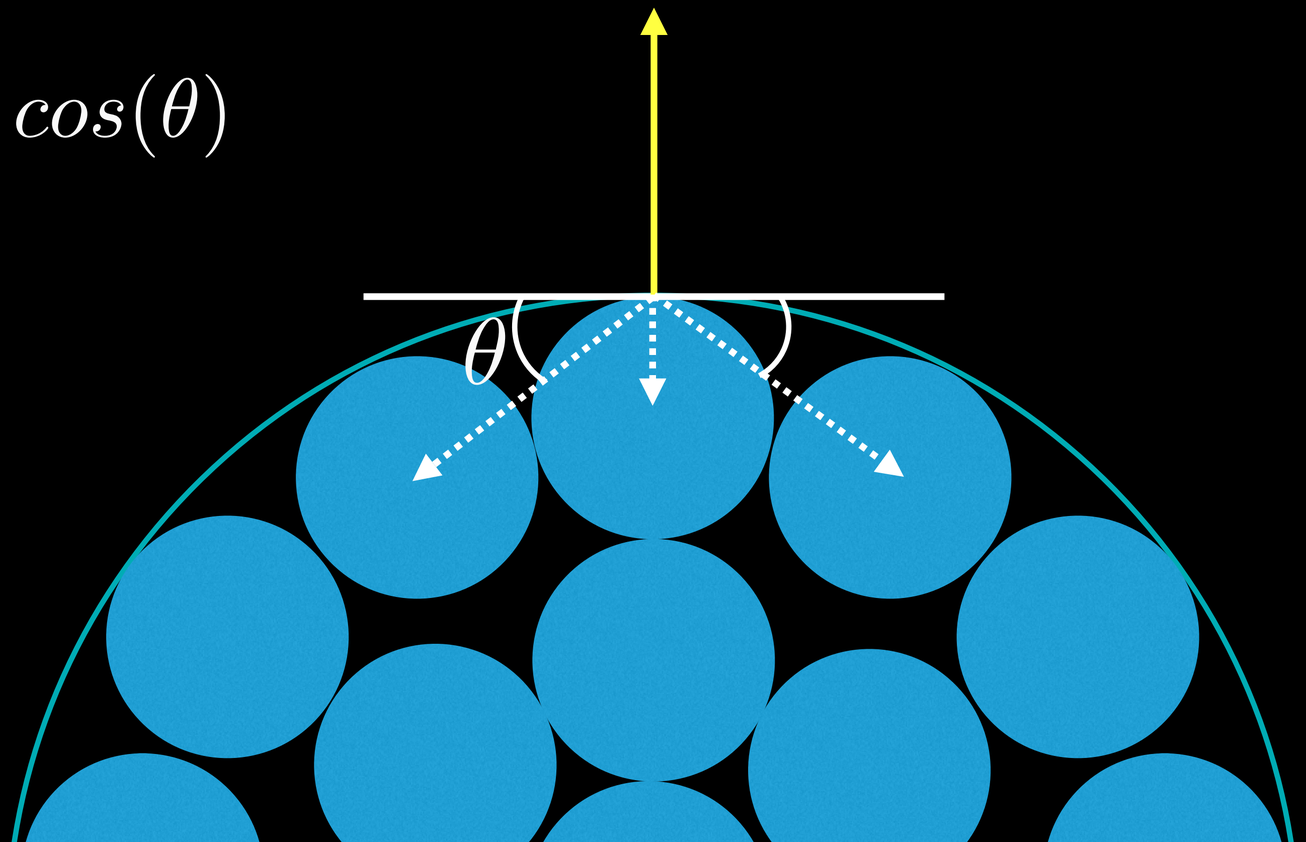


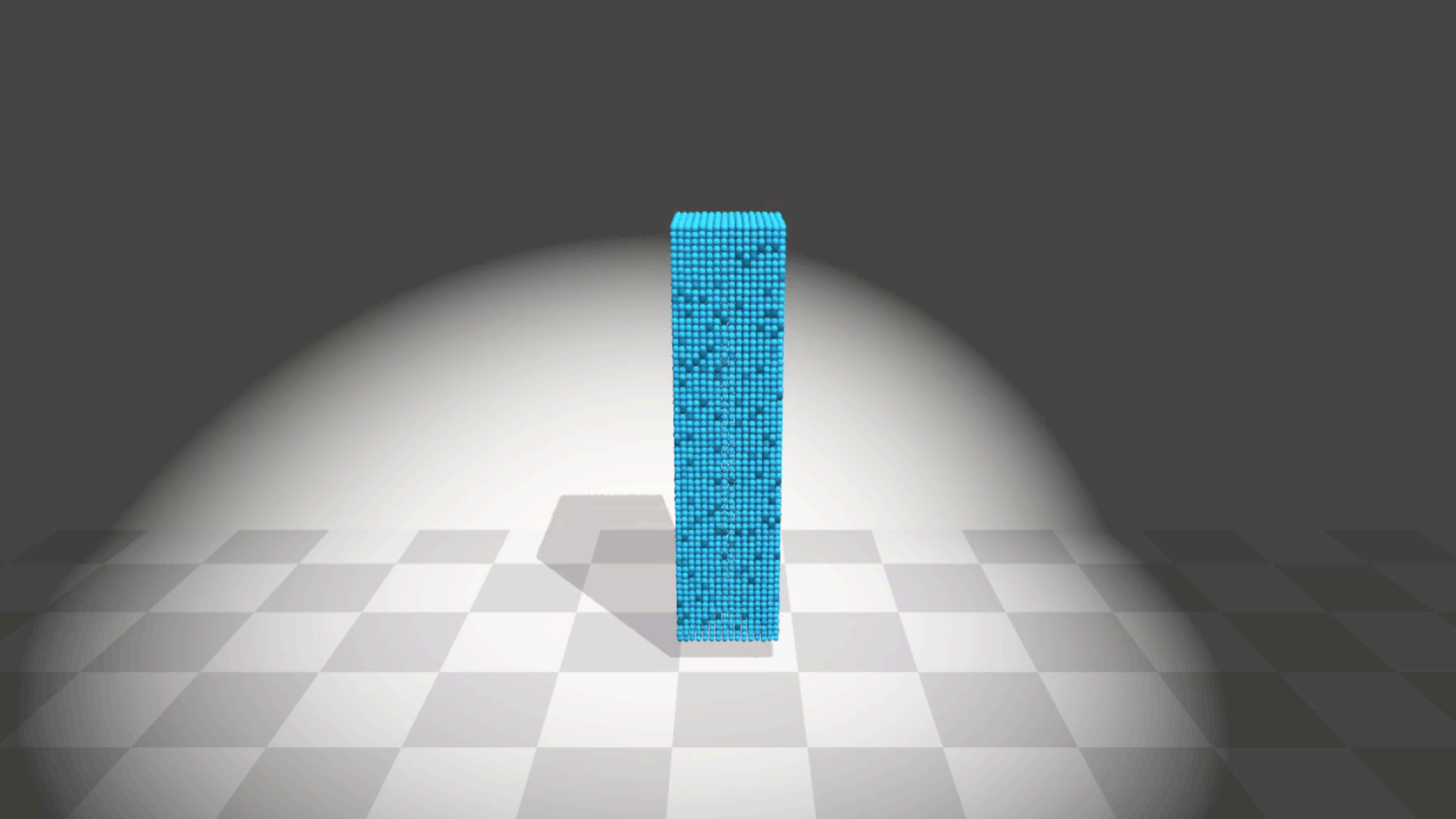


Surface Tension Constraint

- Adapted surface tension model of [Akinci13] to PBD
- Attempts to minimize curvature

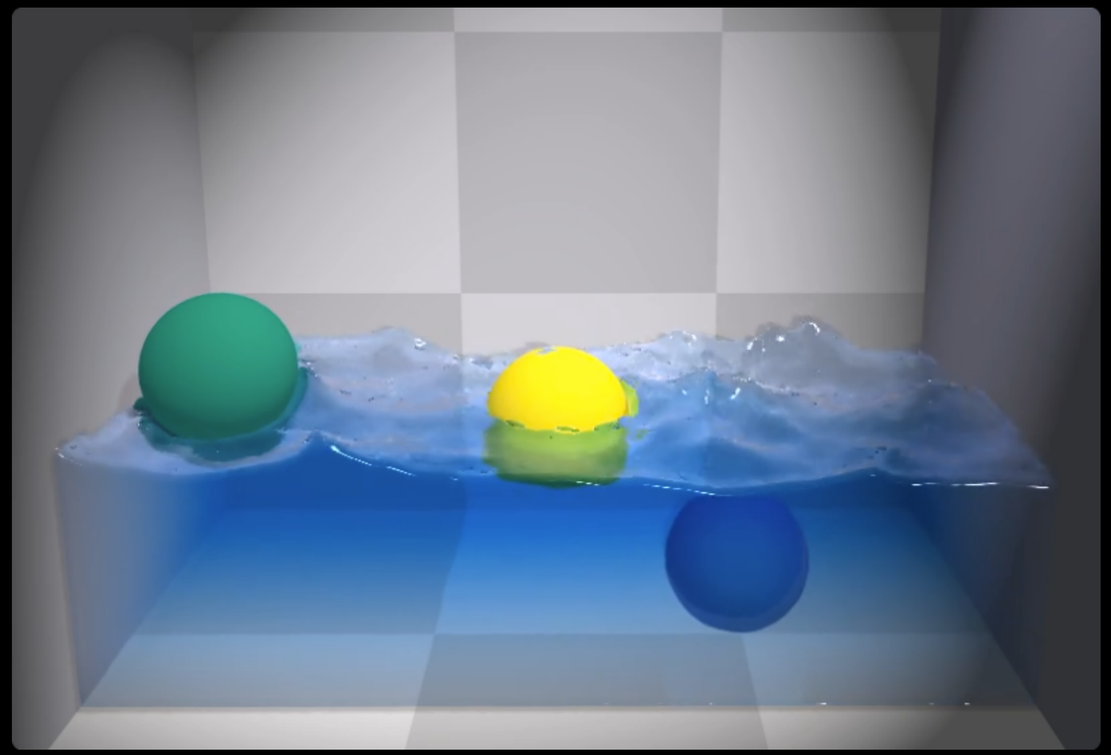
$$C_{tension} = \bar{\mathbf{x}}_{\mathbf{ij}} \cdot \bar{\mathbf{n}}_{\mathbf{i}} = cos(\theta)$$





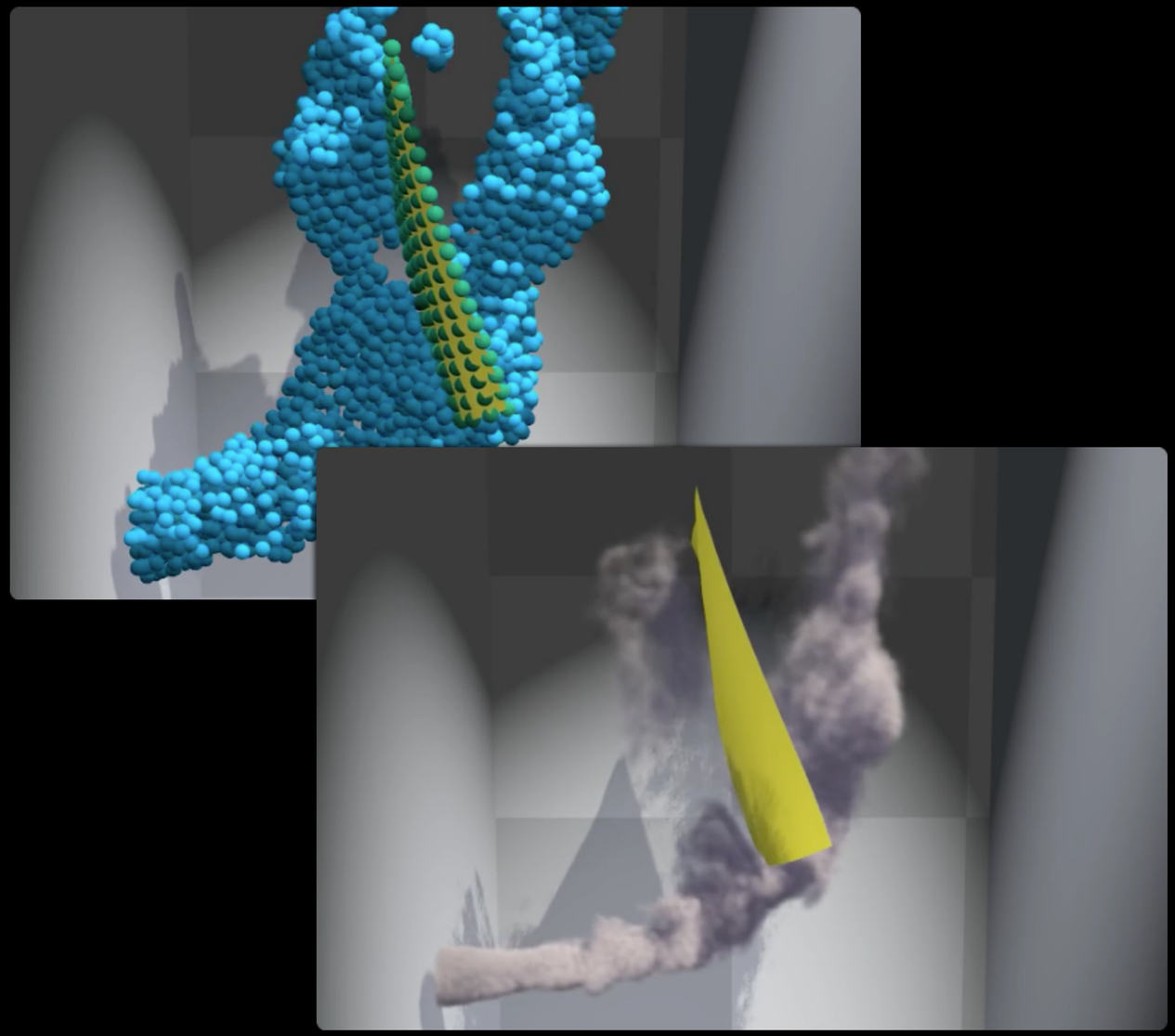
Two-Way Rigid Fluid Coupling

- Mostly automatic
- Include all particles in fluid density estimation
- Treat fluid->solid particle interactions as if both particles solid



Gases

- Treat as an incompressible fluid with density constraint
- Sparse representation
- Passive smoke advection (“diffuse particles”)



Gas Forces

Free surface drag

$\bar{\mathbf{f}}_{\mathbf{drag}}$

Density gradient via SPH derivatives:

$$\nabla \rho = \bar{\mathbf{n}}$$

Pressure gradient via Boussinesq approximation:

$$\nabla p = \bar{\mathbf{g}}$$

$\bar{\mathbf{v}}_{\mathbf{0}}$

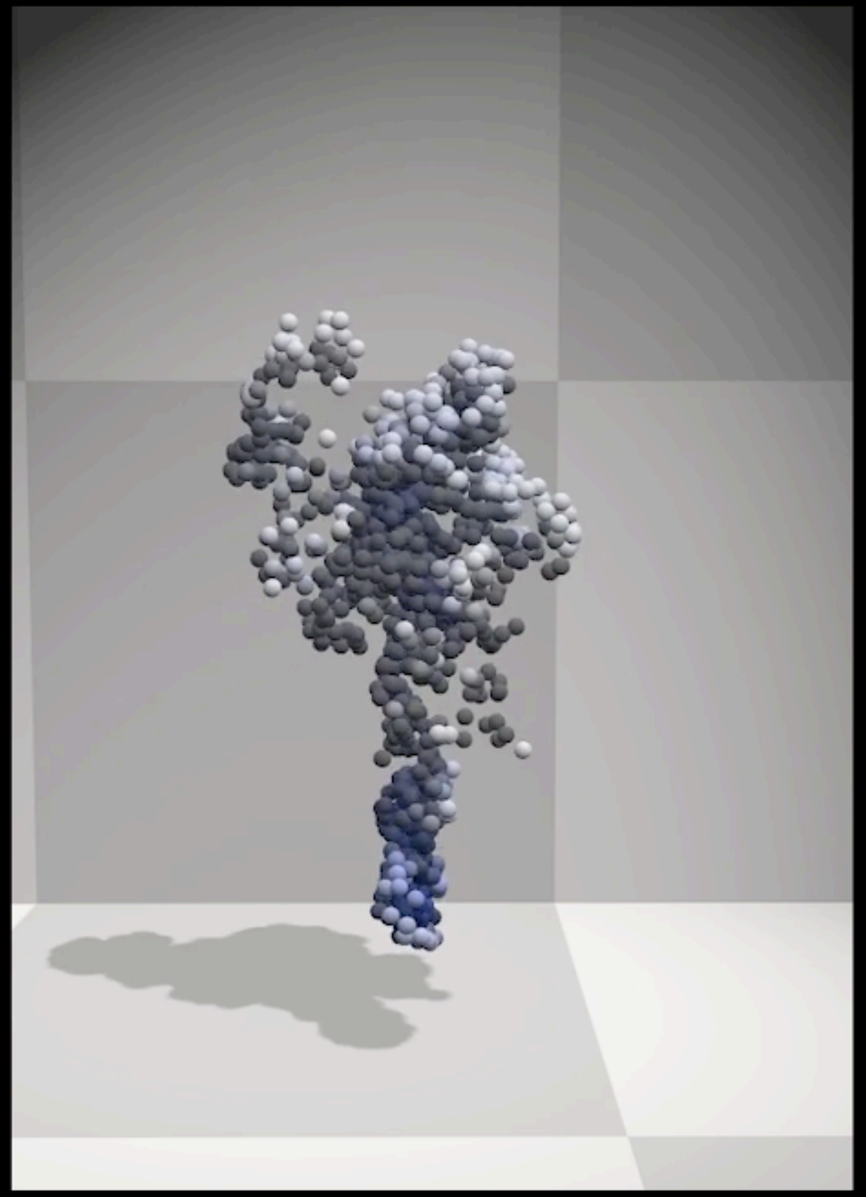
Baroclinic vorticity:

$$\frac{\mathrm{D}\bar{\omega}}{\mathrm{d}t} = \nabla \rho \times \nabla p$$

Driving vorticity:

$$\bar{\mathbf{f}}_{\mathbf{vort}} = \bar{\omega} \times \bar{\mathbf{x}}_{\mathbf{ij}}$$

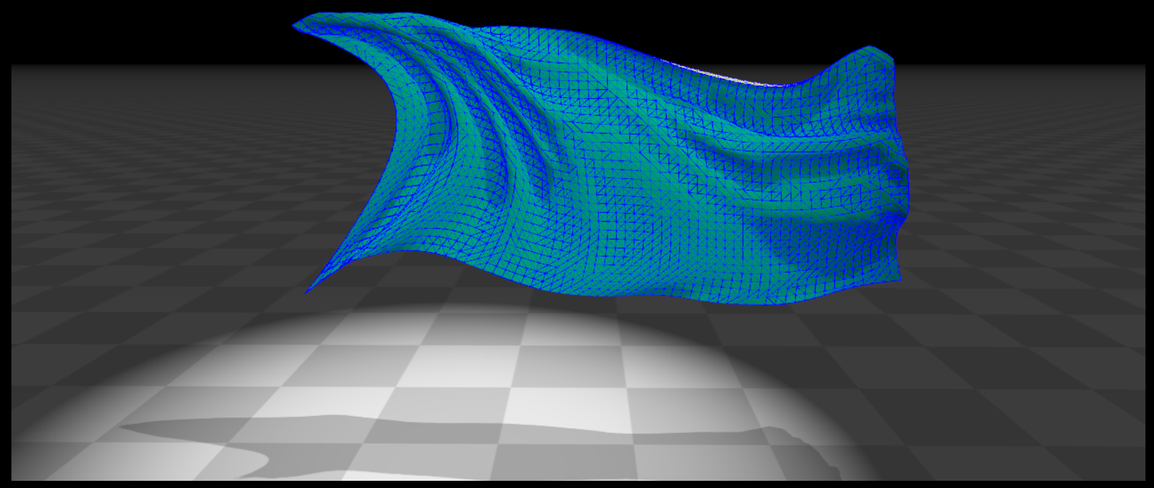
Smoke Particles

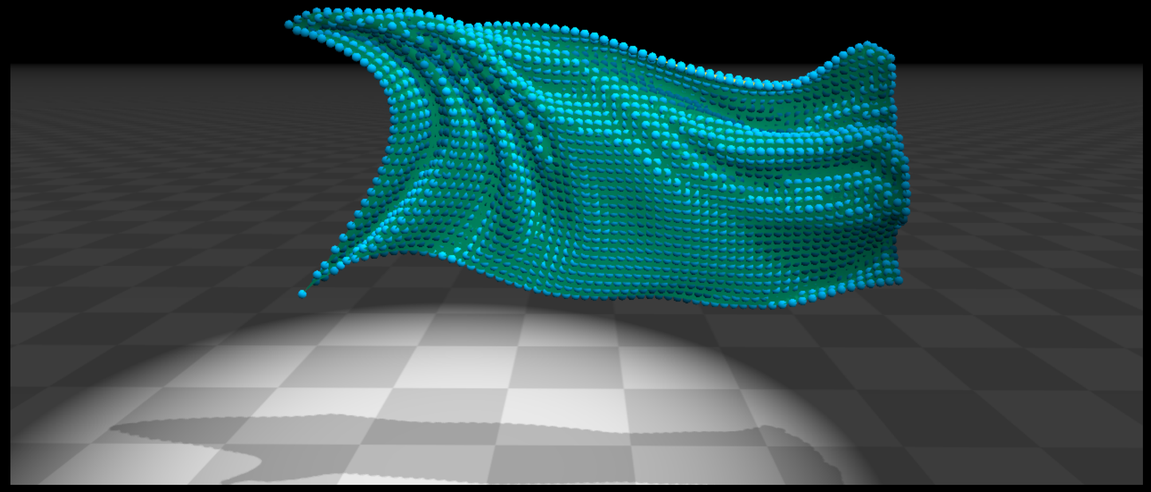


Fluid Particles

Cloth

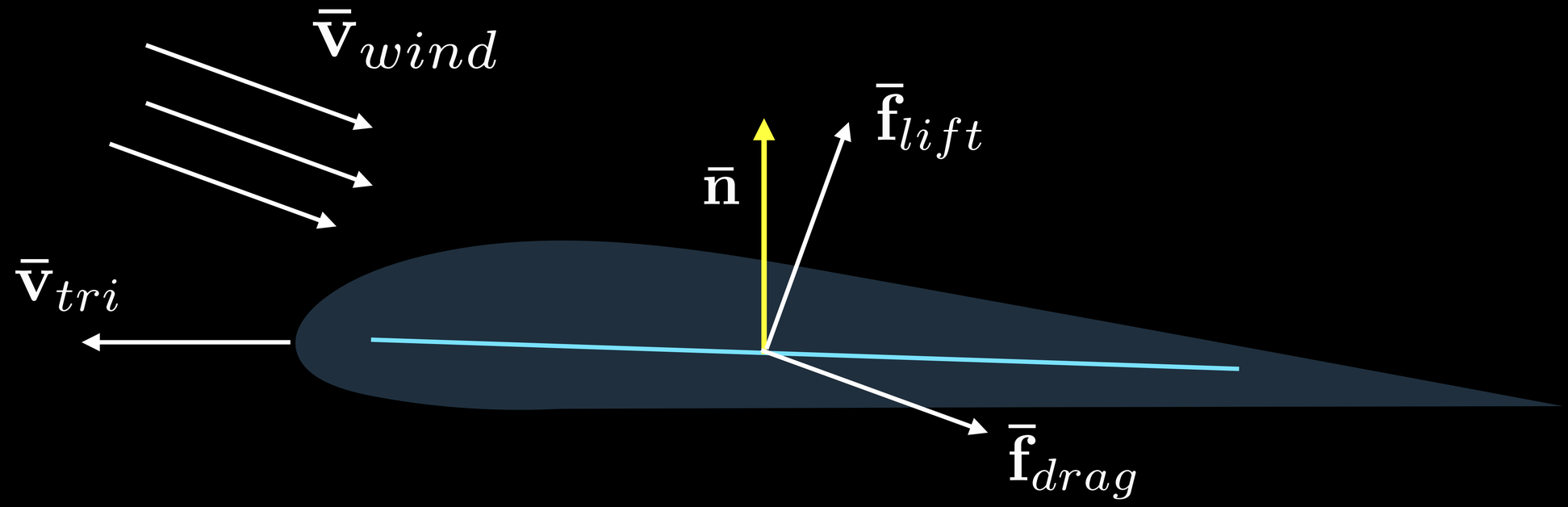
- Graph of distance + tether constraints
- Self-collision / inter-collision automatically handled

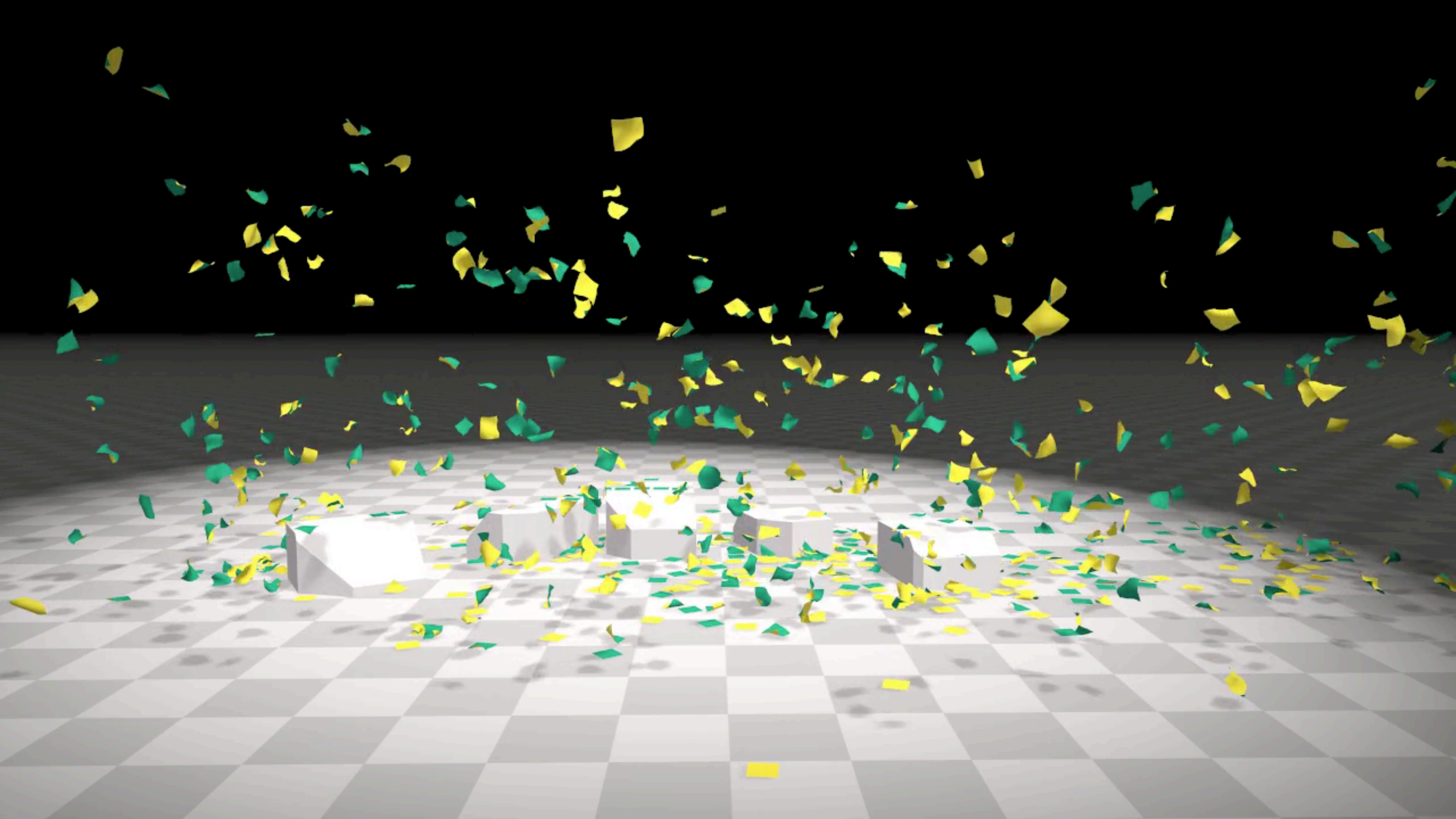




Cloth - Forces

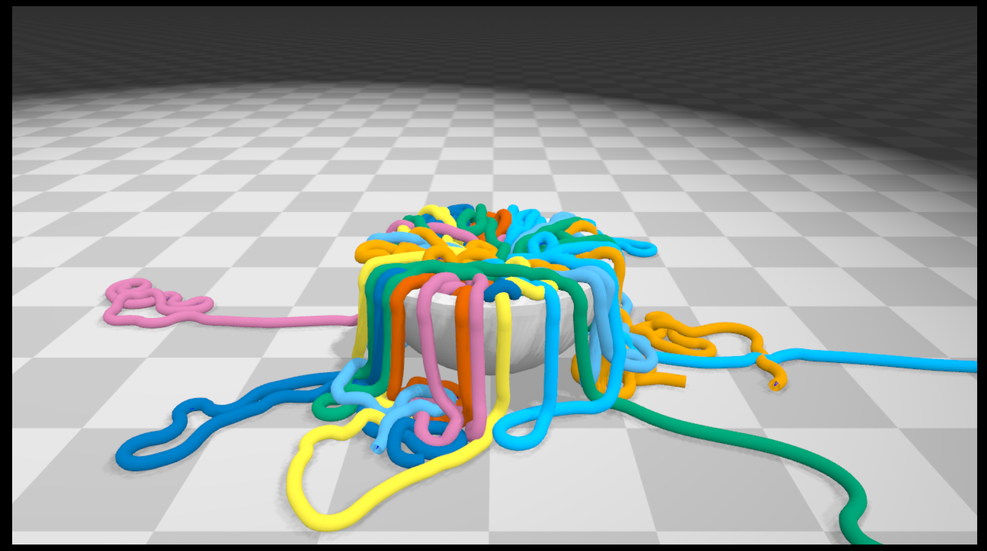
- Basic aerodynamic model
- Treat each triangle as a thin airfoil to generate lift + drag
- Flexible enough to model paper planes

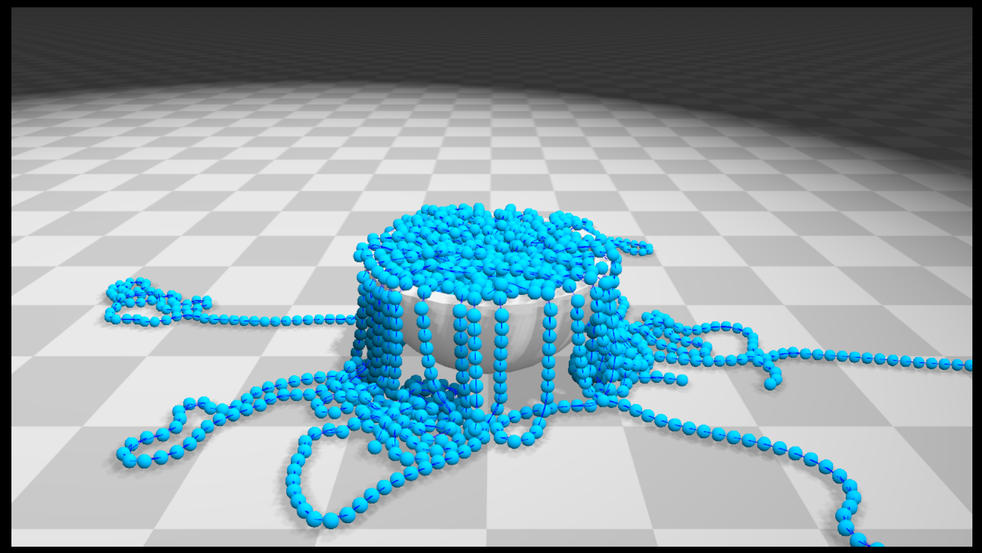


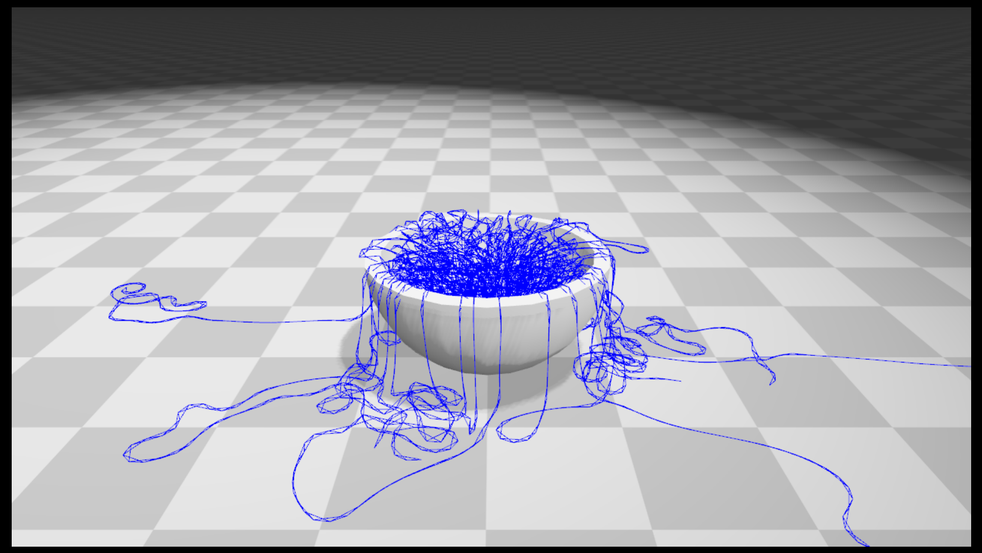


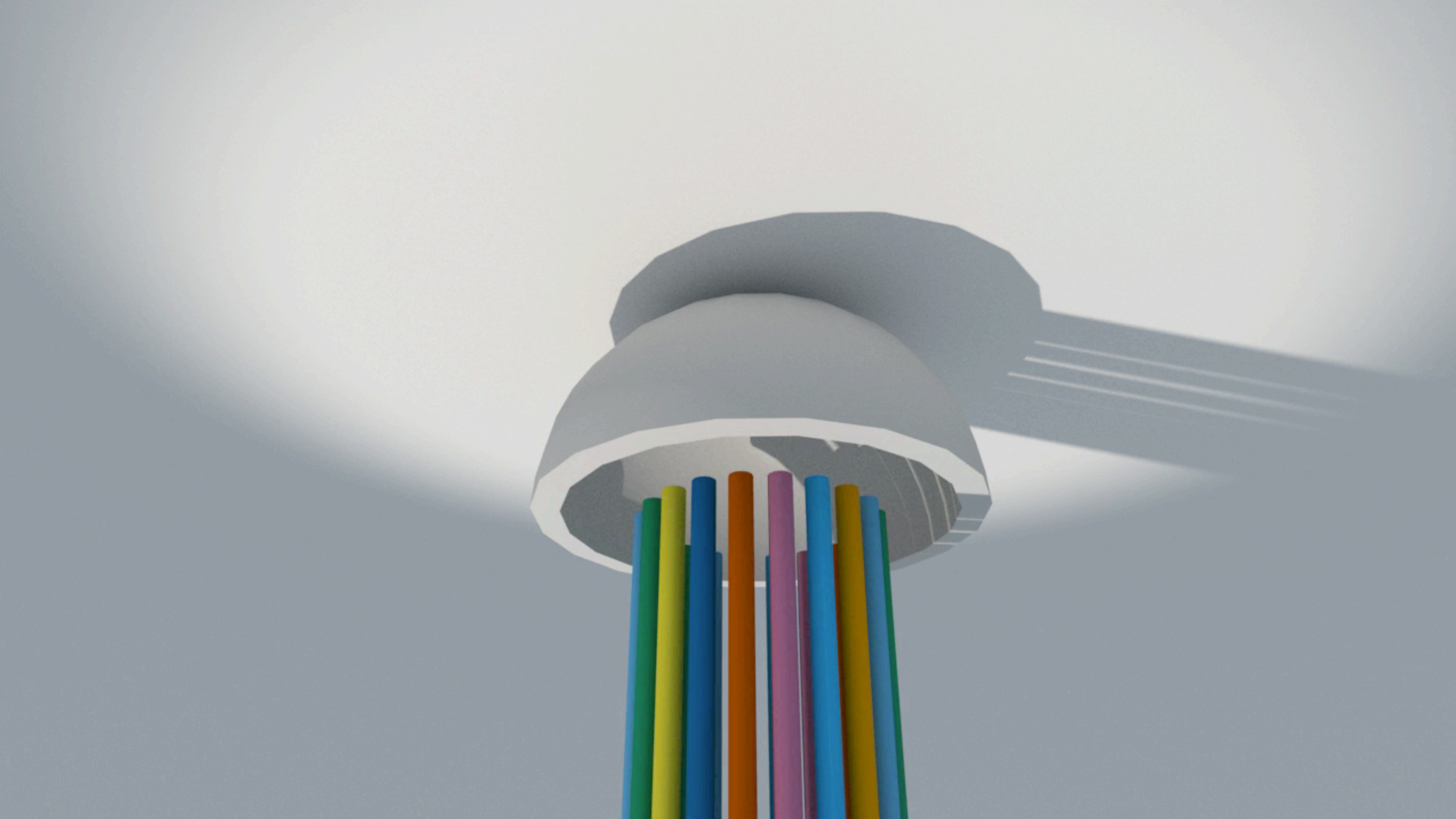
Ropes

- Build ropes from distance + bending constraints
- Fit Catmull-Rom spline to points
- Torsion possible [Umetani 14]

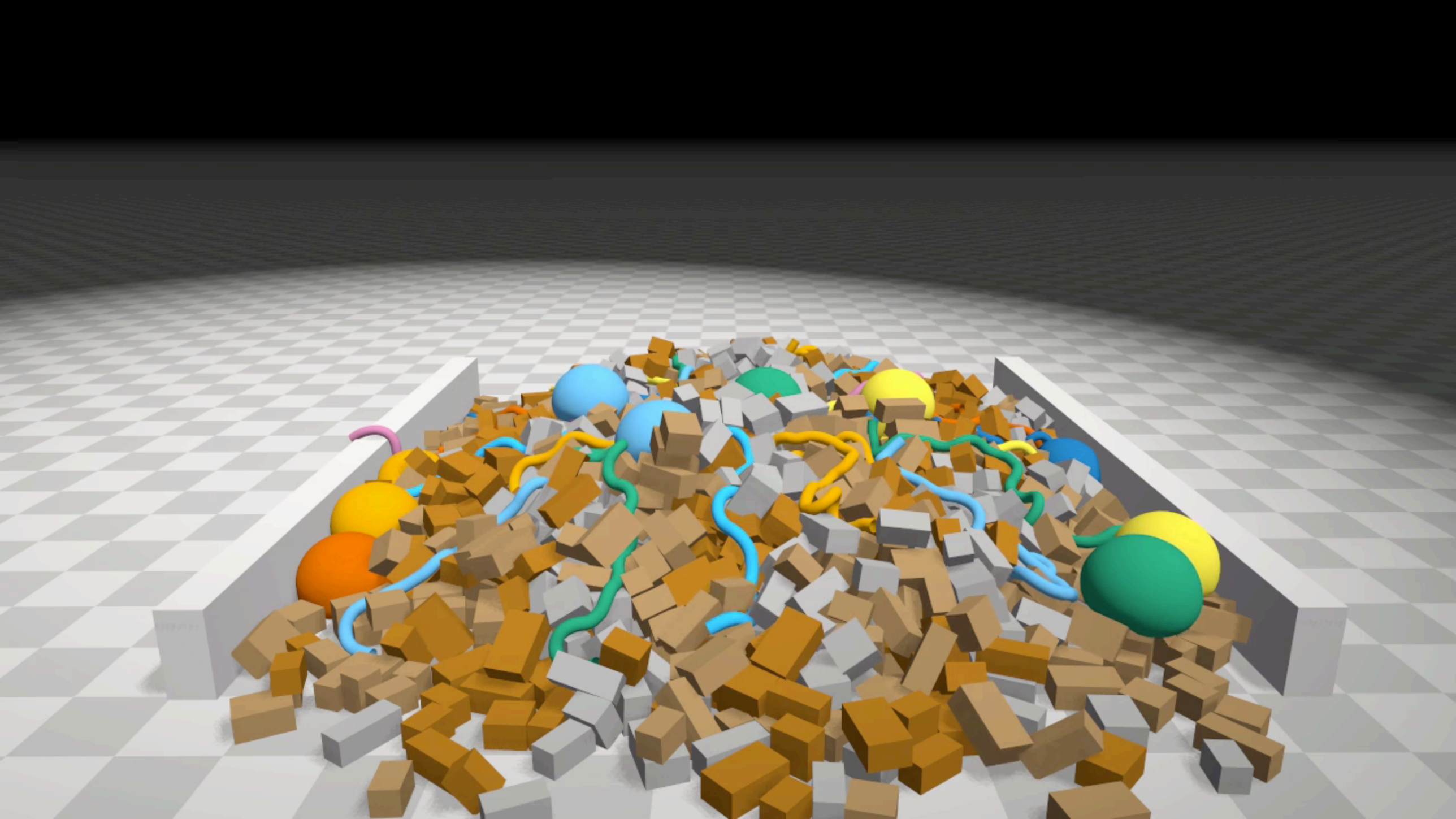


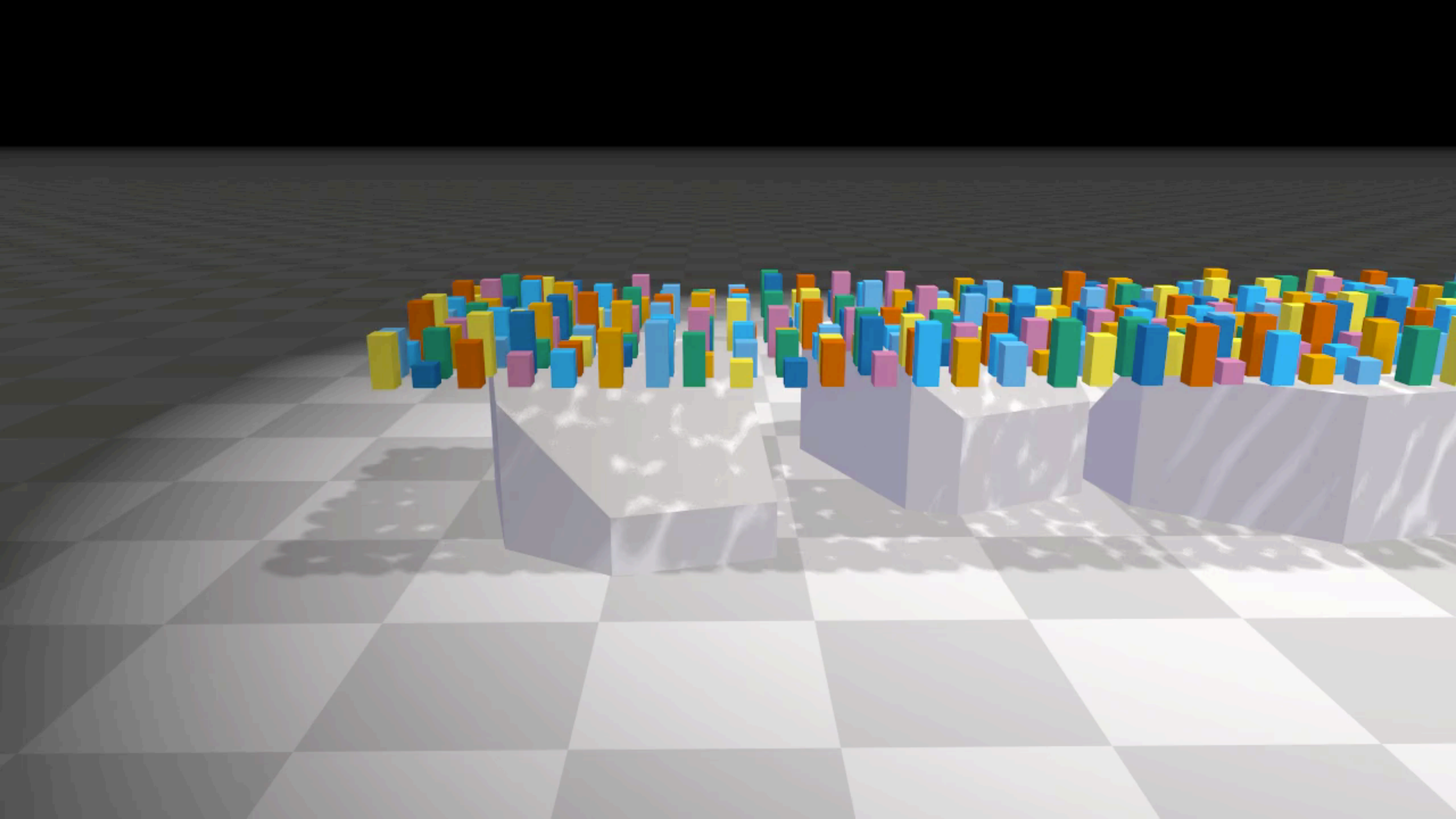


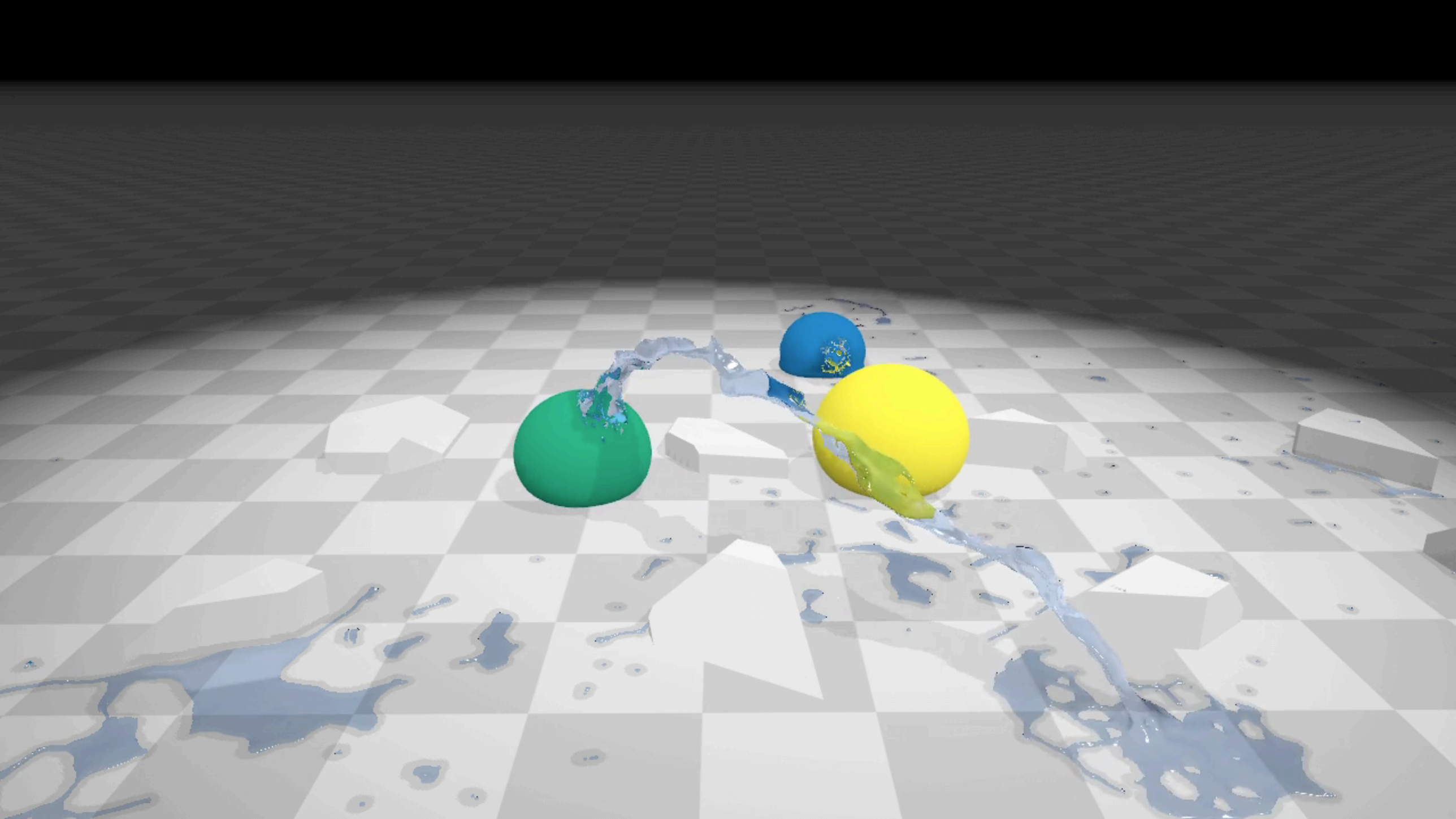




Examples







Limitations / Future Work

- Representing smooth surfaces problematic
- Want parallel and robust collision of simplices
- Hierarchical representation (multi-scale particles)
- Convergence for parallel solver

Questions?

References

- English, Elliot, and Robert Bridson. "Animating developable surfaces using nonconforming elements." ACM Transactions on Graphics (TOG). Vol. 27. No. 3. ACM, 2008.
- Goldenthal, Rony, et al. "Efficient simulation of inextensible cloth." ACM Transactions on Graphics (TOG) 26.3 (2007): 49.
- Bouaziz, Sofien, et al. "Projective dynamics: fusing constraint projections for fast simulation." ACM Transactions on Graphics (TOG) 33.4 (2014): 154.
- Bridson, Robert, Ronald Fedkiw, and John Anderson. "Robust treatment of collisions, contact and friction for cloth animation." ACM Transactions on Graphics (ToG). Vol. 21. No. 3. ACM, 2002.
- Stam, Jos. "Nucleus: Towards a unified dynamics solver for computer graphics." Computer-Aided Design and Computer Graphics, 2009. CAD/Graphics' 09. 11th IEEE International Conference on. IEEE, 2009.
- Green, Simon. "Cuda particles." nVidia Whitepaper 2.3.2 (2008): 1.
- Guendelman, Eran, Robert Bridson, and Ronald Fedkiw. "Nonconvex rigid bodies with stacking." ACM Transactions on Graphics (TOG). Vol. 22. No. 3. ACM, 2003.
- Provot, Xavier. "Deformation constraints in a mass-spring model to describe rigid cloth behaviour." Graphics interface. Canadian Information Processing Society, 1995.
- Fratarcangeli, M., and F. Pellacini. "Scalable Partitioning for Parallel Position Based Dynamics." EUROGRAPHICS. Vol. 34. No. 2. 2015.
- Liu, Tiantian, et al. "Fast simulation of mass-spring systems." ACM Transactions on Graphics (TOG) 32.6 (2013): 214.
- Akinci, Nadir, Gizem Akinci, and Matthias Teschner. "Versatile surface tension and adhesion for SPH fluids." ACM Transactions on Graphics (TOG) 32.6 (2013): 182.
- Ryckaert, Jean-Paul, Giovanni Ciccotti, and Herman JC Berendsen. "Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes." Journal of Computational Physics 23.3 (1977): 327-341.
- Umetani, Nobuyuki, Ryan Schmidt, and Jos Stam. "Position-based elastic rods." ACM SIGGRAPH 2014 Talks. ACM, 2014.