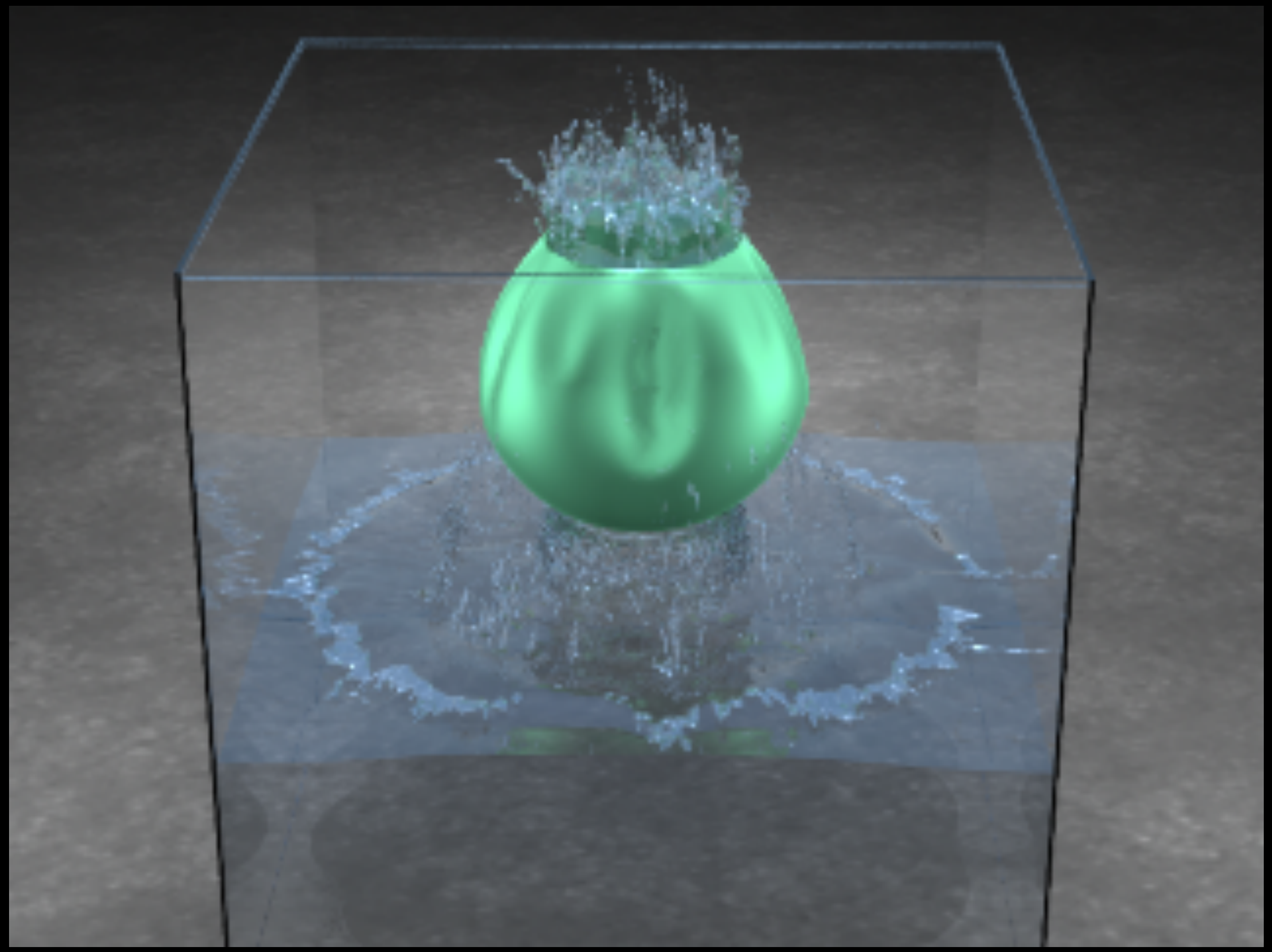
# Unified Particle Physics for Real-Time Applications

Miles Macklin, Matthias Müller, Nuttapong Chentanez, Tae-Yong Kim

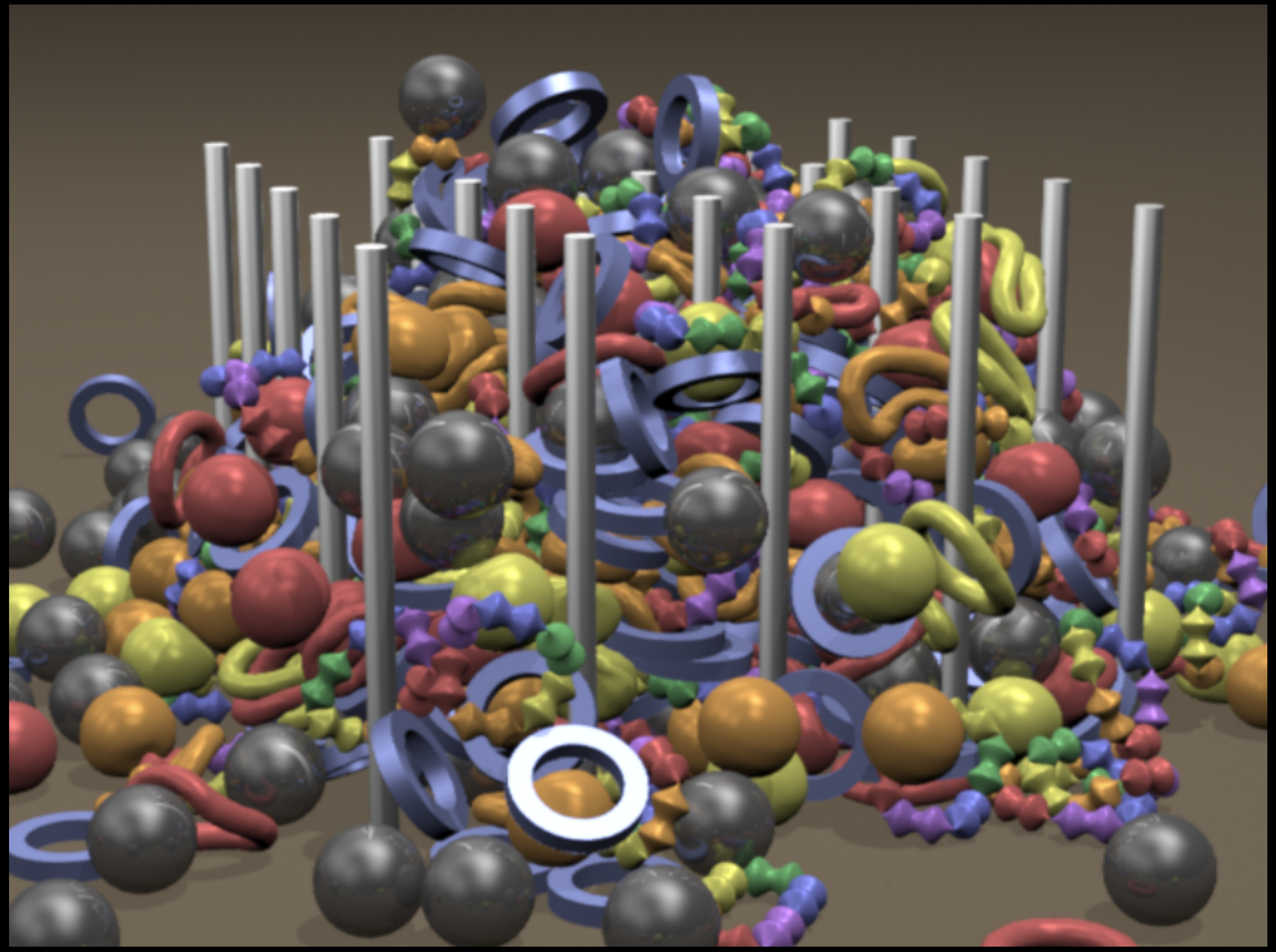
NVIDIA

# Motivation

- Too many solvers
- Creates redundant work
- Want two-way interaction between all object types



[Robinson-Mosher et al. 2008]



[Shinar et al. 2008]

# Core Idea

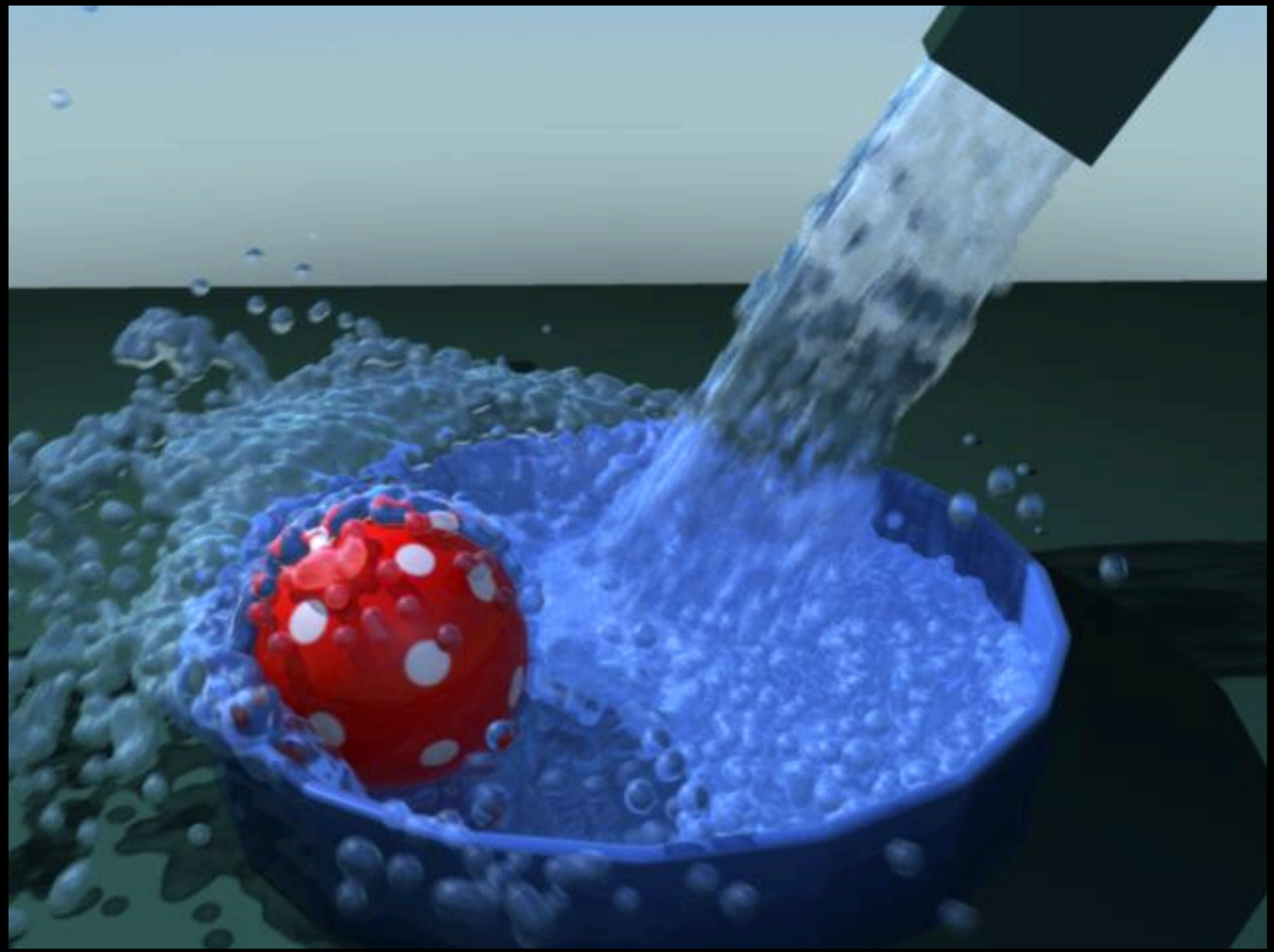
***Everything is a set of particles connected by constraints***

# Advantages

- Simplifies collision detection
- Stable two-way interaction of all object types:
  - Cloth
  - Deformables
  - Liquids
  - Gases
  - Rigid Bodies
- Fits well on the GPU

# Related Work

- Unified solvers popular in offline visual effects, e.g.:
  - Maya’s Nucleus solver (nCloth, nParticles) [Stam09]
  - Softimage's Lagoa (fluids, elastics, granular materials)
- Goal: recreate these packages in real-time



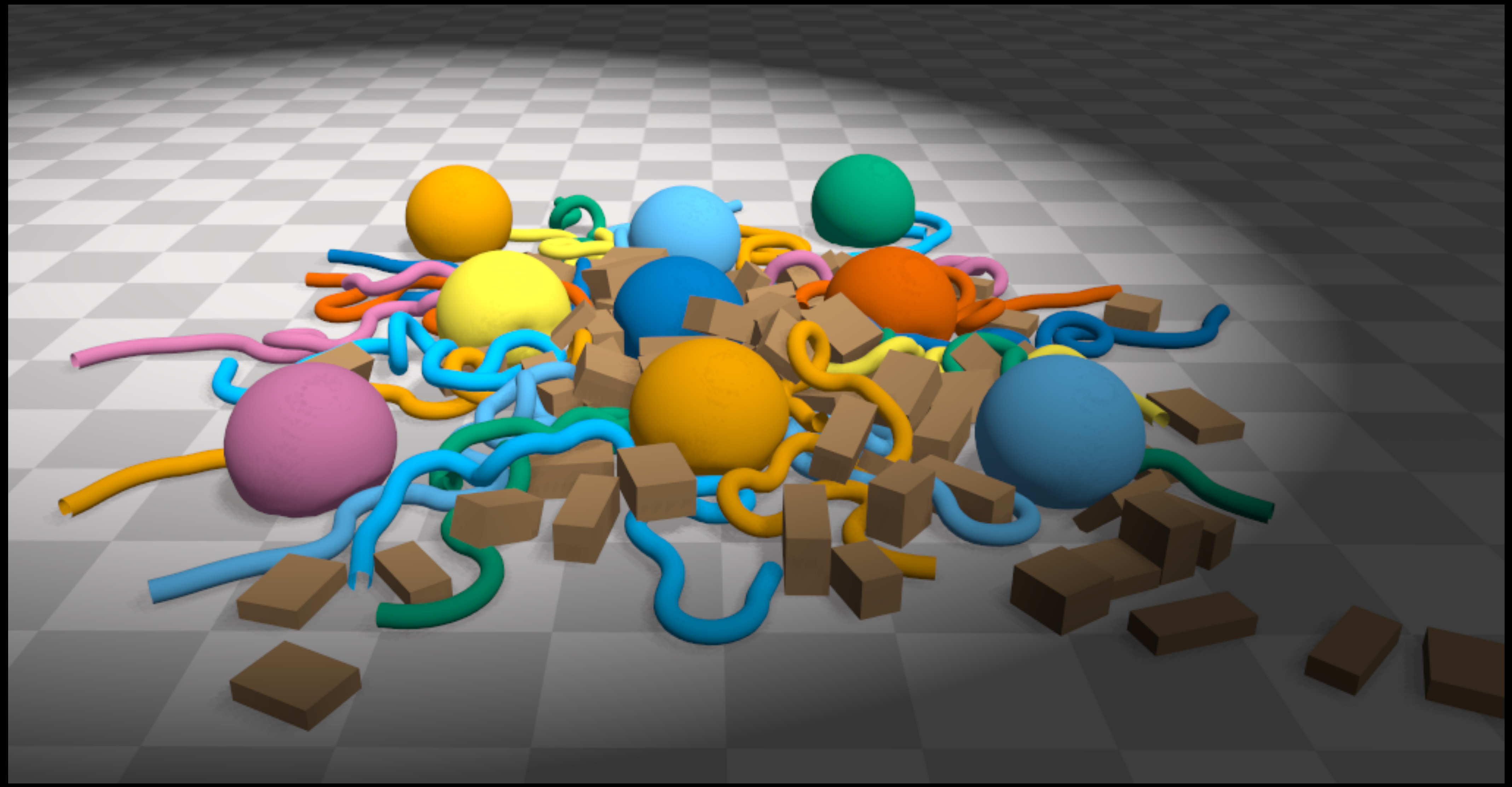
Maya nDynamics

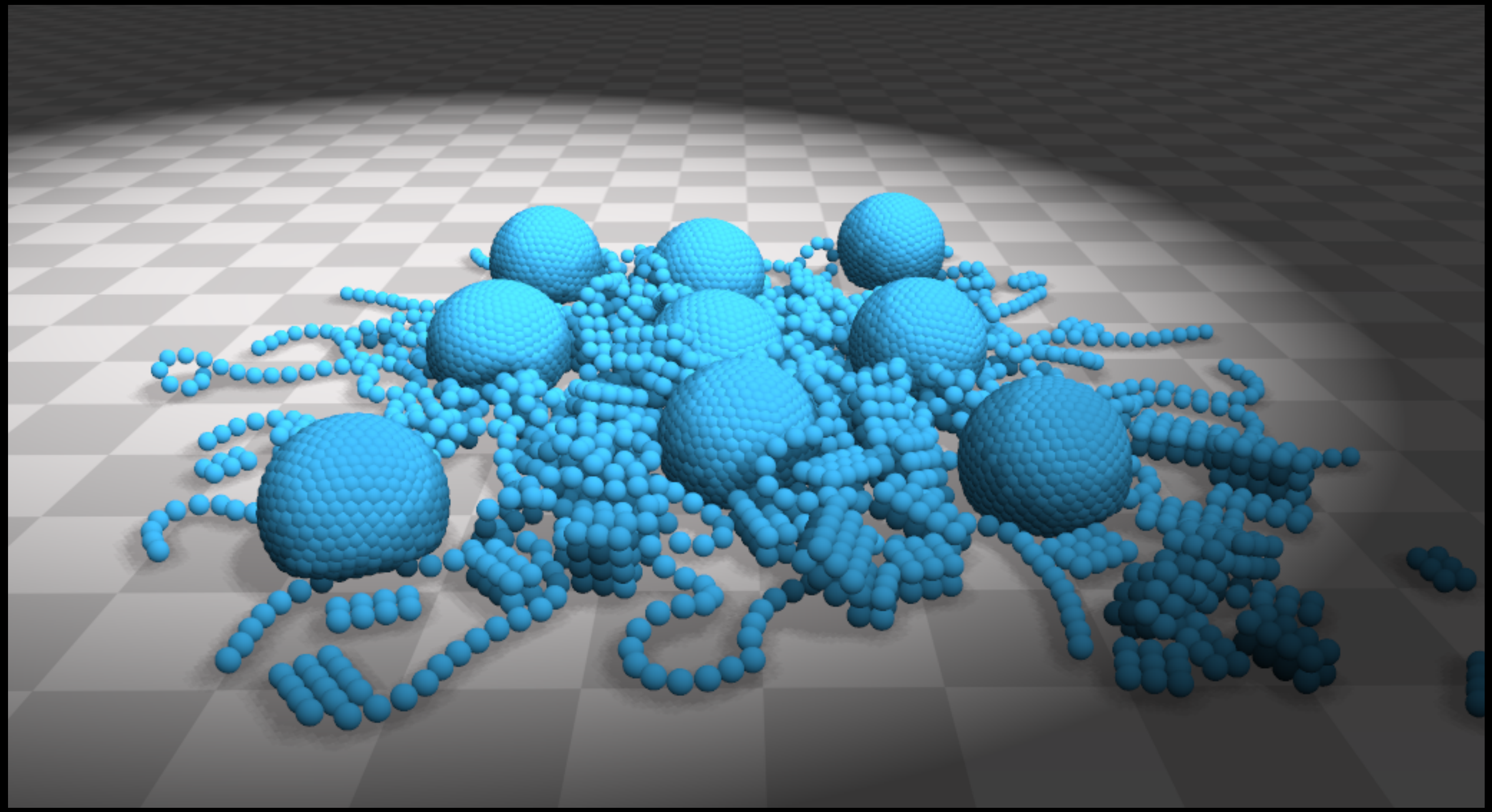
# Examples

# Particles

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

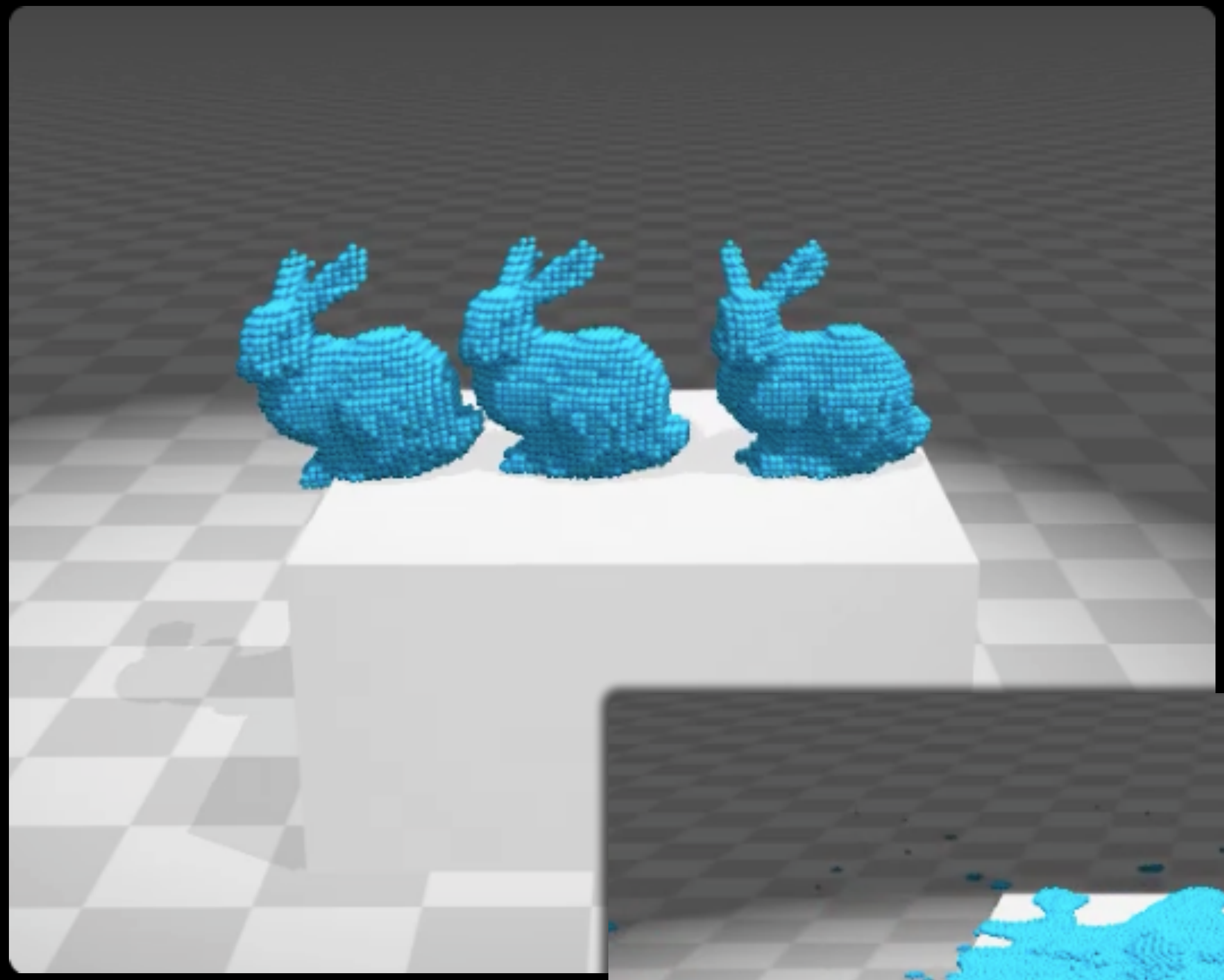
- Phase-ID used to control collision filtering
- Particles do not belong to a particular object
- Single collision radius

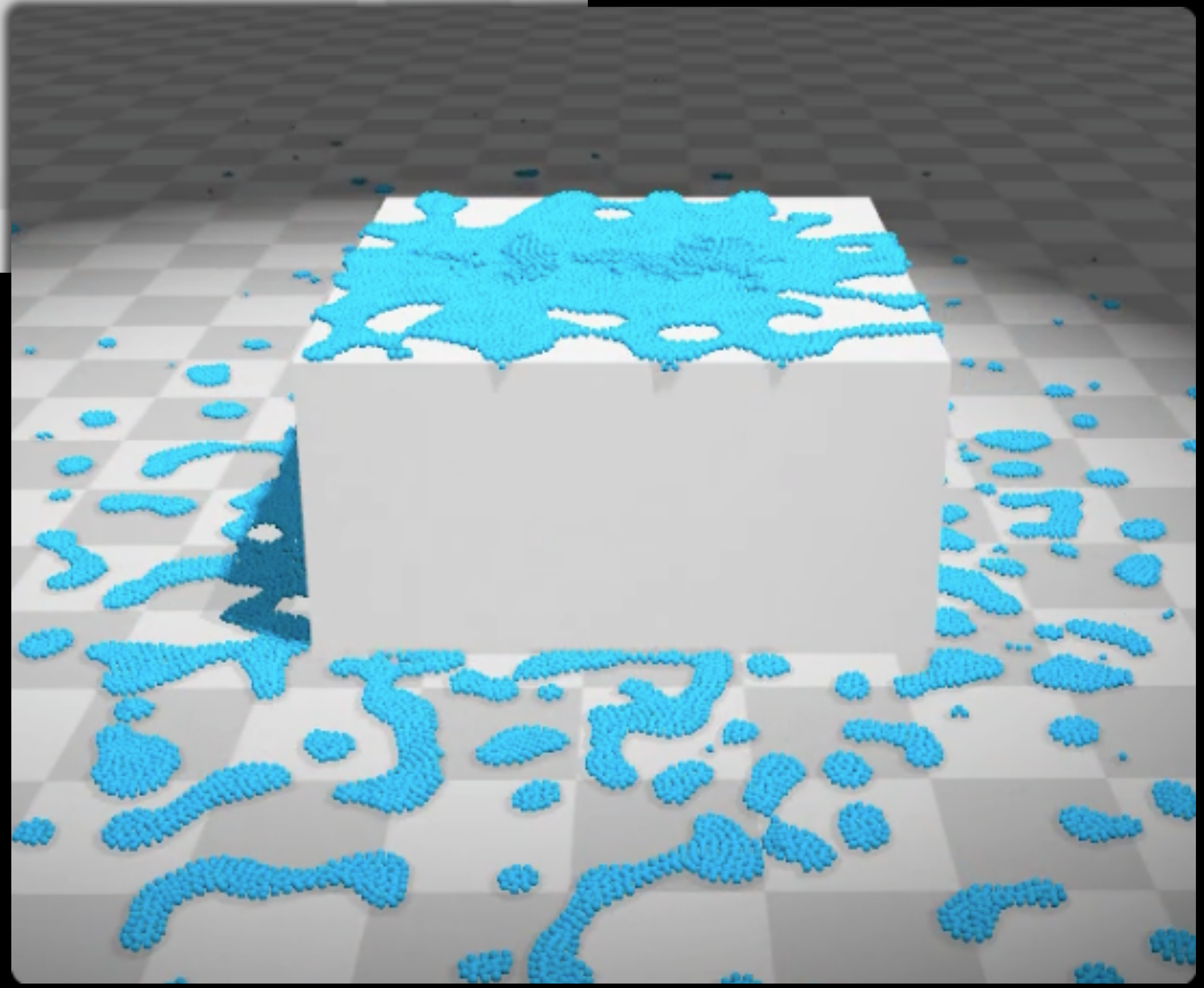




# Constraints

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





# Talk Outline

1. Parallel Solver
2. Contact and friction
3. Rigid bodies
4. Gases

# Position-Based Dynamics (PBD)

- Predict $\tilde{\mathbf{x}} = \mathbf{x}^n + \Delta t \mathbf{v}^n$
- For k=0 to solver iterations
  - Project $\tilde{\mathbf{x}}^{k+1} = \text{ project } (\tilde{\mathbf{x}}^k, C) \text{ along } \mathbf{M}^{-1} \nabla C_{\mathbf{x}^k}$
  - or Minimize
$$\min_{\tilde{\mathbf{x}}^{k+1}} \quad \frac{1}{2}(\tilde{\mathbf{x}}^{k+1} - \tilde{\mathbf{x}}^k)^T \mathbf{M} (\tilde{\mathbf{x}}^{k+1} - \tilde{\mathbf{x}}^k)$$
$$\text{s.t.} \quad C_i(\tilde{\mathbf{x}}^{k+1}) = 0$$
- Velocity Update $\mathbf{v}^{n+1} = \mathbf{x}^* - \mathbf{x}^n$
- Position Update $\mathbf{x}^{n+1} = \mathbf{x}^*$

# Relationship to Implicit Euler

- Position level formulation of backwards 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 first order optimality condition 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: $\begin{aligned}\tilde{\mathbf{x}} &= 2\mathbf{x}^n - \mathbf{x}^{n-1} \\ &= \mathbf{x}^n + \Delta t \mathbf{v}^n\end{aligned}$

# Backward Euler as Constrained Minimization

- Constraints are infinitely stiff potentials

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

- Produces the following constrained optimization:

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

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

# Implicit Euler

- Predict $\tilde{\mathbf{x}} = \mathbf{x}^n + \Delta t \mathbf{v}^n$
- For k=0 to solver iterations
  - Project $\tilde{\mathbf{x}}^{k+1} = \text{ project } (\tilde{\mathbf{x}}, C) \text{ along } \mathbf{M}^{-1} \nabla C_{\mathbf{x}^k}$
  - or Minimize

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

[Martin et al. 2011]

- Velocity Update $\mathbf{v}^{n+1} = \mathbf{x}^* - \mathbf{x}^n$
- Position Update $\mathbf{x}^{n+1} = \mathbf{x}^*$

# Optimality Conditions for Implicit Euler

- Applying Newton's method to the optimality conditions leads to the following KKT matrix for each QP sub-problem

$$\begin{bmatrix} \mathbf{M} & \nabla C(\mathbf{x}_i) \\ \nabla C(\mathbf{x}_i)^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{M}\tilde{\mathbf{x}} \\ -\mathbf{b} \end{bmatrix}$$

- Eliminate x to obtain backward Euler update:

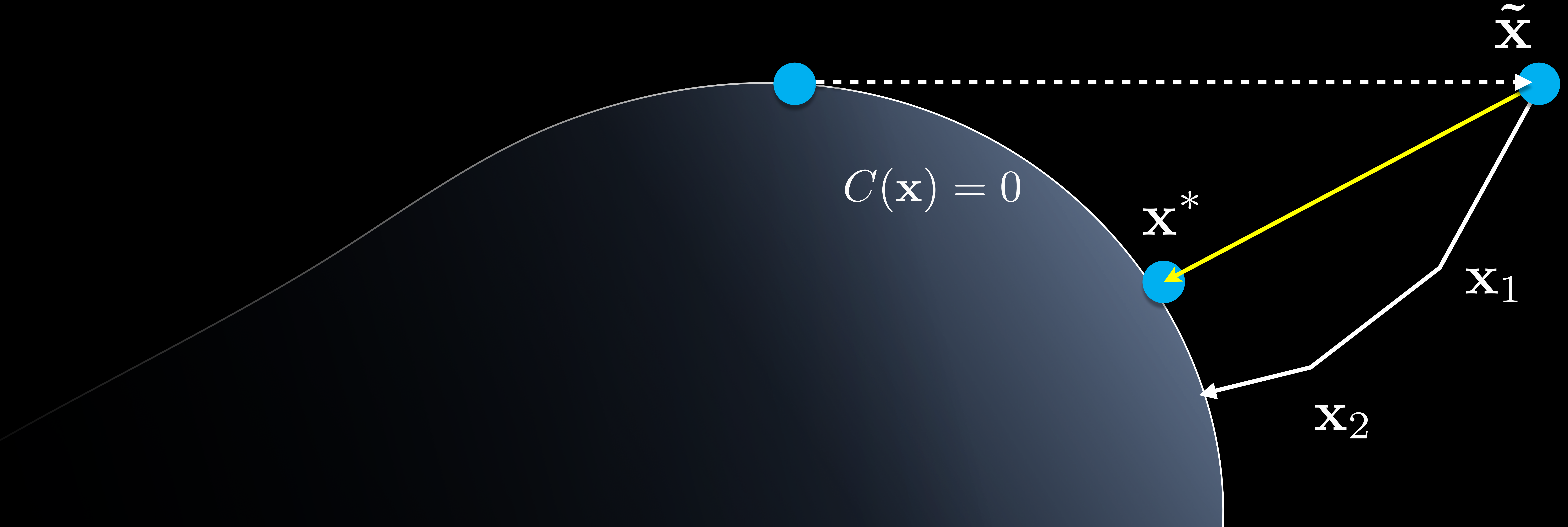
$$\left[\nabla C(\mathbf{x}_i)^T \mathbf{M}^{-1} \nabla C(\mathbf{x}_i)\right] \lambda = -\mathbf{C}(\tilde{\mathbf{x}})$$

- The same as PBD? Not quite, different right-hand side:

$$\left[\nabla C(\mathbf{x}_i)^T \mathbf{M}^{-1} \nabla C(\mathbf{x}_i)\right] \lambda = -\mathbf{C}(\mathbf{x}_i)$$

# PBD and Implicit Euler

- In practice different minimization makes little visual difference
- Identical for linear constraints



# Equivalence to Nucleus

- Nucleus [Stam 09]:

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

- Position Based Dynamics [Müller et al. 06]

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

- PBD converts position changes to impulses applied at the beginning of the time-step

# Parallel Position Based Dynamics

- At each iteration we need to solve the following system:

$$\left[\nabla C(\mathbf{x}_i)^T \mathbf{M}^{-1} \nabla C(\mathbf{x}_i)\right] \lambda = -\mathbf{C}(\mathbf{x}_i)$$

- Position Based Dynamics (PBD) is typically serial
- Use Gauss-Jacobi for parallelism and simple handling of inequalities
- Problem: system matrix can be indefinite, Jacobi will not converge, e.g.: for redundant constraints (cf. figure)

# Constraint Averaging

- 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]:

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

# Constraint Solving on the GPU

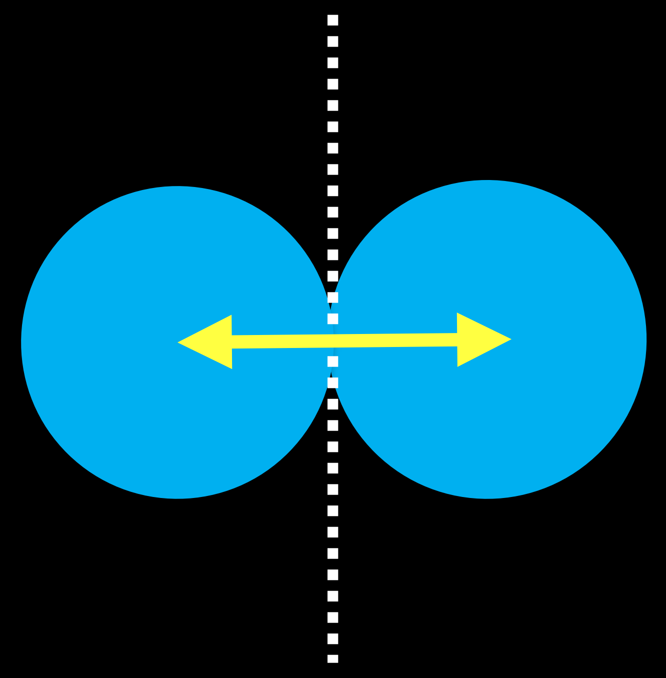
- Two ways to solve constraints:

<table>
<tr><th>Particle-centric approach<br>(gather)</th><th>Constraint-centric approach<br>(scatter)</th></tr>
<tr><td><pre>foreach particle (in parallel)
{
  foreach constraint
  {
    calculate constraint error
    update delta
  }
}</pre></td><td><pre>foreach constraint (in parallel)
{
  calculate constraint error
  foreach particle
  {
    update delta (atomically)
  }
}</pre></td></tr>
</table>

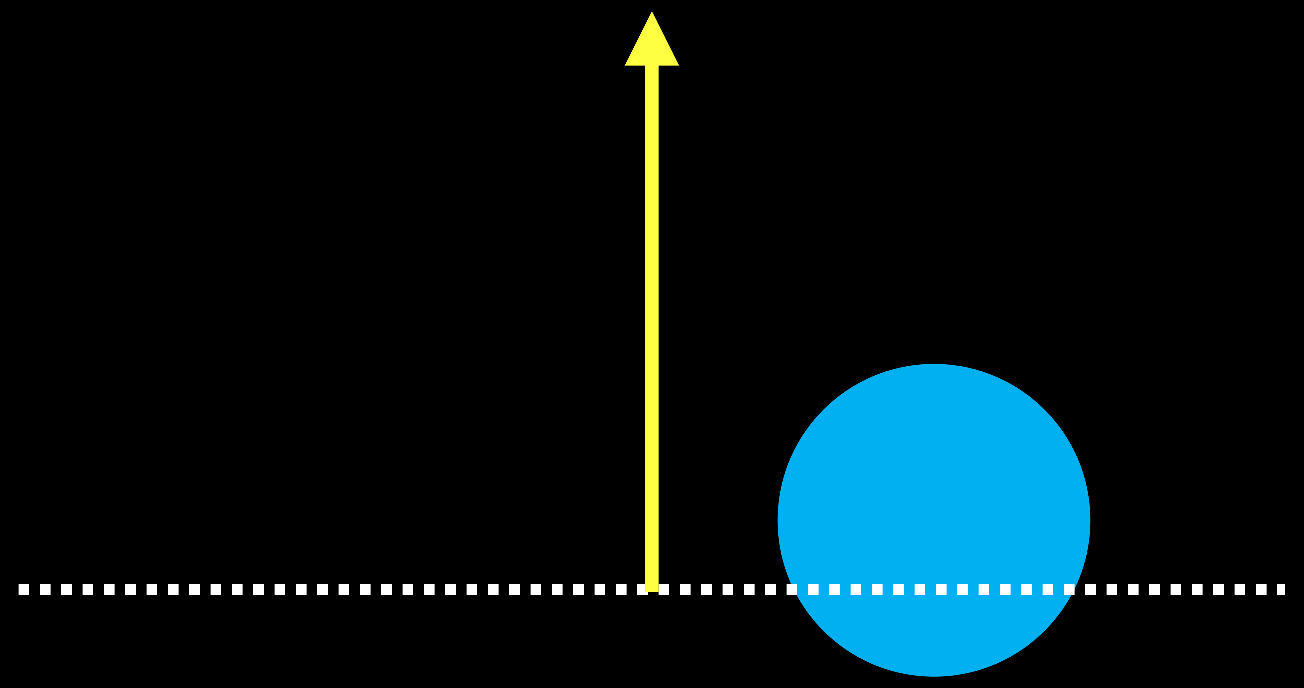
# Contact and friction

# Collision Detection

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



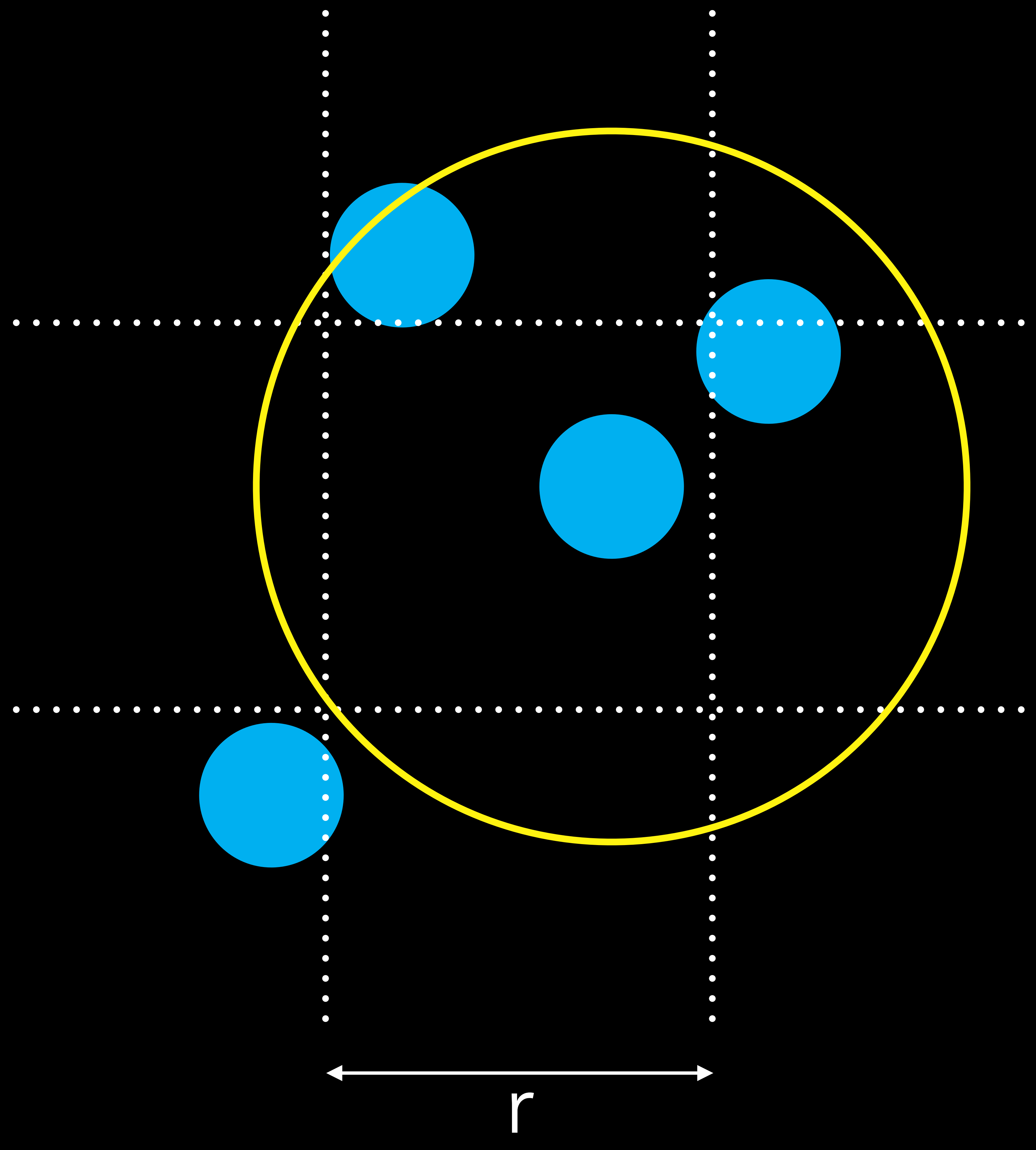
$$C_{contact} = |\mathbf{x}_i - \mathbf{x}_j| - 2r \geq 0$$



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

# Collision Detection

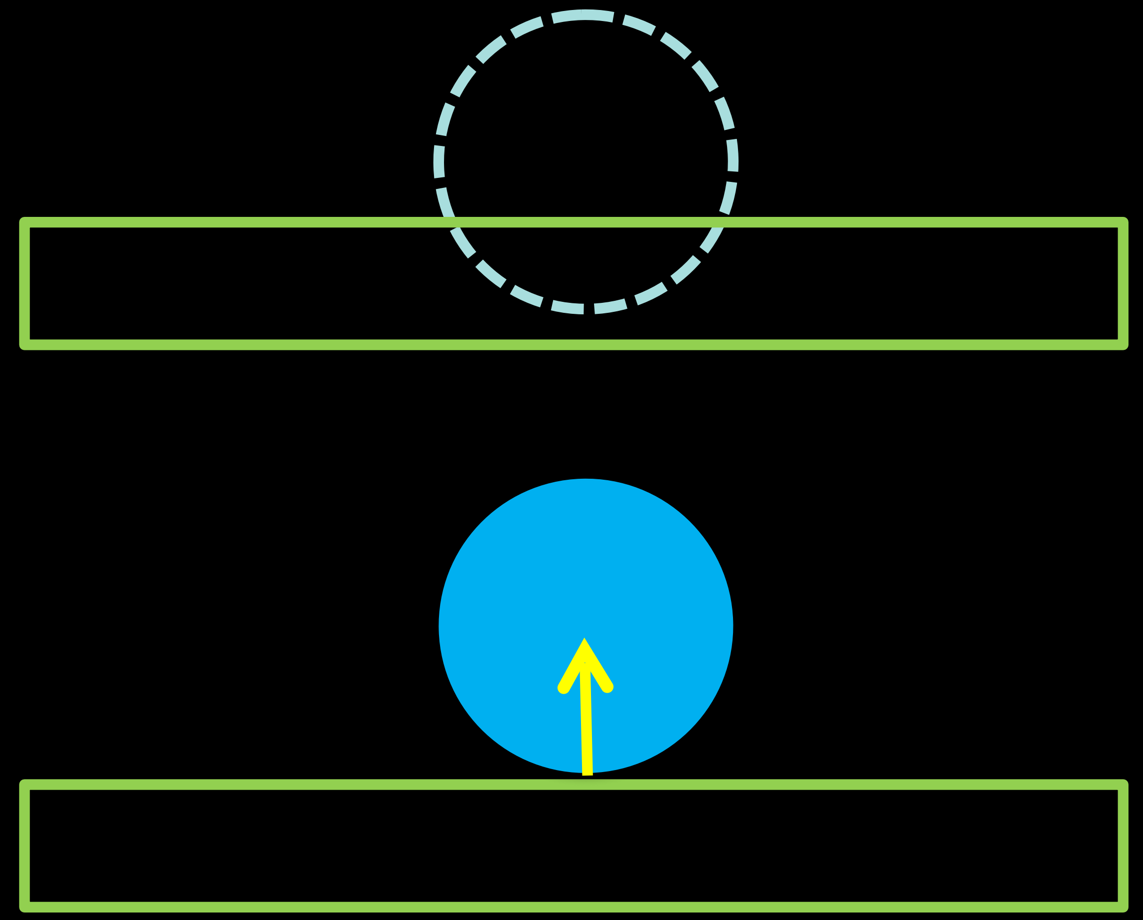
- Particle-Particle
  - Tiled uniform grid
  - Fixed maximum radius
  - Built using cub::DeviceRadixSort
  - Re-order particle data according to cell index to improve memory locality



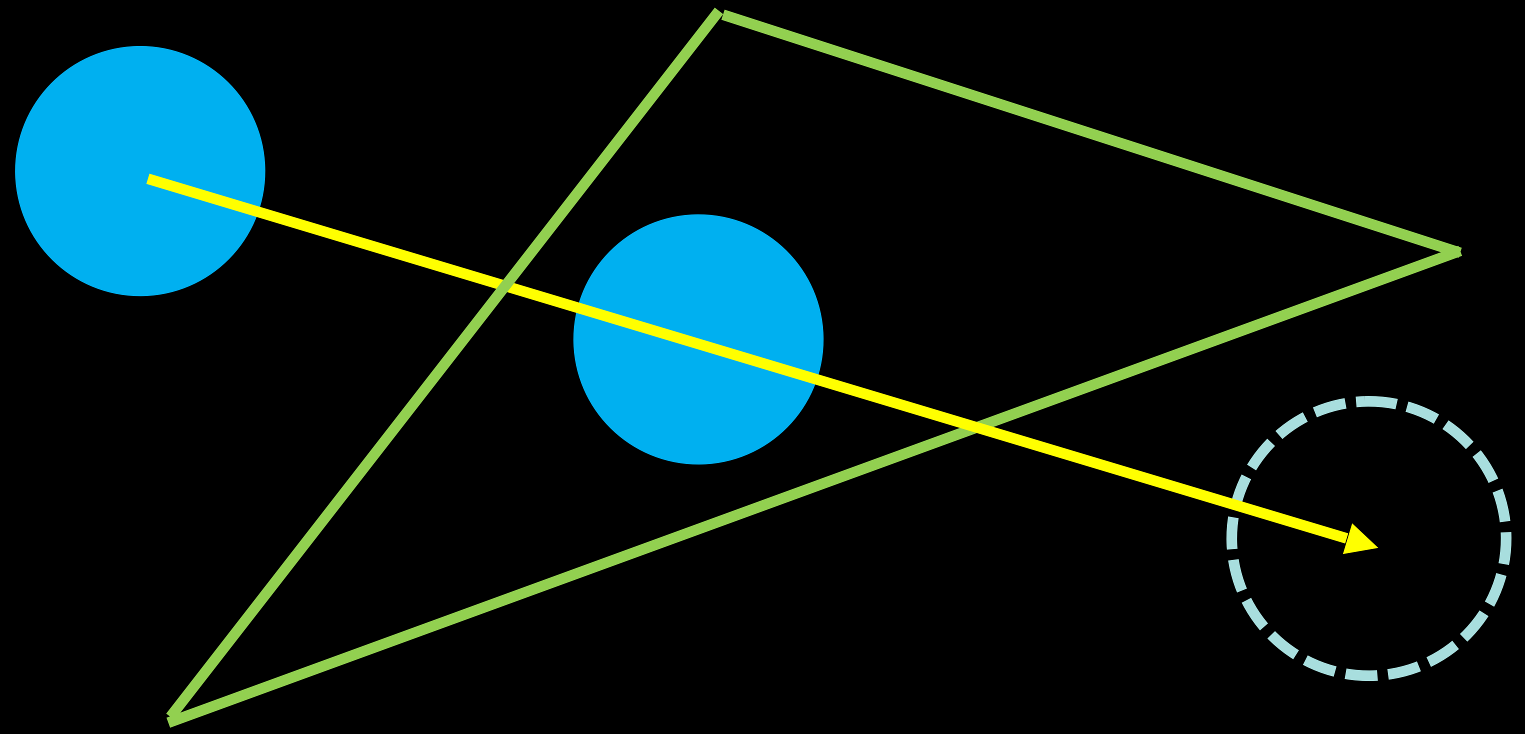
# Collision Detection

- Particle-Convex
  - 2D hash-grid
  - Built on GPU
  - 1 warp-per shape, rasterizes projected bounds to grid (~1500 shapes / ms)

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



Convex Collision (MTD)



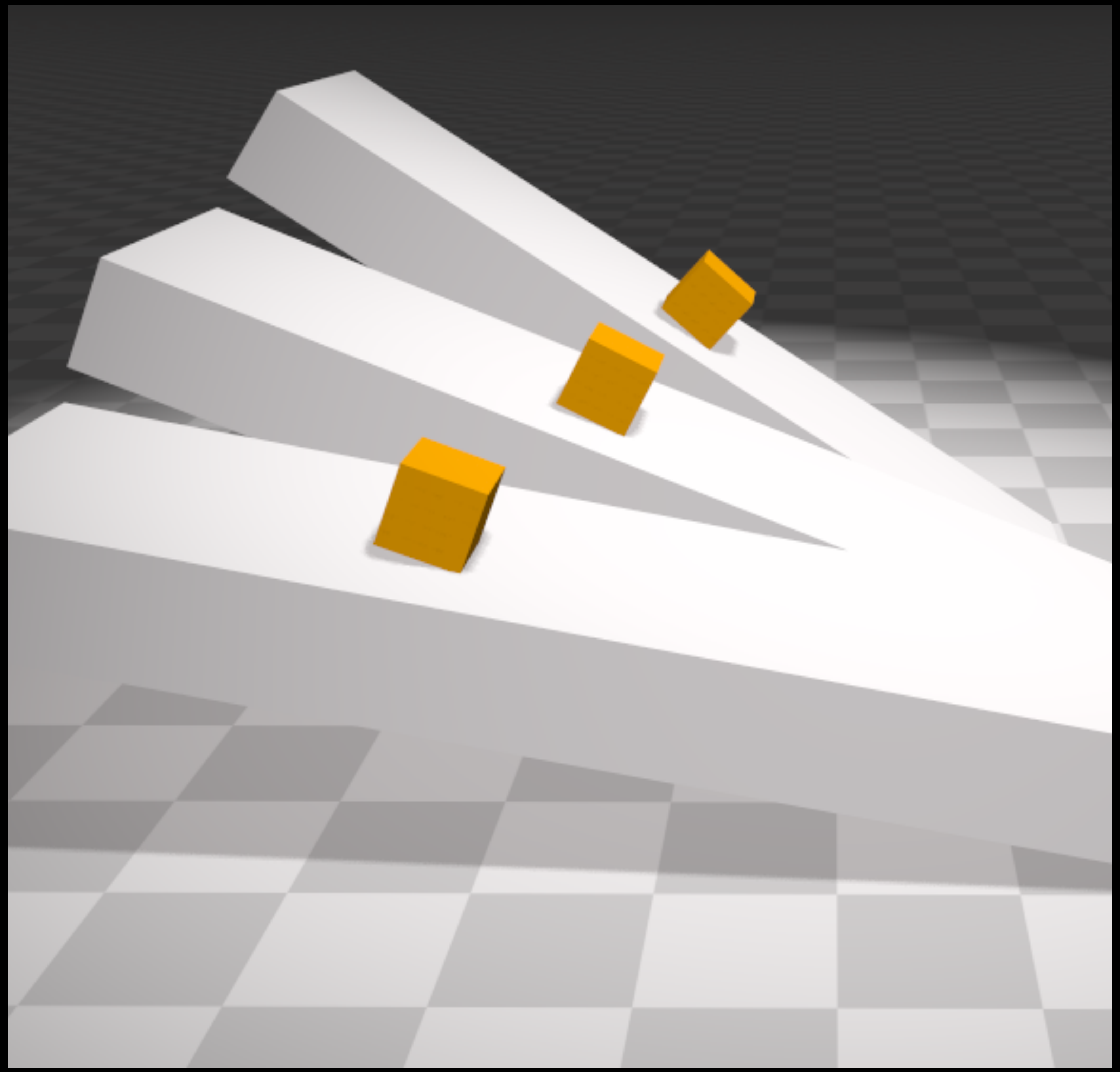
Triangle Collision (TOI)

# Friction

- Friction in PBD traditionally applied using a velocity filter
- We introduce a position-level frictional constraint

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

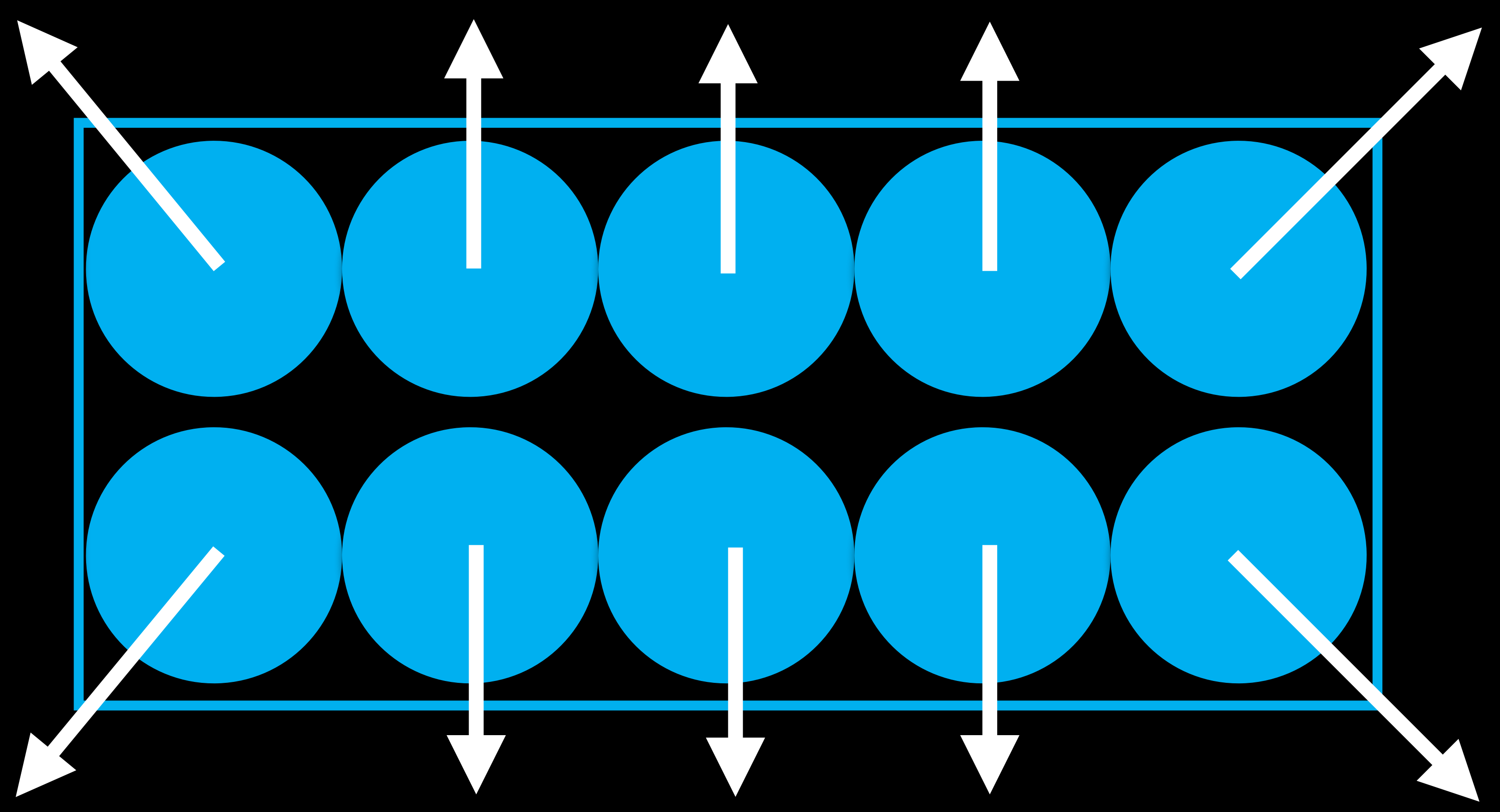
- Approximate Coulomb friction using penetration depth to limit lambda

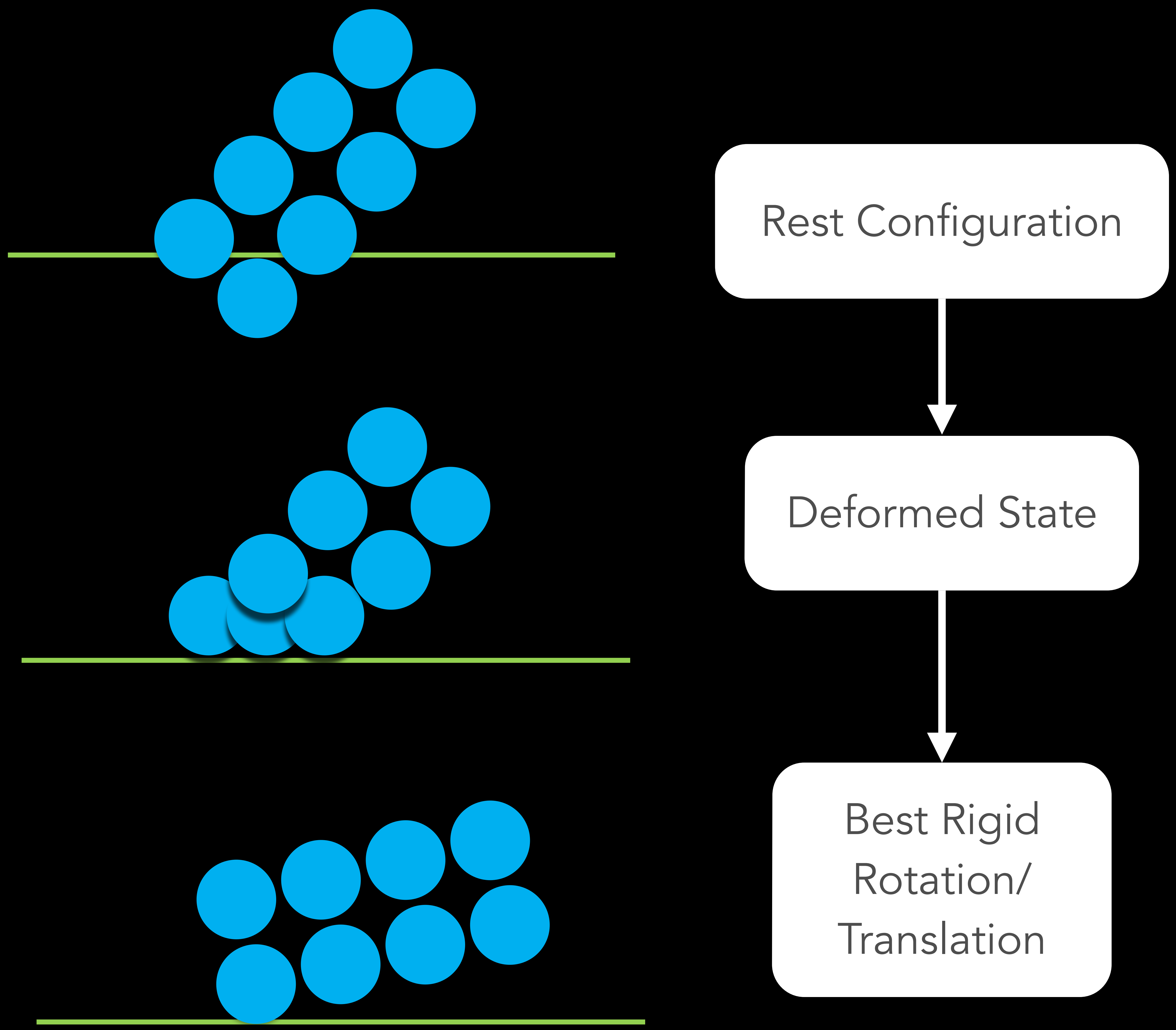


# Rigid Bodies

# Rigid Bodies

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





# Shape matching on the GPU

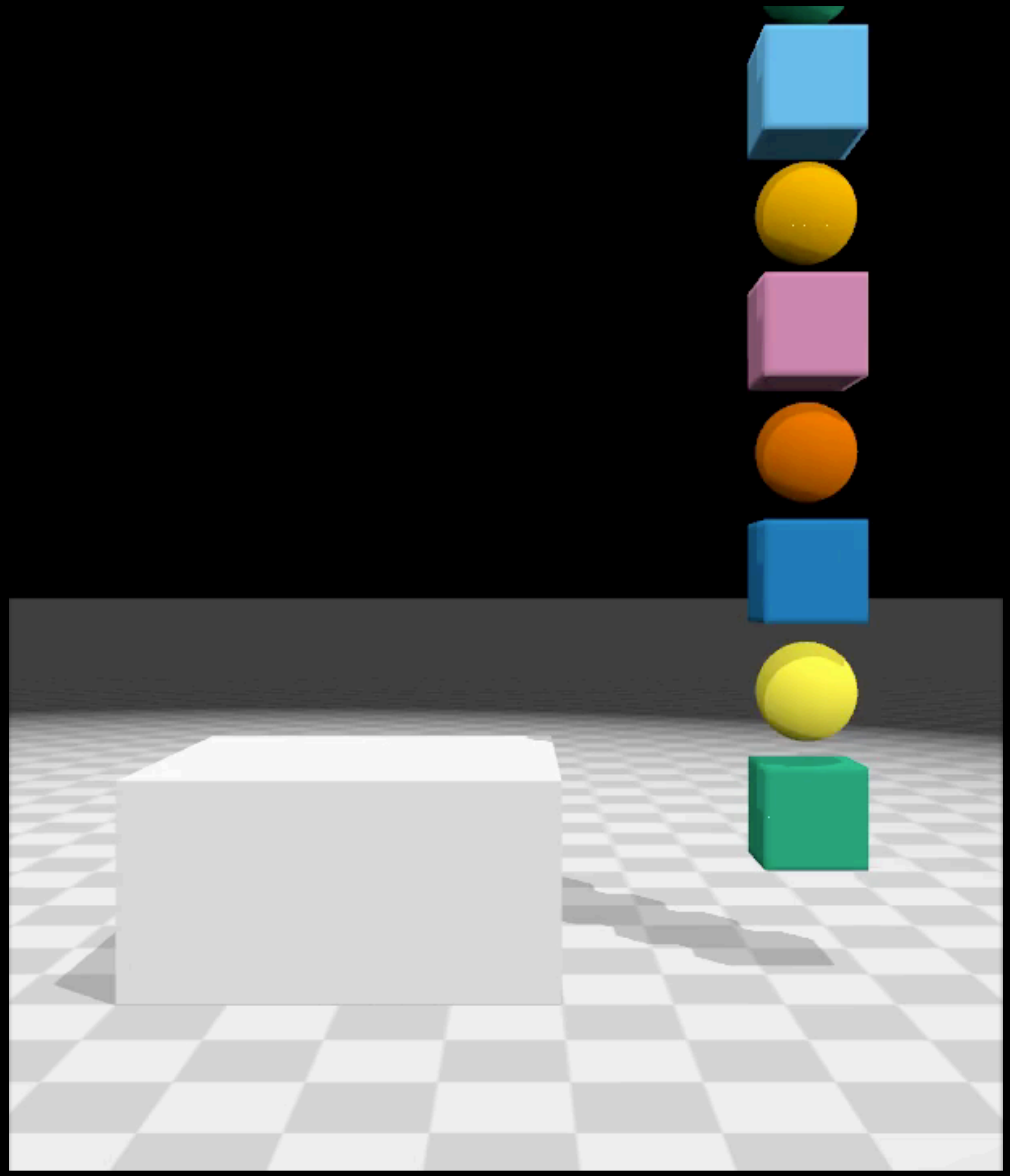
- Shape matching requires computing centre of mass and the moment matrix for particles:

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

- 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

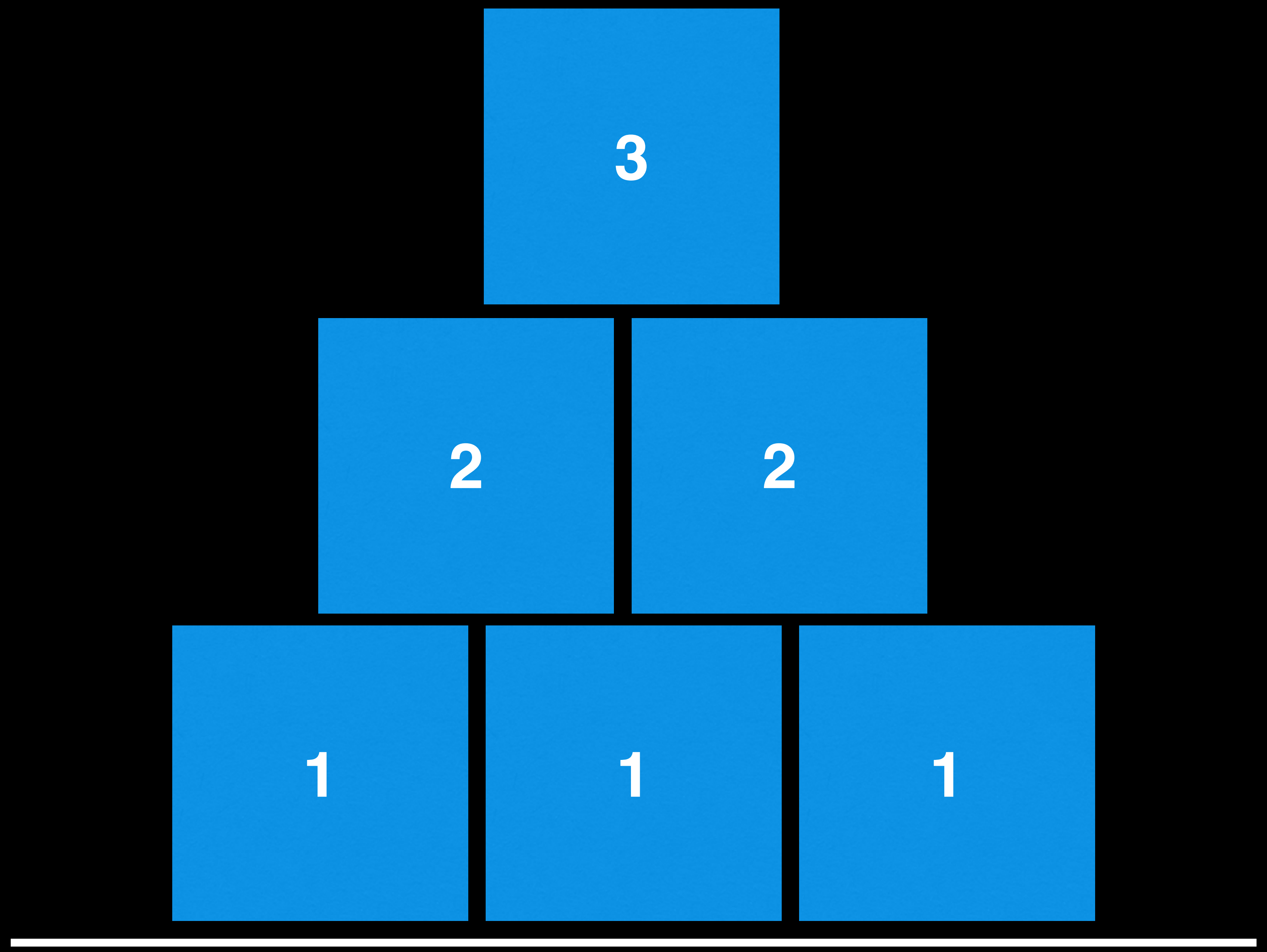
# Plastic Deformation

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



# Rigid Bodies - Piles and Stacks

- Piles of objects can take many iterations to appear stiff
- Common solution: shock propagation [Guendelman et al. 03]
  - Re-orders constraint solve bottom->top
  - Sets mass of each layer = ∞
  - Problem: limited parallelism

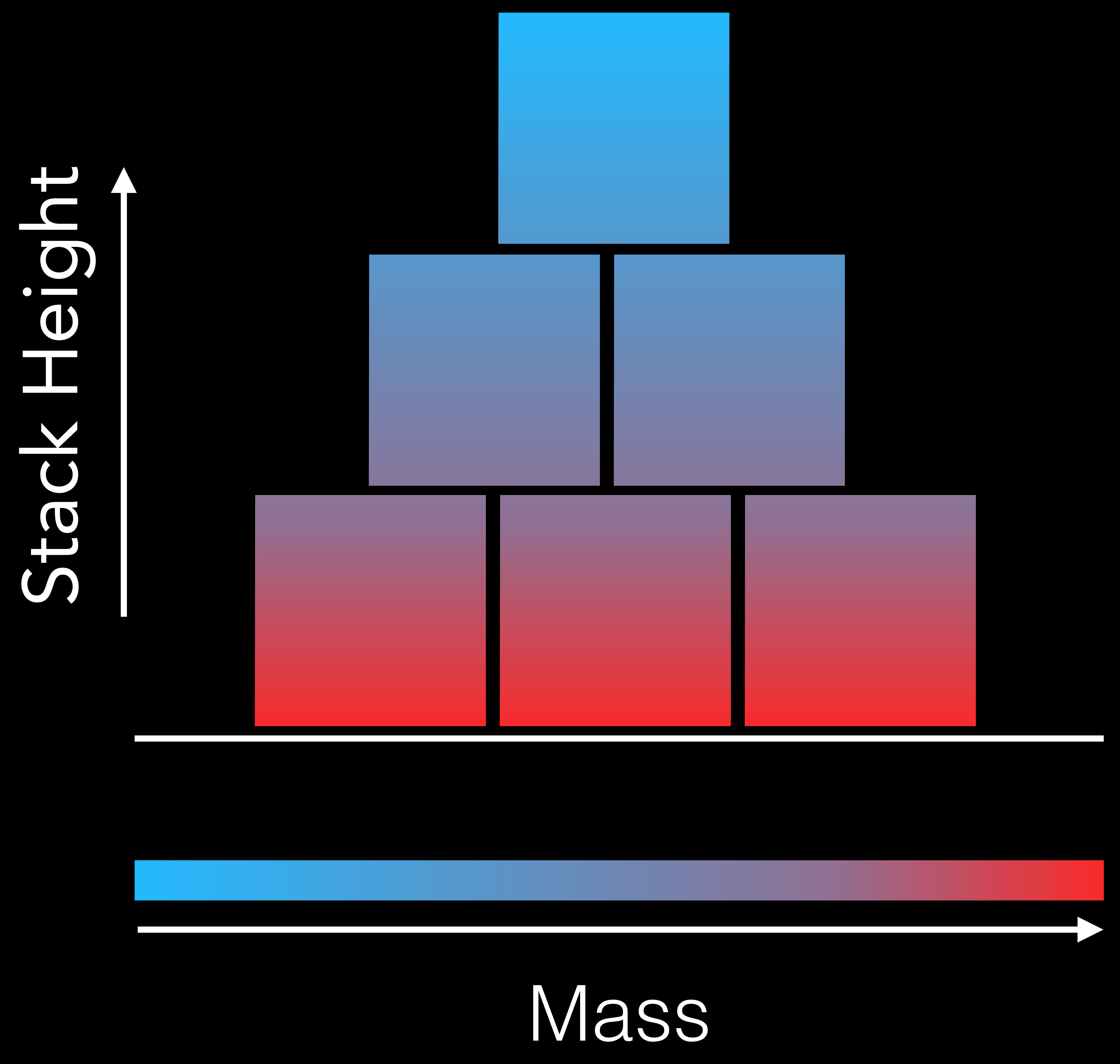


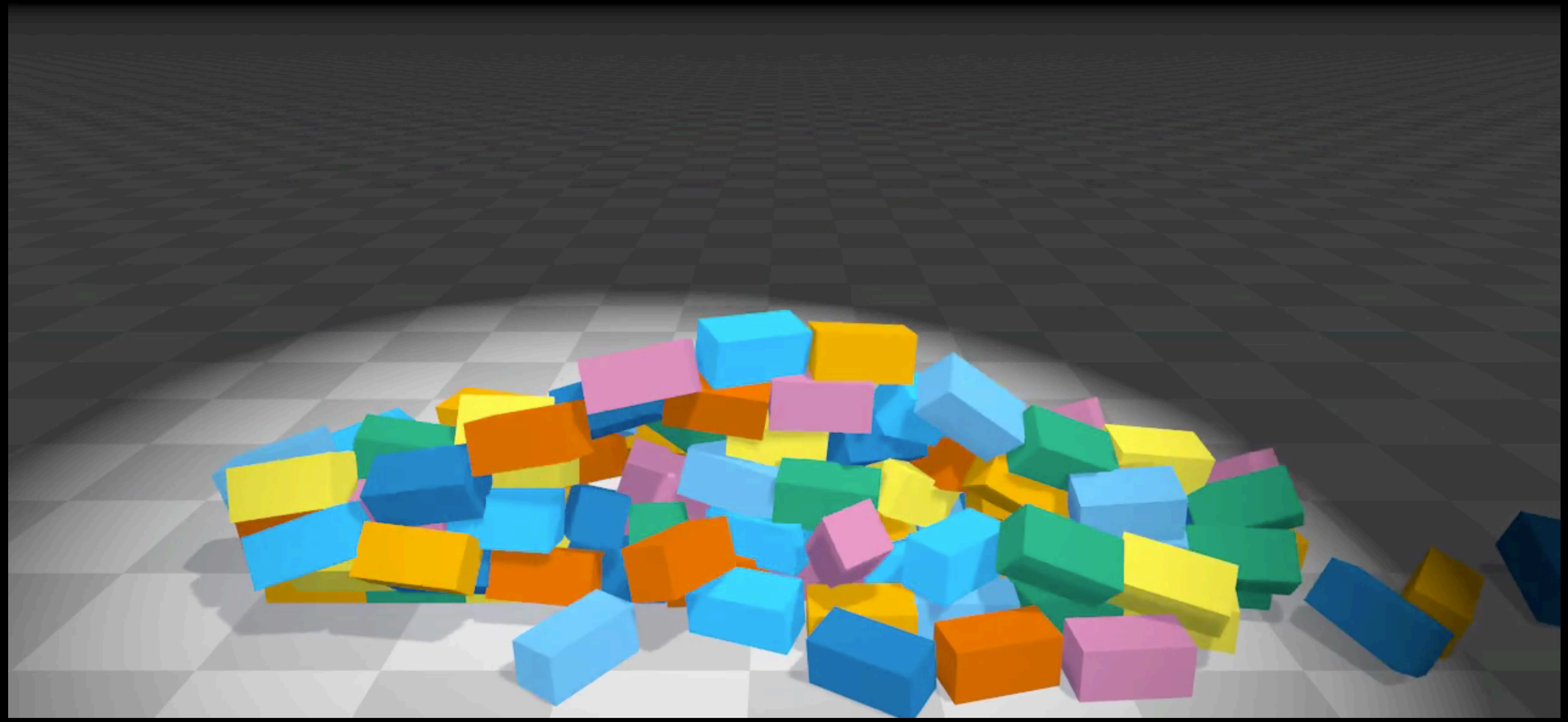
# Approximate Shock Propagation

- A parallel friendly solution
- Instead of discrete layers with infinite mass, we modify mass continuously
- Choose 'stack height' function and evaluate for each particle:

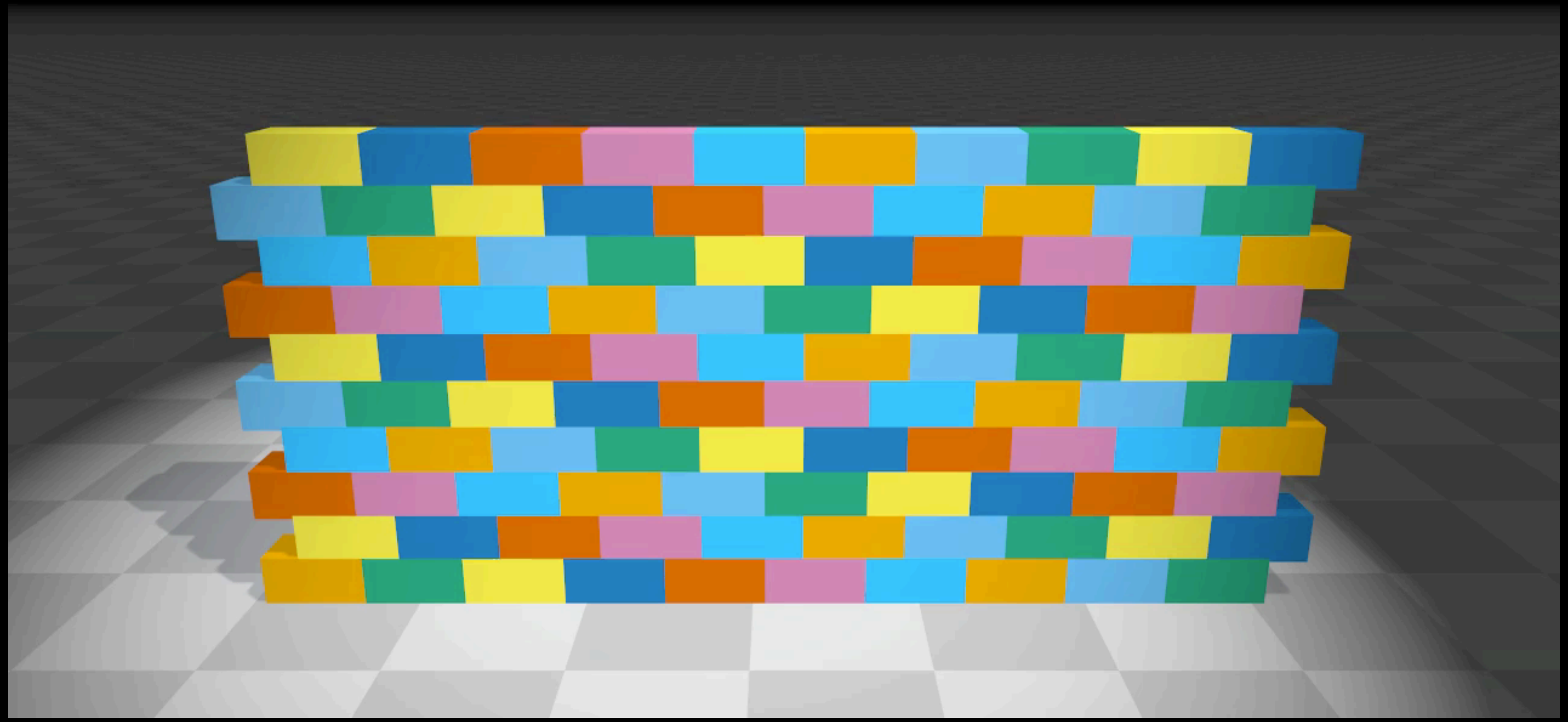
$$height(\mathbf{x}) = |\mathbf{x} - \mathbf{x}_{boundary}|$$

- Temporarily scale particle mass inversely with height





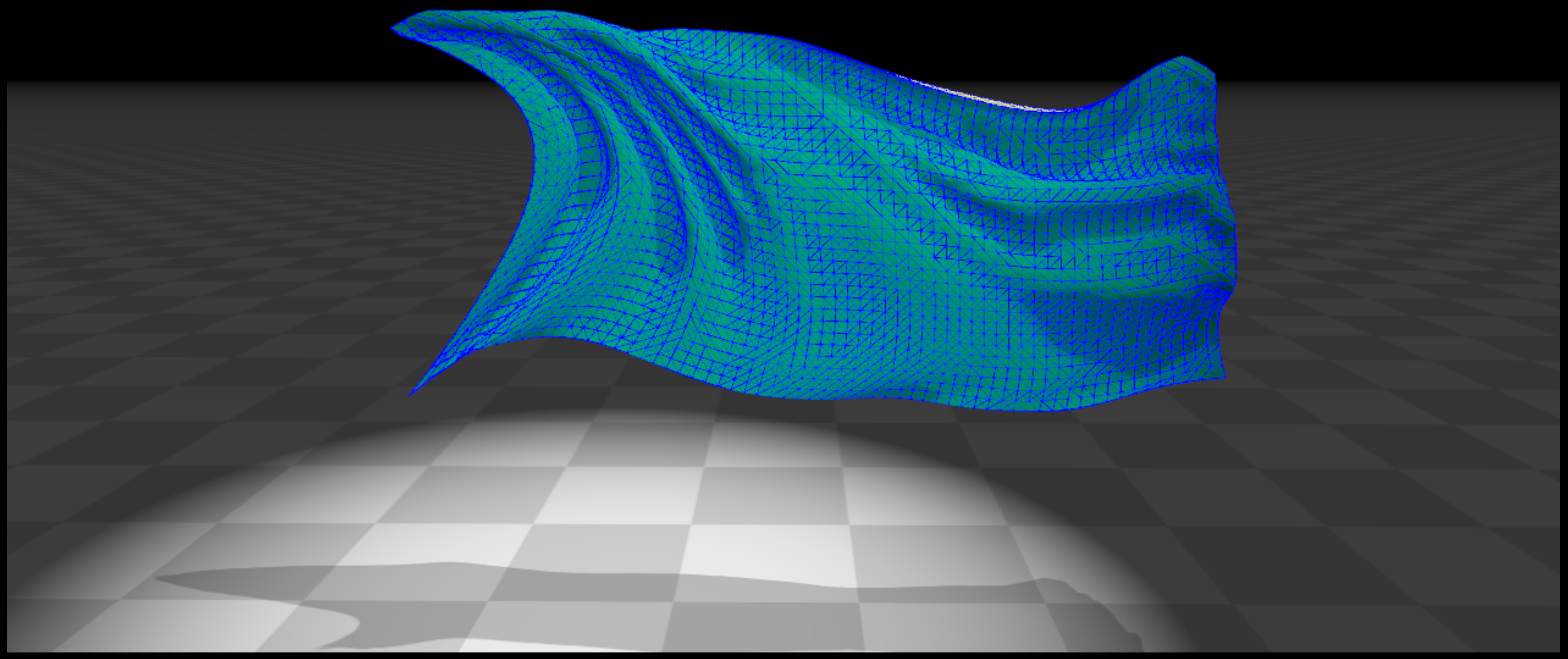
2 Iterations
No Shock Propagation

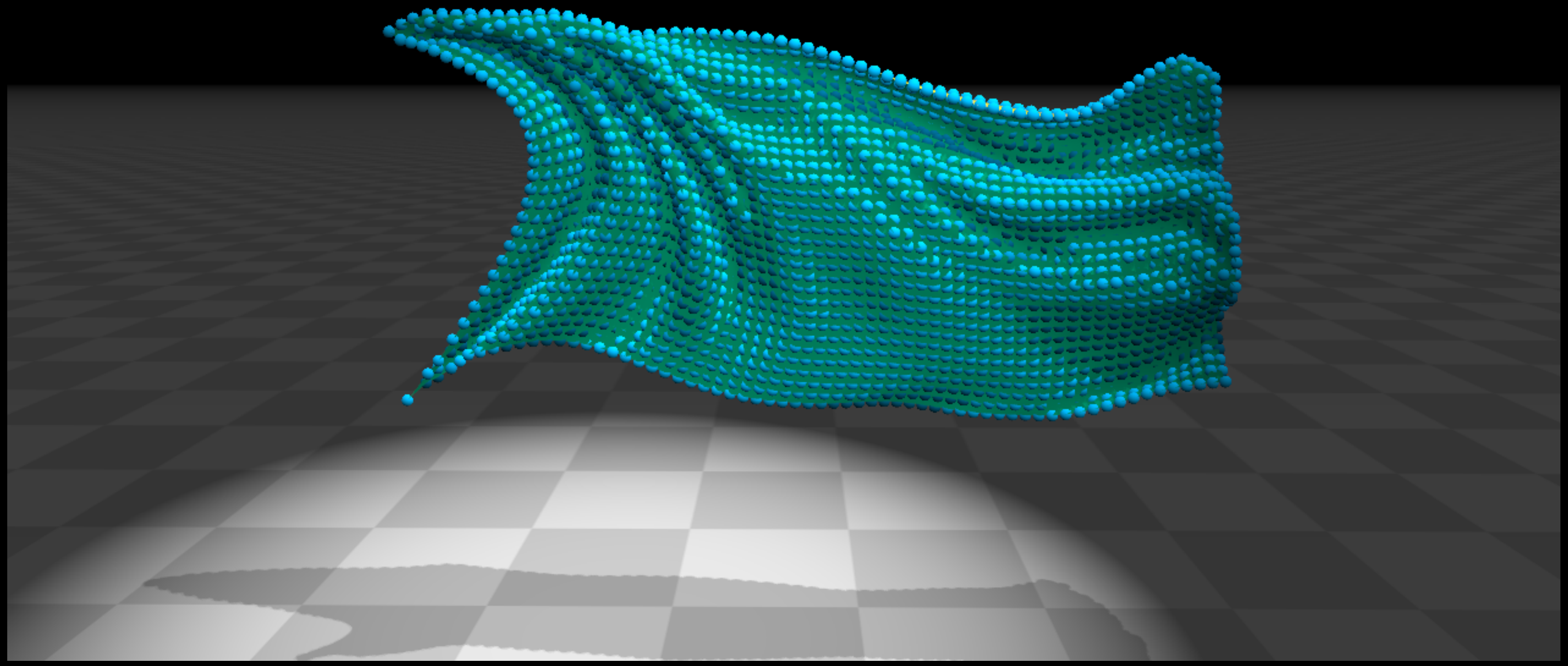


2 Iterations
With Shock Propagation

# Cloth

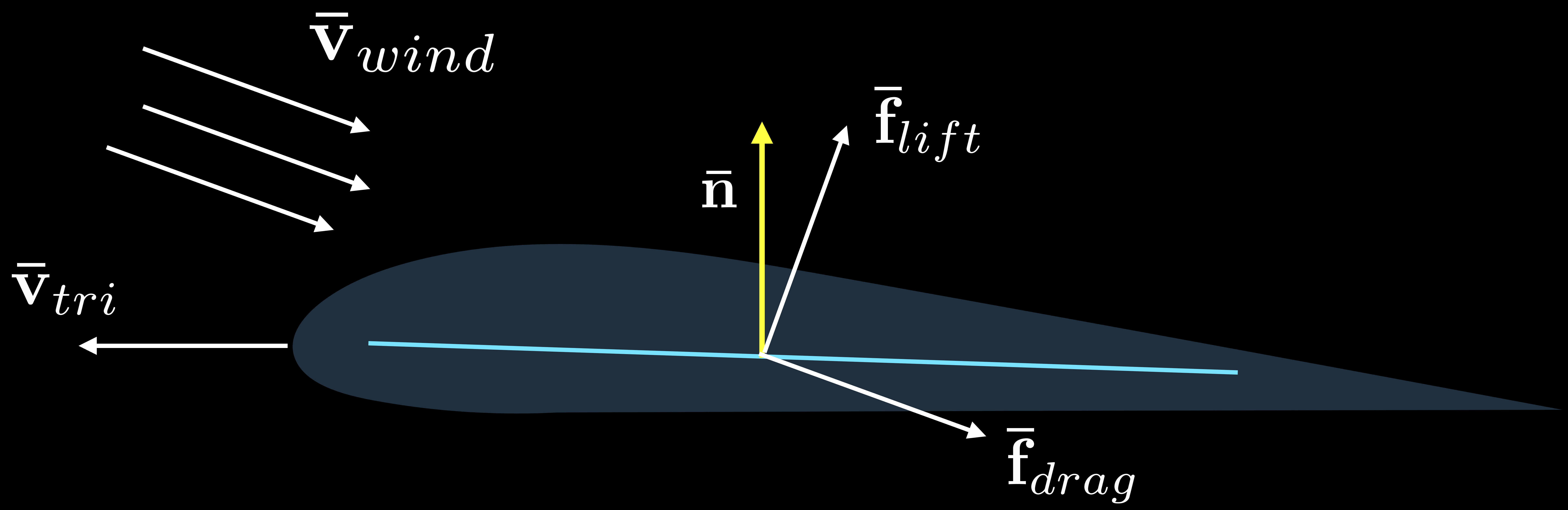
- Graph of distance + tether constraints
- Self-collision / inter-collision automatically handled





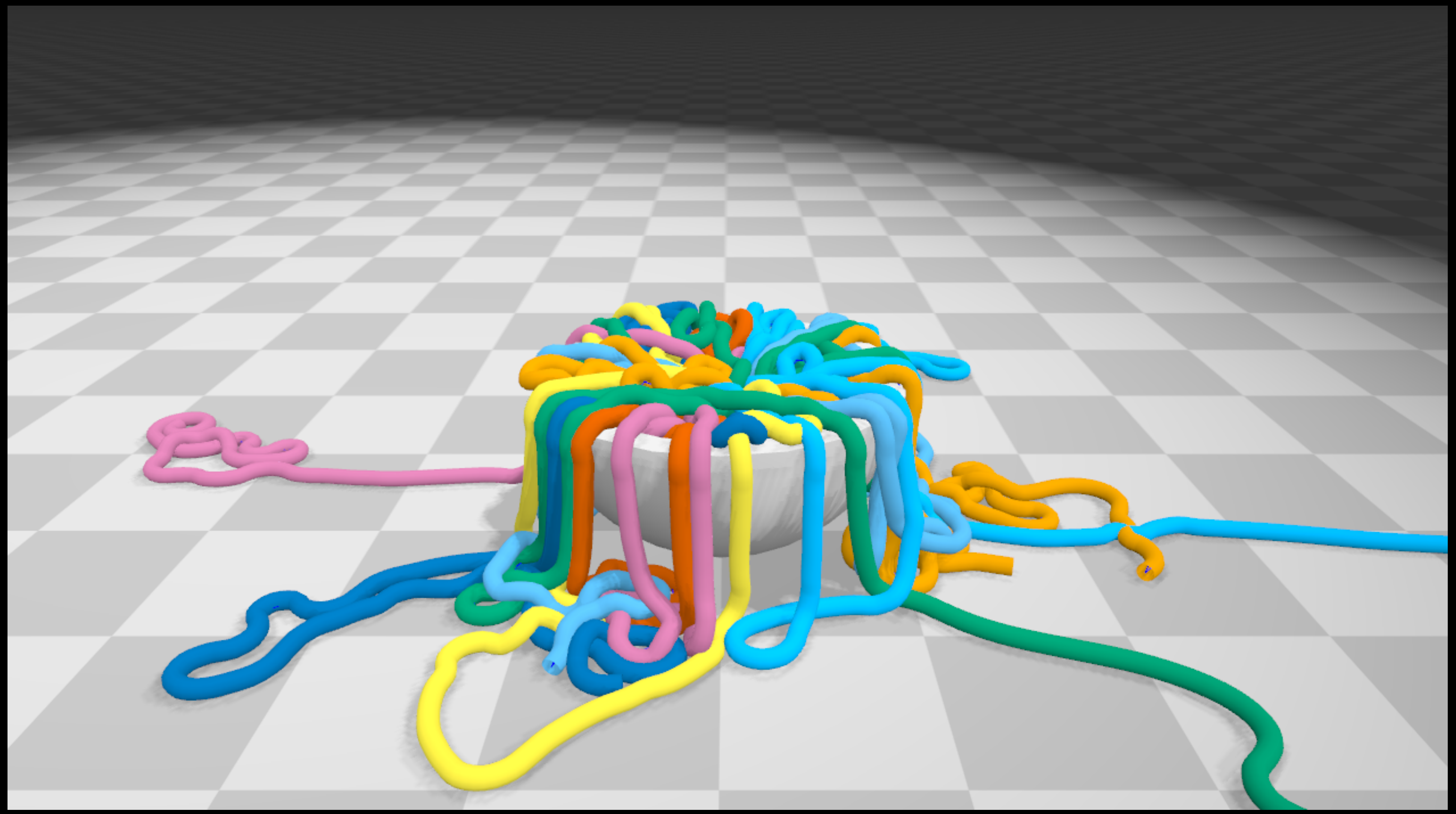
# Cloth - Forces

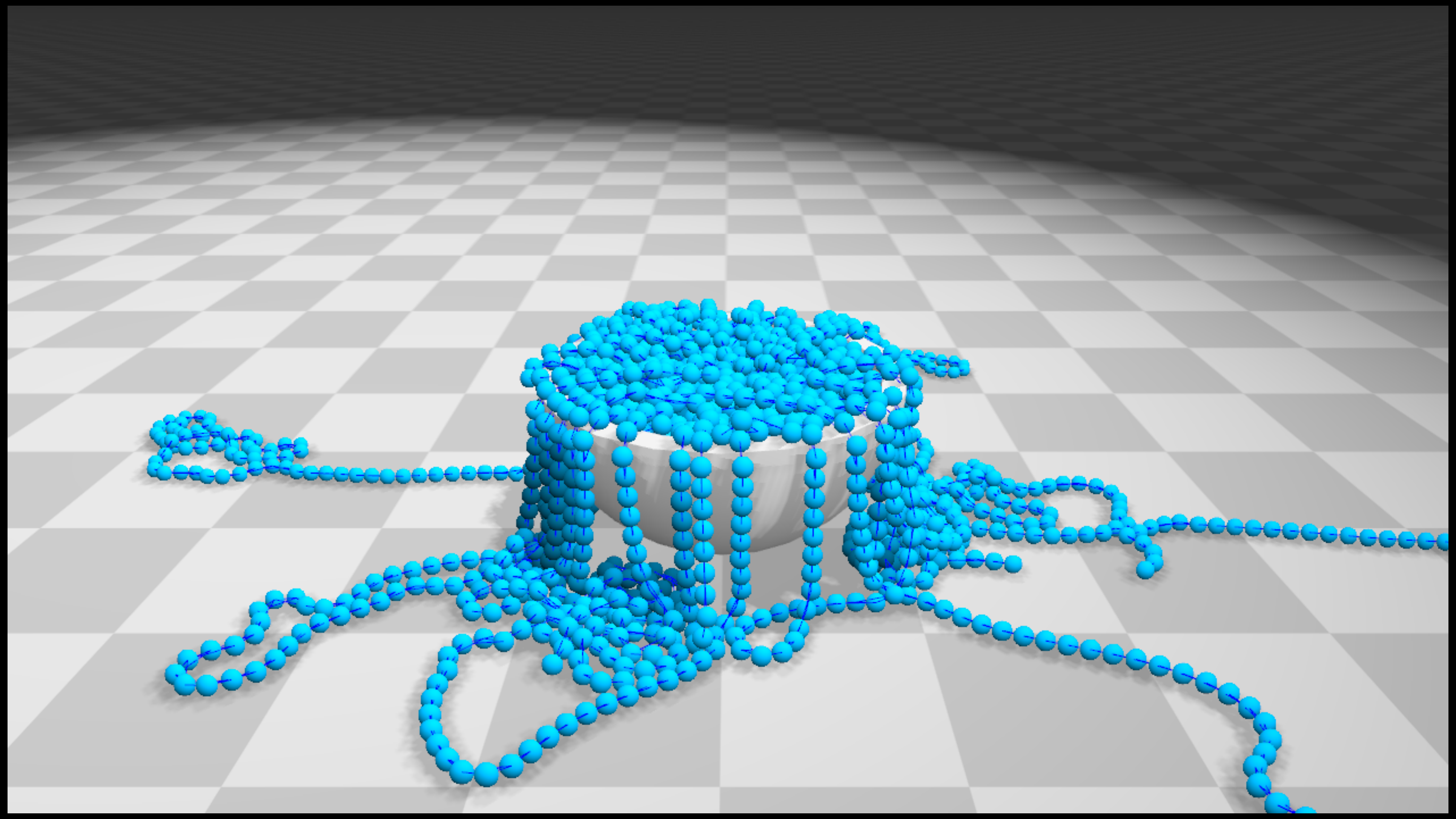
- Basic aerodynamic model
- Treat each triangle as a thin airfoil to generate lift + drag
- Flexible enough to model paper planes

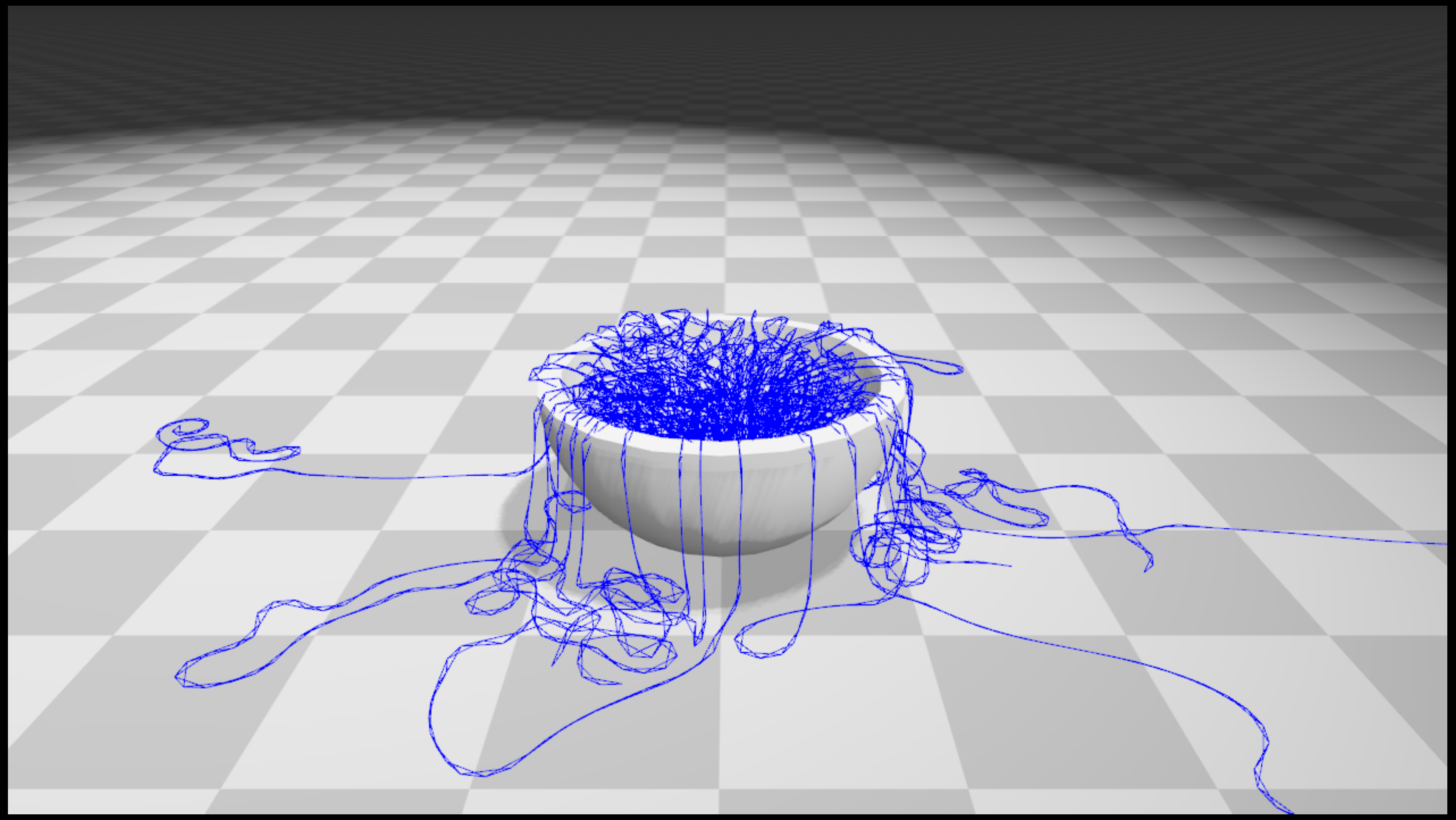


# Ropes

- Build ropes from distance + bending constraints
- Fit Catmull-Rom spline to points
- Good candidate for GPU tessellation unit
- No torsion constraint (need orientation)

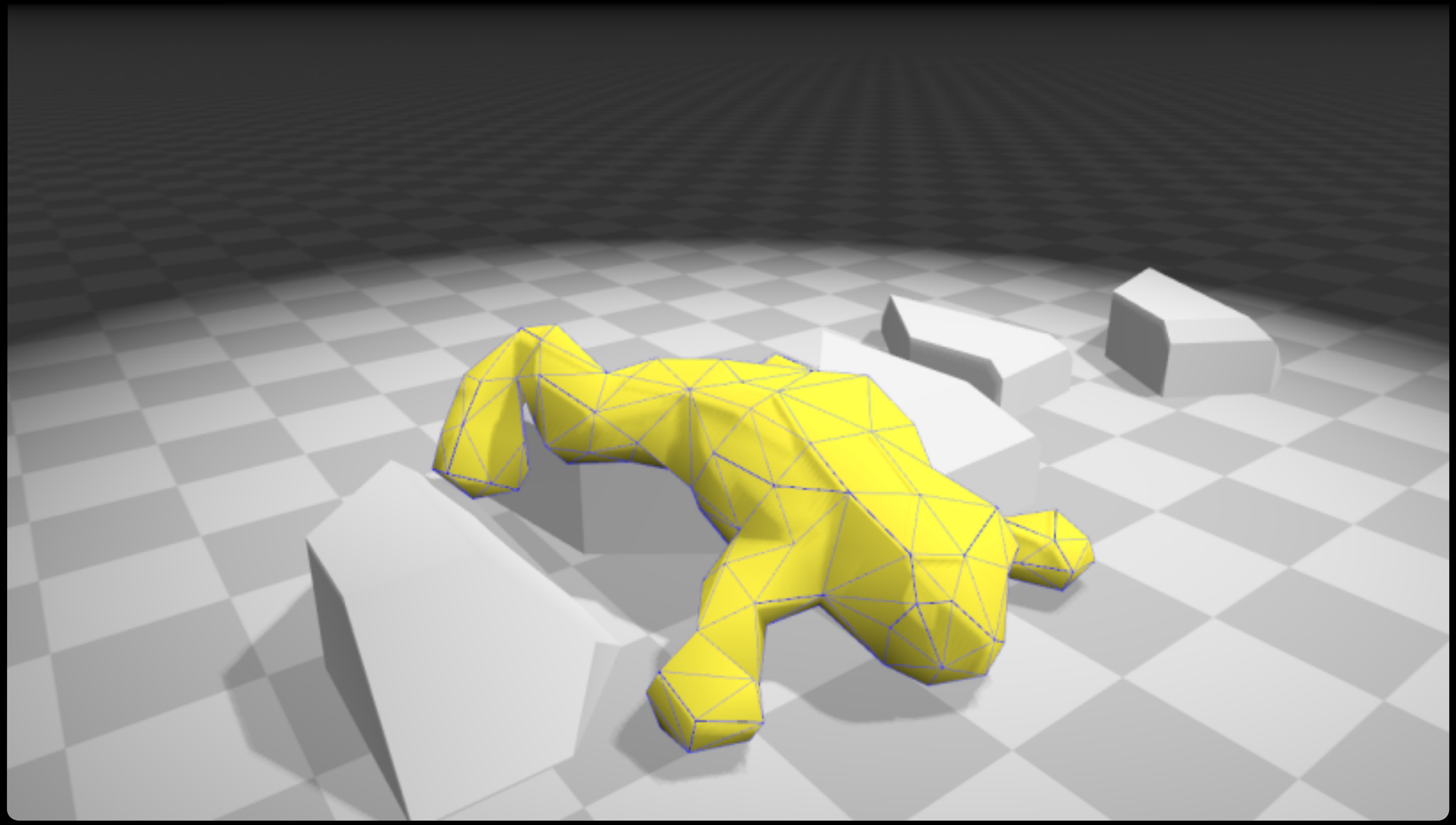






# Deformables

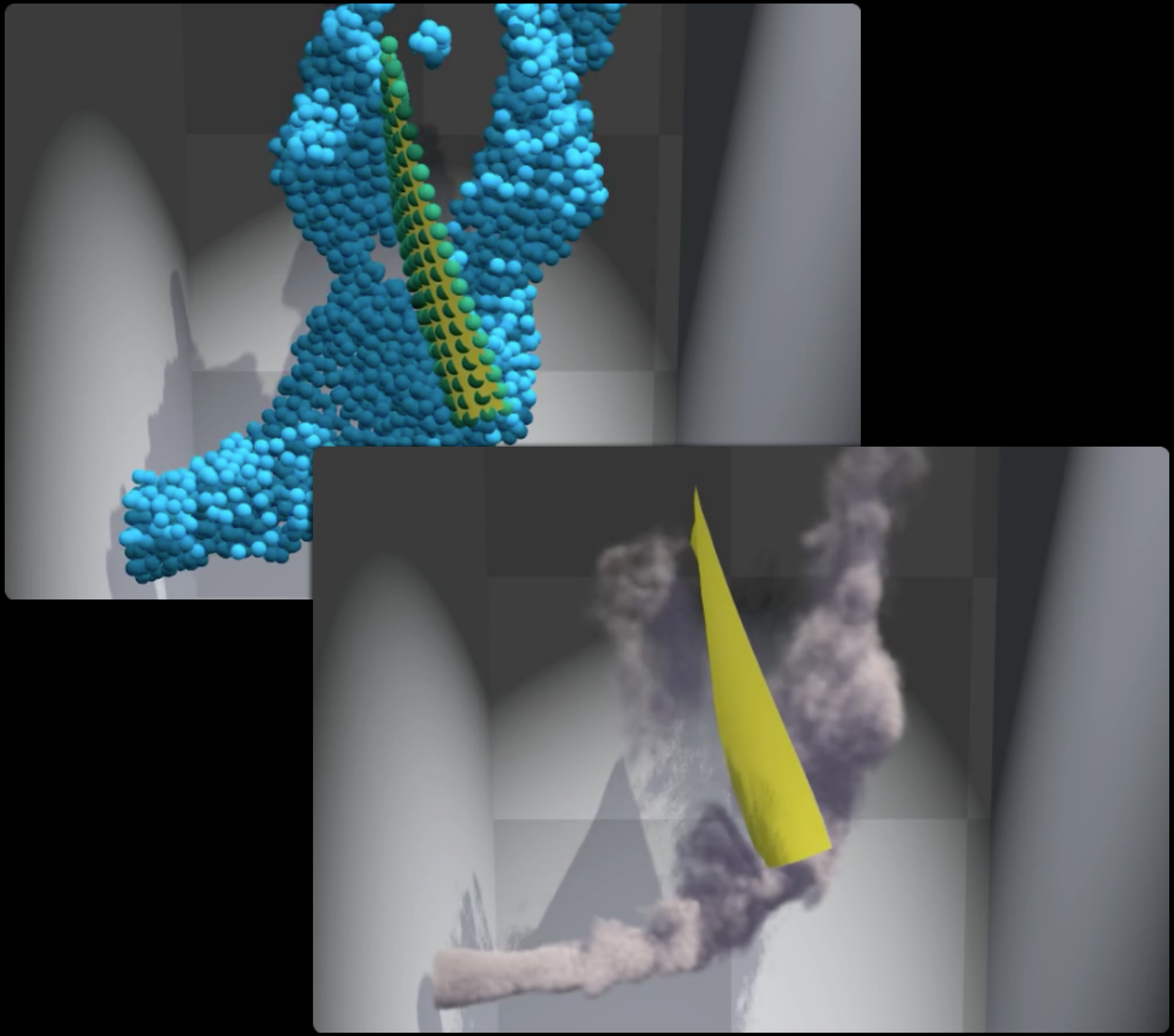
- Tetrahedral meshes -> mass spring system
- Tetrahedral volume constraints
- Soft shape-matching



# Gases

# Gases

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



# Gas Forces

Free surface drag

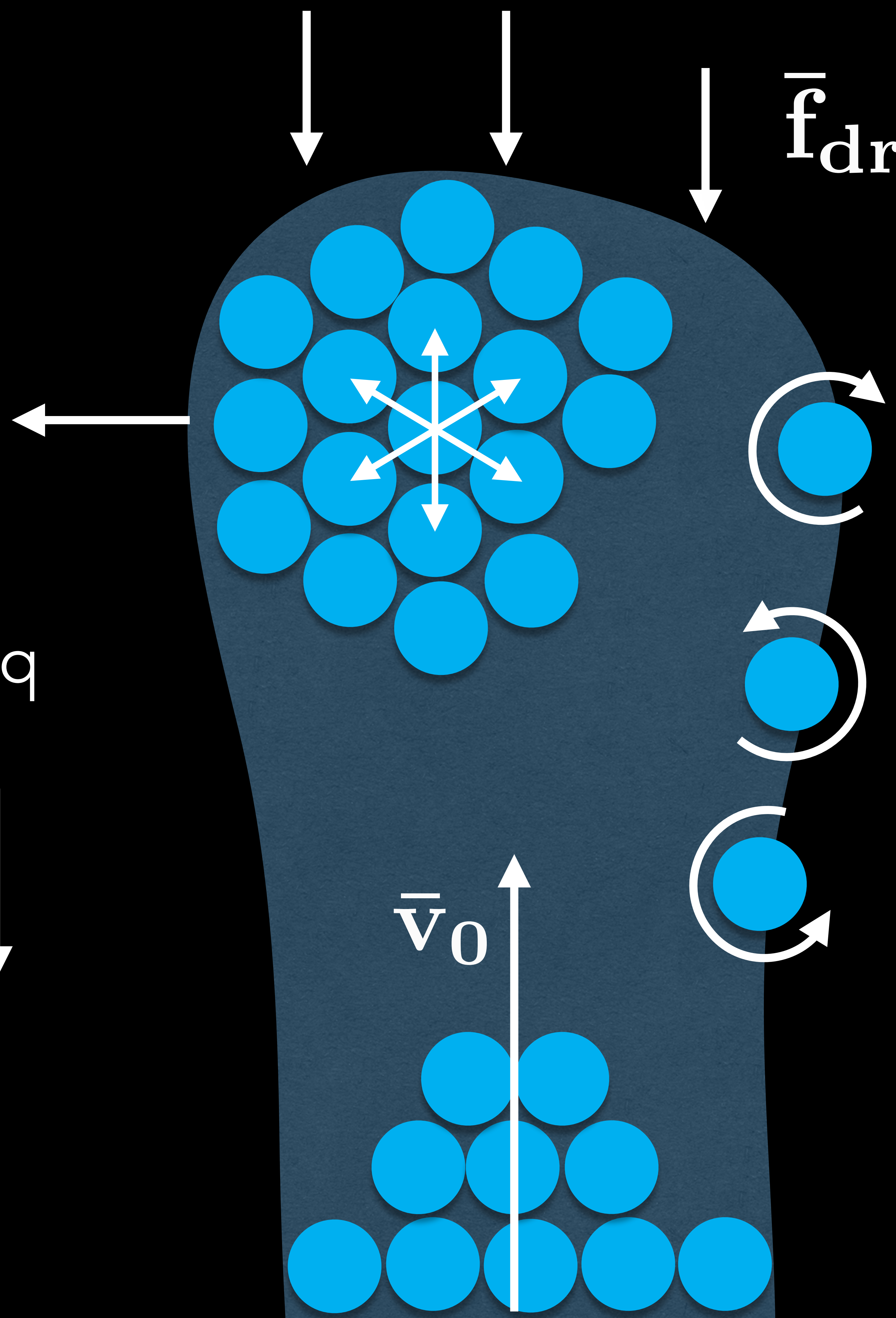
$\bar{\mathbf{f}}_{\mathbf{drag}}$

Density gradient via SPH derivatives:

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

Pressure gradient via Boussinesq approximation:

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



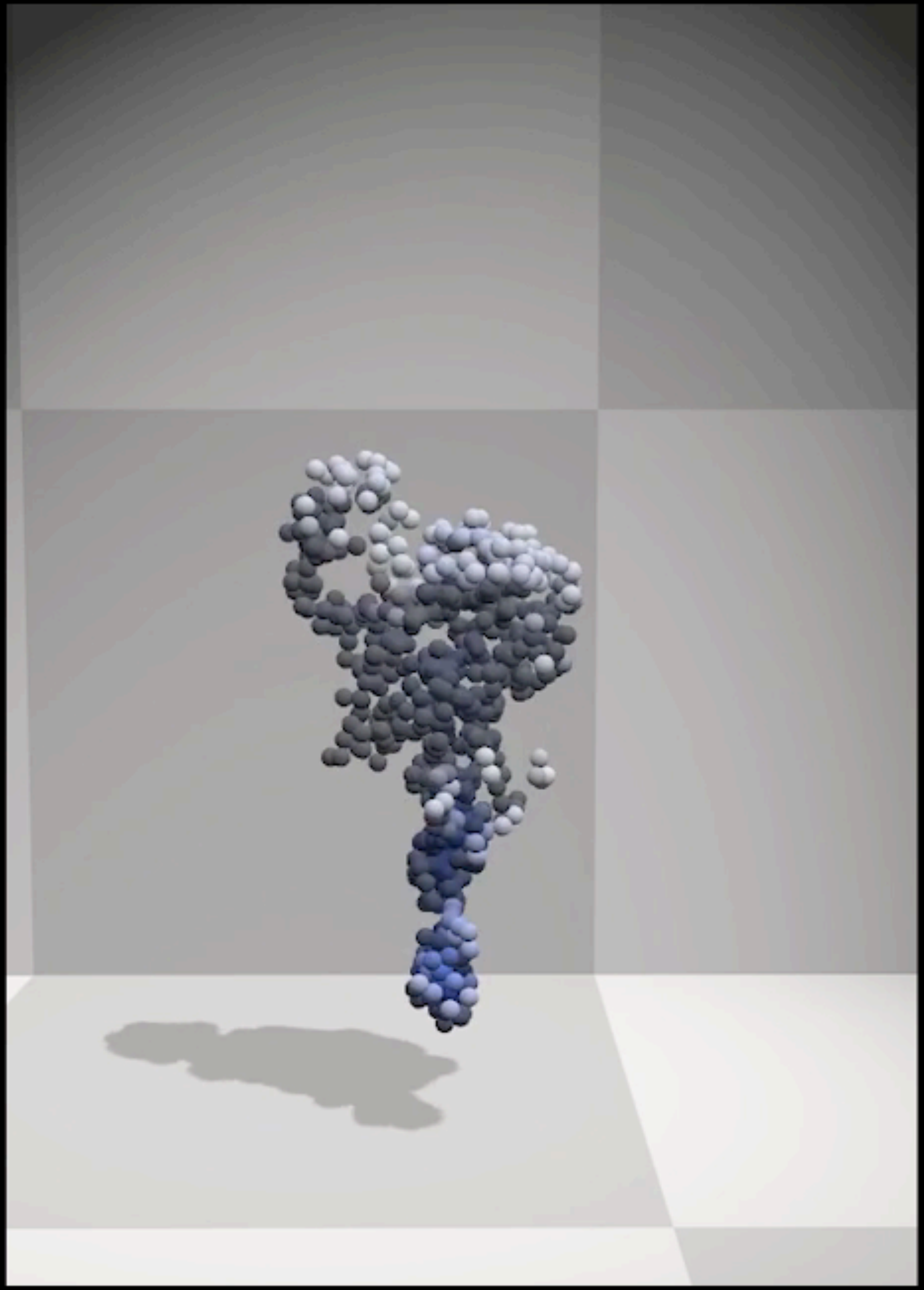

Baroclinic vorticity:

$$\frac{D\bar{\omega}}{dt} = \nabla\rho \times \nabla p$$

Driving vorticity:

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

Smoke Particles



Fluid Particles

# Smoke Rendering

- Back-to-Front sort particles in CUDA
- Point based rendering
- Approximate transmission using shadow-map  depth as input to scattering function

# Examples

# Limitations / Future Work

- Representing smooth surfaces problematic
- Would like parallel and robust collision of simplices
- Dynamic re-seeding for gases
- Iteration independence for non-stiff constraints

# Thank you!

# Acknowledgements

- Thanks to the PhysX team and the paper reviewers

- Contact details:
- mmacklin@nvidia.com
- @milesmacklin

# References


- Two-way coupling of fluids to rigid and deformable solids and shells, Avi Robinson-Mosher, Tamar Shinar, Jón Grétarsson, Jonathan Su, and Ronald Fedkiw, SIGGRAPH 2008
- Full two-way coupling of rigid and deformable bodies, T Shinar, C Schroeder, R Fedkiw, SIGGRAPH 2008
- Nucleus: Towards a unified dynamics solver for computer graphics, J Stam - Computer-Aided Design and Computer Graphics, 2009
- Robust treatment of collisions, contact and friction for cloth animation, R Bridson, R Fedkiw, J Anderson, SIGGRAPH 2002
- Example-based elastic materials, S Martin, B Thomaszewski, E Grinspun, SIGGRAPH 2011
- Nonconvex rigid bodies with stacking, E Guendelman, R Bridson, R Fedkiw, SIGGRAPH 2003