Making Your Game Fully Interactive by NVIDIA FleX

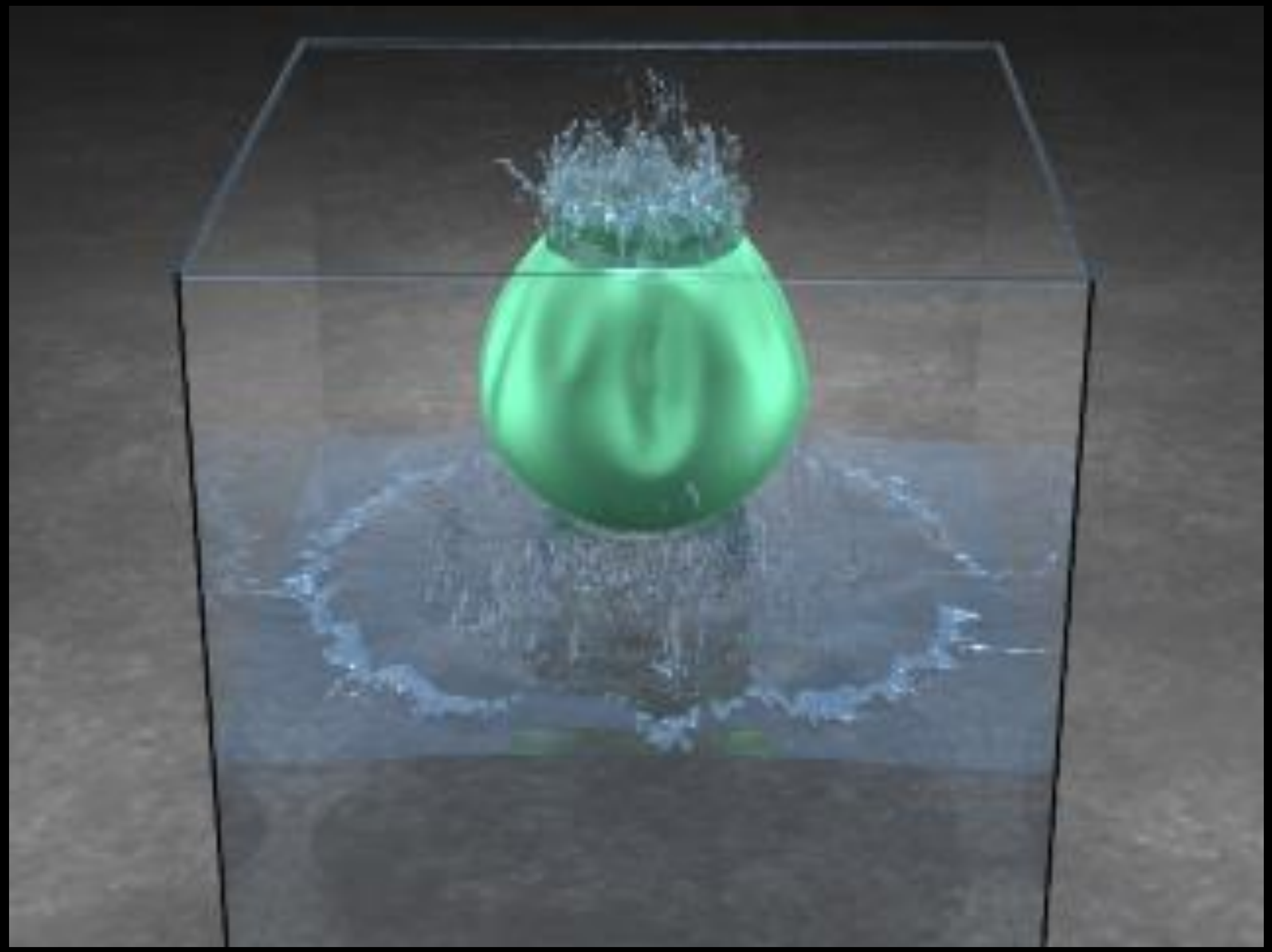
Quan Chen

NVIDIA

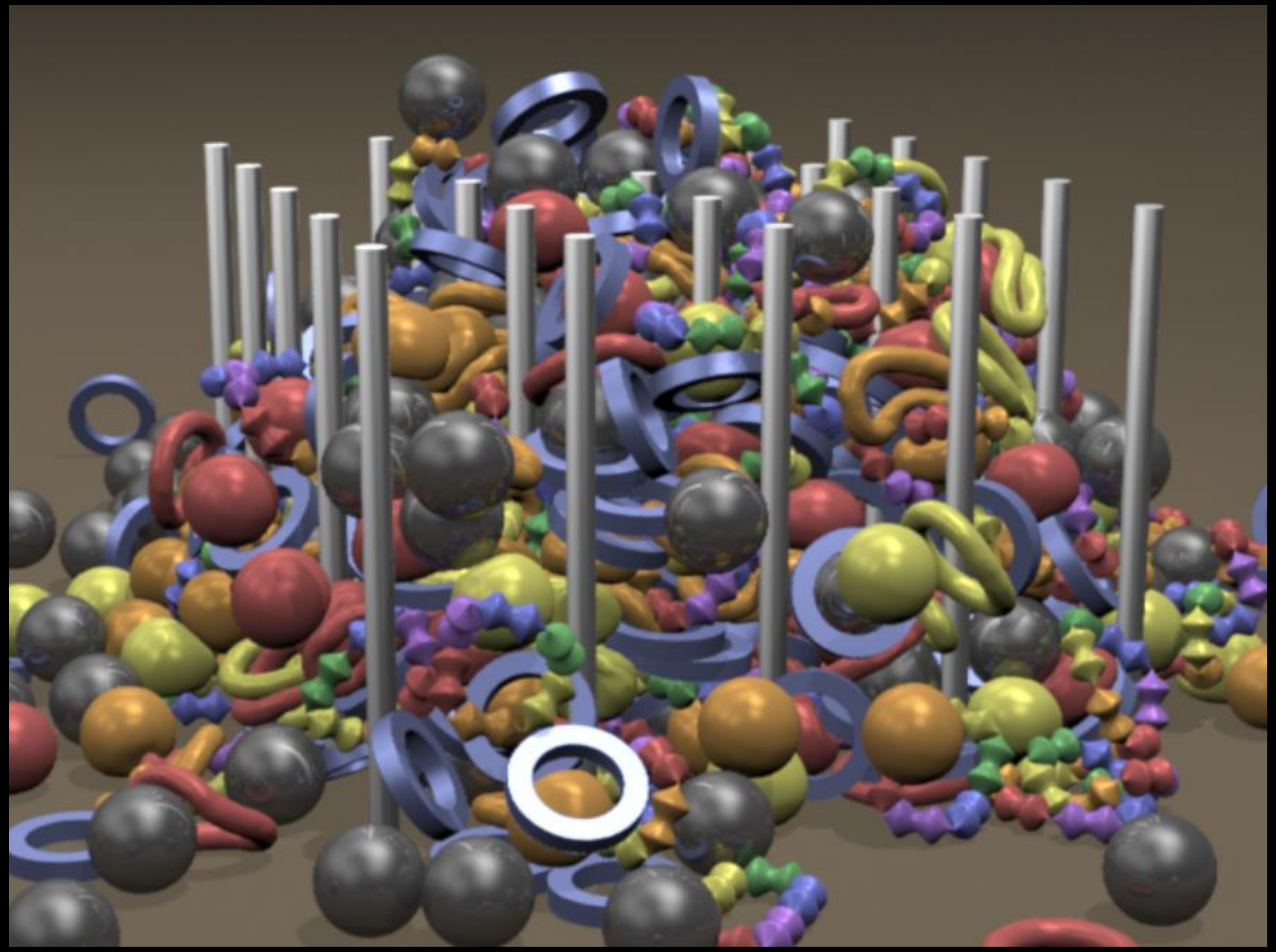
What’s FleX?

Motivation

- Too many solvers
- Creates redundant work
- Want one optimization target
- 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

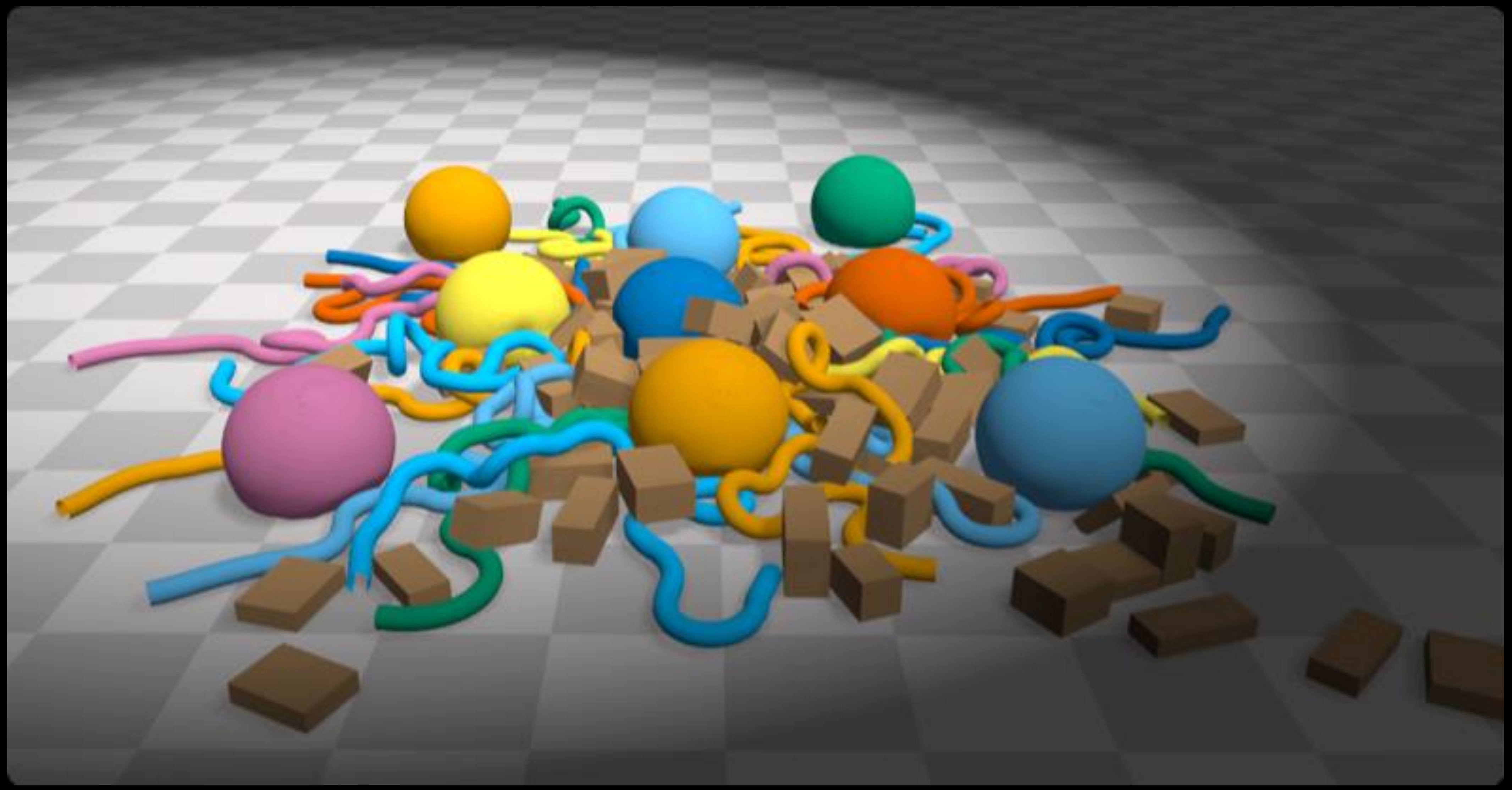
Advantages

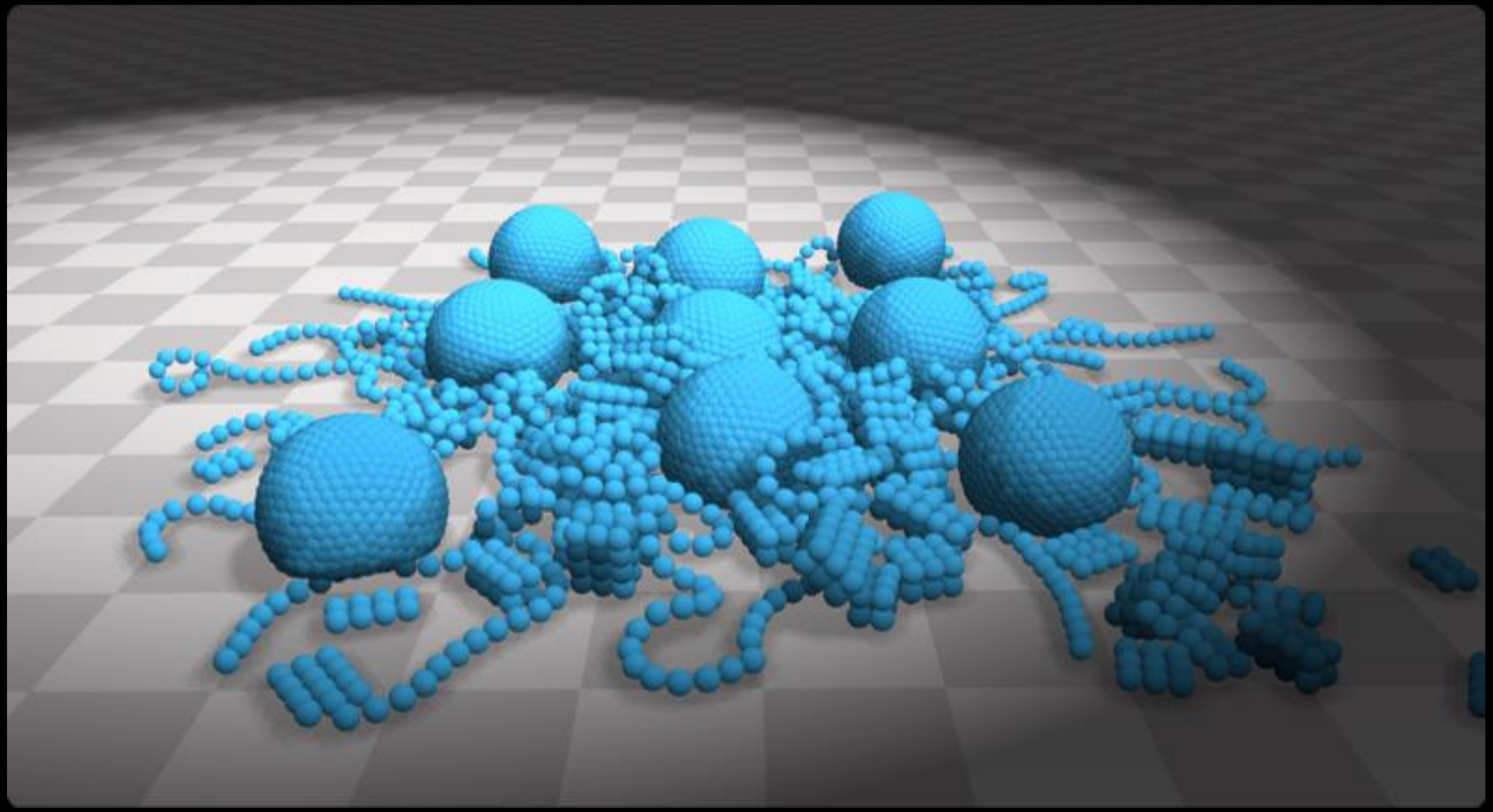
- Simplifies collision detection
- Stable two-way interaction of all object types:
  - Cloth
  - Deformables
  - Liquids
  - Rigid Bodies
  - Gases (not released)
- Fits well on the GPU

Particles

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

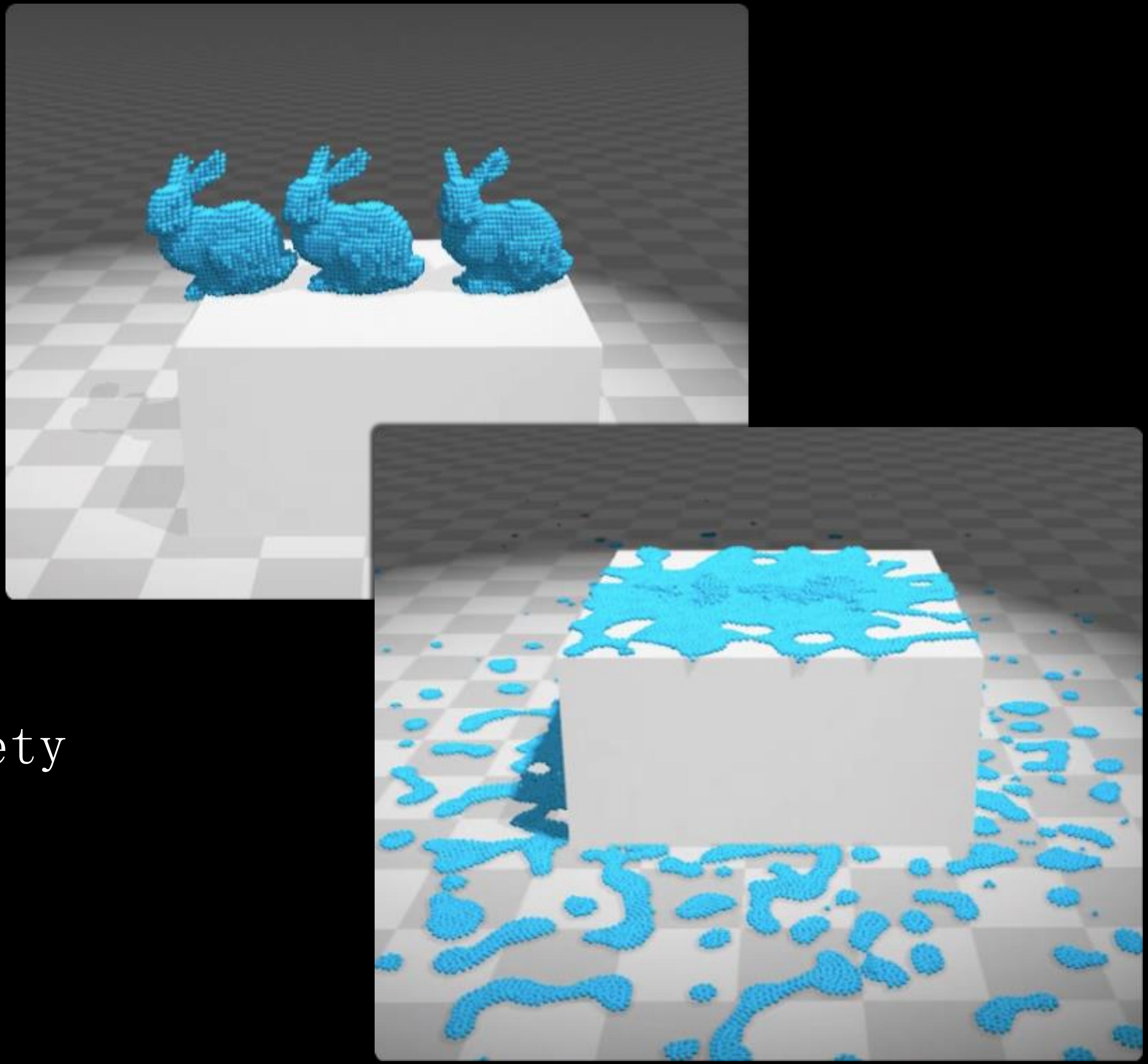
- Phase-ID used to control collision filtering
- Particles do not belong to a particular object
- Single collision radius





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



Solver Loop

1. Apply forces (v = v + 1/m*f*dt)
2. Predict new positions (x* = x + v*dt)
3. Find neighbors, contacts
4. Pre-stabilization
5. For (k iterations)
   1. For each constraint group G, in parallel:

      deltaX = 0

      Solve constraints in G

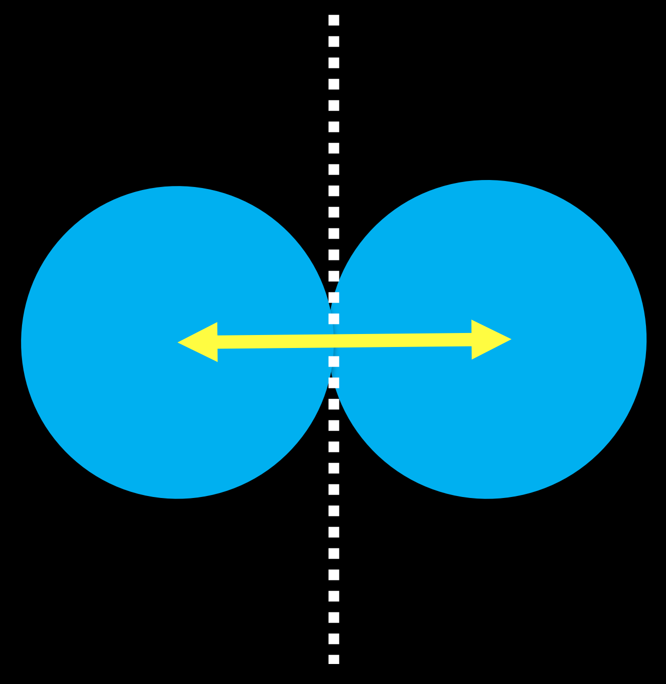
      x* += deltaX*(omega/n)

6. Update velocities (v = (x*-x)/dt)
7. Update positions (x = x*)

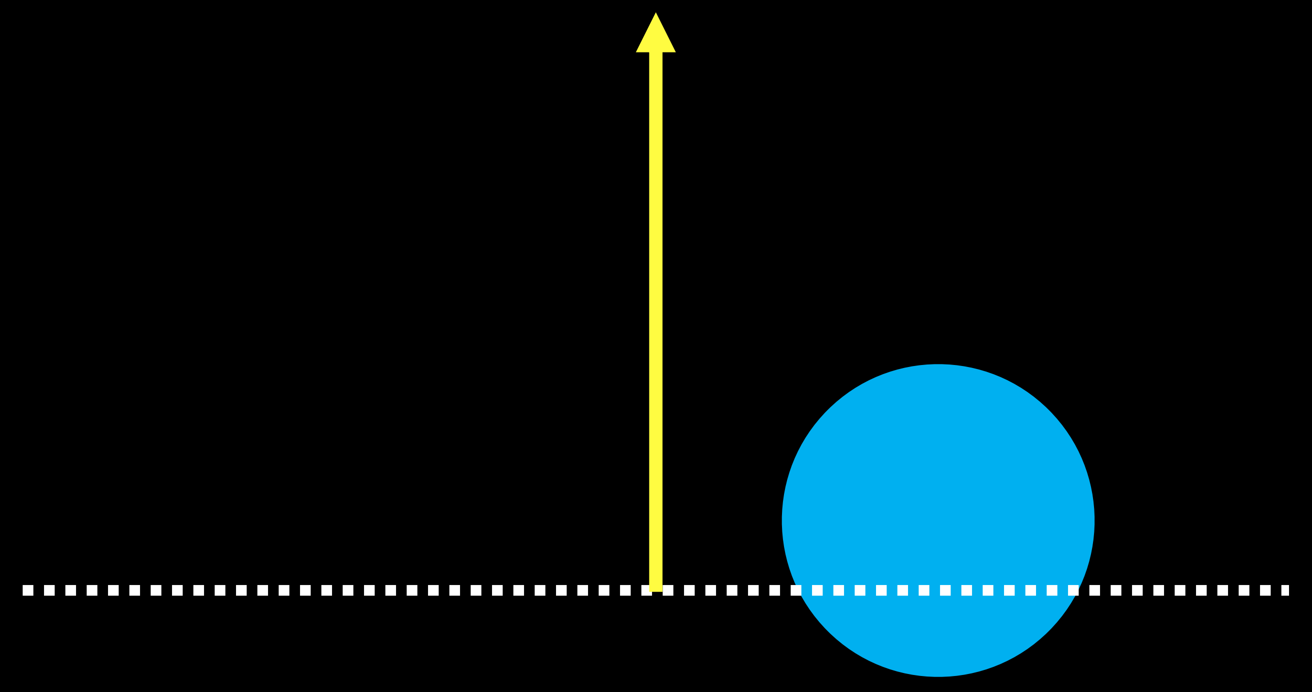
Contact and Friction

Collision Detection

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



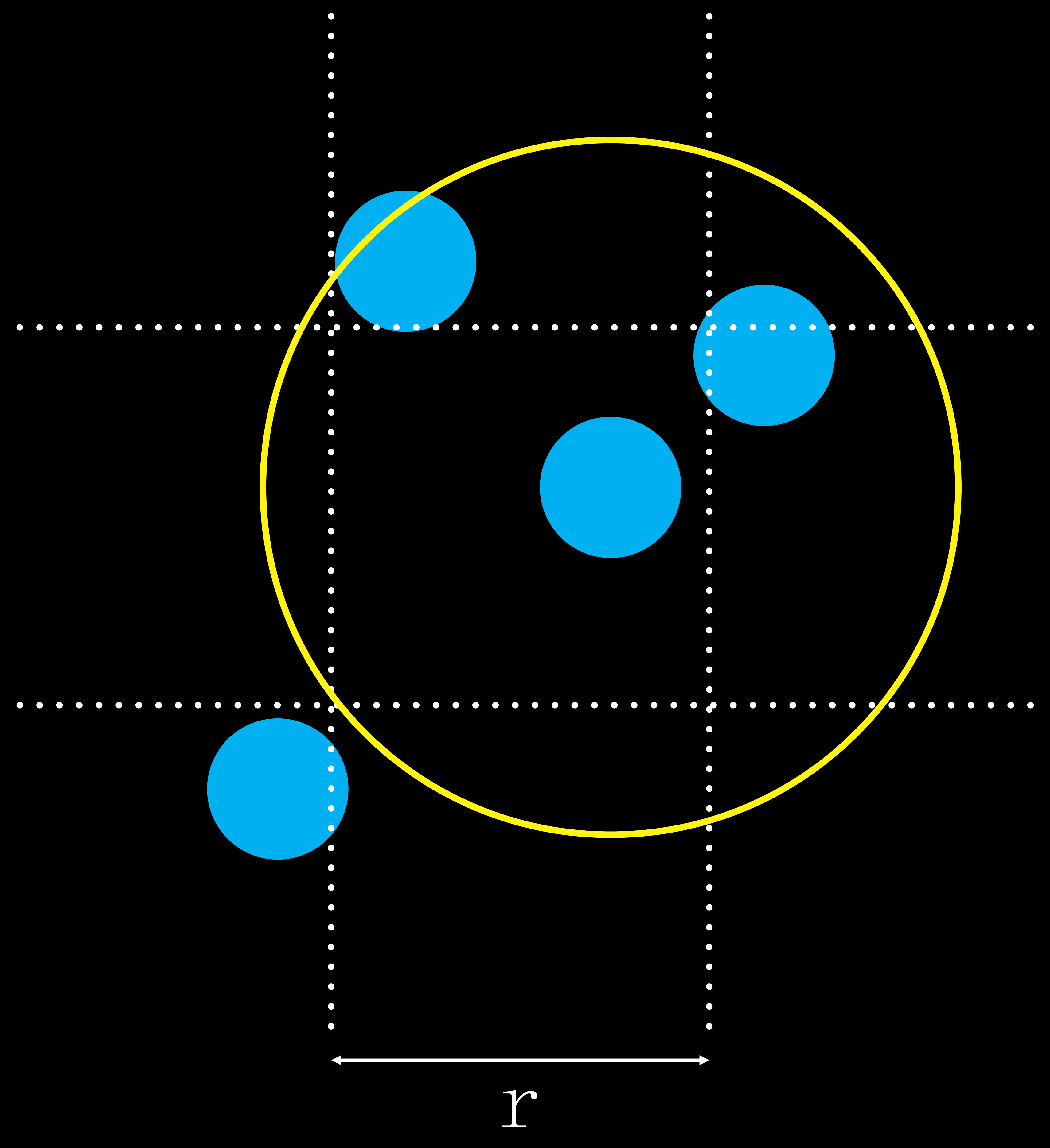
$$C_{contact} = |\mathbf{x}_i - \mathbf{x}_j| - 2r \geq 0$$



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

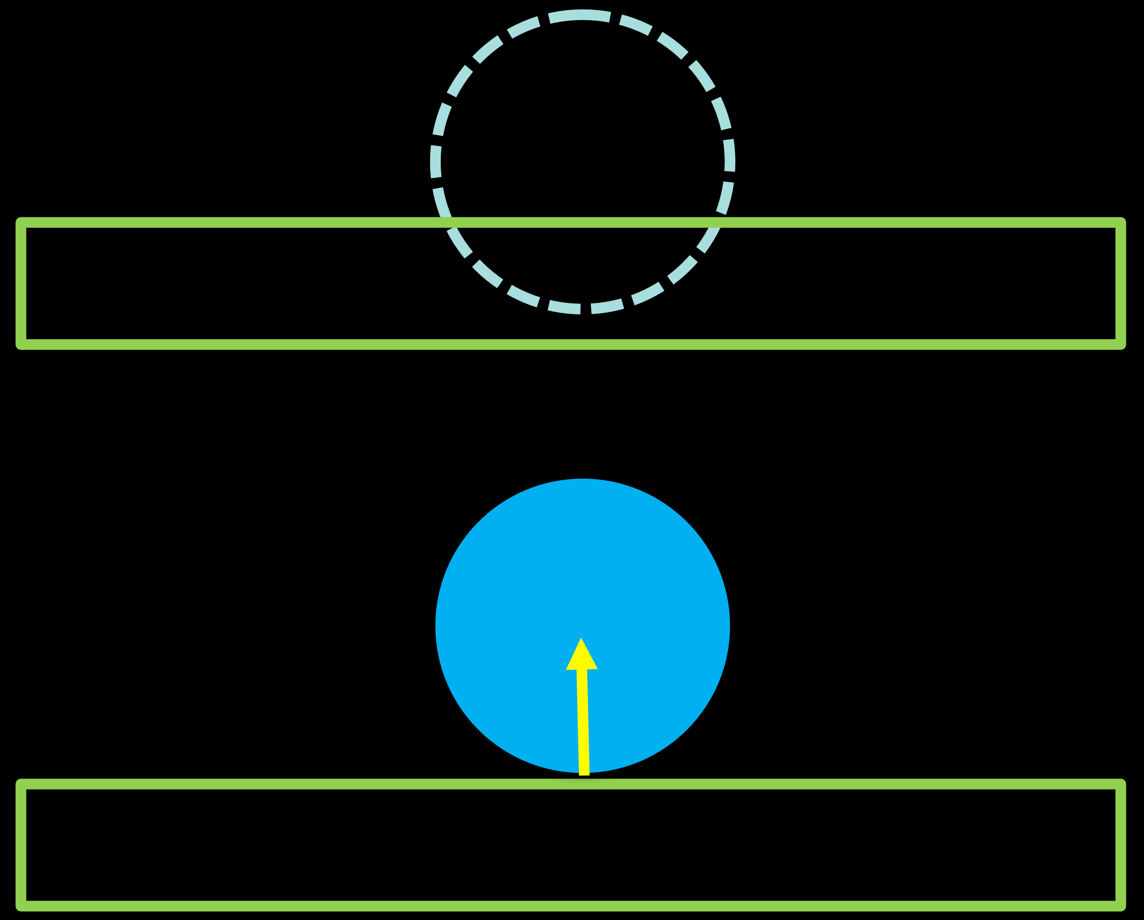
Collision Detection

- Particle-Particle
  - Tiled uniform grid
  - Fixed maximum radius
  - Re-order particle data according to cell index to improve memory locality

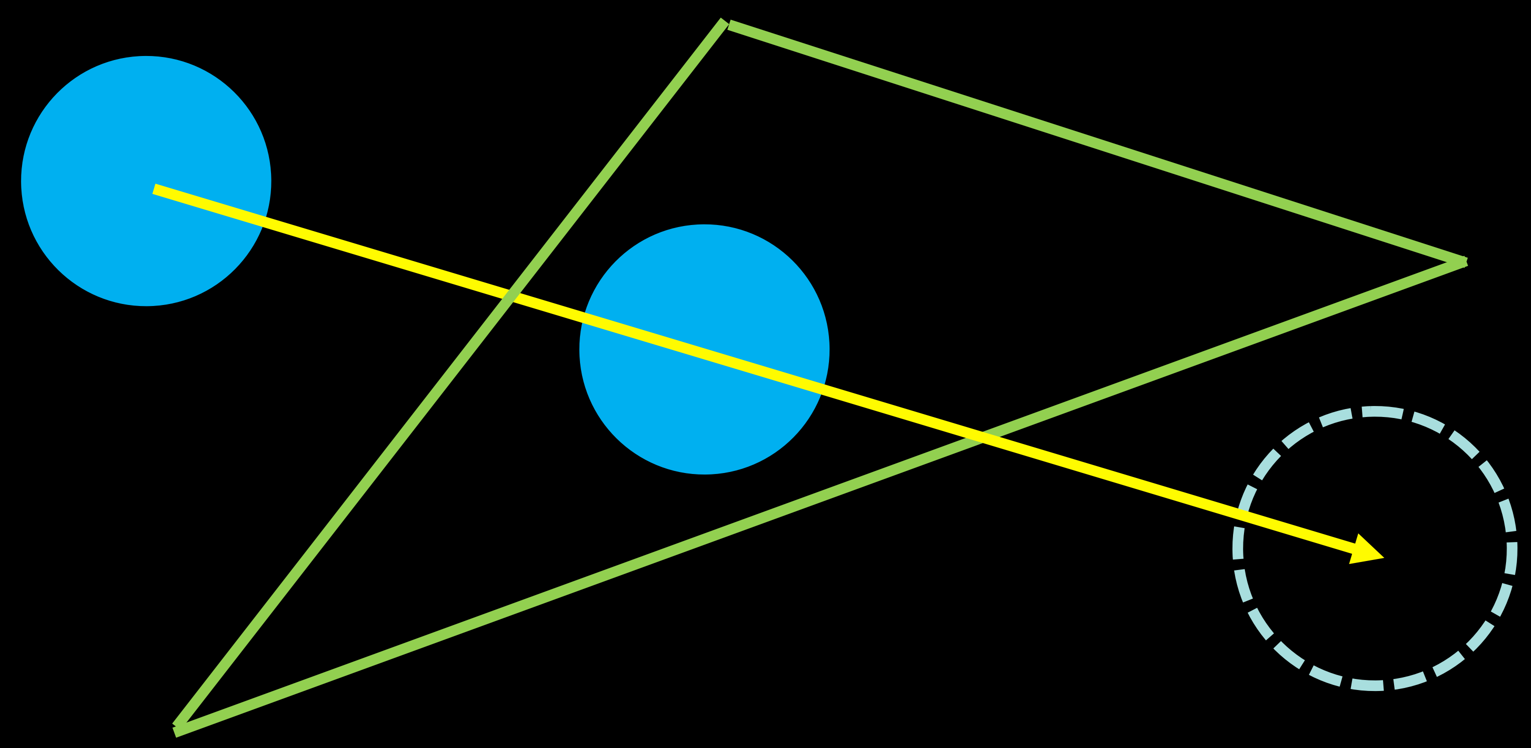


Collision Detection

- Particle-Mesh
  - Collision primitives
    - Plane
    - Sphere & Capsule
    - Convex
    - Triangle mesh (CCD)
    - Signed distance field
  - Friction (Kinetic, static)
  - Restitution



Convex Collision (MTD)
(projection)



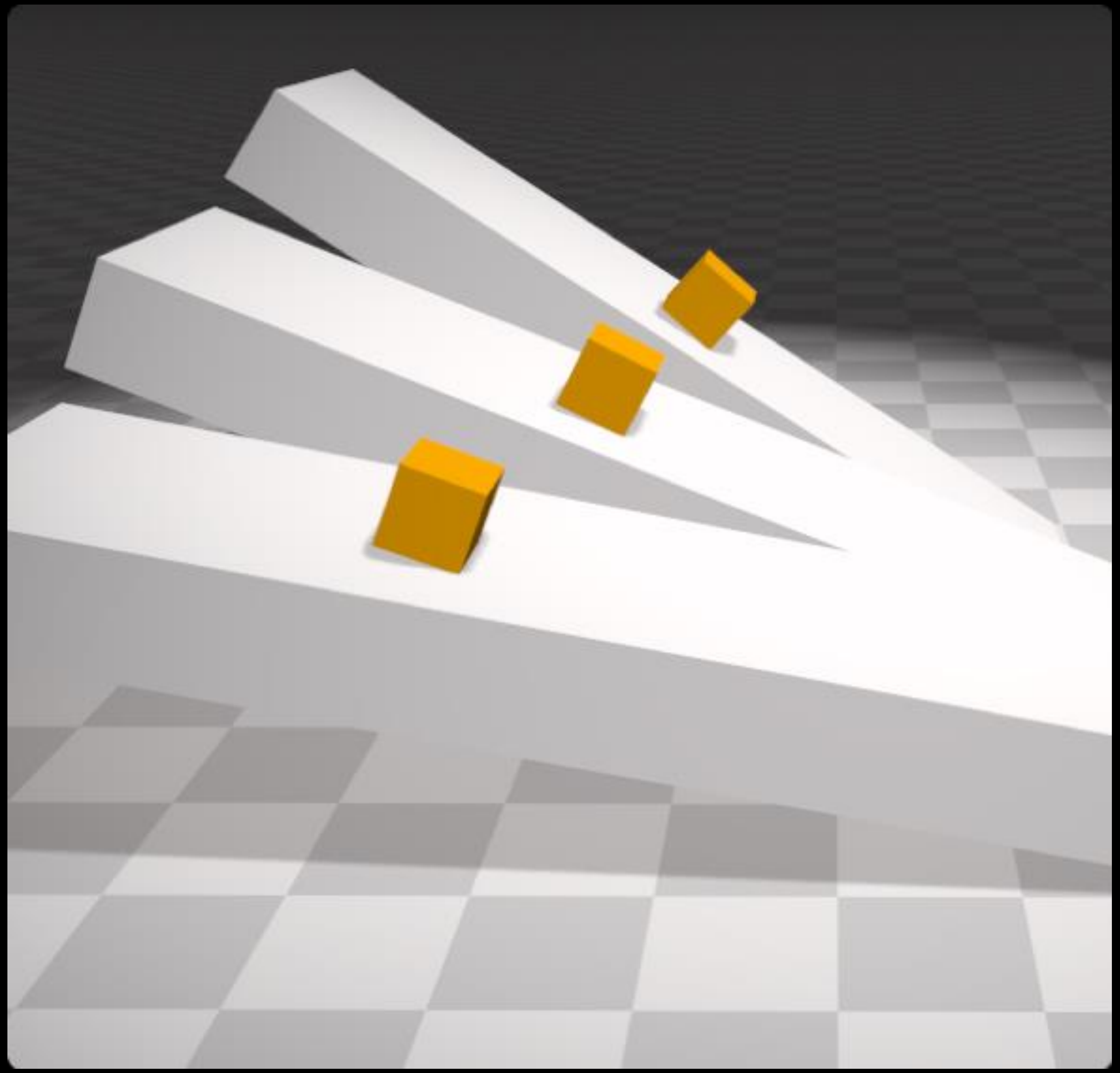
Triangle Collision (TOI)

Friction

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

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

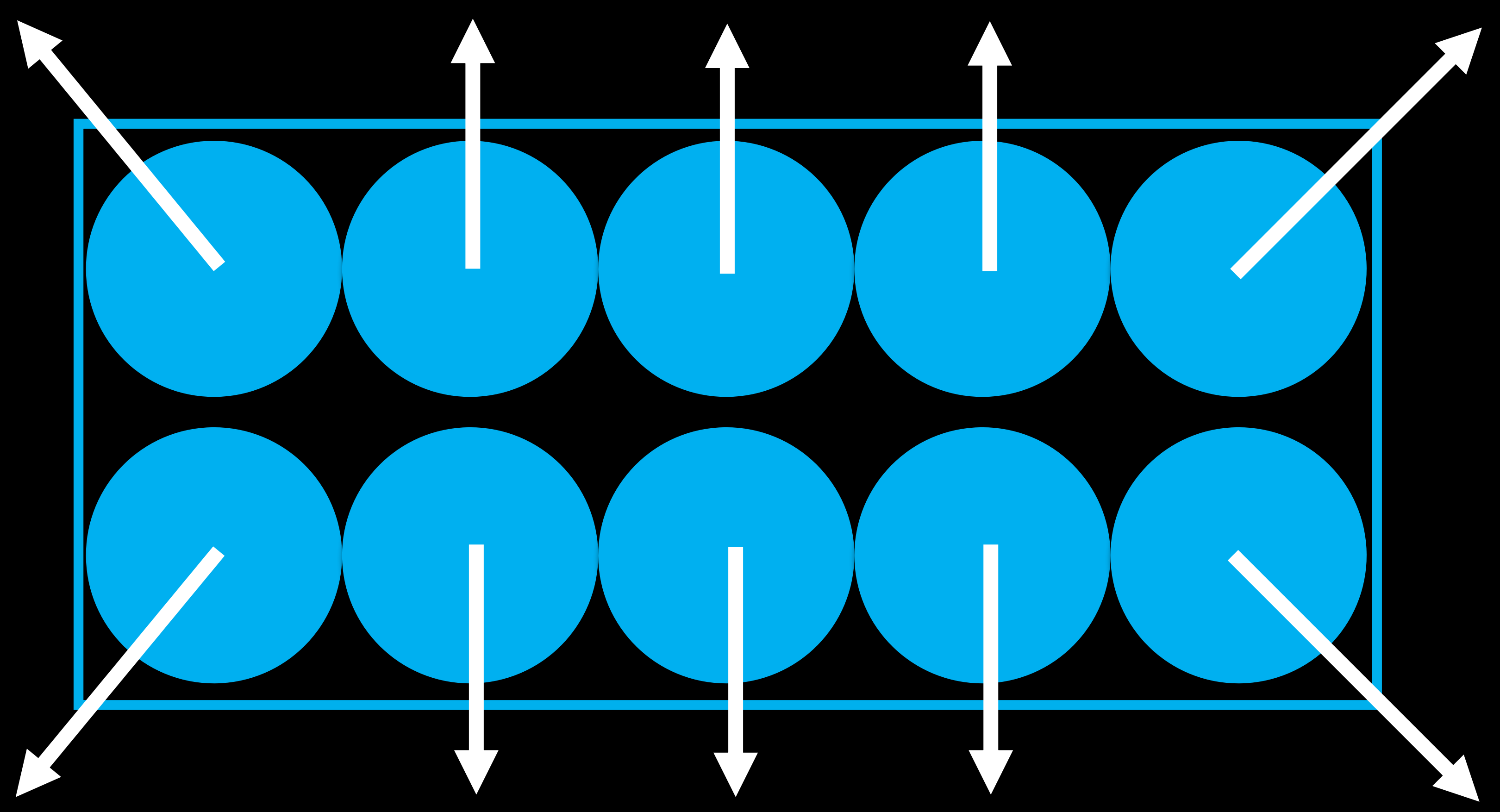
- Approximate Coulomb friction using penetration depth to limit lambda

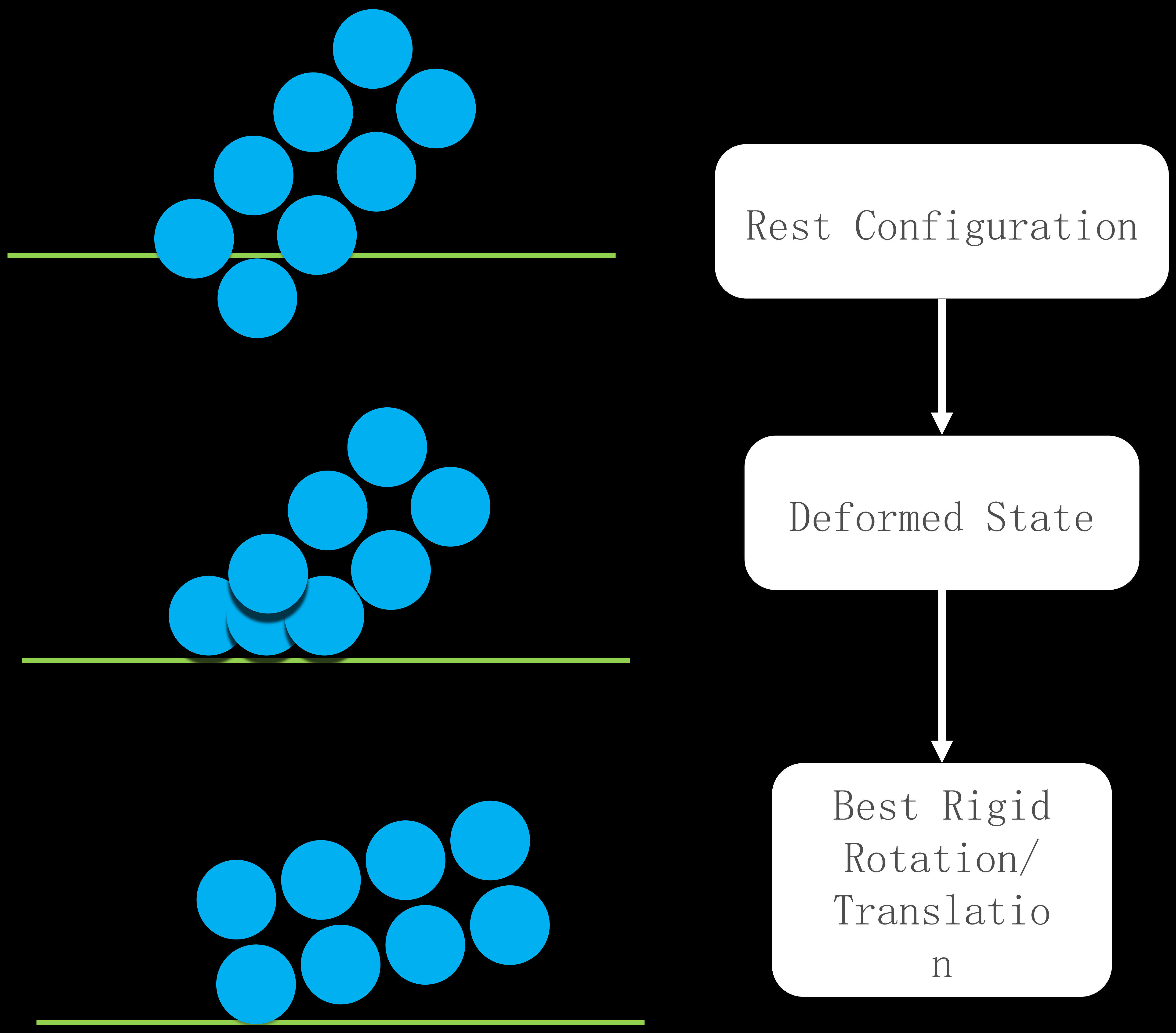


What can FleX do?

Rigid Bodies

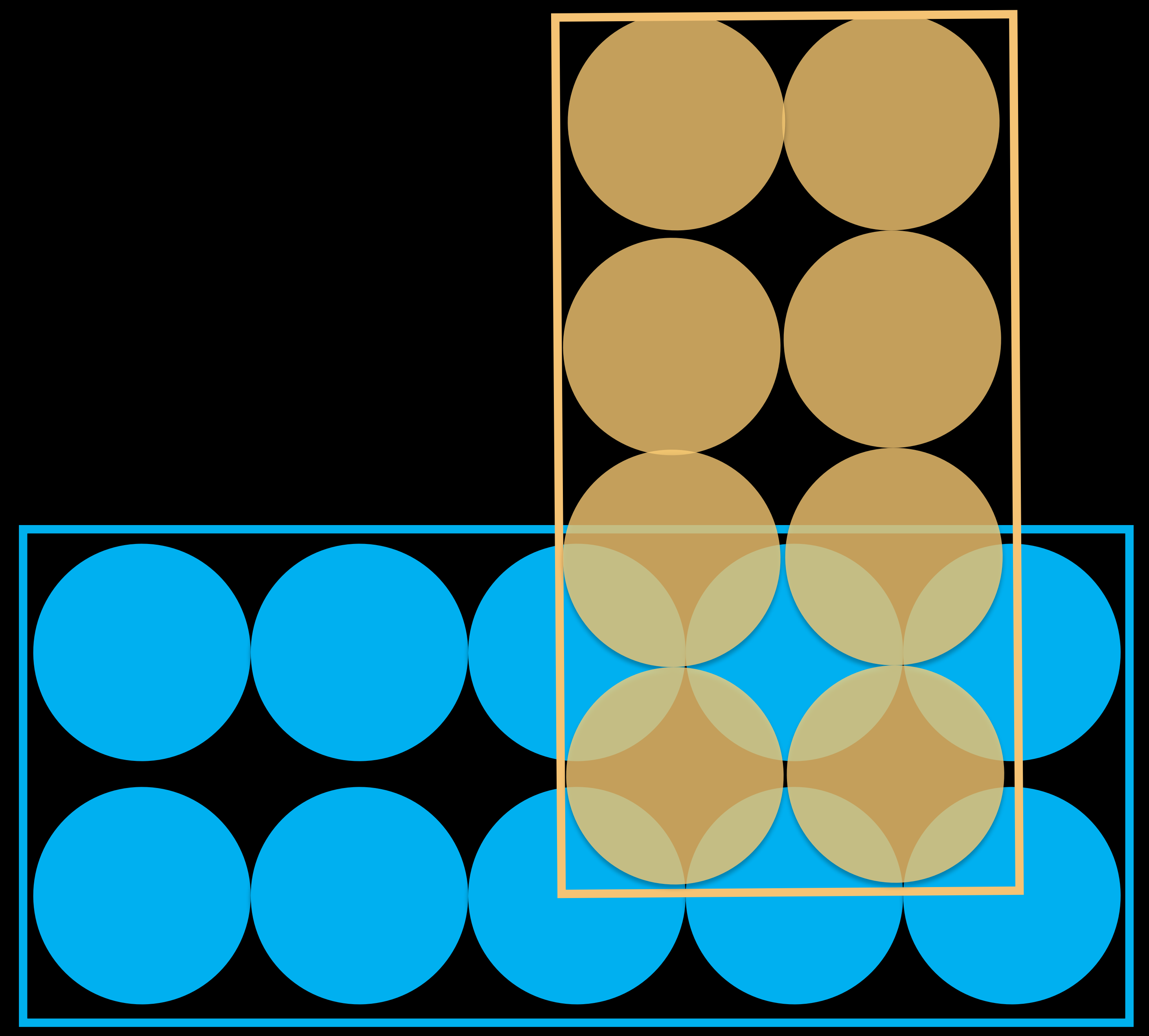
- Convert mesh->SDF
- Place particles in interior
- Add shape-matching constraint
- Store SDF dist + gradient on particles:

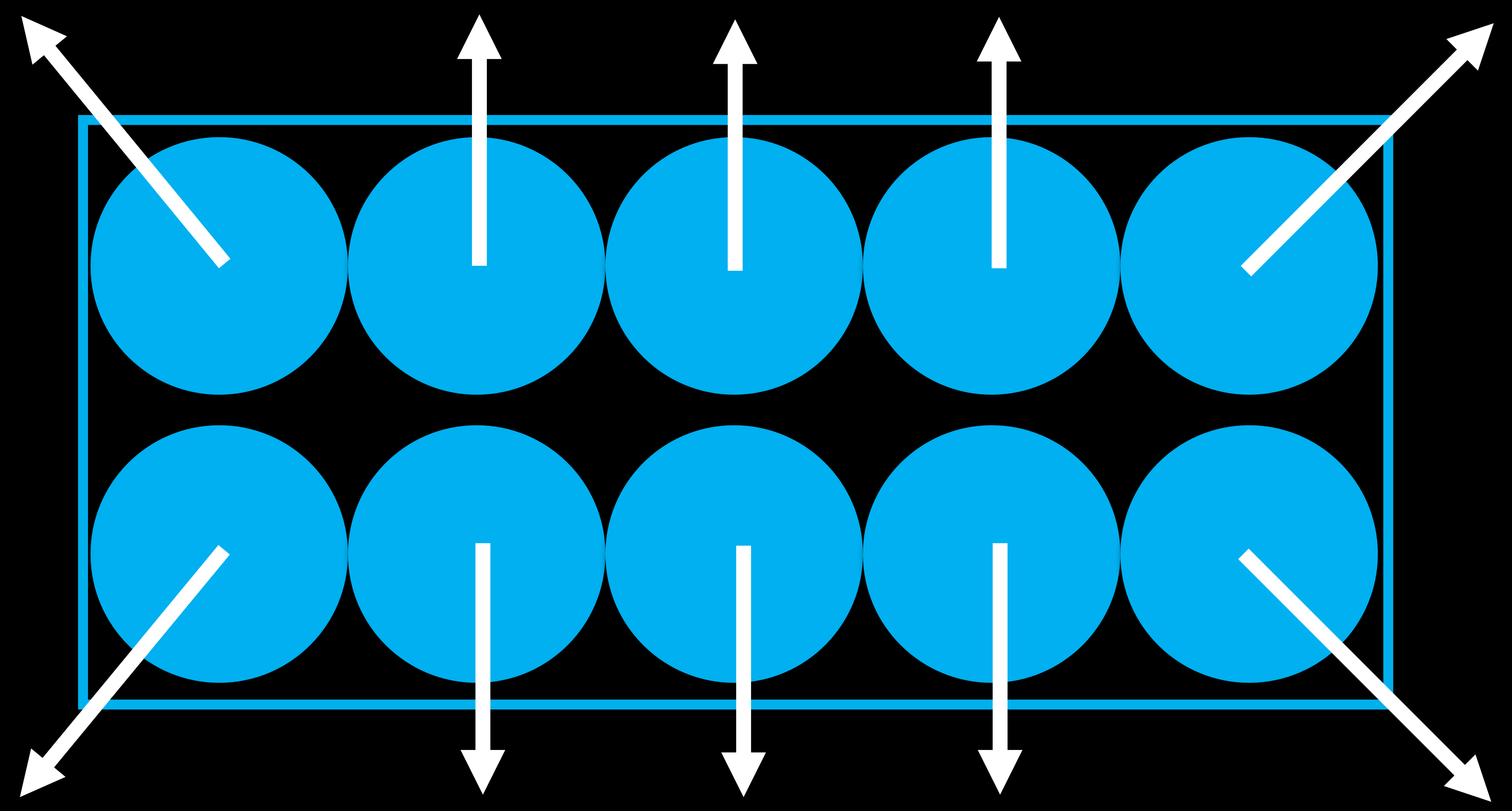




Rigid Collision

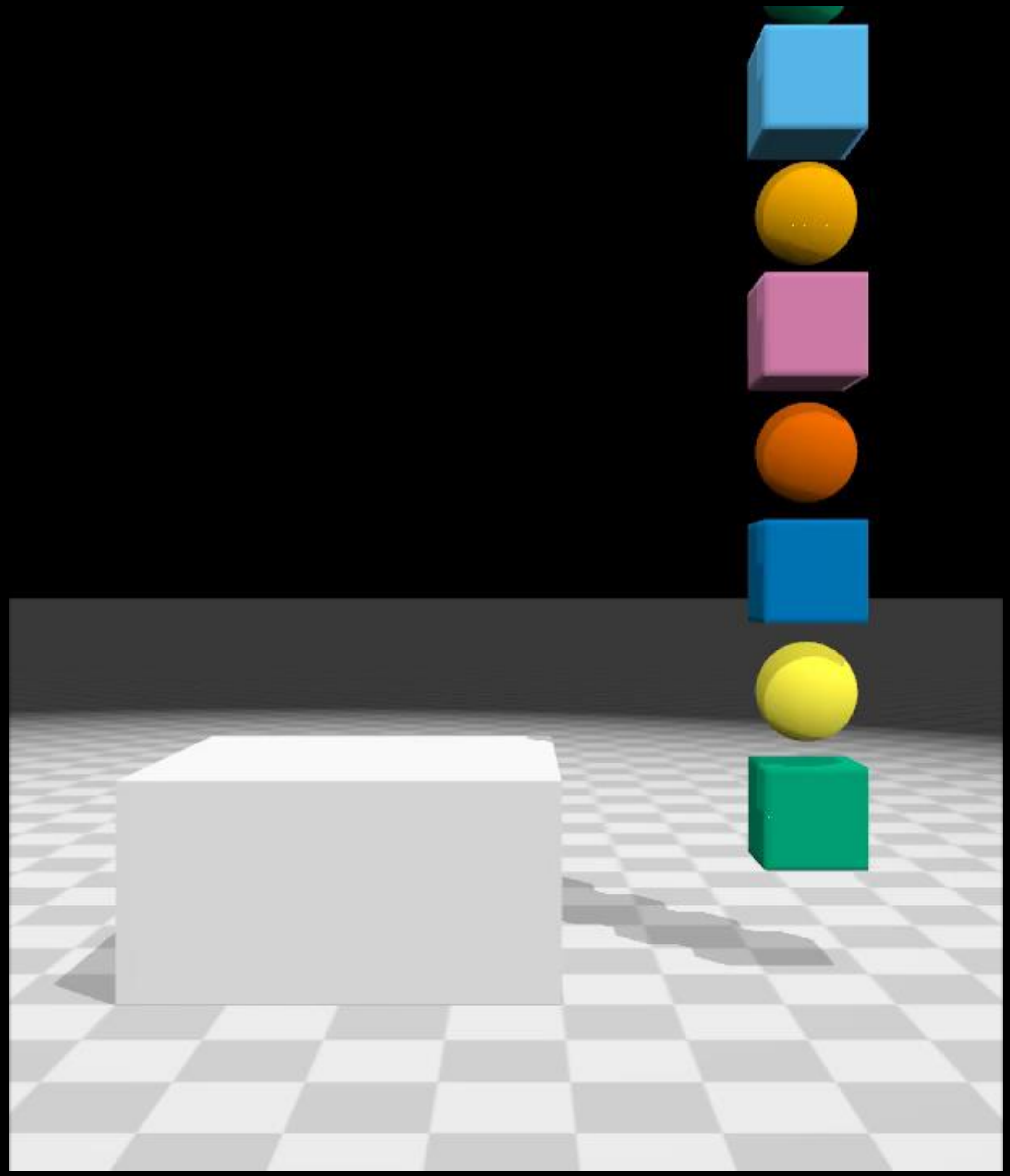
- Just colliding particles is not robust
- Shapes can become interlocked
- Use SDF stored on particles (distance + gradient) for interior
- Use “one-sided” particles at the surface [Müller & Chentanez 11]





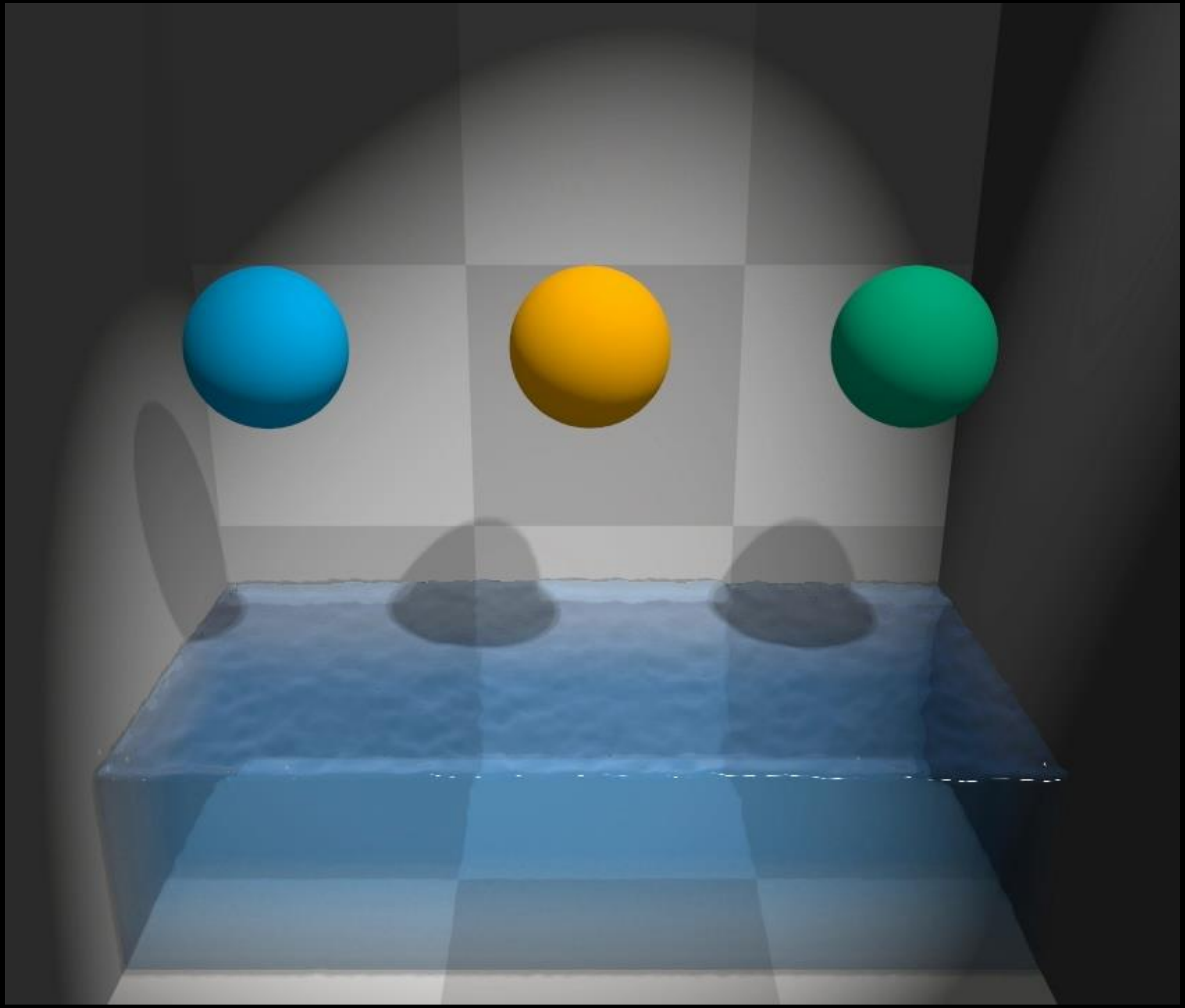
Plastic Deformation

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



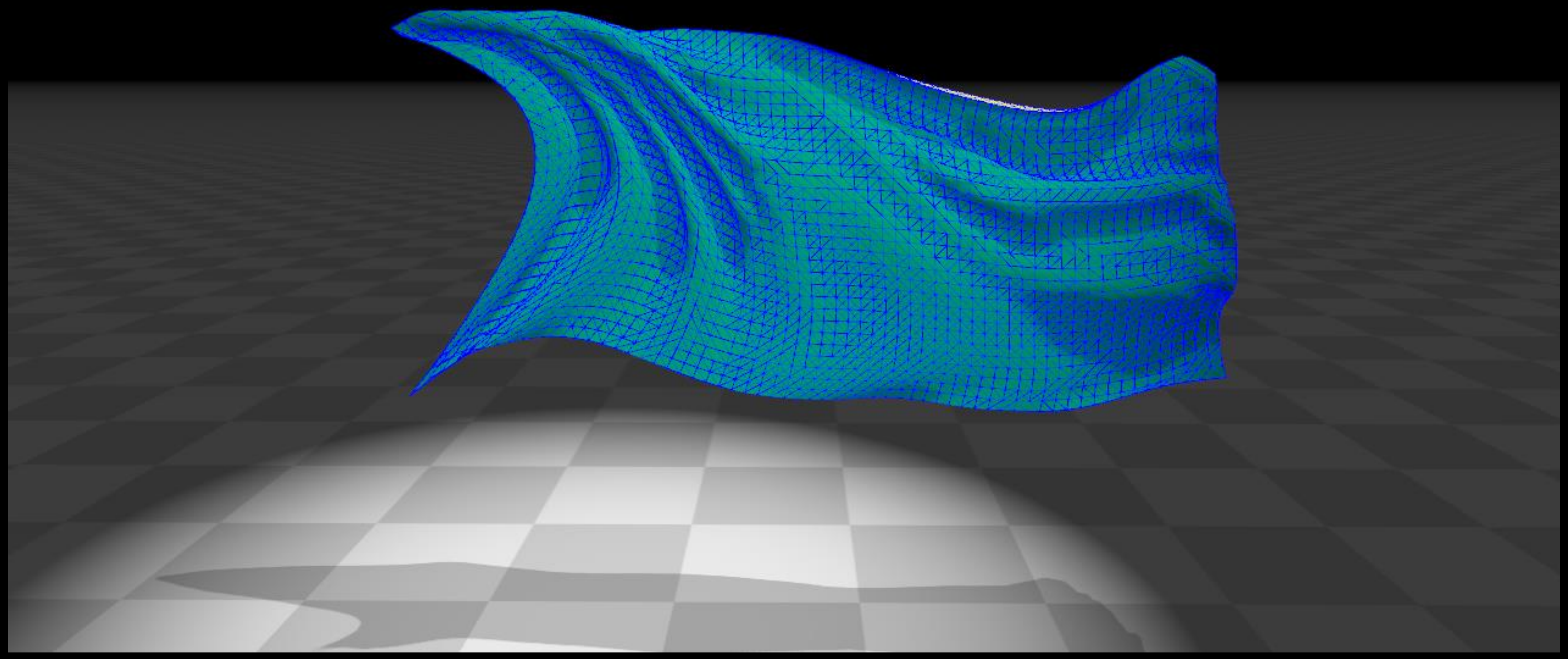
Two-Way Rigid Fluid Coupling

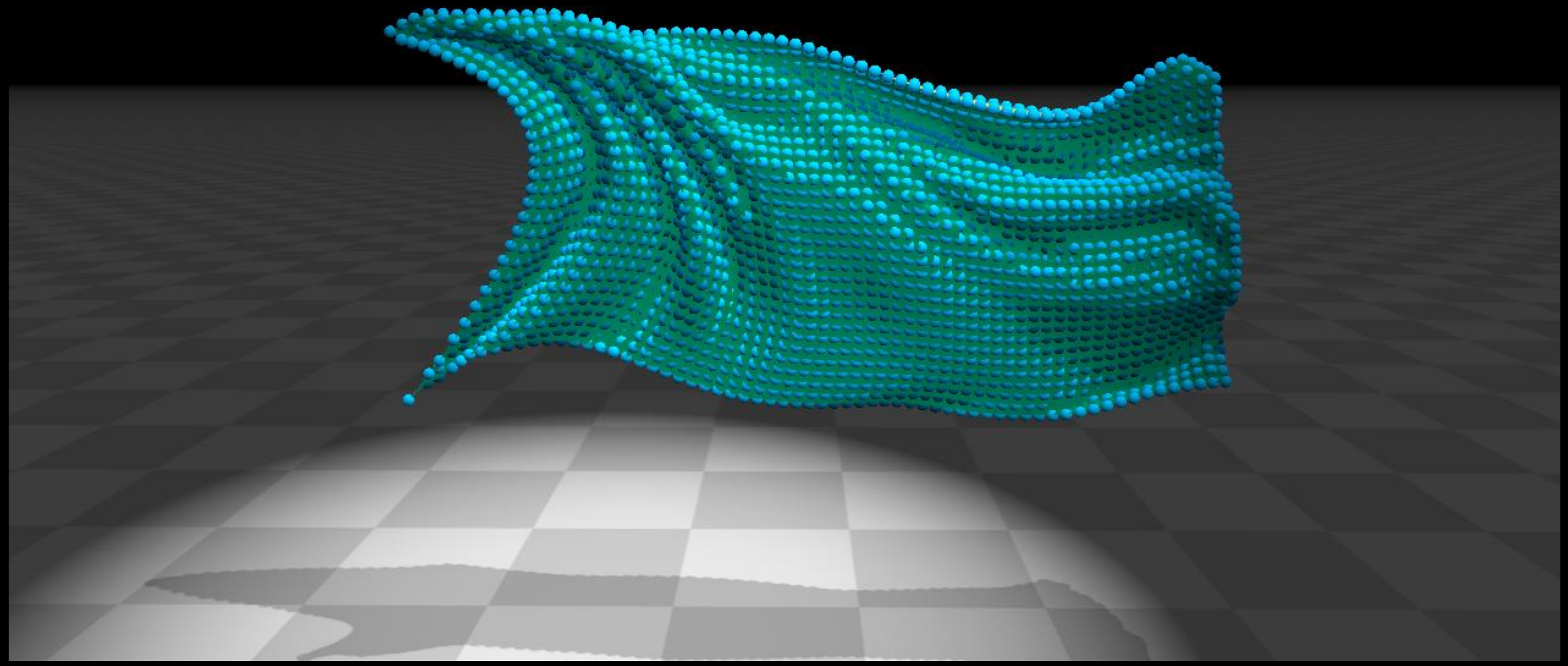
- Mostly automatic
- Include all particles in fluid density estimation
- Treat fluid->solid particle interactions as if both particles solid



Cloth

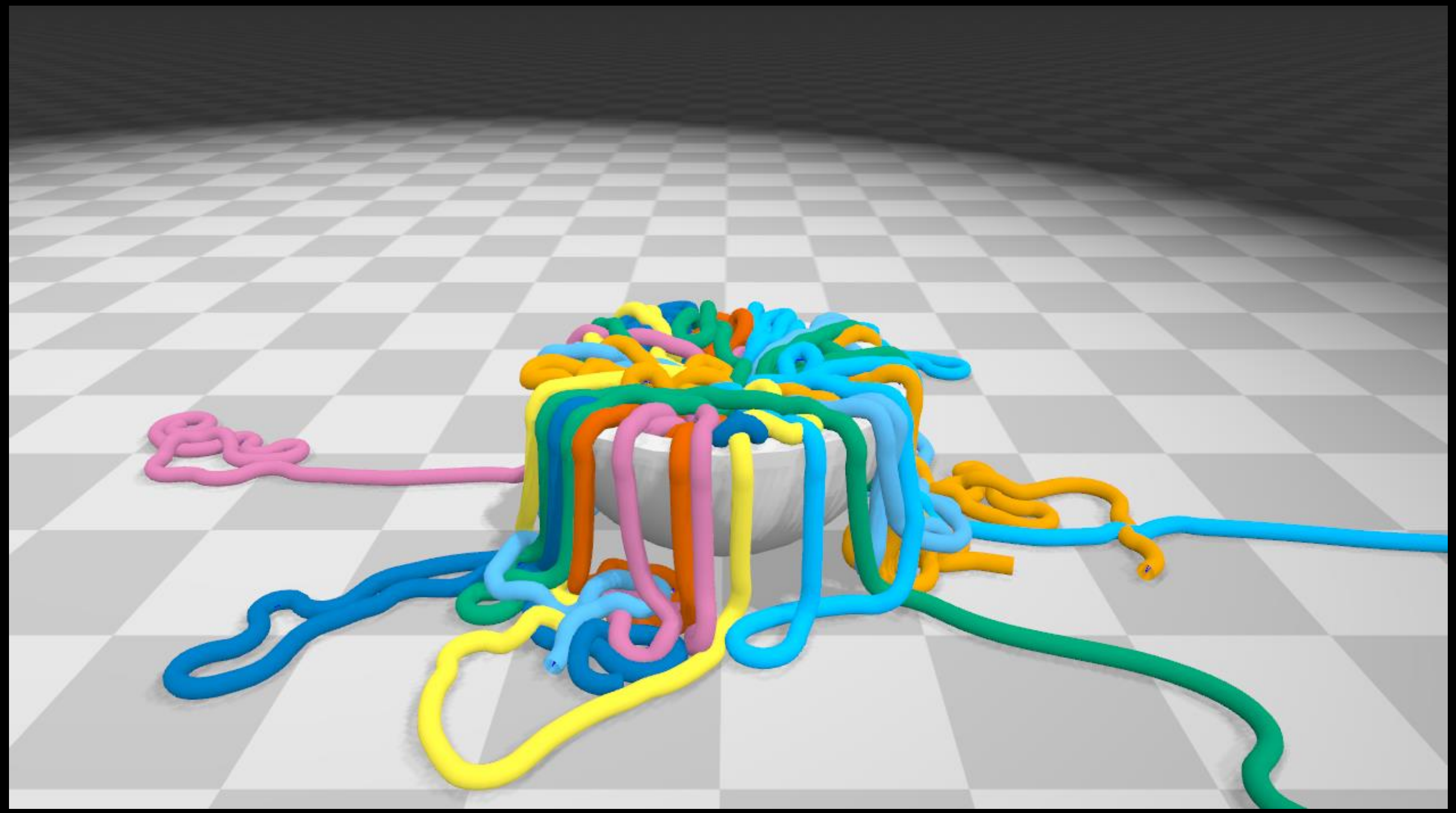
- Graph of distance + tether constraints
- Adding/removing constraints is easy (tearing)
- Self-collision / inter-collision automatically handled

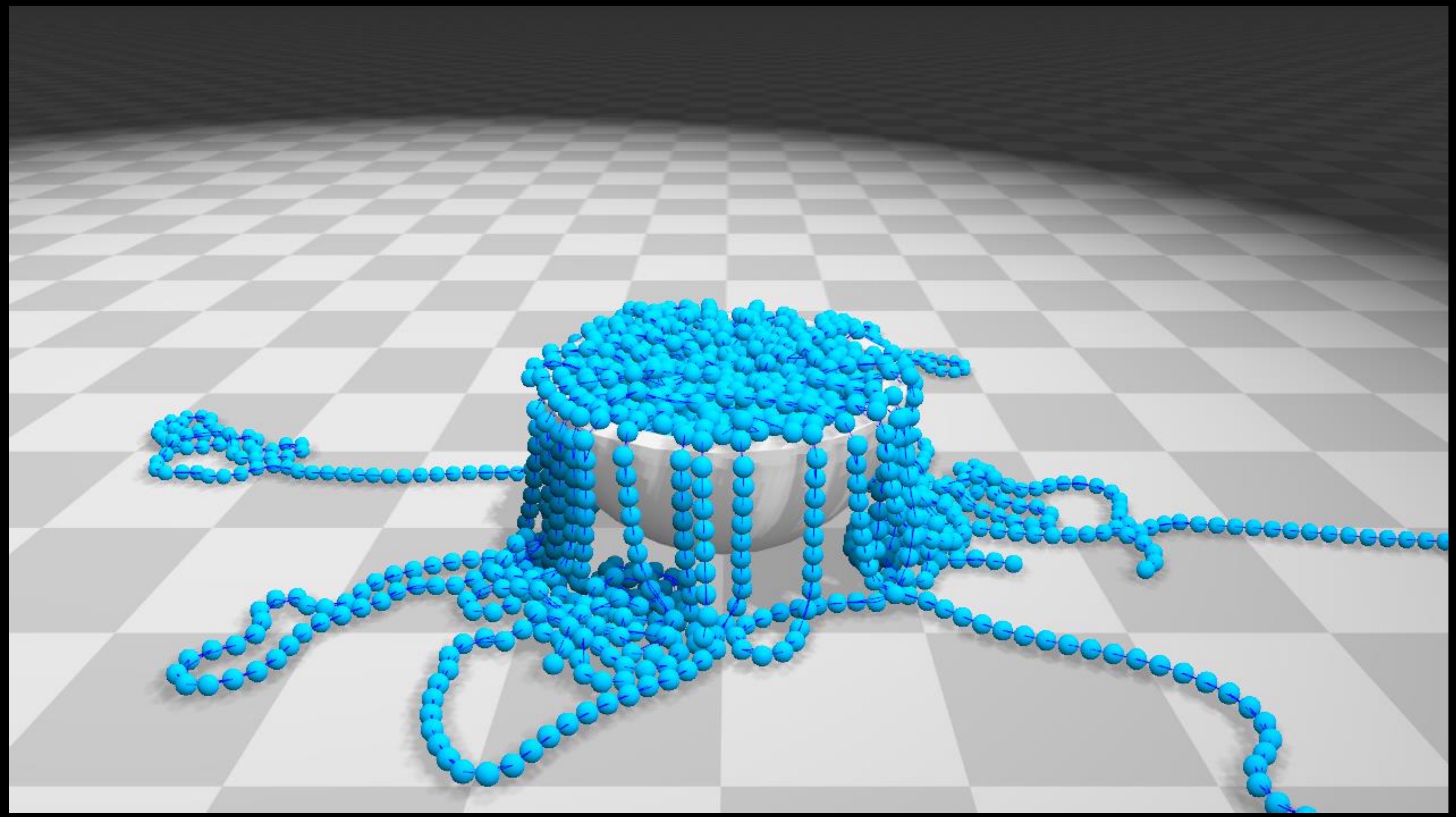


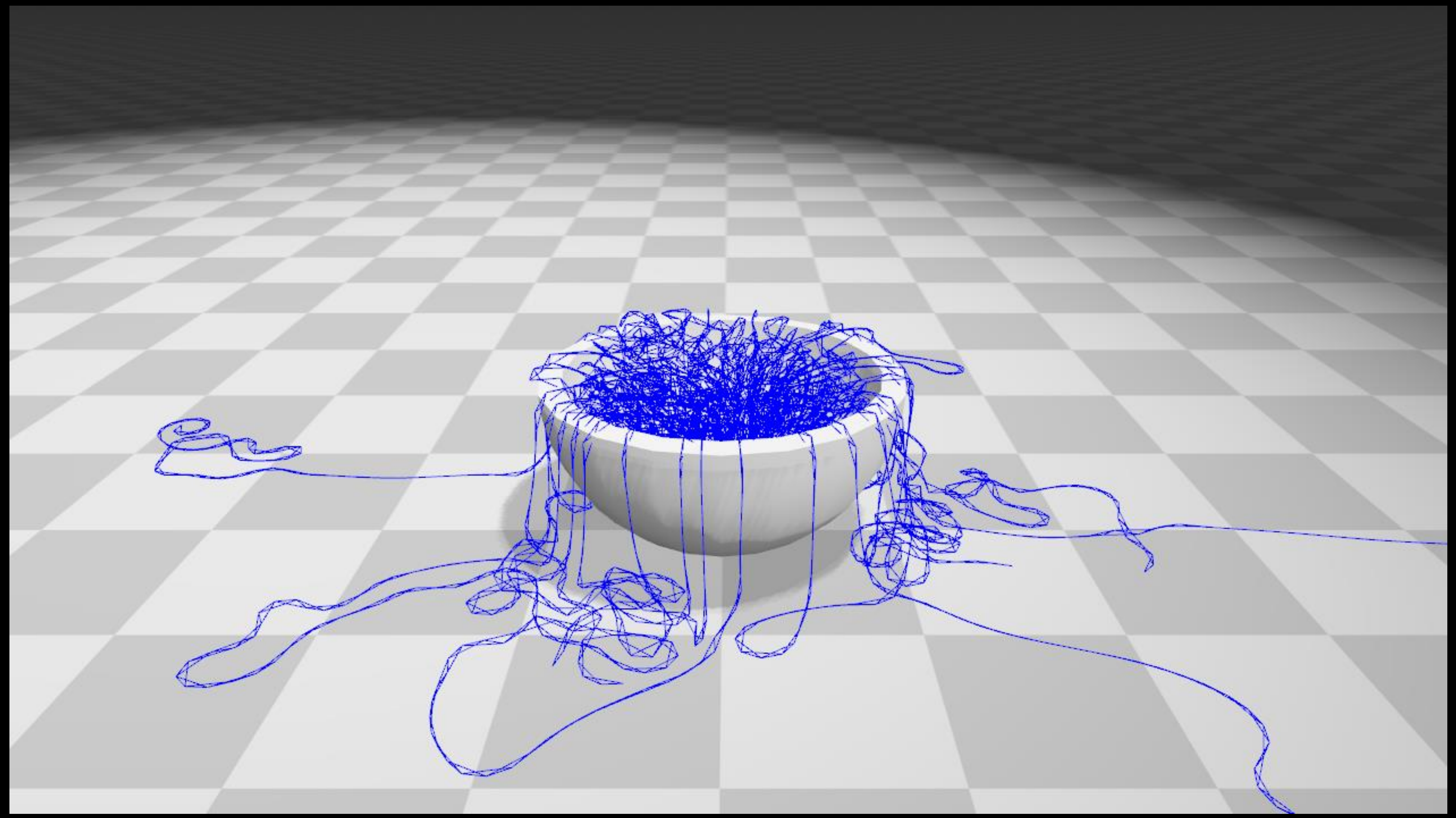


Ropes

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

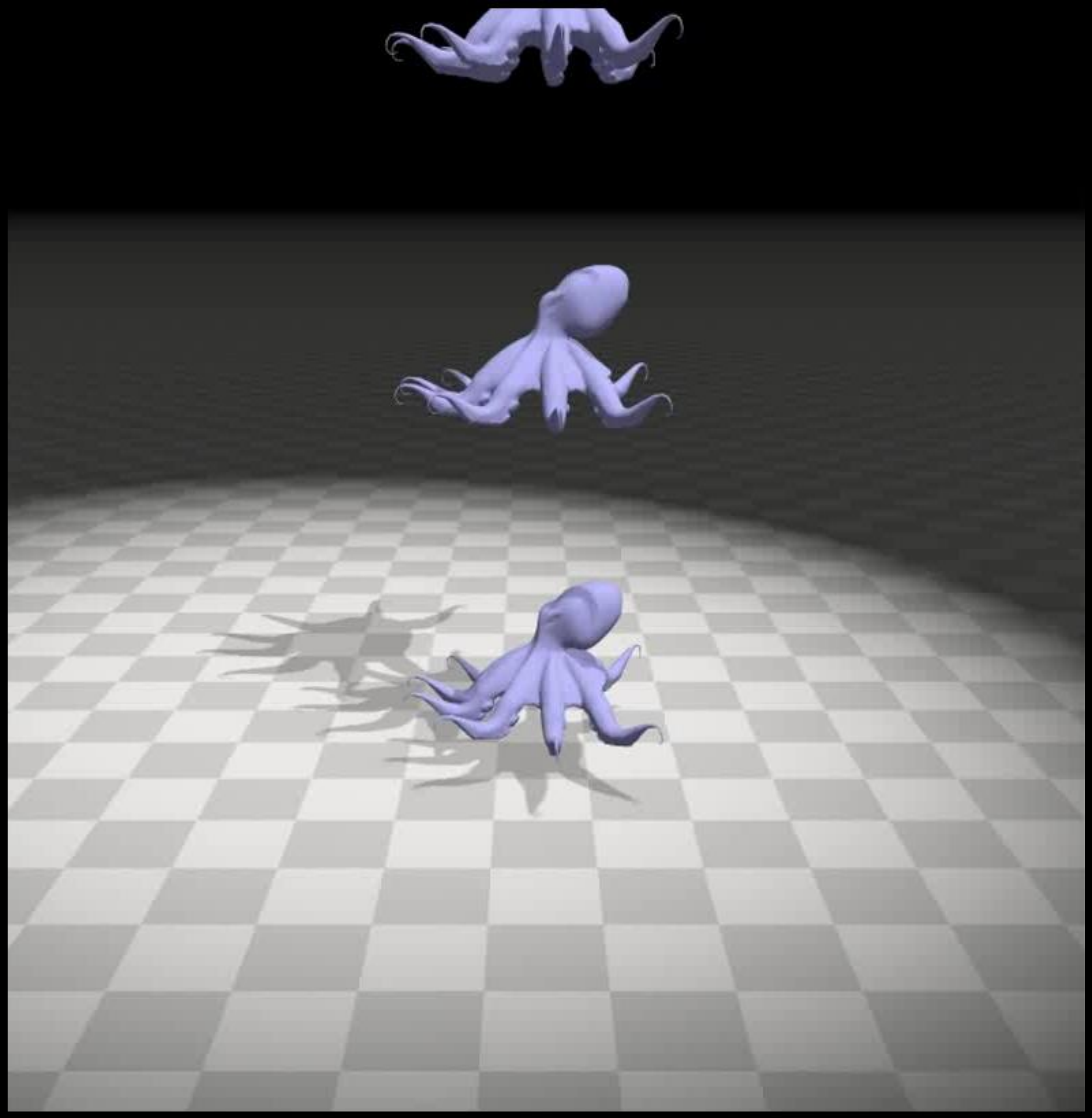






Deformables

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



Gases(not released yet)

PhysX Vs. FleX

PhysX Overview

- PhysX helps developers to make better games
  - PhysX is a complete physics solution
  - PhysX is a core component for game-play *and* effects
  - PhysX is highly competitive on all major platforms: consoles, mobile devices…and PCs, with or without GPU acceleration

What’s the same

- Both are physics simulation engines
- Support similar feature set
  - Rigid Bodies
  - Cloth
  - Fluid & Particles

What’s different

- Platform
  - PhysX: all platforms, from mobile, console, to PC, including GPU acceleration
  - FleX: CUDA
- Solver
  - PhysX: solvers per feature
  - FleX: unified solver
- Game logic
  - PhysX: friendly to game logic
  - FleX: require mapping particles to game actor and need more callbacks

What’ s different

- PhysX has more game related features
  - CCT, joints, vehicle controller, serialization
  - Scene queries, e.g ray cast and overlap tests
- FleX has more inter-feature interactivity in nature
- Usually FleX needs to be coupled with PhysX
  - Large scale terrain, buildings
  - Two-way interaction between CCT and dynamics

FleX Integration

FleX Integration

- FleX SDK has two parts
  - Core Library
  - Extensions Library
- FleX Solver can be embedded inside any authoring tools
  - UE3/4
  - Max/Maya
  - Standalone

Core Library

- C-style API
- Single .h interface, flex.h + flexRelease.dll
- Bulk operations only, example:

```
FLEX_API void flexSetVelocities(FlexSolver* s, const float* v, int n, FlexMemory source);
FLEX_API void flexGetVelocities(FlexSolver* s, float* v, int n, FlexMemory target);

FLEX_API void flexSetPhases(FlexSolver* s, const int* phases, int n, FlexMemory source);
FLEX_API void flexGetPhases(FlexSolver* s, int* phases, int n, FlexMemory target);
```

- CUDA code
- Supports interop through device->device copies

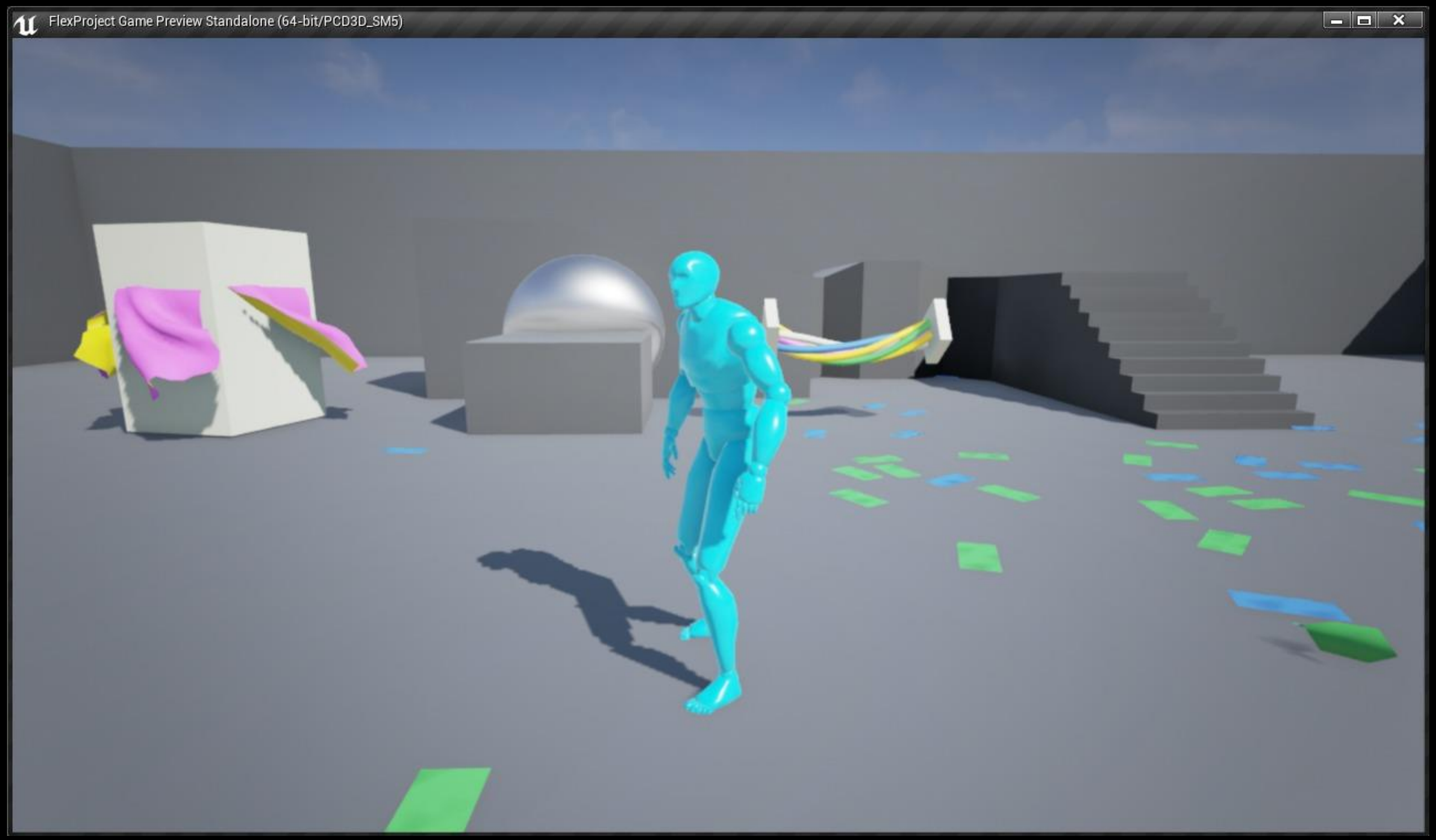
Extensions Library

- C-style API
- Single .h interface, flexExt.h + flexExtRelease.dll
- Helpers for:
  - Allocating and removing particles (freelist management)
  - Converting meshes to particles via voxelization
  - Creating constraint graphs for clothing
  - Creating mass-spring systems from tet-mesh
- Allows users to build lifetime management how they like
- No CUDA code, talks to core API only

Current Status

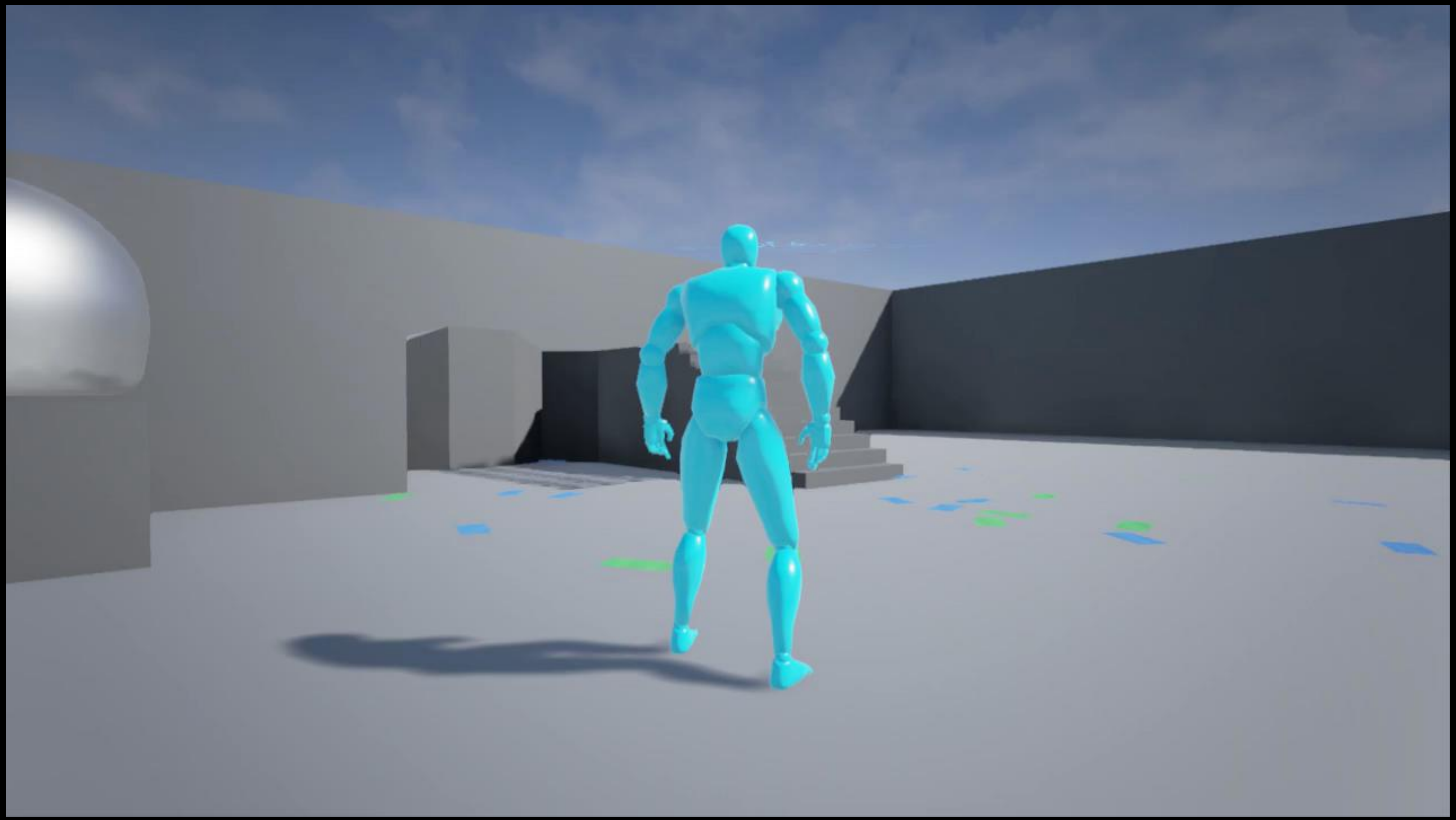
- UE3 and UE4 FleX integrations available now
- Shipping in Batman, Killing Floor
- Components for:
  - Cloth, Rigids, Inflatables, Ropes, Fluids, Particles
- Github distribution available for all UE4 registered developers:

https://github.com/NvPhysX/UnrealEngine/tree/FleX



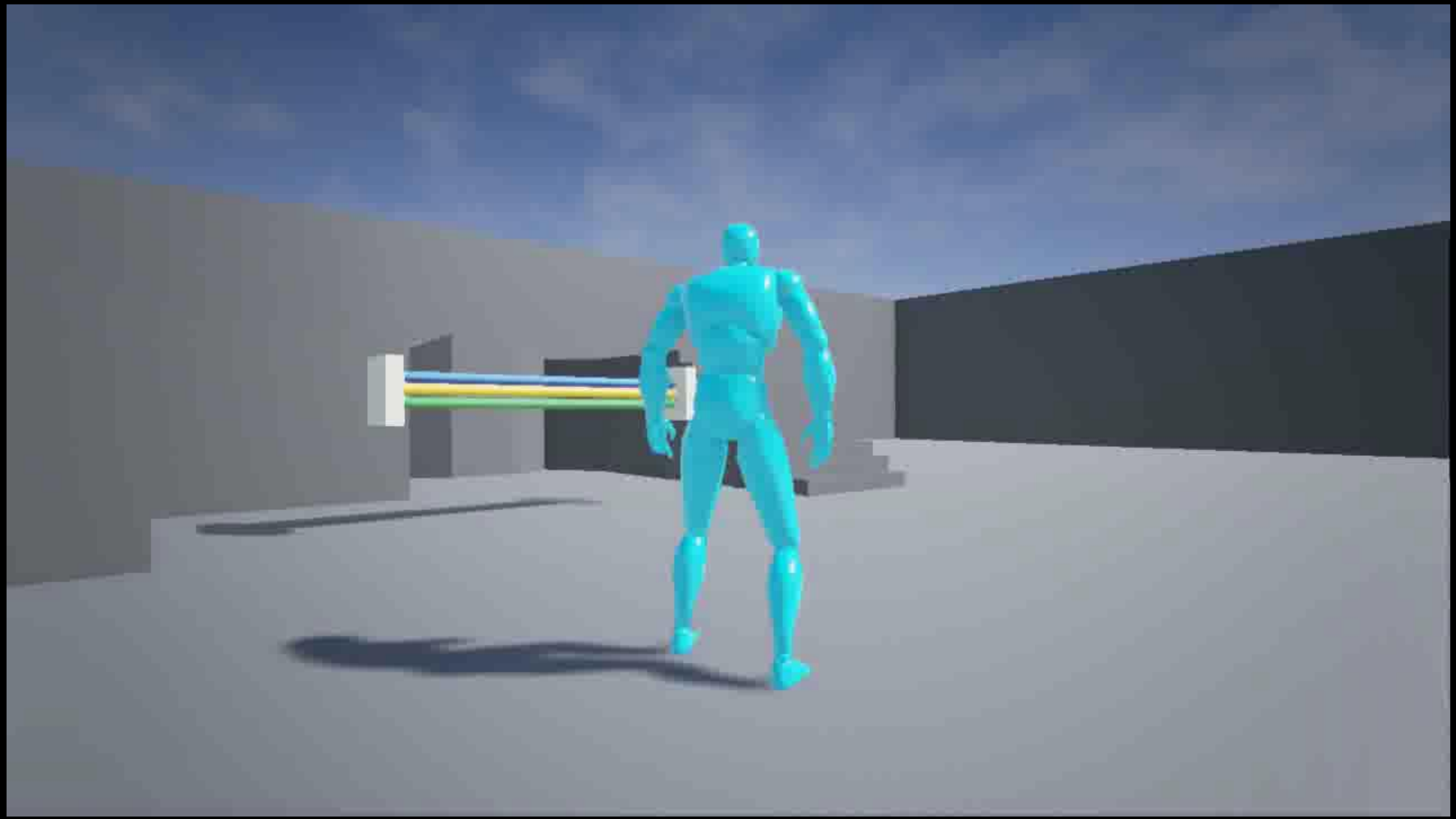
FleX Cloth

- Environmental cloth
- CCD Triangle Tests
- Auto-attachment to static or dynamic actors
- Inflatable constraints



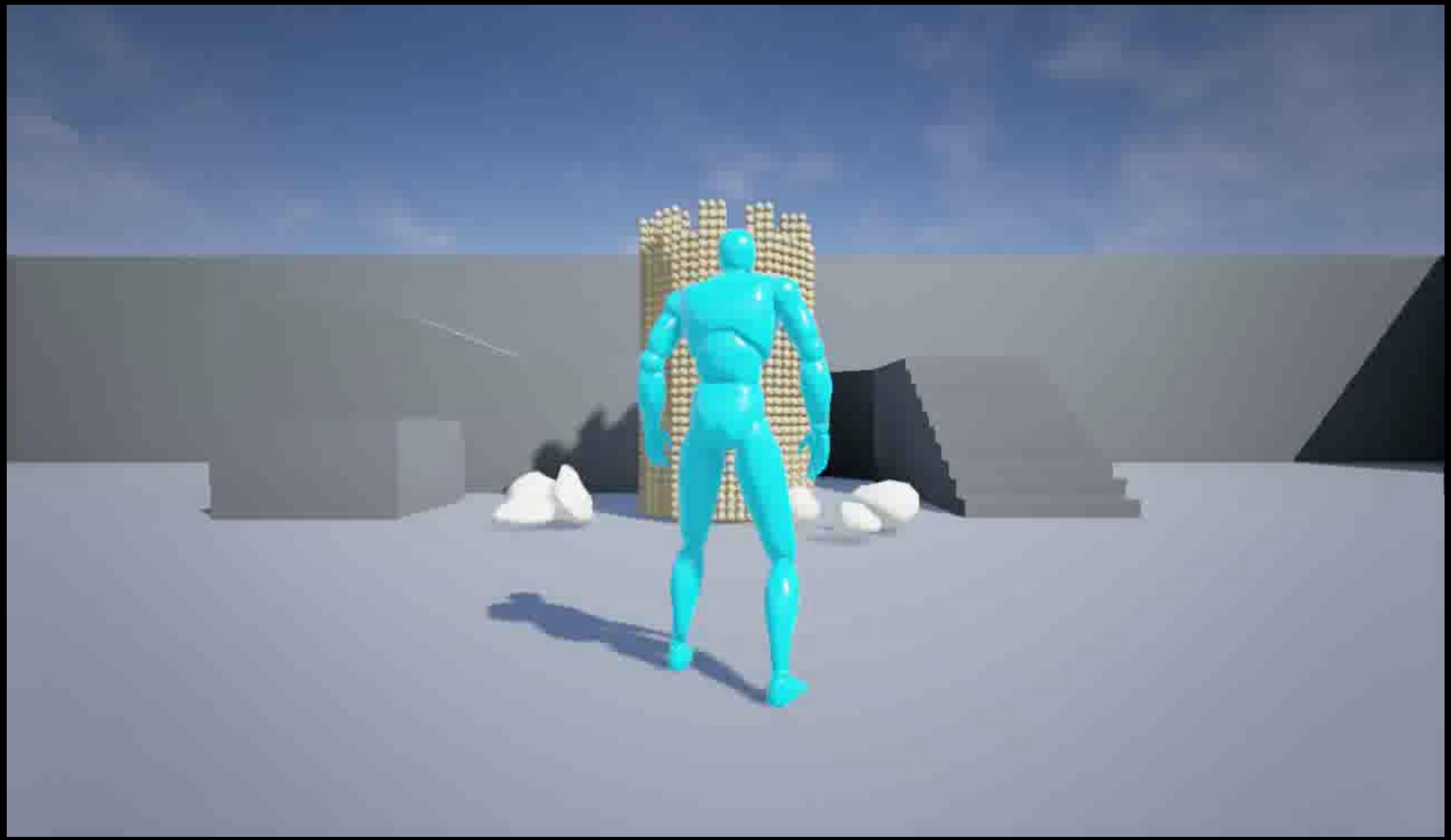
FleX Ropes

- Based on built-in UCableComponent
- Supports bending / self-collision / world collision
- Torsion in the future



FleX Particles

- Integration with Cascade
- New modules for spawning fluids
- New modules for spawning particle shapes
- Modules for spawning inflatables / cloth / etc

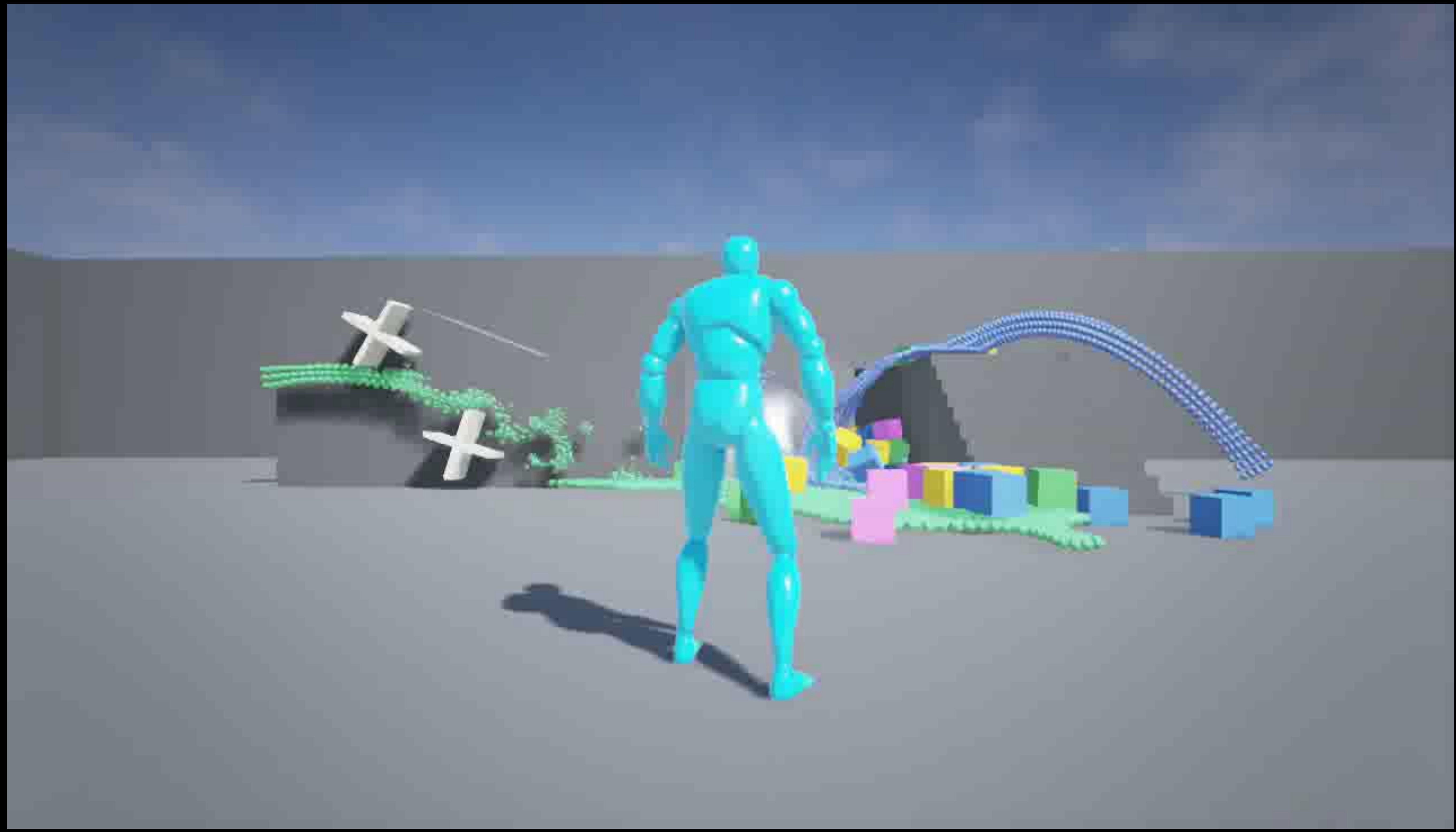


FleX Force Fields

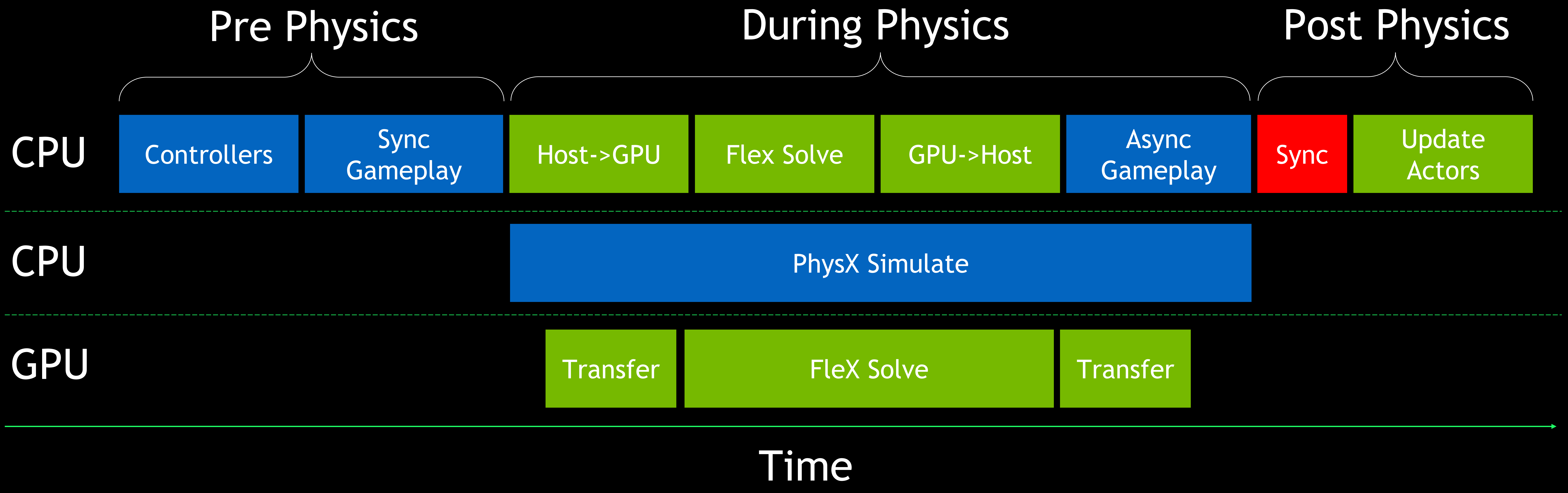
- Integration with UE4 URadialForceComponent
- Scriptable with Blueprints
- Applied in CUDA through FlexExtensions

Interop between PhysX

- Basic two-way interaction between FleX<->PhysX
- FleX actors insert bounds into PhysX scene
- Overlap query per-FleX Actor
- Allows CCT to interact with FleX objects



Frame Timeline



Game Demo:Killing Floor 2

Thank you!

Q&A