20 Particle Systems

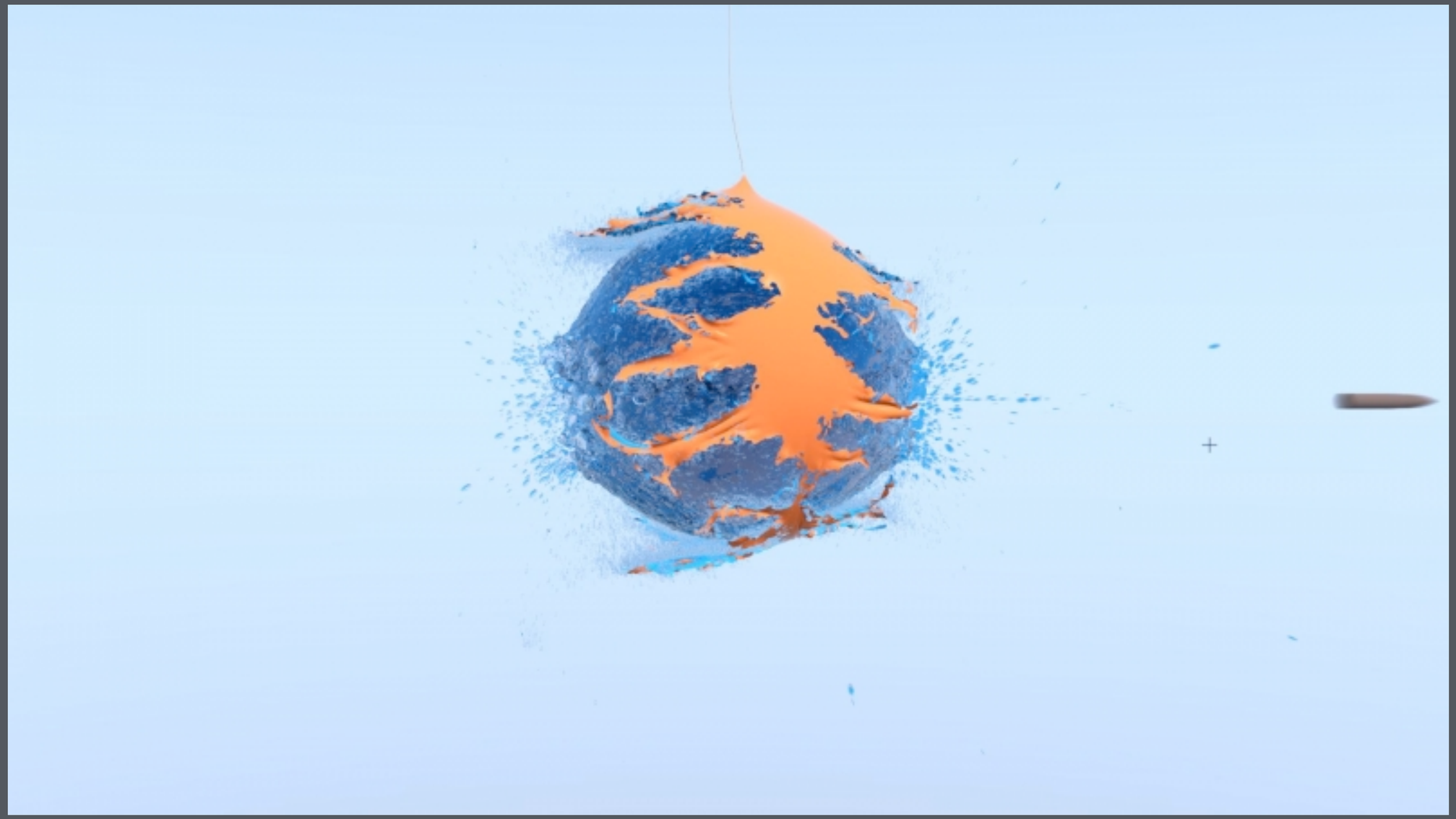
Steve Marschner
Eston Schweickart

CS4620 Spring 2017

Examples of Particle Systems

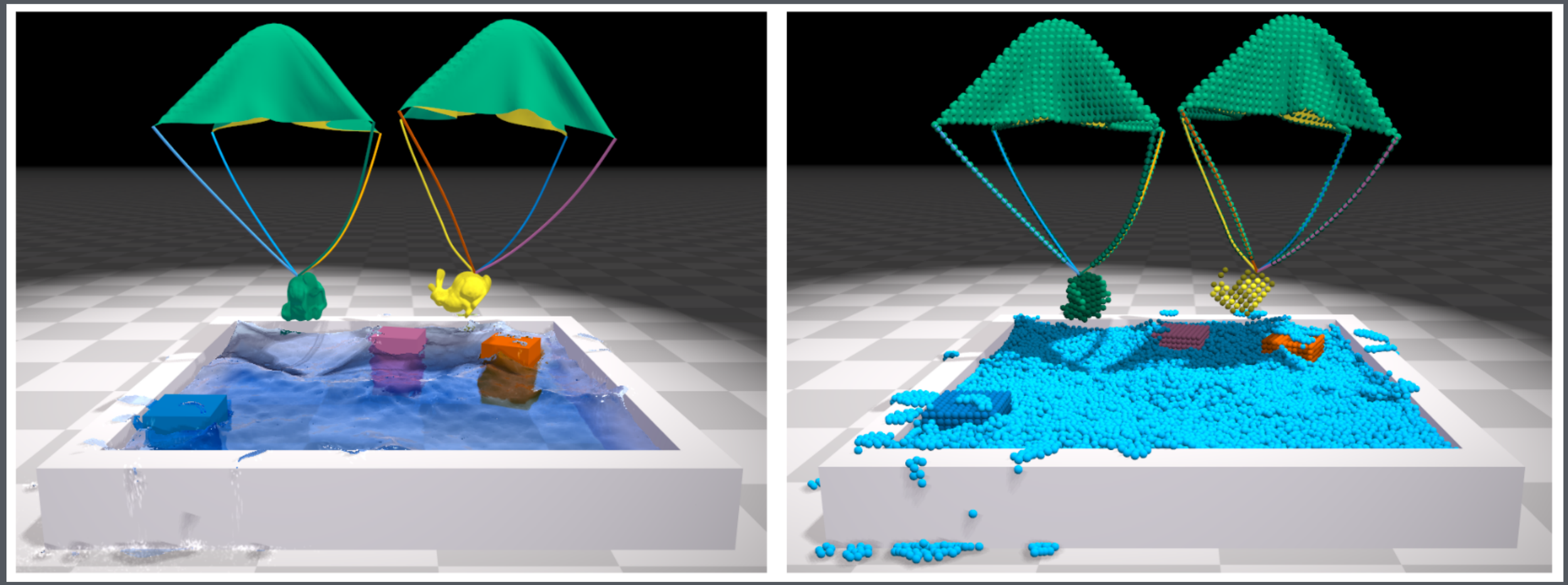
Particle Dreams [Karl Sims, 1988]

Examples of Particle Systems

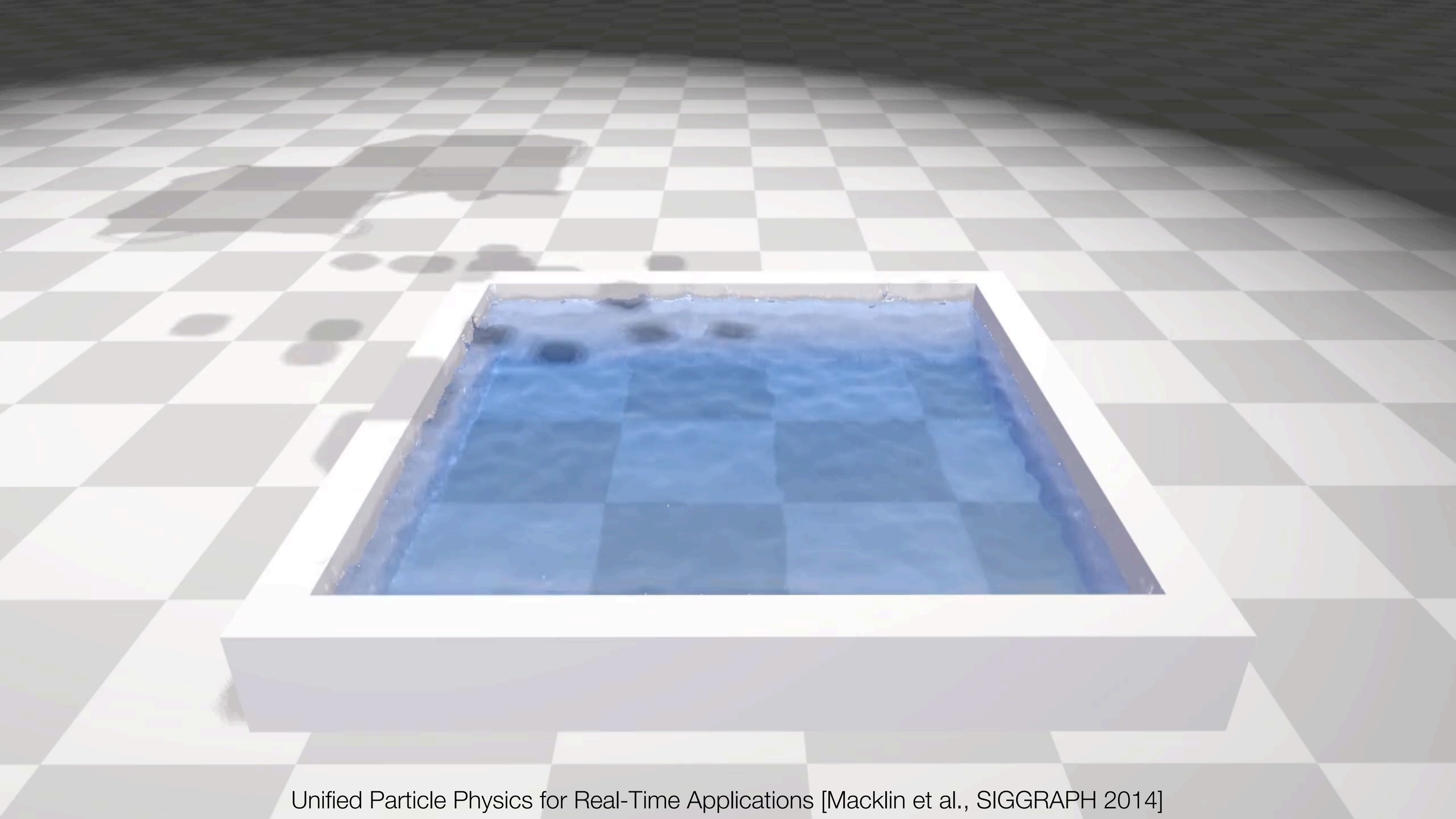


Balloon Burst [Macklin et al., SIGGRAPH 2015]

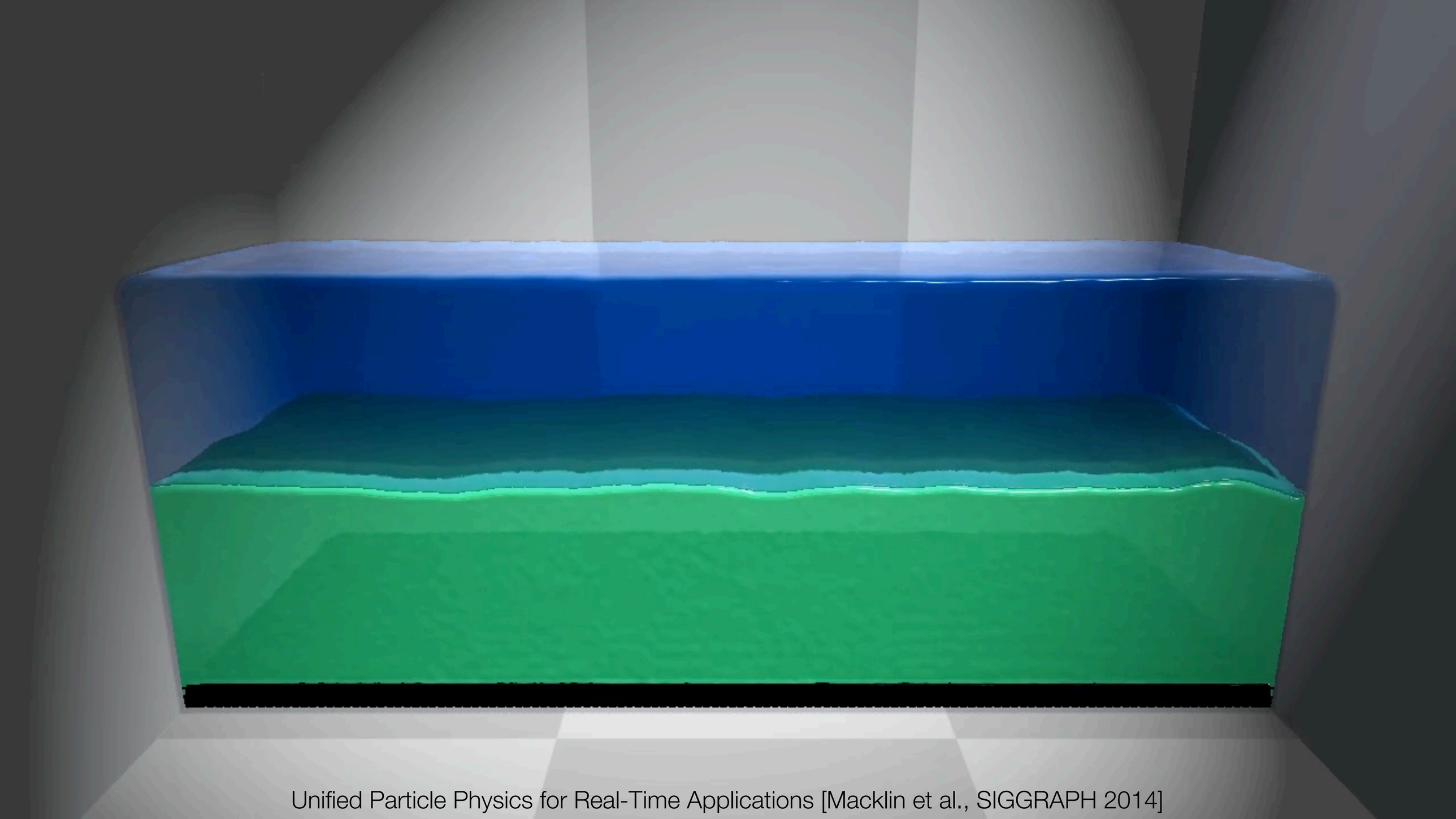
Examples of Particle Systems



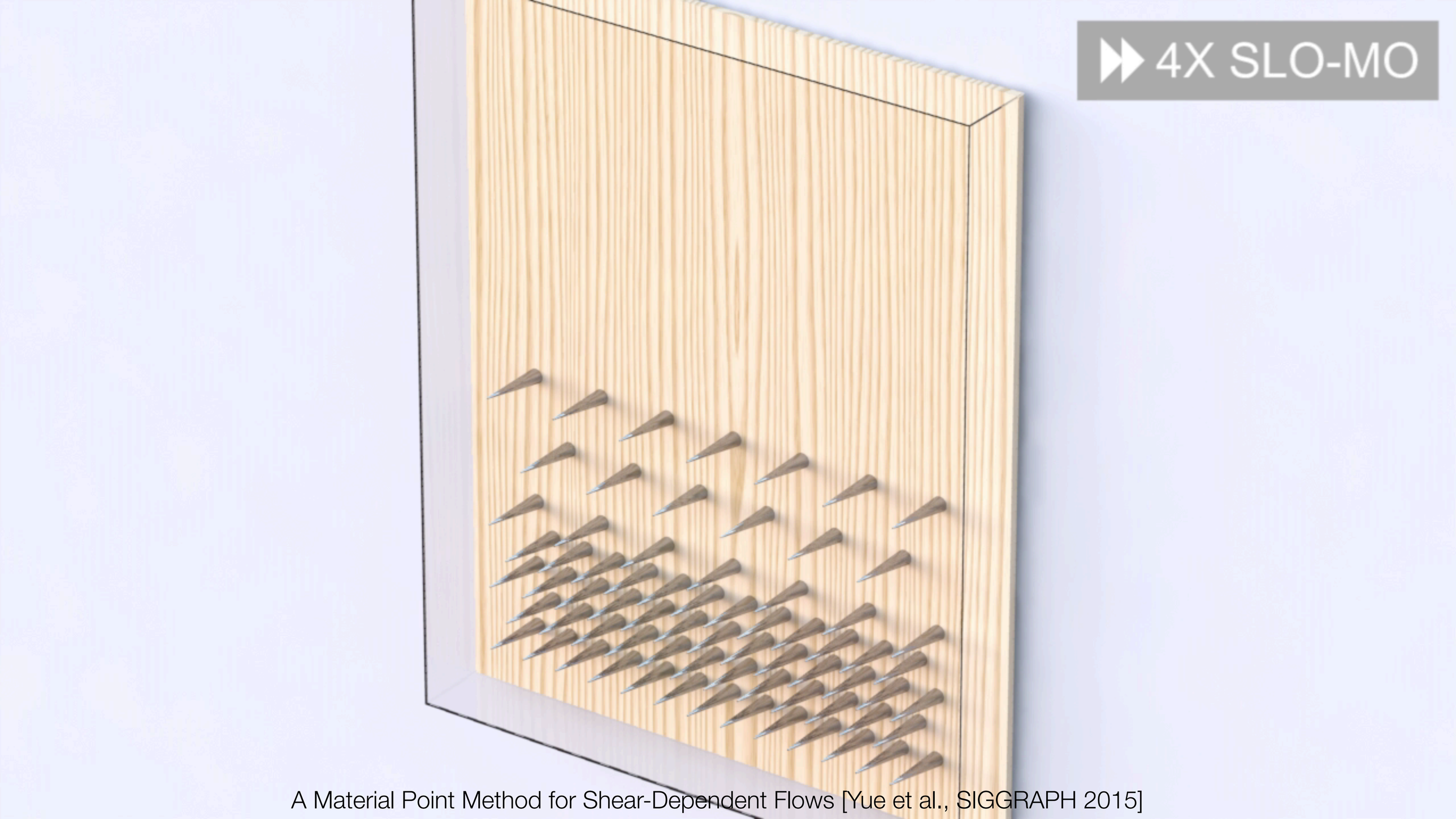
Unified Particle Physics for Real-Time Applications [Macklin et al., SIGGRAPH 2014]



Unified Particle Physics for Real-Time Applications [Macklin et al., SIGGRAPH 2014]



Unified Particle Physics for Real-Time Applications [Macklin et al., SIGGRAPH 2014]



A Material Point Method for Shear-Dependent Flows [Yue et al., SIGGRAPH 2015]

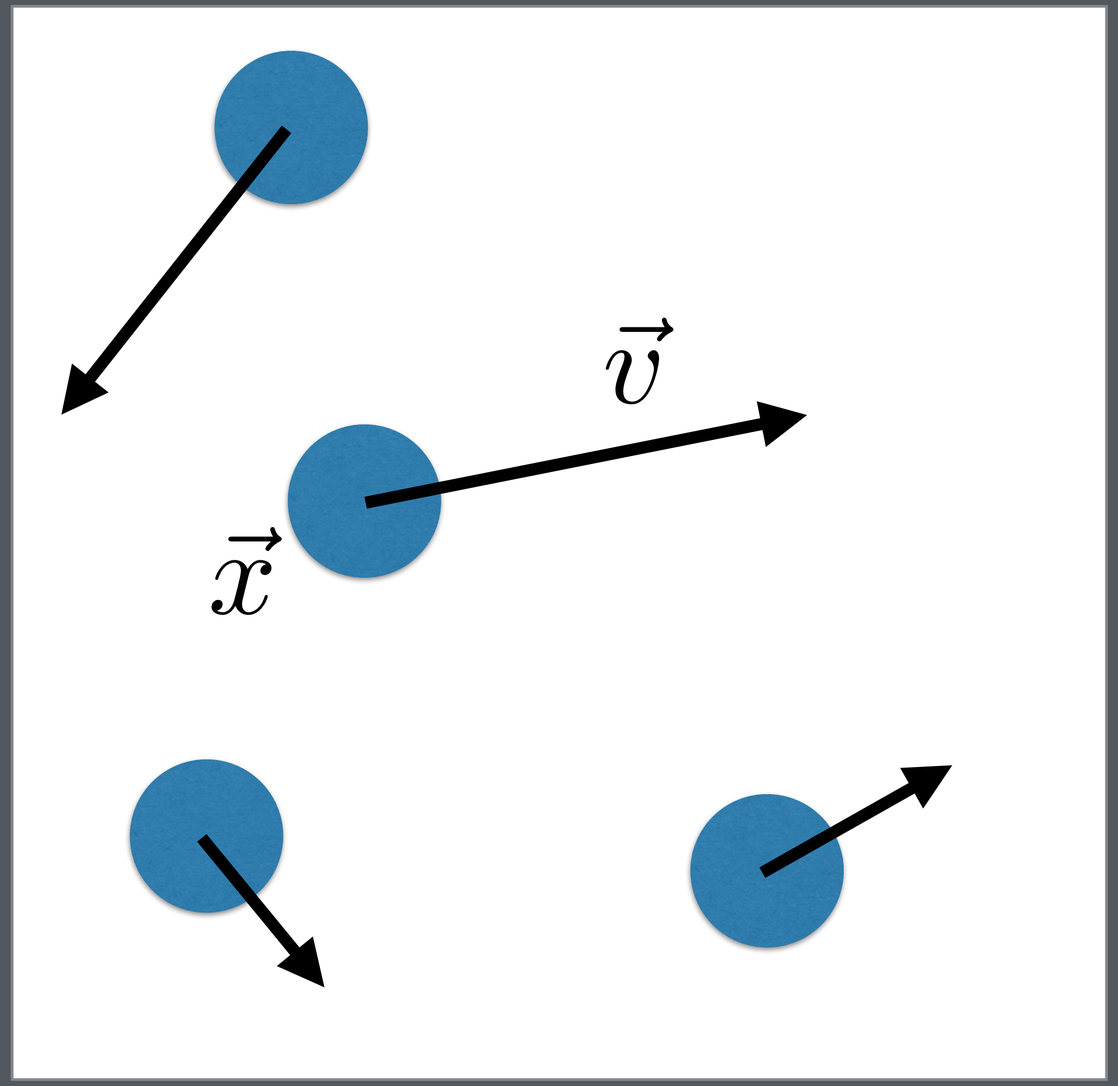
Adaptive Nonlinearity for Collisions in Complex Rod Assemblies [Kaufman et al., SIGGRAPH 2014]

Animating Elastic Rods with Sound [Schweickart et. al, to appear in SIGGRAPH 2017]

Animating Elastic Rods with Sound [Schweickart et. al, to appear in SIGGRAPH 2017]

Particle System Setup

```
class Particle {
   Vector3 position;
   Vector3 velocity;
};
```



Moving Particles

Position is a function of time

- i.e., $\vec{x} \equiv \vec{x}(t)$
- Note that $\vec{v}(t) \equiv \dfrac{\partial \vec{x}}{\partial t}$

Use a function to control the particle's velocity

- $\vec{v}(t) = f(\vec{x}(t))$

This is an Ordinary Differential Equation (ODE)

Solve this ODE at every frame

- i.e., solve for $\vec{x}(t_0), \vec{x}(t_1), \vec{x}(t_2), \ldots$
- Then we can draw each of these positions to the screen

A Simple Example

Let $\vec{v}$ be constant

- e.g., $\vec{v} = f(\vec{x}) = (0, 0, 1)^\top$

Then we can solve for the position at any time:

- $\vec{x}(t) = \vec{x}(0) + t\vec{v}$

Not always so easy

- $f(\vec{x})$ can be anything!
- Might be unknown until runtime (e.g., user interaction)
- Often times, not solved exactly

Moving Particles, Revisited

Now, acceleration is in the mix

- $\vec{a}(t) \equiv \frac{\partial \vec{v}}{\partial t} \equiv \frac{\partial^2 \vec{x}}{\partial t^2}$

Use a function to control the particle's acceleration

- $\vec{a}(t) = f(\vec{x}(t))$

This is a Second Order ODE

Solve this ODE at every frame, same as before

- Can sometimes be reduced to a first order ODE
- Calculate position and velocity together

Physically-based Motion

Acceleration based on Newton's laws

- $\vec{f}(t) = m\vec{a}(t)$ …or, equivalently… $\vec{a}(t) = \vec{f}(t)/m$
- i.e., force is mass times acceleration

Forces are known beforehand

- e.g., gravity, springs, others….
- Multiple forces sum together
- These often depend on the position, i.e., $\vec{f}(t) \equiv \vec{f}(\vec{x}(t))$
- Sometimes velocity, too

If we know the values of the forces, we can solve for particle's state

Unary Forces

Constant

- Gravity

Position/Time-Dependent

- Force fields, e.g. wind

Velocity-Dependent

- Drag

Binary, *n*-ary Forces

Much more interesting behaviors to be had from particles that interact

Simplest: binary forces, e.g. springs

$$\vec{f}_i(\vec{x}_i, \vec{x}_j) = -k_s(\|\vec{x}_i - \vec{x}_j\| - r_{ij}) \frac{\vec{x}_i - \vec{x}_j}{\|\vec{x}_i - \vec{x}_j\|}$$

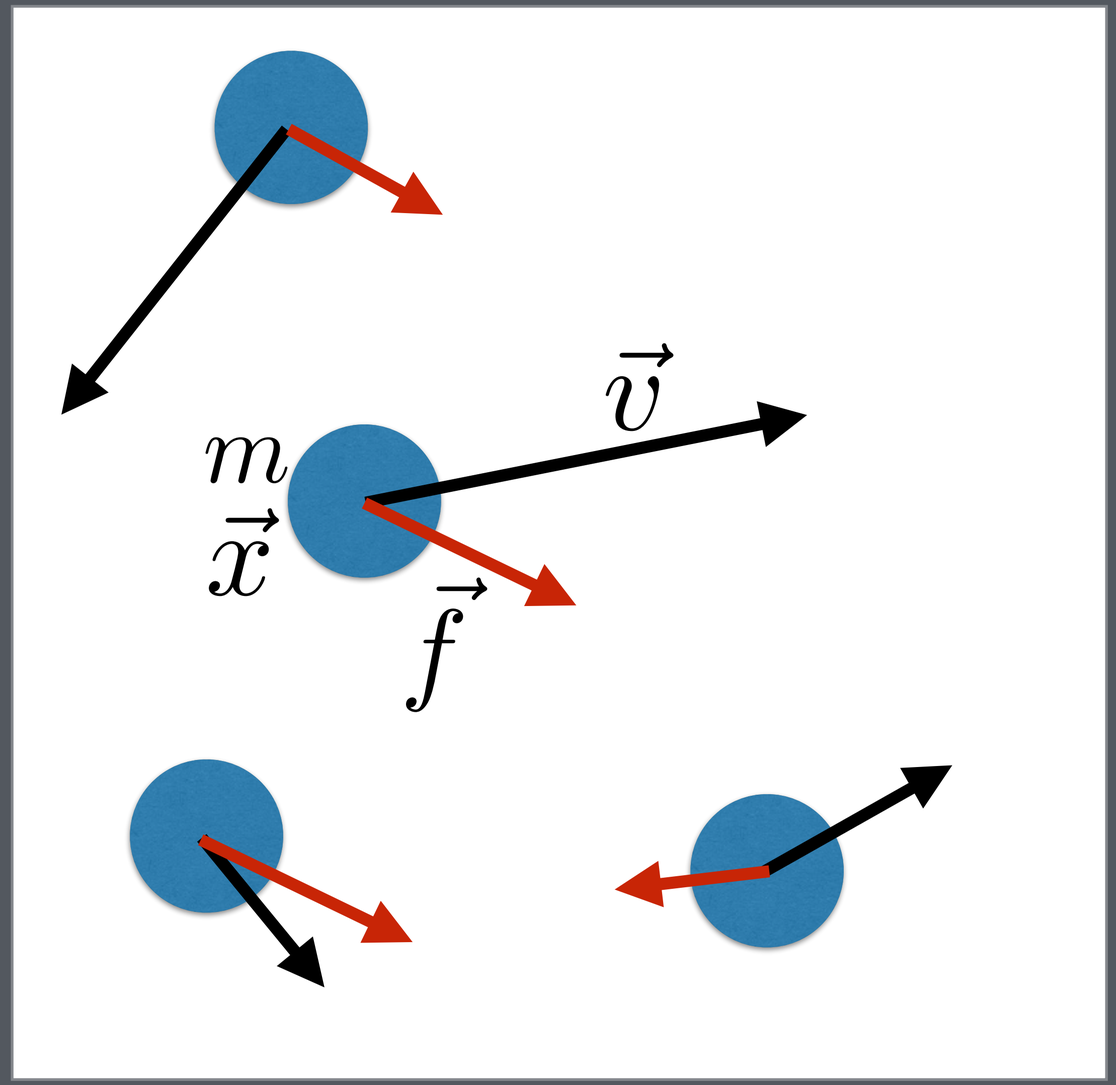
Nice example project with mass-spring systems:

- https://vimeo.com/73188339

More sophisticated models for deformable things use forces relating 3 or more particles

Particle System Setup, Revisited

```
class Particle {
   float mass;
   Vector3 position;
   Vector3 velocity;
   Vector3 force;
};
```



Basic Algorithm

1) Clear forces from previous calculations

2) Calculate/accumulate forces for each particle

3) Solve for particle's state (position, velocity) for the next time step h

Integration Algorithm 1

Calculating Particle State from Forces: First attempt

- Use forces to update velocity: $\vec{v}(t+h) = \vec{v}(t) + \frac{h}{m}\vec{f}(t)$
- Use old velocity to update position: $\vec{x}(t+h) = \vec{x}(t) + h\vec{v}(t)$

Issues

- Unstable in certain cases!
- Reducing time step can help, but this becomes computationally expensive
- Error is $O(h^2)$ per step (and accumulates!). Error is $O(h)$ globally.

This technique is called Forward (Explicit) Euler Integration

Example: circle

Integration Algorithm 2

Another attempt

- Update velocity with forces at next time step: $\vec{v}(t+h) = \vec{v}(t) + \frac{h}{m}\vec{f}(t+h)$
- Use new velocity to update position: $\vec{x}(t+h) = \vec{x}(t) + h\vec{v}(t+h)$

Benefits

- Unconditionally stable if the system is linear!

Issues

- Solving for $\vec{f}(t+h)$ is often expensive
- Can introduce artificial viscous damping
- Error is still $O(h^2)$ per step

This technique is called Backward (Implicit) Euler Integration

Integration Algorithm 3

Next attempt: A compromise

- Update velocity using current forces: $\vec{v}(t+h) = \vec{v}(t) + \frac{h}{m}\vec{f}(t)$
- Use updated velocity to update the position: $\vec{x}(t+h) = \vec{x}(t) + h\vec{v}(t+h)$

Benefits

- All the speed benefits of Forward Euler, but much more stable!
- *Symplectic* implies that it conserves energy much better

Issues

- Not unconditionally stable
- Error still $O(h^2)$ per step

This technique is called Symplectic (Semi-implicit) Euler Integration

Other Integration techniques

Many more complicated schemes

- Newmark-β
- Verlet
- RK-4
- Exponential integrators

Reduce error per step

Increase stability

Caveat: generally more expensive than Symplectic Euler

- Some are approximately as efficient as Implicit Euler

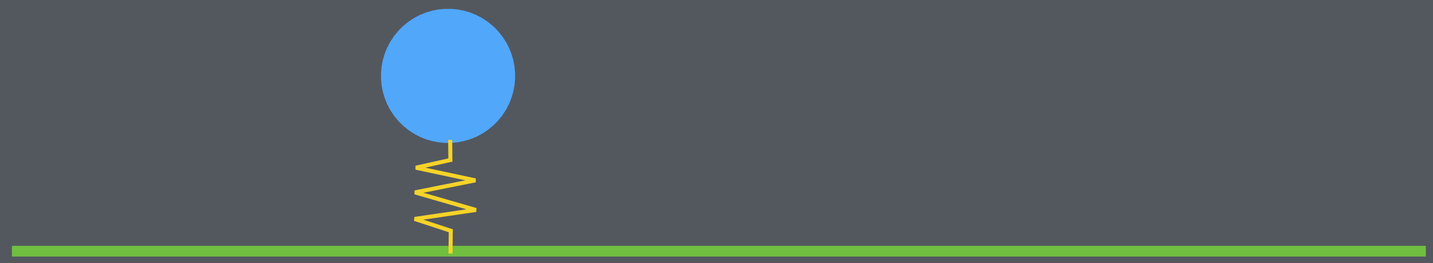
Computational Stiffness

E.g. Bead on Wire

- Can use a spring force to bind bead (particle) to a wire
- If the spring is weak, the particle may drift too far away
- If the spring is strong, we need very small time steps to ensure stability

Known as a “stiff” problem

- One stiff spring makes the whole system stiff!



Constraints

At the end of each step (i.e. after integration), enforce certain properties of the system

- e.g., the bead should not leave the wire

Idea: push unconstrained system towards acceptable configuration by modifying particle momentum as little as possible

Constraint Equations

Usually of the form $C(x) = 0$ or $C(x) \geq 0$

When finding a solution, we are usually interested in the derivatives of these equations with respect to position (x)

These are similar to forces, but are non-physical

Enforcing Constraints

First attempt: Apply constraint equation derivatives iteratively

Benefits

- Fast, parallelizable over particles

Issues

- Constraint application order matters!
- Convergence not guaranteed!
- Successive Over-Relaxation can help (i.e., apply a scaled version of the constraint derivative)
    - But this is finicky, finding the right scaling value can be difficult (or it might not exist)

Enforcing Constraints

Another attempt: Lagrange Multipliers

- Solve a global linear system over all constraints
- Add an extra row/column for each 1D constraint

Benefits

- Order of constraints doesn’t matter
- Solves simultaneous constraints exactly in one pass

Issues

- Non-parallelizable, global linear solve (but this can be done quickly using, e.g., conjugate gradient)
- Makes approximations for nonlinear constraints, which are fairly common in practice

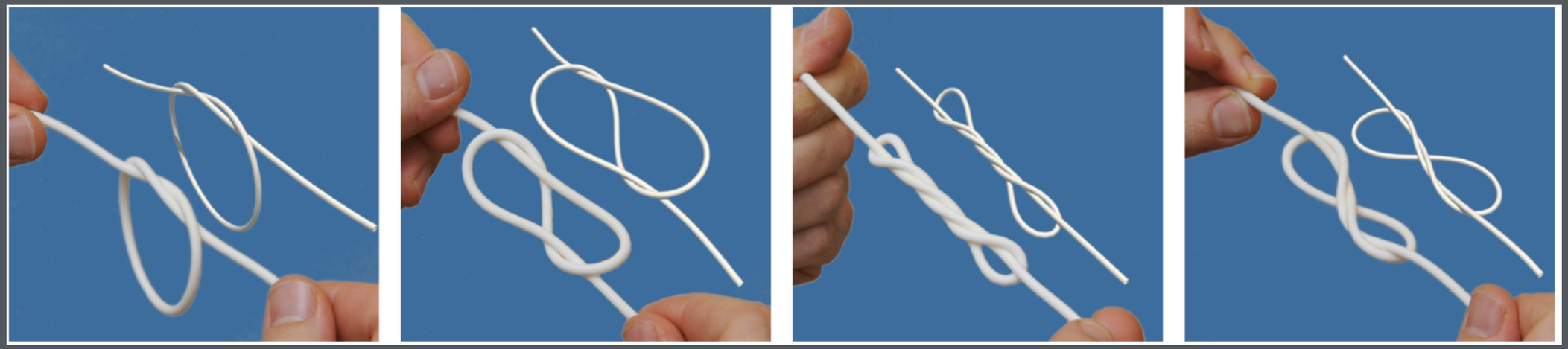
The New Algorithm

1) **Clear forces from previous calculations**
2) **Calculate/accumulate forces for each particle**
3) **Use time integration algorithm of choice to update particle to unconstrained position**
4) **Enforce constraints with algorithm of choice**

Examples of Forces/Constraints

Rods

- Bending (Force)
- Stretching (Force/Constraint)
- Twisting (Force/Constraint)

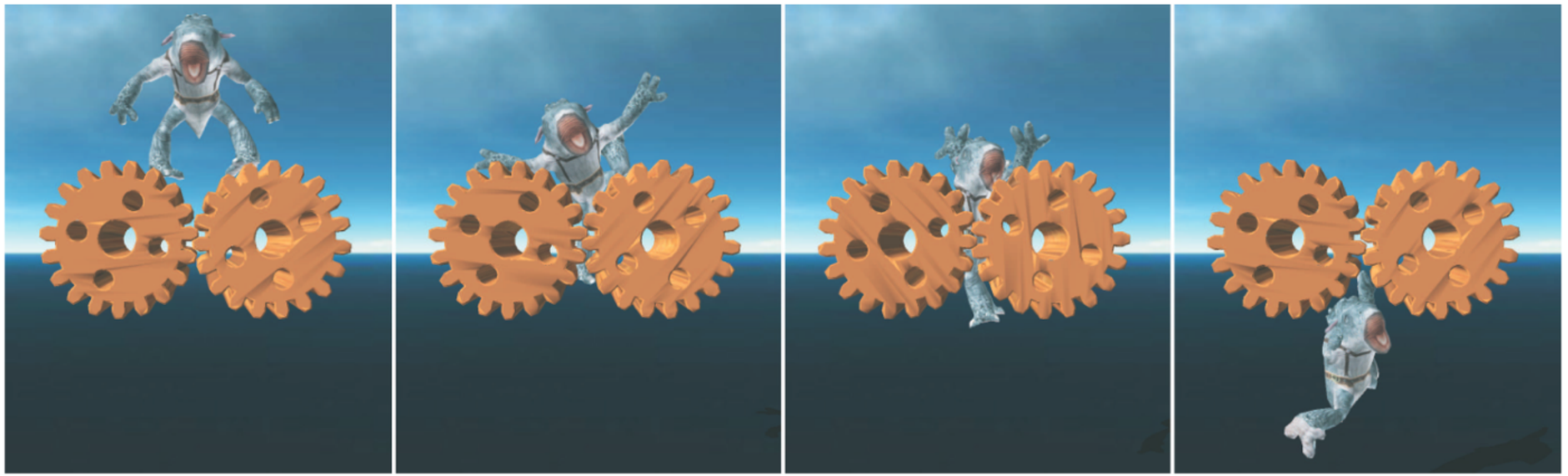


Discrete Elastic Rods [Bergou et al., 2008]

Examples of Force/Constraints

Cloth

- Bending (Force/Constraint)
- Stretching (Force/Constraint)
- Shearing (Force)
- Pressure, e.g. for a balloon (Constraint)

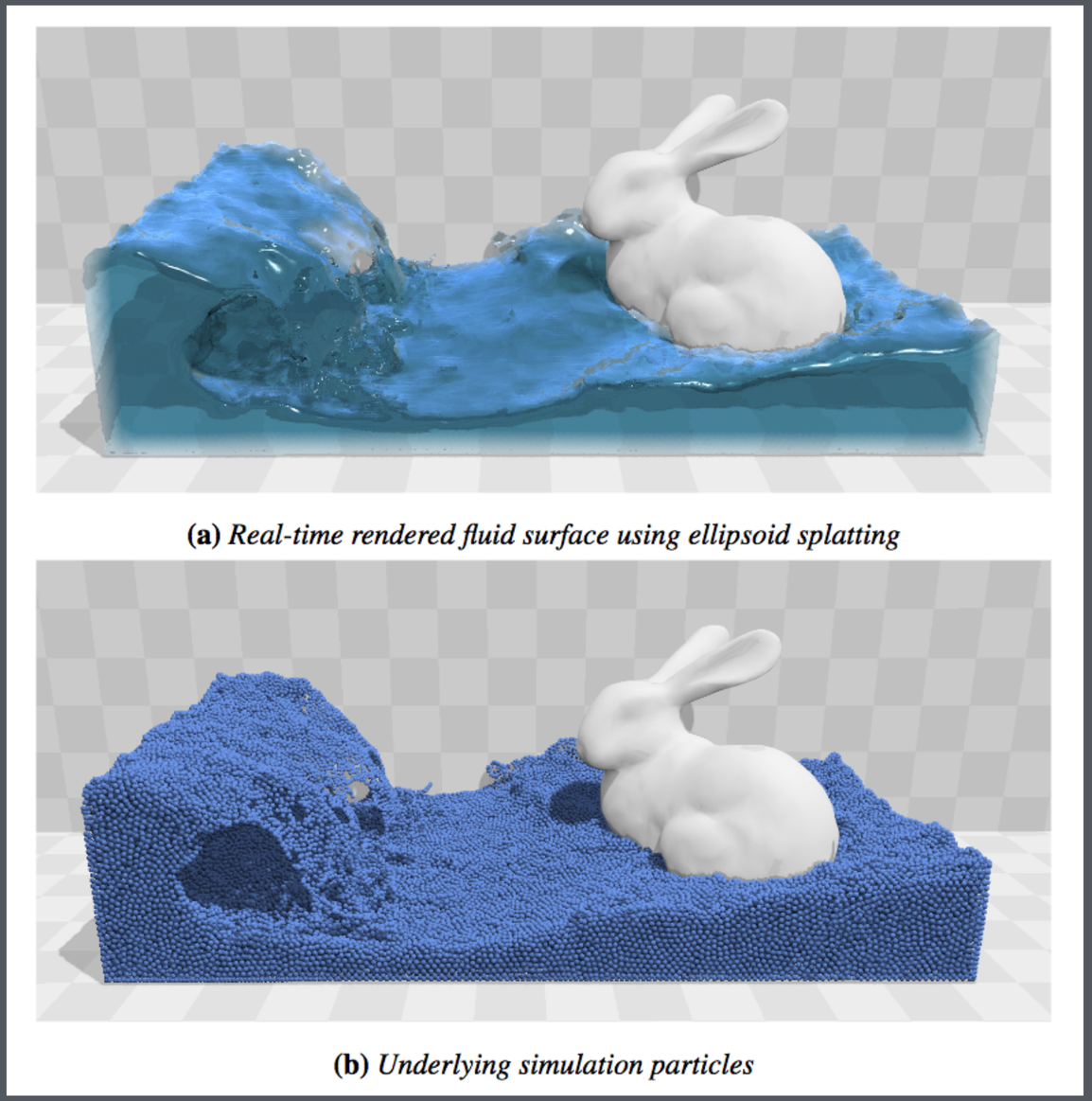


Position Based Dynamics [Müller et al., 2006]

Examples of Force/Constraints

Fluid

- Incompressibility (Constraint)
- Surface Tension (Constraint-ish)
- Vorticity (Force)



(a) *Real-time rendered fluid surface using ellipsoid splatting*

(b) *Underlying simulation particles*

Position Based Fluids [Macklin and Müller, SIGGRAPH 2012]