Constraint fluids in Sprinkle

Dennis Gustafsson
Mediocre

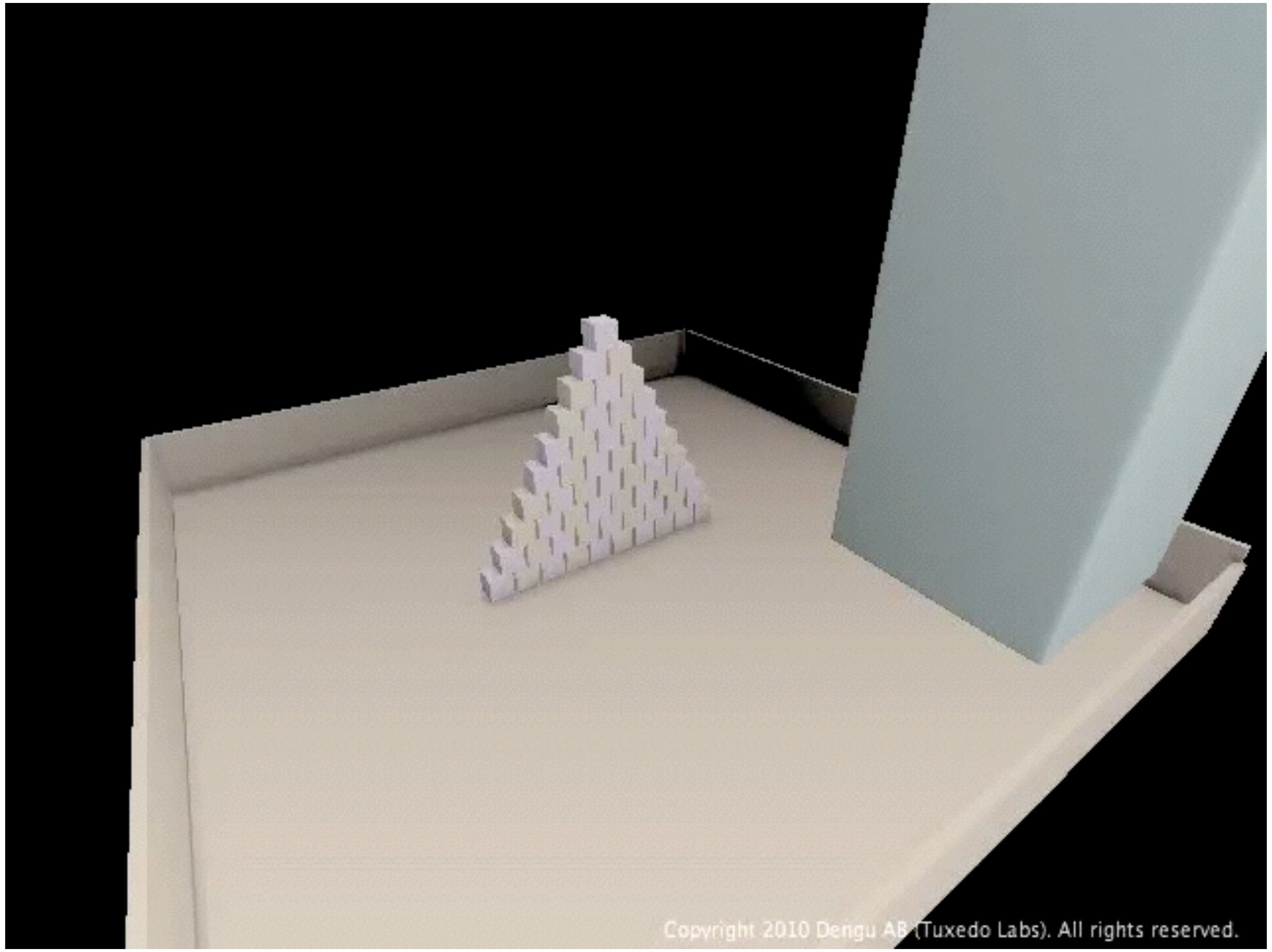
GAME DEVELOPERS CONFERENCE®
SAN FRANCISCO, CA
MARCH 17–21, 2014
EXPO DATES: MARCH 19–21
2014

Presentation outline

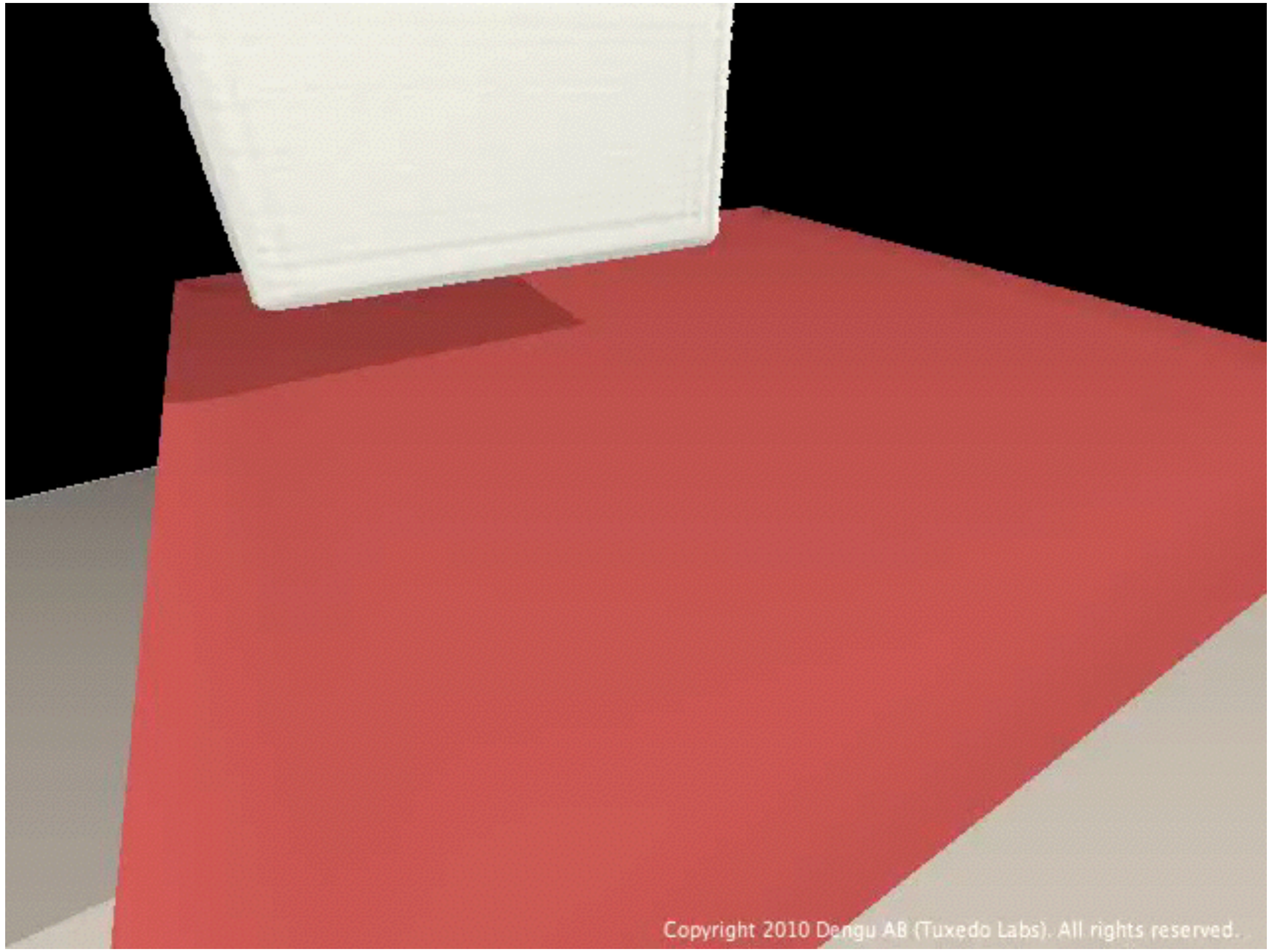
- Background and motivation
- Simulating fluid with constraints
- Implementation in Sprinkle
- Rendering and performance consideration

Experiments in 3D

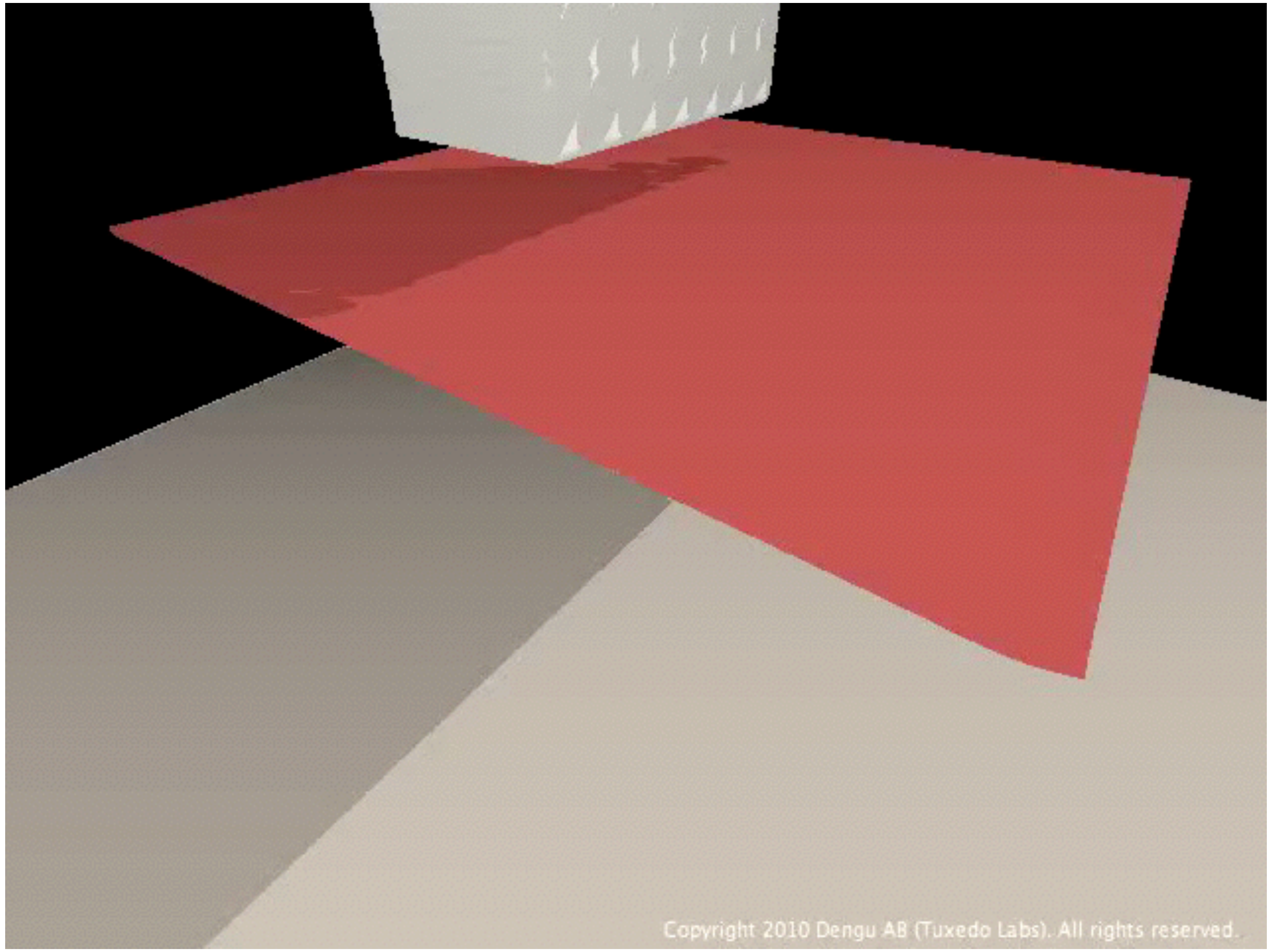
Experiments in 3D



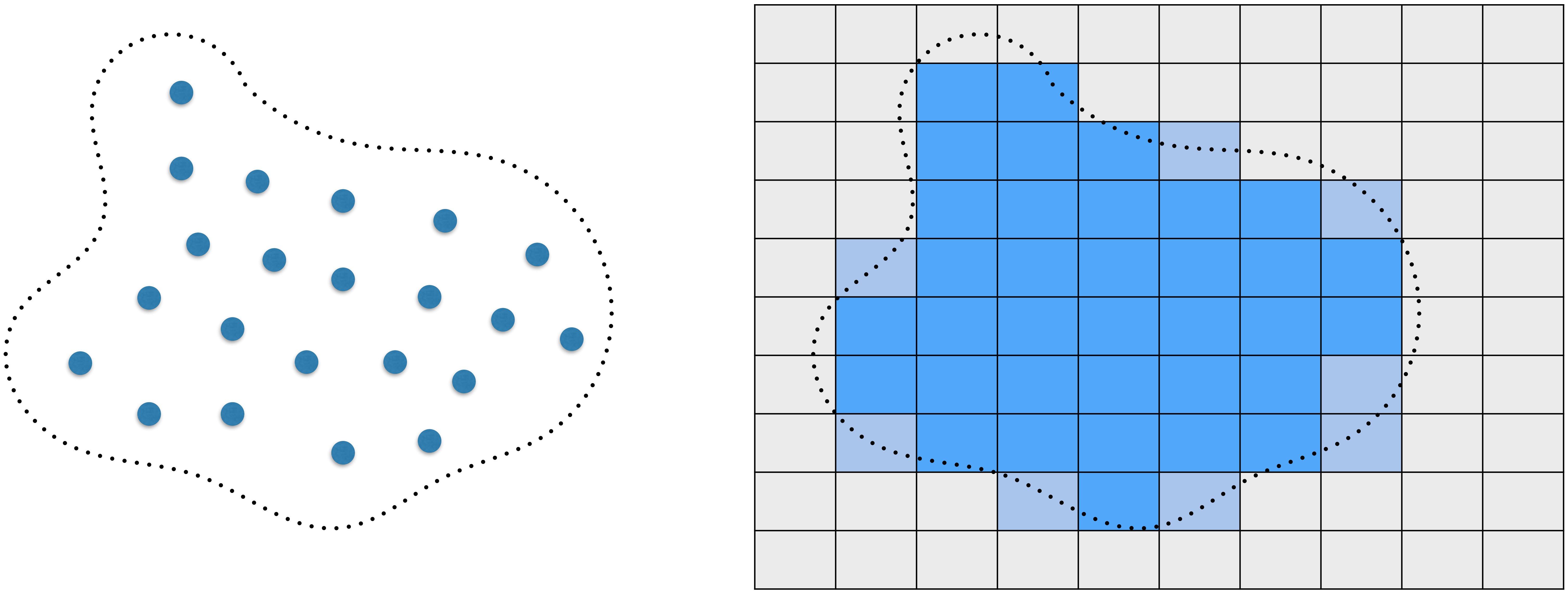
Experiments in 3D



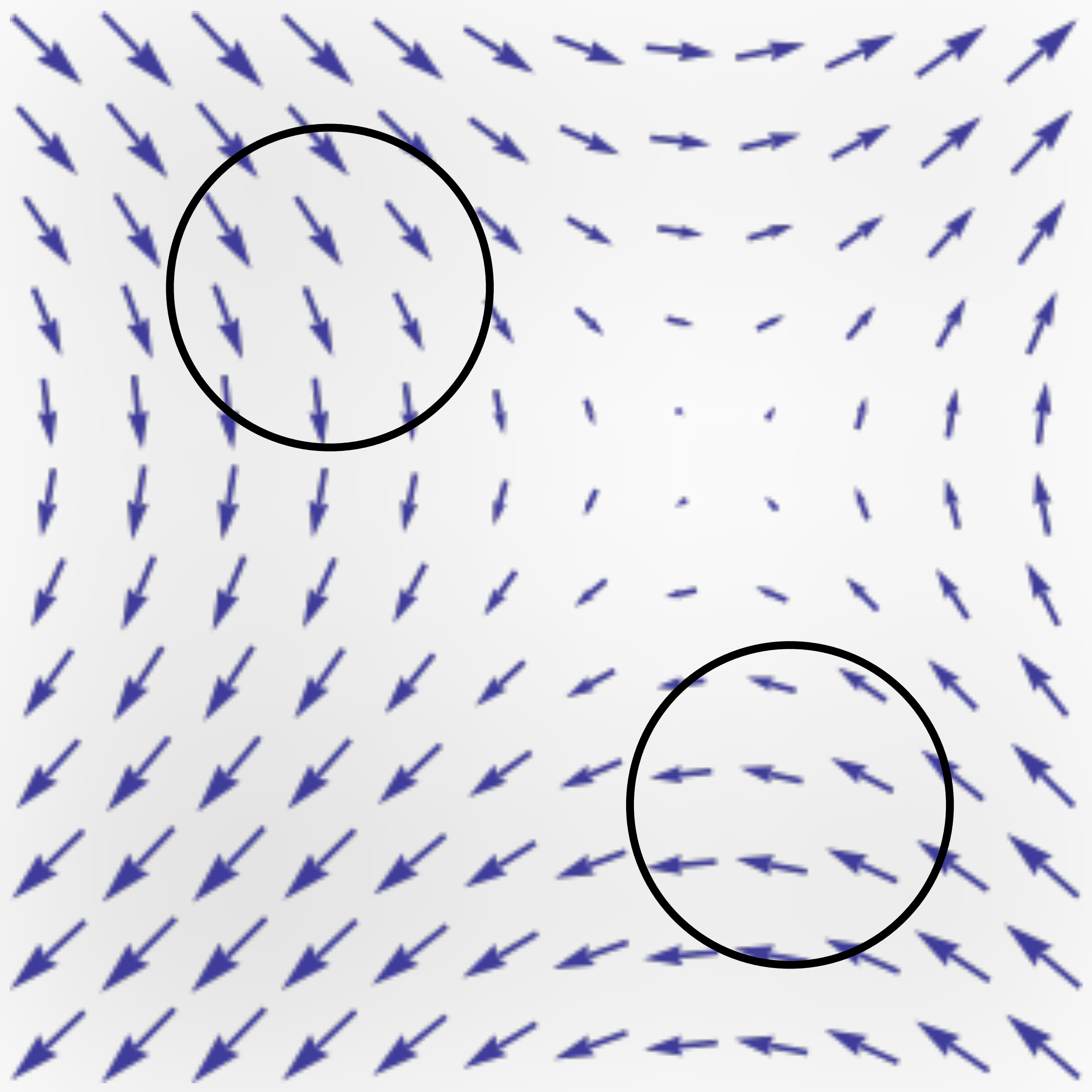
Experiments in 3D



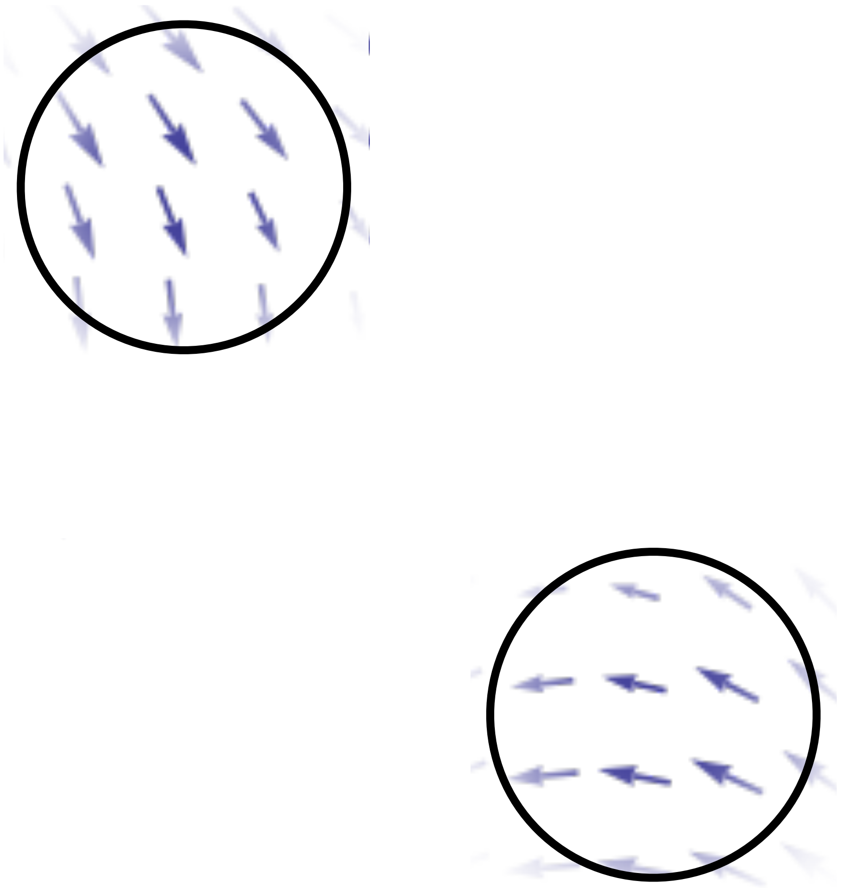
Simulation paradigm



SPH particle ≠ droplet



SPH particle ≠ droplet



Traditional SPH solver recap

1. Find particle neighbors
2. Compute density at each particle
3. Compute and apply pairwise interaction forces
4. Integrate forces to new particle positions

Rigid body solver recap

1. Find body neighbors
2. Setup velocity constraints
3. Sequential impulse solver: Apply impulses at contacts until all are separating*
4. Integrate velocities to new position and orientation

Conceptual differences

- SPH: Particle positions affect interaction forces. Forces are integrated.

- Rigid body: Velocity of other bodies affect impulses. Velocity is integrated.

Expriment

What would a rigid body simulator look like if implemented the same way as SPH?

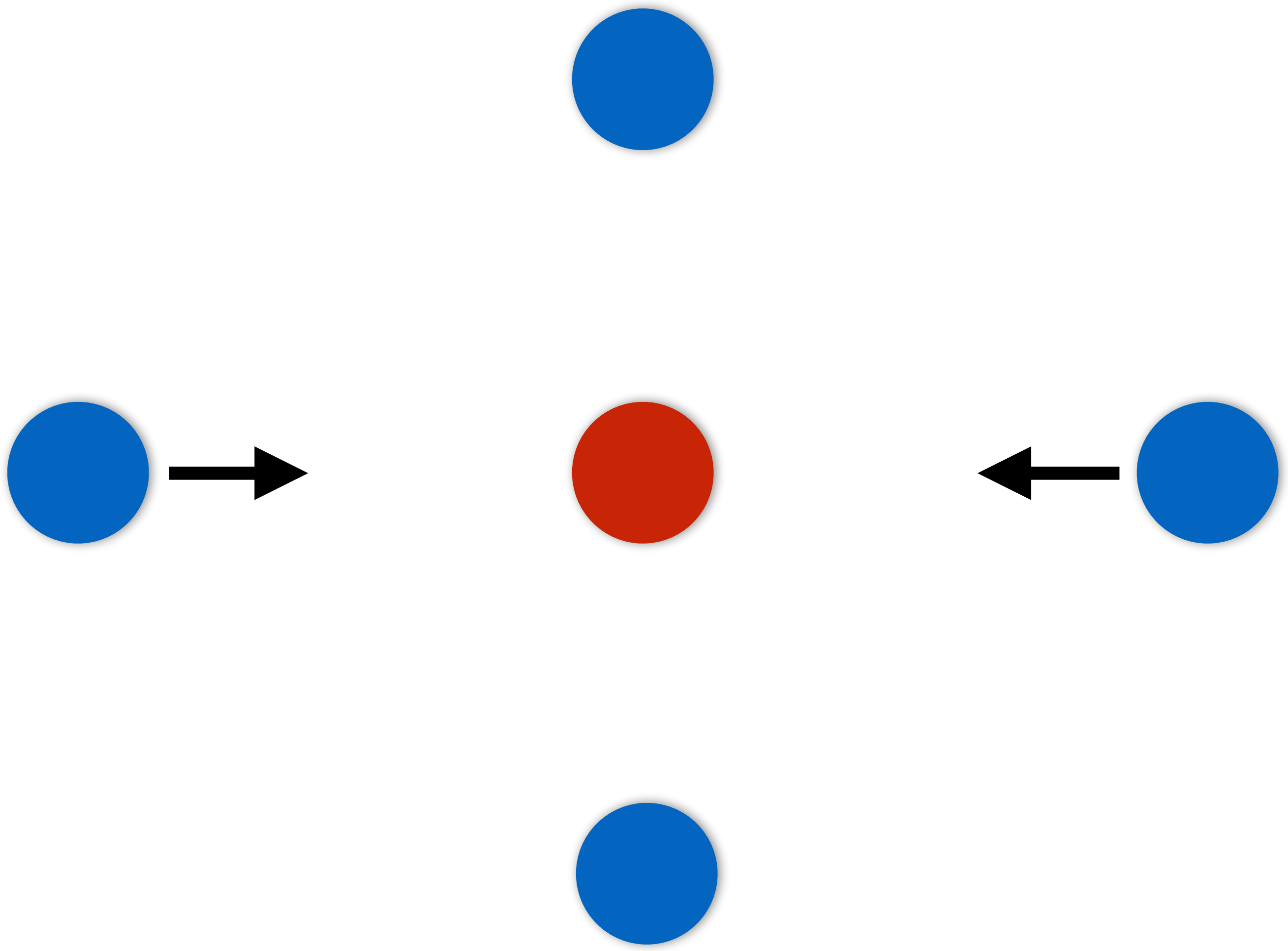
Penalty method

- Springy behavior
- Rigid bodies are supposed to be **rigid**

- Liquids are **not** rigid
- …but liquids are **incompressible**

How can we model fluid motion as a velocity constraint?

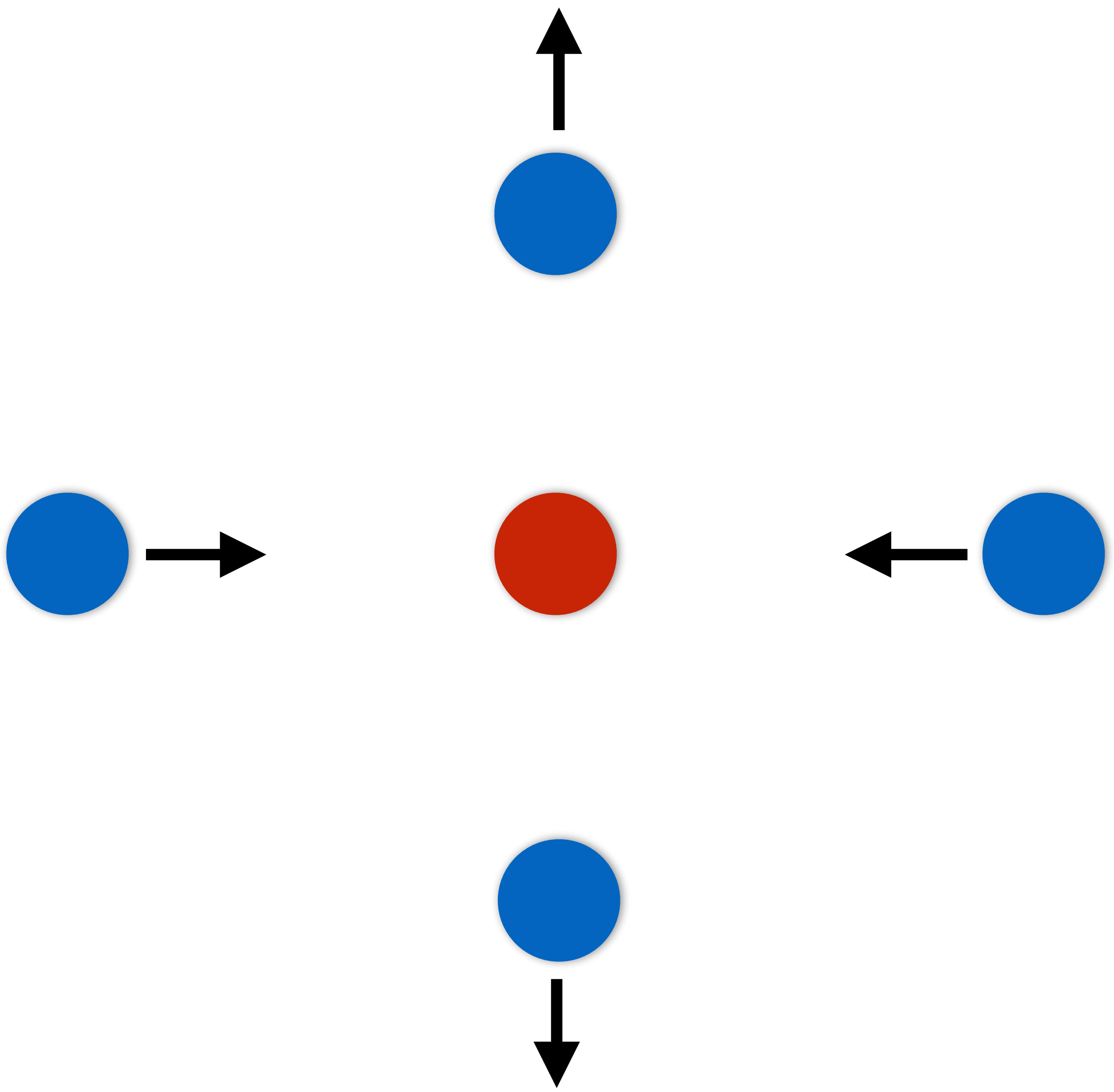
Pair-wise interaction is not enough



A fluid particle is not a point mass.

It’s a discretization of a field.

Fluid can flow through the particle

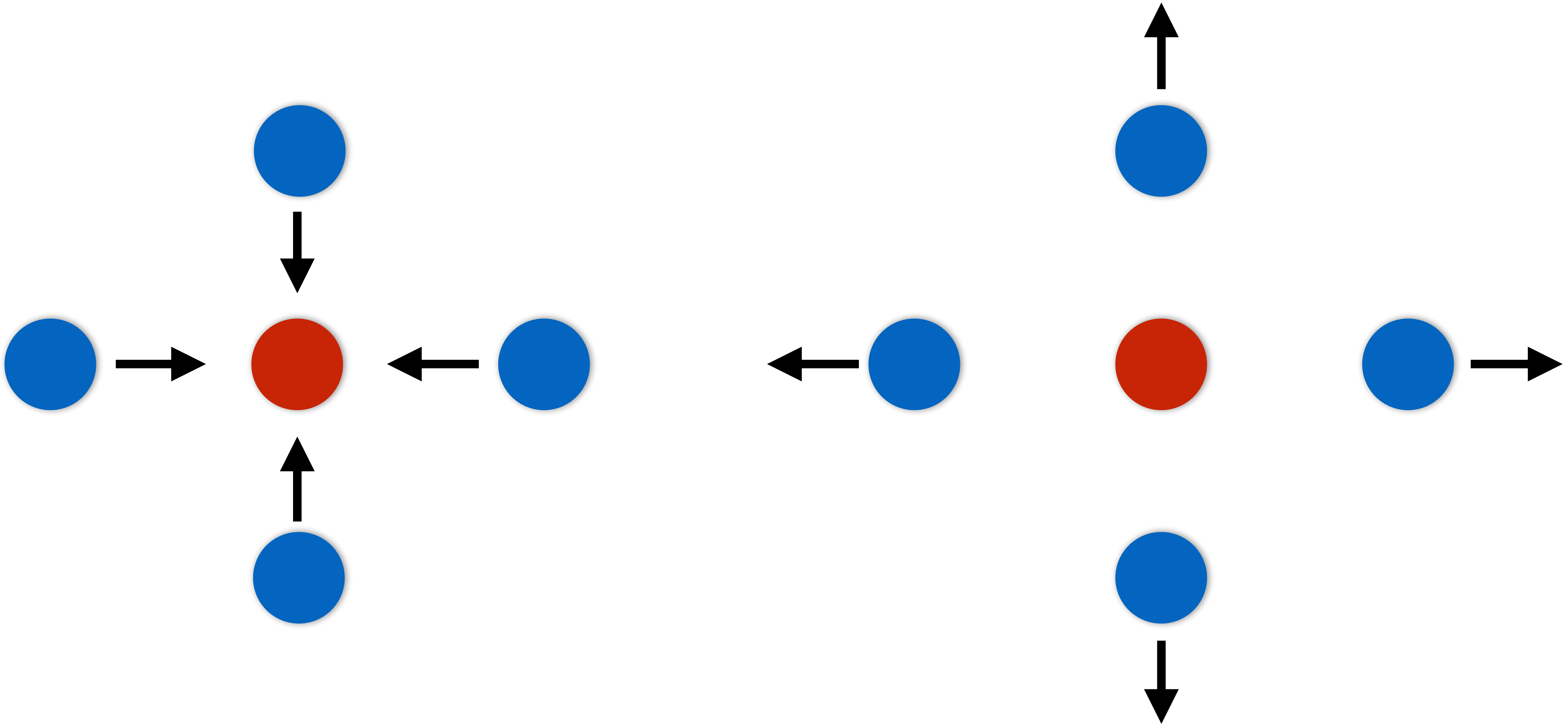


Incompressible flow

- Fluid can flow through a particle, but the density should remain constant.

$$\frac{D\rho}{Dt} = 0$$

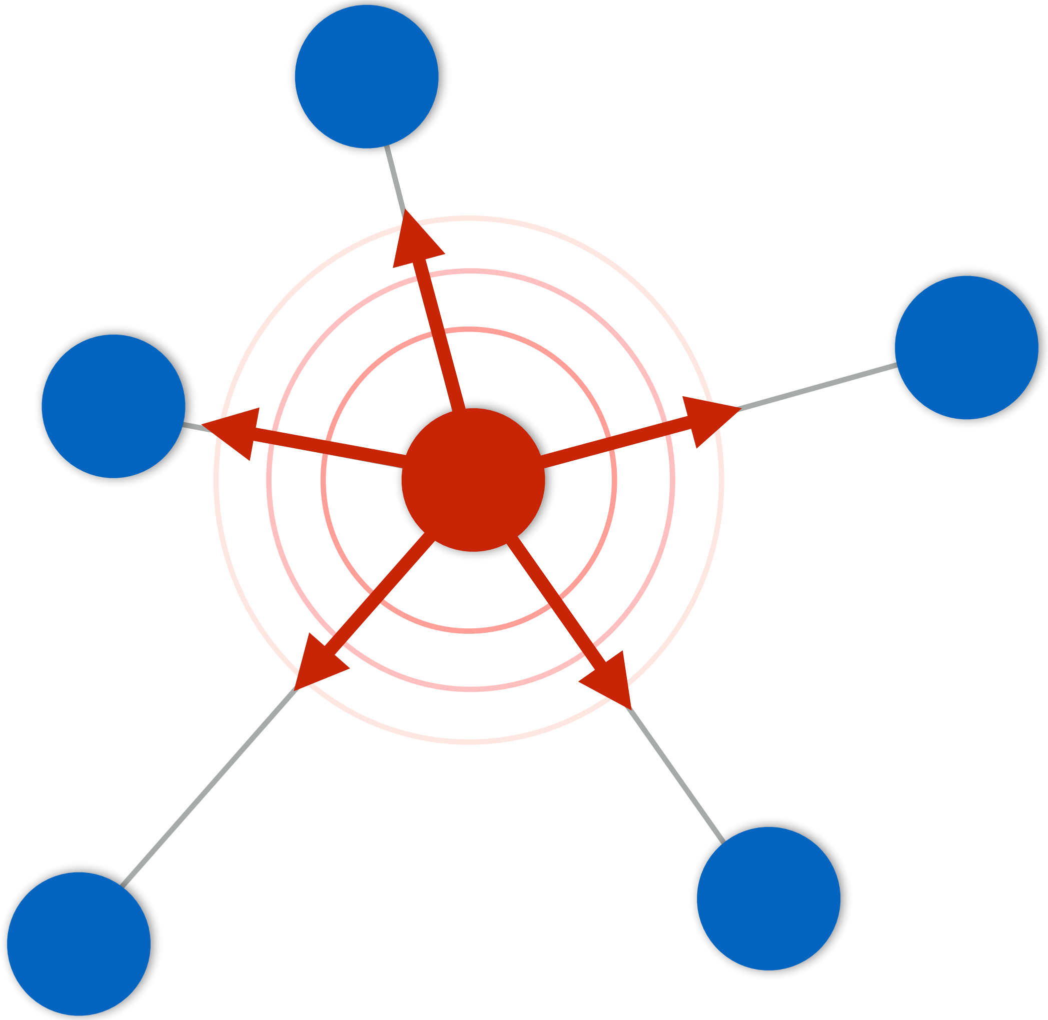
Motion of neighboring particles affect the change in density.



Constrain the motion of neighboring particles so that the net change in density is zero.

The particle itself *is* the constraint

Pressure as impulse

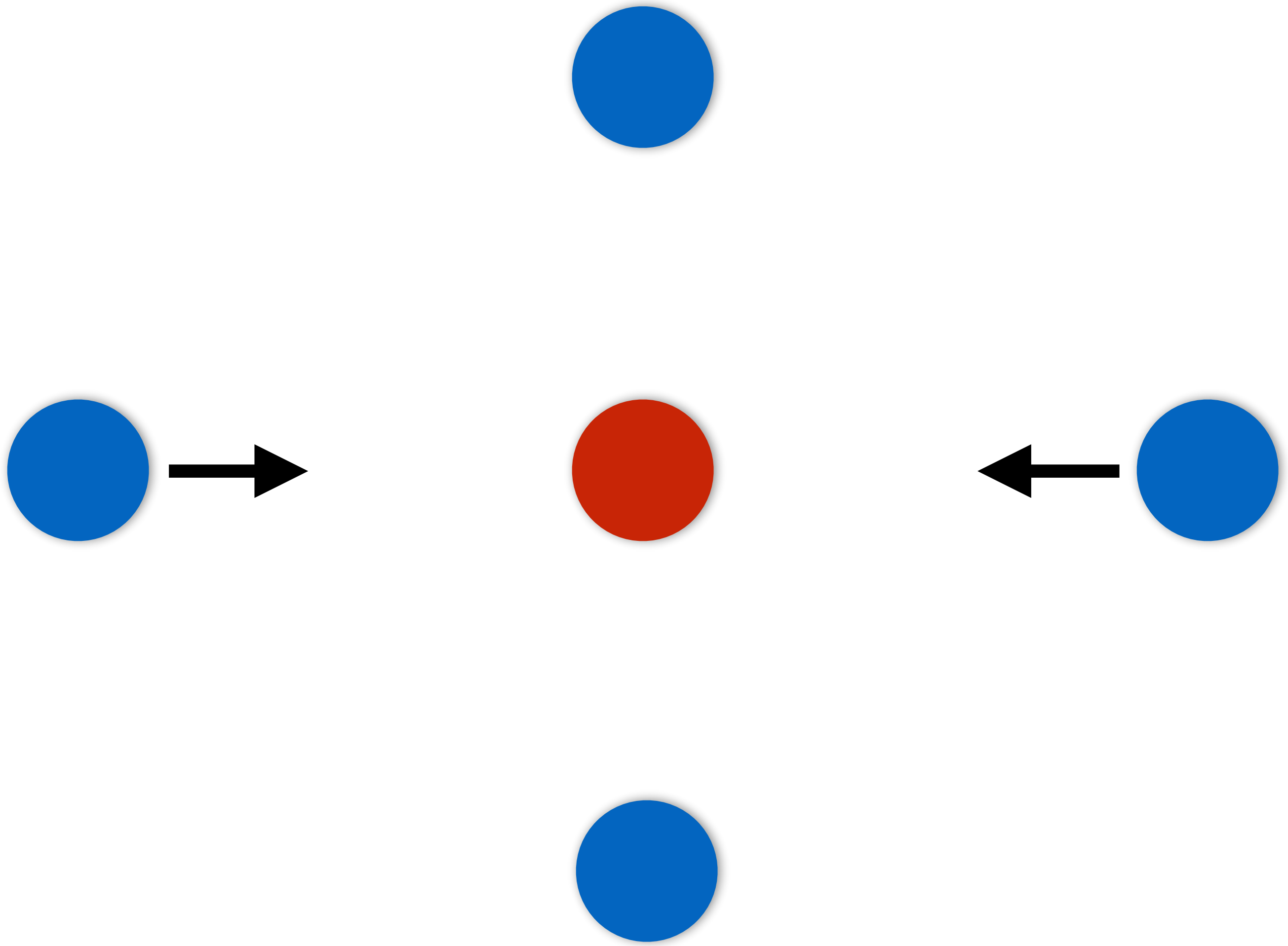


Constraint formulation

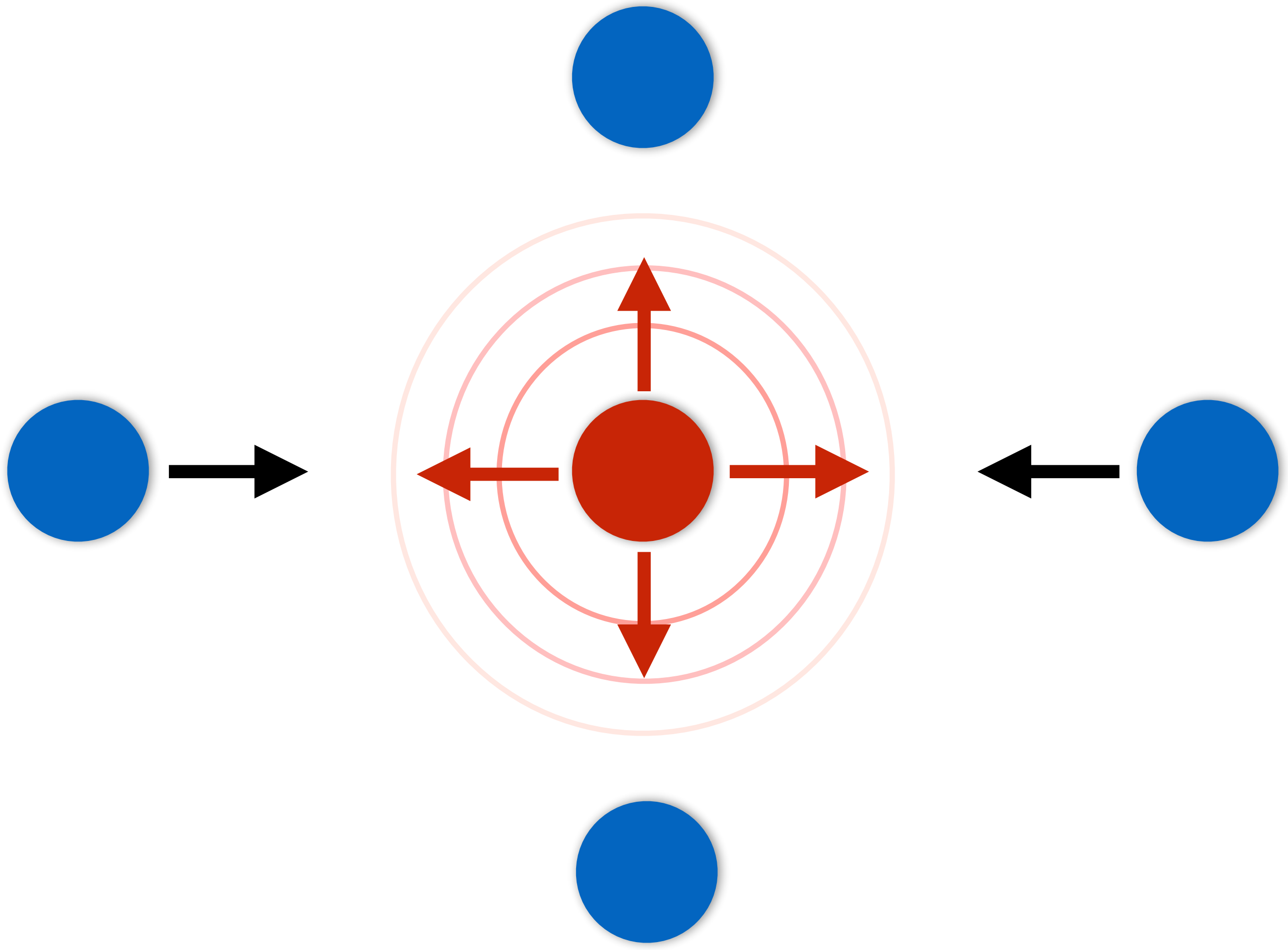
Rigid body: Find the impulse magnitude, to apply to *both* bodies, so that the relative velocity at the contact point is zero.

Fluid constraint: Find the pressure, to apply to *all neighboring* particles, so that the net change in density at the particle position is zero.

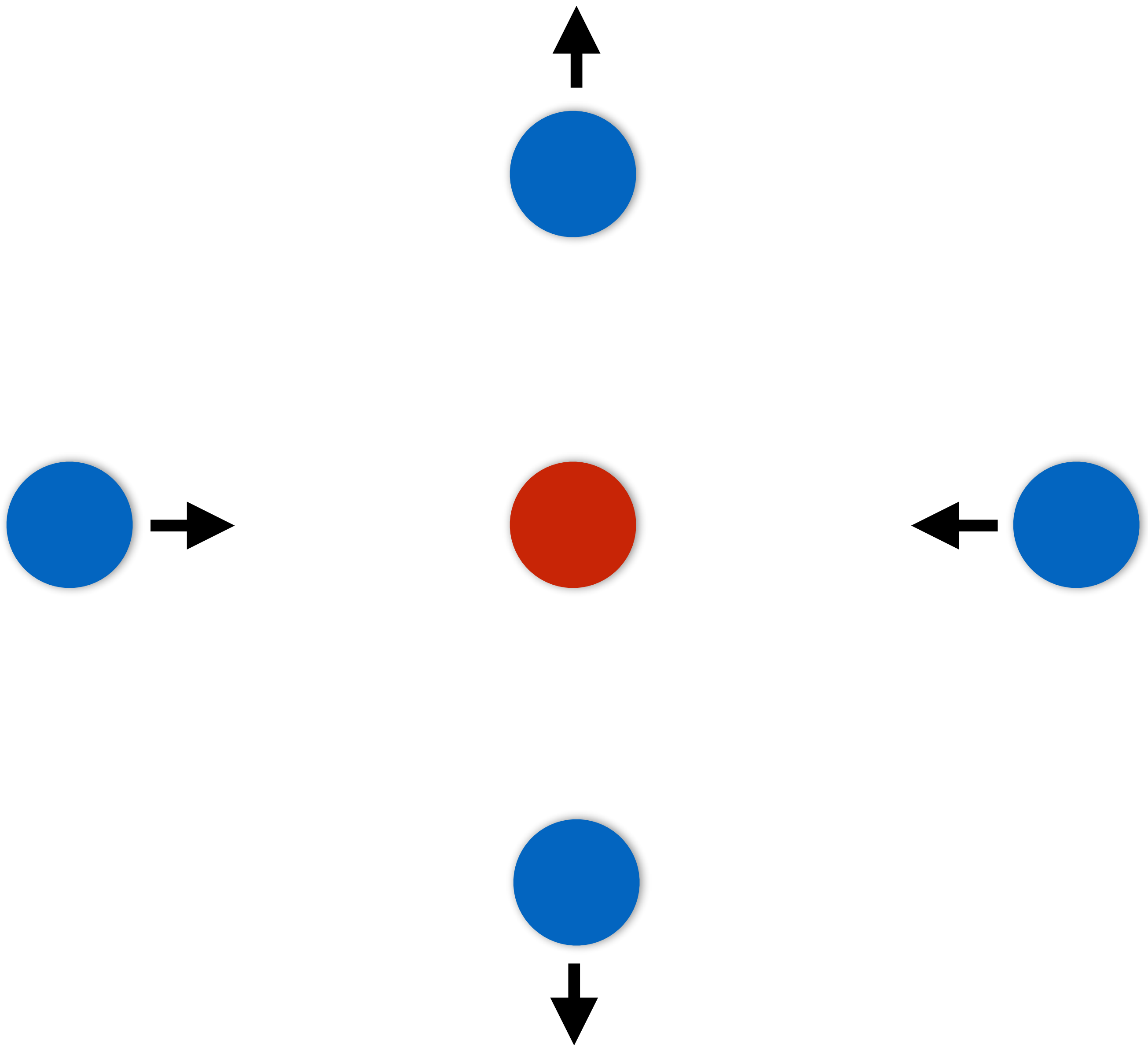
Example



Example

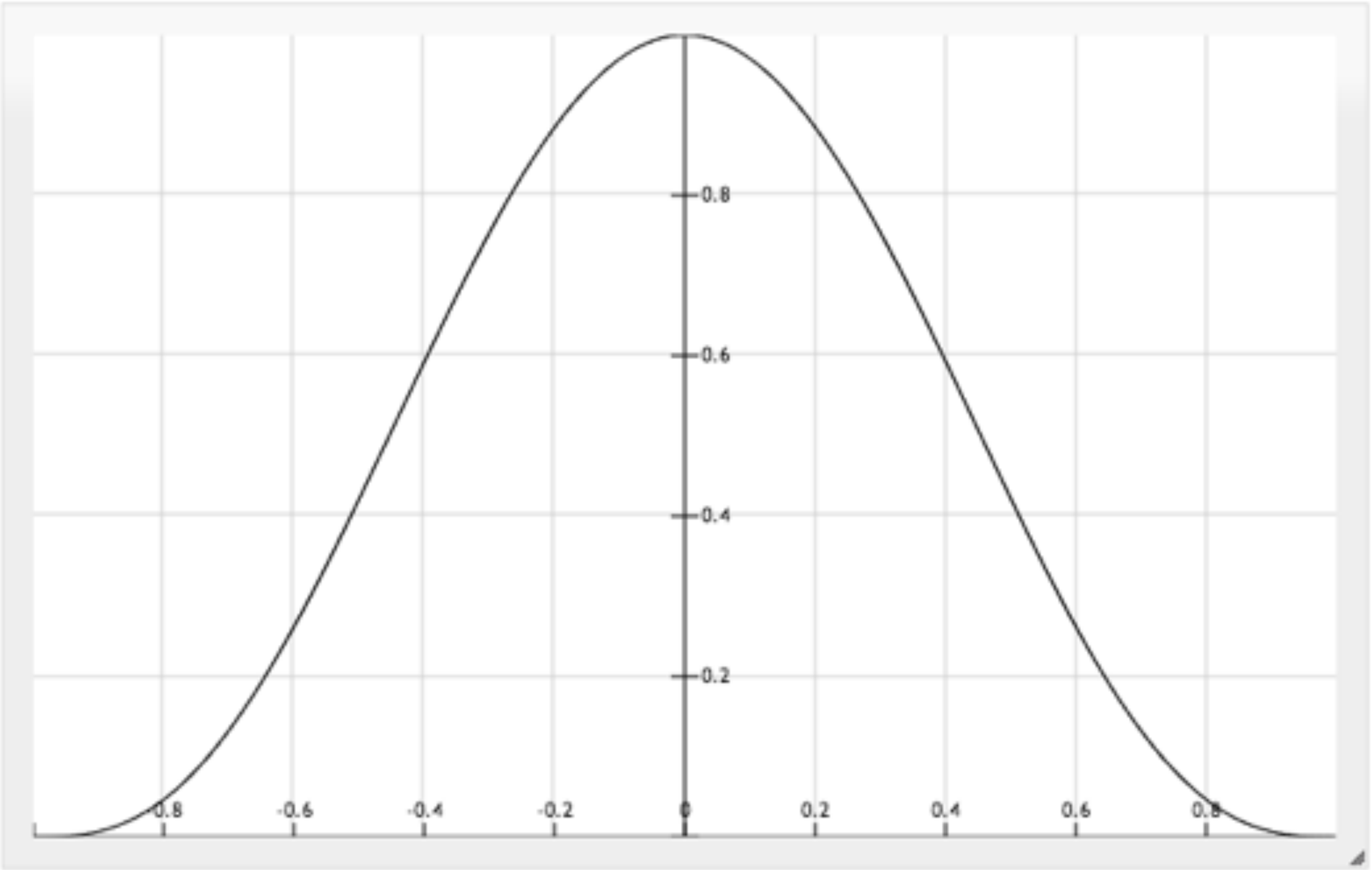


Example



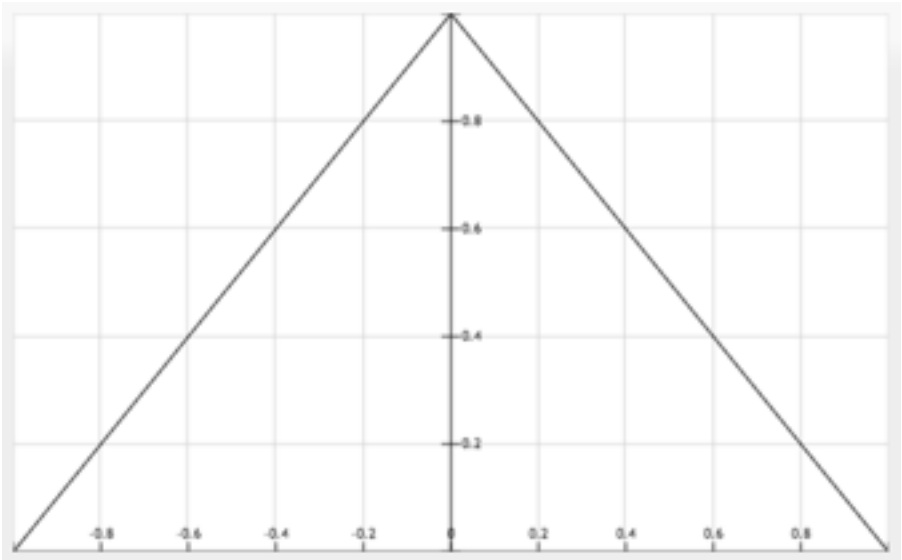
Example

Use a smoothing kernel to avoid discontinuities.



$$(1 - (d/h)^2)^3$$

$$w = 1 - d$$



Iterative solvers aren't perfect

Rigid body simulators compensate for geometric errors (inter-penetration)

The same can by done with a fluid constraint by compensating for deviations in density.

Constraint setup

```
for each particle p
        bias[p] = (restDensity-density)*baumgarte
        for each neighbor n
                d = distance/smoothingLength
                weight[n] = (1-d^2)^3
                A[p] += 2 * weight[n]^2
        next
next
```

Solver iteration

```
for each particle p
        for each neighbour n
                dv = dot(direction[n], vel[n]-vel[p])
                dpSum += weight[n]*dv
        next
        target = dpSum + bias[p]
        magnitude = max(0, target / A[p])
        for each neighbour n
                vel[n] += dir[n] * magnitude * weight[n];
                vel[p] -= dir[n] * magnitude * weight[n];
        next
end
```

measure density change

Solver iteration

```
for each particle p
        for each neighbour n
                dv = dot(direction[n], vel[n]-vel[p])
                dpSum += weight[n]*dv
        next
        target = dpSum + bias[p]
        magnitude = max(0, target / A[p])
        for each neighbour n
                vel[n] += dir[n] * magnitude * weight[n];
                vel[p] -= dir[n] * magnitude * weight[n];
        next
end
```

compute pressure impulse

Solver iteration

```
for each particle p
        for each neighbour n
                dv = dot(direction[n], vel[n]-vel[p])
                dpSum += weight[n]*dv
        next
        target = dpSum + bias[p]
        magnitude = max(0, target / A[p])
        for each neighbour n
                vel[n] += dir[n] * magnitude * weight[n];
                vel[p] -= dir[n] * magnitude * weight[n];
        next
end
```

simplified!

Solver iteration

```
for each particle p
        for each neighbour n
                dv = dot(direction[n], vel[n]-vel[p])
                dpSum += weight[n]*dv
        next
        target = dpSum + bias[p]
        magnitude = max(0, target / A[p])
        for each neighbour n
                vel[n] += dir[n] * magnitude * weight[n];
                vel[p] -= dir[n] * magnitude * weight[n];
        next
end
```

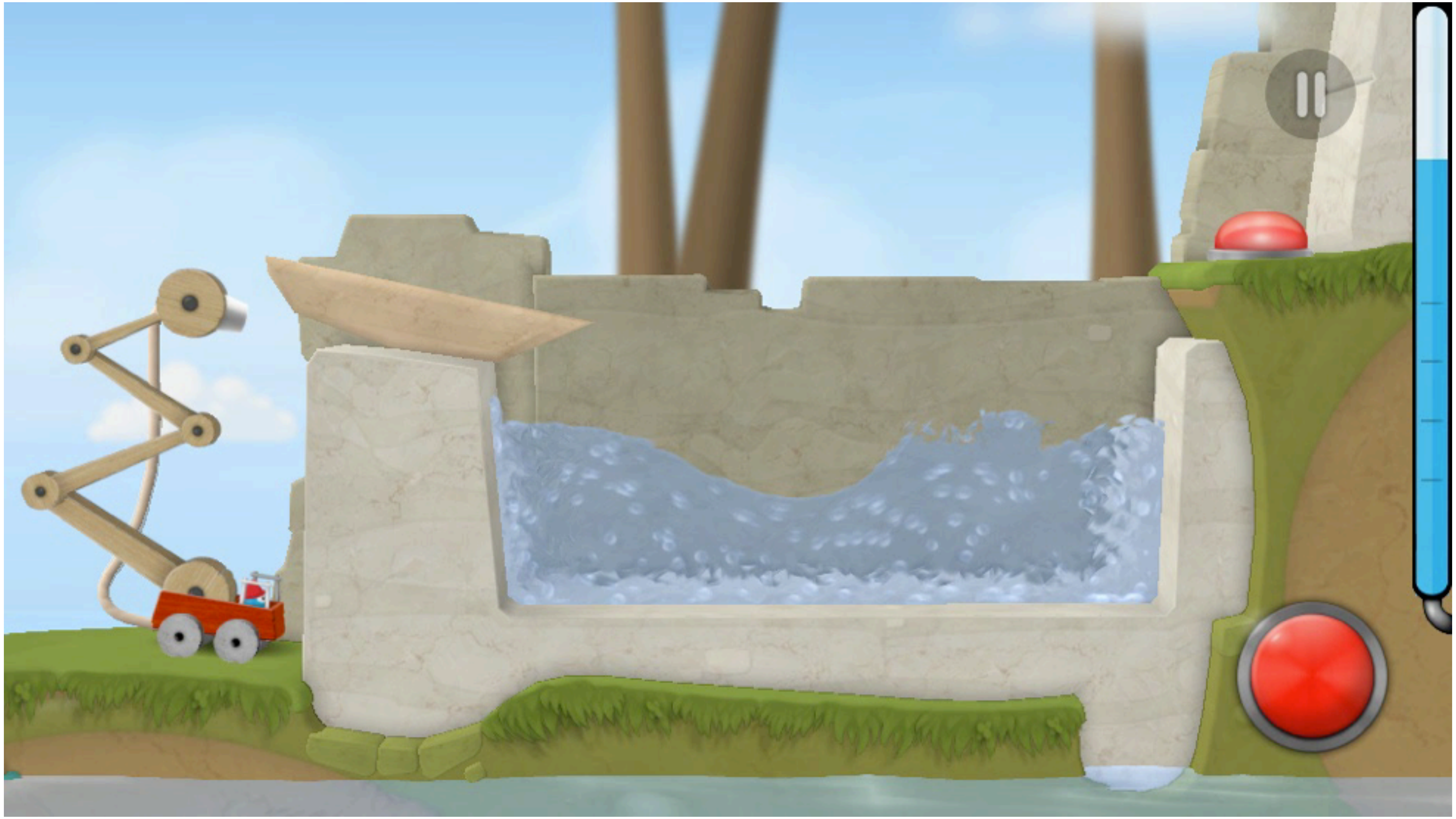
apply pressure impulse

1 iteration

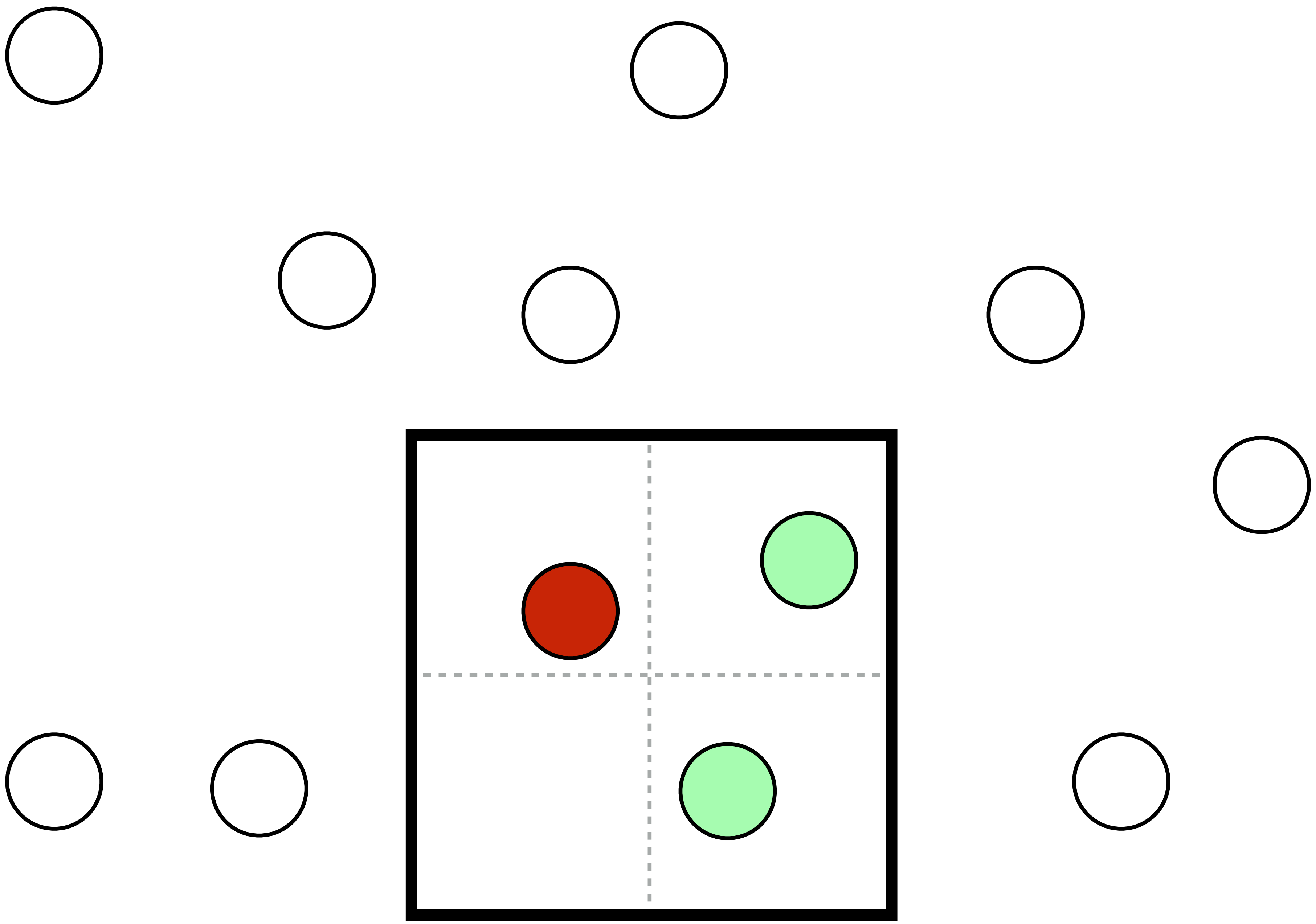
2 iterations

4 iterations

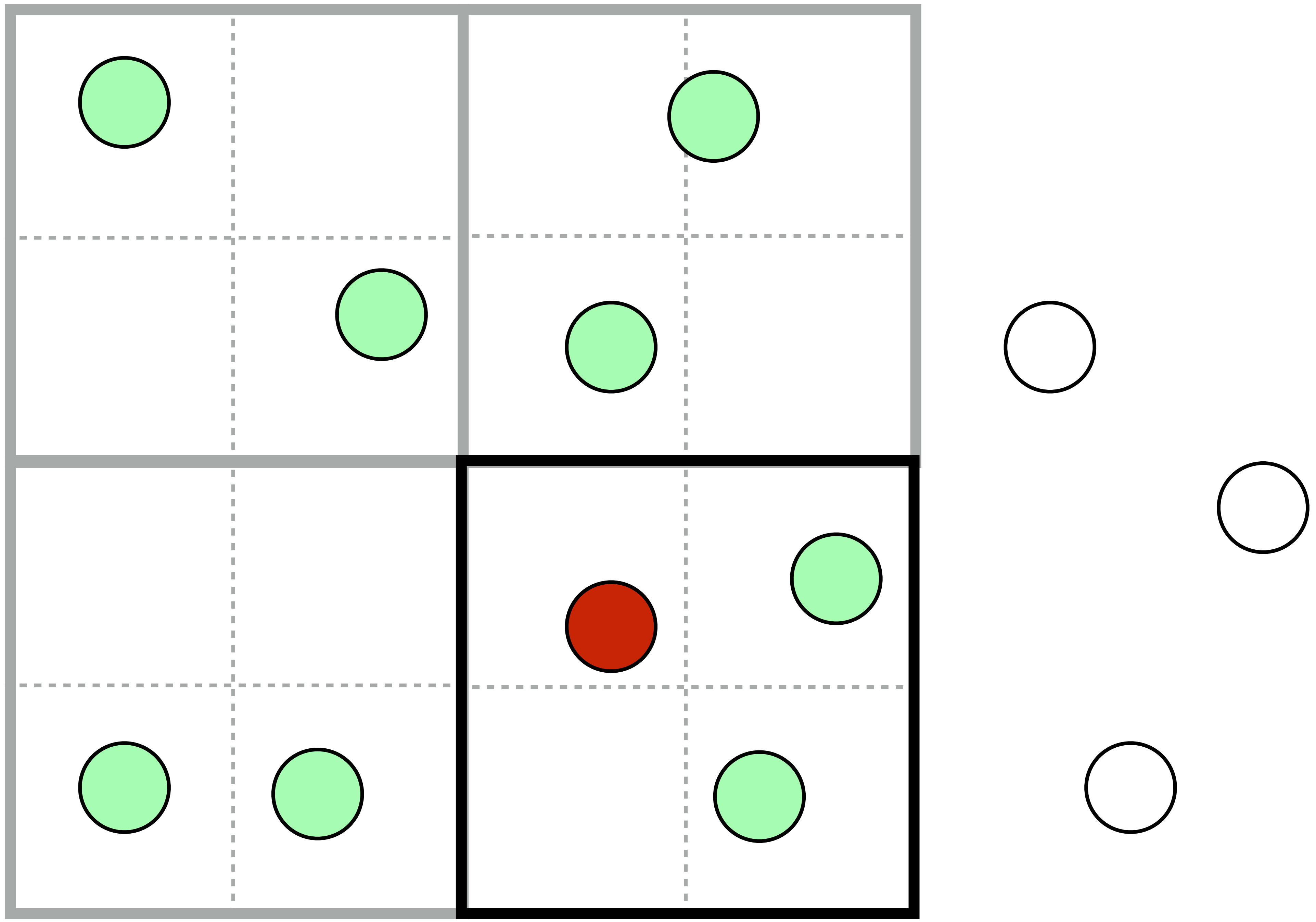
Sub-optimal implementation in Sprinkle



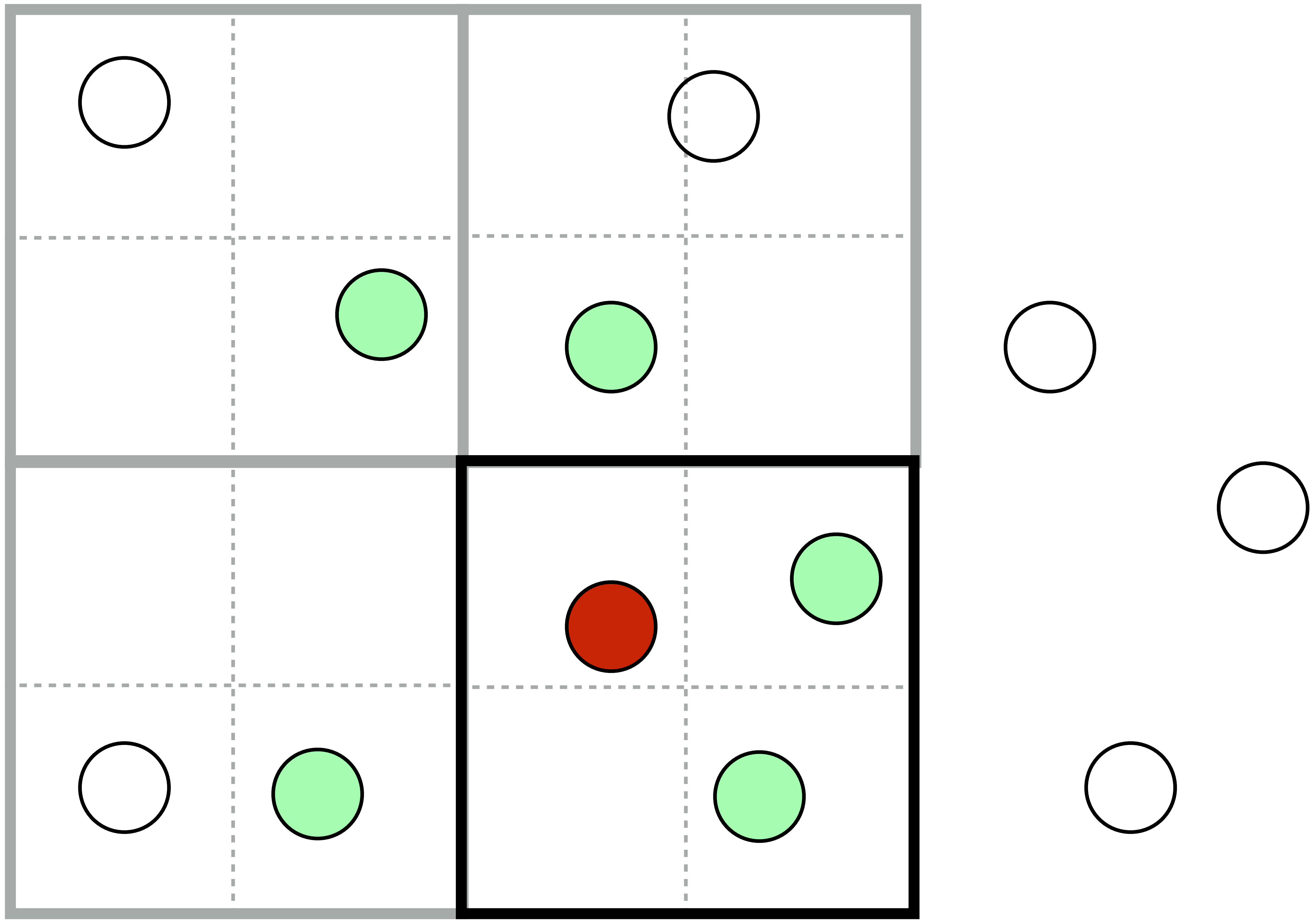
Spatial binning with quadrants



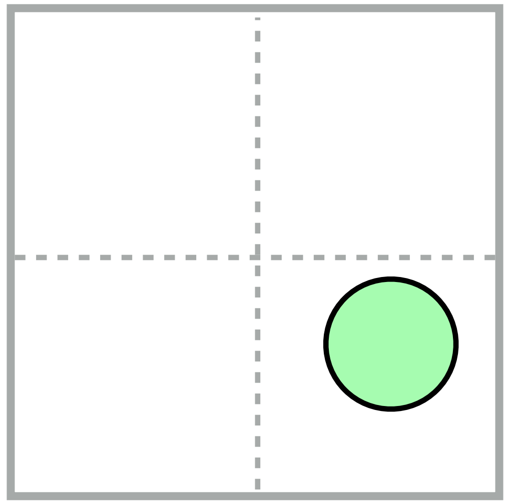
Spatial binning with quadrants



Spatial binning with quadrants



Local quantized cell coordinates



(89, -50)

- Compact 8-bit representation
- Determine quadrant by analyzing sign

Collision detection

- Reuse binning grid cells
- Box2D broad phase to collect shapes
- Dual representation for convex shapes - bounding planes

Solving collisions

- Rigid body interaction as body/particle constraint
- Solve contacts **after** fluid constraints

Memory layout

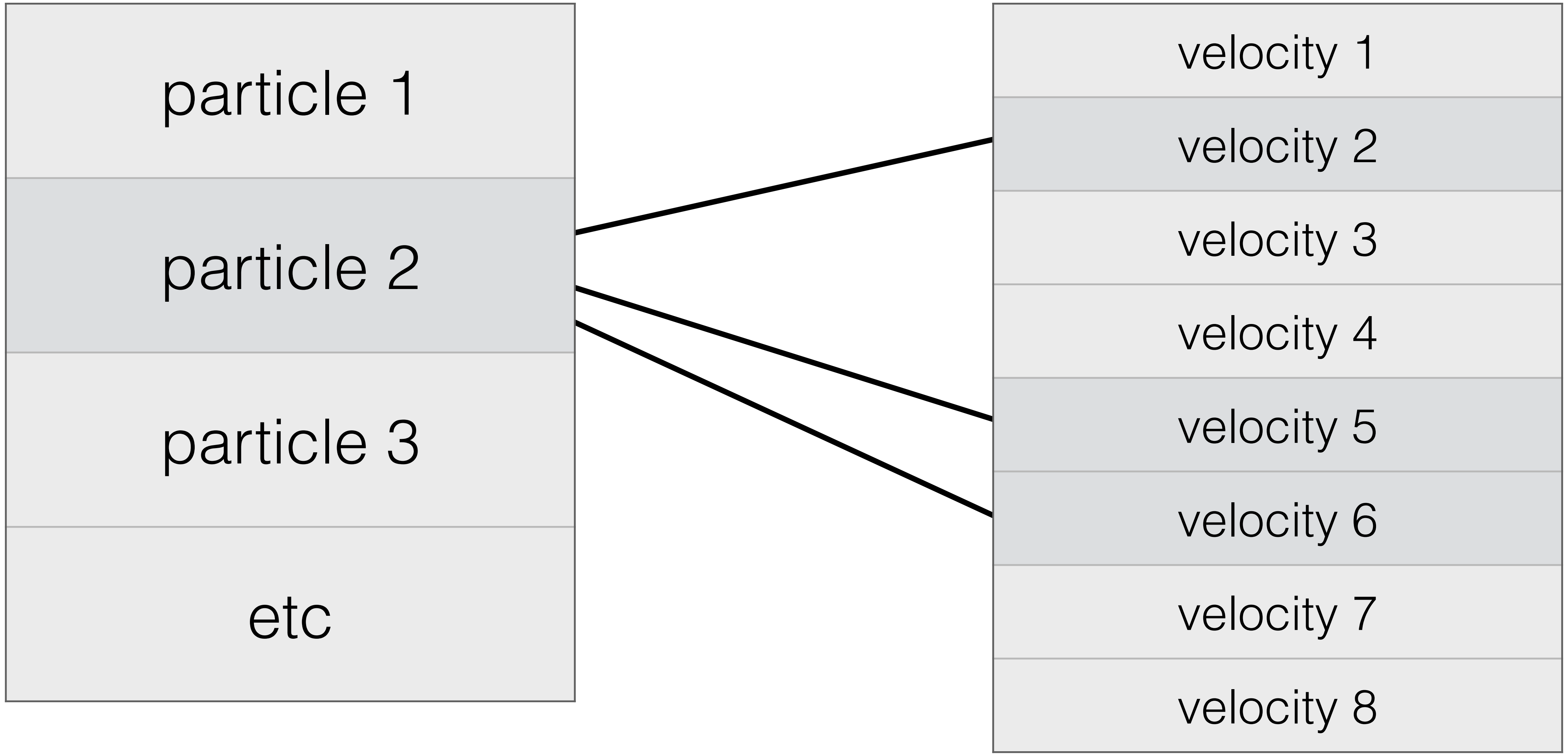
```
struct Particle
{
  vec2 position;
  Neighbor neighbors[MAX_NEIGHBORS];
  int neighborCount;
  Precomputed stuff;
};
```

```
struct Neighbor
{
  int index;
  float weight;
  vec2 direction;
};
```

Memory layout

```
Particle particles[MAX_PARTICLES];
vec2 velocities[MAX_PARTICLES];
```

Memory layout



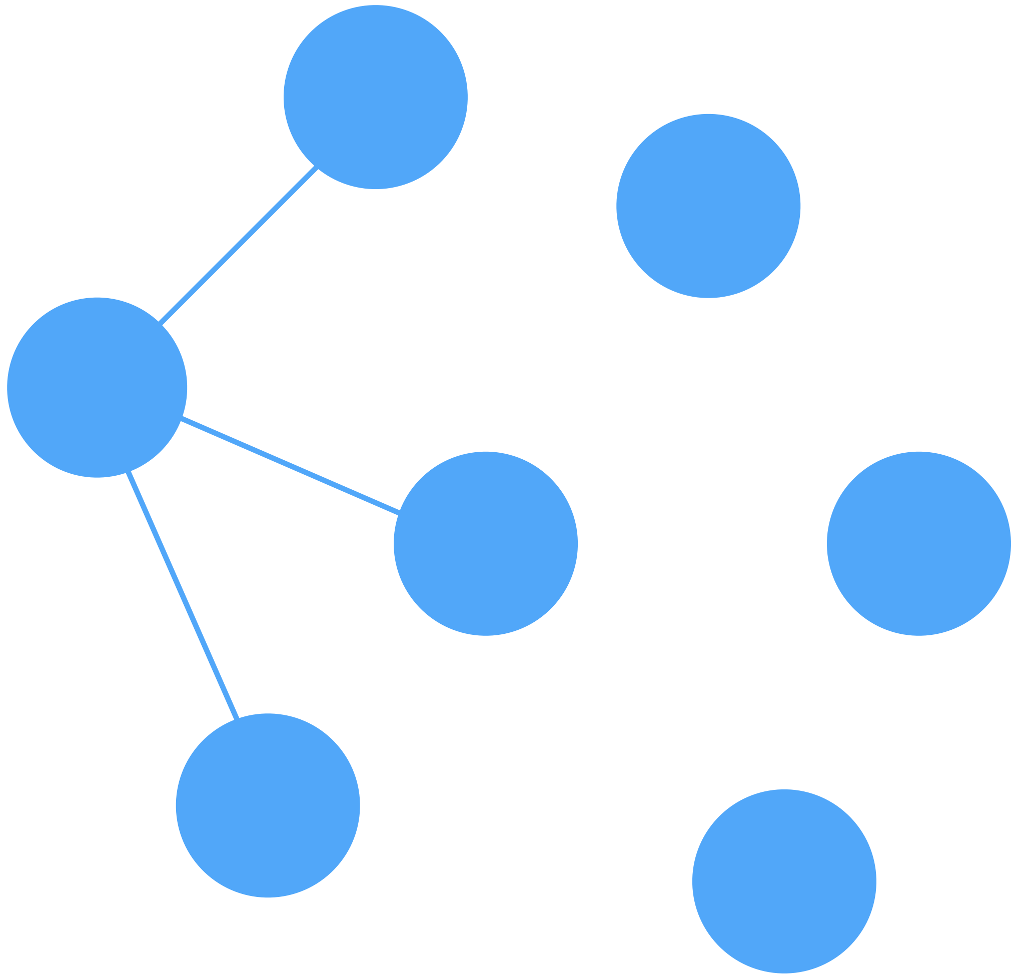
Rendering

We have a lot more information than just particle position!

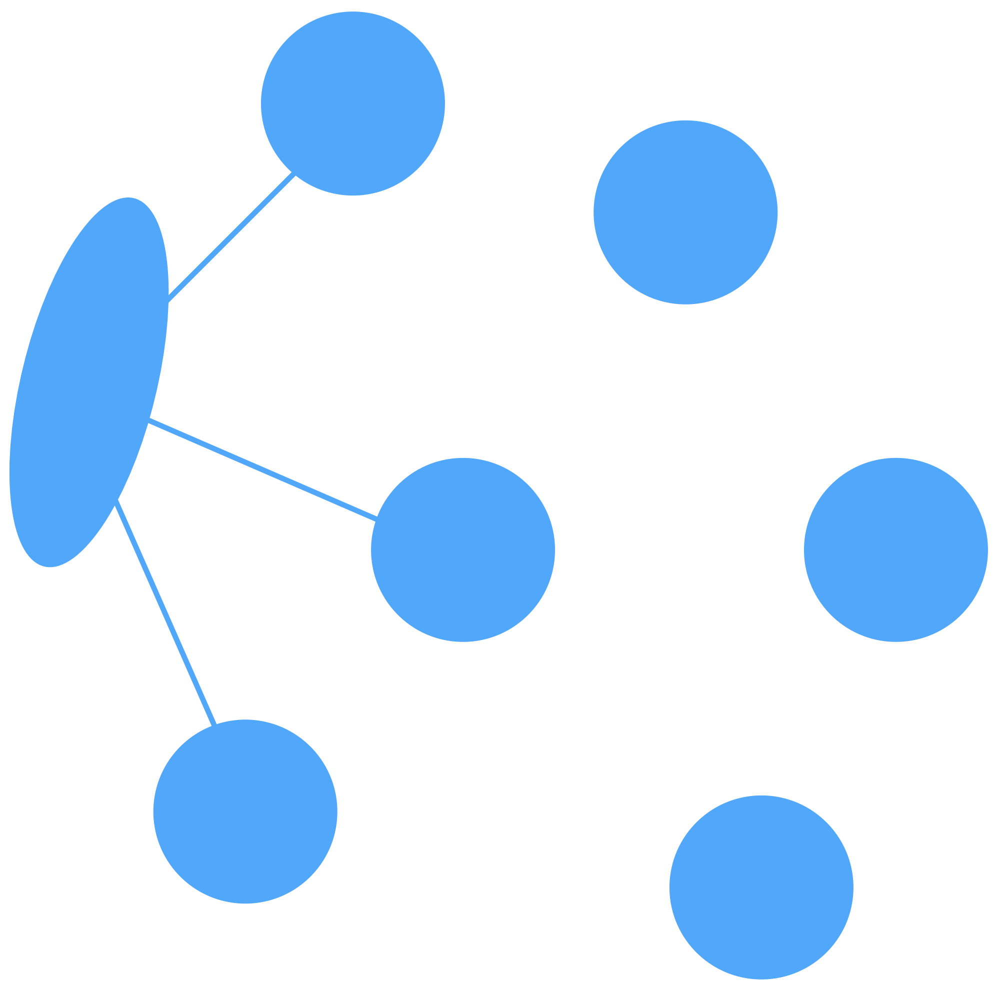
Density — particle size

Pressure gradient — particle orientation

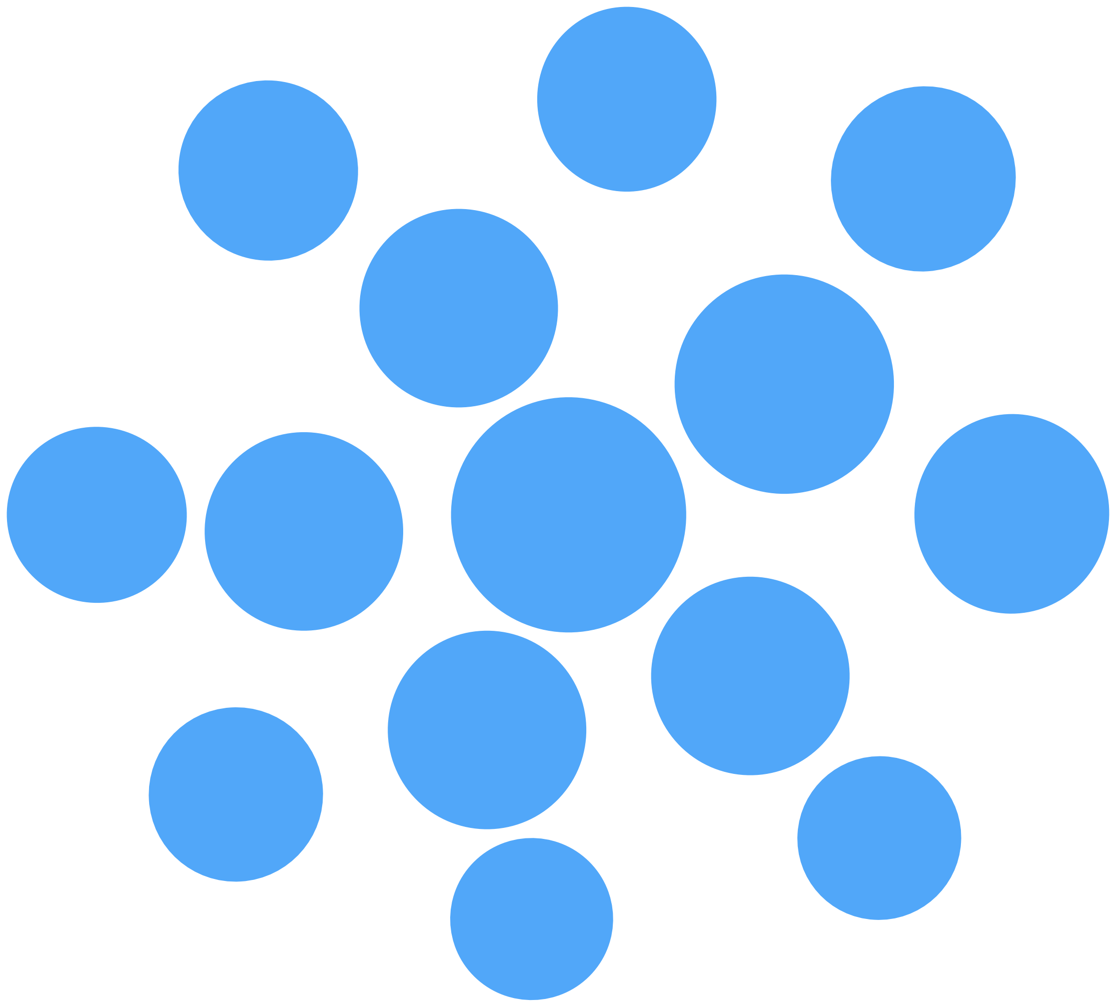
Particle orientation



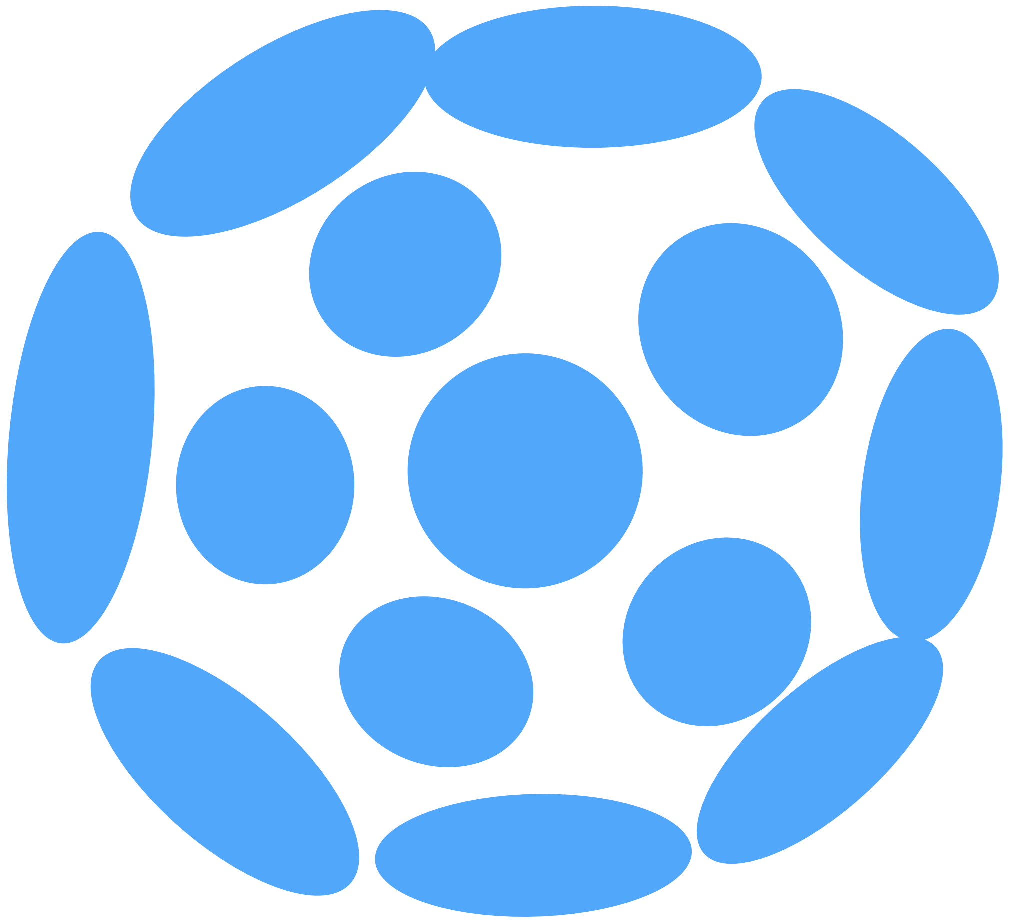
Particle orientation



Particle orientation



Particle orientation



Splashes

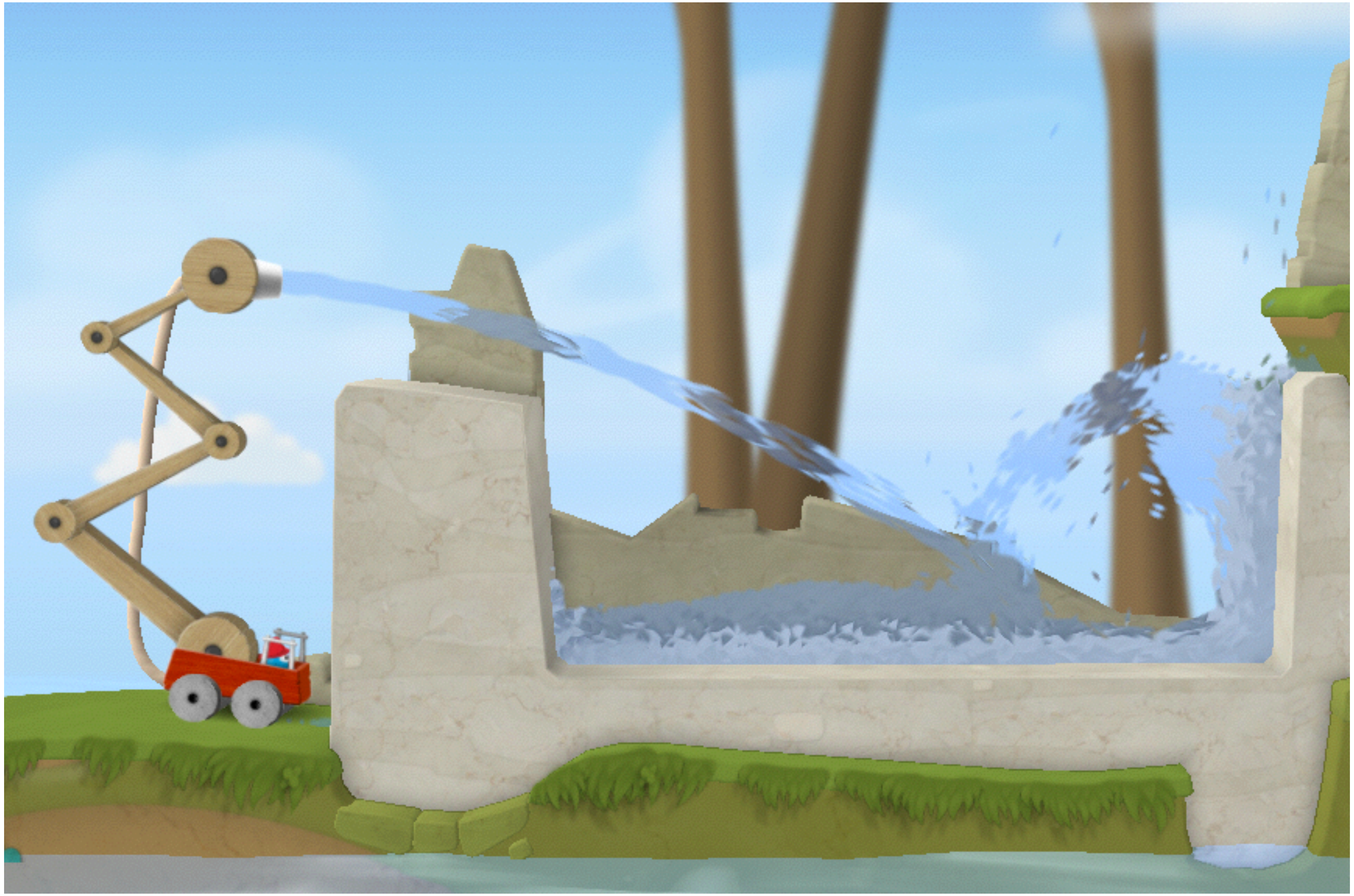
- No fluid simulation
- Ballistic motion
- Full collision detection
- Removed upon collision

Particles

Particles Resize

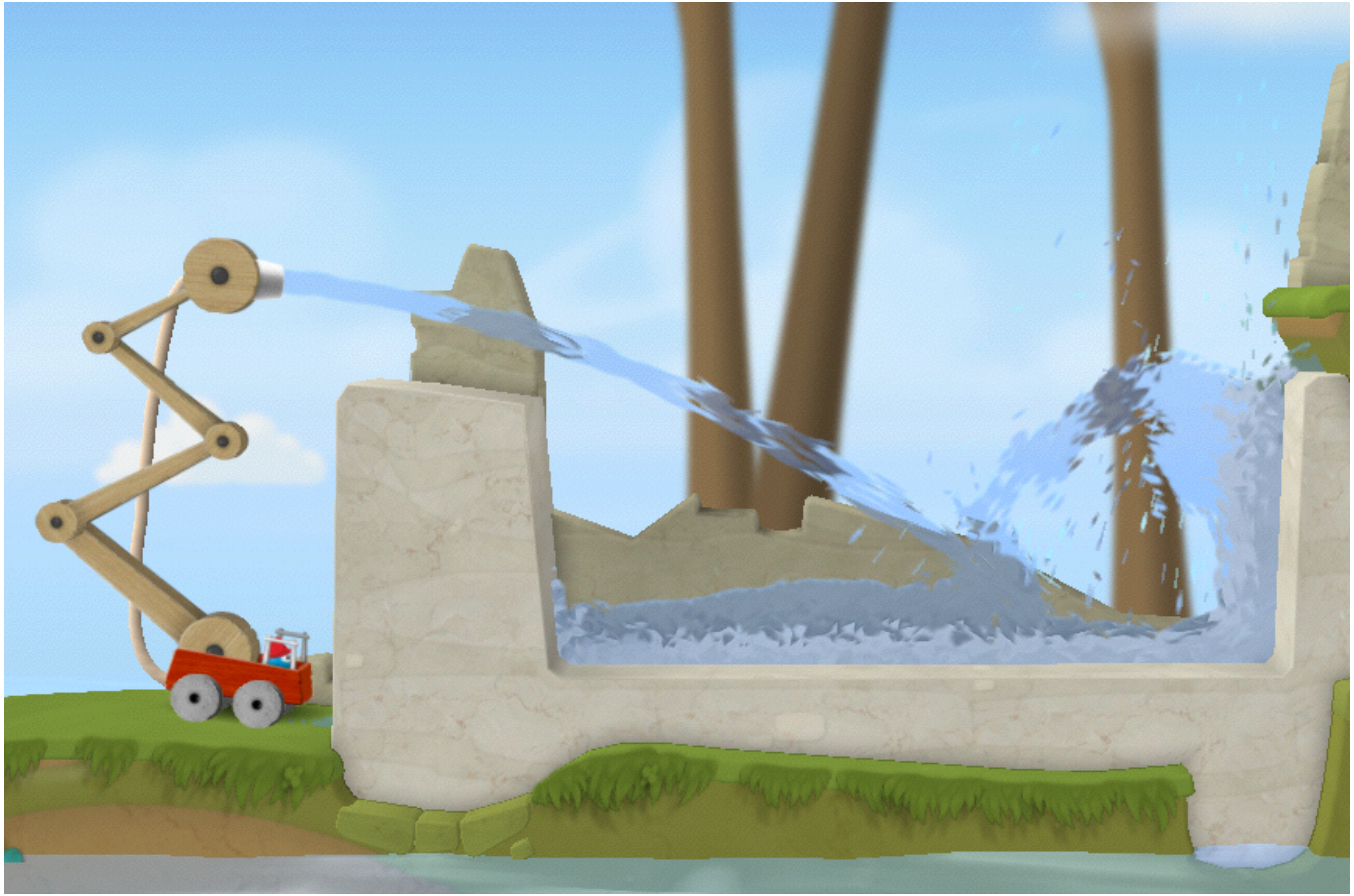
Particles
Resize
Stretch

Particles
Resize
Stretch
Refraction



Particles
Resize
Stretch
Refraction
Edges

Particles
Resize
Stretch
Refraction
Edges
Splashes



Particles
Resize
Stretch
Refraction
Edges
Splashes
Bubbles

Particles
Resize
Stretch
Refraction
Edges
Splashes
Bubbles

Thank you

dennis@mediocre.se

http://tuxedolabs.blogspot.com