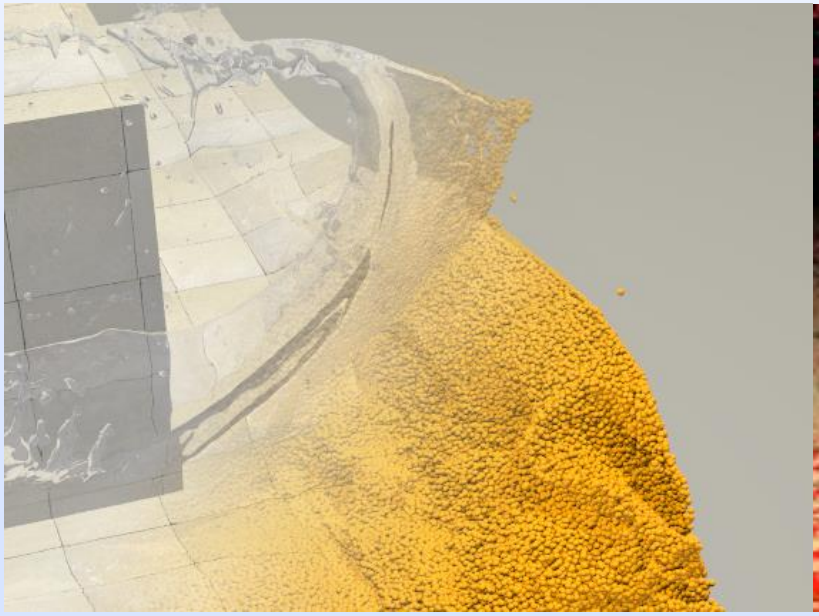
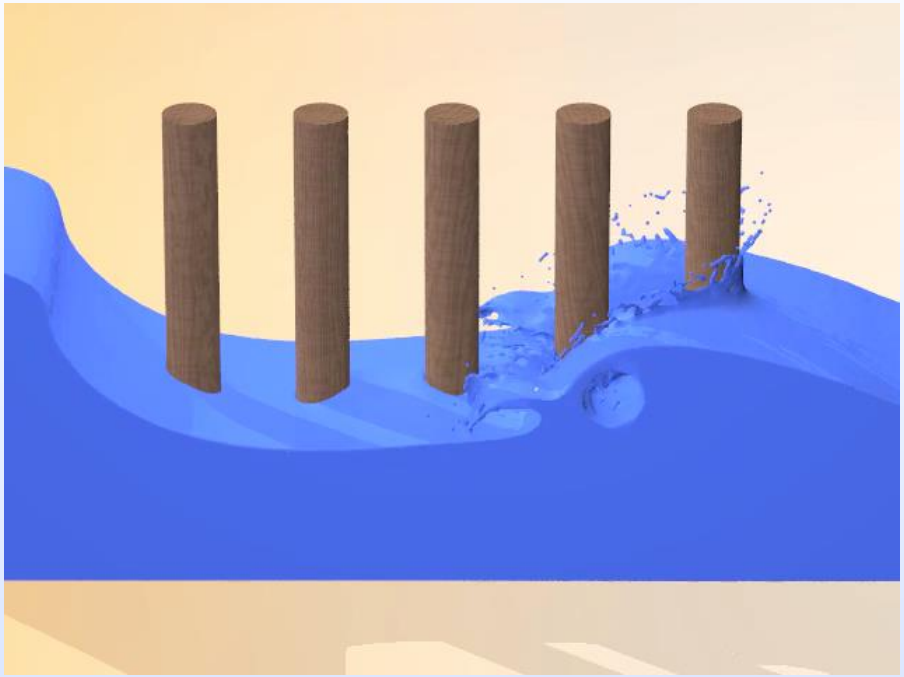
Particle-based Fluids

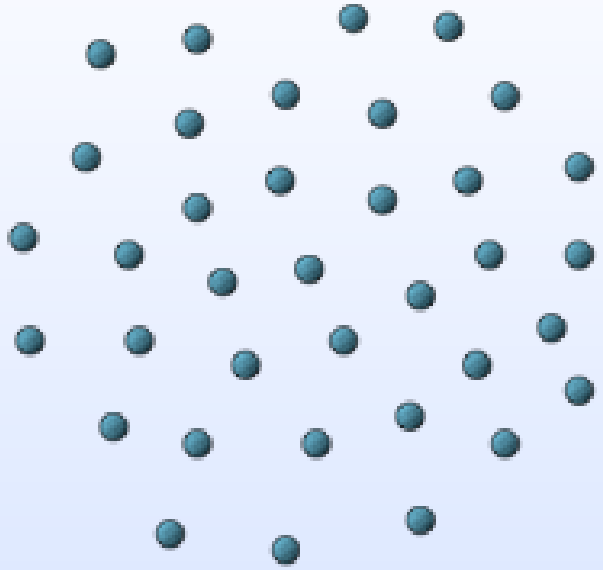


Particle Fluids



Spatial Discretization

Fluid is discretized using particles



Particles = Molecules?

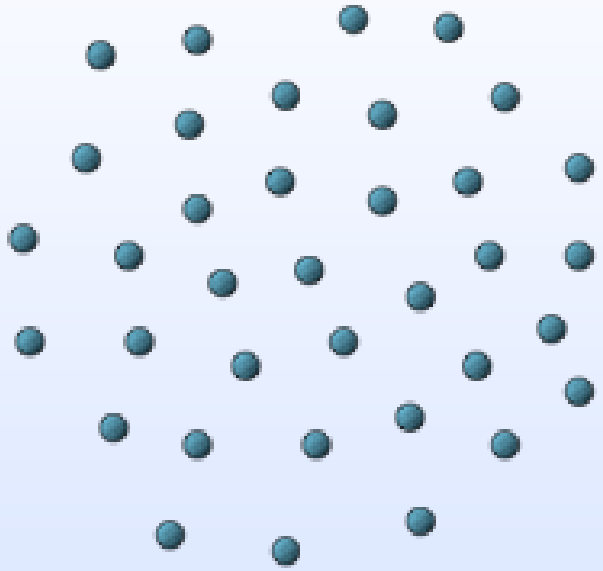
Particle approaches:

- Molecular Dynamics: relates each particle to one molecule
  - Can model molecular forces, integrate
  - 1 liter of water contains about 10^{25} molecules
- SPH: particle represents volume -> continuum assumptions apply
  - Properties such as density, pressure, etc are assumed to vary continuously in space
  - field quantities defined at discrete particle locations, use interpolation to evaluate anywhere

Smoothed Particle Hydrodynamics (SPH)

- Fluid volume is discretized by particles
- Each particles represents a certain amount of fluid volume:

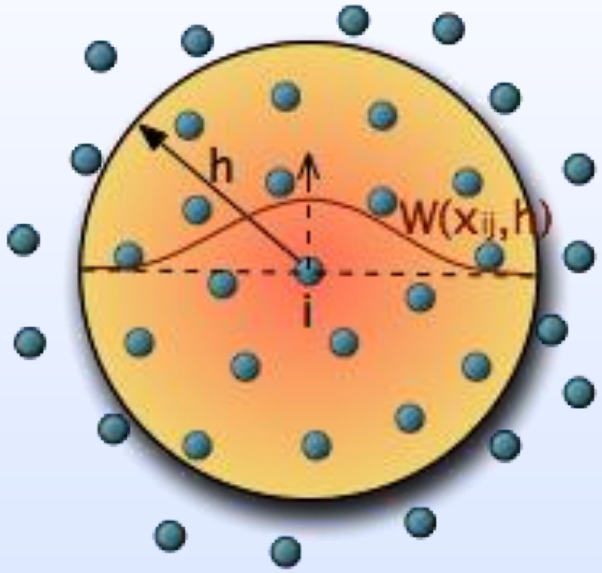
$$V_i = \frac{m_i}{\rho_i}$$



- Note: mass is constant, density is not

Smoothed Particle Hydrodynamics (SPH)

- Particles store attributes
- To evaluate an attribute, take weighted average of particle values within a neighborhood
- Smoothing kernel W prescribes interpolation weights



Kernel Properties

- Typically radially symmetric

  $\boldsymbol{r}$: offset from kernel center

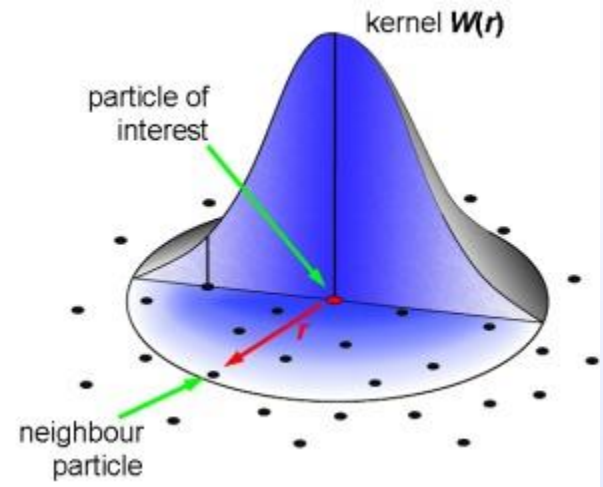
- Normalization condition:

$$\int W(\boldsymbol{r})d\boldsymbol{r} = 1$$

- If W is even ($W(\boldsymbol{r},h)=W(-\boldsymbol{r},h)$): second order accuracy

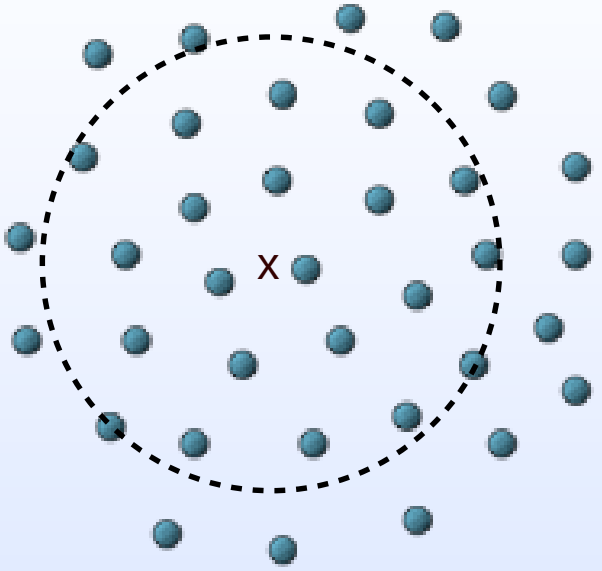
- Compact support

$$W(\boldsymbol{r},h) = 0 \; when \; |\boldsymbol{r}| > h$$



SPH Summation Equation

- Computing some quantity A at an arbitrary position in space
- Sum up contribution of neighboring particles j



$$A(\boldsymbol{x}) = \sum_j \frac{m_j}{\rho_j} A_j \, W(\boldsymbol{x} - \boldsymbol{x}_j, h)$$

Quantity A at arbitrary position x

Sum over all neighbor particles j within h

Particle volume

Quantity A of particle j

Smoothing kernel

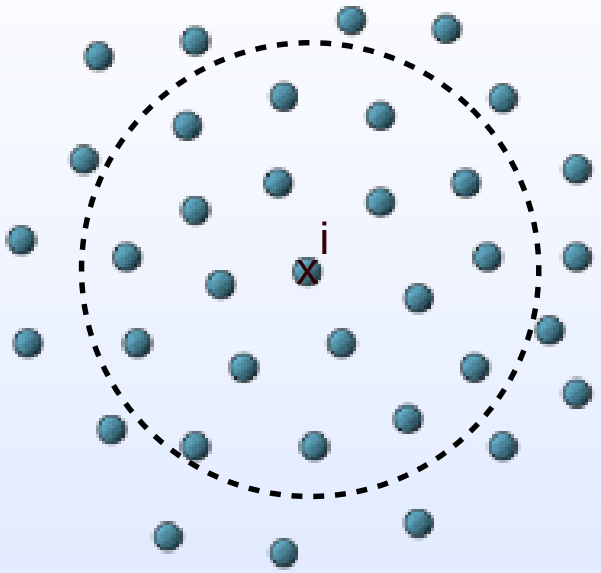
SPH Summation Equation

- A quantity A of particle i can be computed by summing up the contributions from neighbors j:

$$A_i = \sum_j \frac{m_j}{\rho_j} A_j W(\mathbf{x}_i - \mathbf{x}_j, h)$$

$$A_i = \sum_j \frac{m_j}{\rho_j} A_j W(\mathbf{x}_{ij}, h)$$

$$A_i = \sum_j \frac{m_j}{\rho_j} A_j W_{ij}$$



Note: Neighborhood j includes particle i

Differentiation

- Gradient and Laplacian can be easily calculated

$$A_i = \sum_j \frac{m_j}{\rho_j} A_j W_{ij}$$

$$\nabla A_i = \sum_j \frac{m_j}{\rho_j} A_j \nabla W_{ij}$$

$$\nabla^2 A_i = \sum_j \frac{m_j}{\rho_j} A_j \nabla^2 W_{ij}$$

Numerical Solution to NS Equations

$$\rho \frac{\partial \mathbf{u}}{\partial t} = -\nabla p + \mu \nabla^2 \mathbf{u} + \rho \mathbf{g}$$

Neighbor Search

Density, Pressure → Body Force / V + Diffusion / V + Pressure Force / V

Estimating Density

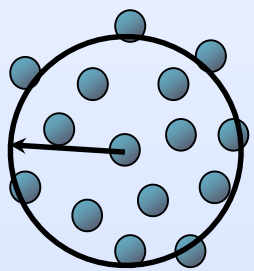
$$\rho\frac{\partial \mathbf{u}}{\partial t} = -\nabla p + \mu\nabla^2\mathbf{u} + \rho g$$

$$A_i = \sum_j \frac{m_j}{\rho_j} A_j W_{ij}$$

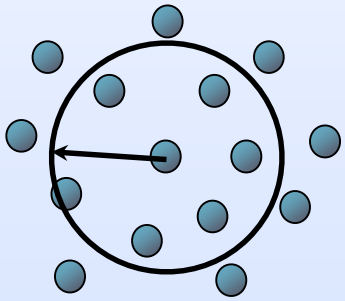
- Attribute A is now density ρ

$$\rho_i = \sum_j \frac{m_j}{\rho_j} \rho_j W_{ij}$$

$$\rho_i = \sum_j m_j W_{ij}$$



Larger density



Smaller density

From Density to Pressure

$$\rho\frac{\partial \mathbf{u}}{\partial t} = -\nabla p + \mu\nabla^2\mathbf{u} + \rho\mathbf{g}$$

- Pressure is computed through the ideal gas law: $p = k\rho$
- Modified version gives pressure of particle i:

$$p_i = k(\rho_i - \rho_0)$$

Stiffness parameter (~Specific Gas Constant)

Rest density of fluid (water: 1000)

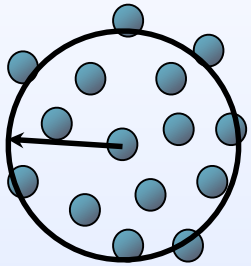
- Modified pressure is proportional to density deviation
- Gradient (forces) should not be affected by offset, but SPH approximation is numerically more stable
- Stiffness k defines speed of response (*note*: stiff system -> smaller time step)

Density and Pressure - Example

$$p_i = k(\rho_i - \rho_0)$$

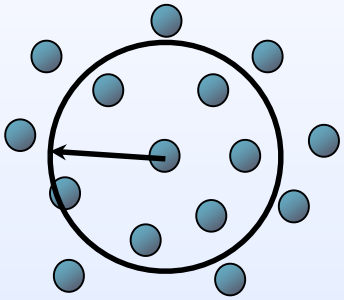
$$\rho\frac{\partial \mathbf{u}}{\partial t} = -\nabla p + \mu\nabla^2\mathbf{u} + \rho g$$

Example: $\rho_0 = 1000$



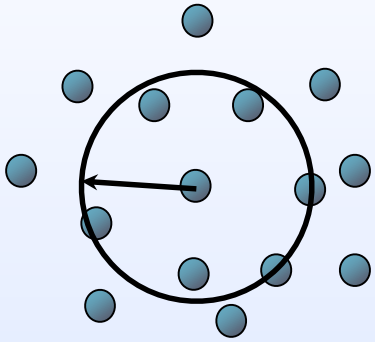
Larger density
Larger pressure

$$\rho_i = 1200$$
$$p_i = k(1200 - 1000)$$
$$= k * 200$$



Smaller density
Smaller pressure

$$\rho_i = 1010$$
$$p_i = k(1010 - 1000)$$
$$= k * 10$$



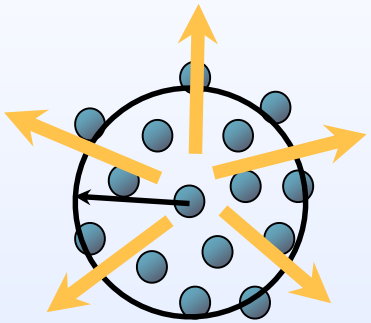
Density below rest density
Negative pressure

$$\rho_i = 800$$
$$p_i = k(800 - 1000)$$
$$= -k * 200$$

Pressure Force Density

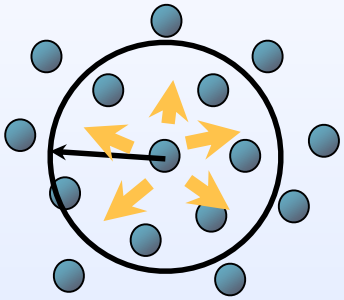
$$f^{pressure} = -\nabla p$$

$$\rho\frac{\partial \mathbf{u}}{\partial t} = -\nabla p + \mu\nabla^2\mathbf{u} + \rho g$$



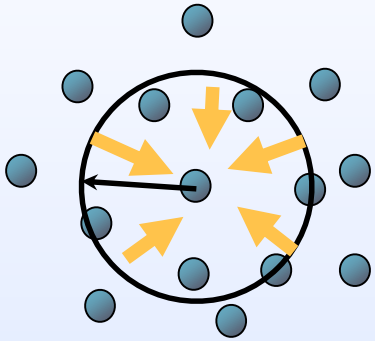
Larger density
Larger pressure
Larger repulsion forces

$$\rho_i = 1200$$
$$p_i = k(1200 - 1000)$$
$$= k * 200$$



Smaller density
Smaller pressure
Smaller repulsion forces

$$\rho_i = 1010$$
$$p_i = k(1010 - 1000)$$
$$= k * 10$$



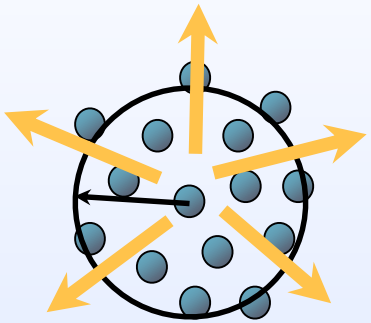
Density below rest density
Negative pressure
Attraction forces

$$\rho_i = 800$$
$$p_i = k(800 - 1000)$$
$$= -k * 200$$

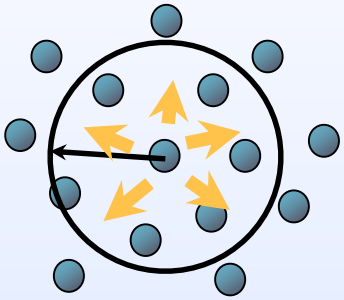
Pressure Force Density

$$f^{pressure} = -\nabla p$$

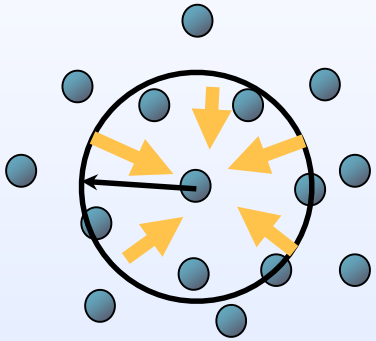
$$\rho\frac{\partial \mathbf{u}}{\partial t} = -\nabla p + \mu\nabla^2\mathbf{u} + \rho g$$



Larger density
Larger pressure
Larger repulsion forces



Smaller density
Smaller pressure
Smaller repulsion forces



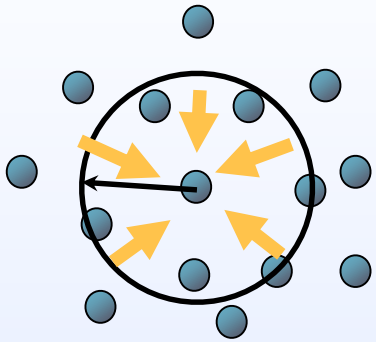
Density below rest density
Negative pressure
Attraction forces

- Pressure force aims to restore the rest state of the fluid (where $\forall_i \rho_i = \rho_0$)

Attraction Forces

- Attraction forces can lead to numerical instability
- Typically, only repulsion forces are used:

$$p_i = max(k(\rho_i - \rho_0), 0)$$



Density below rest density
Negative pressure
Attraction forces

Pressure Force Density

$$\nabla A_i = \sum_j \frac{m_j}{\rho_j} A_j \nabla W_{ij}$$

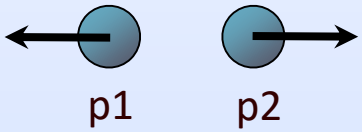
$$\rho \frac{\partial \mathbf{u}}{\partial t} = -\nabla p + \mu \nabla^2 \mathbf{u} + \rho \mathrm{g}$$

For particle i $-\nabla p$ evaluates to:

$$\mathbf{f}_i^{pressure} = -\sum_j \frac{m_j}{\rho_j} p_j \nabla W_{ij}$$

Note: Kernel is multiplied by distance vector x_ij

Pressure force acts along vector between particles!



$$\mathbf{f}_1^{pressure} = -\frac{m_2}{\rho_2} p_2 \nabla W_{12} \qquad \mathbf{F}_1^{pressure} = -\frac{m_1}{\rho_1}\frac{m_2}{\rho_2} p_2 \nabla W_{12}$$

$$\mathbf{f}_2^{pressure} = -\frac{m_1}{\rho_1} p_1 \nabla W_{21} \qquad \mathbf{F}_2^{pressure} = -\frac{m_2}{\rho_2}\frac{m_1}{\rho_1} p_1 \nabla W_{21}$$

$$\mathbf{F} = \mathbf{f}V$$

if p1 != p2: forces not symmetric!

Pressure Force Density

$$\rho\frac{\partial \mathbf{u}}{\partial t} = -\nabla p + \mu\nabla^2\mathbf{u} + \rho\mathbf{g}$$

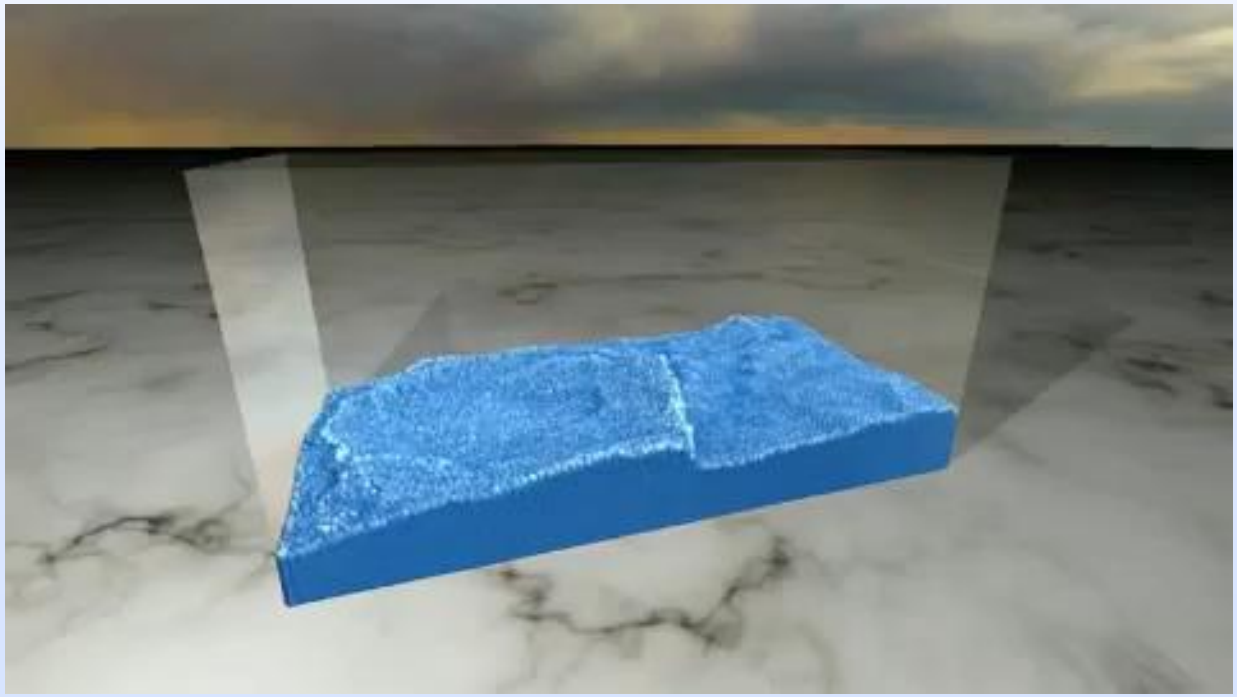
- Non-symmetric forces violate Newton's 3rd law action = reaction
- Use arithmetic mean of pressure values

$$\mathbf{f}_i^{pressure} = -\sum_j \frac{m_j}{\rho_j}\frac{p_i + p_j}{2}\nabla W_{ij}$$

$$\mathbf{F}_1^{pressure} = \mathbf{F}_2^{pressure}$$

Incompressibility

- The stiffer the fluid (the larger k), the less compression
  - Incompressibility by using very (!) large k (density variation < 1%)



Viscosity Force Density

$$\rho\frac{\partial \mathbf{u}}{\partial t} = -\nabla p + \mu\nabla^2\mathbf{u} + \rho g$$

- For particle i, $\mu\nabla^2\mathbf{u}$ evaluates to :

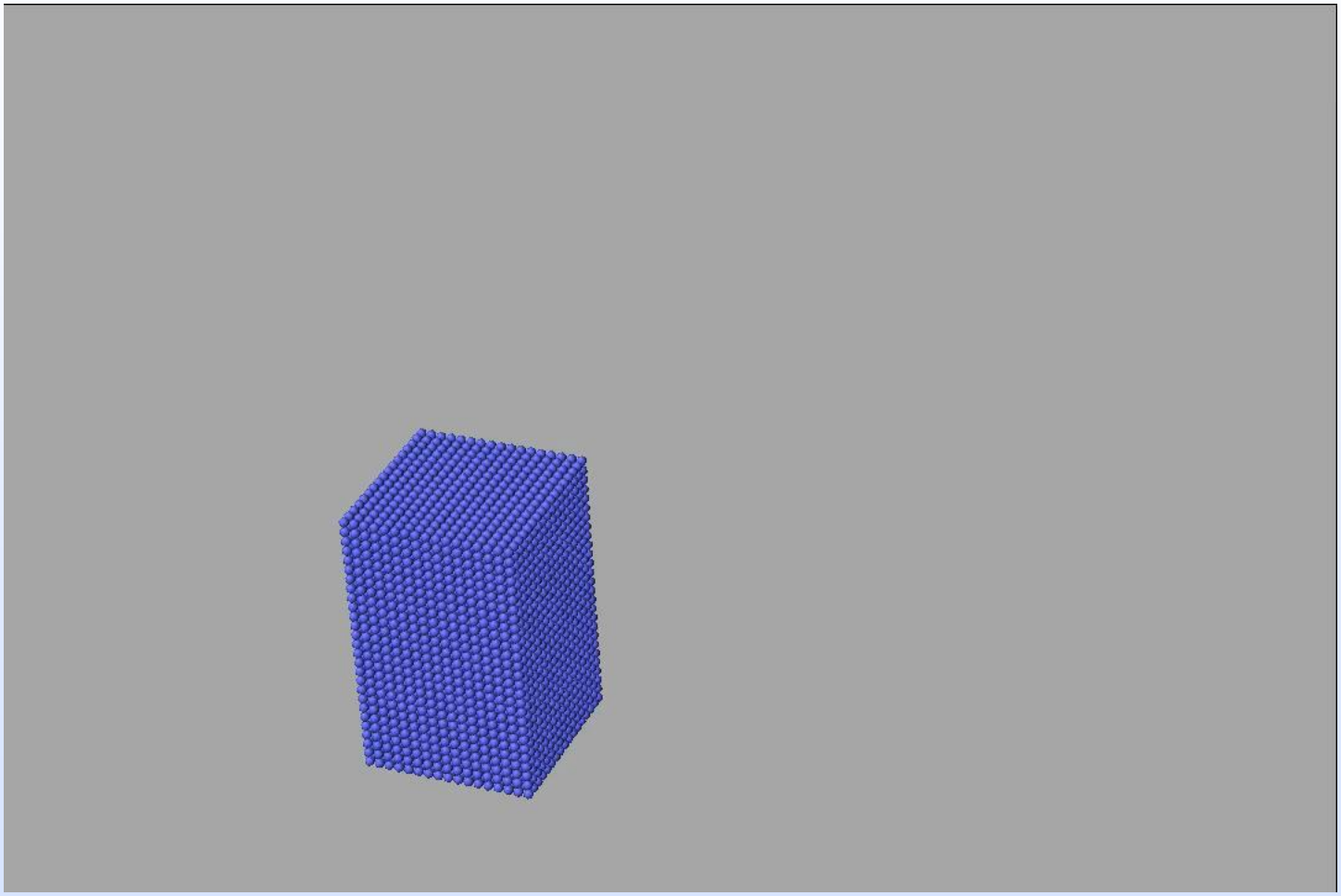
$$\mathbf{f}_i^{viscosity} = \mu\sum_j \frac{m_j}{\rho_j}\mathbf{u}_j\nabla^2 W_{ij}$$

- Once again need to make forces symmetric. Insight: viscosity forces are only dependent on velocity differences, not on absolute velocities

$$\mathbf{f}_i^{viscosity} = \mu\sum_j \frac{m_j}{\rho_j}\mathbf{u}_j - \mathbf{u}_i\nabla^2 W_{ij}$$

Note: Viscosity is always needed to stabilize the particle system

Zero Viscosity vs Normal Viscosity vs High Viscosity



Gravitational Force Density

$$\rho\frac{\partial \mathbf{u}}{\partial t} = -\nabla p + \mu\nabla^2\mathbf{u} + \rho\mathrm{g}$$

- For particle i we get:

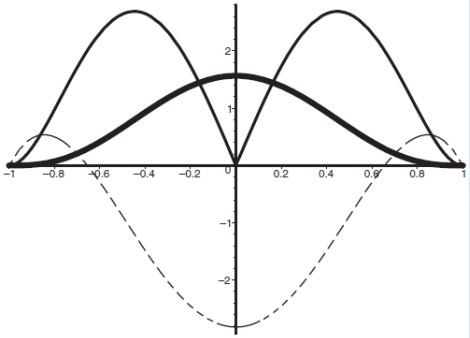
$$\mathbf{f}_i^{gravity} = \rho_i \mathrm{g}$$

  - Other forces can easily be included (collision forces, boundary forces, user interaction, etc.)

Kernel Function

- Different kernels can be used, see for example [Müller03]
- Choice of kernel affects stability, accuracy and speed of SPH methods
- Poly6 kernel: use for everything but pressure forces & viscosity

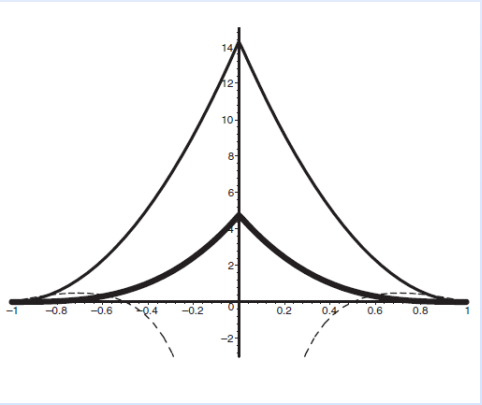
$$W_{\text{poly6}}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & 0 \le r \le h \\ 0 & \text{otherwise} \end{cases}$$



Kernel Function

- Different kernels can be used, see for example [Müller03]
- Choice of kernel affects stability, accuracy and speed of SPH methods
- Spiky kernel: use for pressure forces

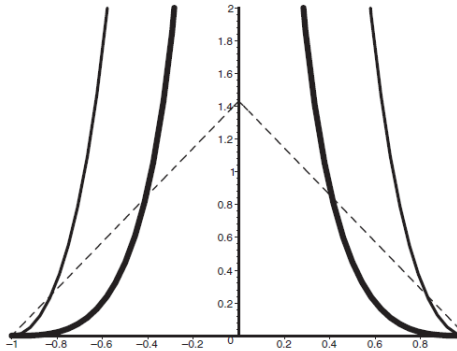
$$W_{\text{spiky}}(\mathbf{r}, h) = \frac{15}{\pi h^6} \begin{cases} (h - r)^3 & 0 \leq r \leq h \\ 0 & \text{otherwise,} \end{cases}$$



Kernel Function

- Different kernels can be used, see for example [Müller03]
- Choice of kernel affects stability, accuracy and speed of SPH methods
- Viscosity kernel: Laplacian is always positive!

$$W_{\text{viscosity}}(\mathbf{r}, h) = \frac{15}{2\pi h^3} \begin{cases} -\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1 & 0 \leq r \leq h \\ 0 & \text{otherwise.} \end{cases}$$



Time Integration

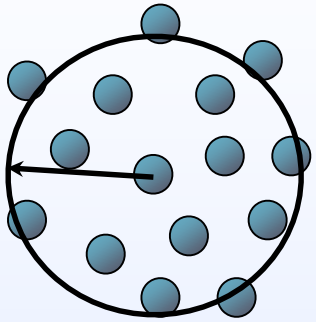
- Total force density: $\mathbf{f}_i = \mathbf{f}_i^{pressure} + \mathbf{f}_i^{viscosity} + \mathbf{f}_i^{gravity}$
- Acceleration: $\mathbf{a}_i = \frac{\mathbf{f}_i}{\rho_i}$

◆ Symplectic Euler scheme:

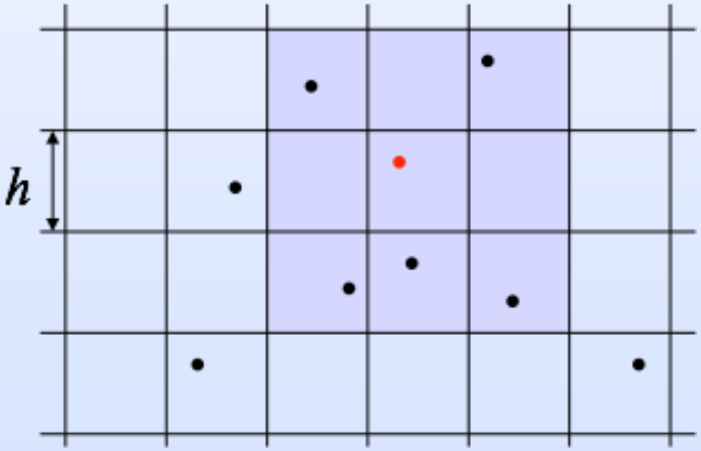
- New velocity: $\mathbf{u}_i(t+1) = \mathbf{u}_i(t) + \Delta t \frac{\mathbf{f}_i}{\rho_i}$
- New position: $\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \Delta t \mathbf{u}_i(t+1)$

Neighbor Search

- 3D: 30-40 neighbors per particle
  →Neighbor computation is most expensive part
  → we need fast data structures



- Domain partitioning into cells of size h
- Potential neighbors in 27 cells
- Create grid, insert particles, compute neighbors



How to get started with SPH...

- Start with simple implementation, worry about performance/optimizations once it works

- Define walls/boundary conditions geometrically → simple collision tests
  (if **x**[y] < 0 then **x**[y] = 0)

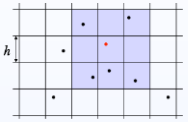
Debugging 1: Visualize particles / quantities (densities, velocities)

Debugging 2: Viscosity const large enough? Time step small enough?

Algorithm

SPH Algorithm

```
1  while animating do
2      for all i do
3          find neighborhoods N_i(t)
4      for all i do
5          compute density ρ_i(t)
6          compute pressure p_i(t)
7      for all i do
8          compute forces F^{p,v,g,ext}(t)
9      for all i do
10         compute new velocity
11         compute new position
```



$$\rho_i = \sum_j m_j W_{ij}$$

$$p_i = k(\rho_i - \rho_0)$$

$$\mathbf{f}_i^{pressure} = -\sum_j \frac{m_j}{\rho_j} \frac{p_i + p_j}{2} \nabla W_{ij}$$

$$\mathbf{f}_i^{viscosity} = \mu \sum_j \frac{m_j}{\rho_j} \mathbf{u}_j - \mathbf{u}_i \nabla^2 W_{ij}$$

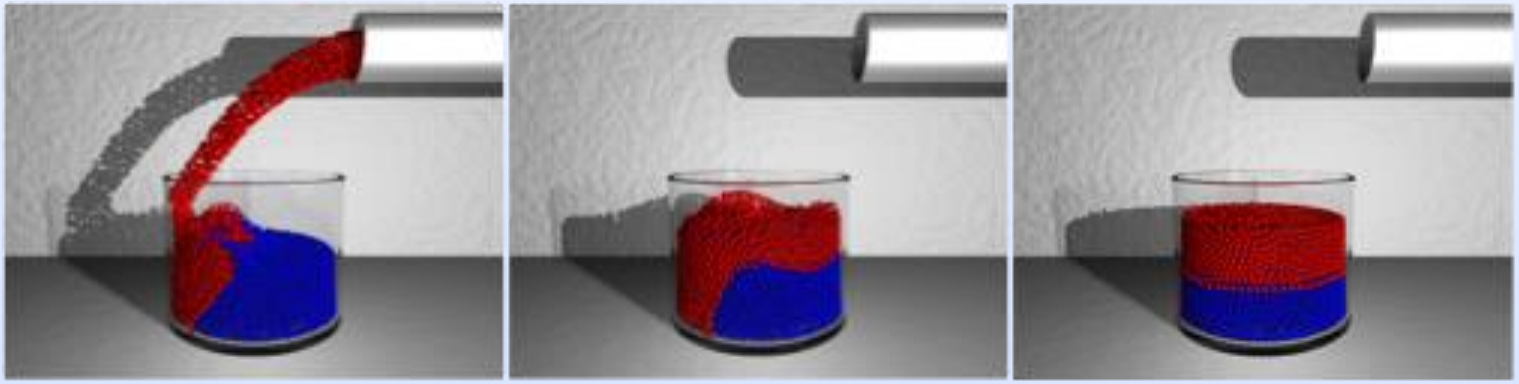
$$\mathbf{f}_i^{gravity} = \rho_i \mathbf{g}$$

$$\mathbf{u}_i(t+1) = \mathbf{u}_i(t) + \Delta t \frac{\mathbf{f}_i}{\rho_i}$$

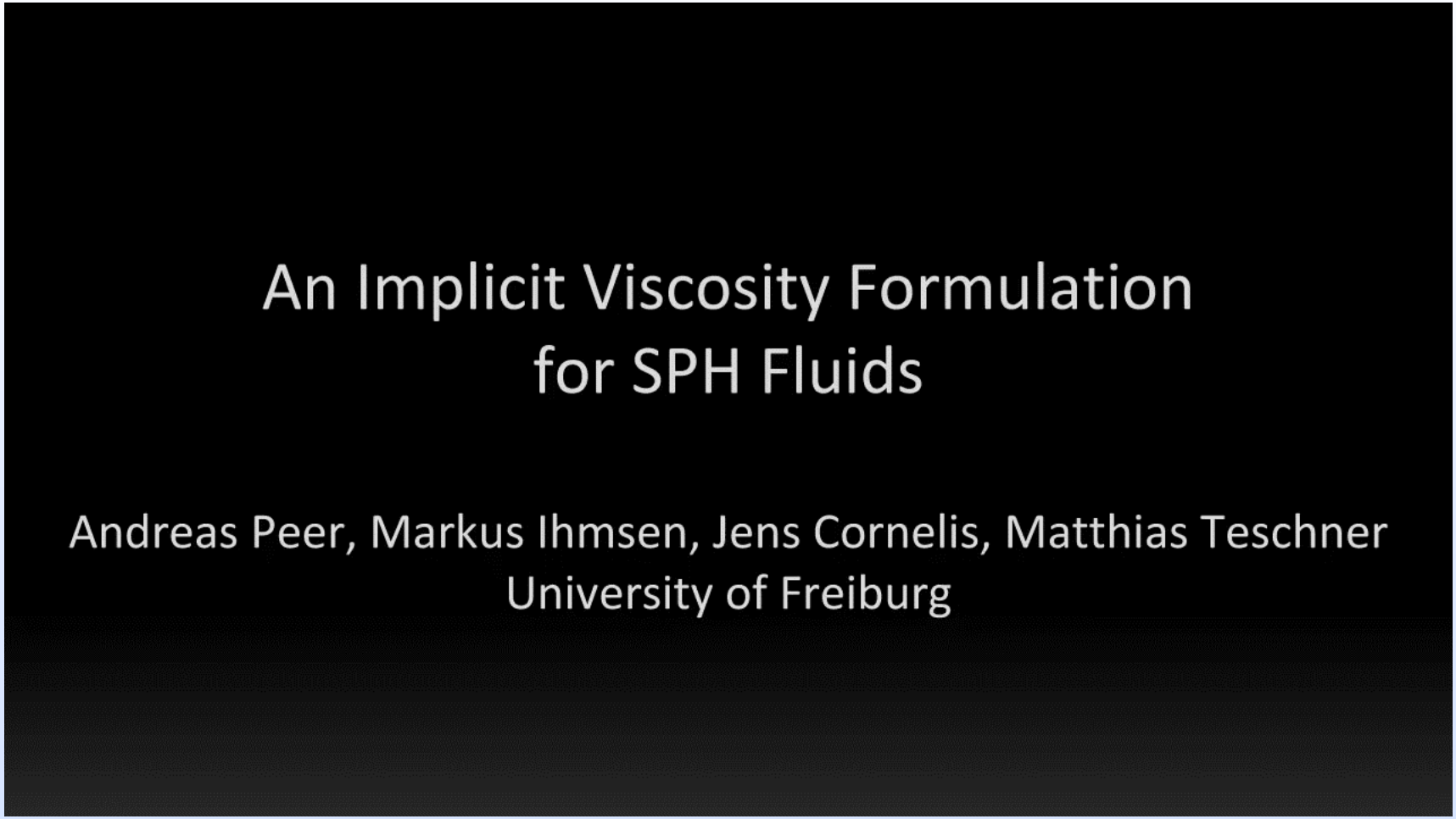
$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \Delta t \mathbf{u}_i(t+1)$$

Multiple Fluids

- Particles carry different attributes
  - Mass
  - Rest density
  - Viscosity coefficient
  - Gas constant (stiffness)
  - Color attributes
  - Temperature
- Buoyancy emerges from individual rest densities *(note:V1=V2)*

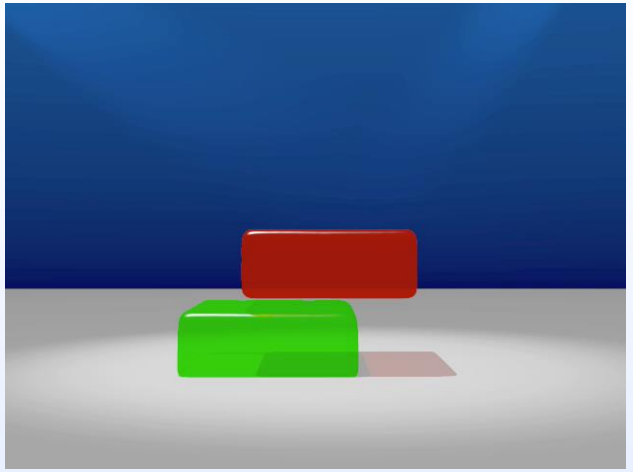


Fun with SPH


An Implicit Viscosity Formulation
for SPH Fluids
Andreas Peer, Markus Ihmsen, Jens Cornelis, Matthias Teschner
University of Freiburg

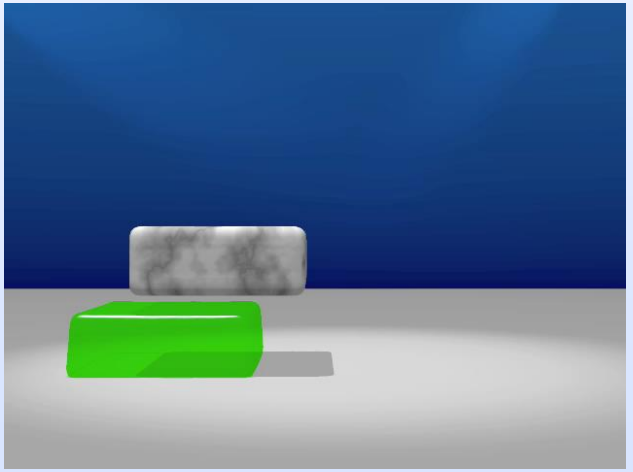
Solids with SPH

- Unified model for fluids, solids, elastic objects -> phase changes
- Displacement from undeformed shape
  -> strain, stress, elastic forces

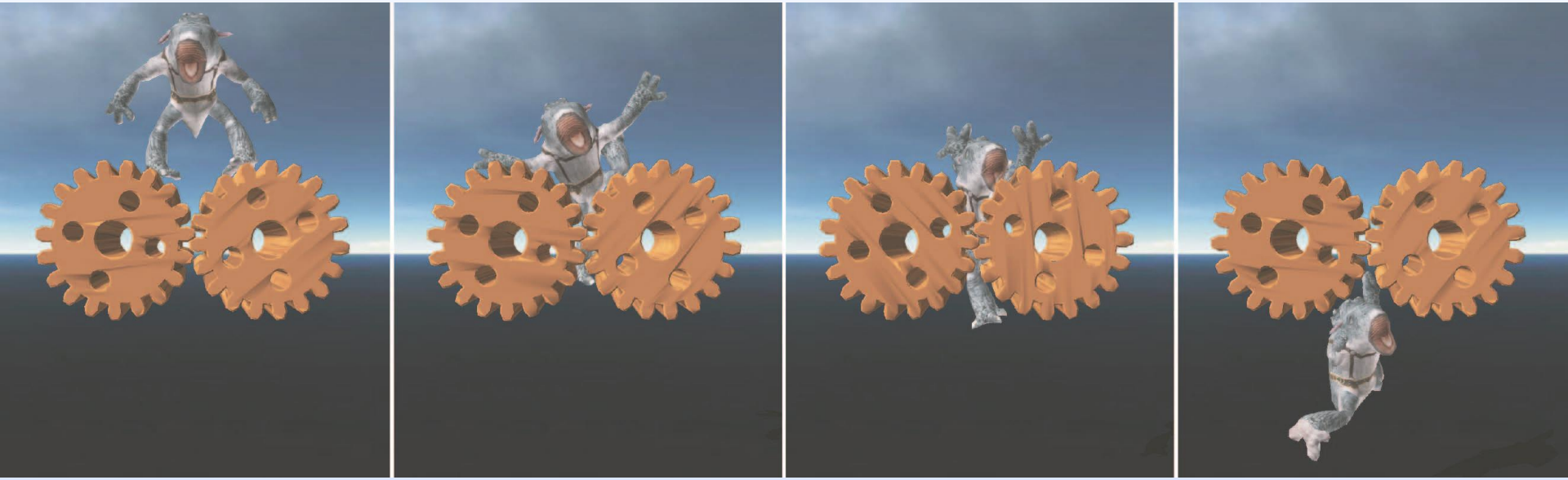


$$\mathbf{a}_i = \frac{1}{\rho_i}(\mathbf{f}_i^{pressure} + \mathbf{f}_i^{viscosity} + \mathbf{f}_i^{external})$$

$$\mathbf{a}_i = \frac{1}{\rho_i}(\mathbf{f}_i^{pressure} + \mathbf{f}_i^{viscosity} + \mathbf{f}_i^{external} + \mathbf{f}_i^{rigid} + \mathbf{f}_i^{elastic})$$



Position-Based Dynamics

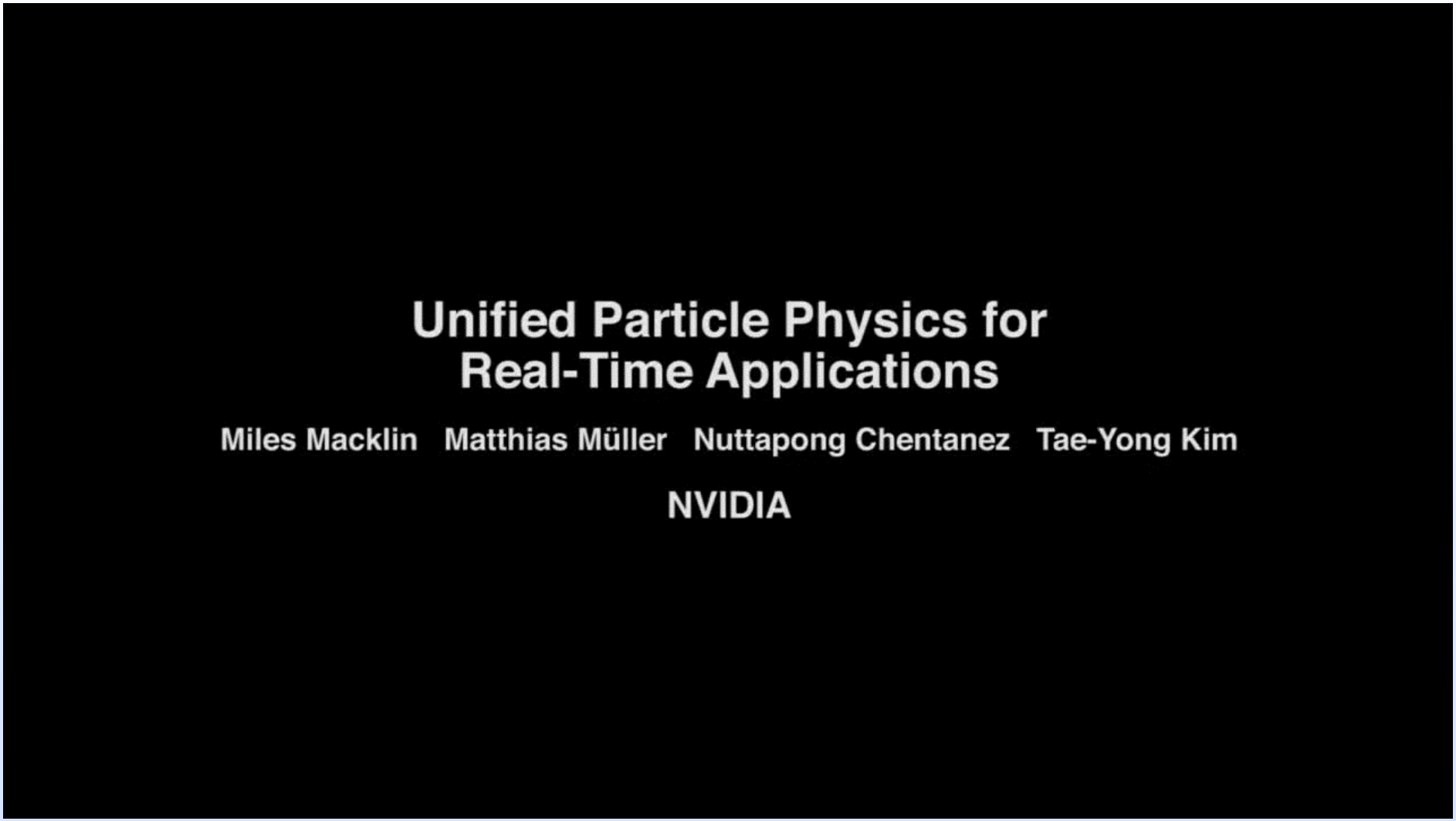


Position-Based Dynamics

- Physics-inspired*: everything is a set of particles connected by *constraints*
  - Replace forces & numerical integration by constraint projection

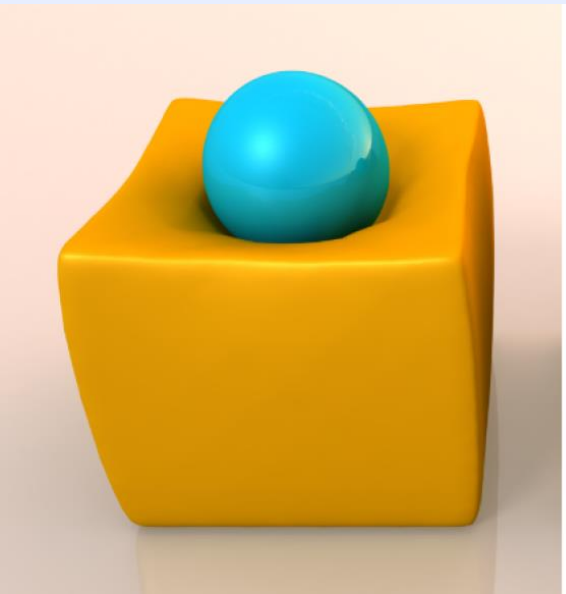
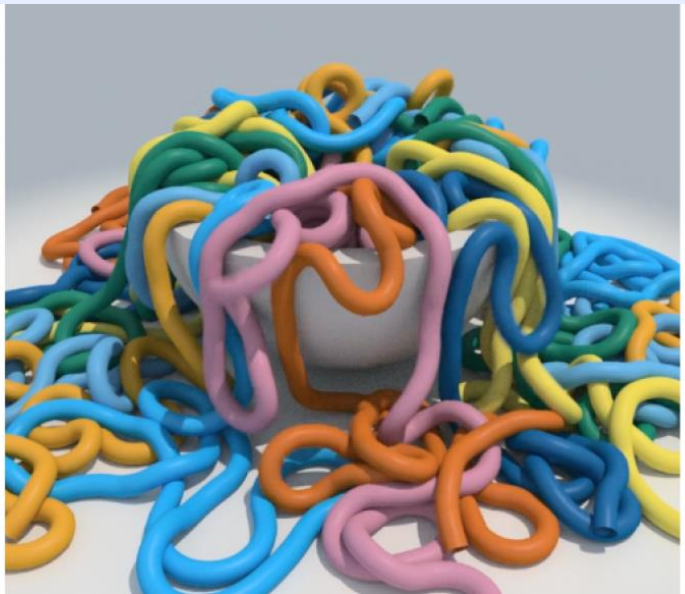
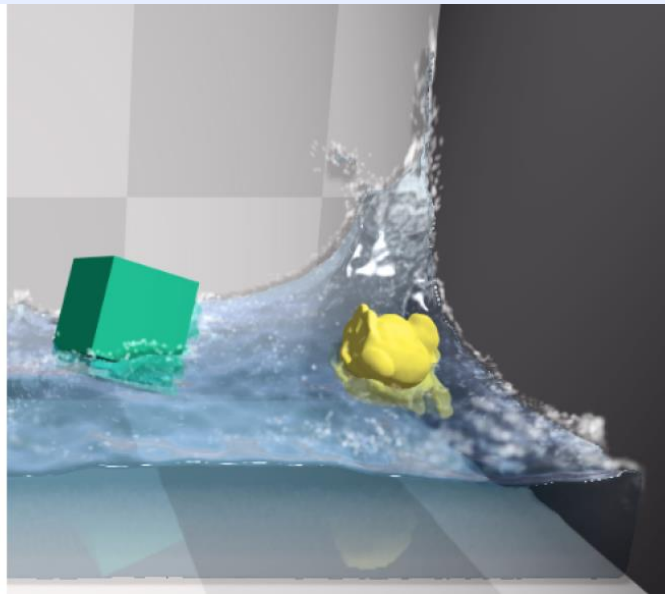
- Very fast, very stable, simple, plausible results
  - used in video games
    - Nvidia PhysX
    - T. Jakobsen, **Advanced Character Physics**, Game Developer Conference, 2001.
  - used in Autodesk's Maya: https://autodeskresearch.com/publications/nucleus

Position-Based Dynamics



Position-Based Dynamics

- Many references, but start here:
  - “Position-Based Simulation Methods in Computer Graphics”, Bender, Muller and Macklin, Eurographics 2015

Position-Based Dynamics

- Setup similar to mass-spring systems
  - Discretize using mass points: mass m_i, position $\mathbf{x_i}$, velocity $\mathbf{v_i}$
- Rather than forces, use constraints:
  - distance constraint, area preservation, etc

PBD: constraints

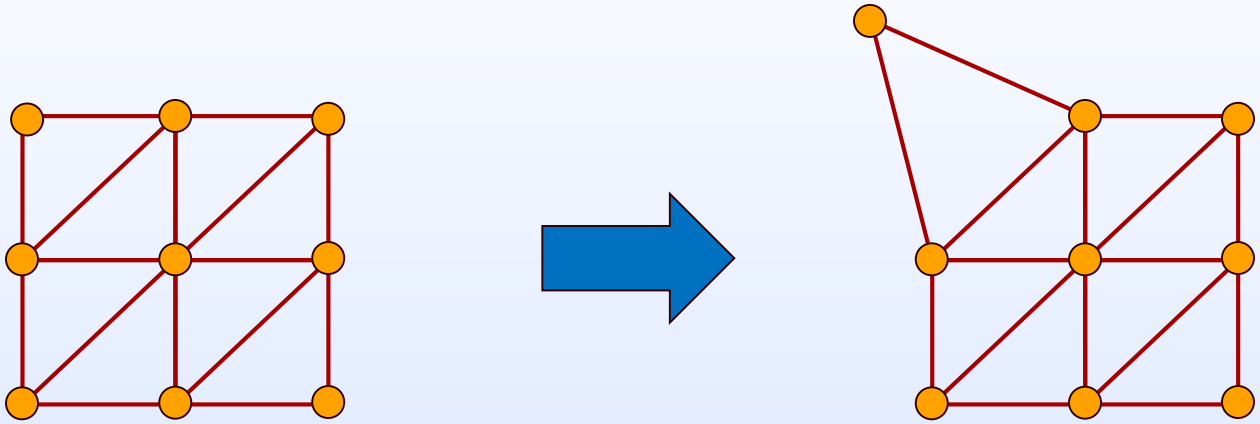
A constraint $j \in [1, \ldots, M]$ consists of

- a cardinality n_j,
- a function $C_j : \mathbb{R}^{3n_j} \rightarrow \mathbb{R}$,
- a set of indices $\{i_1, \ldots i_{n_j}\}$, $i_k \in [1, \ldots N]$,
- a stiffness parameter $k_j \in [0 \ldots 1]$ and
- a type of either *equality* or *inequality*.

PBD: main loop

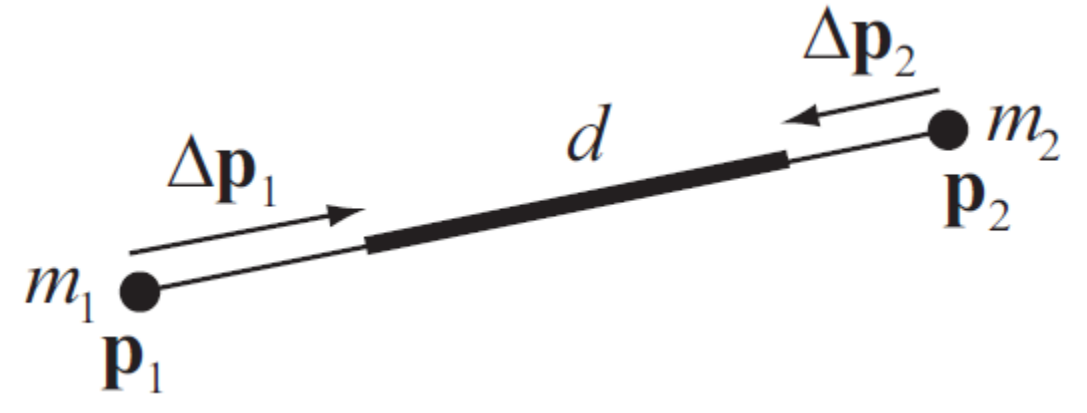
(1) **forall** vertices i
(2) initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
(3) **endfor**
(4) **loop**
(5) **forall** vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
(6) dampVelocities$(\mathbf{v}_1, \ldots, \mathbf{v}_N)$
(7) **forall** vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
(8) **forall** vertices i **do** generateCollisionConstraints$(\mathbf{x}_i \rightarrow \mathbf{p}_i)$
(9) **loop** solverIterations **times**
(10) projectConstraints$(C_1, \ldots, C_{M+M_{coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_N)$
(11) **endloop**
(12) **forall** vertices i
(13) $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
(14) $\mathbf{x}_i \leftarrow \mathbf{p}_i$
(15) **endfor**
(16) velocityUpdate$(\mathbf{v}_1, \ldots, \mathbf{v}_N)$
(17) **endloop**

PBD: constraint projection



$$C(\boldsymbol{p_i}, \boldsymbol{p_j}) = |\boldsymbol{p_i} - \boldsymbol{p_j}| - \mathrm{d}$$

PBD: constraint projection



PBD: constraint projection

Taylor expansion (per constraint):

$$C(\mathbf{p}+\Delta\mathbf{p}) \approx C(\mathbf{p}) + \nabla_{\mathbf{p}} C(\mathbf{p}) \cdot \Delta\mathbf{p} = 0$$

Step must be in direction of constraint Jacobian:

$$\Delta\mathbf{p} = \lambda \nabla_{\mathbf{p}} C(\mathbf{p})$$

Putting it all together:

$$\Delta\mathbf{p} = -\frac{C(\mathbf{p})}{|\nabla_{\mathbf{p}} C(\mathbf{p})|^2} \nabla_{\mathbf{p}} C(\mathbf{p})$$

PBD: constraint projection

$$\Delta \mathbf{p}_i = -s \nabla_{\mathbf{p}_i} C(\mathbf{p}_1, \ldots, \mathbf{p}_n) \qquad s = \frac{C(\mathbf{p}_1, \ldots, \mathbf{p}_n)}{\sum_j |\nabla_{\mathbf{p}_j} C(\mathbf{p}_1, \ldots, \mathbf{p}_n)|^2}$$

Is momentum conserved?

PBD: constraint projection

Conservation of linear momentum:

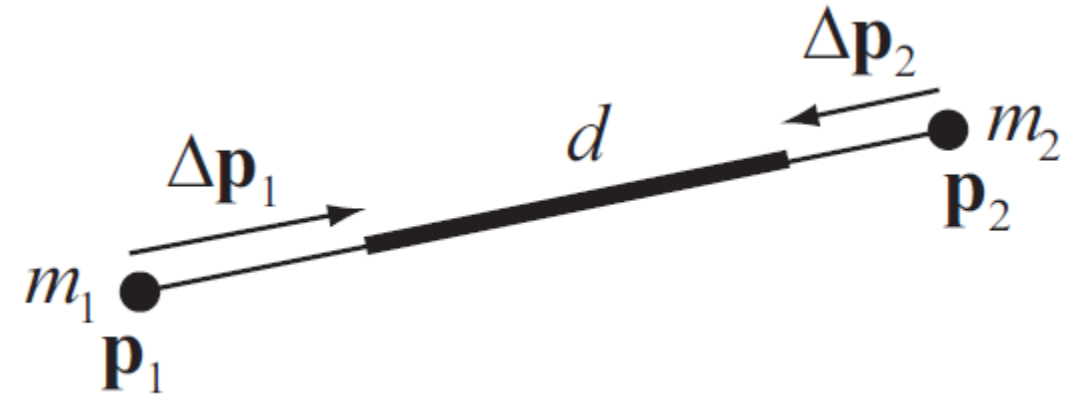
$$\sum_i m_i \Delta \mathbf{p}_i = \mathbf{0}$$

Scale updates by inverse mass $w_i = 1/mi$

$$\Delta \mathbf{p}_i = -s\, w_i \nabla_{\mathbf{p}_i} C(\mathbf{p}_1, \ldots, \mathbf{p}_n)$$

$$s = \frac{C(\mathbf{p}_1, \ldots, \mathbf{p}_n)}{\sum_j w_j |\nabla_{\mathbf{p}_j} C(\mathbf{p}_1, \ldots, \mathbf{p}_n)|^2}$$

PBD: constraint projection



PBD: constraint projection

$$C(\mathbf{p}_1, \mathbf{p}_2) = |\mathbf{p}_1 - \mathbf{p}_2| - d$$

$$\nabla_{\mathbf{p}_1} C(\mathbf{p}_1, \mathbf{p}_2) = \mathbf{n}$$

$$\nabla_{\mathbf{p}_2} C(\mathbf{p}_1, \mathbf{p}_2) = -\mathbf{n}$$

$$\mathbf{n} = \frac{\mathbf{p}_1 - \mathbf{p}_2}{|\mathbf{p}_1 - \mathbf{p}_2|}$$

$$s = \frac{|\mathbf{p}_1 - \mathbf{p}_2| - d}{w_1 + w_2}$$

$$\Delta\mathbf{p}_1 = -\frac{w_1}{w_1 + w_2}(|\mathbf{p}_1 - \mathbf{p}_2| - d)\frac{\mathbf{p}_1 - \mathbf{p}_2}{|\mathbf{p}_1 - \mathbf{p}_2|}$$

$$\Delta\mathbf{p}_2 = +\frac{w_2}{w_1 + w_2}(|\mathbf{p}_1 - \mathbf{p}_2| - d)\frac{\mathbf{p}_1 - \mathbf{p}_2}{|\mathbf{p}_1 - \mathbf{p}_2|}$$

PBD: main loop

(1) **forall** vertices i
(2) initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
(3) **endfor**
(4) **loop**
(5) **forall** vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
(6) dampVelocities$(\mathbf{v}_1, \ldots, \mathbf{v}_N)$
(7) **forall** vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
(8) **forall** vertices i **do** generateCollisionConstraints$(\mathbf{x}_i \rightarrow \mathbf{p}_i)$
(9) **loop** solverIterations **times**
(10) projectConstraints$(C_1, \ldots, C_{M+M_{coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_N)$
(11) **endloop**
(12) **forall** vertices i
(13) $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
(14) $\mathbf{x}_i \leftarrow \mathbf{p}_i$
(15) **endfor**
(16) velocityUpdate$(\mathbf{v}_1, \ldots, \mathbf{v}_N)$
(17) **endloop**

Collision constraints

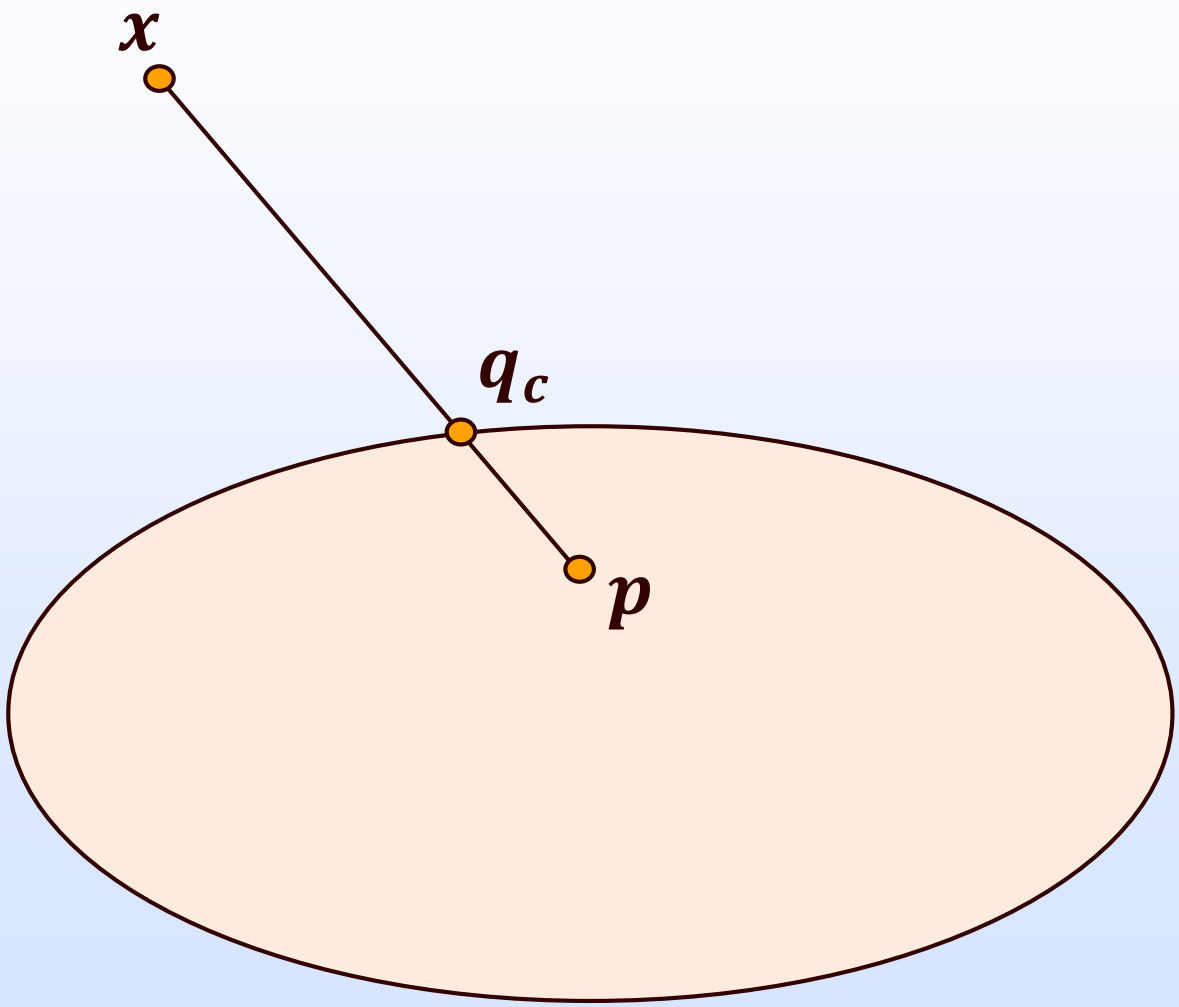
Assume: collisions already detected

- Simple objects/particles: easy

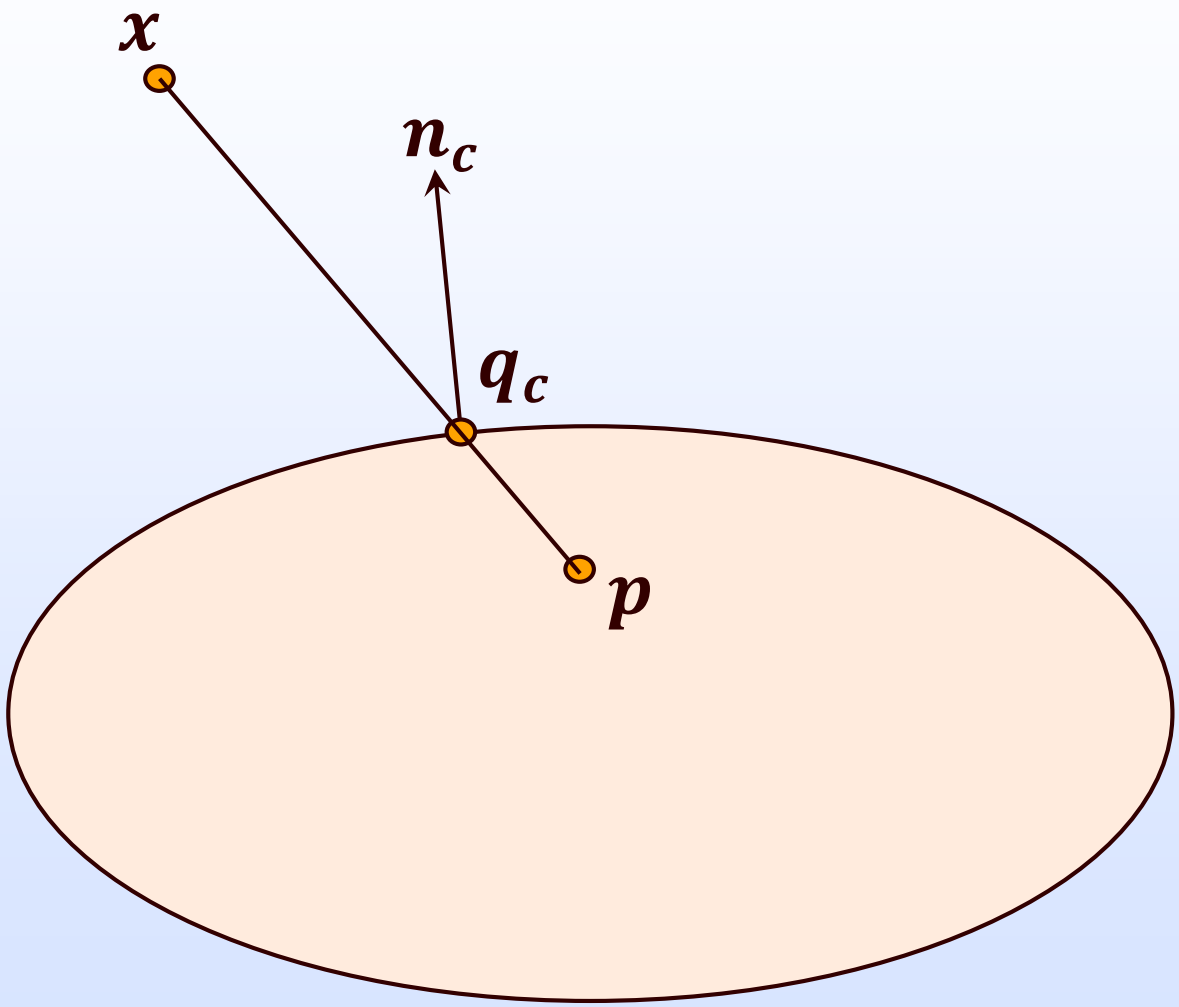
Collision response:

- Unilateral constraints (inequality)
- Potentially changing at every time-step

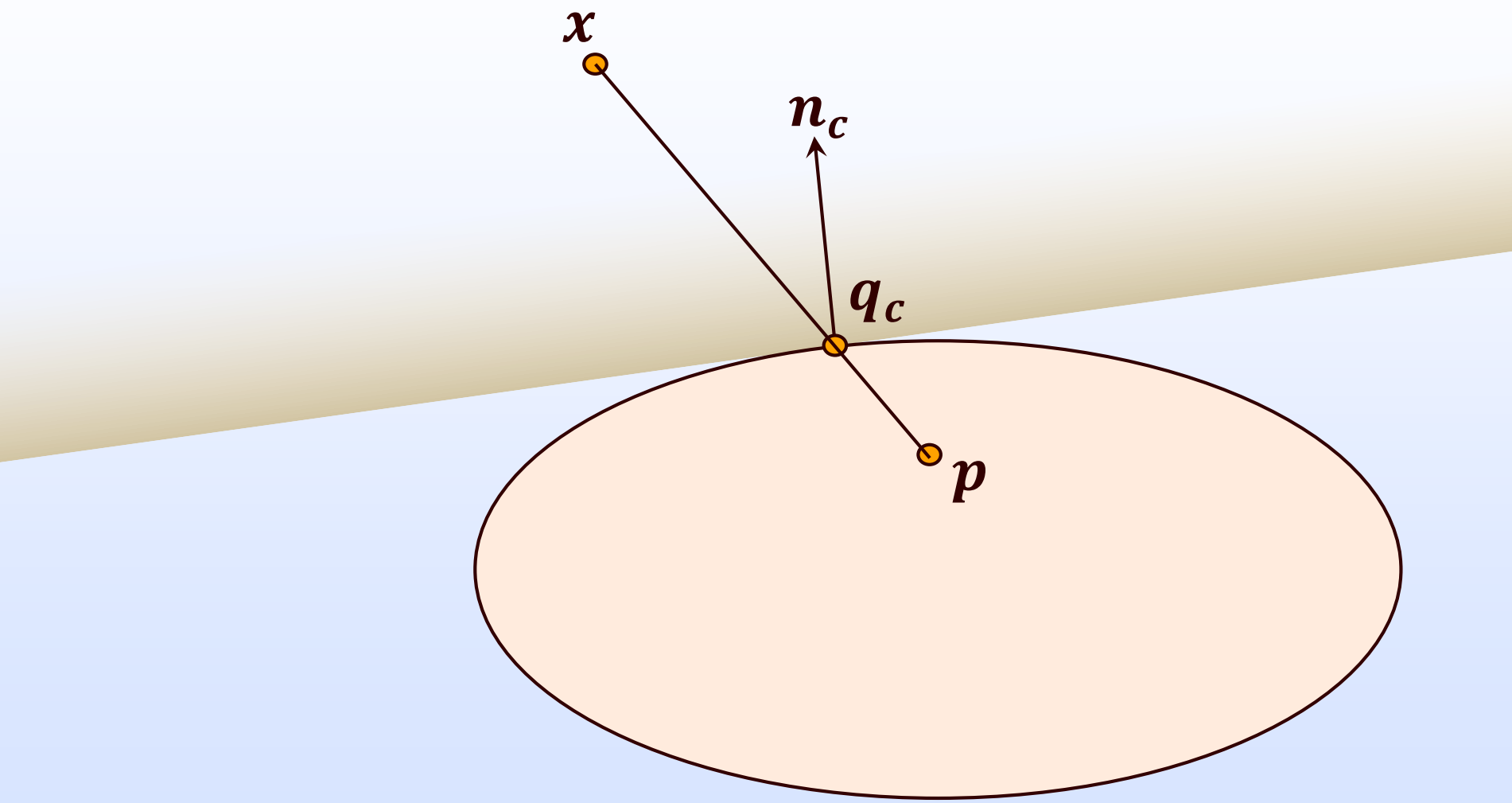
Collision constraints



Collision constraints



Collision constraints



Collision constraint

$$C(\mathbf{p}) = (\mathbf{p} - \mathbf{q}_c) \cdot \mathbf{n}_c$$

Unilateral: $C(\boldsymbol{p}) \geq 0$

- Project only if violated

PBD: main loop

(1) **forall** vertices i
(2) initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
(3) **endfor**
(4) **loop**
(5) **forall** vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
(6) dampVelocities$(\mathbf{v}_1, \ldots, \mathbf{v}_N)$
(7) **forall** vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
(8) **forall** vertices i **do** generateCollisionConstraints$(\mathbf{x}_i \rightarrow \mathbf{p}_i)$
(9) **loop** solverIterations **times**
(10) projectConstraints$(C_1, \ldots, C_{M+M_{coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_N)$
(11) **endloop**
(12) **forall** vertices i
(13) $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
(14) $\mathbf{x}_i \leftarrow \mathbf{p}_i$
(15) **endfor**
(16) velocityUpdate$(\mathbf{v}_1, \ldots, \mathbf{v}_N)$
(17) **endloop**

PBD: main loop

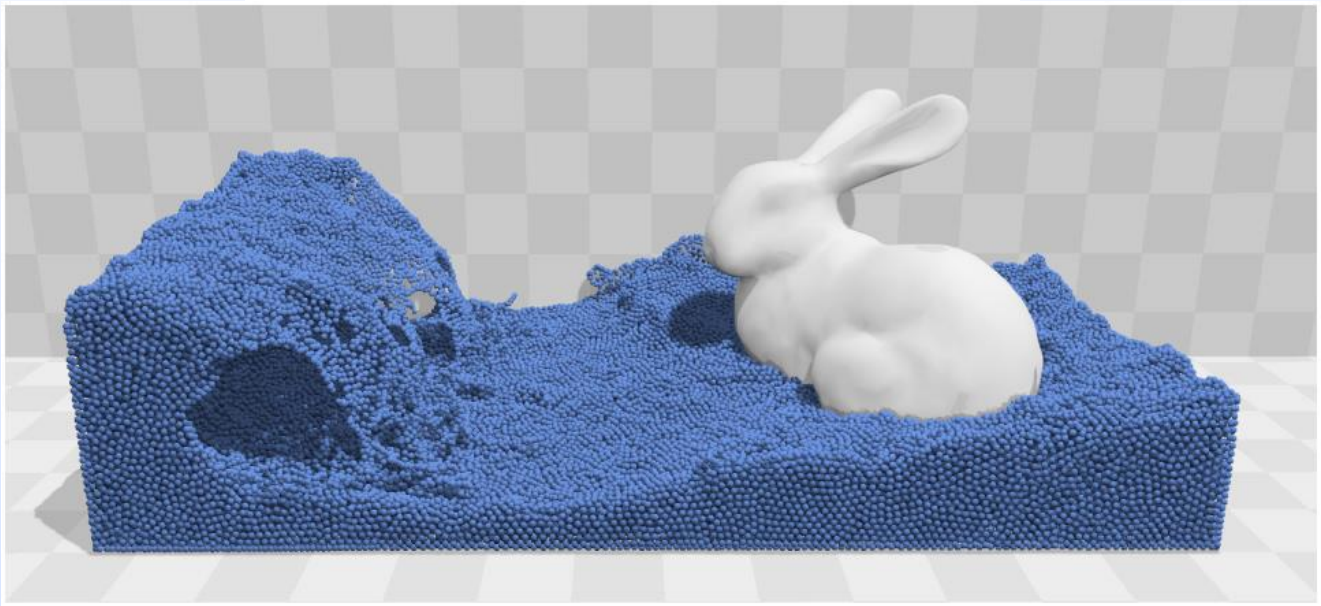
- Not a traditional time-stepping scheme
  - Predictor-Corrector approach
- Stability due to **constraint projection** which acts directly on positions
  - note: no internal forces!

PBD: can you see any downsides to this approach?

(1) **forall** vertices i
(2) initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
(3) **endfor**
(4) **loop**
(5) **forall** vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
(6) dampVelocities$(\mathbf{v}_1, \ldots, \mathbf{v}_N)$
(7) **forall** vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
(8) **forall** vertices i **do** generateCollisionConstraints$(\mathbf{x}_i \rightarrow \mathbf{p}_i)$
(9) **loop** solverIterations **times**
(10) projectConstraints$(C_1, \ldots, C_{M+M_{coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_N)$
(11) **endloop**
(12) **forall** vertices i
(13) $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
(14) $\mathbf{x}_i \leftarrow \mathbf{p}_i$
(15) **endfor**
(16) velocityUpdate$(\mathbf{v}_1, \ldots, \mathbf{v}_N)$
(17) **endloop**

Other types of constraints?

$$C_i(\mathbf{p}_1, \ldots, \mathbf{p}_n) = \frac{\rho_i}{\rho_0} - 1,$$



- Position-based Fluids

  “Position Based Fluids”, Macklin & Muller, 2014

Assignment 2

- Out today
- Due on March 2nd @ midnight

Reminder

- Project Ideas
  - Brief descriptions due today (or plan to talk to us this week)
    - Team info, topic, etc.