SPH Fluids in Computer Graphics

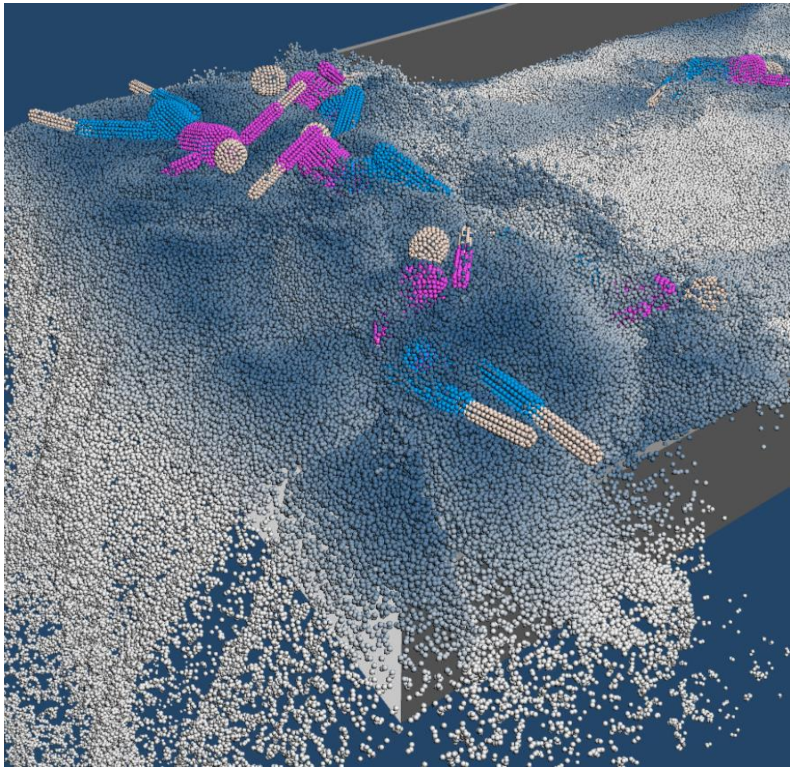
Markus Ihmsen - University of Freiburg
Jens Orthmann - University of Siegen
Barbara Solenthaler - ETH Zürich
Andreas Kolb - University of Siegen
Matthias Teschner - University of Freiburg

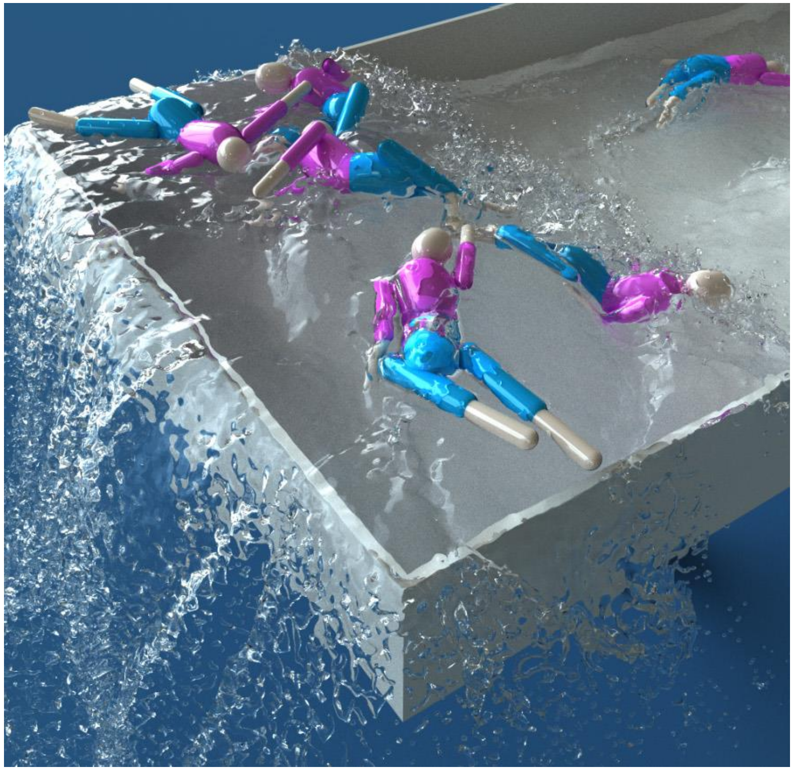
EG Strasbourg 2014

Topics / Research Challenges

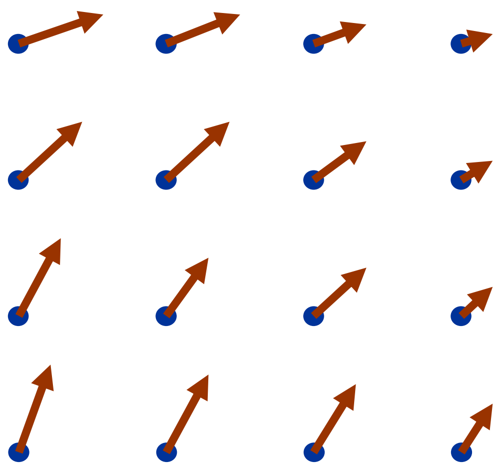
- **SPH fluid solver**
- Neighborhood query
- Incompressibility / pressure computation
- Boundary handling
- Multiple phases
- Multi-resolution
- Surface reconstruction and rendering

Concept

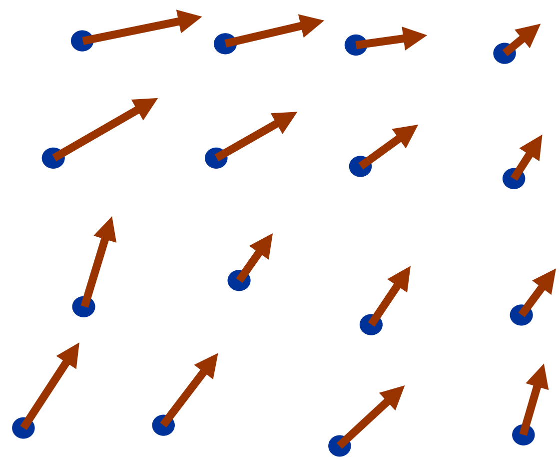




Lagrangian Approach



$$\mathbf{v}(x, y, z, t)$$



$$\mathbf{v}(x + \Delta t \cdot u, y + \Delta t \cdot v, z + \Delta t \cdot w, t + \Delta t)$$

- flow properties are considered at irregular positions $\mathbf{x}_i$
- particles have volume V_i , mass m_i, density ρ_i, pressure p_i
- particles move with their velocity $\mathbf{v}_i$

Momentum Equation

- Navier-Stokes

$$\underbrace{\frac{D\mathbf{v}_i}{Dt}}_{\substack{\text{time rate of change}\\ \text{of the velocity of a}\\ \text{moving fluid element}}} = -\frac{1}{\rho_i}\nabla p_i + \nu\nabla^2\mathbf{v}_i + \frac{\mathbf{F}_i^{other}}{m_i}$$

- Lagrangian form with **advected** fluid samples / particles $\mathbf{x}_i$

$$\frac{D\mathbf{v}_i}{Dt} = \frac{d\mathbf{v}_i}{dt} \qquad \frac{d\mathbf{x}_i}{dt} = \mathbf{v}_i$$

- Eulerian form with **fixed** fluid samples $\mathbf{x}_i$

$$\frac{D\mathbf{v}_i}{Dt} = \frac{\partial\mathbf{v}_i}{\partial t} + \mathbf{v}_i \cdot \nabla\mathbf{v}_i$$

Accounts for the missing movement of the sample

Momentum Equation

- Navier-Stokes

$$\frac{D\mathbf{v}_i}{Dt} = \underbrace{\mathbf{a}_i}_{\substack{\text{acceleration}\\ \text{of a particle}}} = \underbrace{-\frac{1}{\rho_i}\nabla p_i}_{\substack{\text{pressure force}\\ \text{per particle mass}}} + \underbrace{\nu \nabla^2 \mathbf{v}_i}_{\substack{\text{viscosity force}\\ \text{per particle mass}}} + \underbrace{\frac{\mathbf{F}_i^{other}}{m_i}}_{\substack{\text{other forces}\\ \text{per particle mass}}}$$

- Pressure
  - Acceleration due to pressure differences
  - Preserves the fluid volume / density
- Viscosity
  - Acceleration due to friction between particles with different velocities
- Other
  - Gravity
  - Acceleration due to boundary handling

Smoothed Particle Hydrodynamics SPH

- Interpolation method
  - Proposed by Gingold and Monaghan (1977) and Lucy (1977)
- Can be used for sets of arbitrary samples to
  - Interpolate quantities
  - Approximate spatial derivatives
- SPH in a Lagrangian fluid simulation
  - Fluid is represented with a set of particles / samples
  - SPH is used to discretize $\mathbf{a}_i = -\frac{1}{\rho_i}\nabla p_i + \nu \nabla^2 \mathbf{v}_i + \mathbf{g}$

Interpolation with SPH

- Quantity A_i at arbitrary position $\mathbf{x}_i$ is approximately computed with a set of known quantities A_j at $\mathbf{x}_j$ sample positions

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

  - $\mathbf{x}_i$ is not necessarily a sample position
  - If $\mathbf{x}_i$ is a sample position, it contributes to the sum

- W_{ij} is a kernel function that weights the contributions of sample positions $\mathbf{x}_j$ according to their distance to $\mathbf{x}_i$

  $$W_{ij} = W\left(\frac{\|\mathbf{x}_i - \mathbf{x}_j\|}{h}\right)$$

  - h is the so-called smoothing length
  - h is not necessarily the particle distance or the size of the compact support of W_{ij}

Kernel Function

- Close to a Gaussian, but with compact support
- Number of neighboring particles considered in the interpolation
  - Depends on dimensionality, kernel support, particle spacing / mass
  - In 3D, 30-40 neighboring particles are recommended
  - E.g., cubic spline, support $2h$, particle spacing h
  - Trade-off between performance and interpolation accuracy

Spatial Derivatives with SPH

- Original approximations

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

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

- Currently preferred approximations

$$\nabla A_i = \rho_i \sum_j m_j \left(\frac{A_i}{\rho_i^2} + \frac{A_j}{\rho_j^2} \right) \nabla W_{ij}$$

Preserves linear and angular momentum

$$\nabla \cdot \mathbf{A}_i = -\frac{1}{\rho_i} \sum_j m_j \mathbf{A}_{ij} \nabla W_{ij}$$

Sampling independent

$$\nabla^2 A_i = 2 \sum_j \frac{m_j}{\rho_j} A_{ij} \frac{\mathbf{x}_{ij} \cdot \nabla W_{ij}}{\mathbf{x}_{ij} \cdot \mathbf{x}_{ij} + 0.01 h^2}$$

More robust as it avoids the second derivative of W

$$A_{ij} = A_i - A_j \quad \mathbf{A}_{ij} = \mathbf{A}_i - \mathbf{A}_j \quad \mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$$

Momentum Equation – SPH Discretization

- Navier-Stokes

$$\underbrace{\mathbf{a}_i}_{\substack{\text{acceleration}\\ \text{of a particle}}} = \underbrace{-\frac{1}{\rho_i}\nabla p_i}_{\substack{\text{pressure force}\\ \text{per particle mass}}} + \underbrace{\nu\nabla^2\mathbf{v}_i}_{\substack{\text{viscosity force}\\ \text{per particle mass}}} + \underbrace{\frac{\mathbf{F}_i^{other}}{m_i}}_{\substack{\text{other forces}\\ \text{per particle mass}}}$$

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

- Pressure acceleration $-\frac{\nabla p_i}{\rho_i} = -\sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right) \nabla W_{ij}$

- Viscosity acceleration $\nu\nabla^2\mathbf{v}_i = 2\nu \sum_j \frac{m_j}{\rho_j} \mathbf{v}_{ij} \frac{\mathbf{x}_{ij}\cdot\nabla W_{ij}}{\mathbf{x}_{ij}\cdot\mathbf{x}_{ij}+0.01h^2}$

Simple SPH Fluid Solver

- Find all neighbors j of particle i
- Compute density ρ_i
- Compute pressure p_i
- Compute accelerations
- Update velocity and position

Topics / Research Challenges

- SPH fluid solver
- **Neighborhood query**
- Incompressibility / pressure computation
- Boundary handling
- Multiple phases
- Multi-resolution
- Surface reconstruction and rendering

Neighbor Search

- For the computation of SPH sums, each particle needs to know 30-40 neighbors in each simulation step
- Current scenarios
  - Up to 100 million fluid particles
  - Up to 3 billion neighbors per simulation step
- Efficient construction and processing of dynamically changing neighbor sets is essential

Characteristics

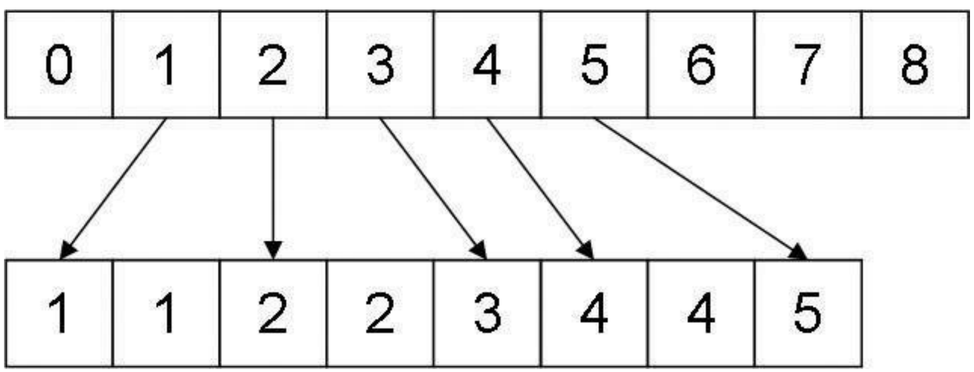
- SPH computes sums
  - Dynamically changing sets of neighboring particles
  - Temporal coherence
- Spatial data structures accelerate the neighbor search
  - Fast query
  - Fast generation - each simulation step
  - sparsely, non-uniformly filled simulation domain
- Similarities to collision detection and intersection tests in raytracing
  - However, cells adjacent to the cell of a particle have to be accessed

Characteristics

- Uniform grid
  - E.g., [Mueller03, Harada07, Green08, Goswami11, Ihmsen11, Macklin13]
  - Generally preferred - construction in O(n), access in O(1)
- Hierarchical data structures
  - E.g., [Vermuri98, Keiser06, Adams07]
  - Less efficient - construction in O(n log n), access in O(log n)
- Verlet lists
  - E.g., [Verlet67, Hieber07]
  - Potential neighbors computed within larger distance than actual support
  - Potential neighbors updated every n-th simulation step
  - Memory-intensive and slow

Index Sort – Uniform Grid

- Cell index $c = k + l \cdot K + m \cdot K \cdot L$ is computed for a particle
  - K and L denote the number of cells in x and y direction
- Particles are sorted with respect to their cell index
  - e.g., radix sort, O(n)
- Each grid cell (k, l, m) stores a reference to the first particle in the sorted list



uniform grid

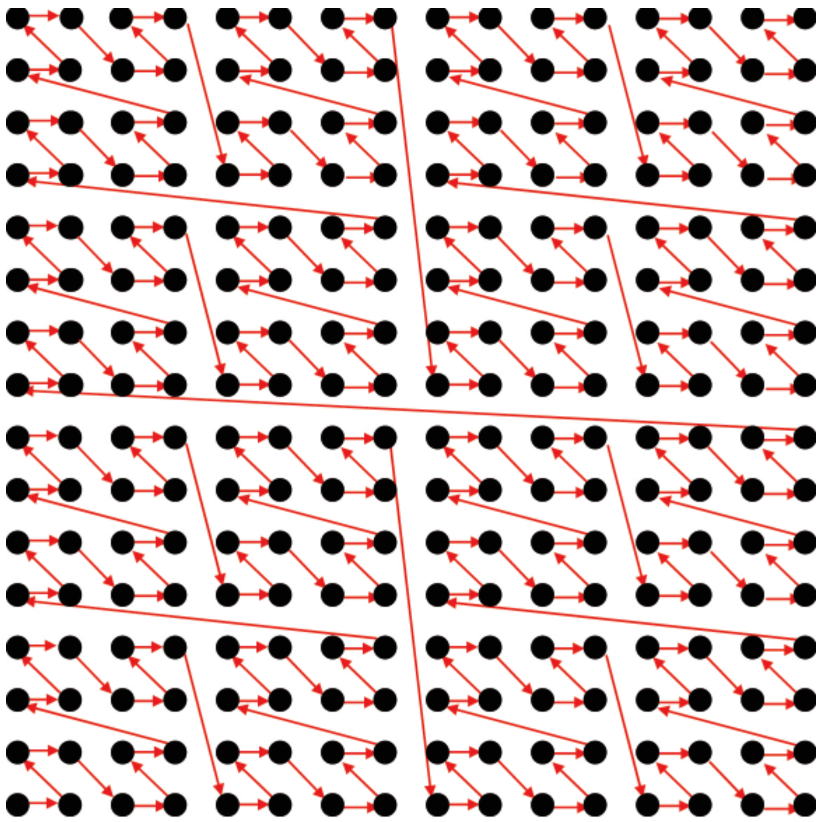
sorted particles with their cell indices

Index Sort – Query

- Sorted particle array is queried
- Particles in the same cell are queried
- References to particles of adjacent cells are obtained from the references stored in the uniform grid
- Cache-hit rate
  - Particles in the same cell are close in memory
  - Particles of neighboring cells are not necessarily close in memory

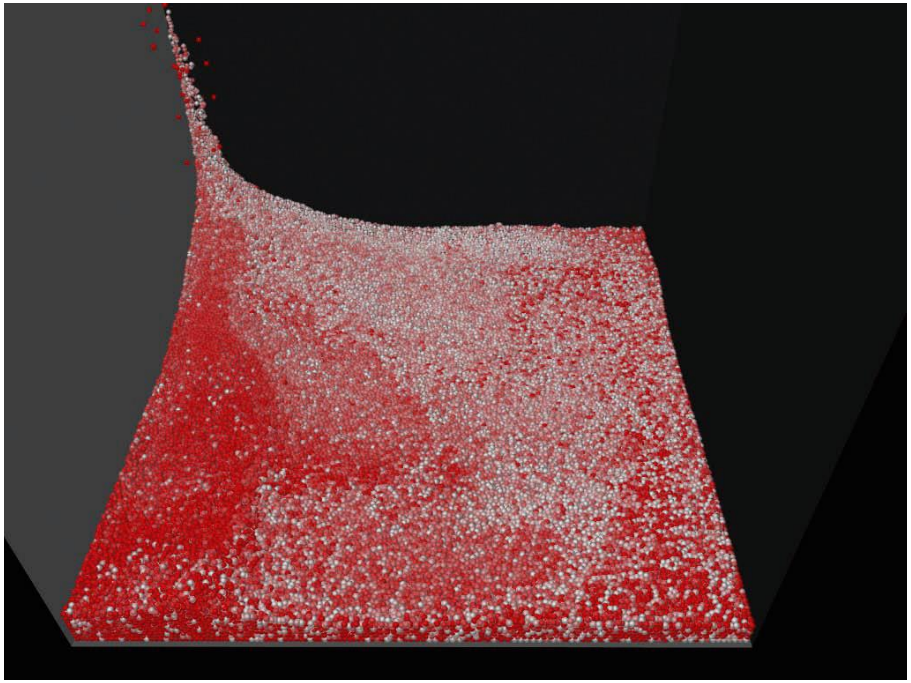
Z-Index Sort

- Particles are sorted with respect to z-curve index
- Improved cache-hit rate
  - Particles in adjacent cells are close in memory
- Efficient computation of z-curve indices possible

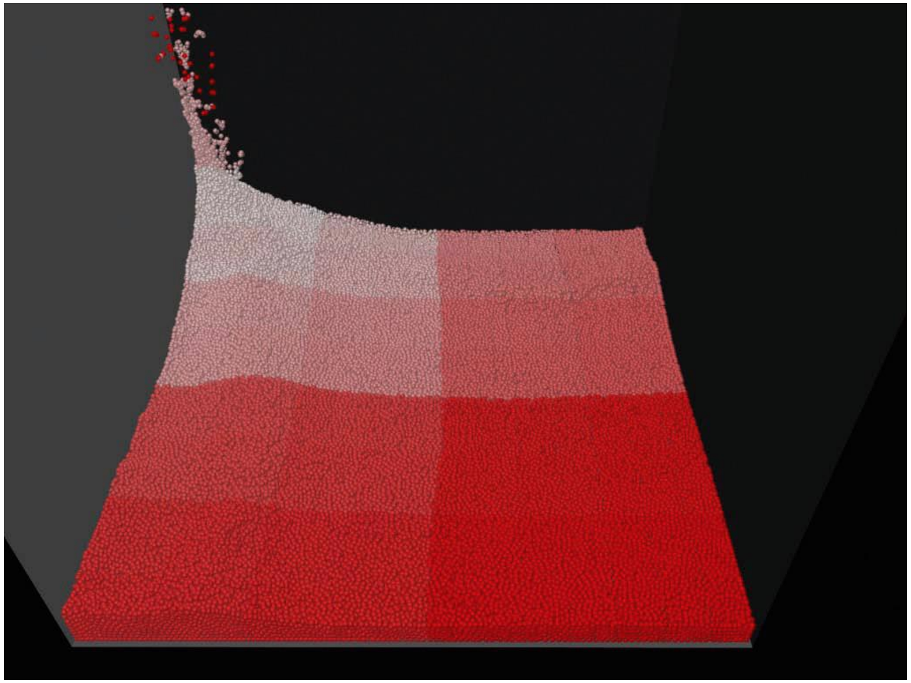


z-curve

Z-Index Sort - Reordering



Particles colored according to their location in memory



Spatial compactness is enforced using a z-curve

Topics / Research Challenges

- SPH fluid solver
- Neighborhood query
- **Incompressibility / pressure computation**
- Boundary handling
- Multiple phases
- Multi-resolution
- Surface reconstruction and rendering

Pressure Computation

- Role of pressure forces
  - Counteracts volume compression
  - Acceleration due to pressure differences

- Incompressibility
  - Is essential for a realistic fluid behavior
  - Inappropriate compression leads, e.g., to oscillations at the free surface
  - Is computationally expensive:
    - Simple computations require small time steps
    - Large time steps require complex computations

Pressure Computation - Models

- Non-iterative state-equation-based
  - Compressible [Müller03]
  - Weakly-compressible [Becker07]
- Iterative state-equation based
  - PCISPH [Solenthaler09]
  - Local Poisson SPH [He12]
  - PBF [Macklin13]
- Pressure projection
  - Divergence free [Cummins99]
  - Density invariant [Shao03]
  - IISPH [Ihmsen13]

State Equations (EOS, SESPH)

- Pressure is locally computed from density, e.g.,
  - Compressible SPH $p_i = k(\frac{\rho_i}{\rho_0} - 1)$
  - Weakly compressible SPH $p_i = k_1\left(\left(\frac{\rho_i}{\rho_0}\right)^{k_2} - 1\right)$
  - Stiffness constants k are user-defined
- Penalty approach
  - Current density fluctuations result in density gradients
  - Density gradients result in pressure gradients
  - Pressure gradients result in pressure force from high to low pressure
- Properties
  - Fast computation, but small time steps
  - Stiffness constant govern compressibility
  - Stiffness restricts the time step (scenario dependent)

Non-iterative EOS Solver (SESPH)

for all *particle i* **do**
  find neighbors j
for all *particle i* **do**
  $\rho_i = \sum_j m_j W_{ij}$
  compute p_i from ρ_i with a state equation
for all *particle i* **do**
  $\mathbf{F}_i^{pressure} = -\frac{m_i}{\rho_i} \nabla p_i$
  $\mathbf{F}_i^{viscosity} = m_i \nu \nabla^2 \mathbf{v}_i$
  $\mathbf{F}_i^{other} = m_i \mathbf{g}$
  $\mathbf{F}_i(t) = \mathbf{F}_i^{pressure} + \mathbf{F}_i^{viscosity} + \mathbf{F}_i^{other}$
for all *particle i* **do**
  $\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \Delta t \mathbf{F}_i(t) / m_i$
  $\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t)$

SESPH with Splitting

- Compute pressure after advecting particles with non-pressure forces
- Splitting concept
  - Compute all non-pressure forces $\mathbf{F}_i^{nonp}(t)$
  - Compute intermediate velocity $\mathbf{v}_i^* = \mathbf{v}_i(t) + \Delta t \frac{\mathbf{F}_i^{nonp}}{m_i}$
  - Compute intermediate position $\mathbf{x}_i^* = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i^*$
  - Compute intermediate density $\rho_i^*(\mathbf{x}_i^*)$
  - Compute pressure p_i from intermediate density ρ_i^* using an EOS
  - Compute final velocity
- Motivation
  - Consider competing forces $\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i^* - \Delta t \frac{1}{\rho_i^*} \nabla p_i$
  - Take (positive or negative) effects of non-pressure forces into account when computing the pressure forces

SESPH with Splitting

for all *particle i* **do**
    find neighbors j
for all *particle i* **do**
    $\mathbf{F}_i^{viscosiy} = m_i \nu \nabla^2 \mathbf{v}_i$
    $\mathbf{F}_i^{other} = m_i \mathbf{g}$
    $\mathbf{v}_i^* = \mathbf{v}_i(t) + \Delta t \frac{\mathbf{F}_i^{viscosity} + \mathbf{F}_i^{other}}{m_i}$
for all *particle i* **do**
    $\rho_i^* = \sum_j m_j W_{ij} + \Delta t \sum_j (\mathbf{v}_i^* - \mathbf{v}_j^*) \nabla W_{ij}$
    compute p_i using ρ_i^*
for all *particle i* **do**
    $\mathbf{F}_i^{pressure} = -\frac{m_i}{\rho_i^*} \nabla p_i$
for all *particle i* **do**
    $\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i^* + \Delta t \mathbf{F}_i^{pressure} / m_i$
    $\mathbf{x}_i(t+\Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t+\Delta t)$

- follows from the continuity equation
- avoids neighbor search

Iterative SESPH with Splitting

- Pressure forces are iteratively accumulated and refined
- Concept
  - Compute non-pressure forces, intermediate velocity and position
  - Iteratively
    - Compute intermediate density from intermediate position
    - Compute pressure from intermediate density
    - Compute pressure forces
    - Update intermediate velocity and position
- Motivation
  - Parameterized by a desired density error, not by a stiffness constant
  - Provides a fluid state with a guaranteed density error

Iterative SESPH with Splitting

for all *particle i* **do**
    find neighbors j
for all *particle i* **do**
    $\mathbf{F}_i^{viscosity} = m_i \nu \nabla^2 \mathbf{v}_i \qquad \mathbf{F}_i^{other} = m_i \mathbf{g}$
    $\mathbf{v}_i^* = \mathbf{v}_i(t) + \Delta t \frac{\mathbf{F}_i^{viscosity} + \mathbf{F}_i^{other}}{m_i} \qquad \mathbf{x}_i^* = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i^*$
repeat
    for all *particle i* **do**
        compute ρ_i^* using $\mathbf{x}_i^*$
        compute p_i using ρ_i^*, e.g. $p_i = k(\rho_i^* - \rho_0)$
    compute ρ_{err}, e.g. average or maximum
    for all *particle i* **do**
        $\mathbf{F}_i^{pressure} = -\frac{m_i}{\rho_i^*} \nabla p_i \qquad \mathbf{v}_i^* = \mathbf{v}_i^* + \Delta t \frac{\mathbf{F}_i^{pressure}}{m_i} \qquad \mathbf{x}_i^* = \mathbf{x}_i^* + \Delta t^2 \frac{\mathbf{F}_i^{pressure}}{m_i}$
until $\rho_{err} < \eta$ user-defined density error
for all *particle i* **do**
    $\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i^* \qquad \mathbf{x}_i(t + \Delta t) = \mathbf{x}_i^*$

Iterative SESPH - Variants

- Different quantities are accumulated
  - Pressure forces (local Poisson SPH)
  - Pressure (predictive-corrective SPH, PCISPH)
  - Distances (position-based fluids, PBF)
    - $\Delta \mathbf{x}_i = -\frac{1}{\rho_0} \sum_j (\frac{p_i}{\beta_i} + \frac{p_j}{\beta_j}) \nabla W_{ij}$ β is a pre-computed constant
- Different EOS and stiffness constants are used
  - Local Poisson SPH: $k = \frac{\rho_i^* r_i^2}{2 \rho_0 \Delta t^2}$
  - PCISPH: $k = \frac{2\rho_0^2}{m_i^2 \cdot \Delta t^2 \left(\sum_j \nabla W_{ij}^0 \cdot \sum_j \nabla W_{ij}^0 + \sum_j (\nabla W_{ij}^0 \cdot \nabla W_{ij}^0) \right)} = \frac{2\rho_0^2}{m_i^2 \cdot \Delta t^2 \sum_j (\nabla W_{ij}^0 \cdot \nabla W_{ij}^0)}$
  - PBF: $k = 1 \quad (p_i = \frac{\rho_i}{\rho_0} - 1)$

Iterative SESPH - Performance

- Typically 3-5 iterations for density errors between 0.1% and 1%
- Typical speed-up over non-iterative SESPH: 50
  - More computations per time step compared to SESPH
  - Significantly larger time step than in SESPH
- EOS and stiffness constant influence the number of required iterations to get a desired density error
  - Rarely analyzed
- Non-linear relation between time step and iterations
  - Largest possible time step does not necessarily lead to an optimal overall performance

Projection Schemes

- Compute pressure with a pressure-Poisson equation

$$\nabla^2 p_i = \frac{\rho_i}{\Delta t}\nabla \cdot \mathbf{v}_i^* = -\frac{1}{\Delta t}\frac{(\rho_i^* - \rho_i)}{\Delta t}$$

  - $\mathbf{v}_i^*$: predicted velocity considering all non-pressure forces
  - ρ_i^* : predicted density with respect to $\mathbf{v}_i^*$, e.g.,

$$\rho_i^* = \rho_i + \Delta t \frac{d\rho_i}{dt} = \rho_i - \Delta t \rho_i \nabla \cdot \mathbf{v}_i^*$$

- Source terms:
  - $\frac{-(\rho_i^* - \rho_i)}{\Delta t^2}$: divergence-free condition, e.g., [Cummins99]
  - $\frac{-(\rho_i^* - \rho_0)}{\Delta t^2}$: density invariance condition , e.g., [Shao03, IISPH]

IISPH - Method

- Continuity equation

$$\frac{d\rho_i}{dt} = -\rho_i \nabla \cdot \mathbf{v}_i$$

- SPH discretization

$$\frac{\rho_i(t+\Delta t)-\rho_i(t)}{\Delta t} = \sum_j m_j \left(\mathbf{v}_i(t+\Delta t) - \mathbf{v}_j(t+\Delta t)\right) \nabla W_{ij}(t)$$

  - Constrain $\rho_i(t+\Delta t)$ to reference density ρ_0
  - Velocities are unknown

IISPH - Method

- Split velocity

$$\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i^{adv}(t+\Delta t) + \Delta t \frac{\mathbf{F}_i^p(t)}{m_i}$$

  - Intermediate velocities without pressure force

$$\mathbf{v}_i^{adv}(t+\Delta t) = \mathbf{v}_i(t) + \Delta t \frac{\mathbf{F}_i^{adv}(t)}{m_i}$$

- Discretization

$$\frac{\rho_0 - \rho_i(t)}{\Delta t} = \sum_j m_j \Big(\mathbf{v}_i(t+\Delta t) - \mathbf{v}_j(t+\Delta t)\Big) \nabla W_{ij}(t)$$

$$\frac{\rho_0 - \rho_i(t)}{\Delta t} = \sum_j m_j \left(\mathbf{v}_{ij}^{adv}(t+\Delta t) + \Delta t \left(\frac{\mathbf{F}_i^p(t)}{m_i} - \frac{\mathbf{F}_j^p(t)}{m_j}\right)\right) \nabla W_{ij}(t)$$

$$\rho_0 - \underbrace{\left(\rho_i(t) + \Delta t \sum_j m_j \mathbf{v}_{ij}^{adv}(t+\Delta t) \nabla W_{ij}(t)\right)}_{\substack{\text{predicted density} \\ \rho_i^{adv}(t+\Delta t)}} = \Delta t^2 \sum_j m_j \left(\frac{\mathbf{F}_i^p(t)}{m_i} - \frac{\mathbf{F}_j^p(t)}{m_j}\right) \nabla W_{ij}(t)$$

IISPH – Pressure Force

- Momentum preserving formulation

$$\mathbf{F}_i^p = -m_i \sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W_{ij}$$

- Linear system with unknown pressure values

$$\rho_0 - \rho_i^{adv} = \Delta t^2 \sum_j m_j \left[\sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W_{ij} - \sum_k m_k \left(\frac{p_j}{\rho_j^2} + \frac{p_k}{\rho_k^2} \right) \nabla W_{jk} \right] \nabla W_{ij}$$

- Properties
  - A particle has up to 40 neighbors
  - Approximately 40*40 non-zero coefficients per equation

IISPH - Implementation

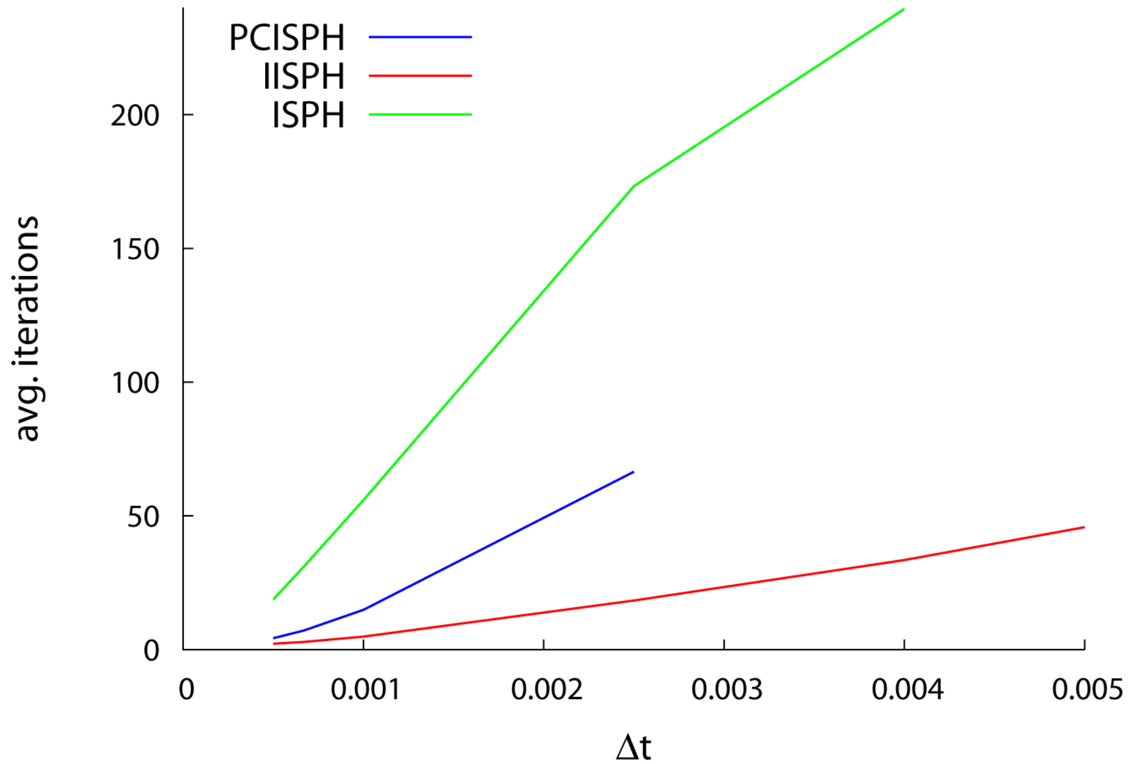
- Relaxed Jacobi
  - Matrix-free implementation
  - Implicit computation of non-diagonal entries
  - Seven scalar values per particle are stored
  - Two loops over set of particles per iteration
  - Fully parallelized
  - Fast convergence

IISPH - Properties

- Efficiency
  - Low number of iterations, typically between 5-15
  - Iterations are cheap to compute
  - ➢ Pressure solver outperforms previous schemes by factor 7
- Plausibility
  - Enforces compression of less than 0.1%
- Robustness
  - Handles larger time steps than previous schemes
  - Adaptive time-stepping is easy

Comparison of Iterative Methods

- Avergage number of iterations to enforce volume preservation with an error of 0.1%

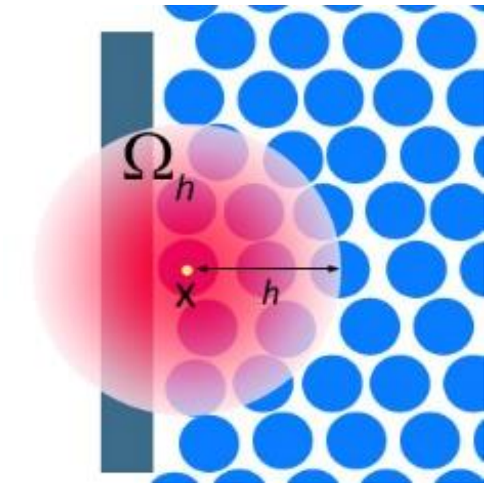


Topics / Research Challenges

- SPH fluid solver
- Neighborhood query
- Incompressibility / pressure computation
- **Boundary handling**
- Multiple phases
- Multi-resolution
- Surface reconstruction and rendering

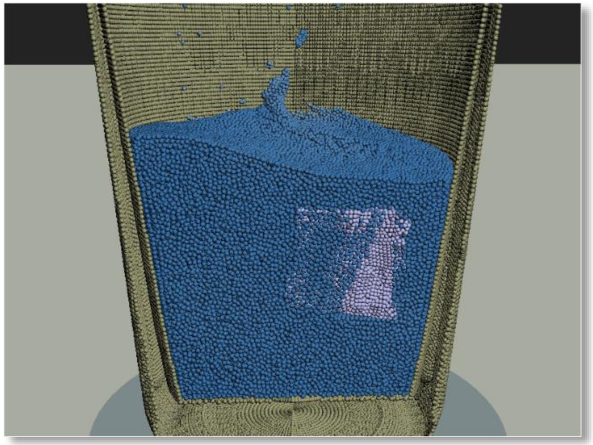
SPH Approximation at the Boundary

- Particle deficiency at the interface
  - Results in discontinuities
  - Large pressure gradients
- Solution:
  - Sample boundaries with particles to approximate field variables



Strategies

- Sampling
  - Pre-sampling, e.g., [Keiser06, Solenthaler07, Akinci12, Schechter12]
  - Online sampling, e.g., [Hu06]
- Field approximation of boundary particle
  - Interpolate, e.g., [Solenthaler07, Ihmsen10]
  - Mirror, e.g., [Akinci12, Schechter12]
- Force computation
  - Penalty forces, e.g., [Müller04, Lenaerts08]
  - Direct forcing, e.g., [Becker09]
  - Pressure-based, e.g., [Solenthaler07,Akinci12]



EG Strasbourg 2014

Sampling of Arbitrary Meshes

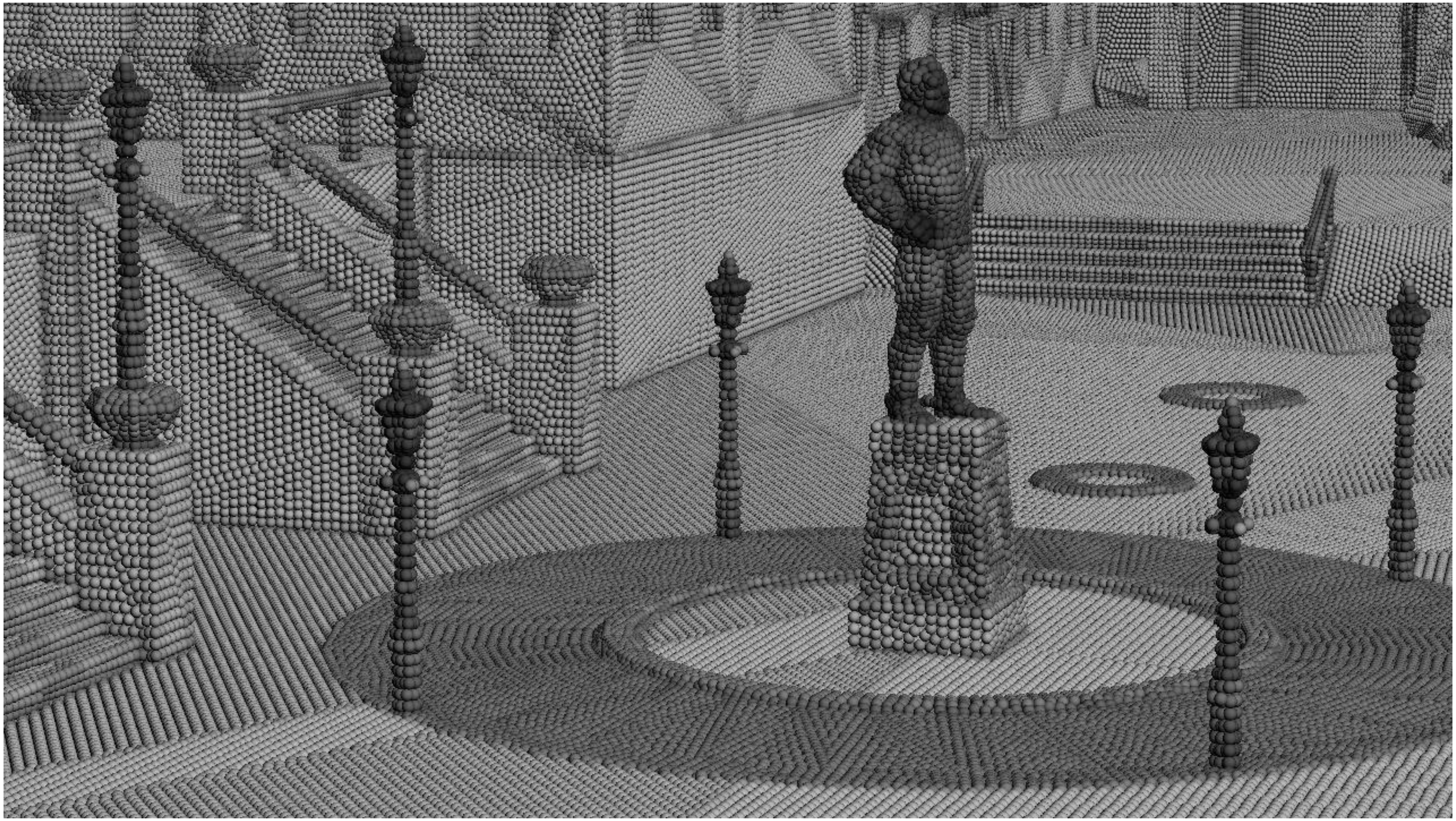
- Uniform sampling not always possible
- Particle spacing must not be larger than smoothing length h
- Correct particle volumes in oversampled boundary regions [Akinci12] $V_{b_i} = \frac{m_b}{\rho_{b_i}} = \frac{m_b}{\sum_k m_b W_{ik}} = \frac{1}{\sum_k W_{ik}}$

- Mirror fluid particle's rest density onto rigid particle to get mass contribution
- Example: adapted density computation

$$\rho_{f_i} = \sum_j m_{f_i} W_{ij} + \sum_k \rho_0 V_{b_k} W_{ik}$$

Strasbourg 2014

Sampling - Mass Contribution



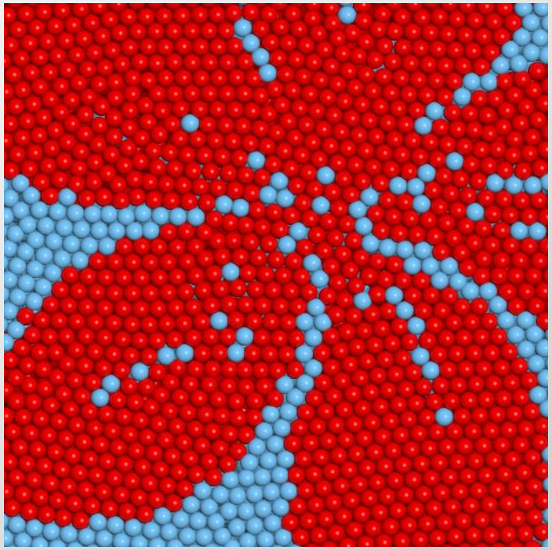
Topics / Research Challenges

- SPH fluid solver
- Neighborhood query
- Incompressibility / pressure computation
- Boundary handling
- **Multiple phases**
- Multi-resolution
- Surface reconstruction and rendering

Multiphase Fluids

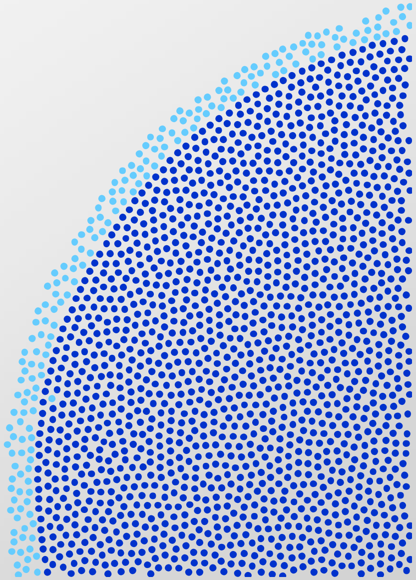
- Particles offer the advantage that the free surface and the interface between two fluids is sharply defined
- [Müller 05, Tartakovsky05, Hu06, Solenthaler08, Schechter12]

Liquid-liquid interface



[Solenthaler08]

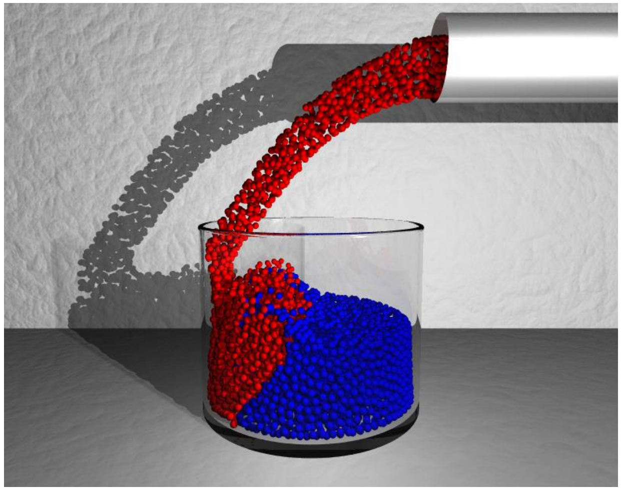
Liquid-air interface



[Schechter12]

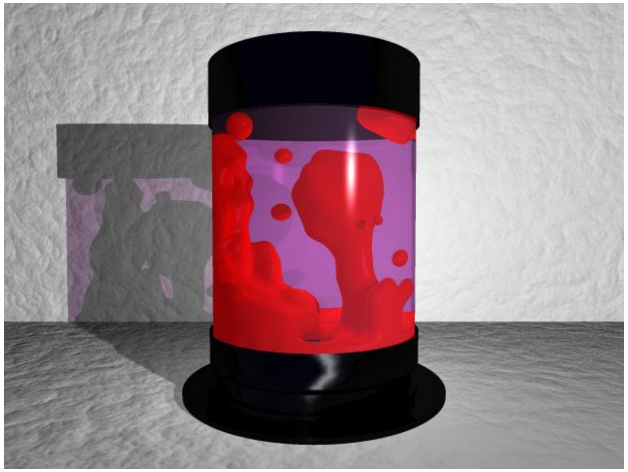
Particle Attributes

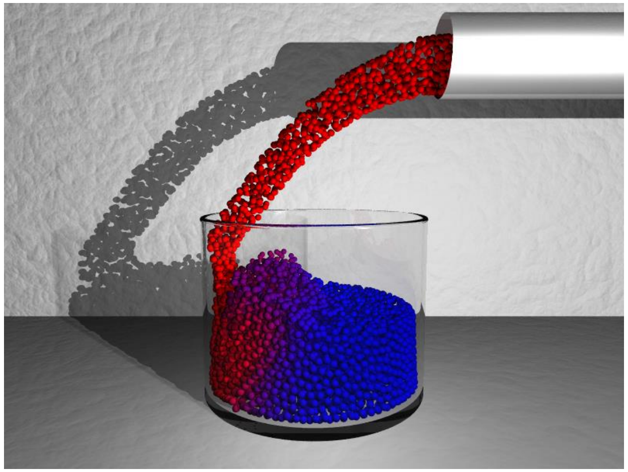
- Particles carry attributes individually
  - Mass
  - Rest density
  - Viscosity coefficient
  - Color attributes
  - Temperature
- Buoyancy emerges from individual rest densities $V_i^{fluid1} = V_j^{fluid2}$



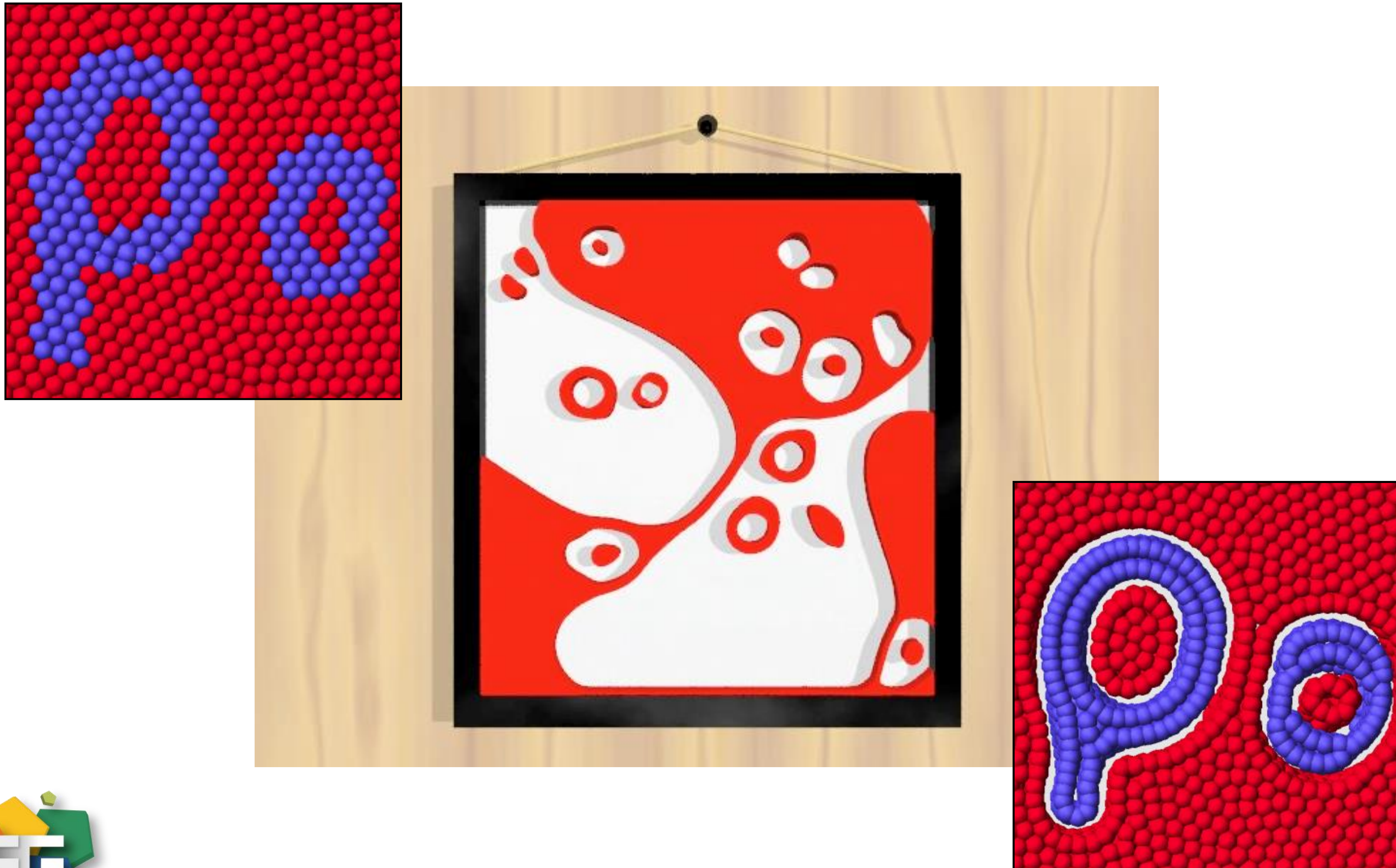
[Müller05]

- Diffusion of concentration, temperature





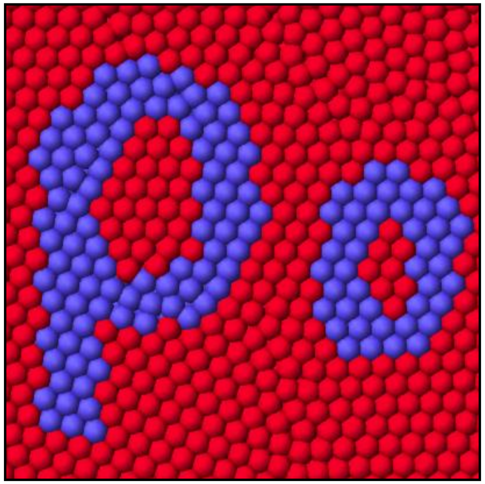
High Density Ratios



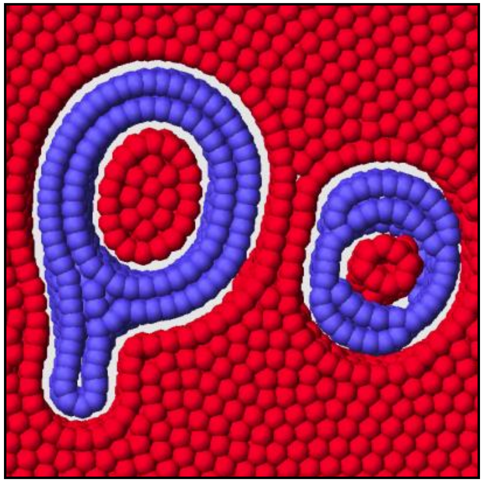
[Solenthaler08]

High Density Ratios

- Adapted SPH
  - Stable simulations despite high density ratios
  - We need full control over behavior



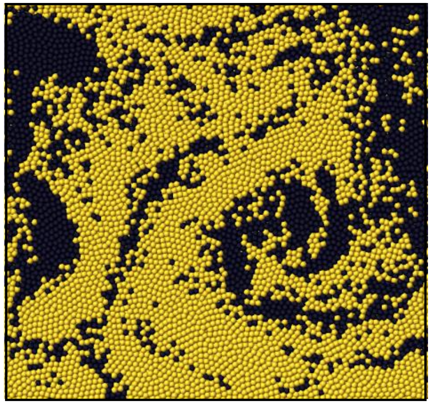
- Standard SPH
  - Cannot handle discontinuities at interfaces
  - Results in spurious and unphysical interface tension
  - Large density differences lead to instability problems



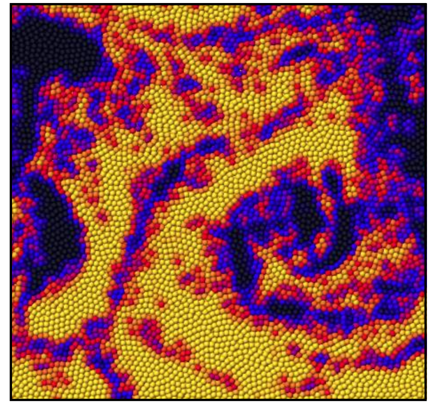
[Solenthaler08]

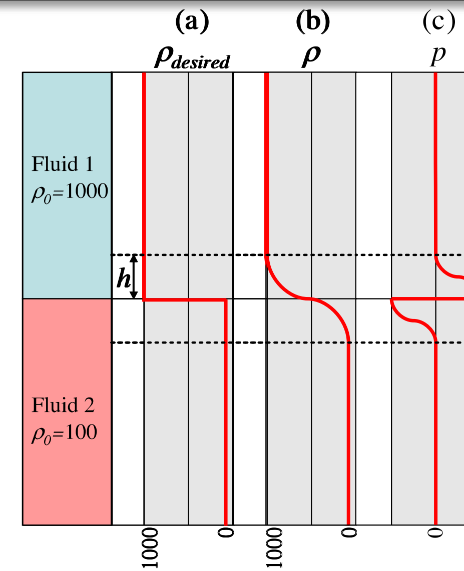
Interface Problems

Desired density field



SPH density field





Particle density

$$\delta_i = \sum_j W(\mathbf{r}_{ij}, h)$$

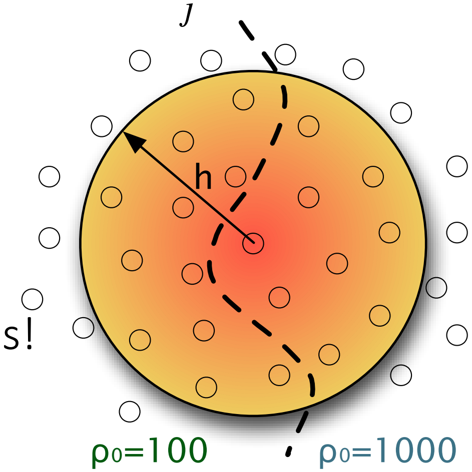
Adapted density

$$\tilde{\rho}_i = m_i \delta_i = m_i \sum_j W(\mathbf{r}_{ij}, h)$$

Apply the same idea to all forces!

[Solenthaler08]

$$\rho_i = \sum_j m_j W(\mathbf{r}_{ij}, h)$$



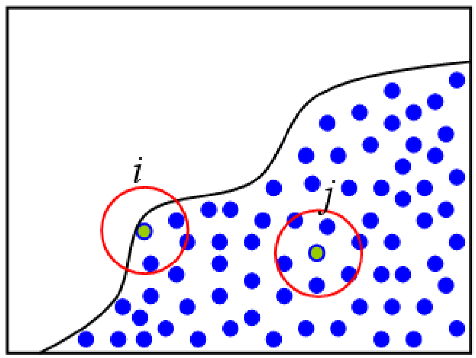
- Problems near interfaces where rest densities and masses vary
- Falsified smoothed quantities

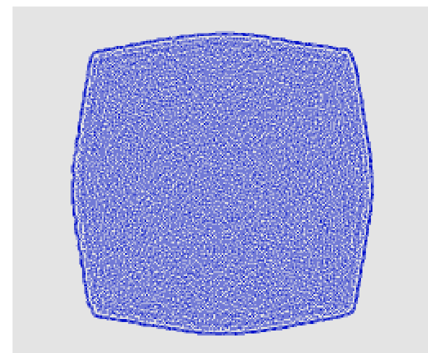
No Artificial Tension Forces

[Solenthaler08]

Liquid-air Interface

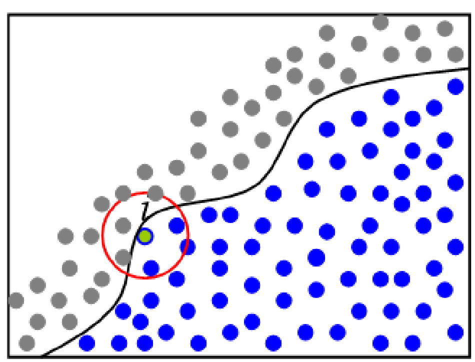
- Density deficiency at the free surface due to lack of neighbors
  -> Surface tension artifacts, clustering in spray

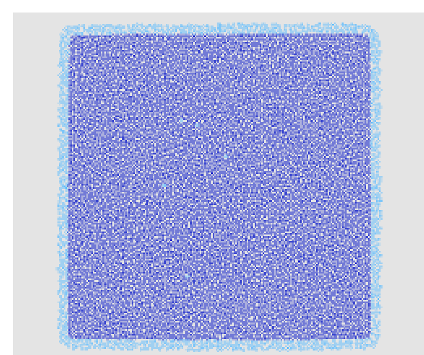




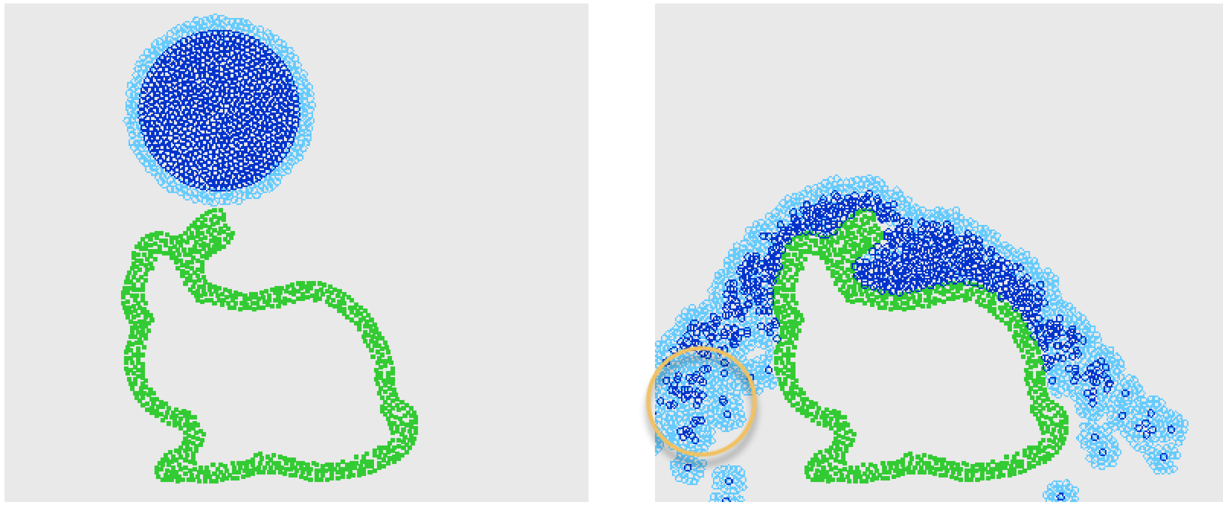
[Schechter12]

- Air particles solve these problems, at the cost of higher memory consumption and computation costs



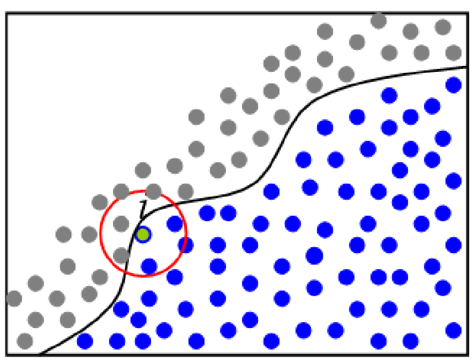


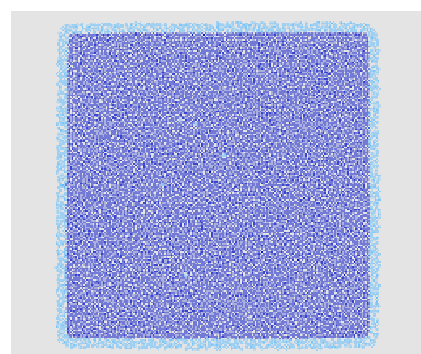
Liquid-air Interface



[Schechter12]

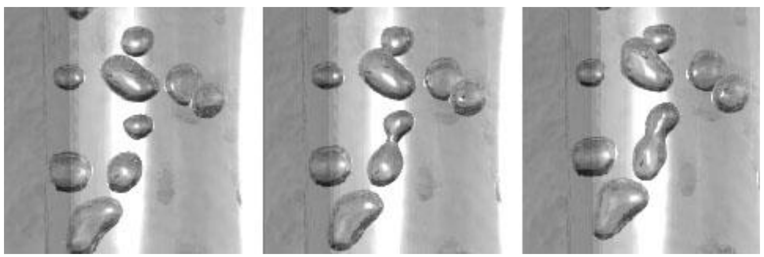
- Air particles solve these problems, at the cost of higher memory consumption and computation costs

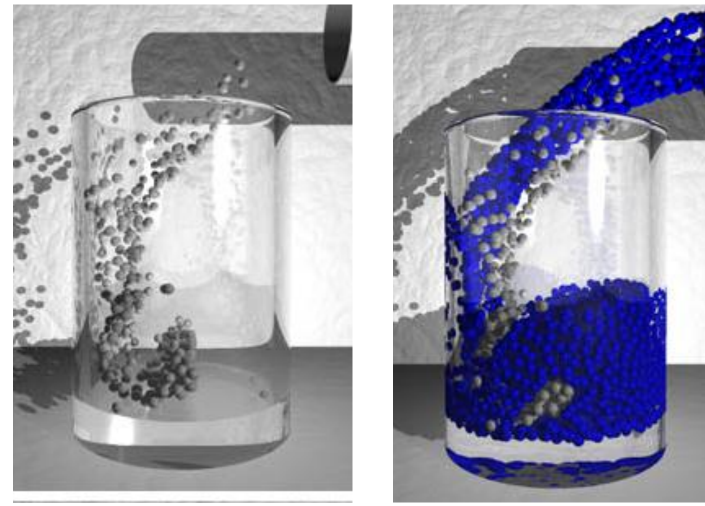




Trapped Air

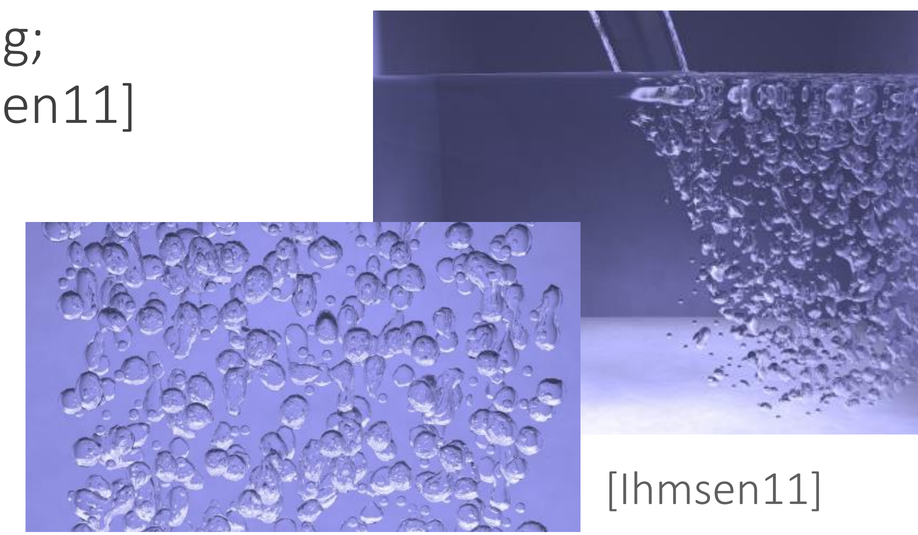
- Similar to [Schechter12], [Müller05] dynamically sample parts of the free surface with air particles -> trapped air





[Müller05]

- High density ratios are challenging; simulate phases separately [Ihmsen11] and couple them via drag force



[Ihmsen11]

Topics / Research Challenges

- SPH fluid solver
- Neighborhood query
- Incompressibility / pressure computation
- Boundary handling
- Multiple phases
- **Multi-resolution**
- Surface reconstruction and rendering

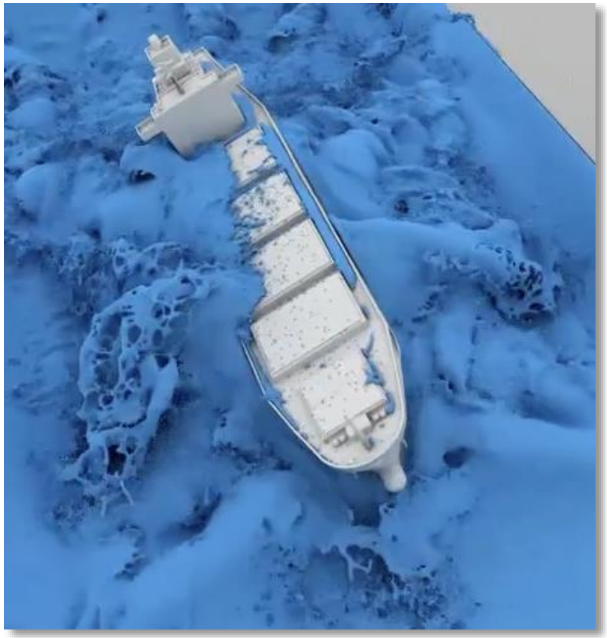
Motivation for Adaptive Spatial Discretization

- Many particles are needed to get the desired visual quality
- Computational cost depends linearly on the particle number

Idea: Allocate computing resources to interesting regions

3K particles
[Müller03]

3M particles
[Solenthaler09]



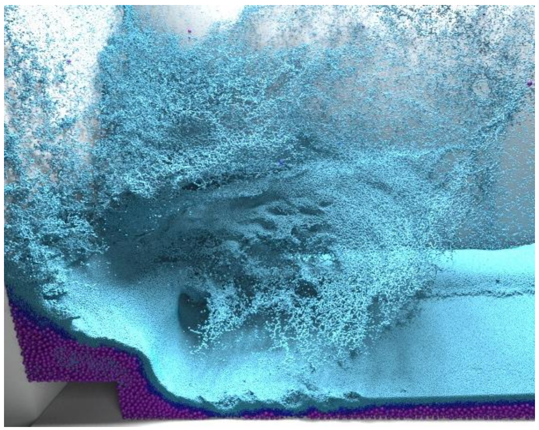
30M particles
[Ihmsen13]

Strasbourg 2014

Criteria

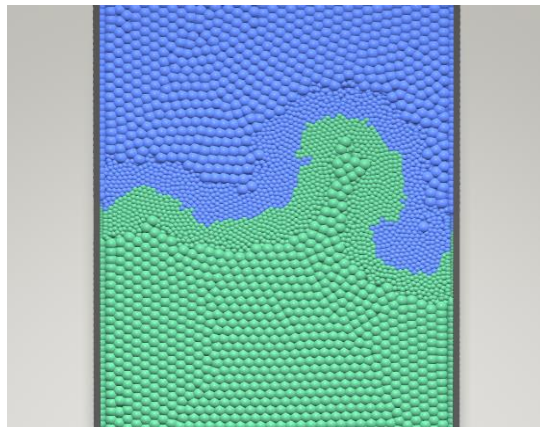
- High resolution in regions of interest and low resolution otherwise

Near the surface



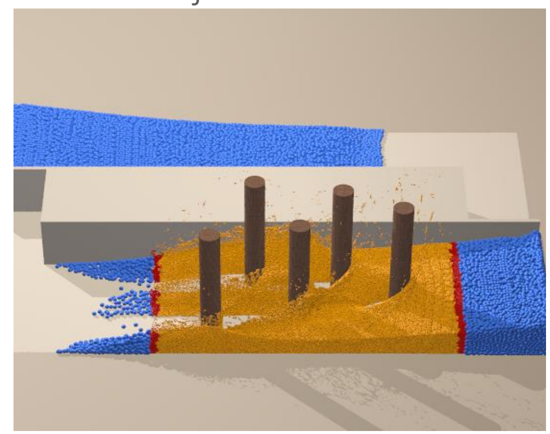
[Horvath13]

Near the interface



[Solenthaler11]

Object interaction

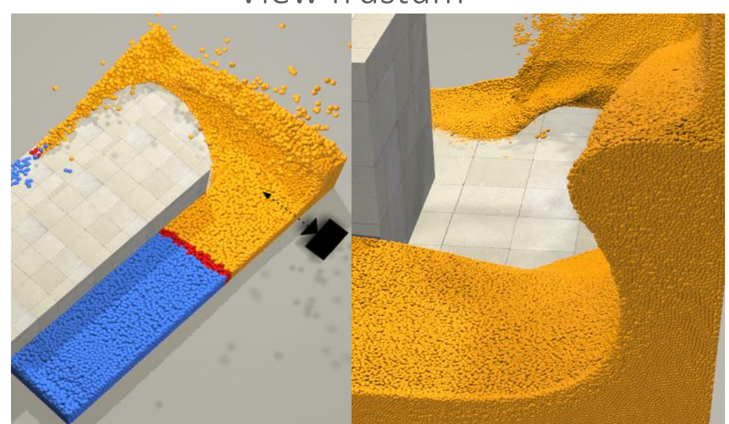


[Solenthaler11]

Distance to Camera

[Horvath13]

View frustum

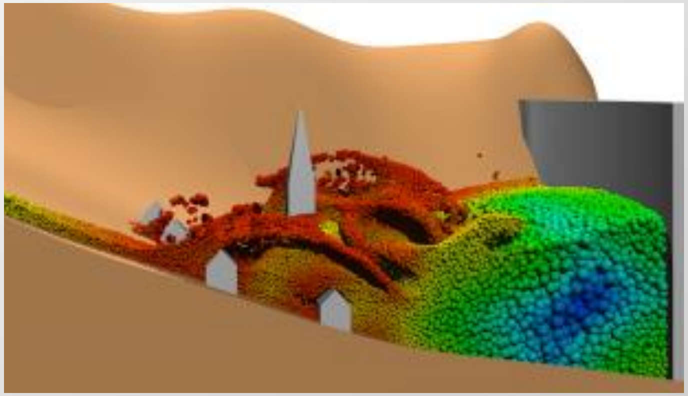


[Solenthaler11]

Approaches

Dynamic particle refinement

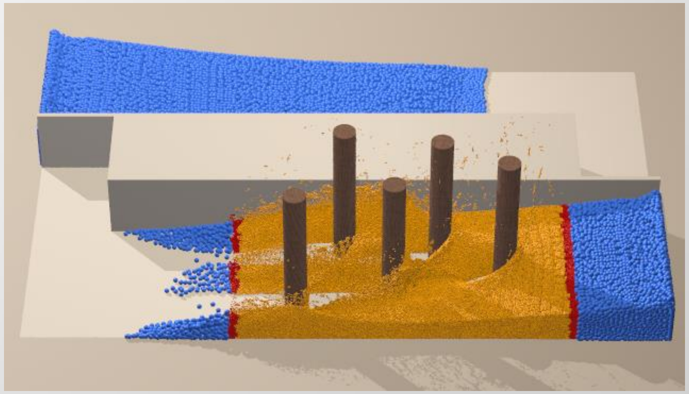
[Desbrun99, Adams07, Orthmann12]



[Adams07]

Multi-scale methods

[Solenthaler11, Horvath13]



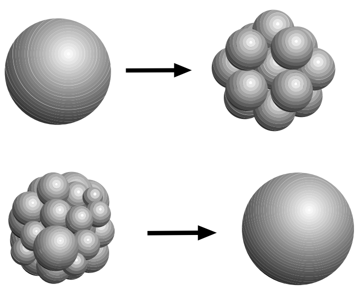
[Solenthaler11]

1 simulation with differently sized particles

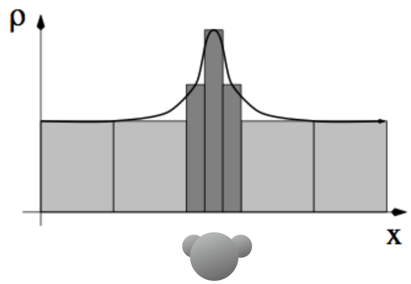
2 (or multiple) coupled simulations, each with equally sized particles

Dynamic Particle Refinement

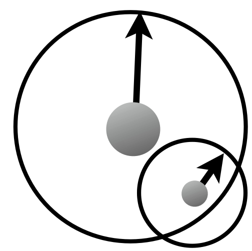
- Dynamically split and merge particles



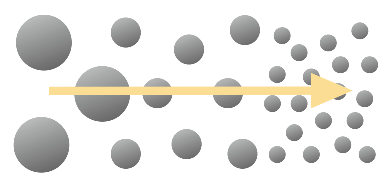
[Desbrun99]



Reproduce field quantities



Symmetric visibility

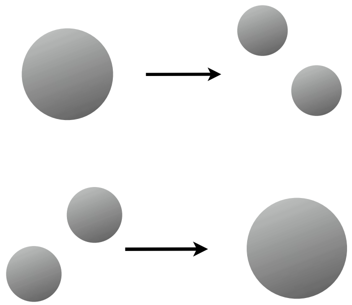


Smooth size transition

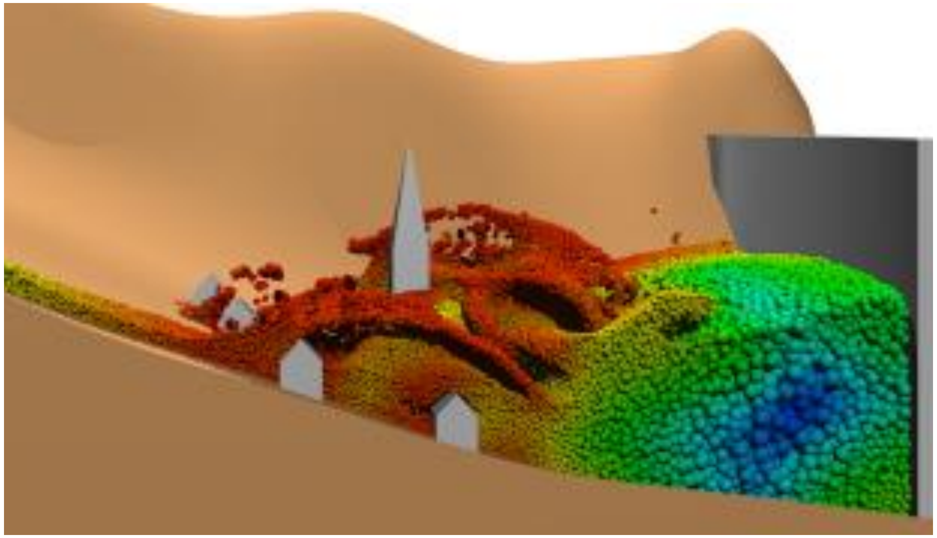
Strasbourg 2014

Dynamic Particle Refinement

- Dynamically split and merge particles

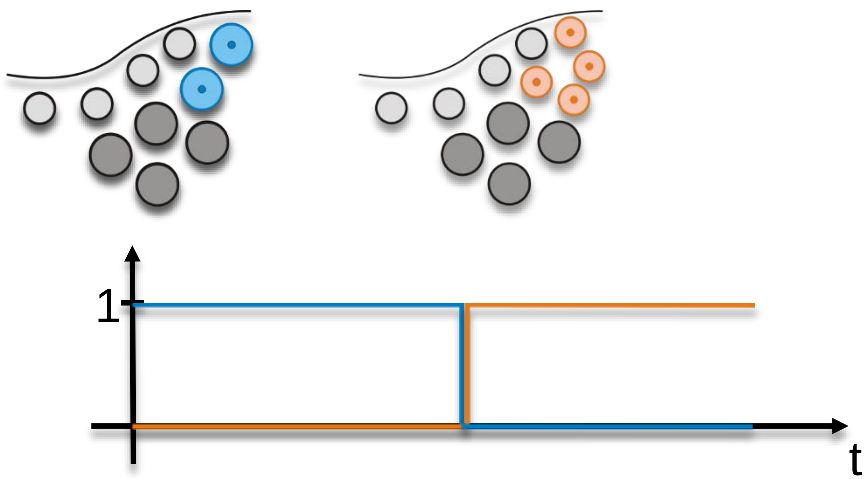


[Adams07]



Field Discontinuities

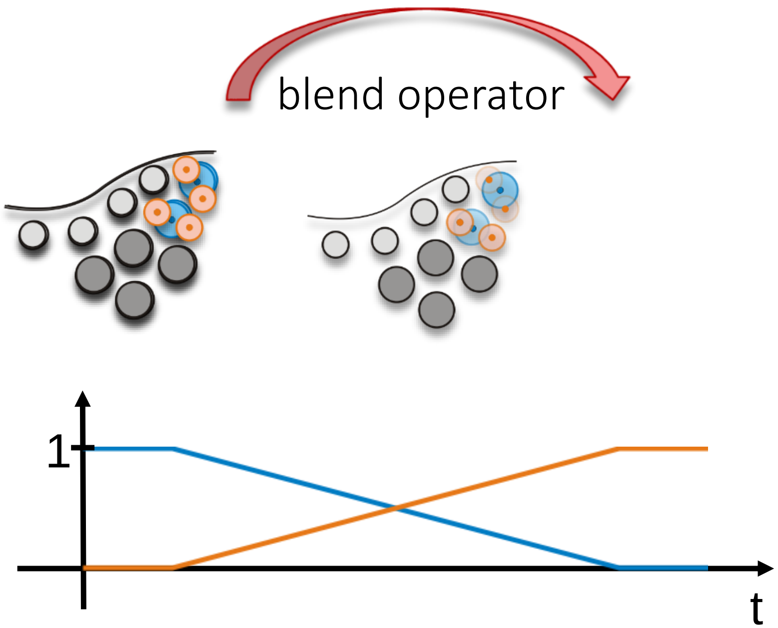
- Supporting incompressibility increases the problems of field discontinuities
  -> shocks, smaller time steps
- Non-continuous sampling over time introduces large errors



[Orthmann12]

Field Discontinuities

- Supporting incompressibility increases the problems of field discontinuities
  -> shocks, smaller time steps
- Non-continuous sampling over time introduces large errors
- Smooth temporal blending of resolution levels [Orthmann12]

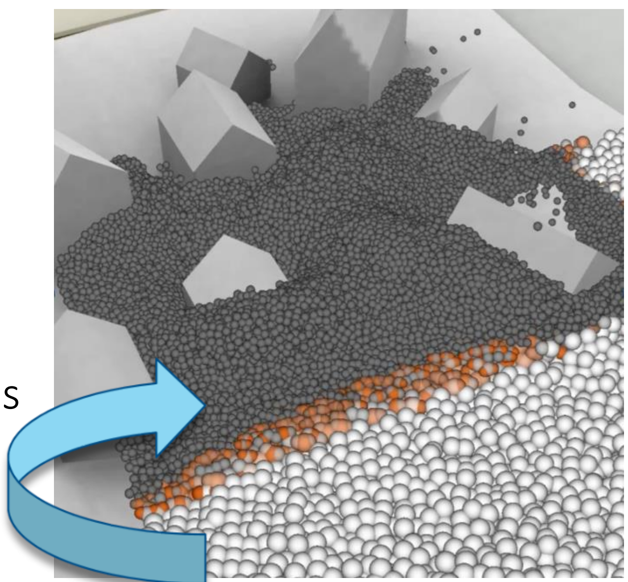


[Orthmann12]

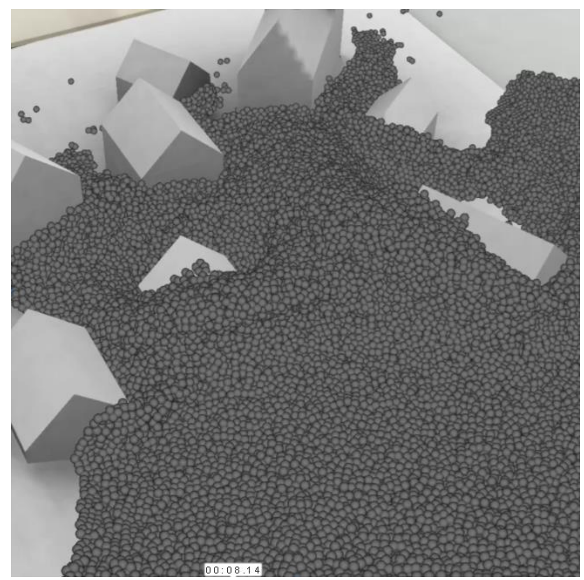
Field Discontinuities

- Supporting incompressibility increases the problems of field discontinuities
  -> shocks, smaller time steps
- Non-continuous sampling over time introduces large errors
- Smooth temporal blending of resolution levels [Othmann12]

Temporal blending



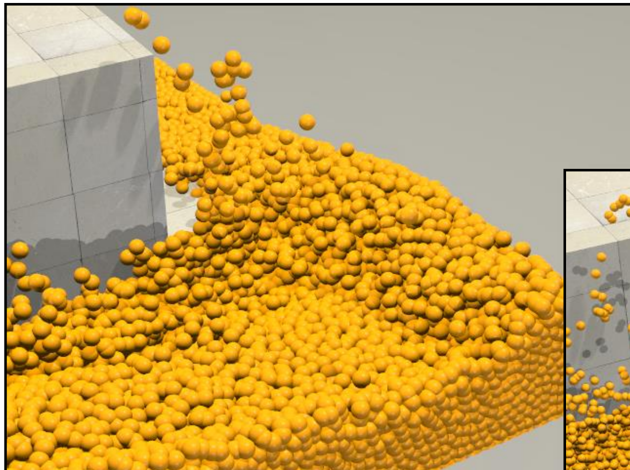
Reference solution



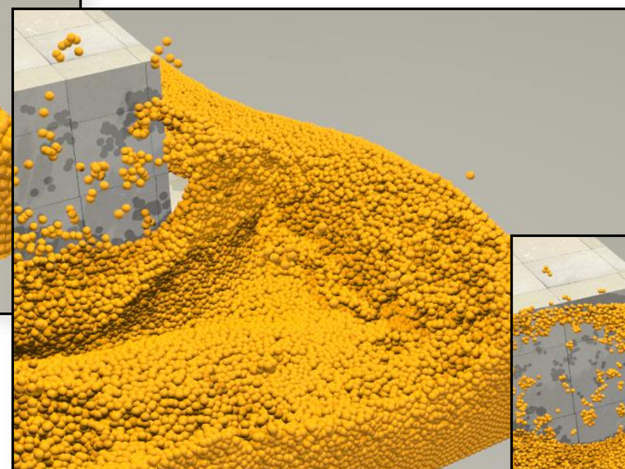
Resolution difference is limited

[Orthmann12]

Resolution Differences



Base resolution



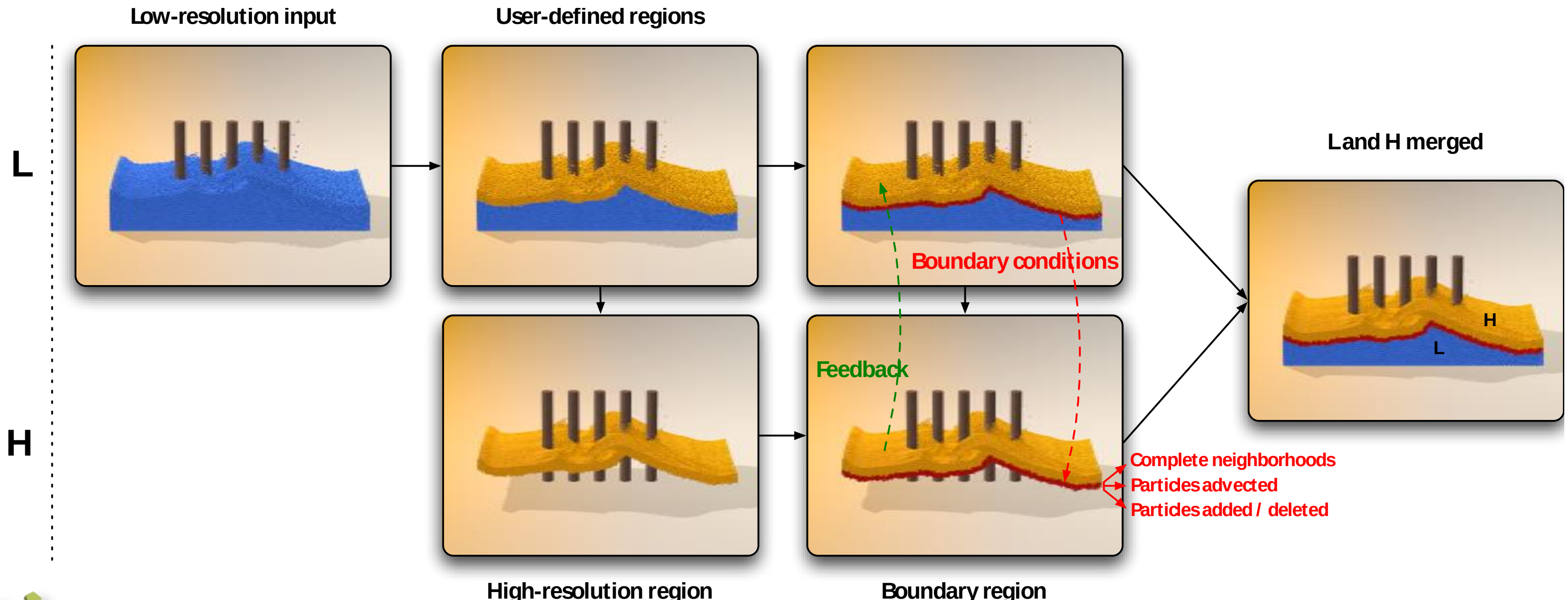
2x resolution
-> 8x more particles

4x resolution
-> 64x more particles

Multi-scale Methods

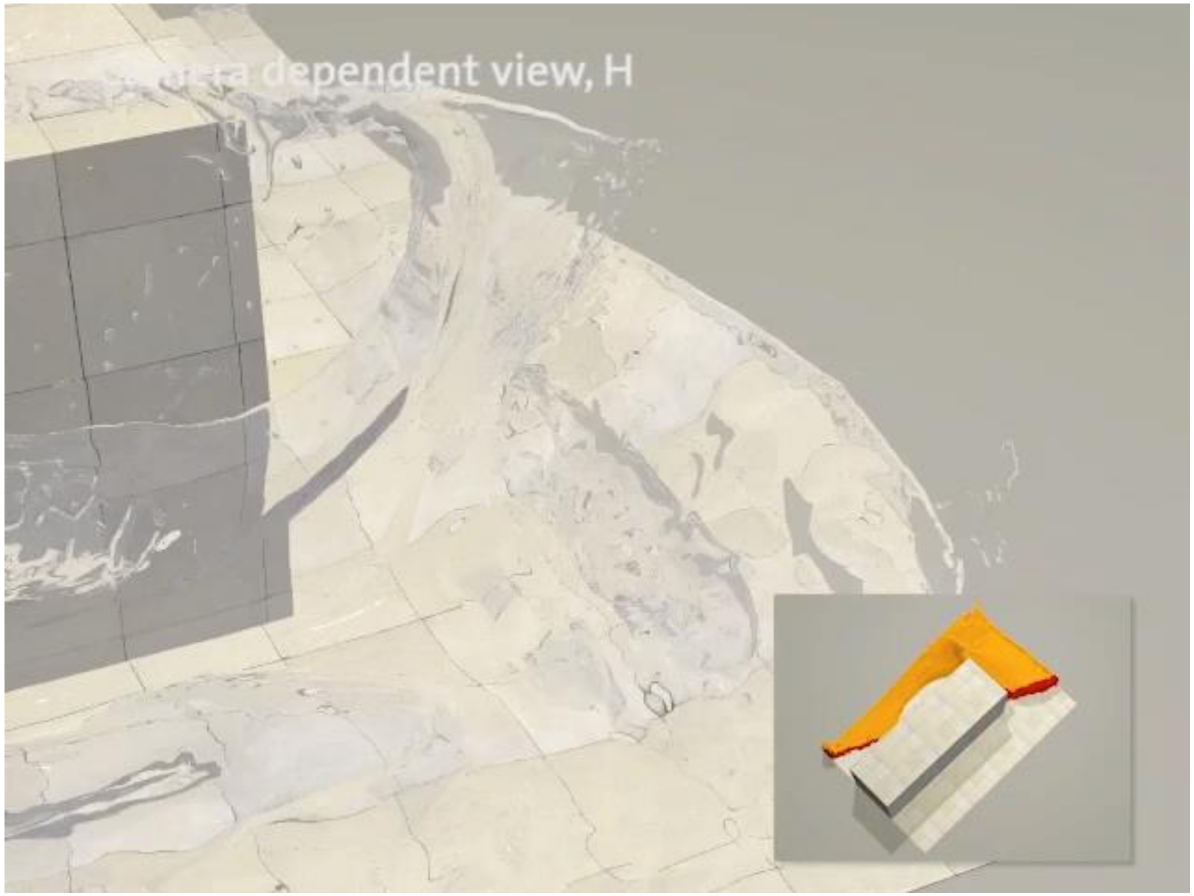
- Allow larger resolution differences
- Avoid splitting / merging and thus field discontinuities

- Use separate but coupled simulations for each level -> m^{level}, h^{level} const
- Two-scale approach [Solenthaler11]



[Solenthaler11]

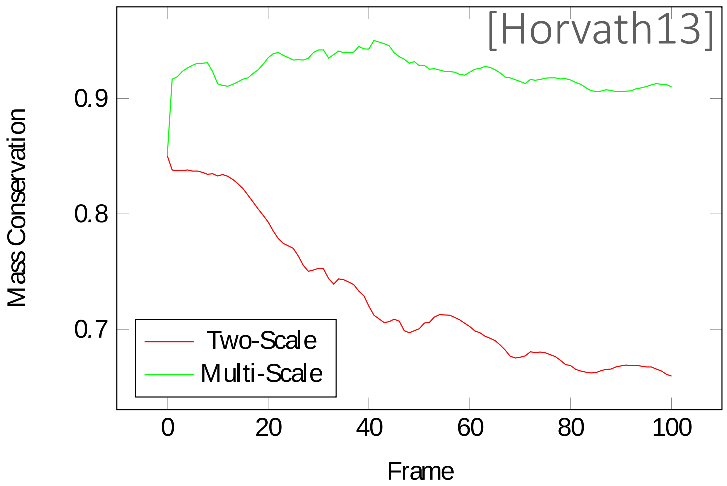
Multi-scale Methods – View Frustum



[Solenthaler11]

Extended to Multi-scale

- Multiple resolution levels
- Combined criteria



[Horvath13]

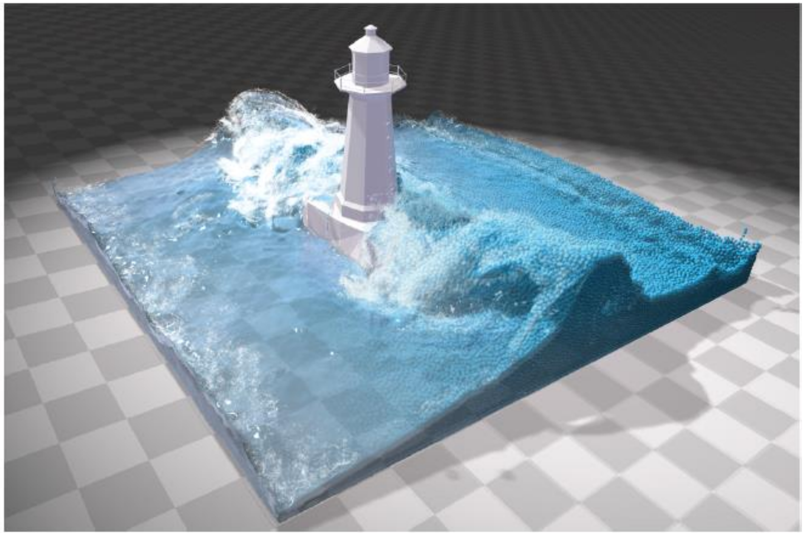
Speed-up:
[Adams07, Solenthaler11]: 3-7x
[Horvath13]: 3-12x
All previous work: less memory

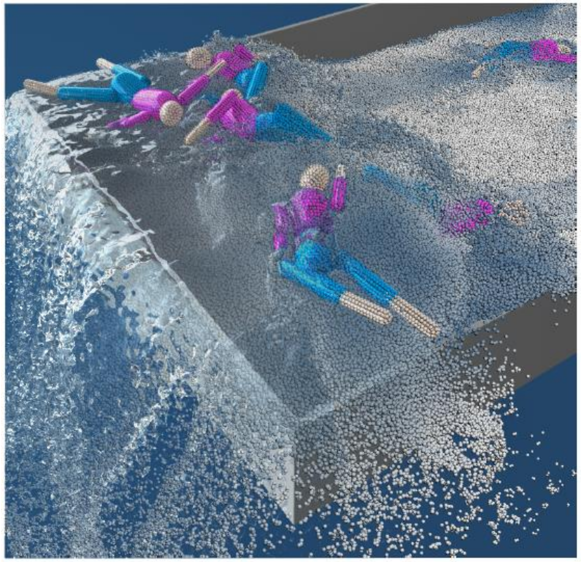
Topics / Research Challenges

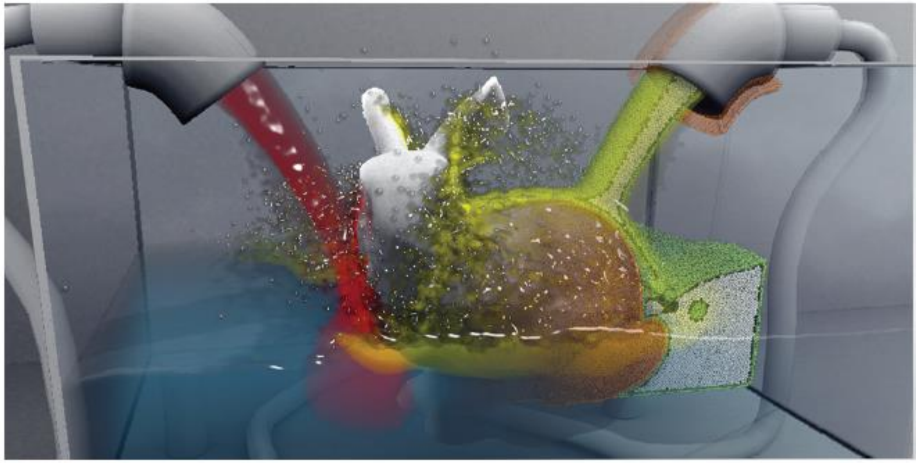
- SPH fluid solver
- Neighborhood query
- Incompressibility / pressure computation
- Boundary handling
- Multiple phases
- Multi-resolution
- **Surface reconstruction and rendering**

Motivation

- Smooth Surfaces
- Efficient Reconstruction
- Combined Volume Rendering





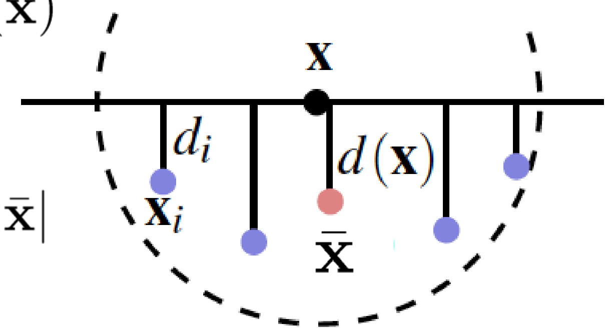


Outlook

- Scalar field functions
- Polygonalization and particle skinning
- Explicit surface tracking
- Direct surface rendering
- Volume rendering

Scalar Field Functions

- General approach: Surface = Iso-surface of scalar field function
- Metaballs [Blinn82]: Superimposed potential function located at particles → yields blobby surfaces
- Color field [Müller03]:
  - Color field ≈ 1 in bulk and 0 in air $c(\mathbf{x}) = \sum_j \frac{m_j}{\rho_j} W_j(\mathbf{x})$
  - Surface normal as color field gradient $\nabla c(\mathbf{x}) = \vec{n}(\mathbf{x})$
  - Disadvantage: Bumpy surface
- Distance to center of mass [Zhu05]: $\phi(\mathbf{x}) = R - |\mathbf{x} - \bar{\mathbf{x}}|$
  - Define level set function
    - Center of mass
      using larger radius $\bar{\mathbf{x}} = \sum_j \mathbf{x} W_j(\mathbf{x}) / \sum_j W_j(\mathbf{x})$
  - This approach yields smoother results



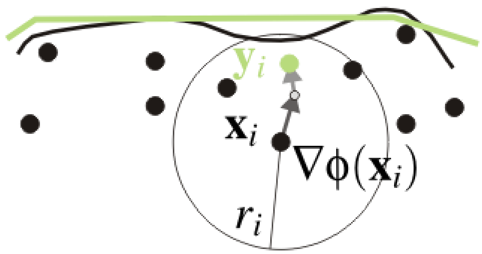
Adopted from [Adams07]

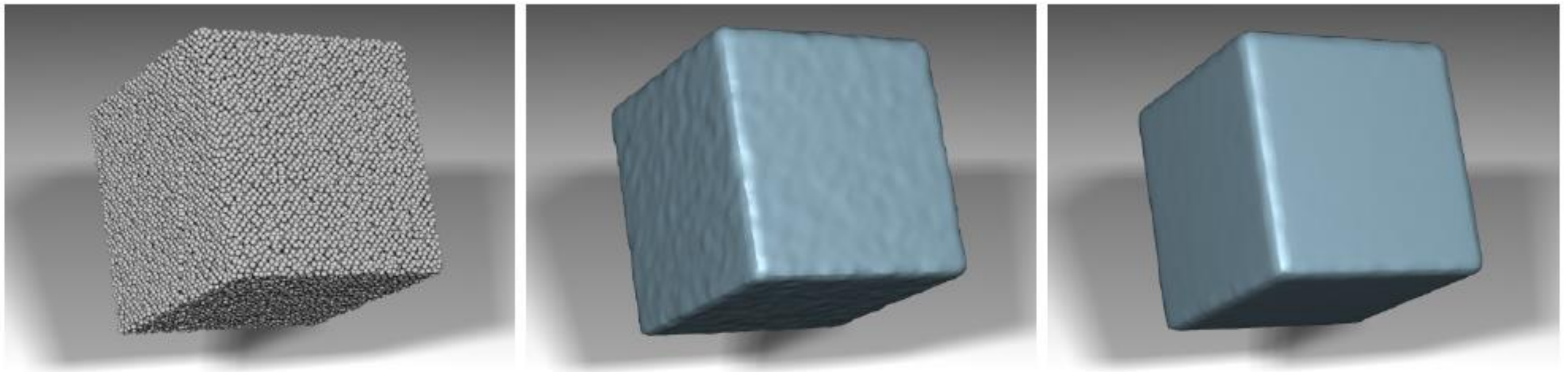
Particle-to-Surface Distance

- Improved particle-to-surface distance function [Adams07]:
  - Level set function with varying distance $\phi(\mathbf{x}) = d(\mathbf{x}) - |\mathbf{x} - \bar{\mathbf{x}}|$
    - Average distance to surface (from prior step):

$$d(\mathbf{x}) = \sum_j d_j W_j(\mathbf{x}) / \sum_j W_j(\mathbf{x})$$

- Surface projection using approximate particle-to-surface distances
  - Binary search along gradient $\mathbf{x}_i + s \cdot \nabla\phi(\mathbf{x}_i)$
  - Surface particle, if surface within radius r_i

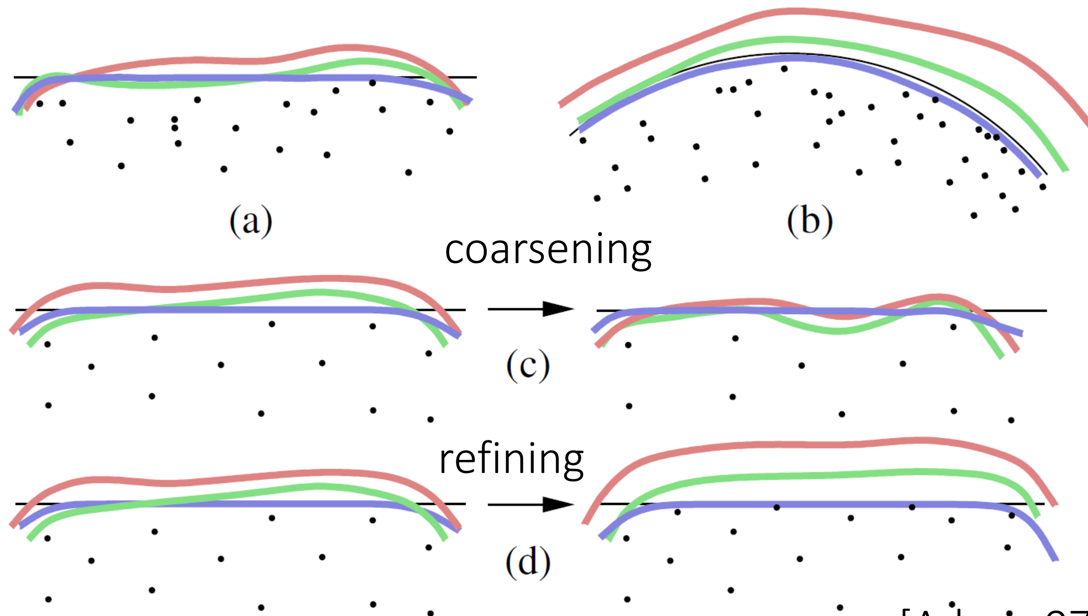




[Adams07] (100K ptcl; several sec on Intel Pentium D)

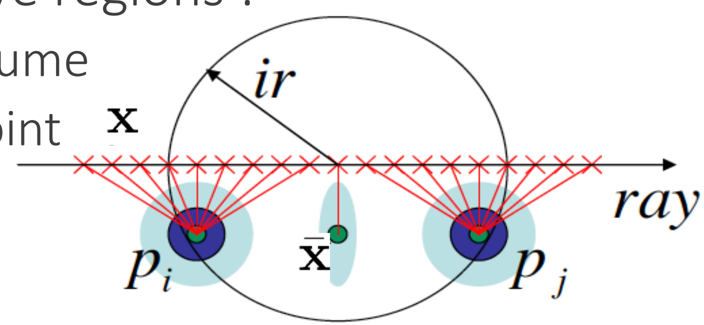
Scalar Field Functions: Comparison & Problems

- Comparison:
  - Metaballs [Blinn82]
  - Constant distance *R* [Zhu05]
  - Particle-to-surface distance $d(\mathbf{x})$ [Adams07



[Adams07]

- Issues with the center of mass in concave regions :
  - Erroneously parts outside of the fluid volume
  - Very sensitive to changes of the query point



Errors in concave regions [Solenthaler07]

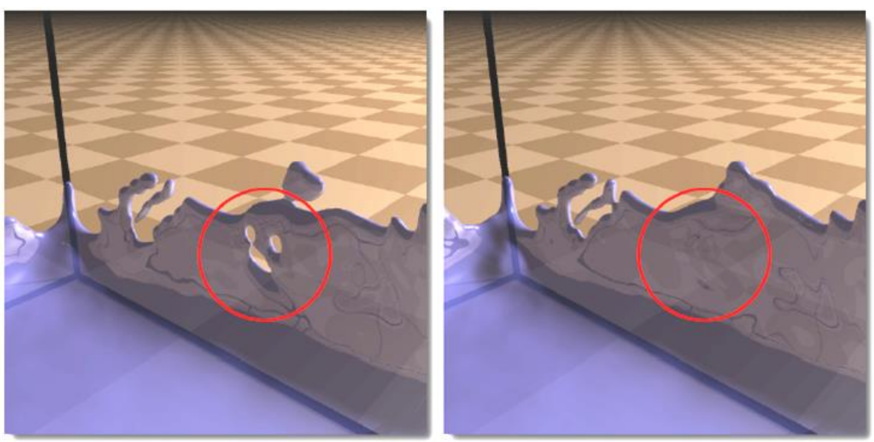
Scalar Field Function: Removal of Artifacts

- Analysis of gradient of mass center [Solenthaler07]
  - Observation: Strong variation of center of mass $\bar{\mathbf{x}}$ at artifacts
  - Solution: Weight distance function according to eigenvalue of $\nabla_{\mathbf{x}}\bar{\mathbf{x}}(\mathbf{x})$

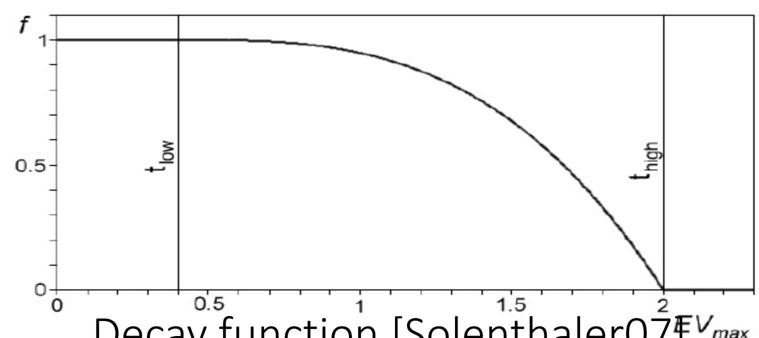
$$\phi(\mathbf{x}) = f(EV_{max}) \cdot d(\mathbf{x}) - |\mathbf{x} - \bar{\mathbf{x}}|$$

- Alternative approach [Onderik13]:
  - Use normalized iso-density instead of EV

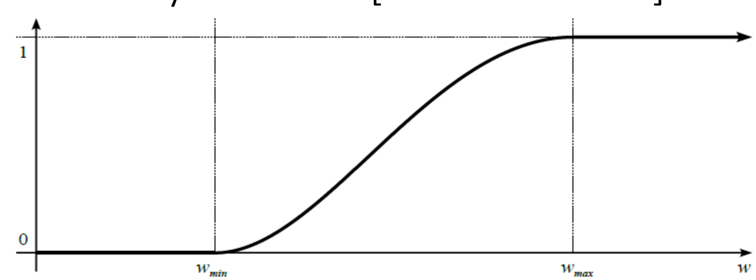
$$w(\bar{\mathbf{x}}) = \sum_j \left(W_j(\bar{\mathbf{x}}) / \sum_k W_k(\mathbf{p}_j) \right)$$



Comparison of [Solenthaler07] (left) and [Onderik13] (right) (14k ptcl, < 1 sec, Intel Core 2 Duo)



Decay function [Solenthaler07]



Transfer function [Ondrik13]

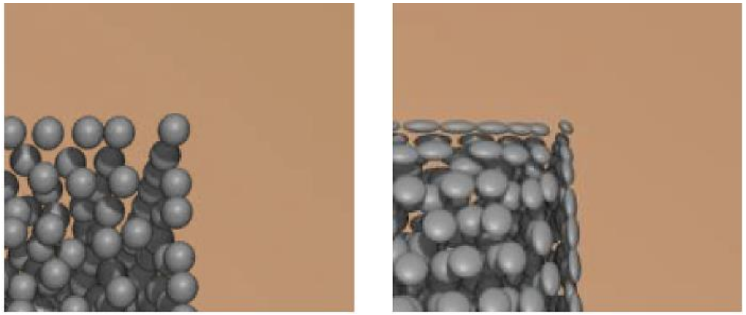
Scalar Field Functions: Anisotropic Kernels

- Goal: Smooth and feature preserving surface reconstruction
- Anisotropic kernels based on covariance matrix over local particle neighborhoods [Yu10].

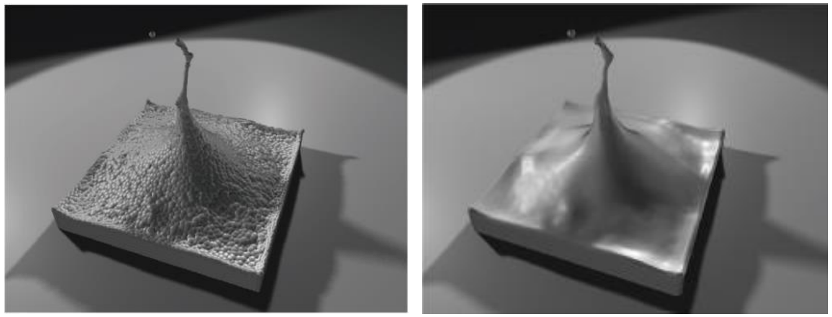
$$C_i = \sum_j W_j(\mathbf{x}_i) \cdot (\mathbf{x}_j - \bar{\mathbf{x}}_i(\mathbf{x}_i)) \cdot (\mathbf{x}_j - \bar{\mathbf{x}}_i(\mathbf{x}_i))^T / \sum_j W(\mathbf{x}_i) = R_i \ldots$$

- Scalar field defined via $\phi(\mathbf{x}) = \sum_j \frac{m_j}{\rho_j} W_j^{G_j}(\mathbf{x})$ using $G_i = \frac{1}{h} R_i \Sigma_i^{-1} R_i^T$

  in order to define anisotropic kernel $W_i^{G_i}(\mathbf{x}) = \det(C_i) W(G_i(\mathbf{x} - \mathbf{x}_i))$



Anisotropic kernel [Yu13]

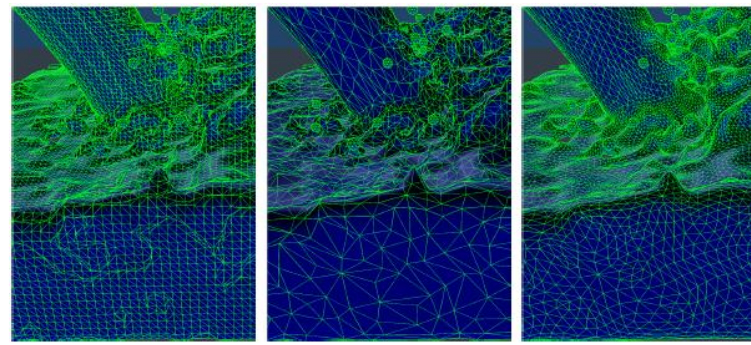


Particle and surface rendering [Yu13]
(24K ptcl; <10 sec on Intel Core 2 Duo)

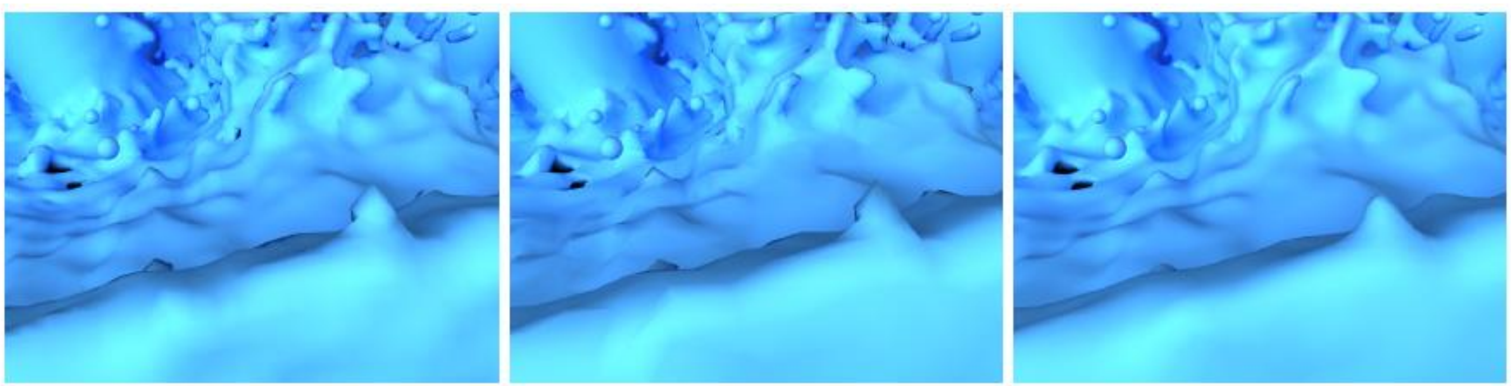
Marching Cubes Reconstruction

Surface Vertex Extraction → Scalar Field Computation → Marching Cubes → Decimation & Subdivision

- Polygonal iso-surfaces w.r.t. scalar function using Marching Cubes
- Idea: Optimization of Marching Cubes on GPU [Akinci12a, Akinci12b]
  - Store grid nodes in a narrow band at surface reduces complexity to $O(n^2)$ [Akinci12a]
  - Specific handling of “double layers”
  - Post-processing [Akinci12b]:
    - Decimation: QEM mesh reduction
    - Refinement: Loop subdivision scheme



Decimation & Subdivision [Akinci12b]



Initial surface [Solenthaler07] (left), after decimation (middle) and subdivision (right) [Akinci12b] (60k ptcl, 3.3 sec, Intel Xeon X5680)

Strasbourg 2014

Particle Skinning with Energy Minimization

- Idea: Find minimal thin plate energy surface between minimal and ma surface [Bhattacharya11].
- Sample potential function of particles onto regular grid

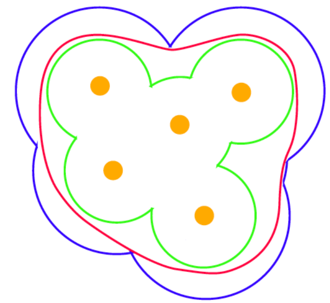
$$\phi_{min}(\mathbf{x}_{klm}) = \min_j |\mathbf{x}_{klm} - \mathbf{x}_j| - r_{min}$$

$$\phi_{max}(\mathbf{x}_{klm}) = \min_j |\mathbf{x}_{klm} - \mathbf{x}_j| - r_{max}$$

- Constrained thin plate optimization with initial

$$\phi_0(\mathbf{x}_{klm}) = \frac{1}{2}(\phi_{min}(\mathbf{x}_{klm} + \phi_{max}(\mathbf{x}_{klm})$$

  - Constraint: $\phi_i(\mathbf{x}_{klm}) \in [\phi_{min}(\mathbf{x}_{klm} + \phi_{max}(\mathbf{x}_{klm}]$



Constrained surface (red) between ϕ_{min} and ϕ_{max}

Amarillo with 0, 20 and 100 iterations [Bhattacharya11]

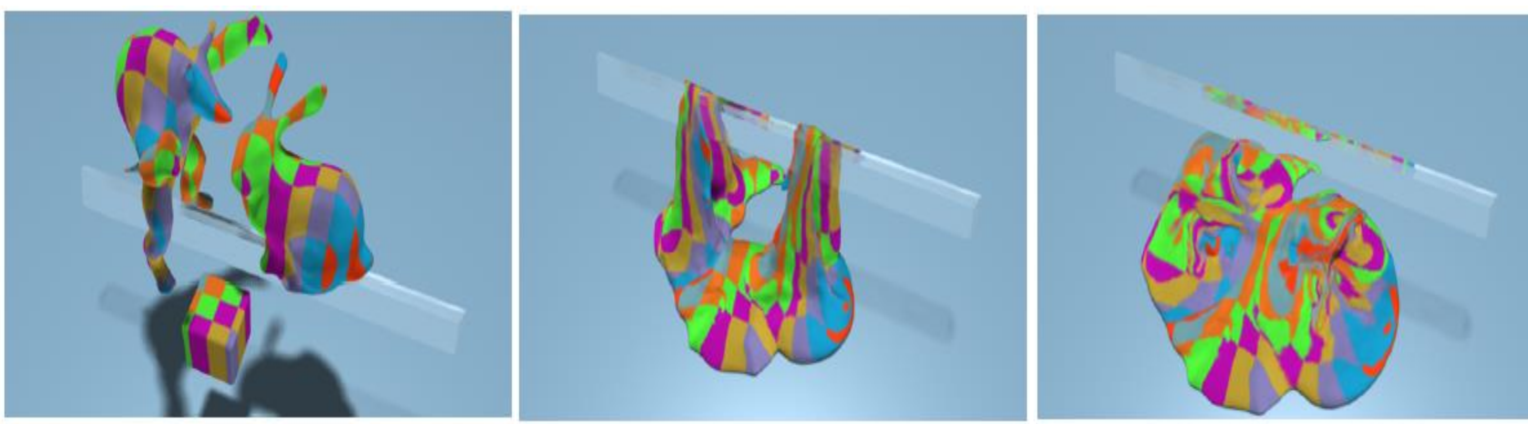
Explicit Surface Tracking

Mesh Initialization → Mesh Advection → Mesh Refinement → Mesh Projection → Topological Changes

- Idea: Attach & track an explicit mesh at the fluid surface [Yu12]
- Initial mesh using anisotropic kernels [Yu10] and MC reconstruction
- Mesh advection: Normalized velocities at mesh vertices $\boldsymbol{v}_i$

$$\vec{v}(\mathbf{v}_i) = \sum_j \vec{v}_j W_j(\mathbf{v}_i) / \sum_j W_j(\mathbf{x})$$

- Mesh refinement using standard split-merge approach
- Mesh vertices are projected onto iso surface [Adams07]
- If projection fails, i.e. $\phi(\mathbf{v}_i) \cdot \phi(\mathbf{v}_i + h \cdot \nabla\phi(\mathbf{v}_i)) > 0$
  - Topological merge if $\boldsymbol{v}_i$ interior ($\phi(\mathbf{v}_i) > 0$) and split else ($\phi(\mathbf{v}_i) < 0$)

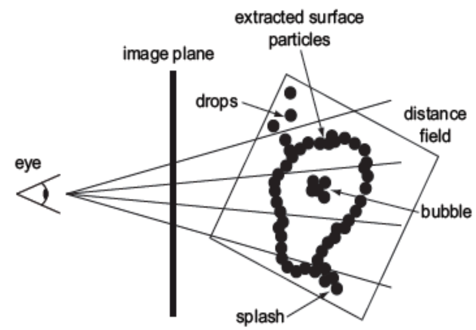


[Yu12] (29k ptls, 41s surface tracking, 24s surface reconstruktion, 2x Intel Xeon E5620)

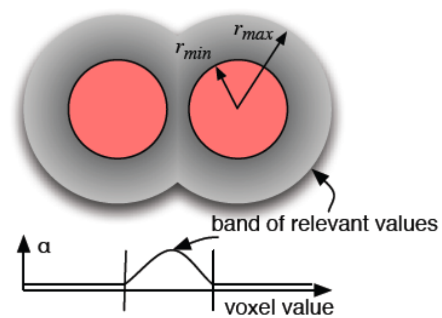
Direct Rendering

Surface Particle Extraction → Scalar Field Computation → Isosurface Raycasting

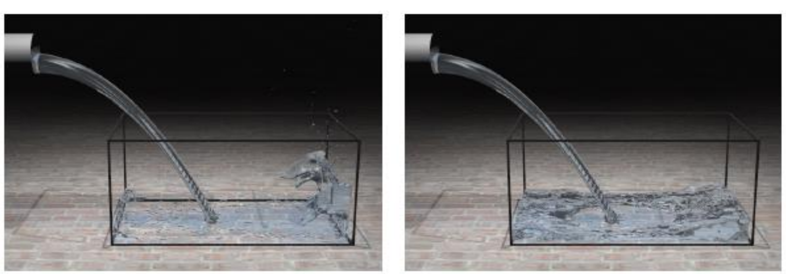
- Idea: Project surface particles onto grid and use iso-surface raycasting [Goswami10]
- Extraction of surface particles similar [Zhu05] (distance to center of mass), but only masses w/o kernel weighting
- Scalar field computation uses 3D splatting in grid
  - Transfer function defines relevant distance values $[r_{min}, r_{max}]$
  - Per grid vertex scalar value $\phi(\mathbf{x}_{klm}) = \min_j(|\mathbf{x}_{klm} - \mathbf{x}_j|)$
- Raycasting with normals computed on grid



Isosurface raycasting



Distance computation on small band around surface particles

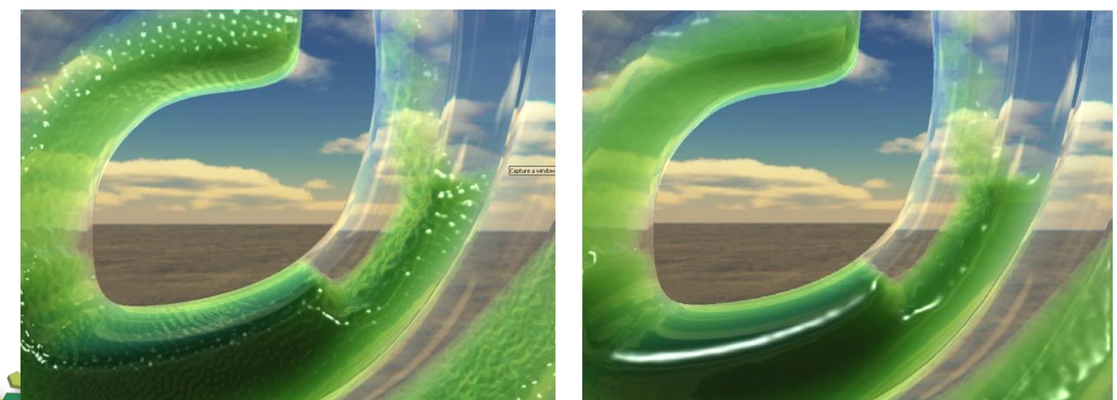


PovRay rendered results [Goswami10] (250k pctl)

Screen-Space Rendering

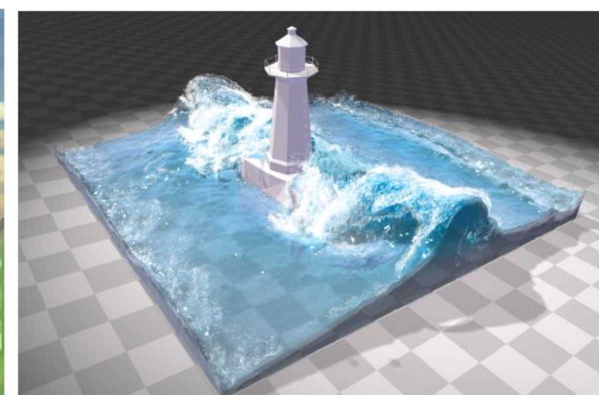
Depth Map Rendering → Screen Space Smoothing → Compositing & Lighting

- Idea: Splat particles as spheres onto image plane apply depth map smoothing [van der Laan09]
- Render particles and store screen-space depth and normal values
- Screen-space smoothing ("curvature flow") $$\frac{\partial z}{\partial t} = H = \frac{1}{2} \nabla \cdot \vec{n}$$
  - Evolve depth according to mean curvature using
  - Apply smoothing iteratively
- Final rendering using compositing



Gaussian smoothing (left) vs. curvature flow (right) [van der Laan09]

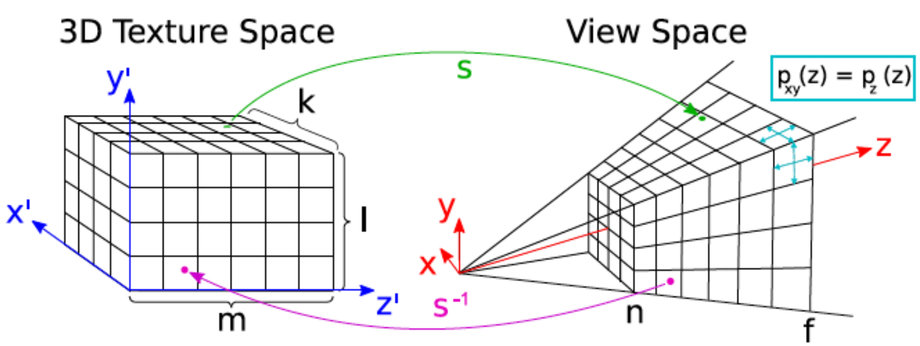
(64k pctl, 18.1 ms (bilateral), 50 ms (curvature flow 100 it), GF8800 GTS 512)



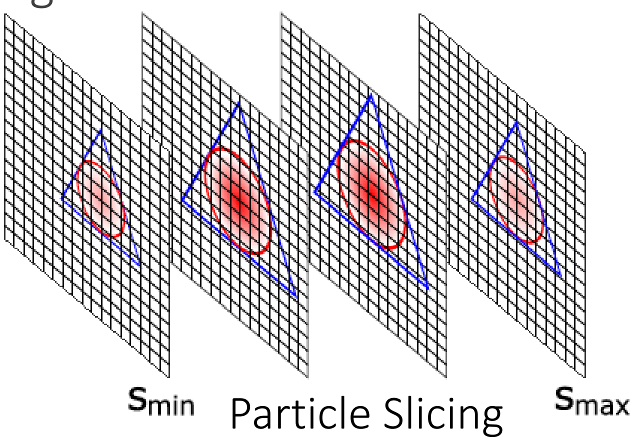
[Macklin13]

Volume Rendering

- Idea: Additionally visualize quantity distribution within bulk
- Approaches for SPH volume rendering
  - Texture slicing based on a view-aligned perspective grid [Fraedrich10]
  - Raycasting on a object aligned octree hierarchy [Orthmann10]
- Texture slicing on perspective grids [Fraedrich10]:
  - Perspective mapping ensures a quasi regular sampling along rays
  - Particle hierarchy allows for “particle size approx. cell size”
  - Particle slicing samples particle contributions onto grid
  - Final rendering via standard texture slicing using front-to-back slabs



Perspective Mapping

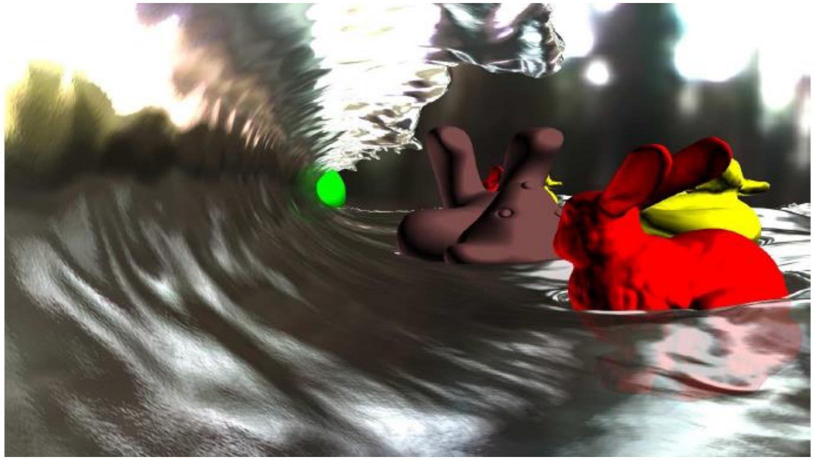


Particle Slicing

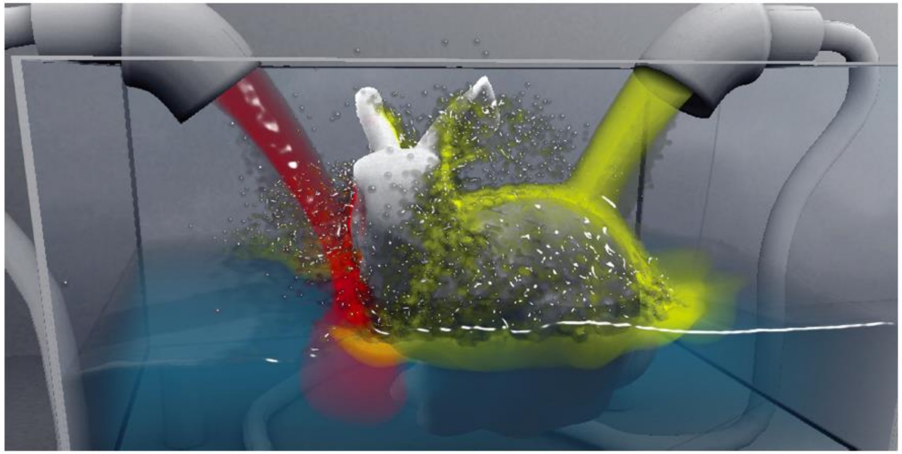
Combined Volume Rendering

Particle-to-Cell Mapping → Volume Sampling → Surface Rendering

- Raycasting on a object aligned octree hierarchy [Orthmann10]
  - Particle-to-cell mapping & hierarchy building per frame
  - Efficient traversal of OA octree using various caches



[Fraedrich10]
(2.5M pctl, 152 ms @ 512^2 res, 676 ms @ 1024^2 res, Intel Core 2 Duo 2.4 GHz + NV GTX 280)



Similar to [Orthmann10]
(2.4M pctl, 1024^2 res, 2s object space hierarchy, 870 ms perspective grid, Nvidia GTX Titan