Position Based Fluids

Miles Macklin [*] Matthias Müller [†]

NVIDIA

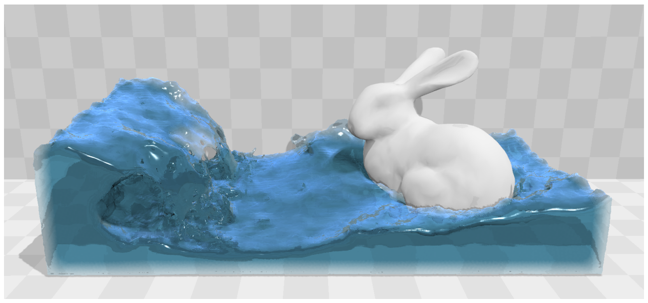
Abstract

In fluid simulation, enforcing incompressibility is crucial for realism; it is also computationally expensive. Recent work has improved efficiency, but still requires time-steps that are impractical for real-time applications. In this work we present an iterative density solver integrated into the Position Based Dynamics framework (PBD). By formulating and solving a set of positional constraints that enforce constant density, our method allows similar incompressibility and convergence to modern smoothed particle hydrodynamic (SPH) solvers, but inherits the stability of the geometric, position based dynamics method, allowing large time steps suitable for real-time applications. We incorporate an artificial pressure term that improves particle distribution, creates surface tension, and lowers the neighborhood requirements of traditional SPH. Finally, we address the issue of energy loss by applying vorticity confinement as a velocity post process.

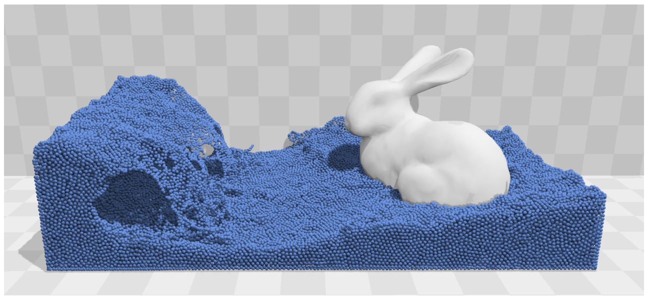
CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: fluid simulation, SPH, PCISPH, constraint fluids, position based dynamics

Links: DL PDF



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



(b) *Underlying simulation particles*

Figure 1: *Bunny taking a bath. 128k particles, 2 sub-steps, 3 density iterations per frame, average simulation time per frame 10ms.*

1 Introduction

Fluids, in particular liquids such as water, are responsible for many visually rich phenomena, and simulating them has been an area of long-standing interest and challenge in computer graphics. There are a variety of techniques available, but here we focus on particle methods, which are popular for their simplicity and flexibility.

Smoothed Particle Hydrodynamics (SPH) [Monaghan 1992][1994] is a well known particle based method for fluid simulation. It has many attractive properties: mass-conservation, Lagrangian discretization (particularly useful in games where the simulation domain is not necessarily known in advance), and conceptual simplicity. However, SPH is sensitive to density fluctuations from neighborhood deficiencies, and enforcing incompressibility is costly due to the unstructured nature of the model. Recent work has improved efficiency by an order of magnitude [Solenthaler and Pajarola 2009], but small time steps remain a requirement, limiting real-time applications.

For interactive environments, robustness is a key issue: the simulation must handle degenerate situations gracefully. SPH algorithms often become unstable if particles do not have enough neighbors for accurate density estimates. The typical solution is to try to avoid these situations by taking sufficiently small time steps, or by using sufficiently many particles, at the cost of increased computation.

In this paper, we show how incompressible flow can be simulated inside the Position Based Dynamics (PBD) framework [Müller et al. 2007]. We choose PBD for its unconditionally stable time integration and robustness, which has made it popular with game developers and film makers. By addressing the issue of particle deficiency at free surfaces, and handling large density errors, our method allows users to trade incompressibility for performance, while remaining stable.

2 Related Work

Müller [2003] showed that SPH can be used for interactive fluid simulation with viscosity and surface tension, by using a low stiffness equation of state (EOS). However to maintain incompressibility, standard SPH or weakly compressible SPH (WCSPH) [Becker and Teschner 2007] require stiff equations, resulting in large forces that limit the time-step size. Predictive-corrective incompressible SPH (PCISPH) [Solenthaler and Pajarola 2009] relaxes this time-step restriction by using an iterative Jacobi-style method that accu-

[*]e-mail:mmacklin@nvidia.com
[†]e-mail:matthiasm@nvidia.com

ACM Reference Format
Macklin, M., Müller, M. 2013. Position Based Fluids. ACM Trans. Graph. 32, 4, Article 104 (July 2013), 5 pages. DOI = 10.1145/2461912.2461984 http://doi.acm.org/10.1145/2461912.2461984.

mulates pressure changes and applies forces until convergence. It has the advantage of not requiring a user-specified stiffness value and of amortizing the cost of neighbor finding over many density corrections.

Bodin et al [2012] achieve uniform density fluid by posing incompressibility as a system of velocity constraints. They construct a linear complementarity problem using linearized constraint functions, which are solved using Gauss-Seidel iteration. In contrast, our method (and PCISPH) attempts to solve the non-linear problem by operating on particles directly, and re-evaluating constraint error and gradients each Jacobi iteration.

Hybrid methods, such as Fluid Implicit-Particle (FLIP) [Brackbill and Ruppel 1986] use a grid for the pressure solve and transfer velocity changes back to particles. FLIP was later extended to incompressible flow with free surfaces by Zhu and Bridson [2005]. Raveendran et al. [2011] use a coarse grid to solve for an approximately divergence free velocity field before an adaptive SPH update.

Clavet et al. [2005] also use a position based approach to simulate viscoelastic fluids. However, because the time step appears in various places of their position projections, their approach is only conditionally stable as in regular explicit integration.

Position Based Dynamics [Müller et al. 2007] provides a method for simulating dynamics in games based on Verlet integration. It solves a system of non-linear constraints using Gauss-Seidel iteration by updating particle positions directly. By eschewing forces, and deriving momentum changes implicitly from the position updates, the typical instabilities associated with explicit methods are avoided.

3 Enforcing Incompressibility

To enforce constant density we solve a system of non-linear constraints, with one constraint per-particle. Each constraint is a function of the particle's position and the positions of its neighbors, which we refer to collectively as $\mathbf{p}_1, \cdots, \mathbf{p}_n$. Following [Bodin et al. 2012] the density constraint on the *ith* particle is defined using an equation of state:

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

where ρ_0 is the rest density and ρ_i is given by the standard SPH density estimator:

$$\rho_i = \sum_j m_j W(\mathbf{p}_i - \mathbf{p}_j, h). \tag{2}$$

We treat all particles as having equal mass and will drop this term from subsequent equations. In our implementation we use the Poly6 kernel for density estimation, and the Spiky kernel for gradient calculation, as in [Müller et al. 2003].

Now we give some background on the position based dynamics method and then show how to incorporate our density constraint. PBD aims to find a particle position correction $\Delta\mathbf{p}$ that satisfies the constraint:

$$C(\mathbf{p} + \Delta\mathbf{p}) = 0 \tag{3}$$

This is found by a series of Newton steps along the constraint gradient:

$$\Delta\mathbf{p} \approx \nabla C(\mathbf{p})\lambda \tag{4}$$

$$C(\mathbf{p} + \Delta\mathbf{p}) \approx C(\mathbf{p}) + \nabla C^T \Delta\mathbf{p} = 0 \tag{5}$$

$$\approx C(\mathbf{p}) + \nabla C^T \nabla C \lambda = 0. \tag{6}$$

Algorithm 1 Simulation Loop

```
1: for all particles $i$ do
2:     apply forces $\mathbf{v}_i \Leftarrow \mathbf{v}_i + \Delta t \mathbf{f}_{ext}(\mathbf{x}_i)$
3:     predict position $\mathbf{x}_i^* \Leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
4: end for
5: for all particles $i$ do
6:     find neighboring particles $N_i(\mathbf{x}_i^*)$
7: end for
8: while $iter < solverIterations$ do
9:     for all particles $i$ do
10:        calculate $\lambda_i$
11:    end for
12:    for all particles $i$ do
13:        calculate $\Delta\mathbf{p}_i$
14:        perform collision detection and response
15:    end for
16:    for all particles $i$ do
17:        update position $\mathbf{x}_i^* \Leftarrow \mathbf{x}_i^* + \Delta\mathbf{p}_i$
18:    end for
19: end while
20: for all particles $i$ do
21:     update velocity $\mathbf{v}_i \Leftarrow \frac{1}{\Delta t}\left(\mathbf{x}_i^* - \mathbf{x}_i\right)$
22:     apply vorticity confinement and XSPH viscosity
23:     update position $\mathbf{x}_i \Leftarrow \mathbf{x}_i^*$
24: end for
```

[Monaghan 1992] gives the SPH recipe for the gradient of a function defined on the particles. Applying this, the gradient of the constraint function (1) with respect to a particle k is given by:

$$\nabla_{\mathbf{p}_k} C_i = \frac{1}{\rho_0} \sum_j \nabla_{\mathbf{p}_k} W(\mathbf{p}_i - \mathbf{p}_j, h) \tag{7}$$

Which has two different cases based on whether k is a neighboring particle or not:

$$\nabla_{\mathbf{p}_k} C_i = \frac{1}{\rho_0} \begin{cases} \sum_j \nabla_{\mathbf{p}_k} W(\mathbf{p}_i - \mathbf{p}_j, h) & \text{if } k = i \\ -\nabla_{\mathbf{p}_k} W(\mathbf{p}_i - \mathbf{p}_j, h) & \text{if } k = j \end{cases} \tag{8}$$

Plugging this into (6) and solving for λ gives

$$\lambda_i = -\frac{C_i(\mathbf{p}_1,...,\mathbf{p}_n)}{\sum_k \left|\nabla_{\mathbf{p}_k} C_i\right|^2} \tag{9}$$

which is the same for all particles in the constraint.

Because the constraint function (1) is non-linear, with a vanishing gradient at the smoothing kernel boundary, the denominator in equation (9) causes instability when particles are close to separating. As in PCISPH this can be solved by pre-computing a conservative corrective scale based on a reference particle configuration with a filled neighborhood.

Alternatively, constraint force mixing (CFM) [Smith 2006] can be used to regularize the constraint. The idea behind CFM is to soften the constraint by mixing in some of the constraint force back into the constraint function, in the case of PBD this changes (6) to

$$C(\mathbf{p} + \Delta\mathbf{p}) \approx C(\mathbf{p}) + \nabla C^T \nabla C \lambda + \varepsilon\lambda = 0. \tag{10}$$

Where ε is a user specified relaxation parameter that is constant over the simulation. The scaling factor is now

$$\lambda_i = -\frac{C_i(\mathbf{p}_1,...,\mathbf{p}_n)}{\sum_k \left|\nabla_{\mathbf{p}_k} C_i\right|^2 + \varepsilon}, \tag{11}$$

and the total position update $\Delta\mathbf{p}_i$ including corrections from neighbor particles density constraint (λ_j) is

$$\Delta\mathbf{p}_i = \frac{1}{\rho_0}\sum_j \left(\lambda_i + \lambda_j\right)\nabla W(\mathbf{p}_i - \mathbf{p}_j, h). \tag{12}$$

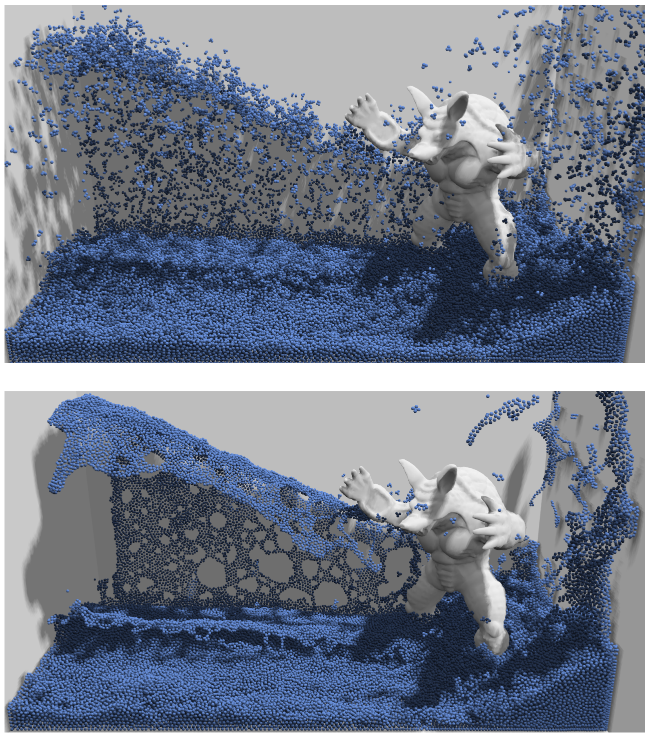


Figure 2: *Armadillo Splash, Top: particle clumping due to neighbor deficiencies, Bottom: with artificial pressure term, note the improved particle distribution and surface tension.*

4 Tensile Instability

A common problem in SPH simulations is particle clustering or clumping caused by negative pressures when a particle has only a few neighbors and is unable to satisfy the rest density (Figure 2). This may be avoided by clamping pressures to be non-negative, but at the cost of reduced particle cohesion. Clavet et al. [2005] use a second 'near pressure' term, while Alduan and Otaduy [2011] use discrete element (DEM) forces [Bell et al. 2005] to push apart particles closer than half the smoothing kernel width. Schechter and Bridson [2012] place ghost particles around the free surface to ensure consistent density estimates.

We follow the approach of [Monaghan 2000] which adds an artificial pressure specified in terms of the smoothing kernel itself as

$$s_{corr} = -k\left(\frac{W(\mathbf{p}_i - \mathbf{p}_j, h)}{W(\Delta\mathbf{q}, h)}\right)^n, \tag{13}$$

where $\Delta\mathbf{q}$ is a point some fixed distance inside the smoothing kernel radius and k is a small positive constant. $|\Delta\mathbf{q}| = 0.1h \cdots 0.3h$, $k = 0.1$ and $n = 4$ work well. We then include this term in the particle position update as

$$\Delta\mathbf{p}_i = \frac{1}{\rho_0}\sum_j \left(\lambda_i + \lambda_j + s_{corr}\right)\nabla W(\mathbf{p}_i - \mathbf{p}_j, h). \tag{14}$$

This purely repulsive term keeps particle density slightly lower than the rest density. Consequently, particles pull their neighbors inwards causing surface tension-like effects similar to the ones described in [Clavet et al. 2005]. We note that this effect is a non-physical artifact of the anti-clustering term and requires a trade off between clustering errors and surface tension strength.

Without clustering problems our algorithm is free from the rule of thumb that in SPH a particle must have 30-40 neighbors at all times, improving efficiency.

5 Vorticity Confinement and Viscosity

Position based methods introduce additional damping which is often undesirable. Fedkiw et al. [2001] introduced vorticity confinement to computer graphics to address numerical dissipation in the simulation of smoke, which was later extended to energy conserving fluid simulation in [Lentine et al. 2011]. In Bubbles Alive, Hong et al. [2008] show how vorticity confinement can be used in a hybrid setup where by vorticity is transferred from a grid to the SPH particles to introduce turbulent motion.

We optionally use vorticity confinement to replace lost energy (Figure 5). This requires first calculating the vorticity at a particle's location, for which we use the estimator given in [Monaghan 1992]:

$$\omega_i = \nabla \times \mathbf{v} = \sum_j \mathbf{v}_{ij} \times \nabla_{\mathbf{p}_j} W(\mathbf{p}_i - \mathbf{p}_j, h) \tag{15}$$

where $\mathbf{v}_{ij} = \mathbf{v}_j - \mathbf{v}_i$. Once we have the vorticity we calculate a corrective force using the location vector $\mathbf{N} = \frac{\eta}{|\eta|}$ with $\eta = \nabla|\omega|_i$

$$\mathbf{f}_i^{vorticity} = \varepsilon\left(\mathbf{N} \times \omega_i\right). \tag{16}$$

Unlike [Hong et al. 2008] we do not use normalized ω as this would increase vorticity indiscriminately. Instead we use the unnormalized value, which only adds vorticity where it already exists, as in [Fedkiw et al. 2001].

In addition, we apply XSPH viscosity [Schechter and Bridson 2012], which is important for coherent motion. The parameter c is typically chosen to be 0.01 in our simulations:

$$\mathbf{v}_i^{new} = \mathbf{v}_i + c\sum_j \mathbf{v}_{ij} \cdot W(\mathbf{p}_i - \mathbf{p}_j, h) \tag{17}$$

6 Algorithm

Our simulation loop is outlined in Algorithm 1. It is similar to the original Position Based Dynamics update except that each constraint is solved independently in a Jacobi fashion, rather than through sequential Gauss-Seidel iteration. We perform collision detection against solids as part of the constraint solving loop.

We recompute particle neighborhoods once per-step and recalculate distance and constraint values each solver iteration. This optimization can lead to density underestimates, for example if a particle separates from the initial set of neighbors. In PCISPH this can cause serious problems, once a particle becomes isolated, each iteration makes its pressure increasingly negative. If it then comes back into contact on a subsequent iteration, large erroneous pressure forces are applied. Our algorithm considers only the current particle positions (not accumulated pressure), so this does not occur.

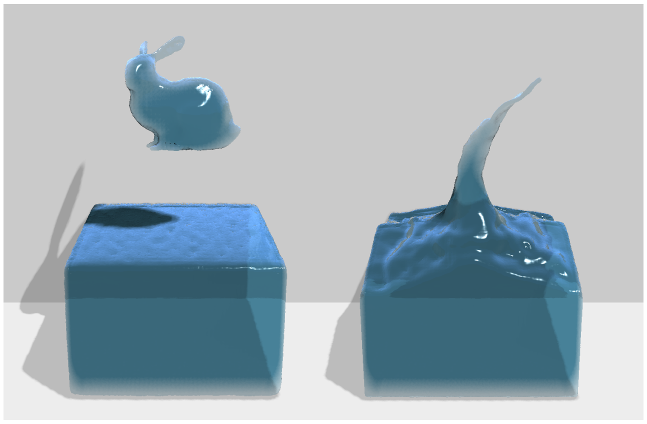
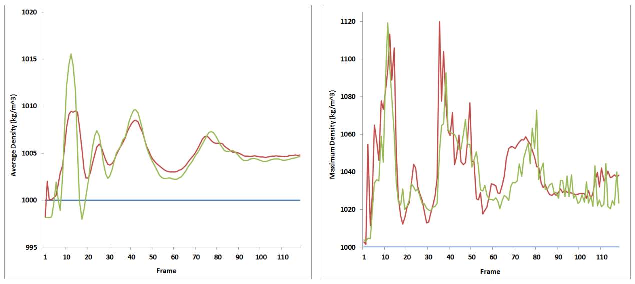


Figure 3: *Dropping a liquid bunny into a pool of water (80k particles).*



(a) *Average Density* **(b)** *Maximum Density*

Figure 4: *Density over the bunny drop simulation. Our algorithm maintains compressibility similar to PCISPH at time-steps more than twice as large.* ***Color key:*** *Blue, rest density. Red, PCISPH. Green, our method.*

7 Rendering

Real-time fluid surfacing is performed using a GPU based ellipsoid splatting technique. Particle anisotropy is first computed using the method of Yu and Turk [2013], and the surface is reconstructed using a method based on the screen-space filtering presented in [van der Laan et al. 2009].

8 Results

We tested our algorithm by dropping a liquid bunny into a pool of water (Figure 3) and compared our results with a PCISPH implementation. For this scenario PCISPH is not stable with less than 10 sub-steps per frame ($\Delta t = 0.0016s$). In contrast our algorithm is stable with a single step ($\Delta t = 0.016s$).

To compare compressibility we run PCISPH with 10 sub-steps and 4 pressure iterations, and our algorithm with 4 sub-steps and 10 iterations per sub-step, so that each performs 40 pressure iterations per-frame in total. The point of this comparison is to show that our method can achieve comparable results with larger time-steps, allowing us to amortize the per-step costs of grid construction and neighbor finding over more density iterations.

Our results are in good accordance, and a plot of density over the simulation confirms that the level of compression is similar despite the larger time-step for our method (Figure 4). Tables 1 and 2 summarize the performance of our algorithm in a selection of scenarios.

Because we are interested in real-time applications with predictable performance, we set the number of iterations to a fixed value (typically 2-4) rather than solving for a specific error threshold. However, we also show the convergence of our method over multiple iterations in Figure 6.

We implemented our algorithm in CUDA and ran our simulations on an NVIDIA GTX 680. Each stage of our algorithm is fully parallelizable so we are able to take advantage of parallel architectures such as GPUs. For neighbor finding we use the method of [Green 2008]. We also perform particle-solid collision detection on the GPU where we use signed distance fields [Bridson et al. 2006] stored as volume textures.

9 Limitations and Future Work

Occasionally particle stacking along boundaries can occur due to incorrect density estimates when particles are in contact with solids. Recent work by Akinci et al. [2012] would help address this issue.

Jacobi methods only propagate information (in our case position corrections) between a particle's immediate neighbors each iteration. This can lead to slow convergence as the number of particles increases. More sophisticated parallel solvers such as red-black or multi-scale schemes such as [Solenthaler and Gross 2011] should help improve convergence speed.

Because our artificial pressure term is dependent on the spatial resolution and time-step it can be difficult to adjust parameters independently. Decoupling these parameters and making anti-clustering independent from surface tension effects would be important future work.

Position based dynamics is popular for simulating deformable objects such as cloth. We have prototyped two-way interaction between position based cloth and fluid with promising results.

Table 1: *Performance results for several examples. A frame time of 16ms is used in all cases.*

Scene	particles	steps/frame	iters/step	time/step [ms]
Armadillo Splash	128k	2	3	4.2
Dam Break	100k	4	3	4.3
Bunny Drop	80k	4	10	7.8

Table 2: *Breakdown of a frame (percentages) for two examples. Constraint Solve includes collision handling with static objects, and Velocity Update includes vorticity confinement and viscosity calculation.*

Step	Armadillo Splash	Dam Break
Integrate	1	1
Create Hash Grid	8	6
Detect Neighbors	28	28
Constraint Solve	38	51
Velocity Update	25	14

10 Acknowledgments

The authors would like to thank NVIDIA for supporting our research, especially Nuttapong Chentanez, Tae-Yong Kim and Simon Schirm for their valuable feedback, Simon Green for his rendering work, and Adam Moravanszky and Christian Sigg for their encouragement. We also wish to thank the anonymous reviewers for their generous comments and suggestions. The Bunny and Armadillo

models are used courtesy of the Stanford Computer Graphics Laboratory.

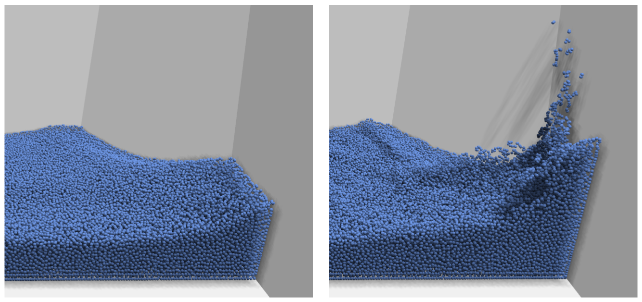


Figure 5: *Dam break scenario at t=6.0, Left: vorticity confinement disabled. Right: vorticity confinement enabled, note the visibly higher splash.*

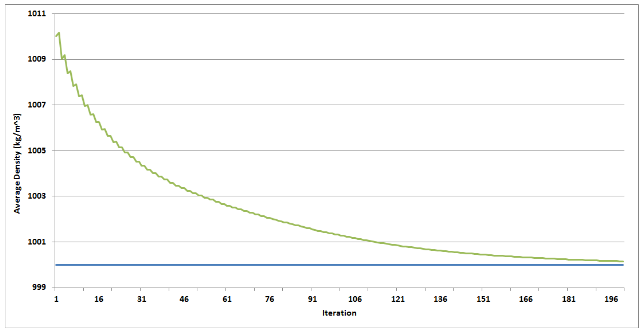


Figure 6: *Convergence of our method over multiple iterations at* $t = 1.0$ *in the dam break scenario.*

References

AKINCI, N., IHMSEN, M., AKINCI, G., SOLENTHALER, B., AND TESCHNER, M. 2012. Versatile rigid-fluid coupling for incompressible sph. *ACM Trans. Graph. 31*, 4 (July), 62:1–62:8.

ALDUÁN, I., AND OTADUY, M. A. 2011. Sph granular flow with friction and cohesion. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, New York, NY, USA, SCA '11, 25–32.

BECKER, M., AND TESCHNER, M. 2007. Weakly compressible sph for free surface flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '07, 209–217.

BELL, N., YU, Y., AND MUCHA, P. J. 2005. Particle-based simulation of granular materials. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM, New York, NY, USA, SCA '05, 77–86.

BODIN, K., LACOURSIERE, C., AND SERVIN, M. 2012. Constraint fluids. *IEEE Transactions on Visualization and Computer Graphics 18*, 3 (Mar.), 516–526.

BRACKBILL, J. U., AND RUPPEL, H. M. 1986. Flip: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *J. Comput. Phys. 65*, 2 (Aug.), 314–343.

BRIDSON, R., FEDKIW, R., AND MÜLLER-FISCHER, M. 2006. Fluid simulation: Siggraph 2006 course notes fedkiw and muller-fischer presenation videos are available from the citation page. In *ACM SIGGRAPH 2006 Courses*, ACM, New York, NY, USA, SIGGRAPH '06, 1–87.

CLAVET, S., BEAUDOIN, P., AND POULIN, P. 2005. Particle-based viscoelastic fluid simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM, New York, NY, USA, SCA '05, 219–228.

FEDKIW, R., STAM, J., AND JENSEN, H. W. 2001. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 15–22.

GREEN, S. 2008. Cuda particles. *nVidia Whitepaper 2*, 3.2, 1.

HONG, J.-M., LEE, H.-Y., YOON, J.-C., AND KIM, C.-H. 2008. Bubbles alive. In *ACM SIGGRAPH 2008 papers*, ACM, New York, NY, USA, SIGGRAPH '08, 48:1–48:4.

LENTINE, M., AANJANEYA, M., AND FEDKIW, R. 2011. Mass and momentum conservation for fluid simulation. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, New York, NY, USA, SCA '11, 91–100.

MONAGHAN, J. J. 1992. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics 30*, 1, 543–574.

MONAGHAN, J. J. 1994. Simulating free surface flows with sph. *J. Comput. Phys. 110*, 2 (Feb.), 399–406.

MONAGHAN, J. J. 2000. Sph without a tensile instability. *J. Comput. Phys. 159*, 2 (Apr.), 290–311.

MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '03, 154–159.

MÜLLER, M., HEIDELBERGER, B., HENNIX, M., AND RATCLIFF, J. 2007. Position based dynamics. *J. Vis. Comun. Image Represent. 18*, 2 (Apr.), 109–118.

RAVEENDRAN, K., WOJTAN, C., AND TURK, G. 2011. Hybrid smoothed particle hydrodynamics. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, New York, NY, USA, SCA '11, 33–42.

SCHECHTER, H., AND BRIDSON, R. 2012. Ghost sph for animating water. *ACM Trans. Graph. 31*, 4 (July), 61:1–61:8.

SMITH, R. 2006. Open dynamics engine v0.5 user guide.

SOLENTHALER, B., AND GROSS, M. 2011. Two-scale particle simulation. *ACM Trans. Graph. 30*, 4 (July), 81:1–81:8.

SOLENTHALER, B., AND PAJAROLA, R. 2009. Predictive-corrective incompressible sph. In *ACM SIGGRAPH 2009 papers*, ACM, New York, NY, USA, SIGGRAPH '09, 40:1–40:6.

VAN DER LAAN, W. J., GREEN, S., AND SAINZ, M. 2009. Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, I3D '09, 91–98.

YU, J., AND TURK, G. 2013. Reconstructing surfaces of particle-based fluids using anisotropic kernels. *ACM Trans. Graph. 32*, 1 (Feb.), 5:1–5:12.

ZHU, Y., AND BRIDSON, R. 2005. Animating sand as a fluid. In *ACM SIGGRAPH 2005 Papers*, ACM, New York, NY, USA, SIGGRAPH '05, 965–972.