

*To appear in ACM TOG 32(4).* **Position Based Fluids**

Miles Macklin

*∗*

Matthias M¨uller

†

NVIDIA

**Abstract**

In ﬂuid simulation, enforcing incompressibility is crucial for real-

ism; it is also computationally expensive. Recent work has im-

proved efﬁciency, but still requires time-steps that are impractical

for real-time applications. In this work we present an iterative den-

sity 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 incom- pressibility and convergence to modern smoothed particle hydro- dynamic (SPH) solvers, but inherits the stability of the geometric, position based dynamics method, allowing large time steps suit- able 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 conﬁne- ment 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;

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

**Keywords:** ﬂuid simulation, SPH, PCISPH, constraint ﬂuids, po- sition based dynamics

**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](#_bookmark4) 1992][1994] is a well known particle based method for ﬂuid simulation. It has many attractive properties: mass-conservation, Lagrangian dis- cretization (particularly useful in games where the simulation do- main is not necessarily known in advance), and conceptual simplic- ity. However, SPH is sensitive to density ﬂuctuations from neigh- borhood deﬁciencies, and enforcing incompressibility is costly due to the unstructured nature of the model. Recent work has im- proved efﬁciency by an order of magnitude [[Solenthaler and Pa- jarola 2009],](#_bookmark4) but small time steps remain a requirement, limiting real-time applications.

**(b)** *Underlying simulation particles*

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

For interactive environments, robustness is a key issue: the simula- tion 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 [[M¨uller et al. 2007].](#_bookmark4) 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.

*∗*

†

e-mail:mmacklin@nvidia.com

e-mail:matthiasm@nvidia.com

**2 Related Work**

M¨uller [[2003]](#_bookmark4) showed that SPH can be used for interactive ﬂuid simulation with viscosity and surface tension, by using a low stiff- ness equation of state (EOS). However to maintain incompressibil- ity, standard SPH or weakly compressible SPH (WCSPH) [[Becker and Teschner 2007]](#_bookmark4) require stiff equations, resulting in large forces that limit the time-step size. Predictive-corrective incompressible SPH (PCISPH) [[Solenthaler and Pajarola 2009]](#_bookmark4) relaxes this time- step restriction by using an iterative Jacobi-style method that accu- mulates pressure changes and applies forces until convergence. It has the advantage of not requiring a user-speci ﬁed stiffness value and of amortizing the cost of neighbor ﬁnding over many density

1

*∗*

*∗*

*⇐***x**

*∗ ∗*

*i*

*i*

*⇐*

1 *∗*

*∗*

∑ **p**

∇

*k*

*j*

*ρ* 

0

*T*

*T*

*T*

2

corrections.

*To appear in ACM TOG 32(4).*

**Algorithm 1** Simulation Loop

Bodin et al [[2012]](#_bookmark4) achieve uniform density ﬂuid by posing incom- pressibility as a system of velocity constraints. They construct a linear complementarity problem using linearized constraint func- tions, 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]](#_bookmark4) use a grid for the pressure solve and transfer ve- locity changes back to particles. FLIP was later extended to in- compressible ﬂow with free surfaces by Zhu and Bridson [[2005].](#_bookmark4) Raveendran et al. [[2011]](#_bookmark4) use a coarse grid to solve for an approx- imately divergence free velocity ﬁeld before an adaptive SPH up- date.

1. :
2. :
3. :
4. :
5. :
6. :
7. :
8. :
9. :
10. :
11. :
12. :
13. :
14. :
15. :

**for all** particles *i* **do**

apply forces **v***i ⇐***v***i* +∆*t* **f***ext* (**x***i* )

predict position **x***i ⇐***x***i* +∆*t* **v***i*

**end for**

**for all** particles *i* **do**

ﬁnd neighboring particles *Ni* (**x***i* )

**end for**

**while** *iter < solverIterations* **do**

**for all** particles *i* **do**

calculate *λi*

**end for**

**for all** particles *i* **do**

calculate ∆**p***i*

perform collision detection and response **end for**

Clavet et al. [[2005]](#_bookmark4) also use a position based approach to simu- late 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.

1. :
2. :
3. :
4. :
5. :

**for all** particles *i* **do** update position **x**

**end for**

**end while**

**for all** particles *i* **do**

+∆**p***i*

Position Based Dynamics [[M¨uller et al. 2007]](#_bookmark4) provides a method for simulating dynamics in games based on Verlet integration. It solves a system of non-linear constraints using Gauss-Seidel itera- tion by updating particle positions directly. By eschewing forces, and deriving momentum changes implicitly from the position up- dates, the typical instabilities associated with explicit methods are

1. :
2. :
3. :
4. :

update velocity **v***i* ∆*t* **x***i −***x***i*

apply vorticity conﬁnement and XSPH viscosity update position **x***i ⇐***x***i*

**end for**

avoided.

**3 Enforcing Incompressibility**

To enforce constant density we solve a system of non-linear con- straints, with one constraint per-particle. Each constraint is a func- tion of the particle’s position and the positions of its neighbors, which we refer to collectively as **p**1 *, · · · ,* **p***n* . Following [[Bodin et al. 2012]](#_bookmark4) the density constraint on the *ith* particle is deﬁned using an equation of state:

*ρi*

*Ci* (**p**1 *, ...,* **p***n* ) = *−*1*,* (1)

*ρ*0

where *ρ*0 is the rest density and *ρi* is given by the standard SPH

[[Monaghan 1992]](#_bookmark4) gives the SPH recipe for the gradient of a func- tion deﬁned on the particles. Applying this, the gradient of the constraint function [(1)](#_bookmark1) with respect to a particle *k* is given by:

1

∇**p***k Ci* = ∇ *k W* (**p***i −***p** *j , h*) (7) *ρ*0 *j*

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



1 ∑ **p** *W* (**p***i −***p** *j , h*) if *k* = *i*

∇**p***k Ci* = (8) *−*∇**p***k W* (**p***i −***p** *j , h*) if *k* = *j*

Plugging this into [(6)](#_bookmark1) and solving for *λ* gives

density estimator:

*ρi*

=

∑

*j*

*m*

*j*

*W* (**p***i*

*−***p**

*j*

*, h*)*.* (2)

*λi* = *−*

*Ci* (**p**

∑*k*

1 *, ...,* **p***n* ) 2

∇**p***k Ci*

(9)

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 gradi- ent calculation, as in [[M¨uller et al. 2003].](#_bookmark4)

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 ∆**p** that satisﬁes the constraint:

*C* (**p** +∆**p**) = 0 (3)

This is found by a series of Newton steps along the constraint gra- dient:

which is the same for all particles in the constraint.

Because the constraint function [(1)](#_bookmark1) is non-linear, with a vanish- ing gradient at the smoothing kernel boundary, the denominator in equation [(9)](#_bookmark1) causes instability when particles are close to separat- ing. As in PCISPH this can be solved by pre-computing a conser- vative corrective scale based on a reference particle conﬁguration with a ﬁlled neighborhood.

Alternatively, constraint force mixing (CFM) [[Smith 2006]](#_bookmark4) 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)](#_bookmark1) to

*C* (**p** +∆**p**) *≈C* (**p**) +∇*C* ∇*C λ*+ *ελ*= 0*.* (10)

∆**p** *≈* ∇*C* (**p**)*λ*

(4)

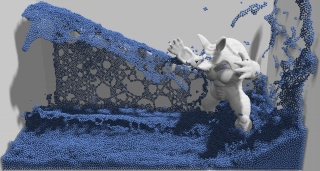
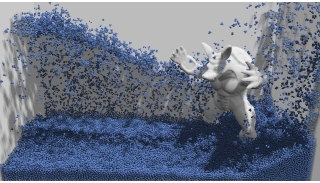
*C* (**p** +∆**p**) *≈C* (**p**) +∇*C* ∆**p** = 0 (5) *≈C* (**p**) +∇*C* ∇*C λ*= 0*.* (6)

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

*Ci* (**p**1 *, ...,* **p***n* )

*λi* = *− ,* (11) ∑*k* ∇**p***k Ci* +*ε*

2



**f**

*new*

∑ *i j*

*To appear in ACM TOG 32(4).*

and the total position update ∆**p***i* including corrections from neigh-

This purely repulsive term keeps particle density slightly lower than

bor particles density constraint (*λ*

*j*

) is

the rest density. Consequently, particles pull their neighbors in-

∆**p***i*

=

1

*ρ*0

∑

*j*

*λi*

+*λ*

*j*

∇*W* (**p***i*

*−***p**

*j*

*, h*)*.* (12)

wards causing surface tension-like effects similar to the ones de- scribed in [[Clavet et al. 2005].](#_bookmark4) 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 of- ten undesirable. Fedkiw et al. [[2001]](#_bookmark4) introduced vorticity conﬁne- ment to computer graphics to address numerical dissipation in the simulation of smoke, which was later extended to energy conserv- ing ﬂuid simulation in [[Lentine et al. 2011].](#_bookmark4) In Bubbles Alive, Hong et al. [[2008]](#_bookmark4) show how vorticity conﬁnement can be used in a hy- brid 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 (Fig- ure [5).](#_bookmark4) This requires ﬁrst calculating the vorticity at a particle’s lo- cation, for which we use the estimator given in [[Monaghan 1992]:](#_bookmark4)

*ω*

*i*

= ∇ *×***v** =

∑

*j*

**v***i j*

*×*∇**p**

*j*

*W* (**p***i*

*−***p**

*j*

*, h*) (15)

where **v***i j*

= **v**

*j*

*−* **v**

*i*

. Once we have the vorticity we calculate a

corrective force using the location vector **N** =

*η*

*|η|*

with *η*= ∇*|ω|*

*i*

*vorticity*

*i*

= *ε*(**N** *×ω*

*i*

) *.* (16)

**Figure 2:** *Armadillo Splash, Top: particle clumping due to neigh- bor deﬁciencies, Bottom: with artiﬁcial pressure term, note the im- proved 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).](#_bookmark2) This may be avoided by clamping pressures to be non-negative, but at the cost of reduced particle cohesion. Clavet et al. [[2005]](#_bookmark4) use a second ’near pressure ’term, while Alduan and Otaduy [[2011]](#_bookmark4) use discrete element (DEM) forces [[Bell et al. 2005]](#_bookmark4) to push apart particles closer than half the smoothing kernel width. Schechter and Bridson [[2012]](#_bookmark4) place ghost particles around the free surface to ensure consistent density estimates.

We follow the approach of [[Monaghan 2000]](#_bookmark4) which adds an arti- ﬁcial pressure speciﬁed in terms of the smoothing kernel itself as

Unlike [[Hong et al. 2008]](#_bookmark4) we do not use normalized *ω* as this would increase vorticity indiscriminately. Instead we use the unnormal- ized value, which only adds vorticity where it already exists, as in [[Fedkiw et al. 2001].](#_bookmark4)

In addition, we apply XSPH viscosity [[Schechter and Bridson 2012],](#_bookmark4) which is important for coherent motion. The parameter *c* is typically chosen to be 0.01 in our simulations:

**v***i* = **v***i* +*c* **v** *· W* (**p***i −***p** *j , h*) (17)

*j*

**6 Algorithm**

Our simulation loop is outlined in Algorithm [1.](#_bookmark0) It is similar to the original Position Based Dynamics update except that each con- straint is solved independently in a Jacobi fashion, rather than through sequential Gauss-Seidel iteration. We perform collision de- tection against solids as part of the constraint solving loop.

*scorr* = *−k*

*W* (**p***i −***p** *j , h*) *W* (∆**q***, h*)

*n*

*,* (13)

We recompute particle neighborhoods once per-step and re- calculate distance and constraint values each solver iteration. This

where ∆**q** is a point some ﬁxed distance inside the smoothing kernel radius and *k* is a small positive constant. *|*∆**q***|* = 0*.*1*h · · ·* 0*.*3*h*, *k* = 0*.*1 and *n* = 4 work well. We then include this term in the particle position update as

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 pres-

∆**p***i*

=

1

*ρ*0

∑

*j*

*λ*

*i*

+*λ*

*j*

+*s*

*corr*

∇*W* (**p***i*

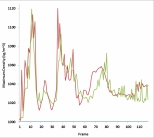
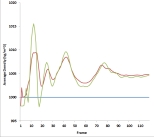
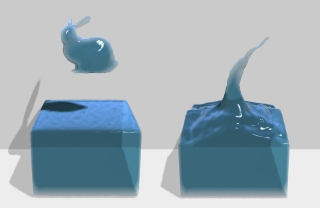
*−***p**

*j*

*, h*)*.* (14)

sure forces are applied. Our algorithm considers only the current particle positions (not accumulated pressure), so this does not oc- cur.

3



*To appear in ACM TOG 32(4).*

Because we are interested in real-time applications with predictable performance, we set the number of iterations to a ﬁxed value (typ- ically 2-4) rather than solving for a speciﬁc error threshold. How- ever, we also show the convergence of our method over multiple iterations in Figure [6.](#_bookmark4)

We implemented our algorithm in CUDA and ran our simulations on an NVIDIA GTX 680. Each stage of our algorithm is fully par- allelizable so we are able to take advantage of parallel architec- tures such as GPUs. For neighbor ﬁnding we use the method of [[Green 2008].](#_bookmark4) We also perform particle-solid collision detection on the GPU where we use signed distance ﬁelds [[Bridson et al. 2006]](#_bookmark4) stored as volume textures.

**9 Limitations and Future Work**

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

**(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.*

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]](#_bookmark4) would help address this issue.

Jacobi methods only propagate information (in our case position corrections) between a particle’s immediate neighbors each itera- tion. 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]](#_bookmark4) should help improve convergence speed.

Because our artiﬁcial pressure term is dependent on the spatial res- olution and time-step it can be difﬁcult to adjust parameters inde- pendently. 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 ob- jects such as cloth. We have prototyped two-way interaction be- tween 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.*

**7 Rendering**

**Scene**

particles steps/frame iters/step time/step [ms]

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],](#_bookmark4) and the surface is reconstructed using a method based on the screen-space ﬁltering presented in [[van der Laan et al. 2009].](#_bookmark4)

**8 Results**

We tested our algorithm by dropping a liquid bunny into a pool of

**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 calcu- lation.*

water (Figure [3)](#_bookmark3) and compared our results with a PCISPH imple-

**Step**

Armadillo Splash Dam Break

mentation. For this scenario PCISPH is not stable with less than 10 sub-steps per frame (∆*t* = 0*.*0016*s*). In contrast our algorithm is stable with a single step (∆*t* = 0*.*016*s*).

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).](#_bookmark3) Tables [1](#_bookmark3) and [2](#_bookmark3) sum- marize the performance of our algorithm in a selection of scenarios.

**Integrate** 1 1

1. **reate Hash Grid** 8 6
2. **etect Neighbors** 28 28

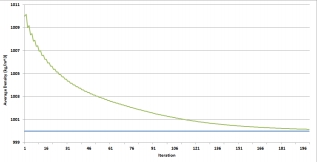
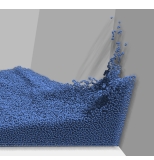
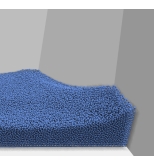
**Constraint Solve** 38 51

**Velocity Update** 25 14

**10 Acknowledgments**

The authors would like to thank NVIDIA for supporting our re- search, 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 encour- agement. We also wish to thank the anonymous reviewers for their generous comments and suggestions. The Bunny and Armadillo

4



*To appear in ACM TOG 32(4).*

models are used courtesy of the Stanford Computer Graphics Lab- oratory.

**Figure 5:** *Dam break scenario at t=6.0, Left: vorticity conﬁne- ment disabled. Right: vorticity conﬁnement enabled, note the visi- bly higher splash.*

**Figure 6:** *Convergence of our method over multiple iterations at t* = 1*.*0 *in the dam break scenario.*

**References**

A KINCI , N., I HMSEN , M., A KINCI , G., S OLENTHALER , B., AND T ESCHNER , M. 2012. Versatile rigid-ﬂuid coupling for incom- pressible sph. *ACM Trans. Graph. 31*, 4 (July), 62:1 –62:8.

1. ´ , I., AND O TADUY, 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.
2. , M., AND T ESCHNER , M. 2007. Weakly compressible sph for free surface ﬂows. In *Proceedings of the 2007 ACM SIG- GRAPH/Eurographics symposium on Computer animation*, Eu- rographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA ’07, 209–217.

B ELL , N., Y U , Y., AND M UCHA , P. J. 2005. Particle-based sim- ulation of granular materials. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM, New York, NY, USA, SCA ’05, 77–86.

B ODIN , K., L ACOURSIERE , C., AND S ERVIN , M. 2012. Con- straint ﬂuids. *IEEE Transactions on Visualization and Computer Graphics 18*, 3 (Mar.), 516–526.

B RACKBILL , J. U., AND R UPPEL , 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.

B RIDSON , R., F EDKIW, R., AND M ULLER¨ -F ISCHER , 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.

C LAVET, S., B EAUDOIN , P., AND P OULIN , P. 2005. Particle- based viscoelastic ﬂuid simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer ani- mation*, ACM, New York, NY, USA, SCA ’05, 219–228.

F EDKIW, R., S TAM , J., AND J ENSEN , H. W. 2001. Visual sim- ulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* , ACM, New York, NY, USA, SIGGRAPH ’01, 15–22.

G REEN , S. 2008. Cuda particles. *nVidia Whitepaper 2*, 3.2, 1. H ONG , J.-M., L EE , H.-Y., YOON , J.-C., AND K IM , C.-H. 2008.

Bubbles alive. In *ACM SIGGRAPH 2008 papers* , ACM, New York, NY, USA, SIGGRAPH ’08, 48:1–48:4.

L ENTINE , M., A ANJANEYA , M., AND F EDKIW, R. 2011. Mass and momentum conservation for ﬂuid simulation. In *Proceed- ings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, New York, NY, USA, SCA ’11, 91– 100.

M ONAGHAN , J. J. 1992. Smoothed particle hydrodynamics. *An- nual Review of Astronomy and Astrophysics 30*, 1, 543–574.

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

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

M ULLER¨ , M., C HARYPAR , D., AND G ROSS , M. 2003. Particle- based ﬂuid simulation for interactive applications. In *Proceed- ings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA ’03, 154–159.

M ULLER¨ , M., H EIDELBERGER , B., H ENNIX , M., AND R AT- CLIFF , J. 2007. Position based dynamics. *J. Vis. Comun. Image Represent. 18*, 2 (Apr.), 109–118.

R AVEENDRAN , K., W OJTAN , C., AND T URK , G. 2011. Hybrid smoothed particle hydrodynamics. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Ani- mation*, ACM, New York, NY, USA, SCA ’11, 33–42.

S CHECHTER , H., AND B RIDSON , R. 2012. Ghost sph for animat- ing water. *ACM Trans. Graph. 31*, 4 (July), 61:1–61:8.

S MITH , R. 2006. Open dynamics engine v0.5 user guide.

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

S OLENTHALER , B., AND PAJAROLA , R. 2009. Predictive- corrective incompressible sph. In *ACM SIGGRAPH 2009 pa- pers*, ACM, New York, NY, USA, SIGGRAPH ’09, 40:1–40:6.

VAN DER L AAN , W. J., G REEN , S., AND S AINZ , 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.

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

Z HU , Y., AND B RIDSON , R. 2005. Animating sand as a ﬂuid. In *ACM SIGGRAPH 2005 Papers*, ACM, New York, NY, USA, SIGGRAPH ’05, 965–972.

5