Highly Adaptive Liquid Simulations on Tetrahedral Meshes

Ryoichi Ando*
Kyushu University

Nils Thürey†
ScanlineVFX

Chris Wojtan‡
IST Austria

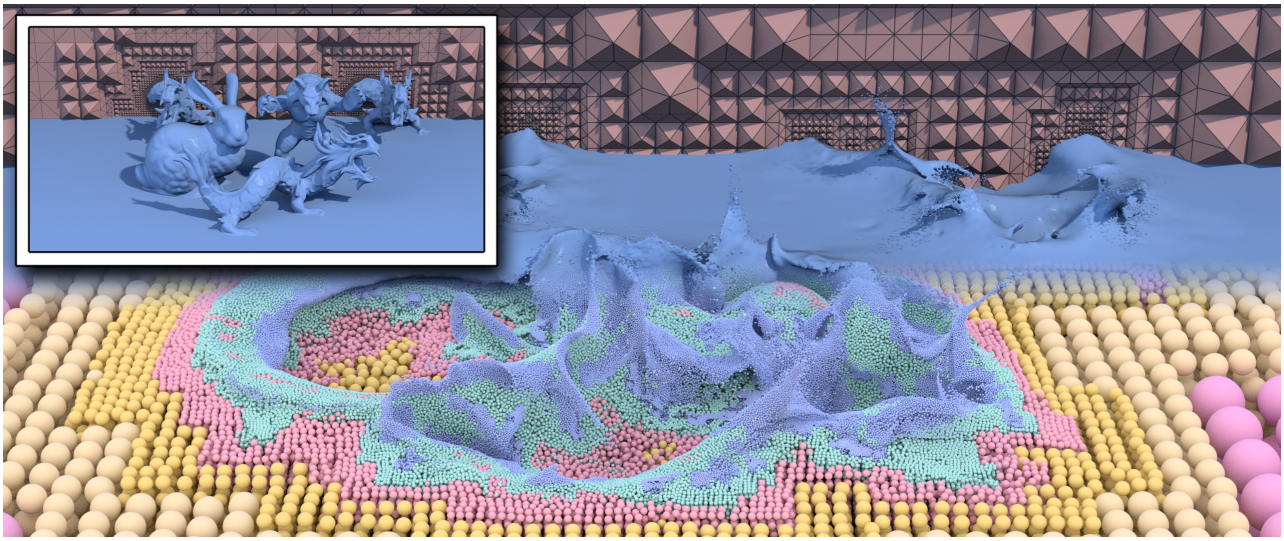

Figure 1: *Our adaptive simulation framework allows us to efficiently simulate highly detailed splashes on large open surfaces. In this case, maximum BCC mesh resolutions from 8 to 1024 cells were used, leading to strong horizontal grading along the surface.*

Abstract

We introduce a new method for efficiently simulating liquid with extreme amounts of spatial adaptivity. Our method combines several key components to drastically speed up the simulation of large-scale fluid phenomena: We leverage an alternative Eulerian tetrahedral mesh discretization to significantly reduce the complexity of the pressure solve while increasing the robustness with respect to element quality and removing the possibility of locking. Next, we enable subtle free-surface phenomena by deriving novel second-order boundary conditions consistent with our discretization. We couple this discretization with a spatially adaptive Fluid-Implicit Particle (FLIP) method, enabling efficient, robust, minimally-dissipative simulations that can undergo sharp changes in spatial resolution while minimizing artifacts. Along the way, we provide a new method for generating a smooth and detailed surface from a set of particles with variable sizes. Finally, we explore several new sizing functions for determining spatially adaptive simulation resolutions, and we show how to couple them to our simulator. We combine each of these elements to produce a simulation algorithm that is capable of creating animations at high maximum resolutions while avoiding common pitfalls like inaccurate boundary conditions and inefficient computation.

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

Keywords: fluid simulation, tetrahedral discretization, adaptivity

*E-mail:and@verygood.aid.design.kyushu-u.ac.jp
†E-mail:nils.thuerey@scanlinevfx.com
‡E-mail:wojtan@ist.ac.at

Links: DL PDF

1 Introduction

This paper aims to produce fluid simulations with a high degree of spatial adaptivity. We desire to enable a simulator to focus its computational resources on the visually interesting regions of a fluid flow, while remaining computationally efficient and avoiding common artifacts due to a spatially adaptive pressure solve.

Previous approaches have made great strides towards this goal, but they often exhibit visual artifacts, a lack of computational robustness, or an unacceptably hefty computational expense. The groundbreaking work of Losasso et al. [2004] introduced an octree for spatial adaptivity, but it suffers from spurious flows at T-junctions. Finite volume methods [Batty et al. 2010] repair these spatial artifacts at the expense of solving a significantly larger system of equations and sacrificing computational stability near poorly-shaped elements. Furthermore, many existing methods still are not truly spatially adaptive in the sense that their computational complexity is still tied to a uniform grid or spatial parameter.

We introduce a combination of techniques that successfully makes

adaptive fluid simulation practical at large scales. We first reduce memory and computational costs by switching from a finite volume method to a discretization with a significantly smaller linear system for the pressure solve, which has the side effect of increasing the simulator's robustness to poor-quality elements and effectively preventing locking artifacts. We next derive second-order Dirichlet boundary conditions consistent with our discretization to benefit from the subtle surface dynamics associated with an accurate pressure solve. We combine this robust and efficient tetrahedral mesh-based fluid simulator with a spatially adaptive method for sampling particles for FLIP-based velocity advection, giving us a method free from any single spatial resolution.

In addition to our adaptive FLIP simulator, we also introduce a new method for computing a surface from a distribution of particles with variable radii. We found that this method out-performs previous methods in cases of extreme spatial adaptivity by exhibiting smoother surfaces without sacrificing detail.

Our fluid simulator works well with spatially adaptive tetrahedral meshes, but it is another question to decide exactly how these adaptive meshes should be generated. We investigate various methods for generating these adaptive meshes by experimenting with several *sizing functions*, allowing us to precisely dictate where simulation detail should occur. Some examples are a surface curvature-based metric that adds detail only where needed on the fluid surface, a turbulence metric that adds detail only where interesting fluid motion occurs, and a visibility metric that adds detail only in front of a virtual camera.

Concretely, the contributions of our work are:

- a novel tetrahedral discretization of the pressure projection step that is efficient to solve and robust to poor-quality elements;
- an accurate treatment of second-order boundary conditions within the tetrahedral mesh;
- a new technique for extracting a smooth surface from particles with varying radii;
- and the inclusion of a flexible sizing function to focus computational resources on important areas of the flow with minimal overhead.

These contributions work together to produce a practical fluid simulator that exhibits low computational and memory complexity, fewer visual artifacts, and a high effective simulation resolution.

2 Related Work

Our work is based on the *Fluid-Implicit Particle* (FLIP) method introduced to the computer graphics community by Zhu and Bridson [2005], which arguably represents the state-of the art for detailed and robust liquid simulations. The algorithm still follows the general ideas of the *Stable Fluid* solver [Stam 1999], and can be readily combined with second-order treatment of free surface boundary conditions [Enright et al. 2003]. FLIP derives its success from the fact that it uses particles to compute an accurate, non-diffusive transport of flow quantities, in combination with a grid-based solve to accurately enforce constraints for mass conservation. The FLIP algorithm is heavily used in the special effects industry, and recent advances have introduced accurate coupling with obstacles [Batty et al. 2007], highly viscous materials [Batty and Bridson 2008], and two-phase flows [Boyd and Bridson 2012].

Traditionally, Cartesian grids are very popular for fluid simulations. The *Marker-And-Cell* (MAC) approach [Harlow and Welch 1965], which stores velocity components at cell faces and pressure samples at cell centers, results in discretizations with good properties in terms of stability and accuracy. An inherent difficulty is that simulations on regular grids become prohibitively expensive for large resolutions. Thus, many works have proposed methods to focus the computations on regions that are of particular interest. One example are octrees, which were used by Losasso et al. [2004; 2005] to refine the computational grid in a controllable way. This approach, however, suffers from numerical diffusion and an inconsistent discretization near the tree's T-junctions. Targeting a similar direction as our work, Hong et al. [2009] and Ando et al. [2012] have demonstrated methods to adapt the resolution of FLIP particles in a simulation. Both methods, in contrast to ours, focus on static computational grids and are restricted to smaller differences in particle size.

Although Cartesian grids are widely used, they are limited in their flexibility to adapt to a simulation setup. Because of this, tetrahedral grids are popular for methods targeting adaptivity. In combination with a suitable method to discretize the problem at hand, they allow for very flexible computational grids. One example is the work of Klingner et al. [Klingner et al. 2006] which demonstrated the use of a Stable Fluids based solver for tetrahedral grids conforming to object boundaries. Another example is the non-linear fluid solver developed by Mullen et al. [2009], which leads to an energy conserving solve. Unlike these methods, we make use of a non-conforming grid with Body-Centered Cubic (BCC) lattices. These meshes were also used by Chentanez et al. [2007] and by Batty et al. [2010] for liquid simulations. We will denote this class of algorithms as *Finite Volume Methods* (FVM). These methods are primarily suitable for uniformly sampled particles, and we will demonstrate in Section 7 that their placement of pressure samples at tetrahedral circumcenters leads to numerical problems in combination with graded BCC meshes.

Another direction of research performs fluid simulations based on arbitrary elements. Clausen et al. [2013] and Misztal et al. [2010] have proposed a method to simulate liquids with a computational grid conforming to a triangulation of a liquid surface. Both methods lead to an increased computational cost in comparison to the more efficient tetrahedral BCC meshes. Sin et al. [2009] proposed an alternative method for hybrid Lagrangian-Eulerian solvers which combines a Voronoi-based pressure solver and particles. Using this Vornoi-based approach for tetrahedral meshes would yield a pressure matrix similar to ours. Like our method, Brochu et al. [2010] used this discretization in combination with embedded second-order boundary conditions. Both of these approaches discretize velocities with per-face flux values, while we store velocity vectors at cell barycenters.

Adaptive simulations have also been explored in the context of SPH simulations without Eulerian grids. The work of [Adams et al. 2007] shares similarities with our approach, as it is able to simulate a wider range of particle radii, and it proposes a surface reconstruction method in the adaptive setting. We will show in Section 5 that our surface creation method results in surfaces with fewer visual artifacts. Additionally, a robust and efficient method for adaptive SPH simulations was introduced by Solenthaler et al. [2011], but this work primarily targets the coupling of two different particle resolutions.

Several other methods have been proposed to reconstruct smooth surfaces around collections of particles without orientation. One approach that is commonly used is to compute a signed distance function with averaged particle radii and centroids [Zhu and Bridson 2005]. A variant of this approach, taking into account information about the spatial variance of the particle's neighborhood was proposed by Yu et al. [2010]. Both methods primarily target particles with constant radius. More recently, a level-set based method was proposed that computes a constrained optimization with bihar-

monic smoothing [Bhattacharya et al. 2011]. However, such an optimization would be complicated to apply in our unstructured setting. In contrast to these methods, our approach for surface creation computes the union of convex hulls around triplets of particles, which leads to a smooth and closed surface around a collection of arbitrarily sized particles.

3 Fluid Solver

The aim of our method is to solve the *Navier-Stokes* equations, which for incompressible, Newtonian, inviscid flows can be written as $\rho D\mathbf{u}/Dt = -\nabla p + \mathbf{f}$, with the additional constraint $\nabla \cdot \mathbf{u} = 0$ to enforce a divergence-free velocity field. Here, $\mathbf{u}$, p and $\mathbf{f}$ denote velocity, pressure and external forces, respectively, while D/Dt denotes the material derivative. The density ρ is constant in our case. We solve these equations using operator splitting [Stam 1999], and a level set $\phi(\mathbf{x}) = 0$ defines the position of the liquid-gas interface.

Spatial Discretization An inherent strength of the FLIP algorithm is its hybrid nature. The motion of the fluid is computed in a Lagrangian manner using particles, while the pressure projection step is computed on an Eulerian grid. We will now describe how we compute the pressure projection using a tetrahedral discretization. This projection of the velocities into a divergence-free state can be formulated as the Poisson problem

$$\frac{\Delta t}{\rho}\nabla^2 p = \nabla \cdot \boldsymbol{u}^*, \tag{1}$$

where $\boldsymbol{u}^*$ denotes an intermediate velocity after the advection. In the following, however, we prefer an alternate view that looks at this problem from an energy minimization perspective: we want to compute the minimal change in kinetic energy necessary to reach a divergence-free state of the flow similar to [Batty et al. 2007]. This can be formulated as:

$$p = \arg\min_p \int_\Omega \frac{1}{2}\, ||\boldsymbol{u}^* - \frac{\Delta t}{\rho}\nabla p||^2\, \rho dV \tag{2}$$

Here, Ω represents the domain of the computational grid, and we choose to discretize this space using tetrahedral cells. This has the advantage of giving us a natural way to handle cells of different size, while yielding a consistent discretization of the differential operators involved. We store pressure samples at the nodes of the tetrahedral mesh, while velocities are stored at cell centers. This configuration is illustrated in Figure 2. Note that by assuming a piece-wise constant velocity and a linear change of pressure within a cell, this setup results in a constant pressure gradient per tetrahedron, by construction. In the following, we denote the number of cells with m and the number of nodes with n, and we indicate discretized quantities with caret notation. Based on this representation we can discretize Eq. (2) with

$$\hat{p} = \arg\min_{\hat{p}} \sum_i^m \frac{1}{2}\, ||\hat{\boldsymbol{u}}_i^* - \frac{\Delta t}{\rho}[\nabla]\hat{p}||^2\, \rho V_i\,, \tag{3}$$

where we denote the volume of a cell with V_i and the discretized gradient operator with $[\nabla]$. It consists of a $m \times n$ matrix, computing a per-tetrahedron gradient from nodal values. Consequently, we define the divergence operator to be the transpose of the discretized gradient. Before we go ahead to define $[\nabla]$, we want to outline the rest of the steps for our pressure solve. We solve equation Eq. (3) with the commonly used least squares technique, yielding

$$\frac{\Delta t}{\rho}[\nabla]^T V[\nabla]\hat{p} = [\nabla]^T V\hat{\boldsymbol{u}}^*, \tag{4}$$

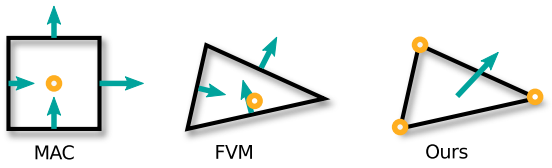

Figure 2: *Our discretization compared to previous approaches: the MAC grid stores velocity components normal to faces and pressure values at the center. The FVM discretizations follow along these lines and store velocities normal to faces and pressure values at the circumcenter of a cell. In our approach we store a 3-component velocity vector at the barycenter of a cell and pressure values at each node.*

where V denotes a matrix containing the V_i as diagonal entries. The $[\nabla]^T[\nabla]$ matrix-matrix multiplication results in a square $n \times n$ matrix, which is symmetric and positive definite. In the following, we will denote the matrix on the left hand side of Eq. (4) as A. Given appropriate boundary conditions, we can use standard tools, such as a commonly used pre-conditioned conjugate gradient solver to compute a solution (we use the one suggested in [Bridson 2008]).

Now, all that is left to construct the left-hand side matrix and the right-hand side terms for Eq. (4) is to define $[\nabla]$. As we assume a linear change of the pressure for each cell, we can use simple barycentric interpolation to retrieve the pressure $\hat{p}$ at a position inside a cell. Given the nodal pressure values $p_{1..4}$ and barycentric weights $\sigma_{1..4}$ this means $\hat{p} = \sigma_1 p_1 + \sigma_2 p_2 + \sigma_3 p_3 + \sigma_4 p_4$. In line with finite element methods using linear elements, we define the gradient based on the partial derivatives of the barycentric interpolation. E.g., the first component of the gradient for a cell is computed with

$$\frac{\partial}{\partial x}\hat{p} = \frac{\partial \sigma_1}{\partial x}p_1 + \frac{\partial \sigma_2}{\partial x}p_2 + \frac{\partial \sigma_3}{\partial x}p_3 + \frac{\partial \sigma_4}{\partial x}p_4\,. \tag{5}$$

To set up the final linear system of Eq. (4) for the pressure solve, we loop over all tetrahedra to compute the derivatives of the barycentric interpolation, adding their contributions to the global matrix.

In contrast to previous work, our pressure solve is a linear system that has n degrees of freedom, n being the number of nodes in the tetrahedral mesh. For our BCC mesh, n is in practice smaller than the number of tetrahedra m (by a factor of 6 on average). A direct implication of this smaller linear system is that it is faster to solve. A second, less obvious implication of the smaller linear system is that it effectively prevents artifacts known as *locking*. These artifacts are commonly observed in finite element methods for problems in elasticity. Different methods have been proposed to circumvent these problems, e.g., using linear elements for pressure instead of piece-wise constant ones [Irving et al. 2007]. Other works explicitly smooth the pressure field to reduce locking problems [Misztal et al. 2010]. In general, locking can be observed if the pressure basis can represent more, and higher-frequency, functions than the basis for the velocity. Thus, choosing a more restrictive basis for pressure, as in [Irving et al. 2007], or explicitly removing high-frequency information from the pressure [Misztal et al. 2010], reduces the chance of locking. Our method, by construction, has more degrees of freedom for representing velocity fields than pressure fields. Although we cannot prove that a local configuration over-constraining the velocities will never occur, the larger number of degrees of freedom for our velocities effectively prevents locking artifacts, and we have not encountered any in our tests.

Boundary Conditions Second-order boundary conditions are a central component for accurate and visually appealing simulations with non-conforming grids. Achieving second-order accuracy for

obstacle boundary conditions is straightforward with our discretization: we can rely on the formulation of previous work [Batty et al. 2007], and set the volume of a cell V_i in Eq. (4) to the volume that is filled with fluid.

For the free surface, we have to ensure that the Dirichlet boundary condition $p = 0$ is satisfied at the interface position. Usually, this means computing a pressure value for nodes outside of the liquid so that a linear interpolation along an edge of a cell gives zero at the correct position [Enright et al. 2005; Lew and Buscaglia 2008]. Considering two pressure samples along an edge, we'll denote values inside the air with a G subscript, and values inside the liquid with an L subscript in the following. For a Cartesian MAC grid, the ghost pressure value p_G is given by $p_G = p_L\phi_G/\phi_L$. In our case, however, this approach does not yield the desired result. The reason is that our velocity samples are not in line with the direct connections of the pressure samples – they are not locally orthogonal to each other. Instead, we have to ensure the boundary conditions result in the correct pressure value at the cell center. In the following we will show how to derive suitable free-surface boundary conditions to ensure second-order accuracy within our framework.

In order to achieve accurate and smooth surface motions with our method, we compute the ghost pressure values p_G with a linear combination of liquid pressure values as:

$$p_G = w_1 p_1 + w_2 p_2 + w_3 p_3 \ , \tag{6}$$

where w_n and p_n denote unknown coefficients and adjacent liquid pressures in the same tetrahedron. Note that for p_i that are not inside of the liquid, we set $w_i = 0$. In line with the traditional ghost fluid method, we define p_G uniquely for each tetrahedron. To handle the most general case, let's suppose that $p_{1..3}$ are all liquid pressure values. Once we have a value for p_G, we can compute a pressure gradient for the tetrahedron and update the velocity at its center with:

$$\hat{\boldsymbol{u}}_{\text{new}} = \hat{\boldsymbol{u}}^* - \frac{\Delta t}{\rho}[\nabla]\begin{bmatrix} p_G & p_1 & p_2 & p_3 \end{bmatrix}^T \tag{7}$$

In order to do this we need to compute the coefficients w_n. p_G can be rewritten in terms of a barycentric interpolation of the three values in the liquid as:

$$p_G = \tilde{p}_L\phi_G/\tilde{\phi}_L \tag{8}$$

with $\tilde{\phi}_L = \theta_1\phi_1 + \theta_2\phi_2 + \theta_3\phi_3$, and $\tilde{p}_L = \theta_1 p_1 + \theta_2 p_2 + \theta_3 p_3$. Here the θ_n are a set of barycentric coordinate coefficients such that $\theta_1+\theta_2+\theta_3 = 1$, and a tilde superscript denotes a value interpolated with the barycentric weights. Substituting Eq.8 into Eq.6 yields

$$w_n = \theta_n\phi_G/\tilde{\phi}_L \ . \tag{9}$$

That means the values w_n are determined by those of the θ_n coefficients, which we will compute in the following. Note that, theoretically, θ_n could take any values as long as they add up to one. Before embedding the boundary conditions, the matrix entries of the pressure solve for a single tetrahedron are, according to Eq. (4), given by $\frac{\Delta t}{\rho}[\nabla]^T V[\nabla]\mathbf{p} = \mathbf{b}$. The computation of the ghost fluid values is independent of the right-hand side $\mathbf{b}$, so we will restrict the discussion to the left hand side. We denote the components of the local symmetric 4×4 matrix on the left hand side with:

$$\begin{bmatrix} \lambda_1 & \alpha & \beta & \gamma \\ \alpha & \lambda_2 & a & b \\ \beta & a & \lambda_3 & c \\ \gamma & b & c & \lambda_4 \end{bmatrix} \tag{10}$$

Assuming, without loss of generality, that the first vertex is the one outside of the liquid volume, we embed the boundary condition into M based on the w_n coefficients. Then the first row of the system is changed to:

$$\begin{bmatrix} 1 & -w_1 & -w_2 & -w_3 \\ \alpha & \lambda_2 & a & b \\ \beta & a & \lambda_3 & c \\ \gamma & b & c & \lambda_4 \end{bmatrix} \begin{bmatrix} p_G \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} . \tag{11}$$

We can extract two constraints for each θ_n from this form, which, together with the barycentric coefficient constraint, give us a 3×3 matrix M' that can be inverted analytically. A full derivation of these steps can be found in Appendix A. With this analytic expression we can compute the ghost pressure coefficients as:

$$\boldsymbol{w} = \frac{\phi_G/\tilde{\phi}_L}{\alpha+\beta+\gamma}\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \tag{12}$$

This boundary condition ensures second-order accuracy while maintaining symmetry when it is assembled into the matrix of Eq. (4). If the quality of a tetrahedron is good, $0 < \theta_n < 1$ is guaranteed. In this case, the resulting matrix is symmetric positive-definite and can be easily inverted by the commonly used preconditioned conjugate gradient methods. However, positive off-diagonal terms of the matrix can result in values of θ outside of the range $[0, 1]$, leading to an indefinite linear system. In these cases we consider the tetrahedron to have a poor quality. When using ghost fluid boundary conditions with a regular MAC grid, it is common practice to clamp small values in the denominator of Eq. (9) to prevent ill-conditioned pressure matrices. Effectively, this means reverting to first-order accuracy when second-order accuracy is intractable. We implement a similar step in our algorithm to overcome numerical problems resulting from badly shaped cells. We check whether the ghost fluid boundary conditions would violate diagonal dominance of an equation in our linear system. If we detect such a case, we smoothly transition to first order accuracy. Specifically, when we have computed M', we check if a resulting diagonal term $M'_{i,i}$ is smaller than $\varphi\lambda_{2..4}$. Here, φ denotes a tolerance factor that we set to $\varphi = 0.25$. Whenever we detect such a case, we compute a coefficient k with

$$k = \min\Big(\frac{\varphi-1}{w_2\alpha}\lambda_2, \frac{\varphi-1}{w_3\beta}\lambda_3, \frac{\varphi-1}{w_4\gamma}\lambda_4\Big) , \tag{13}$$

and multiply each w_n with k when embedding. Note that this scaling does not break the symmetry of the resulting linear system. More specifically, for $k = 1$ this yields full second-order accuracy, while for badly shaped tetrahedra the resulting $k = 0$ means that we revert to the standard rounding strategy of a first order accurate method. With our BCC mesh, all regular BCC tetrahedra have very good quality and valid θ_n values. The graded BCC tetrahedra, on the other hand, can be of lower quality and can require the use of Eq. (13). Luckily, in our tests these tetrahedra make up only a very small fraction of the mesh.

Velocity Interpolation The FLIP advection step traces particles based on the velocities from the Eulerian grid. For this we need to construct a continuous velocity field based on the discrete values in our tetrahedral mesh. As we store velocities at the cell centers, the interpolation would ideally use the dual mesh consisting of the Voronoi cells of each node [Brochu et al. 2010]. Unfortunately, performing interpolations within arbitrary Voronoi cells would be expensive and require a large amount of computation compared to the other steps of our simulator. Instead, we have found the following approach to yield high speed and good accuracy: we first interpolate the centered velocities to the nodes, similar to [Chentanez et al. 2007]. Instead of interpolating these averaged values

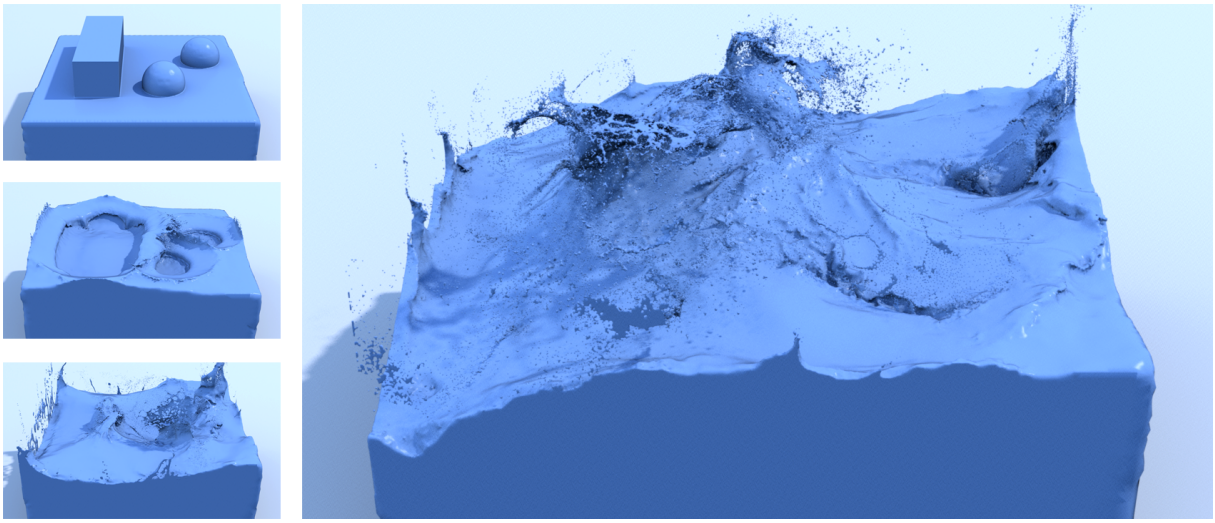

Figure 3: *A simple geometric setup creating a big splash inside a container. This simulation, with a maximum resolution of 256 cells, took* 1.3 *minutes per frame on average.*

directly (which would result in smeared out motion), we temporarily subdivide the cells of our mesh by inserting a vertex at the center where we have an accurate velocity sample.

We then perform barycentric interpolation based on these subdivided cells, ensuring a C^0 continuous velocity that retains the original velocities at cell centers. Note that these four smaller tetrahedra do not have to be stored explicitly. We construct them on the fly when a sample is requested from one of the original cells.

Manipulating FLIP particles The spacing between FLIP particles may drift over time, and high-frequency errors may contribute to a bumpy surface. We combat these problems by directly manipulating particle positions. During each time step, we apply the position correction algorithm of Ando et al. [2012]; this algorithm essentially pushes each particle away from its neighbors to prevent clustering. We also introduce two special behaviors when the particles are close to the liquid surface (less than a distance of six times the particle radius). First, we impose the constraint that the position correction step may only move particles near the surface tangentially to the fluid interface. Secondly, particles near the surface may leave gaps when they spread out quickly. Our method naturally fills in these gaps by slightly pulling each particle towards the fluid interface. For particles near the interface, this pulling force acts in addition to the position correction.

FLIP particles that partake in splashes and sprays can pose a significant burden on computational resources, especially in an adaptive framework like ours. This inefficiency stems from the fact that water droplets undergo extremely simple ballistic motion. Theoretically, we know that such a small region with purely free-surface boundary conditions will yield zero internal forces, so we simply detect individual FLIP particles that have no neighbors within six times their radius, remove them from the pressure solve, and accelerate them with gravity instead. When these particles eventually enter the neighborhood of other particles at some point in the future, we resume treating them like fluid by returning them to the pressure solve. This decision allows us to avoid aggressively refining the tetrahedral mesh in locations where the physical motion is uninteresting. Only a small percentage of the particles are simulated in this way, e.g., 1.7% on average for Figure 6.

4 Adaptivity

Our method achieves adaptivity by varying the mesh resolution over the computational domain. Our FLIP simulation performs computation on both a background volumetric mesh and on a set of particles. Given a sizing function that indicates the desired spatial level of detail, our method first creates a tetrahedral mesh with varying spatial resolution, and then it locally changes the particle density by splitting and merging operations.

To compute the spatially-varying background grid, we start with the Delaunay tetrahedralization of a set of points distributed in a body-centered cubic lattice configuration. In order to make the mesh resolution change over space, we use the octree-based grading method which was proposed by Labelle and Shewchuk [2007] and later adopted in several adaptive simulation environments [Chentanez et al. 2007; Wojtan and Turk 2008; Batty et al. 2010]. Similar to [Batty et al. 2010], we generate a new tetrahedral mesh every ten time steps, instead of rebuilding the mesh on every consecutive step. Also, the tetrahedral mesh is only temporarily used for the pressure solver, so no information is transferred from one time step to the next by storing it on the grid. Thus, we do not worry about re-sampling data when computing a new tetrahedral mesh.

We change the size and number of particles in our simulation with splitting and merging operations. For this, we modify the strategy of Ando et al. [2012] to work within our framework: at each remeshing step, we loop through the particles and determine whether the resolution needs to be changed. If a particle is too small, then we merge it with its nearest neighboring particle, resulting in a particle whose radius is given by the combined volume of the two original particles. If a given particle is bigger than the desired size, then the particle is split in two. The two new particles are placed randomly within the original particle's radius and redistributed with a heuristic that attempts to fill in nearby gaps: We first compute the 24 midpoints $\mathbf{m}_i$ between this particle and its 24 nearest neighbors. Then we find the closest particle to each midpoint and store the squared distance as a weight ω_i. The new particle's position is equal to the weighted average of all nearby midpoints: $\mathbf{x}_{\text{new}} = \sum \omega_i \mathbf{m}_i / \sum \omega_i$.

After a split or merge operation, the new particle's velocity is computed using a volume-weighted average. Also, we must take care to ensure that particles close to the surface do not introduce interfacial bumps when they split or merge; whenever we create a new

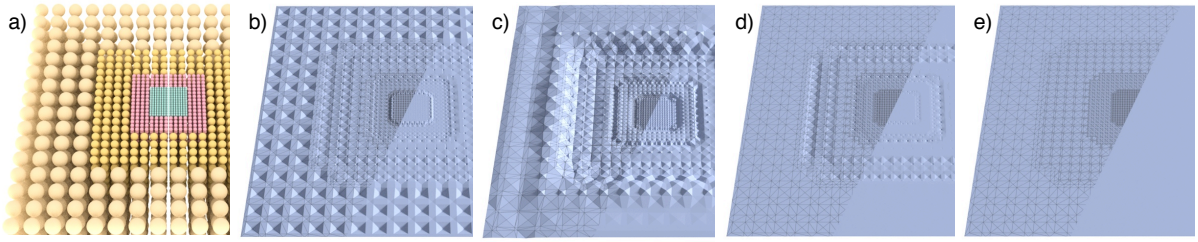

Figure 4: *For the perfectly planar particle placement shown in (a), the methods [Zhu and Bridson 2005] (b), [Yu and Turk 2010] (c), and [Adams et al. 2007] (d) do not result in a flat surface, while our method produces the desired result (e).*

particle that is less than 1.25 times its radius away from the surface (through either a split or a merge event), we move it in the surface normal direction such that its sphere lies exactly tangent to the liquid interface.

Numerical viscosity in fluid simulations is tightly coupled to the spatial resolution resolving the flow. We compensate for spatially-varying numerical viscosity caused by particles of various sizes in our simulation by adjusting the PIC/FLIP blending parameter in our FLIP simulation [Bridson 2008]. Given quantities $Q_{i,\text{PIC}}$ and $Q_{i,\text{FLIP}}$ computed at particle i from PIC and FLIP simulations, respectively, the new quantity Q_i is computed as a weighted blend between the two:

$$Q_i = \frac{\nu \Delta t}{r_i^2} Q_{i,\text{PIC}} + (1 - \frac{\nu \Delta t}{r_i^2}) Q_{i,\text{FLIP}} \tag{14}$$

where r_i is the radius of particle i and ν is the viscosity of the flow. We found that this strategy adequately eliminates any artifacts due to spatially varying numerical viscosity.

Sizing Functions We define the level of detail in our simulations with a spatially varying sizing function $S(\mathbf{x})$. We have experimented with several different sizing functions depending on factors such as distance to a camera, distance to the liquid surface, curvature of the liquid surface, measures of fluid turbulence, and arbitrary analytical number fields. Our simulator is versatile enough to cope with any of these sizing functions, resulting in efficient simulations with highly variable levels of detail. E.g., Figure 1 showcases a simulation where smallest and largest cells differ by a factor of 128.

In all of the examples in this paper, the sizing function is defined as a combination of five different metrics:

$$S(\mathbf{x}) = \max\left(d(\mathbf{x}), V(\mathbf{x}, \min(\kappa_{\text{liquid}}(\mathbf{x}), \kappa_{\text{solid}}(\mathbf{x}), e(\mathbf{x})) \right) \tag{15}$$

where $\mathbf{x}$ is the position of a point in space, and $d(\mathbf{x})$ encodes the depth of the liquid by returning the absolute level set value of the liquid surface. This has the effect that motion near the surface has higher priority than motions far inside the bulk volume of the liquid. $V(\mathbf{x}, y)$ is a view-dependent function that returns the value y if $\mathbf{x}$ is within the camera's visible region and returns the maximum particle radius r_{max} (representing the minimum surface resolution) otherwise. The next two metrics are designed to prioritize geometric detail of the liquid surface and of obstacles by computing a desired resolution based on curvature. $\kappa_{\text{liquid}}(\mathbf{x})$ returns 0.8 divided by the extrapolated curvature of the liquid interface. Similarly, $\kappa_{\text{solid}}(\mathbf{x})$ returns 1.6 $W_{\text{smooth}}(\mathbf{d}_{\text{solid}}, r_{\text{max}})$ divided by the extrapolated curvature of the solid interface, where $W_{\text{smooth}}(\mathbf{x}, h)$ is a smooth kernel function $(1 - ||\boldsymbol{x}||^2/h^2)^3$ and $\mathbf{d}_{\text{solid}}$ is the closest distance to the solid boundary. As a last component of our sizing function we found it beneficial to invest computational resources into keeping interesting motion of the flow field alive. Inspired by turbulence models (used e.g. in [Pfaff et al. 2010]), we have found that the strain tensor of the flow field reliably indicates detailed motions, and we compute $e(\mathbf{x})$ as 30 divided by the Frobenius norm of the fluid strain tensor computed from the velocity field.

5 Surface Representation

We also introduce a new method for computing an implicit surface from a set of particles. Given our set of FLIP particles with variable radii, we aim to implicitly represent the fluid surface by computing its signed distance function. Several useful methods for computing a surface from a collection of particles have been proposed in the past [Zhu and Bridson 2005; Adams et al. 2007; Yu and Turk 2010], but they tend to produce undesirably bumpy surfaces when considering particles of highly variable radii (Figure 4). In this section, we introduce a new strategy for computing an implicit surface from a set of particles of various sizes.

The main idea is to approximate the fluid surface with the union of the convex hulls of each triplet of nearby particles close to the surface. For each set of three FLIP particles near the surface, the convex hull forms a thickened triangle shape with rounded edges (Figure 5). We only consider particles that are less than a given distance apart, with the maximum distance equal to a constant scale factor l times the sum of the two particle radii. A small l shows more surface details, while a larger l tends to fill in small concavities. We used $l = 2$ for most of the simulations in this paper. We ultimately represent our surface as the union of all such local convex hull shapes, and the minimum signed distance from these shapes to a point in space defines the outer part of our level set function.

In practice, we compute the local convex hull by finding the two outermost planes tangent to a set of three spheres. We efficiently compute the distance to these planes by analytically solving the polynomial system: $ax_1+by_1+cz_1+d = r_1$, $ax_2+by_2+cz_2+d = r_2$, $ax_3 + by_3 + cz_3 + d = r_3$, $a^2 + b^2 + c^2 = 1$. where r_i and x_i, y_i, z_i are the radius and x, y, z coordinates of particle i, respectively. a, b, c, d are the variables defining our plane with the signed distance function $ax + by + cz + d = 0$. Intuitively, the first three equations ensure that the plane is the right distance away from each particle with the normal facing away from them, and the final equation ensures that the plane equation is normalized to a distance function. These four equations represent the intersection of three hyperplanes and a hypercylinder in 4D $\{a, b, c, d\}$ space. We solve this system analytically by first finding the line of common intersection of the first three equations, and then intersecting this line with the cylinder represented by the final equation. The system has two solutions, representing the top and bottom planes of our convex hull shape.

The above calculation describes how to find the planar regions of the convex hull of a set of three spheres. By computing the conic and spherical convex hull facets (Figure 5, bottom right) in a similar fashion, we can easily compute the signed distance between this convex hull and a point in space. To evaluate our final level set

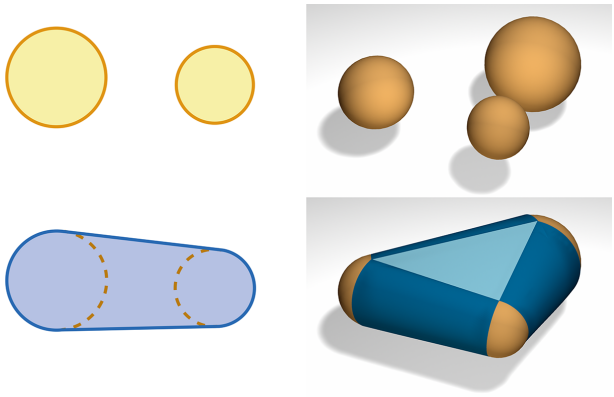

Figure 5: *The 2D version of our surface creation algorithm takes a pair of particles (top left) and computes their convex hull (bottom left). In 3D, we convert three nearby particles (top right) into a convex hull (bottom right) consisting of spherical (orange), conic (dark blue), and planar (light blue) segments.*

value, we compute the minimum signed distance from a query point to all nearby convex hulls. We evaluate the level set on each of the vertices of our adaptive BCC mesh, and we extract a triangle mesh using a marching tetrahedra algorithm. We then perform a light mesh smoothing to increase the reliability of any curvature computations.

The algorithm as described works perfectly for computing the level set outside of our particle surface, but it may lead to small gaps inside. To avoid the creation of holes, we temporarily reassign each particle's radius: $r_i^{\text{temp}} = \max(r_i, -k\phi_i)$ where ϕ_i is the particle's stored level set value from the previous time step, and k is a constant set to 0.75 in our simulations. Using this temporary radius to compute the signed distance as described above will remove erroneous gaps inside the liquid.

We need to compute liquid surfaces both for final visualization as well as for several calculations during the progress of our simulation. For the final visualization, we compute an especially high-resolution BCC mesh from all of our particles and proceed with the algorithm above. The final surface creation is trivially parallelized, and takes around five minutes average per frame for all of our simulations. We attempt to speed up the surface creation routine used for simulation computations by computing on the moderate-resolution BCC mesh used for simulation and ignoring ballistic particles (Section 3). We compare our surface creation routine with a few existing methods in Figure 4. Most previous algorithms perform poorly in this comparison because they were not designed for particles with varying radii.

6 Implementation

At this point we have described all of the components of our simulator. The resulting algorithm can be seen in Algorithm 1. In the beginning of each step (line 2), we typically compute the level-set for the current particle configuration as described in Section 5. We require the distance to the surface in several steps of our algorithm, so we store the level set values for each particle (line 3). When enough time has passed to trigger an update of the mesh, it becomes necessary to evaluate the sizing function. At this point, additional user-defined sizing functions could be computed as well. Having the information from the sizing functions ready, we create a new BCC mesh and perform particle merging and splitting.

For mapping the particle velocities onto the grid (line 9) we use an SPH-like kernel function, which is weighted by the particle volume:

Algorithm 1: One step of our simulation algorithm.

```
1  begin
2      Compute simulation surface S
3      Pre-compute φ for all particles
4      Correct particle positions x_i
5      if Mesh update necessary then
6          Evaluate sizing function S(x) at x_i
7          Build octree and BCC mesh
8          Merge and split particles
9      Compute mesh velocity û from particle velocities
10     Extrapolate û outside the liquid
11     Solve pressure p̂ on the tetrahedral mesh, update û
12     Update particle velocities with gradient of p̂
13     Advect particles with û
```

$\max(v_i(4r_i^2/d^2 - 1), 0)$, where v_i is the particle volume and d is the distance to the particle center normalized with its radius. The particle velocity update of line 12 uses barycentric interpolations as explained in Section 3. Likewise, the grid-based velocity extrapolation of line 10 uses the nodal velocities of Section 3. A time step is completed by performing the pressure projection and advecting the particles in the resulting divergence-free velocity field.

7 Results and Discussion

To evaluate the performance and robustness of our method in comparison to previous work we have performed an extensive series of tests. A selection of these can be found in the accompanying comparison video. One comparison that is particularly interesting is the one comparing our method to an FVM based simulation. Using a graded BCC mesh leads to problems with the latter, as the position of the circumcenter lies exactly on a face for the graded tetrahedrons. In the graded region, this can result in two pressure samples from adjacent tetrahedra being placed at the exact same position. To alleviate this problem, [Batty et al. 2010] propose to slightly offset the pressure samples from the faces. However, our implementation of their method exhibited slow convergence and the velocity artifacts despite this fix. The influence of the different components of our sizing function on the evolution of a simulation is difficult to depict with static images, so we refer to the accompanying video for a comparison.

To evaluate the basis of our adaptive model without any influence of the camera dependent sizing function, we have simulated the simple geometric configuration shown in Figure 3. For this setup, resolutions from 8 to 256 were used, resulting in 6 levels of adaptivity. Timing information for the main steps of our algorithm over the course of this simulation can be found in Figure 8 and Figure 9. For this simulation, the initial configuration consisted of $168,161$ particles, and momentarily peaked up to $1,048,776$ during the maximal extent of the splash (settling down again to around 250 thousand in the end). Note that a full sampling of the initial configuration with a regular grid would have required approximately 6 million particles. Our measurements show that the run-time of our method has a strong linear relationship to the number of particles, and thus the visual complexity of the simulation. The per-frame time is low at the beginning and end of the simulation, but strongly peaks during the complex splash in its middle.

Our visibility sizing function is highlighted by the setup shown in Figure 6. Here the computational resources are focused on the visible region of a rotating camera, as the liquid splashes around a U-shaped corridor. Our solver efficiently resolves the complex motion near the camera, while effectively reducing the computational

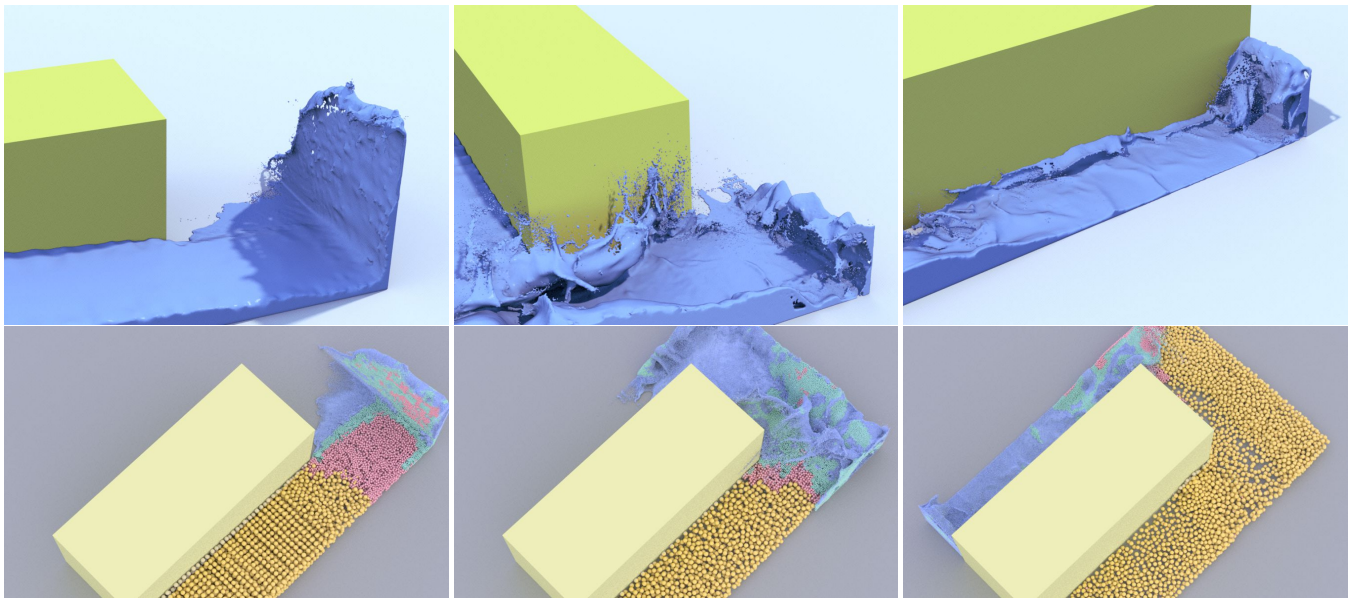

Figure 6: *Our visibility sizing function: as the liquid flows along the U-shaped corridor, the visible volume is simulated with a high resolution (surface shown in top row), while regions outside of the view frustum are coarsened (particle view in bottom row).*

Setup	Min. Δx^{-1}	Min. Δx_s^{-1}	Max. Δx^{-1}	Duration	frames
Figure 3	8	16	256	5h:09m	240
Figure 6	16	32	256	4h:26m	330
Figure 7	16	64	512	5h:55m	480
Figure 1	8	32	1024	12h:8m	160

Table 1: *Resolutions and running times for our simulations (not including final surface creation). Here, Δx^{-1} and Δx_s^{-1} denote the number of BCC cells along one spatial axis for the simulation and for the surface generation, respectively. The simulations were run on a workstation with an Intel Core i7-3960X CPU with 3.30GHz running under Linux.*

cost for parts that are not visible. The simulation of Figure 7 shows a liquid interacting with a highly detailed obstacle. The κ_{solid} component of our sizing function ensures that geometrically complex regions near the obstacle are simulated with higher accuracy. In this way, we can resolve the detailed flow of liquid through the holes in the obstacle.

Figure 1 shows a situation that would be challenging to simulate with a regular solver. Without adaptivity, the large open liquid surface with complex splashes in a localized region would require huge amounts of computational resources. Our method can simulate this setup very efficiently, and in a fully coupled manner with an effective high resolution. The large open region is successfully coarsened by our sizing function, resulting in subtle wave motions around the splashes. In this case, the whole simulation with 8 different octree levels and a maximum resolution of up to 1024 cells took on average only 4.6 minutes per frame to compute. Just to illustrate the amount of detail in this setup – our adaptive version initially used 1.7 million particles, while a regular sampling at the finest resolution would have required roughly 400 million. Further runtime and resolution details for our simulations can be found in Table 1.

Discussion We found our discretization (Section 3) beneficial in a number of ways. We store the pressure variables on tetrahedral vertices, and there are far fewer vertices than tetrahedra in a given mesh. Consequently, the pressure solve has fewer variables and is faster to solve. Also, by counting degrees of freedom and constraints, we can see that our discretization prevents the locking artifacts which are common in other methods. However, the lower number of pressure constraints also implies that the highest frequencies of the velocity field may be unconstrained. In our case a regularization via PIC interpolation acts to diminish any high-frequency artifacts.

Our method uses a FLIP scheme instead of a purely Eulerian

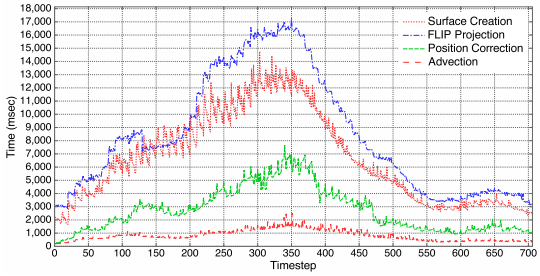

Figure 8: *This graph shows durations for the different parts of our algorithm over the course of the simulation from Figure 3. Note that we only include the computationally more expensive steps here, and not the re-meshing (which is done in intervals).*

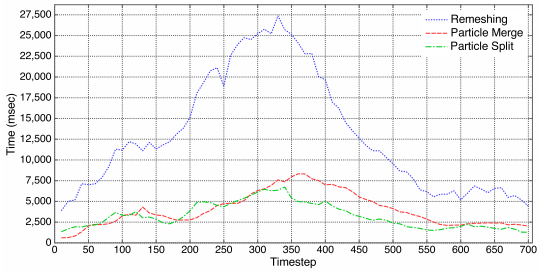

Figure 9: *The time required for our re-meshing and particle merging & splitting computations over time for Figure 3. Note that we perform them only every ten simulation steps.*

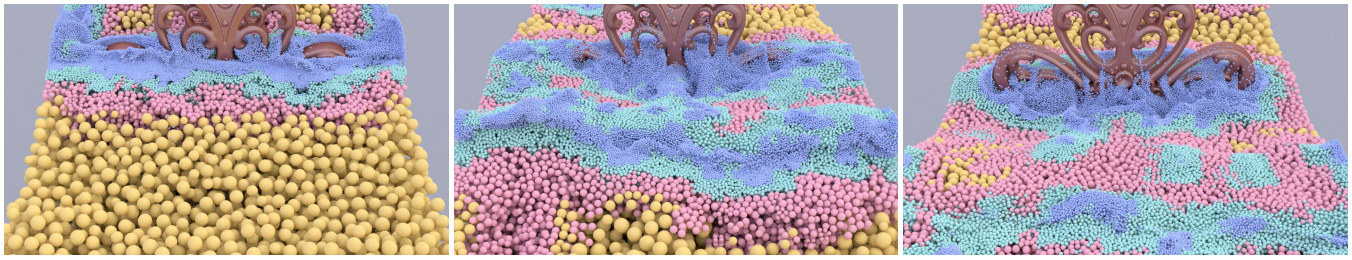

Figure 7: *A simulation of detailed flow through a complex obstacle. The liquid correctly flows through the orifices at the center of the filigree.*

method. We store all physical variables on the FLIP particles, so information is carried from one time step to the next in a Lagrangian manner. As a result, we are allowed to aggressively remesh the tetrahedral background grid without worrying about excessive damping or re-sampling artifacts. On the other hand, FLIP simulations have a well-known problem of creating noisy particle distributions, because there are typically several times more particles than velocity variables on the background grid. We utilize particle repositioning to improve the distribution quality, at the expense of slight inaccuracies due to displacing physical variables.

We noticed that our new surface creation routine is essential for maintaining detailed simulations in the presence of accurate free-surface boundary conditions. One major benefit of our method is that it can easily create perfectly flat surfaces from a mixture of differently-sized particles. These flat surfaces represent the equilibrium state of a fluid simulation, so our animations are able to smoothly settle down as time progresses. Without a method for accurately reproducing flat surfaces, second-order boundary conditions will introduce additional forces in the locations of surface bumps, which artificially prevent a simulation from settling down. While we believe that our surface creation routine is indispensable, it is quite expensive to compute. In the future we would like to optimize the surface computation.

Our simulations perform quite well for large differences in resolutions, but we have only been able to push them to a certain point in our current implementation. We found that using too sharp of a grading in our sizing function can place coarse and fine simulation elements too close together and potentially result in artifacts. For example, when small particles land in very coarse cells after violent splashes, these particles can get stuck in mid air. Occasionally, this can also lead to an overly strong weight for such particles during the velocity mapping, resulting in momentum artifacts. Our adaptive numerical viscosity in Eq. (14) can also exhibit dangerously small damping values for very fine resolutions, so we clamped the blending coefficient to a minimum value of 0.1 in Figure 1.

In Section 4 we introduced a novel collection of sizing functions for adaptively selecting details from a fluid simulation. While we presented specific parameters for the sake of reproducibility, these values were not meticulously tuned and are certainly not optimal. The task of choosing an ideal sizing function is still an open problem that we are interested in pursuing in the future. In particular, we are interested in taking more temporal information into account. This could lead to more gradual changes in resolution, at the expense of a slightly higher particle count.

8 Conclusions and Outlook

We have presented a novel framework for highly adaptive liquid simulation. In our method, a novel, robust discretization works together with accurate embedded boundary conditions and a flexible sizing function to allow for aggressive adaptivity and high computational performance. In this way, we can efficiently compute tough simulation setups, such as large surfaces with very localized details. We have additionally presented a novel surface creation method that yields smooth surfaces in the presence of strongly varying particle radii, which turned out to be an important building block for our framework.

We chose a BCC mesh generation because it is, to the best of our knowledge, the fastest way to generate high-quality meshes. However, despite its efficiency, mesh generation is still a bottleneck for our simulation. This is partly due to the fact that it is a mostly serial operation that is difficult to parallelize (most other steps of our algorithm parallelize easily). So, instead of computing the mesh from scratch each time, we are interested in exploring techniques for continuous re-meshing. Also, our choice of piece-wise constant basis functions for velocity indicates that our discretization could lead to difficulties when it is used for diffusion or viscosity solves. It will be interesting to see how these could be incorporated into our framework. Finally, we are highly interested in applying our method to other types of phenomena, such as smoke and fire simulations, or visco-elastic materials. It will be very interesting to leverage the benefits of our framework for extreme adaptivity in these situations.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful feedback. We also wish to thank Reiji Tsuruno for providing us with computational resources, and Pascal Clausen as well as Ramprasad Sampath for constructive discussions. This work was supported by the Japan Society for the Promotion of Science (JSPS). Finally, we would like to express our gratitude to the authors of the *ANN* library, which we use for kd-tree look-ups, and to the *Mitsuba* renderer, which we have used to render all images in this paper.

References

ADAMS, B., PAULY, M., KEISER, R., AND GUIBAS, L. J. 2007. Adaptively sampled particle fluids. In *ACM SIGGRAPH 2007 papers*, 48.

ANDO, R., THUEREY, N., AND TSURUNO, R. 2012. Preserving Fluid Sheets with Adaptively Sampled Anisotropic Particles. *IEEE Transactions on Visualization and Computer Graphics 18 (8)*, 1202–1214.

BATTY, C., AND BRIDSON, R. 2008. Accurate viscous free surfaces for buckling, coiling, and rotating liquids. In *Proceedings of the 2008 ACM/Eurographics Symposium on Computer Animation*, 219–228.

BATTY, C., BERTAILS, F., AND BRIDSON, R. 2007. A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph. 26*, 3 (July).

BATTY, C., XENOS, S., AND HOUSTON, B. 2010. Tetrahedral embedded boundary methods for accurate and flexible adaptive fluids. In *Proceedings of Eurographics*.

BHATTACHARYA, H., GAO, Y., AND BARGTEIL, A. W. 2011. A level-set method for skinning animated particle data. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

BOYD, L., AND BRIDSON, R. 2012. Multiflip for energetic two-phase fluid simulation. *ACM Trans. Graph. 31*, 2, 16:1–16:12.

BRIDSON, R. 2008. *Fluid Simulation for Computer Graphics*. AK Peters/CRC Press.

BROCHU, T., BATTY, C., AND BRIDSON, R. 2010. Matching fluid simulation elements to surface geometry and topology. *ACM Trans. Graph. 29*, 4 (July), 47:1–47:9.

CHENTANEZ, N., FELDMAN, B. E., LABELLE, F., O'BRIEN, J. F., AND SHEWCHUK, J. R. 2007. Liquid simulation on lattice-based tetrahedral meshes. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, 219–228.

CLAUSEN, P., WICKE, M., SHEWCHUK, J., AND O'BRIEN, J. 2013. Simulating liquids and solid-liquid interactions with Lagrangian meshes. *ACM Trans. Graph.*.

ENRIGHT, D., NGUYEN, D., GIBOU, F., AND FEDKIW, R. 2003. Using the Particle Level Set Method and a Second Order Accurate Pressure Boundary Condition for Free-Surface Flows. *Proc. of the 4th ASME-JSME Joint Fluids Engineering Conference*.

ENRIGHT, D., LOSASSO, F., AND FEDKIW, R. 2005. A fast and accurate semi-Lagrangian particle level set method. *Comput. Struct. 83*, 6-7, 479–490.

HARLOW, F., AND WELCH, E. 1965. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Phys. Fluids 8 (12)*, 2182–2189.

HONG, W., HOUSE, D. H., AND KEYSER, J. 2009. An adaptive sampling approach to incompressible particle-based fluid. In *TPCG*, Eurographics Association, W. Tang and J. P. Collomosse, Eds., 69–76.

IRVING, G., SCHROEDER, C., AND FEDKIW, R. 2007. Volume conserving finite element simulations of deformable models. *ACM Trans. Graph. 26*, 3 (July).

KLINGNER, B. M., FELDMAN, B. E., CHENTANEZ, N., AND O'BRIEN, J. F. 2006. Fluid animation with dynamic meshes. *ACM Trans. Graph. 25*, 3, 820–825.

LABELLE, F., AND SHEWCHUK, J. R. 2007. Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. *ACM Trans. Graph. 26*, 3 (July).

LEW, A. J., AND BUSCAGLIA, G. C. 2008. A discontinuous-galerkin-based immersed boundary method. *International Journal for Numerical Methods in Engineering 76*, 4, 427–454.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. *ACM Trans. Graph. 23*, 3 (Aug.), 457–462.

LOSASSO, F., FEDKIW, R., AND OSHER, S. 2005. Spatially adaptive techniques for level set methods and incompressible flow. *Computers and Fluids 35*, 2006.

MISZTAL, M. K., BRIDSON, R., ERLEBEN, K., BÆRENTZEN, J. A., AND ANTON, F. 2010. Optimization-based fluid simulation on unstructured meshes. In *VRIPHYS*, Eurographics Association, 11–20.

MULLEN, P., CRANE, K., PAVLOV, D., TONG, Y., AND DESBRUN, M. 2009. Energy-preserving integrators for fluid animation. *ACM Trans. Graph. 28* (July), 38:1–38:8.

PFAFF, T., THUEREY, N., COHEN, J., TARIQ, S., AND GROSS, M. 2010. Scalable fluid simulation using anisotropic turbulence particles. In *ACM Transactions on Graphics (TOG)*, vol. 29, ACM, 174.

SIN, F., BARGTEIL, A. W., AND HODGINS, J. K. 2009. A point-based method for animating incompressible flow. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

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

STAM, J. 1999. Stable fluids. In *Proceedings of SIGGRAPH '99*, ACM, 121–128.

WOJTAN, C., AND TURK, G. 2008. Fast viscoelastic behavior with thin features. *ACM Trans. Graph. 27*, 3, 1–8.

YU, J., AND TURK, G. 2010. Reconstructing surfaces of particle-based fluids using anisotropic kernels. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, 217–225.

ZHU, Y., AND BRIDSON, R. 2005. Animating sand as a fluid. *ACM Trans. Graph. 24*, 3 (July), 965–972.

A Ghost Fluid Coefficients

Here we describe how to compute the ghost fluid coefficients θ_n given the 4×4 pressure matrix entries of a single tetrahedron. Reorganizing Eq. (11) gives:

$$\begin{bmatrix} \lambda_2 + \alpha w_1 & a + \alpha w_2 & b + \alpha w_3 \\ a + \beta w_1 & \lambda_3 + \beta w_2 & c + \beta w_3 \\ b + \gamma w_1 & c + \gamma w_2 & \lambda_4 + \gamma w_3 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}. \quad (16)$$

Note that each of the θ_n has two degree of freedom, and thus each w_n also has two degrees of freedom. As we know that the resulting matrix needs to be symmetric, which gives us the following constraints: $a + \alpha w_2 = a + \beta w_1$, $b + \gamma w_1 = b + \alpha w_3$, and $c + \beta w_3 = c + \gamma w_2$. As the w_n linearly depend on θ_n, that means: $\alpha\theta_2 = \beta\theta_1$, $\gamma\theta_1 = \alpha\theta_3$, and $\beta\theta_3 = \gamma\theta_2$. When we re-write these constraints in matrix form, and include the barycentric coordinate constraint $\theta_1 + \theta_2 + \theta_3 = 1$ we get the following linear system:

$$\begin{bmatrix} -\beta & \alpha & 0 \\ \gamma & 0 & -\alpha \\ 0 & -\gamma & \beta \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad (17)$$

As the rank of top three rows of the matrix is 2, we can drop one of them. Removing the first row from the system gives us the following full-rank, 3×3 matrix:

$$\begin{bmatrix} \gamma & 0 & -\alpha \\ 0 & -\gamma & \beta \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (18)$$

The analytical solution of this system is:

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} = \frac{1}{\alpha + \beta + \gamma} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix}. \quad (19)$$

This means that for our discretization, the ghost pressure coefficients θ_n are given by the tetrahedron's matrix entries from Eq.10. More specifically, by those for the vertex that is located outside of the liquid, i.e., α, β and γ.

Substituting this equation into Eq.9 and Eq.6 yields the final equation for the ghost pressure Eq. (12).