Volume 0 (1981), Number 0 pp. 1–13 COMPUTER GRAPHICS *forum*

DYVERSO: A Versatile Multiphase Position-Based Fluids Solution for VFX

Iván Alduán[1,2], Angel Tena[2] and Miguel A. Otaduy[1]

[1]URJC Madrid
[2]Next Limit Technologies

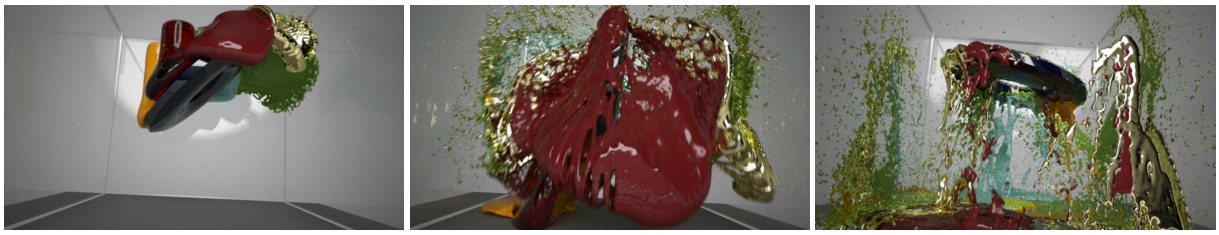

Figure 1: *Three snapshots of the same animation. Seven liquids with different properties interact robustly within our multiphase position-based fluids framework.*

Abstract

Many impressive fluid simulation methods have been presented in research papers before. These papers typically focus on demonstrating particular innovative features, but they do not meet in a comprehensive manner the production demands of actual VFX pipelines. VFX artists seek methods that are flexible, efficient, robust, and scalable, and these goals often conflict with each other. In this paper we present a multiphase particle-based fluid simulation framework, based on the well known Position-Based Fluids (PBF) method, designed to address VFX production demands. Our simulation framework handles multiphase interactions robustly thanks to a modified constraint formulation for density contrast PBF. And it also supports the interaction of fluids sampled at different resolutions. We put special care on data structure design and implementation details. Our framework highlights cache-efficient GPU-friendly data structures, an improved spatial voxelization technique based on Z-index sorting, tuned-up simulation algorithms, and two-way-coupled collision handling based on VDB fields. Altogether, our fluid simulation framework empowers artists with the efficiency, scalability and versatility needed for simulating very diverse scenes and effects.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling

1. Introduction

Research on fluid simulation has a long history in computer graphics, yet animation artists keep demanding methods that enable larger and richer simulations. Even though all new methods contribute to the improvement and evolution of the field, their adoption within the artist community is subject to other practical factors imposed by production requirements.

Computational efficiency is one of those factors. Better simulation times give artists the ability to iterate quickly over a shot and reach faster the desired look. But efficiency often competes with other practical factors such as constraints on the simulation domain, memory consumption, resolution settings, or controllability.

Constraints on the simulation domain, such as the use of fixed resolution domains, are factors that compromise versa-

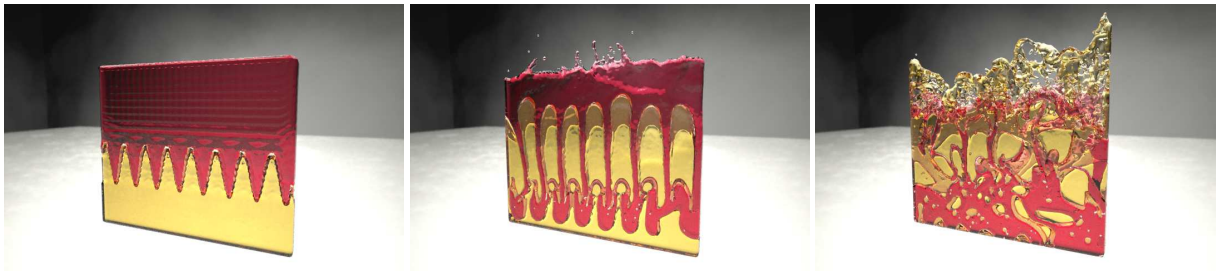

Figure 2: *Initial state and two animation frames of a Rayleigh-Taylor instability. With the proposed density contrast formulation for PBF, our solution is able to keep sharp interfaces and good particle distributions even in the presence of high density ratios between liquid phases.*

tility. Artists creatively circumvent such constraints, but simulation methods that support arbitrary scenes, such as open domains with complicated boundaries, are preferred. High memory consumption and resolution constraints are factors that compromise scalability. Production environments need scalable methods that support high-resolution simulations with highly detailed effects under practical memory costs. Finally, controllability limitations are the factors that compromise the artists' ability to express their creativity. Artists seek user-oriented methods that allow them to explore in an intuitive manner a diverse set of fluid behaviors and their interactions. In practice, artists will even take simulation methods beyond the limits that they were designed for, and expect them to work under diverse and stressful conditions. One typical example is to populate the simulation domain with solid objects represented using non-manifold meshes, degenerate geometry, and/or self-intersecting geometry. Artists prefer robust and stable methods that reach reasonable results even in worst-case scenarios.

This paper presents DYVERSO, a particle-based fluid simulator designed to address VFX production demands. The simulator builds on the PBF method by Macklin and Müller [MM13], but we extend the basic formulation to support multiphase effects of different nature, as shown in Fig. 1, and fluids sampled at different resolutions. In the paper, we combine novel contributions to fluid models that increase the robustness and versatility of the PBF method, with engineering solutions and data structures that enable the adoption of existing research solutions in production environments for the simulation of very diverse scenes and effects.

Section 2 reviews previous work on particle-based fluid simulation following the SPH and PBF methods. Then, Section 3 and Section 4 describe our novel research contributions. Section 3 describes our multiphase PBF formulation, capable of handling robustly multiphase density interfaces and fluids sampled at multiple different resolutions. Section 4 presents other extensions to the basic PBF formulation, namely improved viscosity and surface tension, which further increase the versatility of the framework. Section 5 to Section 7 describe engineering solutions for production-level integration of research methods. Section 5 describes the treatment of two-way-coupled collisions with objects of complex geometry, using VDB sparse fields. Section 6 describes our approach to accelerate neighbor searches with a modified Z-index sort algorithm. And Section 7 discusses our data structures for efficient particle management on a highly parallel heterogeneous implementation. Section 8 and the accompanying videos show our results. Finally, Section 9 discusses avenues for future work.

2. Review of SPH and PBF Methods

Particle systems have been used for animation since the early days of computer graphics [Ree83]. Due to their Lagrangian nature combined with the absence of connectivity requirements, smoothed particles were first used to simulate elements like fire and smoke [SF95], highly deformable bodies [DG96], and viscous fluids like lava [SAC*99]. Starting with the work of Müller et al. [MCG03], Smoothed Particle Hydrodynamics (SPH) has become one of the most popular techniques for fluid animation. Thanks to the versatility of SPH, these formulations have been adapted to support multiphase simulations [MSKG05, SP08], viscoelastic fluids [CBP05], phase transitions [SSP07], deformable solids [BIT09] or granular media [LD09].

One of the major drawbacks of early SPH methods is the challenge to model highly incompressible fluids. Becker et al. proposed some modifications to achieve better incompressibility [BT07] at the cost of smaller time-steps. The stiffer the system becomes, the less feasible it is to use purely explicit formulations. Solenthaler and Pajarola [SP09] introduced the PCISPH method, which iterates pressure adjustments to project the density to acceptable values. PCISPH has also been extended to support other materials [AO11, DGP12]. Implicit SPH formulations have recently gained popularity [ICS*14, PICT15]. As an alternative to SPH, Premoze et al. proposed the use of the Moving-Particle Semi-Implicit method [PTB*03].

Although PCISPH comes from a physically motivated

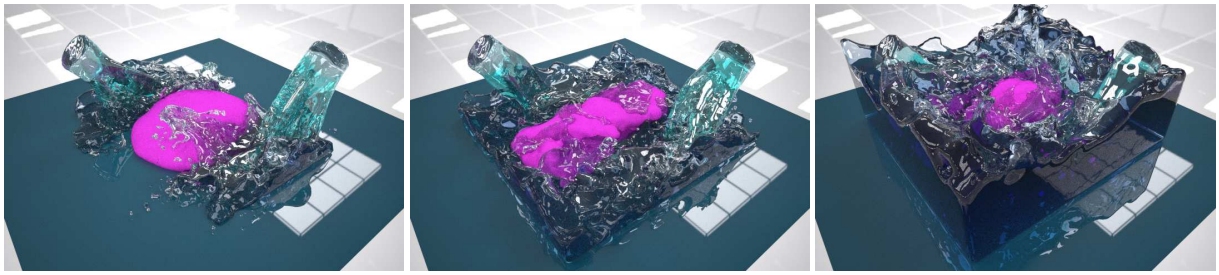

Figure 3: *Two stream emitters of water generate a vortex-like turbulent flow which interacts with a low-density, highly viscous fluid material, creating some interesting deformations on the interface, as well as rotatory behavior resulting from momentum transfer.*

derivation, it largely resembles an iterative solver for constrained optimization too. The connection between particle fluids and constrained optimization was introduced to graphics by Bodin et al. [BLS12]. Macklin and Müller [MM13] recently realized that these constraints could be reformulated to be solved in a position-based manner, gaining additional stability. PBF has been extended to support different media and its interactions [MMCK14].

This work is an extended version of a previously published paper [ATO15]. Here, we add important features that increase the versatility of the fluid solver, in particular treatment of multiphase fluids, improved handling of fluids sampled at different resolutions, and two-way-coupled interaction with solids. All these features are demonstrated on novel examples.

In each section of the paper we will discuss additional work related to each one of our contributions. For more information on relevant publications in the field, we refer to the state of the art report by Ihmsen et al. [IOS*14].

3. Multiphase Position-Based Fluids

As we have anticipated, our solver uses as baseline the PBF method by Macklin and Müller [MM13]. We propose modifications to the basic formulation of incompressibility, such that the PBF method can accommodate in a robust manner fluids with different densities and sampled at different resolutions. Our method adapts the density contrast SPH formulation [SP08] for multiphase simulations to the PBF setting, and it also incorporates multiresolution particle sampling. In practice it ensures good particle distribution and stable behavior even under high density ratios as seen in Fig. 2 and Fig. 3.

To enforce incompressibility, PBF defines one constraint C_i per particle, which measures the deviation of the current SPH-based density ρ_i from the fluid's rest density $\rho_{0,i}$ [BLS12]:

$$C_i = \frac{\rho_i}{\rho_{0,i}} - 1 = 0. \qquad (1)$$

An attractive feature of our method is that it handles both multiple-density and multiple-resolution fluids simply by formulating appropriately the density function in Eq. (1). Then, the rest of the method adopts the standard PBF formulation.

In our multiphase formulation, the rest density depends on each particle's properties. However, unlike the standard SPH density computation, and similar to Solenthaler and Pajarola [SP08], we propose to measure the current density by assuming that all neighboring particles have the same rest

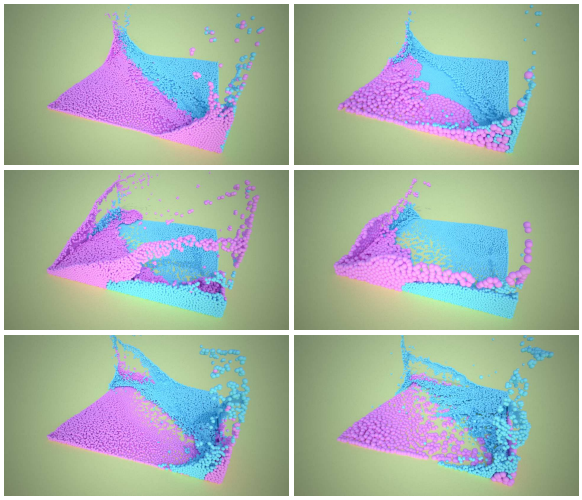

Figure 4: *Comparison of multiphase dam-break simulations. Denser particles sweep less dense particles and appear less splashy, regardless of the sampling resolution. From top to bottom, the ratios of rest densities* $r = \rho_{pink}/\rho_{blue}$ *vary as:* $r = 1.0$ *(top);* $r = 0.2$ *(middle);* $r = 3.0$ *(bottom). The snapshots also compare initializations at different resolutions. On the left column, the number and volume of pink and blue particles is the same. On the right column, the number of blue particles is ten times the number of pink particles.*

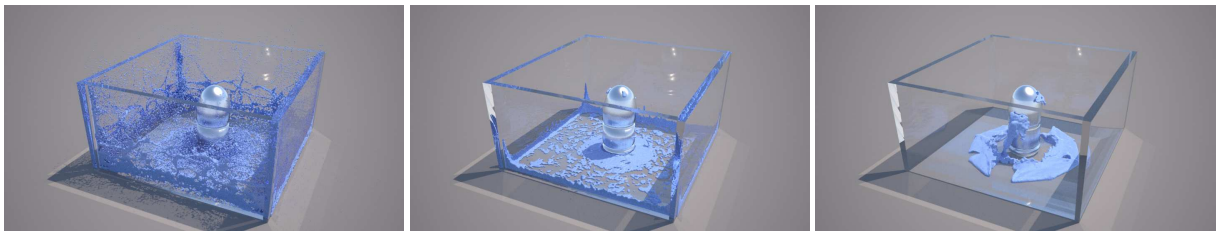

Figure 5: *With our robust XSPH viscosity algorithm, we are able to produce a large range of viscous behaviors. In the snapshots, a sphere of fluid falls over a bullet-shaped object, with behaviors that range from inviscid on the left to toothpaste-like viscous on the right.*

density and mass as the one under study. With uniform particle resolution, particle densities can be computed as:

$$\rho_i = m_i \sum_j W(\mathbf{x}_i - \mathbf{x}_j, h), \tag{2}$$

where m_i is the particle's mass, $\mathbf{x}_i$ and $\mathbf{x}_j$ are particle positions, h is the kernel size, and W is the SPH kernel. As mentioned earlier, this simple change to the original PBF formulation allows handling multiphase fluids robustly.

In our artist-friendly framework, we enable the initialization of different fluids or fluid regions using different sampling resolutions, as shown in Fig. 4. This feature, not to be confused with dynamic adaptivity of particle resolution, is one easy way to set up each fluid emitter according to visual criteria (e.g., distance to camera, rendering properties, etc.). To enforce incompressibility constraints on particles with different sampling resolutions, we average kernel evaluations for different kernel sizes, and we compensate for particle volume differences. Adams et al. [APKG07] also averaged kernel evaluations for the computation of pressure and viscosity forces to ensure force symmetry, but their setting did not require kernel averaging for density evaluations. In our setting, on the other hand, kernel averaging is also required for the computation of particle density, to avoid asymmetric interactions. With average kernels and particle volume compensation, particle densities can be computed as:

$$\rho_i = m_i \sum_j \frac{V_j}{V_i} W(\mathbf{x}_i - \mathbf{x}_j, h_i, h_j), \tag{3}$$

where $W(\mathbf{x}_i - \mathbf{x}_j, h_i, h_j) = \frac{1}{2}(W(\mathbf{x}_i - \mathbf{x}_j, h_i) + W(\mathbf{x}_i - \mathbf{x}_j, h_j))$ is the average kernel evaluation, and h_i and h_j are the kernel sizes of the i^{th} and j^{th} particles respectively.

Constraint gradients take the following form:

$$\nabla_{\mathbf{x}_k} C_i = \frac{m_i}{\rho_{0,i}} \begin{cases} \sum_j \frac{V_j}{V_i} \nabla W(\mathbf{x}_i - \mathbf{x}_j, h_i, h_j) & \text{if } k = i \\ -\frac{V_k}{V_i} \nabla W(\mathbf{x}_i - \mathbf{x}_k, h_i, h_k) & \text{if } k \neq i \end{cases} \tag{4}$$

Following Müller et al. [MHHR07], we aim to find a set of Lagrange multipliers $\{\lambda_i\}$ which, once applied as position corrections $\Delta\mathbf{x}_i$ along the constraint gradients, satisfy the constraints in Eq. (1). PBF solves for Lagrange multipliers following a Jacobi-type iteration, i.e., each Lagrange multiplier is solved independently to satisfy its corresponding constraint, and then all position corrections are added together. We weight the position corrections $\Delta\mathbf{x}_i$ by the inverse masses $\frac{1}{m_i}$ to account for mass differences.

In addition, to compute the Lagrange multiplier λ_i of each particle's incompressibility constraint, we regularize the constraint stiffness in a way similar to Solenthaler and Pajarola [SP09]. This regularization yields simulations that are computationally less expensive, while the overall fluid behavior is very similar to more accurate PBF models. The regularization is particularly inaccurate for surface particles, i.e., particles with few neighbors, but this is not a major problem in practice, as the behavior of such particles is dominated by surface tension. For each fluid type, we setup a prototype filled neighborhood according to the fluid's density ρ_0, and we precompute a constraint stiffness k_{ρ_0}:

$$k_{\rho_0} = \frac{1}{\frac{1}{m_{\rho_0}} \sum_j \left|\nabla_{\mathbf{x}_j} C_i\right|^2}. \tag{5}$$

If a particle interacts with particles of different densities, we do as Macklin et al. [MMCK14] and use the lowest, most conservative stiffness.

Then, the Lagrange multiplier for the incompressibility constraint of the i^{th} particle can be computed as:

$$\lambda_i = -\beta\, k_{\rho_{0,i}} C_i. \tag{6}$$

We use a relaxation coefficient $\beta \leq 1$ to ensure convergence of the Jacobi-type iterations. The value of β needed by the simulation depends on the number of neighbors. However, it can be automatically estimated for a certain number of neighbors, and in practice it is not exposed to the user.

Finally, position corrections for a Jacobi iteration can be computed from the Lagrange multipliers in the multiphase and multiresolution setting as:

$$\Delta\mathbf{x}_i = \frac{1}{m_i} \sum_j \left(\frac{m_i V_j}{\rho_{0,i} V_i} \lambda_i + \frac{m_j V_i}{\rho_{0,j} V_j} \lambda_j \right) \nabla W(\mathbf{x}_i - \mathbf{x}_j, h_i, h_j). \tag{7}$$

4. Extensions to PBF

In this section, we present additional extensions to the basic PBF method, namely viscosity and surface tension models, which extend the types of behaviors that can be simulated in a controllable way.

4.1. Viscosity

Macklin and Müller [MM13] used XSPH [SB12] to maintain coherent motion in their simulations. Beyond this goal, we wish to accommodate robust handling of controllable viscosity, and thus support a wider range of fluid effects. Then, the fluid simulator must be able to support large viscosity values. Unfortunately, we have found that the XSPH velocity smoothing technique easily makes the fluid unstable and gains momentum if the damping coefficient c is larger than 0.5. Mixing explicit forces for viscosity with PBF also results in stability issues.

To model viscosity robustly, we propose to apply the resolution-independent XSPH model [Mon94] in an iterative manner. For damping values c larger than 0.5, we divide the application of the full viscosity into N stable XSPH iterations until they add up to the full desired viscosity behavior. In multiphase simulations, for each particle pair we use the lowest damping value. Each of the N XSPH iterations is computed as:

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \frac{1}{N}\sum_j \min(c_i, c_j)\,\frac{m_j}{\rho_j}\,(\mathbf{v}_j^n - \mathbf{v}_i^n)\cdot W(\mathbf{x}_i - \mathbf{x}_j, h_i, h_j). \tag{8}$$

Note that we average kernel evaluations, as done for incompressibility constraints, to handle fluids sampled at multiple resolutions.

With our easily modified XSPH algorithm, the fluid remains stable even under extreme viscosity without the need to decrease the time step, as shown in Fig. 5.

4.2. Surface Tension

Particle-based fluid simulators are often used for the animation of shots with small-scale liquid effects, such as the glass-filling example in Fig. 6. At such small scales, surface tension becomes a very important factor among fluid forces, hence it becomes compulsory in order to achieve realistic behavior.

In PBF, the major source of attractive forces is the correction imposed by the bilateral incompressibility constraint at low-density regions. Indeed, this attraction may be so large that Macklin and Müller [MM13] added a tensile instability repulsion force to improve results. For small-scale effects, where one expects surface tension to be dominant and hence the attractive behavior should be apparent, their repulsion forces are sufficient. For large-scale effects, however, surface tension should not be apparent, and the attractive behavior in PBF is excessive despite the addition of repulsive forces. A valid alternative for reducing attractive behavior is to convert the incompressibility constraint into an inequality constraint [AO11, GB13, MMCK14].

Figure 6: *Filling some glasses with wine shows the ability of our solver to produce realistic fine-scale scenes with emerging surface-tension details, but it also showcases the efficient support for unbounded simulation domains.*

Inspired by the work of Alduán and Otaduy [AO11] on friction simulation, we achieve a controllable surface tension behavior (see Fig. 7) through the combination of two elements: (i) limits over the range of valid incompressibility attractive forces, and (ii) a configurable stiffness for the tensile instability force. But the key to artist-friendly controllability is to expose only one parameter, a surface-tension coefficient $\kappa \in [0,1]$, and based on this we set linear functions for maximum attractive forces $f(\kappa) = 0.5\,\kappa$, and the tensile stiffness $g(\kappa) = 0.001 + 0.2\,\kappa$.

Limits on the attractive forces are simply implemented by clamping the incompressibility constraint in Eq. (6) as $\lambda_i = -\beta\, k_{\rho_{0,i}} \max(C_i, -f(\kappa))$. The tensile instability force is a correction s_{corr} that is subtracted from the Lagrange multipliers in the position correction in Eq. (7) (See [MM13] for details). By making the tensile stiffness dependent on the surface-tension coefficient, the correction is computed as:

$$s_{corr} = -g(\kappa)\left(\frac{W(\mathbf{x}_i - \mathbf{x}_j, h_i, h_j)}{W(\Delta q, h_i, h_j)}\right). \tag{9}$$

5. Collision Detection Using VDB

Many attractive fluid phenomena emerge as the result of collisions with objects. For this reason, the ability to collide with any kind of geometry robustly and efficiently is one of the major requirements for a VFX fluid simulator.

5.1. Review of Collision Detection Methods

Different methods have been proposed for representing object boundaries. One of the most adopted solutions samples boundary geometry with particles, which interact with the

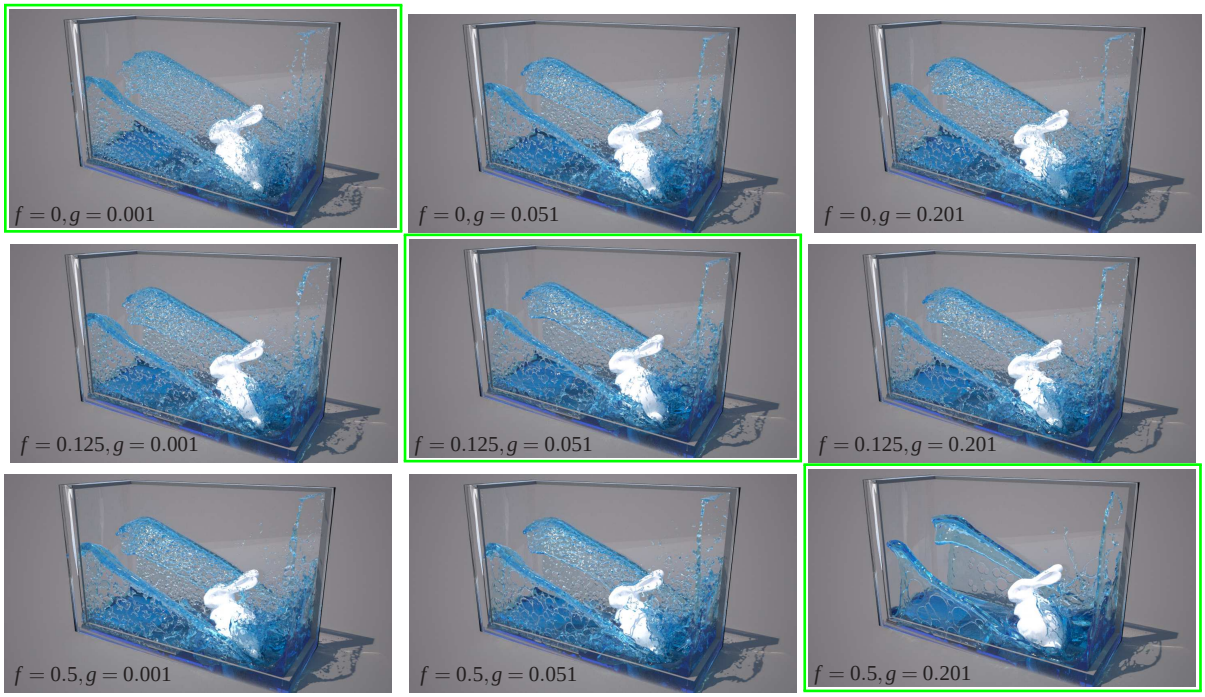

Figure 7: *Each snapshot represents the result of an animation with different surface tension settings. From top to bottom, we increase the maximum attractive force f. From left to right, we increase the tensile stiffness g. By linearly controlling both parameters through a single surface tension coefficient κ, we achieve artist-friendly control of surface tension. The snapshots in the diagonal framed in green correspond to this single-parameter control.*

fluid through penalty forces. This approach has been applied both to rigid [Mon05, AIA*12] and deformable objects [MST*04, ACAT13]. Penalty forces may be unstable under large time steps and difficult to control. Direct forcing approaches alleviate this problem [BTT09, IAGT10]. However, sampling is difficult to apply when fluids of different resolutions interact with the objects.

Alternatively, it is possible to compute the interactions directly with triangle meshes. However, particle-mesh collision detection is costly and time steps must be small to avoid penetrations.

Yet another type of representation is distance fields [JBS06]. The advantage of distance fields is that distance queries are straightforward and inexpensive [FSG03]. In PBF, collision queries need to be executed on each substep iteration, not just once per step, hence fast collision queries are crucial for efficiency. Additionally, dense distance fields can be treated as raw pointers or 3D textures, which simplifies their integration in GPU implementations [HKK07].

5.2. VDB for Fluid Collision Detection

However, production scenes are often large and contain fine details, hence dense distance-field representations would become the memory bottleneck and fail to meet the artist's resolution demands. Instead, we propose the use of adaptive distance fields [FPRJ00]. Specifically, we use narrow-band distance fields stored using VDB grids [Mus13]. The VDB grid is a tree-like data structure with different branching granularity per tree level, support for unbounded domains, activation masks, and efficient cached access.

OpenVDB is the open-source implementation of VDB provided by *Dreamworks, LLC* [Dre15]. This implementation also provides the guidelines on how to compute robust distance fields from geometry, with support for non-manifold surfaces, self-intersections, degenerate faces, and meshes without normals. Such geometric difficulties are common in the models produced by artists (see Fig. 9) and cause simulation problems if they are not properly handled.

Using VDB grids as the base data structure, we have implemented three different methods for distance field computation, which provide the versatility needed to interact with any kind of geometry. Fig. 8, Fig. 16 and Fig. 17 show examples where different methods have been used on complex geometry. For objects with clear inside-outside definition,

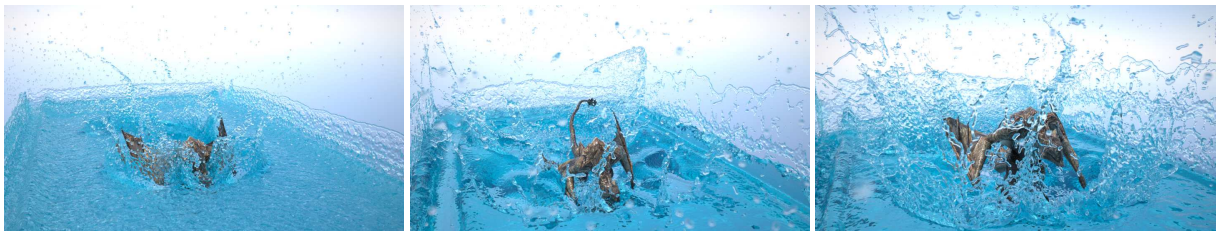

Figure 8: *A non-manifold self-intersecting animated dragon falls violently on a pool of water generating violent splashes with high detail. This is an example of a badly-conditioned model, though common in visual effects (VFX) and 3D animation studios, which our solver handles robustly.*

we provide *solid-inside* and *solid-outside* rasterization methods. For open objects we provide a *shell* rasterization method which creates an unsigned distance field and then dilates the zero-valued isosurface. On top of these methods, the user can configure the rasterization cell size, a surface offset for defining the collision isosurface, and a domain offset to define the narrow band where the distance field is computed.

The simulation of collisions with non-static geometry requires a velocity field too. For rigid bodies, the computation of velocity information is straightforward. For hand-animated deformable bodies, such as the dragon in Fig. 8, vertex velocities can be inferred through finite differences from positions in consecutive frames. Then, we propose an efficient way of converting vertex velocities into a velocity field by extending the distance field computation.

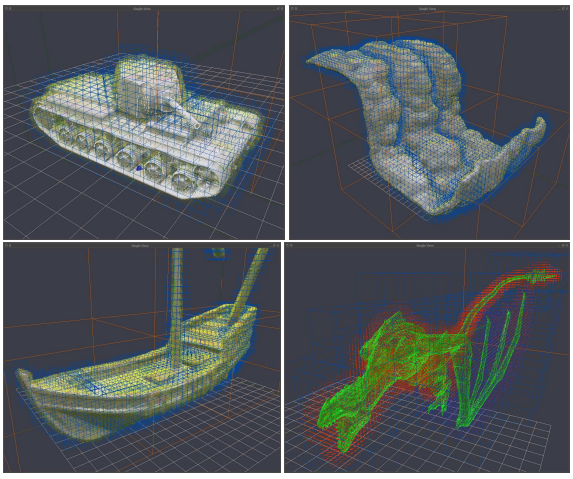

Figure 9: *We compute robustly distance and velocity fields on complex geometry. The tank and the boat are non-manifold self-intersecting geometries. The waterfall is an example of shell-mode rasterization. The wings of the dragon have no volume, hence they need an additional surface offset to be well represented.*

In addition to the distance field, we store in the narrow band an auxiliary field with indices to the closest primitives. This auxiliary field is initialized together with the distance field, on the voxels intersected by surface primitives. The auxiliary field is expanded on the narrow band together with the distance field itself at almost no additional cost. The process is demonstrated in Fig. 10. Using the auxiliary field, we compute the velocity field as follows. We copy the VDB tree topology of the distance field into an empty VDB vector field. Then, per active voxel, we obtain the index of the closest primitive from the auxiliary grid. We compute the actual point in the primitive that is closest to the voxel, use its barycentric coordinates to interpolate vertex velocities, and write the resulting velocity at the voxel. Each voxel's calculation is independent, hence the process is trivial to parallelize.

The approach described above for the computation of the velocity field can be adapted to compute any secondary field needed by the simulator. Fig. 11-bottom shows an example where variable friction coefficients are defined on the geometry as a texture. Using the auxiliary field, we first identify the closest primitive to each voxel, we compute texture coordinates through barycentric interpolation of the closest point, and we simply look-up the friction coefficient. Similarly, Fig. 11-top shows another example where a wet map is simulated by transferring simulation data to a texture thanks to the auxiliary field.

As a limitation, the VDB grid representation is a complex data structure designed to perform well on CPUs, but to date it is too complex to be fully supported on GPUs.

5.3. Two-Way Coupling

We support two-way coupling with rigid and deformable bodies through direct forcing [BTT09]. When a particle collides, we modify its momentum based on bounce, friction, and/or stickiness parameters. If the particle collides against a rigid body, we accumulate the total change of linear and angular momentum. If the particle collides against a deformable body, we use the primitive index auxiliary field of the VDB grid to access the closest triangle, and we

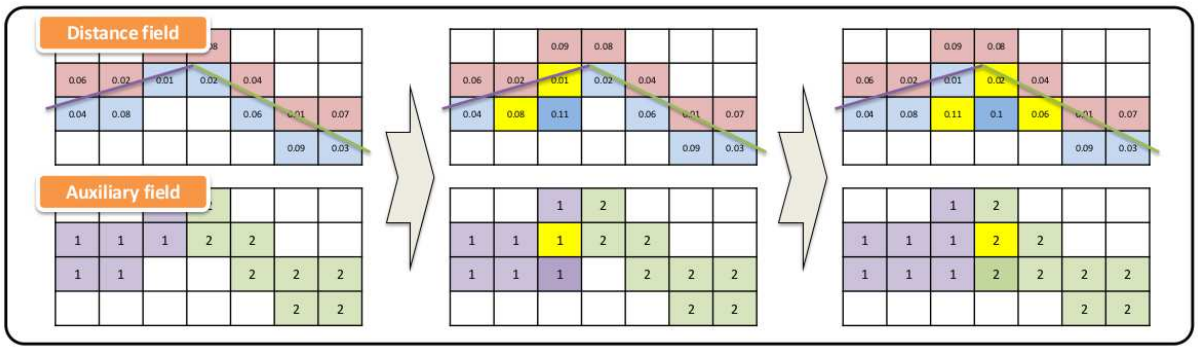

Figure 10: *Illustration of the process to compute our closest-primitive-index auxiliary field along with the expansion of the distance field. Left: the distance field and the auxiliary field are initialized by rasterizing surface primitives. Middle: the fields are expanded into a new voxel, and we compute the distance value from the values of the two neighbors that are already defined (marked in yellow), and we copy the closest-primitive index from the neighbor with shortest distance to the surface (also marked in yellow). Right: the fields are expanded into yet another voxel, and this time the distance is computed from three neighbors.*

use barycentric coordinates to accumulate the momentum change on the three vertices of the triangle. Finally, per-vertex accumulated momentum changes are transferred to the degrees of freedom of the deformable body.

Fig. 12 and Fig. 14 show examples of fluid-solid two-way coupling with rigid and deformable bodies respectively.

6. Neighbor Search Acceleration

Particle-based fluid solvers require identifying, for every particle, all the neighbors inside a predefined radius. Acceleration algorithms are necessary to avoid the brute-force computation with quadratic cost. In this section, we present an efficient solution that extends the Z-index sort method, but overcomes its simulation domain-size limitations and high memory cost.

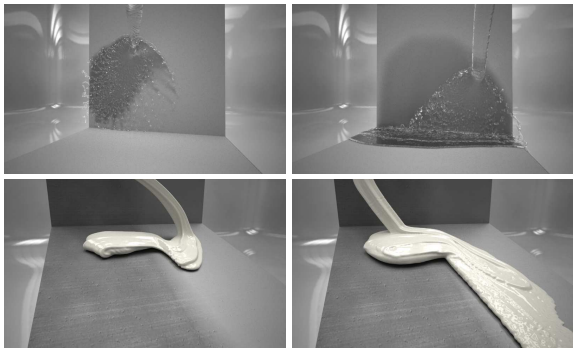

Figure 11: *Additional features provided by the closest-primitive-index auxiliary field. Top: simulation data is transferred to a texture to produce a wet map. Bottom: variable friction coefficients are looked up from a texture.*

6.1. Review of Neighbor Search Methods

Spatial subdivision data structures such as uniform grids can be used to accelerate neighbor queries, but they may become a memory bottleneck. Teschner et al. [THM*03] proposed the use of spatial-hashing to overcome this limitation, but cache-hit rates of this technique are low and hash-collisions cause additional inefficiency.

To avoid the embedding of the geometry into acceleration data structures and gain memory efficiency, index sort approaches maintain during the simulation a cell-ordered array of particle indices, while a dense map points to each cell's range into this array [OD08]. Recently, several authors proposed Z-index sort as a way to accelerate neighbor search queries for SPH, both on multi-core CPUs [IABT11] and on GPUs [GSSP10].

6.2. Improved Z-Index Sort

The Z-index sort algorithm assigns unique indices to cells based on a space-filling Z-curve, which provides good cache locality and can be easily computed by bit-interleaving. Goswami et al. [GSSP10] proposed the use of 32-bit Z-indices with 10 bits available per axis. Although GPU friendly, this solution limits the resolution of the acceleration grid to 1024 cells per axis. In production scenes not restricted to a fixed container, this limitation would result in excessively coarse grids and poor neighbor-search performance.

Instead, we propose to use 64-bit indices, which yields 21 bits per axis (See Fig. 13). This results in an acceleration grid with up to 2^{21} cells per axis, which is far more than necessary by any of today's production scenes. Nonetheless, our choice is GPU-friendly, as GPUs support 64-bit integer computations.

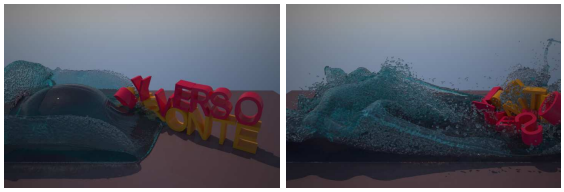

Figure 12: *Two-way interaction with rigid bodies.*

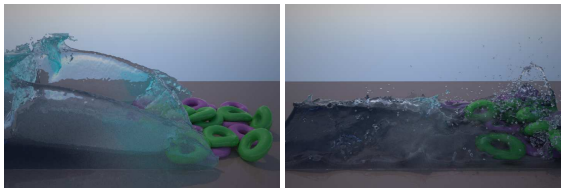

Figure 14: *Two-way interaction with deformable bodies.*

To accelerate the evaluation of the Z-index for each particle, we precompute look-up tables per coordinate that facilitate the bit offsetting. At run-time, a bit-wise *or* operation with values from these tables gives us the Z-index.

The Z-index sort method also requires a grid data structure that stores, for each cell, the range of ordered particles belonging to that cell. Previous work used a dense grid, but with a 64-bit index representation this option is not feasible. Instead, we propose to use a VDB grid to support the Z-Index sort method, thus enabling sparse storage of unbounded domains. On every simulation frame, we first build the corresponding VDB grid on the CPU. To parallelize the construction of the VDB grid, we partition the particles into the number of available cores, create one grid per core, and fill each grid storing on each occupied cell the index of the first particle in that cell. Then, we merge the grids with a reduce operation. Once the CPU VDB grid is built, we take its two lowest levels, and we transform them into two GPU buffers, one for tiles and one for leaves.

With the proposed modifications, our Z-Index sort method for neighbor-search acceleration is able to support virtually any kind of scene with sparse particle distributions. We execute a full sorting of particle data only every 20 simulation steps, and otherwise we sort only particle indices as done by Ihmsen et al. [IAGT10].

7. Particle Data Management

In this section we present details on our particle data management. The choices we made aim to find a good compromise between the efficiency of the simulator and the versatility of the proposed tools.

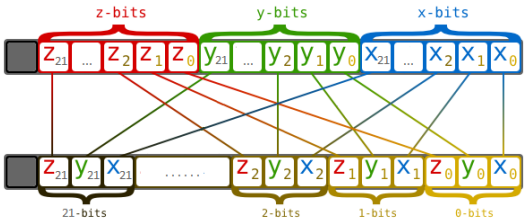

Figure 13: *Cell indices of a Z-curve represented using 64 bits, computed by bit-interleaving with 21 bits available per axis and 1 bit unused.*

In production scenarios, particle data may vary in number and size. The particle count could grow dynamically to tens of millions of particles, or drastically disappear due, e.g., to age-based particle killing. Concerning particle data size, even if the particle properties needed for the simulation are fixed, artists like the possibility to add arbitrary channels of information to the particles, which will aid them later in the rendering or compositing process.

For these reasons, we depart from the Array of Structures (AoS) approach used, e.g., by Macklin et al. [MMCK14], and use instead an Structure of Arrays (SoA), each one representing a property of the fluid. With this data structure, it is trivial to add as many channels of information as desired by the artist (as shown in Fig. 15), cache efficiency during the simulation is higher, and GPU acceleration can be easily supported by transferring to the GPU only the required channels.

However, with raw arrays of information, extra care needs to be taken to keep efficiency and intelligent use of resources as the simulation evolves, because memory copy operations and data reorganization inside the arrays could seriously affect performance. To minimize large data copy and trans-

Figure 15: *Examples where interesting effects are obtained by augmenting particle data with extra channels. On top, reversed advection of UV coordinates; and on the bottom, vorticity animation.*

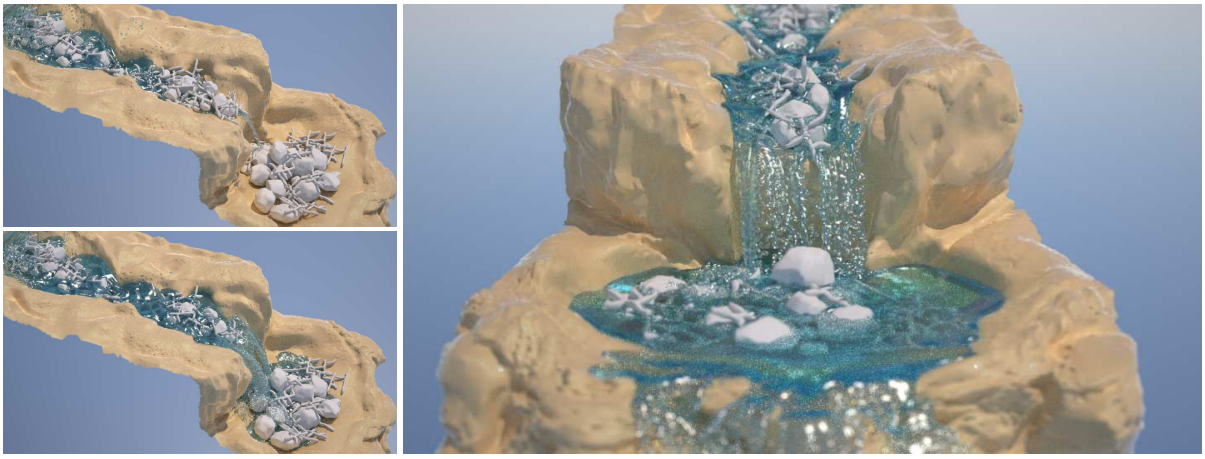

Figure 16: *Our collision detection method based on distance fields and VDB (see Section 5) provides fast and very accurate interaction with detailed geometry even in middle- or large-scale scenarios like the one shown in the images. This scene also showcases robust support of no-volume or shell geometry, often difficult to manage using distance fields.*

fer operations, we change the size of arrays only at predefined capacities chosen by experimentation, balancing memory use and cost of reallocation. To avoid excessive data reorganization as particles disappear, removal operations are handled with an *active-particles* mask, then regularly the arrays are compacted to eliminate gaps, and if occupancy falls below half, the arrays shrink to a lower predefined capacity.

To conclude, we support two different modes of memory operation. When the simulation is paused and the artist interacts with the application and/or constructs the scene, buffers are stored in regular memory. When the simulation starts, buffers are transferred to another pointer to *pinned* or non-pageable memory. The use of pinned memory is more efficient for CPU-GPU heterogeneous executions, but it is risky because it prevents the operating system from reclaiming the reserved memory. As soon as the simulation stops, buffers are returned to regular storage.

8. Results

Our PBF solver has been implemented in a completely parallel manner, using a custom Intel TBB scheduler to support symmetric optimizations on all pairwise computations [DCGG13], and supports both CUDA and OpenCL GPU acceleration. Particle sorting and collisions are handled in the CPU, while force computation is handled in the GPU. Statistics and performance for several of our examples are presented in Table 1. Timings and memory usage were measured on an Intel Core i7-3930K CPU (6 cores, 3.20GHz, 32GB RAM) and a nVidia Quadro K6000 GPU.

In all examples we have used a fixed time step of 16 ms, and the constraint solve is run until convergence. PBF enjoys fast convergence of the incompressibility constraints, particularly on the first iterations, and hence liquids tend to appear practically incompressible even under a small number of constraint solve iterations. Nevertheless, PBF also suffers the classic limitations of relaxation solvers, and it may suffer some residual large-scale error. In some examples, this translates into slight perceived bounciness. The proposed fluid model can also be configured to produce diverse behaviors, as demonstrated in Fig. 5 and Fig. 7, and it also supports two-way coupling with both rigid and deformable solids, as demonstrated in Fig. 12 and Fig. 14.

A novel feature of DYVERSO, presented in this paper, is the possibility to handle multiphase simulations. Fig. 1, Fig. 2 and Fig. 3 show three examples where fluids of different densities interact. In Fig. 1 there are violent splashes with fluids of different density, viscosity, and surface ten-

Scene	#particles ($\times 10^3$)	cost/step (ms)	memory (MB)
Fig. 2	500	364	574
Fig. 3	820	670	712
Fig. 6	1452	1158	585
Fig. 8	6390	3868	1833
Fig. 12	3500	3200	1721
Fig. 14	750	980	1090
Fig. 16	2049	1839	959
Fig. 17	1594	2338	921
Fig. 18	947	649	465

Table 1: *Statistics and performance for several scenes. The table indicates peak values for particle count, computation cost per time step (with* 16 *ms steps), and memory usage. The computation time for the examples in Fig. 12 and Fig. 14 includes rigid-body and deformable-body integration respectively.*

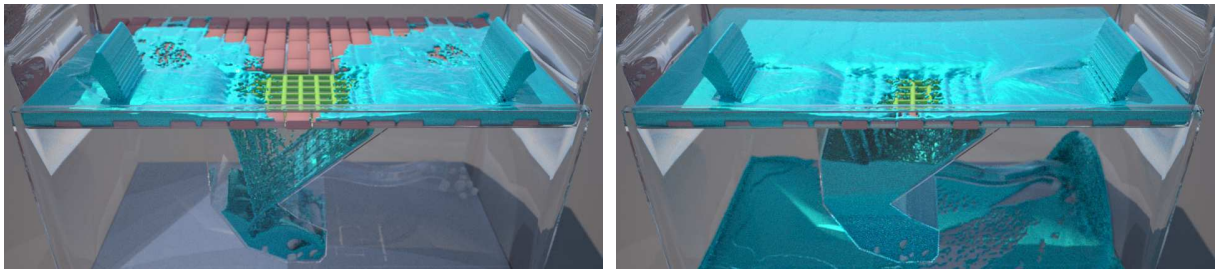

Figure 17: *This sewer drain scene showcases a potentially complex scenario for an animated feature film. The liquid covers the road filling the gaps between cobblestones, falls through the grid-like drain, accumulates in a garbage-filled holder, and the drain overflows as the exit pipe is partially clogged and cannot release all the incoming flow.*

sion. Fig. 2 shows a classic example of Rayleigh-Taylor instability where vortices emerge when the high-density liquid falls to the bottom. The density ratio is 10:1. In Fig. 3, a water-like liquid is poured on a viscous liquid. The two liquids interact, but they do not mix, due to the different density and viscosity. The viscosity ratio is 75:1 and the density ratio 2:1 (the viscous liquid is less dense).

DYVERSO can be configured on a wide range of scenarios, from small-scale to large-scale domains, with their corresponding characteristic effects. Fig. 16 shows a long, large-scale shot, where a river flows into a cascade, and water falls and accumulates naturally forming a lake. Once the lake overflows, the river flows further. This scene demonstrates the benefits of our VDB-based distance-field collision detection. By using a memory-efficient sparse distance-field representation we are able to capture all the fine detail of the original geometry and perform quick particle-boundary collision queries.

Fig. 6 shows a small-scale simulation example, where fine surface-tension details are key for the scene's realism. This scene also shows an unbounded scenario where particles fall away. The voxelization technique presented in Section 6 enables fast neighbor search operations in this scenario. The Z-Index sort algorithm preserves locality, and with the sparse storage provided by VDB it maintains memory consumption low.

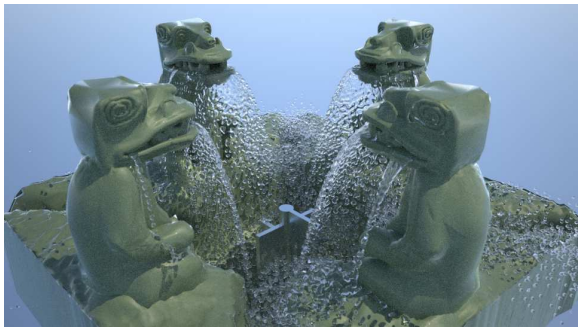

Figure 18: *In this fountain of idols, particles are dynamically created and deleted, but our particle data management handles this situation efficiently.*

Fig. 17 shows a scenario suited for an animation film, with many different geometry elements, and where the fluid naturally evolves around intricate geometry creating interesting effects.

Finally, Fig. 18 shows a scene where particles are dynamically created and deleted. This is a worst-case scenario for our particle data storage based on raw arrays, but our array reallocation policy and mask-based particle removal ensure simulation efficiency.

9. Conclusions

In this paper we have presented a production-oriented particle-based fluid simulation framework able to achieve a good compromise between efficiency, scalability, robustness and versatility. Our solver DYVERSO was integrated within RealFlow and released in 2015, and it has already been adopted by numerous customers and productions. The examples shown in the paper demonstrate just a small number of the effects we can already achieve with our solution. Yet there are many features we wish to incorporate to our simulator.

The current solution also suffers some limitations, and we would like to highlight two of them. First, we have tried to simulate density ratios as high as those in air-water interfaces, but the solver has difficulties to remain stable. Second, the XSPH admits very high viscosity values, but it damps all relative velocities, hence it dissipates angular momentum too. We would like to investigate more accurate ways to model viscous effects.

Our PBF solver is general enough to support any type of constraint. In a similar spirit to the Autodesk Nucleus solver [Sta09], arbitrary types of constraints with different levels of importance can be incorporated to the iterative solver by external programmers. We also show the ability

to simulate multiphase fluids, two-way coupling with solids, and multiple artistic effects.

Even though all the pieces of the simulator were designed targeting heterogeneous computing systems, as outlined in Section 6 and Section 7 full GPU acceleration support for our simulator is still work-in-progress. The CPU components exploit the VDB data structure, which is not trivial to port to the GPU.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments, and Next Limit RealFlow's team members for their great effort and support. This research was partially funded by the Spanish Ministry of Education, Culture and Sports (FPU fellowship ref. AP2010-4118), and grants from the Spanish Ministry of Economy (TIN2014-62143-EXP and TIN2015-70799-R) and the European Research Council (ERC Starting Grant no. 280135 Animetrics).

References

[ACAT13] AKINCI N., CORNELIS J., AKINCI G., TESCHNER M.: Coupling elastic solids with smoothed particle hydrodynamics fluids. *Comp. Anim. Virt. Worlds 24*, 3-4 (2013). 6

[AIA*12] AKINCI N., IHMSEN M., AKINCI G., SOLENTHALER B., TESCHNER M.: Versatile rigid-fluid coupling for incompressible sph. *ACM Trans. Graph. 31*, 4 (2012), 62:1–62:8. 6

[AO11] ALDUÁN I., OTADUY M. A.: Sph granular flow with friction and cohesion. In *Proc. of the 2011 ACM SIGGRAPH/ Eurographics Sympos. Comp. Anim.* (2011), pp. 25–32. 2, 5

[APKG07] ADAMS B., PAULY M., KEISER R., GUIBAS L. J.: Adaptively sampled particle fluids. *ACM Trans. Graph. 26*, 3 (2007). 4

[ATO15] ALDUÁN I., TENA A., OTADUY M. A.: Efficient and robust position-based fluids for VFX. In *Proceedings of the 2015 Spanish Computer Graphics Conference (CEIG XXV)* (2015). 3

[BIT09] BECKER M., IHMSEN M., TESCHNER M.: Corotated sph for deformable solids. In *Proceedings of the Eurographics Workshop on Natural Phenomena* (2009). 2

[BLS12] BODIN K., LACOURSIERE C., SERVIN M.: Constraint fluids. *IEEE Trans. Vis. Comp. Graph. 18*, 3 (2012). 3

[BT07] BECKER M., TESCHNER M.: Weakly compressible sph for free surface flows. In *Proc. of 2007 ACM SIGGRAPH/ Eurographics Sympos. Comp. Anim.* (2007), pp. 209–217. 2

[BTT09] BECKER M., TESSENDORF H., TESCHNER M.: Direct forcing for lagrangian rigid-fluid coupling. *IEEE Trans. Vis. Comp. Graph. 15* (2009), 493–503. 6, 7

[CBP05] CLAVET S., BEAUDOIN P., POULIN P.: Particle-based viscoelastic fluid simulation. In *Proc. of 2005 ACM SIGGRAPH/ Eurographics Sympos. Comp. Anim.* (2005), pp. 219–228. 2

[DCGG13] DOMÍNGUEZ J. M., CRESPO A. J., GÓMEZ-GESTEIRA M.: Optimization strategies for cpu and gpu implementations of a smoothed particle hydrodynamics method. *Computer Physics Communication 184*, 3 (2013), 617–627. 10

[DG96] DESBRUN M., GASCUEL M.-P.: Smoothed particles: A new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation '96* (1996), pp. 61–76. 2

[DGP12] DAGENAIS F., GAGNON J., PAQUETTE E.: A prediction correction approach for stable sph fluid simulation from liquid to rigid. In *Proc. 2012 Comput. Graph. Intern.* (2012). 2

[Dre15] DREAMWORKS, LLC: OpenVDB C++ library, 2015. URL: http://www.openvdb.org/. 6

[FPRJ00] FRISKEN S. F., PERRY R. N., ROCKWOOD A. P., JONES T. R.: Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of SIGGRAPH '00* (2000), pp. 249–254. 6

[FSG03] FUHRMANN A., SOBOTTKA G., GROSS C.: Distance fields for rapid collision detection in physically based modeling. In *Intl. Conf. Comp. Graph. and Vis.* (2003), pp. 58–65. 6

[GB13] GERSZEWSKI D., BARGTEIL A. W.: Physics-based animation of large-scale splashing liquids. *ACM Trans. Graph. 32*, 6 (2013), 185:1–185:6. 5

[GSSP10] GOSWAMI P., SCHLEGEL P., SOLENTHALER B., PAJAROLA R.: Interactive sph simulation and rendering on the gpu. In *Proc. of 2010 ACM SIGGRAPH/ Eurographics Sympos. Comp. Anim.* (2010), pp. 55–64. 8

[HKK07] HARADA T., KOSHIZUKA S., KAWAGUCHI Y.: Smoothed particle hydrodynamics on gpus. In *Proceedings of Computer Graphics International* (2007), pp. 63–70. 6

[IABT11] IHMSEN M., AKINCI N., BECKER M., TESCHNER M.: A parallel sph implementation on multi-core cpus. *Computer Graphics Forum 30*, 1 (2011), 99–112. 8

[IAGT10] IHMSEN M., AKINCI N., GISSLER M., TESCHNER M.: Boundary handling and adaptive time-stepping for PCISPH. In *Proceedings of the 7th Workshop on Virtual Reality Interaction and Physical Simulation* (2010), VRIPHYS '10, pp. 79–88. 6, 9

[ICS*14] IHMSEN M., CORNELIS J., SOLENTHALER B., HORVATH C., TESCHNER M.: Implicit incompressible sph. *IEEE Trans. Vis. Comp. Graph. 20*, 3 (2014), 426–435. 2

[IOS*14] IHMSEN M., ORTHMANN J., SOLENTHALER B., KOLB A., TESCHNER M.: Sph fluids in computer graphics. In *Eurographics 2014 State of the Art Report* (2014). 3

[JBS06] JONES M. W., BAERENTZEN J. A., SRAMEK M.: 3D distance fields: A survey of techniques and applications. *IEEE Trans. Vis. Comp. Graph. 12*, 4 (2006), 581–599. 6

[LD09] LENAERTS T., DUTRE P.: Mixing fluids and granular materials. *Computer Graphics Forum 28*, 2 (2009), 213–218. 2

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *Proc. ACM SIGGRAPH/ Eurographics Sympos. on Comp. Anim.* (2003). 2

[MHHR07] MÜLLER M., HEIDELBERGER B., HENNIX M., RATCLIFF J.: Position based dynamics. *J. Vis. Comun. Image Represent. 18*, 2 (2007), 109–118. 4

[MM13] MACKLIN M., MÜLLER M.: Position based fluids. *ACM Trans. Graph. 32*, 4 (2013), 104:1–104:12. 2, 3, 5

[MMCK14] MACKLIN M., MÜLLER M., CHENTANEZ N., KIM T.-Y.: Unified particle physics for real-time applications. *ACM Trans. Graph. 33*, 4 (2014), 153:1–153:12. 3, 4, 5, 9

[Mon94] MONAGHAN J. J.: Simulating free surface flows with SPH. *J. Comput. Physics 110*, 2 (1994), 399–406. 5

[Mon05] MONAGHAN J. J.: Smoothed particle hydrodynamics. *Reports on Progress in Physics 68* (2005), 1703–1759. 6

[MSKG05] MÜLLER M., SOLENTHALER B., KEISER R., GROSS M.: Particle-based fluid-fluid interaction. In *Proc. ACM SIGGRAPH/ Eurographics Sympos. Comp. Anim.* (2005). 2

[MST*04] MÜLLER M., SCHIRM S., TESCHNER M., HEIDELBERGER B., GROSS M.: Interaction of fluids with deformable solids. *Comp. Anim. Virt. Worlds 15*, 3-4 (2004), 159–171. 6

[Mus13] MUSETH K.: VDB: High-resolution sparse volumes with dynamic topology. *ACM Transactions on Graphics 32*, 3 (2013), 27:1–27:22. 6

[OD08] ONDERIK J., DURIKOVIC R.: Efficient neighbor search for particle-based fluids. *Journal of the Applied Mathematics, Statistics and Informatics 4* (2008), 29–43. 8

[PICT15] PEER A., IHMSEN M., CORNELIS J., TESCHNER M.: An implicit viscosity formulation for sph fluids. *ACM Trans. Graph. 34*, 4 (2015), 114:1–114:10. 2

[PTB*03] PREMOZE S., TASDIZEN T., BIGLER J., LEFOHN A. E., WHITAKER R. T.: Particle-based simulation of fluids. *Computer Graphics Forum 22*, 3 (2003), 401–410. 2

[Ree83] REEVES W. T.: Particle systems: A technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics 2*, 2 (1983), 91–108. 2

[SAC*99] STORA D., AGLIATI P. O., CANI M. P., NEYRET F., GASCUEL J.-D.: Animating lava flows. In *Proceedings of the 1999 Conference on Graphics Interface* (1999). 2

[SB12] SCHECHTER H., BRIDSON R.: Ghost sph for animating water. *ACM Trans. Graph. 31*, 4 (2012), 61:1–61:8. 5

[SF95] STAM J., FIUME E.: Depicting fire and other gaseous phenomena using diffusion processes. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques* (1995), SIGGRAPH '95, pp. 129–136. 2

[SP08] SOLENTHALER B., PAJAROLA R.: Density contrast sph interfaces. In *Proc. 2008 ACM SIGGRAPH/ Eurographics Sympos. Comp. Anim.* (2008), pp. 211–218. 2, 3

[SP09] SOLENTHALER B., PAJAROLA R.: Predictive-corrective incompressible sph. *ACM Trans. Graph. 28*, 3 (2009). 2, 4

[SSP07] SOLENTHALER B., SCHLÃĎFLI J., PAJAROLA R.: A unified particle model for fluid-solid interactions. *Computer Animation and Virtual Worlds 18* (2007), 69–82. 2

[Sta09] STAM J.: Nucleus: Towards a unified dynamics solver for computer graphics. In *Proc. 11th IEEE Intl. Conf. Computer-Aided Design and Computer Graphics.* (2009), pp. 1–11. 11

[THM*03] TESCHNER M., HEIDELBERGER B., MÜLLER M., POMERANETS D., GROSS M.: Optimized spatial hashing for collision detection of deformable objects. In *Proc. of Vision, Modeling and Visualization* (2003), VMV '03, pp. 47–54. 8