Fast reconstruction of implicit surfaces using convolutional neural networks

CHEN ZHAO, TAMAR SHINAR, and CRAIG SCHROEDER, University of California Riverside, USA

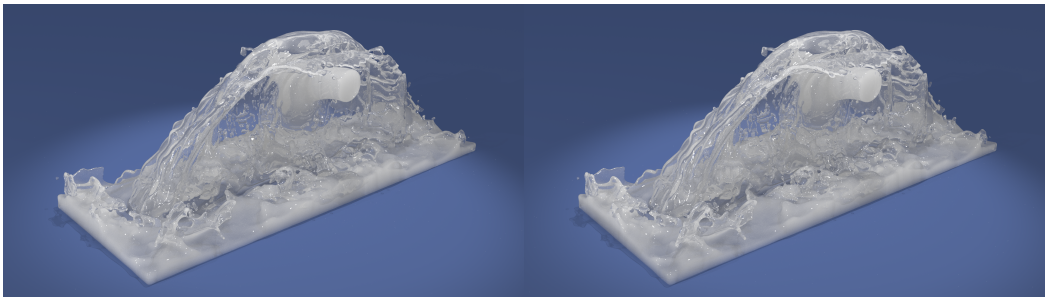


Fig. 1. Surface reconstruction from a simulation of two fluid sources colliding from opposite directions. Left: Our method. Right: Original method. The total reconstruction time using our method is 381 seconds, compared to 1880 seconds for original method, resulting in a 4.93× speedup.

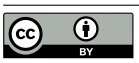
Recently, Zhao et al. [Zhao et al. 2024] proposed a new method for constructing signed distance functions from fluid simulation particles. This method was able to achieve superior surface smoothness, noise reduction, and temporal coherence compared with previous methods. One of the main limitations of the method was its relatively slow construction times, even though it utilized both the CPU and GPU. In this paper, we consider two modifications to this scheme that make the algorithm easier to optimize without introducing any perceptible changes in reconstruction quality, as illustrated in Figure 1. With these improvements, a surface can be reconstructed from a single fluid simulation with 2M particles in 2.21 seconds, compared with 72.3 seconds for the original method, resulting in a single-frame reconstruction speedup of about 33×, making the surface reconstruction fast enough for use within a simulation framework. When reconstructing surface for multiple simulation frames together, we achieve a speedup of about 5× compared with the original. The optimized implementation will be released with the publication.

CCS Concepts: • **Computing methodologies** → **Computer graphics**; **Concurrent computing methodologies**; **Machine learning**; **Shape modeling**.

Additional Key Words and Phrases: Surface reconstruction, Level sets, Particle-based fluid simulation, Rendering fluids, Machine Learning, Neural surface reconstruction

ACM Reference Format:
Chen Zhao, Tamar Shinar, and Craig Schroeder. 2025. Fast reconstruction of implicit surfaces using convolutional neural networks. *Proc. ACM Comput. Graph. Interact. Tech.* 8, 4, Article 51 (August 2025), 15 pages. https://doi.org/10.1145/3747856

Authors' Contact Information: Chen Zhao, czhao078@ucr.edu; Tamar Shinar, shinar@cs.ucr.edu; Craig Schroeder, craigs@cs.ucr.edu, University of California Riverside, Riverside, California, USA.



ACM 2577-6193/2025/8-ART51
https://doi.org/10.1145/3747856

1 Introduction

Particle-based fluid simulation techniques are widely used for special effects and games due to their strengths, which include good resolution of fluid details and natural handling of topological changes. These methods include the Lagrangian smoothed particle hydrodynamics (SPH) [Ihmsen et al. 2014; Koschier et al. 2022; Müller et al. 2003] as well as hybrid Lagrangian-Eulerian methods, such as fluid-implicit-particle (FLIP) [Zhu and Bridson 2005] and the material point method (MPM) [Stomakhin et al. 2013; Sulsky et al. 1994].

One significant challenge that remains for particle-based fluid simulation methods is the reconstruction of a high quality geometric surface for rendering. While rendering is often an offline postprocess, a surface definition is also potentially important for use in-the-loop during simulation, such as for computation of surface tension forces and accurate free-surface boundary conditions [Larionov et al. 2017], making fast reconstruction speed crucial as well. Conceptually, a simple approach to reconstruction is to use blobbies [Blinn 1982] defined by the particles, but this leads to overly bumpy surfaces, especially in regions where the surface should be flat. Typically, an implicit function is defined from the particle data, incorporating smoothing on particle positions or the implicit function to reduce surface bumpiness [Müller et al. 2003; Yu and Turk 2013; Zhu and Bridson 2005] after which methods based on the marching cubes algorithm [Lorensen and Cline 1998] are used to generate a triangle mesh.

Recently, a data-driven approach was introduced by Zhao et al. [Zhao et al. 2024]. This method begins by transferring particle data to a grid to create an implicit function akin to the SPH color field [Müller et al. 2003]. It then uses this grid data as input to a convolutional neural network that predicts signed distance field values. The method was trained on Poisson sampled particle positions of a variety of analytically defined signed distance fields. It constructs smooth, visually pleasing surface, preserves sharp features, and runs at speeds comparable to leading approaches such as Houdini [SideFX [n. d.]] and Pahi [Stomakhin et al. 2023].

In this work, we present a novel surface reconstruction method that generates high quality surface reconstructions comparable to [Zhao et al. 2024] but reconstructs a surface from particles an order of magnitude faster. We identify the bottlenecks in performance of [Zhao et al. 2024], which include feature computation and efficiency in network inference, and propose an algorithm that more efficiently utilizes both CPU and GPU. To facilitate the performance optimization of feature computation, we propose two fundamental changes to the method of [Zhao et al. 2024]. First, we propose a separable tensor-product kernel with parameters derived by minimizing the difference with the spherical kernel used in [Zhao et al. 2024]. The new kernel facilitates an efficient scanline-based parallel rasterization strategy for feature construction. Second, we replace the costly particle-to-particle density interpolation used during feature construction in [Zhao et al. 2024] with a more efficient particle-to-grid followed by grid-to-particle interpolation strategy. We further optimize the network inference stage by using TensorRT and data slicing on the GPU. We demonstrate that our reconstruction algorithm retains the high quality of [Zhao et al. 2024] while improving reconstruction on a single frame by a factor of about 33. About 41% of this cost is postprocessing overhead (reading and writing files), which would be avoided when using the new reconstruction algorithm within a simulation loop. In this case, reconstructing a surface from 2M particles would take about 1.31 seconds, making it practical for use in simulations.

The original method was difficult to parallelize efficiently and instead relied on batch reconstruction, where multiple frames are processed in parallel to improve utilization of both the CPU and GPU. Our algorithm also benefits from batching, since the total time is split approximately equally between the CPU and GPU, and these two parts must be completed sequentially. Even in batch

mode, where most threading benefits are lost, our reconstruction remains about $3-4\times$ faster, with no degradation in visual quality. An overview of our method is shown in Figure 3.

2 Related Work

Particle-based fluid simulation techniques have been widely adopted and developed for visual effects. Smoothed particle hydrodynamics (SPH) was introduced in graphics by [Müller et al. 2003], and a comprehensive survey is given in [Ihmsen et al. 2014]. Hybrid methods such as FLIP [Zhu and Bridson 2005] and MPM [Stomakhin et al. 2013; Sulsky et al. 1994] use particles as a primary material representation and transfer information to a grid to compute spatial derivatives for integrating the equations of motion.

Particle-based fluid simulations method have long struggled with the reconstruction of high quality surfaces for rendering. Müller et al. [Müller et al. 2003] used the SPH color field and yields bumpy surfaces. In the case of constant particle masses, this field corresponds to the input features m_c of [Zhao et al. 2024] and our method. Zhu and Bridson [Zhu and Bridson 2005] were able to reconstruct an implicit surface with reduced bumpiness by using weighted averages of nearby particle positions and radii. The adaptive SPH method of [Adams et al. 2007] proposed a novel surface reconstruction method designed to be less sensitive to their frequently resampled particle representation; they use particle-to-surface distances carried on particles in addition to particle locations to improve surface smoothness. A widely used method was developed by [Yu and Turk 2013], with a recent refinement of the method proposed in [Stomakhin et al. 2023], that utilizes particle position smoothing following by interpolation with anisotropic ellipsoidal, rather than spherical, kernels. The kernel shape is based on the local flow field, yielding sharper features and smoother surfaces, but with some residual artifacts such as reduced volume and bulging at edges [Zhao et al. 2024]. Surface bumpiness was also reduced by [Akinci et al. 2012] which proposed surface decimation and subdivision of the surface triangle mesh as a post-process. Williams [2008] minimized a thin-plate surface energy to generate an explicit surface mesh of particles, and Bhatacharya et al. [2011] adapted this approach to level set skinning of particle data for more smooth and temporally coherent animations. Many approaches first identify surface particles before reconstructing the surface. Sandim et al. [Sandim et al. 2016] detects surface particles based on visibility testing and then uses Screened Poisson Surface Reconstruction [Kazhdan and Hoppe 2013] to solve for a smooth surface mesh. Marrone et al. [Marrone et al. 2010] first identify surface particles and then use their locations and normals to construct a level set function.

Recent work has explored neural implicit representations for surface reconstruction. Neural implicit representations such as DeepSDF [Park et al. 2019] learn a continuous neural signed distance field for shape representation. Tao et al. [Tao et al. 2024] extend this to fluid surfaces, learning a compact signed distance field in a low-dimensional latent space to represent and evolve fluid geometry. Our work focuses on a fast parallel implementation of the recent method [Zhao et al. 2024], which uses a neural network to achieve high quality reconstruction of the fluid surface as a post-process to the fluid simulation.

Parallelization of surface reconstruction for SPH and other particle-based fluid methods has been explored on both CPU and GPU. Akinci et al. [Akinci et al. 2012] presented a parallel surface reconstruction pipeline for SPH focusing on a narrow-band marching cubes approach designed to parallelize well on shared-memory architectures. Following [Ihmsen et al. 2011], they sort particle meta data by grid z-order, and then reconstruct the level set in parallel on designated surface vertices of the grid. This was further improved using a two-level spatial uniform grid structure in [Wu et al. 2017]. Recent methods such as [Chen et al. 2021] and [Yang and Gao 2020] devised a fully parallel pipeline for particle-based fluid visualization, processing all steps on the GPU to avoid CPU-GPU memory transfers. They utilize a spatial hashing strategy to efficiently identify

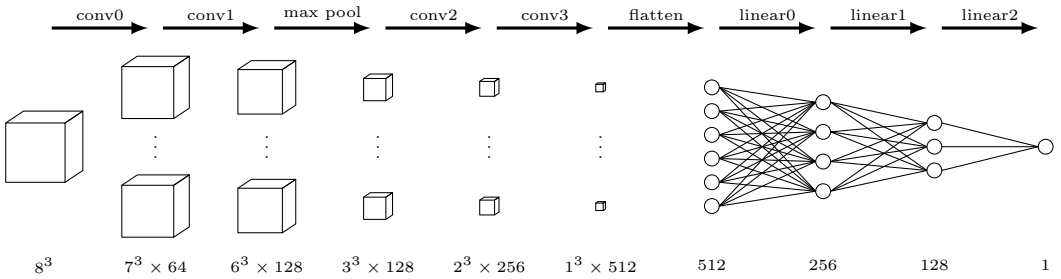


Fig. 2. Network structure for Zhao et al. [Zhao et al. 2024] (reproduced with permission).

surface particles and then apply narrow-band surface reconstruction. Quispe and Paiva [Quispe and Paiva 2022] computes a level set approximation from a smoothed discrete indicator function and proposes a fast GPU-based implementation. [Macklin and Müller 2013] proposed position based fluids with GPU-based ellipsoidal splatting, with anisotropy based on [Yu and Turk 2013], for screen space rendering of the surface in real-time based on [van der Laan et al. 2009]. Domínguez et al. [Domínguez et al. 2013] describes various CPU and GPU optimizations that are implemented in the open-source system DualSPHysics, including a GPU parallelization strategy where they sort particles by cell to improve memory access patterns and facilitate efficient neighbor search.

3 Review of Surface Reconstruction Method

We briefly review the method proposed in [Zhao et al. 2024] for completeness. For full details, we refer the reader to [Zhao et al. 2024]. Zhao et al. [Zhao et al. 2024] proposed a novel network-based approach for reconstructing signed distance fields from fluid particles. First, the method transfers the particle data to a grid. A patch M_i of grid-based features is then input to a convolutional neural network to predict the signed distance field value ϕ_i at a sample grid point i. The network loss incorporates supervised learning from Poisson sampled analytic shapes and a polynomial regularization term.

3.1 Feature construction

We found the feature construction to be a primary bottleneck in performance of [Zhao et al. 2024] and review it next. During feature construction, the particles are rasterized onto a grid with a radially symmetric smoothing kernel W as

$$m_c = \sum_{p \in N_c} \frac{1}{\rho_p} W(\|\mathbf{x}_c - \mathbf{x}_p\|, R), \quad \rho_p = \sum_{q \in N_p} W(\|\mathbf{x}_p - \mathbf{x}_q\|, R), \tag{1}$$

where $\mathbf{x}_c$ is the location of grid cell c, N_c is the set of all particles p within radius R of $\mathbf{x}_c$, and $\mathbf{x}_p$ is the position of the particle p. The density of particle p is given by ρ_p, where N_p is the set of particles within radius R, including the particle p itself. m_c is a dimensionless scalar defined at grid cell centers, which is interpolated from particles to the grid. The smoothing kernel W and particle density ρ_p have units of inverse volume. Zhao et al. [Zhao et al. 2024] used the `Poly6` kernel with $R = 3\Delta x$ as the kernel support radius, defined as

$$W(r, R) = \frac{315}{64\pi R^9}(R^2 - r^2)^3. \tag{2}$$

Note that computation of m_c transfers data from neighboring particles to cells, while the computation of ρ_p transfers data from neighboring particles to particles. While it is relatively expensive

to compute ρ_p, it serves to normalize the data to better describe surface geometry and reduce sensitivity to fluctuations in particle density. In Section 4.1, we replace this kernel with a tensor product kernel that is more efficient to compute. In Section 4.2, we use this kernel to accelerate the particle to grid transfers for calculating the features m_c. In Section 4.3, we reuse these feature transfers to calculate particle densities.

3.2 Network architecture

Another significant performance bottleneck of [Zhao et al. 2024] is network evaluation, which we address in detail below. The network structure for the method of Zhao et al. [Zhao et al. 2024] is shown in Fig. 2. The network consists of convolutional layers conv0, …, conv3 with $2 \times 2 \times 2$ kernels and stride 1, a max-pooling layer with a $2 \times 2 \times 2$ kernel and stride 2. ReLU activations are applied after conv0, max pool, conv2, and conv3. The resulting output is flattened and passed through an MLP with two hidden layers using tanh activation functions. We found that network inference time accounts for a significant portion of the total reconstruction time. Therefore, in Section 4.4, we discuss how to accelerate the inference process using the TensorRT library.

3.3 Loss function

A single forward pass of the network predicts the signed distance function ϕ_i at a grid node i. The loss function is evaluated on a $3 \times 3 \times 3$ array of predicted values centered at i, denoted by Φ_i, and is given by

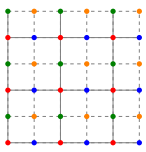
$$L = \sum_i \|\Phi_i - \Phi_i^t\|_2^2 + \lambda L_p(\Phi_i). \tag{3}$$

The first term penalizes the squared error in the predicted ϕ values. The L_p penalizes deviations from a local quadratic polynomial fit of the data. This serves to regularize the result by promoting smoothness without penalizing curvature and can be efficiently implemented as $L_p(\Phi_i) = \|K\Phi\|_2^2$ for a constant matrix K.

3.4 Network training data

The training data is generated by Poisson sampling points from a collection of shapes defined by analytic signed distance functions. Particles sampled closer than $.25\Delta x$ are also rejected to improve resolution of the reconstructed signed distance field on the grid. Four basic shapes (sphere, cone, torus, and cylinder) are generated with random position, orientation, and shape parameters. Both the interior and exterior of these shapes are sampled to capture a variety of fluid volume geometries. Additionally, a thickened boundary of each shape is sampled to train the network in reconstruction of thin fluid sheets. The shapes are sampled with particle separation distances of .5, 1, and 2 particles per cell to train the network to be effective over a variety of fluid particle densities. The training set is uniformly divided among these sampling strategies. Additionally, the data is augmented with isolated particles modeled as spheres of size Δx.

3.5 Double-fine implicit surface reconstruction



MPM simulations are commonly initialized with 8 particles per cell [Jiang et al. 2016]. It can be desirable to reconstruct a finer resolution implicit surface for higher quality results. However, if the grid resolution is doubled and the physical kernel domain shrinks commensurately, the algorithm will see only one particle per cell on average. Zhao et al. [Zhao et al. 2024] showed that this degrades results. They instead proposed to reconstruct the implicit surfaces on 8 staggered grids of the original grid resolution, where each staggered grid consists of a subset of the nodes of the double-fine grid (inset

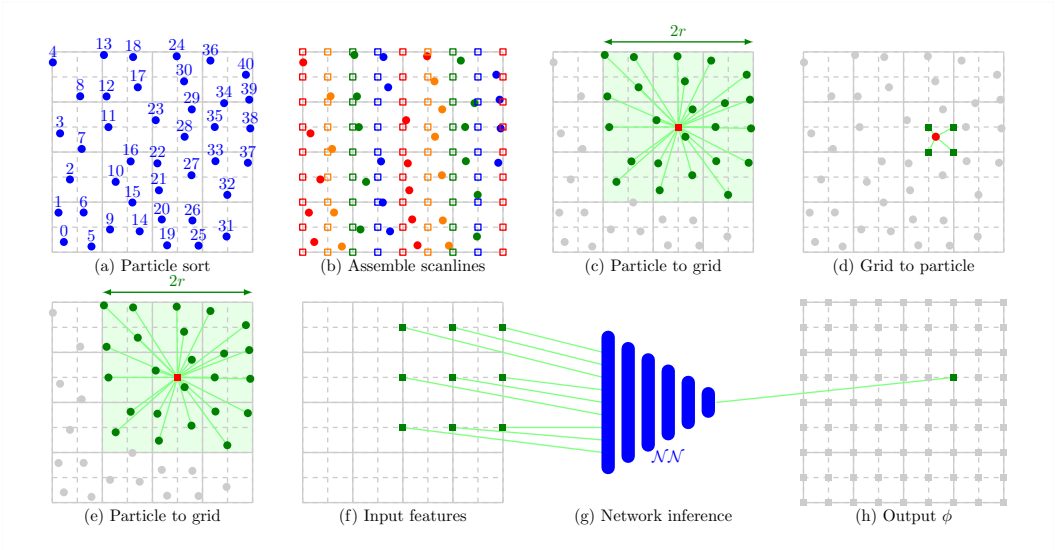


Fig. 3. Overview of our reconstruction algorithm. (a) We begin by sorting particles so that they occur in the same order as the double-fine grid cell they live in. (b) Next, we associate particles and grid nodes with scanlines. (c) To compute particle density ρ_p, we first compute density on the grid nodes and then (d) interpolate the result back to particles. (e) Using the particle density we compute features on grid nodes. (f) For each node on the double-fine grid, we collect an $8 \times 8 \times 8$ neighborhood of features, which are used as input to the (g) network. The output of this network is the level set value, which is stored (h).

figure). In that way, a reconstruction on the double-fine grid is obtained without doubling the grid resolution used for feature construction and network inference.

4 Approach

The primary bottleneck with the reconstruction of Zhao et al. [Zhao et al. 2024] is the feature computation. The core of this computation is transferring information from particles to the grid to assemble the features used as input to the network. This is done by evaluating a kernel for each particle-grid pair. In practice, this is one of the most computationally intensive steps in the algorithm, so we begin our optimization with the kernel itself.

4.1 Kernel

The original implementation used the Poly6 kernel in (2) with a support radius of $R = 3\Delta x$. This kernel had the advantage of being a polynomial and thus relatively efficient to compute. This kernel is especially popular in SPH for particle-particle interactions. The spherical symmetric kernels are the norm for SPH simulations, since it allows them to be rotation invariant. Our reconstruction uses features computed on a regular grid, so it is not rotation invariant. On the other hand, for hybrid particle-grid methods, such as the material point method (MPM), tensor product kernels are the norm: $W(x, y, z) = f(x)f(y)f(z)$, where the kernel depends on the displacement between locations rather than the distance. Tensor product kernels are more efficient when transferring between particles and grids because they take advantage of the regular arrangement of grid nodes; computing all of the weights for one particle at once allows many of the polynomial computations to be reused. We follow this and adopt the tensor product kernel

$$W(\mathbf{x}, r) = W(x, y, z, r) = k(r^2 - x^2)^2(r^2 - y^2)^2(r^2 - z^2)^2. \tag{4}$$

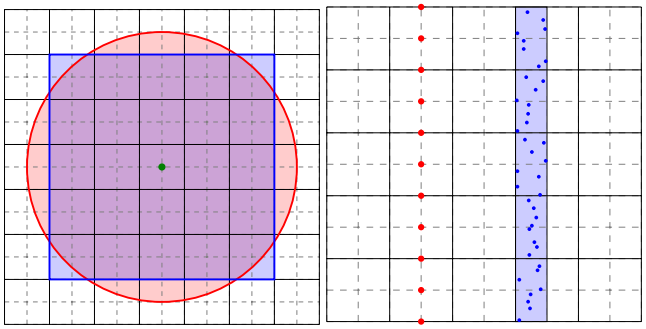


Fig. 4. (Left) Old kernel (red) and new kernel (blue) showing region of space of which particles affect a grid cell (green). Particles are associated with double-fine grid cells, shown dashed. The new kernel's support aligns with the grid. Grid cell centers for the original grid correspond to nodes of the double-fine grid. The green location receives information from particles within a 10×10 neighborhood on the double-fine grid. (Right) Rasterization occurs between an output scanline (red) of double-fine grid nodes and an input scanline (blue) of particles.

Unlike the tensor product cubic or quadratic splines typically used for MPM, we instead opted for tensor product quartic polynomials. This quartic was chosen as the smallest-degree nonzero polynomial that vanishes and has zero derivative at $\pm r$. We chose this kernel because it has a few useful properties. (1) The support of the kernel aligns with the grid cells; since neighbors are identified based on the cell they lie in, no unnecessary particles are visited and no separate radius check is required. (2) Along a scanline only the $(r^2 - z^2)^2$ factor changes; the other factors can be reused. (3) Unlike piecewise polynomial spline-based weights, all grid locations along a scanline are computed using exactly the same quartic polynomial. This allows kernel computations to be accelerated using SIMD instructions.

Kernel rescaling. The new kernel has two degrees of freedom: k and r. The normalization factor k does not matter for our purposes, since this will eventually cancel with the density. We choose $k = r^{-6}$ so that its maximum is $W(\mathbf{0}, r) = 1$. The radius r is more delicate; simply using $r = 3$ as in the original method makes the reconstruction visually worse. We note that the difference between the proposed kernel (for appropriate k) and the Poly6 kernel is minimized in the L_2 sense when $r = 2.43$, so we use $r = 2.5$ for our tensor product kernel. With these parameters, we do not observe any difference in quality between the original reconstruction and the new construction. See Figure 4 for a comparison of the kernels.

4.2 Particle-to-grid transfers

Most of the CPU time in the original method was spent accumulating particle density ρ_p (particle-to-particle transfer) and then accumulating the features m_c (particle-to-grid transfer). We begin by optimizing the particle-to-grid transfer. We note that in our implementation arrays are stored in row-major order, where indices differing in the z coordinate are adjacent in memory. Since we will do a $2 \times 2 \times 2$ staggered reconstruction, we perform grid transfers on a double-fine grid, computing the features for all eight reconstructions at once.

Particle sorting. The first step of the particle-to-grid transfer is identifying particle-cell pairs. Originally, this was done by binning. Although simple, this strategy has a few significant disadvantages. It is difficult to parallelize, since care must be taken to avoid writing to the same or nearby memory on different threads. The construction of large numbers of tiny arrays also does not utilize

Algorithm 1 Accumulate features on grid.

$\Delta s \leftarrow \frac{\Delta x}{2}, R \leftarrow 5\Delta s$ ▷ Double-fine grid size, kernel size
Sort particles by double-fine grid cell
$\{L_0, L_1, \ldots, L_{T-1}\} \leftarrow (0,0), (0,1), \ldots, (2n, 2n)$ ▷ Divide scanlines among threads
for $(g_x, g_y) \in L_t$ **do** ▷ Each thread t visits its output scanlines
  for $p_x \in \{g_x - 5, \ldots, g_x + 4\}$ **do** ▷ Visit neighboring scanlines of particles
    for $p_y \in \{g_y - 5, \ldots, g_y + 4\}$ **do**
      for $p \in P_{p_x,p_y}$ **do** ▷ For each particle in scanline (p_x, p_y)
        $(i, j, k) \leftarrow \textsc{Index}(p)$ ▷ Double-fine grid index for particle p.
        $(x, y, z) \leftarrow \textsc{Location}(p)$
        $a \leftarrow \frac{1}{\rho_p}\left(1 - \left(\frac{x - g_x\Delta s}{R}\right)^2\right)^2\left(1 - \left(\frac{y - g_y\Delta s}{R}\right)^2\right)^2$
        for $r \in \{k - 5, \ldots, k + 4\}$ **do** ▷ For each affected output index (SIMD)
          $m_{g_x,g_y,r} \leftarrow m_{g_x,g_y,r} + a\left(1 - \left(\frac{z - r\Delta s}{R}\right)^2\right)^2$ ▷ Accumulate kernels for scanline

caching well. Instead, we sort the particles based on the index of the cell that they are in, ordered in the same way cells data is stored in memory. Parallel sorting is very efficient, and we simply use the C++ standard library implementation.

Scanlines. Next we need to divide the transfer work between threads. We choose scanlines as our basic unit of work. Our scanlines consist of all grid cells whose x and y indices are the same; scanlines are contiguous blocks of memory. Particles within the grid cells of a scanline are associated with the scanline; these particles are also contiguous in memory due to the sorting. We construct a simple 2D array of structures to store information per scanline, since this information will be used frequently: the range of the output array, the range of the particles array, the location in space and grid index of the first cell of the scanline, and the number of particles that occur in this scanline or any previous scanline. The particle array ranges are computed by binary search. Everything else can be computed directly. The cumulative particle count allows us to quickly divide the scanlines between the threads so that the scanlines for each thread contain approximately the same number of particles. Each thread is responsible for computing the features for grid cells within its scanlines.

Summation of scanline pairs. The final step in the transfers is visiting the grid-particle pairs, computing the kernels, and adding the results. Our basic unit of work for this step is scanline pairs (See Figure 4). For each particle in the first scanline, we add its contributions to grid cells in the second scanline; see Algorithm 1. In this way, rasterization occurs on a relatively small amount of memory at a time, with both particles and grid nodes being visited in order along their scanlines. As noted in Algorithm 1, most of the kernel computations for a particle can be reused across the scanline due to the tensor product structure of the kernel. Since the number of nodes visited along a scanline is constant (ten) and the computations are identical along the scanline (since our kernel is not piecewise), we are able to trivially parallelize the inner loop of the kernel computation using SIMD. We allocate a ghost layer around the grid so that writes can be safely performed without array bounds checks. This also helps separates scanlines in memory to avoid thread conflicts.

Full summation. Each thread is assigned a contiguous subset of the scanlines, which are divided up so that each contains a similar number of particles; this is accomplished using binary search. Although threads also read particles from nearby scanlines, this helps to balance the work among

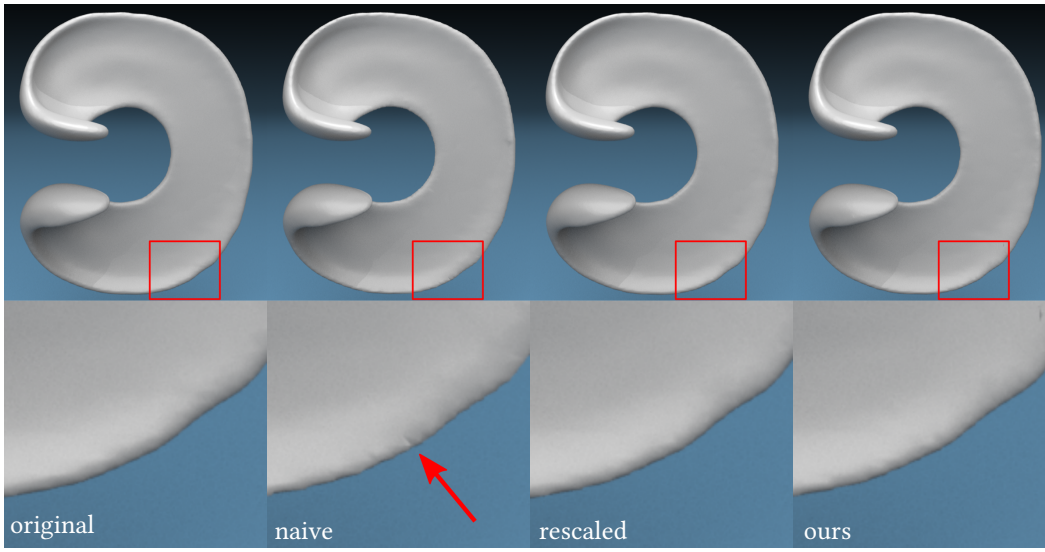


Fig. 5. This figure shows the qualitative effects of replacing the modifications to the surface reconstruction algorithm. (Left) original reconstruction, (left-middle) naively replacing the original kernel with the tensor product kernel, (right-middle) rescaling the kernel to minimize differences between kernels, and (right) using rescaled kernel and interpolating particle density from grid. Naive replacement of the kernel produces noticeable artifacts at the edge of the object (see the arrow in the blow-up). As long as the kernel is properly rescaled, the reconstruction quality is not affected.

threads even when particles are not uniformly distributed through the domain. To do the rasterization, each thread computes the features for the scanlines it has been assigned. For each of these output scanlines, it calls the scanline-scanline rasterization for all particle scanlines within the support of the kernel. Performing summation in this way is very efficient; we are able to evaluate kernels for 4G particle-grid pairs on 32 cores on the CPU in about 75ms.

4.3 Density computation

The feature computation makes use of particles densities ρ_p, which are computed from particle-particle pairs in the original algorithm. Particle-particle neighborhoods are generally more expensive to find and visit than particle-grid neighborhoods. We instead compute densities on grid nodes and then interpolate them to particles using trilinear interpolation. This operation is conveniently identical to the feature computation when $\rho_p = 1$. We are thus able to reuse our feature computation by initializing $\rho_p = 1$, performing the rasterization, interpolating from grid to particles to get the real ρ_p, and then rasterizing a second time. We did not observe any visual degradation in the resulting surface when making this change.

4.4 Network Inference optimization

After optimizing the feature computation on the CPU, the major performance bottleneck shifts to the network inference portion of the algorithm. In this section, we propose some optimizations to the implementation of the network inference process.

4.4.1 Slicing on GPU. In the network inference process of the original paper, the input features are extracted by doing tensor slicing on the feature grid on CPU. Then, batches of input features are

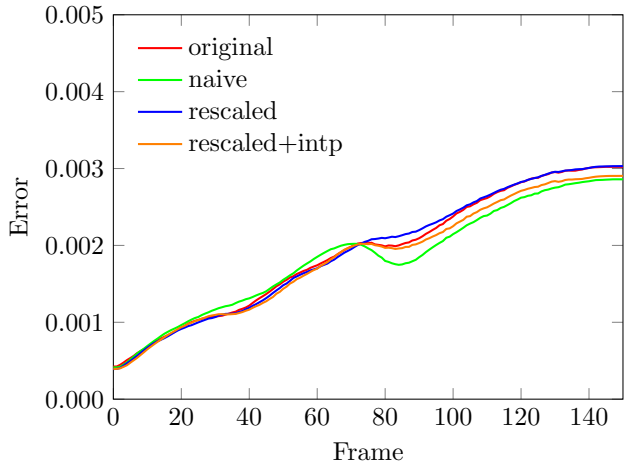


Fig. 6. Quantitative analysis on 3D vortex deforming bumpy sphere between (a) original reconstruction, (b) naively replacing the original kernel with the tensor product kernel, (c) rescaling the kernel to minimize differences between kernels, and (d) using the rescaled kernel and interpolating particle density from the grid.

sent to GPU for the network inference, and the predicted signed distance values are sent back to CPU and written to the signed distance array. The process was repeated until the signed distance value for all grid nodes were obtained. This repeated transfer between CPU and GPU requires multiple PCI transaction initialization which slows down network inference process. To fully utilize the GPU parallelization resources and reduce PCI latency cost, we increase the input feature batch size from 256 to 1024 and implement a CUDA kernel to parallelize the tensor slicing process on GPU, so that both tensor slicing and network inference will be executed on GPU. By doing this, we only need send the kernel feature array to GPU once and retrieve the signed distance field array after all inference are completed, eliminating the redundant transferring process.

4.4.2 TensorRT. NVIDIA TensorRT [Corporation [n. d.]] is a high-performance SDK designed to accelerate deep learning inference. It works alongside popular training frameworks like TensorFlow, PyTorch, and MXNet, focusing on optimizing and executing pre-trained neural networks efficiently on NVIDIA GPUs. TensorRT accelerates inference through various techniques, including precision reduction, layer fusion, and kernel auto-tuning. To optimize the network inference process, we convert the trained network model to the TensorRT engine to accelerate the network inference. We use reduced FP16 precision instead of FP32 as it is faster and does not affect on the surface reconstruction quality. Additionally, since the current network structure has multiple convolution layers followed by ReLU activation layers, the model is further optimized with TensorRT's layer fusion feature.

5 Results

5.1 Ablation study

In this section, we compare the reconstruction method on a warped sphere using four feature computation approaches: (1) original reconstruction, (2) naively replacing the original kernel with the tensor product kernel, (3) rescaling the kernel to minimize differences between kernels, and (4) using the rescaled kernel and interpolating the particle density from grid. Version (2) leads to visual artifacts, as shown in Figure 5. In this case, the edge of the reconstructed surface has become less smooth compared with the original. Rescaling the kernel removes the artifacts, and changing the

Sim	Ours	Original	Speedup
Dam break	134	657	4.90x
Double dam break	113	541	4.79x
Two source	381	1880	4.93x
One source	125	604	4.82x
Sphere drop	113	746	6.6x
Bunny	43	172	4.03x
Ring drop	39	146	3.76x
Average	135	648	5.02x

Fig. 7. Reconstruction times (seconds) for each of the fluid simulation reconstructions.

particle-particle density computation into a particle-grid computation followed by interpolation does not significantly alter the visual appearance.

Reconstruction error. To quantitatively analyze the accuracy of the four reconstruction settings, we follow the approach in [Zhao et al. 2024] and perform the Enright test [Enright et al. 2002]. We seed initial warped sphere with particles and advect these particles using third-order Runge-Kutta. At each time step we reconstruct the surface from the seeded particles. To track the actual interface, we also seed surface particles at the beginning and advect them through the velocity field. The error is computed as $E = \frac{1}{N}\sum_i \frac{|\phi(\mathbf{x}_i)|}{\|\nabla\phi(\mathbf{x}_i)\|}$, where the gradient is obtained via finite differences. For a fair comparison, we shift surface particles to the isocontour that minimizes error in the initial state. The results are shown in Figure 6. We observe similar quantitative errors in the four approaches, suggesting that the changes to the algorithm have not significantly impacted the quantitative surface reconstruction accuracy of the method.

5.2 Timing results

In this section, we investigate the performance of the proposed algorithm more thoroughly. We present all timing experiments on a machine with an AMD Ryzen Threadripper PRO 5975WX CPU with 32 cores and 512GB memory and an NVIDIA GeForce RTX 4090 GPU.

Single-frame timing experiments. In Figure 8, we present the time decomposition for each reconstruction process on a single frame with 2M particles using our method and the original approach. The original reconstruction requires 72.3 seconds. Reconstructing our algorithm using only a single thread reduces this to 5.11 seconds, resulting in a speedup of about 14×. Enabling threading reduces this further to 2.21 seconds, bringing the speedup to about 33×. Of this 2.21 seconds, about 0.90 seconds is spent reading inputs, writing outputs, and loading the network from disk. If surface reconstruction were used within a simulation, such as for computing boundary conditions or surface tension, these costs would be avoided, reducing the time to 1.31 seconds. This is fast enough to be run every time step of a simulation.

The meager speedup from 5.11 to 2.21 using 32 cores requires some explanation. Of this time, the 0.90 seconds spent on IO is not parallel, and the 1.01 seconds spent doing network inference occurs on the GPU and not the CPU. Removing these from the timings brings the threading-eligible times to 3.2 seconds and 0.30 seconds, resulting in a speedup of 10.7× due to the threading.

Batch-mode timing experiments. We repeated all of the surface reconstructions for the large simulation examples from original paper; side-by-side comparisons are shown in Figure 9 for three of the sims for visual comparison. Side-by-side comparisons for all of the large simulations are shown in the accompanying video. The corresponding timing information is shown in Figure 7.

Process	Ours (32 cores)	Ours (1 core)	Original
Total time	2.21	5.11	72.3
Read input from disk	0.11	0.11	0.09
Scanline initialization	0.03	0.21	-
Density computation	0.06	1.19	19.2
Feature computation	0.05	1.08	35.8
Pruning	0.06	0.59	0.58
GPU - model inference	1.01	1.05	16.0
Write to disk	0.68	0.63	0.64
Load model	0.11	0.12	0.10
Others	0.10	0.13	0.13

Fig. 8. Reconstruction times (seconds) for each reconstruction step on a single frame

For these timing comparisons, surfaces for all of the frames are constructed in batch mode, where multiple surfaces can be constructed in parallel. Batch processing is critical for the original method to achieve acceptable performance, since it does not otherwise parallelize well. The proposed method also benefits from it, since GPU and CPU computations can overlap, and IO can overlap with other computations on the CPU. The results show that for all the simulation examples, our proposed algorithm achieves an average of 5× speedup when compared with the original method in batch mode.

6 Conclusion and Limitations

In this paper, we presented a modified version of the network-based surface reconstruction algorithm of [Zhao et al. 2024], which uses a scanline-based transfer algorithm, an efficient tensor-product kernel, and a grid-based density computation strategy. This results in a speedup of about 33× for single surface reconstruction, bringing the cost of high-quality surface reconstruction from fluid simulation particles low enough to be useful within a fluid simulator. For full-simulation surface reconstruction, we achieve a 5× speedup over the original method, making the new method significantly faster than state of the art algorithms for high-quality surface reconstruction from fluid particles.

The proposed algorithm and its implementation divide the work between CPU and GPU, resulting in significant overhead transferring information between them. Further performance improvements may be possible by performing the entire algorithm on the GPU. We occasionally observed mild flickering in animations due to single-frame surface shape errors. While this effect appears in both the original and accelerated methods and does not significantly impact visual quality, reducing such temporal artifacts could be a valuable direction for future work. Although the reconstruction performance is fast enough for use within an MPM-style or SPH-style simulation framework, we have not explored the suitability of the reconstructed surface for this purpose. Curvature computations are particularly sensitive to errors in level sets, and simulations are much less tolerant of large but infrequent errors than rendering, though this may be mitigated somewhat using error reduction techniques such as [Larios-Cárdenas and Gibou 2022]. We leave this for future work.

Acknowledgments

The simulations in this paper were rendered using SideFX Houdini. This work was supported by University of California award M23PL6076.

References

Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J Guibas. 2007. Adaptively sampled particle fluids. In *ACM SIGGRAPH 2007 papers*. 48–es.

Gizem Akinci, Nadir Akinci, Markus Ihmsen, and Matthias Teschner. 2012. An efficient surface reconstruction pipeline for particle-based fluids. (2012).

G. Akinci, M. Ihmsen, N. Akinci, and M. Teschner. 2012. Parallel Surface Reconstruction for Particle-Based Fluids. 31, 6 (2012), 1797–1809. doi:10.1111/j.1467-8659.2012.02096.x

Haimasree Bhatacharya, Yue Gao, and Adam Bargteil. 2011. A level-set method for skinning animated particle data. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics symposium on computer animation*. 17–24.

James F Blinn. 1982. A generalization of algebraic surface drawing. *ACM transactions on graphics (TOG)* 1, 3 (1982), 235–256.

Qiaorui Chen, Shuai Zhang, and Yao Zheng. 2021. Parallel realistic visualization of particle-based fluid. *Computer Animation and Virtual Worlds* 32, 3-4 (2021), e2019.

NVIDIA Corporation. [n. d.]. TensorRT. https://developer.nvidia.com/tensorrt Accessed: [Insert Date of Access].

Jose M Domínguez, Alejandro JC Crespo, and Moncho Gómez-Gesteira. 2013. Optimization strategies for CPU and GPU implementations of a smoothed particle hydrodynamics method. *Computer Physics Communications* 184, 3 (2013), 617–627.

Douglas Enright, Ronald Fedkiw, Joel Ferziger, and Ian Mitchell. 2002. A hybrid particle level set method for improved interface capturing. *Journal of Computational physics* 183, 1 (2002), 83–116.

Markus Ihmsen, Nadir Akinci, Markus Becker, and Matthias Teschner. 2011. A parallel SPH implementation on multi-core CPUs. In *Computer Graphics Forum*, Vol. 30. Wiley Online Library, 99–112.

Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. 2014. SPH fluids in computer graphics. (2014).

Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. 2016. The material point method for simulating continuum materials. In *ACM siggraph 2016 courses*. 1–52.

Michael Kazhdan and Hugues Hoppe. 2013. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)* 32, 3 (2013), 1–13.

Dan Koschier, Jan Bender, Barbara Solenthaler, and Matthias Teschner. 2022. A survey on SPH methods in computer graphics. In *Computer graphics forum*, Vol. 41. Wiley Online Library, 737–760.

Egor Larionov, Christopher Batty, and Robert Bridson. 2017. Variational stokes: A unified pressure-viscosity solver for accurate viscous liquids. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–11.

Luis Ángel Larios-Cárdenas and Frederic Gibou. 2022. A hybrid inference system for improved curvature estimation in the level-set method using machine learning. *J. Comput. Phys.* 463 (2022), 111291.

William E Lorensen and Harvey E Cline. 1998. Marching cubes: A high resolution 3D surface construction algorithm. In *Seminal graphics: pioneering efforts that shaped the field*. 347–353.

Miles Macklin and Matthias Müller. 2013. Position based fluids. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–12.

Salvatore Marrone, Andrea Colagrossi, David Le Touzé, and Giorgio Graziani. 2010. Fast free-surface detection and level-set function definition in SPH solvers. *J. Comput. Phys.* 229, 10 (2010), 3652–3663.

Matthias Müller, David Charypar, and Markus Gross. 2003. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Citeseer, 154–159.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. Deepsdf: Learning Continuous Signed Distance Functions for Shape Representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019). 165–174.

Filomen Incahuanaco Quispe and Afonso Paiva. 2022. Counting Particles: A Simple and Fast Surface Reconstruction Method for Particle-Based Fluids. In *2022 35th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)* (2022), Vol. 1. IEEE, 145–149.

Marcos Sandim, Douglas Cedrim, Luis Gustavo Nonato, Paulo Pagliosa, and Afonso Paiva. 2016. Boundary Detection in Particle-based Fluids. In *Computer Graphics Forum*, Vol. 35. Wiley Online Library, 215–224.

SideFX. [n. d.]. *Houdini*. SideFX. https://www.sidefx.com/products/houdini/ Computer software.

Alexey Stomakhin, Steve Lesser, Joel Wretborn, Sean Flynn, Johnathan Nixon, Nicholas Illingworth, Adrien Rollet, Kevin Blom, and Douglas Mchale. 2023. Pahi: A Unified Water Pipeline and Toolset. In *Proceedings of the Digital Production Symposium* (2023). 1–13.

A. Stomakhin, C. Schroeder, L. Chai, J. Teran, and A. Selle. 2013. A material point method for snow simulation. In *ACM Transactions on Graphics (SIGGRAPH 2013)*. 102:1–10.

D. Sulsky, Z. Chen, and H. Schreyer. 1994. A particle method for history-dependent materials. *Comp Meth in App Mech Eng* 118, 1 (1994), 179–196.

Yuanyuan Tao, Ivan Puhachov, Derek Nowrouzezahrai, and Paul Kry. 2024. Neural Implicit Reduced Fluid Simulation. In *SIGGRAPH Asia 2024 Conference Papers*. 1–11.

Wladimir J van der Laan, Simon Green, and Miguel Sainz. 2009. Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. 91–98.

Brent Warren Williams. 2008. *Fluid surface reconstruction from particles*. Ph. D. Dissertation. University of British Columbia.

Wei Wu, Hongping Li, Tianyun Su, Haixing Liu, and Zhihan Lv. 2017. GPU-accelerated SPH Fluids Surface Reconstruction Using Two-Level Spatial Uniform Grids. 33, 11 (2017), 1429–1442. doi:10.1007/s00371-016-1289-x

Wencong Yang and Chengying Gao. 2020. A completely parallel surface reconstruction method for particle-based fluids. *The Visual Computer* 36 (2020), 2313–2325.

Jihun Yu and Greg Turk. 2013. Reconstructing Surfaces of Particle-Based Fluids Using Anisotropic Kernels. 32, 1 (2013), 1–12.

Chen Zhao, Tamar Shinar, and Craig Schroeder. 2024. Reconstruction of implicit surfaces from fluid particles using convolutional neural networks. In *Computer Graphics Forum*, Vol. 43. Wiley Online Library, e15181.

Y. Zhu and R. Bridson. 2005. Animating sand as a fluid. *ACM Trans Graph* 24, 3 (2005), 965–972.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009

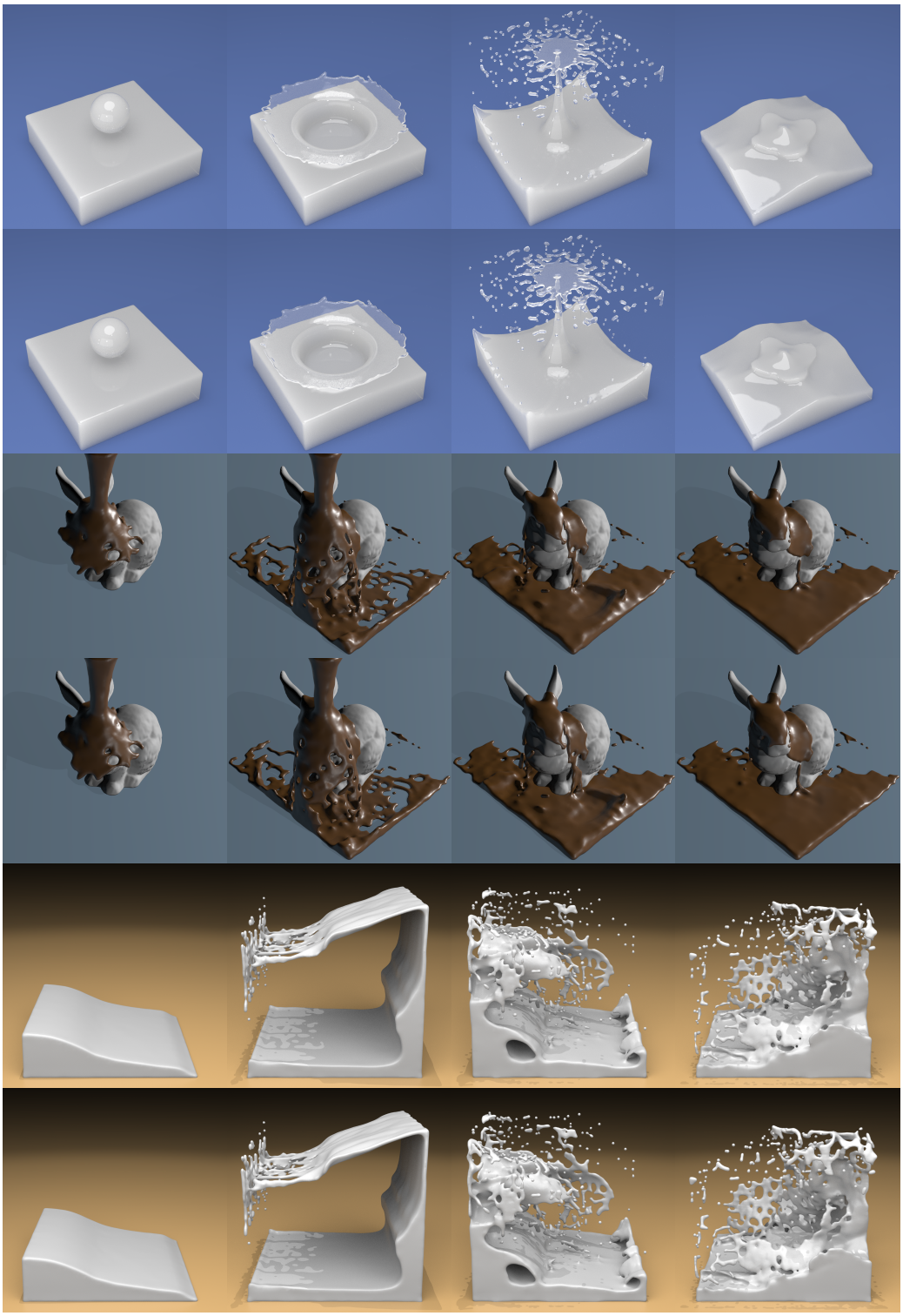


Fig. 9. Side-by-side comparison of renders, with our method above and original method below.