Eurographics Symposium on Parallel Graphics and Visualization (2018)
H. Childs, F. Cucchietti (Editors)

Direct Raytracing of Particle-based Fluid Surfaces Using Anisotropic Kernels

T. Biedert[1], J.-T. Sohns[1], S. Schröder[2], J. Amstutz[3], I. Wald[3] and C. Garth[1]

[1]Technische Universität Kaiserslautern, Germany
[2]Fraunhofer ITWM, Germany
[3]Intel Corporation

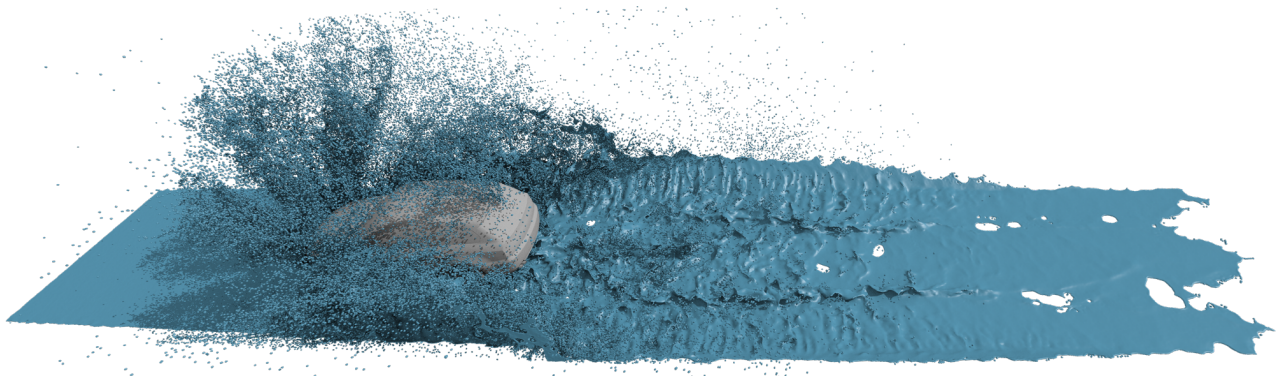


Figure 1: *Watercrossing simulation using 2.1 million particles. Rendered at 4K resolution with ambient occlusion and hard shadows in 2.4 seconds on 32 Xeon Phi KNL nodes of the Stampede2 supercomputer.*

Abstract

Particle-based simulation models have assumed a significant role in the numerical computation of high-fidelity transient flow and continuum mechanical problems. However, direct visualization of surfaces from particle data without intermediary discrete triangulation remains a challenging task. We demonstrate a novel direct raytracing scheme for free surface intersection based on anisotropic smoothing kernels. Our approach efficiently reduces the number of candidate kernels evaluated to converge to the surface threshold, thereby running in image space rather than object space complexity. We conduct comprehensive benchmarks with respect to data set size, scene complexity, visual fidelity and hardware setup. Our versatile system is suitable for both high quality and interactive desktop rendering, scales reasonably well even with trivial parallelization and renders up to 170 million particles on 32 distributed compute nodes at close to interactive frame rates at 4K resolution with ambient occlusion.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing

1. Introduction

The visualization of numerical simulation results based on mere point clouds is a challenging task. Such data sets emerge from Smoothed Particle Hydrodynamics (SPH) or Finite Pointset Method (FPM) simulations, two particle-based simulation techniques in the context of transient flow and continuum mechanical problems. In scenarios with free surfaces or moving geometry, classical grid-based numerical procedures, e.g., Finite Elements or Finite Volumes, fail due to their inherent necessity for remeshing.

However, currently there exist few well-elaborated standard visualization approaches tailored to grid-free methods. A multitude of contemporary rendering techniques are grid-based, and thus inappropriate for the evaluation and analysis of particle-based simulation results. Furthermore, for high density point clouds with great geometric complexity relative to the rastered image, it seems natural to stay within the context of point-based shape representation and directly use the surface points as display primitives.

In this context, numerous visualization approaches for particle-based surface reconstruction rely on scalar field visualization tech-

niques. Using different kinds of overlapping basis functions, a scalar field representation of the fluid volume is computed, and subsequently used for direct volume rendering or isosurface extraction. A particularly interesting approach for SPH settings was presented by Yu and Turk [YT13], who have used each particle's neighborhood structure to compute anisotropic basis functions, thereby capturing local particle distributions more accurately and enabling smoother surfaces with distinct features. While the anisotropic surface definition itself is sound and promising, Yu and Turk have only used Marching Cubes for discrete triangulated surface extraction, thus preventing smooth visualizations which scale transparently with resolution. To achieve a pixel-accurate representation of the surface, the required resolution for the MC grid is prohibitively costly in terms of computational time for most relevant cases.

In this paper, we build upon the rich anisotropic kernel approach, adapt and tune the surface definition to FPM-based fluid simulations, and present a novel direct ray tracing scheme for on-the-fly surface reconstruction. Specifically, after a brief review of relevant prior work (Section 2) and a compendiary introduction to the specifics of the FPM (Section 3), we make the following contributions:

- In Sections 4.1 and 4.2, we present an improved anisotropic kernel-based surface definition that specifically targets FPM simulations, incorporates automatic kernel scaling for variable smoothing lengths and intuitive visuals for isolated particles, and is easily parallelized.
- For this surface definition, we describe a novel direct ray tracing scheme definition (Section 4.3). This on-demand two-pass iterative sampling algorithm intelligently reduces intersection candidates for both opaque and transparent surface rendering, provides optimization opportunities for secondary rays, and allows the dynamic mapping of particle attribute values on to the surface using arbitrary transfer functions. Details of our implementation within the OSPRay raytracer are outlined in section 5.
- We conduct and analyze comprehensive benchmarks to quantify preprocessing and rendering times on different state-of-the-art hardware setups, including workstation, standard cluster and Xeon Phi accelerator systems, and demonstrate the applicability of our approach to a variety of medium and large scale FPM data sets (Section 6).

2. Related Work

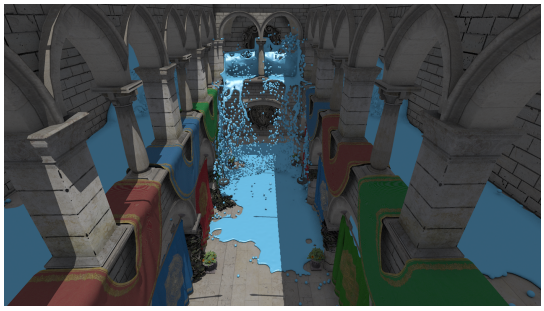
The reconstruction, tracking and visualization of fluid surfaces has been an object of research since the advance of fluid simulation and computational fluid dynamics. In the context of mesh-based simulation, various techniques for surface extraction have been developed such as level-set methods [OF03], particle level-set methods [ELF05], semi-Lagrangian contouring [BGOS06], volume of fluid methods [HN81] and explicit surface tracking [MÖ9].

However, especially for transient flow and continuum mechanical problems, particle-based simulation techniques such as SPH or FPM are more versatile than their grid-based counterparts. Different approaches have been investigated to reconstruct fluid surfaces directly from their point-based representations, such as splatting in combination with image-space curvature flow reduction [vdLGS09], collecting contributing particles using cylindrical rays [SJ00], globally fitting smooth interpolants based on radial basis functions [TO02], or point-set surfaces based on local moving least squares fits [ABCO*01], which can also be used for adaptive advancing front surface triangulation [SFS05]. While point-set surfaces work well for densely sampled surface representations, e.g., from laser scans, they fail in turbulent and noisy scenarios with increasing counts of isolated particles. High quality volume rendering of particle data has been studied by Fraedrich et al. [FAW10] and Hochstetter et al. [HOK16]. Goswami et al. [GSSP10] present a voxel-based rendering pipeline on the GPU which constructs a partial distance field for subsequent ray casting. Reichl et al. [RCSW14] use binary voxel hashing to accelerate ray casting of point-based fluids on the GPU.

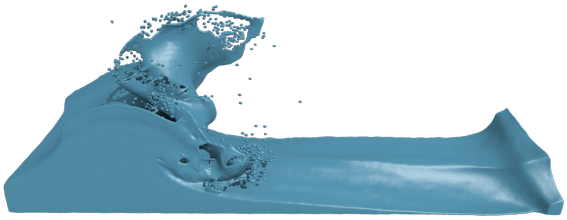
We follow the well elaborated approach of defining the fluid surface as an isosurface of a scalar field constructed from overlapping basis functions. The use of simple isotropic basis functions dates back to Blinn's metaballs [Bli82], which typically result in blobby surfaces. Zhu and Bridson [ZB05] extend this idea to compensate for local particle density variations to create considerably smoother surfaces. Adams et al. [APKG07] track the particle-to-surface distance over time to create smooth surfaces for both fixed-radius and adaptively sized particles. Müeller et al. [MCG03] introduced the idea of creating a normalized scalar field based on the density as estimated by SPH, which we also follow. Solenthaler et al. [SSP07] propose a surface reconstruction technique based on considering the movement of the center of mass to reduce rendering errors in concave regions. Premžoe et al. [PTB*03] use isotropic kernels with interpolation weights stretched along the velocity field.

Owen et al. [OVSM98] inspired the use of anisotropic smoothing kernels, which were later combined by Ding et al. [DTS01] with variational implicit surfaces. Kalaiah and Varshney [KV03] have applied principal component analysis (PCA) to extract anisotropy from point clouds for point-based modeling, whereas Liu et al. [LLL06] have used anisotropic smoothing kernels for material deformation accuracy. Yu and Turk [YT13] have built upon these previous works and extracted a surface from the resulting normalized scalar field using the marching cubes algorithm [LC87]. Ando et al. [ATT12] employed this to determine and visualize thin fluid sheets in fluid simulations, while Macklin and Müller [MM13] combined it with the splatting approach of van der Laan et al. [vdLGS09] for their iterative density solver to achieve real-time fluid simulation. Akinci et al. [AIAT12] parallelized marching cubes-based surface extraction by considering only grid nodes in a narrow band around the surface. Yu et al. [YWTY12] used the anisotropic kernel method to construct an initial explicit triangle mesh, which is advected over time to track the air/fluid interface.

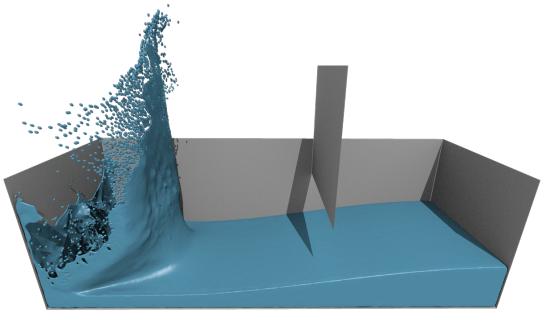
We improve on Yu and Turk's surface definition by presenting a novel direct raytracing scheme for anisotropic smoothing kernels in combination with a modified preprocessing procedure for FPM simulations. We also incorporate a variant of velocity-based stretching of isolated particles, which was shown by Bhatacharya et al. [BGB11] for level-set surface approximation minimizing thin-plate energy.



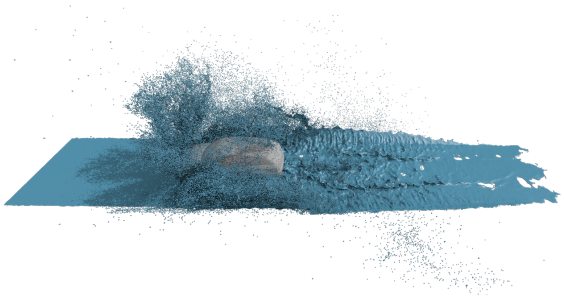
(a) *Sponza*



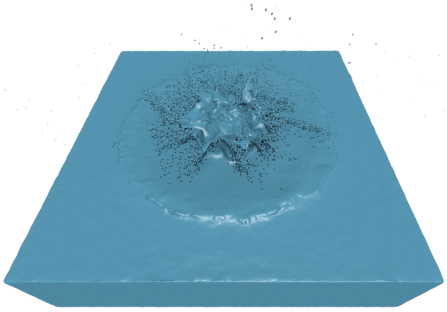
(b) *Sloshing*



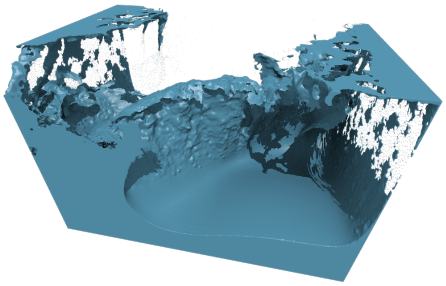
(c) *Dam Break*



(d) *Water Crossing*



(e) *Droplet*



(f) *Double Dam Break*

Figure 2: *Benchmark scenes rendered with ambient occlusion and shadows.*

3. Finite Pointset Method

All simulations for this paper have been performed using MESHFREE, a CFD software developed by Fraunhofer ITWM. MESHFREE uses the *Finite Pointset Method* (FPM, [HJKT05, TAH*07]) to solve the Navier-Stokes equations on a point cloud. As the equations are solved in the Lagrangian formulation, particles move with the current local velocity in every timestep. In contrast to SPH techniques [LL03], particles are only numerical points and carry no mass, allowing to continuously adapt the point cloud by filling and deleting points. Furthermore, this also permits local refinement of the point cloud in areas of interest. Based on a generalized finite difference scheme, the FPM in contrast to SPH supports physical boundary and initial conditions, as well as many well known material models like Darcy, Johnson-Cook, and Drucker-Prager.

Typical real-world simulations discretize using less than 500 000 particles, as there is always a trade-off between computation time, accuracy and available resources. Lower particle counts are preferable as the time step size decreases with a finer resolution due to numerical requirements. For this paper we pushed the number of particles far beyond this to prove that our visualization method will be future-proof.

The visualization requires, per particle, information about position, velocity, smoothing length and kind of boundary. The smoothing length h controls the density of the point cloud; a distance of about $0.4 \cdot h$ among particles is ideal. The velocity information is used to deform isolated particles according to their direction of movement. Kind of boundary assigns each particle their type of boundary, which is divided into inner points, isolated points, wall points and free surface points. Isolated points are determined by the number of their neighbors within their smoothing length, wall points are permanently assigned by the FPM and the detection of

free surface points is based on a local Delaunay tetrahedralization. Our visualization treats both wall and free surface points in the same way as boundary particles.

4. Surface Reconstruction

The proposed surface reconstruction pipeline consists of two steps: preprocessing and rendering. The former includes several computationally intensive steps such as smoothing of particle positions or computing anisotropy information based on local neighborhood structures. The resulting anisotropic kernel representation is used in the subsequent surface visualization via direct raytracing.

In Section 4.1 we will briefly recapitulate the mathematical foundation of the surface definition based on anisotropic kernels, with concrete algorithmic details and optimizations being presented in the following Sections 4.2 and 4.3.

4.1. Surface Definition

Our surface definition is based on the approach proposed by Yu and Turk [YT13], where a scalar field is constructed by overlapping anisotropic smoothing kernels representing the neighborhood structure of each particle. In contrast to previous isotropic approaches, these anisotropic kernels capture local particle distributions more accurately, enabling smooth surfaces, thin streams and sharp features in the reconstruction. We follow the original notation and mark contributed adaptions, optimizations and extensions accordingly.

The surface is defined as an isovalue of the normalized scalar field

$$\phi(\mathbf{x}) = \sum_i \frac{1}{\rho_i} W(\mathbf{x} - \tilde{\mathbf{x}}_i, \mathbf{G}_i), \tag{1}$$

where ρ_i is the sum of the weighted contributions of nearby particles

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

and $\mathbf{W}$ is an anisotropic smoothing kernel of the form

$$W(\mathbf{r}, \mathbf{G}) = det(\mathbf{G}) P(\|\mathbf{G}\mathbf{r}\|). \tag{3}$$

In the preceding equations, $\tilde{\mathbf{x}}_\mathbf{i}$ is a smoothed particle position, $\mathbf{G}_i$ is a $3x3$ linear transformation matrix and P is a symmetric decaying spline with finite support. The linear transformation $\mathbf{G}$ rotates and stretches the radial vector $\mathbf{r}$ to normalized isotropic kernel space, making $W(\mathbf{r}, \mathbf{G})$ an anisotropic kernel with iso-surfaces of ellipsoidal form.

The scalar field $\phi(\mathbf{x})$ is designed as a normalized density field smoothing out the scalar value of 1 at each particle's position over a continuous domain. Thus, an isosurface of $\phi(\mathbf{x})$ gives a surface representation encompassing the particles. We use a surface treshold of 0.2 for all of our use cases.

Note that the original surface definition by Yu and Turk [YT13] was designed for a SPH context and additionally included the mass of each particle. However, since FPM works with massless particles as transient nodes for computation these terms are not required in our definition.

4.2. Preprocessing

For each simulation frame, a dedicated preprocessing step is performed to compute the necessary data for the interactive surface renderer such as per-particle anisotropy information. The complete preprocessing procedure consists of several operations which will be discussed in the following.

Build search structure. We use hash grids for fixed-radius nearest neighbor searches, which are required in several pipeline steps. In this first step, we construct a hash grid over the complete set of fluid particles. Since nearest neighbor searches are the computationally most expensive operation during preprocessing, we try to cache and reuse previous search results as often as possible. Note that since FPM does allow a variable smoothing length (in contrast to SPH), we use the average smoothing length for the bucket size of the hash grid. Thus, the grid implementation needs to support search radii potentially larger than the grid size.

Determine thick boundary. In contrast to Yu and Turk [YT13], we do not use the complete particle set for kernel-based surface evaluation, but consider only particles in a given vicinity of the surface boundary, which we call the *thick* boundary. For this we use the MESHFREE-provided classification of free surface particles. However, if not already available, one could alternatively classify boundary particles based on their neighborhood count. For each boundary particle, we mark all particles within a radius r_b as thick boundary particles. r_b should be chosen as small as possible to reduce the number of candidate ellipsoids which are traversed during sampling, but at the same time large enough such that the surface is sufficiently represented and inner spheres do not intersect the outer surface. To this end, we empirically choose $r_b = 0.8 \cdot h_i$, where h_i is the smoothing length of the i-th particle.

The classification into thick boundary and inner particles is crucial to the subsequent preprocessing and the visualization. While inner particles are directly used by the renderer as a means for fast inner fluid traversal, only thick boundary particles are processed in the remaining preprocessing pipeline. After classification, particle data is regrouped such that inner and thick boundary particles reside in consecutive ranges, and search structure indexing is updated accordingly.

Update search structure. After data restructuring, the neighborhood hash grid is rebuilt to reflect the new particle indexing. In addition to the full particle set hash grid, we also construct a smaller additional hash grid only containing the thick boundary particles to speed up the subsequent computation of connected components.

Compute connected components. In order to alleviate attraction effects between approaching fluid components, a connected component analysis is performed on all particles in the thick boundary. Two particles i and j are defined as being connected if $\|\mathbf{x}_i - \mathbf{x}_j\| \leq r_{cc}$, where $r_{cc} = 0.45 \cdot h_i$. We identify this value since the dynamic point management algorithm of MESHFREE typically results in particles with distance $0.4 \cdot h_i$ to each other. We use a straightforward union-find algorithm to compute connected components and obtain a component id for each thick boundary particle. We only need to consider thick boundary particles, since components which are connected through inner particles necessarily are connected through boundary particles. Thus, when checking

for connected component equality, inner particles are always considered valid.

Smooth particle positions. To improve the visual quality of flat surfaces, a single iteration of Laplacian smoothing is applied to the thick boundary particle positions. The updated particle positions $\bar{\mathbf{x}}_\mathbf{i}$ are computed via

$$\bar{\mathbf{x}}_\mathbf{i} = (1-\lambda)\mathbf{x}_\mathbf{i} + \lambda \sum_j w_{ij}\mathbf{x}_j / \sum_j w_{ij}, \tag{4}$$

where $\lambda \in [0,1]$ is constant describing the degree of smoothing and w_{ij} is a weighting function with finite support. Note that the updated smoothed particle positions are only used by the visualization pipeline and do not affect the underlying simulation. We typically use a value of 0.9 for λ to apply a strong smoothing effect. However, we find that conglomerates of isolated particles tend to be smoothed into a single particle, which can lead to visual artifacts. Thus, we set λ to 0.1 for particles with less than 20 neighbors.

The particle smoothing adheres to the previously established connected component labeling, i.e., for each particle only neighbors which belong to the same connected component as the particle itself are considered. This is reflected in the following weighting function

$$w_{ij} = \begin{cases} 1 - ((\|\mathbf{x}_i - \mathbf{x}_j\|)/r_s)^3 & \text{if } \|\mathbf{x}_i - \mathbf{x}_j\| < r_s \text{ and } c_i = c_j \\ 0 & \text{otherwise} \end{cases}, \tag{5}$$

where c_i denotes the connected component of the i-th particle and r_s is the search radius used for nearest neighbor searches during particle smoothing; we use $r_s = r_b + r_{cc} = 1.25 \cdot h_i$, since inner particles are always considered part of each connected component. Thus, when a particle approaches a surface belonging to another connected component, the inner particles behind that respective thick boundary are not considered for smoothing until the two connected components are close enough to merge.

We cache neighborhood information determined during particle smoothing in order to reuse them in the subsequent computation of anisotropic smoothing kernels and normalization densities. Note that this results in an approximation since particle positions have been smoothed after neighborhood querying. However, our experiments show that reusing unsmoothed neighborhood information as an approximation has only negligible visual impact.

Determine kernel scaling factors. This step is a preliminary optimization for the subsequent computation of the actual anisotropic kernels, where a scaling factor is used to keep the volume of W (Equation 3) approximately constant for all particles with full neighborhood. Yu and Turk [YT13] have used an empirically chosen constant for all particles, which is dependent on the data set at hand and furthermore only makes sense for SPH which has a constant smoothing length. In contrast to this, we employ an automatic randomized sampling strategy to derive a polynomial relationship between local smoothing length and an optimal kernel scaling factor for a given particle.

To achieve this, we pick a random subset of inner particles and perform for each particle a simplified variant of the anisotropic kernel computation (the exact formulae will be outlined in the subsequent paragraph). For each particle, a covariance matrix C is constructed based on a local neighborhood query. Since inner particles are expected to have a full isotropic neighborhood, a convenient scaling factor for the particle at hand is computed based on the determinant of the covariance matrix as $k_s^i = \sqrt[3]{det^{-1}(C)}$.

All resulting (h_i, k_s^i) pairs are collected and averaged into a fixed number of buckets (20 in our experiments), and a least squares polynomial fit of degree 4 is computed, which we denote by $k_s(h)$. This polynomial relationship is used in the following anisotropic kernel computation to pick a suitable kernel scaling factor for each particle based on its respective local smoothing length.

Compute anisotropic kernels. The foundation of the anisotropic kernel method is the determination of an anisotropy matrix $\mathbf{G}_i$ for each particle that describes the particle density distribution around it. By applying weighted Principal Component Analysis (WPCA) to the neighborhood of each particle in the thick boundary, the weighted covariance matrix $\mathbf{C}_i$ is constructed as

$$\mathbf{C}_i = \sum_j w_{ij}(\mathbf{x}_j - \mathbf{x}_i^w)(\mathbf{x}_j - \mathbf{x}_i^w)^T / \sum_j w_{ij}, \tag{6}$$

where $\mathbf{x}_i^w$ is the weighted mean defined as

$$\mathbf{x}_i^w = \sum_j w_{ij}\mathbf{x}_j / \sum_j w_{ij}. \tag{7}$$

Note that the weight function w_{ij} (Equation 5) now uses the updated particle positions originating from the preceeding smoothing operation and still considers connected components. We reuse the previously cached neighborhood information from the smoothing step, but only consider neighbors within the search radius $r_s = h_i$.

After the weighted covariance matrix has been constructed, a Singular Value Decomposition (SVD) is applied, yielding the principal vectors of deformation in the particle set considered. Assuming the SVD is denoted by $\mathbf{C}_i = \mathbf{R}\Sigma\mathbf{R}^T$, where $\mathbf{R}$ is a rotation matrix with principal axes as column vectors and $\Sigma = diag(\sigma_1, \sigma_2, \sigma_3)$ a diagonal matrix with eigenvalues $\sigma_1 \geq \sigma_2 \geq \sigma_3$, extreme deformations are prevented by restricting the proportions between largest and smaller eigenvalues, i.e., $\tilde{\sigma}_{2,3} = max(\sigma_{2,3}, \sigma_1/k_r)$, where k_r is a scaling factor denoting the maximum ratio between largest and smallest axis of the resulting ellipsoid. For our experiments we set $k_r = 4$.

Since the desired transformation matrix $\mathbf{G}_i$ is an inversion of the modified covariance matrix, its computation can be expressed as

$$\mathbf{G}_i = \frac{1}{h_i}\,\mathbf{R}\tilde{\Sigma}^{-1}\mathbf{R}^T, \tag{8}$$

where

$$\tilde{\Sigma}^{-1} = \frac{1}{k_s(h_i)}\, diag(\frac{1}{\sigma_1}, \frac{1}{\tilde{\sigma}_2}, \frac{1}{\tilde{\sigma}_3}). \tag{9}$$

In contrast to Yu and Turk [YT13], we do not represent isolated particles with insufficient neighbors as simple spherical kernels, but also transform those along each particle's velocity vector $\mathbf{v}_i$ as given by the simulation. From our experiments this approach leads to much more intuitive visuals for fast moving isolated particles such as splashing water droplets. Specifically, if a particle has less than 20 neighbors, the normalized velocity $v_n = \|\mathbf{v}_i\|/h_i$ is used to determine the major transformation strength as $m_a =$

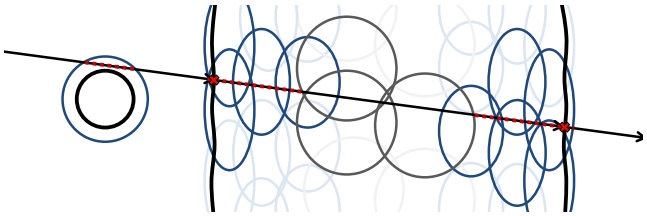


Figure 3: *Intersection scheme: anisotropic smoothing kernels (blue), inner spheres (gray), sampling locations (red dots) and actual surface (black). The smoothing kernel of a candidate isolated particle in front is intersected and sampled, but the surface threshold is not reached until the surface of the fluid bulk is intersected. In transparent rendering, a secondary ray is started, which utilizes the inner spheres to reduce the number of required samples until the exit intersection is reached.*

$1 + d \cdot min(v_n, v_{max})/v_{max}$, where v_{max} is the maximum normalized velocity and d is the maximum degree of deformation. For our experiments we set $v_{max} = 50$ and $d = 0.3$. In order to preserve volume during transformation, the corresponding orthogonal minor transformation strength equals to $m_b = \sqrt{1/m_a}$. The final transformation matrix $\mathbf{G}_i$ is then

$$\mathbf{G}_i = \frac{1}{h_i}\, \mathbf{R}(\mathbf{e}_x, \mathbf{v}_i)\, \mathbf{S}^{-1}, \tag{10}$$

where $\mathbf{R}(\mathbf{e}_x, \mathbf{v}_i)$ is a matrix that rotates the x-axis $\mathbf{e}_x$ onto $\mathbf{v}_i$, $\mathbf{S} = k_n\ diag(m_a, m_b, m_b)$ is a scaling matrix and k_n is a size factor for isolated particles. We use $k_n = 0.35$ to prevent isolated particles from looking too bold.

Compute ellipsoid bounding boxes. Once all anisotropic transformations $\mathbf{G}_i$ have been computed, tight axis-aligned bounding boxes are constructed for each ellipsoid of influence as these are needed by the BVH acceleration structure used in the renderer for fast intersection candidate retrieval.

Using the standard derivation for tight bounding boxes around ellipsoids using projective geometry, the desired axis-aligned bounds can be computed as

$$\begin{aligned} x &= p_x \pm \sqrt{(\mathbf{G}^{-1}_{i,11})^2 + (\mathbf{G}^{-1}_{i,12})^2 + (\mathbf{G}^{-1}_{i,13})^2} \\ y &= p_y \pm \sqrt{(\mathbf{G}^{-1}_{i,21})^2 + (\mathbf{G}^{-1}_{i,22})^2 + (\mathbf{G}^{-1}_{i,23})^2}, \\ z &= p_z \pm \sqrt{(\mathbf{G}^{-1}_{i,31})^2 + (\mathbf{G}^{-1}_{i,32})^2 + (\mathbf{G}^{-1}_{i,33})^2} \end{aligned} \tag{11}$$

where $\mathbf{p}_i = (p_x, p_y, p_z)$ is the particle's center.

Compute weighted contributions per particle. As last preprocessing step, the sum ρ_i of weighted contributions of nearby particles as outlined in Equation 2 is computed for each particle based on the previously constructed anisotropic kernels. For the symmetric decaying spline P in Equation 3 we make use of the reversed *smootherstep* function defined as $P(x) = 1 - (6x^5 - 15x^4 + 10x^3)$ for $x \in [0, 1]$. Also, we precompute the combined coefficient value of $det(\mathbf{G}_i)/\rho_i$ for each particle which is needed in the renderer for surface sampling as illustrated in Equation 1.

After preprocessing has finished, only the data relevant for our direct ray-based rendering technique is kept in memory. For each particle in the thick boundary this boils down to position $\mathbf{p}_i$, transformation matrix $\mathbf{G}_i$, bounding box, coefficient $det(\mathbf{G}_i)/\rho_i$ and optionally a user-selected attribute value which is used for color mapping onto the surface in conjunction with a given transfer function. For inner particles only the position $\mathbf{p}_i$ and a radius $r_i = s \cdot h_i$ is stored, where s is a scaling factor that should be chosen sufficiently large such that the resulting inner spheres overlap completely with themselves and the thick boundary, i.e., there are no holes, however at the same time as small as possible to improve acceleration structure efficiency during traversal. We choose $s = 0.5$ in our approach.

4.3. Intersection

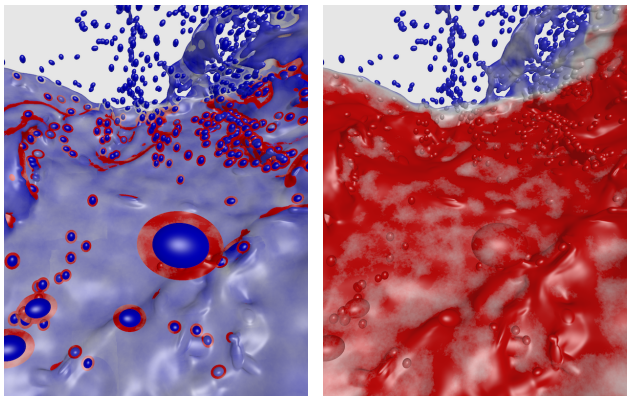
Contrary to previous work based on isosurface extraction via marching cubes, we perform a direct raycasting of the scalar field formed from the preprocessed anisotropic (ellipsoidal) smoothing kernels.

In order to determine ray-surface intersection position, it is necessary to sample and test the scalar field along the ray. Since the field is defined at any point as the sum of contributing kernel values, multiple overlapping kernels may be needed to reach the surface value. On the other hand, intersecting a single arbitrary kernel does not guarantee that the surface is hit. Figure 3 shows a typical scenario, where a fluid volume defined by multiple overlapping anisotropic kernels form in the thick boundary is partially occluded by an isolated particle in front. Additionally, we use spherical inner particles to overlap the complete fluid volume in order to reduce the number of sampling locations and perform fast traversal of inside segments.

While a ray may encounter an isolated particle as a candidates for intersection, the surface threshold is not surpassed during sampling. The first actual surface hit is encountered at the surface of the fluid bulk. If the surface is completely opaque, then rendering for this particular ray stops here. However, e.g., in the case of transparent rendering, a new ray may be started an epsilon behind the former hit point, which continues sampling through the set of contributing anisotropic kernels. It is crucial to keep the number of required samples to a minimum and efficiently skip ray intervals which are completely inside the fluid volume.

To collect contributing kernels for each ray, an all-hit intersection test is performed, computing entrance and exit positions for all candidate anisotropic kernels along the ray. Candidates are provided by the underlying acceleration structure based on their axis-aligned bounding boxes. The resulting events are inserted in an ordered list, which in our case has proven to be faster than saving them unordered and sorting them afterwards, as studied in [AGGW15]. If no kernels were hit, there can be no surface intersection.

However, gathering all intersected ellipsoids along the complete ray can be quite costly and is often not even necessary, e.g., for opaque surfaces, since the essential contributing kernels are located in close vicinity to the frontmost ellipsoid hit. Motivated by this observation, we perform a two-pass approach for opaque surface rendering, where in a first step only kernels close to the first hit are gathered and checked for intersection. Since typical acceleration structures do not guarantee strict sorted ordering of query re-



(a) *Offset culling* **(b)** *Naïve all-hit*

Figure 4: *Total number of ellipsoids collected per ray, from zero (blue) to 200 (red). Using offset culling dramatically reduces the number of smoothing kernels considered for surface sampling. Red halos in Figure 4a indicate areas where the frontmost kernels are hit, but the surface threshold is not reached and offset culling terminates. Thus, an all-hit intersection is performed to reach the real surface intersection.*

sults for performance reasons, we guide the acceleration structure to converge to the frontmost candidates as fast as possible.

To achieve this, we introduced an optimization we call *offset culling*: whenever a candidate ellipsoid is evaluated, the end of the ray is clamped to the respective entry intersection point plus a predefined offset. The offset must be chosen large enough, such that in any case all kernels contributing to the surface are returned by the search structure, even when the first candidate encountered is the frontmost ellipsoid. We use $0.5 \cdot h_{avg}$, where h_{avg} denotes the average smoothing length of the whole data set. Offset culling yields a significant performance improvement over a naïve all-hit query, since the number of candidate kernels potentially contributing to the surface is dramatically reduced, as can be seen in Figure 4. This leads to another improvement as we do not need to sort ellipsoids during gathering, but rather simply store them in any order. If the surface was not hit during the first pass, the ray is cast again without offset culling to gather all kernels as described above. Offset culling is only performed for opaque surfaces, where rays always start outside of the volume and stop at the first surface intersection. When rendering with transparency, large numbers of rays start after the first surface intersection and pass through the inner fluid, where next surface interaction does not lie in vicinity of the frontmost ellipsoids encountered. Listing 1 shows the handling of each candidate ellipsoid during the all-hit phase.

In addition to the anisotropic kernels for particles in the thick boundary, we make use of the remaining inner particles of spherical form for fast traversal of inner ray segments. Each sphere along the ray marks an interval that is always inside the volume. Therefore, there can be no intersection with the surface during this interval and it is safe to skip sampling on this segment of the ray. Similiar to the gathering of candidate ellipsoids, we perform an all-hit intersection in the local spheres scene. To reduce the number of intervals to be checked during sampling, intervals are merged with overlapping ones in a sorted list of intervals as they are detected, as outlined in Listing 2.

Having collected all potentially contributing anisotropic smoothing kernels and having constructed the minimal list of inner intervals, uniform sampling is performed along the ray between all recorded events. For our experiments we employ $0.1 \cdot h_{avg}$ as sampling step size. At each sampling position, we first check if an inside segment can be skipped. Then, the list of currently contributing anisotropic kernels is updated. If no ellipsoids are actually contributing at the sample position, we jump to the next ellipsoid's entry event. Otherwise, the surface's scalar field value is evaluated over the sum of contributing kernels. If the computed value

```
compute intersections of ray with ellipsoid
if (no intersection)
    return
if (offset culling)
    if (ellipsoid behind end of ray)
        return
    set end of ray to entry + offset
    store ellipsoid in unsorted array
else
    construct events for entry and exit
    insert events in sorted array
```

Listing 1: *Per-ellipsoid callback for all-hit intersection.*

```
compute intersections of ray with sphere
if (no intersection)
    return
check existing intervals for overlap
if (no overlap)
    insert new interval in sorted list
else
    merge overlapping intervals with new interval
```

Listing 2: *Per-sphere callback for all-hit intersection.*

```
// 1st pass: offset culling
if (offsetCulling)
    all-hit intersect ellipsoids (with culling)
    if (no intersection)
        return
    sample from first hit to offset:
        sum up scalar field over contributing kernels
        if (surface value passed)
            determine exact hit (value-weighted bisection)
            interpolate normal
            if (color mapping)
                interpolate attribute
            return
        go to next sampling position

// 2nd pass: full traversal
all-hit intersect ellipsoids (no culling)
if (no intersection)
    return
all-hit intersect spheres (construct inner intervals)
sample from first event to last event:
    if (sample point inside inner interval)
        jump to end of interval
    update list of contributing ellipsoids
    if (no contributing ellipsoids)
        jump to next event
        continue
    sum up scalar field over contributing kernels
    if (surface value passed)
        determine exact hit (value-weighted bisection)
        interpolate normal
        if (color mapping)
            interpolate attribute
        return
    go to next sampling position
```

Listing 3: *Two-pass surface intersection scheme.*

passes the defined surface value, the surface was hit between the current and the last sample position. We then perform recursive sub-sampling using value-weighted bisection to refine the intersection point on the given interval. With only a few iterations this approximates the surface position up to single floating point precision in our cases. The surface normal is interpolated from the weighted contributions of the individual kernel normals. The complete surface intersection scheme is outlined in Listing 3.

For secondary rays such as shadow rays or ambient occlusion there is no need for precise hit point sampling. In order to accelerate occlusion tests, we also perform a two-pass approach here. First, only the inner spheres are intersected for occlusion, terminating upon any sphere hit. If there is no occlusion from spheres, the actual surface intersection algorithm is performed as outlined above in a simplified form. In this case, we do not perform recursive subsampling, and we do not compute interpolated normals and attributes.

5. Implementation

We extensively use local OpenMP parallelization in conjunction with MPI distribution across nodes throughout our preprocessing pipeline, in which nearest neighbor searches are the dominant computational effort. Our experiments have shown that in general dynamic scheduling for local parallelization is superior to static scheduling due to the slightly imbalanced workload depending on each particle's neighborhood. For the multi-threaded construction of the hash grids we use the concurrent *libcuckoo* hash map [LAKF14] as back-end. We also investigated search structures based on kD-trees, which exhibited slightly slower performance than the hash grid-based approach for our use cases.

Since the input data sets are relatively small in size even for practically large point clouds, we begin with the complete particle set on each node. All operations on the thick boundary are distributed across nodes, i.e., each node performs processing on only a subset of the particles. After each preproccesing step, state is exchanged through MPI messages. The partial thick boundary markings are all-reduced using a logical or-operator. Partial connected component labelings are exchanged and merged to a global labeling on each node using the same union-find approach as locally. Smoothed particle positions and anisotropic kernels are simply all-gathered on each node. Since the kernel scaling factors are based on random sampling with low performance impact, the scaling curve is computed on a single node only and broadcasted to the others. Eventually, at the end of preprocessing, the partially computed densities are gathered at the master node which then has the fully preprocessed data and sets up the visualization.

The intersection algorithm was implemented as a user geometry in the OSPRay framework [WJA*17]. Internally, the all-hit kernels to collect thick boundary ellipsoids and inner spheres make use of two additional manual Embree scenes, which can be queried for intersection and occlusion independently and are based on the all-hit kernel studies in [AGGW15]. The complete visualization code is automatically vectorized by the ISPC compiler, which compiles a C-based SPMD programming language to run on the SIMD units of CPUs and the Intel Xeon Phi architecture, without the need for tedious writing of manual SSE/AVX intrinsics.

We use OSPRay's scientific visualization renderer and for distributed rendering the MPI offloading device, which replicates the scene model on all worker nodes and performs sort-first compositing at the master node.

6. Results

To investigate the characteristics and potential of our direct raytracing scheme, we conduct comprehensive benchmarks with respect to data set size, scene complexity, visual fidelity and hardware setup. We consider three hardware setups: a desktop workstation with an Intel i7-6700K CPU (4x 4.0 GHz), the *Elwetritsch* cluster using one Intel Xeon E5-2640 v3 (8x 2.6 Ghz) per node with InfiniBand QDR interconnect, and the *Stampede2* supercomputer providing Intel Xeon Phi 7250 Knights Landing accelerator cards (68x 1.4 GHz, 272 hardware threads) with Intel Omni-Path interconnect. We choose three base cases for rendering: opaque rendering, opaque rendering with ambient occlusion (16 samples per hit) and hard shadows, and transparent rendering. All renderings have been performed at standard 4K resolution (3840x2160), except for the low resolution desktop scenario which was performed at 960x540 to represent a downsampled rendering mode for interactive use cases.

Figure 2 shows the different test scenes for our benchmarks, covering a wide spectrum of particle counts and rendered with ambient occlusion and hard shadows. As expected from the anisotropic kernels method, surfaces are smooth while preserving sharp features such as thin shields and isolated particles. The visualization can be considered faithful to the simulation, even if this exposes the dynamic particle management in the form of occasionally popping particles across time steps.

Complete timings for preprocessing and rendering are presented in Table 1. Preprocessing scales roughly linearly with the number of thick boundary particles. Basic opaque rendering times can be considered interactive for reduced resolutions on single nodes, whereas distributed rendering enables interactive frame rates at full 4K resolution. Depending on scene complexity, we observe highly increased rendering workload for ambient occlusion, which is not due to our method per se but generally expected in raytracing contexts. From a hardware perspective, the Stampede2 accelerator cards outperform standard CPU-based machines in rendering due to ample vectorization opportunities in packed raytracing. On the other hand, preprocessing provides less room for vectorization, for instance during hash grid construction or connected component analysis. Distributed rendering requires enough workload in relation to communication overhead in order to scale reasonably well.

To better understand our method's capability to handle large data, we perform an extensive scaling study using the Double Dam Break data set, which was iteratively refined from 1 million particles up to 170 million particles. Thus, we can observe approximately the same scene complexity at different resolutions. Figure 5 shows two fundamentally different scaling studies on Elwetritsch and Stampede2. Figures 5a and 5c show the highest resolution data set preprocessed and rendered on varying node counts, whereas Figures 5b and 5d present timings for different particle counts on 32 nodes of each cluster. In all cases, rendering was performed with

Data Set Particles (Thick) Preprocessed Size	Time	Desktop (4K)	Desktop (Low Res)	Elwetritsch (1 Node)	Elwetritsch (32 Nodes)	Stampede2 (1 Node)	Stampede2 (32 Nodes)
Sponza 161 000 (97%) 12 MB	Preprocess	1.0	1.0	1.2	0.3	4.3	2.2
	Opaque	1.6	0.1	1.9	0.1	0.4	0.6
	AO + Shadows	41.9	2.9	26.4	1.8	5.3	0.6
	Transparent	7.1	0.7	7.8	0.6	2.1	0.6
Sloshing 457 000 (49%) 20 MB	Preprocess	1.9	1.9	1.8	0.5	4.8	1.7
	Opaque	4.9	0.4	5.5	0.4	1.6	0.5
	AO + Shadows	56.2	3.7	56.0	4.3	32.9	2.8
	Transparent	50.5	4.8	51.4	3.3	17.9	1.6
Dam Break 586 000 (49%) 26 MB	Preprocess	2.0	2.0	2.0	0.5	4.8	2.3
	Opaque	3.1	0.3	3.6	0.2	0.9	0.5
	AO + Shadows	166.4	10.5	164.5	10.2	81.2	5.9
	Transparent	28.2	2.7	30.0	1.7	8.7	0.6
Water Crossing 2 153 000 (79%) 136 MB	Preprocess	10.3	10.3	11.9	1.9	11.4	7.1
	Opaque	3.5	0.3	3.8	0.2	1.3	0.6
	AO + Shadows	55.9	3.7	57.4	3.0	29.9	2.4
	Transparent	27.1	3.4	27.9	1.6	11.9	1.5
Droplet 3 765 000 (19%) 103 MB	Preprocess	5.2	5.2	5.0	1.1	6.8	4.6
	Opaque	6.0	0.6	6.7	0.4	1.8	0.7
	AO + Shadows	34.5	2.4	35.4	2.0	17.7	2.1
	Transparent	48.2	4.9	51.7	2.5	13.9	0.7
Double Dam Break 170 000 000 (14%) 4.1 GB	Preprocess	329.9	329.9	230.6	44.9	205.6	102.8
	Opaque	15.0	6.8	13.5	1.0	8.2	2.3
	AO + Shadows	202.9	20.1	199.6	11.3	80.7	5.8
	Transparent	150.2	24.8	147.2	9.4	77.7	14.4

Table 1: *Preprocessing and rendering timings in seconds.*

ambient occlusion and hard shadows to provide enough opportunity for workload distribution during rendering. We highlight that while our core aim was not perfect scalability but rather a feasibility study, our method also performs well in these computationally involved scenarios.

When considering the same full resolution data set on increasing node counts, it becomes obvious that hash grid construction is a purely local operation on each node which does not scale well. Connected component construction scales to some degree, but it still limited by the eventual merge of partial connected components on each node. The increase in rendering time for two nodes is explained by the way we use OSPRay's MPI offloading device, where the master rank has only organizational duties and performs sort-first compositing, while the remaining nodes do the actual rendering. Thus, only one of two ranks is rendering, with additional communication overhead. In contrast to Elwetritsch, the overhead seems to be negligible on Stampede2.

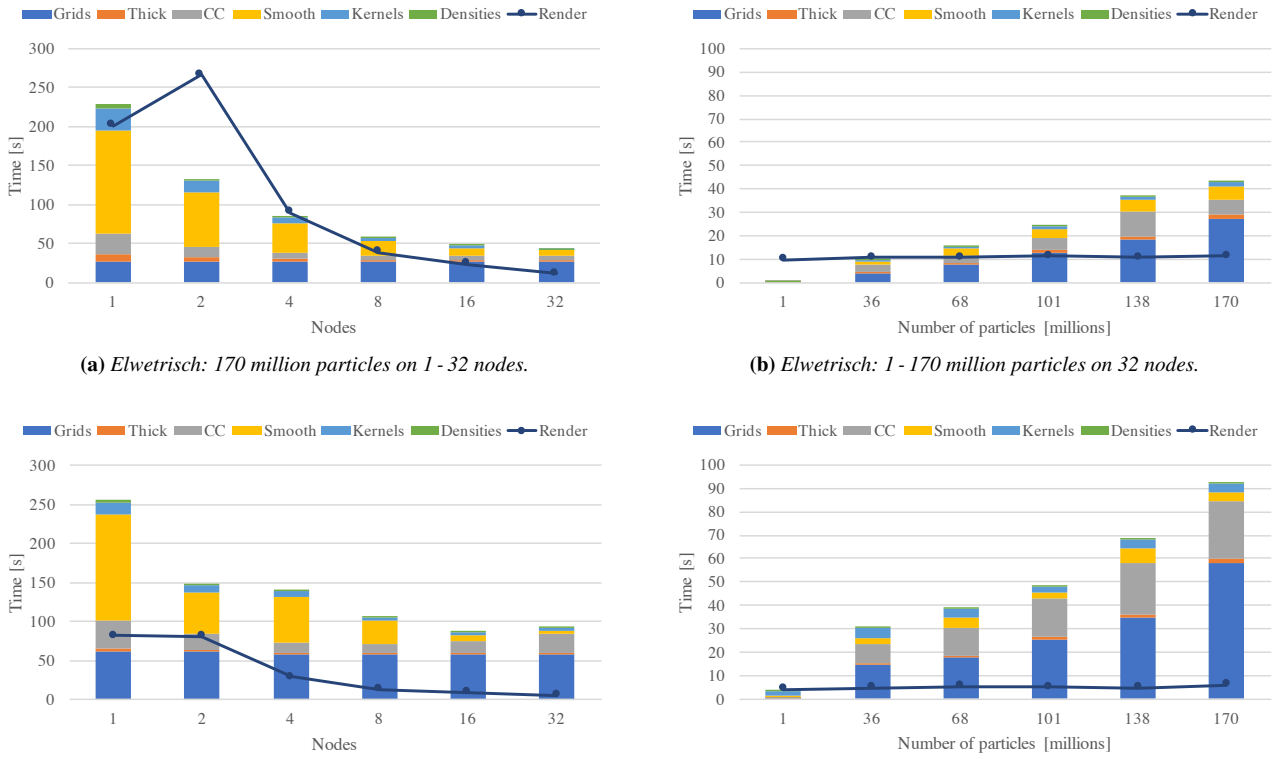
While preprocessing times increase approximately linearly with particle count, rendering appears rather unaffected in these cases. This is expected, as the offset culling optimization in opaque rendering is based on the local smoothing length, resulting in approximately the same magnitude of anisotropic kernels contributing to the surface intersection for each ray. The same behavior can be observed on both Elwetritsch and Stampede2, while the former is in general the better preprocessor and the latter the faster renderer.

We conclude that our direct rendering technique is versatile and suitable for both high fidelity and interactive rendering scenarios. It scales reasonably well even using trivial parallelization, and is thus an option for in-situ use cases by easily enabling preprocessing and rendering on multiple nodes. An additional design aspect of our method is that it runs in image space rather than object space complexity, which is a desirable feature for large scale data applications and is common for raytracing based approaches.

7. Conclusion

We presented a novel approach to the direct visualization of particle-based fluids and have demonstrated its applicability to a wide spectrum of FPM simulations. Our technique is based on an anisotropic kernel model, in which the neighborhood of each particle is used to construct a locally deformed smoothing kernel, allowing smoother surfaces with sharp features. In our raytracing-based rendering scheme we perform optimizations such as offset culling to effectively reduce the number of contributing kernels per surface intersection. We conducted comprehensive benchmarks to study the performance and scaling characteristics of our parallelized and distributed implementation on various hardware configurations and have demonstrated its general versatility.

We anticipate that many improvements to our approach are possible. Since FPM relies on local moving least squares interpolation, it by necessity incorporates nearest neighbor search structures with



(a) *Elwetrisch: 170 million particles on 1 - 32 nodes.*

(b) *Elwetrisch: 1 - 170 million particles on 32 nodes.*

(c) *Stampede2: 170 million particles on 1 - 32 nodes.*

(d) *Stampede2: 1 - 170 million particles on 32 nodes.*

Figure 5: *Scaling on Elwetritsch and Stampede2 using the Double Dam Break data set, rendered at 4K resolution with ambient occlusion and shadows.*

managed ghost particle information in distributed cluster mode, which could be reused in the preprocessing state of our pipeline. Furthermore, we would like to investigate using our system for production visualization, i.e., full path tracing, which is an important use case, e.g., in the automobile industry. Finally, comparing performance of our implementation against a GPU implementation would be interesting to shed light on the relative strength of the differing architectures for our use case. We intend to investigate all these aspects in future work.

References

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. In *Proceedings of the Conference on Visualization '01* (2001), VIS '01. 2

[AGGW15] AMSTUTZ J., GRIBBLE C., GÜNTHER J., WALD I.: An Evaluation of Multi-Hit Ray Traversal in a BVH using Existing First-Hit/Any-Hit Kernels. *Journal of Computer Graphics Techniques (JCGT) 4*, 4 (December 2015), 72–88. 6, 8

[AIAT12] AKINCI G., IHMSEN M., AKINCI N., TESCHNER M.: Parallel surface reconstruction for particle-based fluids. *Comput. Graph. Forum 31*, 6 (Sept. 2012), 1797–1809. 2

[APKG07] ADAMS B., PAULY M., KEISER R., GUIBAS L. J.: Adaptively sampled particle fluids. *ACM Trans. Graph. 26*, 3 (July 2007). 2

[ATT12] ANDO R., THUREY N., TSURUNO R.: Preserving fluid sheets with adaptively sampled anisotropic particles. *IEEE Trans. Visualization and Computer Graphics 18*, 8 (Aug. 2012). 2

[BGB11] BHATACHARYA H., GAO Y., BARGTEIL A.: A level-set method for skinning animated particle data. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2011), SCA '11, pp. 17–24. 2

[BGOS06] BARGTEIL A. W., GOKTEKIN T. G., O'BRIEN J. F., STRAIN J. A.: A semi-lagrangian contouring method for fluid simulation. *ACM Trans. Graph. 25*, 1 (Jan. 2006), 19–38. 2

[Bli82] BLINN J. F.: A generalization of algebraic surface drawing. *ACM Trans. Graph. 1*, 3 (July 1982), 235–256. 2

[DTS01] DINH H. Q., TURK G., SLABAUGH G.: Reconstructing surfaces using anisotropic basis functions. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001* (2001), vol. 2, pp. 606–613 vol.2. 2

[ELF05] ENRIGHT D., LOSASSO F., FEDKIW R.: A fast and accurate semi-lagrangian particle level set method. *Comput. Struct. 83*, 6-7 (Feb. 2005), 479–490. 2

[FAW10] FRAEDRICH R., AUER S., WESTERMANN R.: Efficient high-quality volume rendering of sph data. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (November-December 2010). 2

[GSSP10] GOSWAMI P., SCHLEGEL P., SOLENTHALER B., PAJAROLA R.: Interactive sph simulation and rendering on the gpu. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2010), SCA '10, pp. 55–64. 2

[HJKT05] HIETEL D., JUNK M., KUHNERT J., TIWARI S.: Meshless methods for conservation laws. *Analysis and Numerics for Conservation Laws* (2005), 339–362. 3

[HN81] HIRT C., NICHOLS B.: Volume of fluid (vof) method for the dynamics of free boundaries. *Journal of Computational Physics 39*, 1 (1981), 201 – 225. 2

[HOK16] HOCHSTETTER H., ORTHMANN J., KOLB A.: Adaptive sampling for on-the-fly ray casting of particle-based fluids. In *Proceedings of High Performance Graphics* (2016), HPG '16, pp. 129–138. 2

[KV03] KALAIAH A., VARSHNEY A.: Statistical point geometry. In *Proc. of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (2003), SGP '03, pp. 107–115. 2

[LAKF14] LI X., ANDERSEN D. G., KAMINSKY M., FREEDMAN M. J.: Algorithmic improvements for fast concurrent cuckoo hashing. In *Proceedings of the Ninth European Conference on Computer Systems* (2014), EuroSys '14. 8

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph. 21*, 4 (Aug. 1987), 163–169. 2

[LL03] LIU G. R., LIU M. B.: *Smoothed particle hydrodynamics: a meshfree particle method*. 2003. 3

[LLL06] LIU M. B., LIU G. R., LAM K. Y.: Adaptive smoothed particle hydrodynamics for high strain hydrodynamics with material strength. *Shock Waves 15*, 1 (Mar 2006), 21–29. 2

[MÖ9] MÜLLER M.: Fast and robust tracking of fluid surfaces. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2009), SCA '09. 2

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), SCA '03, pp. 154–159. 2

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

[OF03] OSHER S., FEDKIW R.: *Level Set Methods and Dynamic Implicit Surfaces*. 2003. 2

[OVSM98] OWEN J. M., VILLUMSEN J. V., SHAPIRO P. R., MARTEL H.: Adaptive smoothed particle hydrodynamics: Methodology. ii. *The Astrophysical Journal Supplement Series 116*, 2 (1998), 155. 2

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

[RCSW14] REICHL F., CHAJDAS M. G., SCHNEIDER J., WESTERMANN R.: Interactive rendering of giga-particle fluid simulations. In *Proc. High Performance Graphics* (2014). 2

[SFS05] SCHEIDEGGER C. E., FLEISHMAN S., SILVA C. T.: Triangulating point set surfaces with bounded error. In *Proceedings of the Third Eurographics Symposium on Geometry Processing* (2005), SGP '05. 2

[SJ00] SCHAUFLER G., JENSEN H. W.: Ray tracing point sampled geometry. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000* (2000), pp. 319–328. 2

[SSP07] SOLENTHALER B., SCHLÄFLI J., PAJAROLA R.: A unified particle model for fluid & solid interactions: Research articles. *Comput. Animat. Virtual Worlds 18*, 1 (Feb. 2007). 2

[TAH*07] TIWARI S., ANTONOV S., HIETEL D., KUHNERT J., OLAWSKY F., WEGENER R.: A meshfree method for simulations of interactions between fluids and flexible structures. *Meshfree Methods for Partial Differential Equations III* (2007). 3

[TO02] TURK G., O'BRIEN J. F.: Modelling with implicit surfaces that interpolate. *ACM Trans. Graph. 21*, 4 (Oct. 2002). 2

[vdLGS09] VAN DER LAAN W. J., GREEN S., SAINZ M.: Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (2009), I3D '09, pp. 91–98. 2

[WJA*17] WALD I., JOHNSON G., AMSTUTZ J., BROWNLEE C., KNOLL A., JEFFERS J., GUNTHER J., NAVRATIL P.: OSPRay - A CPU Ray Tracing Framework for Scientific Visualization. *IEEE Transactions on Visualization & Computer Graphics 23*, 1 (2017), 931–940. 8

[YT13] YU J., TURK G.: Reconstructing Surfaces of Particle-based Fluids Using Anisotropic Kernels. *ACM Trans. Graph. 32*, 1 (Feb. 2013), 5:1–5:12. 2, 4, 5

[YWTY12] YU J., WOJTAN C., TURK G., YAP C.: Explicit mesh surfaces for particle based fluids. *Comput. Graph. Forum 31*, 2pt4 (May 2012), 815–824. 2

[ZB05] ZHU Y., BRIDSON R.: Animating sand as a fluid. *ACM Trans. Graph. 24*, 3 (July 2005), 965–972. 2