Volume 0 (1981), Number 0 pp. 1–12 COMPUTER GRAPHICS *forum*

Parallel Surface Reconstruction for Particle-Based Fluids

Gizem Akinci Markus Ihmsen Nadir Akinci Matthias Teschner

University of Freiburg, Germany

Abstract

This paper presents a novel method that improves the efficiency of high-quality surface reconstructions for particle-based fluids using Marching Cubes. By constructing the scalar field only in a narrow band around the surface, the computational complexity and the memory consumption scale with the fluid surface instead of the volume. Furthermore, a parallel implementation of the method is proposed. The presented method works with various scalar field construction approaches. Experiments show that our method reconstructs high-quality surface meshes efficiently even on single-core CPUs. It scales nearly linearly on multi-core CPUs and runs up to fifty times faster on GPUs compared to the original scalar field construction approaches.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Hardware Architecture—Parallel Processing I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Particle-based methods have been gaining increased interest in a broad range of applications, e. g. in movies, commercials and medical simulations. Various fluid phenomena are simulated using particle-based methods, including complex scenerios with more than ten million particles [IABT11]. However, the reconstruction of detailed, smooth, and artifact-free surfaces for large particle sets is generally a bottleneck due to computational complexity and memory requirements.

In recent years, many researchers have worked on the efficient rendering of particle data sets. View-dependent GPU-based methods, e. g. [ALD06, MSD07, FAW10], and view-independent polygonization techniques, e. g. Marching Cubes (MC) [LC87] or Marching Tiles [Wil08], are widely used in this field.

In the context of MC, it is important to note that the employed grid cell size significantly affects the quality of the reconstructed surface and the computation time. In order to obtain smooth surfaces and to catch fine details, small cells are required, which in turn scales the computation time cubically (see Fig. 1).

Another important aspect is the computation of an appropriate scalar field which matches the underlying flow as good as possible together with suppressed surface bumps. Commonly employed smoothing techniques [ZB05, SSP07, APKG07, YT10] consider particles within an influence region in the computation of scalar values at grid points. Similar to the MC cell size, the size of the influence region, i. e. the number of considered particles per query point, significantly affects the quality of the reconstructed surface and also the computation time. Depending on the MC cell size and the size of the influence region, up to 90% of the computation time is spent on constructing the scalar field, whereas the triangulation is rather efficient.

Our contribution. In this paper, we present techniques to improve the performance of high-quality surface generation for particle-based fluids. While many surface reconstruction approaches process all grid nodes (vertices), we propose to improve the efficiency by only considering grid nodes in a narrow band around the surface, without introducing complicated data structures. We also propose a parallelized algorithm for CPUs and GPUs to further improve the efficiency on modern architectures. Our method scales nearly linearly on multi-core CPUs and runs up to fifty times faster on GPU than the original scalar field construction approach. Although narrow band techniques are not new to the literature and there are some sophisticated techniques that were proposed in recent years, e. g. sparse block grids [Bri03], run-length encoding (RLE) methods [HWB04], our approach is especially designed for efficient parallelization on shared

Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

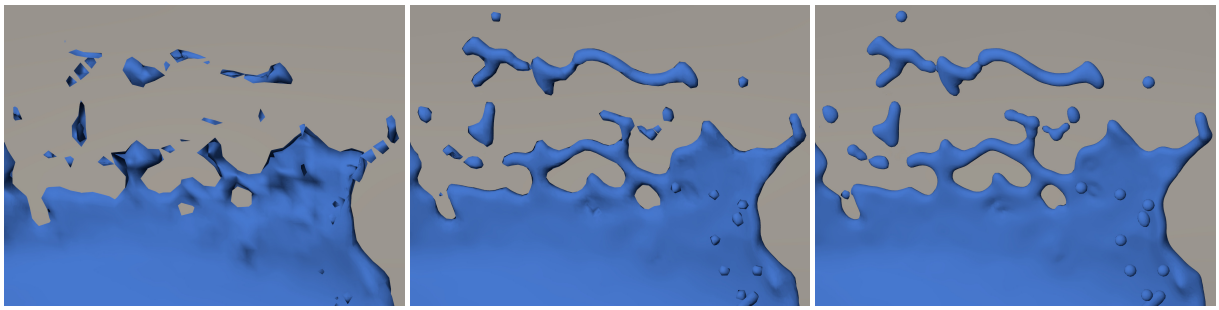

Figure 1: *Close-up view of the reconstructed surface for the corner breaking dam (CBD) scene (1.7 million particles) with three different MC cell sizes: left $2r$, middle r, right $r/2$, with r being the equilibrium distance of the SPH particles. The grid sizes are $123\times107\times123$, $245\times214\times245$, $485\times409\times486$, respectively. The average surface reconstruction time (scalar field computation and triangulation) per frame for the GPU-implementation of the proposed approach is 0.3, 1.18, 15.2 seconds, respectively. While a small cell size produces artifact-free, high-quality surfaces, a larger cell size can be processed very efficiently for, e. g. previewing purposes.*

memory architectures, in contrast to the mentioned methods. Due to its reduced memory requirements and efficient utilization of multi-core architectures, the algorithm can be applied to simulations with large particle sets using small MC cell sizes for generating high-quality surface meshes.

As the smoothed scalar field is only computed in a narrow band, memory consumption and computation time scale with the surface instead of the volume of the simulation domain. Therefore, unnecessary processing and data storage are avoided. In the context of parallelization, inherent data dependencies and race conditions are avoided. Due to the low memory consumption and by enforcing the localism of data, the proposed implementation does not suffer from bandwidth limitations, and scales well on multi-core CPUs and on many-core GPUs.

Our method works with all previously published scalar field construction approaches, e. g. [ZB05, SSP07, APKG07, YT10]. In our experiments, however, [SSP07] is employed as it yields very smooth and artifact-free results, while being comparatively efficient. For the simulations, a variant of smoothed particle hydrodynamics (predictive-corrective incompressible SPH [SP09]) with adaptive time-stepping [IAGT10] is used.

2. Related Work

In this work, we focus on the efficiency of scalar field construction techniques to be incorporated with the commonly used MC method [LC87]. Within the context of these techniques, Blinn proposed one of the earliest approaches by introducing blobbies [Bli82], where the main downside of the approach is that it is not able to generate flat surfaces, especially for particle sets with sharp features. Later, Müller et al. [MCG03] suggested the use of the weighted density information of particles. While this method has improved the quality of classical blobbies, its main issue is still the proneness to bumpiness. Zhu and Bridson proposed to use the signed distance field of the particles [ZB05] to ameliorate the bumpiness issue where each particle within a smoothing radius contributes to the scalar field. This technique achieves smooth surfaces. However, it suffers from artifacts in concave regions and in between splashes. Adams et al. [APKG07] addressed these issues by introducing a distance-based surface tracking technique. This technique is able to capture smooth surfaces better compared to [ZB05]. However, its computational complexity makes it more suitable for frameworks where the distance-to-surface information of particles are computed at each time step, e. g. adaptively sampled particle sets. Solenthaler et al. [SSP07] also improved the method of Zhu and Bridson, where they correct the artifacts on-the-fly by considering the movement of the contributing particles' center of mass at a certain query point. More recently, Yu and Turk [YT10] proposed to use anisotropic kernels. Thereby, the particle kernels are stretched or shrinked along the associated directions of the density distribution in the particle neighborhood. Position smoothing was also suggested to obtain smoother surfaces which causes slight volume shrinkage. The experiments show that the method yields high quality surfaces while being computationally expensive. In contrast to [ZB05, SSP07], the method is highly parameter sensitive and a neighborhood search is required in each reconstruction step for the method itself. Also, stretch/shrink operations are unnecessarily performed for inner particles.

There exist various other approaches that address the efficient representation of fluid surfaces. Witkin and Heckbert [WH94] proposed to use simple constraints in order to track implicit surfaces. Polygonization can be applied for better visual quality, however, it decreases the performance of the method significantly. Explicit surface tracking is a recently used approach [Mül09, BB09] that allows changing the geo-

metric representation and manipulating the regions with thin features. However, small droplets are likely to be missed.

Surface splatting is one of the most popular and widely used surface reconstruction methods, which was first introduced by Zwicker et al. [ZPvBG01]. Adams et al. [ALD06] used splatting for particles, where each particle is projected as a single quad, circle or ellipse to image space using its position and size. Overlapping particles' projections are then blended to obtain a smooth result. More recently, van der Laan et al. [vdLGS09] proposed a splatting method which applies curvature based smoothing and thickness based transparency. The splatting method runs at interactive rates. However, obtaining hole-free results is challenging.

Later within the context of GPU programming, Müller et al. [MSD07] proposed screen space meshes where 2D meshes are constructed for particle sets. The mesh is transformed to 3D world space in order to add shading effects. Fraedrich et al. [FAW10] also proposed a view-dependent GPU rendering method where they discretize the view frustum using an adaptively sampled perspective grid. The particle data is resampled on the grid by using the SPH interpolation technique and poly6 or cubic spline kernel functions. The reconstructed scalar field is finally rendered using ray-casting. Goswami et al. [GSSP10] introduced a distance field volume for surface particles. In this method, the scalar field is reconstructed by simply assigning to each near-to-surface grid vertex the value of the distance between the vertex and the closest particle, without any interpolation. The volume is finally rendered using GPU ray-casting. According to the presented results of [FAW10] and [GSSP10], high performance rates are achieved for high-resolution scenes. However, the demonstrated images show that achieving flat surfaces is still challenging. Similar to our method, [FAW10] and [GSSP10] compute a scalar field over particles. However, the scalar field computation method we prefer [SSP07] is based on the signed distance field method of [ZB05], where an efficient resampling function is used together with a proper surface kernel. The signed distance field method results in an efficient smoothing over the grid, and makes the underlying particle patterns less visible.

In order to scale both the computational amount and the memory consumption with the surface area, different narrow band techniques have been proposed. Müller et al. [MCG03] suggested to visualize the free surface by first identifying the surface cells and later applying MC on the found cells. However, no solution was given for the possible double layer problem and the parallelization was not discussed. Later, more sophisticated methods have been proposed for level set approaches. In [Bri03], Bridson introduced sparse block grids to define the volume over a coarse uniform grid that consists of finer uniform grids in the narrow band region. Houston et al. proposed RLE sparse level sets [HWB04] to encode the regions using run-length encoding with respect to their distance to the narrow band. In [NM06], Nielsen and Museth proposed a new structure, namely dynamic tubular grid (DT-grid), where the narrow band is not constructed on a regular 3d-grid or a tree but without requiring information from outside the narrow band. In terms of efficiency, Nielsen et al. [NNSM07] later proposed a different approach where they handle very high resolutions using out-of-core techniques together with compression strategies. Although these aforementioned structures can reduce the memory footprint efficiently, none of them was designed for parallel architectures and they are not easily adaptable for a parallelization technique due to their sophisticated natures.

To the best of our knowledge, no efficient parallel implementation has been proposed for a particle-based fluid surface reconstruction based on MC. The aim of our paper is to improve the performance of high-quality surface generations for particle-based fluids with an efficient parallelization technique that runs on the narrow band.

3. Scalar Field Estimation

In general, the scalar field computation affects the surface quality and the computation time. While [ZB05] is faster compared to [SSP07] and [YT10], it suffers from artifacts, i. e. spurious blobs, in concave regions. [YT10] yields very smooth surfaces, but is rather expensive to compute, as a neighborhood search has to be performed at each reconstruction step. Besides, we observed that spurious blobs might occur between splashes and in concave regions similar to [ZB05]. Therefore, we performed our experiments with [SSP07], as the approach reconstructs artifact-free and smooth surfaces with reasonable efficiency.

Starting with [ZB05], Solenthaler et al. argue that the mentioned artifacts occur in concave regions if the average position $\bar{\mathbf{x}}$ of neighboring particles around a query point $\mathbf{x}$ changes considerably faster than $\mathbf{x}$. Therefore, they check the largest eigenvalue EV_{max} of $\nabla_{\mathbf{x}}(\bar{\mathbf{x}})$ to detect fast movements. Hence, the isosurface of the scalar field around the query point $\mathbf{x}$ is defined as

$$\phi(\mathbf{x}) = |\mathbf{x} - \bar{\mathbf{x}}| - \bar{r}f \tag{1}$$

where $\phi(\mathbf{x}) = 0$ defines the on-surface points with $\bar{r}$ being the weighted average particle radius within the influence radius of $\mathbf{x}$ and r denoting the equilibrium distance of the SPH particles. In our examples, $\bar{r}$ is equal to r. The factor f is computed as

$$f = \begin{cases} 1 & EV_{max} < t_{low} \\ \gamma^3 - 3\gamma^2 + 3\gamma & otherwise \end{cases} \tag{2}$$

with

$$\gamma = \frac{t_{high} - EV_{max}}{t_{high} - t_{low}} \tag{3}$$

and user-defined threshold values $t_{high} = 3.5$ and $t_{low} = 0.4$.

In [SSP07], $t_{high} = 2.0$ is proposed. However, in our experiments, $t_{high} = 3.5$ yields smoother results. Further, $\bar{\mathbf{x}}$ is computed as

$$\bar{\mathbf{x}} = \frac{\sum_j \mathbf{x}_j k(|\mathbf{x} - \mathbf{x}_j|/R)}{\sum_j k(\mathbf{x} - \mathbf{x}_j|/R)}, \tag{4}$$

with R being the influence radius specifically used in the surface reconstruction, j being the contributing particles that reside within distance R and k being the kernel function which is defined as

$$k(s) = \max(0, (1 - s^2)^3). \tag{5}$$

It can be observed that the quality of the surface is significantly influenced by R. As stated in [ZB05], radii smaller than $4r$ result in bumpy/non-smooth surfaces. I. e., in order to gain smooth and detailed surfaces, a relatively large neighborhood has to be considered. According to our experiments, however, larger influence radii increase the computation time for the surface reconstruction significantly, as the number of particles to be traversed for each query point increases.

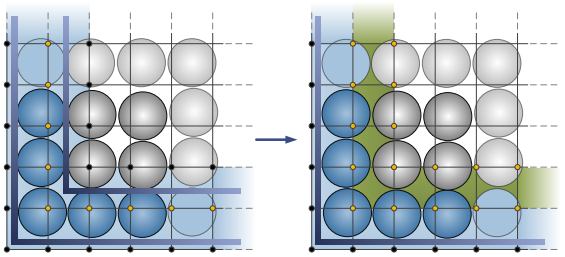

Figure 2: *The double layer problem. Surface vertices are shown by small dots (yellow ones with $\phi(\mathbf{x}) \leq 0$ and black ones with $\phi(\mathbf{x}) > 0$). On the left, the scalar field is constructed using only surface particles (blue disks), causing the inner surface vertices (inner black dots) to have wrong scalar values. By taking the contribution of inner particles (gray disks), inner surface vertices have correct scalar values on the right. All blue cells in both images and also green cells on the right are sent to the triangulation stage. However, since only blue cells pass the triangulation criterion, the double layer problem that occurs on the left is avoided on the right.*

4. Optimized Surface Reconstruction

The computation time for extracting smooth surfaces is mainly influenced by the resolution of the MC grid and the smoothing radius.

In this section, we present an implementation which performs the scalar field computation only for surface vertices (Sec. 4.1). Therefore, robust criteria are proposed for determining grid vertices that are in close proximity to the fluid surface. Also, a parallel formulation for the scalar field construction is described, which avoids inherent data dependencies and is applicable to both CPUs and GPUs. Finally, we propose an efficient triangulation procedure which avoids computing redundant values on shared edges and scales with the number of surface vertices (Sec. 4.2).

In the beginning of the presented surface reconstruction pipeline, surface vertices are extracted. The extraction of surface vertices can be performed by using surface particles. In order to determine surface particles in a preprocessing step, the smoothed color field method [MCG03] is employed. The defined color field criterion catches the surface particles on the main fluid body precisely, while it fails to detect the isolated parts like splashes. Therefore, we also consider particles with less than 25 neighbours as surface particles.

Details of the scalar field computation for surface vertices and the triangulation procedure are explained in the following subsections.

4.1. Scalar Field Computation

The scalar field computation stage consists of three parts: extraction of vertices that reside in the close proximity of the surface, identification of the contributing particles for these vertices, and finally the scalar value computation. Implementation details of these stages together with an efficient parallelization technique are described in this section.

Extracting surface vertices. Any grid vertex that is close enough to a surface particle can be defined as a surface vertex. The close proximity of any surface particle can be easily defined as an AABB around the particle which spans a $2r$ length in each direction. Each grid vertex that resides in such a bounding box is marked as a surface vertex.

Using this technique, a thin region is spanned around surface particles which is sufficient to extract all surface vertices through which the surface is reconstructed. The scalar field computation is performed only for these vertices, and their corresponding data are collected in an array, namely $surfaceVertices$. Accordingly, each MC grid vertex stores just one integer value. For surface vertices, this value is the index of the corresponding entry in $surfaceVertices$, while it is -1 for other vertices.

Identification of the contributing particles. The extraction of surface vertices using surface particles should not be interpreted as if only surface particles would contribute to the final scalar field. One should keep in mind that many surface vertices, generally except the ones in the outermost layers, lie also in the influence radius of some inner particles. Ignoring the contributions of those inner particles leads to erroneous scalar values and bumpy surfaces. Furthermore, a second layer is formed inside the fluid erroneously, as illustrated in Fig. 2, left and dealt with in Sec. 4.2. In order to prevent bumpiness, we suggest to consider not only the sur-

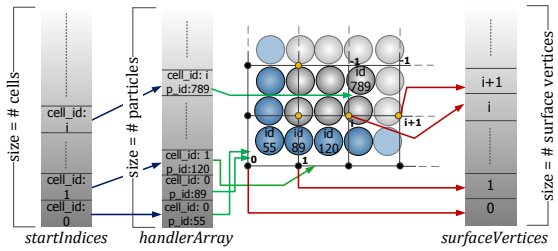

Figure 3: *Illustration of the different arrays used during the scalar field computation stage. Blue and gray disks represent surface and inner particles, respectively. Each particle has a handler (particle id, cell id pair) that is kept in the handlerArray in a cell-based sorted order. The cells of startIndices point to the corresponding start indices in the handlerArray. Each grid vertex is responsible for keeping one integer value, which is either -1 for inner vertices or the corresponding index of the surfaceVertices which keeps the scalar field computation related data.*

face, but also the inner particles' contributions in the influence radius of any surface vertex. In the presented method, this is accomplished by computing an AABB around the vertex along the user defined influence radius, and collecting the particles in cells which are overlapping this AABB. However, identifying the particles that lie in those cells is not straightforward, as will be explained next.

To optimize the performance, this stage is implemented using the Z-indexing method as discussed in [IABT11]. Thereby, the locality of spatially close cells in memory is enforced which reduces the memory transfer. This stage starts by pairing the particles with their cells where the cell id of any particle is computed as:

$$\mathbf{x} = \frac{(\mathbf{p}-\mathbf{g})}{s}, \tag{6}$$

$$\mathbf{cc} = (\lfloor \mathbf{x}.x \rfloor, \lfloor \mathbf{x}.y \rfloor, \lfloor \mathbf{x}.z \rfloor), \tag{7}$$

$$id = iM[\mathbf{cc}.x] \,|\, (iM[\mathbf{cc}.y] \ll 1) \,|\, (iM[\mathbf{cc}.z] \ll 2) \tag{8}$$

where $\mathbf{p}$ is the particle position, $\mathbf{g}$ is the minimum position of the grid, s is the cell size of the grid, $\mathbf{cc}$ is the cell coordinate and iM stands for the interleave map array which is used to store the Z-order indexing entries. For each particle, a simple structure is used to hold the particle id and its corresponding cell id, which is called *handler*. Handlers of all particles are placed in an array, namely *handlerArray*. Once a *handler* for each particle is prepared, the *handlerArray* is sorted with respect to the cell ids. At this point a new array, namely *startIndices*, is generated in the size of total cells. Thereby, each non-empty cell points to the corresponding entry in the *handlerArray* with lowest index (see Fig. 3).

Scalar value computation. The scalar value of each surface vertex is computed using the particles that reside in the AABB of the vertex. As mentioned previously, influence

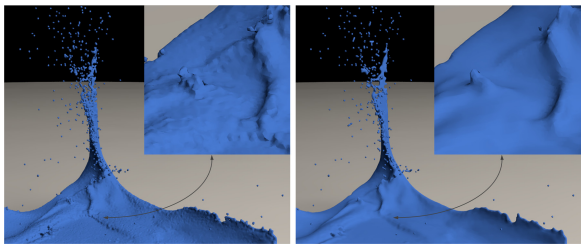

Figure 4: *Surface reconstruction of the CBD scene with $2r$ (left) and $4r$ (right) influence radii. Surface reconstruction times including the scalar field computation and triangulation are 7.5 and 60 seconds average per-frame on our 6-core CPU, respectively.*

radii smaller than $4r$ result in bumpy surfaces (see Fig. 4). Therefore, AABBs are created along the influence radius of $4r$. For each cell that resides in this bounding box, particles inside it are gathered using the previously computed cell-particle pairs. Paired particles of any cell can be easily accessed using the cell id and the *startIndices* array. This array returns the particles inside the cell by checking the *handlerArray* starting from the stored index until the next cell. Finally, the scalar value of the surface vertex is computed by using the contributions of these particles.

Parallelization. A major point in parallel implementations is avoiding race conditions, i. e. multiple threads should never try to write to the same memory address at the same time. However, this is not easy to handle in current surface reconstruction methods due to data dependencies. Our approach is specifically designed to avoid inherent data dependencies. Furthermore, by employing (Z-ordered) sorting, consecutive memory locations are read, which reduces the cache-miss rate and minimizes the memory transfer.

At the first stage, surface vertices are determined by simply traversing along the surface particles in parallel and finding the vertices that reside in their AABB. After the traversal, each surface vertex is pushed into the *surfaceVertices* together with its related data. This is the only serial part in our implementation due to possible race conditions.

At the second stage, the *handlerArray* is constructed in the size of the total number of particles. Thereby, the cell id of all particles is computed in parallel and stored at the corresponding positions. Sorting the pairs with respect to cell ids is a part that is harder to parallelize, and generally does not scale as good as the other parts of the method. For this aim, parallel sorting algorithms provided by the parallel programming libraries are employed. Furthermore, the generation of the *startIndices* array is easily processed in parallel without any race conditions.

At the final stage, the scalar value of each surface vertex is computed by traversing along all surface vertices in par-

Algorithm 1: Optimized Surface Reconstruction

```
foreach particle do
    compute color field quantity;
    mark if surface particle;
foreach surface particle do
    compute AABB that spans 2r distance on each axis;
    mark each vertex that lie in the AABB;
foreach particle do
    compute cell id (8);
    create a handler;
foreach surface vertex do
    compute AABB that spans 4r distance on each axis;
    find cells in AABB;
    foreach cell do
        find particles inside;
        add up each particle's contribution (4) ;
    compute scalar value of the vertex (1);
foreach surface vertex do
    create a cell;
    if all eight vertices are surface vertices then
        pass the cell to triangulation stage;
        if cell is on-surface then
            triangulate the cell
```

allel and by taking the contribution of the particles in the influence radius of each one.

In the most efficient serial implementation of the employed surface reconstruction method, a *scatter* approach would be used. Thereby, the particle list is traversed and the contribution of each particle to any vertex in its influence radius is summed up. Consequently, only the scalar values of vertices, that are influenced by the fluid, are computed. More importantly, the expensive neighborhood query of particles inside the influence region of a vertex is avoided. However, the parallelization of this serial implementation does not scale well since the scalar value of a vertex might be summed up by more than one particle at once.

In order to avoid such race conditions, we use a *gather* approach, i. e. vertices gather the contribution of particles. As we have developed various gather implementations for the scalar field computation, we observed that the performance is significantly affected by the neighborhood search. Therefore, we finally employ Z-index sort [IABT11], as it is advantageous in comparison to a variety of alternatives. However, pure Z-index sort without narrow band incorporation is generally outperformed by the scatter approach for up to four threads, depending on the particle resolution and surface complexity. This is due to additional steps required for the neighborhood search. Besides, using high grid resolutions and large influence radii is mandatory for obtaining high quality surfaces. Together with the fact that the number of contributing particles of the inner vertices are much larger than for the surface vertices, the number of grid vertices to process and queried particles for each vertex increases significantly. Therefore, we incorporated our narrow band technique in order to further improve the performance and decrease the memory consumption. These points are the motivation to compare our method with the scatter implementation and the pure Z-index sorting method without narrow band incorporation (see Sec. 5) .

Our algorithm is designed for parallel architectures. In order to test the scaling, both the CPU version, using OpenMP, and the GPU version, using CUDA, were implemented. No external parallel libraries were employed except for sorting the handlers. For the GPU version, the parallel radix sort is employed which is provided by the CUDA software development toolkit, while for the CPU version, a parallel implementation of the quicksort algorithm is used, which is provided by the OpenMP Multi-Threaded Template Library.

4.2. Triangulation

After computing the scalar values for MC grid vertices, the triangulation stage starts. In order to compute the exact surface intersection points, vertex values are interpolated along the cell edges. In a 3D MC grid, an edge can be shared by 4, 2 or 1 cell, depending on whether it is an inner, a boundary, or a corner edge. The computed intersection point on the edge never changes according to the cell that is sharing the edge. However, this point might be computed redundantly for each sharing cell.

In order to address the issue of redundant computations, we use a simple yet efficient method. According to this method, a grid vertex keeps three intersection point ids, each of them corresponds to one of the three possible edges leaving the vertex and initialized as -1. Besides, a *mesh* structure is used which keeps all intersection points, triangles and normals in separate arrays. Whenever an intersection point is computed on an edge, this point is inserted into the corresponding array. The index of the array element, that keeps this point, is stored in the related vertex of the edge by changing the initial value of -1. After marching to the next cell, vertices are firstly checked for possible past computations on shared edges, i. e. whether the intersection point id is -1 or larger. If such a computation is determined, previously computed point data is simply obtained from the corresponding array. Normals of the intersection points are also computed by interpolating the normals of grid vertices, and collected in their corresponding array. Finally, it remains to determine the triangles using the MC look-up tables, and to collect them in their corresponding array. This type of structure is quite appropriate to be stored in a memory efficient indexed mesh format (e. g. wavefront OBJ format). This technique is used to triangulate the on-surface cells which contribute to the final visualization. The efficient extraction of these

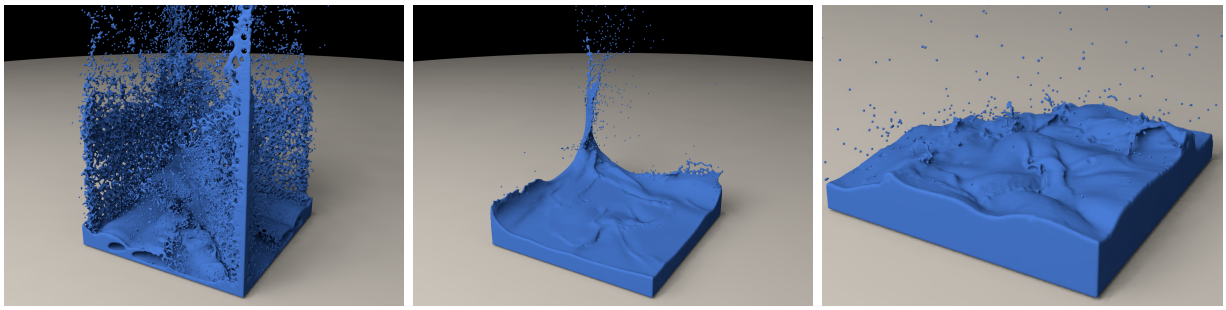

Figure 5: *CBD scene with 1.7 million particles. Note that thin sheets are well preserved (left and middle) due to a small cell size equal to $r/2$. The right image shows the surface in high curvature regions. Even very fine details, e. g. impact points of tiny droplets, are catched in this frame.*

			cell size=r			cell size=$r/2$		
scene	#particles	$\frac{\#\text{particles}_{surf}}{\#\text{particles}_{total}}$	$\frac{\#\text{vertices}_{surf}}{\#\text{vertices}_{total}}$	#triangles	grid res.	$\frac{\#\text{vertices}_{surf}}{\#\text{vertices}_{total}}$	#triangles	grid res.
Drop	360k	0.12	0.075	87k	160×73×147	0.065	351k	320×146×293
CBD	1.7m	0.13	0.1	420k	243×204×243	0.09	1.7m	485×409×486
Hemispheres	up to 2m	0.17	0.18	782k	333×63×438	0.16	3.1m	667×127×876
Hebe	up to 3.2m	0.09	0.08	903k	289×314×259	0.07	6.5m	578×628×518

Table 1: *General information of the presented scenes. The data are given as average per frame.*

scene	t_{sim}[sec]	t_{sp}[msec]	$t_{r\text{-}o}$[sec]	$t_{r\text{-}t}$[sec]
Drop	16	90	18	120
CBD	55	444	144	-
Hemispheres	63	520	168	520
Hebe	150	850	300	854

Table 2: *Averaged per frame timings of simulation, surface particle extraction, opaque and transparent rendering.*

cells is performed as follows. As stated previously, computing correct scalar values for surface vertices is essential to prevent bumpiness. These scalar values are also used to extract on-surface cells and to avoid the double layer problem on-the-fly. In contrast to existing approaches, that traverse all vertices, only surface vertices are traversed at this stage. Each surface vertex is used to create a grid cell by defining a corner point and by using the cell size information. This cell is sent to the triangulation stage only if all eight vertices have been previously marked as surface vertices. Each cell that is sent to the triangulation stage is checked whether it passes the triangulation criterion, i. e. whether it is on-surface or not. Any on-surface cell is then triangulated using the technique which has been explained in the beginning of the section.

The triangulation process is not parallelized, since it takes a rather small amount of time in contrast to the scalar field computation, which is the main bottleneck of the surface reconstruction. However, scaling the triangulation with the surface instead of the fluid volume brings in another significant speed-up (see Sec. 5).

Our optimized surface reconstruction method, including the scalar field computation and the triangulation, is summarized in Alg. 1.

5. Results

In order to demonstrate the utility of our method, we applied it to four different scenarios: *Hebe*, *Drop*, *CBD* and *Hemispheres* with two different cell sizes. For all scenes, the support radius is $4r$. All experiments have been performed on an Intel Xeon X5680 with six 3.33 GHz cores, 24GB RAM and an NVIDIA Quadro 6000 graphic card. All the reconstructed surfaces in still images and the final videos were rendered using POV-Ray [POV11]. A detailed analysis of the presented examples are given in Tables 1 and 2.

All timings, memory consumptions and other related data are given as average per frame. The given surface reconstruction times throughout this section present the total time as a combination of the scalar field computation and the triangulation. GPU timings include memory allocation as well as memory transfer to and from the graphics card. All still images of the scenes are generated with a cell size of $r/2$.

In a first example, we analyze the performance of the popular corner breaking dam (*CBD*) scenerio (see Fig. 5). In this scene, the fluid is simulated with 1.7 million particles. In the CPU version, the surface reconstruction takes only 2.8 sec with a cell size of r , and about 60 sec with $r/2$; while it takes

method	t_{sf}[sec]	t_{tri}[sec]	memory [GB]
scatter	31.5	7.5	4.83
Z-index-1 thread	92	7.5	4.7
Z-index-6 threads	16	7.5	4.7
Z-index-GPU	6.9	7.5	3.9(CPU)+1.2(GPU)
our-1 thread	14.4	0.18	1.39
our-6 threads	2.6	0.18	1.39
our-GPU	1.0	0.18	1.2(CPU)+0.4(GPU)

Table 3: *Comparison of our method to the scatter approach and pure Z-index sorting without narrow band incorporation using [SSP07] for the CBD scene with a cell size of r. In the table, t_{sf} and t_{tri} stand for the scalar field computation and the triangulation time, respectively.*

method	t_{sf}[sec]	t_{tri}[sec]	memory [GB]
scatter	703.5	63.5	22.9
Z-index-1 thread	2000	63.5	21.8
Z-index-6 threads	350	63.5	21.8
Z-index-GPU	NA	NA	NA
our-1 thread	343.4	1.1	4.1
our-6 threads	58.5	1.1	4.1
our-GPU	14.1	1.1	3.9(CPU)+2.1(GPU)

Table 4: *Comparison of our method to the scatter approach and pure Z-index sorting without narrow band incorporation using [SSP07] for the CBD scene with a cell size of $r/2$.*

1.2 and 15 sec for the GPU version, respectively. In order to demonstrate our contribution more clearly, we compared our method with two different implementations of [SSP07] on this scene. Tables 3 and 4 show a performance comparison of our method with a serial scatter implementation and a simple gather implementation with Z-index sorting, but without narrow band. When compared to the scatter approach, the speed-up of our method with one thread is small. This is due to additional data structures and computations which are introduced for the surface vertex determination and in order to avoid race conditions. However, the performance of the method shows a significant improvement proportional to the number of utilized threads since it scales well on multi-core CPUs and GPUs. In order to show the scaling of the method for a larger number of CPU cores, we made an experiment on a 24-core Intel Xeon 2.66 GHz machine for this scene which is illustrated in Fig. 6. This figure shows that the method scales better for smaller cell sizes. The reason for this is that in higher resolution grids, more values are computed in parallel which increases the domination of the parallel working parts over serial parts. In addition to the scalar field computation, the performance of the triangulation is also significantly improved by scaling it with the number of surface vertices and by avoiding redundant computations. In comparison to the original method, a speed-up of up to 60 is achieved, see Tab. 3 and 4. Besides, as no scalar field related data is stored for non-surface vertices, the memory consumption is significantly reduced, which allows for using even higher resolution grids. Furthermore, we compared our method to the pure Z-index sorting method without incorporating our narrow band technique. The triangulation time and the memory consumption with the pure Z-index sorting method are very similar to the scatter approach, since both approaches scale with the volume instead of the surface. However, the scalar field construction takes 16 sec with a cell size of r, and 350 sec with $r/2$ using 6 threads, which shows that the gather implementation without narrow band needs three threads to outperform the scatter implementation. The incorporation of the narrow band significantly improves the gather implementation.

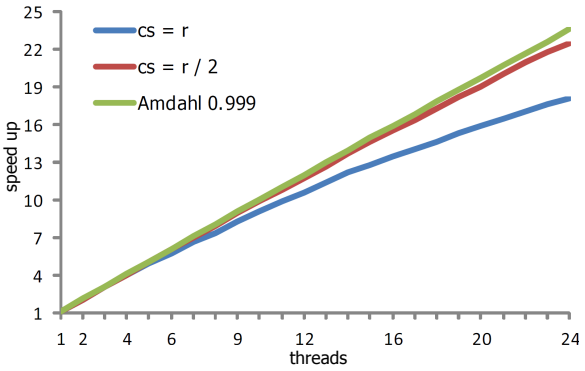

Figure 6: *The scaling of the parallel scalar field computation method for the CBD scene. The method scales well with respect to the number of utilized threads. It yields a better scaling for smaller cell sizes (cs) because of the domination of parallelized parts over the serial part.*

The next example is the *Hemispheres* scene where a fluid with up to 2 million particles flows onto two half spheres (see Fig. 7). The surface reconstruction time of the scene is 4 sec and 108 sec on the CPU and 1.7 sec and 25 sec on the GPU for cell sizes of r and $r/2$, respectively. Note that the smoothness of the thin layer flowing on the plane and thin sheets on the hemispheres are well preserved since a very fine grid resolution is used (see Fig. 7). This is generally challenging to achieve for millions of particles without using the proposed optimization techniques. The analyses in Tables 5 and 6 show that the memory consumption of this scene is smaller than for the *CBD* scene, although the ratio of the number of surface vertices to total vertices is larger. This is due to the average per-frame resolution of the scene which is smaller than for *CBD*, since in many frames of the *CBD* scene, spreading splashes result in a sparsely filled MC grid.

The third example is the *Drop* scene where a water drop falls into a filled pool (see Fig. 8). For this scene, we have used 360k particles with an average surface reconstruction time of 0.65 sec and 10.7 sec on the CPU and 0.5 sec and 3.8

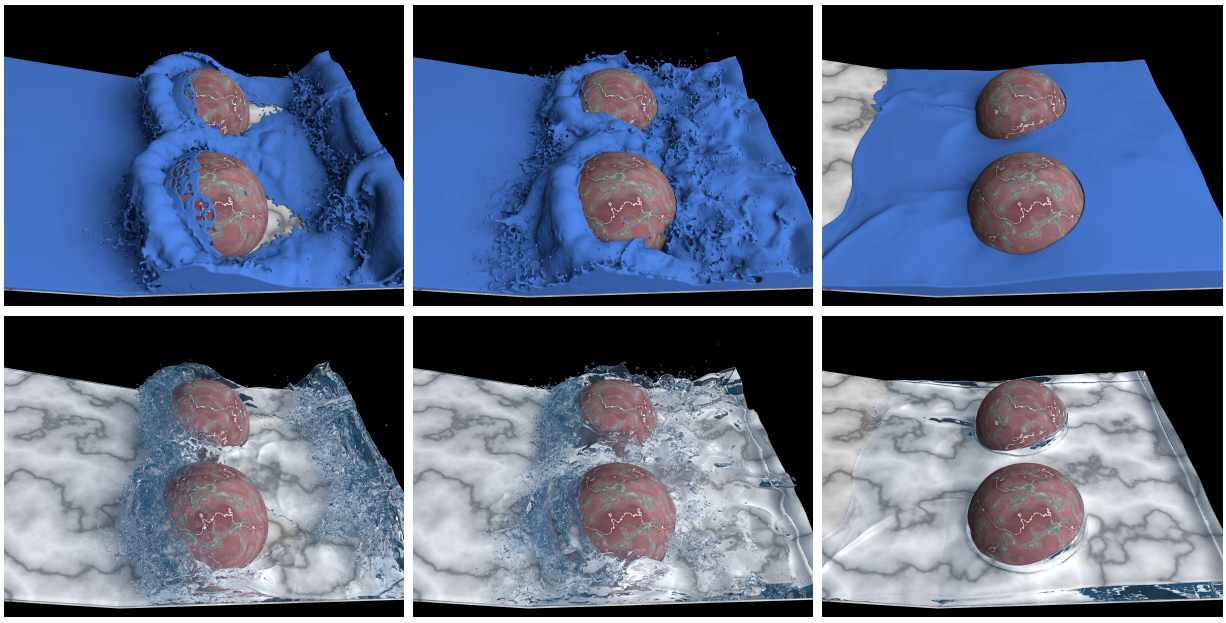

Figure 7: *Surface reconstruction of the Hemispheres scene with a varying particle count of up to 2 million. Note that the smoothness of the thin flowing layer (left and middle) and the fine details of the fluid over the hemispheres (left) are well-preserved.*

scene	$t_{sf\text{-CPU 1 thread}}$	$t_{sf\text{-CPU 6 threads}}$	$t_{sf\text{-GPU}}$	t_{tri}	mem_{CPU}[MB]	mem_{GPU}[MB]
Drop	3.3 sec	0.6 sec	0.37 sec	0.05 sec	271	CPU(265)+GPU(85)
CBD	14.4 sec	2.6 sec	1.0 sec	0.18 sec	1390	CPU(1201)+GPU(410)
Hemispheres	21.1 sec	3.8 sec	1.5 sec	0.27 sec	1143	CPU(980)+GPU(302)
Hebe	36.6 sec	7.1 sec	2.3 sec	0.62 sec	2575	CPU(2224)+GPU(591)

Table 5: *Performance of the presented scenes for scalar field computation (t_{sf}) and triangulation (t_{tri}) together with memory consumption using a cell size equal to r. All data are average-per-frame.*

scene	$t_{sf\text{-CPU 1 thread}}$	$t_{sf\text{-CPU 6 threads}}$	$t_{sf\text{-GPU}}$	t_{tri}	mem_{CPU}[MB]	mem_{GPU}[MB]
Drop	61.1 sec	10.4 sec	3.5 sec	0.28 sec	415	CPU(389)+GPU(170)
CBD	343.4 sec	58.5 sec	14.1 sec	1.13 sec	4076	CPU(3888)+GPU(2130)
Hemispheres	617.3 sec	105.4 sec	21.9 sec	2.92 sec	3356	CPU(2760)+GPU(1322)
Hebe	1640.8 sec	277.1 sec	32.3 sec	6.86 sec	7532	CPU(6145)+GPU(3281)

Table 6: *Performance of the presented scenes for scalar field computation (t_{sf}) and triangulation (t_{tri}) together with memory consumption using a cell size equal to $r/2$. All data are average-per-frame.*

sec on the GPU for cell sizes of r and $r/2$, respectively. The *Drop* scene is an effective example for revealing the generation of smooth surfaces and fine details even for a small number of particles. In this scene, thin sheets arise after the impact and the prominent water crown is generated as well as many splashes.

The last and the largest example is the *Hebe* scene which is an example of revealing the fine details of a relatively viscous fluid which is poured onto the Hebe statue and finally collected in a box (see Fig. 9). For this scene, up to 3.2 million particles have been reconstructed per frame. The average surface reconstruction time for the scene is 7.7 sec and 285 sec on the CPU and 4 sec and 40 sec on GPU for cell sizes of r and $r/2$, respectively.

Comparison. We have applied our parallel algorithm to the scalar field computation method of [SSP07] and [YT10], and compared the performance with an optimal serial implementation of these methods. The performance and scaling comparisons are given for the CBD scene with 150k particles, since using smaller number of particles enables to

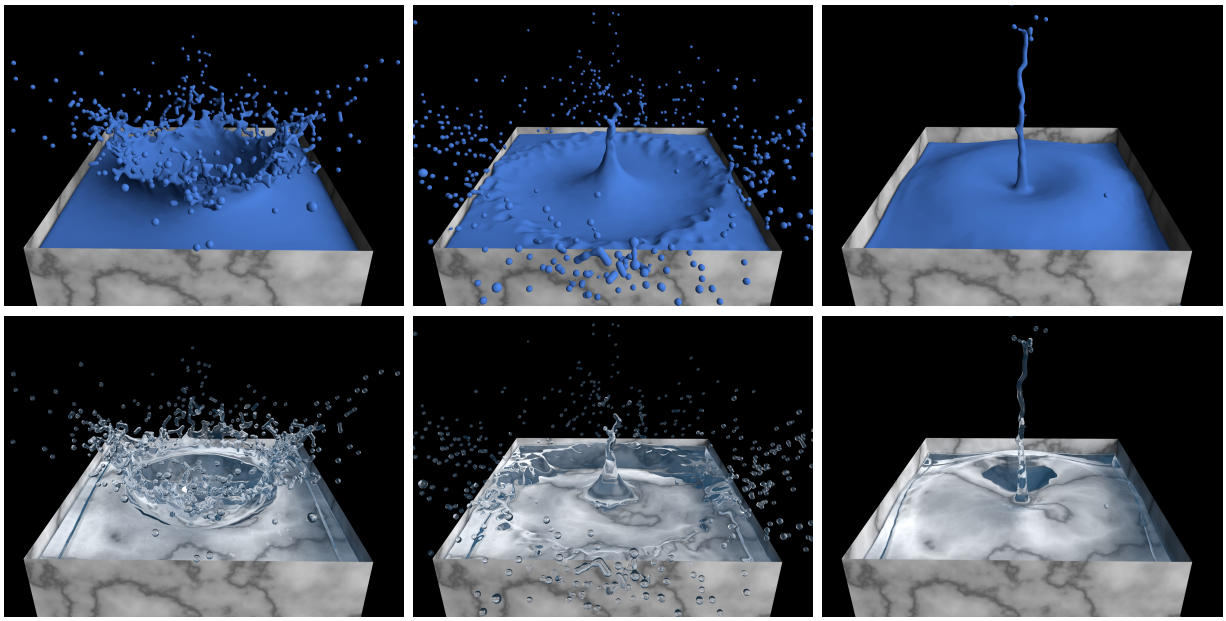

Figure 8: *Surface reconstruction of the drop scene with 360k particles. Fine details and smoothness are well-preserved even for a low number of particles.*

see the visual differences between these two methods more clearly. In the presented scene, the cell size of $r/2$ is used. The grid resolution is $186\times264\times210$, while the ratios between surface/total particles and surface/total vertices are 0.15 and 0.12, respectively on average per frame.

Our experiments indicate that the method of [SSP07] is approximately 2 times faster than the method of [YT10], when comparing the optimal serial implementation of both methods. For [SSP07], the memory requirement is reduced by a factor of five by our method, while the speed-up is almost 50, see Tab. 7. In contrast, for [YT10] the speed-up is approximately 20 (Tab. 8), which shows that the performance of [SSP07] is 5 times faster than [YT10] after applying our algoritm. The reason of this difference is that [YT10] requires additional, performance-critical steps. For instance, an extra neighborhood search increases the number of memory lookups, and the traversal over a non-uniform neighborhood area decreases the cache-hit rate. Besides, anisotropy matrix computation adds extra overhead.

In terms of visual quality, both methods yield very smooth surfaces. However, while spurious blobs in concave regions and in between splashes are eliminated with [SSP07], our experiments indicate that they still occur with [YT10]. Besides, the underlying particle set is not covered exactly since the position smoothing introduces new positions for particles and especially splash particles are likely to be missed as they shift through their neighbourhood center. In addition, slight volume shrinkage occurs. In order to alleviate this problem, we use a small position smoothing constant (λ) with the value of 0.3. The visual comparison of the two methods is shown in Fig. 10.

method	t_{sf}[sec]	t_{tri}[sec]	memory [GB]
[SSP07]	110	5	2.1
opt.-1 thread	58	0.2	0.38
opt.-6 threads	10	0.2	0.38
opt.-GPU	2.5	0.2	0.35(CPU)+0.19(GPU)

Table 7: *Comparison of our optimized method using the scalar field computation defined in [SSP07] to the original method [SSP07] for the CBD-150k scene with a cell size of $r/2$.*

method	t_{sf}[sec]	t_{tri}[sec]	memory [GB]
[YT10]	235	5	3.4
opt.-1 thread	150	0.2	0.82
opt.-6 threads	26	0.2	0.82
opt.-GPU	12	0.2	0.73(CPU)+0.4(GPU)

Table 8: *Comparison of our optimized method using the scalar field computation defined in [YT10] to the original method [YT10] for the CBD-150k scene with a cell size of $r/2$.*

As stated previously, our method is applicable to any scalar field construction method. According to its performance benefits and high-quality results, we prefer [SSP07] over other methods.

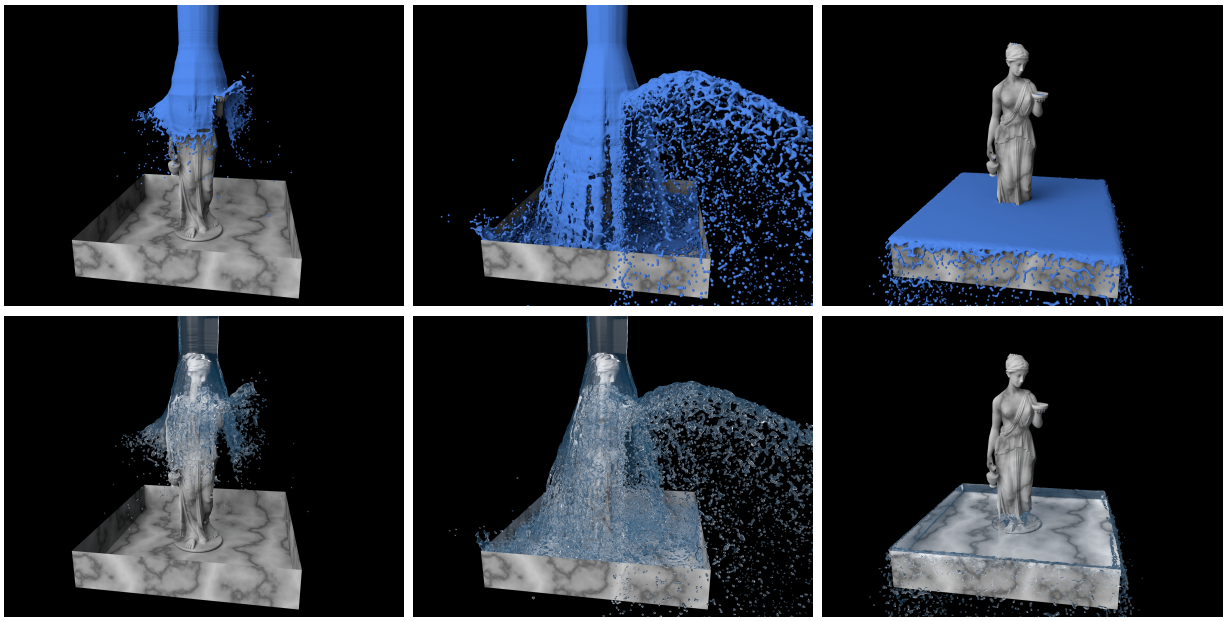

Figure 9: *Surface reconstruction of the Hebe scene with a varying particle count of up to 3.2 million. Even for such a large scene, the surface reconstruction is still efficient with less than 3 sec for a cell size equal to r and less than 40 sec for a cell size equal to $r/2$ on average on the GPU.*

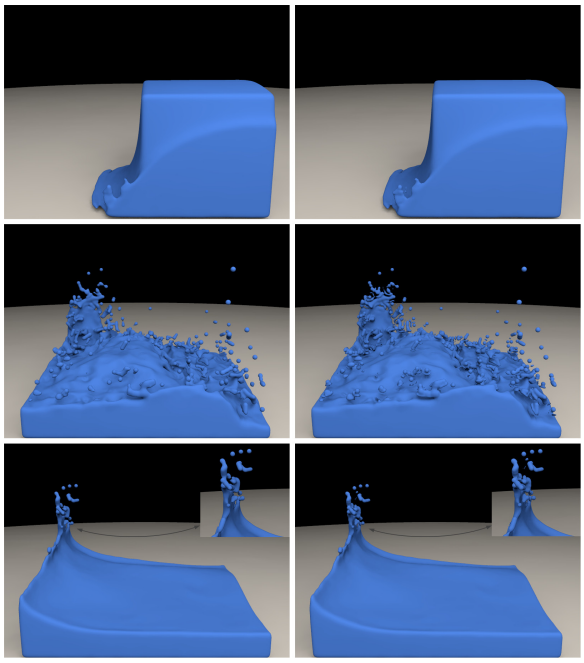

Figure 10: *Visual comparison between [SSP07] (left) and [YT10] (right) on CBD-150k scene. The zoomed-in corner is particularly included in the bottom row, so as to show the differences in splash regions clearly.*

6. Conclusion

In this paper, we presented techniques to improve the performance of high quality surface generation for particle-based fluids. Our method scales with the fluid surface instead of the volume. Unnecessary scalar field computations and data storage for grid vertices that lie far inside the fluid volume or outside the fluid surface are avoided. It can be applied to any scalar field construction approach that has been developed for particle-based fluids. Furthermore, we proposed an efficient parallelization technique which scales well on multi-core CPUs and GPUs.

Obviously, if isolated particles, i. e. particles without any neighbor, spread too much, the outline of the MC grid becomes very large. This causes our method to consume more memory since one integer is still kept for all grid cells as an index of the *handlerArray* in the *startIndices* and one integer for all grid vertices as an index of the *surfaceVertices*, and these arrays scale with the volume. Compared to the original algorithm, however, the memory consumption caused by those arrays is significantly smaller. In the future, we intend to generate smaller MC grids by extracting isolated particles in a preprocessing step and handling them directly in the renderer as spheres, i. e. their ideal shape. Furthermore, instead of sticking to uniform grids, we plan to utilize an adaptive grid structure where we can construct higher resolution grids in the narrow band region or in high-curvature regions, i. e. the regions that have more detailed features. In addition, we also would like to study

whether out-of-core techniques can be incorporated to further improve the memory efficiency.

In the presented parallelization algorithm, we employ general parallelization techniques that are effective on any parallel architecture. Note that even though a speed-up of 50 is achieved on the GPU, we did not employ any GPU specific optimization. In the future, we would like to investigate if *warp partitioning* techniques and *data tiling* strategies can be applied, in order to reduce the number of idle threads and the number of accesses to the global memory. Thereby, we expect the performance to be further improved for the GPU version.

Acknowledgements

We thank Arthur Wahl and Edgar Oswald for their help in the development of the project. This project is supported by the German Research Foundation (DFG) under contract number TE 632/1-1.

References

[ALD06] ADAMS B., LENAERTS T., DUTRE P.: *Particle Splatting: Interactive Rendering of Particle-Based Simulation Data*. Tech. Rep. CW 453, Katholieke Universiteit Leuven, 2006. 1, 3

[APKG07] ADAMS B., PAULY M., KEISER R., GUIBAS L.: Adaptively sampled particle fluids. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (New York, NY, USA, 2007), ACM Press, p. 48. 1, 2

[BB09] BROCHU T., BRIDSON R.: Robust Topological Operations for Dynamic Explicit Surfaces. *SIAM Journal on Scientific Computing 31, 4* (2009), 2472–2493. 2

[Bli82] BLINN J.: A Generalization of Algebraic Surface Drawing. *ACM Trans. Graph. 1*, 3 (1982), 235–256. 2

[Bri03] BRIDSON R.: *Computational aspects of dynamic surfaces*. PhD thesis, Stanford University, 2003. 1, 3

[FAW10] FRAEDRICH R., AUER S., WESTERMANN R.: Efficient High-Quality Volume Rendering of SPH Data. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization / Information Visualization 2010)* (2010), 1533–1540. 1, 3

[GSSP10] GOSWAMI P., SCHLEGEL P., SOLENTHALER B., PAJAROLA R.: Interactive SPH Simulation and Rendering on the GPU. In *Proceedings of the Eurographics / SIGGRAPH Symposium on Computer Animation (SCA)* (Aire-la-Ville, Switzerland, Switzerland, 2010), pp. 55–64. 3

[HWB04] HOUSTON B., WIEBE M., BATTY C. C.: RLE sparse level sets. In *Proceedings of the SIGGRAPH 2004 conference on sketches & applications* (New York, NY, USA, 2004). 1, 3

[IABT11] IHMSEN M., AKINCI N., BECKER M., TESCHNER M.: A Parallel SPH Implementation on Multi-Core CPUs. In *Computer Graphics Forum* (2011), vol. 30, pp. 99–112. 1, 5, 6

[IAGT10] IHMSEN M., AKINCI N., GISSLER M., TESCHNER M.: Boundary handling and adaptive time-stepping for PCISPH. In *Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS* (Aire-la-Ville, Switzerland, Switzerland, 2010), pp. 79–88. 2

[LC87] LORENSEN W., CLINE H.: Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), ACM Press, pp. 163–169. 1, 2

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 154–159. 2, 3, 4

[Mül09] MÜLLER M.: Fast and robust tracking of fluid surfaces. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2009), SCA '09, ACM, pp. 237–245. 2

[MSD07] MÜLLER M., SCHIRM S., DUTHALER S.: Screen Space Meshes. In *Proceedings of ACM SIGGRAPH / EUROGRAPHICS Symposium on Computer Animation (SCA)* (Aire-la-Ville, Switzerland, Switzerland, 2007), pp. 9–15. 1, 3

[NM06] NIELSEN M. B., MUSETH K.: Dynamic Tubular Grid: An efficient data structure and algorithms for high resolution level sets. *J. Scient. Comput. 26, 3* (2006), 261–299. 3

[NNSM07] NIELSEN M. B., NILSSON O., SÖDERSTRÖM A., MUSETH K.: Out-of-core and compressed level set methods. ACM Transactions on Graphics, Vol. 26, No.4, 2007. 3

[POV11] POVRAY: Persistence of vision raytracer (version 3.7 rc3) [computer software]. http://www.povray.org/, October 01 2011. 7

[SP09] SOLENTHALER B., PAJAROLA R.: Predictive-corrective incompressible SPH. In *SIGGRAPH '09: ACM SIGGRAPH 2009 Papers* (New York, NY, USA, 2009), ACM, pp. 1–6. 2

[SSP07] SOLENTHALER B., SCHLÄFLI J., PAJAROLA R.: A unified particle model for fluid-solid interactions. *Computer Animation and Virtual Worlds 18*, 1 (2007), 69–82. 1, 2, 3, 4, 8, 9, 10, 11

[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* (New York, NY, USA, 2009), ACM, pp. 91–98. 3

[WH94] WITKIN A., HECKBERT P.: Using Particles to Sample and Control Implicit Surfaces. In *SIGGRAPH '94: Computer Graphics and Interactive Techniques 28* (New York, NY, USA, 1994), ACM, pp. 269–277. 2

[Wil08] WILLIAMS B.: *Fluid surface reconstruction from particles.* Master's thesis, University Of British Columbia, 2008. 1

[YT10] YU J., TURK G.: Reconstructing Surfaces of Particle-Based Fluids Using Anisotropic Kernels. In *SCA '10: Proceedings of the 2010 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, 2010), Eurographics Association, pp. 217–225. 1, 2, 3, 9, 10, 11

[ZB05] ZHU Y., BRIDSON R.: Animating sand as a fluid. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), ACM Press, pp. 965–972. 1, 2, 3, 4

[ZPvBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), ACM Press, pp. 371–378. 3