Efficient Surface Reconstruction for SPH Fluids

vorgelegt von

Gizem Akinci

Dissertation zur Erlangung des Doktorgrades
der Technischen Fakultät,
Albert-Ludwigs-Universität Freiburg im Breisgau

May 2014

1. Gutachter	Prof. Dr. Matthias Teschner
2. Gutachter	Prof. Dr. Jan Bender
Dekan	Prof. Dr. Yiannos Manoli
Datum der Disputation	15.05.2014

Abstract

Smoothed Particle Hydrodynamics (SPH) is a popular particle-based fluid simulation technique with its wide application area ranging from the entertainment industry (e.g., movies, video games, commercials) to scientific research fields (e.g., medical simulations, dam and flood simulations, petroleum research). As the method has been gaining increased interest, simulation scenarios have been becoming more diverse and more complex. However, surface extraction from particle-based fluid data still remains as one of the bottlenecks due to large computation time and memory consumption requirements, and also because of difficulties in achieving smooth and detailed surfaces. This thesis presents some new techniques that focus on achieving high quality SPH surfaces efficiently.

The thesis starts by reviewing techniques regarding the simulation and surface reconstruction for SPH fluids. Those surface reconstruction techniques predominantly focus on essentials of the field, e.g., lowering memory consumption and computation time, increasing the surface quality or both.

Next, we present a novel parallel algorithm so as to improve the performance of time consuming scalar field construction step. Our method scales nearly linearly on multi-core CPUs and up to fifty times faster on GPUs compared to the original scalar field construction approaches. The method scales with the fluid surface instead of its volume by constructing the scalar field only in the narrow band region where the fluid surface actually lies. Therefore, the method works efficiently even on single-core CPUs. Furthermore, we present some values for an efficient parameter setup which affect the final quality of the reconstructed surface.

In our next contribution, we focus on achieving smooth and detailed fluid surfaces within a reasonable time. Thus, we propose to use a method, based on post-processing of surface meshes, that is applicable to particle position data sets in a frame by frame basis. Our method combines existing scalar field computation techniques with two post-processing steps: decimation and subdivision. This is motivated by an improved representation of smaller surface details, reduction of bumps in flat regions, reduced overall computation time and reduced memory consumption. Our results demonstrate that in comparison to other approaches with comparable surface quality, our pipeline runs up to twenty times faster with up to 80% less memory and secondary storage consumption.

Finally, we present an adaptive spatial data structure, which adapts its cells according to the surface curvature. Low curvature regions are handled in low resolution cells with fewer triangles, while high curvature regions are represented with more triangles using high resolution cells, which helps to preserve surface details. Mesh blocks from different resolution cells are seamlessly stitched by closing cracks with new triangles. Our method produces similar results with less number of triangles, up to four times better memory consumption, and up to 60% better performance when compared to the single level high resolution uniform grid approach.

Throughout this thesis, we present various comparisons of our methods with previous approaches in the literature in order to highlight the benefits of our approaches. Visual comparisons and performance comparisons are given at the end of each corresponding chapter together with their explanations.

Zusammenfassung

Smoothed Particle Hydrodynamics (SPH) ist eine weit verbreitete partikelbasierte Technik zur Simulation von Fluiden, deren breite Anwendung von der Unterhaltungsindustrie (beispielsweise in Filmen, Videospielen, Werbung) bis hin zu wissenschaftlicher Forschung (beispielsweise in medizinischen Simulationen, Überflutungsszenarien, Erdölforschung) reicht. Im Zuge des wachsenden Interesses an der Methode wurden auch die Simulationsszenarien immer verschiedenartiger und komplexer. Dennoch ist die Oberflächenrekonstruktion partikelbasierter Fluiddaten wegen langer Berechnungszeiten, hohem Speicheraufwand und Problemen bei der Rekonstruktion einer glatten und detaillierten Fluidoberfläche einer der Flaschenhälse. Diese Doktorarbeit stellt einige Techniken vor, mittels derer qualitativ hochwertige Fluidoberflächen erreicht werden können.

Wir beginnen mit einer Vorstellung der Oberflächenrekonstruktionstechniken für SPH. Diese konzentrieren sich auf verschiedene Anforderungen, beispielsweise auf die Verringerung des Speicherverbrauchs und der Berechnungszeit, die Verbesserung der Oberflächenqualität oder auf beides.

In dieser Dissertation stellen wir zunächst einen neuen parallelen Algorithmus zur Verbesserung der Performanz des zeitintensiven Erstellungsprozesses skalarer Felder vor. Unsere Methode skaliert fast linear auf Mehrkernprozessoren und ist, verglichen mit herkömmlichen Skalarfelderstellungsansätzen, auf GPUs bis zu fünfzig Mal schneller. Anstelle des Volumens skaliert die vorgestellte Methode mit der Oberfläche, indem das Skalarfeld nur in unmittelbarer Nähe der Fluidoberfläche erstellt wird. Daher arbeitet die Methode selbst auf Einkern-CPUs effizient. Des Weiteren werden Werte für ein effektives Parametersetup vorgeschlagen, das auch die finale Qualität der rekonstruierten Oberfläche beeinflusst.

In unserem nächsten Beitrag konzentrieren wir uns auf das Erreichen einer glatten und detaillierten Fluidoberfläche innerhalb annehmbarer Zeit. Dafür schlagen wir vor, eine Methode zu nutzen, die auf einem Nachbearbeitungsschritt der Mesh-Oberfläche basiert und auf Datensätze von Partikelpositionen, die Bild für Bild zur Verfügung stehen, anwendbar ist. Unsere Methode kombiniert Techniken zur Berechnung existierender Skalarfelder mit zwei Nachbearbeitungsschritten: Dezimierung und Unterteilung. Dies ist durch eine verbesserte Repräsentation kleinformatiger Details, eine Verminderung von Unebenheiten in ebenen Bereichen, eine verringerte Gesamtberechnungszeit und einen geringeren Speicherverbrauch begründet. Unsere Ergebnisse zeigen, dass unsere Pipeline, verglichen mit anderen Ansätzen mit vergleichbarer Oberflächenqualität, mit bis zu 80% weniger Haupt- und Sekundärspeicherverbrauch bis zu zwanzigmal schneller ist.

Zuletzt stellen wir eine adaptive räumliche Datenstruktur vor, die ihre Zellen entsprechend der Oberflächenkrümmung anpasst. Regionen geringer Krümmung werden von grob aufgelösten Zellen mit wenigen Dreiecken, Regionen starker Krümmung von hoch aufgelösten Zellen mit mehr Dreiecken repräsentiert. Mesh-Blöcke aus Zellen verschiedener Auflösungen werden nahtlos mit einander verbunden, indem Zwischenräume mit weiteren Dreiecken gefüllt werden. Unsere Methode erzeugt ähnliche Ergebnisse mit weniger Anzahl der Dreiecke, bis zu vier Mal bessere Speicherverbrauch und bis zu 60% mehr Leistung im Vergleich zu der einzigen Ebene hoher Auflösung gleichmäßigen Raster-Ansatz.

Um den Nutzen unserer Ansätze zu verdeutlichen, zeigen wir im Verlauf dieser Doktorarbeit verschiedene Vergleiche unserer Methoden mit bekannten vorheri-

gen Ansätzen. Am Ende jedes Kapitels werden entsprechende visuelle Vergleiche und Vergleiche der Performanz mit dazugehörigen Erklärungen gegeben.

Contents

Acknowledgments

First of all, I would like to thank my advisor Prof. Matthias Teschner for giving me the oppurtunity to be a part of the Computer Graphics team of the Freiburg University starting from my master studies.

I would like to thank my colleagues Markus Ihmsen, Jens Cornelis, Marc Gissler and Rüdiger Schmedding for their support during my PhD studies. Many thanks go to Cynthia Findlay and Veria Popa for their all time help and patience.

I wish to particularly thank to my husband, colleague and teammate Nadir Akinci. Without his knowledge and support, everything would be different in my life.

My dear and little Defne, you are the joy of my life. Thank you for making me the happiest mother in the world.

At last but not least, I am grateful for having such great parents, such an incredible sister and the best family. They were always there for me whenever I asked. Thank you.

List of Figures

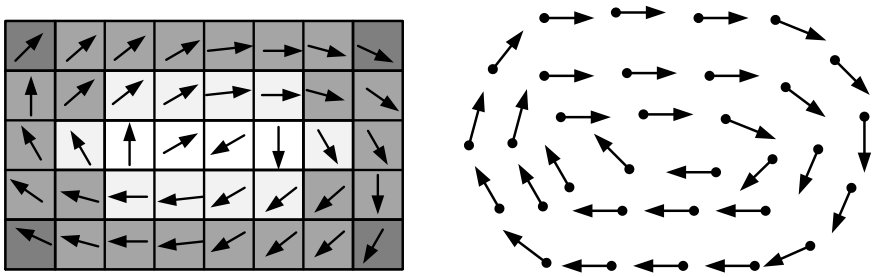
List of Tables

1 Introduction

In computer graphics, the simulation of fluids has been becoming increasingly popular due to its multiple area of usage, for instance, entertainment industry (e.g., movies, video games, commercials), scientific field (e.g., dam break or various flood scenarios, petroleum research), and medicine (e.g., virtual pre-surgery applications). Fluids are among the most challenging phenomena to simulate realistically; they range in complexity from excessively time-consuming high quality offline animations to low-resolution real-time systems. Two mainly competing techniques for simulating fluids are known as Eulerian and Lagrangian approaches, where each technique has its own advantages and disadvantages. In Eulerian techniques, fluid attributes are solved on stationary grid points in a finite simulation domain. In addition to position, velocity and acceleration, the most commonly solved fluid attributes include pressure, density and viscosity. Different from Eulerian viewpoint, Lagrangian techniques represent the continuous fluid as a discrete particle set, and aforementioned fluid attributes are carried by those particles (see Figure 1.1 for a two dimensional comparison). In this thesis, we employ Smoothed Particle Hydrodynamics (SPH) for fluid simulation, which is the most commonly used Lagrangian model in the field. With recent advances in SPH, it is now possible to simulate various fluid phenomena, including complex scenarios with more than 10 million particles, e.g., [IABT11, ICS+13].

In addition to simulation, fluid animations host another equally interesting part which is visualization. So as to visualize SPH fluids in a realistic way, we need a proper surface extracted from the particle set. However, surface reconstruction in SPH still remains as a challenging problem because irregularly sampled particle data requires a time consuming and error-prone parameter set up phase in obtaining smooth surfaces. In addition, obtaining high quality results is computationally expensive, causes large memory footprint and large secondary storage consumption. Therefore, recent studies in the SPH surface reconstruction field focus on improving either the surface quality or the computation time and memory consumption efficiency.

Currently, techniques that have been applied to render surfaces from SPH particles range from high performance methods that include, for instance, point based visualization [Ngu07] or ray-casting of metaballs [KSN08] on GPUs, interactive ray tracing, e.g., [PSLH98], screen space techniques, e.g., [ALD06, MSD07, vdLGS09] or image space 3D metaballs [ZSP07], where these methods aim interactive rates but produce lower quality visualizations, to offline view-independent methods that include, for instance, polygonization of implicit surfaces with, e.g., Marching Cubes [LC87], marching tiles [Wil08], or Delaunay triangulation



(a) **(b)**

Figure 1.1 – (a) Grid-based Eulerian approach. Fluid attributes, e.g., velocity (arrows) or density (grid cell gradient), are solved on non-moving grid points. (b) Particle-based Lagrangian approach. Each freely moving particle carries its own fluid attribute.

[SVK+] that aim high quality surface generations. Among all given examples, the Marching Cubes approach is the most commonly used method because of its simplicity, efficiency and ability to produce high quality surface meshes. In this thesis, the Marching Cubes approach is used to represent fluid surfaces.

Within the context of Marching Cubes approach, it is important to note that an appropriate scalar field computation is required to improve the surface quality. There exist various smoothing techniques, e.g., [ZB05, APKG07, SSP07, YT10, BGB11, OCR11], that aim a proper scalar field computation for SPH fluids. However, advantages and disadvantages coexist for each of these techniques, which enforces the user to be careful on the design decision. Besides, a successful parameter set up is required for all mentioned methods in order to achieve a surface up to the mark. In addition to the corresponding parameter setup of the chosen scalar field computation technique, the Marching Cubes grid cell size plays an important role in the surface quality. However, the grid resolution trades of performance. In other words, using low resolution Marching Cubes grids produces visible mass loss while being computationally efficient, or vice versa. The number of contributing particles per grid node is also another essential factor for the final quality. These mentioned factors all together make the SPH fluid surface reconstruction step error-prone and time consuming.

In this thesis, we present three main research contributions that concentrate on improving some of the major problems of surface reconstruction in SPH: a narrow-band based parallel surface reconstruction presented in Chapter 4, an efficient surface reconstruction pipeline proposing post-processing for SPH fluid surface meshes presented in Chapter 5 and an adaptive surface reconstruction technique based on fluid curvature presented in Chapter 6. Throughout the thesis, we present visual and performance comparisons between different scalar field computation techniques and various corresponding parameter setup.

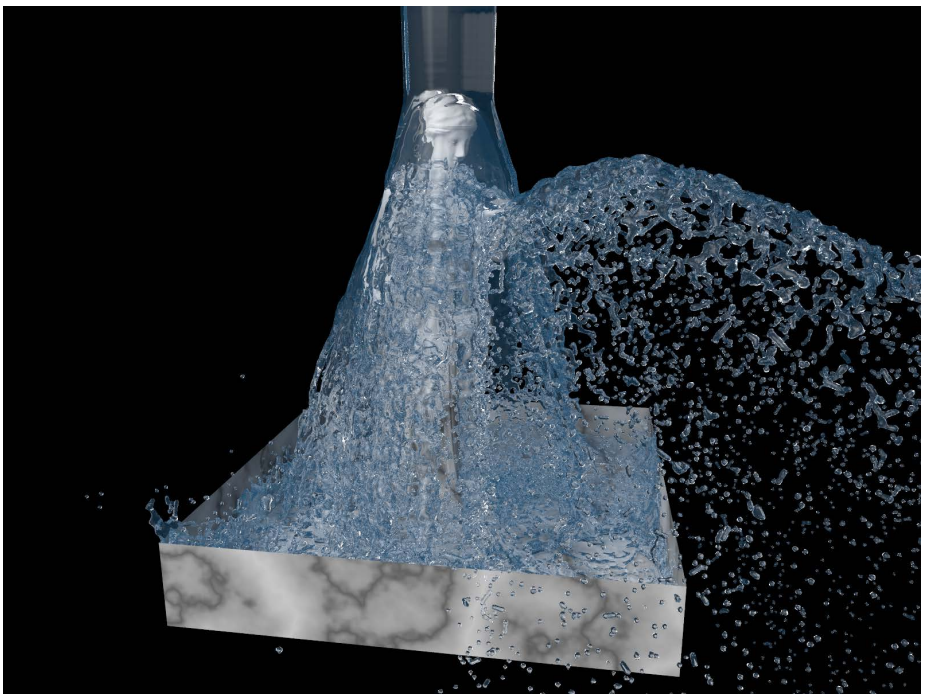


Figure 1.2 – The Hebe Scene. Surface reconstruction of the particle data set poured onto Hebe statue was performed using our parallel approach described in Chapter 4.

1.1 Contributions

Novel Narrow-Band Based Parallel Surface Reconstruction

Whether it is a desktop PC or a supercomputer, parallel computing on CPUs and GPUs have been becoming more popular with the help of increased computing power on modern architectures. Therefore, we present a narrow-band based surface reconstruction algorithm which is specifically designed for efficient parallelization on shared memory architectures. Our method scales nearly linearly on multi-core CPUs and runs up to fifty times faster on GPUs than the original scalar field computation methods. Besides, we present visual and performance comparisons between different scalar field computation techniques and we present advantages and disadvantages of various parameter setup.

Enhanced Surface Quality with post-processing

SPH fluids suffer from bumpy surfaces due to the irregular placement of particles. Smoothing such surfaces either requires sophisticated scalar field computations or resolved within a reasonable time with giving occasion to visible mass loss. Therefore, obtaining both smooth and detailed surfaces within an acceptable computation time is one of the most challenging tasks in the field. For this aim, we propose to combine existing scalar field approaches with two post-processing steps: decimation and subdivision. We address the bumpiness problem by applying a feature sensitive mesh decimation algorithm. This step allows the

decimated mesh to remain faithful to the original topology of the reconstructed surface, while it looks smooth in flat regions. However, the decimation step might sharpen the edges of some mesh features. So as to regain the smoothness of such sharp features, we apply subdivision to the decimated mesh. Our results demonstrate that in comparison to other approaches with comparable surface quality, our pipeline runs 15 to 20 times faster with up to 80% less memory and secondary storage consumption.

Curvature-Based Multi-Level Uniform Grids

In all contributions explained in this thesis, we use the Marching Cubes approach [LC87] for polygonizing SPH fluid surfaces. In order to preserve all of the surface details in high curvature regions and to prevent potential temporal coherence artifacts, the resolution of the underlying uniform Marching Cubes grid should be set up sufficiently high. However, this requirement unnecessarily increases the resolution in relatively flat regions where the surface can be constructed with lower resolutions without changing the quality. Accordingly, excessive number of triangles are generated, the memory consumption increases dramatically, and the performance decreases. In our work, we present a 3-level uniform grid structure which adapts its cells according to the curvature of the fluid surface. In contrast to widely-used octrees, we propose a simple to construct yet efficient hierarchical uniform grid structure. Mesh blocks from different resolution cells are seamlessly stitched by closing cracks with new triangles which establish only 0.15% to 0.6% of overall number of triangles in average. Experiments show that in contrast to the single level low resolution uniform grid approach, the presented method reconstructs fine details properly with a comparable performance; while it produces similar results with less number of triangles, up to four times better memory consumption and up to 60% better performance when compared to the single level high resolution uniform grid approach.

1.2 Publications

This thesis mainly covers the following publications that were published in peer-reviewed journals or peer-reviewed conference proceedings:

- G. Akinci, M. Ihmsen, N. Akinci, M. Teschner. Parallel Surface Reconstruction for Particle-based Fluids. Computer Graphics Forum, vol. 31, no. 6, pp. 1797-1809, 2012, doi: 10.1111/j.1467-8659.2012.02096.x.
- G. Akinci, N. Akinci, M. Ihmsen, M. Teschner. An Efficient Surface Reconstruction Pipeline for Particle-Based Fluids. Proc. VRIPHYS, Darmstadt, Germany, pp. 61-68, Dec. 6-7, 2012
- G. Akinci, N. Akinci, E. Oswald, M. Teschner. Adaptive Surface Reconstruction for SPH using 3-Level Uniform Grids. WSCG proceedings, pp.195-204, Union Agency, 2013

The following publications are also briefly covered in this thesis:

- M. Ihmsen, J. Bader, G. Akinci, M. Teschner. Animation of Air Bubbles with SPH. Int. Conf. on Computer Graphics Theory and Applications GRAPP 2011, pp. 225-234, March 5-7, 2011.

- M. Ihmsen, N. Akinci, G. Akinci, M. Teschner. Unified Spray, Foam and Bubbles for Particle-based Fluids. The Visual Computer , vol. 28, no. 6-8, pp. 669-677, 2012, doi: 10.1007/s00371-012-0697-9
- N. Akinci, M. Ihmsen, G. Akinci, B. Solenthaler, M. Teschner. Versatile Rigid-Fluid Coupling for Incompressible SPH. ACM Transactions on Graphics (Proc. SIGGRAPH 2012), vol. 31, no. 4, pp. 62:1-62:8, July 2012.
- N. Akinci, J. Cornelis, G. Akinci, M. Teschner. Coupling Elastic Solids with SPH Fluids. Journal of Computer Animation and Virtual Worlds (CAVW), vol. 24, no. 3-4, pp. 195-203, CASA 2013 Special Issue, 2013.
- N. Akinci, A. Dippel, G. Akinci, M. Teschner. Screen Space Foam Rendering. Journal of WSCG, Vol.21, No.03, pp.173-182, Union Agency, 2013
- N. Akinci, G. Akinci, M. Teschner. Versatile Surface Tension and Adhesion for SPH Fluids. ACM Transactions on Graphics (Proc. SIGGRAPH Asia 2013), Nov. 2013.

1.3 Overview

We cover the related previous field study in Chapter 2.

Chapter 3 gives a brief description of SPH. The particle concept is introduced in this chapter which establishes a basement for some of the presented variables throughout this thesis.

In Chapter 4, we present our novel parallel surface reconstruction technique for particle based fluids. Our method focuses on the narrow-band region where the fluid surface actually lies. Therefore, the computation complexity scales with the surface area instead of the fluid volume in contrast to traditional methods that employ the Marching Cubes approach.

Chapter 5 introduces an efficient surface reconstruction pipeline for particle based fluids. We show that enhanced surface quality can be achieved very efficiently by introducing two post-processing steps: decimation and subdivision. We address the bumpiness problem by applying a feature sensitive mesh decimation algorithm. The decimated mesh still remains faithful to the original topology of the reconstructed surface, while it looks smooth in flat regions. In order to regain the smoothness of sharp features that arise after the decimation step, we apply subdivision to the decimated mesh.

In Chapter 6, we present an adaptive surface reconstruction technique for SPH using 3-level uniform grids. The surface curvature plays a major role in this technique, since high curvature parts are sampled in high resolution cells and flat regions are sampled in low resolution cells. Cracks, which arise on transition cell faces are closed with newly generated triangles. Finally, we obtain a seamlessly stitched surface mesh with adaptive triangle sampling.

Finally, Chapter 7 concludes the thesis and discusses about future work.

2
Previous Work

2.1 Fluid Simulation

In the fluid simulation field, two main approaches are commonly used, which are called as Lagrangian and Eulerian models. In Eulerian models, quantities, e.g., velocity and pressure are measured at fixed points in space. One advantage of Eulerian methods is that, the time consuming neighborhood search is avoided due to the grid based nature of the model. Besides, the incompressibility can be enforced within a reasonable time. Other advantages that are exhibited by Eulerian models are relatively easy surface extraction and easy parallelization. However, this model falls down on creating small scale effects, e.g., splashes, droplets, small scale waves, which reduces the realism of the fluid flow dramatically. The final quality of the fluid flow highly depends on the resolution of the used staggered grid, and using low resolution grids produces visible mass loss. A detailed overview of the Eulerian model can be found in, for instance, [Sta99, EMF02, FOK05, LIGF06, MCP+09].

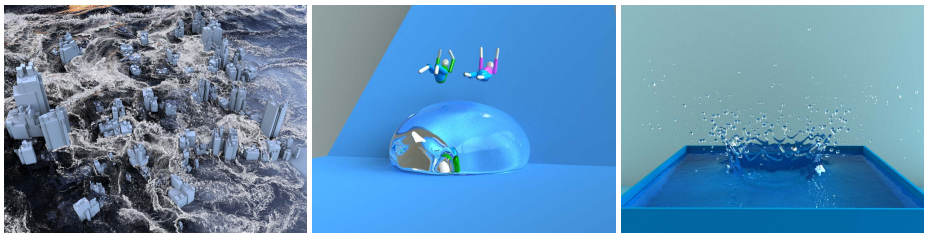
In Lagrangian models, however, the fluid is discretized by freely moving particles that carry those fluid quantities; and the model is able to produce small scale details easily. Another advantage is, the simulation domain does not have to be finite as in Eulerian models. However, Lagrangian models exhibit some disadvantages, for instance, difficulties in neighborhood search, surface extraction and parallelization. In this thesis, we use SPH for fluid simulations which is the most commonly used Lagrangian method in the literature.

Roy was the first to present Lagrangian fluid simulation in computer graphics field [Roy95]. Later, Müller et al. applied SPH to real time computer graphics in [MCG03].

In recent years, SPH has been applied to simulate various fluid phenomena, which include, for instance, multi-phase fluid-fluid interactions, e.g., [MSKG05, SP08], air bubbles and foam, e.g., [IBAT11, IAAT12, AAAT13], granular materials, e.g., [IWT12, IWT13], rivers, e.g., [KW06], and complex scenarios that use multi millions of particles, e.g., [IABT11, AIA+12, ICS+13]. In order to increase performance, adaptive particle schemes, e.g., [APKG07, ZSP08, YWH+09] and efficient data structures, e.g., [IABT11] are commonly used. Besides, so as to alleviate stability issues, different boundary-handling, e.g., [Mon94, HA06, MM97, IAGT10] and adaptive time stepping, e.g., [MK99, IAGT10] schemes are also proposed.

The realism of fluid animations can be improved by incorporating solid or deformable interactions. However, such kind of interactions are not straightfor-

(a) [AIA+12] **(b)** [ACAT13] **(c)** [IBAT11]



(d) [AAAT13] **(e)** [AAT13] **(f)** [AAT13]

Figure 2.1 – SPH fluids can be interacted with rigid objects (a) and deformables (b). Bubble generation (c), foam incorporation (d) and plausible surface tension effects (e, f) further increase the realism of the fluid animation. In all shown images, fluid surfaces have been reconstructed using our models.

ward to solve. Recent papers that address these issues include, e.g., [MST+04, CBP05, KAD+06, ODAF06, LD08b, OKR09, AIA+12, ACAT13]. Plausible surface tension effects are also useful to improve the realism of the fluid animation, e.g., [AAT13], see Figure 2.1.

There is a vast body of literature dedicated to this field and a comprehensive review is beyond the scope of this thesis. However, basic principles and other useful SPH related algorithms can be found, for instance, in [Mon05, MCG03].

Finally, semi-Lagrangian methods combine the best parts of both methods in the literature. For details, see, e.g., [FF01, LTKF08].

2.2 Surface Extraction

2.2.1 Polygonization

In this thesis, we polygonize the fluid surfaces using the Marching Cubes method of Lorensen and Cline [LC87]. Marching Cubes is a surface triangulation algorithm used for viewing 3D point clouds without any connectivity information. The method relies on the voxelized space around the data-of-interest. For every voxel, eight scalar values are computed at respective corners. The scalar value at a voxel corner carries the inside-outside information. Using this information, different triangle configurations are created which lead to the final surface of the data. Since there are eight vertices on a voxel and two possible states (inside or outside) for each of them, there can be 256 different configurations with which a surface intersects the voxel. This number can easily be reduced to 15

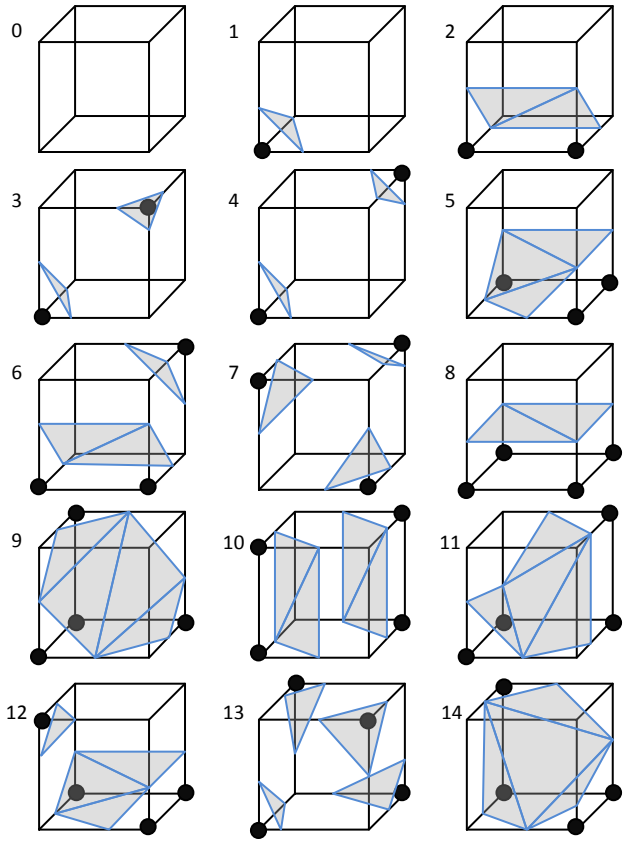


Figure 2.2 – Marching Cubes voxel configurations.

configurations by using the symmetry of the voxel (see Figure 2.2).

Although there are variants of Marching Cubes, e.g., marching tetrahedra [TPG98, CP98] or marching triangles [HI97], Marching Cubes is the commonly preferred technique due to its simplicity and efficiency. Delaunay triangulation is another popular technique that is widely employed for visualization purposes [Del34, Slo87, SVK$^+$, Mau02].

After polygonization, realistic rendering of surface meshes can be produced efficiently by using modern ray tracers, e.g., POV-Ray [Pov10] or Mentalray [NVI11] (see Figure 2.3).

2.2.2 Scalar Field Construction

One of the main focuses of this thesis is the efficiency of scalar field construction techniques to be incorporated with the commonly used Marching Cubes method. 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

Figure 2.3 – The Wine scene. For simulation details, please see [IABT11]. The surface reconstruction has been performed using our model (Chapter 4), and the realistic rendering was performed in POV-Ray using radiosity and high dynamic range imaging (HDRI). The image is taken from [IABT11].

sets with sharp features. Later, Müller et al. suggested the use of the weighted density information of particles in [MCG03]. 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. This problem is also addressed in [OCR11]. 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. 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. Inspired by [Wil08], Bhattacharya et al. [BGB11] proposed a level set method, where the fluid surface that lies between inner and outer surface approximations is processed by Laplacian smoothing. The main downside of this method is, the performed surface approximations and smoothing steps cause the surface to cover the underlying particles coarsely, which gives occasion to lose the details of the particle set.

2.2.3 Efficient Algorithms

One of the issues of the surface reconstruction approaches is scaling the computational region with the volume of the data instead of the surface area which increases the computational complexity cubically. In recent years, different narrow band techniques have been proposed in order to scale both the computational amount and the memory consumption with the surface area. Müller et al. [MCG03] suggested to visualize the free surface by first identifying the surface cells and later applying the Marching Cubes on the found cells. However, no solution was given for the possible double layer problem. 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 those 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. Recently, we proposed a parallel method where the scalar field is efficiently constructed only in the narrow-band [AIAT12].

Among all polygonization techniques, performance improvement of the Marching Cubes approach with parallelization is probably the most commonly investigated one with various proposed techniques, e.g., [Mac92, YC94, SDC09, CZ10, SEL11]. GPU-based applications of marching tetrahedra, e.g., [CKK+12] and parallel algorithms for Delaunay triangulation, e.g., [LPP97, KKA05, Che11] have also been becoming popular.

GPU-based methods is one of the main focuses of researchers with recent advances in the hardware technology. In this context, surface splatting is one of the most popular and widely used surface visualization 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. Projections of the overlapping particles 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.

Further 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 re-sampled 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 presented methods, [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 re-sampling 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.

2.2.4 Surface Post-Processing

Post-processing surface meshes is a practical way of achieving desired effects on surfaces easily. There are various post-processing techniques that can be applied to fluid surfaces, however, we will briefly discuss two of the most commonly used methods: decimation and subdivision.

Within the context of mesh decimation, there exist various approaches to mention, e.g., vertex decimation technique by Schroeder et al. [SZL92], vertex clustering technique by Rossignac and Borrel [RB93], and a rich research on edge contraction techniques, e.g., [HDD$^+$93, Gue95, GH97]. Among the edge contraction techniques, the method of Garland and Heckbert [GH97] has particularly gained attention where their method associates error quadrics to mesh vertices, and computes edge contraction costs accordingly. Error quadric of each vertex is computed as the sum of the squared distances of the vertex to the planes of triangles that meet at the vertex. This method has been further improved in [GH98, Hop99] for associated vertex attributes. More recently, Du et al. [DYK06] proposed a new technique which is again based on quadric error metrics but particularly focuses on feature preservation on high curvature regions after a heavy simplification process.

Doo-Sabin [DS78], Catmull-Clark [CC78], Loop [Loo87] and butterfly [DLG90, ZSS96] are among the most commonly employed subdivision techniques in the field. Doo-Sabin and Catmull-Clark are generalizations of of bi-quadratic and bi-cubic B-splines, respectively, that are applied to polygonal meshes, while butterfly and loop subdivision schemes are specifically designed for triangle meshes. A fair comparison of these approaches depend on many variables, e.g., the surface shape, resultant mesh's triangle uniformity, smoothness, type of the mesh (e.g., triangle mesh or quad mesh) or computational efficiency. In this thesis, we use Loop's subdivision scheme when required since it is easy to implement, and it produces smooth surfaces in only few subdivision steps for our triangulated fluid surfaces.

There are only few researchers who apply decimation or subdivision on fluid surfaces for improving the mesh quality [TFK$^+$03, BBB10], or suggests to use it if necessary [Wil08]. Our recent work [AAIT12] proposes to use both techniques in order to reduce bumpy surfaces by decimation and obtain smooth surfaces with subdivision (see Figure 2.4).

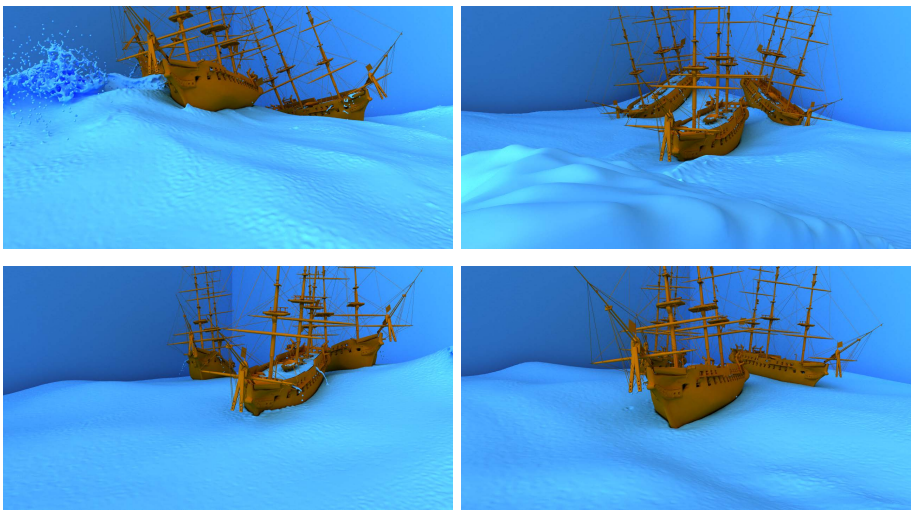


Figure 2.4 – The Three Ships scene has been simulated with more than ten million particles. The surface reconstruction has been performed using our pipeline [AAIT12].

2.2.5 Adaptive Spatial Data Structures

The most straightforward way of generating surface meshes is using uniform grids, since uniform grids reduce computational effort in contrast to sophisticated adaptive data structures. However, the goal of adaptive grid techniques is to save on memory and secondary storage consumption by concentrating the computational effort on detailed surface parts.

One way to generate an adaptive mesh is to use octree structure which was addressed by many researchers, e.g., [WVG92, SFYC96, WKE99, VT01, LY04, JU06, MS10]. An octree is multi-dimensional tree whose internal nodes host exactly eight children. While being very efficient in terms of memory consumption and storage reduction, the construction of octrees is time consuming. Dynamic update of octrees, e.g., shrinking or enlarging the grid and updating child cells, in particular can be time consuming, as one can probably end up with total rebuild of the octree; which makes them less feasible for dynamic scenes. Furthermore, random access to octree cells usually take logarithmic time, which causes additional overhead. In addition, octrees produce many different resolution cells which means that two leaf cells may differ more than one level. In such a case, crack handling gets difficult. A similar discussion can be found in [Bri03] where Bridson proposes an alternative grid-based method that focuses on the narrow-band region by using only one level of detail.

Using kd-trees is an alternative way of generating adaptive meshes. A kd-tree is a multi-dimensional binary tree with arbitrary axis-aligned splitting planes which is in contrast to octrees. General opininon in space-partitioning community is that kd-trees are typically superior than octrees in terms of construction and query times, however, cannot perform as well as octrees for insertion/deletion operations. Within the context of surface extraction, kd-trees have been used by Livnat et al. [LSJ96] and more recently by Gress and Klein [GK03].

One of the bottlenecks of generating meshes using adaptive structures is that if levels of two neighboring cells differ, cracks arise in the corresponding

transition faces. There exist different approaches which address this problem. Using simple crack patching [SFYC96, VKS+04], points that reside on the high resolution edge of one cell are forced to project on the low resolution edge of the neighboring cell. However, this technique produces T-vertices which may lead to visual artifacts during rendering. Filling cracks with new triangles is another popular method for handling cracks. Westermann et al. [WKE99] proposed a method where cracks are fixed by replacing coarse triangles by fans of triangles. Ju and Udeshi [JU06] prevented cracks by adding new polygons using a hybrid method that uses the Marching Cubes and dual contouring. It is stated in [JU06] that the mesh size can get too large by using this approach after newly added triangles. Later, Lengyel [Len10] presented transition cells method where new cells are added in between two different resolution cells for generating new triangles. The newly generated transition cells have to be checked for many cases which is a time consuming process.

Although conventional adaptive structures reduce the storage requirements efficiently, more effective results can be obtained by incorporating hashing. However, this is a challenging task and only few approaches have been proposed so far which address either hashing of octrees [Sig06] or hashing of multi-level grids for ray-tracing, e.g., [LD08a, CPJ10].

To the best of our knowledge, the efficient implementation of adaptive structures for particle-based fluids is limited with the work of Zhou et al. [ZGHG11] which uses octrees, and more recently our work [AAET13] that proposes a three level multi resolution uniform grid.

3

Smoothed Particle Hydrodynamics

SPH was first introduced for astrophysical problems in 1977 by Gingold and Monaghan [GM77], and Lucy [Luc77]. Later, Müller et al. [MCG03] introduced SPH to real time computer graphics area by aiming interactive fluid-based applications.

In SPH, a fluid flow is represented using discrete particle sets. Arbitrary field variable A of a particle is computed by interpolating the values of the same variable that are hold by other particles in the region of interest. This region is commonly called as smoothing area which is described over a smoothing length h, where$h$ is generally chosen as twice the initial distance of adjacent particles. Any particle that lies in the smoothing area can be called as neighbor particle. Contributions of neighbor particles are governed by kernel functions which is commonly denoted as W. A scalar quantity A of an arbitrary particle i is computed as:

$$A_i = \sum_j \frac{A_j}{\rho_j} m_j W_{ij}, \tag{3.1}$$

where W_{ij} is a shorthand for $W(\mathbf{x}_i - \mathbf{x}_j, h)$, j defines neighbor particles, ρ and m are particle density and mass, and $\mathbf{x}$ stands for the particle position. The accuracy of the result is highly dependent to the chosen kernel function. In our implementations, we generally prefer to apply a cubic B-spline function proposed in [ML85] which can be written as:

$$W_{CBS}(q,h) = \sigma \begin{cases} \frac{6q^3}{h^3} - \frac{6q^2}{h^2} + 1 & 0 \le q \le 0.5h \\ 2(1 - \frac{q}{h})^3 & 0.5h < q \le h \\ 0 & else \end{cases}. \tag{3.2}$$

Here, $q = |\mathbf{x}_i - \mathbf{x}_j|$, and σ is a variable dependent to the dimension of the simulation domain which is $1/(\pi^{3/2}h^3)$ in 3D. According to (3.1) and (3.2), only the contribution of those particles, whose distance to the particle of interest is smaller than h are considered (see Figure 3.1). If required, the spatial derivatives of (3.1) can be computed simply by taking the derivative of the kernel function W.

Navier-Stokes Equations

In 19th century, Claude-Louis Navier and George Gabriel Stokes described the motion of fluids by introducing a set of partial differential equations, which

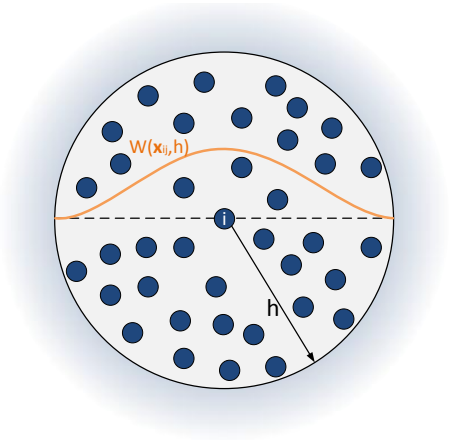


Figure 3.1 – SPH kernel.

are later called as Navier-Stokes equations. By the use of these functions, the momentum equation of incompressible Lagrangian fluids is described as:

$$\rho \frac{d\mathbf{v}}{dt} = -\nabla p + \mu \nabla^2 \mathbf{v} + \mathbf{f}_{ext}, \tag{3.3}$$

which simply represents the fluid motion using Newton's second law, and the stress in the fluid as a sum of viscous and pressure terms. In (3.3), $\mathbf{v}$ is the velocity of the flow,p represents the pressure, μ is the viscosity coefficient and $\mathbf{f}_{ext}$ represents external forces, e.g., gravity and boundary forces. Those external forces can be solved numerically by using SPH interpolation.

Pressure Force

According to [Mon05], the pressure force for an arbitrary particle i is computed as:

$$\mathbf{F}_i^p = -m_a \sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W_{ij}, \tag{3.4}$$

where the pressure p at each particle location is calculated either by using ideal gas equation [MCG03]:

$$p = k(\rho - \rho_0) \tag{3.5}$$

or by using an alternative form of a state equation [Mon94, BT07]:

$$p = \frac{\rho_0 k}{\gamma} \left(\left(\frac{\rho}{\rho_0} \right)^{\gamma} - 1 \right). \tag{3.6}$$

The density of a particle can be computed easily by substituting ρ into (3.1) and resulted as:

$$\rho_i = \sum_j m_j W_{ij}.$$

In (3.5) and (3.6), ρ_0 stands for the rest density of the fluid and k is the stiffness parameter to determine the flow incompressibility. In (3.6), γ is a user defined parameter which is generally chosen as 7.

Viscosity Force

In order to improve the stability of the simulation, another force, namely viscosity force is used which diminishes the intensity of relative velocities of particles. When we apply SPH to the viscosity term $\mu\nabla^2\mathbf{v}$, we obtain

$$\mathbf{F}_i^v = \mu \sum_j m_j \left(\frac{\mathbf{v}_j}{\rho_j} \right) \nabla^2 W_{ij}. \tag{3.7}$$

3.7 relies on the second derivative of the kernel function, which is sensitive to particle disorder. Therefore, in our experiments, we use an artificial viscosity force formulation proposed in [MG83] which is written as:

$$\mathbf{F}_i^v = \begin{cases} -m_i \sum m_j \Pi_{ij} \nabla_i W_{ij} & \mathbf{v}_{ij} \cdot \mathbf{x}_{ij} < 0 \\ 0 & \mathbf{v}_{ij} \cdot \mathbf{x}_{ij} \geq 0 \end{cases}, \tag{3.8}$$

where

$$\Pi_{ij} = -\nu \left(\frac{\mathbf{v}_{ij} \cdot \mathbf{x}_{ij}}{|\mathbf{x}_{ij}|^2 + \epsilon h^2} \right), \tag{3.9}$$

and the viscous factor is:

$$\nu = \frac{\alpha \bar{h_{ij}} c_s}{\bar{\rho}_{ij}}. \tag{3.10}$$

Here, α and c_s describe the viscosity constant and the speed of sound, respectively.

Additional Forces

Although not modeled by Navier Stokes equations, the surface tension force exists in many forms of fluids. Surface tension emerges as a result of cohesive forces between liquid molecules and it is responsible of diverse liquid behaviors. We use the recent method of Akinci et al. [AAT13], which allows to handle large surface tension forces. This surface tension force is presented as:

$$\mathbf{F}_{i \leftarrow j}^{st} = K_{ij} \left(\mathbf{F}_{i \leftarrow j}^{cohesion} + \mathbf{F}_{i \leftarrow j}^{curvature} \right)$$

where K_{ij} is a symmetrized correction factor described using density information as:

$$K_{ij} = \frac{2\rho_0}{\rho_i + \rho_j},$$

and; $\mathbf{F}_{i \leftarrow j}^{cohesion}$ and $\mathbf{F}_{i \leftarrow j}^{curvature}$ stand for fully symmetrized cohesion and curvature forces that are computed as follows.

$$\mathbf{F}_{i \leftarrow j}^{cohesion} = -\gamma m_i m_j C(|\mathbf{x}_i - \mathbf{x}_j|) \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|}$$

with i and j being neighboring particles, γ is the user defined surface tension constant and C is a spline function described as:

$$C(r) = \frac{32}{\pi h^9} \begin{cases} (h-r)^3 r^3 & 2r > h \wedge r \leq h \\ 2(h-r)^3 r^3 - \frac{h^6}{64} & r > 0 \wedge 2r \leq h. \\ 0 & else \end{cases}$$

$$\mathbf{F}_{i\leftarrow j}^{curvature} = -\gamma m_i(\mathbf{n}_i - \mathbf{n}_j),$$

with $\mathbf{n}$ being the particle normal that is computed as

$$\mathbf{n}_i = h\sum_j \frac{m_j}{\rho_j}\nabla W_{ij}.$$

In addition to surface tension force, we apply adhesion force to improve the realism of the animation. For this aim, we use the method of Akinci et al. [AAT13], which together with the above defined surface tension force, allows diverse wetting conditions and plausible rigid-fluid interactions. This force can be written as:

$$\mathbf{F}_{i\leftarrow k}^{adhesion} = -\beta m_i \Psi_{b_k} A(|\mathbf{x}_i - \mathbf{x}_k|)\frac{\mathbf{x}_i - \mathbf{x}_k}{|\mathbf{x}_i - \mathbf{x}_k|},$$

where k denotes solid boundary particles, β is the adhesion coefficient and A is a spline function created as:

$$A(r) = \frac{0.007}{h^{3.25}}\begin{cases}\sqrt[4]{-\frac{4r^2}{h} + 6r - 2h} & 2r > h \wedge r \leq h \\ 0 & else\end{cases}.$$

There is a vast literature about SPH fluid simulations which propose novel technical contributions to the field. However, a thorough discussion of SPH-based fluid simulations is beyond the scope of this thesis. The work of Monaghan et al. [Mon05] and Müller et al. [MCG03] are recommended for reference or further study in SPH.

4

Narrow-Band Based Parallel Surface Reconstruction

4.1 Introduction

For SPH fluids, the reconstruction of detailed, smooth, and artifact-free surfaces is generally a bottleneck due to computational complexity and memory requirements. This situation becomes more pronounced as the complexity of the simulation gets higher.

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 [LC87] or Marching Tiles [Wil08], are among the widely used methods in this field.

In the context of Marching Cubes, 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 catch fine details, small cell sizes are required, which in turn scales the computation time cubically (see Figure 4.1).

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. Scalar values at grid points are generally computed by extensively employed smoothing techniques [ZB05, SSP07, APKG07, YT10, OCR11] which consider particles within an appropriate influence region. Similar to the Marching Cubes 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 Marching Cubes 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.

Together with recent advances in parallel architectures, parallel computing techniques are brought to bear more frequently for performance improvement. However, fluid surface reconstruction techniques are generally performed serial due to high data dependency. In this thesis, we propose a novel parallel technique 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. Our method scales nearly linearly on multi-core CPUs and runs up

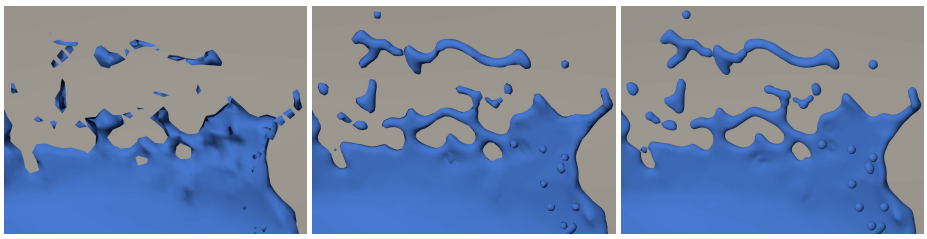


Figure 4.1 – Close-up view of the reconstructed surface for the Corner Breaking Dam (CBD) scene (1.7 million particles) with three different Marching Cubes cell sizes: left $2r$, middle r, right $r/2$, with r being the equilibrium distance of the SPH particles. The grid sizes are 123×107×123, 245×214×245 and 485×409×486, 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.

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 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 Marching Cubes 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, OCR11]. In our experiments, however, [SSP07] is employed as it yields very smooth and artifact-free results, while being comparatively efficient.

4.2 Algorithm

4.2.1 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]. Bhattacharya et al. [BGB11] proposed a level set method, where the main downside of this method is, the performed surface approximations and smoothing steps cause the surface to cover the underlying particles coarsely, which loses the details of the particle set. Therefore, in this thesis, 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:

$$\phi(\mathbf{x}) = |\mathbf{x} - \bar{\mathbf{x}}| - \bar{r}f \tag{4.1}$$

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 since we use a uniform particle size in our experiments.. The factor f is computed as:

$$f = \begin{cases} 1 & EV_{max} < t_{low} \\ \gamma^3 - 3\gamma^2 + 3\gamma & otherwise \end{cases} \tag{4.2}$$

with

$$\gamma = \frac{t_{high} - EV_{max}}{t_{high} - t_{low}} \tag{4.3}$$

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:

$$\bar{\mathbf{x}} = \frac{\sum_j \mathbf{x}_j k(|\mathbf{x} - \mathbf{x}_j|/R)}{\sum_j k(\mathbf{x} - \mathbf{x}_j|/R)}, \tag{4.4}$$

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:

$$k(s) = \max(0, (1 - s^2)^3). \tag{4.5}$$

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. In other words, 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.

4.2.2 Optimized Surface Reconstruction

The computation time for extracting smooth surfaces is mainly influenced by the resolution of the Marching Cubes grid and the smoothing radius R; and

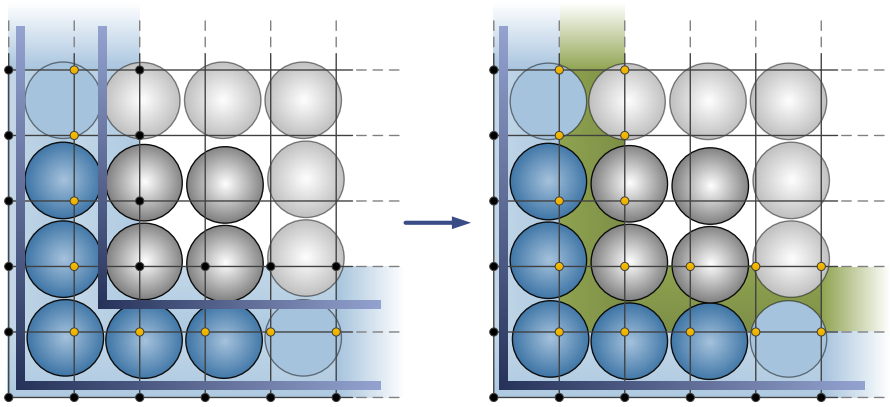


Figure 4.2 – The double layer problem. Surface vertices are shown by small dots (yellow ones with $\phi(\mathbf{x}) \leq 0$ and black ones with $\phi(\mathbf{x}) > 0$). On the left, the scalar field is constructed using only surface particles (blue disks), causing the inner surface vertices (inner black 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.

achieving high quality surfaces is possible at the expense of performance. One of the main causes of this issue is computing the scalar field over the volume of the fluid instead of concentrating on the surface area. In this section, we present an implementation which performs the scalar field computation only for surface vertices. 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.

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 smoothed color field value of a particle at position $\mathbf{x}$ is computed as:

$$cf(\mathbf{x}) = \sum_j m_j \frac{W_{ij}}{\rho_j} \tag{4.6}$$

with W and h are the kernel function and smoothing length of the SPH simulation, j represents neighboring particles and ρ represents the density. 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 neighbors as surface particles.

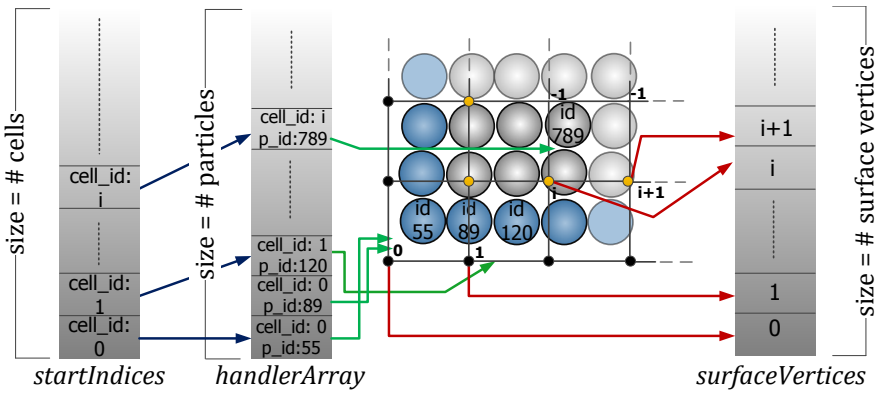


Figure 4.3 – Illustration of the different arrays used during the scalar field computation stage. Blue and gray disks represent surface and inner particles, respectively. Each particle has a handler (particle id, cell id pair) that is kept in the *handlerArray* in a cell-based sorted order. The cells 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.

Step-by-Step Scalar Field Computation Over Narrow Band Region

This phase 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. Details of these stages are described in this section, where parallelization related implementation details are discussed in Sec. 4.2.3.

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 Marching Cubes 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 Figure 4.2, left.

In order to prevent bumpiness, we suggest to consider not only the surface, 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:

$$cell\_id = iM\left[\mathbf{cc}.x\right] | (iM\left[\mathbf{cc}.y\right] \ll 1) | (iM\left[\mathbf{cc}.z\right] \ll 2) \tag{4.7}$$

where

$$\mathbf{cc} = (\lfloor \mathbf{x}.x \rfloor, \lfloor \mathbf{x}.y \rfloor, \lfloor \mathbf{x}.z \rfloor), \tag{4.8}$$

$$\mathbf{x} = \frac{(\mathbf{p} - \mathbf{g})}{s} \tag{4.9}$$

with $\mathbf{p}$ 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 Figure 4.3).

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 radii smaller than $4r$ result in bumpy surfaces (see Figure 4.4). 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 (see Equation (4.1)).

Triangulation

After computing the scalar values for Marching Cubes 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 Marching Cubes 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 (see Figure 4.5, left). 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

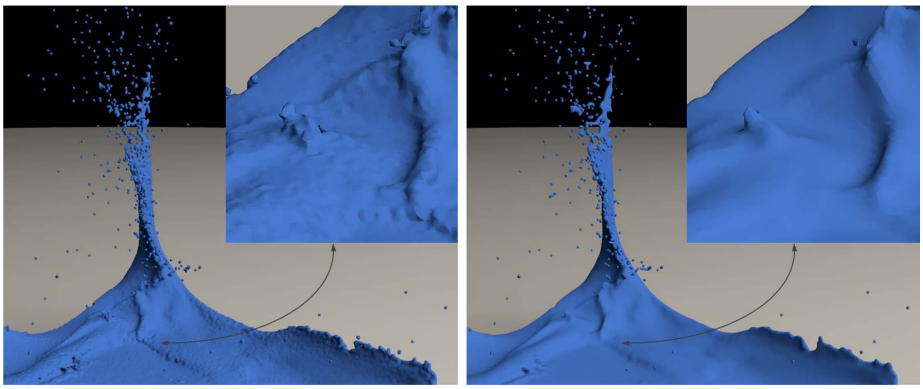


Figure 4.4 – Surface reconstruction of the CBD scene with $2r$ (left) and $4r$ (right) influence radii. Surface reconstruction times including the scalar field computation and triangulation are 7.5 and 60 seconds average per-frame on our 6-core CPU, respectively.

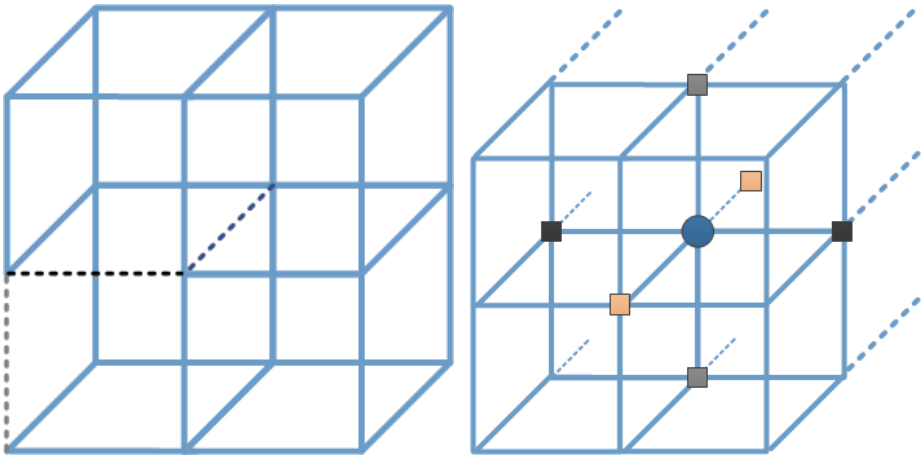


Figure 4.5 – (a) Blue, black and gray dashed edges are shared by 4, 2 or 1 cells, respectively, which causes redundant on-edge computations. (b) Scalar values of the neighboring left-right (black squares), up-down (gray squares) and near-far (cream squares) grid vertices are used to compute the normal of an arbitrary grid vertex (blue circle in the middle).

the vertex and initialized as -1 as illustrated in Figure 4.6. 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 which is responsible of keeping 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. We compute the normal $\mathbf{n}_{vertex} = (\mathbf{n}_x, \mathbf{n}_y, \mathbf{n}_z)$ of arbitrary grid vertex as follows:

$$\mathbf{n}_x = \frac{val_{left} - val_{right}}{2 * cs} \tag{4.10}$$

$$\mathbf{n}_y = \frac{val_{up} - val_{down}}{2 * cs}, \tag{4.11}$$

$$\mathbf{n}_z = \frac{val_{far} - val_{near}}{2 * cs} \tag{4.12}$$

where val_{left}, val_{right}, val_{up}, val_{down}, val_{far} and val_{near} stand for the scalar values of the neighboring grid vertices (see Figure 4.5, right) and cs represents the Marching Cubes grid cell size. Later, we normalize all vertex normals. One should note that some vertices fall outside the scalar field computation area, therefore, has no scalar value. In such a case, we use a pre-defined scalar value for those vertices which is 2^*r in our experiments.

Finally, it remains to determine the triangles using the Marching Cubes 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 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 (see Figure 4.7). 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. 4.3).

Our optimized surface reconstruction method, including the scalar field computation and the triangulation, is summarized in Algorithm 4.1.

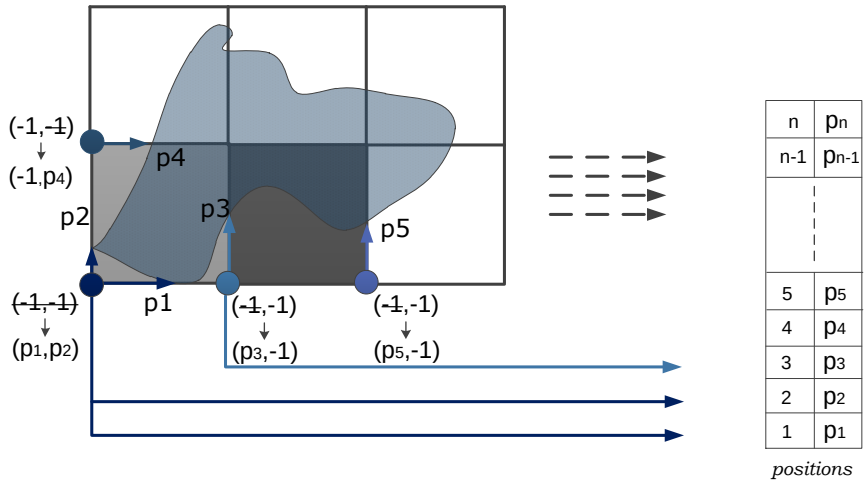


Figure 4.6 – A 2D illustration of the optimized edge intersection point computation.

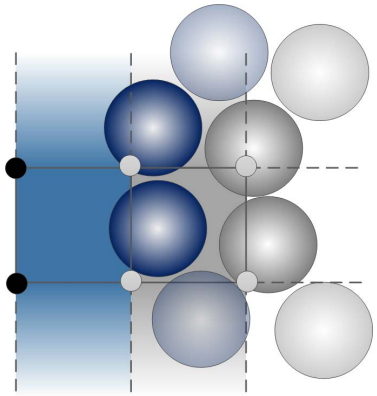


Figure 4.7 – Extraction of the on-surface cells. Small points illustrate the surface vertices. Values of gray points are smaller than zero, since they are in very close proximity of surface particles (blue disks) and also affected by inner particles (gray disks). In contrast, values of black points are larger than zero due to their distance to surface particles and since they are less or never affected by inner particles. Both blue and gray cells are passed to the triangulation stage. However, only on-surface cells (blue cells) are triangulated.

Algorithm 4.1 Optimized Surface Reconstruction

```
1:  foreach particle do
2:      compute color field quantity;
3:      mark if surface particle;
4:  foreach surface particle do
5:      compute AABB that spans 2r distance on each axis;
6:      mark each vertex that lie in the AABB;
7:  foreach particle do
8:      compute cell id;
9:      create a handler;
10: foreach surface vertex do
11:     compute AABB that spans 4r distance on each axis;
12:     find cells in AABB;
13:     foreach cell do
14:         find particles inside;
15:         add up each particle's contribution;
16:     compute scalar value of the vertex;
17: foreach surface vertex do
18:     create a cell;
19:     if all eight vertices are surface vertices then
20:         pass the cell to triangulation stage;
21:         if cell is on-surface then
22:             triangulate the cell;
```

4.2.3 Implementation

Parallelization

The 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

			cell size=r		
scene	#particles	$\frac{\#\text{particles}_{\text{surf}}}{\#\text{particles}_{\text{total}}}$	$\frac{\#\text{vertices}_{\text{surf}}}{\#\text{vertices}_{\text{total}}}$	#triangles	grid res.
Drop	360k	0.12	0.075	87k	160×73×147
CBD	1.7m	0.13	0.1	420k	243×204×243
Hemispheres	up to 2m	0.17	0.18	782k	333×63×438
Hebe	up to 3.2m	0.09	0.08	903k	289×314×259

Table 4.1 – General information of the low resolution versions of presented scenes. The data are given as average per frame.

			cell size=$r/2$		
scene	#particles	$\frac{\#\text{particles}_{\text{surf}}}{\#\text{particles}_{\text{total}}}$	$\frac{\#\text{vertices}_{\text{surf}}}{\#\text{vertices}_{\text{total}}}$	#triangles	grid res.
Drop	360k	0.12	0.065	351k	320×146×293
CBD	1.7m	0.13	0.09	1.7m	485×409×486
Hemispheres	up to 2m	0.17	0.16	3.1m	667×127×876
Hebe	up to 3.2m	0.09	0.07	6.5m	578×628×518

Table 4.2 – General information of the high resolution versions of presented scenes. The data are given as average per frame.

traversing along all surface vertices in parallel 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. 4.3) .

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

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	400
Hemispheres	63	520	168	520
Hebe	150	850	300	854

Table 4.3 – Averaged per frame timings of simulation, surface particle extraction, opaque and transparent rendering.

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 4.4 – 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. In the table, t_{sf} and t_{tri} stand for the scalar field computation and the triangulation time, respectively.

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.

Simulation and Hardware

For the simulations, a variant of SPH (predictive-corrective incompressible SPH [SP09]) with adaptive time-stepping [IAGT10] is used. 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 were rendered using POV-Ray [Pov10].

4.3 Results

In order to demonstrate the utility of our method, we applied it to four different scenarios: Hebe, Drop, Corner Breaking Dam (CBD) and Hemispheres with two different cell sizes. For all scenes, the support radius R is $4r$. A detailed analysis of the presented examples are given in Tables 4.1, 4.2 and 4.3.

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 the first example, we analyze the performance of the popular corner breaking dam (CBD) scenario (see Figure 4.9). 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 1.2 and

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 4.5 – 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 $r/2$.

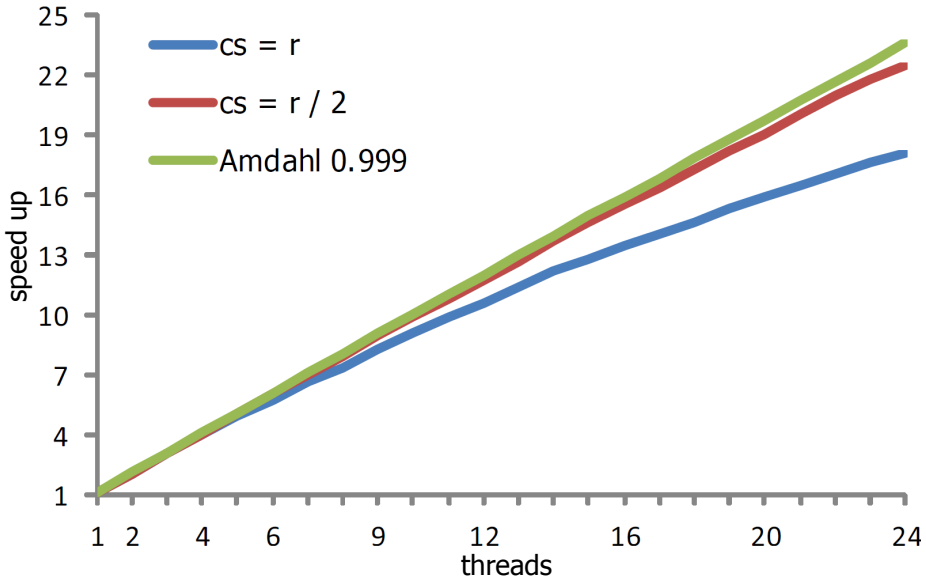


Figure 4.8 – The scaling of the parallel scalar field computation method for the CBD scene. The method scales well with respect to the number of utilized threads. It yields a better scaling for smaller cell sizes (cs) because of the domination of parallelized parts over the serial part.

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 4.4 and 4.5 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 Figure 4.8. This figure shows that the method scales better for smaller cell sizes. The reason for this is that in higher resolution

scene	$t_{\text{sf-CPU 1 thread}}$	$t_{\text{sf-CPU 6 threads}}$	$t_{\text{sf-GPU}}$	t_{tri}
Drop	3.3 sec	0.6 sec	0.37 sec	0.05 sec
CBD	14.4 sec	2.6 sec	1.0 sec	0.18 sec
Hemispheres	21.1 sec	3.8 sec	1.5 sec	0.27 sec
Hebe	36.6 sec	7.1 sec	2.3 sec	0.62 sec

Table 4.6 – Performance of the presented scenes for scalar field computation (t_{sf}) and triangulation (t_{tri}) using a cell size equal to r. All data are average-per-frame.

scene	mem_{CPU}[MB]	mem_{GPU}[MB]
Drop	271	CPU(265)+GPU(85)
CBD	1390	CPU(1201)+GPU(410)
Hemispheres	1143	CPU(980)+GPU(302)
Hebe	2575	CPU(2224)+GPU(591)

Table 4.7 – Memory consumption of the presented scenes for scalar field computation (t_{sf}) and triangulation (t_{tri}) using a cell size equal to r. All data are average-per-frame.

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 Table 4.4 and 4.5). 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.

The next example is the Hemispheres scene where a fluid with up to 2 million particles flows onto two half spheres (see Figure 4.10). 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 Figure 4.10). This is generally challenging to achieve for millions of particles without using the proposed optimization techniques. The analysis in Tables 4.7 and 4.9 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 Marching Cubes grid.

The third example is the Drop scene where a water drop falls into a filled pool (see Figure 4.11). 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 sec on the GPU for cell sizes of r and $r/2$, respectively. The Drop

scene	$t_{sf\text{-}CPU\ 1\ thread}$	$t_{sf\text{-}CPU\ 6\ threads}$	$t_{sf\text{-}GPU}$	t_{tri}
Drop	61.1 sec	10.4 sec	3.5 sec	0.28 sec
CBD	343.4 sec	58.5 sec	14.1 sec	1.13 sec
Hemispheres	617.3 sec	105.4 sec	21.9 sec	2.92 sec
Hebe	1640.8 sec	227.1 sec	32.3 sec	6.86 sec

Table 4.8 – Performance of the presented scenes for scalar field computation (t_{sf}) and triangulation (t_{tri}) using a cell size equal to $r/2$. All data are average-per-frame.

scene	mem_{CPU}[MB]	mem_{GPU}[MB]
Drop	415	CPU(389)+GPU(170)
CBD	4076	CPU(3888)+GPU(2130)
Hemispheres	3356	CPU(2760)+GPU(1322)
Hebe	7532	CPU(6145)+GPU(3281)

Table 4.9 – Memory consumption of the presented scenes for scalar field computation (t_{sf}) and triangulation (t_{tri}) using a cell size equal to $r/2$. All data are average-per-frame.

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 Figure 4.12). For this scene, a surface for 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 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 186×264×210, 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 Table 4.10). In contrast, for [YT10] the speed-up is approximately 20 (see Table 4.11), which shows that the performance of [SSP07] is 5 times faster than [YT10] after applying our algorithm. 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,

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 4.10 – 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 4.11 – 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 $r/2$.

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 neighborhood 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 Figure 4.13.

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.

4.4 Discussion and Future Work

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 Marching Cubes 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. We propose to solve this issue by extracting isolated particles in a preprocessing step and handling them directly in the renderer as spheres, i.e.„ their ideal shape (see Chapter 5).

Furthermore, instead of sticking to uniform grids, we propose to utilize an adaptive grid structure where we can construct higher resolution grids in high-

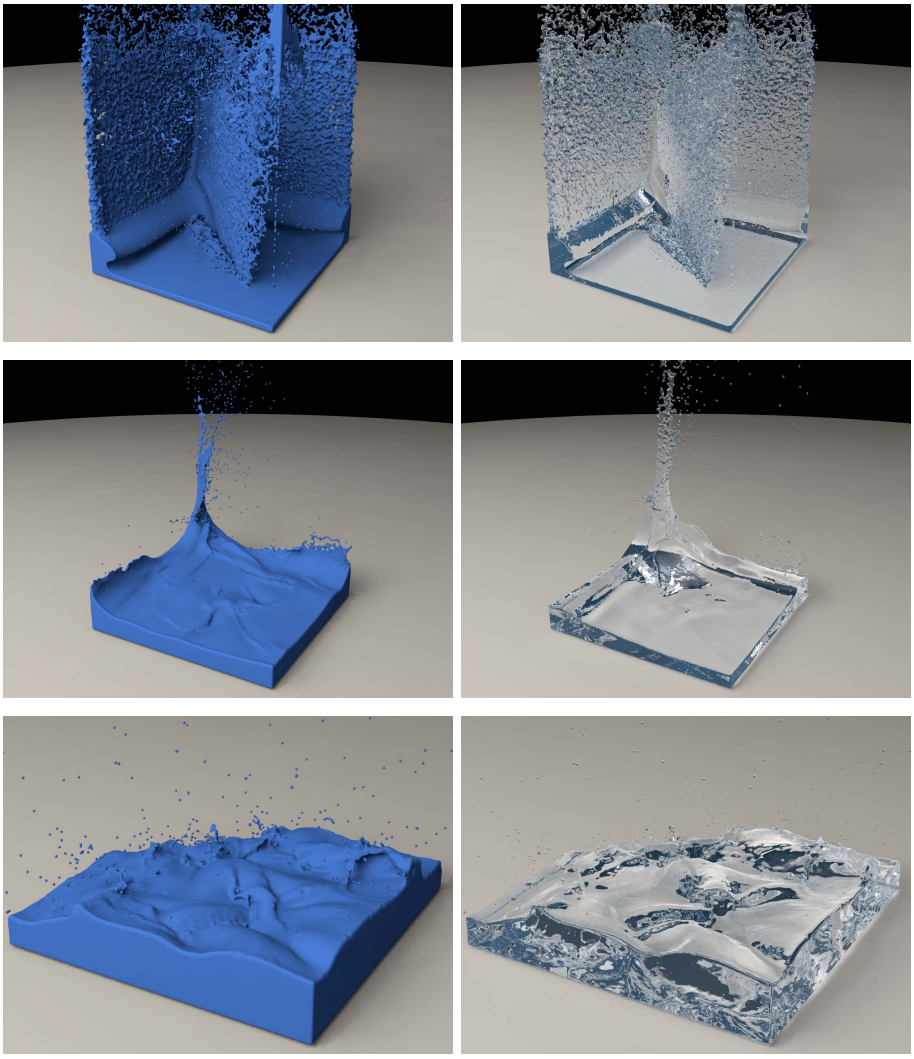


Figure 4.9 – CBD scene with 1.7 million particles. Note that thin sheets are well preserved due to a small cell size equal to $r/2$. The bottom image shows the surface in high curvature regions. Even very fine details, e.g., impact points of tiny droplets, are catched in this frame.

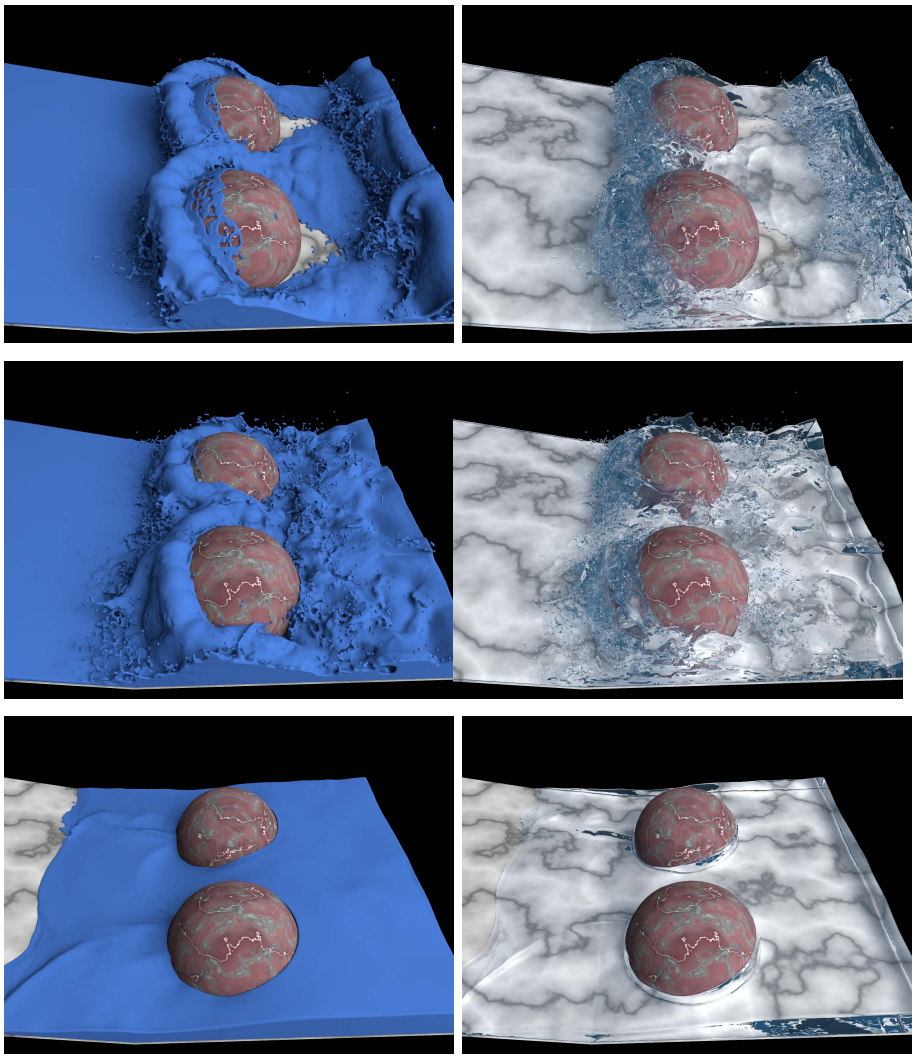


Figure 4.10 – Surface reconstruction of the Hemispheres scene with a varying particle count of up to 2 million. Note that the smoothness of the thin flowing layer (upper and middle) and the fine details of the fluid over the hemispheres (upper) are well-preserved.

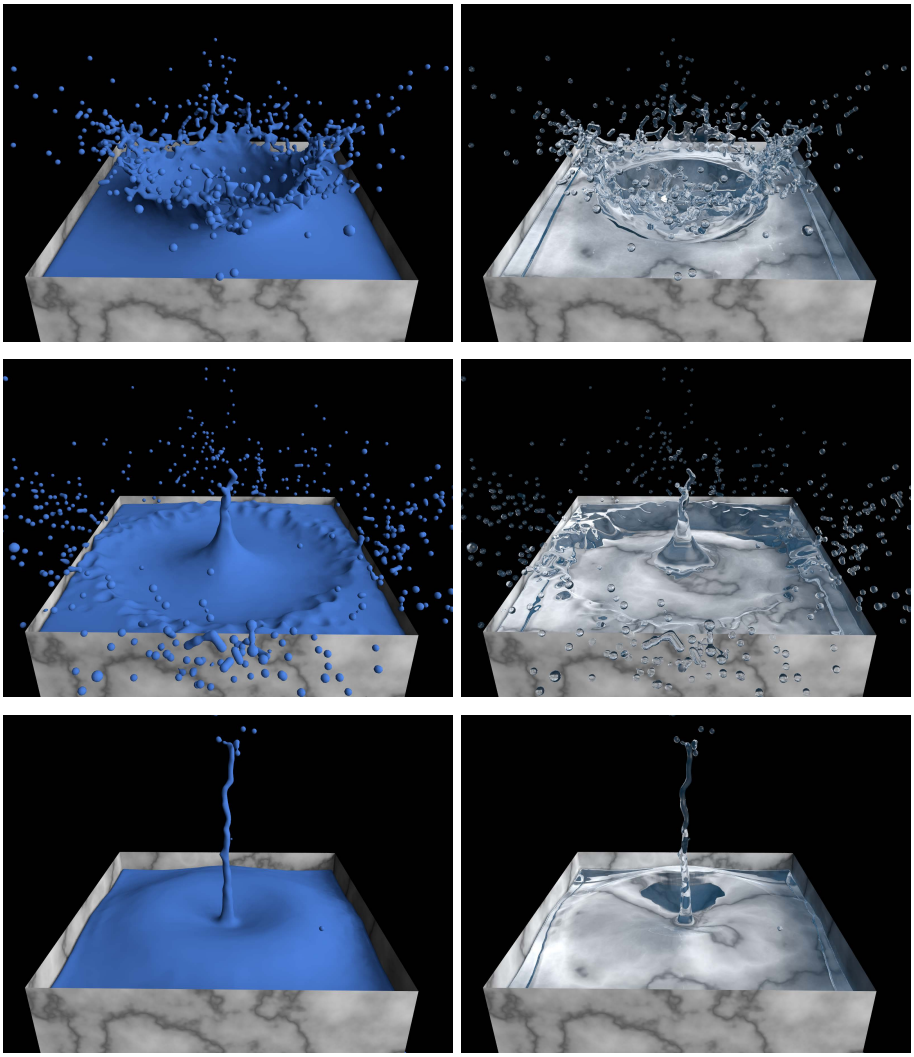


Figure 4.11 – Surface reconstruction of the Drop scene with 360k particles. Fine details and smoothness are well-preserved even for a low number of particles.

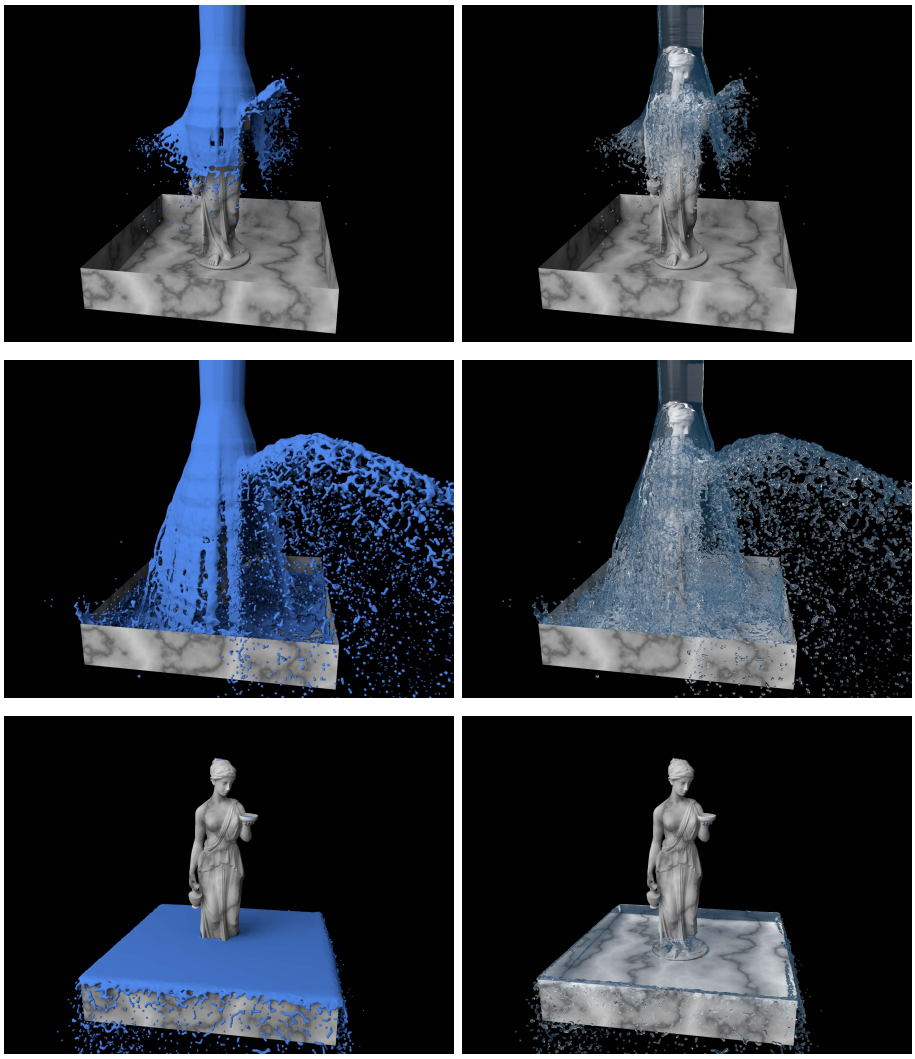


Figure 4.12 – Surface reconstruction of the Hebe scene with a varying particle count of up to 3.2 million. Even for such a large scene, the surface reconstruction is still efficient with less than 3 sec for a cell size equal to r and less than 40 sec for a cell size equal to $r/2$ on average on the GPU.

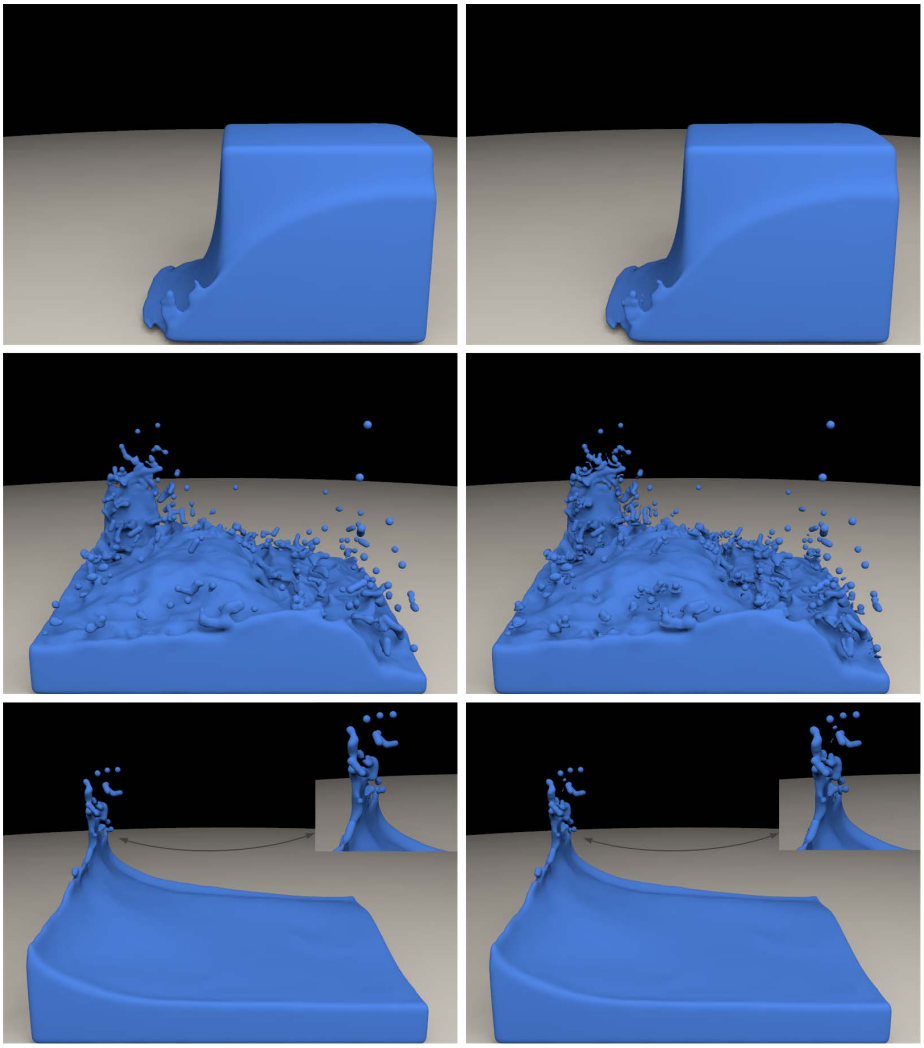


Figure 4.13 – Visual comparison between [SSP07] (left) and [YT10] (right) on CBD-150k scene. The zoomed-in corner is particularly included in the bottom row, so as to show the differences in splash regions clearly.

curvature regions, i.e.„ the regions that have more detailed features (see Chapter 6).

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.

5 Enhanced Surface Quality with Post-Processing

5.1 Introduction

It is a declared problem that surfaces generated for particle-based fluids usually suffer from bumpiness due to irregular particle placement. Bump-free surfaces with smooth features can be generated using smoothing operations, e.g., [YT10, BGB11] or by using very small cell sizes and large influence radii, e.g., Chapter 4. However, most of these approaches either trade off quality with efficiency, or they overlook the amount of detail that the surface provides, which means that the generated surface covers the particles only roughly.

In this thesis, we focus on the efficient reconstruction and processing of SPH surfaces to achieve not only smooth, but also detailed surfaces. For this aim, we propose to combine the existing scalar field computation approaches, in particular [SSP07], with two post-processing steps: decimation and subdivision. This is motivated by an improved representation of smaller surface details, reduction of bumps in flat regions, reduced overall computation time for the surface reconstruction and reduced memory consumption for the resulting mesh. Our experiments show that our approach is able to produce high quality fluid surfaces up to 20 times faster than other approaches with comparable surface quality, while being very efficient in terms of memory and secondary storage consumption.

We address the bumpiness problem by applying a feature sensitive mesh decimation algorithm. This step allows the decimated mesh to remain faithful to the original topology of the reconstructed surface, while it looks smooth in flat regions. However, the decimation step might sharpen the edges of some mesh features (see Figure 5.7, middle). So as to regain the smoothness of such sharp features, we apply subdivision to the decimated mesh. The resolution of the decimated mesh is already adaptive in a way that flat regions are sampled with larger and less triangles, while high curvature regions are sampled with smaller and more triangles. After the subdivision, those high curvature regions with fine details are sampled with even more triangles which are distributed smoothly. Therefore, even though the number of triangles after this step is less than the number of triangles of the input mesh, we achieve a triangle mesh that is even smoother than the input mesh (see Figure 5.7, right). Since the subdivision step ensures the smoothness of mesh features, we gain a significant performance by using lower resolution Marching Cubes grids (see Figure 5.6, (a) and (c) for the

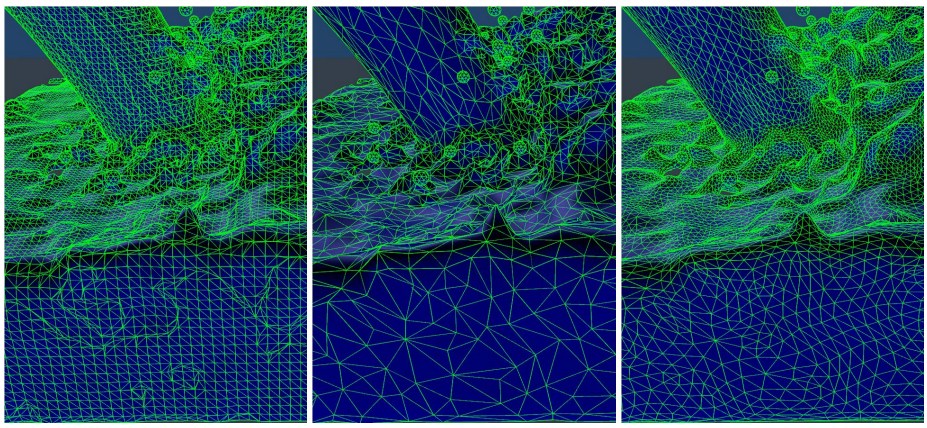


Figure 5.1 – The resolution of the reconstructed mesh (left) becomes adaptive after decimation (middle) and preserves its adaptivity after subdivision (right). Note that the mesh size is reduced without affecting the topology of the mesh; however, the result is smoother than the original mesh. The isolated particles are reconstructed as pre-tessellated spheres in all cases.

comparison).

Furthermore, we handle the isolated particles as pre-tessellated spheres. By extracting them from the main surface computations, the Marching Cubes grid shrinks, the performance increases and the memory consumption decreases.

5.2 Algorithm

In this section, we describe a simple yet efficient pipeline for the surface reconstruction of particle-based fluids. As stated previously, any of the aforementioned scalar field computation techniques can be used in our pipeline. In our experiments, however, we prefer [SSP07] over other techniques, since the method removes the artifacts that arise in concave regions efficiently and covers the underlying particles faithfully.

5.2.1 Decimation

The aim of this step is to reduce bumps in flat regions while maintaining fidelity to the original mesh. Therefore, we need a method that distinguishes curvature regions from low amplitude features (e.g., bumps) in flat regions. Quadric error metric technique [GH97, GH98, Hop99] is a suitable choice for our aim, since the distance-to-original-mesh based decimation priority works properly in our case. Besides, the method is able to simplify even very complex surfaces rapidly. Quadric error metric based decimation technique iteratively contracts vertex pairs in three steps:

1. Two relatively close vertices, v_1 and v_2, are moved to a new position $\bar{v}$.
2. Vertices of v_1 and v_2 are connected to $\bar{v}$.
3. v_1 and v_2 are deleted.

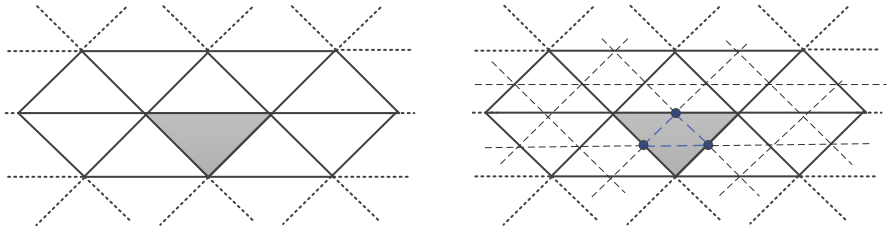


Figure 5.2 – Refinement of each triangle into 4 triangles by splitting each edge at midpoints.

According to the algorithm, contraction costs are computed in order to select the correct vertex pair for contraction. For this aim, a symmetric 4×4 matrix $\mathbf{Q}$ is associated with each vertex in the beginning; and this matrix is used to define the error at the given vertex position $\mathbf{v}$ as $\mathbf{v}^{\mathrm{T}}\mathbf{Q}\mathbf{v}$. The matrix $\bar{\mathbf{Q}}$ that needs to be associated with the new vertex $\bar{v}$ can be computed with a simple additive rule as

$$\bar{\mathbf{Q}} = \mathbf{Q}_1 + \mathbf{Q}_2.$$

The error $\bar{\mathbf{v}}^{\mathrm{T}}\bar{\mathbf{Q}}\bar{\mathbf{v}}$ of this target vertex becomes the cost of contracting the given pair. While contracting the pair v_1 andv_2, $\bar{v}$ is chosen along the line segment $|v_1v_2|$ so that it has the minimal cost. After the computation of contraction costs, the pairs are stored in a heap which is keyed based on the cost. The pair with the minimum cost is kept at the top of the heap, processed first, and the cost of the pairs involving v_1 and v_2 are updated accordingly. On curvature regions which characterize the surface mesh, the distance between $\bar{v}$ and the original mesh is large enough which increases the collapse cost. However, $\bar{v}$ on flat or lower amplitude regions (e.g., bumps) produces less collapse cost, which pushes them on top of the heap.

A sequence of pair contractions are applied until the simplification goal is satisfied. In our experiments, we reduce the total number of triangles by 80%. After this step, the mesh resolution becomes adaptive in a way that the details in high curvature regions are still preserved, while the bumpiness in relatively flat regions is significantly alleviated by using less and larger triangles in those regions (see Figure 5.1, middle).

5.2.2 Subdivision

The decimation step alleviates the bumpiness problem, together with an effective reduction in the mesh size. However, this step causes the edges of some mesh features to sharpen, and the mesh smoothness is not fully preserved. These sharp edges can cause flickering artifacts during rendering, especially on ray-traced transparent surfaces. In order to resolve this problem, we apply one more post-processing step which subdivides the surface mesh, and improves the non-smooth parts efficiently.

For the subdivision, we employ Loop's subdivision scheme [Loo87] since the method is easy to implement, and it is able to produce smooth surfaces very efficiently. In the first pass of this scheme, new vertices are added at the midpoints of the edges of triangles. In the second pass, new edges are added to

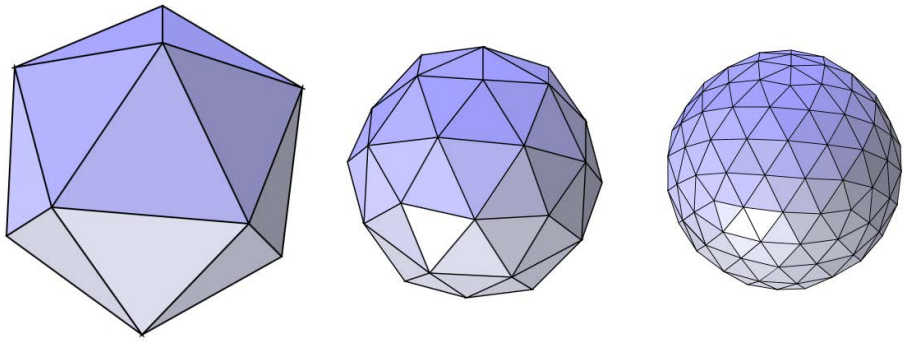


Figure 5.3 – Subdivision of an icosahedron with Loop algorithm after one and after two refinement steps. Image courtesy of Wikipedia.

complete the subdivision into four new triangles (see Figure 5.2). In the final pass, Loop's mask is applied. In other words, vertex positions are averaged so that each vertex in the triangle mesh has a new position which is computed based on the number and relative positions of its neighbors as follows.

$$\mathbf{v}_{new} = \frac{\alpha(n)\mathbf{v} + \mathbf{v}_1 + ... + \mathbf{v}_n}{\alpha(n) + n} \tag{5.1}$$

with

$$\alpha(n) = \frac{n(1 - \beta(n))}{\beta(n)} \tag{5.2}$$

and

$$\beta(n) = \frac{5}{4} - \frac{(3 + 2\cos(2\pi/n))^2}{32} \tag{5.3}$$

where $\mathbf{v}$ is the original position of the respective midpoint, and $\mathbf{v}_1$ through $\mathbf{v}_n$ represent the positions of the neighbor points. At the end of this step, a smoothing effect is achieved (see Figure 5.3).

According to our experiments, the reduction in the original mesh size is usually around 20% after the decimation and subdivision. However, due to the fact that the adaptive mesh is sampled with more and evenly distributed triangles in high curvature and detailed regions (see Figure 5.1, right), we achieve even smoother surfaces when compared to the input mesh (see Figure 5.7, left and right), and still ensure that the main memory and the secondary storage are consumed efficiently. In addition, since the subdivision step ensures the smoothness of mesh features, we do not require a higher resolution of the Marching Cubes grid. This results in a significant performance speed up.

5.2.3 Isolated Particle Extraction

The Marching Cubes approach can require very large grids, if the simulation causes the particles to splash too distant from each other. Such scenarios are especially inefficient in terms of memory consumption. In order to ameliorate this issue, we perform a simple yet efficient step. In our pipeline, we consider a particle as isolated, if it has no neighbors. We determine such particles in the beginning, and exclude them from the steps mentioned in the above sections.

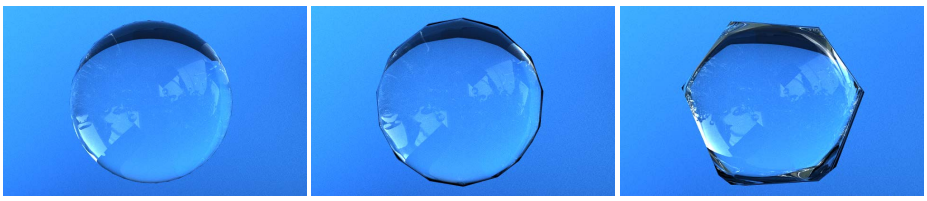


Figure 5.4 – Transparent rendering of an isolated particle using a pre-tessellated sphere (left), and Marching Cubes triangulation with $mcs = r/2$ (middle) and $mcs = r$ (right).

After creating one sphere with radius r initially, we transform this sphere to all positions which belong to isolated particles. A sphere can be tessellated with triangles in an icosahedron form, and can be subdivided until a user defined threshold. This smooth approximation also prevents rendering artifacts which are especially visible on ray-traced transparent surfaces of isolated particles (see Figure 5.4). The cornered edges that cause such rendering artifacts occur due to insufficient Marching Cubes grid resolutions and cause serious flickering artifacts in the animation. So as to generate smooth, spherical shapes for the isolated particles, very high resolution Marching Cubes grids are required. However, this results in very large computation times and memory consumptions, while the tessellation of such particles might still be insufficient (see Figure 5.4, middle).

5.2.4 Implementation

Surface Fitting

The generation of smooth and bump free surfaces has been one of the main concerns in the field, e.g., [BGB11, AIAT12]. However, the point that has been usually overlooked is the amount of detail that the surface provides, which can be achieved by covering the underlying particle set as good as possible. Therefore, the employed scalar field computation technique should fit the particle set faithfully. Besides, a proper setup of the influence radius significantly affects the quality. In order to show this effect, we experimented with two different values of R in our pipeline. Figure 5.8, (b) shows that with $R = 4r$, the underlying particle set is covered properly. However, when we double R (see Figure 5.8, (c)), the particle set is covered very roughly. Such a large influence radius alleviates the bumpiness problem by smoothing out surface details which is an undesired result. For a fair comparison, both settings were tested within our pipeline. However, in terms of bumpiness problem, post-processing steps do not improve the results of the large influence radius since the small scale surface details are already eliminated by the initial surface reconstruction step. Similar to the results of this setting, the method of Bhattacharya et al. also covers the particle set coarsely and the surface details are again lost (see Figure 5.8, (d)). Despite the fact that better coverage of underlying particles increases the bumpiness problem, our pipeline ensures that the bumps in flat regions are alleviated (see Figure 5.8). In addition, proper fitting of the surface also prevents temporal coherency problems which are especially visible in the coarsely covered surfaces.

Scene	Tap	Fountain	CBD	Ship
$\text{time}_{\text{sf-t}}$	2.4	8.5	6	66
time_{dec}	0.77	3.5	1.5	18
$\text{time}_{\text{subdiv}}$	0.1	0.75	0.35	3.6
$\text{time}_{\text{total}}$	3.27	12.75	7.85	87.6

Table 5.1 – Average per frame timings in seconds for each scene. $\text{time}_{\text{sf-t}}$ denotes the computation time for scalar field and triangulation; while time_{dec} and $\text{time}_{\text{subdiv}}$ stand for the computation time of decimation and subdivision, respectively.

Simulation and Hardware

For the fluid simulation, we employed the PCISPH method of Solenthaler et al. [SP09] and for the Ship scene, we used the two-way coupling method of Akinci et al. [AIA+12]. The ship model is courtesy of www.thefree3dmodels.com. For an efficient neighborhood search over particles, we prefer the compact hashing method proposed by Ihmsen et al. [IABT11]. All the scenes were rendered using mental ray v3.9.4 [NVI11]. Experiments have been performed on an Intel Xeon X5680 CPU with 24GB RAM.

5.3 Results

In this section, we discuss the results of our pipeline in terms of visual quality and performance on four different scenes: Fountain, Tap, Corner Breaking Dam (CBD) and Ship. A detailed computation time analysis of the scenes is shown in Table 5.1.

Our first scene is the Fountain with up to 100k fluid particles (see Figure 5.6). Using this scene, we firstly compare the result of our pipeline with [SSP07], which is reconstructed using the Marching Cubes cell size of $r/2$. Neither decimation nor subdivision is applied to the latter. Figure 5.6, (a) and (c) show that our pipeline produces comparable results even without using such a high resolution grid. While our pipeline computes one frame within 12.5 seconds on average, this time increases to 200 seconds for the latter. Besides, the average number of generated triangles are 480k and 2.3m for each approach, respectively. High resolution Marching Cubes grids ensure the smoothness of mesh features. However, bumpiness problem remains (see Figure 5.6, (c)). Secondly, we compare our result with a more recent surface reconstruction method [BGB11]. Our motivation for applying [BGB11] is that, the results of this method are already very smooth and bumpiness is alleviated without applying any post-processing steps. However, as illustrated in Figure 5.6, (d), the particle data set is not exactly covered but smoothed out due to the surface approximations and the smoothing steps of the algorithm. Besides, the average surface reconstruction time per frame of this approach is 240 seconds, which is almost 20 times slower in comparison to our pipeline. In addition, the number of generated triangles is 1.75m, which again shows the memory and secondary storage consumption efficiency of our method.

Our second experiment is the Tap scene where the fluid is poured from three taps as shown in Figure 5.7. This is the smallest test scene with up to 60k particles. Before the post-processing steps, the fluid covers the underlying particle set tightly, however, it looks bumpy in flat regions (Figure 5.7, left).

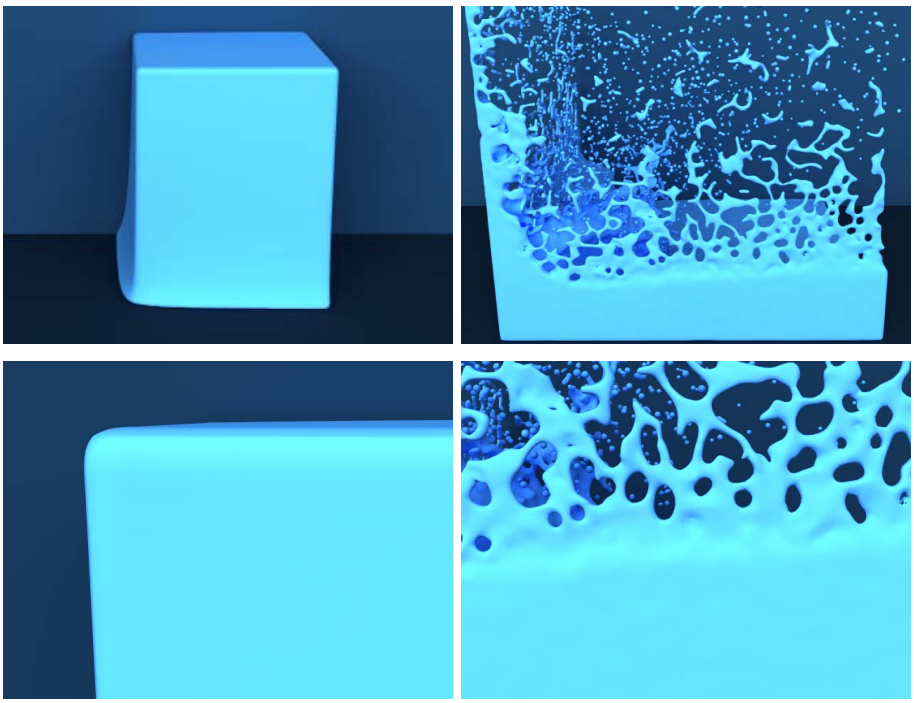


Figure 5.5 – The corner breaking dam (CBD) scene with 130k particles. Close-up views in the bottom row show that our pipeline produces perfectly smooth and bump-free surfaces in flat regions.

Note that after the whole pipeline is applied, the result is less bumpy on sides and inflow, and it is smoother, but still detailed even for such a low particle resolution (Figure 5.7, right).

Our next scene is a corner breaking dam with 130k particles (see Figure 5.5). In order to test the effect of the influence radius R on the surface quality, we experimented with two different parameter settings on this scene as shown in the accompanying video. Using $R = 4r$, the computation takes 7.85 seconds per frame; while it takes 58 seconds for the setting with doubled influence radius. The computation time increases for the latter, since a larger influence radius causes to traverse over more particles in the neighborhood of the grid vertex. Another disadvantage of the doubled influence radius is that, the particle set is coarsely covered and the surface details are lost, which leads to temporal coherency artifacts.

Finally, we present our largest scene, the Ship (see Figure5.9), which was simulated with 4 million fluid particles. Our results show that even for such a large scale simulation, the average computation time is reasonable with 87.6 seconds per frame.

5.4 Discussion and Future Work

Our method post-processes the fluid surfaces so that the particle alignment related and undesired bumps in flat regions are removed, while the characteristic

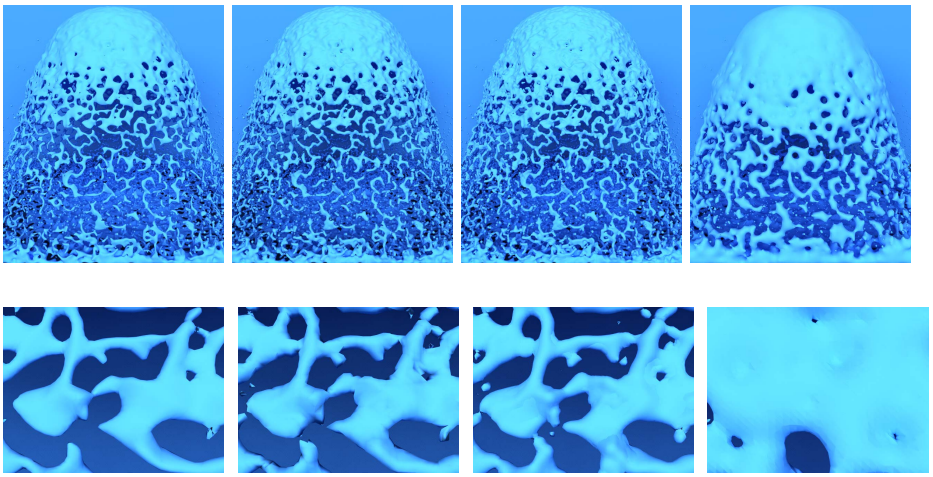
features of the mesh are maintained. After the subdivision step, we obtain a surface whose features are even smoother than the input mesh. Our parameter setup and choice of the employed methods allows us to preserve the surface details, and finally to have a high quality surface. In comparison to other methods with comparable surface quality, our pipeline runs up to 20 times faster with 80% less memory and secondary storage consumption.

The decimation threshold we prefer (80%) may not be optimal in all cases. The reason is that some of the frames allow the mesh to be decimated more than 80% if the surface is perfectly flat or if the high curvature regions are in minority when compared to flat regions. The opposite situation holds if high curvature regions are in majority in the surface. In such a case, using a value smaller than 80% would give better results. However, the threshold we use has been the best choice in all of our test scenes, since larger or smaller values were not appropriate for achieving the same quality in most of the frames. One of the next steps to improve our method can be using an adaptive threshold which depends on the average curvature of the surface in every frame.

After post-processing steps, it is expected to have temporal incoherence in subsequent frames. Even though we did not observe such a behavior in our test scenes, the effect of the post-processing steps in temporal coherence can be analyzed in the future.

Although we exclude the isolated particles from the main computational steps, the Marching Cubes grid can still remain very large depending on the frame, and this causes a large memory footprint when performing surface reconstruction. In the future, we would like to incorporate hashing for the used grid vertices instead of using uniform grid structures.

Another direction for future work can be parallelizing our code and focusing on a GPU implementation.



(a) Our pipeline, $mcs = r$ **(b)** [SSP07], $mcs = r$ **(c)** [SSP07], $mcs = r/2$ **(d)** [BGB11]

Figure 5.6 – The Fountain scene with up to 100k particles. mcs and r denote the Marching Cubes cell size and the particle radius, respectively. (a) The result of our pipeline. We post-process the surface mesh that is reconstructed by using [SSP07] with $mcs = r$. (b) [SSP07] with $mcs = r$ without applying post-processing. (c) [SSP07] with $mcs = r/2$ without applying post-processing. (d) [BGB11] where the marching tetrahedra grid resolution is the same as we employ. The average surface reconstruction time per frame for each approach is: 12.75, 8.5, 200 and 240 seconds; while the average number of generated triangles are: 480k, 600k, 2.3m and 1.75m, respectively. In comparison to (b), our post-processing pipeline increases the quality of the surface with only a small computational overhead, while being even more efficient in terms of the number of generated triangles. In comparison to other two approaches with comparable quality, our approach runs 15 to 20 times faster with up to 80% improved memory and secondary storage consumption.

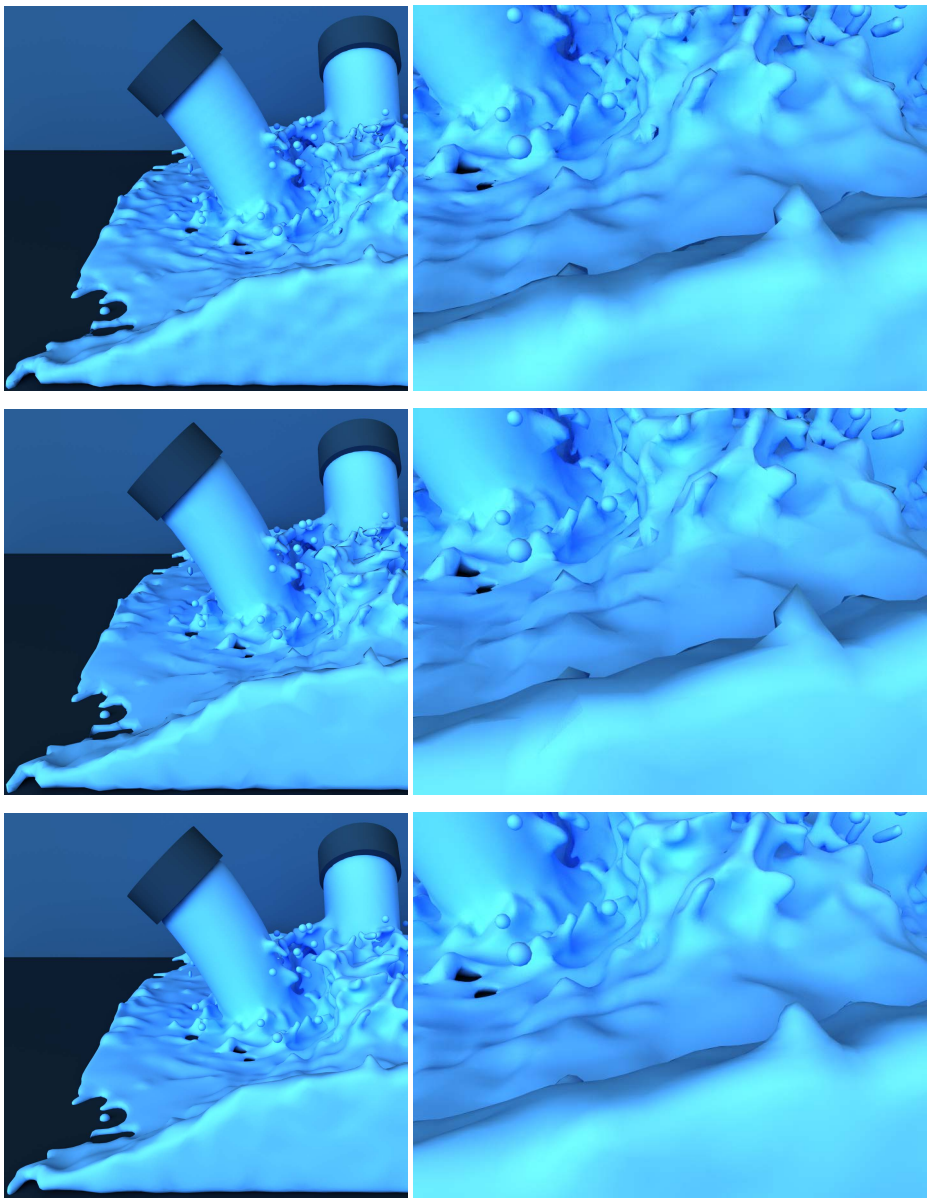


Figure 5.7 – The Tap scene with up to 60k particles. The surface reconstruction is the first step of the pipeline (up). The decimation step reduces bumps in flat regions while it remains faithful to features of input mesh (middle). Finally, undesired sharp features are smoothed by the subdivision step (bottom). Particles without neighbors are rendered as pre-tessellated spheres in all images.

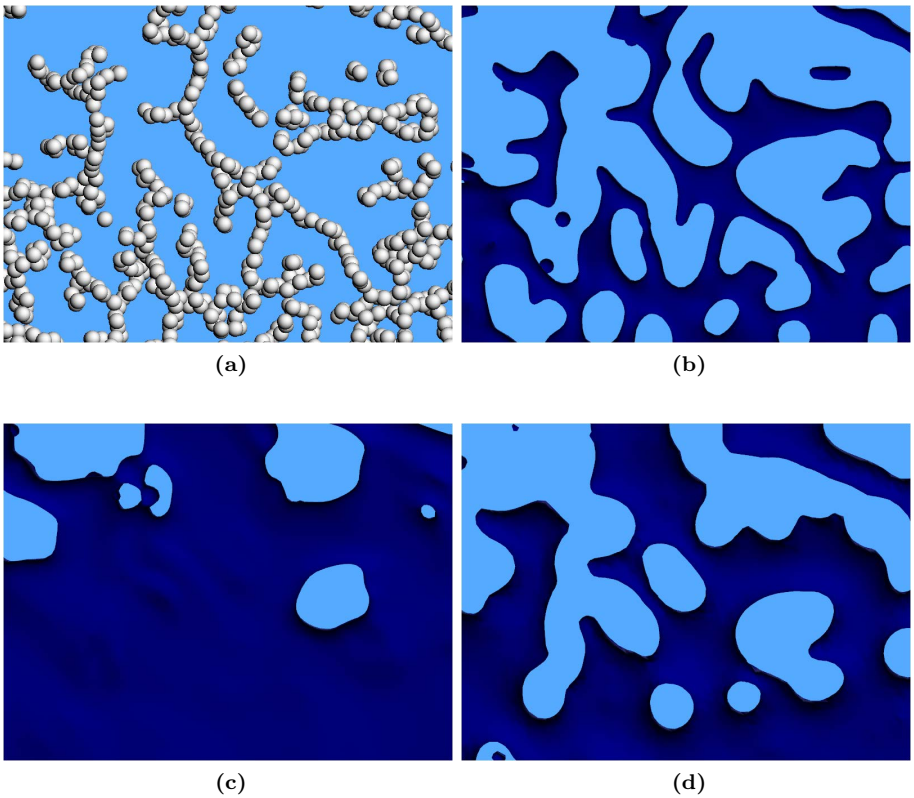


Figure 5.8 – A close-up of the Corner Breaking Dam (CBD) scene. (a) The underlying particle set. (b) By using an influence radius $R = 4r$, we cover the underlying particle set as tightly as possible. (c) Doubled influence radius causes the surface to cover the particle set coarsely. It is worth noting that although the same Marching Cubes cell size is used in both settings, the larger influence radius causes larger computational times, which is discussed in Sec. 5.3. (d) Similar to (c), with the method of Bhattacharya et al. [BGB11], the surface is roughly covered and the details of the underlying particle set are not visible.

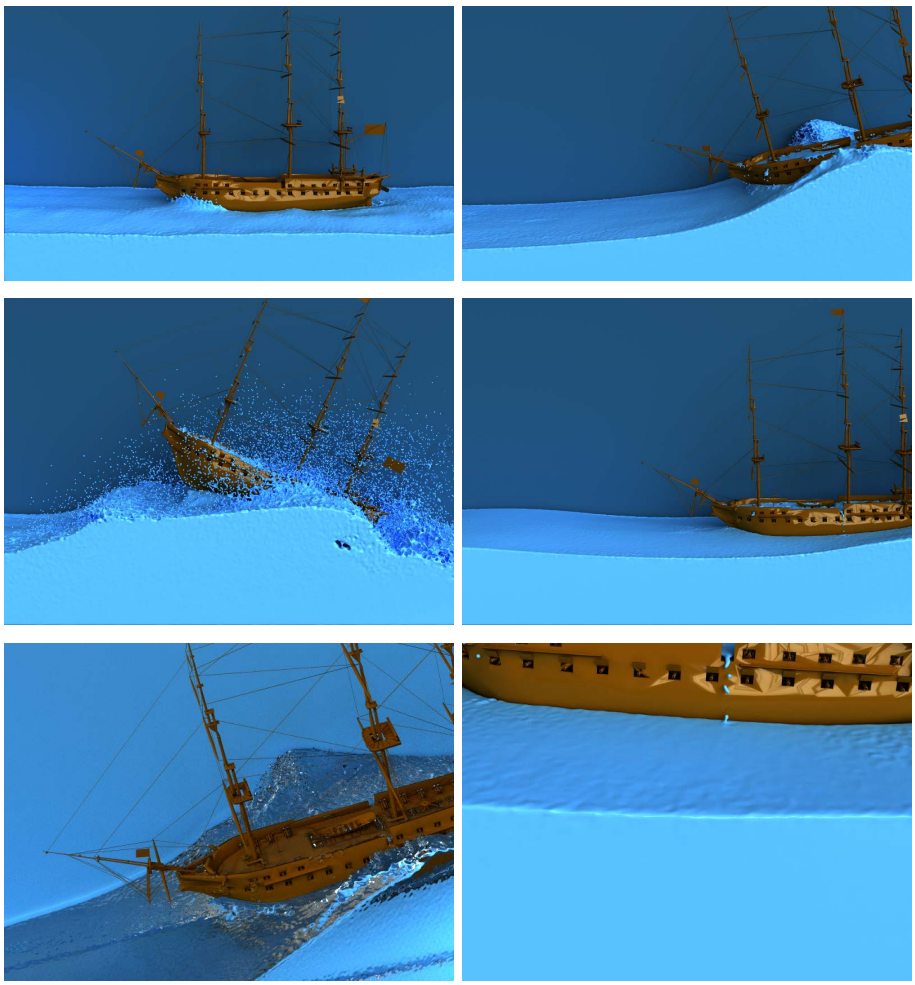


Figure 5.9 – The Ship scene with 4 million particles. Transparent rendering and a zoomed in versions are shown in the last row.

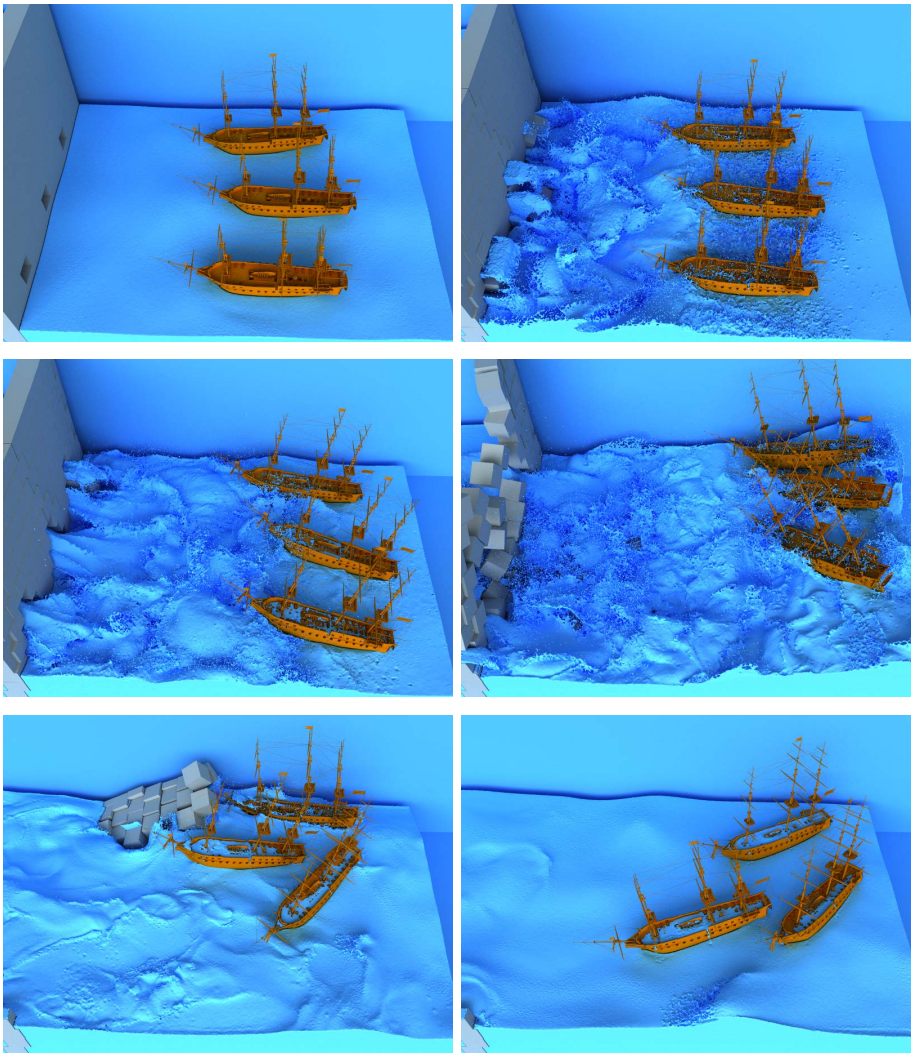


Figure 5.10 – The Three Ships scene with more than 10 million particles. The surface reconstruction of this scene has been performed using our pipeline.

6 Curvature-Based Multi-Level Uniform Grids

6.1 Introduction

The chosen uniform grid resolution of the Marching Cubes method restricts the user since surface details are reconstructed properly only by using high resolution grids at the expense of performance, memory footprint and storage. This issue has prompted many researchers to investigate adaptive mesh refinement techniques, e.g., by using octrees [WVG92, SFYC96, WKE99, VT01, LY04, JU06, MS10]. Although being efficient in terms of memory consumption and storage reduction, the construction of adaptive structures is not straightforward, and they lead to cracks in between different resolution cells which need to be handled carefully. The use of uniform grids, on the contrary, is motivated by the simplicity of the structure which is advantageous for especially dynamic scenes since it allows fast rebuilding of the data structure. To the best of our knowledge, most of the presented adaptive approaches focus on static scenes, e.g., medical visualizations or CAD models, and there are only few researchers who aim to efficiently rebuild those data structures for dynamic scenes, e.g., [ZGHG11].

In this thesis, we present a memory efficient and performance friendly multi-level uniform grid structure for SPH surface reconstructions which allows for fast rebuild in dynamic scenes.

In our technique, the particle data set is covered by an axis aligned bounding box (AABB) which is initially subdivided uniformly with coarse (level-1) cells. These cells are utilized to extract the narrow-band region where the surface is actually defined. Depending on the surface curvature, coarse surface cells are subdivided by one (level-2) or two more levels (level-3). This allows for preserving the surface details even on highly turbulent, high curvature parts by using less triangles, since relatively flat parts are treated as level-2 parts. The described three levels are illustrated in Figure 6.1. We close the arisen cracks eventually by creating new triangles in the size of cracks in between different resolution mesh blocks.

A proper initialization of the cell resolutions and the parameters, e.g., curvature threshold, is essential for obtaining high quality results in a performance and memory friendly way. We discuss the details of such initializations throughout this chapter.

Experiments show that when compared to high resolution single level uniform grids, the presented method produces similar results with up to four times better

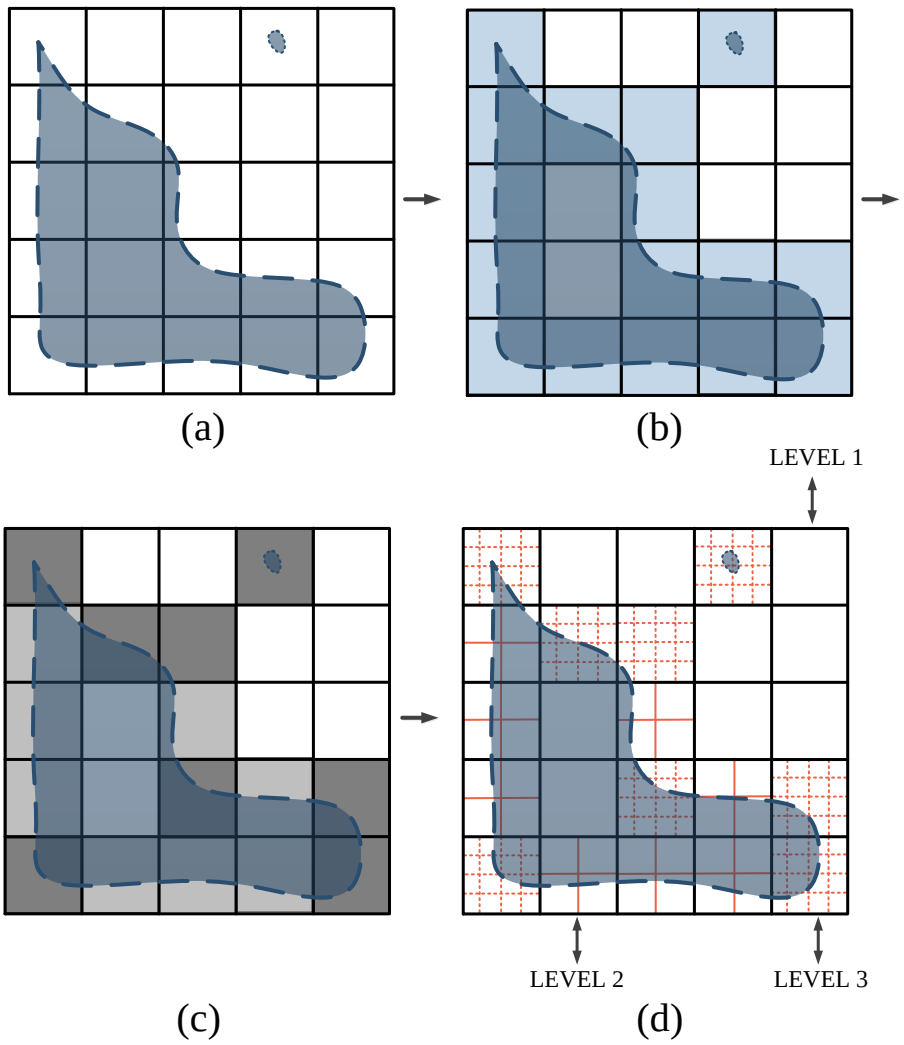


Figure 6.1 – The 3-level adaptive grid. (a) The bounding box of the fluid is used to generate the coarse level uniform grid (level-1). (b) Cells that contain the fluid surface (dashed blue line) are extracted. These surface cells are shown in blue color. (c) Surface cells with low surface curvature are marked as level-2 (light gray cells) and the other surface cells are marked as level-3 (dark gray cells). Cells around the isolated pieces are always considered as level-3. (d) Level-2 cells are subdivided by one more level (straight red lines) while level-3 cells are subdivided by two more levels (dashed red lines)

memory consumption and up to 60% better computation time. Three different test scenarios are discussed in Sec. 6.3, where the computation time and memory consumption data are given for all scenes.

6.2 Algorithm

According to our experiments, in order to obtain proper surface reconstructions, Marching Cubes grid cell size should not be larger than $2r$, with r being the radius of the fluid particles, i.e., the half of the particles' equilibrium distance. However, even $2r$ cell size can be insufficient to preserve all of the surface details. In such a case, experiments show that the cell size of r preserves all surface features appropriately. However, using such a small cell size throughout the whole scene causes a trade-off between the surface quality and the performance-memory consumption. Besides, it is clear that on relatively flat regions, the cell size of $2r$ is already sufficient for obtaining proper results. Therefore, we reconstruct the fluid surfaces using these two resolutions in different regions depending on the surface details.

We initialize our surface reconstruction steps by extracting the narrow-band region where the surface is actually defined. Performing this operation does not require a high resolution grid. We initially create our grid using the cell size of $4r$, extract the narrow-band region using the coarse cells in this resolution, and subdivide the found surface cells using the surface curvature information.

We extract the surface cells using the method explained in Chapter 4. Different than this method, we mark the cells as splash cells instead of surface cells if they are in the neighborhood of splash particles, i.e., particles whose number of neighbors are less than a pre-defined value which is 3 in our test scenes. By doing this, we are easily able to exclude such cells from the curvature computation in future steps and gain performance since they are directly subdivided as level-3 high resolution cells.

Surface particle extraction is performed in the preprocessing step using the smoothed color field method of Muller et al. [MCG03]. The same method is also used to compute particles' normals which are required for the curvature computation in the following step (see Sec. 6.2.1). After refining the cells, the scalar field is computed over all grid cells, the cracks are handled by adding new triangles and different resolution meshes are stitched seamlessly.

6.2.1 Curvature Computation

According to our criterion, a coarse surface cell can be subdivided as either level-2 or level-3 which depends on the surface curvature inside the cell. We approximate the surface curvature using the method described in [IAAT12]. Hence, we firstly compute the curvature for each surface particle that resides in this cell with the help of its neighboring particles as:

$$\kappa_i = \sum_j \kappa_{ij} = \sum_j (1 - \hat{\mathbf{n}}_i \cdot \hat{\mathbf{n}}_j) k_c(\mathbf{x}_i - \mathbf{x}_j, h) \tag{6.1}$$

where j stands for the neighboring particles, $\hat{\mathbf{n}}$ is the normal of any particle and k_c is the kernel function described as:

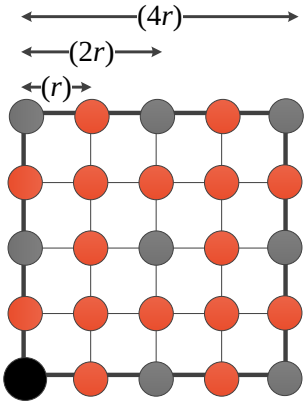


Figure 6.2 – A 2-dimensional illustration of the cell refinement. The thick black line shows the outline of the coarse surface cell whose initial coarse point is colored in black. The gray circles demonstrate level-2 points and both gray and red circles demonstrate level-3 points that are added after the cell refinement.

$$k_c(\mathbf{x}_i - \mathbf{x}_j, h) = \begin{cases} 1 - \|\mathbf{x}_i - \mathbf{x}_j\| / h & \text{if } \|\mathbf{x}_i - \mathbf{x}_j\| \leq h \\ 0 & \text{else} \end{cases}. \tag{6.2}$$

Finally, the curvature of any surface cell is approximated as:

$$\kappa_{cell} = \left(\sum_{\text{for all } i} \kappa_i \right) / N \tag{6.3}$$

where i and N represent the surface particles and the total number of surface particles inside the cell, respectively.

If the curvature is smaller than the predefined threshold, we mark the cell as low resolution level-2 cell. Otherwise, the cell is marked as high resolution level-3 cell. In the remainder of this chapter, we will call any coarse surface cell as level-2 or level-3 cell depending on its refinement level.

6.2.2 Adapted Scalar Field Computation Using Hash Maps

We initialize the second step of our method with cell refinement. Therefore, we traverse through all coarse surface cells. If the cell is marked as a level-2 cell, we generate 8 new cells in the coarse cell which correspond to 27 new grid points. A similar approach is followed for level-3 cells with 64 new cells and 125 new grid points. A 2-dimensional illustration of this refinement is shown in Figure 6.2.

At this part of our implementation, we need a data structure to keep our newly generated grid points. This structure should support easy access to the required grid point. For this aim, we compute an *id* for each grid point and use a hash map structure which allows for easy data access by querying the *id*s that are the keys of the hash map. This structure also stores the corresponding scalar values as the data part of each entry. The *id* of a grid point is computed using its position

$$\mathbf{GridPos_{l-3}} = (gpx, gpy, gpz)$$

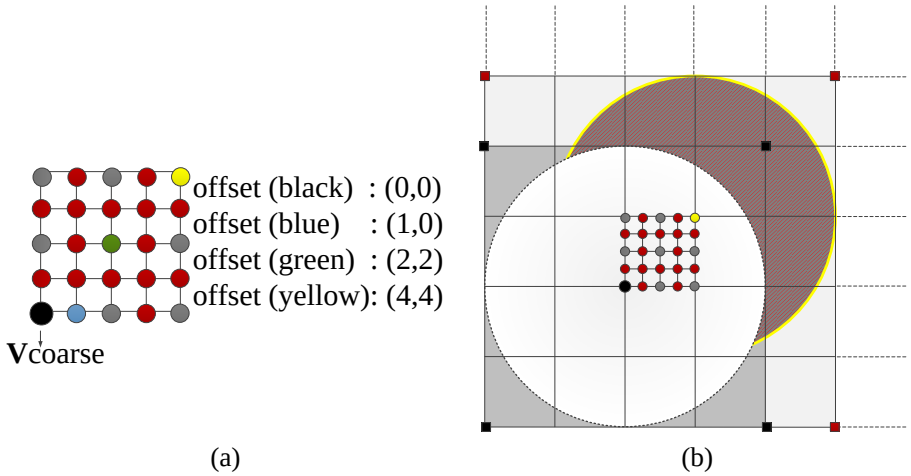


Figure 6.3 – (a) The coarse initial grid point with position **vc** is illustrated with a black circle. Offset values for some fine grid points are given in 2-D which is similar for the 3-D version. (b) The white, large circle shows the influence region of the coarse initial surface grid point. However, this region is not sufficient for newly added fine grid points. The red region which is the influence region of the yellow point in the figure should also be checked for influencing particles.

and number of potential grid points X_{l-3} and XY_{l-3} in x- and xy-directions, respectively, in a virtual, high resolution level-3 grid throughout the whole scene. $\mathbf{GridPos_{l-3}}$ can be written as:

$$\mathbf{GridPos_{l-3}} = 4 \cdot \mathbf{vc} + \mathbf{offset} \tag{6.4}$$

where **vc** is the position of the coarse initial grid point of the cell in the coarse grid. The 3-dimensional **offset** $= (a, b, c)$ vector of any new grid point describes the position of this point in the cell. a, b and c values are defined within the range of [0,5). Thus, these values start with 0 and depending on the refinement level, they increase by 1 or 2 for the high resolution and low resolution cells, respectively (see Figure 6.3). Furthermore, X_{l-3} and XY_{l-3} are computed as:

$$\begin{aligned} X_{l-3} &= 4 \cdot ncx + 1 \\ Y_{l-3} &= 4 \cdot ncy + 1 \\ XY_{l-3} &= X_{l-3} \cdot Y_{l-3} \end{aligned} \tag{6.5}$$

with ncx and ncy being the number of cells in x and y directions in our coarse grid. Finally, the id of a grid point is computed as:

$$id = gpx + X_{l-3} \cdot gpy + XY_{l-3} \cdot gpz \tag{6.6}$$

After computing the id of any grid point, we check whether it is already stored in the hash map. In such a case, we do not compute the scalar value since this information is already kept together with the id. Otherwise, we compute the scalar value of the grid point using the improved signed distance field approach of

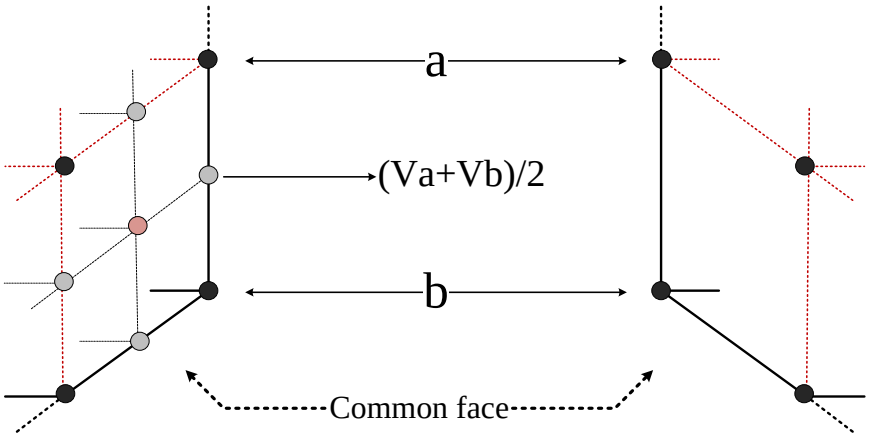


Figure 6.4 – The scalar field continuity is ensured on transition cell faces by sub-sampling the original data.Va and Vb are the scalar values of grid points a and b, respectively.

Solenthaler et al. [SSP07]. At this step, it is important to identify the particles which contribute to the scalar value computation of each grid point. Particles within the influence radius of each coarse grid point can be easily retrieved by using our method, see Chapter 4. However, since the influence region is changed for the newly added fine points, we check all the particles that lie also in the influence region of the final grid point in the cell, i.e., the fine point with **offset**(4,4,4), which is shown with yellow color in Figure 6.3. The computed scalar value can now be stored in our hash map with its corresponding *id*.

6.2.3 Adapted Polygonization and Crack Handling

Triangulation is the most challenging part of our method, since we need a special treatment for handling the cracks which arise in between different resolution transition cells. As mentioned before, various crack handling techniques have been proposed in the literature. Many of these methods, however, are either computationally expensive, or produce T-vertices which cause visual artifacts, or produce large number of triangles and cause memory inefficiency. Therefore, we present a method for an efficient crack closing. We perform this step gradually by ensuring the scalar field continuity first and covering the cracks with new triangles afterwards. In order not to affect the regular triangulation pipeline for uniform grid structures significantly, we leave the Marching Cubes algorithm unchanged for our level-2 cells but change the procedure slightly for level-3 cells.

Scalar Field Continuity

The scalar field that is computed over our multi level grid is not continuous, i.e., the scalar field continuity is broken on the faces of neighboring different resolution cells. Therefore, similar to [OR97], we adjust the values of intermediate grid points on such transition cell faces by sub-sampling the original data (see Figure 6.4). So as to carry out this task in our implementation, we traverse through all level-2 cells and identify the cells that have a level-3 cell neighbor. If any level-2

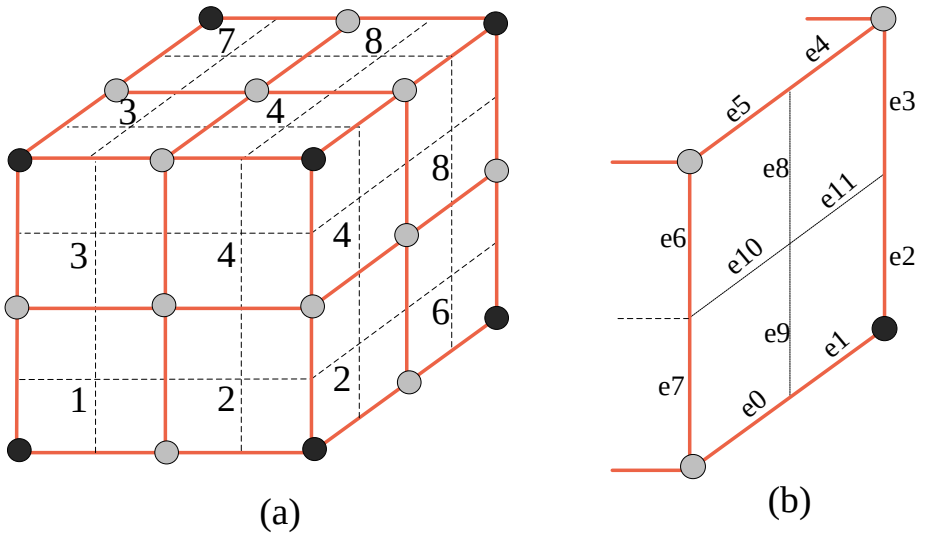


Figure 6.5 – (a) Eight sub-cells of a level-3 cell are illustrated by red cells. (b) Edges are illustrated with their ids for the right transition face of sub-cell-6.

cell meets this condition, we adjust the values in the corresponding face.

Even though this approach alleviates the scalar field discontinuity, cracks can still occur due to the numerical sampling problems which are generally observed after the computations around the middle grid point of the face, e.g., the pink grid point in Figure 6.4. Those cracks are covered with triangles in our method which is explained in the following section.

Crack Problem

At the final step of our method, we close the cracks with new triangles for a proper visualization.

In the beginning of our implementation, we apply standard Marching Cubes algorithm to all level-2 cells. Subsequently, we visit each level-3 cell and handle them sub-cell by sub-cell (see Figure 6.5, (a)). We keep the face information of each sub-cell relative to the coarse cell in which it lies. For instance, the sub-cell which is shown as the first sub-cell in Figure 6.5, (a), shares its near, left and bottom faces with the coarse cell that hosts it. Then, we check whether any of these shared faces has a level-2 cell neighbor using the cell neighborhood information of the coarse cell. As soon as the condition is met, we determine the edges (see Figure 6.5, (b)) which have a potential surface intersection point on itself that will be computed by the Marching Cubes algorithm. It is sufficient to check the scalar values of the end points for each edge to determine potential intersections. According to our criterion, if there is an intersection on at least one of the inner edges (e8...e11), then there exists a crack at that face (see Figure 6.6). Therefore, we keep the intersected edge information of each sub-cell face in a new flag, namely $flag_{intersect}$ which is an array of 6 integers.

Once all of the required information is gathered for the crack handling, we apply Marching Cubes algorithm to all eight level-3 fine cells of the sub-cell. Later, we check each face of our sub-cell to see if there is any previously marked

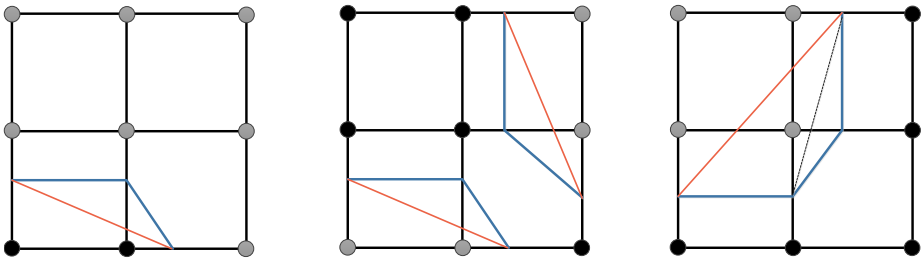


Figure 6.6 – Three of the possible configurations for crack formation. Gray and black colored points have scalar values that are either less or more than the specified isovalue, respectively. Blue lines are generated on high resolution face while red lines are the results of low resolution face. Either one or two cracks can arise (left and middle) or only one crack can cause two triangles (right).

intersection on inner edges. In this case, we firstly create a crack array which is necessary to keep the intersected edge ids, and then we perform the following steps:

1. Go through outer edges (e0...e7). Push the id of the first found edge which has an intersection point into the crack array.
2. For the found outer edge, check neighboring inner (NI), opposing inner (OI) and neighboring outer (NO) edges in its quarter in order to find a new intersection (see Figure 6.7). Push the id of the found edge into the crack array.
3. Jump to the neighboring quarter in which the currently found edge lies and continue to search for a new intersection point by checking neighboring outer, opposing outer (OO) and neighboring inner edges. Push the id of the found edge into the crack array.
4. Follow either step 2 or 3 for any newly found outer or inner intersected edge, respectively.

Instead of simply traversing the edges of each quadrant in any ordering, we follow the edges in an order explained above since this method simplifies having a complete crack array which consists of the intersection points in an order that is easy to follow for triangle generation. For instance, if we have only 3 points in this array, we simply create a new triangle using these points as triangle corners. If we have four points, then we order the triangles' corners as c1, c2, c4 and c2, c3, c4 where c1...c4 are the entries of the crack array.

6.2.4 Implementation

In order to prevent redundant computations on intersected edges, we follow our previous technique (Chapter 4), in which grid points store each computation information for the corresponding three edges that leaves this point. This technique aims single level uniform grids and we pursue a slightly different approach for our multi level grid. We store the information in separate hash maps for level 2 and level 3 fine cells whose keys are again the grid point *id*s, and data part of each entry is composed of 3 integers that are the intersection

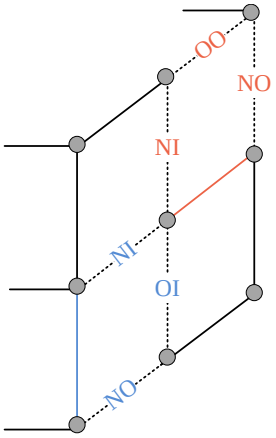


Figure 6.7 – The illustration of an inner (red line) and an outer edge (blue line) together with their neighboring and opposing edges.

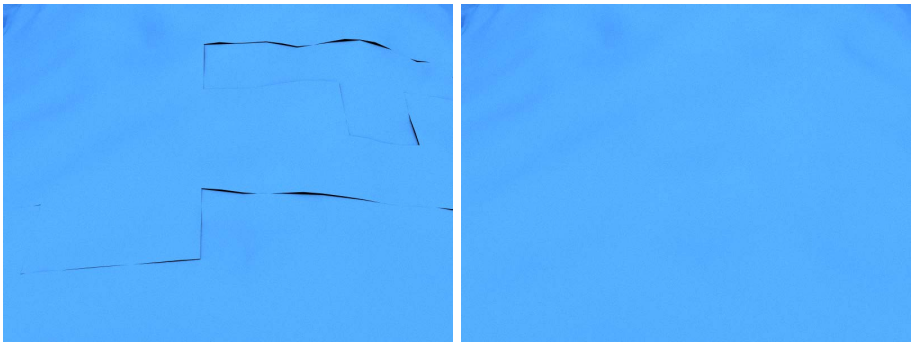


Figure 6.8 – The fluid surface is shown before (left) and after (right) crack handling.

scene	#particles	t_{sf}	t_{tri}	t_{total}
Taps	up to 200k	6 sec	2 sec	8 sec
Splash	up to 500k	25.5 sec	4.5 sec	30 sec
Sink	up to 2.5m	18	2	20

Table 6.1 – Average per frame timings in seconds for each scene. t_{sf}, t_{tri} and t_{total} represent the scalar field computation, triangulation and total timing, respectively.

scene	$\#tri_{l-2}$	$\#tri_{l-3}$	$\#tri_p$	MEM[GB]
Taps	42k	174k	1400	0.9
Splash	140k	1.1m	4000	1
Sink	295k	76k	587	1.25

Table 6.2 – $\#tri_{l-2}, \#tri_{l-3}$ and $\#tri_p$ show the number of level 2, level 3 and patch triangles in order.

points of corresponding edges. We firstly finish the triangulation for both level 2 and level 3 fine cells, and fill their hash maps accordingly. Later, we check the level 2 side of the transition cell faces to see if there is any marked intersection on the corresponding edge. For each transition sub-cell face of the level 3 cell, we check the fine edges of level 2 side and use the information on the associated sub-cell edge. This method is helpful to store the mesh in an indexed format (e.g., Wavefront OBJ). These indexes are also used for an easy retrieval of the corners of the newly generated triangles that fill cracks. At the end of this step, we obtain a surface mesh which contains seamlessly stitched two different resolution mesh blocks.

According to our experiments, the explained data structure is sufficient for a proper surface reconstruction of any fluid data set; and the fourth level is not necessary since it does not improve the quality further but reduces the performance significantly.

Simulation and Hardware

The demonstrated surface reconstructions were run single-threaded on Intel Xeon X5680 CPU with 24GB RAM. For the fluid simulation, we employed the PCISPH method [SP09]. For all examples in the paper, we performed rendering using mental ray [NVI11].

6.3 Results

In this section, we demonstrate the utility of our approach with three different test scenes. Average per frame timings, the number of level-2, level-3, patch triangles and the memory consumption information of both test scenes can be found in Tables 6.1 and 6.2.

In our first experiment, we present a scene with 2 taps pouring water into a box (see Figure 6.13). The scene was simulated using up to 200k particles. This is a simple test scene shows the utility of our approach with a reasonable computation time and memory consumption.

In our next experiment, we show a scene with a rotated plane along which the pouring water scatters (see Figure 6.14). The scene was simulated using up

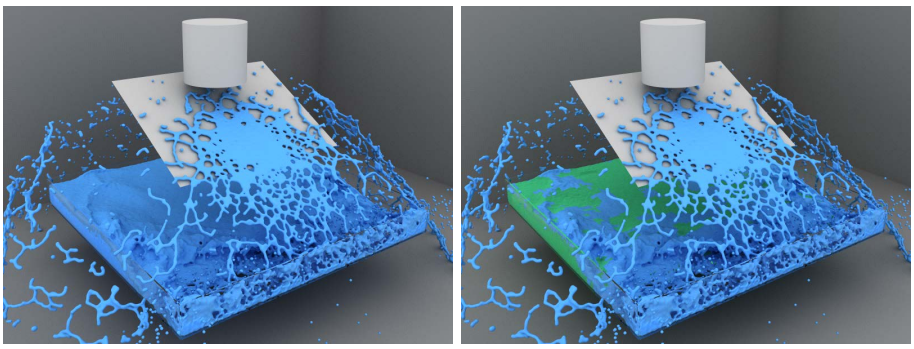


Figure 6.9 – The Splash scene with up to 500k particles. The seamlessly stitched mesh is shown on the left; while blue and green regions on the right image shows the high and low resolution parts, respectively.

to 500k particles. This test scene shows the increase in high resolution parts and so the computation time (see Table 6.1) due to the particle scattering and the dominance of high turbulence regions, see Figure 6.9, right.

Our final scenario is the Sink scene which consists of up to 2.5 million fluid particles (see Figures 6.15 and 6.16). Using this scene, we firstly experimented with the curvature threshold. The curvature threshold should be chosen by taking the nature of the scene into consideration, e.g., the scene can be turbulent and dominated by high curvature regions or vice versa. Thus, the preferred threshold should guarantee sufficient resolution everywhere. In our experiments, we used 1.5 for the Splash scene and 3 for the Sink scene. Figure 6.10 shows the difference between using two different thresholds on the Sink scene. While the value of 3 guarantees a sufficient resolution on high curvature, turbulent regions, e.g., pouring water from the tap; the value of 4 cannot provide sufficient resolution on those parts.

Secondly, we compare our method with low and high resolution single level uniform grid approaches which use the cell size of $2r$ and r, respectively. As shown in Figure 6.11, left, the low resolution single level approach is not able to preserve the details of the pouring water and isolated pieces; and it creates under-tessellated structures. However, it is sufficient to construct the surface properly in the Sink where the fluid surface is relatively flat. On the contrary, high resolution single level uniform grid preserves the details properly as can be seen in the right hand side but this resolution is not necessary for the fluid in the Sink. Figure 6.11 shows that our method uses low resolution inside the Sink and high resolution for the pouring water. Finally, we produce a surface which is of similar quality in comparison to the high resolution single level uniform grid (see Figure 6.11, middle). Table 6.3 and Figure 6.12 illustrates the test results which show that even we introduce many steps for setting up the grid structure and crack handling, our method runs up to 60% faster than the high resolution single level uniform grid approach with up to 4 times better memory consumption. When compared to low resolution single level uniform grid approach, our approach preserves all of the surface details that yields a proper surface visualization, with an acceptable performance and memory overhead.

When we compare the Splash scene and the Sink scene, we see that the computation time and the memory consumption is nearly the same for both

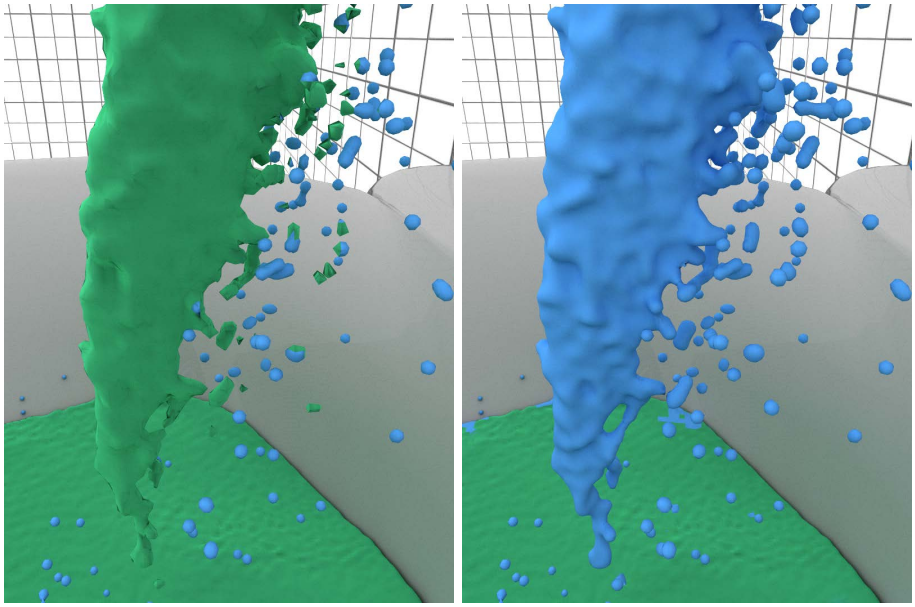


Figure 6.10 – Results with two different curvature thresholds: 4 (left) and 3 (right) are shown. Green and blue regions represent the low and high resolution parts, respectively.

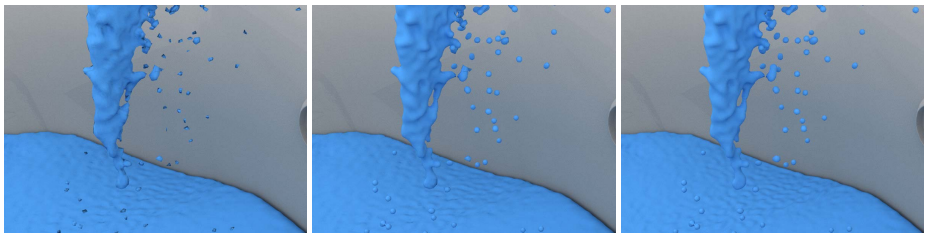


Figure 6.11 – The comparison of the Sink scene with single level uniform grid approaches. The low resolution single level uniform grid (left) neither preserve details nor reconstruct the surface properly as our approach (middle) or the high resolution single level uniform grids (right) do. On the other hand, we achieve a similar quality that high resolution single level grids produce with up to four times better memory consumption and up to %60 better performance.

method	t_{total}	$\#tri_{total}$	MEM[GB]
Uniform low res.	7	310k	1.2
Our method	20	375k	1.25
Uniform high res.	34.5	1.2m	4

Table 6.3 – The comparison of our method with single level uniform grid approaches on the Sink scene. t_{total}, $\#tri_{total}$ and MEM[GB] represent the average per frame computation time, number of triangles and memory consumption, respectively.

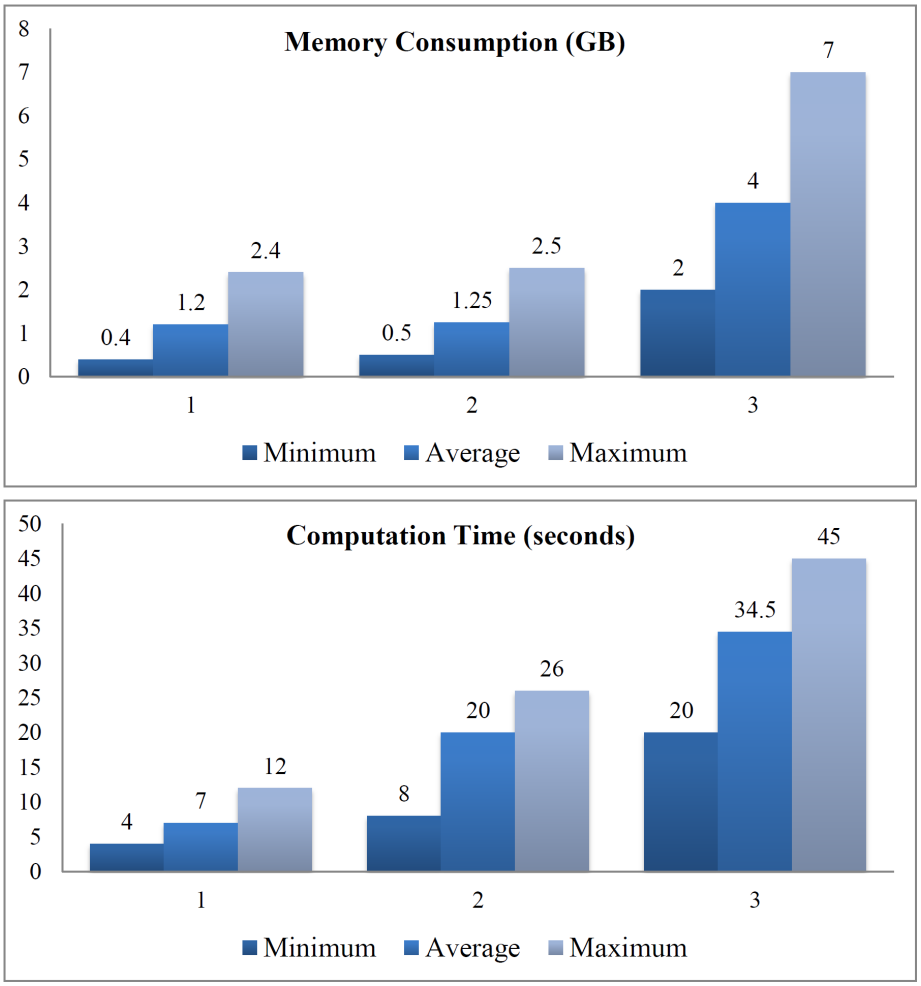
scenes, even though the Sink scene was simulated with larger number of particles. The reason of this similarity is that the particles do not spread around the scene too much in the Sink scene in contrast to the Splash scene. Therefore, both the coarse grid resolution and the number of surface cells in the narrow band region of the Splash scene is slightly larger than the Sink scene. Besides, the Splash scene is dominated by high resolution parts while the Sink scene is dominated by low resolution parts. These points clarify that the computation time depends not only on the resolution of the scene but also on the nature of the scene.

6.4 Discussion and Future Work

In this paper, we presented an adaptive surface reconstruction technique for SPH by using a 3-level uniform grid. The presented grid adapts its cells depending on the surface curvature, which allows for generating less triangles in flat regions but preserving all surface details in regions with high curvature. Consequently, the number of triangles, memory consumption and the computation time decrease as we produce similar quality results in comparison to high resolution single level uniform grids.

In the future, we would like to consider using hashing for also our top level coarse grid so as to prevent unnecessary data storage for unused cells.

Parallel algorithms have been gaining attention in recent years for high performance computing. However, the incorporation of parallelization in our method is a challenging task due to potential thread safety issues that arise in hashing. Another direction for future work can be addressing these issues and parallelizing our algorithm.



1: Uniform low resolution
2: Adaptive grid
3: Uniform high resolution

Figure 6.12 – Two graphs represent the average per frame memory consumption and the computation time of the Sink scene using our method or high/low resolution uniform grid approaches.

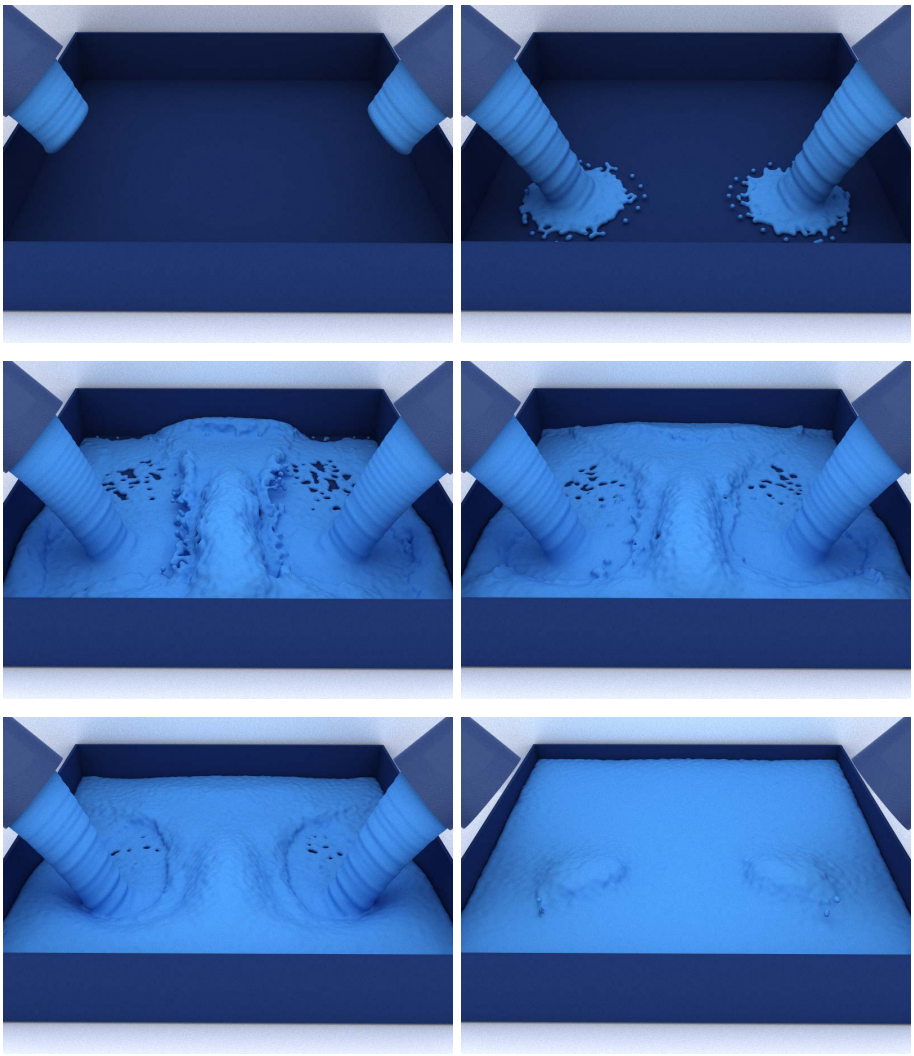


Figure 6.13 – The Two Taps scene with up to 200k particles.

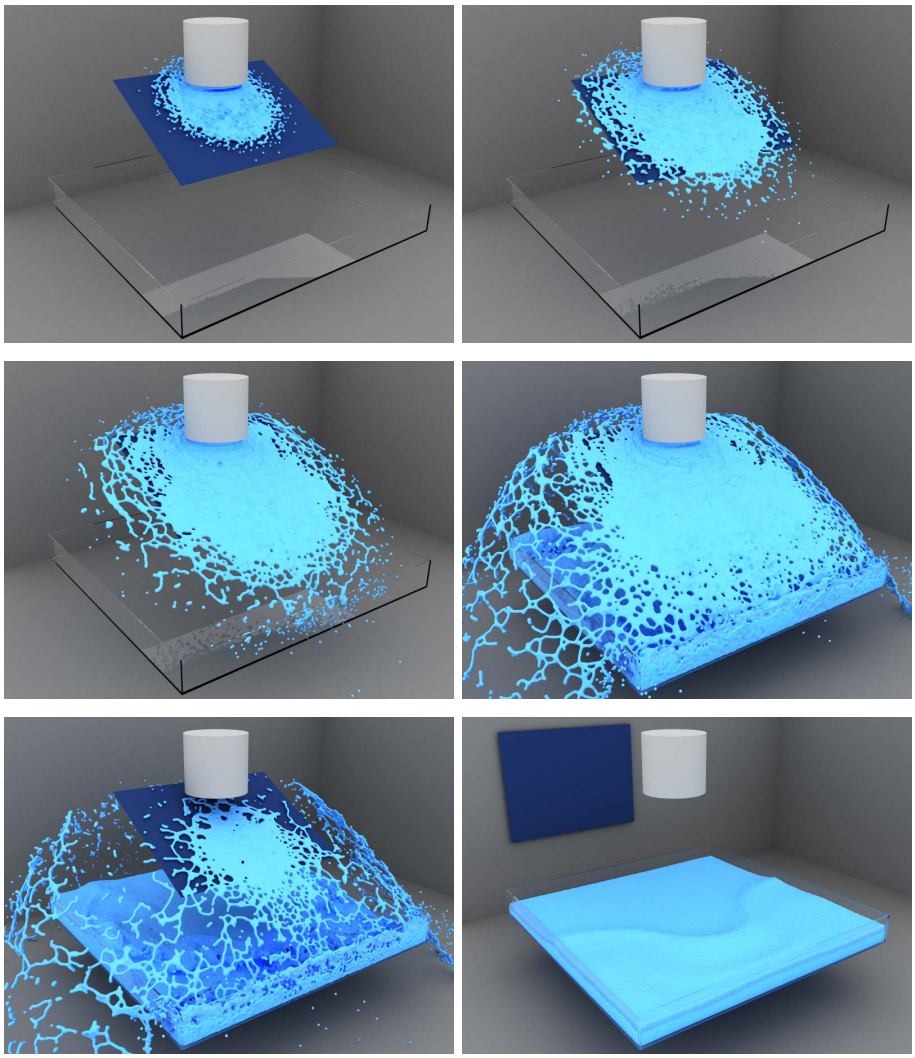


Figure 6.14 – The Splash scene with up to 500k particles.

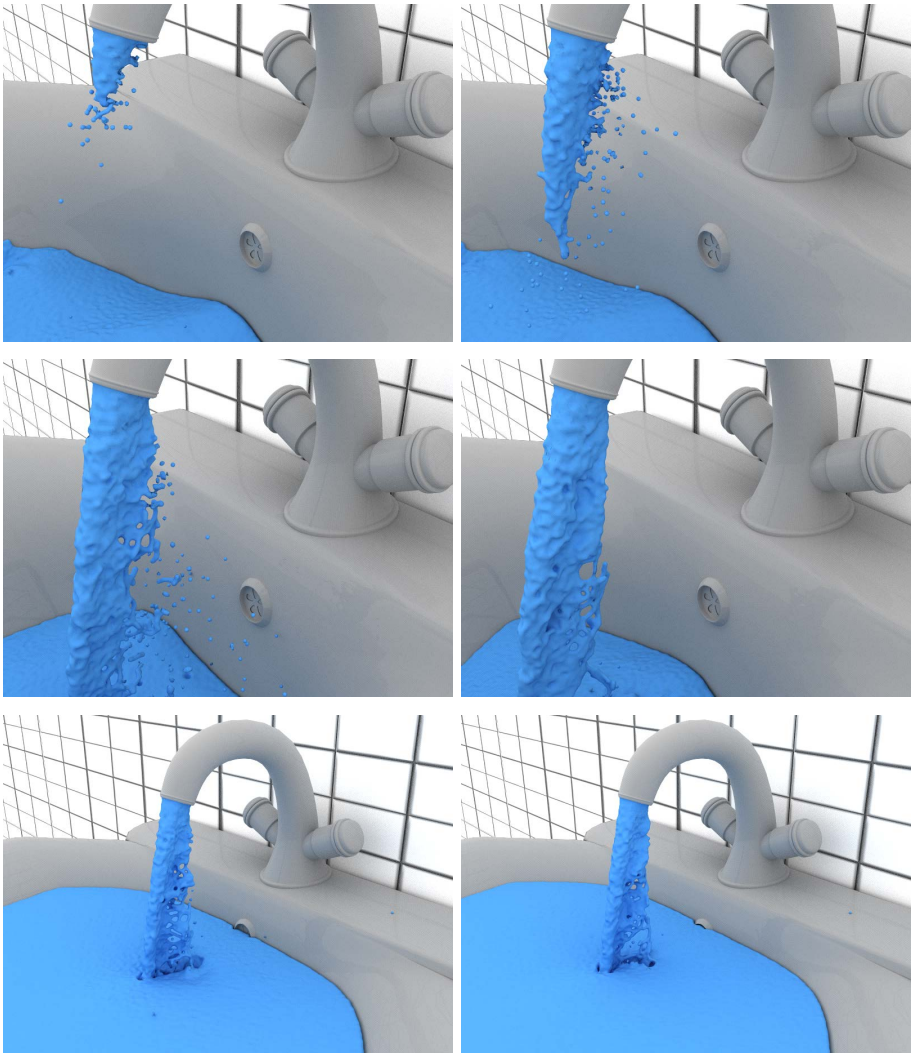


Figure 6.15 – The Sink scene with up to 2.5 million particles. Opaque rendering of the seamlessly stitched mesh is presented.

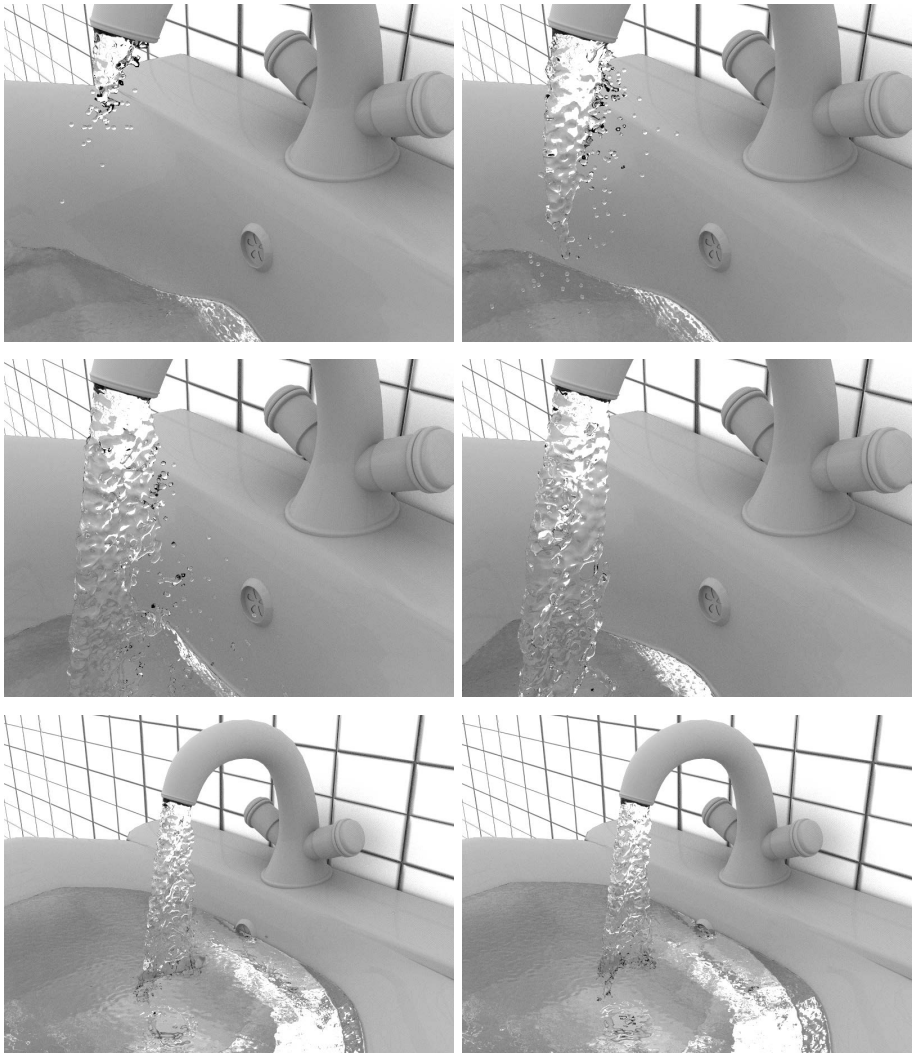


Figure 6.16 – The Sink scene with up to 2.5 million particles. Transparent rendering of the seamlessly stitched mesh is presented.

7 Conclusions

7.1 Summary

In this thesis, new methods are presented which focus on to solve some major problems of the surface reconstruction for SPH fluids. The presented methods aim to improve the computational efficiency, main memory and/or secondary storage consumption, and the resulting surface quality.

One of the extensively acknowledged problems of SPH surface reconstructions is the computational inefficiency. Although there are methods for the fast visualization of particle based fluids, e.g., [ALD06, MSD07, vdLGS09], they generally become ineffective in the visual quality or in realistic rendering step since they either generate a view-dependent open meshes or view-dependent blurred splats, e.g. quads. Therefore, in this thesis, we proposed a Marching Cubes based novel parallel algorithm that focuses on the narrow-band region where the surface actually lies in contrast to the conventional SPH surface reconstruction methods which operates on the entire fluid volume. With this method, we gained a speed-up of fifty on GPUs and our method scales nearly linearly on multi-core CPUs while we preserve the surface quality.

Another widely known problem in the SPH surface reconstruction field is the bumpiness issue due to the irregular placement of particles. Obtaining smooth surfaces depend on many variables, e.g., the chosen scalar field computation technique and its own variables, the Marching Cubes grid resolution, contributing particles per query point. However, establishing a decent set up for all those variables is a challenging and error-prone task. In this thesis, we presented many comparisons with different values of the aforementioned variables and by employing different scalar field computation techniques in order to produce the most plausible surface quality.

Obtaining smooth SPH fluid surfaces may trade off vital surface details due to potential mass loss. So as to prevent this problem, we proposed to employ two post-processing steps on the fluid surface which is reconstructed on a reasonably low resolution Marching Cubes grid. Thus, we addressed the bumpiness problem by applying a feature sensitive mesh decimation algorithm which allows the decimated mesh to remain faithful to the original topology of the reconstructed surface, while it looks smooth in flat regions. Sharp edges which raise after the decimation step are prevented by applying subdivision to the mesh. Our results demonstrate that in comparison to other approaches with comparable surface quality, our pipeline runs up to twenty times faster.

In addition to computational efficiency and visual quality, studies about

the efficiency of main memory and secondary storage consumption takes an important place in the surface reconstruction field. As a well known problem, the Marching Cubes approach requires sufficiently high resolution uniform grids in order to produce high quality surface meshes. Besides, a proper scalar field can be computed if only sufficient number of particles contribute to the query points. These issues increase the memory consumption dramatically. Therefore, in this thesis, we proposed to focus on the narrow band region while we compute the scalar field in order to scale the computational area with the surface instead of the fluid volume. We further reduced the memory consumption by lowering the Marching Cubes grid resolution since we are still able to produce high quality surfaces with the help of post-processing steps. Besides, we shrinked the computational area by excluding isolated particles that has no neighbors and reconstructing them as pre-tessellated spheres. According to our experiments, memory and secondary storage consumption is reduced by up to 80% by using our pipeline. Finally, we proposed a multi level uniform grid which adapts its cells with respect to fluid curvature. With the help of our method, flat regions are solved on low resolution cells that produce less triangles, while high resolution cells produce more triangles for highly curved fluid parts. We also proposed a technique to fill in crack regions efficiently. Our results show that with this method we gain up to four times better memory consumption.

7.2 Future Work

In this thesis, we proposed techniques to solve some of the major problems for the surface reconstruction of SPH fluids. However, there are still various issues that should be handled in the future.

- *Parallelization*

  In order to provide the necessary levels of performance, parallel computing techniques must be brought to bear. Together with recent advances in parallel architectures, many researchers have focused on the parallel computing area; and polygonization techniques are not an exception to this. Among other methods, parallelization of the Marching Cubes approach is probably the most commonly investigated polygonization method in the field, e.g., [Mac92, YC94, SDC09, CZ10, SEL11]. However, parallelization of a proper underlying scalar field computation is still a challenge due to possible thread safety issues. Although these issues have been addressed in our novel parallel algorithm, there is still space for further research. For instance, we employed general parallelization techniques that are effective on any parallel architecture. However, we could get better performance by employing any GPU specific optimization. In the future, some of these techniques, e.g., warp partitioning techniques and data tiling strategies can be investigated which can reduce the number of idle threads and the number of accesses to the global memory.

  Although we presented a novel parallel algorithm; decimation and subdivision steps of the presented pipeline and the proposed multi

level grid approach have not been parallelized which can be a direction for future work.

As the complexity of reconstructed surfaces gets higher, out-of-core techniques become more in demand and are investigated by researchers, e.g., [ZN03, NNSM07]. Efficient load balancing on distributed systems, reducing work duplication and disk seek latency are still existing issues of this type of methods that need to be further improved.

With high levels of parallelism and significant data locality, achieving good scalability rates is still a direction for future work in the field.

- *Temporal Coherence*

  Throughout this thesis, we presented methods which reconstruct SPH fluid surfaces on per-frame basis, where each frame is computed independent of the previous frames. Although we did not observe any temporal incoherence, temporal coherence can be ensured by incorporating the data that is obtained from the previous computation step as employed by Yu et al. [YWTY12] based on [WTGT09, WTGT10] in a form of surface tracking. However, surface tracking is not a simple task for particle-based fluid simulations, since particles do not have any connectivity information. Carrying additional surface information which improves the realism of the animation, e.g., foam, bubbles, textures, from one frame to the next is a challenging task. Besides, in regions where surface divides, e.g., splashing areas, small scale isolated parts, tracking becomes even harder which causes visible mass loss. These issues can be addressed in future for further improvement in the field.

- *Spatial Hashing*

  Spatial hashing is a way of indexing objects in space, which supports reduced memory footprint in contrast to regular uniform grids. Although being used for other simulation purposes, e.g., collision detection [THM$^+$03] or efficient neighbor search in fluid animation [IABT11], application of hashing in the SPH surface reconstruction field still awaits to be investigated. Some issues that need to be taken care of in hashing include, for instance: incorporation of parallelization since this gives occasion to thread safety issues due to high data dependency; resolving hash collisions with better hash functions; and retaining efficient random access.

- *Screen Space Techniques*

  Screen space techniques, e.g.,[MSD07, vdLGS09], are faster to compute when compared to conventional polygonization techniques. These approaches create no close meshes, and generated result is only view-dependent, which are acceptable for fast visualization purposes. However, a thorough investigation for improving the visual quality with such techniques, e.g., more efficient blurring techniques,

efficiency on avoiding mesh collisions, adding details such as foam, spray and bubbles, could be a useful direction for future work which make these techniques a more powerful alternative to traditional polygonization based approaches.

In this thesis, we have shown that with recent advances and proposed methods, the surface reconstruction phase goes out of being a bottleneck in SPH fluid animations. However, this field still has many space to be further investigated. We hope that our work inspires other researchers to work on the surface reconstruction field for particle-based fluids.

Bibliography

[AAAT13] N. Akinci, Dippel A., G. Akinci, and M. Teschner. Screen space foam rendering. *Journal of WSCG*, 21(3):173–182, 2013.

[AAET13] G. Akinci, N. Akinci, Oswald E., and M. Teschner. Adaptive surface reconstruction for sph using 3-level uniform grids. In *Proc. WSCG*, 2013.

[AAIT12] G. Akinci, N. Akinci, M. Ihmsen, and M. Teschner. An efficient surface reconstruction pipeline for particle-based fluids. In *Workshop on Virtual Reality Interaction and Physical Simulation*, pages 61–68. The Eurographics Association, 2012.

[AAT13] N. Akinci, G. Akinci, and M. Teschner. Versatile surface tension and adhesion for sph fluids. *ACM Trans. on Graphics*, 2013.

[ACAT13] N. Akinci, J. Cornelis, G. Akinci, and M Teschner. Coupling elastic solids with smoothed particle hydrodynamics fluids. *Journal of Computer Animation and Virtual Worlds (CAVW)*, 24(3-4):195–203, 2013.

[AIA+12] N. Akinci, M. Ihmsen, G. Akinci, B. Solenthaler, and M. Teschner. Versatile rigid-fluid coupling for incompressible SPH. *ACM Trans. on Graphics (SIGGRAPH Proc.)*, 31(4):62:1–62:8, 2012.

[AIAT12] Gizem Akinci, Markus Ihmsen, Nadir Akinci, and Matthias Teschner. Parallel surface reconstruction for particle-based fluids. *Computer Graphics Forum*, 31:1797–1809, 2012. Presented in Eurographics 2013.

[ALD06] B. Adams, T. Lenaerts, and P. Dutre. Particle Splatting: Interactive Rendering of Particle-Based Simulation Data. Technical Report CW 453, Katholieke Universiteit Leuven, 2006.

[APKG07] B. Adams, M. Pauly, R. Keiser, and L.J. Guibas. Adaptively sampled particle fluids. *ACM Trans. on Graphics (SIGGRAPH Proc.)*, 26(3):48–54, 2007.

[BBB10] Tyson Brochu, Christopher Batty, and Robert Bridson. Matching fluid simulation elements to surface geometry and topology. *ACM Trans. on Graphics (SIGGRAPH Proc.)*, 29(4):47, 2010.

[BGB11] H. BHATTACHARYA, Y. GAO, and A. W. BARGTEIL. A level-set method for skinning animated particle data. In *In roceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, 2011.

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

[Bri03] R. Bridson. *Computational aspects of dynamic surfaces.* PhD thesis, Stanford University, 2003.

[BT07] M. Becker and M. Teschner. Weakly compressible SPH for free surface flows. In *Proc. of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 209–217, 2007.

[CBP05] S. Clavet, P. Beaudoin, and P. Poulin. Particle-based viscoelastic fluid simulation. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 219–228, New York, NY, USA, 2005. ACM Press.

[CC78] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350–355, November 1978.

[Che11] Min-Bin Chen. A parallel 3d delaunay triangulation method. In *IEEE 9th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, 2011.

[CKK+12] Milosz Ciznicki, Michal Kierzynka, Krzysztof Kurowski, Bogdan Ludwiczak, Krystyna Napierala, and Jaroslaw Palczylski. Efficient isosurface extraction using marching tetrahedra and histogram pyramids on multiple gpus. In Roman Wyrzykowski, Jack Dongarra, Konrad Karczewski, and Jerzy Waniewski, editors, *Parallel Processing and Applied Mathematics*, volume 7204 of *Lecture Notes in Computer Science*, pages 343–352. Springer Berlin Heidelberg, 2012.

[CP98] S. L. Chan and E. O.; Purisima. A new tetrahedral tesselation scheme for isosurface generation. *Computers and Graphics*, 22(1):83–90, 1998.

[CPJ10] V. Costa, J. Pereira, and J. Jorge. Multi-level hashed grid construction methods. In *WSCG, Czech Republic.*, 2010.

[CZ10] Dyken Erik Christopher and Gernot Ziegler. Gpu-accelerated data expansion for the marching cubes algorithm. In *GPU Technology Conference*, 2010.

[Del34] Boris N. Delaunay. Sur la sphère vide. *Bulletin of Academy of Sciences of the USSR*, 6:793–800, 1934.

[DLG90] N. Dyn, D. Levin, and J. A. Gregory. A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control. *ACM TOG*, 9(2):160–169, April 1990.

[DS78] D. Doo and M. Sabin. Behavior of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10(6):356–360, 1978.

[DYK06] X. Du, B. Yin, and D. Kong. Feature-Preserving Simplification of Meshes Based on New Quadric Metrics. *Journal of Information & Computational Science*, pages 695–703, December 2006.

[EMF02] D. Enright, S. Marschner, and R. Fedkiw. Animation and rendering of complex water surfaces. In *In Proceedings of Computer graphics and interactive techniques*, pages 736–744, 2002.

[FAW10] R. FRAEDRICH, S. AUER, and R. WESTERMANN. Efficient high-quality volume rendering of sph data. In *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization / Information Visualization 2010)*, pages 1533–1540, 2010.

[FF01] N. Foster and R. Fedkiw. Practical animation of liquids. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 23–30, New York, NY, USA, 2001. ACM Press.

[FOK05] B. E. Feldman, J. F. OBrien, and B. M. Klinger. Animating gases with hybrid meshes. *ACM TOG (Proceedings of SIGGRAPH)*, 24(3):904–909, 2005.

[GH97] M. Garland and P. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH 1997 Proceedings*, pages 209–216, 1997.

[GH98] M. Garland and P. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *In Visualization 1998 Proceedings (1998), IEEE*, pages 263–269, 1998.

[GK03] Alexander Gress and Reinhard Klein. Efficient representation and extraction of 2-manifold isosurfaces using kd-trees. In *In proceedings of The 11th Pacific Conference on Computer Graphics and Applications*, 2003.

[GM77] R.A. Gingold and J.J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375–398, 1977.

[GSSP10] Prashant Goswami, Philipp Schlegel, Barbara Solenthaler, and Renato Pajarola. Interactive SPH Simulation and Rendering on the GPU. In *SCA '10: Proceedings of the 2010 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2010.

[Gue95] A. Gueziec. Surface simplification with variable tolerance. In *In Second Annual Intl. Symp. on Medical Robotics and Computer Assisted Surgery (MRCAS '95)*, pages 132–139, November 1995.

[HA06] X.Y. Hu and N.A. Adams. A multi-phase SPH method for macroscopic and mesoscopic flows. *Journal of Computational Physics*, 213(2):844–861, 2006.

[HDD+93] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *In SIGGRAPH '93 Proc.*, pages 19–26, August 1993.

[HI97] A. Hilton and J. Illingworth. Marching triangles: Delaunay implicit surface triangulation. Technical report, University of Surrey, 1997.

[Hop99] H. Hoppe. New quadric metric for simplifying meshes with appearance attributes. In *Proceedings of the 10th IEEE Visualization 1999 Conference (VIS '99)*, Washington, DC, USA, 1999.

[HWB04] B. HOUSTON, M. WIEBE, and C. C. BATTY. Rle sparse level sets. In *In Proceedings of the SIGGRAPH 2004 conference on sketches & applications*, New York, NY, USA, 2004.

[IAAT12] Markus Ihmsen, Nadir Akinci, Gizem Akinci, and Matthias Teschner. Unified spray, foam and air bubbles for particle-based fluids. *The Visual Computer*, pages 1–9, 2012. 10.1007/s00371-012-0697-9.

[IABT11] Markus Ihmsen, Nadir Akinci, Markus Becker, and Matthias Teschner. A parallel SPH implementation on multi-core CPUs. *Computer Graphics Forum*, 30(1):99–112, 2011.

[IAGT10] M. Ihmsen, N. Akinci, M. Gissler, and M. Teschner. Boundary handling and adaptive time-stepping for PCISPH. In *Proc. of VRIPHYS*, pages 79–88, 2010.

[IBAT11] M. Ihmsen, J. Bader, G. Akinci, and M. Teschner. Animation of air bubbles with SPH. In *International Conference on Graphics Theory and Application*, pages 225–234, 2011.

[ICS+13] M. Ihmsen, J. Cornelis, B. Solenthaler, C. Horvath, and M. Teschner. Implicit incompressible sph. *IEEE T*, 2013.

[IWT12] M. Ihmsen, A. Wahl, and M Teschner. High-resolution simulation of granular material with sph. In *Proc. VRIPHYS*, pages 53–60, December 2012. Best Paper Award.

[IWT13] M. Ihmsen, A. Wahl, and M Teschner. A lagrangian framework for simulating granular material with high detail. *Computers & Graphics*, 38(5):1–10, 2013.

[JU06] T. Ju and T. Udeshi. Intersection-free contouring of an octree grid. In *In Proceedings of Pacific Graphics*, 2006.

[KAD+06] R. Keiser, B. Adams, P. Dutré, L.J. Guibas, and M. Pauly. Multiresolution particle-based fluids. Technical report, ETH Zurich, 2006.

[KKA05] Joseph Kohout, Ivana Kolingerova, and Jiri Ara. Parallel delaunay triangulation in e2 and e3 for computers with shared memory. *Parallel Computing*, 31(5):491–522, 2005.

[KSN08] Y. Kanamori, Z. Szego, and Nis. Gpu-based fast ray casting for a large number of metaballs. *Computer Graphics Forum*, 27(2):351–360, 2008.

[KW06] P. Kipfer and R. Westermann. Realistic and interactive simulation of rivers. *Proceedings of the 2006 conference on Graphics interface*, pages 41–48, 2006.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21:163–169, 1987.

[LD08a] A. Lagae and P. Dutré. Compact, fast and robust grids for ray tracing. In *In ACM SIGGRAPH 2008 talks (SIGGRAPH 2008). Article 20 , 1 pages.*, ACM, New York, NY, USA„ 2008.

[LD08b] T. Lenaerts and P. Dutré. Unified SPH model for fluid-shell simulations. In *ACM SIGGRAPH 2008 posters*, SIGGRAPH '08, pages 12:1–12:1, 2008.

[Len10] E. Lengyel. Transition cells for dynamic multiresolution marching cubes. *Journal of Graphics, GPU, and Game Tools*, 15(2):99–122, 2010.

[LIGF06] F. Losasso, G. Irving, E. Guendelman, and R. Fedkiw. Melting and Burning Solids into Liquids and Gases. *IEEE TVCG*, 12:342–352, 2006.

[Loo87] C. Loop. Smooth Subdivision surfaces based on triangles. Master's thesis, University of Utah, 1987.

[LPP97] Sangyoon Lee, Chan-Ik Park, and Chan-Mo Park. An improved parallel algorithm for delaunay triangulation on distributed memory parallel computers. In *Advances in Parallel and Distributed Computing, 1997. Proceedings*, 1997.

[LSJ96] Y. Livnat, H.W. Shen, and C. R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE TVCG*, 2(1):73–84, 1996.

[LTKF08] F. Losasso, J. Talton, N. Kwatra, and R. Fedkiw. Two-Way Coupled SPH and Particle Level Set Fluid Simulation. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):797–804, 2008.

[Luc77] L.B. Lucy. A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal*, 82:1013–1024, 1977.

[LY04] H. Lee and H. S. Yang. Real-time marchingcube-based lod surface modeling of 3d objects. In *ICAT*, 2004.

[Mac92] Paul Mackerras. A fast parallel marching-cubes implementation on the fujitsu ap1000. Technical report, The Australian National University, 1992.

[Mau02] Pavel Maur. Deladela triangtriangu in 3d. Technical report, University of West Bohemia in Pilsen, 2002.

[MCG03] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 154–159, 2003.

[MCP+09] P. Mullen, K. Crane, D. Pavlov, Y. Tong, and M Desbrun. Energy-preserving integrators for fluid animation. *ACM TOG (Proceedings of SIGGRAPH)*, 28(3):38:1–38:8, 2009.

[MG83] JJ Monaghan and RA Gingold. Shock simulation by the particle method SPH. *Journal of Computational Physics*, 52(2):374–389, 1983.

[MK99] JJ Monaghan and A. Kos. Solitary waves on a Cretan beach. *Journal of Waterway, Port, Coastal, and Ocean Engineering*, 125:145, 1999.

[ML85] JJ Monaghan and JC Lattanzio. A refined particle method for astrophysical problems. *Astronomy and astrophysics*, 149:135–143, 1985.

[MM97] J.P. Morris and J.J. Monaghan. A Switch to Reduce SPH Viscosity. *Journal of Computational Physics*, 136(1):41–50, 1997.

[Mon94] J.J. Monaghan. Simulating free surface flows with SPH. *Journal of Computational Physics*, 110(2):399–406, 1994.

[Mon05] J.J. Monaghan. Smoothed particle hydrodynamics. *Reports on Progress in Physics*, 68(8):1703–1759, 2005.

[MS10] J. Manson and S. Schaefer. Isosurfaces over simplicial partitions of multiresolution grids. *Computer Graphics Forum*, 29(2):377–385, 2010.

[MSD07] M. Müller, S. Schirm, and S. Duthaler. Screen Space Meshes. In *Proceedings of ACM SIGGRAPH / EUROGRAPHICS Symposium on Computer Animation (SCA)*, 2007.

[MSKG05] M. Müller, B. Solenthaler, R. Keiser, and M. Gross. Particle-based fluid-fluid interaction. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 237–244, New York, NY, USA, 2005. ACM.

[MST+04] M. Müller, S. Schirm, M. Teschner, B. Heidelberger, and M. Gross. Interaction of fluids with deformable solids. *Computer Animation and Virtual Worlds*, 15(34):159–171, 2004.

[Ngu07] Hubert Nguyen. *Point-Based Visualization of Metaballs on a GPU*, chapter 7. Addison-Wesley Professional, 2007.

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

[NNSM07] M. B. NIELSEN, O. NILSSON, A. SÖDERSTRÖM, and K. MUSETH. Out-of-core and compressed level set methods. *ACM Transactions on Graphics*, 26(4), 2007.

[NVI11] NVIDIA. mental ray (version 3.9). *http://www.mentalimages.com*, 2011.

[OCR11] J. ONDERIK, M. CHLADEK, and DURIKOVIC R. Sph with small scale details and improved surface reconstruction. In *In In SCCG 2011: Proceedings of the Spring Conference on Computer graphics*, 2011.

[ODAF06] G. Oger, M. Doring, B. Alessandrini, and P. Ferrant. Two-dimensional SPH simulations of wedge water entries. *Journal of Computational Physics*, 213(2):803–822, 2006.

[OKR09] Seungtaik Oh, Younghee Kim, and Byung-Seok Roh. Impulse-based rigid body interaction in SPH. *Comput. Animat. Virtual Worlds*, 20(2-3):215–224, 2009.

[OR97] M. Ohlberger and M. Rumpf. Hierarchical and adaptive visualization on nested grids. *Computing*, 59 (4):269–285, 1997.

[Pov10] *POV-Ray Documentation, Version 3.7 beta*, 2010.

[PSLH98] Steven Parker, Peter Shirley, Yarden Livnat, and Peter-Pike. Hansen, Charles andSloan. Interactive ray tracing for isosurface rendering. In *In Proceedings of the Conference on Visualization 1998*, 1998.

[RB93] J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering complex scenes. In *Modeling in Computer Graphics: Methods and Application*, pages 455–465, 1993.

[Roy95] Trina M. Roy. Physically based fluid modeling using smoothed particle hydrodynamics. Master's thesis, University of Illinois at Chicago, 1995.

[SDC09] L.A. Schmitz, C.A. Dietrich, and J.L.D. Comba. Efficient and high quality contouring of isosurfaces on uniform grids. In *Computer Graphics and Image Processing (SIBGRAPI)*, 2009.

[SEL11] Erik Smistad, Anne C. Elster, and Frank Lindseth. Fast surface extraction and visualization of medical images using opencl and gpus. In *The Joint Workshop on High Performance and Distributed Computing for Medical Imaging (HP-MICCAI)*, 2011.

[SFYC96] R. Shekhar, E. Fayyad, R. Yagel, and J. F. Cornhil. Octree-based decimation of marching cubes surfaces. In *In Proceedings of the 7th conference on Visualization*, 1996.

[Sig06] C. Sigg. *Representation and rendering of implicit surfaces.* PhD thesis, ETH Zurich., 2006.

[Slo87] S. W. Sloan. A fast algorithm for constructing delaunay ttriangulation in the plane. *Adv. Eng. Software*, 9(1):34–55, 1987.

[SP08] B. Solenthaler and R. Pajarola. Density Contrast SPH Interfaces. In *Proc. of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 211–218, 2008.

[SP09] B. Solenthaler and R. Pajarola. Predictive-corrective incompressible SPH. *ACM Trans. on Graphics (SIGGRAPH Proc.)*, 28(3):1–6, 2009.

[SSP07] B. Solenthaler, J. Schläfli, and R. Pajarola. A unified particle model for fluid-solid interactions. *Computer Animation and Virtual Worlds*, 18(1):69–82, 2007.

[Sta99] J. Stam. Stable fluids. *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, 1999.

[SVK+] Claudia Simion, Claudio Dalla Vecchia, Sadegh Khochfar, Klaus Reuter, and Markus Rampp. Visualization of sph data with delaunay tessellation. Electronically.

[SZL92] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. In *Computer Graphics(SIGGRAPH '92 Proc.)*, volume 26(2), pages 65–70, 1992.

[TFK+03] T. Takahashi, H. Fujii, A. Kunimatsu, K. Hiwada, T. Saito, K. Tanaka, and H. Ueki. Realistic Animation of Fluid with Splash and Foam. *Computer Graphics Forum*, 22(3):391–400, 2003.

[THM+03] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, and M. Gross. Optimized Spatial Hashing for Collision Detection of Deformable Objects. In *Proc. of Vision, Modeling, Visualization (VMV)*, pages 47–54, 2003.

[TPG98] G. M. Treece, R. W. Prager, and A.H. Gee. Regularised marching tetrahedra: Improved iso-surface extraction. *Computers and Graphics*, 23:583–598, 1998.

[vdLGS09] W.J. van der Laan, S. Green, and M. Sainz. Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 91–98. ACM, 2009.

[VKS+04] G. Varadhan, S. Krishnan, T. Sriram, , and D. Manocha. Topology preserving surface extraction using adaptive subdivision. In *In Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing (SGP 2004).*, pages 235–244, 2004.

[VT01] F. Velasco and J. C. Torres. Cell octrees: A new data structure for volume modeling and visualization. In *In Proceedings of the Vision Modeling and Visualization Conference 2001 (VMV 2001)*, pages 151–158, 2001.

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

[WKE99] R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of iso-surfaces. *The Visual Computer*, 15:100–111, 1999.

[WTGT09] Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. Deforming meshes that split and merge. *ACM Trans. Graph.*, 28:76:1–76:10, 2009.

[WTGT10] Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. Physics-inspired topology changes for thin fluid features. *ACM Trans. on Graphics (Proc. SIGGRAPH)*, 29(3), 2010.

[WVG92] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM TOG*, 11(3):201–227, 1992.

[YC94] Terry S. Yoo and David T. Chen. Interactive 3d medical visualization: A parallel approach to surface rendering 3d medical data. In *In Proceedings of the Symposium for Computer Assisted Radiology*, 1994.

[YT10] J. Yu and G. Turk. Reconstructing surfaces of particle-based fluids using anisotropic kernels. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 217–225. Eurographics Association, 2010.

[YWH+09] H. Yan, Z. Wang, J. He, X. Chen, C. Wang, and Q. Peng. Real-time fluid simulation with adaptive SPH. *Computer Animation and Virtual Worlds*, 20(2-3):417–426, 2009.

[YWTY12] Jihun Yu, Chris Wojtan, Greg Turk, and Chee Yap. Explicit mesh surfaces for particle based fluids. In *Computer Graphics Forum (Eurographics Proc.)*, volume 31, pages 815–824. Wiley Online Library, 2012.

[ZB05] Y. Zhu and R. Bridson. Animating sand as a fluid. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 965–972, New York, NY, USA, 2005. ACM Press.

[ZGHG11] K. Zhou, M. Gong, X. Huang, and B. Guo. Data-parallel octrees for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 17(5):669–681, 2011.

[ZN03] Huijuan Zhang and Timothy S. Newman. Efficient parallel out-of-core isosurface extraction. In *IEEE Symposium on Parallel and Large Data Visualization and Graphics*, 2003.

[ZPvBG01] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 371–378, New York, NY, USA, 2001. ACM Press.

[ZSP07] Yanci Zhang, Barbara Solenthaler, and Renato Pajarola. Gpu accelerated sph particle simulation and rendering. In *ACM SIGGRAPH 2007 Posters*, 2007.

[ZSP08] Y. Zhang, B. Solenthaler, and R. Pajarola. Adaptive Sampling and Rendering of Fluids on the GPU. In *Proceedings Symposium on Point-Based Graphics*, pages 137–146, 2008.

[ZSS96] D. Zorin, P. Schroder, and W. Sweldens. Interpolating Subdivisions for Meshes with Arbitrary Topology. In *Computer Graphics Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH*, pages 189–192, 1996.

Curriculum Vitae

Personal Information

Name	Gizem Akinci
Date of birth	09.01.1985
Place of birth	Mugla/Turkey

Education

- 2010-2014 PhD. in Computer Science in University of Freiburg/Germany

  Subject of dissertation: "Efficient Surface Reconstruction for SPH Fluids"

  Advisor: Prof. Dr. Matthias Teschner, University of Freiburg

- 2007-2010 MSc. in Computer Science in University of Freiburg/Germany
- 2005-2007 BSc. in Computer Eng. with scholarship in Atılım University/Turkey
- 2005 Transfer to Computer Engineering department
- 2003-2005 BSc. in Industrial Eng. with scholarship in Atılım University/Turkey

Teaching

- Tutorial: Simulation in Computer Graphics, Prof. Matthias Teschner, 2012/2013
- Tutorial: Simulation in Computer Graphics, Prof. Matthias Teschner, 2011/2012

Supervised Theses

- Edgar Oswald. Räumlich adaptive Gitter für SPH-Oberflächenrekonstruktionen. May 2012, Master's Thesis
- Alexander Dippel. Surface Splatting für Dynamische SPH-Animationen. March 2011, Bachelor Thesis.

Publications

- N. Akinci, G. Akinci, M. Teschner. Versatile Surface Tension and Adhesion for SPH Fluids. ACM Transactions on Graphics (Proc. SIGGRAPH Asia 2013), Nov. 2013.
- G. Akinci, N. Akinci, E. Oswald, M. Teschner. Adaptive Surface Reconstruction for SPH using 3-Level Uniform Grids. WSCG proceedings, pp.195-204, Union Agency, 2013
- N. Akinci, A. Dippel, G. Akinci, M. Teschner. Screen Space Foam Rendering. Journal of WSCG, Vol.21, No.03, pp.173-182, Union Agency, 2013

- N. Akinci, J. Cornelis, G. Akinci, M. Teschner. Coupling Elastic Solids with SPH Fluids. Journal of Computer Animation and Virtual Worlds (CAVW), vol. 24, no. 3-4, pp. 195-203, CASA 2013 Special Issue, 2013.
- N. Akinci, M. Ihmsen, G. Akinci, B. Solenthaler, M. Teschner. Versatile Rigid-Fluid Coupling for Incompressible SPH. ACM Transactions on Graphics (Proc. SIGGRAPH 2012), vol. 31, no. 4, pp. 62:1-62:8, July 2012.
- G. Akinci, N. Akinci, M. Ihmsen, M. Teschner. An Efficient Surface Reconstruction Pipeline for Particle-Based Fluids. Proc. VRIPHYS, Darmstadt, Germany, pp. 61-68, Dec. 6-7, 2012
- G. Akinci, M. Ihmsen, N. Akinci, M. Teschner. Parallel Surface Reconstruction for Particle-based Fluids. Computer Graphics Forum, vol. 31, no. 6, pp. 1797-1809, 2012, doi: 10.1111/j.1467-8659.2012.02096.x.
- M. Ihmsen, N. Akinci, G. Akinci, M. Teschner. Unified Spray, Foam and Bubbles for Particle-based Fluids. The Visual Computer , vol. 28, no. 6-8, pp. 669-677, 2012, doi: 10.1007/s00371-012-0697-9
- M. Ihmsen, J. Bader, G. Akinci, M. Teschner. Animation of Air Bubbles with SPH. Int. Conf. on Computer Graphics Theory and Applications GRAPP 2011, pp. 225-234, March 5-7, 2011.

Misc

- G. Akinci . Smooth Surface Reconstruction for SPH . August 2010, Master's thesis
- N. Akinci, G. Kayar , B. Özdemir. Faster Calculation of Ray Tracing with Some Other Lighting Models Using GLSL. June 2007, Bachelor Thesis.