Sparse Paged Grid and its Applications to Adaptivity and Material Point Method in Physics Based Simulations

by

Ming Gao

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2018

Date of final oral examination: 07/03/2018

The dissertation is approved by the following members of the Final Oral Committee:

Eftychios Sifakis, Associate Professor, University of Wisconsin - Madison, Computer Sciences
Michael Gleicher, Professor, University of Wisconsin - Madison, Computer Sciences
Dan Negrut, Professor, University of Wisconsin - Madison, Mechanical Engineering
Jean-Luc Thiffeault, Professor, University of Wisconsin - Madison, Mathematics
Chenfanfu Jiang, Assistant Professor, University of Pennsylvania, Computer and Information Science

To my wife, my family and my friends.

ACKNOWLEDGMENTS

First of all, I would like to thank and acknowledge my Ph.D. advisor, Professor Eftychios Sifakis, for his guidance, support, and motivation over the years. His vision and expertise in research have opened my eyes and enthused me to pursue more in academia; his advise and suggestions also help me get better prepared for future challenges. I also thank my committee members, Professor Michael Gleicher, Professor Dan Negrut, Professor Jean-Luc Thiffeault and Professor Chenfanfu Jiang for providing insightful suggestions for my thesis.

During my Ph.D. studies, I have been fortunate to work with excellent researchers and collaborators: Nathan Mitchell, Tiantian Liu, Lifeng Zhu, Professor Ladislav Kavan, Professor Mridul Aanjaneya, Haixiang Liu, Professor Christopher Batty, Andre Tampubolon, Xuchen Han, Qi Guo, Grant Kot, Xinlei Wang, Kui Wu and Professor Cem Yuksel. Many thanks to my Lab colleagues at UW-Madison: Raj Setaluri, Qisi Wang and Mike Doescher for their help and advise.

Finally, I would like to express my gratitude to my wife and my family. To my parents and my brother: Thank you for always being supportive and considerate. To my wife: Thank you for your endless love and always being patient with me. Without your meticulous care, it would be impossible for me to get through the time when I was unable to see things after the retinal detachment surgery last year. Without your accompany and encouragement, it would also be impossible for me to achieve this far and always be concentrating.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

With a focus and concentration on the simulations of real-world phenomena and materials, physics-based animation has been established as an important research area in computer graphics. However, the focus has been slightly shifted from methods for simulating new phenomena to methods for simulating existing phenomena with higher-resolution and higher-efficiency. This thesis focuses on Sparse Paged Grid (SPGrid), a sparse data structure which leverages modern operating system and hardware for compact storage and efficient stream processing, as well as its applications in physics-based simulation techniques, from pure Eulerian methods for simulating fluid dynamics to particle-grid hybrid methods (e.g., the material point method or MPM) for animating elastic, elastoplastic and granular materials.

First, by locally adapting T-junctions to power diagrams, we can recover the second-order accuracy for octree-based fluid simulations even when the free surface or the fluid-solid interface gets refined. The excellent affinity to SPGrid allows us to easily scale to tens of millions of degrees of freedom on a single computer.

Along with a new adaptive basis particularly designed for adaptive material point method, we then develop a practical strategy for particle-grid transfers that leverages the SPGrid for storing sparse multi-layered grids, leading to an efficient MPM solver that concentrates most of the computation resources in specific regions of interest.

We further present a mixed explicit and semi-implicit method for simulating particle-laden flows which accurately captures the interactions between the fluid and the sediment particles. Both species are modeled by MPM discretization, thus SPGrid again manages to improve the performance by efficiently storing and transferring information between particles and the background grids.

Finally, we introduce a novel, efficient, and memory-friendly GPU algorithm for accelerated MPM simulation on a GPU-tailored sparse storage variation of SPGrid. Our explicit and implicit GPU MPM solvers are further equipped with a heat solver and a novel sand constitutive model to enable fast simulations of a wide range of materials.

1 INTRODUCTION

The objective of this thesis is to investigate *algorithmic techniques*, *data structures* and *implementation practices* to bridge the divide between the pursuit of versatile features and optimized implementation in certain instances of physics-based simulation. The application scope in the techniques we target is focused on visual simulations of solids and fluids in a range of interactions supported by the popular material point method (MPM, e.g., [Stomakhin et al., 2013]). The cornerstone of this pursuit is the sparse paged grid (SPGrid) [Setaluri et al., 2014], an optimized data structure for sparse array storage and high throughput stream processing. We first examine the applications of SPGrid to adaptive discretizations, and then specifically concentrate on exploiting SPGrid to accelerate the MPM pipelines in modern parallel computing hardware.

This thesis has a two-fold claim for its merit of contribution. First, we demonstrate quantitative gains in performance on established simulation tasks, in scenarios that were previously thought to preclude aggressive optimizations. Second, we deliver qualitative improvements by proposing solutions to known deficiencies, artifacts, or impediments to visual accuracy in earlier work.

1.1 Motivation

While simulating new materials or phenomena is invariably the primary target for researchers in the physics-based simulation community, more recent attention has been directed to performance improvements as the demand for simulations with steadily expanding geometrical complexity has become increasingly urgent. This trend is largely industry-driven and covers various subjects, e.g. a sparse data storage scheme (OpenVDB) from Dreamworks [Museth et al., 2013], a multi-threading character simulation technique (PhysGrid) from Walt Disney Animation Studios [Milne et al., 2016] and a distributed solver system for simulations from Side Effects [Lait, 2016]. Smoke is one of the most popular phenomena which has been widely used in both visual effects and video games. It is noticeable that the resolution of the background grid for simulating smokes has been increased from 100^3 [Fedkiw et al., 2001] to 2000^3 [Setaluri et al., 2014]; the latter succeeds in providing the most meticulous details while the former would not be acceptable nowadays. Another example is simulating cloth whose dynamics is generally difficult to correctly capture due to the tremendous

amount of intricate self-collisions. A GPU based method proposed by Tang et al. [2016] is capable of generating one frame within a few seconds for extremely complex scenes.

Even the focus of simulating new featured phenomena has been constantly varying. While most of the earlier work studied the features involved in a *narrow, often isolated* type of material or phenomena, the researchers start to simulate *multi-physics* interactions, e.g. solid-fluid coupling [Robinson-Mosher et al., 2008], solid-fluid phase-changing [Stomakhin et al., 2014], and porous granular material [Tampubolon et al., 2017]. In other words, rather than modeling a completely new phenomenon, there is more and more interest in combining components of different modalities in a single framework to reproduce complicated effects in the real world, e.g. landsliding, liquid blending, volcano eruption, dune migration, which are impossible to capture by a single material. As these interactions usually require us to simultaneously model multiple materials [Guendelman et al., 2005] or solve an enlarged coupling system [Robinson-Mosher et al., 2008], the computational cost could grow tremendously. In such scenarios, performance again becomes a critical concern.

Given those reasons, today, the focus of physics-based simulation has become more balanced between the features and the performance. While new features are still always receiving attention, different kinds of methods for improving performance (i.e., saving computational cost) are now also notable. However, performance enhancements usually do not come at no cost; compromises occur in physical accuracy, regularity of data structures, parallelism potential and even in the quality of the final visual results. For instance, Stam [1999] proposes the popular semi-Lagrangian method to solve the advection equation in fluid simulations. This method is unconditionally stable; thus the computational time step can be increased. As a result, the overall computational cost is largely reduced. The accompanying disadvantage is that the accuracy of this method is downgraded to first-order such that the final visual result can suffer from excessive numerical dissipation and subdued dynamics. Although it seems to be a notable sacrifice, this method still gets widely accepted in the graphics community. In this example, the accuracy is a desired property for acquiring the non-dissipative behavior of fluid dynamics; but it is traded away for the improved performance. Throughout this thesis, I will use the word *feature* interchangeably for its original meaning (the particular material or phenomenon, here, fluid) as well as the corresponding sacrificed property for the exchange of performance (here, higher order accuracy).

It's certainly impossible to earn great performance without the upgrades and advancements of the computational resources themselves. However, considering the CPU/GPU clock rate has almost ceased to increase for a few years, it is actually quite challenging to get the performance required for executing such simulations, especially when one can expect the desired resolution or the complexity of the scene will continue to expand. Parallel computing is a powerful tool for mitigating this problem. I will also explore the possibility of applying two different classes of parallel computing techniques to material point method, namely SIMD (single instruction multiple data) and SIMT (single instruction multiple threads) to largely improve the performance.

In this thesis, my key contributions are listed as follows :

- The development of an efficient and scalable framework for simulating fluids on adaptive-resolution discretizations. The approach mimics the storage data structure of traditional adaptive grids (e.g. octrees), but hybridizes this concept with higher order accuracy representations, namely power diagrams, to alleviate a host of well-known simulation artifacts.

- The derivation as well as the application of a novel basis function that interpolates physical attributes between particles and adaptive background grids. The scheme represents a traditional octree as a pyramid of sparse uniform grids and embeds finer data into coarser levels of the discretization such that efficient uniform grid operations can be used as the building blocks which are further optimized by vectorization techniques.

- The development of a multi-material pipeline for simulating particle-laden flows that is capable of reproducing effects including dune migration, sedimentation, and sediment transportation. The method exploits the hypothesis that multiple materials can co-exist at any spatial point to eligibly resolve the interactions between them.

- The proposal of a highly effective GPU acceleration scheme for particle-to-grid interpolation on sparse background grids. Unlike almost all previous methods which map grid nodes to GPU threads, the method proves that the opposite direction, mapping particles to threads, can achieve superior performance. Specifically we exploit GPU hardware intrinsic instructions to minimize the usage of atomic operations, alleviate the write hazards, and reduce the memory footprint.

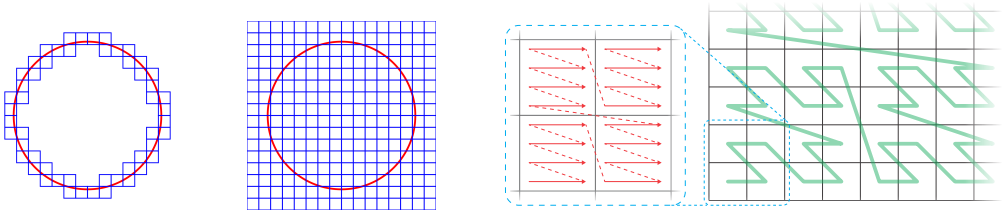
1.2 Sparse paged grid (SPGrid)

All works in this thesis build upon a recently proposed sparse data structure - Sparse Paged Grid (SPGrid) [Setaluri et al., 2014]. SPGrid can reduce the storage footprint just as many other sparse data storage schemes, e.g. OpenVDB [Museth et al., 2013], a popular sparse storage scheme widely accepted in the graphics community. However, SPGrid is more dedicated to simulation related applications, which usually involve heavy sequential and stencil access on data. On the other hand, OpenVDB is applied to various kinds of graphics utilizations, like rendering [Museth, 2014], image processing [Museth, 2013], and cloud modeling [Miller et al., 2012].

Unlike most of the other sparse data structures, SPGrid relies heavily on established software and hardware acceleration mechanisms. For example, SPGrid exploits the `mmap` function in Linux to reserve a large memory span in the virtual memory space without allocating any physical memory. The grid is organized by blocks (typicall 4KB in size), and only blocks ever touched will be mapped to the physical memory space. As shown in Figure 1.1a, although a dense grid is reserved in the virtual memory space, we only allocate physical memory for the blocks (one block maps to one memory page in SPGrid) intersecting with the circle shaped narrow band, leading to a total memory footprint of 52 pages. A similar dense grid (Figure 1.1b) would require 256 memory pages. SPGrid heavily depends on the hardware to retrieve a physical page/a block unless page faults are triggered, which by design could only happen at the precomputation stage.

For sparsely populated grids, the irregular memory access pattern, especially for stencil operations, could easily diminish the prefetching efficiency. To alleviate this problem, a combination of the lexicographic ordering (inside a block) and the Morton code ordering (across blocks) is utilized to improve the data locality to fully exploit the hardware prefetching even for stencil kernels (Figure 1.1c). Given this mapping, translation between the 2D/3D geometric index of a cell and its linear memory address (64-bit offset in SPGrid) in the virtual space can be conducted by bit-interleaving and bit-packing respectively. Furthermore, a hardware instruction (`pdep`) provided on the Haswell architecture can be utilized to significantly reduce the cost of bit-interleaving operations.

Given a particular cell, one needs to access data from its neighboring cells for performing stencil computations. On a dense grid, data of neighboring cells are stored in the memory with a constant offset/distance; while on a sparse grid, it becomes intricate to directly deduce the addresses of cells



(a) SPGrid (b) Dense grid (c) SPGrid mapping scheme

Figure 1.1: **SPGrid.** *Left:* SPGrid in physical memory space; *Middle:* SPGrid in virtual memory space or a corresponding dense grid; *Right:* A combination of lexicographic ordering and Morton coding ordering.

in the vicinity. The simple alternative is to first translate the memory address to its corresponding geometric index, compute the index of its neighboring cell, and then translate back to the offset. However, it is too costly to perform this whole process considering that stencil computations are usually high density operations in applications of grid-based simulations. SPGrid proposes a way to effectively decrease the cost of directly computing the neighboring offsets without translating to geometric index. As a result, SPGrid outperforms OpenVDB in terms of both the sequential and stencil benchmarks on sparse datasets; furthermore, SPGrid reaches about 70% of the throughput of the standard C++ dense storage and access [Setaluri et al., 2014].

For adaptive grids, SPGrid represents an octree as a pyramid of sparsely populated uniform grids (SPGrids of different resolutions) and geometrically overlays the finer cells with their parents to improve the communication between different levels of the octree. Based on this strategy, by carefully designing how to exchange data between different resolutions, we can confine all crossing-level computations to be inside a few narrow bands such that their impacts on the overall performance can be minimized. As demonstrated in Setaluri et al. [2014], SPGrid manages to reduce the cost per degree of freedom on an adaptive grid to be comparable to the cost on a uniform grid, which is uncommon in most previous adaptive data storage schemes.

1.3 Material point method (MPM)

The majority of this thesis adopts material point method (MPM) as the fundamental discretization method, thus I also include a brief introduction of MPM in this section. Sections in later parts of this dissertation review the relevant formalizations and modifications enacted by my work. In the graph-

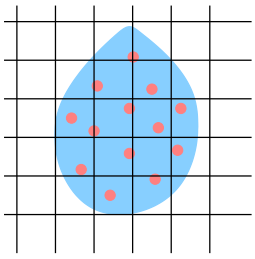
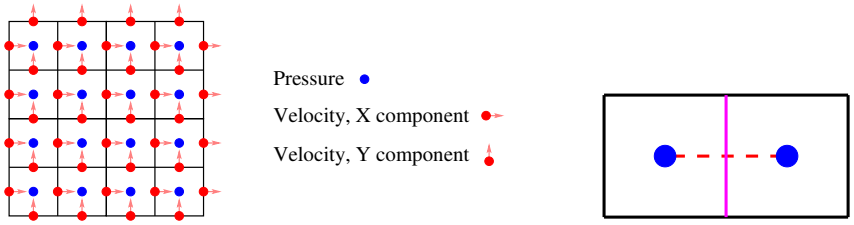


Figure 1.2: **Hybrid discretization:** Particles carry all the physical properties while the background grid acts as a scratch-pad to solve momentum balance.

ics community (both industry and academia), the material point method is an emerging simulation technique, which has been proven to be an effective choice for simulating various materials, including snow [Stomakhin et al., 2013], foam [Yue et al., 2015; Ram et al., 2015], sand [Daviet and Bertails-Descoubes, 2016; Klár et al., 2016], cloth [Jiang et al., 2017a] and solid-fluid mixture [Stomakhin et al., 2014].

MPM is a particle-grid hybrid method that combines the advantages of both Lagrangian and Eulerian methods as shown in Fig. 1.2. MPM particles carry all the physical properties like mass, momentum, deformation gradient, etc. In MPM, the grid serves mainly as a continuum scratch-pad, via which particles can communicate with each other without the costly neighboring-search operation as in mesh-less methods like SPH [Gingold and Monaghan, 1977]. In a typical MPM simulation pipeline, the physical attributes are first transferred from the particles to the Eulerian grid via weighting functions. Then we can conduct either implicit or explicit time integration on the grid for solving the momentum balance. After updating information on the grid, the values of the physical properties are transferred back to the particles for performing advection to finish a complete cycle. The grid resolution is usually far less than the number of particles, reducing the computational cost to a reasonable amount. These properties suggest that MPM is a useful benchmark for investigating sparsity and adaptivity. As Section 1.4 and Chapter 3 focus on the adaptivity, in this section we pay more attention to a uniform but sparsely populated grid.

In typical MPM applications, e.g. simulating sand dynamics, the Lagrangian particles tend to move across a huge space; but at each single time step, the discrete particles always represent a relatively sparse subset of the whole domain, which indicates the usage of a sparse storage scheme for the background grid. Furthermore, the computation of the weighting function is a stencil operation;



(a) MAC grid (b) Orthogonality

Figure 1.3: **Orthogonality in MAC grid.** *Left:* MAC discretization for Poisson solver; *Right:* Uniform orthogonality - the line connecting neighboring pressures is perpendicular to the cell face, and is also parallel to the velocity component stored at that face.

given a single particle, it needs to access 27 grid nodes (e.g. for the quadratic kernel in 3D) in its immediate vicinity.

In this thesis, I explore the possibility of using SPGrid as the MPM background grid to reduce the memory footprint as well as improving the performance for both sequential and stencil computations. Moreover, as the two transfer kernels, i.e. particle-to-grid and grid-to-particle, are identified as the bottleneck of the MPM pipeline, two effective optimization schemes, based on modern CPU and GPU hardware respectively, are outlined in Section 1.5 and further detailed in Chapter 3 and Chapter 5.

1.4 Adaptivity

For the same number of degrees of freedom, adaptive discretizations always generate more visual richness compared to uniform discretizations since most of the computing resources can be dedicated to regions of interest. Thus adaptivity is one of the most natural strategies to achieve the balance between the performance and the feature; however, a perfect balance is generally challenging to achieve. In the literature, most adaptive algorithms care more about performance, i.e. key features may get sacrificed. In other instances, adaptive performance is sought only via decreasing the number of degrees of freedom (DOFs), but not necessarily the cost per DOF. One of the topics in my thesis is mainly about how to restore the sacrificed properties/features while still attaining efficient performance to make a better tradeoff.

Adaptive fluid. The computational bottleneck of fluid simulations is broadly recognized to concentrate on the solution of a Poisson equation, which is part of the computational mechanism that ensures incompressibility of the fluid flow. In a uniform Cartesian grid, the popular Marker-and-Cell method (MAC, Figure 1.3a) discretizes the Laplace operator via a nested standard central difference to attain a linear system for solve. This system is symmetric positive definite (SPD), and can be solved fast and efficiently [McAdams et al., 2010]. Furthermore, this method also attains second-order accuracy. Given a particular problem on the uniform grid, whose solution can be computed analytically, the error between the numerical and the analytical results decreases quadratically with the cell size of the background grid. Both merits, i.e. the SPD system and the second-order accuracy, originate from the orthogonality, which is an essential property inherent from MAC discretization. As shown in Figure 1.3b, the line connecting neighboring pressures is perpendicular to the cell face, and is also parallel to the velocity component stored at that face.

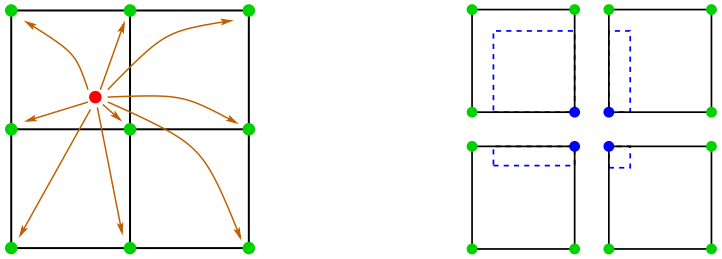
However, when switching to an octree-based discretization, T-junctions (a corner node of a finer cell that resides on the edge of a coarser cell) make it difficult to retain the orthogonality no matter whether the velocity is stored at the finer face or the coarser face (Figure 1.4a). In the literature, many researchers have attempted to tackle this challenge. Popinet [2003] retains second-order convergence but has to solve a non-symmetric linear system. In contrast, by allowing the discrete pressure gradient to be non-parallel to the corresponding discrete velocities, Losasso et al. [2004] obtain a fast SPD solver with the downgraded accuracy. In their follow-up work [Losasso et al., 2006], when restricting the refined region to interior to the fluid (i.e. no T-junctions are allowed on the free surface or the solid-fluid interface), they restore second-order accuracy. However, the interesting visual effects always arise on the free surface (e.g. a boat flees [Enright et al., 2002]) or the solid-fluid interface (e.g. paddles of a submarine generate bubbles [Patkar et al., 2013]). In this thesis, I propose to utilize a power diagram discretization at the transitional regions between different resolutions to recover the desired property - orthogonality (Figure 1.4b) . While keeping the topology of an octree unchanged, we can slightly modify the geometry of the grid cells to make all the dual edges of the grid, which corresponds to the line segment connecting cell centers, become perpendicular to the grid faces.

(a) Quadtree (b) Power diagram

Figure 1.4: **Adaptive orthogonality.** *Left:* In quadtree (octrees in 3D), orthogonality cannot be retained; *Right:* Our power-diagram based discretization restores orthogonality.

Adaptive MPM. In the material point method, Lagrangian particles carry all the physical properties while the Eulerian background grid serves as a scratch pad for solving the equations of motion. Details can be found in the work of Jiang et al. [2016] and a short introduction is in Sec. 1.3. The most crucial operations are the particle-to-grid and grid-to-particle kernels which transfer physical attributes (mass, momentum, etc.) between particles and the grid nodes via the weighting function. For example, when distributing the mass from a particle to grid nodes, the weighting function decides the amount of mass transferred to a particular node. There are a few constraints on the weighting function which are essential for the robustness of the MPM method.

Partition of unity is required for the mass and momentum conservation of the whole system. As discussed by Andersen and Andersen [2007], non-positive weights can render instability issues in critical cases where the nodal mass becomes zero. C^1 continuity is required to obtain a continuous force field to avoid noisy motions when particles move across cells. On a uniform grid, dimension independent B-spline functions are suitable for defining weighting functions and fulfill all requirements described above. However, it becomes challenging for the adaptive case, since no such simple options are available. Most works on adaptive material point methods [Lian et al., 2015; Tan and Nairn, 2002] achieve only C^0 continuity and others [Lian et al., 2015] fail to enforce non-negativity universally. Some works [Daphalapurkar et al., 2007; Ma et al., 2006, 2005] acquire both partition of unity and C^1 continuity, but their applications are restricted to nested grids (not a general octree). In this thesis, I propose a new approach that satisfies all of these constraints simultaneously. Specifically, I will restore the continuity feature required for large deformation scenarios in adaptive MPM method.



(a) MPM (b) GIMP

Figure 1.5: **From MPM to GIMP.** *Left:* MPM relates each particle directly to 3×3 grid nodes ($3 \times 3 \times 3$ in 3D); *Right:* GIMP divides the local domain into cells and each cell contributes to each node independently.

1.5 Parallel optimization for MPM

While Moore's law successfully predicted the fast evolution of integrated circuit technology in the past forty years, the feature size of the transistors is reaching the physical limit; this fact leaves simulation-oriented researchers no choice but to seek to extract the maximum performance from the given hardware. Parallel computing has been playing a more and more critical role in simulations. SIMD (single instruction multiple data) and SIMT (single instruction multiple threads) are the two most popular parallel techniques for computing acceleration on the CPU and the GPU respectively. As discussed in Sec. 1.3, since the transfer kernels are the bottleneck in a typical MPM pipeline, it is worth to exploit modern SIMD and SIMT techniques to boost their performance.

These performance-sensitive kernels are typically tasked with data transfers between a particle and an associated local, but sizable, neighborhood of grid nodes. Given a particle, one needs to compute a set of weights associated with $3 \times 3 \times 3$ grid nodes (3×3 in 2D, Fig. 1.5a) in its vicinity. In CPU, to utilize the data level parallelism provided by modern SIMD hardware support, e.g. Advanced Vector Extensions (AVX), AVX2 or AVX-512, we first need to reformulate the weight computations to a different form, which is better suitable for the application of SIMD instructions. On the other hand, in GPU, particles sharing the same set of grid nodes can simultaneously write data into the same node; thus resolving the write hazards becomes the main challenge.

SIMD. We introduce the generalized interpolation material point method (GIMP [Bardenhagen and Kober, 2004]) to reformulate the $3 \times 3 \times 3$ computations into 8×8 computations (4×4 in 2D, Fig. 1.5b). Consider the transfer of the mass from the particle to the blue grid node in the center.

From the perspective of GIMP, the transferred mass of that node is the sum of the contributions of the same node from the eight different cells sharing it (four different cells in 2D, Fig. 1.5b). Thus the computations of 27 weights are separated into the computations of the eight nodes in the eight different cells; and within each cell, SIMD can be applied to simultaneously compute weights for all the eight nodes by extracting the embedded symmetry property in GIMP. In this way, the workload can be reduced from the total 27 node-computations to 8 cell-computations to achieve at least a 3× speedup.

SIMT. In GPU, we assign each particle to one CUDA thread, and neighboring particles are sorted such that their threads are always executed in sequential order. Again assume we transfer mass from particles to the grid. Threads in the same warp simultaneously write the weighted mass into nodes; and some of them can write into the same node, leading to write hazards, a critical problem which has been examined in [Huang et al., 2008; Parker, 2006; Stantchev et al., 2008; Zhang et al., 2010]. There are generally two schemes for resolving this problem, scattering and gathering. Scattering methods simply use atomic operations to avoid conflicts. On the other hand, for gathering, usually a list of particles is created and maintained for each node during the simulation; thus each node can track down all the particles within its affecting range. Almost all previous papers are opting for a gathering approach since it is widely believed that high frequent atomic-operations in scattering can significantly undermine the performance, especially in GPU-based parallel applications. However, in this thesis, I propose a scattering-based method to handle most of the write conflicts without atomic operations, inspired by the concept of parallel reduction sum [Luitjens, 2014], and show that the performance is superior to the gathering ones.

These two types of interventions are examples of platform-specific optimizations that deliver significant performance boosts over established implementation practices. Chapter 3 and Chapter 5 detail the specific optimization techniques crafted towards this goal.

1.6 Particle-laden flows

As discussed in Sec. 1.1, multi-phase multi-material simulations are increasingly gaining attention from computer graphics researchers. It is generally challenging to correctly capture the interactions

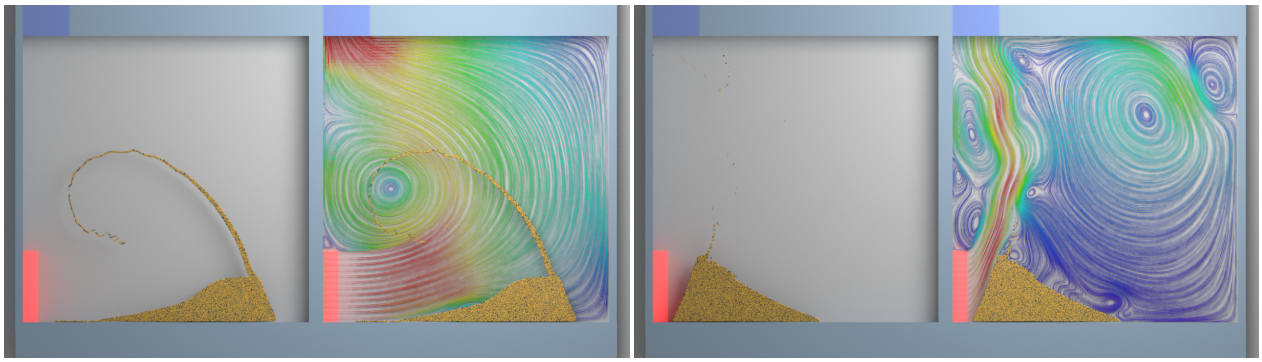


Figure 1.6: **Dune migration:** Two-dimensional simulations of wind blowing sand. *Left:* A one-way coupled simulation where sand does not affect wind fails to produce plausible dynamics. *Right:* Two-way coupled simulation from our method captures the characteristic behavior of a sand dune migrating forward.

between multiple materials. As shown in Fig. 1.6, one-way coupled simulation, where sand does not affect wind, fails to produce plausible dynamics; while two-way coupled simulation from our method captures the characteristic behavior of a dune migrating forward.

Particle-laden sediment flow is ubiquitous in natural systems. Typical examples include sediment transport, sedimentation, volcano eruption, dune migration by erosion with ripples, and dust storms. In the engineering literature, most previous works [Sun and Xiao, 2016a,b; Xiao and Sun, 2011] discretize these problems based on a combination of an Eulerian fluid solver and a Discrete Element Method (DEM) particle solver, which requires velocity interpolation between the two different discretizations to handle the momentum exchange between them. Furthermore, considering that DEM has to handle pair-wise particle collisions explicitly, it is usually too expensive to apply DEM to simulating dynamics in collision-heavy dense flow scenarios.

In this thesis, we propose to employ the MPM discretization for both the fluid phase and the solid phase to mitigate these problems. For the sediment, to support the massive amount of dense particle suspensions, it is preferable to formulate the sediment governing equations through continuum mechanics. However, in practice, sediment particles can get separated from the sediment clump to behave more like a discrete sphere. We design a density approximation method to measure the spatial distribution of sediment particles to identify the status of particles and adjust the treatment accordingly.

The fluid phase has to satisfy the boundary conditions enforced by sediments. However, it is

impossible to exactly resolve these boundaries in the presence of a large number of sediment particles, unless an extremely high fluid resolution is used. We instead incorporate the Locally Averaged incompressible Navier-Stokes theory proposed by Anderson and Jackson [1967] to model the fluid motion. With the assumption that each spatial point is simultaneously occupied by different phases or materials, the boundary conditions from sediments can be implicitly handled by embedding the volume fraction into the traditional mass and momentum equations for fluid.

Furthermore, we propose a momentum-conserving drag force to accurately describe the interactions between the fluid and sediments. Notice that both sediments and fluid are discretized by MPM, thus the drag force can be directly applied at the grid nodes without doing any velocity interpolations as in [Xiao and Sun, 2011; Stomakhin et al., 2014].

1.7 Outline

The remaining chapters are organized as follows. In chapter 2, we present an efficient and scalable octree-inspired fluid simulation framework with the flexibility to leverage adaptivity in any part of the computational domain, even when resolution transitions reach the free surface. In chapter 3, we propose an adaptive Generalized Interpolation Material Point (GIMP) method for simulating elastoplastic materials. In chapter 4, we present a mixed explicit and semi-implicit Material Point Method for simulating particle-laden flows. In chapter 5, we introduce methods that address the computational challenges of MPM and extend the capabilities of general simulation systems based on MPM, particularly concentrating on GPU optimization. We conclude this thesis along with some consideration about the avenues for the future work in the last chapter.

2 POWER DIAGRAMS AND SPARSE PAGED GRIDS FOR HIGH RESOLUTION ADAPTIVE LIQUIDS

In this chapter, we present an efficient and scalable octree-inspired fluid simulation framework with the flexibility to leverage adaptivity in any part of the computational domain, even when resolution transitions reach the free surface. Our methodology ensures symmetry, definiteness and second order accuracy of the discrete Poisson operator, and eliminates numerical and visual artifacts of prior octree schemes. This is achieved by adapting the operators acting on the octree's simulation variables to reflect the structure and connectivity of a *power diagram*, which recovers primal-dual mesh orthogonality and eliminates problematic T-junction configurations. We show how such operators can be efficiently implemented using a pyramid of sparsely populated uniform grids, enhancing the regularity of operations and facilitating parallelization. A novel scheme is proposed for encoding the topology of the power diagram in the neighborhood of each octree cell, allowing us to locally reconstruct it on the fly via a lookup table, rather than resorting to costly explicit meshing. The pressure Poisson equation is solved via a highly efficient, matrix-free multigrid preconditioner for Conjugate Gradient, adapted to the power diagram discretization. We use another sparsely populated uniform grid for high resolution interface tracking with a narrow band level set representation. Using the recently introduced SPGrid data structure, sparse uniform grids in both the power diagram discretization and our narrow band level set can be compactly stored and efficiently updated via streaming operations. Additionally, we present enhancements to adaptive level set advection, velocity extrapolation, and the fast marching method for redistancing. Our overall framework gracefully accommodates the task of dynamically adapting the octree topology during simulation. We demonstrate end-to-end simulations of complex adaptive flows in irregularly shaped domains, with tens of millions of degrees of freedom.

2.1 Introduction

Liquids exhibit complex and detailed motion across a vast range of scales, from tiny ripples to huge waves; this fact motivates the desire for liquid simulation tools that can handle ever increasing

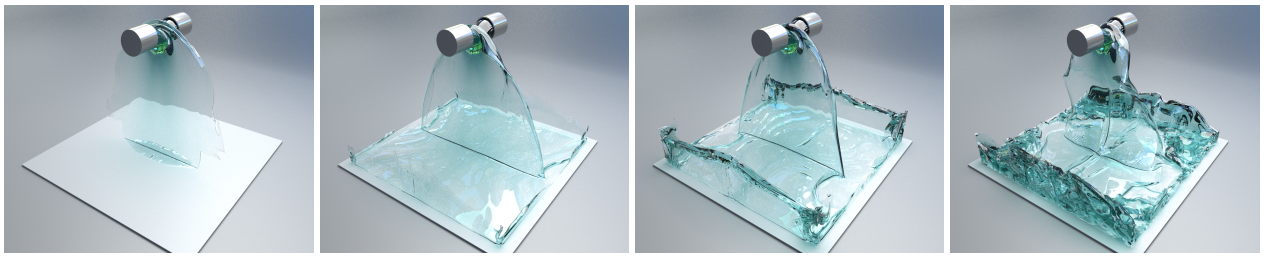


Figure 2.1: Two sources inject water into a container forming a thin sheet, with effective resolution 512^3. The adaptivity pattern is shown in Figure 1; a narrow band level set at resolution 2048^3 is used for interface tracking. As instabilities develop in the flow field over time, the sheet starts to wobble, creating ripples.

levels of resolution. While a key avenue towards this goal is the development of more efficient numerical methods on regular uniform grids that conserve mass with large time steps [Lentine et al., 2011, 2012; Chentanez and Müller, 2012] and allow for fast pressure projection [Molemaker et al., 2008; Lentine et al., 2010; McAdams et al., 2010; Zhang and Bridson, 2014; Ando et al., 2015; Dick et al., 2016; Liu et al., 2016], further computational gains can be obtained through *spatial adaptivity* — dedicating higher resolution to regions of high importance. Amongst the most natural approaches for augmenting standard staggered grid discretizations with spatial adaptivity is to replace the underlying grid structure with that of an octree. The critical challenge that arises for octrees is the treatment of T-junction (or hanging node) configurations where small cells are incident on larger cells. This is not solely a matter of bookkeeping in data structures: the discretization of the physics at these points must be done with care, or disturbing visual and numerical artifacts can emerge.

Losasso et al. [2004] proposed the first method to simulate liquids on octrees. They sacrificed accuracy in the discretization of pressure gradients near T-junctions in exchange for a simpler symmetric and positive definite Laplacian matrix that can be solved by Conjugate Gradient. The price for this choice was the loss of orthogonality between the stored velocity components at the T-junction cell faces and the corresponding discrete pressure gradients across those faces. This approximation reduces the accuracy of the pressure to first order, and gives rise to non-physical parasitic currents at T-junctions even for hydrostatic scenarios, as discussed by various authors [Losasso et al., 2006; Chentanez et al., 2007; Batty et al., 2010b; Ando et al., 2013; Ferstl et al., 2014]. Losasso et al. [2006] subsequently proposed a modification that raises the accuracy of the pressure

field to second order, but allows for adaptivity *only in regions completely interior to the fluid*. It is not clear how to incorporate solid or free surface boundaries in the presence of adaptivity, which necessitates uniformly refining all regions near surfaces and boundaries.

In this chapter, we describe a new staggered octree discretization that retains the positive definiteness and advantageous scaling of Losasso's scheme, while eliminating inconsistent pressure gradients and enabling accurate boundary condition enforcement, even across regions of changing resolution. Inspired by fluid animation approaches relying on Voronoi and power diagrams [Sin et al., 2009; Brochu et al., 2010; de Goes et al., 2015], we modify the implied geometry of the octree near T-junctions to have the form of a power diagram; this effectively eliminates the problematic T-junctions and recovers orthogonality between pressure gradients and cell faces. Embedded free surface and solid boundary conditions can then be directly incorporated on this hybrid mesh, per Brochu et al. [2010]. The resulting octree Poisson solver is second order accurate in pressure and artifact-free. In spite of this different geometric perspective used in our discretization, we retain (with minor caveats) the ability to store our simulation variables on a traditional octree rather than an explicit unstructured mesh, and exploit the regularity of this representation in the interest of performance.

We complete our octree simulator to support end-to-end liquid simulation of complex large-scale scenarios via several complementary enhancements. We leverage the *SPGrid* data structure [Setaluri et al., 2014] to allow our solver to process octree data at performance levels competitive with regular uniform grids containing a similar number of variables. Notably, this standard of efficiency is notoriously difficult to match with pointer-based octree implementations or unstructured mesh discretizations, due to the impact that indirect and irregular memory access (or the overhead of explicit storage of topology) has on the optimal utilization of the available bandwidth. We accurately track the surface using a uniformly high resolution narrow band level set, exploiting the compact footprint of the SPGrid structure and an adaptively multi-stepped semi-Lagrangian advection scheme. To support accurate fast-marching-based level set redistancing on octrees, we perform a localized Delaunay triangulation of the cell centers incident on T-junctions, and adapt standard fast marching strategies to the resulting hybrid tetrahedral/hexahedral mesh. We solve the sparse, symmetric definite linear systems that arise from our Poisson discretization using a lightweight

multigrid preconditioner for the Conjugate Gradient method. We efficiently store the multigrid hierarchy using SPGrid, while showing how a first order accurate multigrid V-cycle can be used as a building block of a very effective preconditioner for the second order problem.

Contributions We developed an accurate and efficient staggered octree-based fluid simulation framework, featuring support for:

- Second order accurate pressure projection allowing adaptivity even along solid and air boundaries;
- Excellent affinity to parallelism-optimized data structures for octree storage (SPGrid), allowing us to easily scale to tens of millions of degrees of freedom on a single computer;
- A simple and effective narrow band level set interface tracking scheme that uses SPGrid to reduce memory footprint;
- An enhanced fast marching method for octrees;
- A tailored multigrid-preconditioner for Conjugate Gradient.

2.2 Related work

Our approach builds on standard grid-based fluid animation techniques; Bridson [2008] provides a useful overview of this family of methods. We adopt a staggered discretization, which naturally avoids instabilities from odd-even decoupling (“checkerboarding") in the pressure field. Our treatment of irregular free surface and solid boundaries draws on the ghost fluid method [Gibou et al., 2002; Enright et al., 2003] and the variational/cut-cell solid scheme [Batty et al., 2007; Ng et al., 2009], respectively. Incorporating these ideas into our solver ensures accurate pressure solutions and avoids voxelization artifacts. While we focus on adaptive approaches for Eulerian methods, Lagrangian SPH and vortex methods have also been applied in the context of adaptivity [Adams et al., 2007; Solenthaler and Gross, 2011; Takahashi and Lin, 2016; Golas et al., 2012].

Octrees Popinet [2003] proposed the first octree-based fluid solver, extending earlier adaptive mesh refinement (AMR) schemes for the Poisson problem in the computational fluid dynamics community [Martin and Cartwright, 1996; Minion, 1996]. Losasso et al. [2004] extended this approach with support for free surface flow, and incorporated a simplification to yield symmetric positive definite systems for pressure projection. Applications of their method have included simulation of bubbles [Hong and Kim, 2005], coupling with dynamic objects [Guendelman et al., 2005], and even lightning [Kim and Lin, 2007]. Unfortunately, this approach reduces the pressure accuracy to first order and introduces visual artifacts in the velocity field. Losasso et al. [2006] later improved the discretization of the pressure projection to recover second order accuracy while preserving symmetry, although it is unclear how to incorporate irregular boundary conditions near T-junctions because the necessary modifications to the Poisson stencil are mutually incompatible; we overcome this limitation to allow refinement and boundaries to seamlessly co-exist without loss of accuracy. Ferstl et al. [2014] proposed a finite element method (FEM) that subdivides octree surface cells cut by surface geometry to yield a conforming mesh, applying Nitsche's method for the free surface and stabilization to minimize checkerboarding. Like Losasso's method, it requires boundaries to be uniformly resolved. Furthermore, solid boundaries were treated in a voxelized fashion. In contrast to the preceding methods, our approach also leverages the recent SPGrid data structure [Setaluri et al., 2014] to overcome the significant efficiency penalties inherent in typical pointer-based octree structures. Nielsen and Bridson [2016] recently discussed some details of the FEM-based adaptive tile-tree scheme used in Maya's Bifrost simulator, although full details are not available. Like our method, they aim to preserve a regular grid-like structure to maximize efficiency, but do so using a more rapidly branching octree involving $5 \times 5 \times 5$ blocks, rather than our pyramid of SPGrids. Their use of FEM was explicitly motivated by the fact that previous finite volume octree schemes have struggled to simultaneously provide matrix symmetry, second order accuracy, and support for non-axis-aligned boundaries; our method achieves all three.

Stretched Grid Cells Another quite useful, albeit more restrictive, approach to spatial adaptivity is the use of stretched grid cells. For example, portions of a regularly sampled domain may be replaced with tall cells that have been stretched along a single axis [Irving et al., 2006b; Chentanez

and Muller, 2011], or larger sections of a regular grid may be stretched along multiple axes by essentially translating entire grid lines [Zhu et al., 2013]. These approaches preserve most of the regular grid efficiency while offering some benefits of adaptivity.

Tetrahedral Meshes An important family of alternatives to octrees are unstructured or partially structured adaptive tetrahedral meshes. An array of staggered Eulerian finite volume discretizations have been presented for such meshes [Feldman et al., 2005; Klingner et al., 2006; Chentanez et al., 2006; Elcott et al., 2007; Chentanez et al., 2007; Batty et al., 2010b]. In particular, Batty et al. [2010b] extended these methods with support for irregular free surfaces and solids to allow straightforward adaptivity even in the presence of boundaries, a feature we likewise pursue. While purely unstructured tetrahedral methods incur non-trivial computational costs due to frequent remeshing and data structure overhead, semi-structured lattice-based variants can somewhat reduce these costs by exploiting an underlying octree structure [Chentanez et al., 2007], although they still incur greater overhead than a pure octree. One clear advantage of tetrahedral meshes over basic octrees is that they support adaptivity without any T-junctions. A drawback is that pressures must be placed at tetrahedron circumcenters to avoid artifacts arising from the pressure solve, yet circumcenters are not guaranteed to be inside their corresponding tetrahedra [Batty et al., 2010b]. Ando et al. [2013] partially circumvent this issue with a finite element variation that instead places velocity vectors at tetrahedron barycenters and pressures at vertices. However, the authors note that they must drop back to a first order discretization for poorly shaped tetrahedra which occur near resolution transitions in their lattice mesh; by contrast, we preserve second order accuracy across transitions and throughout the domain. Node-based Lagrangian schemes that use dynamically remeshed unstructured tetrahedral meshes can also support adaptivity [Misztal et al., 2010; Misztal and Bærentzen, 2012; Clausen et al., 2013]. These methods rely on stabilization terms or extra remeshing procedures to avoid locking or odd-even pressure decoupling artifacts. Moreover, they are generally costlier than Eulerian approaches due to the overhead of continuous remeshing and unstructured mesh manipulation.

Voronoi Diagrams Since the circumcentric dual of a Delaunay tetrahedralization is a Voronoi diagram, the Poisson equation can be readily discretized on this dual mesh too, using essentially

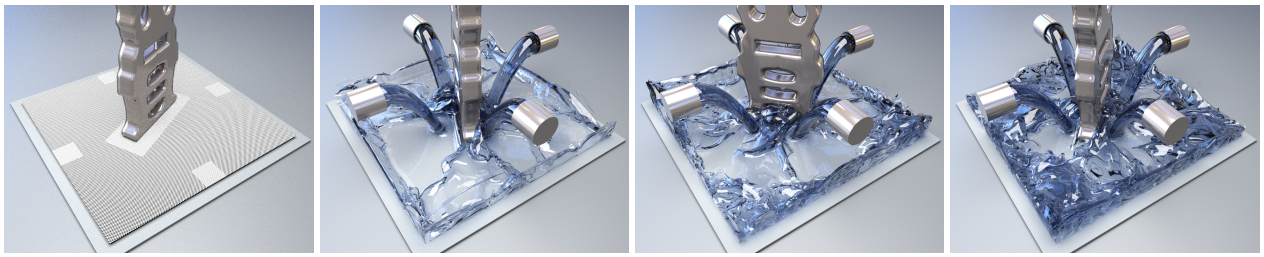


Figure 2.2: Four sources pour water that collides with a kinematic rotating object in a container, with effective resolution 512^3. The adaptivity pattern, illustrated on the left, continuously rotates with the central spinning object. A narrow band level set with 2048^3 resolution is used to track the free surface.

the same staggered finite volume approaches as the tetrahedral schemes above. This has been exploited in several ways. Sin et al. [2009] constructed a particle-based Lagrangian scheme that performs pressure projection on the Voronoi diagram of the particles at each step. Brochu et al. [2010] combined a mesh-based surface tracking scheme with a carefully constructed Voronoi diagram to enable capturing of thin liquid features. English et al. [2013b,a] constructed a nested (Chimera) grid scheme relying on the flexibility of the local Voronoi structure to stitch together the boundaries between regular grids of differing resolutions. Voronoi diagrams have similarly been exploited in computational physics: Guittet et al. improved the ghost-fluid method for two-phase Poisson problems by aligning Voronoi faces with the fluid interface [Guittet et al., 2015a], and separately used the Voronoi diagram of fluid face midpoints to treat viscosity on non-graded octrees [Guittet et al., 2015b].

Power Diagrams Mullen et al. [2011] observed that regular triangulations and their associated dual power diagrams can offer increased flexibility in meshing, while preserving primal-dual orthogonality. This property is critical for constructing accurate discrete gradient and divergence operators [Batty et al., 2010b] in the style of Discrete Exterior Calculus (e.g., [Elcott and Schröder, 2006]). Building on this idea, de Goes et al. [2015] developed a Lagrangian particle method that constructs a power diagram at each time step for enforcing incompressibility. Krivá et al. [2016] used the power diagram of a quadtree in two dimensions to solve a Poisson problem in the context of image processing, although they did not identify their construction as a power diagram. Similarly, Sifounakis et al. [2016] proposed a so-called virtual slanting approach for a Navier-Stokes solver on

quad/octrees; however, their strategy of simply slanting faces breaks down in three dimensions, where proper orthogonality cannot be maintained without (explicitly or implicitly) changing the local cell connectivity. By contrast, we observe that the desired construction is a power diagram, and use this fact to easily determine the correct grid geometry and topology.

Interface Tracking We represent the liquid surface using a level set scheme [Sethian, 1999], a common choice in fluid animation, and specifically a narrow band variant [Adalsteinsson and Sethian, 1995] which we further customize to our application. The particle level set method of Enright et al. [2002] is a popular variant; it maintains Lagrangian particles around the Eulerian interface to accurately reconstruct the surface, building on ideas proposed by Foster and Foster and Fedkiw [2001]. Rather than exploit particles, however, we employ a purely Eulerian narrow band method at a resolution higher than the simulation mesh, similar to previous authors (e.g., [Kim et al., 2009; Bojsen-Hansen and Wojtan, 2013; Goldade et al., 2016]). We differ in proposing a multi-stepping semi-Lagrangian strategy for reduced dissipation, and in leveraging the SPGrid data structure for a highly optimized implementation. Rather than increase resolution, Heo and Ko [2010] obtained greater detail by proposing a pseudo-spectral level set method. The two most common alternatives to level sets are pure particle representations, as in PIC or SPH schemes [Foster and Metaxas, 1996; Desbrun and Gascuel, 1996; Zhu and Bridson, 2005; Jiang et al., 2015], and Lagrangian triangle meshes with dynamic topology [Brochu and Bridson, 2009; Wojtan et al., 2009; Müller, 2009]. Various hybrids have also been proposed [Bargteil et al., 2006; Yu et al., 2012]. While each approach has strengths and weaknesses, level sets stand out for their simplicity, smoothness, and potential for efficiency. The affinity of narrow band level sets to the SPGrid data structure allows for a particularly compact surface representation.

2.2.1 SPGrid arrays and sparse uniform grid pyramids

Our algorithmic formulation draws upon the work of Setaluri et al. [2014] and exploits two of their key proposals. First, it was observed that in lieu of a traditional pointer-based representation, a Cartesian octree can be stored as a pyramid of *sparsely populated uniform grids*. Thus, any cell of the octree can be uniquely identified with a single cell of one of the uniform grids in the pyramid. It

was shown that the global computation of the Laplace operator can be equivalently performed by alternating (a) perfectly uniform stencils on each individual grid, and (b) highly structured data transfers solely between adjacent grids in the pyramid. Section 2.4 describes how we exploit this capability for our power diagram discretization.

Second, a low-level data structure named *SPGrid* (or "Sparse Paged Grid") was proposed for compact storage and efficient stream processing of sparsely populated uniform grids. SPGrid leverages the Virtual Memory subsystem, in conjunction with a Morton ordering, to index into a vast address range, but only materialize into physical memory those parts of the array that are actually touched. Thus, an array that may occupy more than 1TB if densely allocated, can occupy as little as 1GB in physical memory if only a spatially coherent 0.1% of this array was actually populated with data. Recognizing that simulation applications commonly utilize a large number of scalar or vector fields with nearly (or exactly) identical index domains, SPGrid interleaves several of these distinct *"channels"* within a 4KB page. Thus, an SPGrid with 4 float-valued channels fits a geometric block of $8 \times 8 \times 4$ variables for each of those channels in a 4KB page, while 16 float-valued channels would yield a block size of $4 \times 4 \times 4$. The number of channels can easily vary to cater to different tasks; for example, we use only 4 channels for interface tracking, and 16 channels for solving the incompressible Euler equations.

2.3 Method overview

Our simulation methodology necessitates certain interventions in various parts of a standard liquid simulator. A large fraction of our contributions are centered around **pressure projection**, which as many authors have asserted [Molemaker et al., 2008; Lentine et al., 2010; McAdams et al., 2010; Zhang and Bridson, 2014; Ando et al., 2015; Dick et al., 2016; Liu et al., 2016] is typically the most expensive component of the simulation loop in high resolution domains. Section 2.4 discusses several aspects of our pressure solver, including the discretization of the Poisson equation using power diagrams, the efficient storage of the discrete equations and unknowns in an adaptive grid structure, and fast solution of the resulting system using a preconditioner for Conjugate Gradient adapted from a multigrid scheme. Section 2.5 presents our **interface tracking** approach that uses

a highly resolved implicit geometry representation of the free surface, localized to a narrow band around the moving interface and stored in a sparse grid structure (SPGrid) [Setaluri et al., 2014; Liu et al., 2016]. The same section also discusses details on how this representation is advected forward in time, interpolation of scalar and vector quantities, and enhancements to **fast marching and velocity extrapolation** that are mandated by our power diagram discretization. Important implementation details and optimizations in our formulation are reviewed in Section 2.6.

2.4 Pressure projection on modified octrees

To achieve an accurate staggered finite volume discretization of the Poisson equation, the faces of the primal mesh which store velocity components should be orthogonal to the edges of the dual mesh, which correspond to pressure gradients [Perot and Subramanian, 2007; Batty et al., 2010b]. For sufficiently regular meshes this gives second order accurate pressures. In this section, we describe our discretization for the Poisson equation that exploits both the regularity of octrees for aggressive parallelization and the inherent primal-dual orthogonality of power diagrams for accuracy. Our objective is to solve the incompressible Euler equations

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \frac{\nabla p}{\rho} = \mathbf{f} \tag{2.1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{2.2}$$

with a splitting scheme following Stam [1999]. Here, $\mathbf{u} = (u, v, w)$ is the vector velocity field, $\mathbf{f}$ encapsulates external forces, p is the scalar pressure field, t is time, and ρ is the liquid density.

2.4.1 Power diagram discretization

Aurenhammer [1987] defines the *power* of a point x with respect to a sphere of radius r as $d^2 - r^2$, where d is the distance of the point from the center of the sphere. Given a set of spheres, their power diagram is then a partition of space where each point is associated to the sphere with minimum power. We construct an octree-based power diagram by conceptually placing a sphere at each octree cell center with a radius of $\Delta x/\sqrt{3}$ (or $\Delta x/\sqrt{2}$ in 2D), where Δx is the cell's side length; this yields

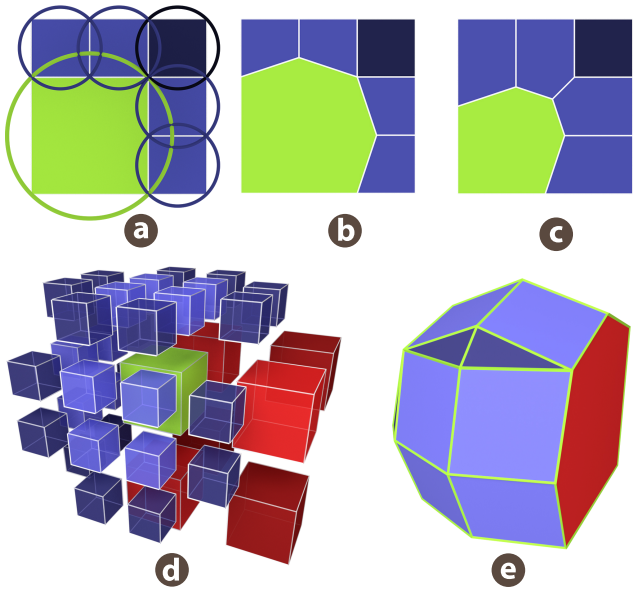


Figure 2.3: (Top) A 2D cell bordering two T-junctions, along with its corresponding power diagram (middle), and the Voronoi diagram of the cell centers (right). The cells of the power diagram have the same neighbor relationships as the original quadtree, but the dual edges are orthogonal to cell faces. The Voronoi diagram of the same geometry creates new face connections between nearby cells at the same level. (Bottom) A 3D cell bordering three T-junctions, along with its corresponding power cell. Note the new faces that emerge between neighbors that previously only shared an edge. Colors imply resolution and neighbor relationship to the central coarse cell.

the circumsphere for each cell. For uniform resolution regions of the tree, the resulting power diagram leaves the regular Cartesian grid pattern unchanged. More remarkably, for T-junctions in two dimensional graded quadtrees, this choice gives a power diagram with *exactly* the same cell connectivity as the original quadtree, but with the geometry adapted to recover primal-dual orthogonality (see Figure 2.3(a,b)). However, in three dimensions a minor wrinkle arises: a given cell remains face-connected to its original face neighbors, but may now also share new faces with its original edge neighbors (see Figure 2.3(d,e)). Nevertheless, as we describe below, this geometry is still amenable to efficient computation of the Poisson equation.

We store pressures p at octree cell centers and normal components of the velocity field $\mathbf{u}$ at power cell faces. For computing the pressure, we discretize the Poisson equation in finite volume fashion over the power cells as

$$V_{\text{cell}} \nabla \cdot \frac{\nabla p}{\rho} = \sum_{\text{faces}} A_{\text{face}} \left(\frac{\nabla p}{\rho} \cdot \mathbf{n} \right) \tag{2.3}$$

where $\mathbf{n}$ is the unit normal vector pointing out of a face, A_{face} is the face area, and V_{cell} is the volume. We similarly compute the discrete divergence as

$$V_{cell} \nabla \cdot \mathbf{u} = \sum_{faces} A_{face} (\mathbf{u} \cdot \mathbf{n}). \tag{2.4}$$

For treating boundary conditions, we adapt an earlier unstructured mesh embedded boundary approach [Batty et al., 2010b; Brochu et al., 2010]. In our setting, this allows both the free surface and solid boundaries to arbitrarily cut through the domain, even near T-junctions. It requires only liquid signed distance values at cell centers and solid signed distance values at cell vertices. By contrast, previous methods [Losasso et al., 2004, 2006] required uniformly refined cells near boundaries. Our resulting linear system remains sparse, symmetric and positive definite, and ensures second order accuracy of the computed pressure field.

Voronoi diagram octrees Another option to recover orthogonality would be the Voronoi diagram of the octree cell centers. For non-graded trees this can yield very general unstructured meshes that discard the regularity benefits of the tree structure (see e.g., [Guittet et al., 2015b]). While the variety of mesh configurations can naturally be reduced by grading, the Voronoi topology still differs more strongly from the original tree than for the power diagram, even in the simpler 2D setting (see Figure 2.3(c)). We therefore prefer the power diagram as it provides the necessary orthogonality while better preserving the original octree structure.

2.4.2 Pyramid of sparse uniform grids

By default, a power diagram would require an explicit mesh for storing the topological connectivity. However, such a representation would necessitate expensive lookups near T-junctions. Since power diagrams preserve the original octree structure, with the caveat of introducing some additional faces between edge neighbors, we adopt an approach similar to Setaluri et al. [2014] of organizing computation on a pyramid of sparsely populated uniform grids that make much better use of the available hardware memory bandwidth. We first briefly review this approach and subsequently highlight the necessary modifications required for power diagrams. To identify cells and faces that

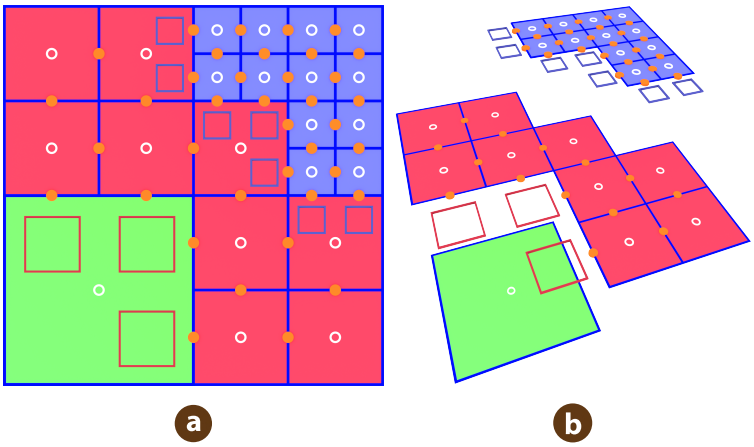


Figure 2.4: Pyramid of sparse uniform grids. All active cell and face degrees of freedom are shown along with ghost cells (square outlines) at each level.

carry degrees of freedom, Setaluri et al. [2014] defined *active* cells and faces as follows:

- A cell at a given level of the pyramid is *active* if it is geometrically present and undivided in the octree.
- A face at a given level is *active* if that face is geometrically present and undivided in the octree, implying that every active face has at least one active cell neighbor at that level.

For efficiently emulating the building blocks of adaptive fluid simulation, they used two fundamental algorithmic kernels: stencil operations within a single level, and transfer operations between adjacent levels in the pyramid. This additionally requires the concept of *ghost* cells. Suppose we index each level with $1 \leqslant l \leqslant L$ where lower indices denote finer grids, and let C_I^l denote a cell that natively lives at level l with multi-dimensional index I. Then, a cell C_I^l is *ghost* if all three of the following conditions are *jointly* met:

1. C_I^l is not active at level l,
2. C_I^l neighbors a cell that is active at level $s \leqslant l$, and
3. An active, *coarser* parent of C_I^l exists at some level $l^\star > l$.

Figure 2.4 shows the pyramid of sparse grids, along with all ghost cells and active cell/face degrees of freedom. To transfer data between adjacent levels of the pyramid, two more operators are used:

1. GhostValuePropagate(): an upsampling routine in which data from level l is *copied to* fine ghost children at level $l-1$,

2. GhostValueAccumulate(): a downsampling routine in which data in level l *accumulates* contributions from any fine ghost children at level $l-1$.

Essentially, ghost cells provide a mechanism to combine information that is shared across different levels of resolution, while only focusing effort one level at a time, enabling optimizations native to uniform grids. A ghost cell can either be a source of a numerical value originating from a coarser level in the pyramid (by mirroring the coarse value), or a placeholder for the partial result of an operation that would have otherwise required access to a coarser variable. We refer the reader to the work of Setaluri et al. [2014] for details on how this representation can be used to efficiently compute the discrete gradient, divergence, and Laplace operators.

Enhancements for power diagrams Since power diagrams may introduce new faces between cells that previously only shared an edge, the definition of ghost cells requires some modifications. In the second of the three criteria we previously listed for identifying a ghost cell, Setaluri et al. [2014] considered only face neighbors of C_I^l to check if any of them were active at level $s \leqslant l$; we now also check all edge neighbors of C_I^l (in 3D). Moreover, their implementation of the discrete Poisson operator (which was based on the formulation of Losasso et al. [2004]) yielded stencils with closed-form coefficients even at level transitions. By contrast, our stencil coefficients are determined by the geometry of the power diagram. To efficiently retrieve these coefficients at run-time, we adopt certain restrictions on our grading scheme that make the power diagram geometry of a cell dependent *only* on its 1-ring neighborhood. This allows us to retrieve stencil coefficients via a look-up table, indexed by a compact descriptor of the local topology as we propose in Section 2.6.

As mentioned in Section 2.4.1, we store velocities at all power cell faces. For faces that are exactly homologous to faces on the original octree, similar to Setaluri et al. [2014], we dedicate 3 channels for storing the individual components of the velocity field (in the standard octree, those would have been the u, v, w axis-components of the velocity field; in our case they are normal velocity components relative to the respective power diagram faces). In our power diagram, however, faces of the power diagram (and correspondingly normal velocity variables that require storage) can exist

in 3D between cells that were edge neighbors in the original octree. An edge of the octree can have up to four cells incident to it; however only two of those four cells can actually be connected in any given power diagram under our formulation. Specifically, it is possible for the pair of cells indexed $C_{i,j,k}$ and $C_{i,j+1,k+1}$ (sharing an edge aligned with the x-axis) to share a face of the power diagram. Likewise, cells $C_{i,j+1,k}$ and $C_{i,j,k+1}$ (incident on the same edge, as far as the octree is concerned) could instead share a face of the power diagram. However, *at most one* of the aforementioned pairs could have a power diagram face connecting them, in any given scenario. Thus, we can associate the normal velocity component associated with either of these two cases with the same octree edge, with the understanding that at most one of these two neighboring relations can be present at the same time; the determination of which of the two cases (if any) we have can be made by examining the local octreee topology. Hence, velocity components associated with such cases of edge neighbors can be conceptually stored on 3 offset grids, each of them centered on the midpoint of edges aligned with the x, y, and z axes, respectively. We simply dedicate three additional channels in SPGrid to store those special velocity components (we would have called them *edge-centered velocities* if such a concept existed in the original octree). We do not explicitly store a description for the direction of this edge-centered velocity component, as this will be automatically reflected in the stencils of those cells incident to it.

2.4.3 Multigrid preconditioner

Our power diagram discretization, paired with a topology encoding scheme (see section 2.6) will allow the Laplace operator to be applied implicitly without building its explicit matrix form. This gives the opportunity to use a matrix-free method such as Conjugate Gradient for solving the Poisson equation. Naturally, a good preconditioner is essential; Setaluri et al. [2014] used a very effective adaptive multigrid preconditioner. However, their approach was specifically tailored for the first order accurate discretization of Losasso et al. [2004]. Since both our power diagram discretization, as well as the first-order accurate approach [Losasso et al., 2004; Setaluri et al., 2014] have exactly the same set of pressure variables, it would be natural to start with the multigrid V-cycle preconditioner proposed by Setaluri et al. [2014] as a baseline, and attempt to adapt it to the power

diagram discretization. The most straightforward way to make such an adaptation would be to replace the discretization at the finest level of the hierarchy with our newly proposed power diagram formulation, retaining the first-order accurate operators at all coarser levels. This adaptation is quite effective; in our tests it consistently achieves preconditioning efficiency comparable with what Setaluri et al. [2014] report for the first-order problem.

Although this approach is effective, there is a modest penalty to be paid in terms of implementation efficiency. The single most costly component of a multigrid V-cycle is the smoothing routine at the finest resolution of the discretization hierarchy. This step is now using our second order power diagram discretization which, although quite efficient, does not permit the full extent of aggressive optimizations of the first order scheme [Setaluri et al., 2014], which does not require local topology queries to define its stencils. Thus, we have implemented a hybrid alternative: We first invoke the smoothing routine, using the second order power diagram discretization, but iterate it only near boundaries and level transitions. We subsequently update the residual, and use a *first order accurate* multigrid V-cycle to solve the error equation and generate a correction across the entire domain. We follow up with a second sweep of smoothing the second order discretization (near boundaries and level transitions) to ensure symmetry of the preconditioner thus crafted. Let us denote by $\mathbf{L}_1$ the first order Laplace operator from Setaluri et al. [2014], and let $\mathbf{M}_1 \approx \mathbf{L}_1^{-1}$ be the multigrid preconditioner they defined in their work. Finally, let $\mathbf{L}_2$ be our second order Laplace discretization, using the power diagram. We define a new preconditioner $\mathbf{M}$, whose action $\mathbf{w} = \mathbf{M}\mathbf{q}$ is computed as follows:

1. Starting with a zero initial guess $\mathbf{p}_0 = \mathbf{0}$, execute k iterations of the damped Jacobi method on the equation $\mathbf{L}_2\mathbf{p} = \mathbf{q}$ (input vector is used as right hand side). This is only applied to a band (about 3 voxels wide) around the boundaries and level transitions. Let $\mathbf{p}_1$ be the iterate that results from this operation, and $\mathbf{r}_1 = \mathbf{q} - \mathbf{L}_2\mathbf{p}_1$ the associated residual.

2. Compute a correction $\delta\mathbf{p} = \mathbf{M}_1\mathbf{r}_1$ by applying the first order accurate preconditioner $\mathbf{M}_1$ from Setaluri et al. [2014] to the residual $\mathbf{r}_1$. Add the correction to $\mathbf{p}_1$, to obtain a new approximation $\mathbf{p}_2 = \mathbf{p}_1 + \delta\mathbf{p}$.

3. Repeat k additional iterations of the same Jacobi method on the equation $\mathbf{L}_2\mathbf{p} = \mathbf{q}$, but this time use $\mathbf{p}_2$ as the initial guess. Let $\mathbf{w}$ denote the result at the end of this iteration.

We can easily verify that vectors $\mathbf{w}$ and $\mathbf{q}$ are related by a linear map $\mathbf{w} = \mathbf{M}\mathbf{q}$ (there is nothing in steps 1-3 above that introduces any nonlinearity, and it is easy to verify that when $\mathbf{q} = \mathbf{0}$ we would also have $\mathbf{w} = \mathbf{0}$). In fact, a careful look at the matrix formula for the Jacobi iteration reveals that the entire matrix $\mathbf{M}$ *is symmetric and positive definite*, assuming that $\mathbf{M}_1$ is SPD as well. We use the algorithm above to implicitly apply $\mathbf{M}$ as a preconditioner to Conjugate Gradient for the system $\mathbf{L}_2\mathbf{p} = \mathbf{f}$. In our examples, this preconditioner achieved very similar convergence performance to the straightforward alternative approach of using a second order operator at the finest level of a V-cycle preconditioner, achieving satisfactory convergence within 6-10 iterations across all resolutions. It was necessary to perform an adequate number ($k \approx 8$) of boundary smoothing iterations in steps (1) and (3) of the aforementioned preconditioner application to achieve this good convergence behavior, but the ability to restrict this effort to just the boundary region (while using an aggressively optimized first order V-cycle) made this a favorable trade-off at large resolutions.

2.5 Interventions in the simulation pipeline

To track the dynamically evolving liquid surface, we designed a level set scheme similar in spirit to prior work [Bargteil et al., 2006; Kim et al., 2009; Bojsen-Hansen and Wojtan, 2013; Goldade et al., 2016] in the sense that we use the SPGrid data structure to store an Eulerian description of a narrow band around the free surface at a significantly higher resolution than the velocity data — typically by a factor of 4 or 8 (see Figure 2.5). Thus, we use two separate meshes, a background simulation mesh that is an octree, and an interface tracking grid (or fine level set grid) that is a single SPGrid (rather than a pyramid). Note that the velocities natively live *only* on the octree, and we also store and evolve a separate (coarse) level set on the octree because our fine representation does not carry inside/outside information beyond a narrow band of the free surface. The fine level set representation only requires an SPGrid with 4 channels (1 channel for Boolean flags associated with domain geometry, 1 for the level set, and 2 more channels for fast marching).

Level set advection The time step Δt is determined by the finest effective resolution of the octree, since the velocities live natively on the octree. However, this time step will be overly dissipative for advecting the fine level set forward in time using a standard semi-Lagrangian update, and may

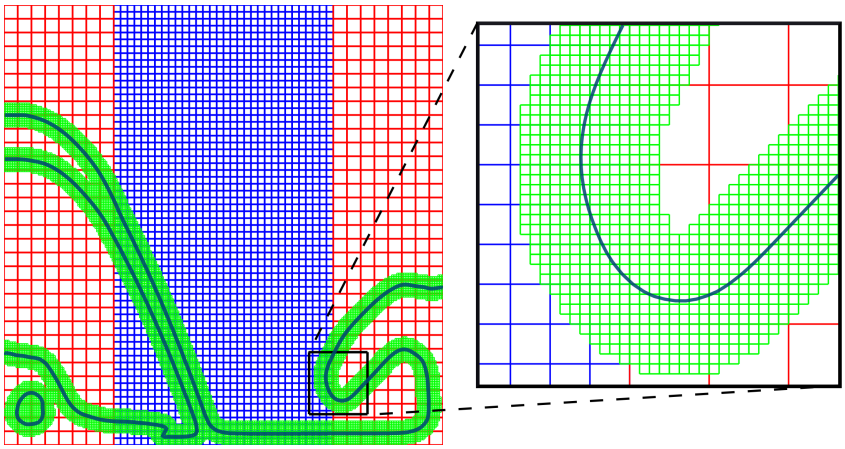
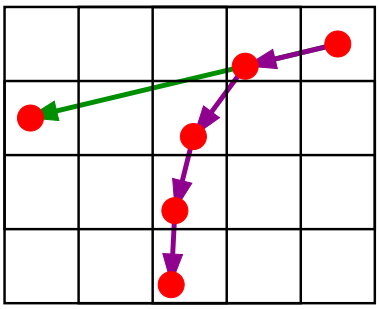


Figure 2.5: A time-evolving Eulerian description of a narrow band around the interface is stored at a significantly higher resolution (shown in green) in our framework. This representation is used to correct the level set field on the octree after every advection step.

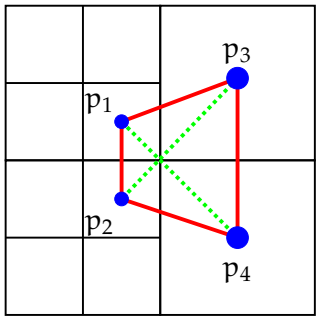
require more expensive schemes for volume conservation [Lentine et al., 2012]. We follow a more accurate approach where we divide this time step by m, where m is the factor by which the fine level set resolution is higher than the effective resolution of the octree. Next, we take m steps (of forward Euler) backwards along the velocity field, where each step is performed with a time step size of $\Delta t/m$.

At each step, we reach a new point p where we compute a new interpolated velocity before taking the next step, tracing a piecewise linear curve in the process (see the figure inset). After the m^{th} step we interpolate the level set, and assign its value back to the starting cell of the trajectory. Note that this approach still allows for efficient parallelization without decreasing the time step size according to the fine level set resolution. Of course, we maintain a wide enough band so that the free surface still falls inside this band after advection. Compared to simply upsampling the background octree, tracing back a piecewise linear curve for semi-Lagrangian advection foregoes the numerical dissipation that would have been introduced by m individual advection and reinitialization steps, producing more accurate results (see Figure 2.6). For advecting forward the coarse level set stored on the octree, we follow the approach of Setaluri et al. [2014]. Subsequently, we use the fine level set to correct the coarse level set wherever we have valid ϕ-values.



Velocity interpolation and advection The discretization of velocity advection is more subtle as not all faces are axis-aligned, and so velocities are not neatly segregated into orthogonal components. Fortunately, this problem has been studied in prior work on unstructured mesh fluid simulation. To interpolate velocities at arbitrary points, we first compute full velocity vectors at all cell centers. For regular uniform cells, we average the axis-aligned face velocities; for all power cells, we perform a least squares fit based on all the face normal components [Feldman et al., 2005]. The dual mesh of our power diagram consists of cubes and tetrahedra, whose vertices are the octree cell centers. Therefore, having constructed velocity vectors at the vertices of this mesh, we apply trilinear interpolation inside cubic cells and barycentric interpolation on tetrahedra. For improved efficiency and accuracy in regular regions away from level transitions, we revert to standard per-axis face-based velocity interpolation. With this interpolation procedure at hand, we compute full velocity vectors at the centroid of each face and advect them using a standard semi-Lagrangian update, subsequently projecting the velocity onto the face normal direction. Likewise, we use this interpolant as needed for advecting the fine resolution level set.

Dynamic topology After each advection step, the topology of the SPGrid storing the high resolution level set needs to be updated; new cells may emerge in the narrow band which require valid signed distance values, and old cells may fall outside the band and should be deleted. We do this by instantiating a new SPGrid and copying over level set values for all cells at the interface. Subsequently, we run fast marching on this SPGrid, activating new cells wherever necessary. For efficiency, we first allocate all pages corresponding to blocks that either contain the interface, or lie in the 1-ring of a block that contains the interface. After this step, cells that lie within the narrow band can be safely marked in parallel without any data hazards. We note that Setaluri et al. [2014] speculated on the utility of an enhancement to SPGrid, that could actually discard physical pages no longer in use (in Linux, this would be via a variant of the `madvise` system call), to allow dynamic additions and removals to the *same* SPGrid. We did not find such an optimization to be worthwhile here, as evaluating refinement criteria at the same time while discarding prior data is an unnecessary complexity.



Fast marching and velocity extrapolation We store level set values ϕ on cell centers of the octree, and solve for fast marching in uniform regions similar to Foster and Fedkiw [2001]. Near T-junctions we adopt the approach of Sethian and Vladimirsky [2000] and run fast marching on a tetrahedral mesh. Note that this approach mandates that the solid angle of each tetrahedron incident at the center of the current cell should be less than $\pi/2$, and so we use the *Delaunay* tetrahedralization of the centers of the face/edge neighbors and the current cell. Note that ϕ-values were stored at grid nodes by Losasso et al. [2004], and they simply ignored missing neighbors near T-junctions as their grading scheme specifically coarsened *away* from the interface. However, our discretization for the pressure specifically requires ϕ-values at the cell centers to obtain second order accuracy near the free surface [Gibou et al., 2002], and we adapt the simulation grid topology even across the interface, thus our different approach. A slight subtlety is that these tetrahedral meshes should be computed *locally* per cell. To understand this consider the 2D illustration shown in the figure inset: $\Delta p_1p_2p_4$ and $\Delta p_1p_3p_4$ must be used at the point p_1 to ensure an acute angle, while $\Delta p_2p_1p_3$ and $\Delta p_2p_3p_4$ should be used at the point p_2. To extrapolate velocities outside the liquid region we first interpolate velocities from power faces to *regular* octree faces and use an approach similar to fast marching, this time operating on faces instead of cells and copying over the velocity value from the face closest to the free surface. For regular uniform cells, identifying face neighbors is straightforward, while near T-junctions we use the connectivity of the tetrahedral mesh to identify all faces incident to cells that correspond to nodes in the mesh. Subsequently, we interpolate back velocities from regular octree faces to all power faces. Figure 2.6 shows a comparison of the particle level set method [Enright et al., 2002] and our interface tracking scheme for a sphere that undergoes a deformation in a circular velocity field. The background grid resolution is 128^2 in 2D (or 128^3 in 3D); particle level set has 256 particles per cell, and our fine level set has 256 fine cells in 2D (or 4096 fine cells in 3D) per coarse cell.

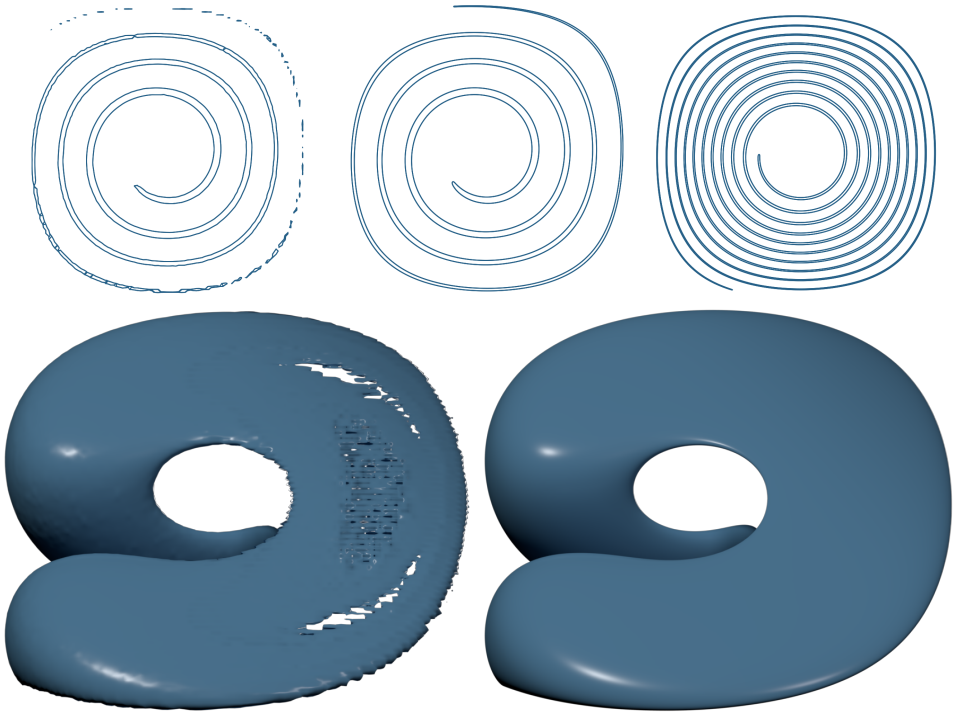


Figure 2.6: (Top) A comparison of (left) particle level set on a 128^2 grid, and (middle) our interface tracking scheme on a 2048^2 grid for a deforming sphere at $t = 5$ seconds; (right) our scheme at $t = 15$ seconds. (Bottom) A comparison of (left) particle level set on a 128^3 grid and (right) our method on a 2048^3 grid. For particle level set, near-interface cells use 256 particles. The narrow band level set conserves volume well, over long time periods.

2.6 Topology encoding and operator storage

The affinity of power diagrams to octrees is a crucial precondition for leveraging aggressive optimizations. In Section 2.4, we discussed how ghost cells in a pyramid of sparse uniform grids can facilitate the transfer of information between variables at different levels. It should be evident from our earlier discussion that the performance potential of our proposed storage paradigm is strongly predicated on the ability to easily load and store neighboring variables, and efficiently perform stencil applications. We now describe our scheme for encoding the local mesh connectivity and the stencils of the discrete Laplace and divergence operators in a compact form.

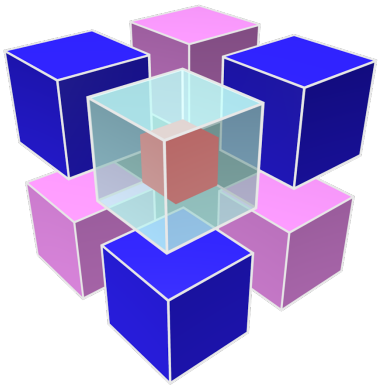
2.6.1 Encoding of local power diagram topology

Our encoding scheme is based on the observation that the local structure of each power cell is completely determined by its *face and edge neighbors*. The octree neighborhood for each cell can be trivially inferred from the SPGrid pyramid; in fact, every relevant property of its neighborhood can be inferred by looking exclusively at a *single* level of the pyramid. If a face/edge neighbor exists at the same resolution, it is flagged as active. Otherwise, if that same neighbor is flagged as ghost, there is a *coarser* neighbor on the other side of that face/edge. If a neighbor is neither present as an active or ghost cell, cells of *finer* resolution must reside on the other side. To ensure that each neighborhood admits a well-defined encoding, we grade such that all neighbors of a cell are (a) at the same resolution as the current cell or one coarser, or (b) at the same resolution as the current cell or one finer. We discuss both these cases below:

Cells with no coarser neighbor If a cell has no ghost neighbors, all its neighbors reside at the same resolution as itself or one level *finer*, giving rise to exactly 2^{18} distinct possibilities: each of the 6 face neighbors and 12 edge neighbors (18 neighbors total) is present at the same resolution, or absent (in which case four finer neighbors are present across each face, and two finer neighbors across each edge). We assemble this into an 18-bit number $N = b_0 b_1 b_2 \cdots b_{18}$, where every bit is defined as

$$b_j = \begin{cases} 1, & \text{if the j-th neighbor exists at same resolution} \\ 0, & \text{otherwise (neighbors are finer)} \end{cases}$$

Cells with no finer neighbor This is the complement of the previous case; our grading restriction ensures that for such a cell, all its neighbors reside at the same resolution as itself or one level *coarser*. To elucidate our encoding scheme, consider a cell (orange) and its coarser parent (light blue); there are 8 possible arrangements of the child cell within its parent, and for each of these configurations, there are only 6 possible coarse neighbors allowable due to parity reasons (see figure inset). We use 3 bits to encode the position of the child cell within its parent, and 6 bits to indicate whether this cell has a coarse neighbor in the X, Y, and Z (face) directions, and XY, YZ, and



XZ (edge) directions. For each configuration of the child cell, there can only be *one* coarse neighbor along each of these directions, making this encoding unique and well-defined. Thus, a total of 9 bits are required for this case.

Ultimately, we dedicate one more bit to encode which of the above two cases we have at hand, for a maximum of 19 bits that can capture all possible local topologies afforded by our grading scheme.

2.6.2 Retrieval of local Delaunay tetrahedralizations

Section 2.5 described our approach of using local Delaunay tetrahedralizations for fast marching and velocity interpolation; here we discuss their storage. Under our grading scheme, there are at most 114 geometrically distinct possibilities for the neighborhood of an octree cell; 18 of those are face and edge neighbors at the same resolution, 48 are cells of one finer resolution (4 cells across each of the 6 faces, and 2 across each of the 12 edges), and 48 are cells of one coarser resolution (8 configurations of a child in its parent cell, and 6 possible coarse neighbors in each case). Hence, all tetrahedra can only choose their vertices from 114 distinct cells, *relative* to the current cell. As a consequence, each vertex only requires a byte to encode (or 4 bytes per tetrahedron). We construct a lookup table with a (conservatively reserved) maximum of 128 tetrahedra incident to each node, requiring 512 bytes per topological case, or slightly more than 128MB to store the table for each of the $2^{18} + 2^9$ different local topologies. We populate this table on-demand, constructing the contents on-the-fly for any newly encountered case; our experiments have indicated that this table is very sparsely occupied, making it even easier for its contents to remain cached.

2.6.3 Hierarchical evaluation of differential operators

Efficient evaluation of the discrete Laplacian operator is critical to the performance of the Conjugate Gradient solver. Our treatment leverages the symmetry of this operator, allowing us to only store the “convenient half" of the coefficient pairs. We first use the routine GhostValuePropagate() helper to populate all ghost cells with the value of their coarser parent. The following three cases arise:

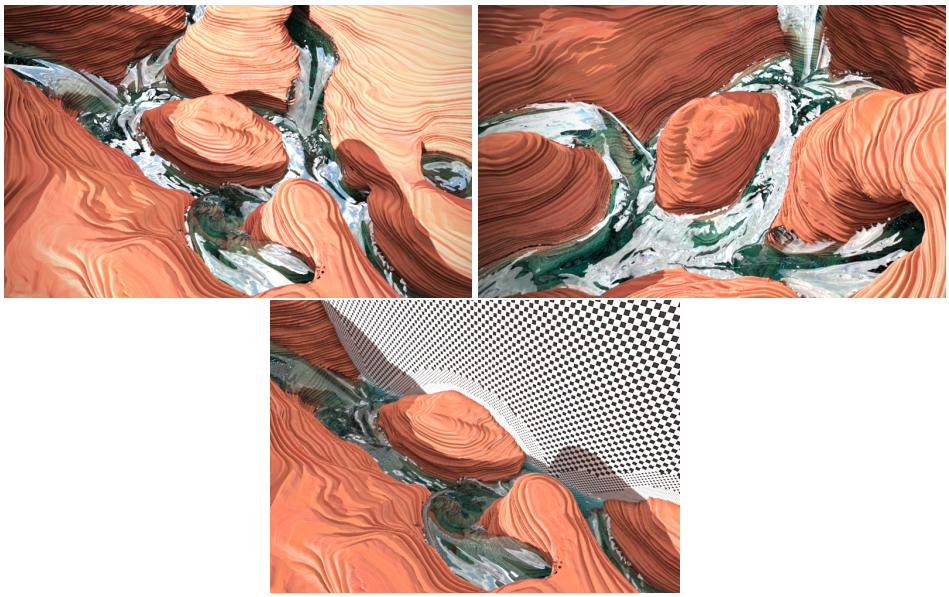


Figure 2.7: Close-up view of the simulation of a river flowing through a canyon. The adaptivity pattern, based on proximity to rigid boundaries, is shown along a vertical cross-section, on the bottom. A narrow band level set with resolution $2048^2 \times 4096$ is used for interface tracking.

Active cells with no finer neighbor The Laplacian for these cells can be readily computed using their 6 face neighbors and 12 edge neighbors (some of these neighbors may be ghost cells). We use a lookup table (identical in terms of indexing to the one storing the Delaunay tetrahedra) to retrieve the 19 stencil coefficients, corresponding to the central cell and its 18 neighbors. The storage cost is 19 floating point values or 76 bytes, for each of the 2^9 different topologies. To maximize alignment with cache lines, we pad this stencil up to 128 bytes, for a still nimble lookup table size of 64KB.

Ghost cells For each ghost cell, we use a 6 bit code to record which of its 6 topologically allowable neighbors (3 face and 3 edge neighbors, the direction being determined by parity) include this ghost cell in their stencil. Depending on the value of these bits, we look up the stencils of those neighboring cells and retrieve the coefficient for this ghost cell. All such contributions are added together. At the end of the Laplacian kernel, the routine GhostValueAccumulate() is invoked to accumulate all partial Laplace contributions temporarily deposited on ghost cells back to their coarse parents.

Active cells with no coarser neighbor In this case, we only consider the contribution of face/edge neighbors at the *same* resolution, and rely on the GhostValueAccumulate() routine to contribute the

missing stencil spokes to all finer neighbors. We retrieve 19 coefficients from our lookup table of 2^{18} possible local topologies. As one would expect, these coefficients are zero for all fine neighbors.

The evaluation of the discrete divergence operator is similar, with the exception of accessing the velocity for the corresponding face from one of the 6 different velocity channels. This approach only requires the storage of 19 coefficients in all cases, and retains the performance potential offered by the SPGrid framework.

2.7 Results

We simulated a number of examples to demonstrate the effectiveness of our framework. Each uses two or three levels of refinement, although this is not a strict limitation. For rendering the liquid surface, we used the fine level set wherever we had valid values, otherwise we used the coarse level set. Figure 2.1 shows two sources pouring water in a container, creating a thin sheet. The octree topology is fine near the sources and coarser away from them, with three levels of adaptivity. Figure 2.2 shows an example with dynamic adaptation, where four sources pour water in a pool with a rotating object. The octree topology is fine within a bounding box of the object and the sources, and coarser elsewhere, with two levels of adaptivity. The box enclosing the object is updated every frame, so that it can track complex solid-fluid interactions. During advection, whenever the mesh topology changes, we trace back from the new mesh face locations, but perform interpolation into the old mesh, thereby avoiding a separate mesh transfer step Klingner et al. [2006]. Finally, Figure 2.7 shows river flow in a canyon, with three levels of adaptivity. The thin geometry of this domain mandates more smoothing effort near the boundary (10 Jacobi smoothing iterations, as opposed to 3 in the other examples), increasing the cost of the projection step.

Table 2.2 shows the memory bandwidth utilization for a streaming copy kernel (for reference), the first order Laplacian of Setaluri et al. [2014], our optimized second order Laplacian, and an unoptimized Laplacian that uses an explicit mesh near T-junctions to store the stencil coefficients. These computations were performed on the river data set from Figure 2.7. Our kernel achieves good performance by exploiting the ghost cell mechanism of the SPGrid pyramid. Table 2.1 shows the average timing breakdown of one time step for all our examples. The memory footprint of our fine

	Fig. 2.1	Fig. 2.2	Fig. 2.7
Time Step	80	58	81
Level Set Advection	26	14	14
Reinitialization	20	15	19
Velocity Advection	5	2	5
Projection	24	18	40
Velocity Extrapolation	4	3	2
Grid Adaptation	N/A	5	N/A
Number of PCG iterations	5	4	10

Table 2.1: Average timing breakdown (in seconds) for all examples

level set representation was 8.09 GB (543M voxels) for the twin source example from Figure 2.1, 7.36 GB (493M voxels) for the rotating paddle example from Figure 2.2, and 3.84GB (253M voxels) for the river example from Figure 2.7. In spite of the exceptionally high resolution afforded in the narrow band, SPGrid yields a storage footprint which is quite acceptable for simulations of this scale. Finally, as Table 2.1 reveals, maintenance of the narrow band is efficient enough to not be the bottleneck, even though we rely on a serial Fast Marching method for reinitialization.

In the Appendix A, we use analytical test cases in two and three dimensions to provide numerical evidence that our projection technique achieves second order accuracy and our fast marching method achieves first order accuracy.

2.8 Limitations and future work

Our proposed combination of a power diagram discretization, storage using an SPGrid pyramid, and the use of a highly refined narrow band level set for interface tracking exposes and exploits a number of opportunities for performance optimizations, but also incurs some conscious limitations. In this section, we highlight the most notable limitations, and discuss whether they are intrinsic to our approach, or a temporary exclusion from our scope that could be amended without fundamental changes to our framework.

Our interface tracking scheme is expressly capable of capturing geometric details at higher resolutions than its counterpart stored on the power diagram. Since the dynamics are driven by the level set as sampled onto the octree power diagram, sub-grid droplets or air pockets may at times be

Streaming Copy	1st Order Laplacian	2nd order Laplacian	Unoptimized Laplacian
14.45	5.71	3.95	0.49

Table 2.2: Memory bandwidth utilization (in GB/sec) for streaming and stencil operations run on an Intel Xeon E3-1241 at 3.5GHz.

overlooked by the simulation, leading to non-physical motions such as the slender suspended splash in the rotating paddle scenario. Therefore, a sub-grid treatment similar to prior work [Bojsen-Hansen and Wojtan, 2013; Goldade et al., 2016] is called for. In addition, our interface tracking scheme uses first order semi-Lagrangian advection and the fast marching method for reinitialization, both of which are known to be overly dissipative. In the future, we would like to explore improved advection schemes, such as FLIP [Zhu and Bridson, 2005] or MacCormack [Selle et al., 2008], and higher order accurate level set reinitialization. Reconstructing full velocities at tetrahedra circumcenters rather than nodes and applying generalized barycentric interpolation over polyhedra would also reduce dissipation [Klingner et al., 2006; Elcott et al., 2007]. Moreover, in principle, it is possible to use adaptivity even for interface tracking while, for simplicity, we have restricted our implementation to a single uniform grid. We are presently reinitializing the signed distance with a serial Fast Marching method on the narrow band; alternative reinitialization schemes that admit parallelism certainly merit attention in future work. Finally, given that our method makes extensive use of the SPGrid data structure to solve the Navier-Stokes equations, tracking the surface with a high-resolution narrow band SPGrid level set was a natural and convenient choice. Nevertheless, our underlying octree fluid solver is not intrinsically tied to this choice, and we expect that it could alternatively be paired with a purely particle- or mesh-based surface tracking strategy if desired.

Thus far we have sought to simulate strictly large-scale single-phase inviscid free surface flows, which leaves ample room for future exploration of more specialized effects such as surface tension, contact-line dynamics, viscosity, multiple phases, solid-fluid interaction, and so on. Some of these extensions should be straightforward; for example, explicit surface tension should involve only a minor modification to the right-hand-side of the linear system based on surface curvature [Enright et al., 2003].

As discussed in Section 2.4, edge neighbors in the octree may share a face in the power diagram.

We use three additional channels associated with edges for storing velocities on these faces. These channels only store meaningful values near T-junctions, resulting in a non-optimal storage density. It is also possible to use the non-graded approach of Losasso et al. [2006] in regions deep interior to the liquid, and our power diagram discretization only near the free surface to obtain a lower cost Laplace operator that still yields a second order accurate pressure field. We have tested this idea on a simple prototype and plan to include it in our simulation pipeline in future work. Finally, we have consciously restricted our scope to a grading scheme that mandates that a cell and all its neighbors span only two levels of resolution; however, there exist weaker grading rules that do not destroy the essential octree structure in the power diagram, and lifting this restriction would not prevent us from using the ghost cell mechanism within the SPGrid framework either. However, in the absence of this grading restriction, our encoding and lookup scheme for local tessellations and stencils would not apply. We look forward to investigating algorithmic paradigms for more intricate adaptive topology patterns in future work.

3 ADAPTIVE GENERALIZED INTERPOLATION MATERIAL POINT METHOD

In this chapter, we present an adaptive Generalized Interpolation Material Point (GIMP) method for simulating elastoplastic materials. Our approach allows adaptive refining and coarsening of different regions of the material, leading to an efficient MPM solver that concentrates most of the computation resources in specific regions of interest. We propose a C^1 continuous adaptive basis function that satisfies the partition of unity property and remains non-negative throughout the computational domain. We develop a practical strategy for particle-grid transfers that leverages the recently introduced SPGrid data structure for storing sparse multi-layered grids. We demonstrate the robustness and efficiency of our method on the simulation of various elastic and plastic materials. We also compare key kernel components to uniform grid MPM solvers to highlight performance benefits of our method.

3.1 Introduction

The Material Point Method (MPM) has been attracting considerable interest since it was introduced to the field of computer graphics by Stomakhin et al. [2013]. Combining advantages from both Lagrangian particle representation and Eulerian grid representation, MPM proves to be especially effective for animating elastoplastic materials undergoing large deformation or topology change [Jiang et al., 2016]. Despite its physical realism and geometrical convenience, a traditional MPM solver has several disadvantages. First, it is more computationally expensive than mesh-based Lagrangian approaches such as those based on Finite Element Methods (FEM) [Sifakis and Barbic, 2012]. The bottleneck of MPM is usually the costly transfer operations between the particles and the grid. The cost of such transfer operations is particularly evident when we realize that MPM has to maintain the same grid resolution and a sufficient particle count throughout the simulation domain. The overhead of this process is highlighted in scenarios such as the example of drawing in a sandbox from Klár et al. [2016], where the majority of sand grains do not move at all.

Another disadvantage of traditional MPM is related to its ability to resolve (self-) collision events.

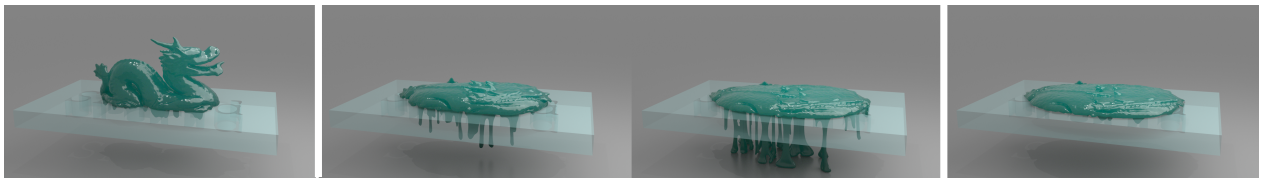


Figure 3.1: **Dragon goo**. An elastoplastic dragon model is placed on a table carved with intricate thin slits. Our adaptive simulation (center) refines the background grid in the vicinity of the collision object, allowing the dragon to seep through, while a simulation on a uniform grid (right) cannot resolve the contact.

MPM automatically enforces non-slip contact. However it treats two particles as being in contact whenever they affect some common grid nodes. As a result, the separation distance is inherently proportional to the grid cell spacing Δx. High resolution is required in order to prevent visually noticeable collision gaps even for the simulation with very simple dynamics. Furthermore, MPM cannot resolve a boundary condition that is finer than the grid resolution. This implies that materials cannot pass through holes smaller than Δx. Similarly, a blade that is thinner than Δx can not cut through materials.

We propose an adaptive variant of the Generalized Interpolation Material Point (GIMP) method [Bardenhagen and Kober, 2004] to alleviate these limitations. By only refining regions of particular interest, we can resolve fine-grained self contact and object collision features with significantly reduced computational cost. We demonstrate the efficacy of our method on the simulation of various elastic and plastic materials. We summarize our main contributions as:

- The introduction of the GIMP paradigm to MPM simulation for computer graphics applications.
- An *adaptive* GIMP discretization framework with significantly improved shape functions to enforce important properties such as C^1 continuity and non-negativity.
- A memory efficient and highly regular parallel particle-grid transfer scheme that achieves attractive performance on both uniform and arbitrary *non-graded* adaptive grids. We show how all the necessary operations in our adaptive grids can be naturally paired with the SPGrid data structure, and implemented using efficient, uniform grid operations as building blocks.

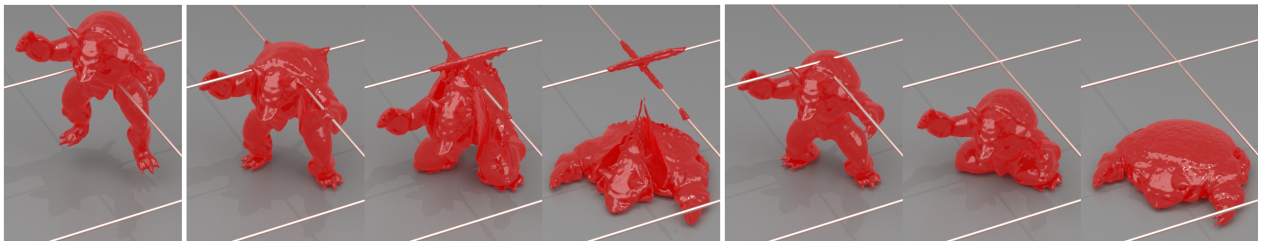


Figure 3.2: **Armadillo wire cut**. A gooey armadillo is dropped through two thin intersecting wires. *Left:* the model just prior to collision; *Center:* grid refined to 4× the base resolution in the vicinity of the wires, per our method; *Right:* A uniform grid with comparable particle count largely misses the collision event.

- A highly optimized (threaded and vectorized) particle-grid transfer approach with a number of aggressive vectorization optimizations that were specifically enabled by our proposed perspective on implementing grid adaptivity via the multi-level SPGrid representation.

Our method is competitive with a uniform gird on a per-cell or per-particle basis. We also note that as opposed to the approach proposed by Lian et al. [2015] that assumes a *graded* adaptive Cartesian grid where neighboring cells do not differ by more than one refinement level, our approach in constructing an adaptive grid is relatively straightforward and simple to implement. Our grid adaptivity also remains efficient regardless of the complexity of refinement levels. Additionally, our transfer operators achieve a very rigorous standard of optimality, matching or exceeding the bandwidth of state-of-the-art parallel uniform MPM codes with the same grid cell and/or particle count.

Scope The principal objective of this work is to propose an adaptive MPM scheme, equipping the weights with all desired properties, which also has the potential to realize competitive performance, and can facilitate and catalyze follow-up work on this topic. We admit, however, that our current framework does not prove that adaptive MPM will always deliver performance superior to uniform MPM, due to the fact that explicit time integration is arguably the least favorable for reaching aggressive end-to-end performance advantages over uniform schemes.

3.2 Related work

Adaptive simulation of deformable solids The use of spatial adaptivity to mitigate the cost of simulating detailed elastic and elastoplastic bodies has been widely documented in the computer graphics literature. Debunne et al. [1999] proposed one of the early continuum-based adaptive volumetric simulation techniques using finite differences, departing from prior schemes based on mass-spring systems. Similar ideas were soon combined with Finite Element (FEM) Methods [Capell et al., 2002] where a hierarchical deformation basis would be constructed via subdivision, with individual bases activated or disabled based on deformation. Similarly, Grinspun et al. [2002] cater to adaptive FEM simulation of volumetric solids and shells by refining the deformation basis, rather than the elements themselves, providing a natural handling of T-junctions at resolution transitions. Topology change, instigated by cutting operations and material fracture provided a compelling context for adaptive simulation, with a number of authors exploring octree-based discretizations of elastic solids [Seiler et al., 2011] which also allowed the use of efficient multi-resolution solvers [Dick et al., 2011], while others combined octrees with shape matching techniques in modeling cutting operations [Steinemann et al., 2008]. Localized adaptation in response to high deformation provided one of the most natural motivations for Discontinuous Galerkin methods [Kaufmann et al., 2009].

Although several of the aforementioned techniques are based on octrees, a number of researchers focused on adaptive tetrahedral discretizations readily produced by robust meshing algorithms [Labelle and Shewchuk, 2007] which have been very popular in modeling elastoplastic deformation [Wicke et al., 2010]. Specifically, adaptive tetrahedralizations based on BCC lattices have been used to model viscoelastic [Wojtan and Turk, 2008] as well as hyperelastic materials [Sifakis et al., 2007]. Although contact and collision handling is often designed independently of adaptive refinement, in certain instances as in the work of Otaduy et al. [2007] the two behaviors are tightly integrated as to concurrently accelerate elastic simulation and collision detection. Finally, for a deeper survey of adaptive techniques for deformable models, we refer to the excellent recent report by Manteaux et al. [2016].

Adaptive fluids Adaptive fluid simulation has been studied extensively by previous approaches since the work of Losasso et al. [2004] on octree-based water and smoke. Tetrahedral based adaptive

fluid simulation was combined with embedded boundary methods by Batty et al. [2010a]. Ando et al. [2012] adopted particle splitting and merging for liquid sheets. Ando et al. [2013] simulated highly detailed splashes on a novel FLIP scheme discretized on an adaptive tetrahedral mesh. Ferstl et al. [2016] proposed a narrow band FLIP that couples with an Eulerian solver. Adaptive sampling methods were also developed for SPH [Adams et al., 2007; Solenthaler and Gross, 2011].

Sparse and adaptive grid structures Although pointer-based tree structures [Losasso et al., 2004] are the most straightforward choice for storing adaptive grids, the under-utilization of memory bandwidth associated with indirect access and suboptimal prefetching that is intrinsic to pointer-based trees has led researchers to explore alternative storage structures. RLE-based techniques [Houston et al., 2006; Irving et al., 2006a; Chentanez and Muller, 2011] combine the regularity of a 2D uniform grid with the compression of a 1D run-length encoding. Adaptive Mesh Refinement (AMR) techniques and variants thereof [Patel et al., 2005; Cohen et al., 2010; English et al., 2013a] combine adaptivity and regularity by patching together uniform grids of different resolutions. One of the most efficient and broadly used adaptive data structures, OpenVDB [Museth et al., 2013], is based on a tree with a high branching factor that yields large uniform grids at leaf nodes. Our present work is based on the recently developed Sparse Paged Grid (SPGrid) data structure [Setaluri et al., 2014], which targets in-core processing and exploits the hardware accelerated mechanisms that support the Virtual Memory subsystem of modern CPUs. Apart from a sparse grid structure, Setaluri et al. [2014] paired SPGrid with a different perspective of octrees, treating them as stacks of sparsely populated *uniform* grids, across which the cells of a geometric octree are scattered. This concept has been subsequently applied to simulation of high-resolution fluids [Liu et al., 2016] and hybridized with second-order accurate techniques for incompressible flow [Aanjaneya et al., 2017].

Material Point Methods MPM [Sulsky et al., 1995] is a generalization of the hybrid Fluid Implicit Particle (FLIP) method [Brackbill et al., 1988; Zhu and Bridson, 2005; Bridson, 2008] to solid mechanics. It has proven to be a promising discretization choice for animating many solid materials including snow [Stomakhin et al., 2013], foam [Yue et al., 2015; Ram et al., 2015], sand [Daviet and Bertails-Descoubes, 2016; Klár et al., 2016], cloth [Jiang et al., 2017a] and solid-fluid mixture [Stomakhin et al., 2014]. The original GIMP concept [Bardenhagen and Kober, 2004] was described in the context

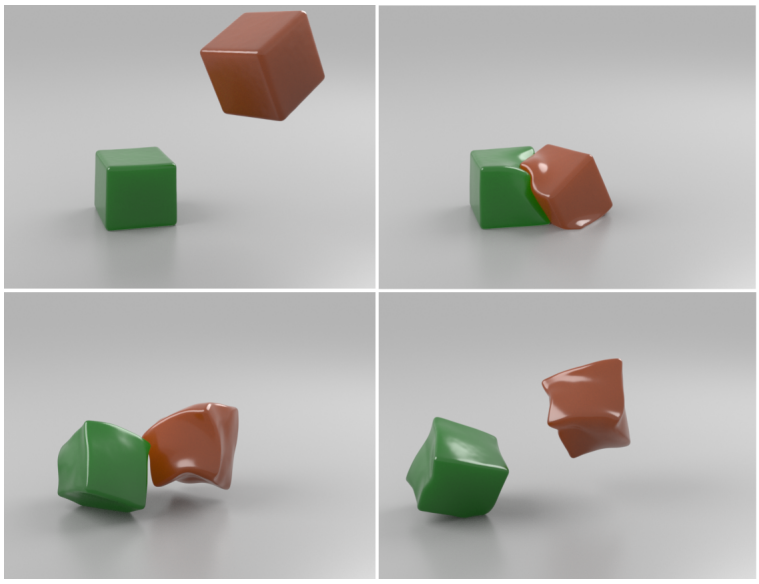


Figure 3.3: **3D colliding jellos.** Two colliding jellos are simulated, where we placed finer particles and grid resolution near the free surface of the material.

of uniform grids. There is some engineering literature exploring adaptive MPM. Early work of Tan and Nairn [2002] mainly focused on particle splitting. Some initial ventures into coupling GIMP and adaptivity [Ma et al., 2005, 2006; Daphalapurkar et al., 2007] managed to enforce both partition of unity and C^1 continuity, but their applications were restricted to nested grids (not a general octree). Lian et al. [2014] employed the concept of embedding fine T-junctions in coarse parents, but only along the surfaces separating resolution levels. More recently, Lian et al. [2015] developed the mesh-grading MPM (MGMPM) that modified the shape functions while maintaining the partition of unity. However their method, as we will see later, is prone to generating negative interpolation weights in certain scenarios, risking potential instability. The shape functions in both [Lian et al., 2014] and [Lian et al., 2015] are not based on GIMP and only C^0 continuous. Our method, on the other hand, guarantees all desired properties: partition of unity, non-negativity and C^1 continuity for the weights.

3.3 Generalized interpolation material point method

The Generalized Interpolation Material Point (GIMP) method, proposed by Bardenhagen and Kober [2004], is a generalization of the original MPM to allow a wider range of interpolation functions

between the particles and the grid.

3.3.1 Governing equations

Before discussing GIMP, we first briefly review the original MPM. We refer to [Stomakhin et al., 2013] for more details. The simulated material is perceived as a continuum body which is a subset of $\mathbb{R}^d$. Here d denotes the spatial dimension, which can be 2 or 3. At any give time t, there is a deformation mapping $\boldsymbol{\varphi}(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^d$, which maps points in the undeformed configuration to a deformed configuration. More precisely, a point $\mathbf{X} \in \Omega^0$ is mapped to $\mathbf{x}(\mathbf{X}, t) = \boldsymbol{\varphi}(\mathbf{X}, t) \in \Omega^t$. The deformation gradient $\mathbf{F} = \partial\boldsymbol{\varphi}/\partial\mathbf{X}$ describes the material deformation and acts as a common strain measure [Bonet and Wood, 2008]. We denote the determinant of the deformation gradient by $J := \det(\mathbf{F})$.

The governing equations we need to solve are the conservation of mass and conservation of momentum. They are written as

$$\frac{D\rho}{Dt} + \rho\nabla\cdot\mathbf{v} = 0 \quad \text{and} \quad \rho\frac{D\mathbf{v}}{Dt} = \nabla\cdot\boldsymbol{\sigma} + \rho\mathbf{g} \tag{3.1}$$

respectively, where D/Dt is the material derivative ($\frac{Df}{Dt} := \frac{\partial f}{\partial t} + \mathbf{v}\cdot\nabla f$ for a generic scalar function f), ρ is the density, $\mathbf{v}$ is the velocity, $\mathbf{g}$ is gravity, $\boldsymbol{\sigma}$ is the Cauchy stress. We assume that there exists an elastic energy density function $\Psi(\mathbf{F})$, so that the Cauchy stress can be written as $\boldsymbol{\sigma} = \frac{1}{J}\frac{\partial\Psi}{\partial\mathbf{F}}\mathbf{F}^T$ (see Section 3.4 for more discussion on the physical model).

3.3.2 Discretization

First, we explain the notation used in this chapter. Subscript i is an index that enumerates grid nodes, while p is an index that enumerates discrete particles. Certain quantities relating grid nodes and particles will carry a double subscript (ip), as will be the case with interpolation functions and weights. Superscript n identifies the discrete time step associated with a time-varying quantity.

In MPM, particles carry attributes such as mass (m_p), position ($\mathbf{x}_p$), velocity ($\mathbf{v}_p$), deformation gradient ($\mathbf{F}_p$), and other material parameters. A background grid is used as a scratch pad to discretize

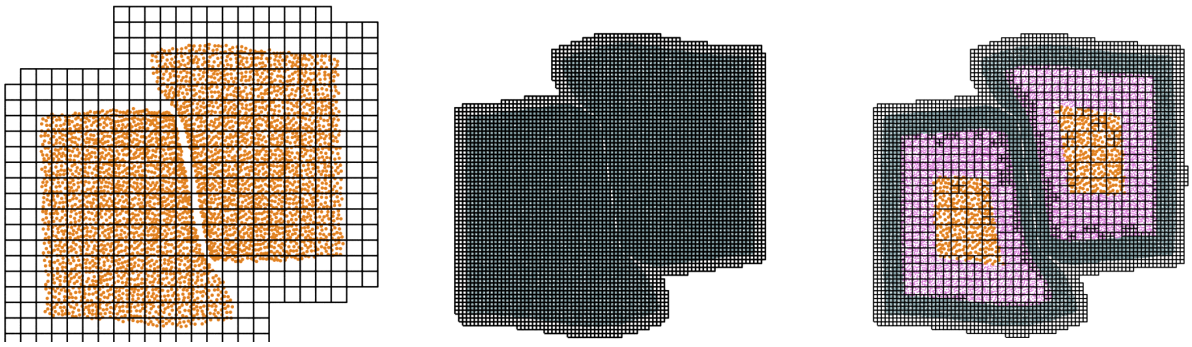


Figure 3.4: **Colliding 2D jello squares**. Two hyperelastic jello squares are driven to collide with each other. *Left:* A uniformly coarse grid leaves large separation gaps. *Middle:* A $4\times$-refined uniform grid nicely resolves the contact, at the expense of gratuitous computation in the interior. *Right:* Our adaptive scheme.

and solve the governing equations. The communication between particles and grid information is handled through an interpolation function. We denote the weight and the corresponding gradient between particle p and grid node i with w_{ip} (a scalar) and ∇w_{ip} (a vector), respectively.

Here we provide an overview of MPM stages in a time step, assuming an explicit symplectic-Euler time integration. GIMP follows exactly the same procedure.

1. **Particle to grid.** Assuming we are at time n, particle mass and momentum are transferred from particles to grid nodes with $m_i^n = \sum_p m_p w_{ip}^n$ and $(m\mathbf{v})_i^n = \sum_p m_p \mathbf{v}_p^n w_{ip}^n$. The velocity of node i is computed as $\mathbf{v}_i^n = (m\mathbf{v})_i^n / m_i^n$ when $m_i^n \neq 0$. It is set to $\mathbf{0}$ otherwise.

2. **Compute grid forces.** This comes from discretizing the conservation of momentum. For node i, the force is given by $\mathbf{f}_i^n = m_i^n \mathbf{g} - \sum_p V_p^0 J_p^n \boldsymbol{\sigma}_p^n \nabla w_{ip}^n$, where V_p^0 is the undeformed particle volume.

3. **Grid velocity update.** Denoting the updated grid node velocities with $\hat{\mathbf{v}}_i$, symplectic Euler computes it as $\hat{\mathbf{v}}_i = \mathbf{v}_i^n + \Delta t \mathbf{f}_i^n / m_i^n$.

4. **Collision treatment.** $\hat{\mathbf{v}}_i$ for each grid node is further processed for nodes inside collision objects. The relative velocity is set to $\mathbf{0}$ inside "sticky" collision objects. The normal component is set to $\mathbf{0}$ in "slip" collision objects.

5. **Strain evolution.** The particle deformation gradient $\mathbf{F}_p$ evolves as $\mathbf{F}_p^{n+1} = \left(\mathbf{I} + \Delta t \sum_i \hat{\mathbf{v}}_i (\nabla w_{ip}^n)^T\right) \mathbf{F}_p^n$.

6. **Grid to particle.** Particle velocities and positions are updated from grid velocities with $\mathbf{v}_p^{n+1} = \alpha(\mathbf{v}_p^n + \sum_i(\hat{\mathbf{v}}_i - \mathbf{v}_i^n)w_{ip}^n) + (1-\alpha)\sum_i \hat{\mathbf{v}}_i w_{ip}^n$ and $\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \sum_i \hat{\mathbf{v}}_i w_{ip}^n$, where $\alpha = 0.95$ [Stomakhin et al., 2013].

3.3.3 From MPM to GIMP

Traditional MPM [Sulsky et al., 1995] defines the weight w_{ip} between particle p and grid node i to be exactly the interpolation function $N_i(\mathbf{x})$ of node i evaluated at $\mathbf{x}_p$:

$$w_{ip} = N_i(\mathbf{x}_p). \tag{3.2}$$

Accordingly the weight gradient is $\nabla w_{ip} = \nabla^{\mathbf{x}} N_i(\mathbf{x}_p)$. Here $N_i(\mathbf{x})$ is the standard finite element trilinear basis function in 3D (bilinear in 2D) defined for node i.

While this choice of particle-grid weights works fine for problems with small deformation, Steffen et al. [2008] observes that it suffers from the "cell crossing instability" for practical problems. This is because piecewise linear basis functions are only C^0 continuous at cell boundaries. The corresponding gradient ∇w_{ip} is therefore discontinuous. Two apparent problems are associated with this property. First of all, it is possible that a huge force is exerted on a node with tiny mass, causing numerical issues. Secondly, discontinuity of the gradient implies discontinuity of the force (cf. Step 2 in the the MPM/GIMP stages). Noise and instability may thus occur as a particle travels across cells. These reasons render this choice of basis function to be ineffective for traditional MPM. We note that this is not a problem for other hybrid particle-grid methods such as a FLIP fluid solver because the weight gradient is not needed.

One convenient solution to remedy the cell crossing instability is to use higher order C^1 or even C^2 continuous interpolation functions such as quadratic or cubic B-splines as in Stomakhin et al. [2013]. Unfortunately there are not many other choices of the interpolation function, given the following constraints:

- **Partition of unity:** $\sum_i w_{ip} = 1, \forall \mathbf{x}_p$. This is required for mass and momentum conservation [Jiang et al., 2015].

- **Non-negativity:** $w_{ip} \geqslant 0$.
- **Interpolation:** $\mathbf{x}_p = \sum_i w_{ip}\mathbf{x}_i$.
- C^1 **continuity:** w_{ip} needs to be at least C^1 continuous.
- **Local support:** w_{ip} is only non-zero for $\mathbf{x}_p$ near $\mathbf{x}_i$.

The last requirement is primarily due to practical considerations. The non-negativity constraint rules out the possibility of high order FEM basis functions such as 9-node quadratic quadrilateral elements or 8-node serendipity quadrilateral elements in 2D (and their corresponding shape functions for hexahedral meshes in 3D) [Hughes, 2012]. As we will discuss more in detail in Section 3.5.1, allowing negative weights may cause serious issues when mass on a grid node becomes negative [Andersen and Andersen, 2007].

GIMP is another alternative to eliminating the cell crossing instability. Instead of choosing $N_i(\mathbf{x})$ to be C^1, GIMP constructs weights with C^1 continuity, using as building blocks bases $N_i(\mathbf{x})$ with just C^0 continuity, often chosen to be the standard multilinear basis.

The full simulation domain is denoted by Ω. Associated to a particle p is the notion of particle domain Ω_p surrounding it (as yet of unspecified shape; will be chosen to be an axis aligned box in our implementation). Thus, the volume of a particle can be computed as

$$\hat{V}_p = \int_{\Omega \cap \Omega_p} d\mathbf{x}. \tag{3.3}$$

Unlike Equation 3.2, GIMP defines the weight to be

$$w_{ip} = \frac{1}{\hat{V}_p} \int_{\Omega} \chi_p(\mathbf{x}) N_i(\mathbf{x})\, d\mathbf{x}, \tag{3.4}$$

with $\chi_p(\mathbf{x})$ being the particle characteristic function defined over the particle domain Ω_p.

The traditional MPM is a special case of GIMP where $\chi_p(\mathbf{x}) = \hat{V}_p \delta(\mathbf{x} - \mathbf{x}_p)$. Common GIMP

schemes often choose $\chi_p(\mathbf{x})$ to be the characteristic indicator function centered at $\mathbf{x}_p$, i.e.,

$$\chi_p(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \Omega_p, \\ 0 & \text{otherwise.} \end{cases}$$

We model Ω_p as an axis-aligned box centered at $\mathbf{x}_p$. Another common simplification that is often made is that Ω_p does not rotate or deform. This is also known as uGIMP (undeformed GIMP). Thus the integral is always evaluated in a box with size $\hat{V}_p = L_p^d$, where L_p is the fixed side length of particle p's box and $d = 2$ or 3 is the dimension. Under this choice of $\chi_p(\mathbf{x})$, Equation 3.4 simplifies to

$$w_{ip} = \frac{1}{\hat{V}_p} \int_{\Omega_p} N_i(\mathbf{x}) d\mathbf{x}. \tag{3.5}$$

Similarly, we define

$$\nabla w_{ip} = \frac{1}{\hat{V}_p} \int_{\Omega_p} \nabla^{\mathbf{x}} N_i(\mathbf{x}) d\mathbf{x}. \tag{3.6}$$

Recall that $N_i(\mathbf{x})$ is piecewise linear and C^0 continuous. Since the weights result from a convolution of $N_i(\mathbf{x})$ with the indicator function of Ω_p, w_{ip} becomes C^1 continuous and ∇w_{ip} is C^0 continuous if they are viewed as functions of $\mathbf{x}$. Additionally, w_{ip} satisfies all constraints mentioned previously.

GIMP weights are inherently smooth. We note that GIMP further recovers traditional MPM with quadratic B-spline interpolation on a uniform grid when $L_p = \Delta x$. We can see this from the convolution definition of uniform B-spline functions. More specifically, assuming $\Delta x = 1$ in 1D, the B-spline of order-0 is given by

$$N^0(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in [0, 1], \\ 0 & \text{otherwise.} \end{cases}$$

If we convolute $N^0(\mathbf{x})$ with itself as $N^1(\mathbf{x}) = N^0(\mathbf{x}) * N^0(\mathbf{x})$, we can see $N^1(\mathbf{x})$ is exactly the piecewise linear order-1 B-spline with support $[-1, 1]$. We can further write $N^2(\mathbf{x}) = N^1(\mathbf{x}) * N^0(\mathbf{x})$ and $N^3(\mathbf{x}) = N^2(\mathbf{x}) * N^0(\mathbf{x})$, or generally $N^k(\mathbf{x}) = N^{(k-1)}(\mathbf{x}) * N^0(\mathbf{x})$ to recursively construct uniform higher order B-splines. Consequently, by noticing $N^0(\mathbf{x}) = \chi_p(\mathbf{x})$ when $L_p = \Delta x$, we see that the

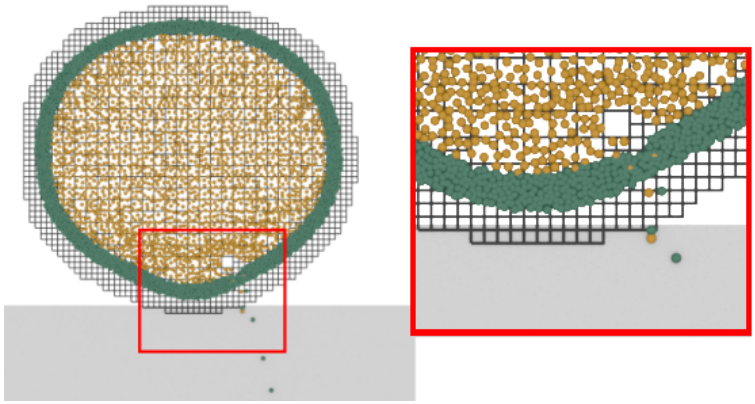


Figure 3.5: Negative weights causing instability in a sphere dropping experiment.

GIMP shape function and the quadratic B-spline shape function are identical in this case. This result generalizes to 2D and 3D due to the property that multi-dimension B-splines are simply tensor products of univariate 1D B-splines.

Even though B-splines MPM and GIMP can be made equivalent on a uniform grid, it is much more difficult to generalize high order B-splines to an adaptive grid. Much of the difficulty lies in the treatment of T-junctions and transition grid cells between different levels. On the other hand, the GIMP view from Equation 3.4 is naturally defined on an adaptive grid, as long as a C^0 continuous piecewise linear function $N_i(\mathbf{x})$ is well defined. We show how to robustly construct such basis functions in Section 3.5.

3.4 Elastoplasticity

We adopt finite strain elastoplasticity to model different material behaviors. In this section we follow [Bonet and Wood, 2008] and assumes that the deformation gradient $\mathbf{F}$ is decomposed into elastic and plastic parts as $\mathbf{F} = \mathbf{F}^E\mathbf{F}^P$. In this chapter we only consider (i) purely elastic objects, (ii) elastoplastic von Mises materials, and (iii) granular Drucker-Prager sand. However, our approach can be easily combined with any constitutive models.

3.4.1 Hyperelasticity

In hyperelasticity setting, there exists an elastic energy density function $\Psi(\mathbf{F}^E)$ that penalizes the deviation of $\mathbf{F}^E$ from a pure rotation. As such, the first Piola-Kirchhoff stress is determined by $\mathbf{P} = \frac{\partial \Psi}{\partial \mathbf{F}^E}(\mathbf{F}^P)^{-T}$. The Cauchy stress can be obtained as $\boldsymbol{\sigma} = \frac{1}{J}\mathbf{P}\mathbf{F}^T$. In this chapter, we decided to choose the fixed-corotated model [Stomakhin et al., 2012] for purely elastic solids due to its robustness under large deformation, where Ψ is defined as $\Psi(\mathbf{F}^E) = \mu\|\mathbf{F}^E - \mathbf{R}^E\|_F^2 + \frac{\lambda}{2}(J^E - 1)^2$, where $\mathbf{R}^E$ is the rotation tensor from the polar decomposition $\mathbf{F}^E = \mathbf{R}^E\mathbf{S}^E$, $J^E = \det(\mathbf{F}^E)$, μ and λ are Lamé parameters. For elastoplastic materials, it is often more convenient to adopt the Hencky strain $\boldsymbol{\epsilon} = \frac{1}{2}\log(\mathbf{F}\mathbf{F}^T)$ from Mast [2013] and to use the St. Venant-Kirchhoff energy density

$$\Psi(\mathbf{F}^E) = \mu \mathrm{tr}(\log(\boldsymbol{\Sigma}^E)^2) + \frac{1}{2}\lambda(\mathrm{tr}(\log(\boldsymbol{\Sigma}^E)))^2, \tag{3.7}$$

where $\boldsymbol{\Sigma}$ comes from SVD of $\mathbf{F}^E$: $\mathbf{F}^E = \mathbf{U}^E\boldsymbol{\Sigma}^E(\mathbf{V}^E)^T$.

3.4.2 Plasticity

In this chapter we consider two plasticity models, namely the Drucker-Prager model with a non-associative flow rule for granular materials, and the von Mises model with an associative flow rule for perfectly plastic materials. We use Equation 3.7 for the elastic response.

Drucker-Prager

The Drucker-Prager yield surface can be derived by applying the Mohr-Coulomb friction law in a continuum. The admissible stress is inside the region given by $y(\boldsymbol{\sigma}) = c_F \mathrm{tr}(\boldsymbol{\sigma}) + \|\boldsymbol{\sigma} - \frac{\mathrm{tr}(\boldsymbol{\sigma})}{d}\mathbf{I}\|_F \leqslant 0$, where c_F is the coefficient of internal friction. Discretely, if the trial stress is outside the yield region, it is projected back onto the yield surface through an closed-form solution of the return mapping [Klár et al., 2016]. The original return mapping causes non-physical volume gain when the material is under expansion. We adopt the volume correction treatment as in Tampubolon et al. [2017] to eliminate this artifact.

von Mises

Von Mises plasticity is widely used for metal and ductile materials. It is also closely related to computer graphics plasticity models that were successfully applied for simulating plastic flow and gooey materials [Bargteil et al., 2007; Wojtan and Turk, 2008]. In 3D, the von Mises yield surface is defined as $y(\boldsymbol{\sigma}) = \sqrt{3J_2} - \sigma_y \leqslant 0$, where σ_y is the yield stress, $J_2 = \frac{1}{2}\mathbf{s} : \mathbf{s}$ is the second invariant of the deviatoric stress $\mathbf{s}$. Here $\mathbf{s} := \boldsymbol{\sigma}^{dev} = \boldsymbol{\sigma} + p\mathbf{I}$ and $p = -\frac{1}{3}\text{tr}(\boldsymbol{\sigma})$. Unlike the Drucker-Prager yield surface for granular materials, von Mises plasticity uses an associative flow rule. The discrete flow rule thus chooses $\frac{\partial y}{\partial \sigma}$ as its direction. In the principal stress space, the shape of von Mises yield surface is an infinitely long cylinder centered at the hydrostatic axis ($\boldsymbol{\sigma}_1 = \boldsymbol{\sigma}_2 = \boldsymbol{\sigma}_3$, where $\boldsymbol{\sigma}_i$ is the i'th eigenvalue of $\boldsymbol{\sigma}$). Following the same methodology in [Klár et al., 2016], we derive a closed form solution of the return mapping for von Mises plasticity. Assuming $\boldsymbol{\epsilon}^E := \log(|\mathbf{F}^E|)$ is the trial Hencky strain in the principal space, when the trial stress is outside the yield region, we project $\boldsymbol{\epsilon}^E$ to the yield surface to obtain a new Hencky strain $\mathbf{H}^E = \boldsymbol{\epsilon}^E - \delta\gamma \frac{\text{dev}(\boldsymbol{\epsilon}^E)}{\|\text{dev}(\boldsymbol{\epsilon}^E)\|}$, where $\delta\gamma = \|\text{dev}(\boldsymbol{\epsilon}^E)\| - \frac{\sigma_y}{2\mu}$ and $\text{dev}(\boldsymbol{\epsilon}^E) = \boldsymbol{\epsilon}^E - \frac{\text{tr}(\boldsymbol{\epsilon}^E)}{3}\mathbf{I}$.

3.5 Adaptive basis functions

We employ a spatially adaptive computational grid. As the simulation proceeds, the grid must locally refine and coarsen to hierarchical refinement levels. Instead of trying to define a C^1 continuous B-spline-like interpolation function that covers all cases and T-junctions, we degrade the problem into defining a C^0 function. As discussed in Section 3.3.3, GIMP will use convolution to convert the C^0 continuous interpolation function $N_i(\mathbf{x})$ to a C^1 continuous shape function w_{ip}.

3.5.1 Adaptive grid basis with free T-junctions

We start by describing the state-of-the-art multi-level grid approach by Lian et al. [2015]. Cell spacing of the grid of level n is $\Delta x_n = \Delta x_0/2^n$, where Δx_0 is the coarsest cell spacing. This results in additional hanging nodes at T-junctions at the interface between cells of different refinement levels.

Taking the case of Figure 3.6(a) as an example, cell 2 and 3 are of a level higher than cell 1. Node b is a T-junction node at the level transition interface. In the formulation of Lian et al. [2015] it is

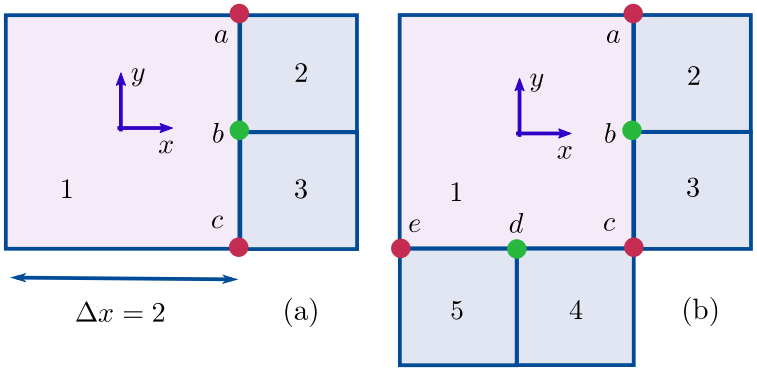


Figure 3.6: **Illustration of free T-junction.** A test case for the adaptive grid basis with free T-junctions described in Section 3.5.1. The interpolation functions remain non-negative in the case of (a). However, in the (b) case where there exist two T-junction nodes, N_c may turn negative and become problematic.

treated as an actual degree of freedom on the grid. Inside cell 2 and 3, the interpolation functions associated with nodes a, b, c are simply the standard bilinear hat functions. For example, if we denote the hat function basis of node γ in cell k with H^k_γ, and the grid basis interpolation function with N^k_γ, we have

$$N^2_a = H^2_a,\ N^2_b = H^2_b,\ N^3_b = H^3_b,\ N^3_c = H^3_c. \tag{3.8}$$

In other words, within cells that have no T-junctions in their periphery, the shape functions of their corner vertices are the standard bilinear hat functions; this is not the case, however for cells containing T-junctions (such as cell 1 in Figure 3.6), neither for the corner vertices or T-junction nodes associated with such cells. N^1_b must be defined in a way that ensures continuity of N_b along edge $a - b - c$. If we parameterize cell 1 with a coordinate system spanning $[-1, 1]^2$ with the origin at its center and assume node b is at $(x = 1, y = 0)$, then $N^1_b = \frac{1}{2}(1 + x)(1 - |y|)$. To enforce continuity of N_a, N_b and partition of unity, the interpolation function of node a and b inside cell 1 are constructed as $N^1_a = H^1_a - \frac{1}{2}N^1_b$, $N^1_c = H^1_c - \frac{1}{2}N^1_b$. A similar strategy can be adopted in 3D, where a transition cell may have 8 to 26 nodes, depending on the number of T-junctions.

While this approach works fine for the case of Figure 3.6(a), it starts to break the non-negativity constraint in the case of Figure 3.6(b) where there exists another T-junction node d. Using the same

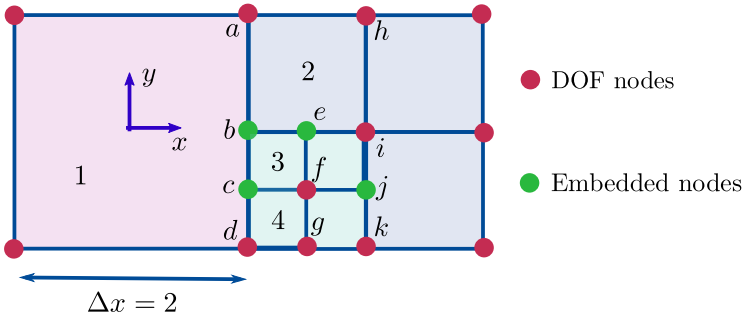


Figure 3.7: **Illustration for constrained T-junctions.** A test case for the our adaptive grid basis with constrained T-junctions, as described in Section 3.5.2.

local coordinate system defined above, we have

$$N_b^1 = \frac{1}{2}(1+x)(1-|y|), \qquad N_d^1 = \frac{1}{2}(1-|x|)(1+y), \tag{3.9}$$

$$H_c^1 = \frac{1}{4}(1+x)(1-y), \qquad N_c^1 = H_c^1 - \frac{1}{2}N_b^1 - \frac{1}{2}N_d^1. \tag{3.10}$$

It is easy to see that N_c^1 can sometimes become negative, for example when $x = 0.5$ and $y = 0.1$. As a result, some grid nodes may have negative mass after the particles-to-grid transfer. Negative weights in MPM causes instability and severe loss of accuracy [Andersen and Andersen, 2007]. For example, we could construct a case with two particles affecting a grid node with exactly opposite weights. As a result, the nodal mass becomes zero, resulting in an incorrect value of grid node velocity. Subsequently, the grid node velocity update and grid-to-particles transfer will be erroneous, causing the simulation to go unstable or behave non-physically. Figure 3.5 shows a practical case where such instability happens at a region where negative nodal mass exists.

3.5.2 Adaptive grid basis with constrained T-junctions

To prevent negative interpolation weights, we propose a different strategy for defining a C^0 continuous $N_i(\mathbf{x})$ for GIMP. The key idea is the same with the classical constrained hanging node treatment in octree FEM simulations [Legrain et al., 2011; Fries et al., 2011]. In contrast to the approach of Lian et al. [2015], our treatment does not construct interpolation functions on the T-junctions. T-junction nodes are constrained to move with their parent nodes at cell corners. Therefore, T-junction nodes are embedded vertices that do not belong to the set of real degrees of freedom.

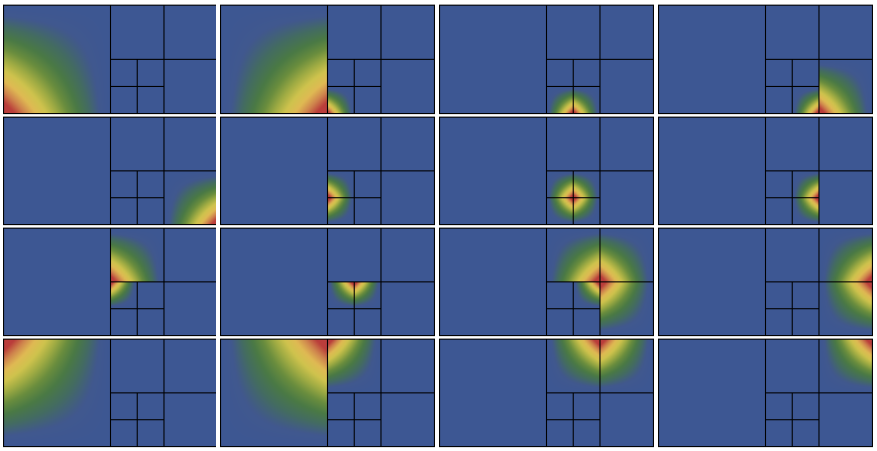


Figure 3.8: Illustration of the shape functions H_γ defined in Section 3.5.2.

We shall use the grid topology shown in Figure 3.7 to illustrate our approach; note that this arrangement includes a number of real degrees of freedom (colored red) as well as T-junctions (colored green). Initially, we will ignore any constraints that the T-junction nodes might be subjected to, and associate a shape function with all enumerated vertices, real or T-junction. For a given node γ we define a shape function $H_\gamma = \sum_n H_\gamma^n$ as the summation of all the standard bilinear hat functions H_γ^n, summed over all cells (indexed by n) for which γ is a *corner vertex* (not a T-junction). For example, we would thus have $H_a = H_a^1 + H_a^2$ and $H_b = H_b^2 + H_b^3$ in the scenario of Figure 3.7. The resulting functions H_γ are plotted in Figure 3.8. Using this basis (in the absence of any constraints), a scalar field $q(\mathbf{x})$ would be interpolated from nodal values as $q(\mathbf{x}) = \sum_\gamma q_\gamma H_\gamma(\mathbf{x})$. Of course, the reconstructed function $q(\mathbf{x})$ would be discontinuous (since the shape functions H_γ are discontinuous themselves, most notably along cell faces that contain T-junctions). Restoring continuity would require us to enforce constraints on the nodal values q_γ, such as $q_b = \frac{1}{2}q_a + \frac{1}{2}q_d$, $q_c = \frac{1}{4}q_a + \frac{3}{4}q_d$, $q_e = \frac{1}{4}q_a + \frac{1}{4}q_d + \frac{1}{2}q_i$ and $q_j = \frac{1}{2}q_i + \frac{1}{2}q_k$ in our specific example. Substituting these constraints into equation $q(\mathbf{x}) = \sum_\gamma q_\gamma H_\gamma(\mathbf{x})$ replaces this summation with a different expression $q(\mathbf{x}) = \sum_\eta q_\eta N_\eta(\mathbf{x})$, where now η enumerates only the *real* (non T-junction) degrees of freedom, and the new functions $N_\eta(\mathbf{x})$ have been formed by accumulating partial contributions from the constrained degrees of freedom that were eliminated after the substitution. For our specific example

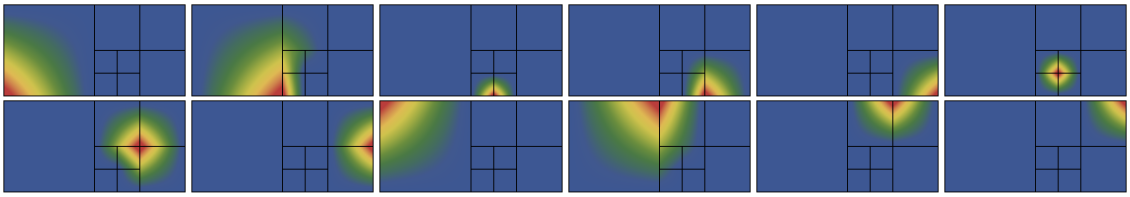


Figure 3.9: Visualization of our C^0 continuous shape function basis, defined in Section 3.5.2. Shape functions are only associated with non-T-junction nodes.

of Figure 3.7, we would have

$$N_a = H_a + \frac{1}{2}H_b + \frac{1}{4}H_c + \frac{1}{4}H_e, \tag{3.11}$$

$$N_d = H_d + \frac{3}{4}H_c + \frac{1}{2}H_b + \frac{1}{4}H_e, \tag{3.12}$$

$$N_i = H_i + \frac{1}{2}H_e + \frac{1}{2}H_j, \tag{3.13}$$

$$N_k = H_k + \frac{1}{2}H_j, \tag{3.14}$$

while the remaining non-T-junction nodes satisfy $N_\eta = H_\eta$. Note that the contribution from a constrained degree of freedom to a real degree of freedom (such as from H_c to N_a) directly results from the corresponding nodal value constraint (such as q_c). We use these newly defined functions $N_\eta(\mathbf{x})$ as our shape basis; these functions are fully C^0 continuous, as illustrated in Figure 3.9. We show a more detailed explanation of how one computes the shape for the basis functions and prove their properties (continuity and partition of unity) in the Appendix B.

We highlight the dual perspective : we can envision this construction either as an alteration of the basis functions of real (non T-junction) nodes, or as the result of constraining the coefficients of the shape functions H_γ associated with all nodes, to enforce embedding constraints; this duality will be exploited in the next section. Each modified interpolation function is a linear combination of hat basis functions. When we construct the GIMP shape functions and their gradients through Equation 3.4 and 3.6, the resulting w_{ip} is C^1 continuous, and ∇w_{ip} is C^0 continuous. In practice, instead of explicitly constructing the modified interpolation functions, we use a ghost cell based multi-layer scattering approach that is efficient and only require simple uniform grid operations (see Section 3.6.1).

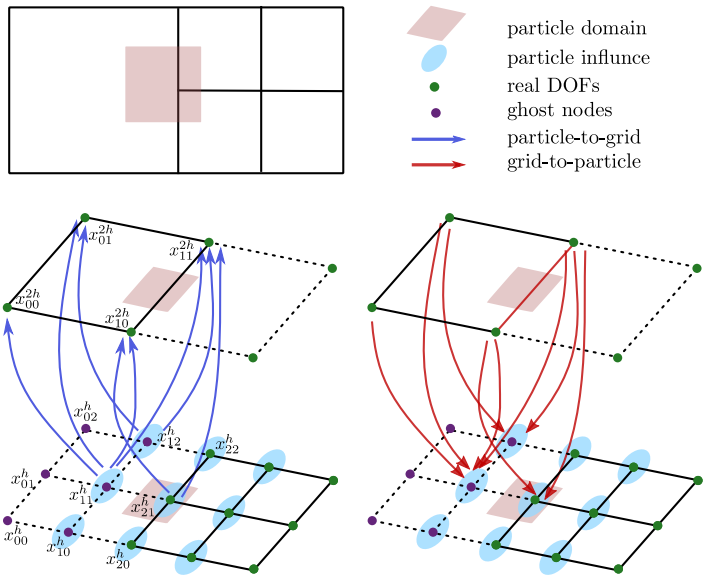


Figure 3.10: Particle-grid transfer operations mapped to a multi-level sparse grid structure; operations involving applications of weights are performed solely on a uniform grid, and information is propagated across resolutions via the embedding relation. The approach trivially extends to non-graded grids.

3.6 Accelerated grid-particle transfers

3.6.1 Multi-layer embedding and interpolation

In the previous section, we momentarily treated T-junctions as actual degrees of freedom, and endowed them with associated shape functions (H_γ). We will see that it is beneficial to take this exercise one step further. Consider the quadtree topology depicted in Figure 3.10. For the purposes of this hypothetical exercise, imagine that we fully subdivide the coarse cell on the left (with vertices $x^{2h}_{00}, \ldots, x^{2h}_{11}$) into four finer cells, straddling vertices now referred to as $x^{h}_{00}, \ldots, x^{h}_{22}$ (we are free to consider collocated vertices i.e. x^{2h}_{10} and x^{h}_{20} either as aliases of one another, or duplicates). Imagine that, instead of defining the starting shape functions H_γ on just the nodes that appear on the octree, we defined them on *all refined nodes* (i.e. define a function H^h_{ij} for every refined node x^h_{ij}). Refined nodes obey similar embedding relationships as previously seen for T-junctions, for example $x^h_{11} = \frac{1}{4}(x^{2h}_{00} + x^{2h}_{10} + x^{2h}_{01} + x^{2h}_{11})$. As in the previous section, we can construct the basis functions of the real degrees of freedom, by eliminating all refined nodes that are not collocated with a real node

in the quadtree; this would yield for example

$$N_{11}^{2h} = H_{22}^{h} + \frac{1}{2}H_{12}^{h} + \frac{1}{2}H_{21}^{h} + \frac{1}{4}H_{11}^{h}.$$

This linear relation could be leveraged to define multi-level weights via the GIMP convolution, resulting for example in an application of Equation 3.4 to an expression of the form:

$$w_{11,p}^{2h} = w_{22,p}^{h} + \frac{1}{2}w_{12,p}^{h} + \frac{1}{2}w_{21,p}^{h} + \frac{1}{4}w_{11,p}^{h} \tag{3.15}$$

Consider the repercussions of this construction: All weights of the form $w_{ij,p}^{h}$ exhibit high regularity, as they result from the integration of the standard bilinear hat function on a *uniform* grid; this is aggressively leveraged in the vectorization optimization discussed in the following subsection. Finally, consider what Equation 3.15 implies for particle-to-grid transfers such as the mass update $m_i = \sum_p m_p w_{ip}$. By substituting Equation 3.15 in this expression, we see that the mass contributions could be equivalently computed in a two-step process: we first compute partial contributions $m_{ij}^{2h} = \sum_p m_p w_{ij,p}^{2h}$, and then distribute these contributions to the real quadtree degrees of freedom by multiplying them with the respective embedding weights, as shown in the left part of Figure 3.10.

We facilitate this hierarchical transfer by introducing the concept of ghost nodes in Figure 3.10, which are similar in motivation with those formulated by Setaluri et al. [2014] to facilitate Laplacian stencil application, but different in the number of them that needs to be instanced. In the vicinity of any T-junction, we create duplicate refined variables by subdividing every cell incident to the T-junction down to the finest level touched by the T-junction. Among such nodes, those that are collocated with a real (non-constrained) quadtree node are labeled as "real" degrees of freedom; the remaining ones are labeled "ghost" and they serve as conduits for implementing the transfer operations in a hierarchical fashion. For any particle p, we detect the finest level intersecting the particle domain Ω_p, and carry out the particle-to-grid transfer exclusively to the refined nodes, some of which will be "ghost". After all particles have transferred their values to the appropriate uniform level, ghost node values are distributed in bulk to their coarser parents (multiplied by their embedding weights), until this process reaches all real quadtree nodes. The opposite process is followed for the grid-to-particle transfer; grid values of ghost nodes are interpolated from their

embedding "real" parents, and then the weighted transfer to the particle is evaluated exclusively at the finest level intersecting Ω_p, as shown in the bottom-right part of Figure 3.10.

3.6.2 Vectorized weight computation

Our hierarchical approach to grid-particle transfers provides two significant regularity properties that can be leveraged to accelerate the computation of transfer weights, which can form the bulk of the computational burden if implemented inefficiently, as they need to be updated at every time step. First, since the actual transfer (modulo the embedding-based distributions) always takes place at a uniform grid, there is always a full set of 8 grid cells/27 grid nodes (in 3D) that participate in this transfer. All eight of such cells reside at the same level of resolution, and their contribution to the weight stencils can be computed in parallel, via SIMD instructions.

In addition to the aforementioned property, our code exploits yet another opportunity for SIMD computation: Since the shape functions in all cells involved in our transfers have now been reduced to the standard trilinear basis, we can consider the possibility of computing the weights of all eight *corner nodes* of every cell in parallel, using SIMD instructions. We illustrate how this materializes for the weights themselves, although the process easily extends to the weight gradients as well. Absorbing the characteristic indicator function into the integral of Equation 3.4 results in

$$w_{ijk,p} = \frac{1}{\hat{V}_p} \int_{\Omega \cap \Omega_p} N_{ijk}(\mathbf{x})\, d\mathbf{x} = \frac{1}{\hat{V}_p} \int_a^b \int_c^d \int_e^f N_{ijk}(\mathbf{x})\, dxdyz$$

where we have explicitly used a triple index $(i, j, k) \in \{0, 1\}^3$ to refer to the eight vertices of a cell, and $\Omega \cap \Omega_p = [a, b] \times [c, d] \times [e, f]$. Since the integrand is separable, it can be easily computed in closed form, e.g. for $N_{111} = \frac{1}{dx^3} xyz$ we have

$$W(a, b, c, d, e, f) := \frac{1}{\hat{V}_p} \int_{\Omega \cap \omega_p} N_{111}(\mathbf{x}) dx = \frac{(b^2 - a^2)(d^2 - c^2)(f^2 - e^2)}{8 \hat{V}_p\, dx^3}$$

We can easily observe that the integral of the shape functions N_{ijk} other than the one given here, can be easily computed by performing a change of variable $x_i \leftarrow 1 - x_i$ for one or more of the coordinate

indices $i = 1, 2, 3$. This yields the following results:

$$\begin{aligned}
w_{000,p} &= W(1-b, 1-a, 1-d, 1-c, 1-f, 1-e) \\
w_{001,p} &= W(a, b, 1-d, 1-c, 1-f, 1-e) \\
w_{010,p} &= W(1-b, 1-a, c, d, 1-f, 1-e) \\
w_{011,p} &= W(a, b, c, d, 1-f, 1-e) \\
w_{100,p} &= W(1-b, 1-a, 1-d, 1-c, e, f), \quad \text{etc...}
\end{aligned}$$

We use this property, in conjunction with the regularity of computation across the eight cells involved, to implement a SIMD-optimized weight computation that computes the weights of all 8 cell vertices at once, by executing a vectorized implementation of the expression $W(\cdot)$ for a properly adjusted set of integration bounds.

The feasibility of these SIMD optimization, and the ability to structure the entire transfer using uniform operations as a building block, was a direct consequence of how our theoretical interpolation scheme was designed. The SIMD optimization would also directly benefit traditional GIMP on a uniform grid.

3.7 Grid rasterization and particle resampling

3.7.1 Grid rasterization

We index grid levels with $1 \leqslant q_g \leqslant Q_g$ and particle types with $1 \leqslant q_p \leqslant Q_p$, where lower case denote finer grid resolution/smaller particles. In pre-process, particles attributes (e.g. position, mass, volume, and type) are prescribed by the user. Our convention is to place coarser grid resolution (hence sparser particle distribution) deep inside, and finer grid resolution (hence denser particle distribution) closer to the free surface or collision boundary. Static grid adaptivity (e.g. Figure 3.1 and 3.2) is accomplished in a pre-process step. However, dynamic grid adaptivity (e.g. Figure 3.4, 3.3, and 3.12) needs to be computed for each time step.

For hyperelasticity, the grid and particle adaptations are completely determined by the particle types. Starting from the coarsest level, the smallest particle type q_p within a cell of the current grid

q_g is computed. We refine the cell if $q_p < q_g$ and the number of particles of each sub-cell is no less than particles per cell prescribed.

For elastoplastic material, we need to perform particle resampling. We first compute the approximate Manhattan distance d from the "free surface" for each finest cell. Then starting from the finest grid level, we merge sub-cells into a single cell only if they are beyond a prescribed distance criterion. However, this simple strategy is prone to the mismatch of particle type and grid level which can easily lead to numerical fracture Lian et al. [2015]. Another problem is that for highly energetic motion (c.f. top right corner of Figure 3.11(a)), sparse particles in the interior are easily driven to the surface, which diminishes surface details.

3.7.2 Particle resampling

To alleviate these problems, we modify the split-and-merge approach proposed by Yue et al. [2015] to better suit our framework. Split-and-merge is applied to a particle depending on both its particle type and the corresponding Manhattan distances of the cell containing the particle. We define three distance parameters d_{small}, d_{medium} and d_{large}, if $d < d_{small}$, neither split nor merge of particles is enforced, otherwise the operations will be visually noticeable. Split is necessary when $d_{small} < d < d_{medium}$ to retain a detailed surface while merge is required for $d > d_{large}$ to reduce computational cost. In the last case, $d_{medium} < d < d_{large}$, split-and-merge depends on the particular application. Moreover, split-and-merge is prohibited if the number of particles within a cell is either too large or too small which could end up with the sparse particles moving closer to the surface (cf. blue circle of Figure 3.11(b)). For applications with more than two levels, (e.g. Figure 3.1 and 3.2), we repeat the process between each level of transition.

Split To split a particle of type q to four particles of type $q-1$ in 2D (eight in 3D), we first put a square(or a cube in 3D) centered at the original particle position with half diagonal length $\frac{dx}{4}$ (dx is the size of the cell containing the particle), then randomly rotate the square/cube and place the new particles in the vertices of the square/cube. To preserve both mass and momentum, the mass and volume are equally distributed to all new particles while the velocity and the deformation gradient are directly duplicated.

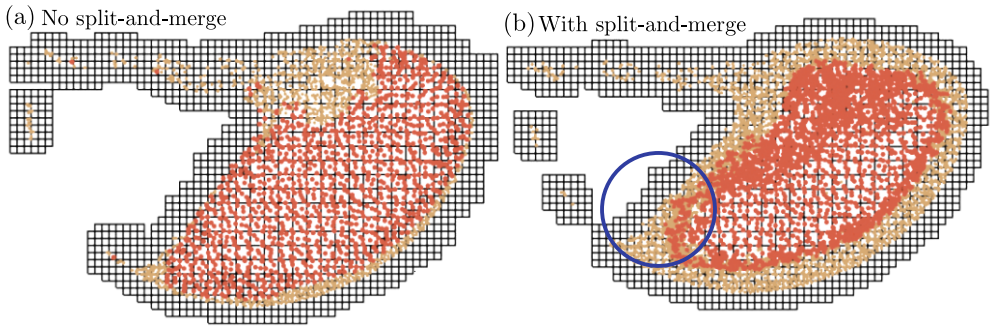


Figure 3.11: **Sand in a rotating circle.** Top: particle split-and-merge turned off. Bottom: particle split-and-merge turned on.

Merge To merge particles, we start with a single particle and then search for its closest neighbors of the same particle type. The position of the new particle is the geometric center of the fine ones. The mass and volume are accumulated while the velocity is computed from a mass-weighted average. For deformation gradient, we first do the singular value decomposition $\mathbf{F}^E = \mathbf{U}^E\mathbf{\Sigma}^E(\mathbf{V}^E)^T$. The matrix $\hat{\mathbf{\Sigma}}$ is computed from the average of the $\mathbf{\Sigma}^E$'s. We apply quaternion average to the $\mathbf{U}^E$'s and $\mathbf{V}^E$'s to get $\hat{\mathbf{U}}^E$ and $\hat{\mathbf{V}}^E$. Finally, we compute the new deformation gradient as $\hat{\mathbf{F}}^E = \hat{\mathbf{U}}^E\hat{\mathbf{\Sigma}}^E(\hat{\mathbf{V}}^E)^T$.

3.8 Results

We list the performance and simulation parameters of our 3D simulations in Table 4.2, in which we also compare against uniform grid simulations. The first two examples (Jello, Figure 3.4 and Hourglass, Figure 3.12) illustrate the speedup of the adaptive approach against uniform methods that yield comparable visual detail with e.g., [Klár et al., 2016]. We highlight that the somewhat modest speedup is due to the fact that we gratuitously refine around the entirety of the free surface, rather than localizing refinement on regions undergoing contact. The last two examples (Dragon, Figure 3.1 and Armadillo, Figure 3.2) show that, when spending comparable computation effort with uniform techniques (with similar particle counts), our adaptive simulation can resolve significantly higher detail. Note that uniform MPM cannot handle our finest Δx case within reasonable amount of time and memory usage. Figure 3.3 shows colliding adaptive jellos in 3D. For this test, the memory usage is 0.18GB for the adaptive case and 0.24GB for the uniform dense case. The cost of particle-grid transfer operations in relate to the cost of a whole time step is 50% and 70% for them respectively.

Figure 3.13 illustrates the fact that our adaptive simulation (left) produces a visually similar detailed dynamics of colliding jellos compared to a regular-dense-grid simulation (right).

Benchmark We compare our accelerated particle-grid transfers to reliable implementations of the MPM solvers in [Klár et al., 2016] (with a dense uniform grid data structure) and [Tampubolon et al., 2017] (with the OpenVDB [Museth et al., 2013] sparse grid) on an Intel(R) Core(TM) i7-4770R. We create a benchmark example with $\Delta x = \frac{1}{128}$. The particles are uniformed sampled on a grid from $(\frac{1}{8}, \frac{1}{8}, \frac{1}{8})$ to $(\frac{7}{8}, \frac{7}{8}, \frac{7}{8})$ with spacing $\frac{1}{256}$, for a total of just over 7 million particles. SPGrid (http://www.cs.wisc.edu/~sifakis/project_pages/SPGrid.html) was sourced from the project website, while the other MPM benchmarks were graciously shared with us by the respective authors. Table 3.1 lists the timing comparison.

Contact The benefit of using an adaptive discretization include better resolution of the contact between different MPM objects (as well as self contact). In Figure 3.4, we show a comparison of 2D colliding jellos on a uniform coarse grid, a uniform dense grid, and our adaptive grid. Adaptivity allows us to achieve very small separation distance. The adaptive discretization reduces the necessary computational cost compared to the uniform dense case.

Dynamic Adaptation We simulate dry sand with the Drucker-Prager plasticity model and the free surface based adaptation criterion (Figure 3.12). Both the particles and the grid are refined near the free surface and coarsened in the interior region. The computational resources are thus focused on the visible part, enabling highly detailed flow resolution.

Table 3.1: **Benchmark.** We compare our accelerated particle-to-grid (P2G) and grid-to-particle (G2P) transfers to a dense uniform grid MPM solver [Klár et al., 2016] and a sparse OpenVDB-based MPM solver [Tampubolon et al., 2017] on an Intel(R) Core(TM) i7-4770R.

	P2G 1 Core	G2P 1 Core	P2G 4 Cores	G2P 4 Cores
Dense	3.58	6.84	1.38	1.85
OpenVDB	2.39	5.8	0.64	1.54
SPGrid	3.05	1.26	0.82	0.28

Table 3.2: **Simulation performance and parameters.** (ρ is material density, E is Young's modulus, ν is Poisson's ratio and σ_y is plasticity yield stress.) (1) Jello and Hourglass are run with Intel(R) Xeon(R) CPU E5-2687W v3. We compare the finest level Δx, first frame particle count, and per-step cost of our adaptive scheme with the simulation on a uniform grid that has the same resolution with our finest grid level. (2) Dragon and Armadillo are run with Intel(R) Xeon(R) CPU E5-1650 v3. Sampling on a uniform grid using our finest grid Δx causes a total number of over 30 million particles and cannot be handled within reasonable amount of time. Therefore we adjust the uniform grid resolution to match the particle count of our adaptive simulation, while maintaining the same particle-per-cell count.

	levels (adaptive)	Δx (finest;adaptive)	particle # (adaptive)	time/step (adaptive)	Δx (uniform)	particle # (uniform)	time/step (uniform)	ρ	E	ν	σ_y	friction angle
Jello (Fig. 3.3)	3	1/256	8.8×10^5	1.67	1/256	1.74×10^6	2.74	1000	8e3	0.3	-	-
Hourglass (Fig. 3.12)	2	1/256	8.1×10^5	1.14	1/256	1.68×10^6	2.84	1000	1e5	0.3	-	30
Dragon (Fig. 3.1)	3	1/512	1.567×10^6	2.78	1/150	1.67×10^6	2.83	800	8e3	0.3	10	-
Armadillo (Fig. 3.2)	3	1/512	5.2×10^5	1.15	1/128	5.2×10^5	0.85	800	8e3	0.3	10	-

Small Features We further demonstrate the effectiveness of our method in resolving small scale collision object features. In Figure 3.1 we carve thin cracks on a glass table and put a dragon-shaped von Mises goo on it. We choose a small grid resolution for efficiency. A uniform grid cannot fully resolve the thin feature. Our method successfully let the goo to slip through the cracks by simply refining the cells near the crack. The robustness of our method on non-graded adaptivity allows us to refine the same cell for multiple times without needing to specially take care of neighboring cells. Note that the refinement is only performed to an extremely small portion of the computational domain and does not cause much overhead. Figure 3.2 shows another example where we cut an armadillo with two wires. Similarly, our method resolves the thin wire with a slight increase in computational cost. Although we adopt three successive levels of refinement in both of these two examples, all cells are either at the very coarsest or very finest level of resolution with a non-graded transition between them.

3.9 Discussion

Limitations Even though our adaptive scheme promises significant performance and detail benefits, it also carries a number of intrinsic limitations. Taking full advantage of adaptivity in simulations involving intricate contact would often require dynamically refining parts of the surface involved in collision events at any point in time (as opposed to preemptively refining the entire surface). In those cases, visual artifacts due to resampling might be evident (especially for simulation of granular

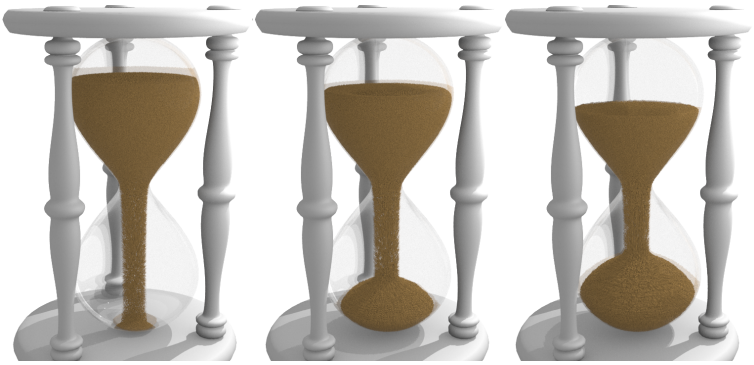


Figure 3.12: **Sand in an hourglass.** Sand in an hourglass is simulated with finer particles and grid resolution strategically placed near the collision boundary.

materials, e.g. sand) and would require special attention to mitigate. The GIMP convolution allowed C^0 multilinear bases to be boosted to C_1 continuity in the computed weights; however, seeking a higher order of continuity is highly nontrivial in the adaptive case, as it would likely require a basis that is C^1 pre-convolution. Our use of a hierarchy of sparse grids allowed us to contain costs associated with expansive uniform grids; nevertheless, tracking the set of particles as they transition across cells of different resolution requires geometric search structures (e.g. box hierarchies) whose traversal and update can be less parallel-friendly than what would be necessary for the uniform case.

Scope restrictions Our initial exploration of adaptive GIMP was consciously restricted in scope to simple simulations of elastoplasticity and refinement rules. In particular, refinement was triggered by simplified heuristics, such as the proximity to collision objects and/or the free surface. We have not investigated more intricate refinement criteria such as those triggered by large values of strain, or spatially localized to regions undergoing self-collision, for which an accurate and efficient detection would be less trivial. We have also restricted our investigation to purely explicit time integration techniques, and did not consider scenarios of fracturing elastic bodies, for which the use of adaptation would be naturally motivated.

Future work We look forward to investigating extensions to implicit MPM approaches and evaluate if the SPGrid paradigm can deliver similar accelerations to operator evaluations in assembly-free iterative solvers. Extensions to MPM fluids and coupling behaviors between solid/granular/fluid

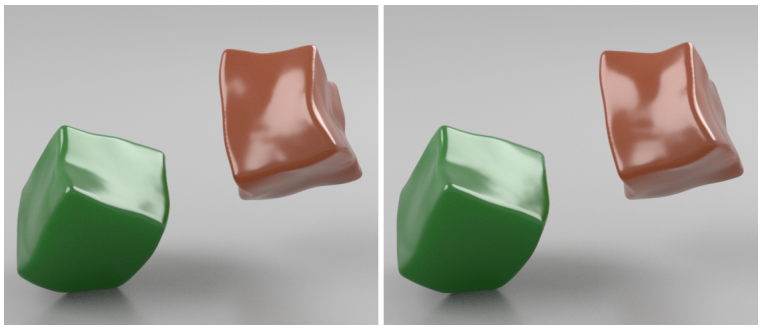


Figure 3.13: **Comparison.** The result of our adaptive formulation (left) compared with regular MPM on a dense grid (right).

phases would also be an exciting direction of investigation. We would also like to investigate the incorporation of the Affine Particle-In-Cell (APIC) method [Jiang et al., 2015] to our adaptive scheme for improved stability and angular momentum conservation. Finally, investigating combined use of adaptivity with fracture scenarios and detailed self-collision would be a very interesting (and likely nontrivial) future thread.

4 ANIMATING FLUID SEDIMENT MIXTURE IN PARTICLE-LADEN FLOWS

In this chapter, we present a mixed explicit and semi-implicit Material Point Method for simulating particle-laden flows. We develop a Multigrid Preconditioned fluid solver for the Locally Averaged Navier Stokes equation. This is discretized purely on a semi-staggered standard MPM grid. Sedimentation is modeled with the Drucker-Prager elastoplasticity flow rule, enhanced by a novel particle density estimation method for converting particles between representations of either continuum or discrete points. Fluid and sediment are two-way coupled through a momentum exchange force that can be easily resolved with two MPM background grids. We present various results to demonstrate the efficacy of our method.

4.1 Introduction

Recently, multi-phase multi-material simulations are increasingly gaining attention from computer graphics researchers. Simulating various phases or materials in a unified framework is particularly favored. Existing work includes coupled Lagrangian particle simulation with Position Based Dynamics (PBD) [Macklin et al., 2014], water-gas mixtures [Nielsen and Østerby, 2013] with an Eulerian method, solid-fluid phase-change [Stomakhin et al., 2014] and porous granular media [Tampubolon et al., 2017] with Material Point Method (MPM), as well as interactive solids and fluids based on the mixture model with Smoothed Particle Hydrodynamics (SPH) [Yan et al., 2016]. Most of the existing approaches are based on *continuum* mixture theory [Manninen et al., 1996]. The continuum assumption for each material phase is essential for simulations of macroscopic porous media (e.g., landslides and liquid blending). However, it may fail to capture the correct behavior of particle-laden flows where the solid phase is on a relatively small scale. Note that particle-laden sediment flow is ubiquitous in natural systems. Typical examples include sediment transport, sedimentation, volcano eruption, dune migration by erosion with ripples, and dust storms. The significance of understanding and simulating these phenomena is also recognized in many engineering applications, such as granular material fluidization [van der Hoef et al., 2006] and coastal erosion prediction [Sun and

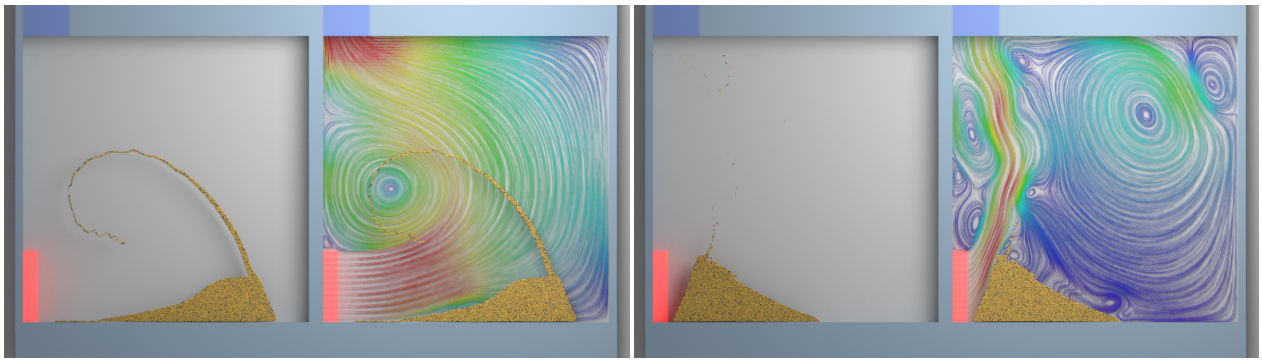


Figure 4.1: **Dune migration:** Two-dimensional simulations of wind blowing sand. *Left:* A one-way coupled simulation where sand does not affect wind fails to produce plausible dynamics. *Right:* Two-way coupled simulation from our method captures the characteristic behavior of a sand dune migrating forward.

Xiao, 2016a].

Our method treats fluid as a continuum and solid sediment as being comprised of small immiscible particles. The majority of engineering literature discretizes these sedimentation problems based on a combination of an Eulerian fluid solver and a Discrete Element Method (DEM) particle solver, which requires velocity interpolation between the two different discretizations to handle the momentum exchange between them. DEM is also too expensive for practical animation in collision-heavy dense flow scenarios. Our framework employs the MPM discretization for both the fluid phase and the solid phase to mitigate these problems.

In standard DEM, each particle is associated with a rigid sphere, and thus trivially represents a discrete spherical body with translational and rotational motions. Even for a collection of many particles, DEM does not make any continuum assumption. In contrast, each MPM particle represents a patch of the continuum and serves as a quadrature point for the spatial discretization of the continuous fields. Furthermore, traditional MPM cannot preserve angular momentum. In this chapter, we tackle both difficulties. We use the Affine Particle-In-Cell (APIC) [Jiang et al., 2015] method to grant each particle an affine velocity field which enables the representation of local rigid rotations. Second, we propose a density criterion to allow certain MPM particles to behave as discrete debris separated from the continuum body. As a result, each MPM particle in the sediment can essentially act as a separate spherical body, while the continuum elastoplastic relationship is only activated for clumps of them.

In contrast to the recent work on wet sand by Tampubolon et al. [2017], which relied on an intricate modification to the sand elastoplasticity model and is limited in the use of weakly compressible fluids, our work does not require an ad-hoc modification to the sand constitutive model and supports an incompressible fluid solver. Unlike the augmented MPM scheme by Stomakhin et al. [2014], our incompressible fluid solver discretizes velocity degree of freedoms on a semi-staggered grid, thus eliminating the computationally costly MPM discretization on a MAC grid which requires the use of cubic kernels. Xiao and Sun [2011] used constant interpolation to approximate the fluid velocities at solid particle positions, introducing dissipation; instead, our framework circumvents such interpolation.

We summarize our contributions as follows:

- A unified solver for purely incompressible fluids and particle-laden-flows using MPM background grids.
- A sub-stepping scheme with accurate momentum exchange for the mixture of solid particles and fluids, allowing for a larger time-step in the fluid solver.
- A density evaluation strategy for converting particles between continuum clumps and discrete debris.
- A semi-staggered semi-implicit MPM discretization for Locally Averaged Navier-Stokes with a variable coefficient multigrid preconditioner.

We demonstrate the efficacy of our method with various sediment-fluid interaction examples.

4.2 Related work

While earlier work in graphics was centered on certain types of physical materials or phenomena, more recent attention has been focused on the simulation of multi-phase multi-material interactions. Teng et al. [2016] simulate two-way coupled deformable solids and incompressible flow. Nielsen and Østerby [2013] used a two-continua mixture model for simulating water droplets in the air, discretized in an Eulerian fashion. Macklin et al. [2014] proposed a unified Lagrangian particle

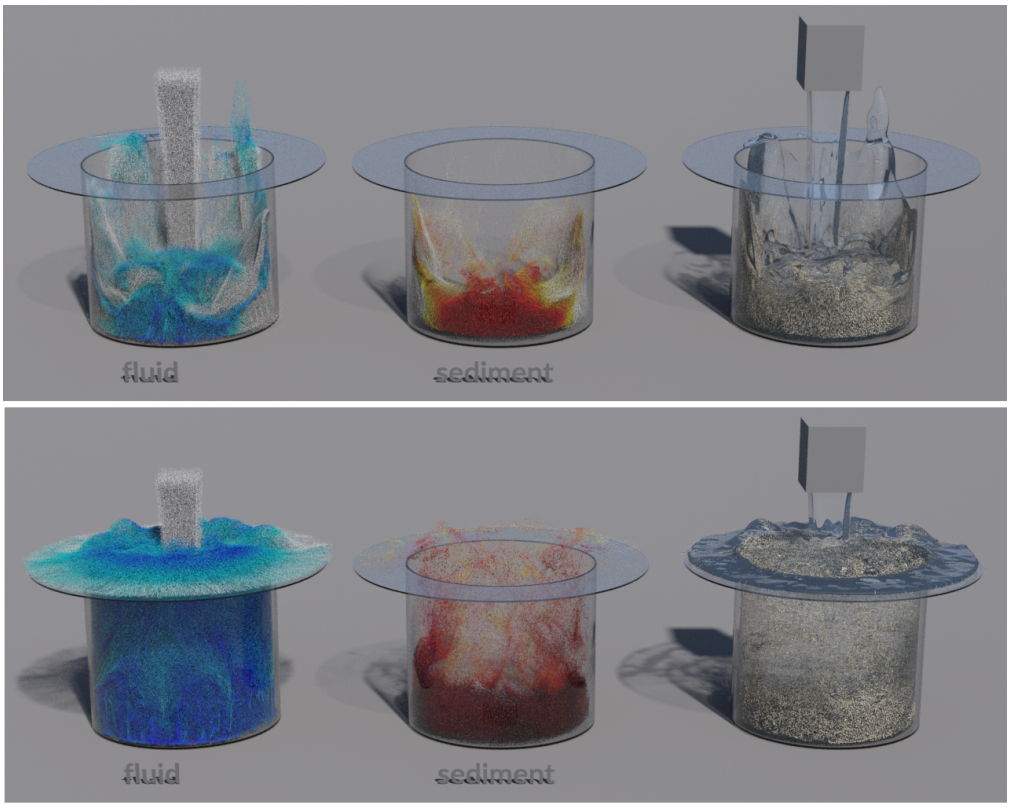


Figure 4.2: **Glass flush:** Water is poured onto sediment in a glass. The overflow of water transports the sediment out of the glass.

based solid-liquid-gas coupling framework with PBD [Müller et al., 2007]. Müller et al. [2005] were the first to simulate fluid-fluid interaction in SPH. Peer et al. [2015] simulated multiphase viscous SPH fluid with pressure and cohesion forces for their interaction. Ren et al. [2014] simulated both miscible and immiscible fluids with a mixture model. Yang et al. [2015] further took a Helmholtz Free Energy based model to achieve improved performance. These SPH frameworks were extended in Yan et al. [2016] to solids and granular materials, and also by Yang et al. [2017] to a variety of fluid-solid interactions. Daviet simulated granular flow in Newtonian fluid Daviet [2016] in two dimensions.

The existance of solid-fluid mixture is also common in porous flow and solid wetting. Lenaerts et al. [2008] simulated fluids in porous deformable media with an Eulerian discretization of Darcy flux. Their approach was extended to mixing fluids and granular materials in [Lenaerts and Dutré, 2009]. Rungjiratananon et al. [2008] simulated wet sand with coupled SPH and DEM. They also

captured capillary flow in anisotropic permeable hair in [Rungjiratananon et al., 2012]. Fei et al. [2017] adopted a multi-scale model for wet hair, where they used Affine Particle-In-Cell (APIC) [Jiang et al., 2015] for water and a discrete rod model [Bergou et al., 2008] for the solid. Liquid is also simulated with APIC in our framework, although we will discretize it on a semi-staggered grid.

In computer graphics, Wojtan et al. [2007] simulated erosion, sedimentation, and corrosion with Eulerian fluids and level set solids. Krištof et al. [2009] used an SPH-based fluid scheme to model erosion and avoided the direct simulation of sediments by introducing the boundary particles to represent the sediment exchange with the terrain.

Our method is based on the Material Point Method (MPM) [Sulsky et al., 1995], which has recently been popularized in computer graphics for simulating granular materials [Stomakhin et al., 2013; Klár et al., 2016; Daviet and Bertails-Descoubes, 2016], viscoplasticity [Ram et al., 2015; Yue et al., 2015] and cloth [Jiang et al., 2017a]. MPM was also used for simulating solid-incompressible fluid coupling in [Stomakhin et al., 2014] and porous sand-water mixture in [Tampubolon et al., 2017]. We compare the feature sets of these methods in Table 4.1.

There is a large body of related work in Engineering literature on simulating particle-laden flows. One of the standard approaches is to couple mesh-based Eulerian fluids with particle systems described with the discrete element method (DEM). The pioneering work of Xu and Yu [1997] modeled gas-solid flow in a fluidized bed by combining the Discrete Particle Method (DPM) and standard Computational Fluid Dynamics (CFD) solvers. Li et al. [1999] extended it to gas–liquid–solid systems. Xiao and Sun [2011] applied a similar idea to general particle-laden flows and recently released a general-purpose, open-source CFD-DEM coupling solver called SediFoam [Sun and Xiao, 2016b]. They treated the fluid-particle interaction term implicitly and this resulted in a highly

Table 4.1: Comparing Augmented MPM (AMPM) [Stomakhin et al., 2014], Multi-species MPM (MMPM) [Tampubolon et al., 2017] and our method for simulating the mixture of granular solid particles and fluids.

	AMPM	MMPM	Ours
Reducible to incompressible flow (fluid)	✓	✗	✓
Unmodified hyperelasticity (solid)	✓	✗	✓
Discrete debris support (solid)	✗	✗	✓
Saturation dependent cohesion (solid)	✗	✓	✗
Multi-phase miscibility	✗	✓	✓
Mixed integration	✗	✗	✓

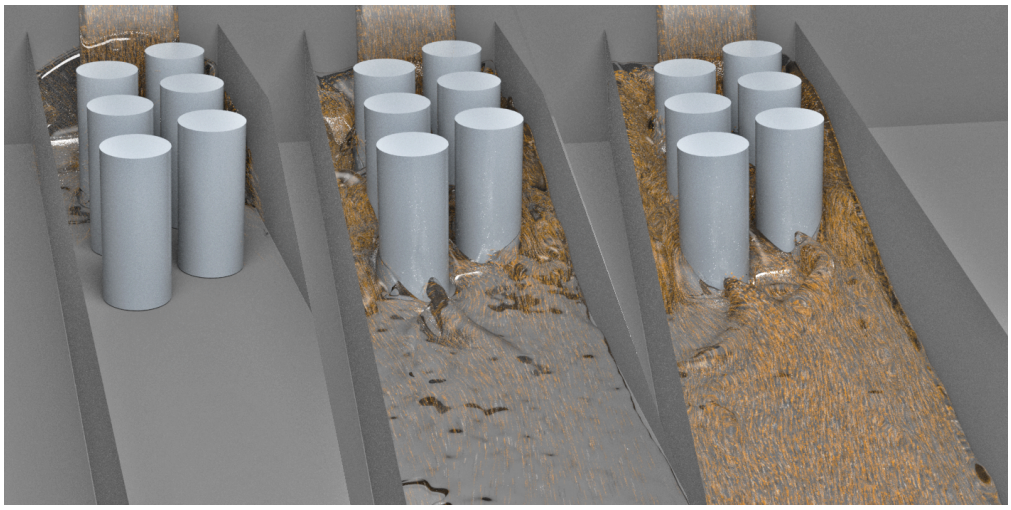


Figure 4.3: **Debris flow:** We simulate water-saturated debris flow, which is a common occurrence of particle-laden flows in nature.

stable integration scheme. Snider [2001] considered dense particle fluid flows with multiphase Particle-In-Cell (MPIC). Fluid-particle interaction was also achieved with coupled SPH and DEM by Robinson and Ramaioli [2011]. More recently Mutabaruka and Kamrin [2017] simulated particle-laden dynamics with lattice Boltzmann fluids and DEM.

Narain et al. [2010] and Daviet and Bertails-Descoubes [2016] investigated free-flowing granular media as disperse grains in low density regions. Our strategy for treating debris sediment is similar to the approach by Dunatunga and Kamrin [2015]. Their approach associates a unilateral pressure with each particle by comparing their density with a threshold and enforcing a *disconnected* stress-free state when the density is low. The main difference is that they evolve material point volume using velocity gradient over time, while we always evaluate the current particle "clumpiness" based on world space locations which avoids accumulated numerical advection error in highly dynamic scenarios.

4.3 Continuous equations

Here we describe the governing equations for both fluid and sediment materials in particle-laden flows. We will use superscript f for fluid and s for sediment. Fluid and sediment velocities are denoted with $\mathbf{u}$ and $\mathbf{v}$ respectively.

4.3.1 Fluid

In particle-laden flows, the fluid has to satisfy the impermeable boundary conditions enforced by discrete particles. However, it is impossible to exactly resolve these boundaries in the presence of a large number of sediment particles, unless an extremely high fluid resolution is used. We instead incorporate the Locally Averaged incompressible Navier-Stokes theory proposed by Anderson and Jackson [1967] to model the fluid motion. As in other previous work on multiphase simulation, each spatial point is simultaneously occupied by different phases or materials. Note that the use of Locally Averaged Navier-Stokes in our work essentially shares the same form for describing the fluid with the mixture model used by Nielsen and Østerby [2013]. Specifically, the mass equation for fluid is given by

$$\frac{\partial(\epsilon\rho^f)}{\partial t} + \nabla\cdot(\epsilon\rho^f\mathbf{u}) = 0, \tag{4.1}$$

and the momentum equation is

$$\frac{\partial(\epsilon\rho^f\mathbf{u})}{\partial t} + \nabla\cdot(\epsilon\rho^f\mathbf{u}\otimes\mathbf{u}) = -\epsilon\nabla p + \epsilon\rho^f\mathbf{g} + \mathbf{f}^{fd}, \tag{4.2}$$

where ϵ, ρ^f, $\mathbf{u}$, p, $\mathbf{g}$, and $\mathbf{f}^{fd}$ are the fluid volume fraction, fluid *intrinsic* density, fluid velocity, pressure, the gravitational constant, and the fluid drag force density respectively. We use the superscript fd, sd and se for fluid drag force, sediment drag force and sediment elastic force respectively. Note that $\mathbf{f}^{fd}$ and $\mathbf{f}^{sd}$ denote the interaction drag *force of the sediment on the fluid* and *force of the fluid on the sediment*. As such, they obey $\mathbf{f}^{fd} = -\mathbf{f}^{sd}$. Combining Eq. (4.1) and Eq. (4.2), we arrive at

$$\frac{\partial\mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} = -\frac{1}{\rho^f}\nabla p + \mathbf{g} + \frac{1}{\epsilon\rho^f}\mathbf{f}^{fd}. \tag{4.3}$$

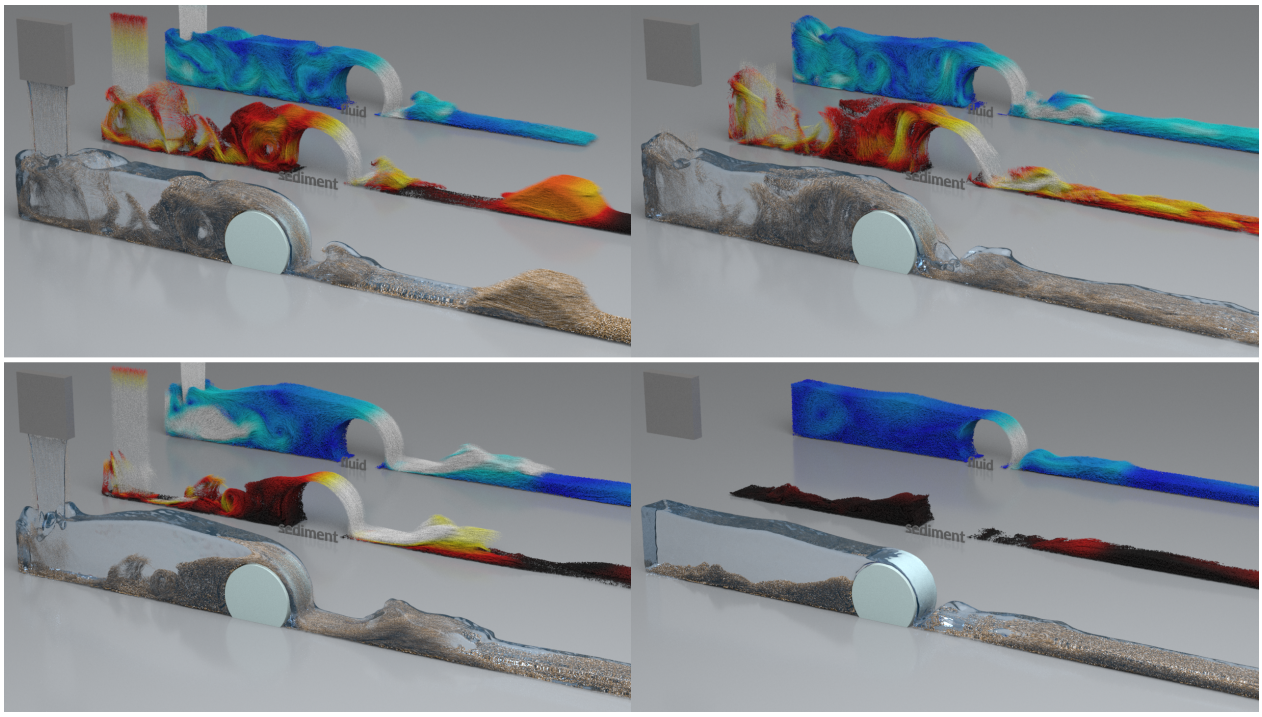


Figure 4.4: **Hydraulic jump and sediment transport:** A mixture of water and sediment being poured into a thin channel with a cylindrical obstacle in the middle. The top figure uses an RPIC ratio of 0.2 (and a smaller gravity value), creating a more turbulent flow, while the bottom figure uses an RPIC ratio of 0.4, creating a more laminar flow.

4.3.2 Sediment

In the scale of particle-laden flows, granular solid materials are usually treated as discrete spherical bodies. This perspective eliminates the need to formulate governing equations for the volume fraction scaled solid material or for the entire mixture as in [Yan et al., 2016]. Normally in a CFD-DEM coupling framework such as [Sun and Xiao, 2016b], Newton's second law would be directly enforced on the discrete spherical particles to describe their translational and rotational motions. However, in order to support the massive amount of dense particle suspensions, it is preferable to formulate the sediment governing equations through continuum mechanics, with the extra drag force density term $\mathbf{f}^{sd}$:

$$\frac{\partial(\delta\rho^s)}{\partial t} + \nabla\cdot(\delta\rho^s\mathbf{v}) = 0, \tag{4.4}$$

$$\frac{\partial(\delta\rho^s\mathbf{v})}{\partial t} + \nabla\cdot(\delta\rho^s\mathbf{v}\otimes\mathbf{v}) = \delta\nabla\cdot\boldsymbol{\sigma}^s + \delta\rho^s\mathbf{g} + \mathbf{f}^{sd}, \tag{4.5}$$

where δ, ρ^s, $\mathbf{v}^s$, and $\boldsymbol{\sigma}^s$ denote sediment volume fraction, sediment *intrinsic* density, sediment velocity, and the Cauchy stress describing the mechanical responses inside solid clumps. From mixture theory, the volume fraction of fluid and sediment obeys $\epsilon + \delta = 1$. In the case of debris in the air without liquid, we have $\delta = 1$ and $\epsilon = 0$.

As previously mentioned, we treat a clump of sediment particles as an elastoplastic material which obeys Drucker-Prager plasticity. Our density approximation method provides a measure of the spatial distribution of sediment particles. As a sediment particle is separated from its neighbor, it is no longer subject to the elastoplastic constitutive law. We discuss the strategy we adopt for this density evaluation in more detail in Sec. (§4.4.5).

4.3.3 Mixture incompressibility and interaction term

From the assumption that $\epsilon + \delta = 1$, we can combine Eq. (4.1) and Eq.(4.4) to get

$$\nabla \cdot (\epsilon \mathbf{u} + \delta \mathbf{v}) = 0. \tag{4.6}$$

We would like to note that in this work, the interaction terms between the fluid and the sediment are limited to drag forces ($\mathbf{f}^{sd}$ and $\mathbf{f}^{fd}$), which is a point of departure from [Anderson and Jackson, 1967]. The exact model used is described in Sec. (§4.4.3). This model does not include the effect of buoyancy (necessary for capturing liquefication phenomena) and lift force. A caveat is that to conserve momentum, we only apply the drag force model to the sediment whereas in the fluid step the drag force is computed from the exchanged momentum accumulated during the sediment step.

4.4 Method

We discretize the velocity degrees of freedom and the forces of each constituent with the help of an MPM grid with grid-spacing Δx. We will use subscript i for quantities stored at grid node i. We take a semi-staggered discretization similar to [Zhang et al., 2017] for the fluid momentum equation. Once we commit to simulating sand using MPM, it is natural to simulate fluid on a semi-staggered grid since both fluid and sediment will be discretized on colocated nodes. Another advantage of

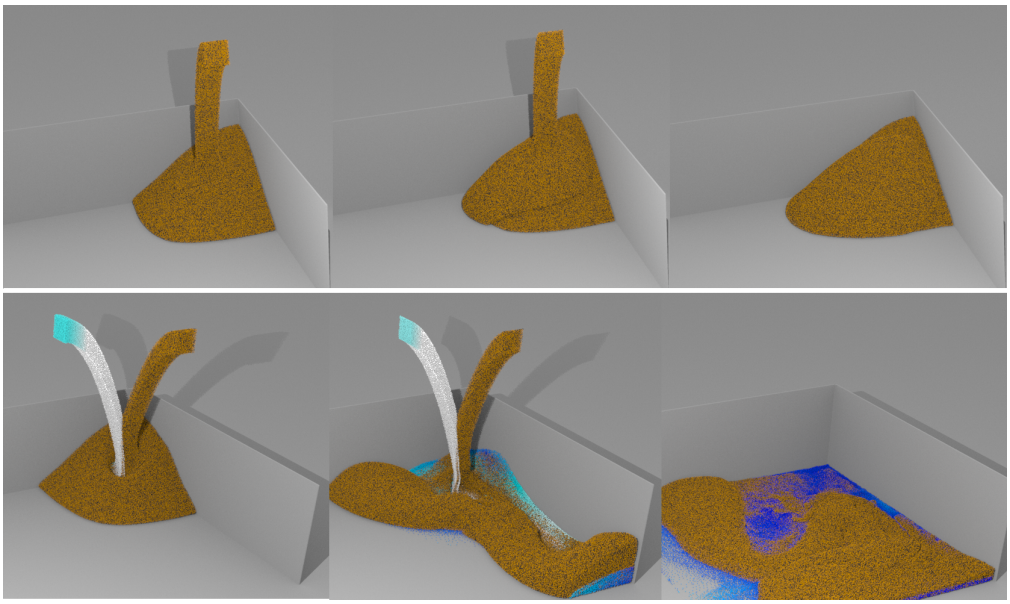


Figure 4.5: **Pile:** *Top:* Pure sediment particles form a pile with static friction. *Bottom:* By injecting liquid, the pile collapses and interacts with the fluid.

this is to avoid the need to extrapolate velocity on the fluid surface, which is needed with FLIP. As a result, pressure and velocity divergence are stored at cell centers with subscript c. We use subscript f to denote variables on the faces. Both fluid and sediment materials are tracked and advected with MPM particles, which will be denoted with subscript p. The physical dimension is denoted by $d \in \{2, 3\}$.

4.4.1 Algorithm overview

We use a semi-implicit discretization for the fluid step and a symplectic Euler time integration for the sediment step. The fluid can thus take a larger time step than the sediment. We use $\Delta t^{f,n}$ to denote a fluid time step from t^n to $t^{n+1} =: t^n + \Delta t^{f,n}$. We divide one sediment timestep into K substeps, i.e. $\sum_{k=1}^{K} \Delta t^{s,n_k} = \Delta t^{f,n}$. Note that both time steps are chosen to satisfy the CFL condition [Stomakhin et al., 2013]. We summarize the essential steps in our method below and provide an illustration in Fig. 4.6.

First we describe a fluid step from t^n to t^{n+1}. We use APIC [Jiang et al., 2015] with a multi-linear kernel for particle-grid transfers.

1. **Particles to grid:** Transfer fluid particle mass m_p^f and momentum $(m\mathbf{u})_p^n$ to the grid and divide the mass, giving nodal mass $m_i^{f,n}$ and velocity $\mathbf{u}_i^n$.

2. **Apply forces:** Add *explicit* drag force and gravity to compute the intermediate velocities $\mathbf{u}_i^* \leftarrow \mathbf{u}_i^n$ on the background grid.

3. **Pressure projection:** Solve mixture pressure projection with proper boundary conditions to correct the nodal velocities $\hat{\mathbf{u}}_i^{n+1} \leftarrow \mathbf{u}_i^*$ so that Eq. (4.9) is satisfied.

4. **Grid to particles:** Particles get velocities $\mathbf{u}_p^{n+1}$ (and APIC coefficients) by interpolating from $\hat{\mathbf{u}}_i^{n+1}$. Advection is done as $\mathbf{x}_p^{f,n+1} = \mathbf{x}_p^{f,n} + \Delta t^{f,n}\mathbf{u}_p^{n+1}$.

Once we are done with the fluid step, we then integrate the sediment governing equations from t^n to t^{n+1} (equivalently $t^{n_0}, ..., t^{n_K}$) with symplectic Euler. Multiple sub-steps are usually required since sediment (which is similar to dry sand) has a considerably high Young's modulus which imposes a strong time step restriction. We use APIC with the quadratic B-spline kernel which is optimized, as in [Gao et al., 2017], for particle-grid transfers. We use the notation $N_i(\mathbf{x}_p^{s,n_k})$ as the evaluation of the kernel centered at grid node i at position $\mathbf{x}_p^{s,n_k}$. In each sub-step from t^{n_k} to $t^{n_{k+1}}$, the procedures are

1. **Particles to grid:** Transfer sediment particle mass m_p^s and momentum $(m\mathbf{v})_p^{n_k}$ to the grid and then average to get nodal mass m_i^{s,n_k} and velocity $\mathbf{v}_i^{n_k}$. We also approximate mass gradient on grid nodes with $(\nabla m^{s,n_k})(\mathbf{x}_i)$. Nodal sediment volume fraction $\delta_i^{n_k}$ is also updated (§4.4.5).

2. **Apply forces:** We compute the elastic force, fluid drag force, and gravity on the grid and update grid velocities based on these forces, i.e. $\hat{\mathbf{v}}_i^{n_{k+1}} \leftarrow \mathbf{v}_i^{n_k}$. Note that the drag forces conserve total momentum of the fluid-sediment system. We show more details in Sec. (§4.4.3). Next, we apply boundary conditions according to [Stomakhin et al., 2013], i.e. $\mathbf{v}_i^{n_{k+1}} \leftarrow \hat{\mathbf{v}}_i^{n_{k+1}}$ (§4.4.5).

3. **Update particles' states:** Particles' velocities and positions are updated by interpolation from the grid (§4.4.5).

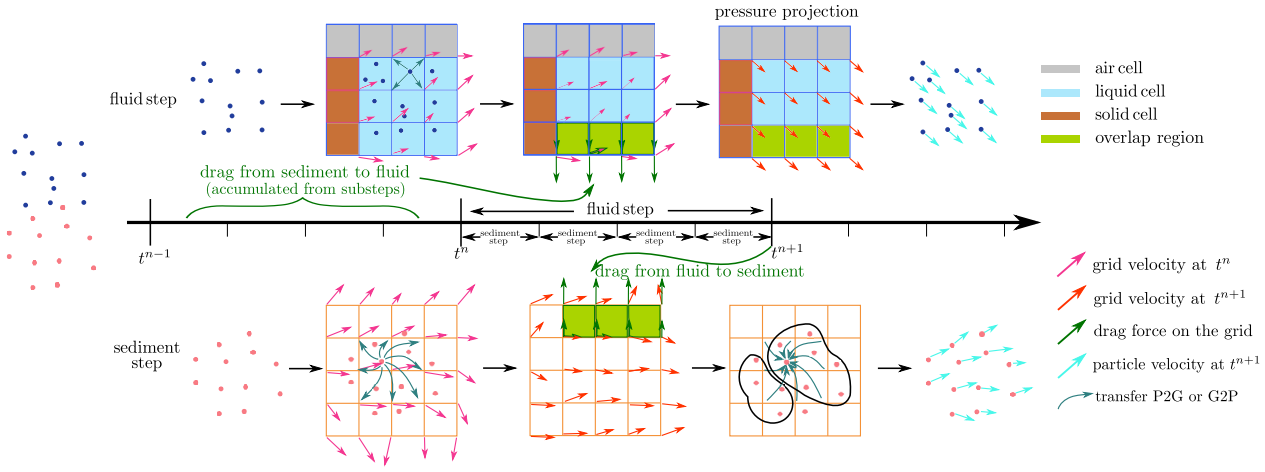


Figure 4.6: **Algorithm step:** The top row illustrates the fluid step, where we perform pressure projection to solve for the new velocity on the grid. The result of this computation is then used in the sedimentation step, illustrated in the bottom row. Note that for a single fluid step, there are multiple (explicit) MPM sediment sub-steps. The drag forces from each of the sediment sub-steps are accumulated to compute $\mathbf{f}^{fd}$ for the fluid step.

4. **Strain and representation update:** Particle deformation gradients $\mathbf{F}_p^{n_{k+1}}$ are updated using the Drucker-Prager plastic flow return mapping [Klár et al., 2016] with volume correction as proposed by [Tampubolon et al., 2017]. We also evaluate the world space sediment particle density $\rho_p^{s,n_{k+1}}$ to determine whether a particle should be treated as being part of the continuum or as a discrete debris (§4.4.5).

4.4.2 Volume fraction

Sediment particles are extremely stiff elastically, and conserve volume during the non-associative Drucker-Prager plastic flow [Klár et al., 2016]. In contrast to traditional MPM algorithms, we assume that the volume of each particle almost remains unchanged during the simulation ($V_p^{s,n_k} = V_p^{s,0}$ where $V_p^{s,0}$ is the original volume). Volume fraction of sediment is then given at each node as :

$$\delta_i^{n_k} = \min\left\{\delta_{max}, \frac{V_i^{s,n_k}}{\Delta x^d}\right\}, \tag{4.7}$$

where $V_i^{s,n_k} = \sum_p V_p^{s,n_k} N_i(\mathbf{x}_p^{s,n_k})$. The value of δ_{max} is chosen for numerical stability ($\delta_{max} \in [0.5, 0.8]$ in all of our examples). The corresponding nodal fluid volume fraction is then $\epsilon_i^n = 1 - \delta_i^{n_0} \in$

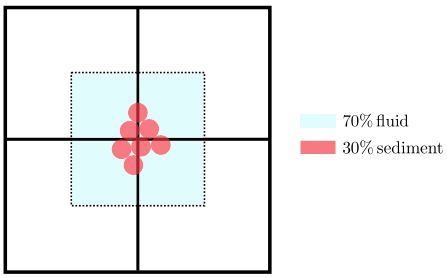


Figure 4.7: **Volume fraction:** In this point of view, water cannot enter into sediment. The node at the center represents 70% fluid and 30% sediment.

$[1-\delta_{max}, 1]$. Note that $\epsilon_i^n = 1$ if $m_i^{s,n_0} = 0$. In the absence of sediment particles, our locally averaged fluid discretization (§4.4.4) reduces to the standard incompressible flow discretization from [Zhang et al., 2017].

4.4.3 Drag force

On sediment

In contrast to CFD-DEM approaches where particle-wise drag forces are evaluated by interpolating fluid velocities to sediment locations, MPM allows us to directly apply drag forces to grid nodes. Corresponding to the drag force density $\mathbf{f}^{sd}$ in the continuum equation, we use a drag force model proposed by Di Felice [1994] in a discrete setting. On grid nodes where fluid and sediment materials are present we compute

$$\mathbf{f}_i^{sd,n_k} = \frac{1}{2} c_i (\epsilon_i^n)^{-\chi} \rho^f A_i^{s,n_k} |\mathbf{u}_i^{n+1} - \mathbf{v}_i^{n_k}| (\mathbf{u}_i^{n+1} - \mathbf{v}_i^{n_k}), \tag{4.8}$$

with c_i and χ denoting the empirical coefficient and exponent respectively. For high Reynolds number fluids, $\chi = 3.7$ and $c_i = 0.39$. The term A_i^{s,n_k} is the cross-sectional area of an imaginary spherical geometry, given by $A_i^{s,n_k} = 2\left(V_i^{s,n_k}/\pi\right)^{1/2}$ in 2D and $A_i^{s,n_k} = \pi\left(3V_i^{s,n_k}/(4\pi)\right)^{2/3}$ in 3D. This term guarantees nodes with tiny mass will receive a tiny force, thus avoiding potential numerical instabilities. In practice we adjust c_i for controlling the overall strength of the drag force. This drag force model is more suitable for particle laden flows and differs from the model used by Tampubolon et al. [2017].

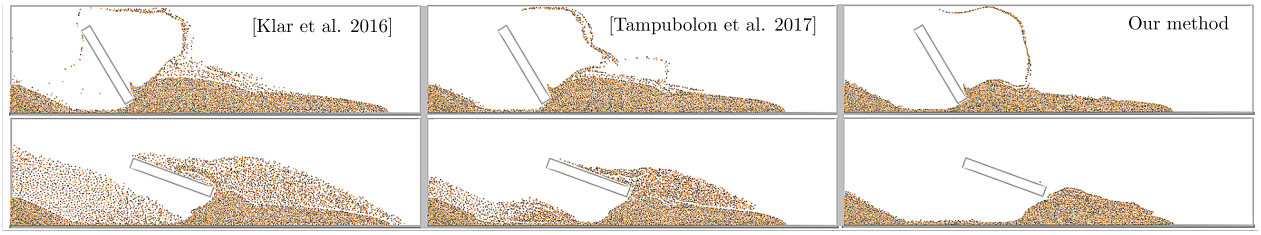


Figure 4.8: **Volume gain problem:** Previous MPM granular methods can introduce volume gain for highly dynamic scenarios. Tampubolon et al. [2017] proposed a method to mitigate this volume gain effect to a certain extent. Our method successfully resolves this problem.

On fluid

As we proceed from t^{n_0} to t^{n_K}, the total momentum change on the sediment is $\Delta \mathbf{P}_i^{sd,n} = \sum_{k=1}^{K} \mathbf{f}_i^{sd,n_k} \Delta t^{s,n_k}$. To enforce total momentum conservation in the fluid-sediment system, we set $\mathbf{f}_i^{fd,n} = -\Delta \mathbf{P}_i^{sd,n-1} / \Delta t^{f,n}$ when we apply the drag force to fluid. As a result $\Delta \mathbf{P}_i^{fd,n} = -\Delta \mathbf{P}_i^{sd,n-1}$ is always satisfied, i.e. any momentum change to the sediment caused by the drag force is always negated and applied to the fluid in the upcoming fluid step.

4.4.4 Fluid

Splitting

We apply the standard splitting treatment of fluid simulation (cf. [Bridson, 2008] for details) to Eq. (4.3) to get separate steps, which includes advection, application of external forces, pressure projection and velocity correction. Here we focus on the grid solve. Particle-grid transfer steps are described in Sec. (§4.4.1).

Advection In hybrid particle-grid fluid solvers, the nodal velocities are transferred from fluid particles, which have been advected at the end of the previous time step through particle advection. In other words, there is no additional advection required on the grid in this new cycle.

External forces This step includes the application of both drag forces from sediment particles and gravity to get the intermediate velocities $\mathbf{u}_i^* = \mathbf{u}_i^n + \left(\frac{\mathbf{f}_i^{fd,n}}{m_i^{f,n}} + \mathbf{g} \right) \Delta t$. Note that our method is explicit on the drag force and uses $\mathbf{f}_i^{fd,n}$ directly.

Pressure projection We adopt a semi-implicit discretization to rewrite Eq. (4.6) as

$$\nabla \cdot (\epsilon_i^n \mathbf{u}_i^{n+1}) = -\nabla \cdot (\delta_i^{n_0} \mathbf{v}_i^n), \tag{4.9}$$

where the divergence of volume fraction weighted sediment velocities is treated explicitly. This allows us to combine it with

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{1}{\rho} \nabla p = 0$$

to derive a variable coefficient Poisson's equation for the fluid pressure, namely

$$-\frac{\Delta t}{\rho^f} \nabla \cdot (\epsilon_f^n \nabla p) = -\nabla \cdot (\delta_i^{n_0} \mathbf{v}_i^n) - \nabla \cdot (\epsilon_i^n \mathbf{u}_i^*), \tag{4.10}$$

where both ϵ_f^n and ∇p are discretized at cell faces. Considering that ϵ_f^n is always positive, Eq. (4.10) corresponds to a symmetric positive (semi-)definite system.

Velocity correction After the pressure values are solved at cell centers, they can be used to correct fluid velocities as

$$\frac{\hat{\mathbf{u}}_i^{n+1} - \mathbf{u}_i^*}{\Delta t} = -\frac{1}{\rho^f} \nabla p, \tag{4.11}$$

where $\hat{\mathbf{u}}_i^{n+1}$ is further transferred back to particles using APIC (§4.4.1).

A Semi-staggered Discretization

As discussed in Sec. (§5.1), constructing an incompressible solid-fluid coupling solver on a fully staggered MAC grid as done by Stomakhin et al. [2014] would require multiple axis-independent MPM transfers between particles and cell faces with a cubic kernel. To avoid this overhead, we adopt the recently proposed semi-staggered MPM incompressible fluid discretization [Zhang et al., 2017] and adapt it to solve the Locally Averaged Navier-Stokes with a variable coefficient multigrid preconditioner.

We first summarize the MPM discretization for the pressure solve in a pure fluid pipeline as in

[Zhang et al., 2017]. A standard pressure solve can be described by

$$-\frac{\Delta t}{\rho}\nabla \cdot (\nabla p) = -\nabla \cdot \mathbf{u}. \tag{4.12}$$

Pressures p are stored at cell centers, while all velocities are stored at cell nodes and are transferred between MPM particles and the background grid via the standard multi-linear kernel with FLIP [Brackbill and Ruppel, 1986]. We use the angular momentum conserving APIC [Jiang et al., 2015] instead for stability and low dissipation. We further adopt the RPIC damping as described in [Jiang et al., 2017a] as a controllable artificial viscosity for changing liquid behavior. The left-hand side of Eq. (4.12) can be discretized with a standard 7-point Laplacian stencil. For the right-hand side, the standard way to compute the divergence of velocity in the semi-staggered discretization is to first compute the partial derivatives on the edges and then average them to the cell center. Zhang et al. [2017] compute this differently by directly applying the divergence operator to the interpolated velocity which is then transferred to the weighting function, ending up with the identical form. It is well known that this semi-staggered discretization causes spurious hourglass velocity modes which cannot be eliminated by the pressure projection. We mitigate this problem by using an hourglass damping as proposed by Zhang et al. [2017].

Ghost pressure We mark a cell to be a water cell whenever there is a water particle present in the cell, and mark a cell to be Neumann when the cell center is inside an object or a wall. All other cells are air (Dirichlet) cells.

We use a simplified ghost pressure treatment [Gibou et al., 2002] to get a smooth free surface without tracking a level set or doing reconstruction from particles. One popular interpretation of the ghost pressure method is to compute the pressure gradient using a more accurate distance from the cell center to the free surface $\theta\Delta x$ (cf. Fig. 4.9(a)), instead of the distance between two cell centers Δx. We first identify the water particle closest to the face separating the water cell and the air cell. We then assign a sphere of radius $\frac{\Delta x}{2}$ to the particle to define an axis-aligned free surface and the corresponding θ (illustrated in Fig. 4.9(b)). It can be shown that when the water particle crosses the cell, this new free surface is continuous and passes through the air cell center; thus this scheme is free from cell-crossing discontinuity artifacts.

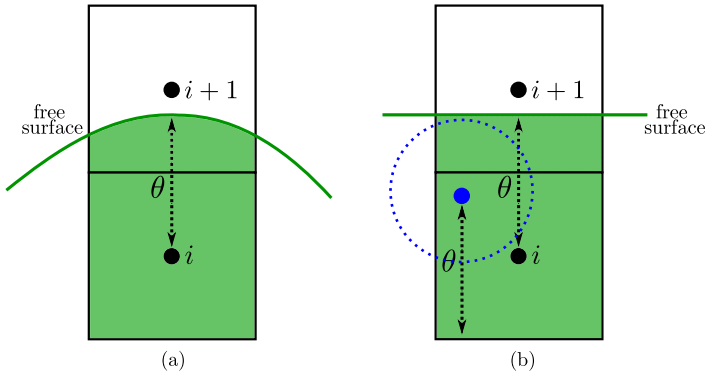


Figure 4.9: **Simple ghost pressure computation:** *Left:* With a more accurate free-surface representation, e.g. using level set, one can compute θ. *Right:* We propose a simplified way to compute θ even without the presence of a level set.

Adaptation to the mixture pipeline For solving the mixture projection Eq. (4.10), we need to compute the divergence of the volume-fraction-averaged mixture velocity: $\hat{\mathbf{u}} = \delta_i^{n_0}\mathbf{v}_i^n + \epsilon_i^n\mathbf{u}_i^*$. It is also located at cell nodes, so the divergence operator remains the same. Moreover, the 7-point stencil becomes a variable coefficient Laplacian $\nabla\cdot(\epsilon_f^n\nabla p)$. ϵ_f^n is usually stored at the cell faces to be collocated with the standard finite difference pressure gradient. Instead, we choose to interpolate nodal water volume fractions ϵ_i^n to cell centers to store them as ϵ_c^n and compute the face average whenever needed. In this way, the system to be solved remains symmetric. This choice also simplifies the definition of the restriction operator for the volume fraction in the multigrid preconditioner as discussed below.

Notice that the velocity correction Eq. (4.11) remains the same as in the standard pure fluid pipeline [Zhang et al., 2017]. Pressure gradients are first computed for each face; then the weighted average of these is used to update the nodal velocities. The face weight is set to be one whenever that face belongs to a water cell; otherwise it is set to be zero.

A variable coefficient multigrid preconditioner A multigrid preconditioner is utilized to improve the convergence rate of the conjugate gradient solver. Since the fluid volume fraction term ϵ_f^n is confined within a narrow range (due to clamping), we are able to slightly modify the technique proposed by McAdams et al. [2010] to achieve a reasonable convergence acceleration. In the multigrid V-cycle, two operators, namely the prolongation and restriction operators, are defined to act on the

pressure degrees of freedom. The restriction operator averages the finer data to be more coarse, and the prolongation operator interpolates finer values from the coarser grid. During each iteration, the pressures at each level are updated. In contrast, volume fraction coefficients only need to be restricted from finer levels to coarser levels once because they do not act as real degrees of freedom and can be re-used during the whole V-cycle.

As mentioned, although ϵ_f^n is required to solve the variable coefficient Poisson equation, we choose to store the volume fraction at cell centers and use them to compute ϵ_f^n whenever needed. This choice preserves the symmetry of the system. Furthermore, the restriction of volume fractions can be computed by averaging the interior cells. Our method differs from McAdams et al. [2010] in that we do not average values from all finer cells with zeros substituted into non-interior cells. With this modification, we make sure that when all of the volume fractions are one, the variable coefficient multigrid preconditioner can transform back to the constant coefficient case.

4.4.5 Sediment

The following sections describe each of the steps involved in advancing the states of the sediment particles from sub-step n_k to n_{k+1}, with $0 \leqslant k < K$.

Particles to Grid

Following [Klár et al., 2016], we transfer mass and momentum from particles to the grid using quadratic basis function according to

$$m_i^{s,n_k} = \sum_p m_p^s N_i(\mathbf{x}_p^{s,n_k}), \tag{4.13}$$

$$(m\mathbf{v})_i^{n_k} = \sum_p m_p^s N_i(\mathbf{x}_p^{s,n_k}) \left(\mathbf{v}_p^{n_k} + \frac{4}{(\Delta x)^2} \mathbf{B}_p^{s,n_k} (\mathbf{x}_i - \mathbf{x}_p^{s,n_k}) \right), \tag{4.14}$$

where the APIC $\mathbf{B}_p^{s,n_k}$ is originally set to zero and is updated at each substep according to Eq.(4.21). This is an APIC transfer [Jiang et al., 2015], with the factor $4/(\Delta x)^2$ applied to a quadratic kernel. The velocity on the grid is then computed as $\mathbf{v}_i^{n_k} = \frac{(m\mathbf{v})_i^{n_k}}{m_i^{s,n_k}}$.

In addition to this standard MPM transfer, we also compute mass gradients on grid nodes which

will be used later in our particle density approximation, namely

$$(\nabla m^{s,n_k})(\mathbf{x}_i) = -\sum_p m_p^s \nabla N_i(\mathbf{x}_p^{s,n_k}). \tag{4.15}$$

Furthermore, we update the nodal sediment volume fraction according to Eq.(4.7) during the transfer.

Update grid momentum and boundary conditions

In this step, we update the velocity $\mathbf{v}_i^{n_k} \to \hat{\mathbf{v}}_i^{n_{k+1}}$ by taking into account the drag force in regions where the fluid and sediment materials overlap along with the elastic forces from the sediment constitutive law. The drag force is given by Eq.(4.8). The elastic force is computed as in [Klár et al., 2016]:

$$\mathbf{f}_i^{se,n_k} = -\sum_p V_p^0 \left(\frac{\partial \psi}{\partial \mathbf{F}}(\mathbf{F}_p^{n_k})\right) (\mathbf{F}_p^{n_k})^\top \nabla N_i(\mathbf{x}_p^{s,n_k}), \tag{4.16}$$

with

$$\frac{\partial \psi}{\partial \mathbf{F}}(\mathbf{F}) = \mathbf{U}\left(2\mu\boldsymbol{\Sigma}^{-1}\ln\boldsymbol{\Sigma} + \lambda\,\mathrm{tr}(\ln\boldsymbol{\Sigma})\boldsymbol{\Sigma}^{-1}\right)\mathbf{V}^\top, \tag{4.17}$$

where $\mathbf{F} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top$ is the SVD of $\mathbf{F}$. The parameters λ and μ are the Lamé coefficients. The velocity update on the sediment grid is given by

$$m_i^{s,n_k}\hat{\mathbf{v}}_i^{n_{k+1}} = m_i^{s,n_k}\mathbf{v}_i^{n_k} + \Delta t\left(\mathbf{f}_i^{sd,n_k} + \mathbf{f}_i^{se,n_k} + m_i^{s,n_k}\mathbf{g}\right). \tag{4.18}$$

Finally, we apply the boundary conditions following [Stomakhin et al., 2013] to get the final update on the grid $\mathbf{v}_i^{n_{k+1}} \leftarrow \hat{\mathbf{v}}_i^{n_{k+1}}$.

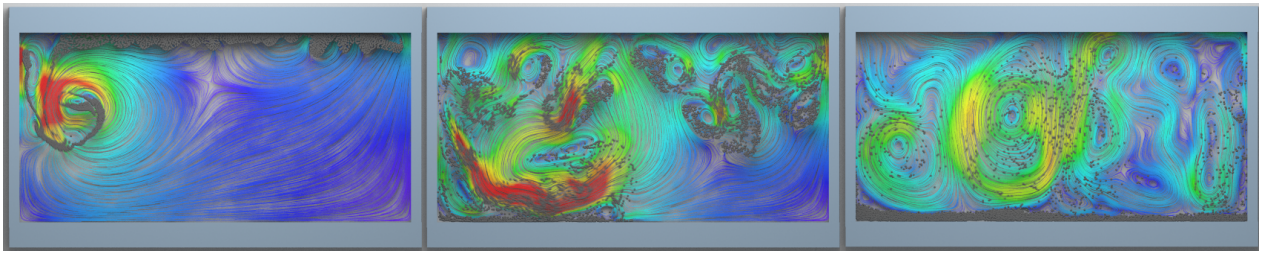


Figure 4.10: **Sedimentation 2D:** A box of dirt is dropped into a water container, creating interesting vortices before settling down.

Grid to particles

In this step, we update the velocity, position, and $\mathbf{B}_p^{s,n_{k+1}}$ of each particle following [Jiang et al., 2015]:

$$\mathbf{v}_p^{n_{k+1}} = \sum_i N_i(\mathbf{x}_p^{s,n_k})\mathbf{v}_i^{n_{k+1}}, \tag{4.19}$$

$$\mathbf{x}_p^{s,n_{k+1}} = \mathbf{x}_p^{s,n_k} + \Delta t^{s,n_k}\mathbf{v}_p^{n_{k+1}}, \tag{4.20}$$

$$\mathbf{B}_p^{s,n_{k+1}} = \sum_i N_i(\mathbf{x}_p^{s,n_k})\mathbf{v}_i^{n_{k+1}}(\mathbf{x}_i - \mathbf{x}_p^{s,n_k})^\top. \tag{4.21}$$

Strain update with density-based projection

To update the deformation gradient, we first compute

$$\mathbf{F}_p^{n_{k+1}} = \left(\mathbf{I} + \Delta t \sum_i \mathbf{v}_i^{n_{k+1}} \nabla N_i(\mathbf{x}_p^{s,n_k})^\top\right)\mathbf{F}_p^{n_k}.$$

This deformation is then projected so that it obeys the Drucker-Prager plasticity model as in [Tampubolon et al., 2017]. However, particles that are separated from clumps should not be subject to plasticity and should be treated as discrete debris with zero stress.

We approximate the particle density using a first-order accurate formula. Specifically, we expand $m(\mathbf{x})$ around $\mathbf{x}_i$ as $m(\mathbf{x}) \approx m(\mathbf{x}_i) + (\nabla m)(\mathbf{x}_i)\cdot(\mathbf{x} - \mathbf{x}_i)$. If we reconstruct this mass field at $\mathbf{x}_p$ accordingly, we get $m^i(\mathbf{x}_p) \approx m(\mathbf{x}_i) + (\nabla m)(\mathbf{x}_i)\cdot(\mathbf{x}_p - \mathbf{x}_i)$, where we use the superscript i to denote the use of node i for linearizing the mass field. We estimate particle density using the weighted

Table 4.2: **Simulation performance and parameters.** Superscript f and s represent fluid and sediment quantity respectively. The terms c, RPIC, and ρ are the drag force coefficient, RPIC damping coefficient, and density respectively. Inflow $\|\mathbf{v}\|$ indicates the magnitude of source velocity. The parameter ρ^f choices are adhoc and do not always reflect reality. We denote the number of particles by #P and grid resolution by Δx. The frame rate of all the examples listed above are 48. The young's modulus and Poisson's ratio of all examples are 1000 and 0.3 respectively. Both Transports (Fig. 4.4 Top and Bottom) were run with Intel(R) Xeon(R) CPU E3-1270 v3; Pile (Fig. 4.5) was run with Intel(R) Xeon(R) CPU E3-1241 v3; Flushing (Fig. 4.2) was run with Intel(R) Xeon(R) CPU E5-1650 v3; Sedimentation (Fig. 4.11) was run with Intel(R) Core(TM) i7-8700K; Debris (Fig. 4.3) was run with Intel(R) Core(TM) i7-7700K.

	Min/frame	Initial $\#P^f$	Initial $\#P^s$	Max $\#P^f$	Max $\#P^s$	Δx	c	RPIC	ρ^f	ρ^s	Inflow $\|\mathbf{v}^f\|$	Inflow $\|\mathbf{v}^s\|$
Transport (Fig. 4.4 Top)	4.9	0	3.1×10^5	1.0×10^6	3.9×10^5	1/512	0.9	0.2	1000	17	0.6	0.06
Transport (Fig. 4.4 Bottom)	4.9	0	3.1×10^5	1.0×10^6	3.9×10^5	1/512	0.9	0.4	1000	17	0.6	0.06
Pile (Fig. 4.5)	11.1	0	6.5×10^5	3.0×10^5	9.5×10^5	1/256	4	0	100	10	0.4	0.4
Flushing (Fig. 4.2)	21	0	2.3×10^5	1.9×10^6	2.3×10^5	1/256	2	0	1000	17	0.6	0
Debris (Fig. 4.3)	2.5	0	0	9.3×10^5	1.0×10^5	1/256	0	0	1000	150	0.15	0.015
Sedimentation (Fig. 4.11)	6.7	1.5×10^6	0	1.5×10^6	1.0×10^5	1/256	12	0.2	100	20	0	0.01

average of grid masses:

$$\rho_p = \sum_i N_i(\mathbf{x}_p) m^i(\mathbf{x}_p) \approx \sum_i N_i(\mathbf{x}_p)(m_i + (\nabla m)(\mathbf{x}_i) \cdot (\mathbf{x}_p - \mathbf{x}_i)). \tag{4.22}$$

In the discrete case, this is written as

$$\rho_p^{s,n_{k+1}} = \sum_i N_i(\mathbf{x}_p^{s,n_k}) \left(m_i^{s,n_k} + (\nabla m^{s,n_k})(\mathbf{x}_i)^\top (\mathbf{x}_p^{s,n_k} - \mathbf{x}_i) \right),$$

where $\nabla m^{s,n_k}(\mathbf{x}_i)$ is defined in Eq. (4.15). One can view the gradient of mass term as the first order approximation of nodal mass from the mass of the surrounding particles. We validated the accuracy of this strategy for a fluid simulation based on a weakly compressible Equation-of-State model [Becker and Teschner, 2007] and found that an interpolation solely based on mass produces grid artifacts. We would like to note that the gradient term can be seen as a second-order term in the Taylor expansion of the mass expression.

To utilize this density approximation, at time $t = 0$, we compute the initial average density

$$\hat{\rho} := \frac{1}{N_p} \sum_{p=1}^{N_p} \rho_p^{s,0}, \tag{4.23}$$

where the average is taken over all sediment particles that exist at time 0. If the current density

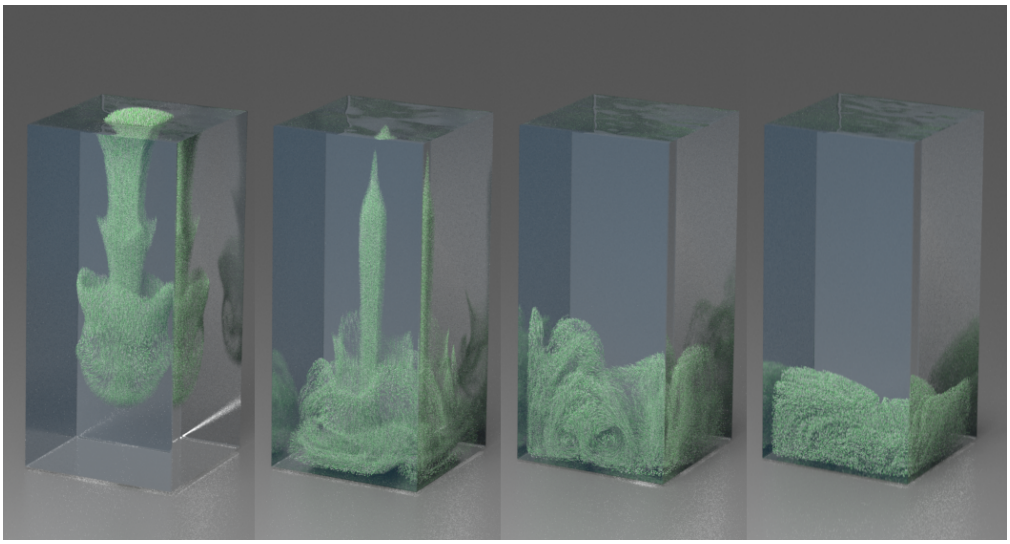


Figure 4.11: **Sedimentation 3D:** We pour sediment into a box of water until it settles at the bottom of the container.

approximation is lower than this initial average density, i.e. $\rho_p^{s,n_{k+1}} < \hat{\rho}$, we then identify the particle as a discrete kinematic body that does not undergo any elastic deformation. This is equivalent to setting

$$\mathbf{F}_p^{n_{k+1}} = \mathbf{I}.$$

Note that due to the usage of APIC, the discrete particle still acts as an independent body with a finite volume and maintains its angular momentum.

4.5 Results

We summarize the performance of our implementation in Table 4.2. In our pipeline, the Preconditioned Conjugate Gradient (PCG) projection remains the most expensive component (this step takes about 6.25 seconds in the 3D transport example, compared to 3.3 seconds for all the other operations). The sediment substep count depends on the CFL condition. In the debris flow simulation, we take $60 - 100$ sediment substeps per fluid step.

Fig. 4.8 shows the benefit of our debris treatment. As a block stirs a pile of sand, previous elastoplastic MPM algorithms introduce a severe volume gain. Tampubolon et al. [2017] proposed a

method to mitigate this volume gain effect to a certain extent. However, in a highly dynamic scenario like this, disconnected particles tend to suffer from intense resistance when connecting back to a more massive bulk. Our method offers a way to approximate state change, i.e., when particles act like debris based on the surrounding density. For a more accurate ballistic motion, more careful modeling is necessary [Smith et al., 2012].

Our algorithm can reproduce classical particle-laden phenomena such as sand dune migration (Fig. 4.1). In this example, a source of wind blows over a pile of sand as it moves from the left to the right, forcing sand to migrate. The left of the figure depicts the result of a simulation that is one-way-coupled (the wind affects the sand), which produces a non-physical behavior. The right figure depicts a two-way-coupled simulation, which creates more physically plausible dynamics on both the fluid and sand materials. The fluid (wind) part of this example is based on the node-based Eulerian fluid solver as proposed by [Guendelman et al., 2005] with vorticity confinement as described in [Fedkiw et al., 2001]. Note that all of the other examples with water as fluids (i.e., based on [Zhang et al., 2017]) do not use vorticity confinement.

Fig. 4.4 depict a particle-laden phenomenon that is created when a mixture of sand and water is being poured into a thin channel. The movement of the fluid transports the sediment. The presence of a cylindrical obstacle in the channel creates a hydraulic jump that carries the sand over to the other side. The use of RPIC blending helps to create a more laminar motion of the fluid (Fig. 4.4 bottom). A similar phenomenon of debris flow, where a water and sand mixture is poured into a wider channel with more complex collision objects is illustrated in Fig. 4.3.

Our algorithm can also handle sedimentation phenomena, which is depicted by Fig. 4.10 and Fig. 4.11. In these 2D and 3D simulations, a box of sand is being dropped into a water container. We can observe vortices that are created by the interaction between water and sand materials. The sand materials eventually settle down at the bottom of the container.

Fig. 4.5 depicts a simulation where pure sediment particles form a pile due to static friction (top). When liquid is injected, the pile collapses and interacts with the fluid. Similarly, water is poured on top of a pile of sediment that initially sits at the bottom of a glass container in Fig. 4.2. The overflow of water washes the sediment out of the glass.

4.6 Discussion

Our model qualitatively captures a number of distinctive dynamic features of particle-laden flows (such as dune migration and sediment transport) and stands as a practical tool in animation and VFX. As suggested in Sec. (§4.3.3), our model of the interaction term only uses drag forces based on [Di Felice, 1994]. We do not make any claims on how broadly this model effectively captures empirical experiments. As such, a more rigorous study of sediment-fluid interaction would require careful modeling of the other nontrivial effects such as the lift force as discussed by Sun and Xiao [2016b].

Another limitation of our method is the clamping of the sediment volume fraction (at locations where sediment and fluid co-exist), which is motivated by numerical stability considerations. As such, our simulations cannot capture the variations of granular flow as studied in [Pailha et al., 2008]. Furthermore, at the interface of dry and immersed sand, we do not apply any special treatment for the sudden jump of sediment volume fraction from $\delta = 0.8$ (the case of liquid and sediment co-existing) to $\delta = 1$ (the case of pure sediment) as no noticeable artifacts were observed.

We presently rely on artificial viscosity from RPIC for adjusting the extent of fluid turbulence. More physical and accurate viscosity models incorporated into the Locally Averaged Navier-Stokes discretization would be appropriate to investigate in future work.

Due to the explicit treatment of the drag force, we occasionally encounter numerical instability when the relative velocity is too high, or the fluid volume fraction is too low (thus resorting to the usage of a clamping threshold). Exploring a fully implicit treatment of the interaction term would be a potential direction for future study. Furthermore, the design of a preconditioner explicitly tailored to the variable-coefficient Poisson system is an interesting research question.

5 GPU OPTIMIZATION OF MATERIAL POINT METHODS

The Material Point Method (MPM) has been shown to facilitate effective simulations of physically complex and topologically challenging materials, with a wealth of emerging applications in computational engineering and visual computing. Borne out of the extreme importance of regularity, MPM is given attractive parallelization opportunities on high-performance modern multiprocessors. Unlike the conceptually simple CPU parallelization, a GPU optimization of MPM that fully leverages computing resources presents challenges that require exploring an extensive design-space for favorable data structures and algorithms. In this chapter we introduce methods for addressing the computational challenges of MPM and extending the capabilities of general simulation systems based on MPM, particularly concentrating on GPU optimization. In addition to our open-source high-performance framework, we also perform performance analyses and benchmark experiments to compare against alternative design choices which may superficially appear to be reasonable, but can suffer from suboptimal performance in practice. Our explicit and implicit GPU MPM solvers are further equipped with a Moving Least Squares MPM heat solver and a novel sand constitutive model to enable fast simulations of a wide range of materials. We demonstrate that more than an order of magnitude performance improvement can be achieved with our GPU solvers. Practical high-resolution examples with up to ten million particles run in less than one minute per frame.

5.1 Introduction

The Material Point Method (MPM) is a hybrid Lagrangian/Eulerian computational scheme that has been shown to simulate a large variety of traditionally-challenging materials with visually rich animations in computer graphics. Recent examples of MPM-based methods developed for such materials include simulations of snow [Stomakhin et al., 2013], granular solids [Klár et al., 2016], multi-phase mixtures [Stomakhin et al., 2014; Tampubolon et al., 2017; Gao et al., 2018], cloth [Jiang et al., 2017a] and many others. MPM has been shown to be particularly effective for simulations involving a large number of particles with complex interactions. However, the size

and the complexity of these simulations lead to substantial demands on computational resources, thereby limiting the practical use cases of MPM in computer graphics applications.

Using the parallel computation power of today's GPUs is an attractive direction for addressing computational requirements of simulations with MPM. However, the algorithmic composition of an MPM simulation pipeline can pose challenges in fully leveraging compute resources in a GPU implementation. Indeed, MPM simulations include multiple stages with different computational profiles, and the choice of data structures and algorithms used for handling some stages can have cascading effects on the performance of the remaining computation. Thus, discovering how to achieve a performant GPU implementation of MPM involves a software-level design-space exploration for determining the favorable combinations of data structures and algorithms for handling each stage.

In this chapter we introduce methods for addressing the computational challenges of MPM and extending the capabilities of general simulation systems based on MPM, particularly concentrating on a high-performance GPU implementation. We present a collection of alternative approaches for implementing all components of the MPM simulation on the GPU and provide test results that identify the favorable design choices. We also show that design choices that may superficially appear to be reasonable can suffer from suboptimal performance in practice. Furthermore, we introduce novel methods for thermodynamics and simulation of granular materials with MPM. More specifically, this chapter includes the following contributions:

1. A novel, efficient, and memory-friendly GPU algorithm for accelerated MPM simulation on a GPU-tailored sparse storage variation of CPU SPGrid [Setaluri et al., 2014].

2. A performance analysis of crucial MPM pipeline components, with several alternative strategies for particle-grid transfers.

3. A collocated, weak form-consistent, MLS-MPM-based implicit heat solver that enables thermo-mechanical effects on elastoplasticity.

4. An easy-to-implement unilateral hyperelasticity model with non-associative flow rule for cohesionless granular media.

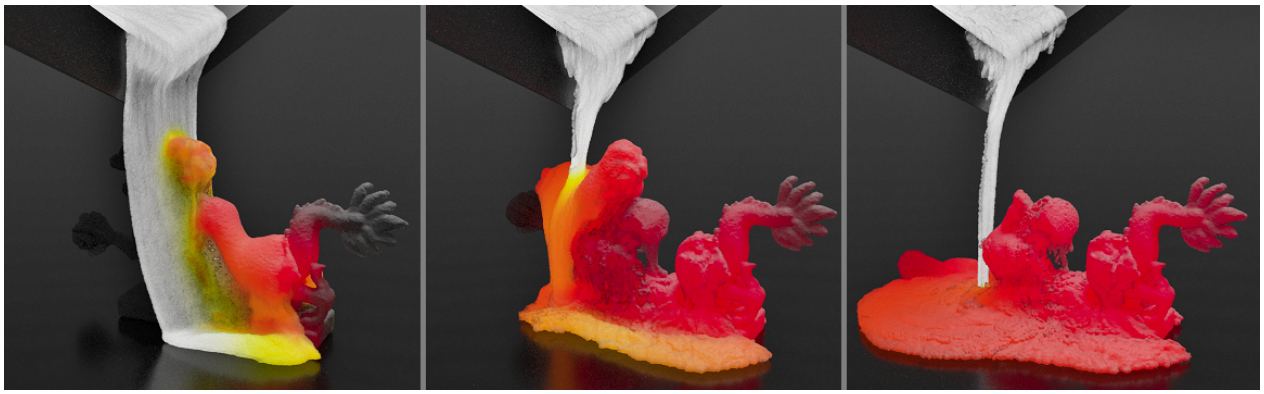


Figure 5.1: **How to melt your dragon.** Melting an elastoplastic dragon with 4.2 million particles on a 256^3 grid using our GPU-optimized implicit MPM dynamics and heat solvers on a Nvidia *Quadro* P6000 GPU at an average 10.5 seconds per 48Hz frame.

Our experiments show that more than an order of magnitude performance improvement can be achieved using the favorable choices for data structures and algorithms, as compared to optimized solutions using different computational models. We demonstrate that complex MPM simulations with up to 10 million particles can be simulated within a minute per frame using the methods we describe. We also present results showing that our heat solver can effectively handle phase transition effects and that our hyperelasticity model for granular materials allows achieving consistent simulation results with explicit and implicit integrations.

5.2 Background

Material point method. MPM was introduced by Sulsky et al. [1995] as the generalization of the hybrid Fluid Implicit Particle (FLIP) method [Brackbill, 1988; Zhu and Bridson, 2005; Bridson, 2008] to solid mechanics. It has been recognized as a promising discretization choice for animating various solid materials including snow [Stomakhin et al., 2013], foam [Ram et al., 2015; Yue et al., 2015], sand [Daviet and Bertails-Descoubes, 2016; Klár et al., 2016], cloth [Jiang et al., 2017a; Guo et al., 2018], fracture [Wretborn et al., 2017], cutting [Hu et al., 2018] and solid fluid mixture [Stomakhin et al., 2014; Gao et al., 2018; Tampubolon et al., 2017].

GPU-based simulation methods. Many researchers also divided Eulerian simulation domain in order to to parallelize computation on the GPU [Horvath and Geiger, 2009; Liu et al., 2016; Chu

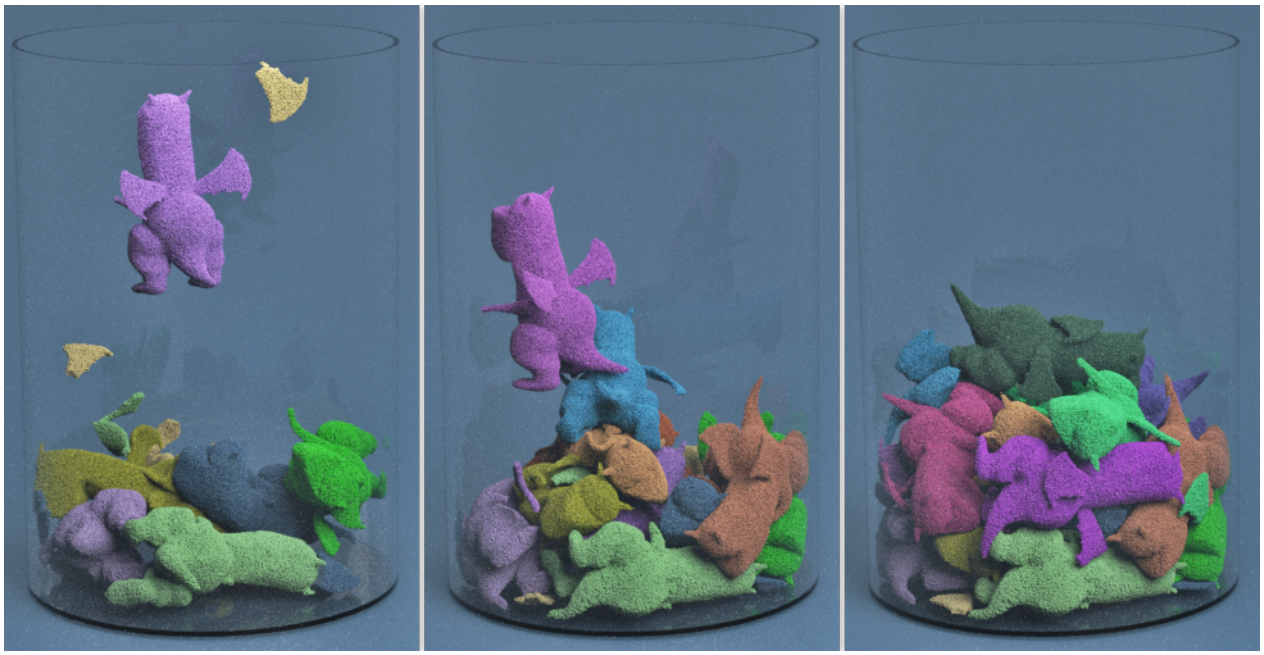


Figure 5.2: **How to stack your dragon.** Stacking elastic dragons in a glass. This simulation contains 9.0 million particles on a 512^3 grid with an average 21.8 seconds per 48Hz frame.

et al., 2017]. Wang [2018] performed a GPU optimization on sewing pattern adjustment system for cloth. Others explored solutions to the computational heavy task of self-collision detection, using GPUs [Govindaraju et al., 2005, 2007] with improved spatio-temporal coherence and spatial hashing [Tang et al., 2013, 2016, 2018; Weller et al., 2017; Wang et al., 2018]. In particular, the spatial hashing table [Weller et al., 2017] has been proposed as both an acceleration structure and a substitution to the hierarchy of uniform grids for collision query in order to lower the memory consumption. Furthermore, the histogram sort (alternative for radix sort) has been proven to be capable to significantly reduce the overhead of sorting if the number of bins is adequately smaller than the total elements count [Wang et al., 2018]. These practices provided great foundations for our MPM pipeline as they fit well with the sparse grid structure.

Sparse grid data structures. From the Eulerian view, simulation domain is represented by discretized grid. Museth et al. [2013] developed OpenVDB, which is a tree with a high branching factor that yields a large uniform grid at leaf nodes. This adaptive data structure has been shown to be very efficient and broadly used. Hoetzlein [2016] proposed an GPU sparse grid structure, GVDB

Voxels, inspired by OpenVDB. The voxel data is represented in dense n^3 bricks allocated from a pool of sub-volumes in a *voxel atlas*. The atlas is implemented as a 3D hardware texture to enable trilinear interpolation and GPU texture cache. Recently, Wu et al. [2018] extended GVDB [Hoetzlein, 2016] with dynamical topology update and GPU optimized matrix-free conjugate gradient solver for fluid simulations with tens of millions particles. Although the use of texture to store volumetric data can benefit performance for general purpose usage, such as hardware trilinear interpolation and fast data accessing, it prevents GVDB using scattering rather than gathering because atomic operations on texture are not allowed to be used under current GPU hardware.

SPGrid [Setaluri et al., 2014] provided an alternative sparse data structure; it has been adopted in large-scale fluid simulations [Aanjaneya et al., 2017; Setaluri et al., 2014; Liu et al., 2016] and in the MPM context [Gao et al., 2017, 2018; Hu et al., 2018]. SPGrid improves the data locality by mapping from a sparse 2D/3D array to a linear memory span by following a modified Morton coding map. Furthermore, it exploits hardware functions to accelerate the translation between geometric indices and the 64-bit memory offsets. The neighborhood accesses can be achieved in $\mathcal{O}(1)$, rather than $\mathcal{O}(\log n)$ for traditional tree-based sparse storage schemes (n is the depth of the tree).

5.2.1 MPM Overview

MPM, as a hybrid spatial discretization method, benefits from the advantages from both Lagrangian and Eulerian views. MPM uses Lagrangian particles to carry material states including mass m_p, position $\mathbf{x}_p$, velocity $\mathbf{v}_p$, volume V_p, deformation gradient $\mathbf{F}_p$, etc. The grid acts as an Eulerian scratchpad for computing stress divergence and performing numerical integration. Grid nodes represent the actual degrees of freedom, which store mass m_i, position $\mathbf{x}_i$ and velocity $\mathbf{v}_i$ on each node i.

A typical first-order MPM time integration scheme for incremental dynamics from t^0 to t^1 (with $\Delta t = t^1 - t^0$) contains the following essential steps:

1. Particles-to-grid (P2G) transfer of masses and velocities: $\{m_i^0, \mathbf{v}_i^0\} \leftarrow \{m_p, \mathbf{v}_p^0\}$;
2. Grid velocity update using either explicit or implicit integration: $\mathbf{v}_i^1 \leftarrow \mathbf{v}_i^0$;
3. Grid-to-particles (G2P) transfer of velocities and strain increments: $\{\mathbf{v}_p^1, \mathbf{F}_p^1\} \leftarrow \mathbf{v}_i^1$

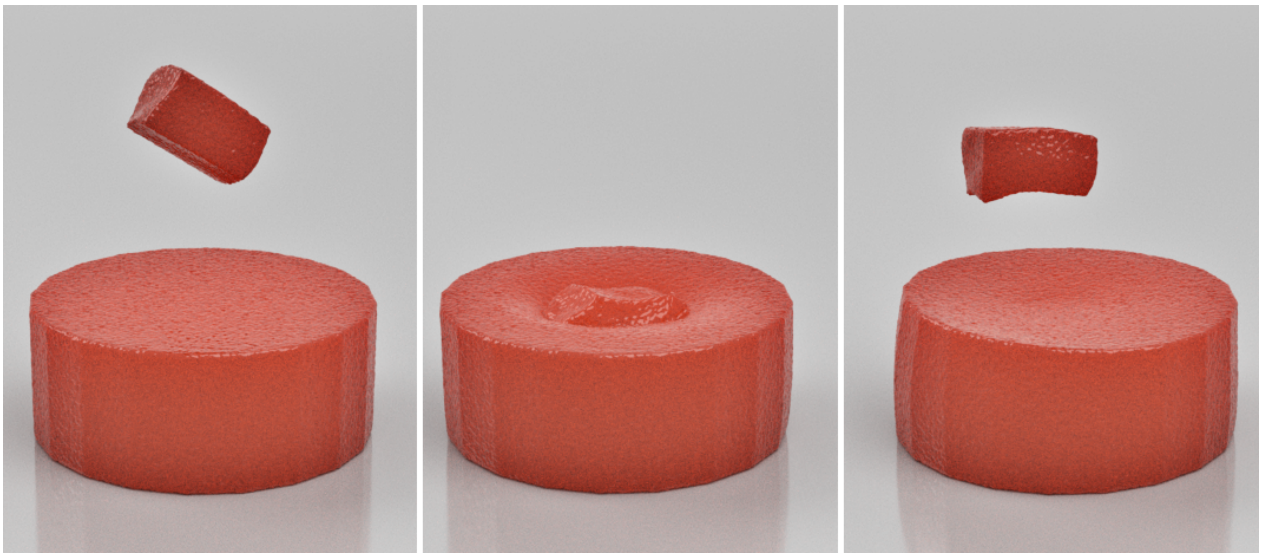


Figure 5.3: Elasticity simulation of Gelatin bouncing off Gelatin with 6.9 million particles on a 512^3 grid at an average 6.72 seconds per 48Hz frame.

4. Particle-wise stress evaluation and plasticity treatment that modifies $\mathbf{F}_p^1$.

Assuming the usage of a matrix-free Krylov solver for the linearized system (due to its superior efficiency), both explicit and implicit Euler time integrations of MPM break the entire computation procedures in (1)-(3) into particle-grid transfer operations of physical quantities and their differentials. As such, the key to high-performance MPM is the optimization of particle-grid transfer operators.

Unsurprisingly, the algorithmic choice of the transfer scheme plays a considerable part in affecting the computing performance. We design our benchmarks based on three possible choices of the transfer scheme: FLIP [Zhu and Bridson, 2005], APIC [Jiang et al., 2015] and the recently proposed MLS-MPM [Hu et al., 2018]. Note that MLS-MPM provides an additional algorithmic speed-up by avoiding kernel weight gradient computations in step (2) (discussed in more detail in §5.4 and by Hu et al. [2018]).

5.2.2 Particle-to-Grid Transfer

As in many other Lagrangian-Eulerian hybrid methods, the particles carry material states and transfer them to the grid as mentioned in §5.2.1 step (1). In GPU, the two most common approaches for performing parallel state transfer are *gathering* and *scattering*. The former one gathers states from

all nearby particles to one particular grid node; while the latter one distributes all states of one particular particle to all influenced grid nodes.

Domain decomposition technique is commonly used in the gathering strategy [Huang et al., 2008; Parker, 2006; Zhang et al., 2010; Stantchev et al., 2008]. Using this strategy the simulation domain is divided into several small sub-domains such that each node only needs to check all the particles within the neighboring sub-domains instead of the whole domain. [Klár et al., 2017] not only group particles but also group nearby sub-domains to obtain better data accessing pattern. Ghost particles are introduced to minimize the number of communications and barriers within sub-domains [Parker, 2006; Ruggirello and Schumacher, 2014]. Chiang et al. [2009] maintain and update particle lists for each node during the simulation. Obviously, it requires a huge amount of GPU memory as well as the processing time for updating particle lists. In order to reduce the workload, Dong et al. [2015] extend the influencing range the grid nodes and further adapt their method to multiple-GPU [Dong and Grabe, 2018]. Wu et al. [2018] pre-compute a subset of the particles for each grid cell. However, their method still needs to unnecessarily examine a large amount of particles within each voxel. Furthermore, for storing the subsets, it not only requires expensive data movements but also consumes a large amount of memory. To summarize, all the fundamental issues of gathering are due to the need for accessing associated particles of a node. There is no satisfying solution reading the data efficiently as well as saving the extra memory for particle lists, etc. The other inherent problem is that the workload of each GPU thread is closely related to the length of particle list of each grid node, thus thread divergence generally exists and slows down P2G.

On the contrary, the scattering method is free from all these critical issues, but its performance suffers from the write-conflict. Both Harada et al. [2007] and Zhang et al. [2008] introduced parallel scattering methods to distribute particle attributes to neighboring particles in SPH. In their methods, particle attributes are written into 2D texture buffer simultaneously using graphic rasterization pipeline to solve write-conflicts. However, their scattering methods cannot be easily employed in a 3D Eulerian simulation framework because the current hardware rasterization technique only works for 2D texture, not to mention the added complexity to build a mapping between a sparse grid and 2D texture during rasterization. To the best of our knowledge, this chapter proposes the first

parallel particle-to-grid transferring technique based on scattering, which utilizes modern graphics hardware features.

5.3 Optimized GPU scheme for MPM

In this section we describe our optimized GPU scheme for MPM using a GPU-tailored sparse paged grid data structure. The overall algorithm is summarized in Algorithm 1.

5.3.1 GSPGrid Tailored to MPM

We introduce *GSPGrid*, the GPU adaptation of the SPGrid [Setaluri et al., 2014] data structure which facilitates the sparse storage required for efficient simulations. Though not limited to such a choice, a $4 \times 4 \times 4$ spatial dissection of the computational domain into GSPGrid blocks is assumed in our implementation.

We take a similar strategy to [Gao et al., 2017], and briefly summarize it here. We use the quadratic B-spline functions as the weighting kernel, so each particle is associated with $3 \times 3 \times 3$ grid nodes (3×3 in 2D as shown in the Fig. 5.4). We assign each particle to the cell whose "min" corner collocates with the "smallest" node in the particle's local $3 \times 3 \times 3$ grid. All particles whose corresponding cells are within the same GSPGrid block are attached to that block. Geometrically, the particles of a particular GSPGrid block all reside in the same dual block, i.e. the $\Delta x/2$-shifted block (the red dashed block in Fig. 5.4). All computations for transferring particles' properties to/from the grid will be conducted locally in the same GSPGrid block.

We associate each particle with one CUDA thread. The particles are sorted such that the computations of particles sharing the same dual cell are always conducted in sequential threads. In scenarios involving high particle densities, the number of particles within a single GSPGrid block can easily go beyond the maximum number of threads allowed in a CUDA block under the current graphic architecture. To solve this issue, we assign each GSPGrid block to one or several CUDA blocks and generate the corresponding *virtual-to-physical* page mapping. In this way, we can treat each CUDA block separately without considering their geometric connections.

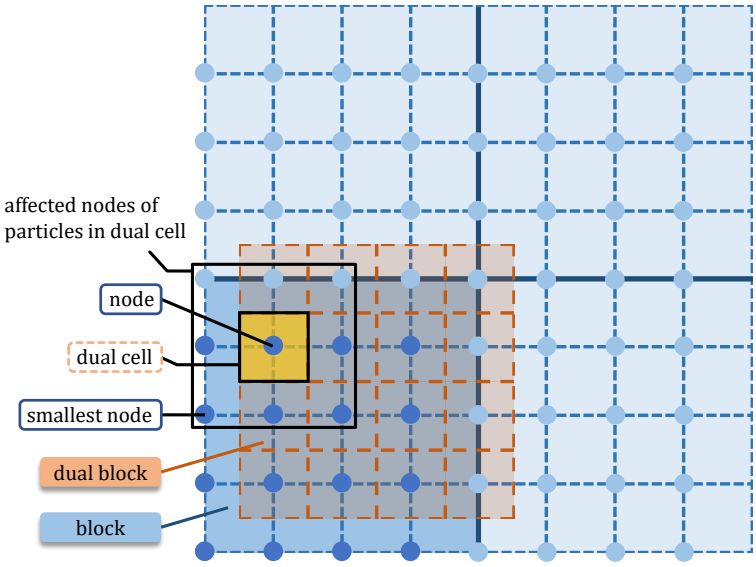


Figure 5.4: **Mapping from particles to blocks:** All particles in the yellow dual cell interact with the same set of the 27 nodes (9 in 2D). Notice that these particles also distribute their states to the grid nodes of the top neighboring block.

Notice that, for some particles, e.g. the ones in the yellow cell in Fig. 5.4, reading and writing will involve nodes from neighboring blocks. To deal with such particles, the shared memory of each CUDA block temporarily allocates enough space for all 8 neighboring blocks (4 in 2D). One CUDA block usually handles hundreds of particles in parallel, which makes it affordable to allocate enough shared memory for all neighboring blocks. For example, in Grid-To-Particle transfer, we can first fetch all required data into the shared memory from the global grid, and then use the data to update particles' properties.

As discussed above, it is possible for a particular block to access data from its neighboring blocks. In SPGrid, the offsets of the neighboring blocks can be easily computed for addressing them. However, without the support of virtual memory space in GPU, we also need to store the address information of neighboring blocks. Fortunately, the spatial hashing algorithm is able to construct the topology of neighboring blocks in $\mathcal{O}(1)$ complexity on the GPU.

Algorithm 1 Sparse MPM simulation on the GPU

```
1:  procedure SparseMPM( )
2:      P ← initial points
3:      P ← sort(P)                                          ▷ Section 5.3.3
4:      for each timestep do
5:          dt ← compute_dt(P)
6:          V ← GSPGrid structure(P)                         ▷ Section 5.3.1
7:          M ← particle_grid_mapping(P, V)                  ▷ Section 5.3.1
8:          V ← particles_to_grid(P, M)                      ▷ Section 5.3.2
9:          V ← apply_external_force(V)
10:         V ← grid_solve(V, dt)
11:         P ← grid_to_particles(V, M)
12:         P ← update_positions(P, dt)
13:         P ← sort(P)                                      ▷ Section 5.3.3
```

5.3.2 Parallel Particle-to-Grid Scattering

For particle-to-grid transfers, geometrically-neighboring particles can write into the same nodes, making write hazards a critical problem which has been examined in many Eulerian-Lagrangian hybrid methods as discussed in §5.2.2. There are generally two schemes for resolving this problem, scattering and gathering. Scattering methods simply use atomic operations to avoid conflicts. On the other hand, for gathering, usually a list of particles are created and maintained during the simulation; thus each node can track down all the particles within its affecting range.

Almost all previous papers are opting for a gathering approach since it is widely believed that high frequent atomic-operations in scattering can significantly undermine the performance especially in GPU-based parallel applications. However, we propose a method to handle most of the write conflicts for scattering without atomic operations, inspired by the concept of parallel reduction sum [Luitjens, 2014] and show that the performance is superior to the gathering ones. As we divide the whole grid domain as well as their corresponding particles into blocks, there are two levels of write hazards in P2G, within-block hazards and crossing-block hazards. As observed in practice, the within-block conflicts are the absolute majority. The within-block ones can be further divided into within-warp and crossing-warp conflicts and our solution particularly tackles the within-warp ones.

As shown in Fig. 5.5, within a warp, a few groups of particles tend to add their attributes to the corresponding nodes. For particles of a particular group (e.g., particles 1-4), simultaneous write

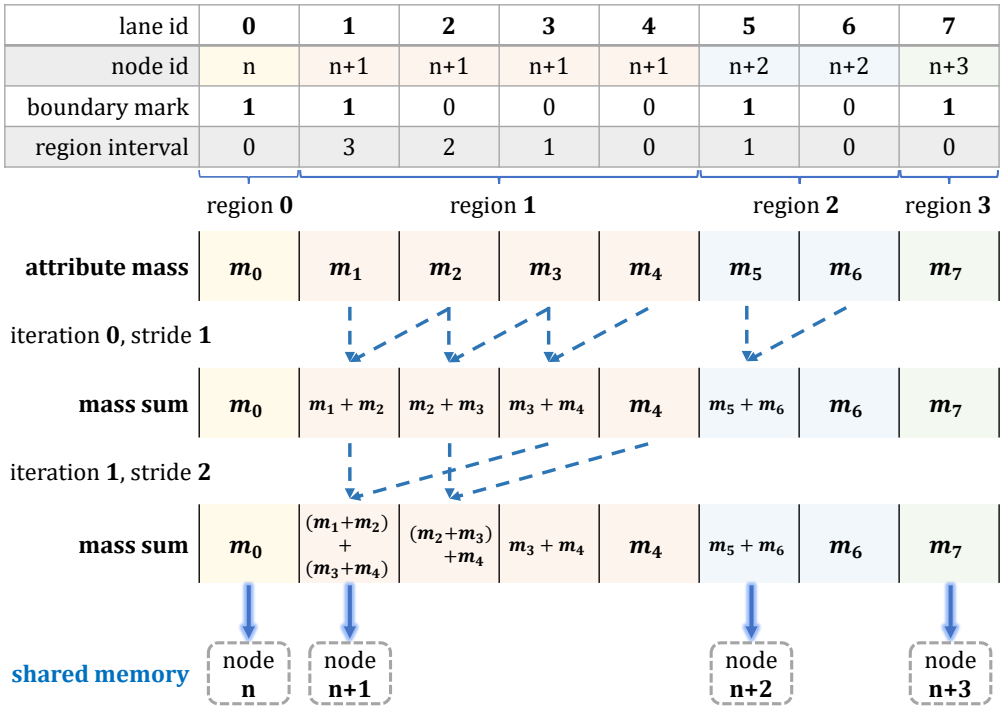


Figure 5.5: **Optimized transfer from particles to grid nodes**. We apply CUDA warp intrinsics to accelerate hierarchical attribute summations. The results are first written to the shared memory and then transfered to global memory in bulk.

operations into the same place need careful treatments to guarantee correctness. We exploit the warp-level CUDA intrinsics, i.e. *ballot* and *shfl*, to resolve this problem. First, a representative of each group is chosen (i.e. the left most one whose boundary mark is set on). Then attributes from all the other particles within the same group are added to the representative one by iteratively shuffling from right to left. The number of iterations can be further reduced by using Algorithm 2. Finally, the representative thread is responsible for writing the sum to the target node, as illustrated in Algorithm 3. In this way, all warp-level conflicts are eliminated. To further reduce the impact of crossing-warp conflicts, the shared-memory acts as a buffer for temporarily holding the results from different warps by applying atomic-adds. Notice this crossing-warp conflicts only infrequently happen such that the cost of the corresponding atomic-adds is almost negligible.

5.3.3 Cell-based Particle Sorting

Since the positions of the particles are updated in each time-step, the GSPGrid data structure should be refreshed accordingly. To build the new mapping from the GSPGrid blocks to the continuous GPU memory, we need first to re-sort the particles to help identify all the blocks occupied or touched by the particles.

SPGrid translates the cell indices to the 64-bit offsets, which can be used as the keyword for the re-sorting. Radix sort is generally considered the fastest sorting algorithm on GPU. However, in practice the fraction of the occupied offsets to the maximum capacity that a 64-bit offset can represent is too sparse. We instead use spatial hashing to register GSPGrid blocks. Whenever a new block is inserted, the block number to be assigned increments by one. The transformed block indices of all the particles are thus consecutive, and the maximum block index is usually several orders of magnitude smaller than the particle number. Also, all particles within the same block are further partitioned by $4 \times 4 \times 4$ cells. This particular layout is required by the follow-up computations including pre-calculating the smallest index of each particle and reducing write-conflicts during P2G transfer (§5.3.2).

With CPU SPGrid, radix sort facilitates proximity (in memory) of geometrically neighboring blocks to improve prefetching efficiency. In GPU, this is not a concern anymore; thus we use histogram sort instead of the conventional radix sort. The new keyword, which is a combination of

Algorithm 2 Warp Computation

```
1:  procedure ComputeBoundaryAndInterval(int laneid, int* cellids)
2:      cellid ← cellids[laneid]
3:      cellid_prev ← cellids[laneid − 1]
4:      if laneid = 0 ∨ cellid ≠ cellid_prev then
5:          boundary ← true
6:      mark ← brev(ballot(boundary))
7:      interval ← countFollowingZeroes(mark, laneid)
8:      stride ← 1
9:      iter ← interval
10:     while stride < 32 do
11:         tmp ← shfldown(iter, stride)
12:         iter ← max(tmp, iter)
13:         stride ← stride << 1                    ▷ //move on to a higher level
14:     iter ← shfl(iter, 0)                        ▷ //broadcast the maximum iterations
```

Algorithm 3 Warp Write

```
1:  procedure GatherAndWrite(T* buffer, T attrib, int iter)
2:      stride ← 1
3:      val ← attrib
4:      while stride <= iter do                          ▷ //hierarchical summation
5:          tmp ← shfldown(val, stride)
6:          if stride <= interval then                   ▷ //only sum within the group
7:              val ← val + tmp
8:          stride ← stride << 1                         ▷ //move on to a higher level
9:      if boundary then                                 ▷ //only the boundary node needs to write
10:         *buffer ← *buffer + val                      ▷ //AtomicAdd is applied
```

the transformed block index and dense cell index within the block, works as the reference to the bin. This algorithm provides $5\times$ speedup, and in collaboration with *delayed ordering technique* (§5.3.4), reordering the particles is no longer a bottleneck.

5.3.4 Particle Reordering

Given the sorted indices from §5.3.3, one natural thing to do next is to reorder all particles' properties accordingly since sequential memory accesses are always preferable in CUDA kernels. However, for a high-resolution simulation, this reordering itself can be the bottleneck, since each particle carries various different properties such as positions, velocities, and deformation gradient, etc., and loading and writing those data in a non-coalesced manner can be time consuming. We propose a practical way to completely dispense with the reordering. We observe that any functions which take in scattered data and write the updated result back in a sequential order can actually reorder the data as a byproduct. In essence, a pure reordering function is one of this kind of functions which simply writes untouched inputs back in a different order. If we choose the functions carefully, instead of simply using a pure reordering function, the cost/latency due to the scattered memory reads can be largely mitigated.

The particles' positions are the only property we decide to reorder since they are essential for almost all kernels that build mappings between particles and grid blocks/cells, and for all transfer kernels which need to compute weights and weight gradients from the positions. We treat all the other properties in several different ways based on their attributes and the scenarios they are being

used. We list a few of them as the most typical cases. Notice that most CUDA kernels follow the positions' order, except for certain material-based computations.

Mass

Since each particle's mass remains constant, their sequence will never be changed; for retrieving the correct mass, we only need to map from the current positions' order to the original masses' order.

Velocity

Velocities are updated every time-step in the G2P kernel. The new velocities are forced to follow the order of the threads/positions by simply writing them back sequentially. However, when the velocities are being used as the inputs to some kernels in the next time-step, they are not matching with new positions' order since we choose only to reorder positions at the end of each time-step. As a result, a mapping from the current order to the previous order is required.

Stress

When computing the stresses, we choose not to change the order of either, the inputs $\mathbf{F}$, or the outputs $\mathbf{P}$. One reason is that the GPU SVD function can be optimized (§5.4.3) so fast that it may be inefficient to amortize the scattered writes.

5.4 Benchmarks and performance evaluation

To evaluate our GPU MPM algorithm, we create several benchmarks, starting with the uniform particle distribution (§5.4.1). Firstly, we compare the performance of the transfer kernels between our method and one SIMD-optimized multi-core CPU implementation as well as one gathering-based GPU implementation with GVDB as the sparse background grid. Secondly, with the total number of particles being fixed, we vary the particle densities (particles per cell, PPC) to examine how our pipeline can be affected.

We further evaluate the performance with non-uniform particle distributions, e.g. Gaussian distributions (§5.4.2). In this experiment, we fix both the total number of particles and the total

number of cells being occupied by the particles; while the PPC of each single cell can vary following particular Gaussian distributions.

In addition to the explicit pipeline, we also measure the performance of the key kernels merely used in the implicit time integration [Stomakhin et al., 2013]. Since the results of the singular value decomposition (SVD) of the deformation gradient are required in the computation of the stress derivative in every single iteration, one can always pre-compute the SVD and store the results in advance. However, our experiment (§5.4.3) reveals some interesting findings.

Notice that, as mentioned before, the three-dimensional GSPGrid block with 16 channels is of resolution $4 \times 4 \times 4$ and eight particles per cell are usually required for stability considerations in MPM applications. Hence, we choose to allow each CUDA block to process at most 512 particles, due to the limitation of the current hardware architecture.

Unless otherwise stated, all of our GPU tests are performed on Quadro P6000.

5.4.1 Uniform Distribution Benchmarks

Comparisons with two state-of-the-art implementations

We create a benchmark with $\Delta x = \frac{1}{128}$. The particles are uniformly sampled on a grid from $(\frac{1}{8}, \frac{1}{8}, \frac{1}{8})$ to $(\frac{7}{8}, \frac{7}{8}, \frac{7}{8})$ with spacing $\frac{1}{256}$. The total number of the particles is just over 7 million particles. For this test, the CPU benchmark was performed on an 18-cores Intel(R) Xeon(R) Gold 6140 CPU and all GPU measurements were performed on a Nvidia Titan Xp. The results are in Fig. 5.6.

We first compare our scattering-based GPU implementation of particle-grid transfers to one SIMD-optimized CPU implementation in [Gao et al., 2017]. Both P2G and G2P achieve more than 15x speedups.

Then we compare with one most recent GPU gathering implementation with GVDB as the sparse grid structure [Wu et al., 2018]. Their idea is basically to pre-compute a subset of the particles for each grid cell. All particles influencing the grid nodes inside the cell will be included in that list. When performing the particle-to-grid transfer, each node needs to check all points in the list to determine whether they are close enough. Therefore, each node has to check much more unnecessary points than needed (only 20% utilization for MPM FLIP with subcell size 4^3). In order

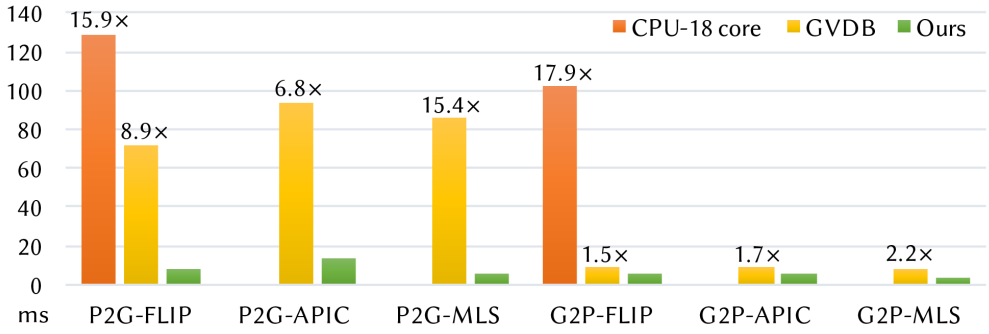


Figure 5.6: **Transfer benchmark.** We compare our scattering GPU transfers (Nvidia Titan Xp) to a gathering GPU transfer implementation using GVDB [Wu et al., 2018] (Nvidia Titan Xp) and a SIMD CPU implementation [Gao et al., 2017] on an 18-cores Intel(R) Xeon(R) Gold 6140 CPU. The transfer scheme of the CPU implementation is FLIP.

to do a comprehensive comparison, we use three different transfer schemes, including FLIP [Zhu and Bridson, 2005], APIC [Jiang et al., 2015], and MLS [Hu et al., 2018]. Notice that since their gathering method need to create lists for all particle attributes, such as velocity, position, stress, and deformation gradients, it takes half of computation time to load and store data. More importantly, it consumes tremendous amounts of GPU memory.

For our P2G kernels, the computing workload and the memory access workload are well balanced. Therefore, the timing is bounded by both memory and computations. FLIP only needs to compute nodal mass, traditional translational momentum and forces; while APIC also needs to load one additional matrix for including the affine velocity modes. In contrast, MLS completely avoids the computation of weight gradients, and the two matrix-vector multiplications of APIC (i.e. computing the force and the affine modes) can be merged into one [Hu et al., 2018].

For our G2P kernels, memory is utilized more heavily than computing units. Compared to APIC and MLS, FLIP also needs to load the nodal velocity increments for updating the particles' velocities. However, APIC and MLS have to refresh one extra matrix for recording the affine velocity modes; while MLS can merge the updates of $\mathbf{F}$ and that extra matrix into one to reduce the total cost [Hu et al., 2018]. GVDB uses a 3D texture to store volume data to utilize the hardware tri-linear texture interpolation functions; however MPM cannot benefit from this because of the quadric weighting functions. Furthermore, we exploit the shared memory to pre-load grid data for all particles within the same block.

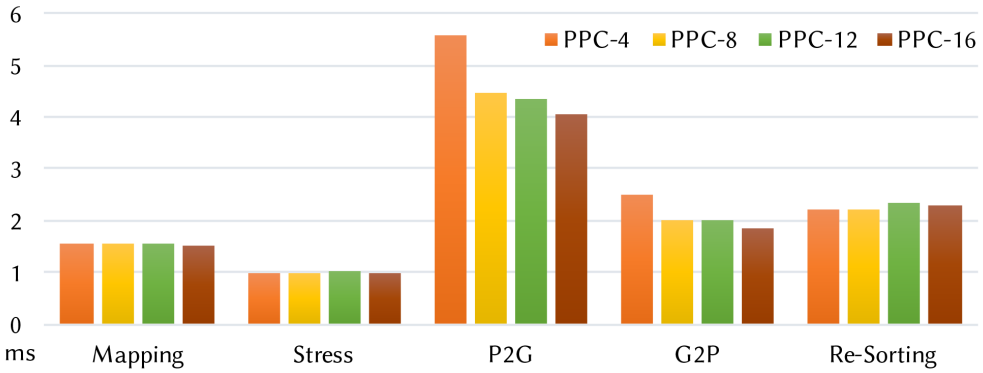


Figure 5.7: **Particle density benchmark.** Using our GPU implementation with MLS transfers, we compare the performance of each critical kernel between four cases with different numbers of particle per cell on a Nvidia Quadro P6000. The total number of particles is approximately fixed as 3.5M. The stress kernel also includes the SVD computation.

Particle density benchmark

In this subsection, while fixing the total number of particles, we run the benchmarks for cases with different particle densities (particles per cell, PPC). And we start to also include all other critical kernels in addition to the transfer kernels; all tests are with MLS transfers. As shown in Fig. 5.7, it is reasonable to observe that when the particle density increases, the transfer kernels take less time to finish, since the higher PPC renders a smaller sparse grid structure. For the other kernels, which are mostly particle-oriented, i.e. the underlying grid structure does not really interfere with them, the impacts of the varying PPC seem to be negligible.

5.4.2 Gaussian Particle Distribution Benchmarks

To further examine the impacts of non-uniform particle distributions, we also run some benchmarks in which the particle-per-cell varies based on a Gaussian distribution. All tests are with MLS. We use the same box domain from $(\frac{1}{8}, \frac{1}{8}, \frac{1}{8})$ to $(\frac{7}{8}, \frac{7}{8}, \frac{7}{8})$. For the two Gaussian distributions, the minimum particle-per-cell are 4 while the maximums are 16 and 32; while the corresponding uniform cases are with particle-per-cell being 10 and 18 respectively. As shown in Fig. 5.8, the performance are almost identical, proving that our scheme is not affected by the particle distribution when the background sparse grid remains the same.

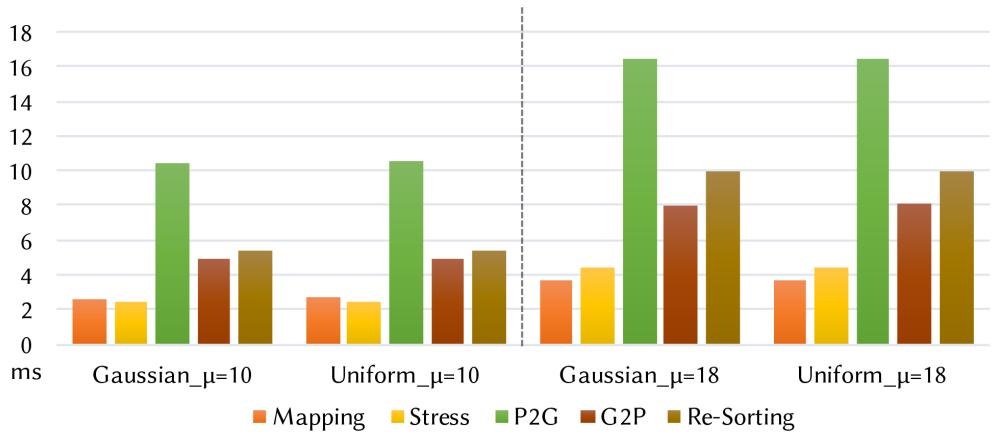


Figure 5.8: **Gaussian benchmark.** We compare the performance of each critical kernel when the particle-per-cell distributions following Gaussian distributions and uniform distributions on a Nvidia Quadro P6000. The transfer scheme is MLS. The stress kernel also includes the SVD computation.

5.4.3 Implicit Iteration and SVD

We adopt the matrix-free Krylov solver for the implicit step in which the multiplication of the system matrix and a vector can be expressed by concatenating a G2P transfer and a P2G transfer (cf. [Stomakhin et al., 2013] for more details). Notice those two transfer kernels are not the same as the ones used in the explicit MPM solver. We name them as P2G-Implicit and G2P-Implicit kernels as in Fig. 5.9. Both FLIP and APIC (non-MLS) have to compute weight gradients while MLS approximates weight gradients with weights. From the right half of Fig. 5.9, the G2P-MLS is slightly slower than G2P-non-MLS because G2P-MLS needs to do additional 9 multiplications due to the extra term $(\mathbf{x}_p - \mathbf{x}_i)$ at all 27 nodes.

We also consider the possibility of precomputing and storing the results of SVD at the beginning of one time step. Whenever the Stress kernel or Stress-derivative kernel needs, we simply load the SVD results from memory. It is interesting to notice that, for Stress kernel, it is faster to simply re-compute SVD repeatedly; while for Stress-derivative kernel, there is only negligible difference. The main reason for this discrepant behavior in the two kernels is that, the computing workload in Stress kernel is already lighter than the memory workload, loading more data in can significantly impede the performance. On the other hand, Stress-derivative has enough computing workload to mitigate the memory cost for loading SVD results.

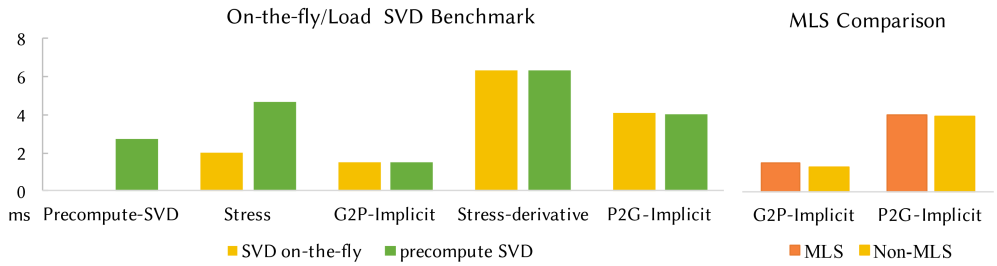


Figure 5.9: **On-the-flyload SVD benchmark and MLS comparison.** *Left:* when we pre-compute SVD and store the results, the stress and stress-derivative kernels load the SVD results directly from the memory; otherwise they recompute SVD on-the-fly whenever necessary; *Right:* we compare the performance for one implicit iteration of MLS and non-MLS implicit integrations.

In the same CPU used in Sec. 5.4.1, a AVX512 SVD implementation of [McAdams et al., 2011] takes about 2.3 ns per particle; while our GPU implementation takes about 0.37 ns.

5.4.4 Solver benchmark

We compare our explicit solver with the matrix-free implicit Krylov solver by simulating the collision between two elastic dragons and list the performance in Fig. 5.10. The Δt of them are set to be 1×10^{-4} and 1×10^{-3} respectively. Obviously, implicit solver spends less time to converge but also can use larger time step.

5.5 MPM heat solver with MLS shape functions

MPM can be generalized to derive an implicit scheme for solving the heat equation. We follow Hu et al. [2018] in deriving a moving least squares (MLS) weak form. The resulting algorithm allows us to accurately capture heat conduction in virtual materials and enables us to thermo-mechanical phenomena such as melting.

Stomakhin et al. [2014] also investigated thermo-mechanical effects in the context of MPM. Their formulation is based on a staggered-grid finite difference discretization which requires heuristic boundary cell labeling. Our method, in contrast, naturally enforces the zero Neumann boundary condition (insulated at the free surface) when no surface heat flux is specified. This boundary condition is analogous to the zero traction boundary condition in discretizing the momentum

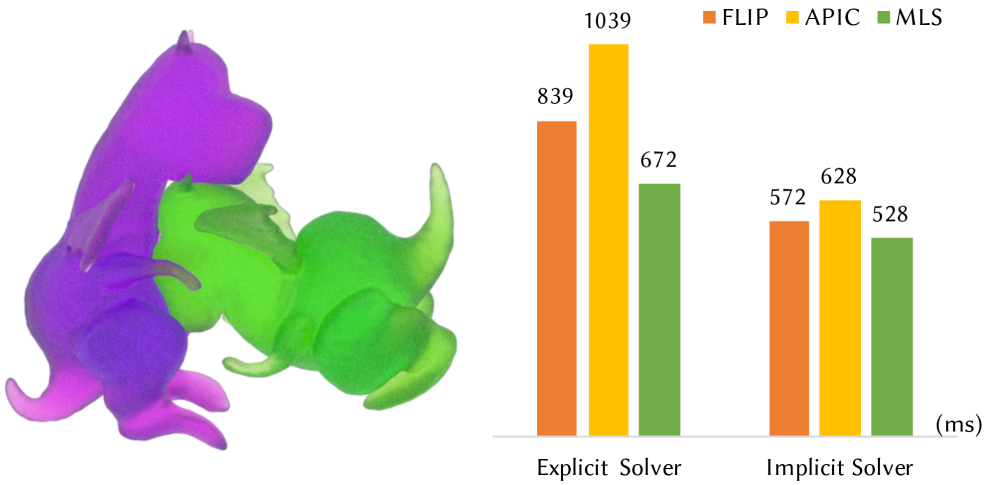


Figure 5.10: **Solver benchmark.** We compare the performance of the two dragons colliding example (left) with our GPU explicit and implicit solver along with three different schemes (right). The total number of particles is 775K and the grid resolution is 256^3. The Δt for them are 1×10^{-4} and 1×10^{-3} respectively.

equation with MPM. Performing a weak form consistent discretization allows us to treat particles as mass-full quadrature points. Consequently, unlike Stomakhin et al. [2014], we do not need to transfer particle-wise heat capacity or conductivity to the grid. We also maintain a consistent discretization function space for both the momentum and heat equations.

Additionally, our method shares the same collocated MPM grid and transfer kernels as the ones used for solving the momentum equation. The existing optimization strategies for velocity and force transfers on the GPUs directly apply to temperature and "thermal force" transfers with negligible modifications.

5.5.1 Continuous Equation

We start from the Eulerian-form heat equation

$$\rho c \frac{D\theta}{Dt} - \nabla \cdot \kappa \nabla \theta + q^{ext} = 0,$$

where $\theta(\mathbf{x}, t)$ is temperature, $\rho(\mathbf{x}, t)$ is density, $c(\mathbf{x}, t)$ is Eulerian specific heat capacity (with its Lagrangian counter part $C(\mathbf{X}, t)$ and unit $J/(kg \cdot K)$), $\kappa(\mathbf{x}, t)$ is heat conductivity, q^{ext} encodes any

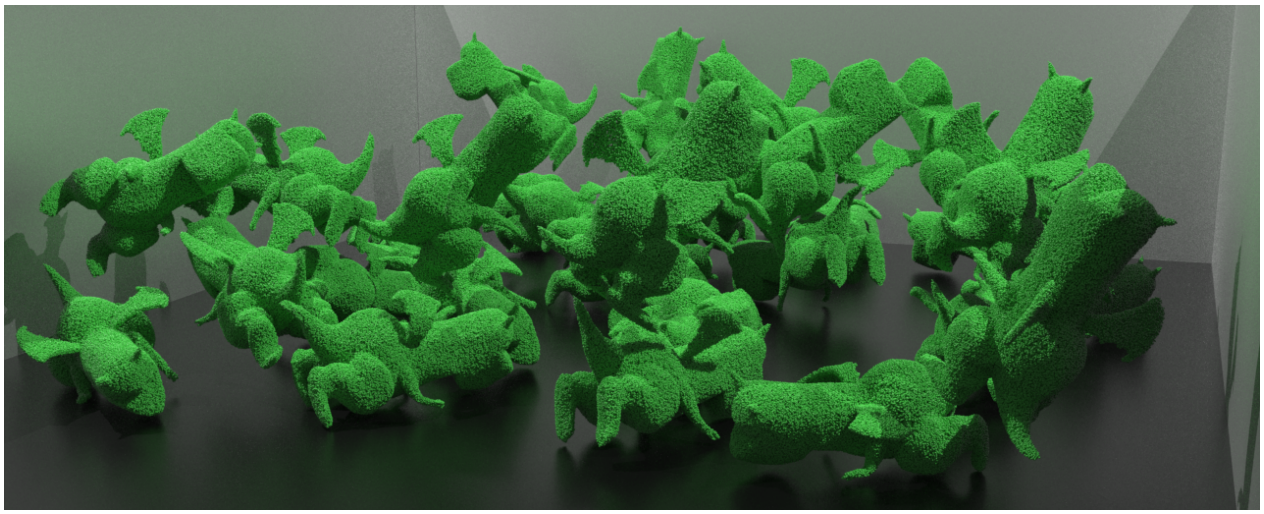


Figure 5.11: **How to collide your dragon.** Two arrays of elastic dragons colliding with each other.

external body heat source such as radiation.

5.5.2 MLS Discretization

In deriving the weak form of the heat equation, we follow the discretization strategy of Hu et al. [2018] closely. The backward Euler discretization (from t^0 to t^1) of the weak form of the heat equation is given by

$$\frac{\hat{\mathcal{M}}_i^0(\theta_i^1 - \theta_i^0)}{\Delta t} = \mathbf{q}^1 + \mathbf{y}, \tag{5.1}$$

where $\mathbf{q}^1(\mathbf{x}) = \int_{\partial\Omega^0} \left(\Phi_i \kappa^0(\mathbf{x}) \theta_j^1 \nabla\Phi_j\right) \cdot \mathbf{n}\, ds(\mathbf{x})$ encodes the heat flux Neumann boundary condition,

$$\mathbf{y}(\theta^1) = -\int_{\Omega^0} \nabla\Phi_i \cdot \left(\kappa^0(\mathbf{x}) \theta_j^1 \nabla\Phi_j\right)\, d\mathbf{x}$$

is the implicit "thermal force". $\hat{\mathcal{M}}_i^0 = \sum_p m_p C_p N_i(\mathbf{x}_p)$ is the lumped thermal mass, with $N_i(\mathbf{x})$ being the quadratic B-spline interpolation function. MLS shape functions $\Phi_i(\mathbf{x})$ are used to reconstruct a function space near each particle. The integration domain is further expressed as a summation over particle domains, where each integral over a particle domain is approximated using one point

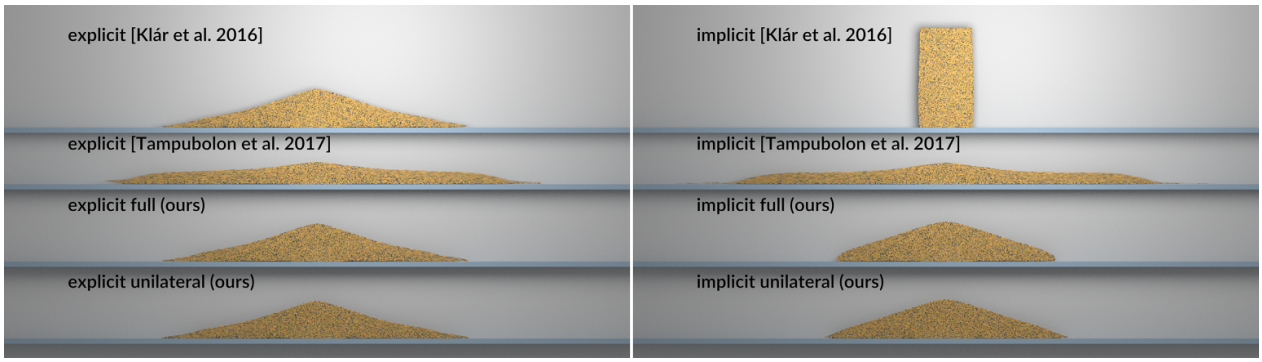


Figure 5.12: **New sand constitutive model** The left and right columns depict explicit and semi-implicit simulations respectively. Semi-implicit scheme with the regular St. Venant-Kirchhoff constitutive model used in [Klár et al., 2016] introduces a severe numerical viscosity, while the modification proposed by Tampubolon et al. [2017] introduces spreading effect and non-physical column-collapse profile. Our proposed energy density functions mitigate both of these shortcomings.

quadrature. The resulting formulation for an insulated body with no boundary heat flux is then

$$\frac{1}{\Delta t}(\theta_i^1 - \theta_i^0)\hat{\mathcal{M}}_i^0 = -\sum_j \left(\sum_p \kappa_p V_p^0 \nabla \Phi_i(\mathbf{x}_p) \cdot \nabla \Phi_j(\mathbf{x}_p) \right) \theta_j^1, \tag{5.2}$$

where θ_i is temperature of node i, C_p is the specific heat capacity of particle p, and κ_p is the heat conductivity of particle p.

After the temperature increment is solved on the grid, we transfer it back to the particles during the grid-to-particles step. A more detailed explanation of the discretization step is provided in Appendix C.

5.6 Constitutive models

5.6.1 Temperature Dependent Elasticity

The elastic response of the simulated material is modeled in the isotropic hyperelasticity framework. In this context, the energy density function is a function of the singular values of the deformation gradient whose SVD is given by $\mathbf{F} = \mathbf{U}\hat{\mathbf{F}}\mathbf{V}^T$, with $\hat{\mathbf{F}} = \text{diag}\{\hat{f}_0, \hat{f}_1, \hat{f}_2\}$. For temperature dependent physical model, we adopt the fixed corotated energy density function as proposed in [Stomakhin

et al., 2012]:

$$\hat{\psi}(\hat{\mathbf{F}}) = \mu(\theta) \sum_{i=0}^{d-1} (\hat{f}_i - 1)^2 + \frac{\lambda(\theta)}{2} (\det(\hat{\mathbf{F}}) - 1)^2. \tag{5.3}$$

Here, d represents the number of spatial dimensions. We use interpolated grid temperature values at particle locations instead of θ_p on particles to drive the phase change. This prevents the influence of the ringing instability [Jiang et al., 2015, 2017b], i.e., particle temperature modes that are invisible to the integrator due to the null space in the particles-to-grid transfer operator.

We additionally apply a numerical RPIC damping [Jiang et al., 2017a; Gao et al., 2018] to achieve the look of viscous flow in the fluid phase.

5.6.2 Stabilizing Shear Compliant Particles

When a phase change occurs, the shear modulus μ is set to 0. In this case the energy density only penalizes volumetric change without penalizing shearing. This process has been shown to cause the entries of $\mathbf{F}$ to grow unbounded quickly (even close to the square root of `FLT_MAX`) while its determinant stays close to 1. Floating point accuracy is correspondingly drastically affected and floating point overflow can easily get triggered. One solution is to use an equation of states constitutive model which only depends on the update of the determinant of $\mathbf{F}$ as in [Tampubolon et al., 2017]. Instead, we propose a simple solution by projecting $\mathbf{F}$ to the hydrostatic axis and setting the diagonal entries of it to be $J^{1/d}$. This corresponds to a plasticity return mapping that absorbs the isochoric part of the elastic deformation gradient into the plastic part while only keeping its dilational part. This strategy improves the numerical stability of our algorithm significantly.

5.6.3 Unilateral Model for Cohesionless Granular Material

We simulate cohesionless sand as an elastoplastic material. We propose a *full* quartic model whose energy density function is given by

$$\hat{\psi}(\hat{\mathbf{F}}) = \alpha\mu \sum_{i=0}^{d-1} (\log(\hat{f}_i))^4 + \frac{\alpha\lambda}{2} \big(\mathrm{tr}(\log(\hat{\mathbf{F}}))\big)^4. \tag{5.4}$$

Figure 5.13: **How to granulate your dragon.** Granulated dragons fall on elastic ones. This simulation contains 6.7 million particles on a 512^3 grid at an average 39.4 seconds per 48Hz frame.

The coefficient α is approximately 6.254 and is obtained by minimizing the L^2-norm of the difference between the quartic function with the original logarithm function over the interval $[0.25, 1]$.

The above model gives visually pleasing results for the explicit time integration scheme. Semi-implicit scheme, where plasticity is treated as a post-process after the elastic response is resolved implicitly, has been shown to suffer numerical cohesion problem [Klár et al., 2016]. We propose a *unilateral* version of the quartic energy density function which mitigates this problem and improves the visual result of [Tampubolon et al., 2017], namely

$$\begin{aligned}\hat{\psi}(\hat{\mathbf{F}}) = \alpha\mu \sum_{i=0}^{d-1} (\log(\hat{f}_i))^4 H_{\{\log(\hat{f}_i)<0\}}(\log(\hat{f}_i)) + \\ \frac{\alpha\lambda}{2}\left(\mathrm{tr}(\log(\hat{\mathbf{F}}))\right)^4 H_{\{\mathrm{tr}(\log(\hat{\mathbf{F}}))<0\}}(\mathrm{tr}(\log(\hat{\mathbf{F}}))),\end{aligned} \tag{5.5}$$

where H symbolizes the indicator function.

We use the Drucker-Prager plasticity yield function with a rigorously derived return mapping algorithm corresponding to the quartic model. Unlike the model in [Tampubolon et al., 2017], our unilateral elasticity does not require additional parameter tuning and much more closely

Table 5.1: **Average time per frame.** All timings are in seconds and frame rate is 48.

	Particles #	Domain	Δt	Mapping	Stress	P2G	Solver*	Solver w/ Heat*	G2P	Sorting	Others	Total (s)
Dragon Cup	9.0M	512^3	1×10^{-4}	0.64	0.57	2.30	14.87	-	1.00	1.35	1.15	**21.88**
Granulation	6.7M	512^3	2.5×10^{-4}	0.26	1.34	1.88	34.50	-	0.74	0.41	0.32	**39.45**
Gelatin	6.9M	512^3	1×10^{-3}	0.08	0.05	0.26	5.95	-	0.10	0.26	0.02	**6.72**
Melting	4.2M	256^3	8×10^{-3}	0.02	0.01	0.07	-	10.32	0.02	0.03	0.01	**10.48**

** Implicit solvers were used for all tests in this table.*

approximate the visual behavior of the original model from [Klár et al., 2016]. The details of our algorithm is explained in Appendix C.

5.7 More results

In addition to the benchmarks, we also demonstrate the efficiency of our GPU implementation and the efficacy of our new heat discretization and the semi-definite sand model with several simulation demos. We list the performance and the parameters used in these simulations in Table 5.1. Note that all particles are sampled using Poisson Disk [Bridson, 2007] for uniform coverage.

In Fig. 5.2, eighteen elastic dragons are stacked together in a glass to generate interesting dynamics; and in Fig. 5.11, two arrays of dragon-shaped jellies are dropped to the ground. We can also simulate a Gelatin jello bouncing off another larger one in Fig. 5.3.

Lava is poured to a cool elastic dragon in Fig. 5.1. Our heat solver is capable of accurately capturing the process of heat transport and phase change. As the temperature of certain parts of the dragon increases, the dragon liquefies. Finally, we demonstrate in Fig. 5.13 that our new granular material model manages to produce visually pleasing dynamics with a semi-implicit solver.

5.7.1 Memory Footprint

Time performance is not the only concern of our implementation. The memory consumption should also be dealt with appropriately, especially in high-resolution animations. In our simulator, the memory budget is partitioned into three categories.

Particle This part of the memory is used to store all particles' attributes. Its size grows linearly with the particle count and the number of attributes in each particle. It also depends on the type of

each particle as different materials store different parameters.

Grid This part of the memory is used for the sparse grid structure, where the actual amount of memory in use has a linear correlation with the number of occupied GSPGrid blocks. The worst case degenerates to a uniform grid. In our test cases, we mostly set the total amount of memory allocated to be 60% to that of the uniform grid.

Auxiliary Logically speaking, this part consists of two blocks of memory. Since we use spatial hashing for particle sorting and block topology construction, we need a hash table to perform the task. Its capacity also shares a linear correlation with the number of cells (64× that of sparse GSPGrid blocks). To reduce hash collision conflicts as well as saving memory, the coefficient is set to 64 as a compromise. Note that the key value in our MPM pipeline is a 64-bit integer. The other trunk of memory works as the storage for intermediate computation, including the ordering of particle positions, velocities, deformation gradient $\mathbf{F}$, etc. Its size is basically equivalent to that of the original particle data. In our actual implementation, since the values in hash table are no longer in use once its job is finished, we use the memory trunk to perform spatial hashing rather than allocating another hash table.

As a result, up to 2.4GB is spent on the cube example with 7M particles, 900K cells and 17K GSPGrid blocks, of which the majority is particle-related, while the memory budget for the dragon collision example with 95K particles, 6K cells and 500 GSPGrid blocks is only 70MB.

5.8 Limitations and future work

A number of the design choices that led to our demonstrated performance gains also carry some associated limitations that we consciously commit to. Our decision to mimic the design of the CPU-oriented SPGrid data structure in our GPU counterpart allows for implementations on the respective platforms to use similar semantics and maximize code reuse. However, the explicit use of the virtual memory system in the CPU version of SPGrid allows for computational kernels to be implemented (at reasonable, albeit not fully optimal efficiency) with computations performed at per-node granularity or, more realistically, SIMD-line granularity; for example, accessing a stencil

neighbor of any individual grid node can be done at a reasonable cost, without any set-up overhead. On the GPU, however, such computations can only reach high efficiency if performed at a larger scale, e.g. at block granularity, since the overhead of fetching the neighboring blocks of the one being processed needs to be borne for each kernel invocation. This is not a prohibitive limitation, as performing computations at block granularity is by-and-large a necessity for efficiency for any similar GPU kernel.

In addition, for a CPU implementation of SPGrid, accessing a grid node that has not been referenced before is an operation that can be done without any requisite setup or pre-processing (any page-faults that might occur are handled transparently). On the GPU, however, our need to explicitly allocate all active blocks necessitates that the set of all active indices be fully known before their data storage can be allocated and accessed. Once again, we regard this as a reasonable limitation, since the frequency at which the topology of the computational domain changes is small relative to the computational cost of operating on such data during MPM simulation.

Finally, GPU MPM simulations are still limited by the smaller amount of on-board memory, and it would be an interesting investigation to explore multi-GPU methods, or heterogeneous implementations to circumvent the size limitation.

While our optimization strategies greatly utilize computational resources on the GPU, high fidelity MPM simulations are still far from being real-time. This is largely due to the strict CFL restriction on time step sizes especially in high resolution. It would be interesting future work to further combine additional algorithmic acceleration of MPM time stepping with our GPU framework. We would also explore possibilities with spatially adaptive GSPGrid following Gao et al. [2017] for superior performance.

6 DISCUSSION

In both industry and academia of the graphics community, in-depth exploitations of the present computational resources to promote the performance of animation algorithms or pipelines have been attracting more and more attention. SPGrid, a sparse grid storage scheme, is one of the most recent explorations in this direction, which is capable of squeezing the maximum boost from the computing potential provided in operating systems and hardware. As presented in [Setaluri et al., 2014], SPGrid achieves to reduce the cost per degree of freedom on an adaptive grid to be comparable to the cost on a uniform grid. In this thesis, the application of SPGrid has been extended from pure Cartesian grid based fluid simulations to particle-grid hybrid techniques. It has been verified that, equipped with modern parallel techniques, SPGrid or its GPU adaptation can be used as the scratch-pad in material point method to accelerate the whole pipeline. A future investigation would be to further introduce SPGrid to non-simulation scenarios, e.g. rendering, geometric modeling, etc.

6.1 General observations

In addition to the findings reported in the individual chapters of this dissertation, my journey through the challenges in these separate (but occasionally crossing) research threads revealed certain recurring findings and observations that are worth to mention, in the interest of perspective.

Many elements of my work built upon earlier papers that introduced something new in the feature set of dynamics simulation. In some cases, compromises are made knowingly, either in performance or generality, in support of these new features. Sometimes, when enough attempts meet the same set of challenges, and accept the same set of compromises, an image is painted that such compromise or limitation is the normal cost of supporting some new capability. For example, many works that count on adaptivity for improving the performance of simulations rely mostly on the reduction of the number of degrees of freedom as the way to gain performance. Often times, an impression is formed that the overhead of using adaptivity instead of a more regular approach is necessary and unavoidable. I have found this argument to be more complex to answer fully and honestly. There are many instances that reducing the number of degrees of freedom in a simulation by, say, 20x-50x would barely result in a performance improvement by more than 2x-3x because (i)

the time steps needed in a dynamic simulation become smaller, (ii) data structures and algorithms become less efficient, and (iii) parallelism becomes complicated, not necessarily because an effective parallel version is impossible, but often tricky to implement.

If I can extract one conclusion from my experience, it would not be that prior work was not accurate, or truthful in saying that such conflicts exist. I did observe, however, that the things that one can sacrifice to get the best of features and efficiency come in more different types that one would think. One possibility is to consider how broad or narrow our solutions can be. If we are willing to focus our solutions to a specific type of physics-based simulation, and a more narrow range of simulated phenomena, some opportunities are presented about how to recover good performance while not disallowing important features. For this reason, it can be said that many of the solutions presented in this work are specific to MPM simulation, for example. However, the ever-expanding reach of MPM methods as seen in the activity in this area would make us hopeful that we can still reach a sizeable spectrum of applications.

Another type of generality that we might have to compromise is the ability to map our algorithms to any parallel, throughput-optimized computing platform, among those that are available today or in the future. My work has included optimizations that used low-level programming on GPUs (e.g. CUDA intrinsics), that are not guaranteed to have an equivalent on CPUs, or even future generations of GPUs themselves. I hope that the spirit of some of these optimizations would still be applicable, although the implementation might be different.

Finally, there is an important question that remains: How easy or practical would it be for a graphics programmer to reproduce the optimizations in this work, or implement similar optimizations for a slightly different problem. I would expect that somebody with the necessary background and experience would be in a better position to do that after reading the work that my colleagues and myself have produced, but I have to be honest that this is not an easy task. In many cases, significant performance optimizations were made possible not just by re-arranging code, but by re-thinking the entire algorithm, the discretization, and its properties. I do believe that being open to change the theory, the numerical algorithm, and the software engineering is worthwhile, but it also requires more diverse skills and knowledge from the implementer than what might be practical in some environments. I hope that, either by using software libraries that hide the complexity of

low-level operations from implementers, or by some well-designed Domain Specific Language for this purpose, this task would become more accessible to a larger user base.

6.2 Future work

Rendering Many rendering techniques utilize hierarchical grids for accelerations. Stencil operations, e.g. the computation of transmission in volume rendering, are crucial to the overall efficiency. SPGrid specifically optimizes stencil access, making it also suitable for fast rendering applications. However, large resolution rendering tasks always require the support of out-of-core processing which is now not implemented in SPGrid. Unlike other sparse data structures, SPGrid relies on the virtual memory system to allocate physical memory. As a result, a memory page continues to be occupied unless the user explicitly informs the operating system to free it.

Temporal adaptivity. While spatial adaptivity has been widely studied, temporal adaptivity nearly remains to be a new research topic. In both Chapter 2 and Chapter 3, although we can refine the grid at the regions of interest, the whole domain is still running with the same Δt, which is determined by the finest resolution. In temporally adaptive simulations, one can imagine that "temporal" T-junctions may also exist to mess with the final visual results and require appropriate treatments. Fang et al. [2018] recently propose a way to apply different local time steps to regions with different stiffness to save the total computations. An even more intimidating case would be to integrate both spatial and temporal adaptivity - the finer regions can evolve with smaller Δt.

Geometric skinning with self-collision support. Physics-based skinning methods are attractive due to their ability to resolve object-collisions and self-collisions; however, the high-valence computing expenses prevent them from being widely adopted in industry, especially in real-time applications. On the other hand, geometric skinning methods always have enviable computing performance. Considering that MPM automatically enforces non-slip contact, one interesting future work would be to equip geometric skinning methods with MPM particles to help handle collisions.

A APPENDIX: POWER DIAGRAM TOPOLOGY (CH. 2)

Lemma A.1. *No face in the power diagram can arise between two cells in the octree that only shared a vertex.*

Proof. Consider the points p_1 and p_2 at the centers of two power cells C_1 and C_2. From the definition of the power diagram, the plane $\mathcal{P}$ between them satisfies the equation,

$$d_1^2 - r_1^2 = d_2^2 - r_2^2 \tag{A.1}$$

where $d_1 = |p - p_1|$ and $d_2 = |p - p_2|$ are the distances of an arbitrary point p, and r_1, r_2 are the radii of the circumspheres S_1, S_2 for the power cells C_1, C_2. $\mathcal{P}$ is the *secant plane* that passes through the intersection circle of S_1 and S_2. The primal-dual orthogonality property of power diagrams ensures that p_1p_2 is perpendicular to $\mathcal{P}$, let p_0 be the intersection point. It follows that when a face exists between C_1 and C_2 in the power diagram, then $|p_0 - p_1| < r_1$ and $|p_0 - p_2| < r_2$.

Now assume that the octree cells O_1, O_2 centered at the points p_1, p_2 only shared a vertex q. When the radius of each power cell is $\Delta x/\sqrt{3}$ (or $\Delta x/\sqrt{2}$ in 2D), then q lies on both S_1 and S_2. However, $|p_1 - q| = r_1$ and $|p_2 - q| = r_2$, so q also satisfies equation (A.1), implying that it lies on the plane $\mathcal{P}$. It follows that the plane $\mathcal{P}$ is *tangent* to both S_1 and S_2. Thus, there is no face between C_1 and C_2 in the power diagram. □

Note that the proof for Lemma A.1 does not assume any grading restrictions on the octree, suggesting that this property holds in general.

Numerical validation

We now show the numerical convergence of our discretization on some analytic problems. Consider an analytic pressure field satisfying $p = x^2 + y^2 - r^2$ and a level set field $\phi = \sqrt{x^2 + y^2} - r$ in the domain $[-0.5, 0.5] \times [-0.5, 0.5]$, where $r = 0.25$. Regions inside the level set contain pressure degree of freedom, while those outside serve as Dirichlet boundary conditions. Our quadtree has two levels

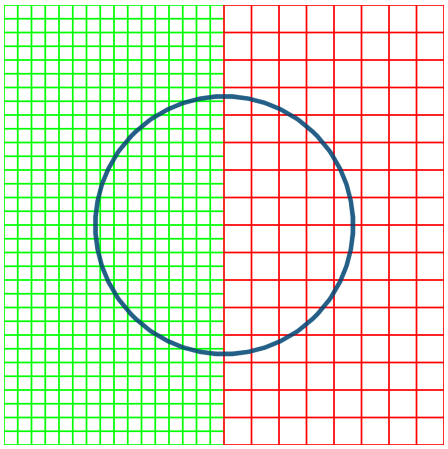


Figure A.1: Computational domain for the two dimensional Poisson problem

Effective resolution	32^2	64^2	128^2	256^2	512^2
L_∞ error	0.000753682	0.000207968	$5.58272e^{-5}$	$1.44169e^{-5}$	$3.68431e^{-6}$
Order of accuracy	–	1.86	1.90	1.95	1.97

Table A.1: Convergence results for the two dimensional Poisson problem

Effective resolution	32^3	64^3	128^3	256^3
L_∞ error	0.00115148	0.00030693	$8.1405e^{-5}$	$2.15657e^{-5}$
Order of accuracy	–	1.91	1.91	1.92

Table A.2: Convergence results for the three dimensional Poisson problem

Effective resolution	32^2	64^2	128^2	256^2	512^2
L_∞ error	0.0159432	0.0100688	0.00494522	0.0024499	0.00121924
Order of accuracy	–	0.58	1.02	1.01	1.01

Table A.3: Convergence results for our fast marching scheme in two dimensions

of adaptivity, with fine resolution on one side and coarse on the other (see Figure A.1). Table A.1 shows the convergence results from our discretization. We similarly consider an analytic pressure field $p = x^2 + y^2 + z^2 - r^2$ and a level set field $\phi = \sqrt{x^2 + y^2 + z^2} - r$ in three dimensions. Table A.2 shows the convergence results. As can be seen, our discretization achieves second order accuracy. We also evaluated the order of accuracy of our hybrid fast marching scheme. Table A.3 shows the convergence behavior of our method in the two dimensional setting of Figure A.1, while Table A.4 shows the corresponding behavior in three dimensions.

Effective resolution	32^3	64^3	128^3	256^3	512^3
L_∞ error	0.0227256	0.0138875	0.00731823	0.00342631	0.00187874
Order of accuracy	–	0.71	0.92	1.09	0.87

Table A.4: Convergence results for our fast marching scheme in three dimensions

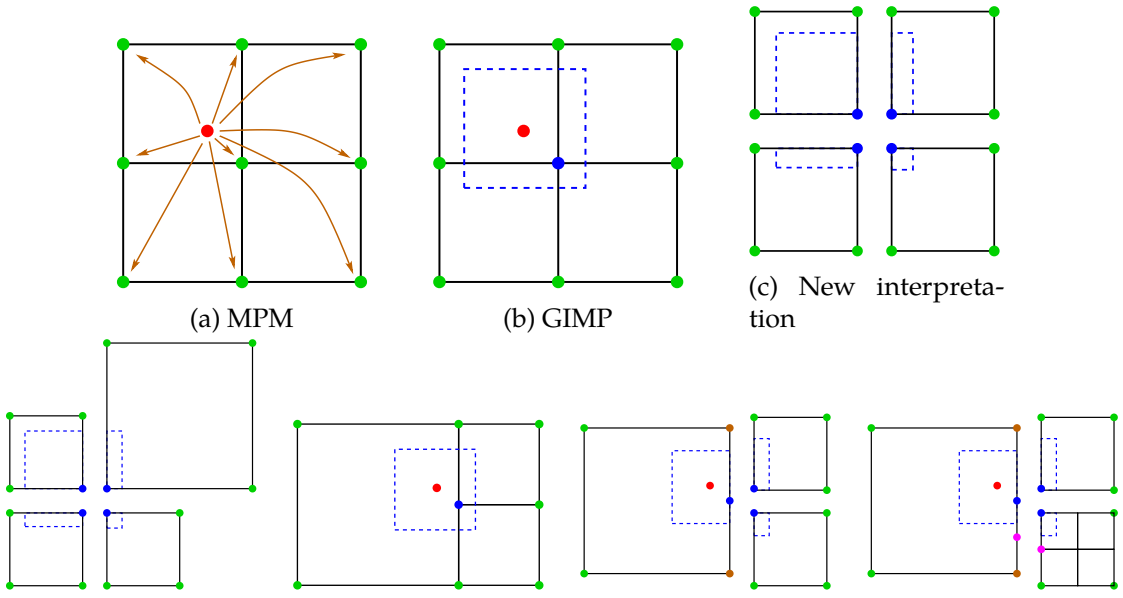
B APPENDIX: PROPERTIES OF ADAPTIVE GIMP SHAPE FUNCTIONS (CH. 3)

B.1 From MPM to adaptive GIMP

Since the introduction of the material point method to the graphics community, researchers tend to focus on defining the weighting function as a direct relation between a particle and the grid nodes in its immediate vicinity, e.g. a quadratic B-spline function, as shown in Figure B.1a. GIMP is another alternative which downgrades the C^1 requirement to C^0 and then uses a convolution to resume the C^1 continuity. We assign an axis aligned range to each particle as in Figure B.1b and the general form for the weight is

$$w_{ip} = \frac{1}{V_p} \int_{\Omega} \chi_p(\mathbf{x}) N_i(\mathbf{x}) d\mathbf{x} \tag{B.1}$$

where V_p is the volume of the dashed blue box Ω, χ_p is usually an indicator function, and $N_i(\mathbf{x})$ is the basis function (e.g. a simple hat function) .



(a) MPM (b) GIMP (c) New interpretation

(d) Corner node (e) T-junction node (f) GIMP T-junction (g) Non-graded case

Figure B.1: From MPM to Adaptive GIMP

The conventional GIMP is simply to do an integration for each affected grid node. However, we propose a new interpretation which is quite essential for all the following procedures. We divide the particle range into regions which intersect with each single grid cell as $\Omega_j = \Omega \cap C_j$ and then $\Omega = \sum_j \Omega_j$. So B.1 becomes

$$w_{ip} = \frac{1}{V_p} \sum_{\Omega_j} \int_{\Omega_j} N_i(\mathbf{x}) d\mathbf{x} \tag{B.2}$$

For example, the weight of the center grid node in Figure B.1c is now the sum of the contributions computed from all four cells. When we switch to quadtrees, the corner nodes are easier to handle as in Figure B.1d, and the same interpretation works in exactly the same way except that the basis function used in the right top cell is different.

For dealing with T-junction nodes (blue node in Figure B.1e), we take the classical constrained hanging node treatment from octree FEM. The blue node is initially regarded as a real degree of freedom (DOF) in both smaller cells (Figure B.1f), thus contributions from those two cells can be computed. As to the coarser cell, the T-junction is constrained and its property is completely determined by its parents (brown nodes). Instead of doing any weight computations in the coarser cell, we just need to redistribute the contributions collected from finer cells to its parents.

This new approach of computing weights for adaptive grids fulfills all the aforementioned requirements, and it especially enforces both partition of unity and C^1 continuity at the T-junctions, which are proved in the next sections. Furthermore, there is no principal or practical restrictions for its extension to non-graded trees. However, one obvious concern is that for dealing with one T-junction node, weight computations can go across several levels (with different basis functions, as in Figure B.1g). And the corresponding data retrieval from several levels can soon jeopardize the memory bandwidth. For this performance related issue, we propose an equivalent way to convert all cross-level computations into uniform kernels, and the details are presented in Chapter 3.

Lemma B.1. ***Partition of unity*** *: given a particle, the weights of all grid nodes sum up to one* $\sum_i w_i = 1$

Proof. Consider the case in Figure B.2, the T-junction (blue dot H) is constrained by its parents G and C. In this proof, we concern about the sum of the weights (not the continuity), thus regarding H as a real DOF (in cell 2 and 3 only) can ease the reasoning.

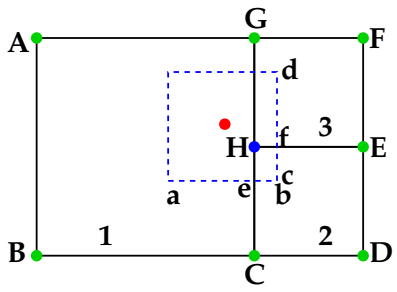


Figure B.2: **Partition of unity.**

Taking the view of GIMP, the weight can be written as

$$w_i = \frac{1}{V}\int_a^b \int_c^d N_i \, dydx \tag{B.3}$$

where $V = (b-a) \times (d-c)$ is the area of the particle range. Furthermore, we can divide the particle range into the intersections with each cell:

$$\begin{aligned}
w_i &= w_i^1 + w_i^2 + w_i^3 && \text{(B.4)}\\
w_i^1 &= \frac{1}{V}\int_a^e \int_c^d N_i^1 \, dydx && \text{(B.5)}\\
w_i^2 &= \frac{1}{V}\int_e^b \int_c^f N_i^2 \, dydx && \text{(B.6)}\\
w_i^3 &= \frac{1}{V}\int_e^b \int_f^d N_i^3 \, dydx && \text{(B.7)}
\end{aligned}$$

In this way, each weight w_i can be splitted into three of them as $w_i^{1,2,3}$. It is clear that for some nodes, not all the three are non-zeros. For example, both w_A^2 and w_A^3 are zeros. We collect the splitted non-zero weights of cell 1 as

$$\begin{aligned}
\sum_i w_i^1 &= w_A^1 + w_B^1 + w_C^1 + w_G^1 && \text{(B.8)}\\
&= \frac{1}{V}\int_a^e \int_c^d (N_A^1 + N_B^1 + N_C^1 + N_G^1) \, dydx && \text{(B.9)}\\
&= \frac{1}{V}\int_a^e \int_c^d 1 \, dydx && \text{(B.10)}\\
&= \frac{(e-a) \times (d-c)}{V} && \text{(B.11)}
\end{aligned}$$

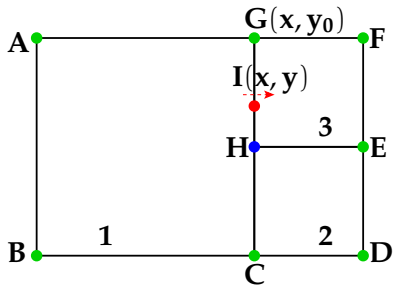


Figure B.3: **Continuity.**

where we used the identity $N_A^1 + N_B^1 + N_C^1 + N_G^1 = 1$ because all are standard hat functions. Similarly, we can get $\sum_i w_i^2 = \frac{(b-e)\times(f-c)}{V}$ and $\sum_i w_i^3 = \frac{(b-e)\times(d-f)}{V}$. Thus

$$\sum_i w_i = \frac{(e-a)\times(d-c)}{V} + \frac{(b-e)\times(f-c)}{V} + \frac{(b-e)\times(d-f)}{V} \tag{B.12}$$

$$= \frac{(b-a)\times(d-c)}{V} = 1 \tag{B.13}$$

□

Lemma B.2. C^0 ***continuity****: the shape function defined in Chapter 3 is* C^0 *continuous.*

Proof. In this section, we denote a modified basis function associated with grid node i by the notation $\hat{N}_i$. Consider the case in Figure B.3, a particle (at position I) is moving across the interface edge GH between cell 1 and cell 3. The T-junction (blue dot H) is constrained by its parents G and C, thus it is not a real DOF anymore, and the weight from it should be redistributed to its parents. Let us focus on node G, since all others are either the same or much easier cases.

When the particle is still in cell 1, the shape function can be computed :

$$\hat{N}_G^{1,x} = 1 \tag{B.14}$$

$$\hat{N}_G^{1,y} = 1 - \frac{y - y_0}{2\Delta x} \tag{B.15}$$

where Δx is the cell size of finer cells. In this scenario, the shape function in the x-direction is always 1, we just ignore it for now.

After particle moves into cell 3, the shape function should be updated :

$$N_G^{3,y} = 1 - \frac{y - y_0}{\Delta x}. \tag{B.16}$$

Notice that the node H should also be computed and then redistributed to its parent as

$$N_H^{3,y} = \frac{y - y_0}{\Delta x} \tag{B.17}$$

$$\hat{N}_G^{3,y} = N_G^{3,y} + \frac{1}{2} N_H^{3,y} \tag{B.18}$$

$$= 1 - \frac{y - y_0}{\Delta x} + \frac{1}{2} \frac{y - y_0}{\Delta x} \tag{B.19}$$

$$= 1 - \frac{y - y_0}{2\Delta x} \tag{B.20}$$

Thus $N_G^{1,y} = \hat{N}_G^{3,y}$, i.e. the shape function of node G is continuous when particle moves across the interface edge. Eventually the weight function can acquire C^1 continuity after computing the convolution of the shape function. □

C APPENDIX: MPM-BASED HEAT SOLVER AND UNILATERAL SAND CONSTITUTIVE MODEL (CH. 5)

C.1 MLS-MPM implicit heat solver

C.1.1 Weak Form

MPM is a general spatial discretization scheme (just like FEM) and can be used to solve the heat equation. A straightforward application in computer graphics is melting and freezing, where mechanical material parameters depend on temperature. A more complex case is thermo-elastoviscoplasticity, where thermal expansion and frictional heating effects are taken into account through thero-mechanical coupled equations. Here we only focus on deriving the weak form discretization on MLS-MPM [Hu et al., 2018] for the heat conduction equation.

Let's start from the Eulerian-form heat equation

$$\rho C \frac{D\theta}{Dt} - \nabla \cdot \kappa \nabla \theta + q^{ext}(\mathbf{x}) = 0,$$

where $\theta(\mathbf{x}, t)$ is temperature, $\rho(\mathbf{x}, t)$ is density, $c(\mathbf{x}, t)$ is Eulerian specific heat capacity per unit mass (with its Lagrangian counter part $C(\mathbf{X}, t)$), $\kappa(\mathbf{x}, t)$ is heat conductivity, q^{ext} is external body heat source (let's ignore this term, this one is similar to gravity in the momentum equation and can encode effects such as radiation). The Eulerian weak form of this PDE is

$$\int_{\Omega^t} \rho c \frac{D\theta}{Dt} - \nabla \cdot \kappa \nabla \theta \, d\mathbf{x} = 0.$$

We follow the same way of discretizing the momentum equation using a Galerkin style weak form [Jiang et al., 2016]. Commonly in MPM, we take an updated Lagrangian view and look at t^n. Now our temperature θ is like the velocity in the momentum equation. It relates to its Lagrangian

counter-part Θ as $\theta^{n+1}(\mathbf{x}) = \Theta(\Phi^{-1}(\mathbf{x}, t^n), t^{n+1})$ and $\theta^n(\mathbf{x}) = \Theta(\Phi^{-1}(\mathbf{x}, t^n), t^n)$.

C.1.2 MLS Shape Functions

For any test function $w(\mathbf{x}, t^n)$ in the proper function space, the time discretization reveals

$$\begin{aligned}&\frac{1}{\Delta t}\int_{\Omega^{t^n}} w(\mathbf{x}, t^n)\rho(\mathbf{x}, t^n)c(\mathbf{x}, t^n)\left(\theta^{n+1}(\mathbf{x}) - \theta^n(\mathbf{x})\right) d\mathbf{x}\\ &= \int_{\partial\Omega^{t^n}} (w(\mathbf{x}, t^n)\kappa(\mathbf{x}, t^n)\nabla\theta(\mathbf{x}, t^n)) \cdot \mathbf{n} ds(\mathbf{x}) - \int_{\Omega^{t^n}} \nabla w(\mathbf{x}, t^n) \cdot (\kappa(\mathbf{x}, t^n)\nabla\theta(\mathbf{x}, t^n))\, d\mathbf{x}.\end{aligned}$$

The next step is MLS style spacial discretization. Following [Hu et al., 2018], we adopt the MLS shape function $\Phi_i(\mathbf{x})$ at each node near a particle to discretize both the test function and the physical fields. That is, we do

$$w^n = w_i^n \Phi_i, \;\; \theta^n = \theta_j^n \Phi_j, \;\; \theta^{n+1} = \theta_j^{n+1} \Phi_j. \tag{C.1}$$

We refer to [Hu et al., 2018] for the construction of the MLS shape functions in a way that is consistent with Element Free Galerkin (EFG) methods.

Then

$$\begin{aligned}&\frac{1}{\Delta t}\int_{\Omega^{t^n}} w_i^n \Phi_i \rho(\mathbf{x}, t^n)c(\mathbf{x}, t^n)\theta_j^{n+1}\Phi_j d\mathbf{x} - \frac{1}{\Delta t}\int_{\Omega^{t^n}} w_i^n \Phi_i \rho(\mathbf{x}, t^n)c(\mathbf{x}, t^n)\theta_j^n \Phi_j d\mathbf{x}\\ &= \int_{\partial\Omega^{t^n}} \left(w_i^n \Phi_i \kappa(\mathbf{x}, t^n)\theta_j^n \nabla\Phi_j\right) \cdot \mathbf{n} ds(\mathbf{x}) - \int_{\Omega^{t^n}} w_i^n \nabla\Phi_i \cdot \left(\kappa(\mathbf{x}, t^n)\theta_j^n \nabla\Phi_j\right) d\mathbf{x}.\end{aligned}$$

Utilizing the MLS shape functions avoids differentiating B-spline kernels in high dimensions. More specifically, if a linear polynomial space with quadratic B-spline weighting is chosen for the MLS reconstruction, we have [Hu et al., 2018]

$$\nabla\Phi_i(\mathbf{x}_p) = D_p^{-1} N_i(\mathbf{x}_p^n)(\mathbf{x}_i - \mathbf{x}_p^n),$$

where $D_p = \frac{1}{4}\Delta x^2$ for quadratic B-spline weighting in $N_i(\mathbf{x})$.

C.1.3 Lumped Mass

Similarly to how mass matrix was definened in the momentum case, we can define a "thermal mass" matrix

$$\mathcal{M}^n_{ij} = \int_{\Omega^{t^n}} \Phi_i(\mathbf{x})\rho(\mathbf{x},t^n)c(\mathbf{x},t^n)\Phi_j(\mathbf{x})d\mathbf{x}$$

and rewrite the equation as

$$\begin{aligned}&\frac{1}{\Delta t}w_i^n\theta_j^{n+1}\mathcal{M}^n_{ij} - \frac{1}{\Delta t}w_i^n\theta_j^n\mathcal{M}^n_{ij}\\ &= \int_{\partial\Omega^{t^n}}\left(w_i^n\Phi_i\kappa(\mathbf{x},t^n)\theta_j^n\nabla\Phi_j\right)\cdot\mathbf{n}ds(\mathbf{x}) - \int_{\Omega^{t^n}} w_i^n\nabla\Phi_i\cdot\left(\kappa(\mathbf{x},t^n)\theta_j^n\nabla\Phi_j\right)d\mathbf{x}.\end{aligned} \tag{C.2}$$

Note that

$$\mathcal{M}^n_{ij} = \int_{\Omega^{t^n}} \Phi_i(\mathbf{x})\rho(\mathbf{x},t^n)c(\mathbf{x},t^n)\Phi_j(\mathbf{x})d\mathbf{x}$$

Pull back from the Eulerian frame to the Lagrangian frame..

$$= \int_{\Omega^0} \Phi_i(\mathbf{x}(\mathbf{X}))R(\mathbf{X},t)C(\mathbf{X},t^n)\Phi_j(\mathbf{x}(\mathbf{X}))J(\mathbf{X},t)d\mathbf{X}$$

$$= \int_{\Omega^0} \Phi_i(\mathbf{x}(\mathbf{X}))R(\mathbf{X},0)C(\mathbf{X},t^n)\Phi_j(\mathbf{x}(\mathbf{X}))d\mathbf{X}$$

Adopting one point quadrature over particle domains..

$$\approx \sum_p m_p C_p \Phi_i(\mathbf{x}_p)\Phi_j(\mathbf{x}_p)$$

Now in Equation C.2 we choose

$$w_i^n = \begin{cases} 1, & \mathbf{i} = \hat{\mathbf{i}} \\ 0, & \text{otherwise}\end{cases}$$

then

$$\sum_j \frac{1}{\Delta t}(\theta_j^{n+1} - \theta_j^n)\mathcal{M}^n_{\hat{i}j} = \int_{\partial\Omega^{t^n}}\left(\Phi_{\hat{i}}\kappa(\mathbf{x},t^n)\theta_j^n\nabla\Phi_j\right)\cdot\mathbf{n}ds(\mathbf{x}) - \int_{\Omega^{t^n}}\nabla\Phi_{\hat{i}}\cdot\left(\kappa(\mathbf{x},t^n)\theta_j^n\nabla\Phi_j\right)d\mathbf{x}. \tag{C.3}$$

This is the equation for node $\hat{i}$.

Inverting a full mass matrix is not possible in MPM due to its potential singularity. A common strategy is diagonal mass lumping, meaning replacing $\mathcal{M}$ with its diagonal row-sums. Let's call each new entry $\hat{\mathcal{M}}_i^n$, then

$$\begin{aligned}
\hat{\mathcal{M}}_i^n &= \sum_j \mathcal{M}_{ij}^n \\
&= \sum_j \int_{\Omega^{t^n}} \Phi_i(\mathbf{x}) \rho(\mathbf{x}, t^n) c(\mathbf{x}, t^n) \Phi_j(\mathbf{x}) d\mathbf{x} \\
&= \int_{\Omega^{t^n}} \Phi_i(\mathbf{x}) \rho(\mathbf{x}, t^n) c(\mathbf{x}, t^n) d\mathbf{x} \\
&\text{Pull back..} \\
&= \int_{\Omega^0} \Phi_i(\mathbf{x}(\mathbf{X})) R(\mathbf{X}, 0) C(\mathbf{X}, t^n) d\mathbf{X} \\
&\approx \sum_p m_p C_p \Phi_i(\mathbf{x}_p).
\end{aligned}$$

Now we use $\mathcal{M}_{\hat{i}j}^n \approx \hat{\mathcal{M}}_{\hat{i}}^n \delta_{\hat{i}j}$ to rewrite Equation C.3:

$$\frac{1}{\Delta t}(\theta_{\hat{i}}^{n+1} - \theta_{\hat{i}}^n)\hat{\mathcal{M}}_{\hat{i}}^n = \int_{\partial\Omega^{t^n}} \left(\Phi_{\hat{i}} \kappa(\mathbf{x}, t^n) \theta_j^n \nabla\Phi_j\right) \cdot \mathbf{n} ds(\mathbf{x}) - \int_{\Omega^{t^n}} \nabla\Phi_{\hat{i}} \cdot \left(\kappa(\mathbf{x}, t^n) \theta_j^n \nabla\Phi_j\right) d\mathbf{x}. \quad \text{(C.4)}$$

Let's replace $\hat{i}$ with i to get the final equation for the temperature on node i:

$$\frac{1}{\Delta t}(\theta_i^{n+1} - \theta_i^n)\hat{\mathcal{M}}_i^n = \int_{\partial\Omega^{t^n}} \left(\Phi_i \kappa(\mathbf{x}, t^n) \theta_j^n \nabla\Phi_j\right) \cdot \mathbf{n} ds(\mathbf{x}) - \int_{\Omega^{t^n}} \nabla\Phi_i \cdot \left(\kappa(\mathbf{x}, t^n) \theta_j^n \nabla\Phi_j\right) d\mathbf{x}. \quad \text{(C.5)}$$

Note that mC is called thermal mass and is fixed on each particle.

C.1.4 Heat Force

Now let's deal with the right hand side. Let's look the second one (volume integral):

$$\int_{\Omega^{t^n}} \nabla\Phi_i \cdot \left(\kappa(\mathbf{x}, t^n) \theta_j^n \nabla\Phi_j\right) d\mathbf{x} \approx \sum_p \left(\nabla\Phi_i(\mathbf{x}_p)\right) \cdot \left(\kappa(\mathbf{x}_p, t^n) \theta_j^n \nabla\Phi_j(\mathbf{x}_p)\right) V_p^n.$$

This is the 'internal force' for heat equation. Note that to go implicit we need to change the known θ_j^n to the unknown θ_j. We have

$$f_i(\theta) = \sum_p \left(\nabla\Phi_i(\mathbf{x}_p)\right) \cdot \left(\kappa_p \theta_j \nabla\Phi_j(\mathbf{x}_p)\right) V_p^n \tag{C.6}$$

$$= \sum_j \sum_p \left(\nabla\Phi_i(\mathbf{x}_p)\right) \cdot \left(\kappa_p \theta_j \nabla\Phi_j(\mathbf{x}_p)\right) V_p^n \tag{C.7}$$

$$= \sum_j \sum_p \kappa_p V_p^n \nabla\Phi_i(\mathbf{x}_p) \cdot \nabla\Phi_j(\mathbf{x}_p)\theta_j \tag{C.8}$$

$$= \sum_j H_{ij}\theta_j \tag{C.9}$$

where $H_{ij} = \sum_p \kappa_p V_p^n \nabla\Phi_i(\mathbf{x}_p) \cdot \nabla\Phi_j(\mathbf{x}_p)$. We can see the force is linear in θ. It is easy to verify that if we use a linear basis, and one particle per cell center, then this is equivalent to finite difference Laplacian operator.

The boundary term

$$h_i = \int_{\partial\Omega^{t^n}} \left(\Phi_i(\mathbf{x})\kappa(\mathbf{x}, t^n)\theta_j^n \nabla\Phi_j\right) \cdot \mathbf{n} ds(\mathbf{x})$$

is the heat flux boundary condition. To apply this, quantity $\eta(\mathbf{x}, t^n) = (\kappa(\mathbf{x}, t^n)\nabla\theta) \cdot \mathbf{n}$ is specified at the boundary. This is analygoous to the traction boudnary conditon in MPM.

C.1.5 Summarization

Finally, let's summarize the implicit formulation of heat equation assuming no boundary heat flux:

$$\frac{1}{\Delta t}(\theta_i - \theta_i^n)\hat{\mathcal{M}}_i^n = -\sum_j \left(\sum_p \kappa_p V_p^n \nabla\Phi_i(\mathbf{x}_p) \cdot \nabla\Phi_j(\mathbf{x}_p)\right)\theta_j, \tag{C.10}$$

where θ_i is temperature of node i, $\hat{\mathcal{M}}_i^n = \sum_p m_p C_p \Phi_i(\mathbf{x}_p)$ is lumped thermal mass, C_p is specific heat capacity per unit mass, κ_p is heat conductivity. The resulting implicit system is SPD and can be efficiently solved with Conjugate Gradient on the GPU.

C.2 Unilateral sand constitutive model

C.2.1 Notations

The letter $d \in \{2, 3\}$ denotes the spatial dimension of the problem. We use $\mathbf{F}$ to denote the deformation gradient of the flow-map $\boldsymbol{\varphi}$. For isotropic constitutive model, it is useful to work with the singular value decomposition of $\mathbf{F} = \mathbf{U}\hat{\mathbf{F}}\mathbf{V}^T$ [Bonet and Wood, 2008]. We write

$$\hat{\mathbf{F}} = \begin{pmatrix} \hat{f}_0 & 0 \\ 0 & \hat{f}_1 \end{pmatrix} \quad (\text{in 2D}), \quad \text{or} \quad \hat{\mathbf{F}} = \begin{pmatrix} \hat{f}_0 & 0 & 0 \\ 0 & \hat{f}_1 & 0 \\ 0 & 0 & \hat{f}_2 \end{pmatrix} \quad (\text{in 3D}). \tag{C.11}$$

Sometimes, it's convenient to use an abuse of notation and to write $\hat{\mathbf{F}}$ as a vector

$$\hat{\mathbf{F}} = \begin{pmatrix} \hat{f}_0 \\ \hat{f}_1 \end{pmatrix} \quad (\text{in 2D}), \quad \text{or} \quad \hat{\mathbf{F}} = \begin{pmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \hat{f}_2 \end{pmatrix} \quad (\text{in 3D}). \tag{C.12}$$

Mast [2013] and Klár et al. [2016] have shown the usefulness of Hencky-strain in modeling granular material. The Hencky-strain $\mathbf{h}$ is the logarithm of the singular values of $\mathbf{F}$, i.e.

$$\hat{\mathbf{h}} = \log(\hat{\mathbf{F}}). \tag{C.13}$$

C.2.2 Energy density function

The original St. Venant-Kirchhoff with Hencky strain model is [Klár et al., 2016]

$$\hat{\psi}(\hat{\mathbf{F}}) = \mu \text{tr}\Big((\log \hat{\mathbf{F}})^2\Big) + \frac{\lambda}{2}\big(\text{tr}(\log \hat{\mathbf{F}})\big)^2. \tag{C.14}$$

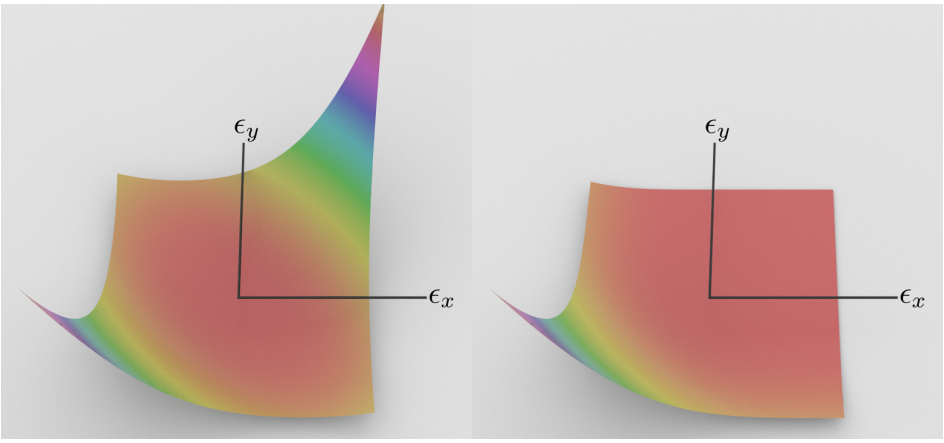


Figure C.1: **Full v.s. unilateral.** Two-dimensional depiction of the full and unilateral quartic energy density function in the Hencky-strain space.

In lieu of the original model, we propose a *quartic* energy density function

$$\hat{\psi}(\hat{\mathbf{F}}) = a\mu \sum_{i=0}^{d-1} (\log(\hat{f}_i))^4 + \frac{a\lambda}{2} \big(\mathrm{tr}(\log(\hat{\mathbf{F}}))\big)^4. \tag{C.15}$$

The coefficient a is chosen to be 6.254421582537118, which is the solution of the following minimization problem

$$a = \mathrm{argmin}_a \int_{0.25}^{1} \left(ax^4 - \log(x)^2\right)^2 \, dx. \tag{C.16}$$

In order to mitigate numerical cohesion that occurs in a semi-implicit time integration scheme (where the elastic response is computed using an implicit time-integration scheme and plasticity is done as a post-process), we also consider the unilateral version of (C.15), namely

$$\hat{\psi}(\hat{\mathbf{F}}) = a\mu \sum_{i=0}^{d-1} (\log(\hat{f}_i))^4 H_{\{\log(\hat{f}_i)<0\}}(\log(\hat{f}_i)) + \frac{a\lambda}{2} \big(\mathrm{tr}(\log(\hat{\mathbf{F}}))\big)^4 H_{\{\mathrm{tr}(\log(\hat{\mathbf{F}}))<0\}}(\mathrm{tr}(\log(\hat{\mathbf{F}}))), \tag{C.17}$$

which is twice continuously differentiable. The function H_I denotes an indicator function, i.e. given a set $I \subseteq \mathbb{R}$, we have

$$H_I(x) = \begin{cases} 1 & \text{if } x \in I \\ 0 & \text{otherwise.} \end{cases}$$

C.2.3 Stress-strain relationship

Let $\mathbf{P}$ be the first Piola-Kirchhoff stress tensor. All of the models above are isotropic, and as such $\mathbf{P}$ can be written as

$$\mathbf{P} = \mathbf{U}\hat{\mathbf{P}}\mathbf{V}^\top, \tag{C.18}$$

with

$$\hat{\mathbf{P}} = \frac{\partial \hat{\psi}}{\partial \hat{\mathbf{F}}}. \tag{C.19}$$

The Cauchy stress tensor is given by

$$\boldsymbol{\sigma} = J^{-1}\mathbf{P}\mathbf{F}^\top = J^{-1}\mathbf{U}\hat{\mathbf{P}}\mathbf{V}^\top\mathbf{V}\hat{\mathbf{F}}\mathbf{U}^\top = J^{-1}\mathbf{U}\hat{\mathbf{P}}\hat{\mathbf{F}}\mathbf{U}^\top, \tag{C.20}$$

which motivates us to define

$$\hat{\boldsymbol{\sigma}} := \hat{\mathbf{P}}\hat{\mathbf{F}}. \tag{C.21}$$

The Kirchhoff stress is $\boldsymbol{\tau} = J\boldsymbol{\sigma}$. Arguably, this is a better stress measure for our purposes since we're avoiding a division by J. From (C.20), we have

$$\boldsymbol{\tau} = \mathbf{U}\hat{\boldsymbol{\sigma}}\mathbf{U}^\top \quad \Longrightarrow \quad \hat{\boldsymbol{\tau}} = J\boldsymbol{\sigma}.$$

The stress-strain relationship for the constitutive model defined by (C.15) is

$$\hat{\boldsymbol{\sigma}} = 4\alpha\mu(\hat{\mathbf{h}})^3 + 2\alpha\lambda\,\mathrm{tr}(\hat{\mathbf{h}})^3\mathbf{1}, \tag{C.22}$$

with $\mathbf{1}$ denoting the all-ones vector and $(\hat{\mathbf{h}})^3$ is defined component-wise.

C.2.4 Yield surface

To model sand, we use the Drucker-Prager yield criterion. For completeness, we'll include its form for both the Cauchy and Kirchhoff stress

$$\hat{y}(\hat{\boldsymbol{\sigma}}) = \operatorname{tr}(\hat{\boldsymbol{\sigma}})\alpha + \left\| \hat{\boldsymbol{\sigma}} - \frac{\operatorname{tr}(\hat{\boldsymbol{\sigma}})}{3}\mathbf{I} \right\|_F \leqslant 0, \qquad \text{(Cauchy)}, \tag{C.23}$$

$$\hat{y}(\hat{\boldsymbol{\tau}}) = \operatorname{tr}(\hat{\boldsymbol{\tau}})\alpha + \left\| \hat{\boldsymbol{\tau}} - \frac{\operatorname{tr}(\hat{\boldsymbol{\tau}})}{3}\mathbf{I} \right\|_F \leqslant 0, \qquad \text{(Kirchhoff)}. \tag{C.24}$$

Combined with (C.22), we can derive the yield function in terms of the Hencky-strain $\hat{\mathbf{h}}$. We denote this relationship by

$$\bar{y}(\hat{\mathbf{h}}) \leqslant 0. \tag{C.25}$$

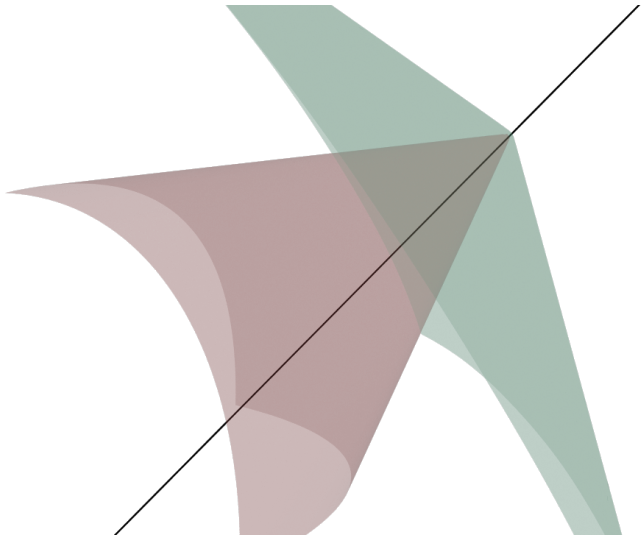


Figure C.2: **Yield surface in Hencky-strain space.** The outer green surface is the yield function corresponding to the proposed quartic energy density and the inner red cone corresponds to the regular St. Venant-Kirchhoff with Hencky strain model. The line denotes the hydrostatic axis.

C.2.5 Trial and projected strain

We adopt the following notation

	Principal strain	Hencky strain	Cauchy stress
trial state	$\hat{\mathbf{F}}^{\text{tr}}$	$\hat{\mathbf{h}}^{\text{tr}}$	$\hat{\boldsymbol{\sigma}}^{\text{tr}}$
projected state	$\hat{\mathbf{F}}$	$\hat{\mathbf{h}}$	$\hat{\boldsymbol{\sigma}}$

The relationship between the trial state and the projected state is given by

$$\hat{\mathbf{h}} = \delta\gamma\hat{\mathbf{G}}|_{\hat{\mathbf{h}}} + \hat{\mathbf{h}}^{\text{tr}}. \tag{C.26}$$

where

$$\hat{\mathbf{G}} = \text{dev}\left(\frac{\partial\hat{y}}{\partial\hat{\boldsymbol{\tau}}}\right). \tag{C.27}$$

which is completely determined by the yield surface.

C.2.6 Solving the system of equations

In the projection step, there are $d+1$-unknowns, namely $\delta\gamma$ and the projected strain $\hat{\mathbf{h}}$. Equations (C.26) and (C.25) gives a total of $(d+1)$-equations to be satisfied.

Solving the system of equations in three-dimensions

It can be proven that the solution $\mathbf{h}$ lies in the space of $\text{span}\{(1,1,1)^\top, \text{dev}(\hat{\mathbf{h}}^t)\}$. The deviatoric part of $\mathbf{h}$ is

$$\text{dev}(\hat{\mathbf{h}}^t) = \begin{pmatrix} -2\hat{h}_0 + \hat{h}_1 + \hat{h}_2 \\ \hat{h}_0 - 2\hat{h}_1 + \hat{h}_2 \\ \hat{h}_0 + \hat{h}_1 - 2\hat{h}_2 \end{pmatrix}.$$

So we can write

$$\hat{\mathbf{h}} = p\mathbf{o} + q\text{dev}(\hat{\mathbf{h}}^t). \tag{C.28}$$

We can substitute this to (C.26) and solve for p (see `mathematica_tech_doc.nb`) to get

$$p = \frac{\hat{h}_0^t + \hat{h}_1^t + \hat{h}_2^t}{3}. \tag{C.29}$$

The yield function as a function of p and q is quite long. Its form and its derivative is attached in the `mathematica` document. One can solve for the zero of this function using Newton's method.

The algorithm pseudocode is

$p \leftarrow (h_0^{tr} + h_1^{tr} + h_2^{tr})/3$

if $y(\mathbf{h}^{tr}) \leqslant 0$ **then**

return

else if $p \geqslant 0$ **then**

$\mathbf{h} \leftarrow (0, 0, 0)$

else

$\mathbf{dev} \leftarrow (-2h_0^{tr} + h_1^{tr} + h_2^{tr}, h_0^{tr} - 2h_1^{tr} + h_2^{tr}, h_0^{tr} + h_1^{tr} - 2h_2^{tr})$

$q \leftarrow -1./3.$

solve q using Newton

$\mathbf{h} \leftarrow p(1, 1, 1) + q\,\mathbf{dev}$

D APPENDIX: DISSERTATOR'S CONTRIBUTION TO CO-AUTHORED WORK

Chapter 2

Publication: M. Aanjaneya*, M. Gao* (joint first authors), H. Liu, C. Batty, E. Sifakis, "Power Diagrams and Sparse Paged Grids for High Resolution Adaptive Liquids", ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH), 2017.

Conceptualization: Jointly with Professor Sifakis, Professor Batty; 50% contribution stake among student participants.

Problem investigation/analysis: 50% contribution stake among students.

Original techniques:

- Co-developed the encoding scheme
- Co-developed the new multigrid preconditioner
- Co-developed the new fine levelset tracking scheme

Implementation/testing: 50% contribution stake among students.

- Co-implemented the final pipeline
- Implemented the tetrahedralization scheme
- Implemented the prototype of the new Poisson solver and verified its convergence
- Implemented the prototype of the new fast marching method and verified its convergence
- Implemented the particle levelset pipeline to generate comparisons

Writing: 15% of the paper and 100% of the supplemental document.

Chapter 3

Publication: M. Gao, A. Tampubolon, C. Jiang, E. Sifakis, "An Adaptive Generalized Interpolation Material Point Method for Simulating Elastoplastic Materials", ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia), 2017.

Conceptualization: Jointly with Professor Sifakis, Professor Jiang; 80% contribution stake among student participants.

Problem investigation/analysis: 80% contribution stake among students.

Original techniques:

- Co-developed the construction of the C^0 basis function
- Co-developed the application of GIMP to boost the continuity to C^1
- Co-developed the parallelization scheme to accelerate the pipeline
- Co-developed the application of SIMD to uniform particle-to-grid transfers
- Co-developed the particle resampling and grid rasterization

Implementation/testing: 80% contribution stake among students.

- Implemented the prototype and the final pipeline
- Implemented the method of Lian et al. [2015] and identified the negative weight problem

Writing: 35% of the paper and 100% of the supplemental document.

Chapter 4

Publication: M. Gao, A. Tampubolon, X. Han, Q. Guo, G. Kot, E. Sifakis, C. Jiang, "Animating Fluid Sediment Mixture in Particle-Laden Flows", ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH), 2018. (TOG Cover Image)

Conceptualization: Jointly with Professor Sifakis, Professor Jiang; 80% contribution stake among student participants.

Problem investigation/analysis: 80% contribution stake among students.

Original techniques:

- Co-developed the coupling scheme
- Co-developed the sub-stepping scheme
- Co-developed the variable coefficient multigrid preconditioner

Implementation/testing: 80% contribution stake among students.

- Implemented the prototype and the final pipeline

Writing: 40% of the paper.

Chapter 5

Publication: M. Gao*, X. Wang*, K. Wu*(joint first authors), A. Tampubolon, E. Sifakis, C. Yuksel, C. Jiang, "GPU Optimization of Material Point Method", SIGGRAPH Asia 2018 (under review).

Conceptualization: Jointly with Professor Sifakis, Professor Jiang and Professor Yuksel; 50% contribution stake among student participants.

Problem investigation/analysis: 50% contribution stake among students.

Original techniques:

- Co-developed the new particle-to-grid transfer scheme
- Co-developed the GSPGrid data structure
- Co-developed the implicit re-ordering scheme
- Co-developed the MPM-based heat solver

Implementation/testing: 50% contribution stake among students.

- Implemented the prototype and the final pipeline

Writing: 40% of the paper.

REFERENCES

Aanjaneya, M., M. Gao, H. Liu, C. Batty, and E. Sifakis. 2017. Power diagrams and sparse paged grids for high resolution adaptive liquids. *ACM Trans Graph* 36(4):140.

Adalsteinsson, David, and James A Sethian. 1995. A fast level set method for propagating interfaces. *J Comp Phys* 118(2):269–277.

Adams, B., M. Pauly, R. Keiser, and L. Guibas. 2007. Adaptively sampled particle fluids. In *Acm trans graph*, vol. 26, 48. ACM.

Andersen, Søren Mikkel, and Lars Andersen. 2007. Material-point method analysis of bending in elastic beams. In *The international conference on civil, structural and environmetal engineering computing*. Citeseer.

Anderson, T.B., and R. Jackson. 1967. Fluid mechanical description of fluidized beds. equations of motion. *Ind & Eng Chem Fund* 6(4):527–539.

Ando, R., N. Thurey, and R. Tsuruno. 2012. Preserving fluid sheets with adaptively sampled anisotropic particles. *IEEE Trans Vis Comp Graph* 18(8):1202–1214.

Ando, R., N. Thurey, and C. Wojtan. 2013. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Trans Graph* 32(4):103:1–103:10.

Ando, Ryoichi, Nils Thuerey, and Chris Wojtan. 2015. A dimension-reduced pressure solver for liquid simulations. *Computer Graphics Forum (Eurographics)* 34(2):473–480.

Aurenhammer, Franz. 1987. Power diagrams: properties, algorithms, and applications. *SIAM Journal on Computing* 16(1):78–96.

Bardenhagen, SG, and EM Kober. 2004. The generalized interpolation material point method. *Comp Mod in Eng and Sci* 5(6):477–496.

Bargteil, A., C. Wojtan, J. Hodgins, and G. Turk. 2007. A finite element method for animating large viscoplastic flow. *ACM Trans Graph* 26(3).

Bargteil, Adam W., James F. O'Brien, Tolga G. Goktekin, and John A. Strain. 2006. A semi-Lagrangian contouring method for fluid simulation. *ACM Trans. Graph.* 25(1):19–38.

Batty, C., F. Bertails, and R. Bridson. 2007. A fast variational framework for accurate solid-fluid coupling. *ACM Trans Graph* 26(3).

Batty, C., S. Xenos, and B. Houston. 2010a. Tetrahedral embedded boundary methods for accurate and flexible adaptive fluids. In *Proc of eurographics*.

Batty, Christopher, Stefan Xenos, and Ben Houston. 2010b. Tetrahedral embedded boundary methods for accurate and flexible adaptive fluids. *Computer Graphics Forum (Eurographics)* 29(2): 695–704.

Becker, M., and M. Teschner. 2007. Weakly compressible sph for free surface flows. In *Proc acm siggraph/eurograph symp comp anim*, 209–217.

Bergou, M., M. Wardetzky, S. Robinson, B. Audoly, and E. Grinspun. 2008. Discrete elastic rods. In *Acm trans graph*, vol. 27, 63. ACM.

Bojsen-Hansen, Morten, and Chris Wojtan. 2013. Liquid surface tracking with error compensation. *ACM Trans. Graph. (SIGGRAPH)* 32(4):79:1–79:10.

Bonet, J., and R. Wood. 2008. *Nonlinear continuum mechanics for finite element analysis*. Cambridge University Press.

Brackbill, J. 1988. The ringing instability in particle-in-cell calculations of low-speed flow. *J Comp Phys* 75(2):469–492.

Brackbill, J., D. Kothe, and H. Ruppel. 1988. FLIP: A low-dissipation, PIC method for fluid flow. *Comp Phys Comm* 48:25–38.

Brackbill, J., and H. Ruppel. 1986. FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *J Comp Phys* 65:314–343.

Bridson, R. 2007. Fast poisson disk sampling in arbitrary dimensions. In *Acm siggraph 2007 sketches*.

Bridson, Robert. 2008. *Fluid simulation for computer graphics*. Taylor & Francis.

Brochu, T., C. Batty, and R. Bridson. 2010. Matching fluid simulation elements to surface geometry and topology. *ACM Trans Graph* 29(4):47:1–47:9.

Brochu, Tyson, and Robert Bridson. 2009. Robust topological operations for dynamic explicit surfaces. *SIAM J. Sci. Comput.* 31(4):2472–2493.

Capell, S., S. Green, B. Curless, Tom D., and Z. Popović. 2002. A multiresolution framework for dynamic deformations. In *Proc of the 2002 acm siggraph/eurographics symp on comp anim*, 41–47. ACM.

Chentanez, N., and M. Muller. 2011. Real-time eulerian water simulation using a restricted tall cell grid. *ACM Trans Graph* 30(4):82:1–82:10.

Chentanez, Nuttapong, Bryan E. Feldman, François Labelle, James F. O'Brien, and Jonathan Richard Shewchuk. 2007. Liquid simulation on lattice-based tetrahedral meshes. In *Symposium on computer animation*, 219–228. ACM Press.

Chentanez, Nuttapong, Tolga G Goktekin, Bryan E Feldman, and James F O'Brien. 2006. Simultaneous coupling of fluids and deformable bodies. In *Proc of the 2006 acm siggraph/eurographics symposium on computer animation*, 83–89. Eurographics Association.

Chentanez, Nuttapong, and Matthias Müller. 2012. Mass-conserving eulerian liquid simulation. In *Proc of sca*, 245–254. Eurographics Association.

Chiang, W-F., M. DeLisi, T. Hummel, T. Prete, K. Tew, M. Hall, P. Wallstedt, and J. Guilkey. 2009. Gpu acceleration of the generalized interpolation material point method. *Symp App Accel High Perf Comp*.

Chu, J., N. B. Zafar, and X. Yang. 2017. A schur complement preconditioner for scalable parallel fluid simulation. *ACM Trans Graph* 36(5):163:1–163:11.

Clausen, Pascal, Martin Wicke, Jonathan Richard Shewchuk, and James F. O'Brien. 2013. Simulating liquids and solid-liquid interactions with Lagrangian meshes. *ACM Trans. Graph.* 32(2):17.

Cohen, J. M., S. Tariq, and S. Green. 2010. Interactive fluid-particle simulation using translating eulerian grids. In *Proc of the 2010 acm siggraph symp on interactive 3d graph and games*, 15–22. ACM.

Daphalapurkar, N., H. Lu, D. Coker, and R. Komanduri. 2007. Simulation of dynamic crack growth using the generalized interpolation material point (gimp) method. *Int J Fract* 143(1):79–102.

Daviet, G. 2016. Modeling and simulating complex materials subject to frictional contact. Ph.D. thesis, Universite Grenoble Alpes.

Daviet, Gilles, and Florence Bertails-Descoubes. 2016. A semi-implicit material point method for the continuum simulation of granular materials. *ACM Trans Graph* 35(4):102:1–102:13.

Debunne, G., M. Desbrun, A. Barr, and M-P. Cani. 1999. Interactive multiresolution anim of deformable models. In *Comp anim and simulationâĽ™99*, 133–144. Springer.

Desbrun, Mathieu, and Marie-Paule Gascuel. 1996. Smoothed particles: a new paradigm for animating highly deformable bodies. In *Eurographics workshop on computer animation and simulation*, 61–76.

Di Felice, R. 1994. The voidage function for fluid-particle interaction systems. *Int J Mult Flow* 20(1): 153–159.

Dick, C., J. Georgii, and R. Westermann. 2011. A hexahedral multigrid approach for simulating cuts in deformable objects. *IEEE Trans on Vis and Comp Graph* 17(11):1663–1675.

Dick, Christian, Marcus Rogowsky, and Rüdiger Westermann. 2016. Solving the fluid pressure Poisson equation using multigrid - evaluation and improvements. *IEEE TVCG* 22(11):2480–2492.

Dong, Y., and J. Grabe. 2018. Large scale parallelisation of the material point method with multiple gpus. *Comp and Geo* 101:149–158.

Dong, Y., D. Wang, and M. F. Randolph. 2015. A gpu parallel computing strategy for the material point method. *Comp and Geo* 66:31–38.

Dunatunga, S., and K. Kamrin. 2015. Continuum modelling and simulation of granular flows through their many phases. *J Fluid Mech* 779:483–513.

Elcott, Sharif, and Peter Schröder. 2006. Building your own DEC at home. In *Acm siggraph course notes*, 55–59.

Elcott, Sharif, Yiying Tong, Eva Kanso, Peter Schröder, and Mathieu Desbrun. 2007. Stable, circulation-preserving, simplicial fluids. *ACM Trans. Graph.* 26(1):4.

English, R. E., L. Qiu, Y. Yu, and R. Fedkiw. 2013a. Chimera grids for water simulation. In *Proc of the 12th acm siggraph/eurographics symp on comp anim*, 85–94. ACM.

English, Robert Elliot, Linhai Qiu, Yue Yu, and Ronald Fedkiw. 2013b. An adaptive discretization of incompressible flow using a multitude of moving Cartesian grids. *J. Comp. Phys.* 254:107–154.

Enright, Doug, Duc Nguyen, Frédéric Gibou, and Ron Fedkiw. 2003. Using the particle level set method and a second order accurate pressure boundary condition for free surface flows. In *Proceedings of the 4th asme-jsme joint fluids engineering conference*, 337–342. ASME.

Enright, Douglas, Stephen Marschner, and Ronald Fedkiw. 2002. Animation and rendering of complex water surfaces. In *Acm transactions on graphics (tog)*, vol. 21, 736–744. ACM.

Fang, Yu, Yuanming Hu, Shi-Min Hu, and Chenfanfu Jiang. 2018. A temporally adaptive material point method with regional time stepping. In *Proceedings of the 2018 acm siggraph/eurographics symposium on computer animation*. Eurographics Association.

Fedkiw, R., J. Stam, and H.W. Jensen. 2001. Visual simulation of smoke. In *Proc of the 28th ann conf on comp graph and int techs*, 15–22. ACM.

Fei, Y.R., H.T. Maia, C. Batty, C. Zheng, and E. Grinspun. 2017. A multi-scale model for simulating liquid-hair interactions. *ACM Trans Graph* 36(4):56.

Feldman, Bryan E, James F O'brien, and Bryan M Klingner. 2005. Animating gases with hybrid meshes. In *Acm trans graph*, vol. 24, 904–909. ACM.

Ferstl, F., R. Ando, C. Wojtan, R. Westermann, and N. Thuerey. 2016. Narrow band flip for liquid simulations. In *Comp graph forum*, vol. 35, 225–232. Wiley Online Library.

Ferstl, Florian, Rüdiger Westermann, and Christian Dick. 2014. Large-scale liquid simulation on adaptive octree grids. *IEEE TVCG* 20(10):1405–1417.

Foster, N., and D. Metaxas. 1996. Realistic animation of liquids. *Graph Mod Imag Proc* 58:471–483.

Foster, Nick, and Ronald Fedkiw. 2001. Practical animation of liquids. In *Proceedings of the 28th annual conference on computer graphics and interactive techniques*, 23–30. SIGGRAPH '01.

Fries, T., A. Byfut, A. Alizada, K. Cheng, and A. Schröder. 2011. Hanging nodes and xfem. *Int J Numer Meth Eng* 86(4-5):404–430.

Gao, M., A. P. Tampubolon, C. Jiang, and E. Sifakis. 2017. An adaptive generalized interpolation material point method for simulating elastoplastic materials. *ACM Trans Graph* 36(6).

Gao, M., A.P. Tampubolon, X. Han, Q Guo, G. Kot, E. Sifakis, and C. Jiang. 2018. Animating fluid sediment mixture in particle-laden flows. *ACM Trans Graph* 37(4).

Gibou, F., R.P. Fedkiw, L.T. Cheng, and M. Kang. 2002. A second-order-accurate symmetric discretization of the poisson equation on irregular domains. *J Comp Phys* 176(1):205–227.

Gingold, Robert A, and Joseph J Monaghan. 1977. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society* 181(3):375–389.

de Goes, Fernando, Corentin Wallez, Jin Huang, Dmitry Pavlov, and Mathieu Desbrun. 2015. Power particles: an incompressible fluid solver based on power diagrams. *ACM Trans. Graph. (SIGGRAPH)* 34(4):50.

Golas, Abhinav, Rahul Narain, Jason Sewall, Pavel Krajcevski, Pradeep Dubey, and Ming Lin. 2012. Large-scale fluid simulation using velocity-vorticity domain decomposition. *ACM Trans. Graph. (SIGGRAPH Asia)* 31(6):148.

Goldade, Ryan, Christopher Batty, and Chris Wojtan. 2016. A practical method for high-resolution embedded liquid surfaces. *Computer Graphics Forum (Eurographics)* 35(2):233–242.

Govindaraju, N. K., I. Kabul, M. C. Lin, and D. Manocha. 2007. Fast continuous collision detection among deformable models using graphics processors. *Comp Graph* 31(1):5–14.

Govindaraju, N. K., D. Knott, N. Jain, I. Kabul, R. Tamstorf, R. Gayle, M. C. Lin, and D. Manocha. 2005. Interactive collision detection between deformable models using chromatic decomposition. *ACM Trans Graph* 24(3):991–999.

Grinspun, E., P. Krysl, and P. Schröder. 2002. Charms: a simple framework for adaptive simulation. *ACM Trans on Graph (TOG)* 21(3):281–290.

Guendelman, E., A. Selle, F. Losasso, and R. Fedkiw. 2005. Coupling water and smoke to thin deformable and rigid shells. *ACM Trans Graph* 24(3):973–981.

Guittet, Arthur, Mathieu Lepilliez, Sebastian Tanguy, and Frédéric Gibou. 2015a. Solving elliptic problems with discontinuities on irregular domains: the Voronoi Interface Method. *J. Comp. Phys.* 298:747–765.

Guittet, Arthur, Maxime Theillard, and Frédéric Gibou. 2015b. A stable projection method for the incompressible Navier-Stokes equations on arbitrary geometries and adaptive Quad/Octrees. *J. Comp. Phys.* 292:215–238.

Guo, Q., X. Han, C. Fu, T. Gast, R. Tamstorf, and J. Teran. 2018. A material point method for thin shells with frictional contact. *ACM Trans Graph* 37(4).

Harada, T., S. Koshizuka, and Y. Kawaguchi. 2007. Smoothed particle hydrodynamics on gpus. In *Comp graph int*, vol. 40, 63–70. SBC Petropolis.

Heo, Nambin, and Hyeong-Seok Ko. 2010. Detail-preserving fully-Eulerian interface tracking framework. *ACM Trans. Graph. (SIGGRAPH Asia)* 29(6):176:1—-176:8.

van der Hoef, M.A., M. Ye, M. van Sint Annaland, A.T. Andrews, S. Sundaresan, and J.A.M. Kuipers. 2006. Multiscale modeling of gas-fluidized beds. *Adv Chem Eng* 31:65–149.

Hoetzlein, R. K. 2016. GVDB: Raytracing sparse voxel database structures on the GPU. In *Proc of high perf graph*, 109–117. Eurographics Association.

Hong, Jeong-Mo, and Chang-Hun Kim. 2005. Discontinuous fluids. *ACM Trans. Graph. (SIGGRAPH)* 24(3):915–920.

Horvath, C., and W. Geiger. 2009. Directable, high-resolution simulation of fire on the GPU. *ACM Trans Graph* 28(3):41:1–41:8.

Houston, B., M. B. Nielsen, C. Batty, O. Nilsson, and K. Museth. 2006. Hierarchical rle level set: A compact and versatile deformable surface representation. *ACM Trans on Graph (TOG)* 25(1): 151–175.

Hu, Y., Y. Fang, Z. Ge, Z. Qu, Y. Zhu, A. Pradhana, and C. Jiang. 2018. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Trans Graph* 37(4).

Huang, P., X. Zhang, S. Ma, and H. K. Wang. 2008. Shared memory openmp parallelization of explicit mpm and its application to hypervelocity impact. *Comp Mod in Eng and Sci* 38:119–148.

Hughes, T. 2012. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation.

Irving, G., E. Guendelman, F. Losasso, and R. Fedkiw. 2006a. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. In *Acm trans on graph (tog)*, vol. 25, 805–811. ACM.

Irving, Geoffrey, Eran Guendelman, Frank Losasso, and Ronald Fedkiw. 2006b. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. *ACM Trans. Graph. (SIGGRAPH)* 25(3):805–811.

Jiang, C., T. Gast, and J. Teran. 2017a. Anisotropic elastoplasticity for cloth, knit and hair frictional contact. *ACM Trans Graph* 36(4).

Jiang, C., C. Schroeder, A. Selle, J. Teran, and A. Stomakhin. 2015. The affine particle-in-cell method. *ACM Trans Graph* 34(4):51:1–51:10.

Jiang, C., C. Schroeder, and J. Teran. 2017b. An angular momentum conserving affine-particle-in-cell method. *J Comp Phys* 338:137–164.

Jiang, Chenfanfu, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. 2016. The material point method for simulating continuum materials. In *Acm siggraph 2016 course*, 24:1–24:52.

Kaufmann, Peter, Sebastian Martin, Mario Botsch, and Markus Gross. 2009. Flexible simulation of deformable models using discontinuous galerkin fem. *Graphical Models* 71(4):153–167.

Kim, Doyub, Oh-young Song, and Hyeong-Seok Ko. 2009. Stretching and wiggling liquids. *ACM Trans. Graph. (SIGGRAPH Asia)* 28(5):120.

Kim, Theodore, and Ming C. Lin. 2007. Fast animation of lightning using an adaptive mesh. *IEEE TVCG* 12(2):390–402.

Klár, G., J. Budsberg, M. Titus, S. Jones, and K. Museth. 2017. Production ready mpm simulations. In *Acm siggraph 2017 talks*, 42:1–42:2.

Klár, G., T. Gast, A. Pradhana, C. Fu, C. Schroeder, C. Jiang, and J. Teran. 2016. Drucker-prager elastoplasticity for sand anim. *ACM Trans Graph* 35(4).

Klingner, Bryan M, Bryan E Feldman, Nuttapong Chentanez, and James F O'brien. 2006. Fluid animation with dynamic meshes. In *Acm trans graph*, vol. 25, 820–825. ACM.

Krištof, P., B. Beneš, J. Křivánek, and O. Šťava. 2009. Hydraulic erosion using smoothed particle hydrodynamics. In *Comp graph forum*, vol. 28, 219–228. Wiley Online Library.

Krivá, Zuzana, Angela Handlovičová, and Karol Mikula. 2016. Adaptive cell-centered finite volume method for diffusion equations on a consistent quadtree grid. *Advances in computational mathematics* 42(2):249–277.

Labelle, F., and J. R. Shewchuk. 2007. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. In *Acm trans on graph (tog)*, vol. 26, 57. ACM.

Lait, Jeff. 2016. Inside houdini's distributed solver system. In *Acm siggraph 2016 talks*, 42. ACM.

Legrain, G., R. Allais, and P. Cartraud. 2011. On the use of the extended finite element method with quadtree/octree meshes. *Int J Numer Meth Eng* 86(6):717–743.

Lenaerts, T., B. Adams, and P. Dutré. 2008. Porous flow in particle-based fluid simulations. *ACM Trans Graph* 27(3):49.

Lenaerts, T., and P. Dutré. 2009. Mixing fluids and granular materials. *Comp Graph Forum* 28(2): 213–218.

Lentine, Michael, Mridul Aanjaneya, and Ronald Fedkiw. 2011. Mass and momentum conservation for fluid simulation. In *Symposium on computer animation*, 91–100.

Lentine, Michael, Matthew Cong, Saket Patkar, and Ronald Fedkiw. 2012. Simulating free surface flow with very large time steps. In *Symposium on computer animation*, 107–116.

Lentine, Michael, Wen Zheng, and Ronald Fedkiw. 2010. A novel algorithm for incompressible flow using only a coarse grid projection. *ACM Trans. Graph. (SIGGRAPH)* 29(4):114.

Li, Y., J. Zhang, and L.S. Fan. 1999. Numerical simulation of gas–liquid–solid fluidization systems using a combined cfd-vof-dpm method: bubble wake behavior. *Chem Eng Sci* 54(21):5101–5107.

Lian, Y., X. Zhang, F. Zhang, and X. Cui. 2014. Tied interface grid material point method for problems with localized extreme deformation. *Int J Imp Eng* 70:50–61.

Lian, Y.P., P.F. Yang, X. Zhang, F. Zhang, Y. Liu, and P. Huang. 2015. A mesh-grading material point method and its parallelization for problems with localized extreme deformation. *Comp Meth App Mech Eng* 289:291 – 315.

Liu, Haixiang, Nathan Mitchell, Mridul Aanjaneya, and Eftychios Sifakis. 2016. A scalable schur-complement fluids solver for heterogeneous compute platforms. *ACM Transactions on Graphics (TOG)* 35(6):201.

Losasso, Frank, Ronald Fedkiw, and Stanley Osher. 2006. Spatially adaptive techniques for level set methods and incompressible flow. *Computers & Fluids* 35(10):995–1010.

Losasso, Frank, Frédéric Gibou, and Ron Fedkiw. 2004. Simulating water and smoke with an octree data structure. In *Acm transactions on graphics (tog)*, vol. 23, 457–462. ACM.

Luitjens, Justin. 2014. Faster parallel reductions on kepler.

Ma, J., H. Lu, and R. Komanduri. 2006. Structured mesh refinement in generalized interpolation material point (gimp) method for simulation of dynamic problems. *Comp Model Eng & Sci* 12(3): 213.

Ma, J., H. Lu, B. Wang, S. Roy, R. Hornung, A. Wissink, and R. Komanduri. 2005. Multiscale simulations using generalized interpolation material point (gimp) method and samrai parallel processing. *Comp Model Eng & Sci* 8(2):135–152.

Macklin, M., M. Muller, N. Chentanez, and T. Kim. 2014. nified particle physics for real-time applications. *ACM Trans Graph* 33(4):153:1–153:12.

Manninen, M., V. Taivassalo, S. Kallio, et al. 1996. On the mixture model for multiphase flow.

Manteaux, P-L., C. Wojtan, R. Narain, S. Redon, F. Faure, and M-P. Cani. 2016. Adaptive physically based models in comp graph. In *Comp graph forum*. Wiley Online Library.

Martin, David F., and Kelley L. Cartwright. 1996. Solving Poisson's equation using adaptive mesh refinement. Tech. Rep., EECS Department, University of California, Berkeley.

Mast, Carter M. 2013. Modeling landslide-induced flow interactions with structures using the material point method. Ph.D. thesis, University of Washington.

McAdams, A., A. Selle, R. Tamstorf, J. Teran, and E. Sifakis. 2011. Computing the singular value decomposition of 3×3 matrices with minimal branching and elementary floating point operations. *University of Wisconsin Madison*.

McAdams, Aleka, Eftychios Sifakis, and Joseph Teran. 2010. A parallel multigrid poisson solver for fluids simulation on large grids. In *Proceedings of the 2010 acm siggraph/eurographics symposium on computer animation*, 65–74. Eurographics Association.

Miller, Brett, Ken Museth, Devon Penney, and Nafees Bin Zafar. 2012. Cloud modeling and rendering for âŁœpuss in bootsâŁž. *ACM SIGGRAPH Talks*.

Milne, Andy, Mark McLaughlin, Rasmus Tamstorf, Alexey Stomakhin, Nicholas Burkard, Mitch Counsell, Jesus Canal, David Komorowski, and Evan Goldberg. 2016. Flesh, flab, and fascia simulation on zootopia. In *Acm siggraph 2016 talks*, 34. ACM.

Minion, Michael L. 1996. A projection method for locally refined grids. *J. Comp. Phys.* 127(1): 158–178.

Misztal, Marek, and Andreas Bærentzen. 2012. Topology-adaptive interface tracking using the deformable simplicial complex. *ACM Trans. Graph.* 31(3):24.

Misztal, Marek, Robert Bridson, Kenny Erleben, Andreas Bærentzen, and François Anton. 2010. Optimization-based fluid simulation on unstructured meshes. In *Vriphys*, 11–20.

Molemaker, J., J. M. Cohen, S. Patel, and J. Noh. 2008. Low viscosity flow simulations for animation. In *Proc of sca*, 9–18. Eurographics Association.

Mullen, Patrick, Pooran Memari, Fernando de Goes, and Mathieu Desbrun. 2011. HOT: Hodge-optimized triangulations. *ACM Trans. Graph. (SIGGRAPH)* 30(4):103.

Müller, M., B. Heidelberger, M. Hennix, and J. Ratcliff. 2007. Position based dynamics. *J Vis Comm Imag Repre* 18(2):109–118.

Müller, M., B. Solenthaler, R. Keiser, and M. Gross. 2005. Particle-based fluid-fluid interaction. In *Proc of the 2005 acm siggraph/eurograp symp comp anim*, 237–244. ACM.

Müller, Matthias. 2009. Fast and robust tracking of fluid surfaces. In *Symposium on computer animation*, 237–245. New York, NY, USA: ACM.

Museth, Ken. 2013. A flexible image processing approach to the surfacing of particle-based fluid animation. *Mathematical Progress in Expressive Image Synthesis* 4.

Museth, Ken. 2014. Hierarchical digital differential analyzer for efficient ray-marching in openvdb. In *Acm siggraph 2014 talks*, 40. ACM.

Museth, Ken, Jeff Lait, John Johanson, Jeff Budsberg, Ron Henderson, Mihai Alden, Peter Cucka, David Hill, and Andrew Pearce. 2013. Openvdb: an open-source data structure and toolkit for high-resolution volumes. In *Acm siggraph 2013 courses*, 19. ACM.

Mutabaruka, P., and K. Kamrin. 2017. A simulation technique for slurries interacting with moving parts and deformable solids with applications. *arXiv preprint arXiv:1703.05158*.

Narain, R., A. Golas, and M. Lin. 2010. Free-flowing granular materials with two-way solid coupling. *ACM Trans Graph* 29(6):173:1–173:10.

Ng, Yen Ting, Chohong Min, and Frédéric Gibou. 2009. An efficient fluid-solid coupling algorithm for single-phase flows. *J. Comp. Phys.* 228(23):8807–8829.

Nielsen, M.B., and O. Østerby. 2013. A two-continua approach to eulerian simulation of water spray. *ACM Trans Graph* 32(4):67.

Nielsen, Michael B., and Robert Bridson. 2016. Spatially adaptive FLIP fluid simulations in Bifrost. In *Acm siggraph talks*, 41.

Otaduy, M. A., D. Germann, S. Redon, and M. Gross. 2007. Adaptive deformations with fast tight bounds. In *Proc of the 2007 acm siggraph/eurographics symp on comp anim*, 181–190. Eurographics Association.

Pailha, M., M. Nicolas, and O. Pouliquen. 2008. Initiation of underwater granular avalanches: Influence of the initial volume fraction. *Phys of Fluids* 20(11):111701.

Parker, S. G. 2006. A component-based architecture for parallel multi-physics pde simulation. *Fut Gen Comp Sys* 22(1):204 – 216.

Patel, S., A. Chu, J. Cohen, and F. Pighin. 2005. Fluid simulation via disjoint translating grids. In *Acm siggraph 2005 sketches*, 139. ACM.

Patkar, Saket, Mridul Aanjaneya, Dmitriy Karpman, and Ronald Fedkiw. 2013. A hybrid lagrangian-eulerian formulation for bubble generation and dynamics. In *Proceedings of the 12th acm siggraph/eurographics symposium on computer animation*, 105–114. ACM.

Peer, A., M. Ihmsen, J. Cornelis, and M. Teschner. 2015. An implicit viscosity formulation for sph fluids. *ACM Trans Graph* 34(4):114.

Perot, Blair, and V. Subramanian. 2007. Discrete calculus methods for diffusion. *J. Comp. Phys.* 224(1):59–81.

Popinet, Stéphane. 2003. Gerris: a tree-based adaptive solver for the incompressible euler equations in complex geometries. *Journal of Computational Physics* 190(2):572–600.

Ram, D., T. Gast, C. Jiang, C. Schroeder, A. Stomakhin, J. Teran, and P. Kavehpour. 2015. A material point method for viscoelastic fluids, foams and sponges. In *In proc symp comp anim*, 157–163.

Ren, B., C. Li, X. Yan, M.C. Lin, J. Bonet, and S.M. Hu. 2014. Multiple-fluid sph simulation using a mixture model. *ACM Trans Graph* 33(5):171.

Robinson, M., and M. Ramaioli. 2011. Mesoscale fluid-particle interaction using two-way coupled sph and the discrete element method. In *Proc of the 6th spheric work*, 72–78.

Robinson-Mosher, Avi, Tamar Shinar, Jon Gretarsson, Jonathan Su, and Ronald Fedkiw. 2008. Two-way coupling of fluids to rigid and deformable solids and shells. *ACM Trans Graph* 27(3): 46:1–46:9.

Ruggirello, K. P, and S. C. Schumacher. 2014. A comparison of parallelization strategies for the material point method. In *11th world cong on comp mech*, 20–25.

Rungjiratananon, W., Y. Kanamori, and T. Nishita. 2012. Wetting effects in hair simulation. In *Comp graph forum*, vol. 31, 1993–2002. Wiley Online Library.

Rungjiratananon, W., Z. Szego, Y. Kanamori, and T. Nishita. 2008. Real-time animation of sand-water interaction. In *Comp graph forum*, vol. 27, 1887–1893. Wiley Online Library.

Seiler, M., D. Steinemann, J. Spillmann, and M. Harders. 2011. Robust interactive cutting based on an adaptive octree simulation mesh. *The Vis Comp* 27(6-8):519–529.

Selle, Andrew, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. 2008. An unconditionally stable MacCormack method. *SIAM J. Sci.Comput.* 35(2-3):350–371.

Setaluri, Rajsekhar, Mridul Aanjaneya, Sean Bauer, and Eftychios Sifakis. 2014. Spgrid: A sparse paged grid structure applied to adaptive smoke simulation. *ACM Trans Graph* 33(6):205:1–205:12.

Sethian, James. 1999. *Level set methods and fast marching methods*. Cambridge University Press.

Sethian, James A., and Alexander Vladimirsky. 2000. Fast methods for the eikonal and related hamilton-jacobi equations on unstructured meshes. *Proceedings of the National Academy of Sciences* 97(11):5699–5703.

Sifakis, E., and J. Barbic. 2012. Fem simulation of 3d deformable solids: A practitioner's guide to theory, discretization and model reduction. In *Acm siggraph 2012 courses*, 20:1–20:50. SIGGRAPH '12.

Sifakis, E., T. Shinar, G. Irving, and R. Fedkiw. 2007. Hybrid simulation of deformable solids. In *Proc acm siggraph/eurograph symp comp anim*, 81–90.

Sifounakis, Adamandios, Sangseung Lee, and Donghyun You. 2016. A conservative finite volume method for incompressible Navier-Stokes equations on locally refined nested Cartesian grids. *J. Comp. Phys.* 326:845–861.

Sin, F., A. Bargteil, and J. Hodgins. 2009. A point-based method for animating incompressible flow. In *Proc acm siggraph/eurograph symp comp anim*, 247–255.

Smith, B., D. M. Kaufman, E. Vouga, R. Tamstorf, and E. Grinspun. 2012. Reflections on simultaneous impact. *ACM Trans Graph* 31(4):106:1–106:12.

Snider, D. M. 2001. An incompressible three-dimensional multiphase particle-in-cell model for dense particle flows. *J Comp Phys* 170(2):523–549.

Solenthaler, Barbara, and Markus Gross. 2011. Two-scale particle simulation. *ACM Trans. Graph. (SIGGRAPH)* 30(4):81.

Stam, Jos. 1999. Stable fluids. In *Proceedings of the 26th annual conference on computer graphics and interactive techniques*, 121–128. ACM Press/Addison-Wesley Publishing Co.

Stantchev, G., D. Dorland, and N. Gumerov. 2008. Fast parallel particle-to-grid interpolation for plasma pic simulations on the gpu. *J Par Dis Comp* 68(10):1339 – 1349.

Steffen, Michael, Robert M Kirby, and Martin Berzins. 2008. Analysis and reduction of quadrature errors in the material point method (MPM). *Int J Numer Meth Eng* 76(6):922–948.

Steinemann, D., M. A. Otaduy, and M. Gross. 2008. Fast adaptive shape matching deformations. In *Proc of the 2008 acm siggraph/eurographics symp on comp anim*, 87–94. Eurographics Association.

Stomakhin, A., R. Howes, C. Schroeder, and J. Teran. 2012. Energetically consistent invertible elasticity. In *In proc symp comp anim*, 25–32.

Stomakhin, A., C. Schroeder, L. Chai, J. Teran, and A. Selle. 2013. A material point method for snow simulation. *ACM Trans Graph* 32(4):102:1–102:10.

Stomakhin, A., C. Schroeder, C. Jiang, L. Chai, J. Teran, and A. Selle. 2014. Augmented MPM for phase-change and varied materials. *ACM Trans Graph* 33(4):138:1–138:11.

Sulsky, D., S. Zhou, and H. Schreyer. 1995. Application of a particle-in-cell method to solid mechanics. *Comp Phys Comm* 87(1):236–252.

Sun, R., and H. Xiao. 2016a. Cfd–dem simulations of current-induced dune formation and morphological evolution. *Adv Water Res* 92:228–239.

Sun, Rui, and Heng Xiao. 2016b. Sedifoam: A general-purpose, open-source cfd–dem solver for particle-laden flow with emphasis on sediment transport. *Comp & Geo* 89:207–219.

Takahashi, Tetsuya, and Ming C. Lin. 2016. A multilevel sph solver with unified solid boundary handling. *Computer Graphics Forum* 35(7):517–526.

Tampubolon, A. P., T. Gast, G. Klár, C. Fu, J. Teran, C. Jiang, and K. Museth. 2017. Multi-species simulation of porous sand and water mixtures. *ACM Trans Graph* 36(4).

Tan, Honglai, and John A Nairn. 2002. Hierarchical, adaptive, material point method for dynamic energy release rate calculations. *Computer Methods in Applied Mechanics and Engineering* 191(19): 2123–2137.

Tang, M., Z. Liu, R. Tong, and D. Manocha. 2018. PSCC: Parallel self-collision culling with spatial hashing on GPUs. *Proc ACM on Comp Graph Int Techn* 1(1):18:1–18.

Tang, M., R. Tong, R. Narain, C. Meng, and D. Manocha. 2013. A gpu-based streaming algorithm for high-resolution cloth simulation. *Comp Graph Forum* 32(7):21–30.

Tang, Min, Huamin Wang, Le Tang, Ruofeng Tong, and Dinesh Manocha. 2016. Cama: Contact-aware matrix assembly with unified collision handling for gpu-based cloth simulation. *Comp Graph Forum* 35(2):511–521.

Teng, Y., D. Levin, and T. Kim. 2016. Eulerian solid-fluid coupling. *ACM Trans Graph* 35(6):200:1–200:8.

Wang, H. 2018. Rule-free sewing pattern adjustment with precision. *ACM Trans Graph* 37(4).

Wang, X., M. Tang, D. Manocha, and R. Tong. 2018. Efficient bvh-based collision detection scheme with ordering and restructuring. In *Comp graph forum*, vol. 37, 227–237. Wiley Online Library.

Weller, R., N. Debowski, and G. Zachmann. 2017. kdet: Parallel constant time collision detection for polygonal objects. In *Comp graph forum*, vol. 36, 131–141.

Wicke, M., D. Ritchie, B. Klingner, S. Burke, J. Shewchuk, and J. O'Brien. 2010. Dynamic local remeshing for elastoplastic simulation. *ACM Trans Graph* 29(4):49:1–11.

Wojtan, C., M. Carlson, P.J. Mucha, and G. Turk. 2007. Animating corrosion and erosion. In *Eurograph work nat phen*, 15–22.

Wojtan, C., and G. Turk. 2008. Fast viscoelastic behavior with thin features. *ACM Trans Graph* 27(3): 1–8.

Wojtan, Chris, Nils Thuerey, Markus Gross, and Greg Turk. 2009. Deforming meshes that split and merge. *ACM Trans. Graph. (SIGGRAPH)* 28(3):76.

Wretborn, J., R. Armiento, and K. Museth. 2017. Animation of crack propagation by means of an extended multi-body solver for the material point method. *Comp and Graph*.

Wu, K., N. Truong, Yuksel C., and Hoetzlein R. 2018. Fast fluid simulations with sparse volumes on the GPU. *Comp Graph Forum*.

Xiao, H., and J. Sun. 2011. Algorithms in a robust hybrid cfd-dem solver for particle-laden flows. *Comm Comp Phys* 9(2):297–323.

Xu, B.H., and A.B. Yu. 1997. Numerical simulation of the gas-solid flow in a fluidized bed by combining discrete particle method with computational fluid dynamics. *Chem Eng Sci* 52(16):2785 – 2809.

Yan, X., Y. Jiang, C. Li, R. Martin, and S. Hu. 2016. Multiphase sph simulation for interactive fluids and solids. *ACM Trans Graph* 35(4):79.

Yang, T., J. Chang, M.C. Lin, R. Martin, J. Zhang, and S.M. Hu. 2017. A unified particle system framework for multi-phase, multi-material visual simulations. *ACM Trans Graph* 36(6).

Yang, T., J. Chang, B. Ren, M.C. Lin, J.J. Zhang, and S.M. Hu. 2015. Fast multiple-fluid simulation using helmholtz free energy. *ACM Trans Graph* 34(6):201.

Yu, Jihun, Chris Wojtan, Greg Turk, and Chee Yap. 2012. Explicit mesh surfaces for particle based fluids. *Computer Graphics Forum (Eurographics)* 31(2):815–824.

Yue, Y., B. Smith, C. Batty, C. Zheng, and E. Grinspun. 2015. Continuum foam: a material point method for shear-dependent flows. *ACM Trans Graph* 34(5):160:1–160:20.

Zhang, F., X. Zhang, K.Y. Sze, Y. Lian, and Y. Liu. 2017. Incompressible material point method for free surface flow. *J Comp Phys* 330:92–110.

Zhang, Xinxin, and Robert Bridson. 2014. A PPPM fast summation method for fluids and beyond. *ACM Trans. Graph. (SIGGRAPH Asia)* 33(6):206.

Zhang, Y., B. Solenthaler, and R. Pajarola. 2008. Adaptive sampling and rendering of fluids on the gpu. In *Proc of the fifth eurographics / ieee vgtc conf on point-based graph*, 137–146.

Zhang, Y., X. Zhang, and Y. Liu. 2010. An alternated grid updating parallel algorithm for material point method using openmp. *Comp Modeling in Eng and Sci* 69(2):143–165.

Zhu, Bo, Wenlong Lu, Matthew Cong, Byungmoon Kim, and Ronald Fedkiw. 2013. A new grid structure for domain extension. *ACM Trans. on Graph.(SIGGRAPH)* 32(4):63.

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