

AN ADAPTIVE SAMPLING APPROACH TO INCOMPRESSIBLE
PARTICLE-BASED FLUID

A Dissertation
by
WOO-SUCK HONG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

May 2009

Major Subject: Computer Science

AN ADAPTIVE SAMPLING APPROACH TO INCOMPRESSIBLE
PARTICLE-BASED FLUID

A Dissertation
by
WOO-SUCK HONG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

Approved by:

Co-Chairs of Committee,	Donald H House
	John Keyser
Committee Members,	Frederic I. Parke
	Ricardo Gutierrez-Osuna
Head of Department,	Valerie E. Taylor

May 2009

Major Subject: Computer Science

ABSTRACT

An Adaptive Sampling Approach to
Incompressible Particle-Based Fluid. (May 2009)
Woo-suck Hong, B.S., Pukyong National University;
M.S., Pukyong National University

Co-Chairs of Advisory Committee: Dr. Donald H. House
Dr. John Keyser

I propose a particle-based technique for simulating incompressible fluid that includes adaptive refinement of particle sampling. Each particle represents a mass of fluid in its local region. Particles are split into several particles for finer sampling in regions of complex flow. In regions of smooth flow, neighboring particles can be merged. Depth below the surface and Reynolds number are exploited as our criteria for determining whether splitting or merging should take place. For the fluid dynamics calculations, I use the hybrid FLIP method, which is computationally simple and efficient. Since the fluid is incompressible, each particle has a volume proportional to its mass. A kernel function, whose effective range is based on this volume, is used for transferring and updating the particle's physical properties such as mass and velocity. In addition, the particle sampling technique is extended to a fully adaptive approach, supporting adaptive splitting and merging of fluid particles and adaptive spatial sampling for the reconstruction of the velocity and pressure fields. Particle splitting allows a detailed sampling of fluid momentum in regions of complex flow. Particle merging, in regions of smooth flow, reduces memory and computational overhead. An octree structure is used to compute inter-particle interactions and to compute the pressure field. The octree supporting field-based calculations is adapted to provide

a fine spatial reconstruction where particles are small and a coarse reconstruction where particles are large. This scheme places computational resources where they are most needed, to handle both flow and surface complexity. Thus, incompressibility can be enforced even in very small, but highly turbulent areas. Simultaneously, the level of detail is very high in these areas, allowing the direct support of tiny splashes and small-scale surface tension effects. This produces a finely detailed and realistic representation of surface motion.

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Prof. Donald H. House. I could not have asked for a better advisor. Thanks as well to my other advisor, Prof. John Keyser, and to my committee members, Prof. Frederic I. Parke and Prof. Ricardo Gutierrez-Osuna. Also, thanks to Prof. CheungHun Kim, Prof. Yoonsuck Choe and Prof. Jinxiang Chai for their encouragement. Finally I would like to thank my parents and sister. My study in the US would have not been possible without their love for me.

TABLE OF CONTENTS

CHAPTER	Page
I INTRODUCTION	1
II SIMULATION OF WATER	4
A. Navier-Stokes equations	4
B. Eulerian fluid	5
1. Stam's stable fluid	9
C. Lagrangian fluid	12
1. Smoothed particle hydrodynamics	12
a. Modeling fluid forces	13
b. Pressure	14
c. Viscosity	14
d. Surface tension	15
e. Smoothing kernels	15
2. Moving Particle Semi-implicit Method	16
a. Modeling of incompressibility	18
D. Hybrid fluid simulation	19
1. Particle-in-cell and fluid-implicit-particle method	21
E. Fluid simulation coupled with octree based structure	22
1. Combination of octree	22
2. Previous work	24
3. Introduction to octree	24
4. Representation of quadtree and octree	25
5. Finding a leaf node from an arbitrary location	25
6. Finding the smallest common enclosing cells by region	26
7. Finding neighbors	27
8. Constructing the poisson equation	28
a. The pressure gradient	28
b. The divergence	31
F. Surface reconstruction	32
1. Particle level set	32
a. Reinitialization	34
2. Blobbies	37
3. Bridson's method	38

CHAPTER	Page
III ADAPTIVE PARTICLES	40
A. Introduction	40
1. Our contribution	41
B. Previous works	43
C. The adaptive fluid simulation algorithm	45
D. Computing deformability	46
1. Distance from surface	46
2. Reynolds number	47
E. Determining when to merge and split	48
F. Splitting and merging	49
1. Merging	50
2. Splitting	50
3. Upper and lower bounds	51
G. Kernels and the auxiliary grid	51
H. Results and evaluation	54
1. Experimental results	54
2. Evaluation	60
IV FULLY ADAPTIVE FLUID SIMULATION FOR INCOMPRESSIBLE FLOW	63
A. Introduction	63
B. Overview of our adaptive fluid simulation	66
C. Octree generation	68
1. Relating adaptive particles and octree cells	68
2. Creating the octree grid	70
D. Splitting and merging	72
1. Splitting and merging process	72
2. Layers for splitting and merging	73
E. Kernel choice and transferring particle velocity to octree grid	74
F. Constructing the poisson equation	75
G. Divergence	76
H. Results and discussion	76
1. Results	76
2. Contrast with previous methods	77
V SURFACE TENSION WITH FULLY ADAPTIVE FLUID SIMULATION	81

CHAPTER	Page
A. Introduction	81
B. Modeling surface tension	82
C. Adding surface tension to fully adaptive fluid simulation . .	85
D. Fluid-object interaction	86
E. Results and comparison	88
VI CONCLUSION/FUTURE WORK	92
A. Future work	94
REFERENCES	96
VITA	101

LIST OF TABLES

TABLE	Page
I Parameters used in experiments	55
II Simulation time table for a 2D simulation similar to Figure 21 . . .	58

LIST OF FIGURES

FIGURE	Page
1 Staggered MAC grid	6
2 Types of cells	7
3 A result from Successive Over Relaxation scheme	8
4 Eulerian fluid simulation	11
5 Smooth Particle Hydrodynamics	17
6 Moving Particle Semi-implicit Method	20
7 Hybrid method	23
8 Quadtree data structure and its representation	26
9 Quad tree and locational codes	27
10 The possible neighbors of a cell	28
11 Computing the pressure on a quadtree	30
12 Fluid simulation and particle level set	36
13 Blobbiness	38
14 Fluid simulation with adaptive particle sampling	41
15 Layers for Splitting and Merging	48
16 Mass and volume conservation when splitting and merging particles	50
17 Gaussian and tent kernel functions	52
18 2D simulation using Gaussian and tent kernel functions	52

FIGURE	Page
19 Sampling of particles with small radii	54
20 Fluid simulation with adaptive particle sampling	55
21 2D simulation for splitting and merging	56
22 3D simulation using merging only	57
23 3D simulation with splitting and merging	58
24 Surface without (left) and with (right) surface smoothing	60
25 Surface without (left) and with (right) surface smoothing	60
26 Animation result with surface-smoothing	62
27 Quadtree hybrid simulation with uniform cells	65
28 Overall simulation structure	67
29 The size of particles determines the size of an octree cell	69
30 Relation between adaptive particles and the octree cell	71
31 Layers for splitting and merging	74
32 Circular kernel functions	75
33 Contrast with Previous Methods.(Comparison of Normal simulation, Adaptive Particle and Fully Adaptive Simulation)	78
34 2D Fully Adaptive Fluid simulation	79
35 3D fluid simulation using adaptive particles and an octree grid	80
36 Non realistic splash effect without surface tension	82
37 Curvature in concave region and convex region	84
38 Surface tension in Fully Adaptive Fluid Simulation	85
39 Interaction between fluid particles and an object	87

FIGURE	Page
40 Comparison of different surface tension parameters	89
41 2D fluid simulation without (left) and with (right) surface tension . .	90
42 Image sequence of 3D surface tension test	91

CHAPTER I

INTRODUCTION

In the simulation of incompressible fluid, one is faced with two phenomena that seem to demand quite different simulation methodologies. One dominant factor in fluid flow is that moving fluid is transporting real material, and thus the properties of the fluid are being transported (adverted) with this material. This seems to call for a Lagrangian or particle-based simulation scheme, which naturally tracks the motion of discrete “chunks” of fluid mass, and thus provides a vehicle for handling advection in a straightforward way. The other dominant factor in fluid flow is that it is shaped largely by pressure gradients. Unlike advection, pressure is a massless field property that is poorly represented by moving particles, but quite naturally represented by an Eulerian or spatial-grid scheme. The usual way to handle advection within an Eulerian simulation is by backtracing, as in the *semi-Lagrangian* method [34], but this approach is well known to lead to considerable dissipation of flow complexity. It is possible to approximate the pressure effects in compressible flow using the metaphor of interparticle spring forces. However, this method is poorly suited to handling water flow, which is essentially incompressible, yielding an unnatural looking “bouncy” water surface.

Another important factor in fluid representation is that, within a flow, some regions will be highly complex while others are quite smooth. Similarly, for rendering purposes, having a detailed and convincing representation of the moving surface geometry may be considerably more important than having high detail below the surface. Thus, within a simulation, the ability to refine or coarsen the representation

This dissertation follows the style of The Visual Computer.

can be quite useful, regardless of whether an Eulerian or a Lagrangian computational methodology is used. To capture the complex flow and fine geometric detail at the surface, finer grid spacing or particle sampling can be used, while regions of smooth flow permit coarsening of grid or particle representations without loss of detail, saving computation time and space.

Finally, if the simulation level of detail is to be very fine, a representation of surface tension becomes important. This is especially true when it is desired to represent spray and splashes. A particle-based scheme naturally accounts for fine spray, but will not directly support the representation of surface tension, which is needed for the simulation of small water drops. Again, there is a need for both a particle-based representation of the material of the fluid, and a grid-based representation of the small-scale fluid surface. Our work attempts to find a suitable simulation framework that exploits the right methodologies for the various stages of the simulation, while maintaining a representation that is appropriately adapted to the required simulation granularity. For our approach, we have developed an adaptive version of the hybrid Flip method [37], which is essentially particle-based, but augmented by a computational grid for speeding the calculation of inter-particle interactions and for computing the pressure field. To make the approach adaptive, we have implemented a particle splitting and merging scheme using both depth from the surface and flow complexity to determine when particles should be split or merged. We use an octree spatial data structure to represent the computational grid, with octree cell size determined at each time step by the particle sampling in the neighborhood of the cell. Thus, the method is essentially particle based, with particle advection providing the transport mechanism in the fluid, and the pressure, inter-particle interaction and surface geometry being calculated on an octree whose resolution is locally determined by the particles.

The main contribution of this work is to demonstrate how to support a fully

adaptive simulation of both particle and grid-based components for incompressible fluids. As part of this, we present a new method for determining spatial grid resolution based on adaptive particle sizes, and coupling this grid with the particles. We also present improvements to particle splitting and merging. Our approach allows us to have the resolution needed to incorporate surface tension effects into an adaptive particle-based simulation, and we present a new method for computing these forces.

CHAPTER II

SIMULATION OF WATER

A. Navier-Stokes equations

We first briefly review vector calculus before describing the Navier-Stokes equations. The nabla operator ∇ is a vector differential operator. In a two-dimensional space it takes the form:

$$\nabla p = \left(\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y} \right) \quad (2.1)$$

Its discrete form is:

$$\frac{p_{i+1,j} - p_{i-1,j}}{2\delta x}, \quad \frac{p_{i,j+1} - p_{i,j-1}}{2\delta y} \quad (2.2)$$

The divergence, denoted by $\nabla \cdot u$ is an operator that measures the magnitude of a vector field's source at a given point. It is defined as follows:

$$\nabla \cdot u = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \quad (2.3)$$

Its discretization is :

$$\frac{u_{i+1,j} - u_{i-1,j}}{2\delta x} + \frac{v_{i,j+1} - v_{i,j-1}}{2\delta y} \quad (2.4)$$

The Laplace operator is a second order differential operator which is defined as the divergence of the gradient. In two dimensions it has the form:

$$\nabla^2 p = \nabla \cdot \nabla p = \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} \quad (2.5)$$

Its discretization is:

$$\frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{(\delta x)^2} + \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{(\delta y)^2} \quad (2.6)$$

Fluid simulations typically attempt to reproduce the effects of the Navier-Stokes equations that describe fluid flow. A typical representation of the incompressible form of these equations, after some simplification, is:

$$\frac{\partial u}{\partial t} = f - (u \cdot \nabla)u - \frac{1}{\rho} \nabla p + \nu \nabla^2 u \quad (2.7)$$

$$\nabla \cdot u = 0 \quad (2.8)$$

where u is the velocity field, p is the pressure field, f represents the sum of all external forces, ρ is a density measure, and ν is kinematic viscosity. The first equation is derived from Newton's Second Law, which states that momentum is always conserved and accounts for pressure gradients, diffusion, and external forces such as gravity. The $(u \cdot \nabla)u$ term accounts for advection of the fluid, and the $\nu \nabla^2 u$ term accounts for velocity diffusion due to the fluid's viscosity. The second equation ensures that the fluid is divergence free, and thus incompressible.

B. Eulerian fluid

Eulerian fluid simulations generally solve the Navier-Stokes equations over a fixed spatial grid. Most Eulerian fluid simulations in Computer Graphics are based on Harlow and Welch's original marker and cell (MAC) method [17] which was originally developed for the computational fluid dynamics community. Later, Foster and Metaxas [13] were the first in the computer graphics community to achieve full 3D liquid simulation based on the Navier-Stokes equations for incompressible flow. Their method is based largely on the work of Harlow and Welch [17]. A staggered MAC grid is used in this method to compute the physical properties such as the pressure and velocity fields.

As shown in Figure 1, the scalar quantities such as pressure are set to the cell center,

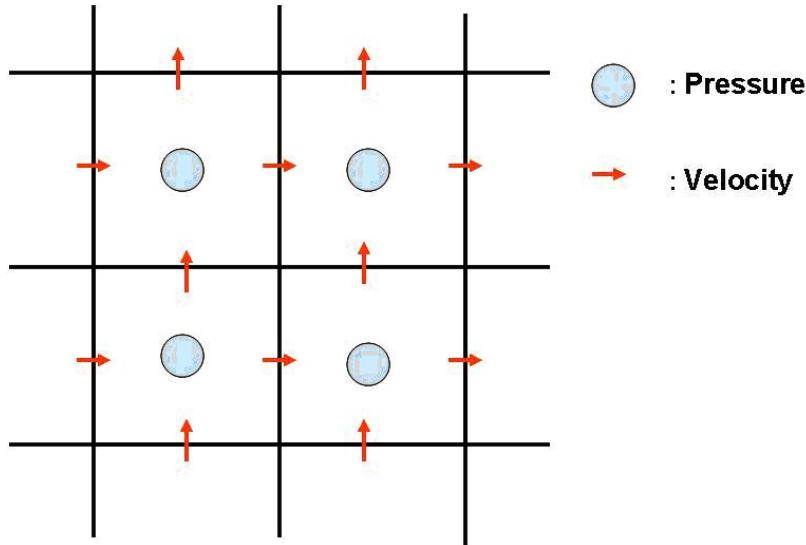


Fig. 1. Staggered MAC grid

and vector quantities such as velocity are located at the face of each cell. We can obtain a more accurate approximation for the gradient of the velocity field using a staggered MAC grid rather than a cell-centered method in which all physical properties are positioned at the center of cell. Massless marker particles are generated to represent the fluid, but have no effect on its motion. Also, marker particles play an important role in determining which cells are on the surface of the fluid. If solid or barrier cells are marked, then fluid cell types, such as air (containing no particles), surface (containing particles, and adjacent to an air cell), and full (containing particles and not adjacent to air cells) cells can be recognized by marker particles at each step as shown in Figure 2. Cells with no particles are marked as an air cells. If a cell contains particles and the adjacent cell is an air cell with no particles, it is marked as a surface cell. Otherwise, all remaining cells containing particles are considered fluid cells.

The Navier-Stokes equations are integrated explicitly by a forward finite difference approximation to obtain accurate fluid motion for advection and viscosity. We

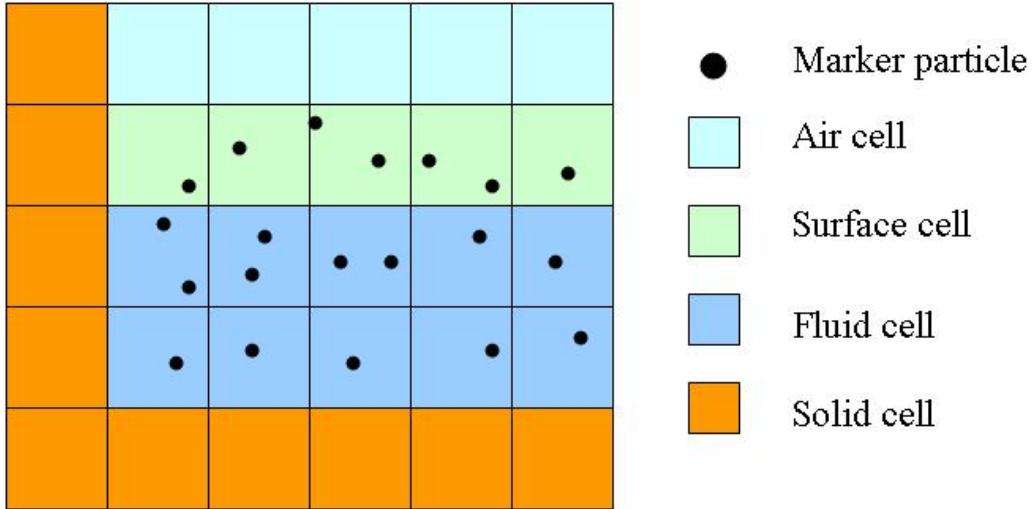


Fig. 2. Types of cells

need to consider the Courant-Friedrichs-Lowy condition (CFL condition) [7] for simulation stability, which is a condition for convergence while solving certain partial differential equations. Unfortunately, it strictly restricts the relationship between cell size and simulation time step to be $1 > \max[u\frac{\delta t}{\delta x}, v\frac{\delta t}{\delta y}, w\frac{\delta t}{\delta z}]$ where u, v, w are velocities in the x,y,z directions; $\frac{\delta t}{\delta x}, \frac{\delta t}{\delta y}, \frac{\delta t}{\delta z}$ are ratios of the time step to the cell size in the x,y,z directions. Thus, $u\frac{\delta t}{\delta x}, v\frac{\delta t}{\delta y}, w\frac{\delta t}{\delta z}$ are dimensionless quantities which must be less than 1 to assure that fluid cannot cross more than one cell in one time step. This means that the performance of a simulation will be significantly slower (i.e. will require a smaller time step) at a high spatial resolution.

In order to conserve mass, the new pressure field must be determined in each time step and the mass conservation equation $\nabla \cdot u = 0$ must be satisfied as well. Positive divergence represents an influx into a cell, meaning that the cell pressure is being increased, while negative divergence represents an outflow from the cell, meaning that the cell pressure is being decreased. Figure 3 shows a specific example of this. The cell marked in red has a fluid influx, and the figure shows that the pressure is

increased to 0.035715 and the pressure in the left cell is decreased to -0.033280. This is propagated throughout the overall grid.

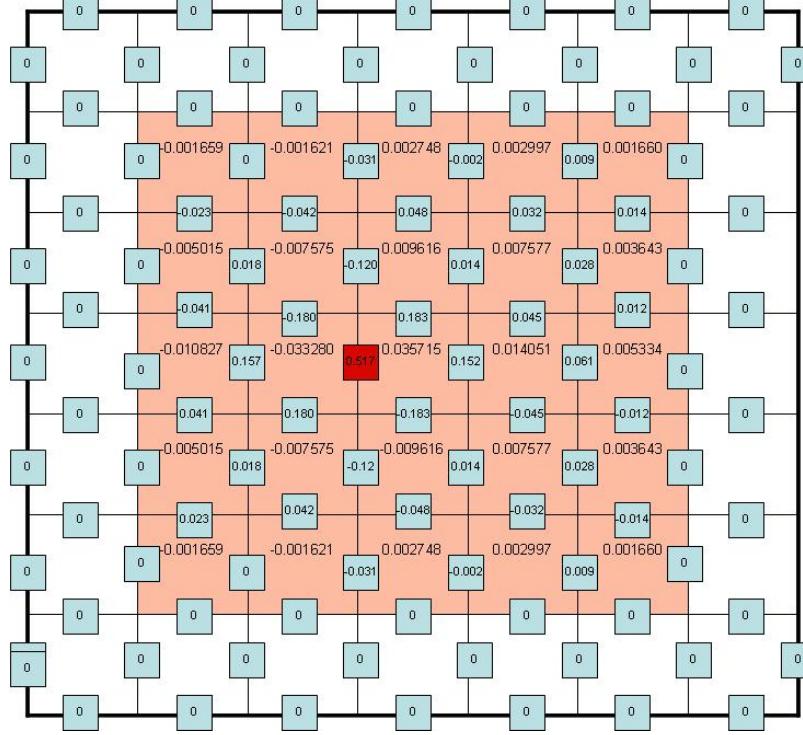


Fig. 3. A result from Successive Over Relaxation scheme

In work by Foster and Metaxas [13], Successive Over Relaxation (SOR) was used to enforce incompressibility, but this method usually has poor convergence and needs many iterations. Three cases of boundary conditions need to be set on the free surface so that the Navier-Stokes equations can be used to find the velocity for fluid cells. Thus, boundary conditions are needed for the velocity on the empty cells near the surface. First, if a cell is surrounded by three fluid cells, the velocity on the remaining face of the cell is set to $v_{i,j+1} = v_{i,j} - (\frac{\delta y}{\delta x})(u_{i+1,j} - u_{i,j})$. Second, if the cell has two sides labeled as cells, the velocity of each side equals that of the opposite side. Third, if the cell has one side that is air, the velocity of the open side equals that of the opposite side while the remaining two sides do not change. In most cases, this

procedure seems to work well, however the solution is not unique when more than one face of a cell needs to be updated and the cell has two or three sides of air cells. Later, an extrapolation method [11] was proposed, allowing higher order accuracy and more visually pleasing results for the implicit surface. The boundary condition velocity can be computed similar to the reinitialization step of the level set method [28]. Once the signed distance field is constructed away from the surface, the fluid velocity field can be extrapolated using $\nabla\phi \cdot \nabla u = 0$, where u is the extrapolated velocity and ϕ is a field quantity representing signed distance from the level set. This can be created using the fast marching method [32].

1. Stam's stable fluid

The stable fluid method, introduced by Stam [34], uses a semi Lagrangian scheme for advection as well as a projection method to ensure unconditional stability and divergence-free flow. As mentioned in the previous section, Foster and Metaxas's simulation [13] has difficulties in producing a stable fluid using large timesteps due to the forward finite differencing approach. This method requires many iterations for the SOR method to converge to a solution that enforces incompressibility. The stable fluid method can be considered as an alternative which resolves the above difficulties. The method is outlined as follows, where the w 's are the velocity field after successive refinement steps:

$$w_0(x) \xrightarrow{\text{add force}} w_1(x) \xrightarrow{\text{advect}} w_2(x) \xrightarrow{\text{diffuse}} w_3(x) \xrightarrow{\text{project}} w_4(x) \quad (2.9)$$

The solution is given by the last velocity field w_4 . The step is iterated every time step for the duration of simulation. First, the external force f is added using Euler

integration in order to create a good approximation of the force effect:

$$w_1 = w_0 + \Delta t f \quad (2.10)$$

In order to solve the advection step, Stam introduces a technique called the semi-Lagrangian scheme. This step satisfies the $-(u \cdot \nabla)u$ term in the momentum equation:

$$w_2 = \text{transport}(w_1, -\Delta t) \quad (2.11)$$

Euler integration is used to integrate backwards through the velocity field, determining where it would have been at $t - \Delta t$. The velocity at the original position is sampled at the backtraced position from the velocity field. The maximum value of the new field is never larger than the maximum value of the previous field, which result in an unconditionally stable solver. The third step accounts for the effect of viscosity forces:

$$\frac{\partial w_2}{\partial t} = \nu \nabla^2 w_2. \quad (2.12)$$

An implicit integration scheme is used to ensure stability rather than an explicit Euler method, because high stiffness and large time steps lead to an unstable simulation:

$$(I - \nu \Delta t \nabla^2) w_3 = w_2. \quad (2.13)$$

When the equation is discretized, it leads to a matrix form which can be inverted using a sparse linear system for the unknown field w_3 . The fourth step is used to guarantee a divergence free-velocity field:

$$w_4 = w_3 - \frac{\Delta t}{\rho} \nabla p = w_3 - \nabla q \quad (2.14)$$

The main idea is to use *Helmholtz-Hodge Decomposition*, which states that the velocity field can be decomposed into a divergence-free field and the gradient of a scalar field

q :

$$\nabla^2 q = \nabla \cdot w_3 \quad (2.15)$$

In this case, the scalar field q is proportional to the pressure. The equation produces a sparse diagonal matrix and is solved by a Poisson solver using a bi-conjugate gradient method, which is more accurate and faster than a solution via SOR. Once the pressure is determined, the divergence-free velocity field can be easily computed. Figure 4 is a frame from an animation of a fluid surface computed using the approach outlined above. The fluid surface is obtained using the particle level set approach [10].

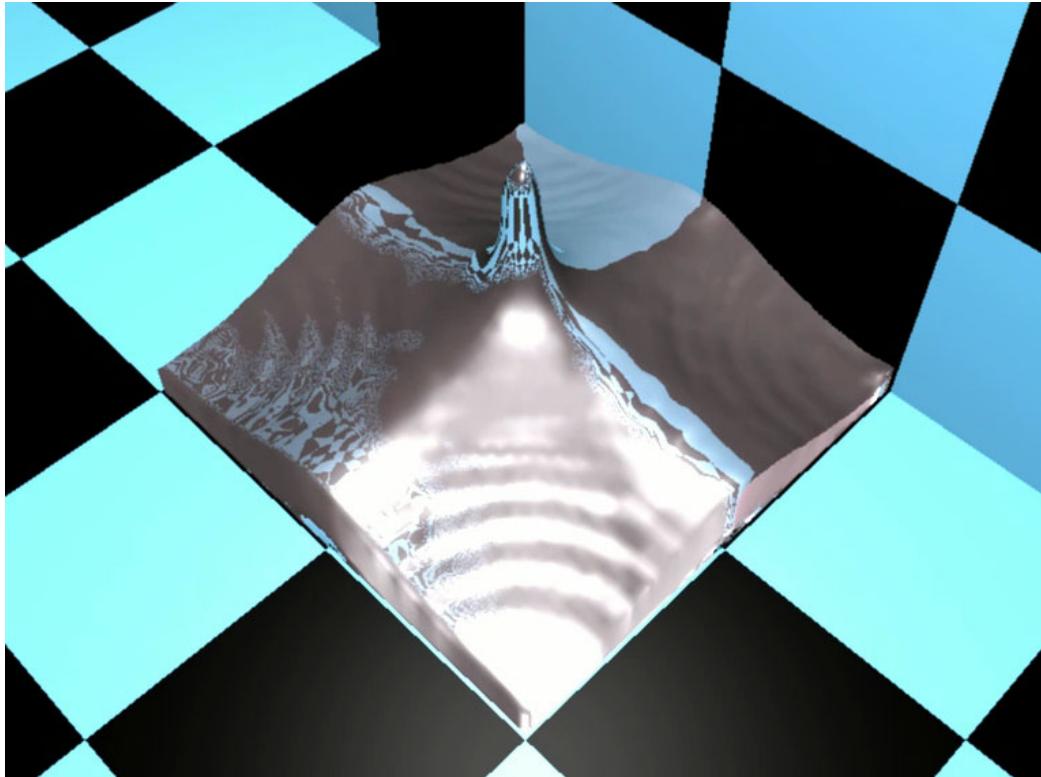


Fig. 4. Eulerian fluid simulation

C. Lagrangian fluid

Particles are the fundamental momentum-carrying simulation element in Lagrangian simulations. The most popular Lagrangian fluid simulation method is the Smooth Particle Hydrodynamics (SPH) Method [25]. In SPH, the fluid is composed of a set of particles, with inter-particle forces such as pressure and viscosity computed at the position of each particle. Desbrun and Cani-Gasculel [8] introduced SPH into computer graphics to simulate highly deformable substances such as lava flow.

Müller et al. [27] extended the SPH approach for interactive fluid simulation, and introduced a new technique for complex behaviors, such as air-water interaction, boiling water, and trapped air. In a recent paper, Adams et al. [2] have proposed an improved adaptive method for SPH compressible fluid simulation. Though aspects of this paper are similar to ours, we emphasize that our approach handles incompressible fluid, and is in fact the first adaptive particle method to do so.

Premoze [29] introduced Koshizuka's original work [22] on the Moving-Particle Semi-Implicit Method (MPS) to simulate incompressible liquid and multifluid flow. Unlike SPH, MPS handles incompressible fluid. However, the approach is slow, tends towards instability, and is not adaptive. Clavet [6] also extended the formulation of SPH by using a method called Double Density Relaxation for enforcing incompressibility and to oppose clustering. Several small-scale simulations of substances such as mud and paint were created realistically with surface tension using springs between particles.

1. Smoothed particle hydrodynamics

In SPH, the fluid is sampled by a set of particles. A particle is a sample point where field quantities of the fluid are defined locally. A particle carries physical quantities

such as mass, density, velocity and pressure in its local area. Such field quantities and their derivatives can be interpolated at particle radius r by a weighted sum of contributions.

$$f(r) = \sum_j m_j \frac{f_j}{\rho_j} W_h(r - r_j) \quad (2.16)$$

where f_j denotes the field quantity at particle j , m_j is the mass of particle, r_j is its position, ρ_j is the density and W_j is a smoothing kernel determining the local support for f . In order to simulate a fluid, derivatives of field quantities such as pressure or velocity are needed. The following equation is used in order to evaluate the gradient of f :

$$\nabla f(r) = \sum_j m_j \frac{f_j}{\rho_j} \nabla W_h(r - r_j) \quad (2.17)$$

The Laplacian of f is as follows:

$$\nabla^2 f(r) = \sum_j m_j \frac{f_j}{\rho_j} \nabla^2 W_h(r - r_j) \quad (2.18)$$

These equations are the basis of the SPH simulation. Using the above equations, density in the location of a particle can be evaluated as:

$$\rho(r) = \sum_j m_j W_h(r - r_j) \quad (2.19)$$

a. Modeling fluid forces

First, we consider the Navier-Stokes equations in terms of a Lagrangian fluid model:

$$\rho \left(\frac{\partial v}{\partial t} + v \cdot \nabla v \right) = -\nabla p + \rho g + \mu \nabla^2 v \quad (2.20)$$

The term $v \cdot \nabla v$ of the left side is not necessary in Lagrangian fluid simulation since the particle has its own physical properties, such as velocity, and it moves with fluid.

Thus, the rest of terms in right side of the equation, pressure $-\nabla p$, external forces ρ , and viscosity $\mu\nabla^2v$ are all the terms that we need to consider.

b. Pressure

We model the pressure force, a factor which causes a change in momentum as follows,

$$-\nabla p = - \sum_j m_j \frac{p_j}{\rho_j} \nabla W_h(r - r_j) \quad (2.21)$$

Müller [27] pointed out that the equation leads to non-symmetric forces, since the gradient of the kernel is zero and the particle j is only used for particle i to compute its pressure force. He proposed that the symmetric pressure force can be computed using the arithmetic mean of the pressure among the interacting particles.

$$-\nabla p = - \sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W_h(r - r_j) \quad (2.22)$$

c. Viscosity

Viscosity can be considered a force of friction resisting the relative motion between the surface of two objects. This force can be modeled using the average of the velocities at the particle positions as follows:

$$\mu \nabla^2 v = \mu \sum_j m_j \frac{v_j}{\rho_j} \nabla^2 W_h(r - r_j) \quad (2.23)$$

This equation also produces an asymmetric viscosity force. The following equation was proposed by Müller [27] in order to compute the symmetric force.

$$\mu \nabla^2 v = \mu \sum_j m_j \frac{v_j - v}{\rho_j} \nabla^2 W_h(r - r_j) \quad (2.24)$$

In the equation, the particle i is accelerated by the average of the velocity differences. If the fluid in the position of a particle is flowing faster, and the fluid in a neighboring

position of the particle is flowing slower, the velocity in the particle position is adjusted in the direction of the relative speed.

d. Surface tension

Surface tension is a phenomenon caused by cohesive forces between liquid molecules. The force acting on the surface of a liquid tends to minimize the area of the surface. Its strength depends on the forces of attraction among the particles. For example, surface tension results in spherical drops of liquid, as the liquid tends to minimize its surface area. In order to compute surface tension, we need an additional distance field as follows:

$$C_s = \sum_j m_j \frac{1}{\rho_j} W_h(r - r_j) \quad (2.25)$$

The gradient of the color field $n = \nabla c_s$ and the normalized scalar field $\delta_s = |n|$ are used to compute the surface normal field. The Laplacian of the color field represents the curvature of the surface $\kappa = \frac{-\nabla^2 c_s}{|n|}$. The surface tension force is evaluated using:

$$\sigma \kappa n = -\sigma \nabla^2 c_s \frac{n}{|n|} \quad (2.26)$$

e. Smoothing kernels

Each particle has its own area of influence that is weighted by a kernel. The choice of kernel is very important in particle based simulation since the performance of the simulation, as well as its accuracy and stability depend on it. The kernel function is used to determine the contribution that the particle has on other nearby particles. Müller [27] used the following three smoothing kernels:

$$w_h^{poly6} = \frac{315}{64\pi h^9} (h^2 - r^2) \quad (2.27)$$

$$w_h^{spiky} = \frac{15}{\pi h^6} (h - r)^3 \quad (2.28)$$

$$w_h^{viscosity} = \frac{15}{2\pi h^3} \left(-\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1 \right) \quad (2.29)$$

The kernels are mainly used to compute the density, pressure force, and viscosity force for the duration of the simulation. The kernel *poly6* is used for sampling the density at each particle location, but if it is used for the computation of pressure force, particles tend to cluster, or overlap, since the repulsive force vanishes when two particles are very close to each other, producing a high pressure region. Desbrun [8] used a *spiky* function to resolve the problem, since its gradient vanishes around the center of a particle. The *viscosity* kernel is used for computing the viscosity force, whose Laplacian is positive and produces the correct effect for smoothing the velocity field. Figure 5 shows two images from a sequence of images from an SPH simulation of pouring water into a cup. The particles are colored blue before colliding with the wall and are colored green afterward.

2. Moving Particle Semi-implicit Method

Premoze [29] introduced Koshizuka's original work [22] on the Moving-Particle Semi-Implicit Method (MPS) to simulate incompressible liquid and multifluid flow. Unlike SPH, MPS handles incompressible fluid. However, the approach is slow and tends towards instability. For an incompressible fluid, the fluid density must be constant and the number of particles in a unit volume is approximated by the particle number density as follows:

$$\langle \rho_n \rangle_i = \frac{\langle n \rangle_i}{\int w(r) dv} \quad (2.30)$$

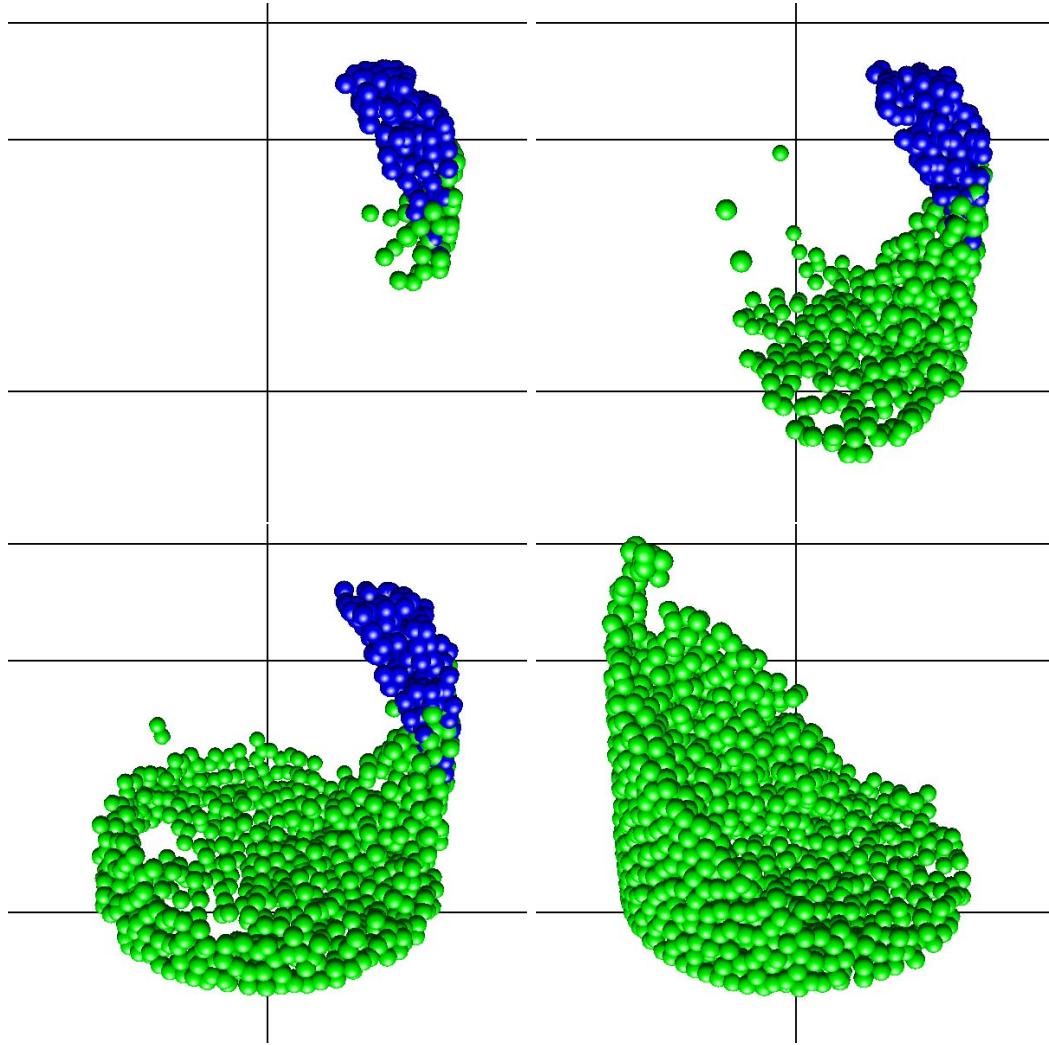


Fig. 5. Smooth Particle Hydrodynamics

where $w(r)$ is $\frac{r_e}{r}$ within the effective range of particle radius, and the particle number density is computed as $\langle n \rangle_i = \sum_{j \neq i} w(|r_j - r_i|)$. Later, the particle density is fixed to enforce the incompressibility. A gradient vector is defined by $(\phi_j - \phi_i)(r_j - r_i)/|r_j - r_i|^2$ and is evaluated using the weighted kernel between particle i and its neighbor j . The gradient of field quantities is computed as follows:

$$\langle \nabla \phi \rangle_i = \frac{d}{n^0} \sum_{j \neq i} i \frac{\phi_j - \phi_i}{|r_j - r_i|^r} w(|r_j - r_i|) \quad (2.31)$$

where d represents the space dimensions. The equation is used to compute the pressure gradient in MPS. When a particle get close to its neighbor, it provides a large force, which helps to avoid clustering throughout the duration of simulation. As presented in the previous section, Desbrun [8] used a *spiky* function to resolve the problem, since its gradient vanished around the center of the particle in the use of *poly6*. However, a big pressure difference between a particle and its neighbor can lead to instability. The Laplacian of field quantities is computed as follows,

$$\langle \nabla^2 \phi \rangle_i = \frac{2d}{\lambda n^0} \sum_{j \neq i} (\phi_j - \phi_i) w(|r_j - r_i|) \quad (2.32)$$

This model of the Laplacian is used for computing the viscosity force.

a. Modeling of incompressibility

Unlike SPH, MPS handles incompressible fluid, which means that the particle number density n^0 should be a constant throughout the simulation. After computing the pressure and viscosity forces, the incompressibility of the fluid is violated. When the particle number density n^* is not n^0 , it should be corrected to n^0 as follows:

$$n^* + n' = n^0 \quad (2.33)$$

The divergence of velocity in the particle location can be computed by the correction value n' as follows:

$$\frac{1}{\Delta t} \frac{n'}{n^0} = -\nabla \cdot u' \quad (2.34)$$

The modification velocity u' is derived from the implicit pressure term:

$$u' = -\frac{\Delta t}{\rho} \nabla P^{n+1} \quad (2.35)$$

With the above three equations, a Poisson equation for pressure is obtained:

$$\langle \nabla^2 P^{n+1} \rangle_i = -\frac{\rho}{\Delta t^2} \frac{\langle n^* \rangle_i - n^0}{n^0} \quad (2.36)$$

The right side of the equation is similar to the divergence of the velocity since it is represented by the deviation of the particle number density from a constant value. A linear symmetric matrix is constructed in the equation, which is sparse and symmetric and it can be solved using the conjugate gradient method. The overall steps of MPS algorithm are as follows:

- Apply forces to particles and find temporary particle positions and velocities u^* , r^* using Euler forward integration $r^*_i = r^n_i + \Delta u^*_i$
- Solve pressure Poisson equation using Conjugate Gradient method. $\nabla^2 P^{n+1} = -\frac{\rho}{\Delta t^2} \frac{\langle n^* \rangle_i - n^0}{n^0}$
- Calculate the modification of velocity u' and find new particle position $(u_i^{n+1}, r_i^{n+1}, P_i^{n+1})$

The above steps are iterated throughout the duration of the simulation. When a symmetric matrix is constructed for solving the pressure Poisson equation, the surface particles can be considered with $n_i^* < \beta n^0$, where β is a parameter below 1.0. Figure 6 shows a single frame from an animation that we obtained using the Moving Particle Semi-implicit Method. The surface particles are represented in blue and the remaining particles are shown in green. The figure shows an example of a splashing effect.

D. Hybrid fluid simulation

First, we need to go over the strengths and weaknesses of the grid-based and particle-based methods before describing the Hybrid method. In the grid-based method, each cell size is much larger than the size of a particle in the particle-based simulation.

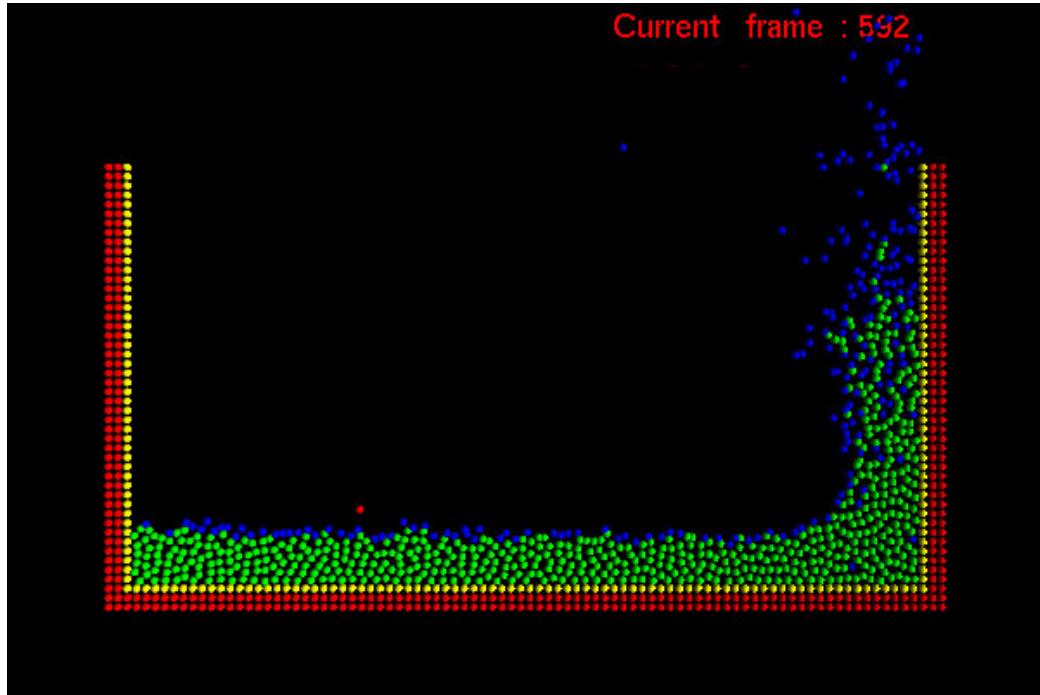


Fig. 6. Moving Particle Semi-implicit Method

The discretization and solution of the incompressibility condition is resolved in a unit cell and very simply compared to the corresponding step of a particle-based simulation such as MPS [29]. In MPS, the size of the linear symmetric matrix for solving the Poisson equation for pressure is n^2 , where n is the number of particles in the simulation. Even if only eight particles are used in a tentative unit cell for a 3D simulation, the size of the matrix will be significantly increased, causing the simulation to be very slow. This is because the projection step dominates the overall computation time. The grid-based method has difficulties in fluid advection, which is conducted using the Semi-Lagrangian method. The Semi-Lagrangian method allows us to take large time steps, but suffers from excessive numerical dissipation due to the accumulated interpolation error. Mass dissipation in a liquid simulation can cause critical problems in terms of the visual effect, since it dampens out interesting flow

features. Thus, the time step still needs to be limited by a CFL condition [7] for the liquid simulation.

On the other hand, in particle-based simulations, particles are the fundamental representation of fluid in the simulation and have their own physical properties such as mass, velocity, and density. Therefore, they let particles flow for themselves through the velocity field using forward Euler integration with excellent accuracy. The accuracy in the advection step allows the simulation to capture interesting flow features, but it has difficulty solving the incompressibility condition as presented above.

1. Particle-in-cell and fluid-implicit-particle method

Zhu and Bridson [37] introduced the hybrid fluid simulation approaches called Particle-in-Cell (PIC) [17], and Fluid-Implicit-Particle (FLIP) [5] to computer graphics in a paper simulating sand flow. Both approaches are fundamentally particle-based, as particles carry the momentum of the fluid, but a MAC grid is used for efficient computation of the spatial interactions required to compute diffusion and guarantee incompressibility. Using this auxiliary grid, incompressibility and boundary conditions can be enforced much more efficiently than in a pure Lagrangian scheme like MPS. In both PIC and FLIP, mass particles have their own velocity and position, which are integrated numerically using velocity updates obtained from the grid. Since the methods are Lagrangian, the velocity backtracing step of the semi-Lagrangian method can be avoided, greatly reducing numerical dissipation and loss of flow detail. Kim and Ihm [21] adapted these approaches to water animation, demonstrating realistic turbulent splashing without volume loss. The overall steps of the hybrid simulation are as follows,

- Construct an auxiliary MAC grid around the particles.

- Use the particle velocities to reconstruct a velocity field on the MAC grid.
- Integrate accelerations due to body forces such as gravity and other external forces.
- Integrate accelerations due to diffusion.
- Calculate the pressure field, and project the flow field onto the nearest divergence free field.
- Transfer velocity changes in the flow field back onto the particles.
- Advect each particle with its velocity through the flow field.

Figure 7 shows an example obtained using the standard hybrid simulation approach. The particles are drawn as black dots, and the grid cells are drawn in red.

E. Fluid simulation coupled with octree based structure

1. Combination of octree

Losasso and Fedkiw [24] proposed a technique which simulates water and smoke on an octree-based data structure using refinement and coarsening. The proposed technique shows how the poisson equation can be discretized on the Octree grid and allows us to use fast solution methods such as the conjugate gradient method with preconditioning. A cell is subdivided into more grid cells in regions of complex flow, such as at a nearby surface. This subdivision is performed from the larger cells to smaller cells. In regions of smooth flow, coarsening is performed from smaller cells to larger cells, which means that neighboring leaves are merged into a larger node and the resulting node values are deleted or unchanged in the octree structure. The authors were able to reduce the total simulation time to approximately 25 percent of

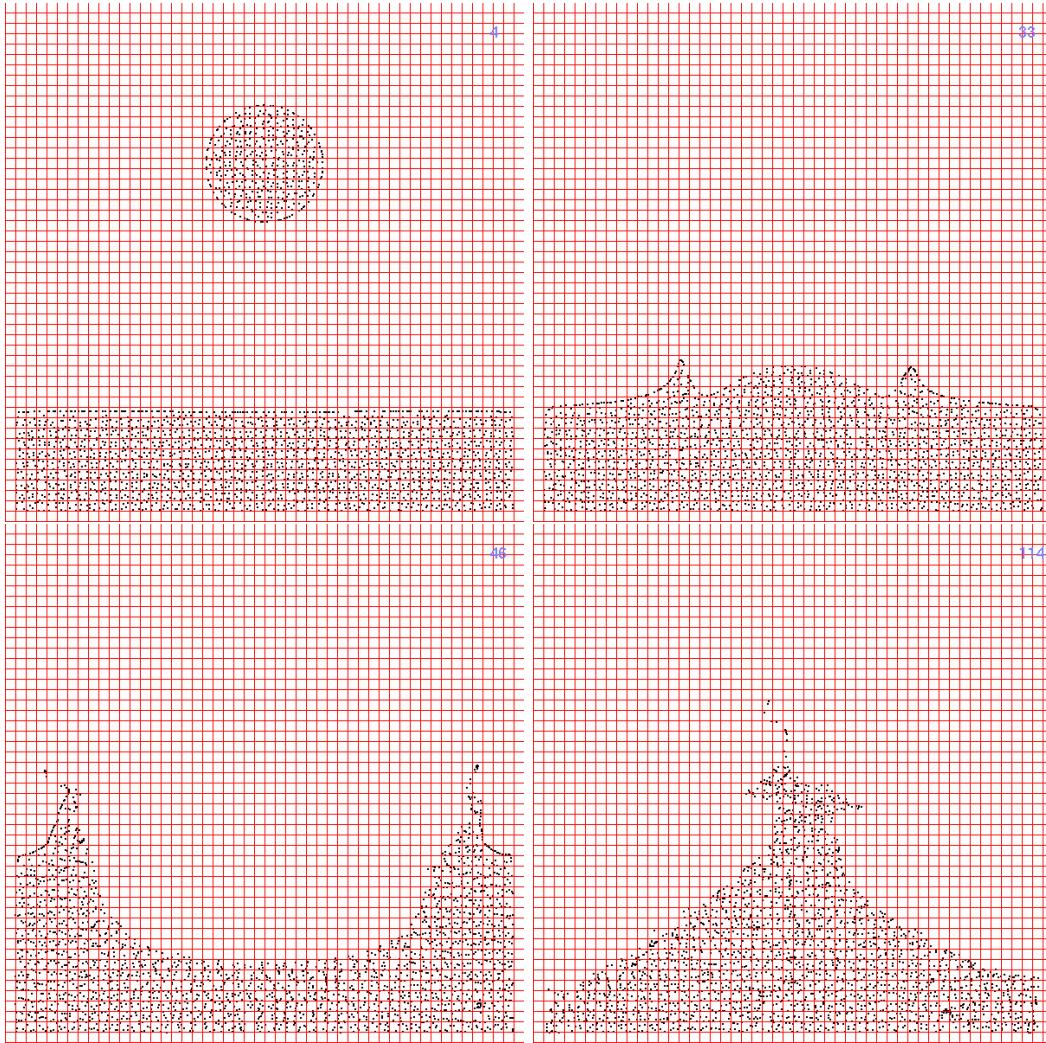


Fig. 7. Hybrid method

the original, and the finer sampling enabled them to capture fine detail of the flow in smoke and water. However, octree method [24] suffers from numerical dissipation due to repeated averaging and interpolation. The dissipation can be reduced in [16] using FLIP [37]. Inspired by their approach, I propose a fully adaptive technique which couples our adaptive particle approach with the Octree structure. It will be described in the next chapter. In this chapter, the detail algorithms are presented in order to apply an octree for fluid simulation.

2. Previous work

A number of authors have looked at adaptive methods for simulating fluid. Within the Eulerian simulation community, Losasso et al. [24] promoted the use of an adaptive octree data structure, instead of a fixed grid, to achieve high surface resolution in complex flows. More recently, Irving et al. [20] used a simpler idea, simply to coarsen cells with distance from the surface. In Lagrangian methods, early work by Desbrun and Cani-Gascuel [9] looked at splitting and merging particles in an SPH simulation to sample the fluid adaptively. Most recently, Adams et al. [2] have proposed an improved adaptive method for SPH compressible fluid simulation, and Hong et al. [19] demonstrated adaptive particle sampling within an incompressible framework.

3. Introduction to octree

Quadtrees and Octrees are tree data structures which are often used in computer graphics. They can be used to partition a region of space into 4 or 8 quadrants or octants. When a cell (i.e., a leaf node) contains an object boundary, the cell is subdivided into smaller cells, which enables us to save more of memory compared to methods that partition space uniformly when an object is represented in a region of space. The hierarchical tree structure is composed of a root node, intermediate nodes, and leaf nodes. Each node has a pointer to its own parent and children. There are three important traversal operations such as point location, region location and neighbor searches for querying and managing information. Each node's pointers are used for traversing the tree when these operations are performed. In this section, I will focus on efficient methods for performing these operations through a survey of the previous octree papers [31], [26] and [3].

4. Representation of quadtree and octree

Quadtrees and octrees are hierarchical tree data structures which branch from a root node to a leaf cell when traversing the tree. Each node has a pointer to its parent node and four child nodes, for a quadtree, or eight child nodes for an octree. A quadtree representation is assumed in this section. Figures 8 and 9 illustrate a quadtree data structure, where each node is labeled by its location code. Figure 8 shows the tree structure, and Figure 9 shows the corresponding spatial breakdown. The depth of the quadtree is set to n levels before generating. The level of the root node is set to: $ROOT_LEVEL = n - 1$, which is one less than the maximum depth of the tree. The level of the smallest size node is set to 0. As shown in Figure 9, the quadtree is defined over $[0, 1] \times [0, 1]$ (an arbitrarily sized space can be represented by simply prescaling). Each node has a location code, which is used as a search key to locate a cell. The location code is represented in a binary encoding form in which each bit corresponds to the branching pattern of the corresponding level in a tree. Figure 9 assumes that the quadtree has 6 levels and the level of the root is 5 (i.e., $ROOT_LEVEL = 6 - 1$). In order to determine the location code for each cell, the location of the cell's left is multiplied by $2^{ROOTLEVEL}$. For example, the cell $[0.375, 0.5]$ has a location code ($0.375 \times 2^5 = 12 = 001100$). In the case of a quadtree or an octree, the location code should be computed for each dimension.

5. Finding a leaf node from an arbitrary location

It is important to find the corresponding node in the octree from any arbitrary location. First, the coordinate values in a location are converted into locational codes by multiplying each coordinate value by $2^{ROOTLEVEL}$. The resulting products are truncated to integers. Finally, the integers are represented in binary form. For

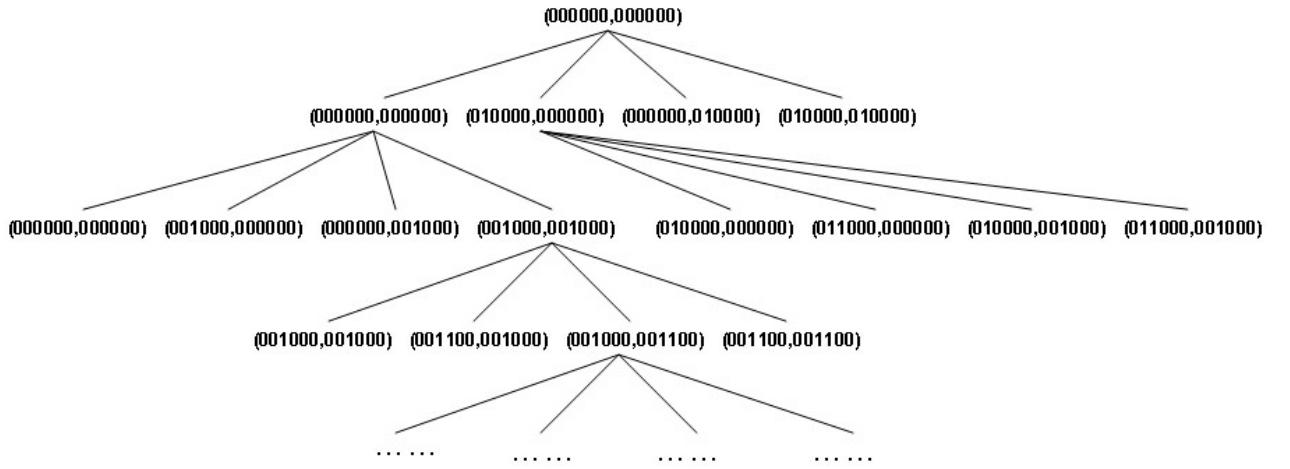


Fig. 8. Quadtree data structure and its representation

the example in Figures 8 and 9, the root level is 5, we multiply all values by $2^5 = 32$, or 6 binary digits. An arbitrary location at (0.44,0.32) is converted into binary ($\text{trunc}(0.44 \times 32) = 001110$) for x, binary ($\text{trunc}(0.32 \times 32) = 001010$) for y. This branching pattern of the locational code is used to traverse the tree from the root node to a leaf node, which is the desired cell that we are looking for.

6. Finding the smallest common enclosing cells by region

[26] and [3] present a method for finding the smallest possible enclosing cells. We use a method similar to [26], which determines the size of the smallest possible enclosing cells using an XOR operation, with left and right locational codes and top and bottom locational codes in a quadtree. For example, the region [0.38,0.45] for the x direction has left and right locational codes of 001100 and 001111. An XOR operation with the two locational codes yields 000011 and the smallest enclosing cell is found at level 2 because the first '1' bit is encountered at the 5th possible bit from the left. As a special case, let us see the following example in which the region [0.31,0.65] for the x direction has left and right locational codes as 001001 and 010101. The XOR

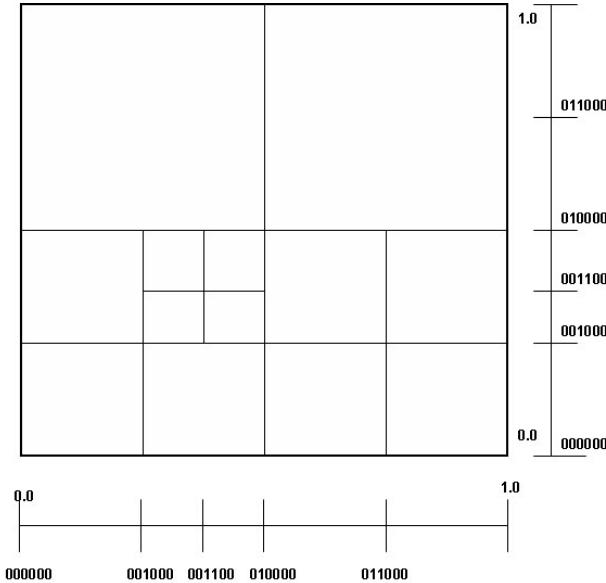


Fig. 9. Quad tree and locational codes

operation with the two locational codes yields 011100. The smallest possible enclosing cell is located at level 5 but the enclosing cell for the region is the root cell because the first '1' bit is the second bit from the left, meaning that the smallest common cell enclosing the location is not exactly matched with the cell.

7. Finding neighbors

Neighbor information is important when discretizing the Poisson equation when using an octree grid. This discretization requires exact information from the right, left, top, bottom, up and down neighbors in order to accurately compute pressure. As shown in Figure 10, the number of neighbors in each direction can be more than a single cell since the octree includes different sized cells as leaf nodes.

In order to determine the neighbors of a given cell, we note that the bit patterns of the locational codes of two neighboring cells differ by the binary distance between the two cells. The left boundary of every right neighbor of a cell is offset from

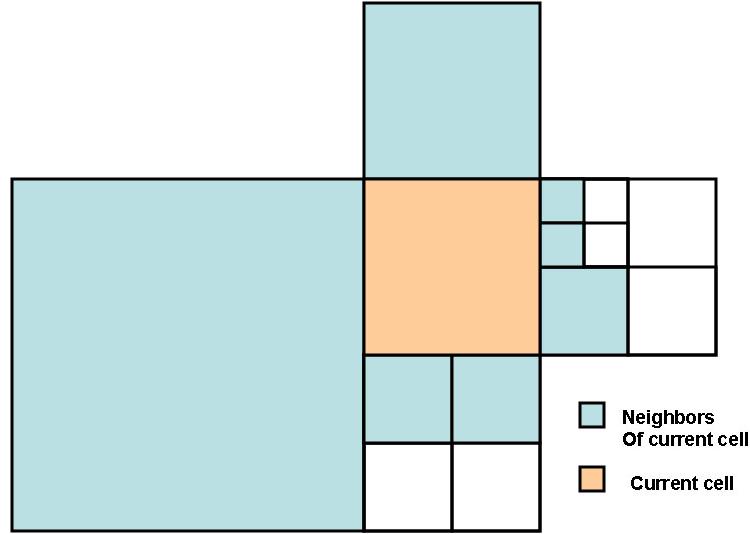


Fig. 10. The possible neighbors of a cell

the cell's left boundary. Thus, the x location code of every right neighbor can be determined by adding the binary form of the cell's size to the cell's x location code. For left neighbors, it can be found by locating the smallest possible left neighbor by subtracting the current cell's x location code from binary 1 and then traversing downwards from the smallest possible ancestor until the leaf node is reached. Using the bit pattern, all leaf nodes should be reached to obtain information about neighbors in each direction in order to discretize the Poisson equation.

8. Constructing the poisson equation

a. The pressure gradient

First, we derive the generalized form for discretizing the Poisson equation in terms of a two dimensional quad tree. The Laplacian of the pressure is defined at cell ij as

$$\nabla^2 p_{ij} = \frac{\frac{p_{i+1,j} - p_{i,j}}{\Delta x_0} - \frac{p_{i,j} - p_{i-1,j}}{\Delta x_1}}{\Delta x_2} + \frac{\frac{p_{i,j+1} - p_{i,j}}{\Delta y_0} - \frac{p_{i,j} - p_{i,j-1}}{\Delta y_1}}{\Delta y_2}, \quad (2.37)$$

where Δx_0 is the distance between $p_{i+1,j}$ and $p_{i,j}$ in the x direction, Δx_1 is the distance between $p_{i,j}$ and $p_{i-1,j}$ in the x direction, Δy_0 is the distance between $p_{i,j+1}$ and $p_{i,j}$ in the y direction and Δy_1 is the distance between $p_{i,j}$ and $p_{i,j-1}$ in the y direction. This can be rewritten as,

$$(\Delta x_2 \Delta y_2) \nabla^2 p = \frac{(p_{i+1,j} - p_{i,j}) \Delta y_2}{\Delta x_0} - \frac{(p_{i,j} - p_{i-1,j}) \Delta y_2}{\Delta x_1} + \frac{(p_{i,j+1} - p_{i,j}) \Delta x_2}{\Delta y_0} - \frac{(p_{i,j} - p_{i,j-1}) \Delta y_2}{\Delta y_1}. \quad (2.38)$$

As equation 2.38 shows, we need the pressure of the neighbor cells in order to construct the Laplacian used in the Poisson equation. Only four neighbors are considered on a MAC grid which is composed of uniform size cells but more than four neighbors may be needed on a quadtree based structure. For Figure 11, the cell enclosing p_4 is connected to five cells. We must consider three cases for discretizing the Poisson equation on this quadtree grid. In the first case, the size of the current cell is the same as the size of the neighbor. In the second case, the size of the current cell is bigger than the size of the neighbors, meaning that the current cell has several neighbors. In the third case, the size of the current cell is smaller than the size of the neighbors meaning that there is only one neighbor. The standard pressure gradient formulation is used when the cells are the same size. We have to concern ourselves with the gradient of pressure between smaller neighbors and the current cell and the gradient of pressure between a bigger neighbor and the current cell. We first consider the gradient of pressure p_4 with its smaller neighbors. According to [24], p_x is approximated as $(p_{52} - p_a)/(.75\Delta x)$ since the distance between p_4 and p_{52} is perturbed by a small amount proportional to the size of a cell, making $p_a = p_4$ to avoid the dependency of p_a yielding a convergent approximation. The denominator (.75 Δx) is

also approximated to the size of a large cell.

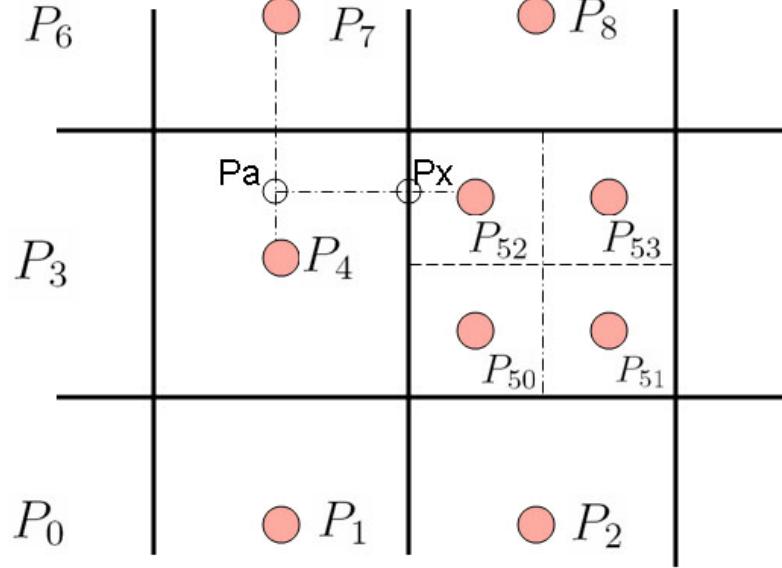


Fig. 11. Computing the pressure on a quadtree

In Figure 11, the equation of the pressure gradient for p_4 with its neighbor's pressure using this approximation is:

$$(\Delta x \Delta y) \nabla^2 p_4 = \frac{\frac{\Delta x}{2} (p_{50} - p_4)}{\Delta x} + \frac{\frac{\Delta x}{2} (p_{52} - p_4)}{\Delta x} + \frac{\Delta x (p_3 - p_4)}{\Delta x} + \frac{\Delta x (p_7 - p_4)}{\Delta y} + \frac{\Delta x (p_1 - p_4)}{\Delta y}. \quad (2.39)$$

The denominators are set to the size of the larger cell between the current cell and its neighbors. The numerators are set to the size of the smaller cells. When the current cell is larger than its neighbors, we can use the same method to approximate the pressure gradient, yielding

$$(\Delta x/2)(\Delta y/2) \nabla^2 p_{50} = \frac{(\Delta y/2)(p_{51} - p_{50})}{\Delta x/2} - \frac{(\Delta y/2)(p_4 - p_{50})}{\Delta x} +$$

$$\frac{(\Delta x/2)(p_{52} - p_{50})}{\Delta y/2} + \frac{(\Delta x/2)(p_2 - p_{50})}{\Delta y} . \quad (2.40)$$

The denominators are set to the size of the larger cell between the current cell and its neighbor and the numerators are set to the size of the smaller cell in equation 2.40.

The generalized form for all three cases is

$$A_{area} \nabla^2 p = \sum_j \left(\frac{p_j - p_i}{\Delta} \cdot n \right) L_{size} , \quad (2.41)$$

where A_{area} is the area of the current cell which is $\Delta x \Delta y$, j , the index of neighbors, n is the outward unit normal for the current cell, Δ is the size of the large cell, and L_{size} is the size of the current cell. The generalized form for a 3-dimensional octree is

$$V_{cell} \nabla^2 p = \sum_j \left(\frac{p_j - p_i}{\Delta} \cdot n \right) A_{area} , \quad (2.42)$$

where V_{cell} is the volume of the current cell, j the index of neighbors, n is the outward unit normal of the current cell, Δ is the size of the large cell and A_{size} is the face area of the current cell.

b. The divergence

The divergence of velocity in 2-dimensional uniform sized cells is

$$(\Delta x \Delta y) (\nabla \cdot u)_{ij} = \Delta y (u_{i+1,j} - u_{i,j}) + \Delta x (v_{i+1,j} - v_{i,j}) . \quad (2.43)$$

Following the same logic that was used for the pressure gradient in octree cells, the generalized form for the divergence term is

$$V_{cell} \nabla \cdot u = \sum_j (u_j - u_i) \cdot n A_{area} \quad (2.44)$$

where A_{area} is the area of a cell face, n is the outward unit normal of the current cell and V_{cell} is the volume of the cell in the fluid simulation based on the octree. Once the equation for divergence and pressure gradient are computed in all fluid cells, the linear system representing the finite difference approximation to the Poisson equation $\nabla^2 p = \nabla \cdot u$ can be constructed. An efficient iterative method such as preconditioned conjugate gradients can be applied to solve the system.

F. Surface reconstruction

In this section, we review several methods for surface reconstruction, which are used for Eulerian and Lagrangian fluid model.

1. Particle level set

The level set method is a tool for representing an evolving fluid surface. The zero level set of a signed distance field ϕ represents the fluid surface. Negative values of ϕ are considered fluid and positive values considered air:

$$\frac{\partial \phi}{\partial t} + \mathbf{u} \cdot \nabla \phi = 0, \quad (2.45)$$

where $\frac{\partial \phi}{\partial t}$ is a temporal partial derivative in the time variable t , \mathbf{u} is the given fluid velocity at time t and $\nabla \phi$ denotes the vector of spatial derivatives of ϕ . Once fluid velocity is updated at each time step by a solver, this equation is integrated by using the updated velocity. The first order Eulerian advection method , e.g. Upwind scheme can be applied in order to discretize the spatial terms $\mathbf{u} \cdot \nabla \phi$ in the equation,

which is extended for high-order spatial accuracy by using Hamilton-Jacobi (W)ENO method(a high order accurate Eulerian advection method) [28]. Also, the forward Euler method can be extended to a higher-order temporal discretization such as TVD Runge-Kutta method [28] in time, which is a high order time-integration method. The standard Eulerian advection algorithm provides enough accuracy for tracking the interface, but it is strictly restricted by a stability-based CFL criterion [7], which causes the simulation to be significantly slowed.

Enright et al. [10] proposed a particle level set method which exploits the combination of a Grid based semi-Lagrangian advection method and a characteristic-based particle method. A semi-Lagrangian method is not limited by a stability-based CFL condition but it suffers from a large amount of numerical dissipation. Massless marker particles near the interface are used to correct errors caused by back-tracing ϕ values for the duration of the simulation. Massless marker particles are placed in a thin layer inside and outside of the zero level set. Particles placed below the surface ($\phi < 0$) carry a negative sign and particles outside of the surface ($\phi > 0$) carry a positive sign. A band of thickness $\pm 6 \max(\Delta x, \Delta y)$ is fine on each side of the interface. The particles are integrated forward in time using the fluid velocity with a second order accurate midpoint rule. The level set function is then advected with a semi-Lagrangian method. Finally the particles are used to correct any errors in the function. When particles appear on the wrong side of the interface (a negative particle in the $\phi > 0$ or a positive particle in the $\phi < 0$) by a distance more than its radius, it is said to be have escaped. In this case, the level set function is corrected as follows,

$$\phi_p(x) = s_p(r_p - |x - x_p|), \quad (2.46)$$

where x is a sampling point on the computational grid, x_p is the Particle position and s_p is the sign of the particle. For error correction, the minimum magnitude for the

grid sample is used to replace the existing level set value. Overall steps for using the particle level methods are as follows:

- Evolve the marker particles and level set function in time.
- Correct errors in the level set function using particles.
- Reinitialize ϕ using the fast marching method.
- Correct errors in the level set function using particles.
- Adjust the particle radii using the following rule:

$$r_p = \begin{cases} r_{max} & \text{if } s_p\phi(x_p) > r_{max} \\ s_p & \text{if } r_{min} \leq s_p\phi(x_p) \leq r_{max} \\ r_{min} & \text{if } s_p\phi(x_p) \leq r_{min} \end{cases} \quad (2.47)$$

a. Reinitialization

Once the level set function is updated, the signed distance field may no longer be accurate. Reinitialization of the level set function is necessary to recover the signed distance field, which is important for visualizing fluid and extending the velocity field outward from the interface. There are two approaches for solving the Eikonal equation [36] for reinitialization. One approach is to move the zero isocontour in the normal direction at times equal to the distance from the interface. In this approach, many iterations are necessary until a steady state is reached. The other approach is a discrete algorithm that mimics the crossing-time approach to update the signed distance field one grid point at a time. This is known as the Fast Marching Method [32]

which needs to solve the Eikonal equation [36] to construct a signed distance field. When marching out, each grid point is updated with the appropriate value of signed distance. The final minimization is computed over all quadrants. Three conditions are considered to solve the quadratic equation. First, there is one neighboring point which is not equal to ∞ in the spatial dimension. In this case, two of the three terms can vanish since the current grid point is chosen as the minimum as follows:

$$\left(\frac{\phi_{i,j,k} - \phi_1}{\Delta x}\right)^2 = 1 \quad (2.48)$$

The larger term $\phi_s + \Delta x_s$ is chosen when marching out in the normal direction to construct the positive distance function. When there are two nonzero terms, the equation is:

$$\left(\frac{\phi_{i,j,k} - \phi_1}{\Delta x_i}\right)^2 + \left(\frac{\phi_{i,j,k} - \phi_2}{\Delta x_j}\right)^2 = 1 \quad (2.49)$$

If $P(\max\{\phi_1, \phi_2\}) > 1$, then the larger ϕ is ignored and we proceed with $\left(\frac{\phi_{i,j,k} - \phi_s}{\Delta x}\right)^2 = 1$. Otherwise, the above quadratic equation is solved and the larger one is used as the solution. When there are three nonzero terms, it follows the same way, after checking if $P(\max\{\phi_s\})$ is greater than 1 or not. The equation in this case is:

$$\left(\frac{\phi_{i,j,k} - \phi_1}{\Delta x_i}\right)^2 + \left(\frac{\phi_{i,j,k} - \phi_2}{\Delta x_j}\right)^2 + \left(\frac{\phi_{i,j,k} - \phi_3}{\Delta x_k}\right)^2 = 1 \quad (2.50)$$

The time complexity of the FMM is of order $O(N \log N)$, where N is the number of grid points and $\log N$ is the height of the tree for the heapsort algorithm. The method was originally proposed by Tsitsiklis. FMM is generally considered a fast method but the extra memory assignment is necessary for heapsorting and it is not simple to implement.

There is one more iterative algorithm for solving the Eikonal equation [36], called

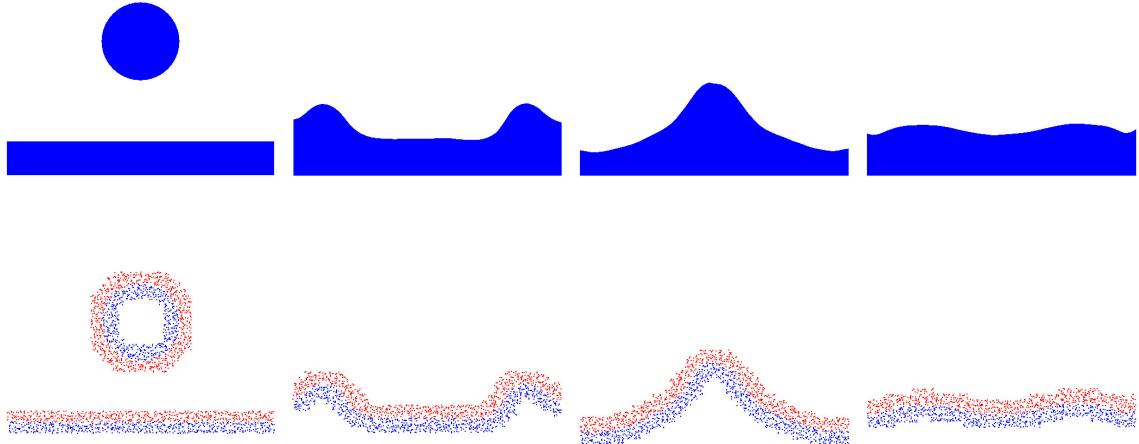


Fig. 12. Fluid simulation and particle level set

the fast sweeping method [36]. The main idea of the fast sweeping method is to use the Godunov upwind differencing scheme [28], as we did in FMM, and Gauss-Seidel iteration with alternating sweeping. First, check the neighbors in the current grid point that have the closest distance from the surface. If the closest distance is smaller than the distance of the current grid point, update the distance. We sweep the whole domain with four alternating orderings(in two dimensions)as follows,

$$(1)i = 1 : I, \quad j = 1 : J, \quad (2)i = I : 1, \quad j = 1 : J, \\ (3)i = I : 1, \quad j = J : 1, \quad (4)i = 1 : I, \quad j = J : 1. \quad (2.51)$$

A finite number of iterations is needed to reach a steady state and the complexity of the algorithm is $O(N)$, where N is the number of grid points. It is extremely simple to implement and fast compared to the Fast Marching method. Figure 12 shows a 2D example of surface extraction using a particle level set, for a fluid simulation. The

top row shows the extracted surface. The bottom row shows the particles inside the surface in blue, and outside the surface in red.

2. Blobbies

The basic concept of blobbies is introduced in this section. Blinn proposed the idea of blobbies in his early work [4]. Blobbies model electron density maps of molecular structures, and thus are well suited to surface reconstruction from particles. For these reasons, it is used as a basic mathematical model for constructing surfaces in Lagrangian fluid simulations.

The electron in an atom can be represented as a density function in Quantum mechanics as

$$D(x, y, z) = \exp(-ar) \quad (2.52)$$

where r is a distance between the spatial location and a hydrogen atom, and a is the standard deviation. Polynomial density functions can be used as an alternate definition of density functions, even though Blinn used a Gaussian distribution. When it is applied to a large number of atoms, the density function can be represented by summing the contribution from each atom as follows:

$$D(x, y, z) = \sum_i D_i(x, y, z) \quad (2.53)$$

A surface can be chosen by a threshold T as follows,

$$S = \{(x, y, z) \mid D(x, y, z) = T\} \quad (2.54)$$

The electron densities are greater than T inside the surface and less than T outside the surface. The resulting surface of this model is naturally blobby. Figure 13 shows our result for blobbies .

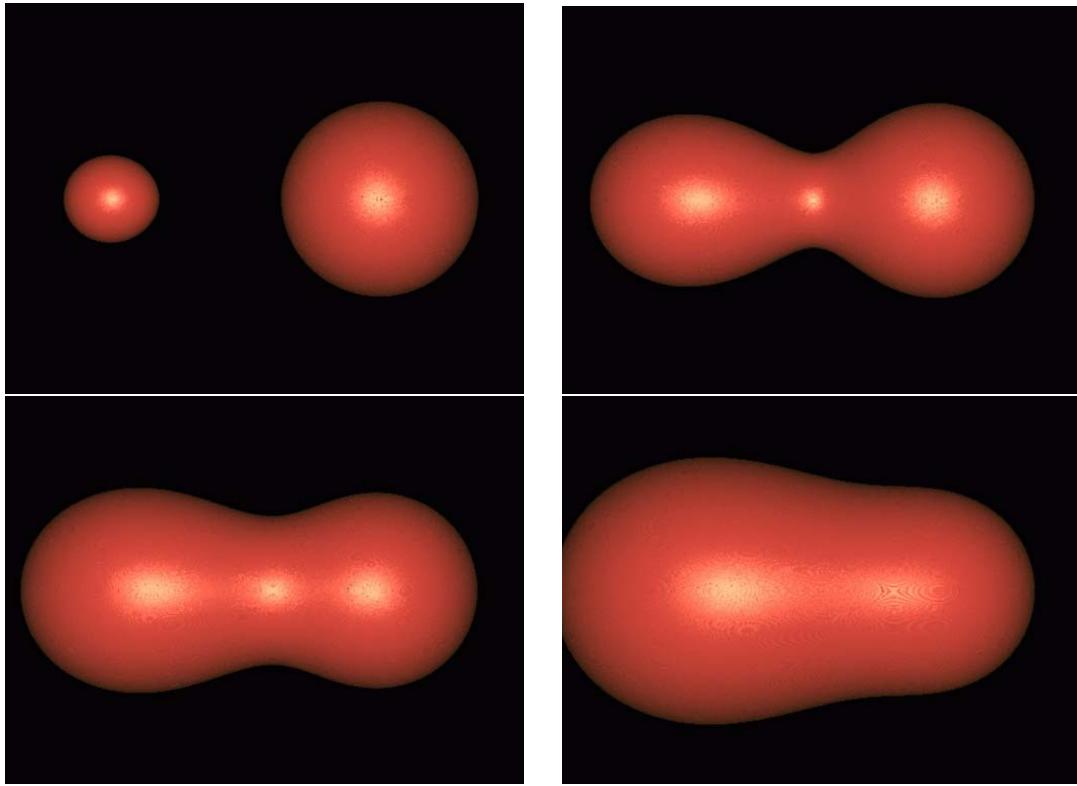


Fig. 13. Blobbiness

3. Bridson's method

More recently, Bridson et al [37] proposed a smoothing technique which uses a moving average. In order to construct a signed distance field from particles, he used the following equations:

$$\phi(x) = |x - \bar{x}| - \bar{r} \quad (2.55)$$

$$\bar{x} = \sum_i w_i x_i \quad (2.56)$$

$$\bar{r} = \sum_i w_i r_i \quad (2.57)$$

$$w_i = \frac{k(|x - x_i|/R)}{\sum_j k(|x - x_j|/R)} \quad (2.58)$$

where $k(s) = \max(0, (1 - s^2)^3)$, x_i and r_i are the position and radius of particle i , and R is the radius of the neighborhood. Unfortunately, his model produces spurious blobs due to the moving average in concave areas and can remove small detailed features near the surface. Another method similar to this is found in [Adams07].

CHAPTER III

ADAPTIVE PARTICLES¹

A. Introduction

It is well known that the complexity of the flow of fluids, especially water, differs depending upon the region of the flow. For example, in large bodies of water, surface flow is generally much more complex than the flow deep below the surface. This concept has been used in Eulerian fluid simulation when developing schemes for grid refinement for detailed behavior and coarsening for efficiency, as in [20]. In a Lagrangian fluid simulation, refinement and coarsening can be done by merging and splitting the particles representing the mass of the fluid. For example, within a Smoothed Particle Hydrodynamics (SPH) simulation, Desbrun and Cani [9] used simplification and refinement of particles to animate soft bodies such as lava. Particle size adaptation is attractive as a simulation mechanism, since particles provide an efficient means for capturing the small-scale detailed behavior near the surface of a fluid. Even simulations based on an Eulerian grid typically use particles to restore surface detail lost due to smoothing [11]. In Figure 14, we show examples from an adaptive particle-based fluid simulation, where very fine details of the surface are visible, while in the deep areas, particle merging has been used to avoid oversampling and lend efficiency to the simulation. Although the idea is attractive, it has, to date, only been reported for compressible fluid simulation.

¹Reprinted with permission from "Adaptive particles for incompressible fluid simulation" by Woosuck Hong, Donald H. House and John Keyser, 2008. The Visual Computer, Volume 24, 535-543 ,Copyright[2008] by Springer Berlin / Heidelberg

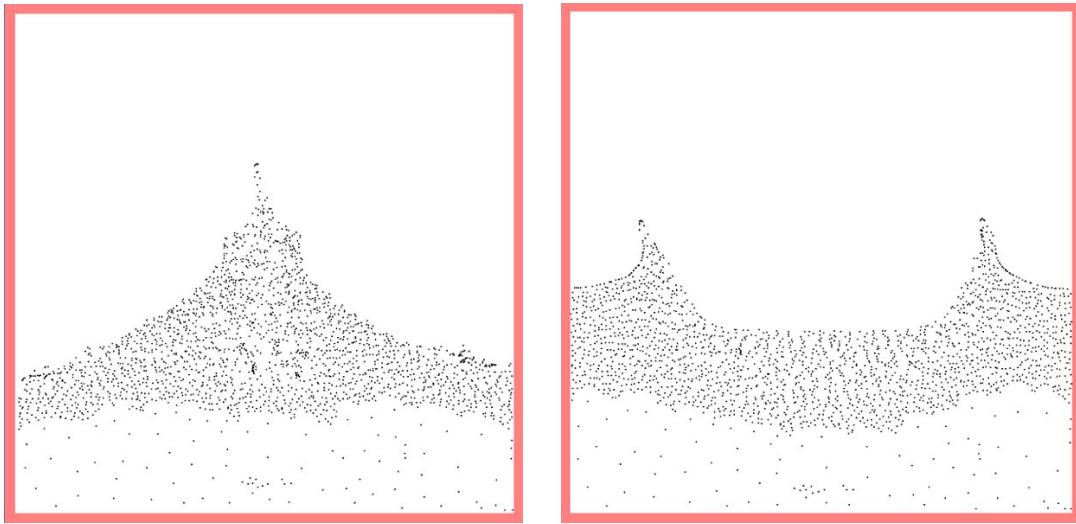


Fig. 14. Fluid simulation with adaptive particle sampling

1. Our contribution

We propose an efficient incompressible Lagrangian fluid simulation method utilizing splitting and merging of particles to adaptively sample the evolving fluid motion. The approach uses the hybrid FLIP method [37]. This is an extension to the original work of Desbrun [9], and more recently Adams et al. [2], in which refinement and simplification were performed for compressible flows based on the SPH approach.

The fact that we model incompressible flow makes our method much more suitable than SPH for simulating water. Water is well known to be nearly incompressible, and this fact contributes greatly to its visible behavior. For example, water being poured into a cylindrical glass would maintain an average height proportional to the volume of fluid in the glass. On the other hand, the height of the surface of a compressible fluid would exhibit damped periodic oscillations, giving it an unnatural springy look. To confirm this, one can compare animations in support of the incompressible Particle Level Set Method [11], and those in support of the compressible SPH method [27].

In order to determine the level of particle sampling needed in a region of fluid, we quantify what we call the *deformability* [9] of each region of the fluid and then base particle splitting and merging on this measure. Our method for finding the deformability of a fluid region considers both depth or distance from the surface, and the local Reynolds number [30]. Depth is determined by constructing a signed distance field from the fluid interface as determined by particle positions. The local Reynolds number is estimated on the computational grid, using a ratio of terms from the Navier Stokes equations.

The use of both depth and Reynolds number to obtain a deformability measure allows us to provide the necessary detail to represent both surface shape and complex turbulent flow below the surface, while avoiding over computation in areas where the flow is smooth. Splitting is done in highly deformable areas, and merging in areas of low deformation. With merging, we do not need to assign particles to every grid cell, since the effective radius of a particle can be bigger than the cell width, thus we do not need to worry about occasional gaps in the flow of fluid. Significant memory and computing time savings come from reducing the number of particles in the simulation. We have been able to reduce the number of particles by more than 80% of the number that would have to be used without merging, while producing results that are visually indistinguishable. Likewise, the speed of the simulation is improved significantly.

By using multiple particle sizes, we can also better decouple the size of the auxiliary grid from the size of the particles themselves. The auxiliary grid can be used in the computation of external forces (such as finding curvature and normal vectors when computing surface tension). By supporting multiple particle sizes, we can allow for better computation of these external forces without extra computational cost.

B. Previous works

Eulerian fluid simulations generally solve the Navier-Stokes equation over a grid. Foster and Metaxas [14] were the first in the computer graphics community to achieve full 3D liquid simulation based on the Navier-Stokes equations for incompressible flow. Their simulation method was based on Harlow and Welch's original Marker and Cell (MAC) method [17]. They used a fixed finite-difference grid for computing partial spatial derivatives, an explicit integration scheme for advection and diffusion, a relaxation method for incompressibility, and massless marker particles for surface representation. Stam[34] made significant improvements on the stability of this approach using a semi-Lagrangian scheme to handle velocity advection, implicit integration to handle diffusion, and a Poisson solver for computation of pressure to achieve divergence free flow. Foster and Fedkiw [12] and Enright et al. [11][10] extended Stam's method to handle the fluid-air interface using level set methods augmented by marker particles.

In addition to the above work, several extensions have been attempted to the basic Eulerian method. Greenwood and House [15] simulated fluid with the visual effect of bubbles, Song [33] attempted to reduce the numerical diffusion and dissipation caused by the semi-Lagrangian method by adopting a constrained interpolation profile-based advection scheme. Hong and Kim [18] developed a numerical method to resolve the discontinuities of the interface between two fluids by using the interpolated pressure field and velocity gradient.

Particles are the fundamental momentum-carrying simulation element in Lagrangian simulations. In SPH [25], the fluid is composed of a set of particles, with inter-particle forces such as pressure and viscosity computed at the position of a particle by a smoothing kernel. Desbrun and Cani-Gasculel [8] introduced SPH to

computer graphics to simulate highly deformable substances such as lava flow. In [9] refinement of particles by splitting and merging allowed sampling the fluid adaptively. The basic concept of the adaptive sampling scheme for Lagrangian fluid corresponds to Geoffrey’s idea [20] who used refinement near the interface for detailed representation and coarsening of fluid cells away from the interface for efficiency in Eulerian fluid. Müller et al. [27] extended the SPH approach for interactive fluid simulation, and introduced a new technique for complex behaviors, such as air-water interaction, boiling water, and trapped air. In a recent paper, Adams et al. [2] have proposed an improved adaptive method for SPH compressible fluid simulation. Though aspects of this paper are similar to ours, we emphasize that our approach handles incompressible fluid, and is in fact the first adaptive particle method to do so.

Premoze [29] introduced Koshizuka’s original work [22] on the Moving-Particle Semi-Implicit Method (MPS) to simulate incompressible liquid and multifluid flow. Unlike SPH, MPS handles incompressible fluid. However, the approach is slow, tends towards instability, and is not adaptive. Clavet [6] also extended the formulation of SPH by using a method called double Density Relaxation for enforcing incompressibility and to oppose clustering. Several small-scale simulations of substances such as mud and paint were created realistically with surface tension using springs between particles.

Zhu and Bridson [37] introduced the hybrid fluid simulation approaches called Particle-in-cell (PIC) [17], and Fluid-Implicit-Particle (FLIP) [5] to computer graphics, in a paper simulating sand flow. Both approaches are fundamentally particle-based, as particles carry the momentum of the fluid, but a MAC grid is used for the efficient computation of the spatial interactions required to compute diffusion and to guarantee incompressibility. Using this auxiliary grid, incompressibility and boundary conditions can be enforced much more efficiently than in a pure Lagrangian

scheme like MPS. In both PIC and FLIP, mass particles have their own velocity and position, which are integrated numerically using velocity updates obtained from the grid. Since the methods are Lagrangian, the velocity backtracing step of the semi-Lagrangian method can be avoided, greatly reducing numerical dissipation and loss of flow detail. Kim and Ihm [21] adapted these approaches to water animation, demonstrating realistic turbulent splashing without volume loss.

C. The adaptive fluid simulation algorithm

Our fluid simulation is based on the FLIP method, where particle velocities are transferred to a staggered Marker-and-Cell (MAC) grid, body and diffusion forces are applied, the divergence free property of incompressible fluid is enforced by solving a Poisson equation to obtain a pressure field, and velocities are corrected based on the gradient of the pressure field. The resulting velocity changes are transferred to the fluid mass particles and their positions are integrated through the velocity field. An auxiliary grid is recreated at each time step, serving a computational role only in reconstructing the flow field from the particles. This allows us to avoid costly inter-particle interaction calculations, and provides a more natural field representation for computing pressure.

We have augmented this approach to support merging and splitting of particles. Our overall scheme thus becomes:

- Construct an auxiliary MAC grid around the particles.
- Use the particle velocities to reconstruct a velocity field on the MAC grid.
- Integrate accelerations due to body forces such as gravity and other external forces.

- Integrate accelerations due to diffusion.
- Calculate the pressure field, and project the flow field onto the nearest divergence free field.
- Transfer velocity changes in the flow field back onto the particles.
- Compute a fluid deformability measure based on depth and Reynolds number.
- Split or merge the particles based on the local deformability measure.
- Convect each particle with its velocity through the flow field.

When particle velocity is transferred to the auxiliary grid, the weight of a particle's effect is determined by using a spherical kernel function whose volume is proportional to the mass of fluid represented by the particle.

D. Computing deformability

Splitting and merging are adaptively performed based on what we call the fluid's local deformability. In a local region, our deformability measure is based both on depth from the surface, and on the Reynolds number. Since the Reynolds number is the ratio of convection to diffusion, our method requires the fluid to have non-zero viscosity.

1. Distance from surface

Intuitively speaking, the highly deformable area in a fluid is mainly near the surface, i.e. the interface with another fluid. Areas away from this interface tend to have smoother flow and are thus less deformable. Further, we wish to be able to reconstruct a smooth surface for rendering, and so we want to be sure to sample finely near

the surface. Therefore, we use distance from the surface as our basic criterion for deformability. We compute an implicit representation of the fluid surface directly from the particles, and then use the Fast Sweeping Method [36], which is $O(n)$ in time to construct a signed distance field. This is more easily implemented and faster than the Fast Marching Method [1], which is $O(n \log n)$. Then, the distance from the surface can be computed anywhere inside the fluid using trilinear interpolation.

2. Reynolds number

In 1883, Osborne Reynolds defined the terms laminar and turbulent flow to describe his experimental results with fluid motion [30]. He proposed the use of a number Re , which is the ratio of inertial forces to viscous forces. This number has come to be known as the Reynolds Number. In fluid mechanics, the Reynolds number can be used to identify highly deformable areas of fluid. It is given by

$$Re = \frac{v_s L}{\nu} = \frac{\text{Inertial forces}}{\text{Viscous forces}} \quad (3.1)$$

where v_s is the mean fluid velocity, L is the characteristic length, and ν is the kinematic fluid viscosity. It is known that the flow rate of turbulence is proportional to the square root of the pressure gradient. That is, high deformation occurs when a high pressure gradient is present, since this will cause high accelerations. We can estimate the square root of the pressure gradient by computing a gradient from the convection term of the Navier-Stokes equation via differencing on the associated grid. We estimate a “deformability factor” roughly proportional to Re ,

$$Df = \frac{(u \cdot \nabla) u}{\nu \nabla^2 u} \quad (3.2)$$

by taking a ratio of the convection term and the diffusion term of the Navier Stokes equation 2.7.

E. Determining when to merge and split

To determine where it is appropriate to merge and split particles, we divide our simulation domain into a set of layers, based on the distance to the surface. The basic process is illustrated in Figure 15. The number and placement of these layers is somewhat arbitrary; we want to choose layers that allow detailed simulation (small particles) where needed, and less detail (large particles) where acceptable. Numerous possible schemes could be developed relying on the distance from surface and deformability factor. We describe here a four-layer approach that has proven effective in practice.

Note that in all cases, we must set some minimum and maximum bounds to limit the amount of splitting or merging allowed. Splitting must be limited in order to avoid excessive memory space and noise. Merging must be limited in order to avoid excessive smoothing of particle motion.

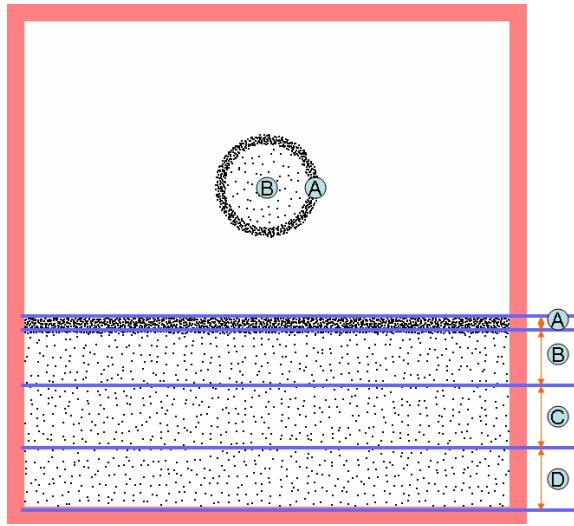


Fig. 15. Layers for Splitting and Merging

Figure 15 shows the four layers into which we divide the fluid. These layers represent distance from the fluid surface, which is not necessarily the same as depth.

Layer A represents the region closest to the fluid/air interface. We desire the highest resolution in this region, and as a result, we want to use particles of the minimum volume possible. Any particles that enter this region are immediately split to the minimum allowable particle size. This ensures that there is always a highly detailed representation of the surface.

In *Layer B*, all particles are of a nominal, or standard size, which is larger than that in layer *A*. When smaller particles move into layer *B*, merging will be performed, and when larger particles move into this area, splitting will be performed.

Layer C and *Layer D* allow both splitting and merging, based on the deformability measure. The choice of whether to split or merge in these regions is based solely on the deformability factor. Particles are split in higher deformability factor regions and are merged in lower deformability factor regions. Note that instead of layers, one could also easily define an approach where the deformability factor thresholds for merging and splitting are a continuous function of distance from the surface.

F. Splitting and merging

Figure 16 shows particle splitting on the left, and particle merging on the right. This process must preserve mass, which, for a constant density incompressible fluid, is equivalent to conserving volume, as indicated by the V labels on the figure. One stage of particle splitting splits a single particle into two particles, each of half the volume of the original particle. Particles within some sphere of radius r are merged into a single particle whose volume is the sum of the original particles. Momentum is conserved using different approaches for merging and splitting.

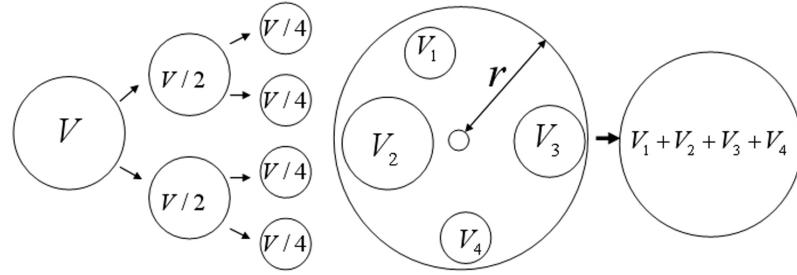


Fig. 16. Mass and volume conservation when splitting and merging particles

1. Merging

When merging a group of particles, the new particle is placed at the center of gravity of the original particles. The mass, radius, position and velocity of a newly created particle i from neighbor particles are given by

$$m_i = \sum_j m_j, r_i = \sqrt[3]{\sum_j r_j^3}, x_i = \frac{\sum_j V_j x_j}{\sum_j V_j}, u_i = \frac{\sum_j V_j u_j}{\sum_j V_j}. \quad (3.3)$$

2. Splitting

Contrary to merging, splitting of a particle is performed by generating a new set of n particles within the radius of the parent particle. We currently use $n = 2$ but this is arbitrary. Particle mass is distributed evenly among the child particles, and the velocities are simply copied from the original particle, thus conserving momentum. The mass, radius, and velocity of each new particle j , generated by splitting particle i , are given by

$$m_j = \frac{m_i}{n}, r_j = \sqrt[3]{\frac{r_i^3}{n}}, u_j = u_i \quad (3.4)$$

There are multiple reasonable ways to determine position x_j of the new particles. We choose to place it randomly at a position within the radius of the original particle, using spherical coordinates.

3. Upper and lower bounds

Note that bounds on minimum and maximum size can be set per particle. These maximum and minimum bounds can be set layer by layer (or change continuously) if desired. In practice, we adjust these by layer.

We limit the maximum size of a merged particle so that its radius should not extend beyond the fluid/air interface. A smaller maximum volume is set near the surface, and a larger maximum volume is used in the deep areas of the fluid. The maximum volume is interpreted as an upper limit, beyond which no more merging will occur. We also limit the minimum size of the particles, so that there is not excessive splitting. This minimum volume limit is interpreted as a bound for which no particles will ever be split if it would result in a particle lower than that bound.

Note that these limits restrict the splitting and merging *process*, rather than the sizes of the particles themselves directly. For example, a small particle (below the “minimum” radius) might enter a region, and be unable to merge with any other particle without exceeding the maximum value. In this case, the smaller particle would remain. Likewise, a particle that exceeds the “maximum” might enter a region and be unable to split, since splitting might reduce it below the minimum bound. In our implementation, for example, both of these can occur in layer B .

G. Kernels and the auxiliary grid

When the particle velocities must be reconstructed in the auxiliary grid, a kernel function is used around each particle to determine its contribution to a grid point. Each particle is given a radius that determines the volume of fluid it represents. Each cell point that is within a particle’s radius is affected by that particle by an amount determined by distance and the kernel function shape. We have tested two kernel

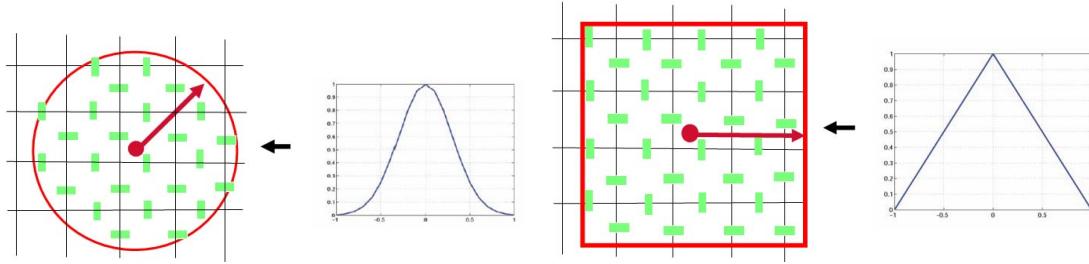


Fig. 17. Gaussian and tent kernel functions

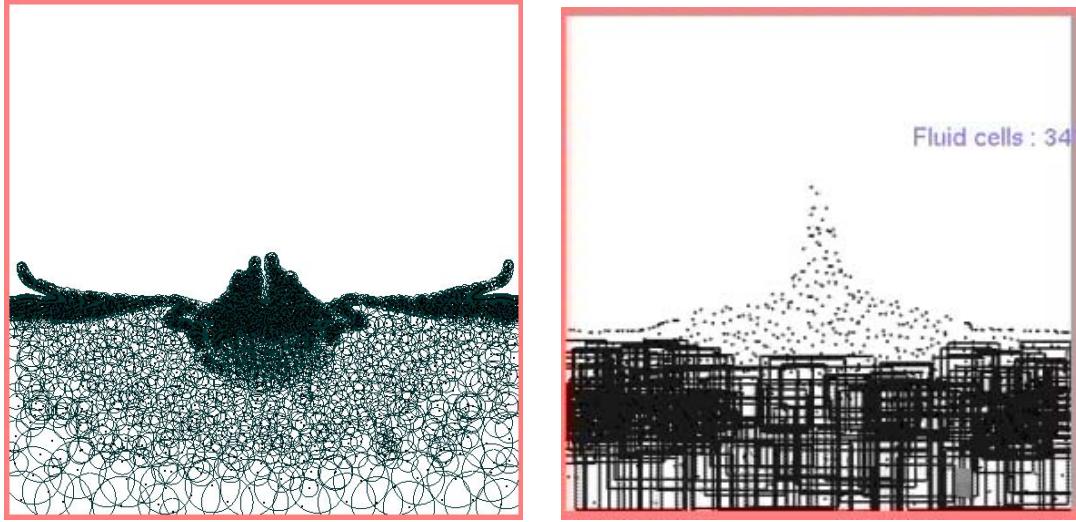


Fig. 18. 2D simulation using Gaussian and tent kernel functions

functions, Gaussian and tent. Illustrations of these two kernel functions are given in Figure 18. We tested the Gaussian kernel over a circular region, and the tent kernel over a square region, as indicated as shown in Figure 17. For computational efficiency, we prefer the *tent* kernel function, which is implemented by trilinear interpolation. This simplification does not produce a noticeable effect on the simulation accuracy. Our 2D simulations for Gaussian and tent kernels are shown in Figure 18

The size of the auxiliary grid can be set independently of the size of the particles themselves. In practice, we usually use a grid cell spacing equal to the “nominal” particle radius of layer B . We will call the grid cell width d_g .

To determine velocity values on the grid, we use different schemes depending on the radii of the nearby particles. If the particle radii are generally greater than or equal to d_g (as is generally the case in layers B through D in our implementation), each sample point on the grid will potentially be affected by multiple particles. In this case, we simply weight the contribution of each particle at the sample point by the value of the kernel function:

$$u_{cell} = \frac{\sum_i W_i(x_{cell}; r_i) u_i}{\sum_i W_i(x_{cell}; r_i)}, \quad (3.5)$$

where the W_j for particle i is defined either radially:

$$W_i(x; r) = K(|x - x_i|; r), \quad (3.6)$$

or trilinearly:

$$W_i(x; r) = K((x - x_i) \cdot \vec{x}; r) K((x - x_i) \cdot \vec{y}; r) K((x - x_i) \cdot \vec{z}; r). \quad (3.7)$$

Note that $K(d; r)$ is the value of the kernel function of radius r at distance d .

As shown in Figure 19, when particle radii are near to or smaller than d_g , the sample point(red dot) might not lie within the volume of any particle(pink dots), and thus the weighted average described above is not applicable. To avoid such cases, for each sample point we check whether there is a particle with radius less than d_g , within a distance of d_g (in each dimension). If there are such particles, we determine the velocity by taking a volume-weighted average of all such cells, using a d_g radius kernel function for each:

$$u_{cell} = \frac{\sum_i V_i W_i(x_{cell}; d_g) u_i}{\sum_i V_i W_i(x_{cell}; d_g)}, \quad (3.8)$$

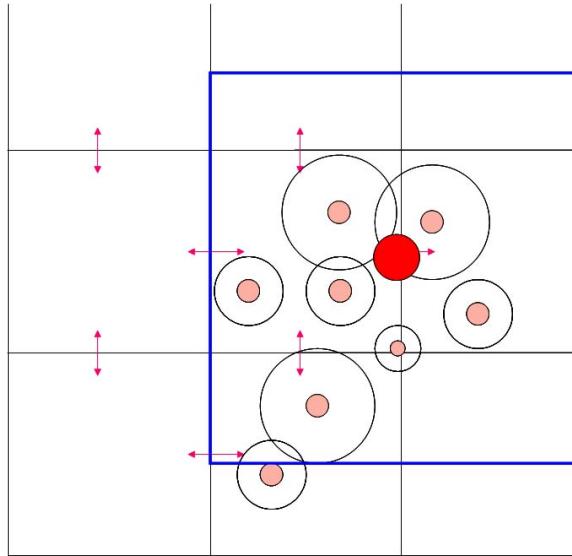


Fig. 19. Sampling of particles with small radii

where V_i is the volume of particle i .

H. Results and evaluation

1. Experimental results

We have implemented the above process to simulate in both 2 and 3 dimensions. Note that the 2D simulation requires slightly different but straightforward reformulation of some of the listed equations. Examples from some 2D simulations are shown in Figure 20. Table I lists the parameters used in our experiments, including the distance of each layer from the surface, relative to the radius of the “standard” particles in layer B , the deformability factor values used to determine when splitting and/or merging occur, and the minimum and maximum bounds to use when splitting and merging. For each layer, columns give the distance from surface range (upper and lower bound), the deformability factors above which splitting occurs and below which merging occurs, and the minimum and maximum particle radii (relative to the nom-

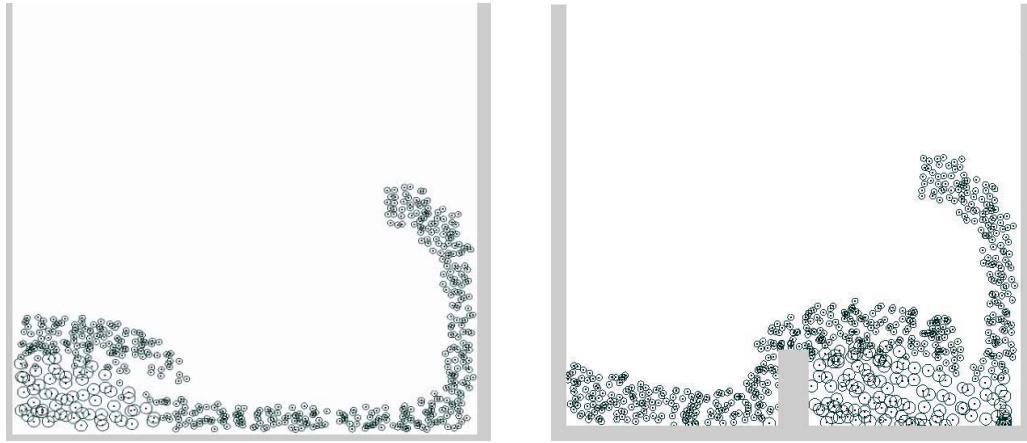


Fig. 20. Fluid simulation with adaptive particle sampling

inal size) for splitting and merging. The auxiliary grid was set with grid cell size d_g equal to the nominal radius of particles in layer B . For all calculations, constants typical of water were used. These are are kinematic viscosity $\nu = 8.90 \cdot 10^{-7} m^2/s$, with density $\rho = 1000 kg/m^3$. The 2 dimensional water simulation shown in Figure

Table I. Parameters used in experiments

Layer	Lower	Upper	Df	Df	Split	Merge
	Bound	Bound	Split	Merge	Min	Max
A	0	1	-	-	0.25	0.25
B	1	4	-	-	1	1
C	4	7	0.3	0.2	3	6
D	7	-	0.1	0.05	5	10

21 was sampled with 3,545 particles on an auxiliary 30×30 grid. Green, blue and red particles are used for split, standard (radius 1) and merged particles. Splitting and merging were performed based on our algorithm for finding the deformable area.

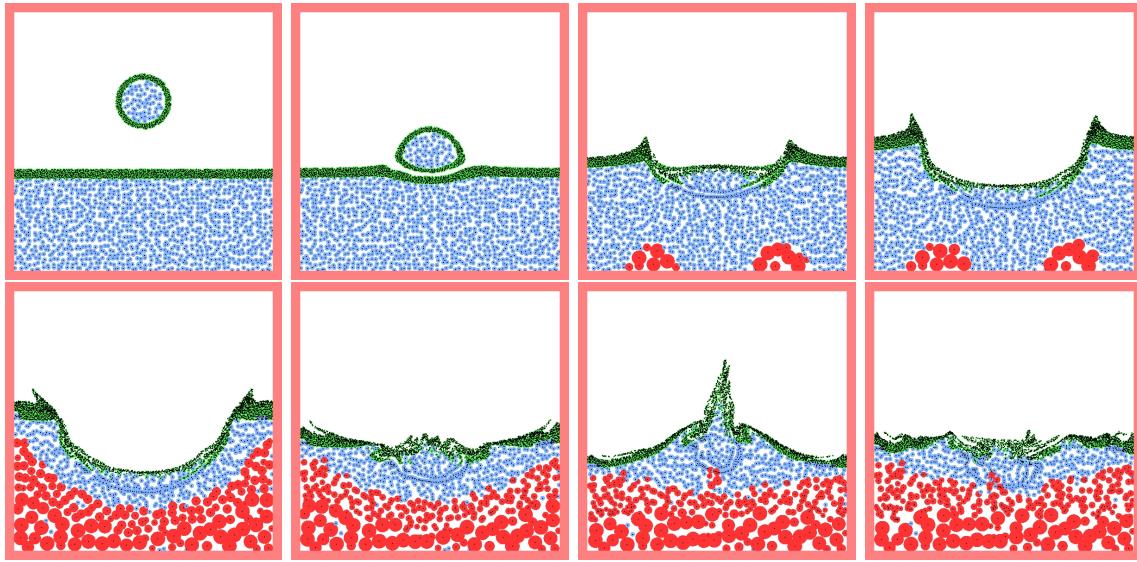


Fig. 21. 2D simulation for splitting and merging

The total number of particles was reduced by as much as 61% (to 1,372 particles) by merging during the simulation. The split particles near the interface still support a highly detailed surface representation as shown in the figure.

The image sequence shown in Figure 22 demonstrates a three-dimensional water simulation that uses only merging, originally sampled with 87,965 particles and an auxiliary $30 \times 30 \times 30$ grid. For the animation shown in Figure 23, we used 113,958 particles in the initial step. Splitting was performed near the surface before the simulation began, in which each particle's volume was split to the minimum size of 0.25 standard particle sizes. Splitting and merging were performed during the simulation. The number of particles was reduced to about 60 percent (69,438 particles) due to merging.

Green particles are standard particles which are radius 1. Red particles are the merged particles and blue particles are the split particles with radius less than 1. The grid resolution was $30 \times 30 \times 30$. Table II presents running times for a sample 2D simulation as in Figure 21. Timings were the average seconds per frame over a 200

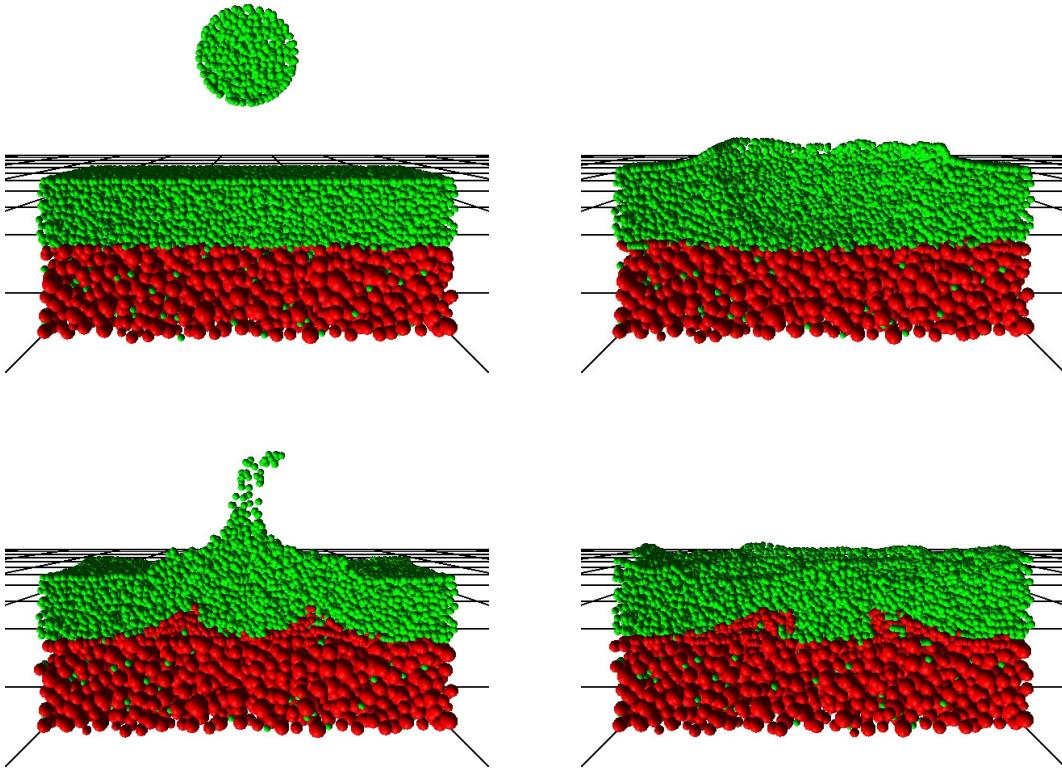


Fig. 22. 3D simulation using merging only

frame sequence, on a Pentium4 1.60 GHz CPU. NA is the timing for a non-adaptive method with no merging or splitting. All particles were of the nominal size, and 1,326 particles were used. M1 presents timing when the particles were merged (but not split) as a preprocess, and no further merging (or splitting) occurred during the simulation. Only 972 particles were used. The M2 simulation starts from the same point as M1, but allows particles to merge (but not split) during the simulation, and the number of particles was eventually reduced to 776. Note that this demonstrates that our particle merging causes noticeable improvements in running time. The AS simulation presents our adaptive sampling approach, including particle splitting. With this approach, we achieve a much finer scale representation of the fluid surface, at a cost of only about 10% more than the original running time with no splitting or merging. In order to

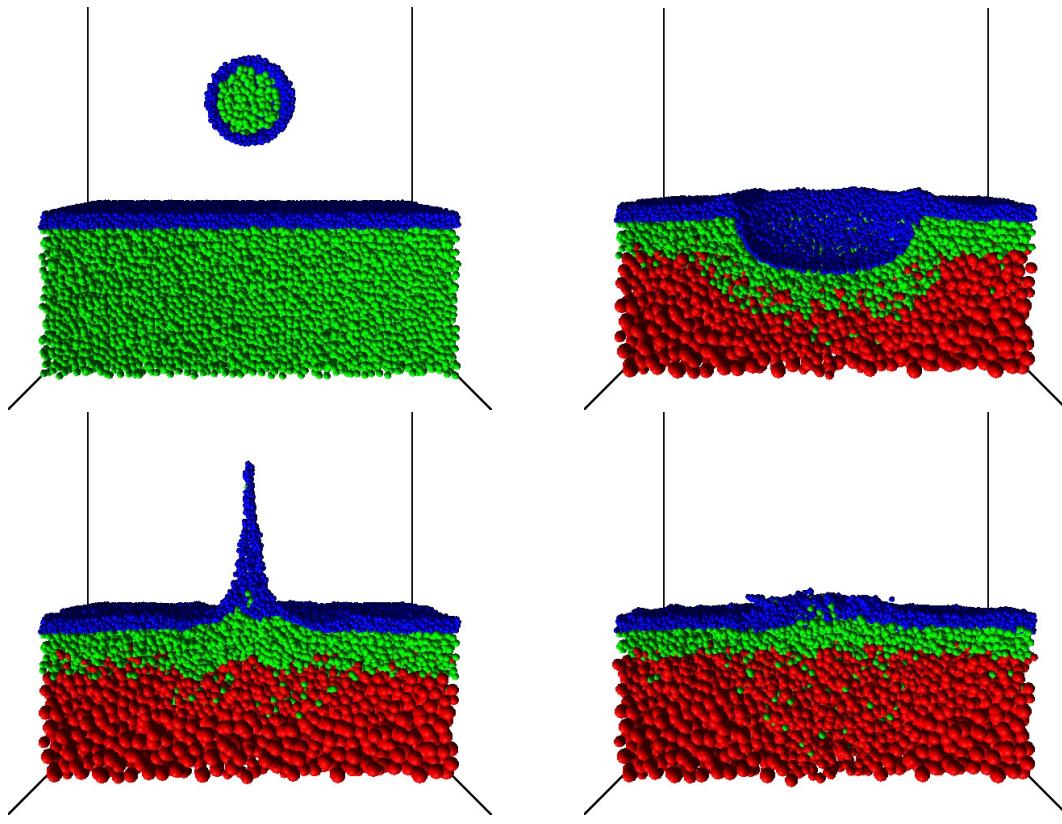


Fig. 23. 3D simulation with splitting and merging

produce a visually plausible fluid surface, surface reconstruction process is required. In our approach, particles are the fundamental representation in our hybrid method, and must be used to determine the location of the fluid surface. This guarantees that the fluid surface interface will be maintained at the full available level of detail, and then there will be no loss in the volume of the fluid. Also, the adaptive approach allows us to capture the small detail features through split particles. Thus, the following

Table II. Simulation time table for a 2D simulation similar to Figure 21

	NA	M1	M2	AS
Simulation time(sec.)	0.208	0.167	0.136	0.226

equation is proposed:

$$\phi(x) = \left(\frac{\sum_i V_i W_i(x; r_i) (1 - \frac{|x-x_i|}{r_i})^2}{\sum_i V_i W_i(x; r_i)} \right)^{\frac{1}{2}}, \quad (3.9)$$

where V and W are the particle volume and kernel function, and x, x_i and r_i are the sampling point, neighboring particle location and its radius. $\phi(x)$ is computed from the particles on an auxiliary grid of the desired surface resolution and then the final value is normalized by the summation of density. We use the combination of a root and square in order to produce a particle-defined surface that gives more contribution to particles closer to the surface, while reducing bumpy artifacts. The surface interface is simply the zero level set of this function.

Once the surface cell is detected, a signed distance field is easily constructed by fast sweeping, which is exploited for detecting the deformable area in the next step of simulation. The *Marching Cubes Method* [23] is exploited to extract a polygonal isosurface for high quality rendering. Once the signed distance field is constructed, bump artifacts are reduced by averaging the neighboring vertices and the normal vector of their vertices of the resulting mesh. In order to accelerate the smoothing operation, k-D tree algorithm is used for searching the neighbors. we found that three or four iterations are sufficient for smoothing. Excessive iterations of smoothing lead to elimination of important small detailed features but eliminating the smoothing operation leads to a poor fluid surface with lots of bump artifacts near the surface. Figures 24 and 25 show the effect of surface smoothing in both 2D and 3D.

In the 3D simulation shown in Figure 26, we rendered the surface using the marching cubes technique through our surface reconstruction method. This simulation was also performed with splitting and merging. The grid resolution for surface generation was $150 \times 150 \times 150$ and 623,575 particles were generated in the initial

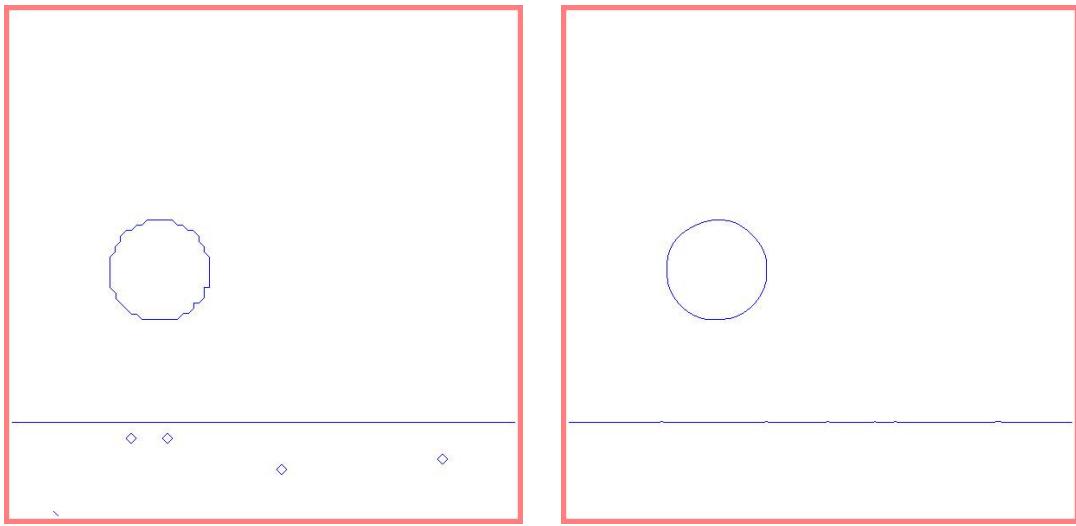


Fig. 24. Surface without (left) and with (right) surface smoothing

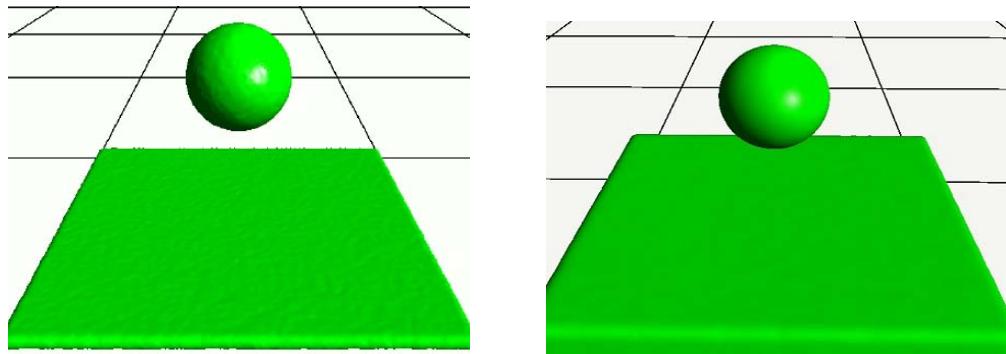


Fig. 25. Surface without (left) and with (right) surface smoothing

step. More than 40 percent of the initial particles were eliminated due to merging.

2. Evaluation

Mass particle refinement and simplification based on depth and Reynolds number give several advantages in terms of the performance of the simulation and a detailed representation of the fluid surface. In order to obtain interesting flow motion, splitting is performed near the fluid surface. Merging away from the surface reduces the

number of particles. This enables us to save considerable memory space in large scale simulations and improves the speed of the simulation significantly, while maintaining detail where it is needed.

Distance from the surface is used as our basic criterion for determining whether to split or merge particles, and we use a signed distance field for computing this distance. As a complementary criterion, we have proposed a calculation based on the Reynolds number to determine deformability of areas below the surface. Particle splitting and merging are performed adaptively using these criteria.

Next, we will investigate more faithful capture of the character of the small-scale detail of turbulent water, including splashes and bubbles. To do this, I will implement surface tension, using techniques similar to those of Hong [18] and Kim [21]. In addition, I will couple an octree based MAC grid approach as in Losasso et al. [24] with our adaptive particle approach, so that appropriate spatial resolution is maintained in all phases of our calculations, further improving the performance of our simulation. Our fully adaptive particle approach will be presented in the next chapter.

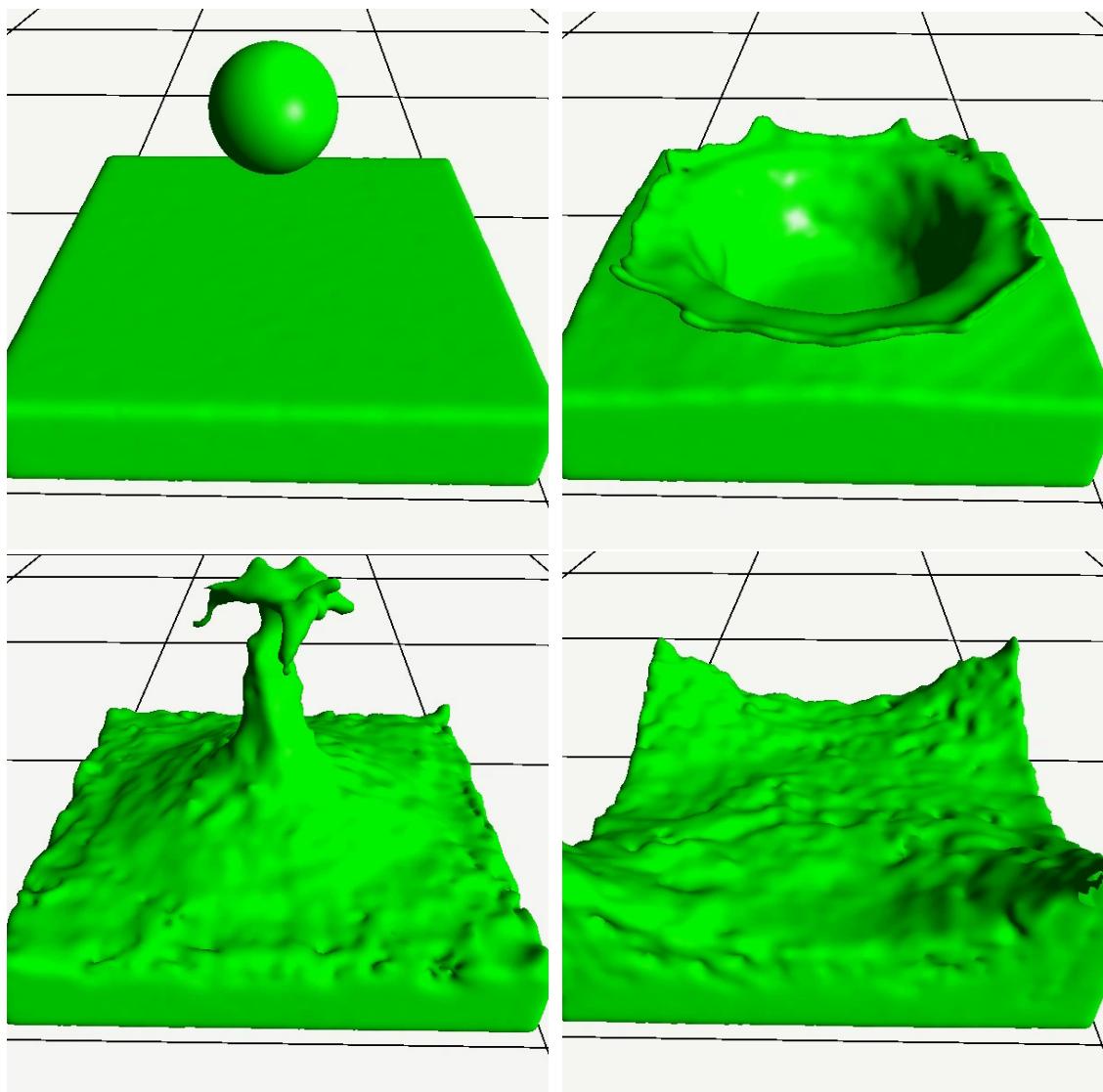


Fig. 26. Animation result with surface-smoothing

CHAPTER IV

FULLY ADAPTIVE FLUID SIMULATION FOR INCOMPRESSIBLE FLOW

A. Introduction

Thus, my work in this chapter, I describe an adaptive particle-based technique for simulating incompressible fluid that uses an octree structure to compute inter-particle interactions and to compute the pressure field. As presented in the previous chapter, an efficient incompressible Lagrangian fluid simulation was proposed to utilize splitting and merging of particles to adaptively sample the evolving fluid motion.

Significant memory and computing time savings came from reducing the number of particles through the merging scheme. The adaptive sampling scheme was performed only on particles using splitting and merging to capture the complex flow. It was not performed on the computational grid. The uniform-size grid was still used as an auxiliary grid in the method to enforce incompressibility and boundary conditions, which can lead to waste of memory, low performance of simulation and numerical dissipation. For an example, when particle size is near to or smaller than the size of a cell in the uniform grid, the sample point might not lie within the volume of any particle, and thus the weighted average is not applicable. If there are such particles, the fluid velocity is determined by taking a volume-weighted average with the uniform size radii in the previous method. This method may cause a serious numerical dissipation when splitting is performed several times within a cell. Also, numerical dissipation may occur when a large particle is generated through merging and occupies several uniform-size cells on the auxiliary grid.

Thus, my work in this chapter attempts to find a suitable simulation framework that exploits the right methodologies for the various stages of the simulation, while

maintaining a representation that is appropriately adapted to the required simulation granularity. To make our approach fully adaptive, a particle splitting and merging scheme is exploited using both depth from the surface and flow complexity to determine when particles should be split or merged and an octree spatial data structure is used to represent the computational grid, with octree cell size determined at each time step by the particle sampling in the neighborhood of the cell. Eventually, the octree supporting field-based calculations provides a fine spatial reconstruction where particles are small and a coarse reconstruction where particles are large. This scheme places computational resources where they are most needed, to handle both flow and surface complexity.

Thus, incompressibility can be enforced even in very small, but highly turbulent areas. Simultaneously, the level of detail is very high in these areas, allowing the direct support of tiny splashes and small-scale surface tension effects. If the simulation level of detail is to be very fine, a representation of surface tension becomes important. This is especially true when it is desired to represent spray and splashes. A particle-based scheme naturally accounts for fine spray, but will not directly support the representation of surface tension, which is needed for the simulation of small water drops. Again, there is a need for both a particle-based representation of the material of the fluid, and a grid-based representation of the small-scale fluid surface. Thus, the method is essentially particle based, with particle advection providing the transport mechanism in the fluid, and the pressure, inter-particle interaction and surface geometry being calculated on an octree whose resolution is locally determined by the particles.

The main contribution of this work is to demonstrate how to support a fully adaptive simulation of both particle and grid-based components for incompressible fluids. As part of this, we present a new method for determining spatial grid resolution

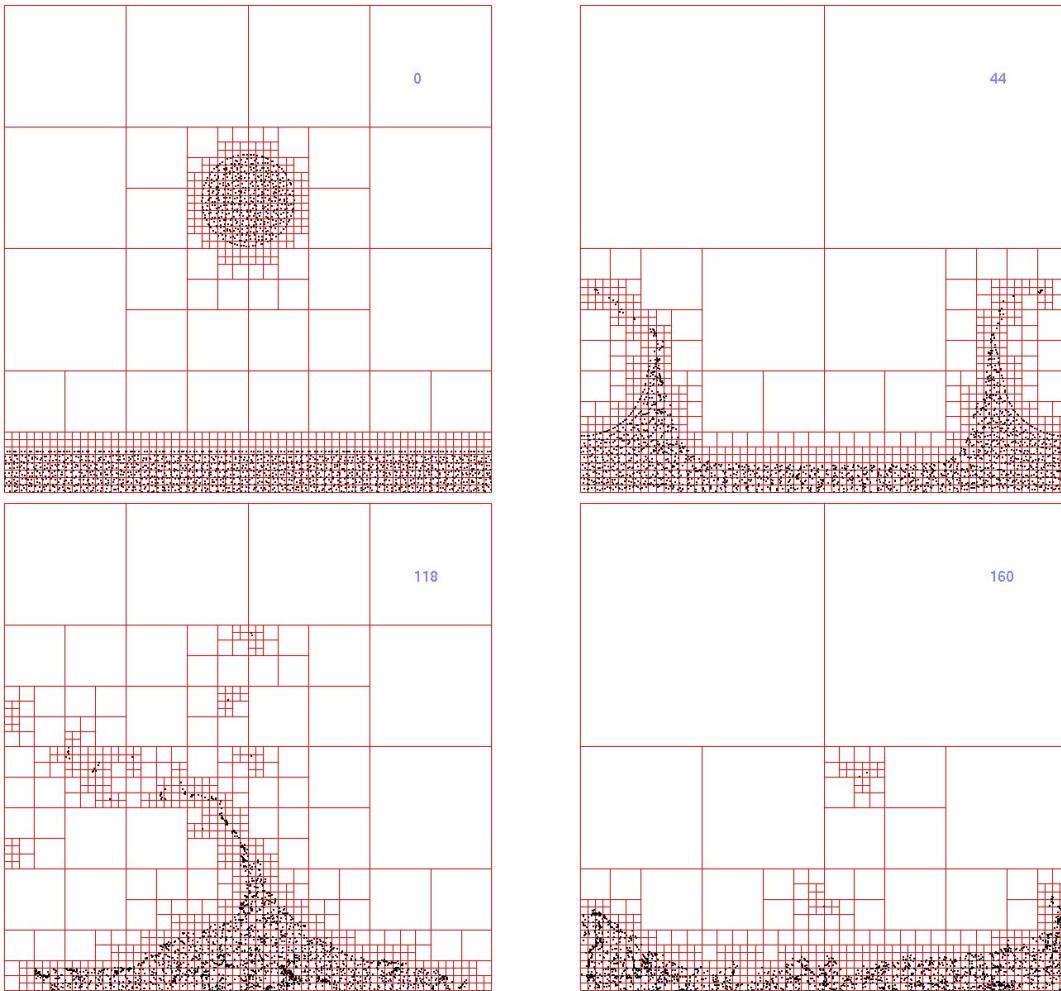


Fig. 27. Quadtree hybrid simulation with uniform cells

based on adaptive particle sizes, and coupling this grid with the particles. We also present improvements to particle splitting and merging. Our approach allows us to have the resolution needed to incorporate surface tension effects into an adaptive particle-based simulation, and we present a new method for computing these forces.

Figure 27 shows an example of our result when using the hybrid method, coupled with an octree data structure but using uniform size particles.

B. Overview of our adaptive fluid simulation

Our overall method is an extension to the hybrid FLIP method [37]. We maintain the general FLIP approach of using particles to capture fluid advection and a grid to perform pressure calculations. In order to make the simulation more adaptive, we incorporate adaptive techniques in both key aspects: namely particle splitting and merging for Lagrangian calculation, and octree grids for Eulerian calculation. A surface tension force is added as an external force to more accurately capture small-scale fluid motions (the detailed description of the surface tension force will be given in Chapter V). More detail explanation will be given in the next chapter. While each of these ideas has been applied individually in prior work, we show that these techniques can be applied collectively in a unified manner. Specifically, we show how the adaptive particles can be used to drive the formation of the octree grid. Figure 28 presents a diagram illustrating our overall process. The following steps are iterated during the simulation.

- Update particle positions based on velocity.
- Generate an octree grid with resolution based on local particle radii.
- Identify surface cells from among the current fluid cells. Subdivide cells in the air based on the distance to the fluid surface.
- Construct a signed distance field from the particles using fast sweeping on the octree grid, allowing identification of particles near the surface. Local surface curvature is computed for these particles to provide a surface tension force.
- Use the particle velocities to reconstruct a set of fields on the octree. Fields are maintained, at the center of each octree cell, for velocity, pressure, distance

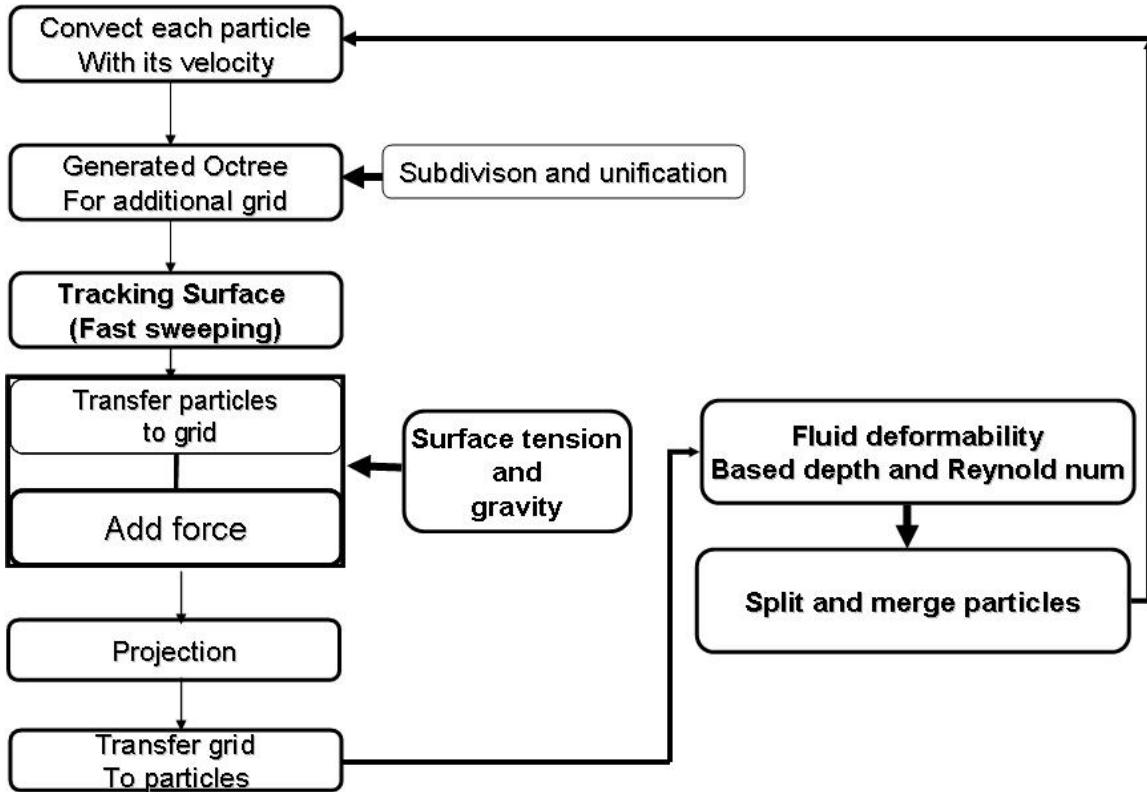


Fig. 28. Overall simulation structure

from the surface, and applied forces. Forces include gravity, surface tension, and any externally applied forces. A circular kernel is used for the reconstruction.

- Create a copy of the octree. Using two octrees avoids the additional steps and computation that would be needed to update the field in place.
 - Calculate the pressure field, and project the flow field onto the nearest divergence free field [24].
 - Transfer velocity changes in the flow field back onto the particles' velocities.
 - Compute a fluid deformability measure based on depth and local motion.

- Split or merge the particles based on the local deformability measure.

Several of these steps are common to the hybrid FLIP method, or are explained in detail in the previous chapter, and we will not discuss them further here. We will focus our discussion in the following sections on the key areas where we present new ideas or improvements on prior approaches. Specifically, we will describe the creation of octree cells based on particles (Section C), splitting and merging particles (Section D), communicating velocity information between the particles and the grid (Section E), and computing surface tension (Chapter V).

For illustrative purposes, most of our figures and examples will be presented in 2D, while the descriptions and formulae will be in 3D (e.g. octrees instead of quadtrees). However, all the methods discussed have been implemented and work equally well in both two and three dimensions.

C. Octree generation

Key to our method is the generation of an octree data structure at each iteration. During simulation, both particle volume and octree cell size can be changed adaptively, conserving mass for a constant density incompressible fluid. We choose to have the octree structure adapt to both the positions and the sizes of the particles. Thus (as described in Section D), we adapt the particle size/distribution to the needs of the simulation, and then the octree to the particles.

1. Relating adaptive particles and octree cells

First, we need to discuss the relation between adaptive particles and octree cells. In a sense, these values can be considered independently – the grid size and particle size/distribution do not necessarily have to be related. This is illustrated in prior

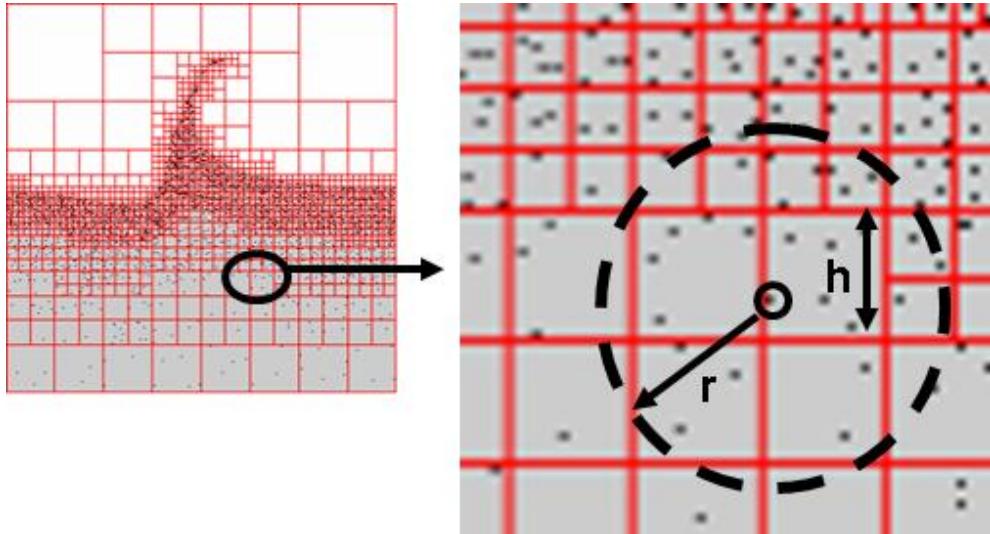


Fig. 29. The size of particles determines the size of an octree cell

adaptive particle methods that use fixed grids [19]. However, in general the areas where we want additional (or less) detail in the pressure calculation are the same as those where we want more (less) detail in the mass advection, therefore it makes sense for the grid size to be related to the particle size. In order to communicate information between the grid and the particles, we need to be able to map velocity information from the particles to the grid and vice-versa (see Section E). When a size h cell is matched with a radius r particle, where $r = \sqrt{3}h$ (or $r = \sqrt{2}h$ in 2D), each particle can be expected to cover at least eight (four) nodal points on the octree (quadtree) grid. This coverage guarantees that (if all adjacent cells are the same size), a particle will affect all nodes of the grid within the cell in which it lies. This is illustrated in Figure 29. We note that although this is a heuristic, it works very well in practice, and better than other values we tested. We discuss next how this heuristic is used to determine the actual octree subdivision.

2. Creating the octree grid

It is straightforward to create an octree grid according to the “target” particle radius. All particles are placed into a single octree node for the simulation, and this node is recursively subdivided to form the octree. In the recursion, each node is examined to determine whether it contains a particle with radius less than the “target” value (i.e. in our implementation, with $r < h/\sqrt{3}$). If two particles in the same cell have different sizes, the cell will be subdivided based on the smaller particle radius. When a cell of the octree is of sufficiently small size, the cell is identified as a leaf node. Typically, there will be only a single particle per leaf node, though it is possible to have more.

Leaf nodes containing particles are identified as fluid cells and the physical values for the cell (velocity and pressure) are initialized from the particles (Section E).

As the octree is created, the areas near the surface will thus be described by a fine grid (assuming the particles were of small radius). However, for the velocity field to be extrapolated accurately, we must also ensure a fine sampling of the velocity grid on the “air” side of the fluid-air interface. Essentially, we want neighboring air cells to be no larger than the fluid cells. So, we will further process the octree data to add detail in the air near the fluid.

Figure 30 demonstrates the relationship between the particles and the octree cells created. All fluid cells are stored into a list and used to construct this extra detail. For each fluid cell, we determine any neighboring air cells, and repeatedly apply octree subdivision to those cells, until the neighboring cells are as small as the fluid cell. Within this process, we also identify which fluid and air cells form the fluid/air boundary, and store these in a list for subsequent computation.

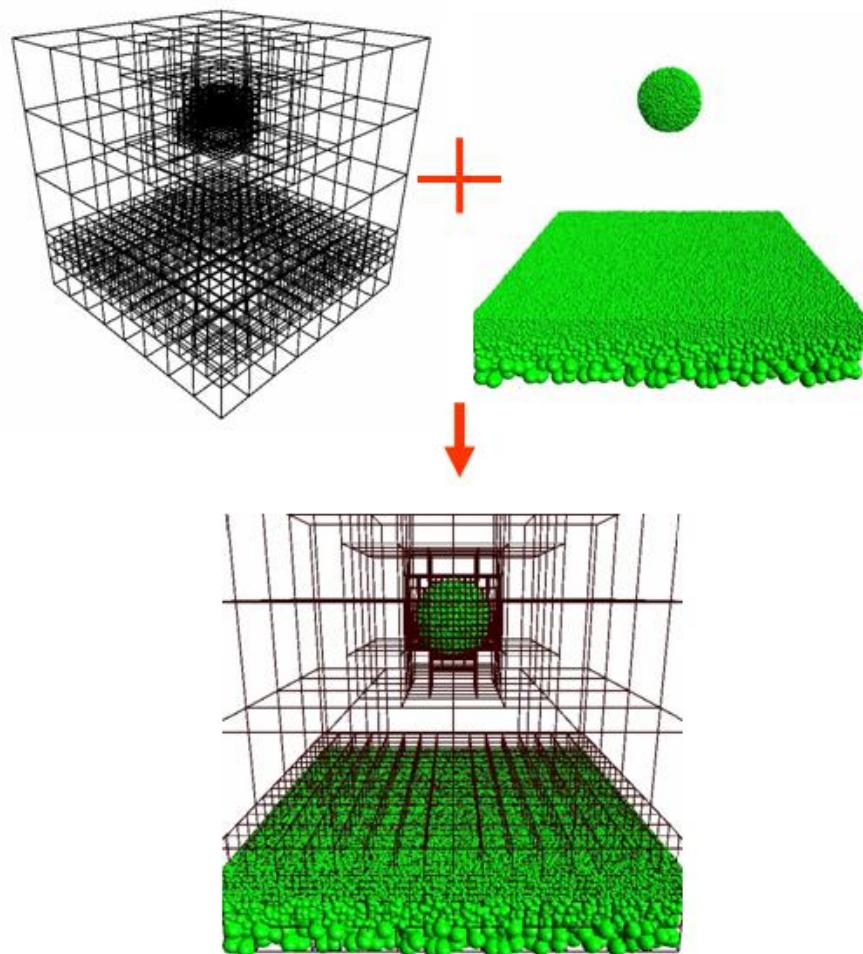


Fig. 30. Relation between adaptive particles and the octree cell

D. Splitting and merging

In order to adapt our simulation to the areas of most interest, we implement a particle splitting and merging approach. As presented in chapter III, we base our splitting and merging on both distance to the surface and on local fluid deformation similar to the Reynolds number. We summarize our approach here, which offers an improvement over prior adaptive particle methods, and focus on the unique characteristics.

1. Splitting and merging process

The process of splitting and merging particles must preserve mass and momentum. For a constant density incompressible fluid, mass and volume conservation are equivalent.

Merging particles is straightforward; the merged particle's mass and volume are the sum of the smaller particles' mass and volume, while position and velocity are chosen to conserve momentum and center of mass. When we decide to merge particles (Section 2), we use a greedy approach. We choose one particle arbitrarily, and merge it with the nearest particle within the particle's radius r . The process is continued with this particle, which will now have a larger radius, until its volume reaches the maximum volume allowed in the region of the particle, or no further particles are within its radius.

Splitting is somewhat more complicated. When splitting, a single particle is divided into two or more particles, each of equal volume/mass, and with velocity equal to that of the parent particle. Given the original particle position, we form a “box” around the particle position, of size equal to the size of a grid cell, h . The new subparticles are generated within this box. In practice, to ensure good distribution of child particles, we subdivide the box into a 2 by 2 by 2 grid of subboxes, and ensure

that no particles are placed into the same subbox (we never subdivide into more than 8 particles).

When a particle is split near the surface, we modify the positions of the subparticles to ensure a good sampling remains near the surface. If the parent particle is located within a threshold of the surface, then the new subparticles are projected to the surface using the gradient of the distance field (already computed):

$$\vec{N} = \frac{\nabla\phi}{|\nabla\phi|}. \quad (4.1)$$

2. Layers for splitting and merging

With the splitting and merging process available, we must determine exactly when to split and merge particles. We divide the fluid into a number of “layers,” in each of which we determine whether to merge or split particles.

Layers are determined by distance from the surface. Within each layer, we determine whether to split/merge based in part on a local measure of fluid deformation based on the Reynolds number; this determination is similar to the approach which presented in previous chapter, so we will not discuss it further. Generally, each layer has a nominal target particle size, with radius twice that of the layer nearer the surface. Local motion of the fluid within the layer can cause smaller particles to occur in those areas.

We create hysteresis around the layer transition boundary by defining a small range, ϵ , in which merging (in the “smaller layer”) or splitting (in the “larger” layer) are not allowed. This is illustrated in Figure 31. The range of ϵ is determined by the nominal radius of particles in the farther layer. Specifically, we set $\epsilon = h$ for the grid cell size, h of the layer with larger particles. This hysteresis is used to prevent particles newly split/merged from immediately being merged/split. For example,

splitting a particle could cause some subparticles to be placed in the “larger” layer, and thus immediately merged. Likewise, a merged particle could be positioned within the “smaller” layer and immediately split.

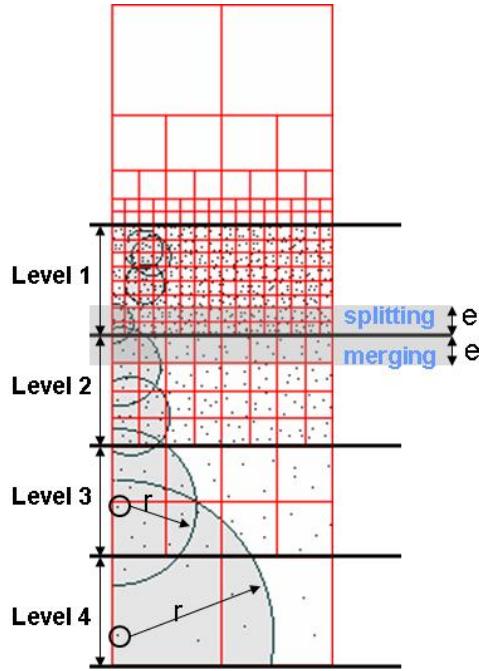


Fig. 31. Layers for splitting and merging

E. Kernel choice and transferring particle velocity to octree grid

Once the octree grid is formed, the velocities of particles must be transferred to the grid. To do this, a kernel function is associated with each particle, and the velocity value at each node point is determined by the weighted average of the kernels that overlap it. Likewise, this kernel is used to transfer velocities from the grid to the particle (velocities of the nodes are weighted by the kernel value).

We use a circular kernel function, w :

$$w(r) = \left(1 - \frac{r}{r_e}\right)^3 \quad (0 \leq r < r_e) \quad (4.2)$$

$$w(r) = 0 \quad (r_e \leq r) \quad (4.3)$$

where r is the distance from the current particle, and r_e is the current particle radius. The power of 3 in the kernel function was experimentally determined to give somewhat better results than a linear “tent” kernel or a quadratic kernel in producing detailed splashing effects. Using a circular kernel, rather than a rectangular kernel makes finding which cell centers are within the particle’s radius simpler. Figure 32 shows circular kernels on an octree grid.

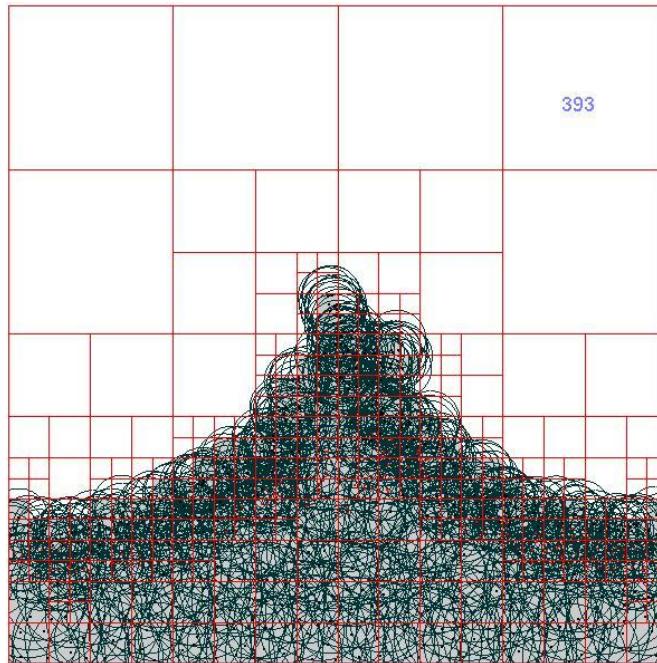


Fig. 32. Circular kernel functions

F. Constructing the poisson equation

To construct and solve the Poisson equation, we simply use Lossaso’s octree based approach [24]. The only difference is that we minimize octree neighbor cell queries by storing all field quantities in the center of each grid cell, speeding some of the

computations. In a fixed grid system this could lead to unwanted smoothing of spatial derivatives of the velocity field, since interpolation is required to compute derivatives where they are needed. However, in the spatially adaptive octree formulation, this effect is negligible. Pressure is computed as follows:

$$V_{cell} \nabla^2 p = \sum_j \left(\frac{p_j - p_i}{\Delta} \cdot n \right) A_{area}, \quad (4.4)$$

where V_{cell} is the volume of the current cell, j is the index of neighbors, n is the outward unit normal of the current cell, Δ is the size of the large cell and A_{size} is the face area of the neighbor cell.

G. Divergence

Following the same logic that was used for the pressure gradient, in octree cells the generalized form for the divergence term is

$$V_{cell} \nabla \cdot u = \sum_j (u_j - u_i) \cdot n A_{area} \quad (4.5)$$

where A_{area} is the area of a cell face, n is the outward unit normal of the current cell and V_{cell} is the volume of the cell in the fluid simulation based on the octree.

H. Results and discussion

1. Results

We have implemented the algorithm as described, in both two and three dimensions. Figure 33 shows a comparison of a standard method, a method with adaptive particles only, and our fully adaptive method. Results of our implementation can be seen in

Figures 34 and 35. In Figure 34, the maximum grid resolution is 128^3 . 7122 particles were used in the starting configuration. The average number of octree cells was 2419. In Figure 35, the maximum grid resolution is 128^3 . 668,261 particles were used in the starting configuration. The average number of octree cells was 88,033. These two figures show sequences from a 2D and a 3D simulation conducted using our approach, illustrating both the particle sampling and the octree structure. The detail at the surface and the coarsening of the representation below the surface can be clearly seen. Since the simulation is grounded in a particle representation, splashes and spray are handled directly, and actually account for true fluid mass. At the same time, the octree representation provides the fine detail needed around spray particles so that nearby particles continue to form a surface over which surface tension forces can be computed.

2. Contrast with previous methods

Note that our approach for merging and splitting is in contrast to that of Adams et al. [2], which can allow some particles of larger size to end up near the surface. While the larger kernel they then use for surface smoothing allows the method to produce smooth results, it also eliminates the small scale surface features that can be vital for producing splashing and similar effects. Since a major goal of our method is to capture fine surface detail (such as from surface tension), we do not adopt their splitting and merging criterion. More fundamentally, that approach is based on SPH, and does not support incompressibility, which is a key feature for realistic water behavior.

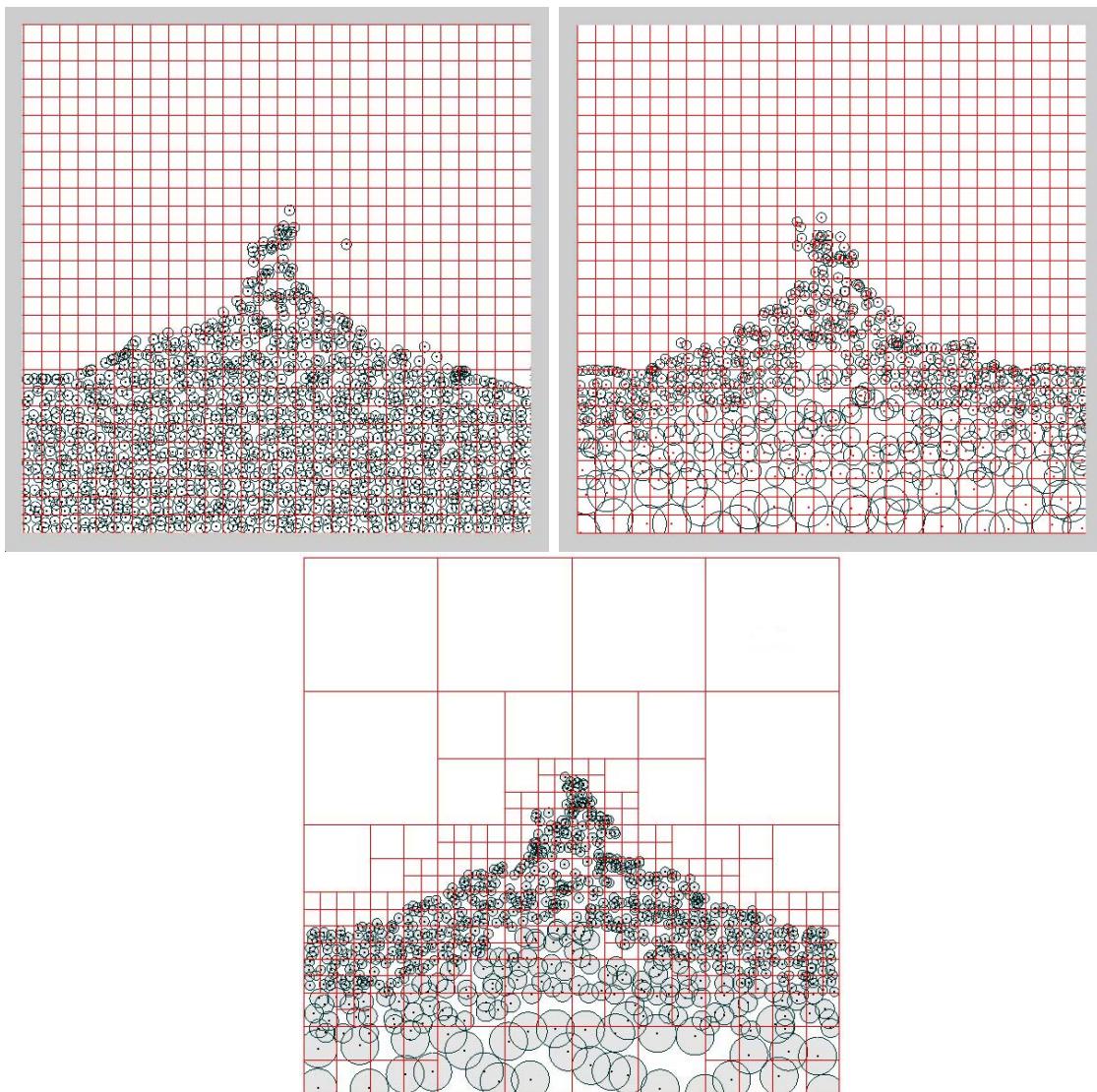


Fig. 33. Contrast with Previous Methods.(Comparison of Normal simulation, Adaptive Particle and Fully Adaptive Simulation)

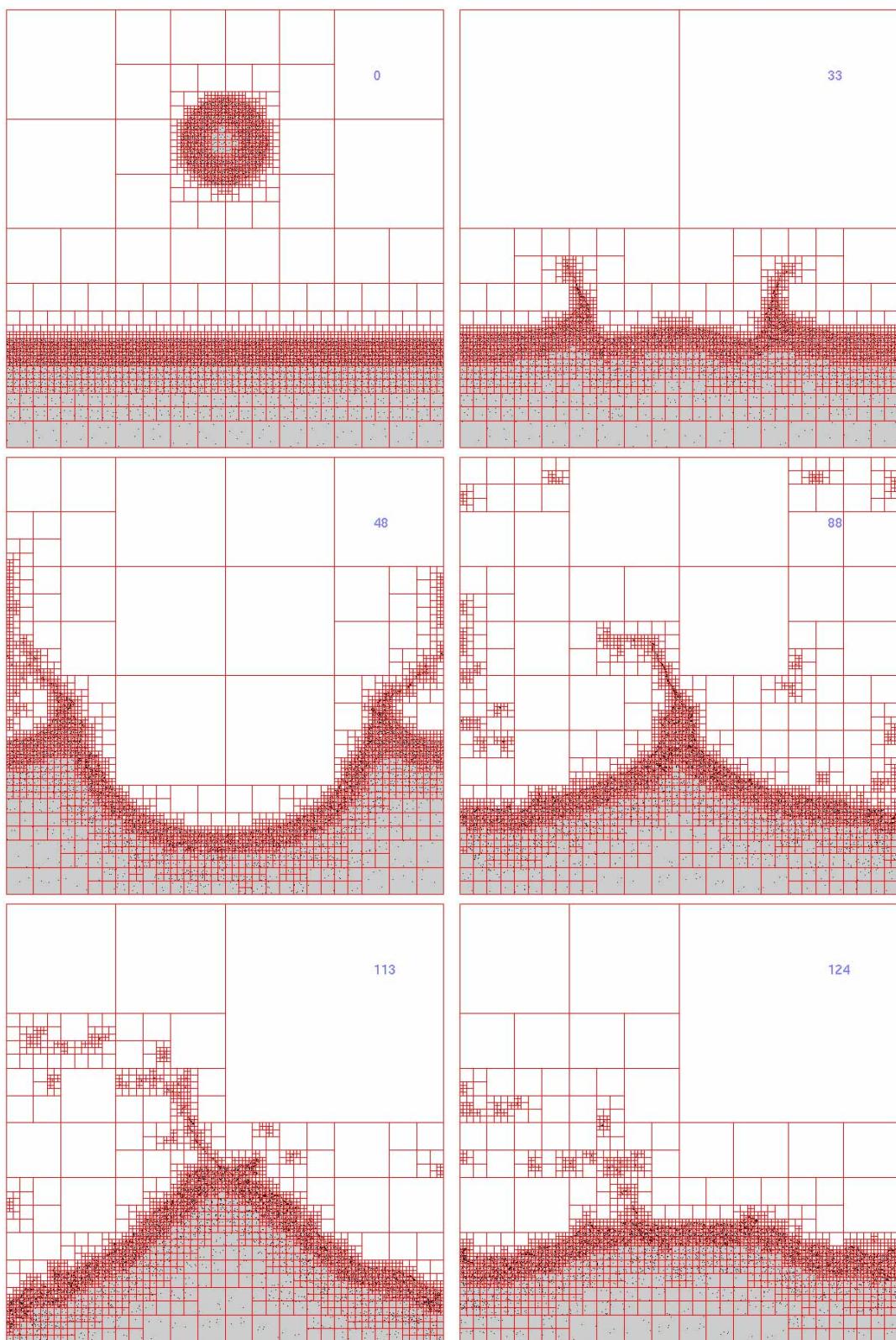


Fig. 34. 2D Fully Adaptive Fluid simulation

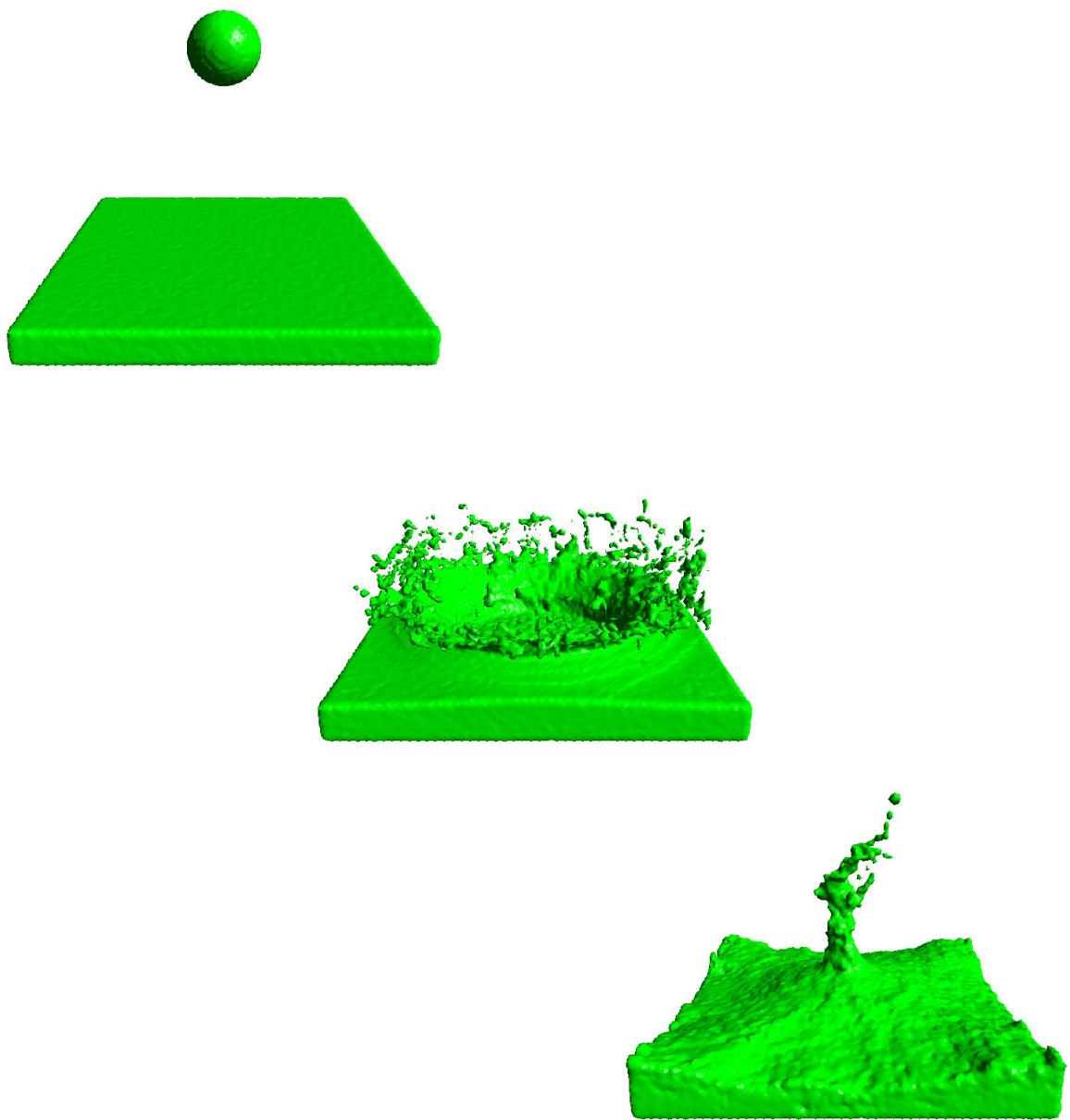


Fig. 35. 3D fluid simulation using adaptive particles and an octree grid

CHAPTER V

SURFACE TENSION WITH FULLY ADAPTIVE FLUID SIMULATION

A. Introduction

In our adaptive simulation, particle splitting is performed near the surface of the fluid in order to capture the characteristic small-scale details of the water. This results in the generation of many particles near the surface of the fluid, leading to non-realistic small scale effects as shown in Figure 36. We believe that by introducing a surface tension representation, these small scale effects can be minimized. In order to provide a more faithful representation of the small-scale details of turbulent water, including splashes and bubbles, we propose implementing surface tension, using techniques similar to those of [18, 35].

Hong and Kim[18] describe a technique for animating fluids that extends previous techniques to handle multi-phase fluids with surface tension effects and viscosity changes at their interfaces. Wang[35] presents an algorithm to solve the capillary solid coupling problem by modeling surface tension between the liquid and a solid object. The method replaces the liquid-solid interface with a virtual surface beneath the solid surface so that the estimated boundary pressure can represent all surface tension on the contact front.

For our fluid simulation, we expect that the octree based structure will be very useful for representing surface cells including surface tension. Even though the techniques of Hong and Wang[18, 35] show new aspects of small-scale fluid motion and Wang tried the Sparse Grid Representation technique which activates or deactivates regions based on particle presence, their fluid simulations must be performed in a high resolution MAC grid to capture the small details. This is excessively complex

in both memory and time. We propose a more efficient technique in which surface tension is applied directly to the boundary of each cell which corresponds to each leaf node of an octree and the grid size is changed adaptively during the simulation based on the radius of particles. We review how to model the surface tension on a MAC grid, then we observe some shortcomings of our technique and now they are resolved in the following section.

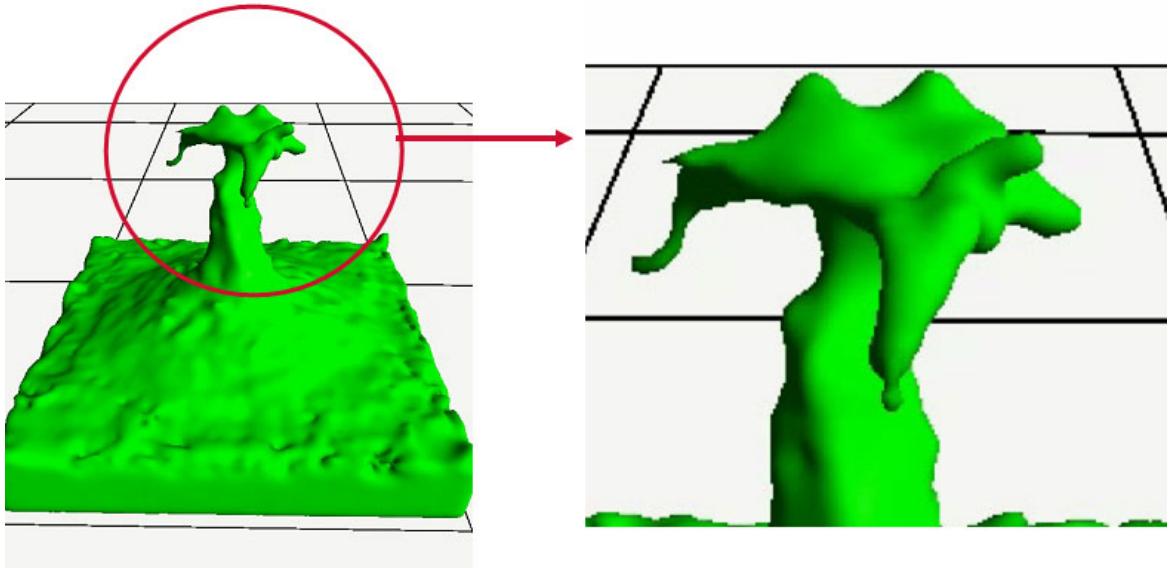


Fig. 36. Non realistic splash effect without surface tension

B. Modeling surface tension

Surface tension is physically caused by attraction between molecules. Within a fluid, these attractive forces are canceled out, but near the surface they are unopposed in the direction of the surface. In physics, surface tension is usually considered to be an external force oriented towards the negative surface normal with magnitude proportional to the surface curvature.

To compute surface tension, we will need to know surface curvature. Let the

function $\phi(x, y, z)$ be a scalar field defined implicitly as a signed distance field around a level set $\phi = 0$. The gradient of the implicit function is

$$\nabla\phi = \left(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}, \frac{\partial\phi}{\partial z} \right). \quad (5.1)$$

$\nabla\phi$ is perpendicular to the interface (i.e. the surface determined by the level set) and points in the direction of increasing ϕ . Thus, the outward unit normal for an arbitrary point on the interface is

$$\vec{N} = \frac{\nabla\phi}{|\nabla\phi|}. \quad (5.2)$$

The derivatives in equation 5.1 can be approximated using a first-order accurate forward difference such as

$$\left(\frac{\partial\phi}{\partial x} \right)_i \approx \frac{\phi_{i+1} - \phi_i}{\Delta x}. \quad (5.3)$$

This equation is abbreviated $D^+\phi$, and the first-order backward difference is

$$\left(\frac{\partial\phi}{\partial x} \right)_i \approx \frac{\phi_i - \phi_{i-1}}{\Delta x}, \quad (5.4)$$

which is abbreviated $D^-\phi$. The second-order central difference is defined as

$$\left(\frac{\partial\phi}{\partial x} \right)_i \approx \frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x}, \quad (5.5)$$

which is abbreviated $D^0\phi$. The formulas for the derivatives in the y and z directions can be obtained in the same manner. The mean curvature κ of the interface is defined as the divergence from the normal,

$$\kappa = \nabla \cdot N = \frac{\partial n_1}{\partial x} + \frac{\partial n_2}{\partial y} + \frac{\partial n_3}{\partial z}, \quad (5.6)$$

where $\vec{N} = (n_1, n_2, n_3)$. Equation 5.2 and 5.6 yield

$$\kappa = \nabla \cdot \left(\frac{\nabla \phi}{|\nabla \phi|} \right) . \quad (5.7)$$

Thus, we can write the curvature as

$$\begin{aligned} \kappa = & (\phi_x^2 \phi_y y - 2\phi_x \phi_y \phi_{xy} + \phi_y^2 \phi_{xx} + \phi_x^2 \phi_{zz} - 2\phi_x \phi_z \phi_{zz} + \phi_z^2 \phi_{xx} + \phi_y^2 \phi_{zz} - \\ & 2\phi_y \phi_z \phi_{yz} + \phi_z^2 \phi_{yy}) / |\nabla \phi|^3 , \end{aligned} \quad (5.8)$$

where ϕ_{xy} is defined as $D_x^0 D_y^0 \phi$. By equation 5.3

$$\phi_{xy} = \frac{(\phi_{i+1,j+1} - \phi_{i-1,j+1}) - (\phi_{i+1,j-1} - \phi_{i-1,j-1})}{4\Delta x^2} . \quad (5.9)$$

The forces arising from surface tension are based on the above equation. As illustrated in Figure 37, $\kappa > 0$ indicates a convex region, $\kappa < 0$ indicates a concave region, and $\kappa = 0$ indicates a planer region.

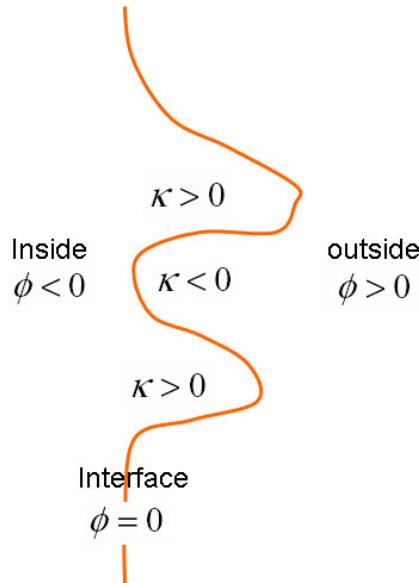


Fig. 37. Curvature in concave region and convex region

C. Adding surface tension to fully adaptive fluid simulation

Surface tension J can be described as a force in the negative surface normal direction, with magnitude proportional to the surface curvature, κ_Γ :

$$J = -\sigma \kappa_\Gamma \vec{N} \quad (5.10)$$

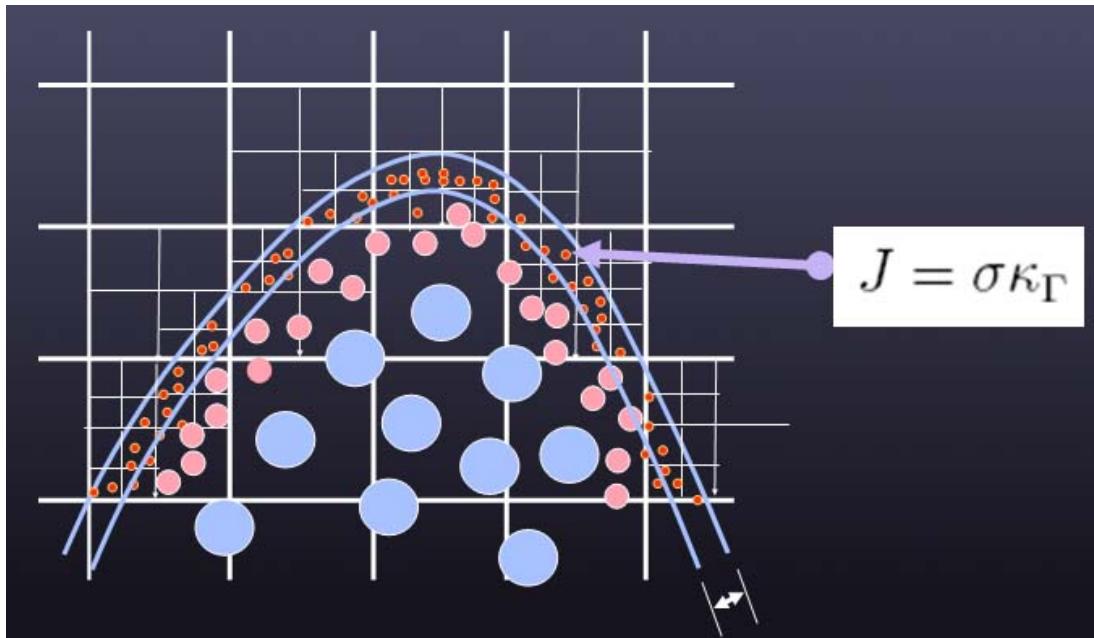


Fig. 38. Surface tension in Fully Adaptive Fluid Simulation

In the hybrid FLIP method, we have two ways to apply surface tension. The prior work for incorporating surface tension, presented by Hong and Kim [18], is based on computing these external forces on an Eulerian grid. Likewise, we could compute the tension force by generating an additional MAC grid in the FLIP method and using this grid to compute the surface tension forces. However, doing so would require a very high resolution grid and significant additional computational time in order to capture the small detailed features that are the whole point of incorporating surface tension in the first place.

We prefer to instead apply the surface tension force directly within the FLIP method, as we present here. First, surface particles are identified as those within a small depth ϵ from the surface in the signed distance field. Figure 38 illustrates the region in which we calculate surface tension.

The normal $\vec{N} = \frac{\nabla\phi}{|\nabla\phi|}$ and curvature $\kappa_\Gamma = \nabla \cdot N$ are computed at each of these points, based on the distance function. This force is then transferred to the Octree grid along with particle velocities, and is then used along with other external forces (such as gravity and viscosity) when performing the Eulerian force computations. Because this alternative has significantly lower computational cost and makes use of the Lagrangian nature of the approach, we believe it provides a better alternative for surface tension calculation within a hybrid FLIP implementation.

D. Fluid-object interaction

First, the computation of the normal vector for an object is important. In order to compute it, a signed distance field should be computed for the object and then the normal vector can be computed at any position. We need to consider one problem: that many particles could collide in the fluid. If a response to a collision is computed whenever a particle collides, it would be very costly. In order to prevent this problem, we consider a common technique by Muller, Keiser and Simon, who worked on Lagrangian fluid. The collision between many particles in a fluid can be detected at the same time. First, an object is evolved in time in terms of linear velocity and angular velocity without collision detection with particles. Then, all particles located in the object can be detected as shown in Figure 39.

The relative velocity is important to accumulate particle impulses on the body. Assume that there are two objects A and B, and the relative velocity A to B is

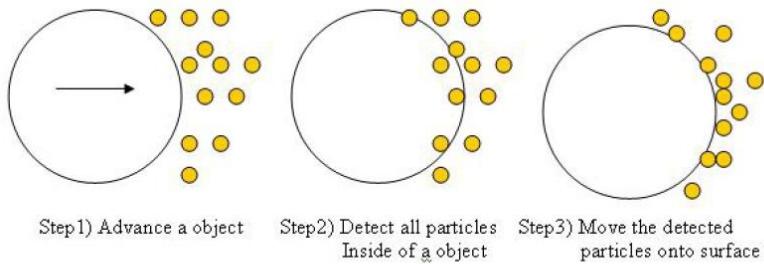


Fig. 39. Interaction between fluid particles and an object

(velocity of A - velocity of B). Using the relative velocity, we consider both object-to-fluid and fluid-to-object interaction. The velocity can then be separated into normal and tangential components.

$$v^{normal} = (v_r \cdot n)n \quad (5.11)$$

$$v^{tangent} = (v - v^{normal}) \quad (5.12)$$

$$I = v^{normal} - \mu v^{tangent} \quad (5.13)$$

The impulse, including friction, is computed as in the above equations and then the impulse is accumulated to determine the force and torque for the object. In terms of particles, the relative velocity of the particle to the object is computed and used to apply the impulse to the particle. Particles are moved to the surface of the object before applying the impulse.

E. Results and comparison

In this section, we demonstrate the effect that surface tension has within the simulation. For all calculations, constants typical of water were used. As physical properties, kinematic viscosity $\nu = 8.90 \cdot 10^{-7} m^2/s$ with density $\rho = 1000 kg/m^3$ and surface tension between the liquid-air interface, $\sigma = 73 g/s^2$ (a room temperature) are assumed in our tests. Figure 40 shows the results using 2 different surface tension parameters. The first image shows no surface tension, the second image σ and the third image 2σ as the surface tension coefficient. Notice that the water particles cluster more when surface tension is included. In Figure 41, a water ball drop was performed onto an object(green circle). The fluid-object interaction algorithm was already presented in the previous section. The simulation shows two results without surface tension(left) and with surface tension(right). The *Marching Square Algorithm* [23] is used to extract a polygonal isosurface. Note that the clustering of the particles is shown in the right picture. Figure 42 demonstrates the surface tension effect in our 3D fluid simulation.

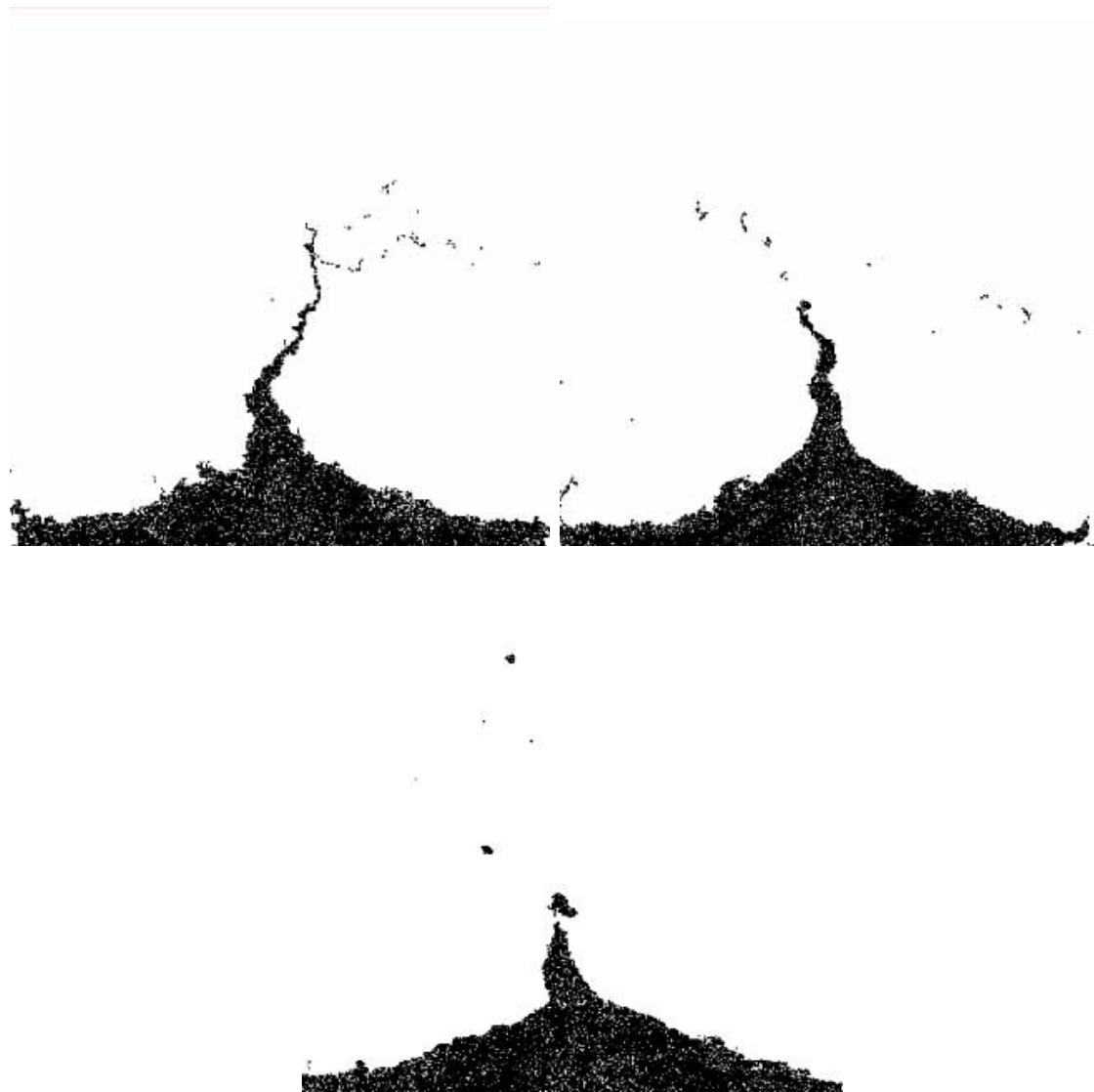


Fig. 40. Comparison of different surface tension parameters

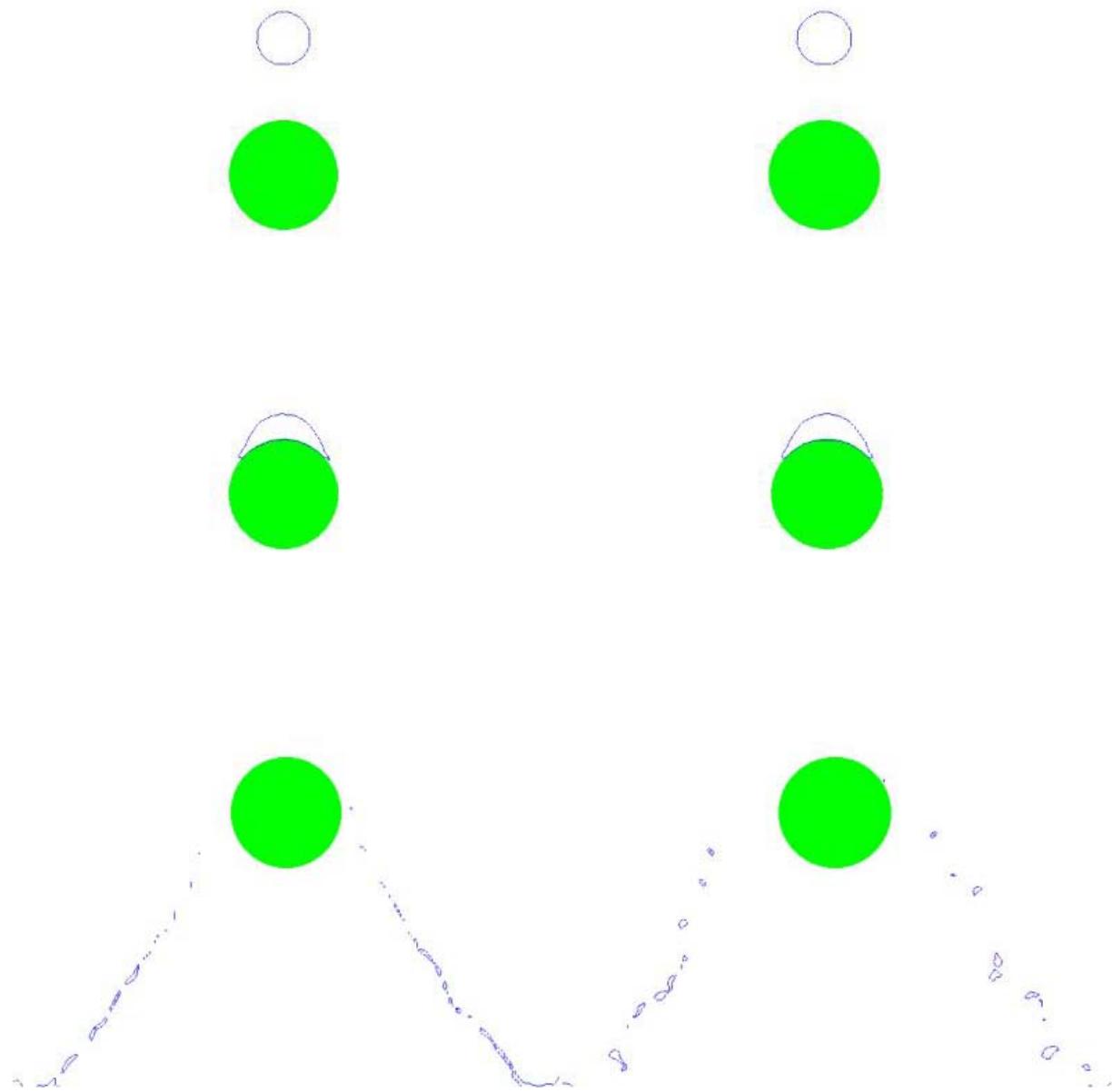


Fig. 41. 2D fluid simulation without (left) and with (right) surface tension

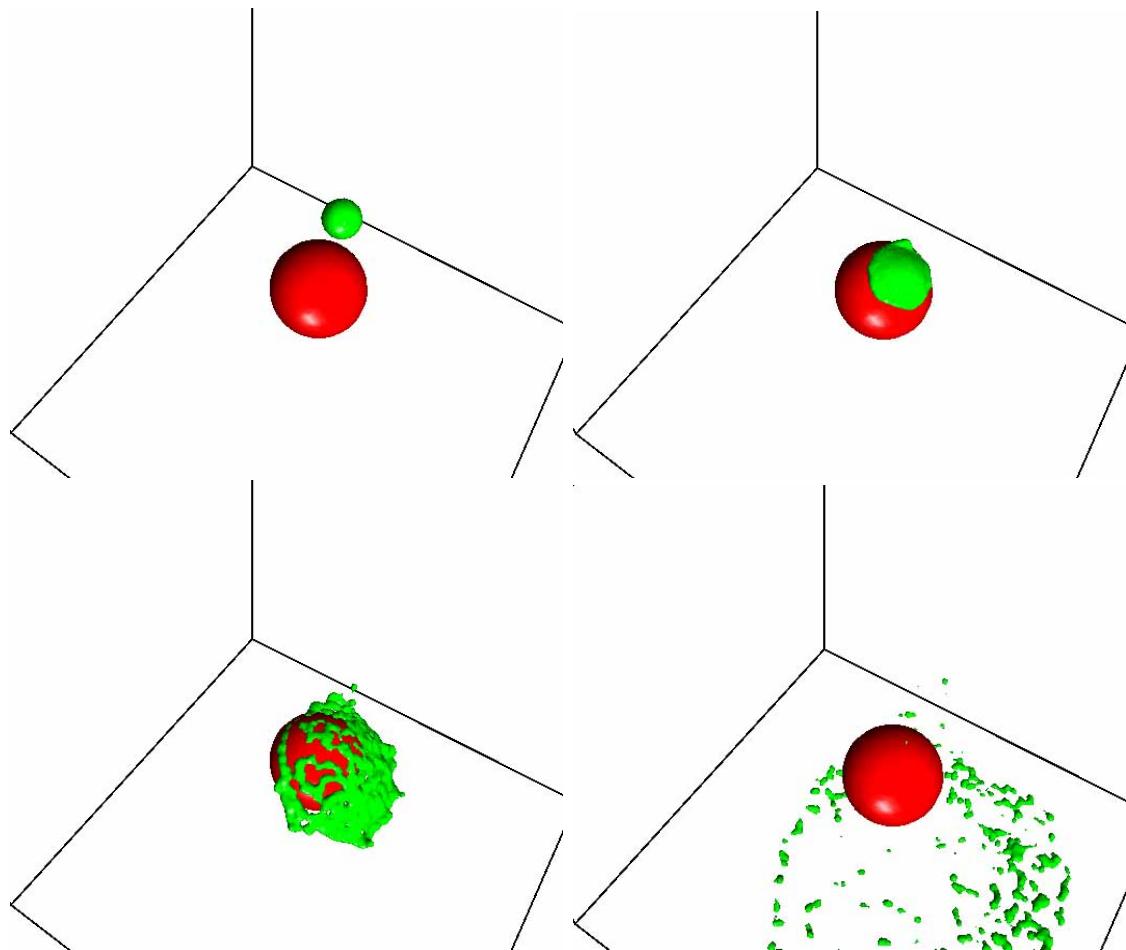


Fig. 42. Image sequence of 3D surface tension test

CHAPTER VI

CONCLUSION/FUTURE WORK

I have proposed a fully adaptive technique for incompressible fluid simulation that uses both adaptive fluid particles and an octree spatial grid. The technique exploits the strengths and avoids the weaknesses of Eulerian and Lagrangian methods, handling advection using a Lagrangian model and the pressure field using an Eulerian model.

First, I proposed an efficient incompressible Lagrangian fluid model using a splitting and merging process for particles in a grid to adaptively sample the fluid motion. Memory space and computing time were lowered significantly by reducing the number of particles through a merging process but the adaptive sampling scheme was performed only on particles, not on the grid. The uniform-size grid was still used as an auxiliary grid to enforce incompressibility which can lead to waste of memory and low performance of the simulation when using a high resolution grid. Also, this method may cause numerical dissipation when splitting is performed several times within a cell. Thus, I needed to extend the adaptive approach to the fully adaptive method, which adaptively samples fluid motion in terms of particles and grid. This is performed at each time step of the simulation.

The fully adaptive method provides several advantages in fluid simulation. Compared to the adaptive particle approach, memory space is saved inside and outside the fluid by using the octree algorithm. Coupling the creation of an octree cell to the particle size enables us to limit the generation of split particles efficiently near the surface, while reducing numerical dissipation. Also, small cells and split particles near the surface provide support for a detailed representation of the fluid surface, including the representation of surface tension.

Surface representation can be made fine-grained enough to allow the computation

of surface tension forces, adding to the realism of the detail in surface shape and spray particles. This force is then transferred to the octree grid along with particle velocities, and is then used along with other external forces (such as gravity and viscosity) when performing the Eulerian force computations. Because this alternative has significantly lower computational cost and makes use of the Lagrangian nature of the approach, we believe it provides a better alternative for surface tension calculation within a hybrid FLIP implementation.

Particle splitting and merging are controlled by both depth from the surface and the local flow complexity. Fluid depth from the surface is determined locally by the size and number of particles. Thus, in deep quiet regions, particles are merged together to form large particles, and correspondingly large cells of the octree grid are created in this region, which enables a significant savings of memory space, while reducing the number of particles and cells. On the other hand, near the surface, where there is considerable turbulence, particles are split into very small particles and correspondingly very fine octree cells are created. The creation of small cells in very high resolution areas allows us to enforce incompressibility even in very small, but highly turbulent areas.

The approach provides a considerable improvement in simulation behavior over other fluid solvers. It is possible to maintain the stability in performance obtainable with Eulerian (semi-Lagrangian) solvers, but without the problems of dissipation and loss of vorticity, and without the problem of volume loss. It is possible to simulate the high detail in surface and splashes that can be obtained with the SPH method, but at the same time to maintain fluid incompressibility. By allowing adaptive particle sizes, the bottleneck of the Lagrangian stage (particle updates) is reduced, at the cost of overhead to maintain the splitting and merging operations. By using an octree grid, the large number of cells that govern performance in the Eulerian stage are reduced,

though there is notable additional overhead in the generation and use of the octree itself. The net result is the production of the first fully adaptive (in both the Eulerian and Lagrangian stages) incompressible hybrid fluid simulation.

A. Future work

Future work remaining to be done on this method includes improving fluid surface reconstruction from particles, performance acceleration and dealing with an occasional gap problem.

First, it will be necessary to develop a surface reconstruction technique, which can produce a renderable surface from particles without a smoothing operation. Fluid reconstruction is difficult, especially when the fluid simulation includes lots of tiny features such as splashing or splay. These difficulties occur in both Eulerian and Lagrangian fluid models. Currently, a dominant surface reconstruction method for the Eulerian fluid model, the particle level set method, suffers from numerical difficulties such as mass loss and the diffusion of small-scale features. However, it is still popular since it is able to generate a high-quality surface. In the Lagrangian fluid models, surface reconstruction is even more difficult, compared to the particle level set method, due to the difficulties in reconstructing a smooth surface from particles.

One of our research goals throughout this dissertation was to capture interesting flow features and small detailed surface features from the particles. Our fully adaptive approach allows us to capture surface features using small octree cells and split particles, together with a surface tension force. However, this creates one of the worst conditions for producing a visually pleasing and renderable fluid surface, since there is too much fine detail. Using a smoothing operation for the fluid surface would ruin many of the small detailed features but the absence of smoothing leads

to a fluid surface with lots of bump artifacts. Thus, it is still necessary to develop a surface reconstruction technique which can extract a visually pleasing surface that does not smooth out desired features of the fluid surface, yet eliminates unwanted bump artifacts.

Second, the reimplementation of the fully adaptive approach using hardware-based programming could be an important way to improve the performance of our simulation. One of the primary costs for our approach is in finding neighboring nodes or finding all nodes in a specific region within the octree algorithm. The simulation performance could be significantly improved through the reimplementation of the octree algorithms on a programmable GPU.

Third, our current implementation uses a simple method for finding the fluid cells, which is based on particle location in the octree grid. Actually the method provides better performance for our CPU-based simulation due to avoiding a step for finding the neighbor nodes on the octree grid. Using this approach, the simulation suffers from an occasional gap problem. When a big cell on the octree grid is recognized as an air cell inside the fluid, the fluid flow toward the cell leads to volume loss with high pressure. The problem can be resolved by removing all air cells inside the fluid by adding an extra step. Even though the simple method speeds up the simulation, it is not a valid method in terms of physically based simulation because the fluid cells ought to be determined based on particle volume. However, using particle volume for finding the fluid cells causes low performance compared to the simple method. An area for future work, then, is to find a method for determining fluid cells that is faster than computing coverage for every particle, but more reliable than just examining grid cells.

REFERENCES

- [1] Adalsteinsson, D., Sethian, J.A.: The fast construction of extension velocities in level set methods. *J. Comput. Phys.* **148**(1), 2–22 (1999). DOI <http://dx.doi.org/10.1006/jcph.1998.6090>
- [2] Adams, B., Pauly, M., Keiser, R., Guibas, L.J.: Adaptively sampled particle fluids. In: SIGGRAPH '07: ACM SIGGRAPH 2007 papers, p. 48 (2007)
- [3] Bhattacharya, P.: Efficient Neighbor Finding Algorithms in Quadtree and Octree. M.T. Thesis, Dept.Comp. Science, India Inst. Technology, Kanpur (2001)
- [4] Blinn, J.F.: A generalization of algebraic surface drawing. *ACM Trans. Graph.* **1**(3), 235–256 (1982). DOI <http://doi.acm.org/10.1145/357306.357310>
- [5] Brackbill, J.U., Ruppel, H.M.: Flip: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *J. Comput. Phys.* **65**(2), 314–343 (1986). DOI [http://dx.doi.org/10.1016/0021-9991\(86\)90211-1](http://dx.doi.org/10.1016/0021-9991(86)90211-1)
- [6] Clavet, S., Beaudoin, P., Poulin, P.: Particle-based viscoelastic fluid simulation. In: SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 219–228. ACM Press, New York, NY (2005). DOI <http://doi.acm.org/10.1145/1073368.1073400>
- [7] Courant, R., Friedrichs, K., Lewy, H.: On the partial difference equations of mathematical physics. *IBM Journal* pp. 215–234 (1967)
- [8] Desbrun, M., Gascuel, M.P.: Smoothed particles: a new paradigm for animating highly deformable bodies. In: Proceedings of the Eurographics workshop on Computer animation and simulation '96, pp. 61–76. Springer-Verlag New York, Inc., New York, NY (1996)

- [9] Desbrun, M., Gascuel, M.P.: Space-time adaptive simulation of highly deformable substances. In: INRIA Technical Report (1998)
- [10] Enright, D., Losasso, F., Fedkiw, R.: A fast and accurate semi-lagrangian particle levle set method. *Computers and Structures* **83**, 479–490 (2005)
- [11] Enright, D., Marschner, S., Fedkiw, R.: Animation and rendering of complex water surfaces. In: SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques, pp. 736–744. ACM Press, New York,NY (2002). DOI <http://doi.acm.org/10.1145/566570.566645>
- [12] Foster, N., Fedkiw, R.: Practical animation of liquids. In: SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 23–30. ACM Press, New York,NY (2001). DOI <http://doi.acm.org/10.1145/383259.383261>
- [13] Foster, N., Metaxas, D.: Realistic animation of liquids. *Graphical Models and Image Processing* (58(5)), 471–483 (1996)
- [14] Foster, N., Metaxas, D.: Realistic animation of liquids. *Graph. Models Image Process* **58**(5), 471–483 (1996). DOI <http://dx.doi.org/10.1006/gmip.1996.0039>
- [15] Greenwood, S.T., House, D.H.: Better with bubbles: enhancing the visual realism of simulated fluid. In: SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 287–296. Eurographics Association, Aire-la-Ville, Switzerland (2004). DOI <http://doi.acm.org/10.1145/1028523.1028562>
- [16] Guendelman, E., Bridson, R., Fedkiw, R.: Nonconvex rigid bodies

- with stacking. ACM Trans. Graph. **22**(3), 871–878 (2003). DOI <http://doi.acm.org/10.1145/882262.882358>
- [17] Harlow, F., Welch, J.: Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. J. Fluids. Phys. **181**(8) (1965)
- [18] Hong, J.M., Kim, C.H.: Discontinuous fluids. In: SIGGRAPH '05: ACM SIGGRAPH 2005 Papers, pp. 915–920. ACM Press, New York, NY (2005). DOI <http://doi.acm.org/10.1145/1186822.1073283>
- [19] Hong, W., House, D.H., Keyser, J.: Adaptive particles for incompressible fluid simulation. The Visual Computer **24**(7-9), 535–543 (2008)
- [20] Irving, G., Guendelman, E., Losasso, F., Fedkiw, R.: Efficient simulation of large bodies of water by coupling two and three dimensional techniques. In: SIGGRAPH '06: ACM SIGGRAPH 2006 Papers, pp. 805–811. ACM Press, New York, NY (2006). DOI <http://doi.acm.org/10.1145/1179352.1141959>
- [21] Kim, J., Cha, D., Chang, B., Koo, B., Ihm, I.: Practical animation of turbulent splashing water. In: SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 335–344. Eurographics Association, Aire-la-Ville, Switzerland (2006)
- [22] Koshizuka, S., Tamako, H., Oka, Y.: A particle method for incompressible viscous flow with fluid fragmentation. Computational Fluid Dynamics **181**(4), 29–46 (1995)
- [23] Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. In: SIGGRAPH '87: Proceedings of the 14th annual

- conference on Computer graphics and interactive techniques, pp. 163–169. ACM Press, New York, NY (1987). DOI <http://doi.acm.org/10.1145/37401.37422>
- [24] Losasso, F., Gibou, F., Fedkiw, R.: Simulating water and smoke with an octree data structure. In: SIGGRAPH '04: ACM SIGGRAPH 2004 Papers, pp. 457–462. ACM Press, New York, NY (2004). DOI <http://doi.acm.org/10.1145/1186562.1015745>
- [25] Lucy, B.L.: A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal* **82**, 1013–1024 (1977)
- [26] M.Pritchard: Direct Access Quadtree Lookup. Game Programming Gems 2, ed. M. Deloura, Charles River Media, Hingham, MA, pp.394-401 (2001)
- [27] Müller, M., Charypar, D., Gross, M.: Particle-based fluid simulation for interactive applications. In: SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 154–159. Eurographics Association, Aire-la-Ville, Switzerland (2003)
- [28] Osher, S., Fedkiw, R.: Level Set Methods and Dynamic Implicit Surfaces. Springer Verlag (2002)
- [29] Premoe, S., Tasdizen, T., Bigler, J., Lefohn, A., Whitaker, R.T.: Particle based simulation of fluids. In: Eurographics, vol. 22, pp. 401–410 (2003)
- [30] Reynolds, O.: An Experimental Investigation of the Circumstances Which Determine Whether the Motion of Water Shall Be Direct or Sinuous, and of the Law of Resistance in Parallel Channels. *Philosophical Transactions Series I* **174**, 935–982 (1883)

- [31] Samet, H.: *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA (1990)
- [32] Sethian, J.A.: *Level Set Methods and Fast Marching Methods*. Cambridge University Press (1999)
- [33] Song, O.Y., Shin, H., Ko, H.S.: Stable but nondissipative water. *ACM Trans. Graph.* **24**(1), 81–97 (2005). DOI <http://doi.acm.org/10.1145/1037957.1037962>
- [34] Stam, J.: Stable fluids. In: *SIGGRAPH '99: ACM SIGGRAPH 1999 Papers*, pp. 121–128 (1999)
- [35] Wang, H., Mucha, P.J., Turk, G.: Water drops on surfaces. In: *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pp. 921–929. ACM Press, New York, NY (2005). DOI <http://doi.acm.org/10.1145/1186822.1073284>
- [36] Zhao, H.: A fast sweeping method for eikonal equations. In: *Mathematics of Computation*, vol. 74, pp. 603–627 (2004)
- [37] Zhu, Y., Bridson, R.: Animating sand as a fluid. In: *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pp. 965–972. ACM Press, New York, NY (2005). DOI <http://doi.acm.org/10.1145/1186822.1073298>

VITA

Woo-suck Hong started his Ph.D. study in 2003 at Department of Computer Science, Texas A&M University, TAMU 3112 College Station, TX 77843-3112. He received the B.S. and M.S. degrees in Computer Science from Pukyong National University, Korea in 1997 and 2001. His research interests include physically based modeling, scientific visualization, and computer animation.