Gaming Media and Social Effects

Chang-Hun Kim
Sun-Jeong Kim
Soo-Kyun Kim
Shin-Jin Kang

# Real-Time Visual Effects for Game Programming

Springer

# Gaming Media and Social Effects

**Editor-in-chief**

Henry Been-Lirn Duh, Hobart, Australia

**Series editor**

Anton Nijholt, Enschede, The Netherlands

More information about this series at http://www.springer.com/series/11864

Chang-Hun Kim · Sun-Jeong Kim
Soo-Kyun Kim · Shin-Jin Kang

# Real-Time Visual Effects for Game Programming

Springer

Chang-Hun Kim
Computer Science and Engineering
Korea University
Seoul
Korea, Republic of (South Korea)

Soo-Kyun Kim
Game Engineering
Pai Chai University
Daejeon
Korea, Republic of (South Korea)

Sun-Jeong Kim
Convergence Software
Hallym University
Chuncheon
Korea, Republic of (South Korea)

Shin-Jin Kang
School of Game
Hongik University
Sejong
Korea, Republic of (South Korea)

ISSN 2197-9685 ISSN 2197-9693 (electronic)
Gaming Media and Social Effects
ISBN 978-981-287-486-3 ISBN 978-981-287-487-0 (eBook)
DOI 10.1007/978-981-287-487-0

Library of Congress Control Number: 2015934438

Springer Singapore Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer Science+Business Media Singapore Pte Ltd. is part of Springer Science+Business Media (www.springer.com)

# Acknowledgments

We would like to specially thank Jeong-Mo Hong. In our lab, he was a pioneer in the field of fluid simulation and inspired colleagues with a lot of brilliant ideas. Also we really appreciate the great support of Jaeho Im, MyungJin Choi, Jong-Hyun Kim, Ki-hoon Kim, and TaeHyeong Kim. Finally, we wish to express gratitude and love to our family.

This book was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (No. 2014R1A2A2A01007143).

# Contents

# Introduction

Over the past 20 years, Computer Graphics (CG) have improved significantly, and now contribute to various industries. In particular, CG enables the realistic rendering of water, fire, smoke, fabric, fur, and wind effects in the virtual world. Movies and games are two representative industries to which CG technology is applied commercially. In these fields, CG is called *Visual Effects*.

Visual effects in movies often focus on realism. They are planned and manufactured on the assumption that sufficient hardware resources are available. Each frame for a movie takes several hours to be rendered and is displayed on DVD at a rate of 24/30 frames per second. These high-quality realistic visual effects provide people with a high degree of immersion in the virtual world. In movies, the storytelling is linear and cameras move along planned tracks. Therefore, audiences are permitted to see only limited scenes following these camera paths. These features impose spatiotemporal restrictions on the visual effects in movies, allowing them to be produced in independent studios.

Visual effects in games focus on real-time rendering under limited memory and CPU capacity. Frames should be rendered at 30/60 frames per second on the players machine. Storytelling within games is nonlinear, with players able to control the camera and navigate around a virtual world with few geometric restrictions. The main purpose of games is that players should enjoy themselves. Because the emotional state of enjoyment is difficult to evaluate, game development follows a long-term iterative process. During this process, games are frequently modified, requiring the organization of additional effects teams.

The visual effects of water, fire, smoke, and wind for movies and games have different development processes, but share common CG algorithms. Specifically, water, fire, smoke, and ice effects are realistically implemented in movies, but the complexity of their algorithms and hardware limitations mean they cannot be successfully implemented in games. This book introduces some common physics-based rendering algorithms for games, and then discusses methods for improving the real-time rendering process.

The remainder of this book is organized as follows. The basics of visual effects are introduced in Chap. 1. Chapters 2–4 then present the simulation schemes for water, smoke, fire, ice, and snow. Chapter 5 introduces techniques for controlling fluid animation and fluid interaction, such as mechanisms for collision detection between solids and fluids. Programming issues related to real-time visual effects are discussed in Chap. 6.

# Chapter 1
# Basic Concepts of Visual Effects

**Abstract** This chapter introduces the fundamental theory for understanding the entire contents of this book. A method to solve fluid equation for fire, water, and smoke simulations is explained. And also, background knowledge and research trends for modeling the natural environment such as ice formation are described. In addition to fluid dynamics, a coupling problem for solid–fluid collision detection and various methods of controlling fluid animation are dealt with. This chapter is organized as follows: underlying concepts for simulations of water, smoke, fire and ice are presented in Sects. 1.1–1.3. Collision response of fluid and target-driven fluid animation are discussed in Sect. 1.4.

## 1.1 Water

In CG, the simulation of water is implemented by various schemes. There are two main implementation methods: grid-based and particle-based. The grid-based method defines water as small particles, and controls the movement of the water by storing the physical quantity of each particle in a grid. In contrast, the particle-based method controls the movement of water by replacing its molecules with $N$ particles and solving the Navier–Stokes equation using the physical quantities of each particle, such as their density, pressure, and velocity. The grid-based method can solve very complex equations and is faster than the particle-based method for large-scale simulations, although some numerical dissipation can occur. The performance of the particle-based method can deteriorate when large numbers of particles are used, but it is still possible to generate relatively accurate simulations. The current trend for water simulation is moving to particle-based simulations.

Because the surface tension property of water is unlike that of any other fluid, surface tension is used to produce realistic water simulations. Whereas smoke spreads and dissipates in the air during simulation, water simulations must guarantee that the volume is conserved and that an explicit surface is maintained. Therefore, in water simulations, the main issues are to minimize the numerical dissipation occurring in the calculation process and accurately track the surface of the water. The level-set method is a typical surface tracking technique that generates and uses a signed distance field (SDF). Further details about SDF are given in Appendix A.1.

C.-H. Kim et al., *Real-Time Visual Effects for Game Programming*,
Gaming Media and Social Effects, DOI 10.1007/978-981-287-487-0_1

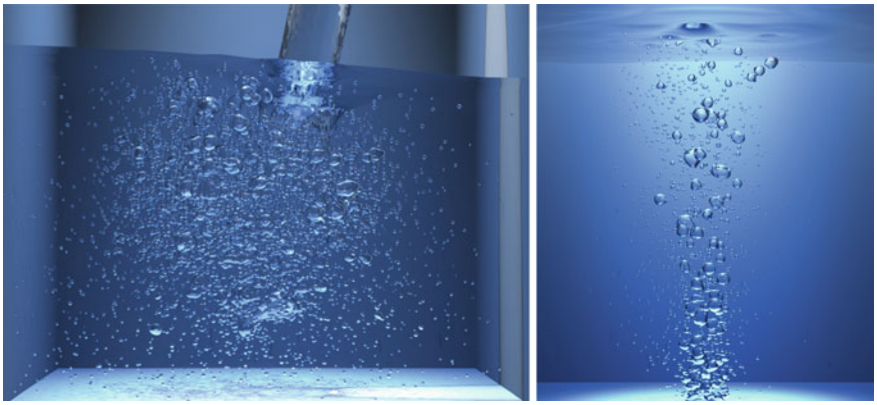


**Fig. 1.1** Bubbles alive [10]: Pouring water with turbulent multiphase bubble flows (*left*) and rising bubbles in calm water (*right*)

We cannot achieve sufficiently good water simulation using elaborate surface tracking to calculate the movement of the water, because motion is generated from a variety of splashes, bubbles, and so on. Therefore, there has been a considerable amount of research in the representation of splashes and bubbles (Fig. 1.1).

This book presents water simulation techniques that track the surface and precisely calculate the motion using a grid-based approach and the level-set method. In addition, to ensure a realistic simulation, a variety of techniques are presented to generate bubbles and express the movement of water under surface tension.

## *1.1.1 Eulerian Method for Water and Bubbles*

### 1.1.1.1 Multiphase Flows

We consider a multiphase Navier–Stokes solver that includes buoyancy and surface tension. Our system is designed to use the merits of Navier–Stokes simulation schemes developed for CG animation.

$$\frac{\partial(\rho\mathbf{u})}{\partial t} = \mu\nabla\cdot(\nabla\mathbf{u}) - \nabla\cdot(\rho\mathbf{u}\mathbf{u}) - \nabla P + \rho\mathbf{g} + \int_{\Gamma(t)} \sigma\kappa\mathbf{n}\delta(\mathbf{x}-\mathbf{x}_f)ds \tag{1.1}$$

### 1.1.1.2 Bubbles

We introduce one more subject pertaining to liquid animation: bubbles. Bubbles are pockets of air enclosed by liquid, and can be found wherever liquid and air coexist. Two flows must be considered: those occurring inside and outside of the bubble.

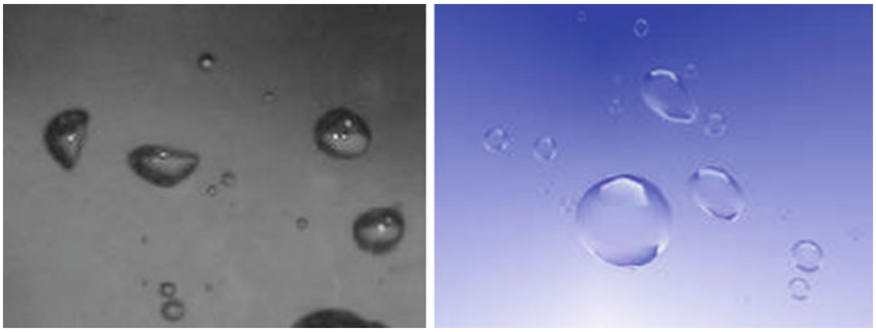


**Fig. 1.2** Rising bubbles in liquids [7]: Photo image (*left*) and rendered image (*right*)

Differences in specific gravity between the two fluids generate buoyancy forces, and surface tension forces are exerted at the interface between the two fluids. We will develop a new fluid animation method in which liquid and gas interact with each other using the example of bubbles rising in water (Fig. 1.2).

#### 1.1.1.3 Minimum-Stress Surface Tension

In the multiphase fluid configuration, the surface is constructed as the blue lines in Fig. 1.3. To construct the blue lines, we first find the zero-stress surface of each cell (red line), and gradually move toward these surfaces. This tendency is equivalent to surface tension defined on the simulation grids. Finally, we add this force to the velocity as a body force. The Navier–Stokes solver can then handle this scenario naturally.

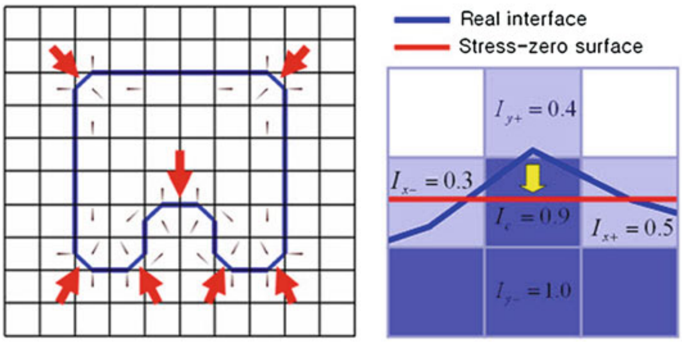


**Fig. 1.3** Minimum-stress surface tension method (*left*); the velocity field induced by the surface tension forces (*right*)

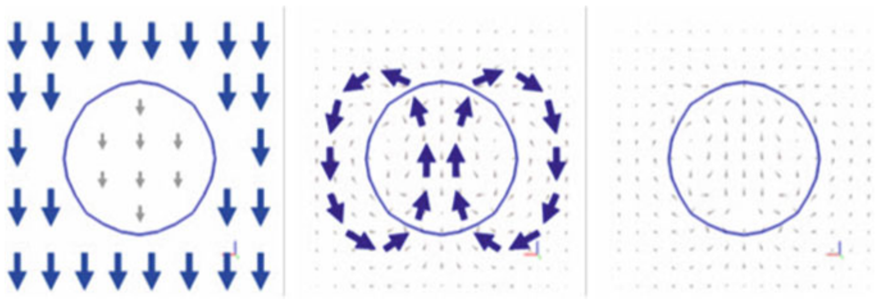


**Fig. 1.4** *Left* step A, *middle*: step B, and *right* step C

### 1.1.1.4 Numerical Simulation of Buoyancy

Instead of employing experimental equations, we numerically simulate buoyancy effects (Fig. 1.4). This method can be easily implemented as part of a multiphase simulation system.

**[Step A]** Add gravity forces to the velocity field, as the buoyancy comes from the density difference between two materials.
**[Step B]** Correct the velocity field by solving the hyperbolic partial differential equations.
**[Step C]** Trace the interfaces following the velocity field.

### 1.1.1.5 Mass Conservation

Because this method is partially based on the volume-of-fluid scheme, we can explicitly correct the total mass of the enclosed fluids after each interface-capturing step (Fig. 1.5).

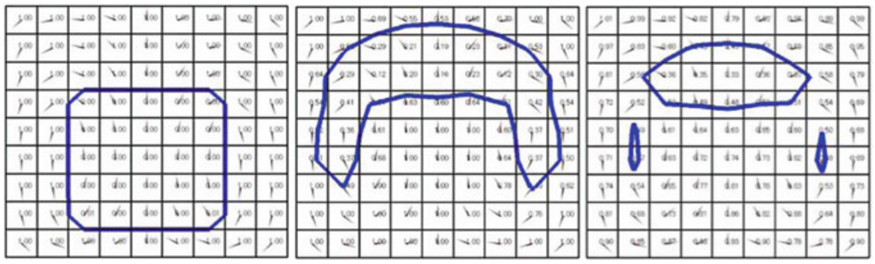


**Fig. 1.5** *Left* initial state; *middle*: mass conservation step skipped; and *right* mass conservation step used

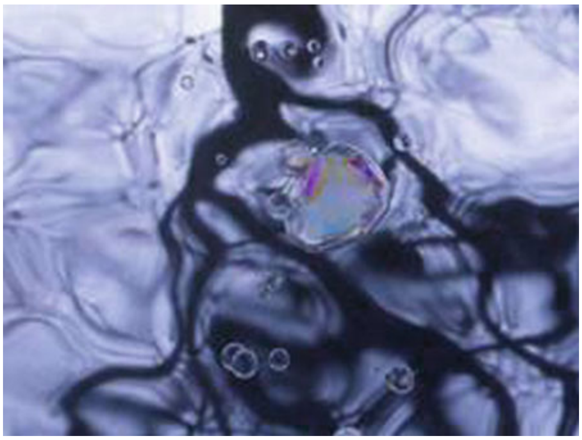


**Fig. 1.6** Rising bubbles in a liquid [7]

#### 1.1.1.6 Results

Finally, we can animate rising bubbles in a liquid within the Navier–Stokes fluid simulation scheme. A number of particles were employed to make the animated scene more lively, and the interfaces were rendered by the vertex-shader technique (Fig. 1.6).

### *1.1.2 Hybrid Method for Water and Bubbles*

We now propose a hybrid method for simulating multiphase fluids such as bubbly water. The appearance of subgrid visual details is improved by incorporating a new bubble model based on smoothed particle hydrodynamics (SPH) into a Eulerian grid-based simulation that handles the background flow of large bodies of water and air. To overcome the difficulty of simulating small bubbles in the context of multiphase flows on a coarse grid, we heuristically model the interphase properties of water and air by means of the interactions between bubble particles. As a result, we can efficiently animate the lively motion of bubbly water with small-scale details.

#### 1.1.2.1 Hybrid Eulerian and Lagrangian Approach

We can use the Eulerian method to generate the background motion of water and air bodies that are large enough to be captured using a simulation grid using a single-CPU machine. The bubbling details that are too small to be handled on the grid are simulated by SPH. We build our system on the particle level-set (PLS) fluid solver to generate bubble particles by incorporating escaped particles back into the SPH system as bubbles (Fig. 1.7).

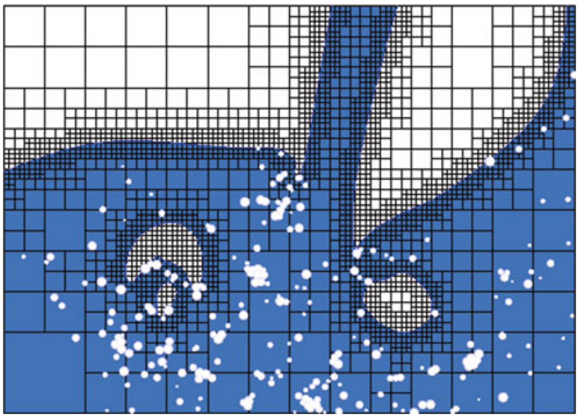


**Fig. 1.7** Schematic outline of our hybrid system [10]. The body of water is colored *blue* and bubble particles are drawn as *white circles*

#### 1.1.2.2 Drag Force at a Bubble Surface

Because the density of water is 800 times that of air, the force applied to water by air is insignificant. Conversely, water induces buoyancy and a drag force on bubbles. A drag force is applied to the upper portion of the air bubble by buoyancy. Ellingsen and Risso [4] observed that an air bubble injected into water retains its ellipsoidal shape while it rises, implying that the drag force only acts on the upper portion of the air bubble. We can control the shape of air bubbles using the drag force (Fig. 1.8).

#### 1.1.2.3 Interchangeable SPH and Level Set

We will display small-scale bubble motion based on Lagrangian SPH in a grid-based simulation to detect detailed subgrid features. In fact, the cohesion force in SPH enables particles to merge, yielding a higher density and creating air bubbles that are large enough to be depicted in a grid. Though SPH was originally created to describe subgrid details, the integration of particles that are larger than subgrid size reduces the simulation accuracy. When the size of the particles in SPH outgrows that of the subgrid, we turn to a grid-based level set.

#### 1.1.2.4 Swirling Bubbly Water

The effect of surface tension is dynamically and realistically represented within a multiphase fluid simulation. Air bubbles are seeded with ‘bubble particles,’ which move randomly. These molecule-like movements modify the surface of the air bubbles and generate turbulence in the water. The surface tension between air bubble and water, determined by the composition of the water, remains constant regardless of bubble size. However, external forces produce unstable fluid motion as the surface tension strives to exert itself, causing bubbles to split and merge. The bubble particles

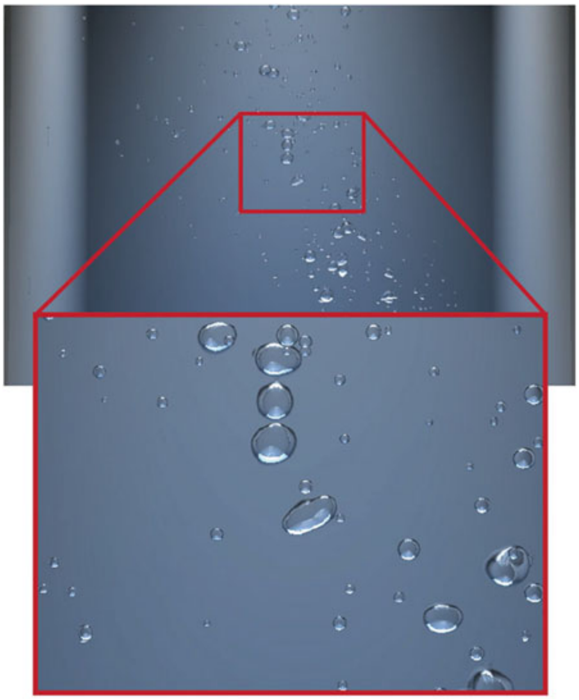


**Fig. 1.8** Ellipsoidal shape of air bubbles using our proposed drag force [16]

can also be used to mitigate the numerical dissipation usually experienced in grid-based fluid simulations, restoring the lost volume of individual bubbles. The realistic tearing of bubble surfaces is shown in a range of examples (Figs. 1.9 and 1.10).

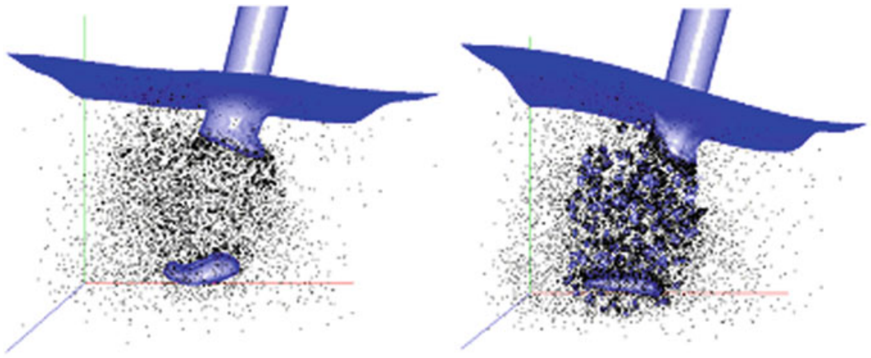


**Fig. 1.9** Example of pouring water [17]: SPH particles showing the motion of the subgrid (*left*) and air bubbles formed by converting merged SPH particles into a level set (*right*). *Dots* denote SPH particles

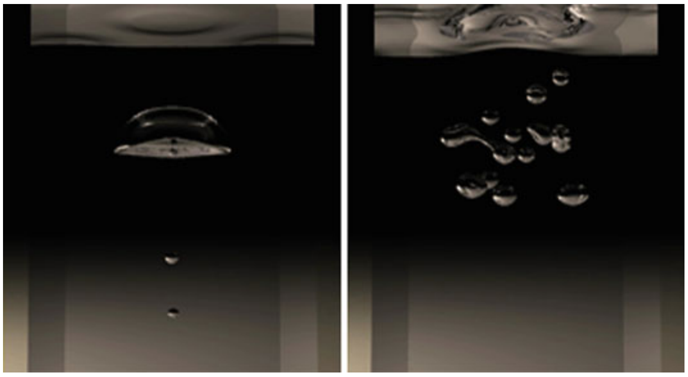


**Fig. 1.10** An air bubble rising in calm water (*left*); air bubbles spiraling upwards (*right*) [18]

### 1.1.3 *High-Order Surface Tracking*

At the interface between different fluids, properties such as density, viscosity, and molecular cohesion are discontinuous. To realistically animate small-scale details of incompressible viscous multiphase fluids, we focus on the discontinuities in the state variables that express these properties. The surface tension of both the free surface and the bubble is modeled using a jump condition in the pressure field. Discontinuities in the velocity gradient field, driven by viscosity differences, are also considered. To obtain the derivatives of the pressure and velocity fields with subgrid accuracy, we extrapolate these quantities across the interface using continuous variables based on physical properties. The numerical methods that we present are easy to implement and do not affect the performance of existing solvers. Small-scale fluid motion, such as capillary instability, breakup of liquid sheets (Fig. 1.11), and bubbly water, can all be successfully animated.

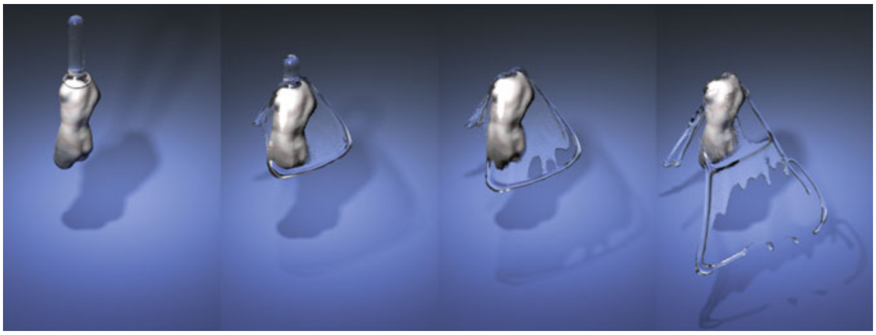


**Fig. 1.11** Breakup of a liquid sheet [9]. The effective resolution is $512^3$

#### 1.1.3.1 Discontinuous Fluid

There is a discontinuous pressure profile at the interface Γ between two different fluids. Figure 1.12 illustrates this discontinuous pressure at the interface. The pressure on the right of Γ is different to that on the left. This makes it difficult to differentiate the pressure across Γ using standard finite differencing.

#### 1.1.3.2 Anisotropic Particle Level Set

We will demonstrate how to track the surface of a multiphase fluid more accurately using the PLS method with anisotropic particles (APLS) instead of spherical particles. We use a weighted principal component analysis (WPCA) to construct the anisotropic particles, although the computational cost of this approach is high. We use the directional derivative to generate the distribution of anisotropic particles. Compared with PLS, this approach provides more surface detail, corrects numerical dissipation, and preserves the volume of the fluid (Fig. 1.13). Furthermore, we will present anisotropic particle-based fluid simulations with surface reconstructions.

### *1.1.4 Miscible Multiphase Fluids*

By modeling mass transfer phenomena, we will simulate solids and liquids dissolving or changing to other substances, and also deal with the very small-scale phenomena that occur when a fluid spreads out at the interface of another fluid. We will model the pressure at the interfaces between fluids with Darcy's Law, and represent the viscous fingering phenomenon in which a fluid interface spreads out with a fractal-like shape. Hybrid grid-based simulations and SPH will be used to simulate intermolecular

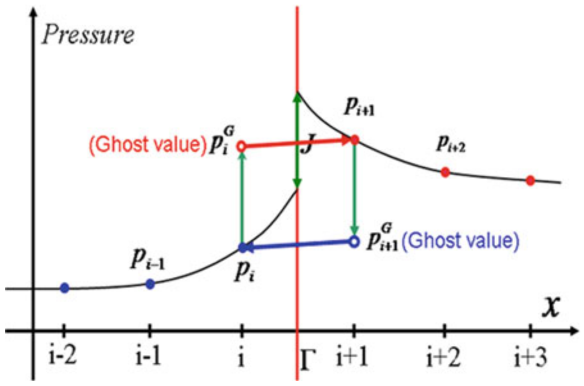


**Fig. 1.12** Discontinuous pressure field near an interface Γ

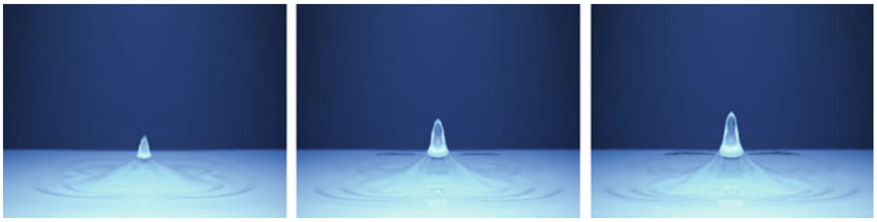


**Fig. 1.13** Water drop tower: PLS (*left*), APLS with WPCA (*middle*), and APLS with directional derivative (*right*) [15]

diffusion and attraction using particles at a computable scale. As a result, we can animate fluids mixing and objects dissolving.

#### 1.1.4.1 Viscous Fingerling

When fluids mix, we can observe that they spread out irregularly as their mixing surface makes a fractal-like shape. Viscous fingering refers to the onset and evolution of these instabilities in the displacement of fluids. The unstable flow of a fluid in a porous medium, or by analogy in a Hele–Shaw cell, has been studied for 50 years. The results have applications in areas such as enhanced oil recovery and microfluidics (Fig. 1.14).

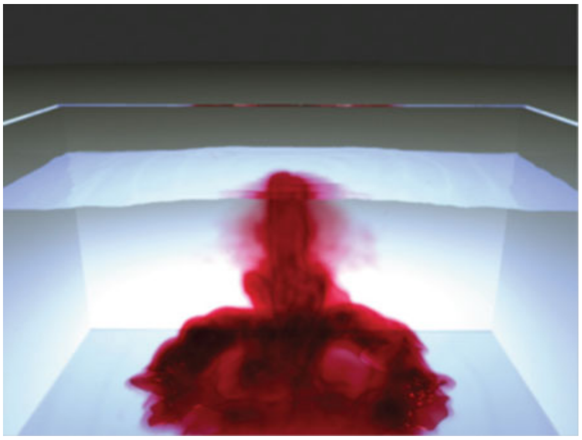


**Fig. 1.14** Dropping *red ink* [23]. Two different liquids mix (grids:256 × 256 × 128, particles: about 120,000)

#### 1.1.4.2 Conclusions

Our technique enables the modeling of miscible multiphase fluid flow by improving the handling of interfacial properties and chemical reactions. With this technique, we can construct naturalistic scenarios in which a solid body melts or liquids are mixed. These combinations of viscous fingering, chemical-based mass transfer, and molecular forces are relatively easy to model with techniques familiar to the CG community.

## 1.2 Smoke

Modeling the flow of smoke is the basis of fluid simulation. Based on smoke simulations, other fluid flow techniques are often developed.

In natural phenomena, smoke consists of nanoscale particles that form various shapes under external forces such as wind. One of the main issues in smoke simulation is the calculation and expression of the movement of a substance that is very sensitive to movement in the surrounding air. The complex motion of smoke is calculated by dividing the simulation space into a grid, and then solving the Navier–Stokes equation to discover the changes in velocity and mass across the grid. In the process of solving the equation, mathematical techniques from computational fluid dynamics (CFD) and the finite difference method (FDM) are used. Since Jos Stam presented the “Stable Fluid” approach [24], which guarantees numerical stability, his scheme has become the basis for subsequent research.

Generally, a large number of calculations are required to produce the desired smoke simulation. This is due to the problem of solving the Navier–Stokes equation in a grid, and is common to other fluid simulations. The higher the grid resolution, the more realistic and accurate the resulting simulation, although the computational cost will increase sharply. This is the trade-off between quality and cost. To overcome this problem, an adaptive approach is proposed: the space in which the smoke is actively moving is divided finely, whereas the other space is divided coarsely. A data-driven method has also been studied to give faster results using previously simulated data.

Recently, the development of multi-core and many-core processors has enabled multiple calculations to be performed at high speed. Researchers have studied the Navier–Stokes equation in a multi-thread environment, and NVIDIA presented a 3D simulation generated in real time using CUDA (Compute Unified Device Architecture). Because these results are achieved by techniques that occupy the majority of a devices resources, they are difficult to directly reproduce in games.

Some researchers have attempted to optimize the trade-off between quality and cost. That is, they employ additional operations to produce more realistic results at the same grid resolution. For example, vorticity confinement is a commonly used scheme introduced by Fedkiw et al. in “Visual Simulation of Smoke” [5]. The turbulence of smoke is represented by adding noise to the grid to compute the curl. In addition, techniques such as using a subgrid or inserting particles to increase the number of vortices have been studied (Fig. 1.15).

Smoke has also been represented using particles rather than a grid. Filament-based simulations can produce more sophisticated results than grid-based methods in small scenes. However, it is difficult to apply the filament-based scheme to the large-scale scenes that are often shown in movies and games.

### *1.2.1 Animating Smoke with Dynamic Balance*

We will propose a numerical method for maintaining a dynamic rolling motion in animated gaseous phenomena, such as smoke, while avoiding dissipation due to numerical error. The errors induced by a semi-Lagrangian scheme are compensated using an error estimate for each time interval. We develop a new advection term, and perform vortex advection based on a vorticity confinement force. Example simulations show that this method retains smoke features, even near the center of a vortex (Figs. 1.16 and 1.17).

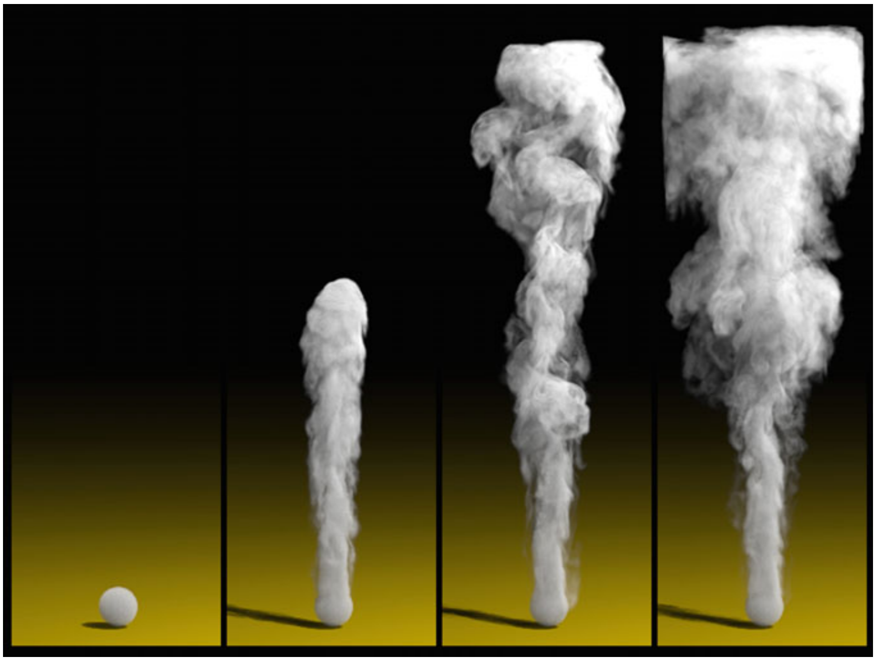


**Fig. 1.15** Results of procedural synthesis using vortex particle method for fluid simulation [26]

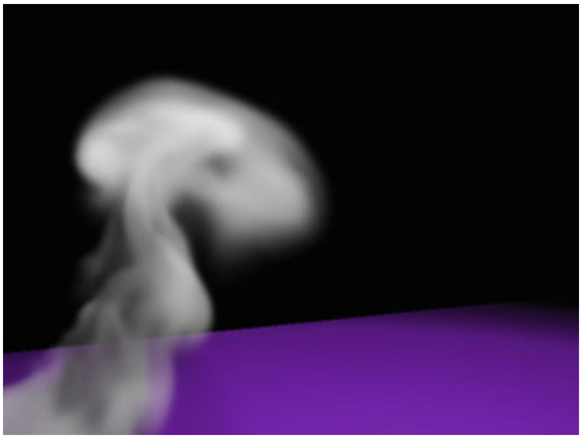


**Fig. 1.16** Smoke motion with vortex advection and error compensation [8]

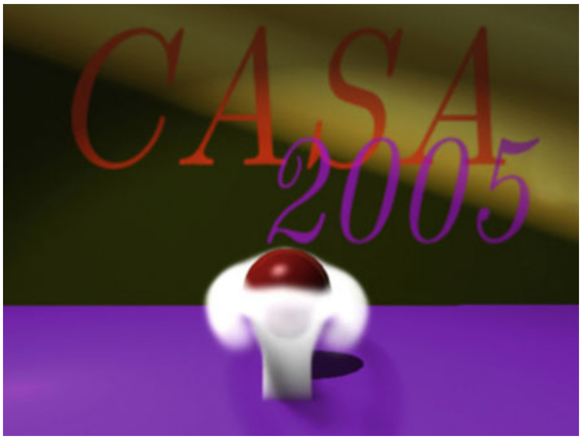


**Fig. 1.17** Rising smoke swirling over a sphere [8]

### *1.2.2 Procedural Synthesis Using Vortex Particle Method*

We will propose a fast and effective technique to improve the subgrid visual details of the grid-based fluid simulation. Our approach procedurally synthesizes the flow fields coming from the incompressible Navier–Stokes solver and the vorticity fields generated by the vortex particle method for subgrid turbulence. We can efficiently animate highly turbulent, swirling smoke with small-scale details. Because this technique does not solve the linear system in high-resolution grids, the fluid simulation can be performed more rapidly. We can easily estimate the influence of turbulence and swirling on the fluid flow (Figs. 1.18 and 1.19).

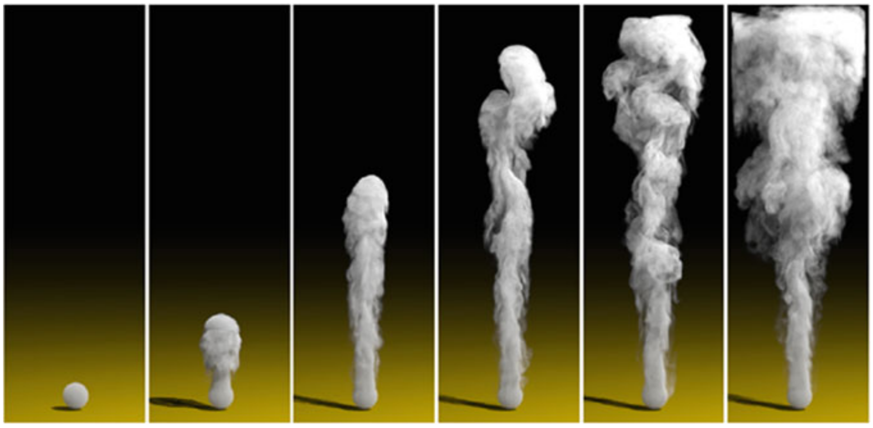
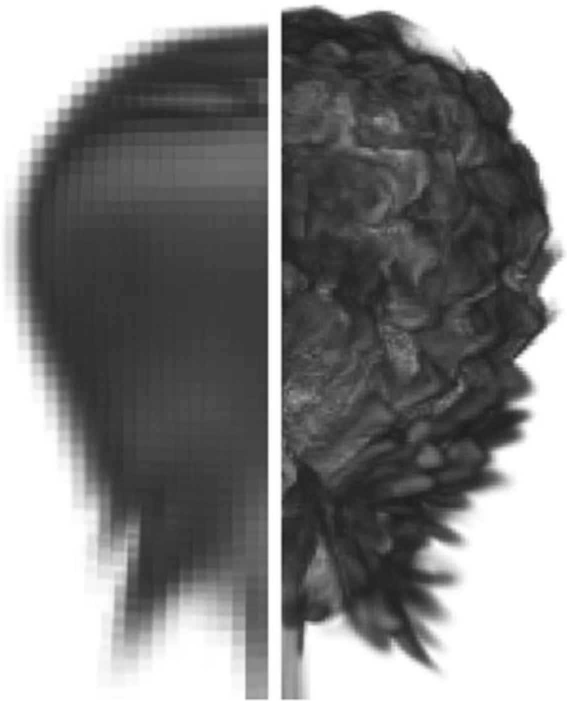


**Fig. 1.18** Rising smoke [26]

**Fig. 1.19** Explosion-like effect [26]



## 1.3 Fire and Ice

Fire simulation is a very popular effect in movies and computer games, and is a challenging research field. In natural phenomena, fire occurs through extremely complex chemical processes involving fuel and high temperatures. In CG, the aim is to render and control flame realistically with a simplified model.

Early research attempted to control the fire motion using particles. “Realistic and Controllable Fire Simulation” [1] is a representative introduction to this approach, simulating fire with an object mesh via propagation, animation and rendering, and producing flame motion with its skeleton.

Since then, grid-based simulation has become the main approach to modeling fire. The framework described in “Physically Based Modeling and Animation of Fire” [3] proposed a state model that handled the process of combustion, i.e., burning fuel results in high temperature, and simulated flame using the location and temperature of the fuel. The position of the fuel (specifically, the reaction zone) was traced by the level-set method, and flame was modeled using the surface normal direction and curvature term. In addition, buoyancy caused by temperature differences was modeled, and a rendering technique was presented that defines the color of the flame according to the theory of black body radiation.

Based on this research, schemes that produce more realistic and detailed flame shapes and motion have been studied. “Wrinkled Flames and Cellular Patterns” [11] proposed the detonation shock dynamics (DSD) framework for flame wrinkling and cellular patterns.

Recently, industrial demands and advances in computing power have resulted in a change in research direction. The game industry requires plausible flame movement to be drawn in real time. Various approaches have been developed for this problem, such as creating a sprite image that is precisely rendered in advance or adding lighting effects with GPU shaders to generate realistic results. The film industry often requires large-scale explosions and flames caused by strong turbulence. Approaches that reduce the computational load of these effects have been studied, such as coarsely dividing the space in which the motion of fire is to be calculated and then adding details later. “Directable, High-Resolution Simulation of Fire on the GPU” [13] proposed a technique to calculate a vector field on a coarse grid by solving the Navier–Stokes equation, advecting particles through a vector field, and performing a further computation to obtain detailed information according to the viewpoint of the user. The scene is then composited by mapping a fire texture to each particle.

Researchers have also studied the burning or deformation of objects by temperature variation. “Multi-Representation Interaction for Physically-Based Modeling” [21] proposed a technique to model and simulate the heat transfer inside an object and its effect on the air around the object. “Melting and Burning Solids into Liquids and Gases” [20] presented a simulation of the deformation of objects by heat, and “Physically Based Simulation of Solid Objects’ Burning” [19] simulated the free-form deformation of objects.

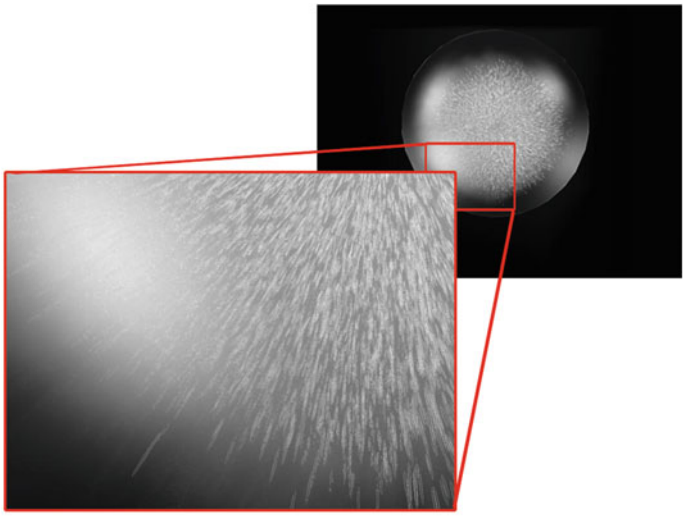


**Fig. 1.20** Simulation of air bubble shapes like needles [14]

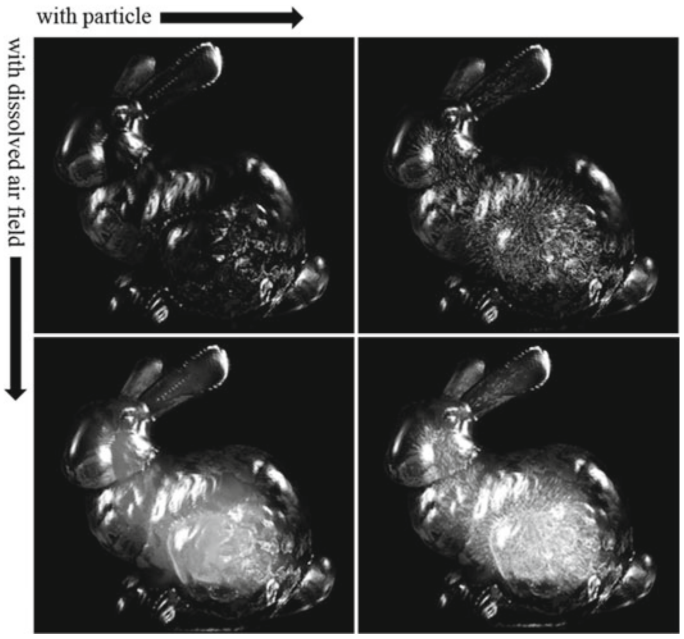


**Fig. 1.21** Without any opaque area (*top left*); Using particles only (*top right*); Dissolved air field only (*bottom left*); Both particles and the field (*bottom right*) [14]

In CG, the simulation of natural phenomena has several important issues, such as how to produce fast and realistic simulation results. In the winter, we often see snow and ice, so simulations of these phenomena are often used in films and advertising. However, the rendering scheme for a fluttering snowflake and snow piled-up on the surface of an object requires a large number of particles and considerable computation to produce realistic results. In the case of ice simulation, rendering opaque ice requires designers to spend a lot of time drawing the internal representation of the ice. Thus, transparent ice simulations are sometimes used in films and computer games. To solve the above problems, this book presents a physically based simulation method to rapidly produce exact, detailed ice and snow simulation results.

Many researchers have proposed grid-based or particle-based simulation schemes, and these give very artistic results. Recently, hybrid simulation methods have been proposed to simulate realistic ice. “A Particle-Grid method for Opaque Ice Formation” [14] presents the simulation of explicit air bubbles in the inside of ice using a particle-based method and creates nanoscale air bubbles to simulate mist using a grid-based technique (Figs. 1.20 and 1.21).

## 1.4 Fluid Interaction

Fluids often interact with the same or different materials, such as in fluid–fluid or solid–fluid interactions. In particular, fluid–fluid interactions are called a “coupling” (Fig. 1.22 left). As CG techniques are being applied to films and advertising more frequently, the simulation of solids (e.g., rigid and deformable bodies, cloth) has become an attractive research topic. This includes solid–fluid and solid–solid interactions, as well as fluid dynamics simulations of water, smoke, and fire. In CG, such fluid–fluid and solid–fluid interactions require separate fluid and solid simulation techniques, as well as highly precise and rapid computational methods to represent the visual effects.

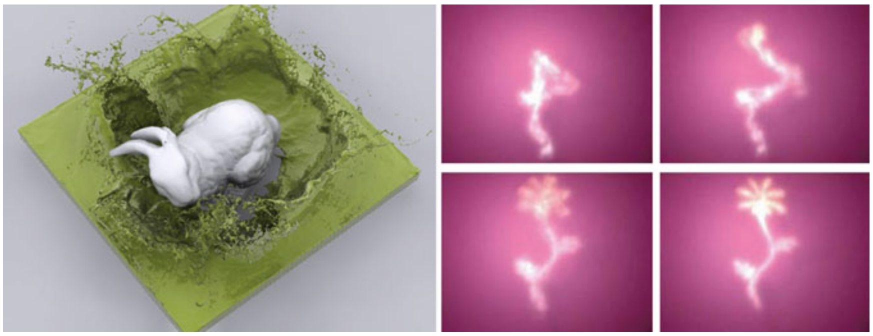


**Fig. 1.22** Coupling (*left*) and controlling a fluid (*right*)

### *1.4.1 Coupling*

One important technique for fluid interactions is collision detection. The particle-based simulation of fluid, cloth, and hair using the Lagrangian method requires a great many particles to precisely represent their deformation (Figs. 1.23 and 1.24). Hence, collision detection schemes for large numbers of particles have been studied. In a typical solid–fluid collision detection method, the velocity of fluid particles at the interface is interpolated and then used to determine the force of the solid.

Figure 1.25 illustrates a boundary condition that checks whether the ray between two adjacent cell centers whose pressures are defined intersects an object. If the boundary condition is not examined properly, a problem occurs in the thin surface layer of water formed by the collision with a thin solid object (namely, the thin water surface is quickly compressed and its density vanishes). That is, if the collision between a solid surface and a fluid particle is not detected correctly, a sticking problem or penetration problem will arise.

For particle-based simulations, adaptively sampled distance fields (ADFs) are used for the rapid computation of fluid velocities (Fig. 1.26). Because this method adaptively divides the simulation space, its simulation speed is relatively fast compared with previous methods. However, because the total computational load of this method depends on the number of particles, large-scale problems can result in a decline in simulation speed.

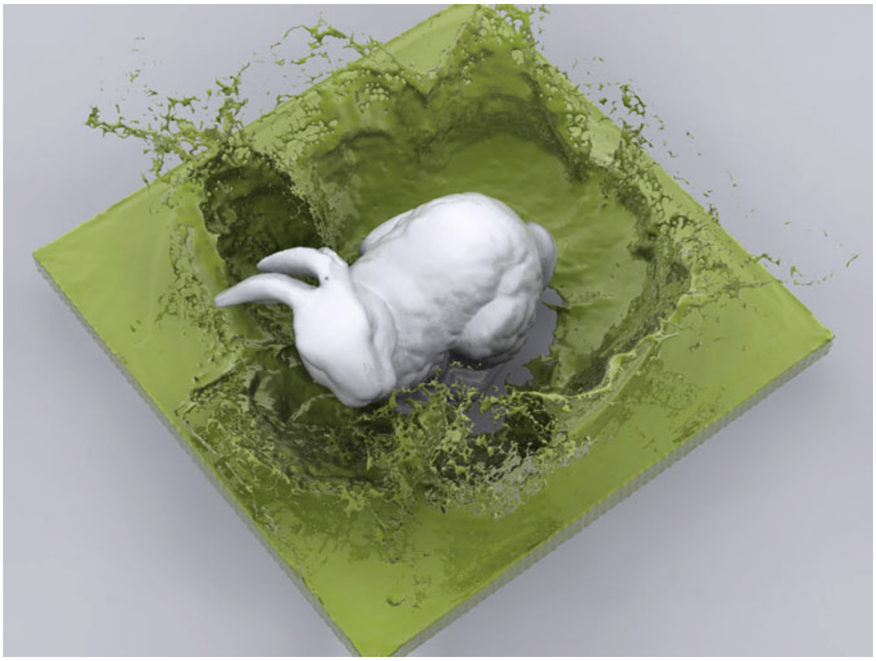


**Fig. 1.23** Coupling a fluid with a deformable body

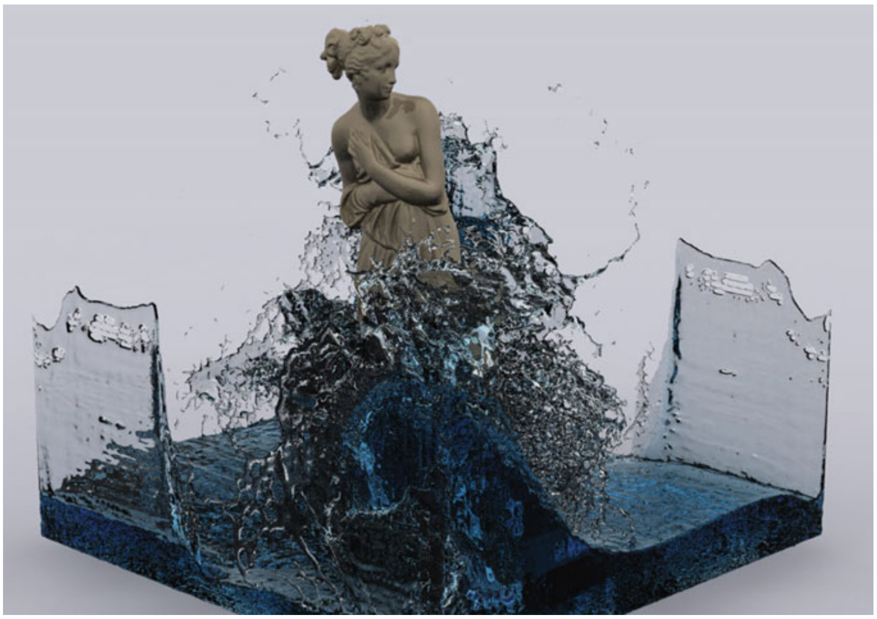


**Fig. 1.24** Coupling a fluid with a rigid body

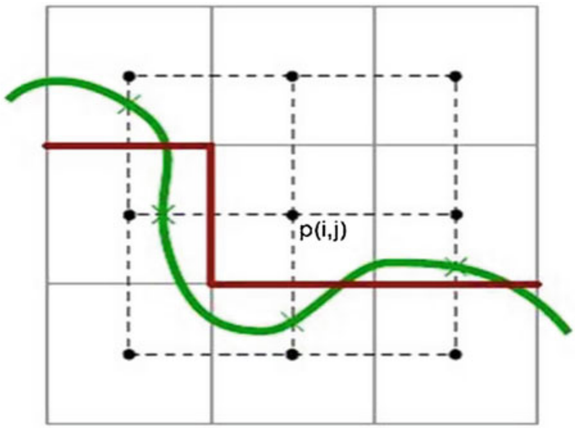


**Fig. 1.25** Neumann boundary condition (denoted in *bold red*) is enforced at a cell face if the ray between two adjacent cell centers (where pressures are defined) intersects an object

## *1.4.2 Controlling Fluid*

This section introduces a new fluid control technique that uses a geometrically induced potential field. Instead of optimizing the control forces exerted at each frame, as in previous work, a potential is added as an extra dimension to the simulation space. This coerces the fluid inside this space to form the target shape (Fig. 1.22 right). This type of shape control requires practically no additional run-time computation by the

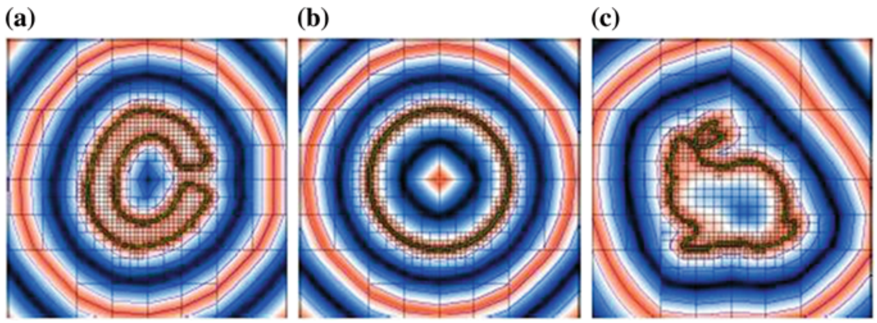


**Fig. 1.26** Various example ADFs **a** C glyph. **b** Sphere **c** Stanford Bunny

Navier–Stokes solver, and adds little overhead to the implementation. The confinement potentials are induced from geometric information given by animators, and so the control forces that take fluids to a lower potential can be determined in a preprocessing step. A slightly generalized Navier–Stokes equation for fluids in potential fields can be simulated without changing the solver itself, and a harmonic potential function can be quickly found with the Poisson solver, which is already implemented as part of the Navier–Stokes solver. Two- and three-dimensional flows designed by common methods such as hand drawing, traditional shape modeling, and key-framing can be animated efficiently with our control technique (Figs. 1.27 and 1.28).

A simple control problem in one dimension can be modeled as shown in Fig. 1.29. There is a physical entity $A$, currently at $x_0$, that we want to place at $x_{target}$. Control means checking the error and taking action to reduce it. In Fig. 1.29, the error $e$ is $x_{target} - x_0$. Control forces will be continuously exerted on $A$ such that $e$ tends toward zero. Determining these forces is the major concern of classical control theory (interested readers are referred to [6]). This framework was used by Treuille et al. [25], who developed novel techniques to determine and optimize the control forces for fluids. However, the very existence of an external controller can be burdensome to a

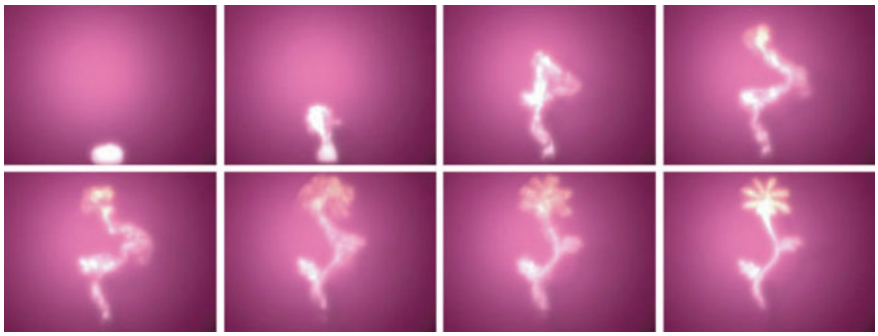


**Fig. 1.27** Two-dimensional flow control: a blossoming flower [12]

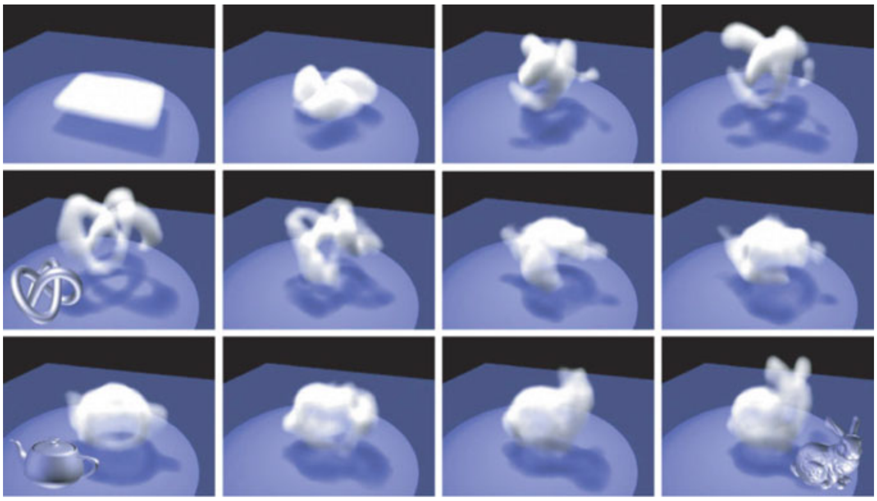


**Fig. 1.28** Three-dimensional keyframing: knot, teapot, and rabbit model [12]

**Fig. 1.29** Sample control problem

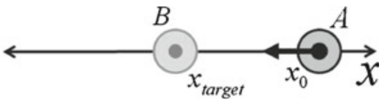
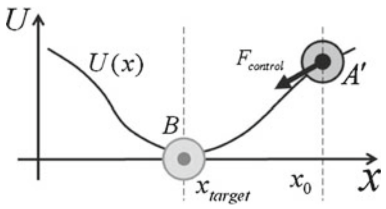


**Fig. 1.30** Our control frame work



Navier–Stokes solver. Furthermore, because fluids have many degrees of freedom, it is questionable whether this approach is applicable to complicated three-dimensional target shapes. The alternative is to insert the uncertainty into the physical model, as was done by Chenney and Forsyth [2]; for similar reasons, this is still computationally burdensome.

Figure 1.30 shows the key idea of our method. Intuitively, we can confirm that, without external control, $A'$ will start moving and will stop at $x_{target}$ at some later time, as for a rock rolling into a ravine.[1] This means that our method—adding a potential dimension to the simulation space and adjusting it to design the animation—reformulates the control problem as an initial-value problem. There is no need for control during the simulation; we control the animation by designing $U(x)$ in a preprocessing step. Because the vertical axis that we have introduced is not a geo-

[1] This is known as a potential well in physics.

metric dimension, the dimensionality and size of the simulation space have not been increased, and so the performance of the numerical simulation is maintained. The only change to the fluid dynamics resulting from the potential field is the introduction of $F_{potential}$, which is

$$F(x)_{potential} = -\nabla U(x), \tag{1.2}$$

from the energy-force relationship of Newtonian mechanics. As the potential scalar field $U(x)$ is determined during preprocessing, the gradient vector field $\nabla U(x)$ can be found at the same time. This makes it possible to simulate fluids in a potential field without lowering the performance of the Navier–Stokes solver. Details are given in the next section.

### *1.4.3 Target Driven Animation*

This section introduces a novel method of controlling a multiphase fluid so that it flows into a target shape in a natural manner (Figs. 1.31 and 1.32). To preserve the sharp detail of the target shape, we represent it as an implicit function, and construct the level set of that function. Previous approaches have added a target-driven control force as an external term. This term is then attenuated during the velocity projection step, making the convergence process unstable and causing sharp detail to be lost from the target shape. In contrast, we calculate the force on the fluid from the pressure discontinuity at the interface between phases, and integrate the control force into the projection step so as to preserve its effect. The control force is calculated using an enhanced version of the ghost fluid method (GFM), which guarantees that the fluid flows from the source shape and converges on the target shape, while achieving a more natural animation than other approaches. Our control force is merged during the projection step, avoiding the need for a post-optimization process to eliminate divergence at the liquid interface. This makes our method easy to implement using existing fluid engines, and it incurs little computational overhead. Experimental results demonstrate the accuracy and robustness of this technique.

Unlike morphing techniques, methods of controlling fluid produce a natural-looking fluid flow between a source shape and a target shape. Some of these techniques also use a multiphase fluid simulation so that liquids can be included in the animation. Control is achieved by adding external forces to the Navier–Stokes equation, ensuring the fluid flows from the source to the target configuration. However, existing techniques do not take account of the interfacial discontinuities in pressure and density that occur when two different fluids are adjacent to one another. Some important fluid properties, such as surface tension and capillary phenomena, occur at such discontinuities. In addition, the flow around objects is greatly influenced by these discontinuities at the interface. Our method improves the control of multiphase fluid simulations by considering interfacial discontinuities. To allow a user to change the direction and shape of a fluid flow, we add control forces at the projection step of a modified GFM, which is able to take account of discontinuities at the interface

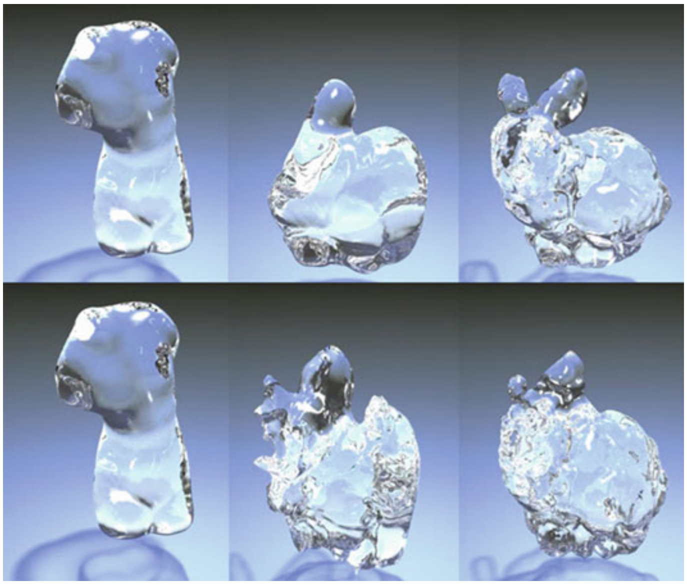
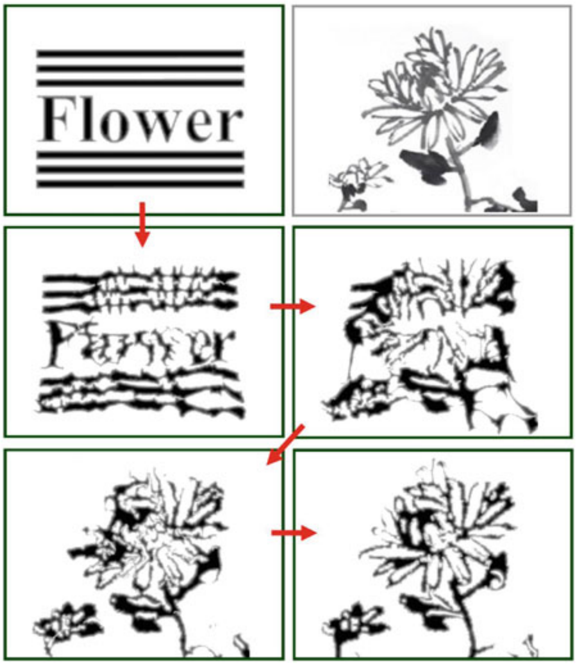


**Fig. 1.31** Liquid flowing from a Venus to a bunny. Our method (*bottom*) [22] is compared with a previous method (*top*) that uses an external force. The resolution is $128^3$

**Fig. 1.32** Liquids making the word 'Flower' flow into the target image, which is an oriental painting (*top right*) [22]



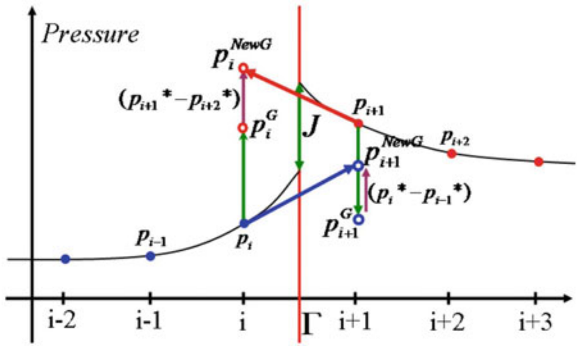


**Fig. 1.33** Discontinuous pressure field and the new ghost value

between different fluids. Users may provide an image, 3D mesh data, or sketches. Based on this input, the level set of the target shape is constructed as an SDF, from which pressure jump values can be determined.

Controlling a liquid animation by adding forces at the interface between two immiscible fluids has several advantages:

- Because we are adding the control force at the projection step, the force is accurately preserved.
- The fluid simulation is still divergence-free and robust, despite the control force, and no optimization is required.
- Additional natural forces such as gravity and buoyancy can be added to the environment as external forces. Control and external forces are added independently at different stages.
- Control forces can easily be added within an existing fluid simulation pipeline.

There is a discontinuous pressure profile at the interface Γ between two different fluids. Figure 1.33 shows the discontinuous pressure at this interface. The different pressures to the right and left of Γ make it difficult to differentiate the pressure across the interface using standard finite differencing.

The ghost values $p_i^G$ and $p_{i+1}^G$ are based on the assumption that the differential values of the pressure on either side of the interface are always equal [9]. To accommodate a pressure differential, the ghost values must be modified as follows:

$$p_i^{NewG} = p_i + J + (p_{i+1}{}^* - p_{i+2}{}^*) \tag{1.3}$$

$$p_{i+1}^{NewG} = p_{i+1} - J + (p_i{}^* - p_{i-1}{}^*), \tag{1.4}$$

where $(p_{i+1}{}^* - p_{i+2}{}^*)$ and $(p_i{}^* - p_{i-1}{}^*)$ are obtained using the pressure value $p^*$ from the time-step before the projection.

To make a liquid flow into the target shape, we impose a control pressure using the pressure jump. This causes the pressure values to change rapidly, which affects the pressure differential across the interface. The modified GFM deals with this situation efficiently.

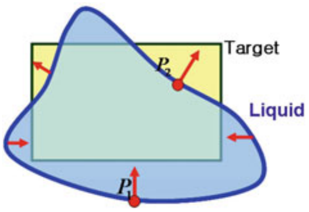


**Fig. 1.34** Shape feedback force on the liquid interface

To ensure a fluid converges to a detailed target shape, the desired shape must be clearly described. We represent the target shape as a level set with an SDF modeled by an octree. From this shape information, we can calculate the shape feedback force.

As shown in Fig. 1.34, the force on a point $P_1$ outside the target shape is calculated in the direction $-\frac{\nabla\phi_{target}}{||\nabla\phi_{target}||}$, while the force on $P_2$ inside the target shape is calculated in the direction $\frac{\nabla\phi_{liquid}}{||\nabla\phi_{liquid}||}$. The term $\phi_{target}(x, t)$ is an SDF of the target shape at time $t$, and $\phi_{liquid}(x, t)$ is an SDF of the liquid boundary.

## References

1. Beaudoin P, Paquet S, Poulin P (2001) Realistic and controllable fire simulation. Proc Graph Interface 2001:159–166
2. Chenney S, Forsyth DA (2000) Sampling plausible solutions to multi-body constraint problems. In: Proceedings of ACM SIGGRAPH 2000, pp 219–228
3. Nguyen DQ, Fedkiw R, Jensen HW (2002) Physically based modeling and animation of fire. In: Proceedings of ACM SIGGRAPH 2002. ACM transactions on graphics (TOG), July 2002, vol 21(3), pp 721–728
4. Ellingsen K, Risso F (2001) On the rise of an ellipsoidal bubble in water: oscillatory paths and liquid-induced velocity. J Fluid Mech 440:235–268
5. Fedkiw R, Stam J, Jensen HW (2001) Visual simulation of smoke. In: Proceedings of SIGGRAPH, pp 15–22
6. Franklin GF, Powell JD, Emami-Naeini A (2002) Feedback control of dynamic systems. Prentice Hall, Englewood Cliffs
7. Hong J-M, Kim C-H (2003) Animation of bubbles in liquid. Computer graphics forum (Eurographics 2003 Proceedings), vol 22(3), pp 253–262, September 2003
8. Hong J-K, Kim C-H (2005) Animating smoke with dynamic balance. In: Proceedings of computer animation and social agents 2005. Computer animation and virtual worlds, July 2005, vol 16(3–4), pp 405–414
9. Hong J-M, Kim C-H (2015) Discontinuous fluids. In: Proceedings of ACM SIGGRAPH 2005. ACM transactions on graphics, July 2005, vol 24(3), pp 915–920
10. Hong J-M, Lee H-Y, Yoon J-C, Kim C-H (2008) Bubbles alive. In: Proceedings of ACM SIGGRAPH 2008. ACM transactions on graphics, August 2008, vol 27(3), p 48,
11. Hong J-M, Shinar T, Fedkiw R (2007) Wrinkled flames and cellular patterns. In: Proceedings of ACM SIGGRAPH. ACM transactions on graphics (TOG), July 2007, vol 26(3), Article 47
12. Hong J-M, Kim C-H (2004) Controlling fluid animation with geometric potential. Comput Animat Virtual Worlds (CASA 2004) 15(3–4):147–157
13. Horvath C, Geiger W (2009) Directable, high-resolution simulation of fire on the GPU. In: Proceedings of ACM SIGGRAPHACM. ACM transactions on graphics (TOG), August 2009, vol 28(3), Article 41

14. Im J, Park H, Kim J-H, Kim C-H (2013) A particle-grid method for opaque ice formation. Comput Graph Forum 3(2):371–377
15. Kim P-R, Lee H-Y, Lee Jung, Kim C-H (2012) Anisotropic particle level-set method for multiphase fluid. J Res Pract Inf Technol
16. Kim P-R, Lee H-Y, Kim J-H, Kim C-H (2012) Controlling shapes of air bubbles in a multi-phase fluid simulation. Vis Comput 28(6–8):597–602
17. Lee H-Y, Hong J-M, Kim C-H (2009) Interchangeable SPH and level set method in multiphase fluids. Vis Comput 25(5–7):713–718
18. Lee H-Y, Hong J-M, Kim C-H (2009) Simulation of swirling bubbly water using bubble particles. Vis Comput 25(5–7):707–712
19. Liu S, An T, Gong Z, Hagiwara I (2012) Physically based simulation of solid objects' burning. In: Pan Z, Cheok AD, Chang W, Zhang M (eds) Transactions on edutainment VII. Springer, Berlin, pp 110–120
20. Losasso F, Irving G, Guendelman E, Fedkiw R (2006) Melting and burning solids into liquids and gases. IEEE Trans Vis Comput Graph 12(3):343–352
21. Melek Z, Keyser J (2005) Multi-representation interaction for physically based modeling. In: Proceedings of the 2005. ACM symposium on solid and physical modeling (SPM'05), pp 187–196
22. Shin S-H, Kim C-H (2007) Target-driven liquid animation with interfacial discontinuities. Comput Animat Virtual Worlds (CASA 2007) 18(4–5):447–453
23. Shin S-H, Kam HR, Kim C-H (2010) Hybrid simulation of miscible mixing with viscous fingering. Comput Graph Forum 29(2):675–683
24. Stam J (1999) Stable fluids. In: Proceedings of ACM SIGGRAPH 1999, pp 121–128
25. Treuille A, McNamara A, Popović Z, Stam J (2003) Keyframe control of smoke simulations, In: Proceedings of ACM SIGGRAPH 2003. ACM transactions on graphics, July 2003, vol 22(3), pp 716–723
26. Yoon J-C, Kam HR, Hong J-M, Jin Kang S, Kim C-H (2009) Procedural synthesis using vortex particle method for fluid simulation. Comput Graph Forum 28(7):1853–1859

# Chapter 2
# Water and Bubbles

## 2.1 Animation of Bubbles in Liquid

**Abstract** This section introduces a new fluid animation technique in which liquid and gas interact with each other, using the example of bubbles rising in water. In contrast to previous studies which only focused on one fluid, this system considers both the liquid and gas simultaneously. In addition to the flowing motion, the interactions between liquid and gas cause buoyancy, surface tension, deformation, and movement of the bubbles. For the natural manipulation of topological changes and the removal of the numerical diffusion, we combine the volume of fluid method and the front-tracking method developed in the field of computational fluid dynamics. Our minimum stress surface tension method enables this complementary combination. The interfaces are constructed using the marching cubes algorithm. Optical effects are rendered using vertex shader techniques.

### *2.1.1 Introduction*

Liquids are very attractive substances. As well as having beautiful optical properties, their movements are mysterious, or as an eastern saying goes, "just observing water can provide good meditation." Many studies have been done in an attempt to animate and render liquids in the computer graphics field. And thanks to recent improvements in computing powers and simulation techniques, more phenomena related to liquids have become subjects of animation.

This section introduces one more subject pertaining to liquid animation, i.e., bubbles.

Bubbles are pockets of air enclosed by liquid and exist everyplace where liquid and air coexist. As opposed to air skimming over liquid surfaces, bubbles are governed by the interactions between air and liquid. There are many factors to be considered when attempting to simulate the deformation and movement of bubbles. There are two flows to consider—i.e., those occurring inside and outside of the bubble bodies. Differences in specific gravity between the two fluids generate buoyancy forces. Surface tension forces are exerted at the interfaces between the two fluids.

C.-H. Kim et al., *Real-Time Visual Effects for Game Programming*,
Gaming Media and Social Effects, DOI 10.1007/978-981-287-487-0_2

In general, the density of liquids is much higher than that of gases. For example, water is eight hundred times heavier than air. This fact is one of the reasons for the free surface approximation in which the existence of air is generally ignored in liquid simulations. Many recent studies on liquid animation have referred to the free surface studies which have been done in computational fluid dynamics (CFD). These studies showed very natural results and some air flows were able to be inserted as surface boundary conditions—e.g., wind. However, since enclosed air is an altogether different affair, those studies are not suitable for bubbles. Besides the additional factors described above, one more consideration needs to be taken into account—i.e., two fluids have to be simulated at the same time. This problem is studied in the form of multiphase flows in CFD with the phase change problem.

Like other fluid problems, many techniques have been developed for the simulation of multiphase flows in CFD. However, since all techniques have their own characteristic approach, in order to decide which technique to use for computer animation, a set of selection criteria are needed. The criteria that we use in this section are the ease of programming, the numerical stability and the fast simulation, even at the cost of accuracy. However, since the main virtue of CFD is accuracy, no existing technique matched these characteristics exactly, therefore we combine and modify various existing techniques for our purposes.

This section presents a new fluid animation technique in which liquid and gas interact with each other, using the example of bubbles rising in water (Fig. 2.1). This system is based on the complementary combination of the volume of fluid (VOF) method and the front-tracking method which were developed in the field of CFD. The VOF method is an efficient and fast scheme for free surface simulation with the inherent capability of topological changes. It can be easily extended for the simulation of multiphase flows. However, to reduce the effect of numerical diffusion in the VOF scheme, the interfaces between the two fluids in the simulation grid need to be decided exactly, which is simple in 2D, but complicated and computationally expensive in 3D with fluid volume constraints. In contrast to the VOF method, the front-tracking method introduces no numerical diffusion. However, a bookkeeping process to maintain the front connectivity is needed to handle the topological changes and physically accurate interfacial geometry is required for the calculation of surface tension. Since our minimum stress surface tension method calculates the surface tension effects not from the interfacial geometry but directly from the simulation data, it was possible to combine these two methods.

Due to the VOF scheme being used, fast interface construction is possible with the marching cubes algorithm. Interfaces composed of polygon meshes are rendered by means of the vertex shader. Optical effects—refract, reflection, and dispersion—are included.

Section 2.1.2 presents the previous works on liquid animation and some related CFD techniques, in order to explain the limitations of previous works and the characteristics of our approaches. Section 2.1.3 introduces some new concepts for the representation of multiphase fluids and overviews our method. In Sect. 2.1.4, the simulation process is discussed in relation to the Navier–Stokes equation. Section 2.1.5 discusses the techniques introduced for visualization. In Sect. 2.1.6, we present our

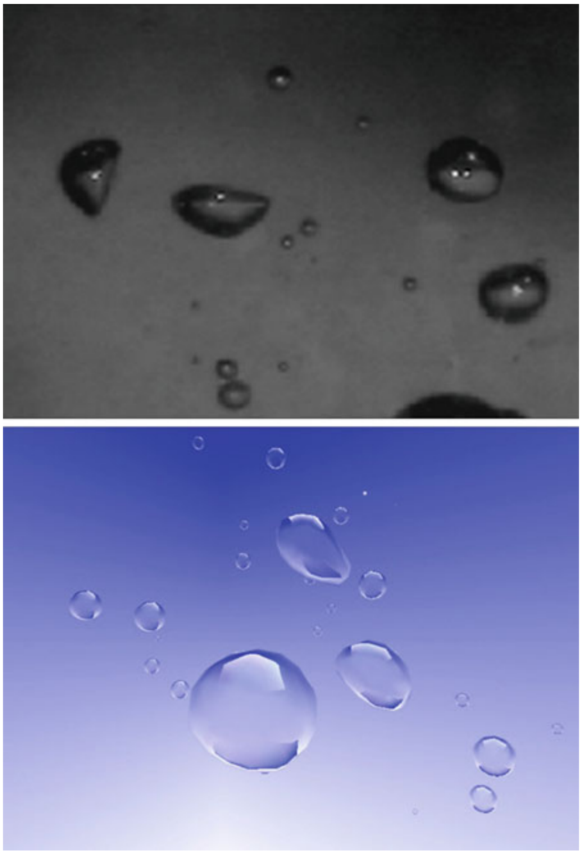


**Fig. 2.1** Rising bubbles in liquids. Photo image (*top*) and rendered image (*bottom*)

results. We conclude and discuss ideas for the future research in Sect. 2.1.7. All figures are explained in two dimensions and their extension to three dimensions should be fairly evident.

### *2.1.2 Previous Work*

The characteristics of a physically based model are strongly influenced by the physical and mathematical foundation of that model. Therefore, a combination of both models and CFD techniques is necessary in order to provide a more meaningful explanation. CFD researches include many topics—the accuracy of simulation, numerical techniques, the handling of geometry, and so on. Among them, we will concentrate on only those parts which are directly related to our purpose.

The governing equation of fluids is known as the momentum or Navier–Stokes equations. Following some initial approaches using simplified versions of the Navier–Stokes equations [37, 62], the animation of complex water was studied [19] using the marker and cell (MAC) method [32] with the full 3D Navier–Stokes equation. In the MAC method, the Navier–Stokes equation is discretized within some fixed uniform cells and fluids are expressed by Marker particles. Marker particles were able to describe both natural and detailed scenes [19, 32]. This scheme is also applied to melting animations [5]. In order to treat the smooth and detailed surfaces using these marker particles, the implicit surfaces and level-set methods were used [16]. Realistic optical properties were rendered with the physically based ray tracer. In the simulation of very complex scenes, volume loss occurred and to fix this problem the particle level-set method was introduced [14]. This approach enabled the animation of very complex scenes and velocity extrapolation gave us more control with coarse grids. Although the MAC method presents the explicit expression of liquids with marker particles, it is difficult to estimate the volume of liquid in a cell from marker particles. To represent and simulate two fluids with one grid system, we have to know the volume of each fluid in one grid. Therefore, the animation techniques based on MAC method [5, 14, 16, 19] are not suitable for our purposes.

For fast animation of liquids [43], the volume of fluid (VOF) method [29] was used, in which the liquid surfaces were constructed using the marching cubes algorithm [44] and rendered with polygonal techniques. The VOF method handles topological changes naturally with the marching cubes algorithm, and basically uses only one scalar value—the volume of fluid—for one cell, through which we can know the total volume of fluid in the simulation space. The VOF method assumes that the liquid in a cell is gathered in one corner. From the volume value of one cell and its adjacent cells, the exact position of the liquids needs to be estimated to eliminate the effects of numerical diffusion. This is a problem involving the intersection of a line and a square in 2D cases [67]. In 3D, these become a plane and a cube [23], and some numerical iteration is required in order to find a solution. Therefore, it is inefficient to eliminate numerical diffusion within VOF scheme for computer animation purposes.

Some spherical objects related to fluids such as liquid foams [42], water droplets [17, 89], and soap bubbles [10] have been studied. However for air bubbles enclosed in liquids, the simulation of environmental liquids is unavoidable. This phenomenon can be explained as a kind of multiphase flows. The front-tracking method [80] involves the simulation of multiphase flows without numerical diffusion. The original front-tracking method explicitly discretized the free surface using particles and maintains a connectivity list between these particles [85]. This connectivity list is difficult to maintain when parts of the free surface break apart or merge together as is often seen in complex flows of water and other liquids. To avoid this difficulty, the point-set method was introduced [79]. Although this approach unchains the front-tracking method from its dependence on logical interface point connectivity, the point regeneration algorithm is complex and computationally expensive. The level contour reconstruction method [72] is similar to the combination of the VOF method and the marching cubes algorithm used in [43], which possesses the inherent capability of being able to deal with topological changes. The feedback from interfaces to simulation grids still removes numerical diffusion. However, for the calculation of surface tension forces and for numerical accuracy, the physically exact interfaces are needed.

Our minimum stress surface tension method is implemented independently from the details of interfacial geometry with the sufficient convergence for computer animation. Moreover, the feedback provided by the front-tracking method removes the numerical diffusion and guarantees mass conservation with the benefits coming from the VOF scheme.

In solving the Navier–Stokes equations, the initial approach was based on explicit finite difference scheme [19, 32]. For computer graphics, the stable fluid scheme [77] based on implicit approaches such as semi-Lagrangian method and implicit diffusion was proposed for large time-steps and numerical stability. Subsequently, efficient pressure iteration was introduced [16]. Our method utilizes these techniques instead of the standard CFD techniques (see Sect. 2.1.4.1).

## *2.1.3 Overview*

### 2.1.3.1 Representation of Multiphase Fluids

In contrast to previous works which have dealt with the free surface problem, we consider two fluids simultaneously. To represent two fluids with one fixed grid system, we define an *indicator function* $I = I(\mathbf{x}, t)$. $I(\mathbf{x}, t)$ takes the value 1 in one fluid and 0 in the other fluid. A *material field* is defined by the values of $I$ in each cell and Fig. 2.2a shows an example. As you can see in Fig. 2.2a, there are some transition zones between 0 and 1, which are at the interfaces between the two fluids.

We can define the interfaces between two fluids with some isosurface construction algorithm. As is shown in Fig. 2.2b, we used the marching cubes algorithm with a threshold value of 0.5. Details are discussed in Sect. 2.1.5.1.

### 2.1.3.2 System Outline

In this section, we divide our animation process into three steps in order to provide a conceptual explanation. They are the velocity field update, material field update, and visualization processes. A more detailed explanation of the general processes involved in fluid animation can be found in the literature [16, 19, 29, 32, 77].

**Velocity Field Update**

In this step, the velocity field is updated from the initial or previous velocity field by solving the Navier–Stokes equation. Material field data are needed for the calculation of the gravity forces and surface tension forces. The details are discussed in Sects. 2.1.4.1 and 2.1.4.2.

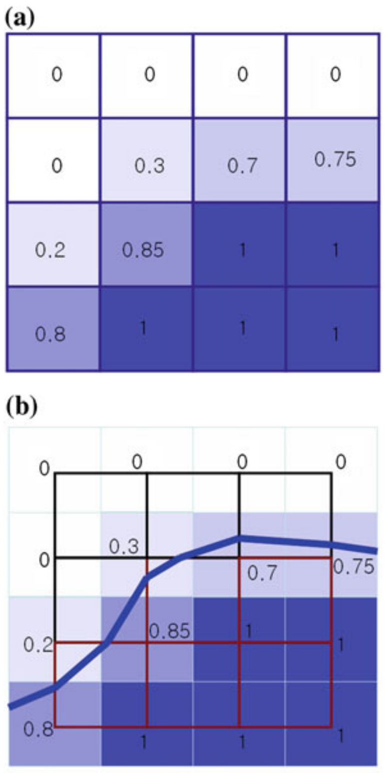


**Fig. 2.2** Material field **a** and interfaces **b** constructed from this material field

## Material Field Update

After updating the velocity field, we should update the material field by evolving the indicator function to reflect the movement of the fluids caused by the velocity field. This involves the flow of materials. The details are discussed in Sect. 2.1.4.3.

## Visualization

From the updated material field, we construct rendering primitives and render them. As well as the polygonal meshes representing the interfaces, some particles are included for the sake of providing more detailed scenes. The details are discussed in Sect. 2.1.5.

### *2.1.4 Simulation of Multiphase Flows*

#### 2.1.4.1 Navier–Stokes Equation

The momentum equation, the so-called Navier–Stokes equation for multiphase flows [80] is

$$\frac{\partial(\rho \mathbf{u})}{\partial t} = \mu \nabla \cdot (\nabla \mathbf{u}) - \nabla \cdot (\rho \mathbf{u}\mathbf{u}) - \nabla P + \rho \mathbf{g} + \int_{\Gamma(t)} \rho \kappa \mathbf{n} \delta(\mathbf{x} - \mathbf{x}_f) ds \tag{2.1}$$

where **u** is the velocity, $\rho$ is the density, $\mu$ is viscosity, $P$ is the pressure, and **g** is the gravity. The surface integral is a surface tension term. The physical definition and finite difference scheme of surface tension are described in Sect. 2.1.4.2.

Conservation of mass written for the entire flow field is

$$\nabla \cdot (\rho \mathbf{u}) = -\frac{\partial \rho}{\partial t}. \tag{2.2}$$

The discrete forms for the finite difference method of Eqs. (2.1) and (2.2) can be written as

$$\frac{\mathbf{w}^{n+1} - \mathbf{w}^n}{\Delta t} = \mathbf{A}^n + \mathbf{F}^{n+1} - \nabla_h P \tag{2.3}$$

$$\nabla_h \cdot \mathbf{w}^{n+1} = M^{n+1}. \tag{2.4}$$

Here $\mathbf{w} = \rho \mathbf{u}$ is the fluid mass flux. The advection, diffusion, and external forces terms in Eq. (2.1) are lumped into **A**, the right side of Eq. (2.2) is denoted by $M$, and the surface integral in Eq. (2.1) is denoted by **F**.

Following the spirit of Chorin's projection method, we split the momentum equation into

$$\frac{\tilde{\mathbf{w}} - \mathbf{w}^n}{\Delta t} = \mathbf{A}^n + \mathbf{F}^{n+1} \tag{2.5}$$

and

$$\frac{\mathbf{w}^{n+1} - \tilde{\mathbf{w}}}{\Delta t} = -\nabla_h P \tag{2.6}$$

where we introduce the variable $\tilde{\mathbf{w}}$, which is the new fluid mass flux if the effect of pressure is ignored. The first step is to find this mass flux using Eq. (2.5)

$$\tilde{\mathbf{w}} = \mathbf{w}^n + \Delta t (\mathbf{A}^n + \mathbf{F}^{n+1}). \tag{2.7}$$

The pressure is found by taking the divergence of Eq. (2.6) and using Eq. (2.4). This leads to a Poisson equation for $P$

$$\nabla^2 P = \frac{\nabla \cdot \tilde{\mathbf{w}} - M^{n+1}}{\Delta t}, \tag{2.8}$$

which can be solved using a standard Poisson solver. The updated mass flux is found from Eq. (2.6)

$$\mathbf{w}^{n+1} = \tilde{\mathbf{w}} - \Delta t \nabla P. \tag{2.9}$$

The updated velocity is $\mathbf{u}^{n+1} = \mathbf{w}^{n+1}/\rho^{n+1}$.

In this section, the phase change problem coming from heat transfer is not included. In isothermal cases, $\partial\rho/\partial t = 0$, which reduces Eq. (2.2) to

$$\nabla \cdot \mathbf{u} = 0 \tag{2.10}$$

and Eq. (2.5) to

$$\nabla_h \cdot \mathbf{w}^{n+1} = 0 \tag{2.11}$$

with $M = 0$. If we consider Eq. (2.10) as a *volume* conserving condition, the whole process of finding a solution becomes similar to one involving free surface conditions.

Since there is no *vacant* space in our simulation, unlike free surface simulations, all cells should be simulated. Therefore, the free surface conditioning such as classifying cells and modifying the velocities of surface cells [19, 32], is not needed.

In the first projection step of Eqs. (2.5) and (2.7), the stable fluids scheme [77] is incorporated, in which the advection is calculated using the semi-Lagrangian method and the diffusion is calculated with implicit method. The second projection step of Eqs. (2.6), (2.8), and (2.9) is solved in the form of a mass conservation process [16], in which we use a standard conjugate gradient solver as a Poisson solver. All equations are discretized on the standard staggered MAC grids [32].

#### 2.1.4.2 Surface Tension

Surface tension is the apparent interfacial tensile stress (force per unit length of interface) that acts whenever a liquid has a density interface, such as when the liquid is in contacts with a gas, vapor, second liquid, or solid. The mathematical definition of surface tension **F** in Eq. (2.1) is

$$\mathbf{F} = \int_{\Gamma(t)} \sigma\kappa\mathbf{n}\delta(\mathbf{x} - \mathbf{x}_f)ds \tag{2.12}$$

where $\sigma$ is the surface tension coefficient, $\kappa$ is twice the mean interface curvature, $\mathbf{n}$ is the unit normal to the interface, $\mathbf{x}_f = \mathbf{x}(x, t)$ represents the parameterization of the interface $\Gamma(t)$, and $\delta(\mathbf{x} - \mathbf{x}_f)$ is a three-dimensional delta function that is nonzero only where $\mathbf{x} = \mathbf{x}_f$. Figure 2.3 visually explains the surface tension forces defined in Eq. (2.12). The black lines refer to a portion of the surfaces. Light blue arrows represent the tension forces being exerting at the interfaces. The red arrow,

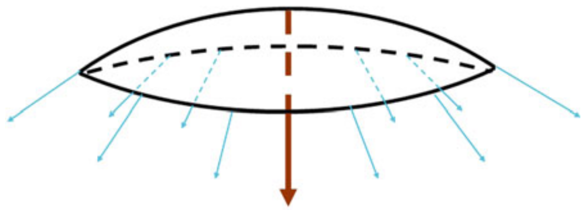


**Fig. 2.3** Surface tension forces

representing the sum of these tension forces, represents the total force being exerting on this portion of the surface.

In front-tracking scheme, these forces are calculated using the polygon meshes representing interfaces and distributed to the simulation grids as body forces [72, 80, 85]. Since, the interfaces are constructed from material field discretized on the simulation grids, it is inefficient and unreliable to distribute surface tension forces to the simulation grids estimated from those interfaces. To overcome this inefficiency and remove dependency on interfacial geometry in surface tension calculation as discussed in Sect. 2.1.2, our *minimum stress surface tension method* calculates surface tension forces directly from the material field. The physical meaning of Eq. (2.12) is that the surface tension is a tendency to minimize the total stress of interfacial surfaces. So, we define the stress of material field and let surface tension forces to minimize this stress.

To define the stress of a position on the material field, $S(\mathbf{x})$, first, we define an imaginary stress-zero isosurface whose value is $I^0(\mathbf{x})$, on which $S(\mathbf{x}) = 0$. Then, $S(\mathbf{x})$ can be defined by the deviation of $I(\mathbf{x})$ from $I^0(\mathbf{x})$. In Cartesian coordinate system, $S(\mathbf{x})$ is defined as

$$S(\mathbf{x}) = c \sum_l (I_l^0(\mathbf{x}) - I(\mathbf{x})) \cdot \mathbf{n}_l, \tag{2.13}$$

where $c$ is a control coefficient, $l$ is $\{x, y\}$ in 2D and $\{x, y, z\}$ in 3D, and $\mathbf{n}_l$ is the unit normal of $l$ direction. In our implementation, we define $I_l^0(\mathbf{x})$ as

$$I_l^0(\mathbf{x}) = \sum_l \mathbf{n}_l \cdot \left\{ \sum_{p=m-l} (I(\mathbf{x}_{p+}) + I(\mathbf{x}_{p-})) \right\} / a, \tag{2.14}$$

where $m = \{x, y\}$ and $a = 2$ in 2D, and $m = \{x, y, z\}$ and $a = 4$ in 3D. Finally, we can define the material field version of Eq. (2.12) as

$$F(\mathbf{x}) = -\sum_l (S(\mathbf{x}) \cdot \mathbf{n}_l) \nabla_l I(\mathbf{x}), \tag{2.15}$$

where $\nabla_l I(\mathbf{x}) = (\nabla I(\mathbf{x}) \cdot \mathbf{n}_l)\mathbf{n}_l$.

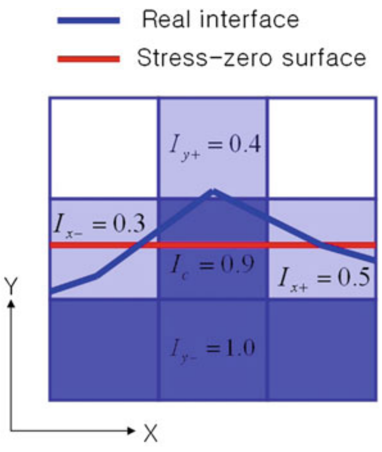


**Fig. 2.4** The minimum stress surface tension method

Figure 2.4 shows an example of our method. To find the $y$ portion of $F(center)$, first, we assume an imaginary stress-zero isosurface (red line). In this case, the value of this isosurface, $I_j^0(center)$, is $(I_{x-} + I_{x+})/2 = (0.3 + 0.5)/2 = 0.4$ using Eq. (2.14). The material value of the center cell 0.9 is bigger than 0.4 and this implies that the interfaces constructed by marching cubes algorithm (blue line) would not be on the stress-zero surface. Now, we can calculate the direction and magnitude of the $y$ portion of $F(center)$ using Eq. (2.14). The $x$ portion of $F(center)$ can be calculated in the same way. The extension to 3D cases are fairly evident.

The calculated surface tension forces are inserted to Eq. (2.3) as body forces for Navier–Stokes simulation. Figure 2.5 shows an example. The small lines—the direction is heading from black to white—are normalized surface tension forces inserted as body forces. Red arrows are introduced as visually understandable explanations of the surface tension forces.

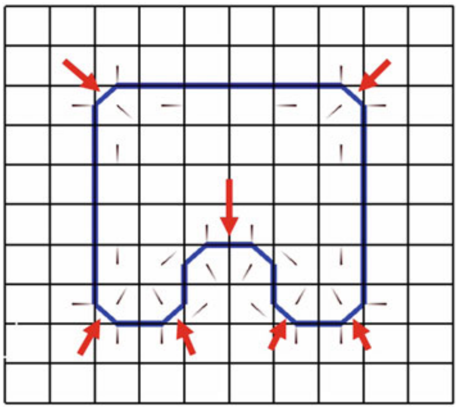


**Fig. 2.5** The surface tension forces inserted as body forces

### 2.1.4.3 Update of Material Field

The last step in the simulation is the update of the material field. As described in Sect. 2.1.3.1, our system describes the positioning of fluids by means of an indicator function. After getting the velocity field as in Sect. 2.1.4.1, we should evolve the indicator function to reflect the movement of the fluids caused by the velocity field.

The time dependence of indicator function $I$ on a velocity field is governed by the equation [29],

$$\frac{\partial I}{\partial t} + u\frac{\partial I}{\partial x} + v\frac{\partial I}{\partial y} = 0. \tag{2.16}$$

In the VOF representation, Eq. (2.16) can be solved by transporting the volume of fluid from one cell to another cell [29]. Through some experiments to get the smoothness of animation in the combination of marching cubes algorithm, we can decide our discretized form of Eq. (2.16). In the case of Fig. 2.6, the change of center cell with our discretization is

$$\frac{\Delta I_C}{\Delta t} = -I_C \cdot v_N - I_C \cdot v_W - I_C \cdot v_E + I_S \cdot v_S. \tag{2.17}$$

While Eq. (2.17) is easy to implement and shows very smooth animation in combination of marching cubes algorithm (will be discussed in Sect. 2.1.5.1), it has the inherent property of numerical diffusion. In ideal simulations, material values of the cells far from interfaces must be 0 or 1. Numerical diffusion occurs when this condition is not fulfilled as shown in Fig. 2.7b. Numerical diffusion prevents the robust and correct liquid simulation. As discussed in Sect. 2.1.2, it is difficult to meet this condition within VOF scheme. However, with the aid of front-feedbacks used in front-tracking method, numerical diffusion can be corrected. In contrast to the MAC representation, we can know the total volume or mass of fluids explicitly with the VOF representation. The total mass of a volume at time $t$, $M^t$ is

$$M^t = \int_V \rho I^t(\mathbf{x})dV. \tag{2.18}$$

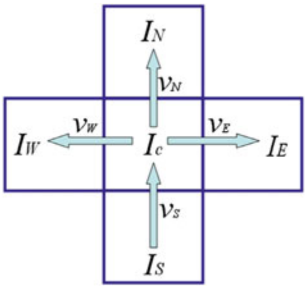


**Fig. 2.6** An example of indicator function update

Therefore, what we have to do to correct the mass loss is just to modify $I^{t+\Delta t}$ to meet $M^{t+\Delta t} = M^t$ by changing some material values.

Since we use marching cubes algorithm for constructing interfaces, it is easy to find the location of interfaces or fronts, and move them by scaling adjacent material values. In this modifying step, we fix the value of the indicator function to 0 or 1 *except for* the cells near interfaces or fronts in order to remove numerical diffusion and maintain the location of the interfaces at the same time. Subsequently, we pull out or push back the interfaces to maintain the total mass by scaling the material values near the interfaces. The scaling factor is decided as

$$SF = \frac{M^t - M_{fixed}}{M^{t+\Delta t} - M_{fixed}}. \tag{2.19}$$

Figure 2.7 is a rising bubble example. Figure 2.7a represents initial configurations. Unlike Fig. 2.7b, with our correcting step, Fig. 2.7c shows the numerically perfect mass conservation and no numerical diffusion.

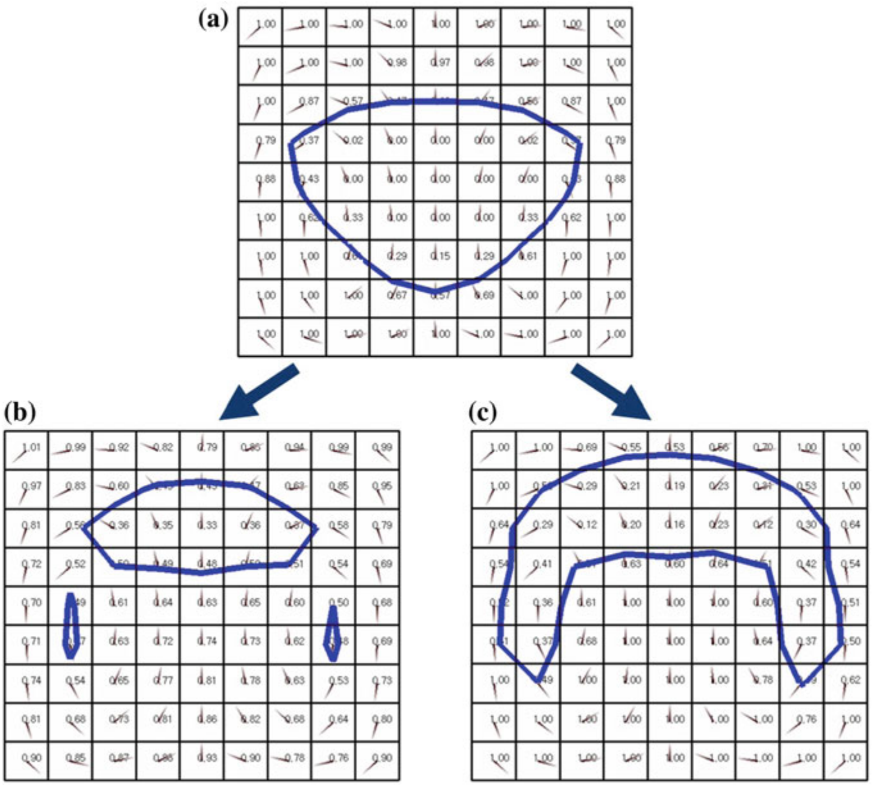


**Fig. 2.7** Restricting the numerical diffusion

## 2.1.5 Visualization

### 2.1.5.1 Interface Construction

In the front-tracking method, the interfaces between two materials—in our case, water and air—are composed of polygon meshes for the easy calculation of the surface tension. Bookkeeping method [85] in the case of polygon meshes is difficult because of the topological changes of the fluids. Recently, the isosurface construction method for front tracking [72] was used to solve this problem. This approach handles topology changes in natural way, which is appropriate for the purpose of animation. Since our use of surface tension steps using the minimum stress tendencies discussed in Sect. 2.1.4.2 reduces the need for a detailed expression of the interfaces, we were able to use the marching cubes algorithm for isosurface construction. In addition to its compatibility with our staggered grid system, the lookup table style of the marching cubes algorithm supports fast animation [43, 44]. In this case, the material field plays a role of the intensity field needed in marching cubes algorithm (see Fig. 2.2b). The vertex normal was calculated by interpolating the gradient of the material field.

With the marching cubes algorithm, there are many possibilities of discontinuities arising in the animation. Furthermore, since our indicator function is defined by a discontinuous delta function, the continuity of the animation could be damaged. However, though the approach we used to update the indicator function introduces the numerical diffusion without front-tracking steps, it shows smooth animation with the marching cubes algorithm. This is one more benefit which arises from the use of our indicator function update method (discussed in Sect. 2.1.4.3).

### 2.1.5.2 Particle System

Small bubbles are spherical due to the domination of the surface tension forces. We use the particle system for small bubbles with no deformation. While the computational cost associated with the particles is small, they provide for a lively animation. The velocity of a particle is determined by the linear interpolation of six facial velocities of the cell containing that particle. For natural behavior, buoyant forces are added as body forces, which is similar to the approach taken in the MAC method. Sizes and initial positions are randomly decided. In some cases, the use of the particle system alone could provide for a good animation of bubbles.

### 2.1.5.3 Rendering

Unlike other approaches using implicit surfaces [14, 16], in our system, the interfaces are composed of polygon meshes, which enables fast rendering supported by hardware acceleration [43]. Some optical effects were able to be implemented by means of a vertex shader. Reflection, refraction, and dispersion effects were applied using conventional vertex shader codes [93], which results in visually pleasing scenes.

## *2.1.6 Results and Discussion*

Following are the results of our animation system implemented using the OpenGL APIs and the NVIDIA vertex shader codes. This system was tested on Windows PC system and the test machine is a PC with 512 MB of RAM and an Intel Pentium IV processor running at 1.4 GHz. It uses an NVIDIA GeForce 2 MX graphics card with 64 MB of video RAM.

Before simulation process, the properties and the initial conditions of fluids should be given. They are the initial velocity field, viscosity and density of each fluid, gravity, surface tension coefficient between two fluids, and initial configuration.

### Surface Tension

Figures 2.8 and 2.9 are examples provided to show the convergence of our minimum stress surface tension method. The influence of gravity was omitted for clarity. Even though our algorithm calculates the surface tension forces using material field independently from the details of interface geometry, any arbitrary shapes converged to the spherical ones with no volume loss. In spherical shapes, all surface tension forces are canceled by each other. Some oscillatory phenomena were also included, which are similar to those observed in nature. $13 \times 13 \times 13$ simulation grids were used and frame rate was 7.6 fps.

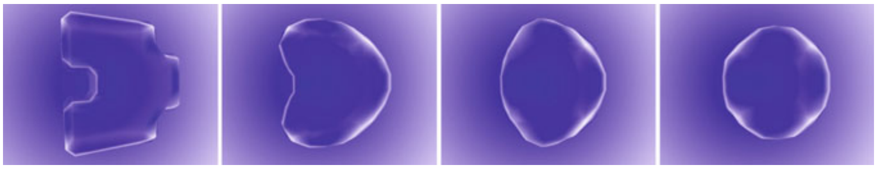


**Fig. 2.8** Surface tension convergency

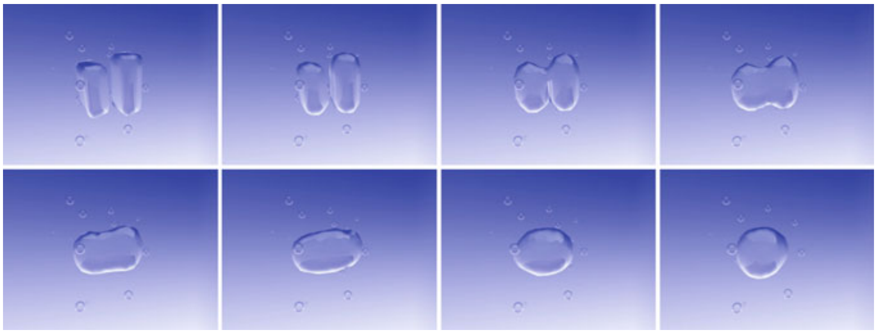


**Fig. 2.9** Deformation and merging caused by surface tension

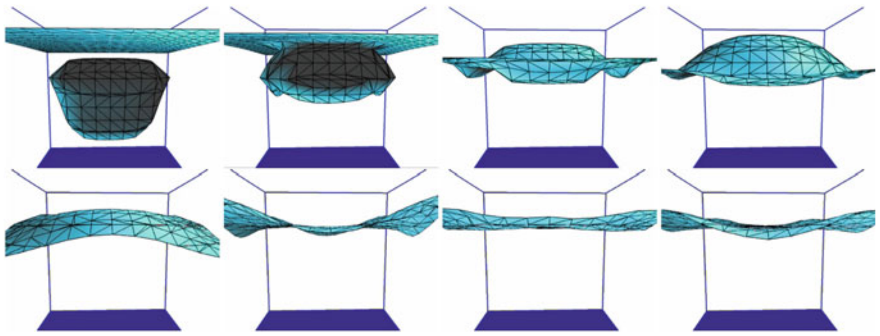


**Fig. 2.10** A bubble merges into a free surface

## A Bubble Near a Free Surface

When rising bubbles arrive at free surfaces, they are absorbed by the atmospheric air leaving violent impacts on the free surfaces. Figure 2.10 shows this phenomenon. In spite of the severe shape changes, this simulation shows natural animation. The merging of bubble meshes and surface meshes, is done naturally. $15 \times 15 \times 15$ simulation grids were used and frame rate was 5.2 fps.

Although this result proves that it is possible to deal with the free surface condition within our simulation frameworks, there occurs a problem of volume gain of atmospheric air—i.e., the free surfaces lower before meeting the bubble. The reason of this problem is that we conserve *whole air volume or whole liquid volume* in our current implementation as discussed in Sect. 2.1.4.3. To fix this problem, we have to check each separated air volumes—i.e., $M^t = \sum_i M_i^t$—and conserve each of them. With our front-feedbacks, we can easily find the separated volumes and conserve them. This problem with free surface conditions can be handled as a future work.

## Rising Bubbles

Figure 2.11 shows a decorated version of the rising bubbles problem. The bubble rises due to their buoyancy. After merging with other small bubble, it rises with certain fixed shape. The shape constitutes a kind of balance point between buoyancy forces and the surface tension forces. The small bubbles are animated using particle system with no deformation as discussed in Sect. 2.1.5.2. Visually pleasing optical effects were included using vertex shader techniques. $9 \times 9 \times 25$ simulation grids were used and frame rate was 1.2 fps.

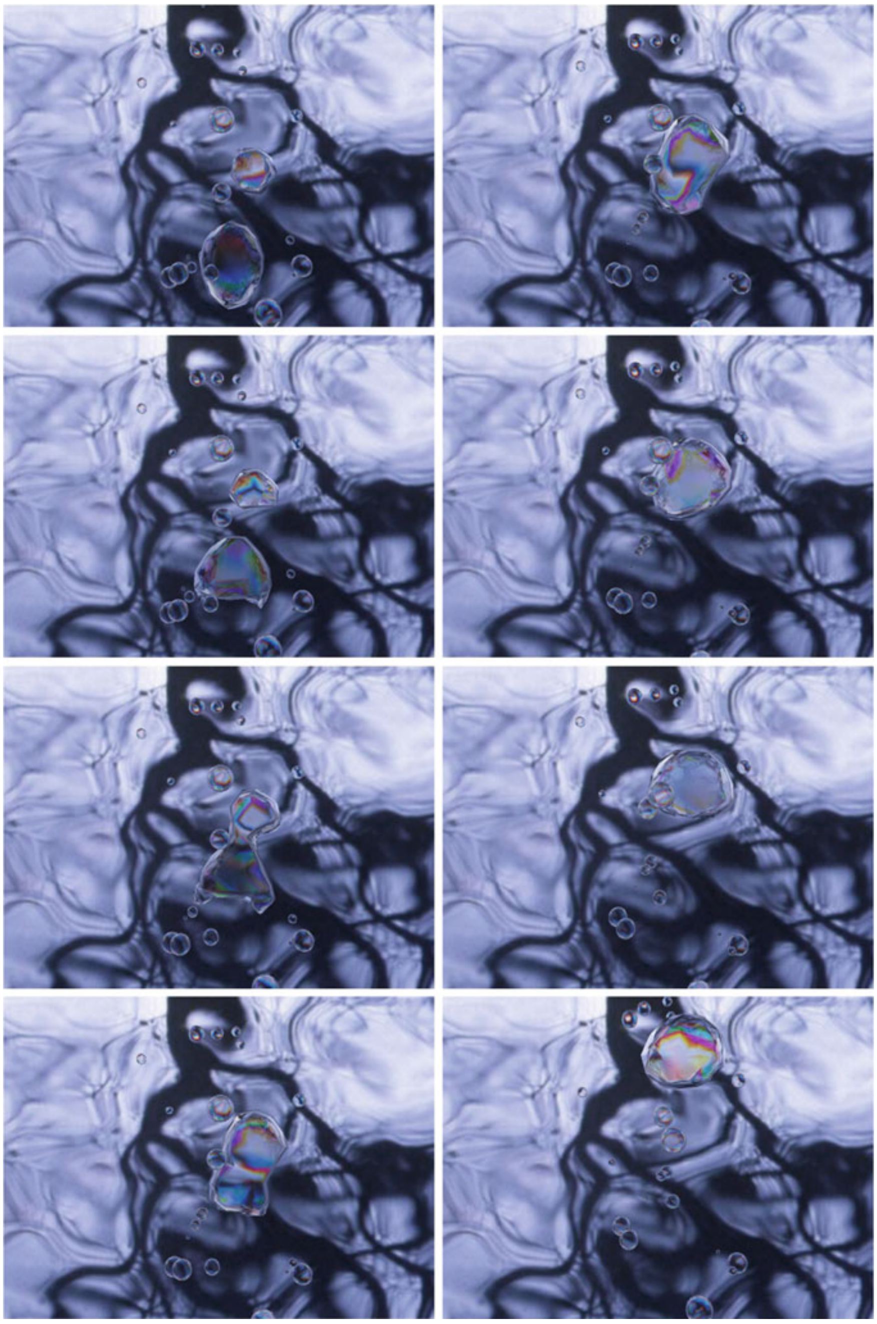


**Fig. 2.11** Rising bubbles in a liquid (from *left-up* to *right-down*)

### *2.1.7 Conclusion and Future Work*

In this section, we studied a new fluid animation technique in which liquid and gas interact with each other. This algorithm is based on a complementary combination of various CFD techniques which are selected and modified for computer animation purposes with the aid of our minimum stress surface tension method. The finite difference scheme for the simulation of the multiphase Navier–Stokes equation was introduced and we used appropriate visualization techniques using the marching cubes algorithm and the hardware acceleration.

Since this algorithm can handle topological changes and surface tension fairly easily and with no volume loss or numerical diffusion, we can extend it to the physically based simulation of water droplet model interacting with static environments or other droplets.

## 2.2 Discontinuous Fluids

**Abstract** At interfaces between different fluids, properties such as density, viscosity, and molecular cohesion are discontinuous. To animate small-scale details of incompressible viscous multiphase fluids realistically, this section focuses on the discontinuities in the state variables that express these properties. Surface tension of both free and bubble surfaces is modeled using the jump condition in the pressure field; and discontinuities in the velocity gradient field, driven by viscosity differences, are also considered. To obtain derivatives of the pressure and velocity fields with subgrid accuracy, they are extrapolated across interfaces using continuous variables based on physical properties. The numerical methods presented in this section are easy to implement and do not impact the performance of existing solvers. Small-scale fluid motions, such as capillary instability, breakup of liquid sheets, and bubbly water can all be successfully animated.

### *2.2.1 Introduction*

Close-up scenes of splashing water have mysterious attractions. To emphasize the luxurious image of their products in an advertisement, to depict the tense atmosphere before the sword fight in a movie, or just to show off the performance of their brand-new digital camera, many people are trying to catch a moment of this beauty. Recently, the computer graphics community has made great advances in fluid animation, and we are taking one more step toward small-scale realism.

All fluids in our environment are essentially multiphase. This means that property variables are discontinuous at the interfaces between different phases. The small-scale motion of fluids is strongly influenced by these discontinuities. For example,

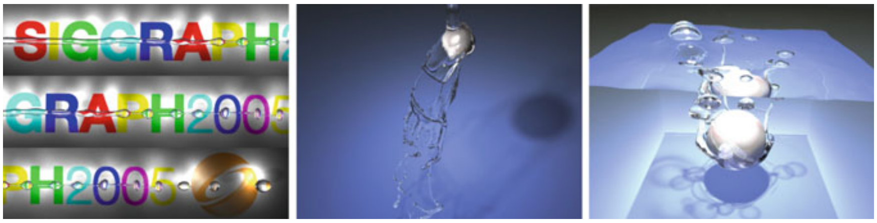


**Fig. 2.12** Capillary instability of a liquid jet; liquid pouring on to a sphere; and bubbly water

the discontinuity of molecular cohesion induces surface tension, which is the phenomenon that smooths out liquid surfaces. It is an interesting fact that surface tension is also responsible for the capillary instability that can break up fluid into small droplets or bubbles. Similarly, discontinuity of density is the reason for Rayleigh–Taylor instability, as well as for buoyancy; and the discontinuity of viscosity influences the shape of air bubbles in water.

In this chapter, we extend previous fluid simulation techniques based on Eulerian grids [14, 16, 47, 77] to incompressible viscous multiphase fluids, focusing on surface tension effects and viscosity changes at both free surfaces and bubble surfaces, as well as on buoyancy. This requires a robust treatment of discontinuities in the pressure and velocity gradient fields. To differentiate them accurately across interfaces, we deploy the ghost fluid method (GFM) of Fedkiw et al. [15], which was developed in computational physics. In combination with the implicit representation of level-set surfaces [63], GFM can treat discontinuities accurately. Surface tension can be modeled using the jump condition in the pressure field, and discontinuities in the velocity gradient field, driven by viscosity differences, can also be considered, permitting subgrid accuracy. Since the numerical methods derived in this section are formulated as simple modifications of previous techniques, they are very easy to implement and do not influence the performance of existing solvers. Results show interesting aspects of small-scale fluid motions such as capillary instability, breakup of liquid sheets, and bubbly water (Fig. 2.12).

### *2.2.2 Previous Work*

The first simulation of the fully three-dimensional Navier–Stokes equation for animating liquids [19] was based on the marker and cell method [32] from computational fluid dynamics. Foster and Metaxas [19] used explicit finite differencing for advection and viscosity, successive over relaxation (SOR) for pressure projection and incompressibility, and massless marker particles for surface representation. Explicit integration methods were subsequently replaced by implicit methods [77] such as semi-Lagrangian advection and implicit viscosity integration, which greatly increased the numerical stability of fluid simulators both for liquid and gas, and made them easier to implement. Later [16], SOR was replaced by more efficient linear solvers, such as the conjugate gradient method, and the particle-based surface representation was reinforced by implicit level-set surfaces, which greatly improved the smoothness of liquid surfaces and their robustness under topological changes. This hybrid surface representation was enhanced by the particle level-set method [14], which has a much improved mass conservation.

While free surface animation techniques, in which the environmental and enclosed air are ignored, have been extensively developed for liquid animation, the dynamics of multiphase fluids have received less attention. Takahashi et al. [81] reported a multiphase fluid simulator that handles liquid and gas simultaneously, but gave no attention to the dynamic characteristics of liquid–gas interactions. On the other hand, [25] mainly focused on buoyancy and surface tension in their animation of bubbles in liquids. Although their results showed the interesting characteristics of multiphase fluids containing bubbles, it is not clear that their heuristic implementation of surface tension is generally useful. Furthermore, the effect of viscosity differences was not considered, in spite of the large influence of viscosity on bubble shapes. Carlson et al. and Rasmussen et al. [5, 66] used the variational viscosity method to handle thermal changes of viscosity, but they did not consider the large changes that occur across interfaces. More attention to the small-scale features of multiphase fluid was paid by [76]. They demonstrated the characteristics of enclosed air and modeled surface tension using the continuum surface force model [2] that has been generally used in computational fluid dynamics; but Song et al. commented that surface tension effects were not visually significant in their work. We believe that is because they replaced small-scale features by undeformable particles instead of simulating them directly. Similarly, [22] used escaped particles within a particle level-set method to represent air bubbles. A breakthrough was made by [47], who animated the crown phenomenon exhibited by milk by accurately simulating the surface tension of free surfaces. They were able to use a sufficiently large grid, for example $512^3$, so that they did not lose small-scale details. However, neither surface tension in bubble surfaces nor viscosity was considered. Their fluid simulator was based on an octree data structure, which is also the basis of the work of this section.

In computational physics, extensive studies have been undertaken to simulate multiphase fluids. For a good survey, [9, 78, 80] and their references are recommended. Although it is difficult to evaluate the techniques reported in those papers from the viewpoint of computer graphics, the work of [36] is found most compatible with the liquid simulation techniques widely used in computer graphics, such as the particle level-set method [14]. The methods used by [36] are motivated by the ghost fluid method [15], and developed using the variable coefficient Poisson equation [45]. In computer graphics, GFM has been used for physically based modeling of fire [61].

Because of the different requirements of computational physics and computer graphics, we separate the pressure jump condition from the density and velocity gradient discontinuities. This is much easier for computer graphics programmers to understand and implement. Although we are approximating the accurate method of [36], our techniques are powerful and robust in animating multiphase fluids with relatively coarse grids.

## *2.2.3 Overview of Navier–Stokes Simulation*

The Navier–Stokes equation for an incompressible viscous fluid is

$$\mathbf{u}_t = -(\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla \cdot (\nu \nabla \mathbf{u}) - \frac{\nabla p}{\rho} + \mathbf{f} \tag{2.20}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{2.21}$$

where $\mathbf{u} = \{u, v, w\}$ is the velocity, $\rho$ is the density, and $\nu$ is the (kinematic) viscosity, which is the ratio between the absolute viscosity $\mu$ and $\rho$. The term $\mathbf{f}$ can be used to add external forces such as gravity, buoyancy [20], surface tension forces [25, 76], and control forces [18, 26, 54, 83].

The numerical simulation of Eqs. (2.20) and (2.21) advances by updating the value of $\mathbf{u}$ at the $n$th time-step, $\mathbf{u}^n$ to $\mathbf{u}^{n+1}$ during a finite time-step $\Delta t$. Following Chorin's projection method [4], we discretize Eq. (2.20) by splitting it into two equations with intermediate status $\mathbf{u}^*$:

$$\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} = -(\mathbf{u}^n \cdot \nabla)\mathbf{u}^n + \nabla \cdot (\nu \nabla \mathbf{u}^n) + \mathbf{f} \tag{2.22}$$

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^*}{\Delta t} = -\frac{\nabla p}{\rho}. \tag{2.23}$$

To obtain $\mathbf{u}^*$ from $\mathbf{u}^n$, we compute the advection term, $-(\mathbf{u}^n \cdot \nabla)\mathbf{u}^n$, using a semi-Lagrangian method [77], and the viscosity term, $\nabla \cdot (\nu \nabla \mathbf{u}^n)$, using explicit finite differencing or an implicit variable viscosity formulation [5].

The final step is determining $\mathbf{u}^{n+1}$ from $\mathbf{u}^*$. We can write the divergence of Eq. (2.23) as a form of Poisson's equation,[1]

$$\nabla^2 p = \frac{\rho}{\Delta t} \nabla \cdot \mathbf{u}^*, \tag{2.24}$$

since Eq. (2.21) tells us that $\nabla \cdot \mathbf{u}^{n+1}$ should be zero. Once the pressure profile is determined by solving Eq. (2.24), we can get the final velocity profile:

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \frac{\Delta t}{\rho} \nabla p. \tag{2.25}$$

Buoyancy can be implemented in a simple way by exerting buoyant forces with the spatial constant $\rho$ [20], or more accurately by allowing for the fact that $\rho = \rho(x)$ in the solution of Eq. (2.23) [76]. Since the modeling of buoyancy is not new to the computer graphics community, we will focus on surface tension and viscosity in

[1] If $\rho$ is spatially varying, Eq. (2.24) is only an approximation, but one that is necessary to decouple the pressure jump condition and the density discontinuity.

Sect. 2.2.4. Instead of exerting continuous surface tension forces [76] using **f**, we use the discontinuity of $p$ in solving Eq. (2.24) to model surface tension effects and thus obtain subgrid accuracy. Viscosity change across the interfaces is also taken into account in solving Eq. (2.22).

One more topic to be considered is the interface tracking method. We use a signed distance function, $\phi$, to represent implicitly the interfaces of two immiscible fluids, at least one of which is a liquid. The advection of $\phi$, driven by $\mathbf{u}$, can be described by the level-set equation:

$$\phi_t + \mathbf{u} \cdot \nabla \phi = 0 \tag{2.26}$$

To solve Eq. (2.26) numerically, we use the semi-Lagrangian particle level-set method [13].

### 2.2.4 Discontinuous Interfacial Dynamics

We will now describe numerical methods to implement surface tension and viscosity changes at interfaces. Discontinuous variables are extrapolated across interfaces using continuous variables based on their physical properties. This is applied to both the pressure and velocity field, to obtain accurate derivatives at interfaces. Examples are provided in a single dimension for clarity. Extension to two or three dimensions is straightforward.

#### 2.2.4.1 Surface Tension

We will assume the existence of a pressure profile near the interface $\Gamma$ between two different, immiscible fluids. Using the signed distance function $\phi$ to represent the geometry of the situation, $\Gamma$ exists where $\phi = 0$. As shown in Fig. 2.13, surface tension causes a jump $J$ in pressure across $\Gamma$, and the magnitude of $J$ is $\sigma\kappa_\Gamma$.

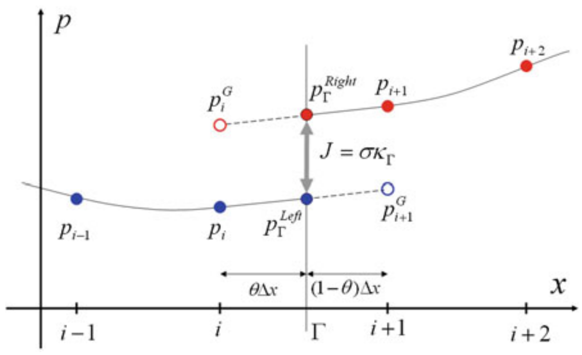


**Fig. 2.13** The discontinuous pressure field near an interface $\Gamma$

At $\Gamma$, $\sigma$ is the surface tension bcoefficient and $\kappa_\Gamma$ is the curvature, which can be determined by interpolating between the curvatures $\kappa = \nabla \cdot (\nabla\phi/|\nabla\phi|)$ of near nodes, using $\theta\kappa_i + (1-\theta)\kappa_{i+1}$, since $\kappa$ is continuous across $\Gamma$ due to the implicit surface representation. When $\kappa_\Gamma$ is positive, $J$ is positive and vice versa.

The existence of the pressure jump induces a discontinuity in the pressure at $\Gamma$, i.e., in Fig. 2.13 the pressure of the left side of the interface, $p_\Gamma^{Left}$, and the pressure of the right side, $p_\Gamma^{Right}$, are different. This makes it difficult to differentiate $p$ across $\Gamma$, in order to discretize Eqs. (2.24) and (2.25) using standard finite differencing. Instead of resolving this problem by smearing out the pressure profile at $i-1, i, i+1, i+2$, we follow the implementation of the variable coefficient Poisson's equation in [45] to keep the profile sharp. First, the pressure at node $i$, $p_i$, and the pressure at node $i+1$, $p_{i+1}$, are extrapolated across $\Gamma$ to decide the ghost values, $p_{i+1}^G$ and $p_i^G$ :

$$p_i^G = p_i + J \tag{2.27}$$
$$p_{i+1}^G = p_{i+1} - J. \tag{2.28}$$

Using these ghost values, accurate derivatives at $\Gamma$ can be determined:

$$p_{x,\Gamma}^{Left} = p_{x,i+\frac{1}{2}}^{Left} = \frac{p_{i+1}^G - p_i}{\Delta x} \tag{2.29}$$
$$p_{x,\Gamma}^{Right} = p_{x,i+\frac{1}{2}}^{Right} = \frac{p_{i+1} - p_i^G}{\Delta x}. \tag{2.30}$$

We can discretize Poisson's equation (2.24) at $i$ and $i+1$ as

$$\nabla^2 p_i = p_{xx,i} = \frac{p_{x,\Gamma}^{Left} - p_{x,i-\frac{1}{2}}}{\Delta x} = D(x_i) \tag{2.31}$$
$$\nabla^2 p_{i+1} = p_{xx,i+1} = \frac{p_{x,i+\frac{3}{2}} - p_{x,\Gamma}^{Right}}{\Delta x} = D(x_{i+1}). \tag{2.32}$$

where $D$ represents the right-hand side of Eq. (2.24) in one dimension. Equations (2.31) and (2.32) can be rewritten, using Eqs. (2.29) and (2.30), as

$$\frac{\frac{p_{i+1}^G - p_i}{\Delta x} - \frac{p_i - p_{i-1}}{\Delta x}}{\Delta x} = D(x_i) \tag{2.33}$$
$$\frac{\frac{p_{i+2} - p_{i+1}}{\Delta x} - \frac{p_{i+1} - p_i^G}{\Delta x}}{\Delta x} = D(x_{i+1}) \tag{2.34}$$

and using Eqs. (2.27) and (2.28) as

$$\frac{\frac{(p_{i+1}-J)-p_i}{\Delta x} - \frac{p_i - p_{i-1}}{\Delta x}}{\Delta x} = D(x_i) \tag{2.35}$$

$$\frac{\frac{p_{i+2}-p_{i+1}}{\Delta x} - \frac{p_{i+1}-(p_i+J)}{\Delta x}}{\Delta x} = D(x_{i+1}). \tag{2.36}$$

Fortunately, Eqs. (2.35) and (2.36) can be rewritten as follows:

$$\frac{p_{i+1} + p_{i-1} - 2p_i}{\Delta x^2} = D(x_i) + \frac{J}{\Delta x^2} \tag{2.37}$$

$$\frac{p_{i+2} + p_i - 2p_{i+1}}{\Delta x^2} = D(x_{i+1}) - \frac{J}{\Delta x^2}. \tag{2.38}$$

These equations can be assembled into a linear system $\mathbf{Ax} = \mathbf{b}$, where the matrix $\mathbf{A}$ is symmetric and positive definite with appropriate boundary conditions. Note that the left-hand side terms of Eqs. (2.37) and (2.38) are identical to those used in [16]. A small modification of $\mathbf{b}$ involving the pressure jump, $J$, is all that is required for accurate implementation of surface tension. This method is applicable to both free surfaces and internal interfaces (bubble surfaces) and can be combined with discretization using an octree data structure [47] without any inconsistency. For free surfaces, the ambient air pressure is used as the Dirichlet boundary condition at nodes which adjoin ambient air, and then the jump condition activates the surface tension effect. Extension to the two- or three-dimensional case is simple. A $J$ term for each coordinate is superposed and then added to or subtracted from $D(\mathbf{x})$, in the same way as in Eqs. (2.37) and (2.38). We can then solve this linear system using the conjugate gradient method with a modified ILU preconditioner [69]. After solving Poisson's equation (2.24), the pressure derivatives of Eq. (2.25) are also determined from Eqs. (2.29) or (2.30), which allows us to compute $\mathbf{u}^{n+1}$.

#### 2.2.4.2 Viscosity

We will assume a velocity profile near an interface $\Gamma$ between two viscous, immiscible fluids, with viscosities denoted[2] as $\nu^-$ and $\nu^+$. Note that the discontinuity of viscosity brings about a discontinuity of velocity gradient field across $\Gamma$, even though velocity is continuous at $\Gamma$. In general, the exact velocity at the interface, $u_\Gamma$, cannot be stored in a Eulerian grid. This means that we cannot determine the exact derivatives of $u$ across $\Gamma$, which leads to errors in the viscosity step of Eq. (2.22), which will be propagated to adjacent grid elements. Larger differences in viscosity will cause more serious errors.

To resolve this problem, the ghost values, $u^G_{i+1}$ and $u^G_i$, are obtained by extrapolation $u_i$ and $u_{i+1}$ across $u_\Gamma$, as shown in Fig. 2.14. Then we can express the exact

[2]The notation $+$ or $-$ originated from the level-set representation of the interfaces. We use $\phi < 0$ regions for water and $\phi \geq 0$ regions for air. Instead of $+$ or $-$, "left" or "right" was used in Sect. 2.2.4.1 since the sign of the pressure jump is not dependent on the sign of $\phi$.

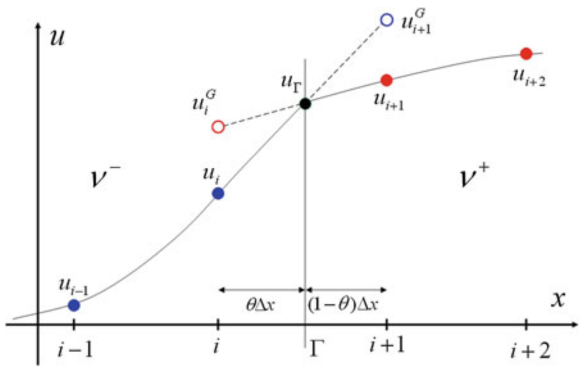


**Fig. 2.14** Velocity profile near an interface

derivatives just left and right of Γ as

$$u^-_{x,\Gamma} = u^-_{x,i+\frac{1}{2}} = \frac{u^G_{i+1} - u_i}{\Delta x} \tag{2.39}$$

$$u^+_{x,\Gamma} = u^+_{x,i+\frac{1}{2}} = \frac{u_{i+1} - u^G_i}{\Delta x}, \tag{2.40}$$

while the standard finite difference formulation is

$$u_{x,i+\frac{1}{2}} = \frac{u_{i+1} - u_i}{\Delta x}. \tag{2.41}$$

Now we can use the known physical properties of interfaces. First, the velocity should be continuous:

$$u^-_{x,\Gamma}\theta\Delta x + u^+_{x,\Gamma}(1-\theta)\Delta x = u_{x,\Gamma}\Delta x. \tag{2.42}$$

And, due to the no-slip condition, the viscous acceleration should be the same on both sides of the interface:

$$\nu^- u^-_{x,\Gamma} = \nu^+ u^+_{x,\Gamma}. \tag{2.43}$$

After rearranging Eqs. (2.42) and (2.43), we can rewrite $\nu^- u^-_{x,\Gamma}$ and $\nu^+ u^+_{x,\Gamma}$ as

$$\nu^- u^-_{x,\Gamma} = \hat{\nu} u_{x,i+\frac{1}{2}} \tag{2.44}$$

$$\nu^+ u^+_{x,\Gamma} = \hat{\nu} u_{x,i+\frac{1}{2}}, \tag{2.45}$$

where the effective viscosity[3] $\hat{\nu}$ is $(\frac{\theta}{\nu^-} + \frac{1-\theta}{\nu^+})^{-1}$. We note that it is unnecessary to decide the values of $u_i^G$ and $u_{i+1}^G$, since the discontinuity is not in the value of $u$ but in that of $u_x$.

The diffusion equation,

$$u^{new} = u + \nabla \cdot (\nu \nabla u)\Delta t, \tag{2.46}$$

is part of Eq. (2.22); we can discretize it at node $i$ as

$$u_i^{new} = u_i + \frac{\nu^- u^-_{x,\Gamma} - \nu^- u_{x,i-\frac{1}{2}}}{\Delta x}\Delta t. \tag{2.47}$$

Using Eq. (2.44), we can rewrite this equation as

$$u_i^{new} = u_i + \frac{\hat{\nu} u_{x,i+\frac{1}{2}} - \nu^- u_{x,i-\frac{1}{2}}}{\Delta x}\Delta t. \tag{2.48}$$

After repeating a similar process at node $i+1$, we obtain two equations which express the velocities near $\Gamma$:

$$u_i^{new} = u_i + \frac{\hat{\nu}\frac{u_{i+1}-u_i}{\Delta x} - \nu^- \frac{u_i - u_{i-1}}{\Delta x}}{\Delta x}\Delta t \tag{2.49}$$

$$u_{i+1}^{new} = u_{i+1} + \frac{\nu^+ \frac{u_{i+2}-u_{i+1}}{\Delta x} - \hat{\nu}\frac{u_{i+1}-u_i}{\Delta x}}{\Delta x}\Delta t. \tag{2.50}$$

High viscosities, large differences, and large time-steps are all troublesome in this explicit scheme. Therefore, we rewrite Eqs. (2.49) and (2.50) in implicit form:

$$-\lambda\nu^- u_{i-1}^{new} + (1 + \lambda\hat{\nu} + \lambda\nu^-)u_i^{new} - \lambda\hat{\nu}u_{i+1}^{new} = u_i \tag{2.51}$$

$$-\lambda\hat{\nu}u_i^{new} + (1 + \lambda\nu^+ + \lambda\hat{\nu})u_{i+1}^{new} - \lambda\nu^+ u_{i+2}^{new} = u_{i+1}, \tag{2.52}$$

where $\lambda = \Delta t/\Delta x^2$.

These equations form two rows of a linear system $\mathbf{Ax} = \mathbf{b}$. This is a modification of the single-phase case of [77], but nevertheless $\mathbf{A}$ is still symmetric and positive definite, in the same manner as the variable viscosity of [5]. When implemented, based on an octree data structure, the unpreconditioned conjugate gradient method showed better performance than preconditioned methods when the initial guess for $\mathbf{x}$ was $\mathbf{u}^n$. Using an adaptive grid, the terms $\Delta x$ in Eq. (2.39) and in Eq. (2.47) are different. The errors caused by T-junctions [47] do not significantly impact the visual results.

[3]This approach separates the pressure jump condition from the viscosity gradient jump condition. For a more complicated and coupled treatment, see [36].

### 2.2.5 Results

Let us explain the small-scale features of fluid motion simulated by the techniques just described with examples. The first example is the capillary instability of the liquid jet in Fig. 2.15. A 5 mm diameter jet is introduced into the computational domain from the left and gravity accelerates it rightward. First, the head of the jet becomes rounded by surface tension. As the head gets bigger, necking develops, and then the head is pinched off. This is the start of instability. This process is propagated to the liquid following, causing repeated sequential pinching-off. The oscillation of drops due to surface tension makes them look like elastic balls. The very thin and short filaments left after breaking also coalesce into small droplets, but some of them are lost, as reported by [21]. Refer to Picture 122 and 123 in [11] as a comparison of the simulated results with a real picture.

Similar phenomena are found in a liquid sheet, as shown in Fig. 2.16. Surface tension rounds the edges of a thin liquid sheet, then forms it into pipes, which tear the sheet. The torn parts coalesce with adjacent regions of the original sheet, but some fragments are lost because they are so thin. These phenomena also appear in photographs, such as Picture 149 in [11]. Some more interesting scenes are shown in Fig. 2.17. Here, liquid is poured on to a static sphere which causes it to spread into sheets. These are agglomerated by surface tension, finally forming many drops. The strong surface tension makes the liquid sheet behave as an elastic membrane.

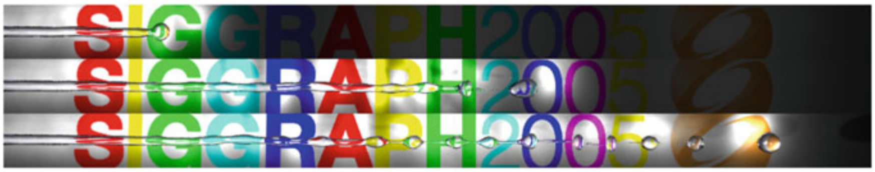


**Fig. 2.15** The capillary instability of a liquid jet. The effective resolution is $64^3$ by 25

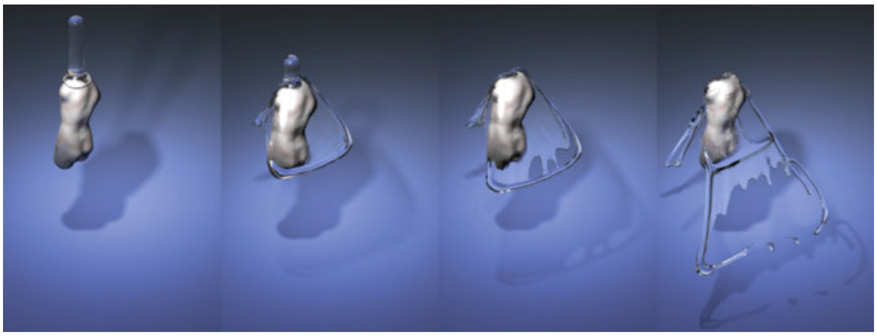


**Fig. 2.16** The breakup of a liquid sheet. The effective resolution is $512^3$

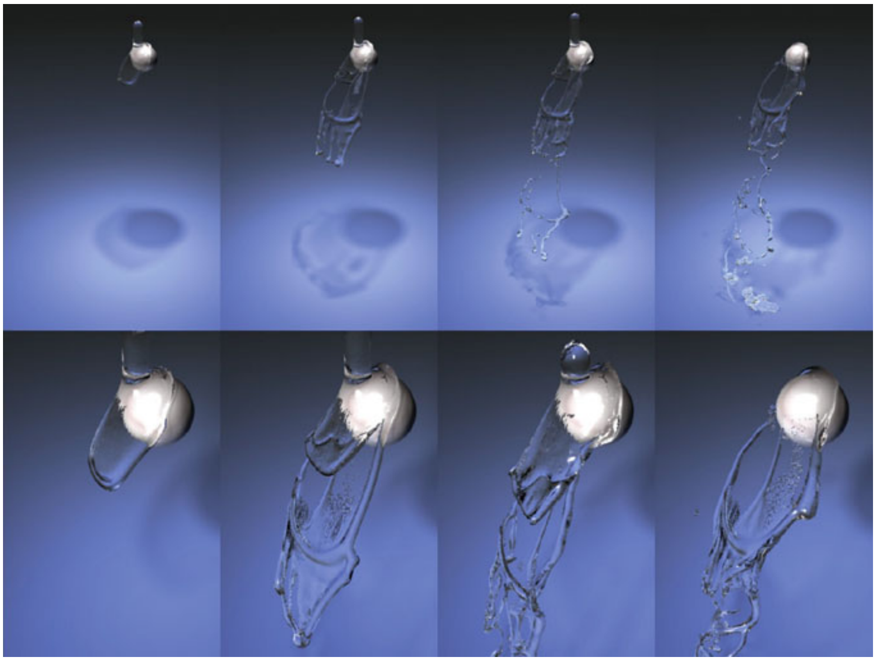


**Fig. 2.17** A small-scale scene of liquid pouring. The effective resolution was $512^3$ by 2

Figure 2.18 shows an animation of bubbly water. Air is introduced from the bottom and a static sphere disturbs its flow. The breakup of air, the formation and rising of bubbles, and explosions at the surface are all animated. The phenomena simulated in previous examples are also seen in this animation, but viscosity has now become significant. It is difficult to get visually pleasing bubbles without considering the viscosity jump, because the ratio between the viscosities of two fluids influences the position of the buoyant vortex center which affects bubble shapes as much as surface tension.

These simulations were performed on a desktop PC with 3.4 GHz CPU and 2 GB RAM. For the octree data structure, the implementation of [46, 47] was followed. Each time-step took at most 2 min of computation time with an average of 1 min. At most, 20 time-steps per frame were used, allowing one example sequence to be generated in two days of computing. This method is not primarily intended to compete with existing techniques in terms of efficiency; however, it is known that surface tension effects can induce surface oscillations when large time-steps are used, thus slowing the simulation as a whole. However, what limits the time-step in our experience is not the surface tension, but the swirling near very small droplets or bubbles.

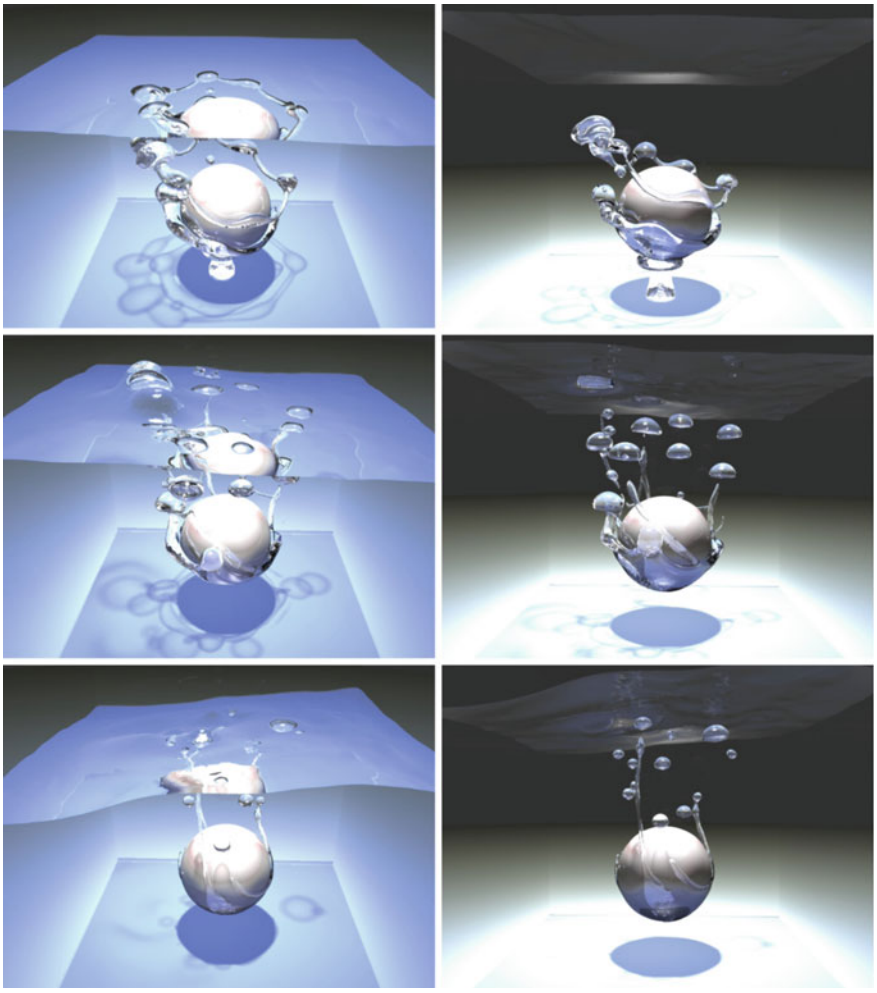


**Fig. 2.18** The animation of bubbly water. The effective resolution was $256^3$

### 2.2.6 Conclusion

This section has described a technique for animating fluids which have a discontinuity in their state variables. We have extended previous techniques to multiphase fluids with surface tension effects and viscosity changes at their interfaces, as well as modeling buoyancy. Discontinuities in the pressure and velocity gradient fields were treated in a sharp fashion which preserved subgrid details. The resulting numerical methods are easy to implement and do not influence the performance of existing solvers. Based on these techniques, we have been able to show new aspects of small-scale fluid motions.

One technical extension of this work would be to consider the discontinuity of velocities tangent to interfaces. And also this method of modeling viscosity could be enhanced to include elastic or plastic bodies. An efficient shape control algorithm for multiphase fluids would be another interesting project.

## 2.3 Bubbles Alive

**Abstract** This section proposes a hybrid method for simulating multiphase fluids such as bubbly water. The appearance of subgrid visual details is improved by incorporating a new bubble model based on smoothed particle hydrodynamics (SPH) into a Eulerian grid-based simulation that handles background flows of large bodies of water and air. To overcome the difficulty in simulating small bubbles in the context of the multiphase flows on a coarse grid, we heuristically model the interphase properties of water and air by means of the interactions between bubble particles. As a result, we can animate lively motion of bubbly water with small-scale details efficiently.

### *2.3.1 Introduction*

The lively but chaotic motion of bubbles has enchanted and challenged many scientists. Besides the engineering applications, including ship hydrodynamics, cooling of nuclear reactors, and laundry machines, an understanding of bubbles is indispensable to the visual realism of computer-generated animations that show the multiphase characteristics of fluids. In computer graphics, many researchers are struggling to get more realistic bubbles and foams by means of physics-based fluid animation, powered by computational fluid dynamics.

The two major approaches, based on Eulerian grids and Lagrangian particles, have been competing with each other, but are now being combined. This is desirable because they are complementary methods: a particle system based on smoothed particle hydrodynamics (SPH) can be much more flexible and controllable if it concentrates on small-scale details, while large bodies of water and air can be handled efficiently and faithfully by a grid-based solver, without requiring excessive resolution.

The hybrid approach to multiphase flows (including bubbles) has received less attention than the simulation of splashes and droplets, because the difference in scale as compared to the background flow is more severe for bubbles than droplets. Water dominates the inertia because its density is 800 times higher than that of air, and thus bubbles require the surrounding water to be simulated in more detail than the air around a splash. In our experience, each bubble should occupy at least $3^3$ nodes (or $3^2$ in 2D) to have numerical meaning. This makes it infeasible to refine the grid sufficiently to capture all the small details, especially in a graphics context. That is why it is desirable to develop a dynamic model appropriate for representing details at the subgrid scale.

This section proposes a hybrid method for simulating multiphase fluids, especially focusing on bubbles. To avoid excessive refinement of the background grid, while maintaining the subgrid details of bubble motion including path instability, we model the interphase properties of water and air in terms of the interactions between bubble particles. While this is ultimately a heuristic approach, it is underpinned by the SPH vorticity confinement method and an analysis of the cohesive forces that generate subgrid turbulence. This combination enables us to capture the natural look of moving bubbles in a way that harmonizes with an underlying grid-based simulation of multiphase flows.

### *2.3.2 Previous Work*

The success of grid-based liquid animation techniques that use a free surface single-phase model (see [14, 16, 19] for examples) led to work on the direct numerical simulation of multiphase phenomena [25, 27, 31, 39, 41, 51, 57, 76].

Premoze et al. [65] presented a particle-based method for fluid simulations that can handle multiphase liquids. Müller et al. [60] applied the SPH method to multiple phases, and [6] modeled the nucleation, collision, and drag interactions of bubbles and foams, based on a background SPH simulation.

Kim et al. [35] used the SPH method to model escaped particles within the particle level-set method [12], so as to resolve subgrid splashes. Losasso et al. [52] improved this approach by coupling a model of dense water volume to diffuse sprays. Greenwood and House [22] also modeled escaped particles to give a more detailed look to bubbles and foams, but without using SPH. Thüerey et al. [84] coupled SPH bubbles to shallow water simulations using locally defined vortices on particles.

### *2.3.3 A Hybrid Approach*

We use the Eulerian method to model the background motions of water and air bodies which are large enough to be captured using a simulation grid which can be managed by an ordinary single-CPU computer. The bubbling details that are too small to be handled on such a grid are simulated by SPH particles. We build our system on the particle level-set fluid solver [14] in order to generate bubble particles by incorporating the escaped particles back into the SPH system as bubbles, similar to [22]. However, this hybrid framework and our bubble model would also integrate well with other grid-based techniques such as the CIP method [76], the BFECC method [39], the CLSVOF method [57], or the Lattice Boltzmann method [82], if appropriate ways of generating bubbles were available.

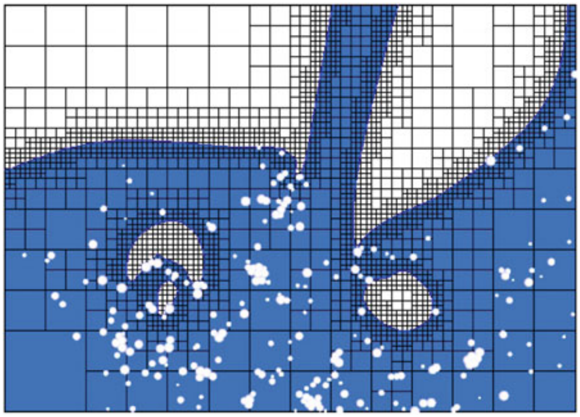


**Fig. 2.19** A schematic outline of our hybrid system. The body of water is colored *blue* and *bubble* particles are drawn as *white circles*

Figure 2.19 is a schematic overview of our hybrid system. Since we accelerate our solver by using an octree grid [47], the scale difference between the grid spacing and the particle radius is large. This difficulty is resolved by our subgrid-scale bubble dynamics, which we develop in Sect. 2.3.4.

#### 2.3.3.1 Grid-based Background Simulation

The Navier–Stokes equations describing inviscid incompressible fluid motion are

$$\mathbf{u}_t + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p/\rho = \mathbf{f} \tag{2.53}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{2.54}$$

where $\mathbf{u}$ is the velocity, $p$ is the pressure, $\rho$ is the density, and $\mathbf{f}$ is the aggregate of the external forces including gravity and the momentum exchange from the SPH bubbles that occur during coupling. Since numerical methods of solving Eqs. (2.53) and (2.54) are well known, we refer readers to [14, 27, 47] for details.

#### 2.3.3.2 SPH Overview

The acceleration of a particle $i$ is determined by a sum of forces exerted by adjacent particles, $\mathbf{f}_{ij}$, as follows:

$$\mathbf{a}_i = \sum_j \mathbf{f}_{ij}/\rho_i, \tag{2.55}$$

where the density of a particle $i$ is defined as $\rho_i = \sum m_i W(\mathbf{x}_{ij}, r_i)$. We use the radially symmetric kernel functions $W(\mathbf{x}, r)$ with support $r$, as defined in [59]. The velocity and the position of a particle can be determined by sequential Euler integrations such as $\mathbf{v}^{t+\Delta t} = \mathbf{v}^t + \mathbf{a}^t \Delta t$ and $\mathbf{p}^{t+\Delta t} = \mathbf{p}^t + \mathbf{v}^{t+\Delta t} \Delta t$, where $\Delta t$ is a time-step. Following the adaptive radius approach of [1], which provides a versatile description of bubble details, the pressure force can be expressed as

$$\mathbf{f}_{ij}^{pressure} = -V_i V_j (P_i + P_j)(\nabla W(\mathbf{x}_{ij}, r_i) + \nabla W(\mathbf{x}_{ij}, r_j))/2, \tag{2.56}$$

where the volume $V_i$ is $m_i/\rho_i$, $r$ is the radius, the mass $m_i$ is proportional to $r_i^3$, $\mathbf{x}_{ij} = \mathbf{x}_j - \mathbf{x}_i$, and the pressure $P_i = k\rho_i$ with a control parameter $k$. In general, SPH systems largely depend on viscosity, especially to improve stability when they are used to simulate large bodies. Since we use a grid-based solver to deal with large bodies, the viscous forces can be omitted.

#### 2.3.3.3 Two-way Coupling

The major coupling forces which make the bubble particles follow the background flows are drag and lift forces [6, 53], given by

$$\mathbf{f}_i^{drag} = -k_{drag} r_i^2 |\mathbf{v}_i - \mathbf{u}_i| (\mathbf{v}_i - \mathbf{u}_i) \tag{2.57}$$

$$\mathbf{f}_i^{lift} = -k_{lift} V_i (\mathbf{v}_i - \mathbf{u}_i) \times \omega_i, \tag{2.58}$$

where $\mathbf{u}_i$ and $\omega_i = \nabla \times \mathbf{u}_i$ are the velocity and the vorticity, which are interpolated at $\mathbf{p}_i$ from the grid values. Initially, we tried to simulate a simulation for the path instability of bubbles with lift forces, but this did not work well enough since Eq. (2.58) relies on the vorticity field around $\mathbf{p}_i$ being highly refined. This is one of the motivations to develop the heuristic bubble model of Sect. 2.3.4.

The forces reacting to these coupling forces are transferred to the surrounding fluid through Eq. (2.53) after being distributed across a number of adjacent nodes. We also use reaction forces to model the popping of bubbles when they merge with the ambient air. In many cases, the SPH time-step needs to be smaller than the grid simulation time-step. Since the reaction forces change the grid velocities and repeated updating makes their values diverge, they must be stored separately and only added to the right-hand side of Eq. (2.53) once per grid simulation time-step.

### *2.3.4 Bubbles*

#### 2.3.4.1 SPH Vorticity Confinement

Unlike droplets moving through ambient air, bubble particles are subject to strong velocity diffusion because they are coupled to the surrounding fluid by drag and lift forces. Furthermore, these forces are determined from values interpolated on the coarse grid. To simulate the motion of bubbles in more detail, we therefore introduce a heuristic representation of the vorticity confinement [20] into the SPH method.

First, we measure the vorticity $\omega = \nabla \times \mathbf{v}$ at the mass center of two SPH particles, $\mathbf{p}_{\oplus} = (m_i \mathbf{p}_i + m_j \mathbf{p}_j)/(m_i + m_j)$. In contrast to the grid-based method of [20], we are able to express the vorticity location vector $\eta$ as $\eta = \mathbf{p}_{\oplus} - \mathbf{p}_i$. We can use a normalized version of $\eta$, $\mathbf{N} = \frac{\eta}{|\eta|}$, to determine the confinement force:

$$\mathbf{f}_{ij}^{vorticity} = \varepsilon \left( \mathbf{N} \times \frac{\omega}{|\omega|} \right) \rho_i . \tag{2.59}$$

The original vorticity confinement method used by [20] can amplify the existing vorticity over time because the incompressibility enforced by the projection step ensures stability. Taking a similar approach makes the SPH system diverge and we therefore use a normalized $\omega$.

#### 2.3.4.2 Cohesive Forces

Due to the very large density ratio of water to air, water exerts a high pressure on air bubbles causing them to merge rapidly. To achieve a physically accurate simulation, multiphase SPH methods such as those of [24, 60] are desirable. However, because we simulate the water on a coarse grid, we have to take care of the multiphase interactions without explicit models of water particles or detailed velocities around air particles. By assuming that air particles are surrounded by water except where they are explicitly modeled, we can handle this multiphase property by simulating the attraction forces between touching particles, rather than attempting to model the forces exerted by water particles on air particles. Finally, we introduce a cohesive attraction force between particles:

$$\mathbf{f}_{ij}^{attraction} = k_{attraction} W_{attraction}(\mathbf{x}_{ij}, r_i + r_j) \rho_i . \tag{2.60}$$

We use a constant-valued function for $W_{attraction}$ to make it easy to establish a force that balances the pressure forces. The pressure force in Eq. (2.56) pushes adjacent particles outward when the density $\rho_i$ becomes high due to the attraction forces. Becker and Teschner [3] introduced a similar force to represent surface tension, but this can be adequately modeled by the intrinsic properties of SPH in the physical situation with which we are dealing.

One way of inducing clustering would be to use the pressure kernels of [1] or [3] with a negative term so that attraction forces are exerted on adjacent particles when the particle density is low. This is a reasonable approach, but our attraction force will be physically more plausible for bubbles under large water pressure. It also works better with the vorticity confinement techniques explained in the previous section, since it can suppress the scattering of particles by centrifugal effects to the extent required.

### 2.3.4.3 Subgrid Turbulence

The beauty of bubble motions is mainly a result of their unstable paths. Even a single bubble rising in calm water moves along a zigzag or spiral path due to its own wake (see [74] for an example). Our combination of a cohesive attraction force and SPH vorticity confinement approximates this characteristic motion (see Fig. 2.20) when two or more particles are close together. For single bubbles, it is simplest to add disturbances to the particles' velocities based on random numbers. This also helps to generate an initial vorticity and our system generates the natural look of turbulent bubble motion with the combination of these techniques.

### 2.3.4.4 Buoyancy

The rising velocities of bubbles are determined by the balance between drag and buoyancy, which establishes a terminal upward velocity. We generally make the buoyant force $\mathbf{f}_{buoyancy}$ proportional to the volume of each particle. An alternative is to make the buoyant force proportional to the difference between the current velocity and a terminal velocity approximately proportional to the particle radius [55]. This could be used to improve the upward motion of the bubbles.

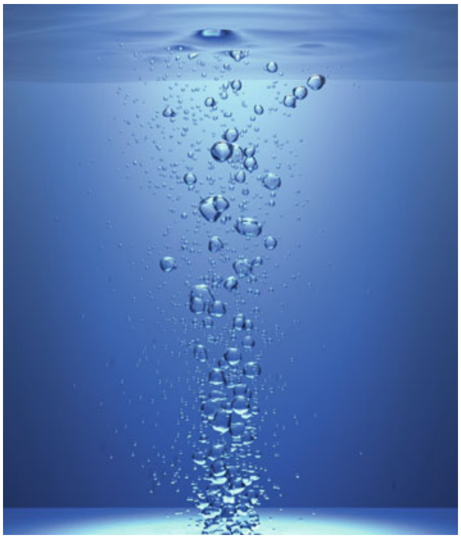


**Fig. 2.20** Rising bubbles in calm water. This example shows the realistic motion of bubbles generated by our bubble model coupled to background flows. Simulation took 3 h on an octree grid with an effective resolution of $256 \times 128^2$. A maximum of 2,600 SPH particles were used

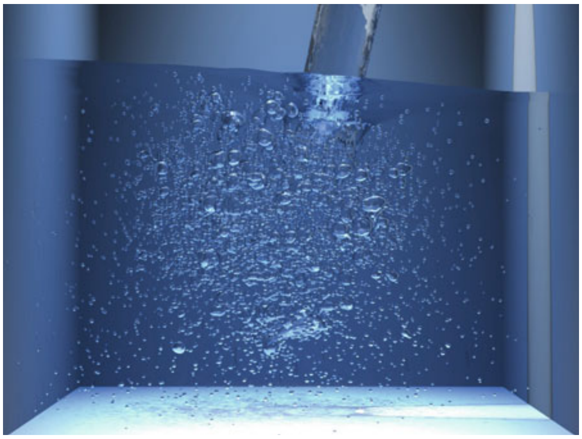


**Fig. 2.21** Water pouring with turbulent multiphase bubble flows. Simulation took 6 h on an octree grid with an effective resolution of $256 \times 128^2$. A maximum of 8,000 SPH particles were used

## 2.3.5 Examples

Both water surfaces and bubbles can be ray traced as a single level-set surface by performing on-the-fly Boolean operations that subtract air bubbles from water bodies. The particle radii were set between 0.3 and 0.8 of the grid spacing. Simulations were performed on a PC with an Intel Core2 CPU running at 3 GHz.

Figure 2.20 shows bubbles freely rising in water. In this example, bubble particles are seeded randomly at the bottom and then rise, demonstrating the basic capabilities of our bubble model. The lively and natural motion of bubbles, including flickering, merging, separation, and spiral path instability were simulated successfully. On the accompanying video there are animations with and without our vorticity confinement heuristic, which show that simply adding random disturbances is not adequate. Our bubbles pop as soon as they reach the surface, rather than persisting as foam, which could be implemented using the methods already investigated by [6, 22]. Figure 2.21 shows water being poured.

The atomization of large bodies of air is naturally modeled by escaped particles, and the coupled motion of level-set surfaces and SPH particles achieves realistic bubbly water.

## 2.3.6 Conclusion

This section has presented a hybrid of Eulerian grid-based simulation and Lagrangian SPH for the realistic simulation of multiphase fluids, focusing on bubbles. Using this heuristic bubble model, we can generate natural looking computer-generated bubbly water.

## 2.4 Hybrid Simulation of Miscible Mixing with Viscous Fingering

**Abstract** In this section, we simulate solids and liquids dissolving or changing to other substance by modeling mass transfer phenomena, and deal with the very small-scale phenomena that occur when a fluid spreads out at the interface of another fluid. We model the pressure at the interfaces between fluids with Darcy's Law and represent the viscous fingering phenomenon in which a fluid interface spreads out with a fractal-like shape. We use hybrid grid-based simulation and smoothed particle hydrodynamics (SPH) to simulate intermolecular diffusion and attraction using particles at a computable scale. As a result, we animate fluids mixing and objects dissolving.

### *2.4.1 Introduction*

In computer graphics, many fluid simulation techniques have been developed and used to create realistic animations. However, most of those techniques focus on immiscible fluids such as water, air, and bubbles. Losasso et al. [51] simulated fire and more than two liquids in the same scene, but did not deal with miscible fluids. Recently, Zhu et al. [91], Mullen et al. [58], and Park et al. [64] have presented miscible fluid simulations. However, they excluded the physical and chemical phenomena in which fluids are mixed and react with each other. When two different fluids meet, they spread out in a fractal shape because of physical pressure differences and diffusion laws. We can see this happen when ink is dropped into water. Substances can also melt and be dissolved by mass transfer caused by chemical reaction, and then change into other substances. Molecules of solute float about in the flowing fluid and spread out in a complicated fashion. This section proposes methods of simulating complicated fluid phenomena like those described above, and present animations of the interaction of miscible fluids such as ink, water, bubbles, and melting solids.

Stam [77] demonstrated stable fluid simulation using a semi-Lagrangian advection method and a decomposed version of the Navier–Stokes equation. Following Foster and Fedkiw [16], many researchers have developed multiphase fluid simulation techniques that use a level-set method. These techniques are used by the special effects industry and help to produce movies, advertisements, and games. The technique, in this section, is built on these existing technologies, with the aim of achieving stability and straightforward implementation. Other researches on miscible fluids [58, 64, 91] have used the lattice Boltzmann method (LBM), the density-based weighted essentially non-oscillatory (WENO) method, or the phase field method (PFM), but we employ none of these.

We track the interfaces of a large number of fluids using a similar approach to multiple level-set techniques. We use a separate level-set for each separate substance, and name the interface where the substances mix the mixing surface. At the mixing surfaces defined by these multiple level-sets, intricate mixing phenomena occur that create complicated fractal shapes because of the differences in concentration, viscosity, and pressure between the different substances. We assume that these phenomena are sufficiently like the flow of liquids through porous media, which follows Darcy's Law, and we model the viscous fingering exhibited by mixing fluids and simulate it using the ghost fluid method (GFM). The viscous fingering model proposed in this section is simple and easy to implement since it is modeled with pressure jump. We also model the mass transfer phenomena caused by chemical reactions using the equation of heat-dependent mass transfer proposed by Mihalef et al. [57] and Son et al. [71]. These techniques allow us to animate substances that change phase and melt to form other substances.

Hong et al. [28] and Losasso et al. [52] simulated detailed splashing and bubble motion by combining a grid-based version of Euler's method with a particle-based Lagrangian approach. We simulate the motion of molecule-like particles that represent a concentration using this hybrid method. These concentration particles experience forces that include diffusion and quasi-intermolecular attraction and repulsion. We control and simulate these forces simply using smoothed particle hydrodynamics (SPH).

### 2.4.2 Related Work

Numerical simulation of the Navier–Stokes equation has become a standard technique for the realistic animation of fluids. Foster and Metaxas [19] introduced a fully three-dimensional Navier–Stokes solver into computer graphics, and an effective and robust solution to the Navier–Stokes equation that includes semi-Lagrangian advection was reported by Stam [77]. Foster and Fedkiw [16] used a conjugate gradient method to solve the Poisson equation and an implicit level-set surface to represent the interface area effectively. Their method smooths the fluid interface, and changes of topology are represented robustly. This method has been extended to a particle level-set method by Enright et al. [14], with the addition of conservation of mass. Losasso et al. [47] used an adaptive octree data structure to show detailed fluid effects such as the crown phenomenon. Losasso et al. [51] then simulated various immiscible fluids such as oil, water, or fire using multiple level-sets. Hong and Kim [27] dealt with interface discontinuities using the GFM [15, 36]. They considered surface tension at the interface between two immiscible fluids at the projection step and introduced a discontinuous viscosity condition. Shin and Kim [73] modeled the force that drives liquids to a target shape using a pressure jump within the GFM. We model the viscous fingering phenomenon in a similar way.

There has been a spate of recent research in the computer graphics community, on the animation of phase transitions. Mihalef et al. [57] animated air bubbles in boiling water. They controlled the number and the volume of the bubbles produced by applying the equation of mass transfer by heat. Kim et al. [39] conserved the volume of air bubbles by revising the local value of divergence. Losasso et al. [49]

simulated phase transitions such as ice melting and paper burning. Wojtan et al. [86] animated corrosion and erosion of solid. In our scenario, phase changes, liquid to liquid as well, result from chemical reactions.

Desbrun and Gascuel [7] modified the SPH method to handle viscous fluids, and Müller et al. [59] proposed an interactive method in which SPH underpins the simulation of water; these authors also developed a multiphase SPH method to describe fluids of different compositions [60]. Cleary et al. [6] have improved the realism with which the collision of foam and bubbles on a complicated surface can be modeled.

Recently, much of effort has been applied to improving the efficiency of fluid simulations by modeling the details of fluid behavior on an underlying subgrid, which can be achieved by combining a Eulerian grid with Lagrangian particles. Kim et al. [35] modeled splashing by combining the SPH method with escaped particles from a particle level-set. Losasso et al. [52] developed a two-way coupling between a particle level-set and SPH to simulate diffuse regions such as splashing. Hong et al. [28] used SPH to model lively air bubbles on a coarse grid while retaining small-scale features of the flow. Lee et al. [48] proposed a way of making particle and level-set representations more interchangeable. We use the techniques mentioned above to simulate floating particles representing concentration factors.

### *2.4.3 Modeling Miscible Fluids with Multiple Level-Sets*

To simulate fluids mixing, it is necessary to simulate more than two fluids and track the interfaces between them. Losasso et al. [51] tracked numerous interfaces of immiscible fluids using multiple level-sets, and this is the approach that we use here. We use multiple level-sets in miscible fluids to trace mixing surfaces in which we make a pressure jump according to Darcy's Law and simulate pressure term. We then trace the new mixing surfaces that viscous fingering creates. Chemical mass transfer also occurs at mixing surfaces.

We use fields containing the velocity of each substance for each multiple level-set. When using a single velocity field, it is difficult to represent the mixing surfaces' characteristics described above, because the velocity field of each fluid is scattered by the effect of the diffusion and pressure terms. Therefore, we must simulate scenarios that satisfy the divergence-free condition for each fluid while considering that fluid's changes of phase. We calculate the pressure term for each fluid incorporating the effect of Darcy's Law and mass transfer. Each velocity field is extrapolated and then advected by the semi-Lagrangian advection method. We create a single velocity field for the fluid by combining the calculated velocity fields and calculate the pressure term for the velocity field of the entire fluid in an additional step. The velocity field must be adjusted so that it fulfills the divergence-free condition, and finally we calculate the diffusion term. We perform this simulation repeatedly while dividing the velocity field to match the multiple level-sets. This method is expensive, but it is able to control the changes to the multiple level-sets that occur after reactions and interactions, so it is necessary. Figure 2.22 shows a schematic outline of our method.

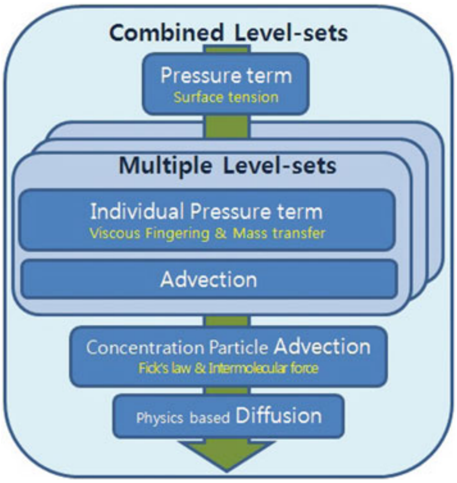


**Fig. 2.22** Schematic outline of our simulation

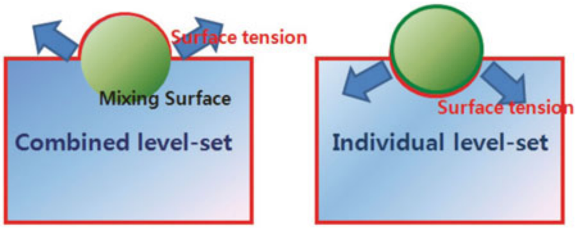


**Fig. 2.23** When using a combined level-set, surface tension (*left*) is more accurate than when using individual level-sets (*right*)

In this section, we create and use combined level-sets to describe the mixing surfaces of entire fluids, as shown in Fig. 2.23. This method allows us to choose what data to assign to the combined level-set of the entire fluid, and to the level-set of each fluid when calculating surface tensions. When using numerous level-sets, it is possible for the direction of the surface tension to be miscalculated at the intersection between level-sets, as shown in Fig. 2.23 (right). We therefore consider all the level-sets to represent a single fluid, and calculate the surface tension using the combined level-set data. This allows us to determine the correct direction for the surface tension across the whole mixed fluid, as shown in Fig. 2.23 (left).

### 2.4.4 Basic Fluids Simulation

The Navier–Stokes equation, which is the basis of our simulation, preserves mass and momentum:

$$\mathbf{u}_t = -(\mathbf{u}\cdot\nabla)\mathbf{u} + \frac{\nabla\cdot\tau}{\rho} - \frac{\nabla p}{\rho} + \mathbf{f} \tag{2.61}$$

$$\nabla\cdot\mathbf{u} = 0, \tag{2.62}$$

where $\mathbf{u}$ is the velocity, $\tau$ is the viscous stress tensor, and $\rho$ is the density. The term $\mathbf{f}$ can be used to add external forces such as gravity and buoyancy. The numerical simulation of Eqs. (2.61) and (2.62) requires the value of $\mathbf{u}$ to be updated from $\mathbf{u}^n$ to $\mathbf{u}^{n+1}$ at the $n$th time-step. We discretize Eq. (2.61) by splitting it into two equations by introducing an intermediate velocity $\mathbf{u}^*$:

$$\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} = -(\mathbf{u}^n \cdot \nabla)\mathbf{u}^n + \frac{\nabla \cdot \tau}{\rho} + \mathbf{f} \tag{2.63}$$

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^*}{\Delta t} = -\frac{\nabla p}{\rho}. \tag{2.64}$$

The variable $\mathbf{u}^*$ can be used to compute the advection term using the semi-Lagrangian method of Stam [77]. We can write the divergence of Eq. (2.64) as a form of Poisson's equation:

$$\nabla^2 p = \frac{\rho}{\Delta t} \nabla \cdot \mathbf{u}^*. \tag{2.65}$$

Once the pressure profile has been determined by solving this equation, we can obtain the final velocity profile:

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \frac{\Delta t}{\rho} \nabla p. \tag{2.66}$$

There is a discontinuous pressure profile at the interface between two different fluids. It is possible to take the discontinuous pressure at the interface into account, using GFM [27, 36]. The pressure at node $i$, which is $p_i$, and the pressure at node $i+1$, which is $p_{i+1}$, are extrapolated across $\Gamma$ to determine the ghost values, $p^G_{i+1}$ and $p^G_i$ :

$$p^G_i = p_i + J \tag{2.67}$$
$$p^G_{i+1} = p_{i+1} - J. \tag{2.68}$$

Using these equations, Eq. (2.65) can be expanded as follows [27]:

$$\frac{p_{i+1} + p_{i-1} - 2p_i}{\Delta x^2} = D(x_i) + \frac{J}{\Delta x^2} \tag{2.69}$$
$$\frac{p_{i+2} + p_i - 2p_{i+1}}{\Delta x^2} = D(x_{i+1}) - \frac{J}{\Delta x^2}, \tag{2.70}$$

where $D$ represents the right-hand side of Eq. (2.65) in one dimension. These equations can be solved using the linear system that has been used to solve Poisson's equation.

Generally, pressure jumps are modeled by surface tension, but we model $\nabla P^{Darcy}$ as the pressure jump $J$ generated at the interface of mixing fluids. This $J$ causes viscous fingering, and will be explained in Sect. 2.4.5.

We use an octree in the same way as Losasso et al. [47] to focus on the fluid interface, and can create a smooth surface from a complicated liquid interface by means of the particle level-set method [14].

### 2.4.5 Viscous Fingering

When fluids mix, we can see them spread out irregularly as their mixing surface makes a fractal-like shape. Viscous fingering refers to the onset and evolution of these instabilities in the displacement of fluids. The unstable flow of a fluid in a porous medium, or by analogy in a Hele-Shaw cell, has been studied for 50 years [30]. The results have applications in areas such as enhanced oil recovery and microfluidics.

In our scenario, we model viscous fingering with Darcy's Law, which expresses the state of a fluid passing through a porous medium. We assume that a mixing process is equivalent to a fluid infiltrating a fluid consisting of molecules. At the mixing surface between two fluids, there is a pressure jump that results from the properties of the fluid. According to Darcy's Law, the pressure gradient vector is:

$$\nabla P^{Darcy} = \frac{\hat{\mu}\mathbf{U}}{k} + \hat{\rho} g, \tag{2.71}$$

where $\hat{\mu}$ is viscosity, $\mathbf{U}$ is velocity, $k$ is permeability, $\hat{\rho}$ is density, and $g$ is gravity.

There is a discontinuity of density and viscosity at an interface between two fluids. To calculate the density at a fluid interface exactly, we use an equation proposed by Losasso et al. [51]:

$$\hat{\beta} = (\beta^- \beta^+)/\{\theta\beta^+ + (1-\theta)\beta^-\}, \tag{2.72}$$

where $\theta = |\phi(x_i)|/(|\phi(x_i)| + |\phi(x_{i+1})|)$, and the $-$ and $+$ superscripts refer to values from different sides of the interface. The viscosity at the interface can be calculated in a similar way. As mentioned in Sect. 2.4.3, each substance has its own velocity field, and $\mathbf{U}$ is the velocity of the solute.

The porosity equation is formulated using the curvature of the interface of the solute that is infiltrating into a fluid. As shown in Fig. 2.24, if the curvature is high, the fluid surface is convex and the probability of infiltration is high, but if this shape is flat or concave, the probability of infiltration is low. Thus, we express porosity as $k = \alpha\kappa_{levelset}$, where $\alpha$ is a constant that we set to 0.1, and $\kappa_{levelset}$ is the curvature of the level-set, which can be expressed as follows: $\kappa_{levelset} = \nabla \cdot \left(\frac{\nabla\phi}{||\nabla\phi||}\right)$.

We define $\nabla P^{Darcy}$ as the pressure jump $J$ generated at the interface of mixing fluids. The $u$, $v$, and $w$ velocities and pressure jump $J$ are defined at the centers of each face of a cell and referenced locally [19]. The pressure jump is therefore

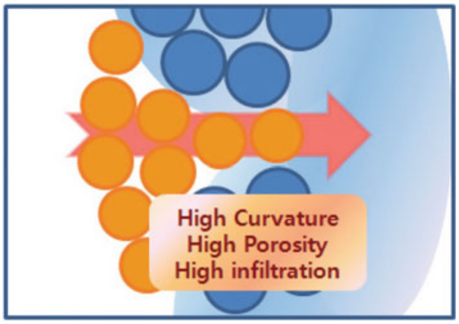


**Fig. 2.24** In our model, if the curvature is high, the porosity is high and the probability of infiltration is high

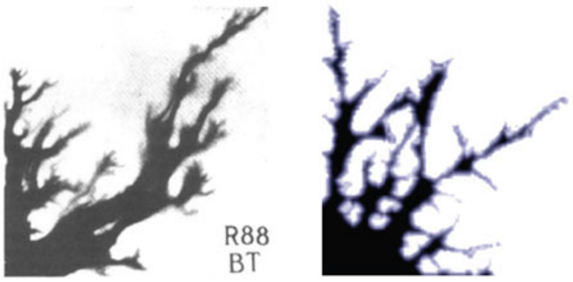


**Fig. 2.25** Viscous fingering. Miscible fluids mixing (*left*), and our simulation result (*right*)

calculated on the face of the cell that is defined as the interface in which the neighbor cell has opposite sign. Therefore, **U** is used for the velocity of the cell face, which is a scalar. For example, if there are interfaces on three cell faces, individually calculated $J$ value for each cell face are stored. $J$ consists of a scalar value for each dimension $(x, y, z)$:

$$J_{(x,y,z)} = \frac{\hat{\mu}\mathbf{U}_{(x,y,z)}}{k} + \hat{\rho}\mathbf{g}_{(z)}. \tag{2.73}$$

This formulation allows us to demonstrate viscous fingering at a mixing surface, as shown in 2D in Fig. 2.25. This is a similar structure to that shown in a real photograph of mixing fluids, due to Habermann [30], also reproduced in Fig. 2.25.

## 2.4.6 Chemical Mass Transfer

When fluids mix, chemical reactions can occur. Then, the volume of fluid may decrease or increase and some of the fluid may change into another type of fluid. We model and simulate this kind of mass transfer. Mihalef et al. [57] simulated the mass transfer that occurs when water boils. The mass transfer because of chemical reaction is similar to the phase transition that can be caused by heat. The rate of mass transfer resulting from a chemical reaction can be expressed as follows:

$$\dot{m}_c = -\frac{\mathbf{cf}_{solute} \cdot \mathbf{N} - \mathbf{cf}_{solvent} \cdot \mathbf{N}}{H}, \tag{2.74}$$

where **cf** is the chemical flux defined by $\mathbf{cf} = -D\nabla C_{cell}$, **N** is the outward facing normal from $\mathbf{cf}_{solute}$ to $\mathbf{cf}_{solvent}$, $H$ is the heat of reaction, $C_{cell}$ is the concentration of each cell, and $D$ is the coefficient of diffusivity for the concentration.

The rate of mass transfer by chemical reaction depends on the concentration of the substance. Depending on the rate of mass transfer, the level-set of the fluid is updated as follows:

$$\phi_* = \phi_n - \Delta t \frac{\dot{m}_c}{\hat{\rho}} |\nabla \phi|. \tag{2.75}$$

$\phi_*$ is advected to $\phi_{n+1}$ using level-set method after updated. Because the volume of the fluid changes, we must revise the divergence value of the pressure term by using the rate of mass transfer to control the volume of fluid, as follows:

$$\nabla \cdot \mathbf{u}^{n+1} = \frac{\dot{m}_c}{\hat{\rho}} |\mathbf{N}|, \tag{2.76}$$

$$\nabla^2 p = \frac{\hat{\rho}}{\Delta t} \nabla \cdot \mathbf{u}^* - \frac{\dot{m}_c}{\Delta t} |\mathbf{N}|. \tag{2.77}$$

This method is similar to those explained in [39, 57]. This allows us to simulate phenomena in which the volume of the substance changes, such as the melting of a solid because of a chemical reaction. Figure 2.26 shows a solid teapot melting in a transparent liquid. Because of mass transfer, the volume of the teapot decreases. The solid teapot shape is defined by an implicit surface using level-set data, which makes it easy to implement this scenario.

### 2.4.7 Hybrid Method

It is hard to model intermolecular forces using only grid-based advection of concentration. We therefore simulate the effect of molecular diffusion and interaction on concentration using what we call *concentration particles*. The advection of concentration is simulated using both a grid-based model of semi-Lagrangian advection and particle-based advection. We can assume that the concentration particles are not completely absorbed in the solvent in which they dissolve, but float about like oversize molecules. Figure 2.27 shows a 2D simulation of the mixing of two substances using this hybrid method.

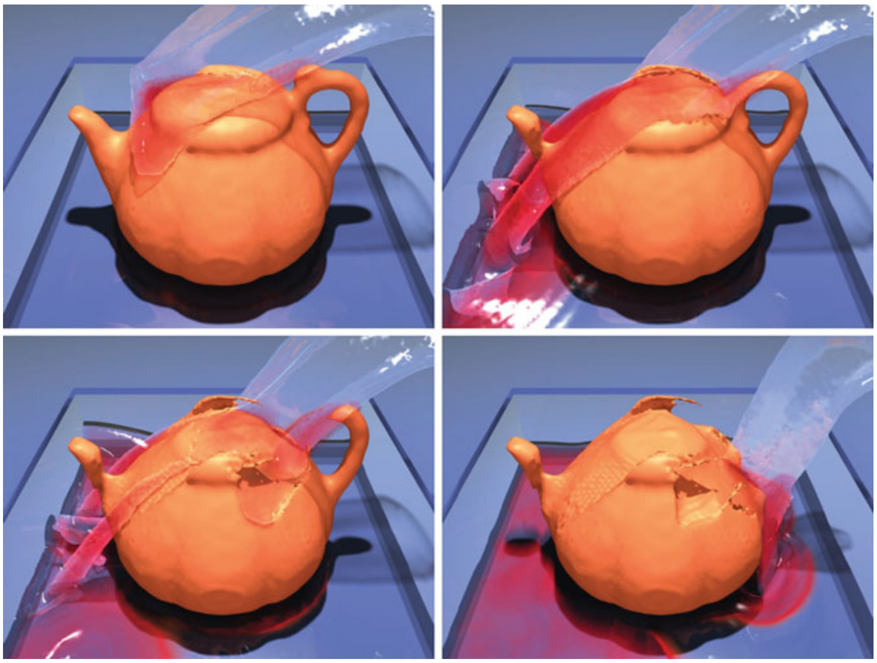
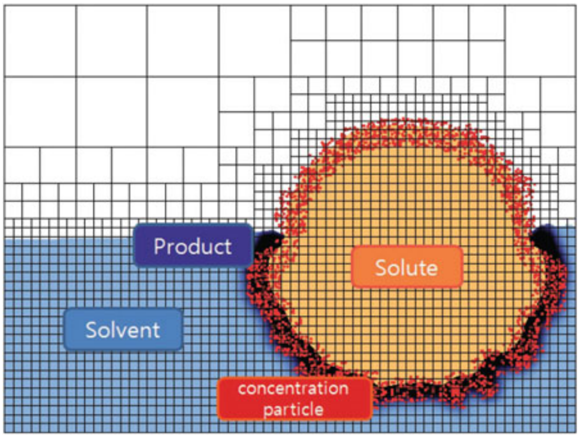


**Fig. 2.26** Pouring water on the teapot. The solid teapot melts by chemical mass transfer. (grids:128 × 128 × 128; particles: about 20,000)

**Fig. 2.27** Hybrid simulation of particle and grid-based method. The grid is 64 × 64 (6-level octree). The *blue* and *yellow* points are the level-sets and the *red* points are concentration particles. The *dark blue* points are particles modeling the concentration after the reaction



### 2.4.7.1 Concentration Particles and Absorption

Concentration particles are generated in the grid cells corresponding to a mixing surface. They are defined by position, velocity, radius, and concentration. The number of particles that are reseeded depends on the maximum that has been set for each cell. That is, we simulate more particles with reseeding to have $max_p$ particles in a mixing surface cell if the number of particles is less than $max_p$. $max_p$ is the maximum number of particles that a cell can have. In our experiment, we set $max_p$ to four or eight. When a particle is seeded, its concentration is initialized to $C_{equilibrium}/max_p$.

When one or more concentration particles exist in a cell, absorption occurs from the particles to the grid. The reaction rate is proportional to the concentration of the substance. In general, the reaction rate of two substances is $k_{reaction}[A][B]$, where $[A]$ is the sum of the particle concentrations existing in each cell ($\sum_i^{np} C^n_{particle_i}$) and $[B]$ is the concentration of solvent in the cell.

In the simulation, we limit the maximum number of cell concentrations to $C_{equilibrium}$, and we assume that the concentration of solvent $[B]$ is initialized to $C_{equilibrium}$. Because the concentration of substance $[B]$ is proportionally reduced as reaction product is generated, we can model $[B]$ as $(C_{equilibrium} - C_{cell})$. Thus, the cell concentration of reaction product absorbed from a particle for the next time-step, $C^{n+1}_{cell}$ is:

$$C^{n+1}_{cell} = C^n_{cell} + \sum_i^{np} k_{reaction} C^n_{particle_i} (C_{equilibrium} - C^n_{cell}), \tag{2.78}$$

where $C_{cell}$ is the concentration of product in the cell, $C_{particle_i}$ is the $i$th particle's concentration in the cell, $np$ is the number of particles in the cell, and $k_{reaction}$ is a coefficient that defines the rate of the chemical reaction. In our experiment, we use a small number, typically 0.001, for $k_{reaction}$ and set $C_{equilibrium}$ to 1.0.

When a concentration particle is absorbed into the cell, the particle concentration of the solute is reduced:

$$C^{n+1}_{particle} = C^n_{particle} - k_{reaction} C^n_{particle} (C_{equilibrium} - C^n_{cell}). \tag{2.79}$$

Depending on Eq. (2.79), if a particle's concentration is less than a fixed threshold, which is 0.01, the particle is deleted. If $C^n_{cell}$ is larger than $C^n_{equilibrium}$, the concentration of product decreases and the particle concentration of solute increases in the next step. The process is similar to maintaining a state of chemical equilibrium. This allows the computation to be simplified without a significant effect on accuracy.

### 2.4.7.2 Concentration Particle Advection

Concentration particles are advected by their own velocity according to a Lagrangian method. Each of their positions is updated using the equation: $\mathbf{p}^{n+1}_{particle} = \mathbf{p}^n_{particle} + \mathbf{u}^n_{particle} \cdot \Delta t$. The velocity of a particle $\mathbf{u}_{particle}$ is calculated using the equation: $\mathbf{u}^{n+1}_{particle} = \mathbf{u}^n_{particle} + \mathbf{f}_{\text{advection}} \cdot \Delta t$. The advection of concentration particles depends on the intermolecular diffusion, infiltration, and coupling forces:

$$\mathbf{f}_{\text{advection}} = \mathbf{f}_{\text{diffusion}} + \mathbf{f}_{\text{capillary}} + \mathbf{f}_{\text{molecular}} + \mathbf{f}_{\text{coupling}}. \tag{2.80}$$

Concentration particles spread out in the direction of the concentration gradient. The diffusive flux follows Fick's Law:

$$\mathbf{f}_{\mathsf{diffusion}} = -D\nabla C_{cell}, \tag{2.81}$$

where $D$ is the diffusion coefficient. It is not easy to model complicated diffusion scenarios in which irregular filaments are produced and spread out, using only the above Eq. (2.81). Thus, we use a capillary force and intermolecular forces.

The capillary force is modeled using the gradient of curvature of the concentration field. Because regions of high curvature have a high porosity, it is easy for a fluid to infiltrate in that region. We therefore model the capillary force as follows:

$$\mathbf{f}_{\mathsf{capillary}} = I|\kappa_{concentration}|\nabla\kappa_{concentration}, \tag{2.82}$$

where $\kappa_{concentration}$ is the curvature of the concentration field and $I$ is the infiltration coefficient. The value of $\kappa_{concentration}$ can be expressed as $\kappa_{concentration} = \nabla \cdot (\nabla C_{cell}/||\nabla C_{cell}||)$.

We simulate intermolecular forces using the SPH method. The intermolecular forces themselves are modeled by the interaction forces between concentration particles. The following equation, which is typically used to calculate pressure in SPH-based fluid simulations, has been found experimentally to be effective in modeling the attractive force between concentration particles:

$$\mathbf{f}_{\mathsf{molecular}} = A\sum_{j} m_j \left( \frac{P_i}{{C_{particle_i}}^2} + \frac{P_j}{{C_{particle_j}}^2} \right) \nabla W(\mathbf{x}_j - \mathbf{x}_i), \tag{2.83}$$

where the pressure $P_i = \alpha C_{particle_i}$ with the control parameter $\alpha$, $A$ is the attraction coefficient, and $m$ is the mass of a particle. We assume that the mass of all the particles is the same, with a value of 1.

When a concentration particle moves between different fluids, a resistance force is provided by the fluid into which the particle is moving. This force causes the coupling between the velocity of grid and particle. The magnitude of this force is proportional to the relative velocity of the concentration particle and the receiving fluid, and can be expressed as:

$$\mathbf{f}_{\mathsf{coupling}} = -R(\mathbf{u}_{particle} - \mathbf{u}_{cell}), \tag{2.84}$$

where $R$ is the resistance coefficient. The overall advection of the particles is provided by the sum of the forces in Eqs. (2.81)–(2.84).

### 2.4.8 Results

Simulations were performed on an Intel PC with a 3.0 GHz CPU. In general, simulation grids are economically implemented as octrees, but we must use a regular grid for scenarios where concentration must be modeled. In this case, the value of the level-set is less than 0. In other words, we can simulate concentrations where liquid is present without ambient air. Therefore, if $\phi$ over the whole level-set is more than 0, we could use an octree data structure instead of the dense regular grid that we used where $\phi$ is less than 0.

Because an octree data structure is being used, the computation cost increases in proportion to the volume of fluid that is simulated. The simulation shown in Fig. 2.26 takes at least 15 s per frame and at most 120 s. Mental Ray in Maya is used for rendering. Simulation data are saved as a mesh and a scalar field to be rendered in Maya.

Figure 2.28 shows that the Venus-shaped volume of fluid dives into an open water surface and is mixed. The two different fluids intermingle and spread out just as ink mixes with water. This experiment was performed using a $256 \times 256 \times 128$ grid. The simulations took approximately 9 min per frame, including running the fluid solver and the file I/O.

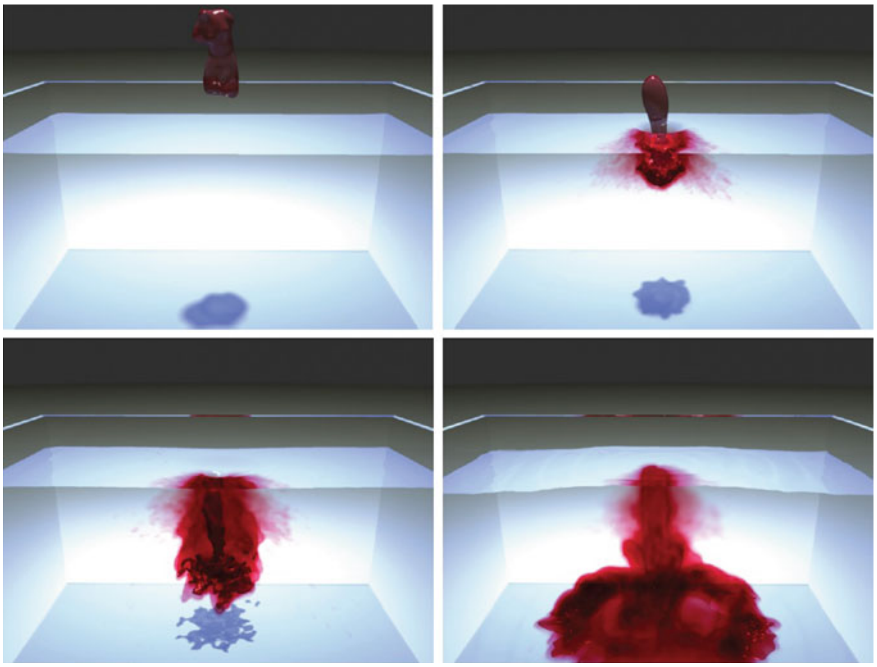


**Fig. 2.28** Dropping *red ink*. Two different liquids mix (grids: $256 \times 256 \times 128$, particles: about 120,000)

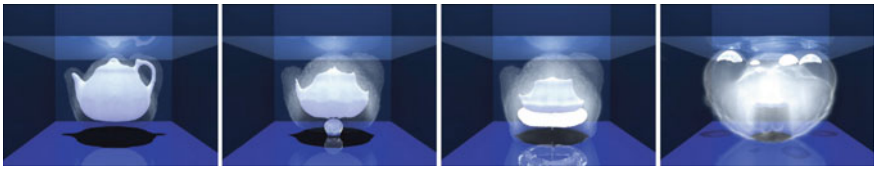


**Fig. 2.29** Dissolving a solid teapot. By mass transfer, the volume of the solid decreases (grids: $128 \times 128 \times 128$, particles: about 30,000)

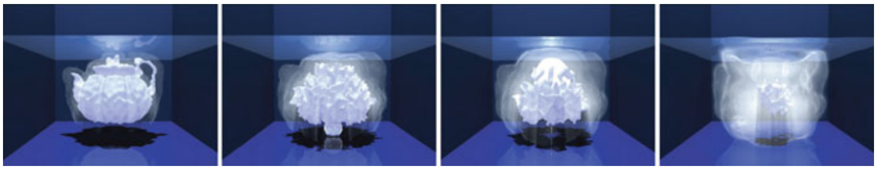


**Fig. 2.30** Dissolving a liquid teapot. Viscous fingering occurs at the mixing surface (grids, particles: same as above)

Figure 2.29 shows how a solid teapot shape dissolves in a liquid. As the volume of the solid is reduced by mass transfer, the solid teapot dissolves in water. Violet material from the teapot that has been mixed in water floats on the surface and interacts with air bubbles. Figure 2.30 shows a somewhat different scenario in which a liquid teapot mixes with the water that surrounds it. In contrast to Fig. 2.29, viscous fingering now takes place because of the difference in pressure at the interface between the two fluids and a liquid core itself flows. Bubbles were intentionally inserted to show the comparison between the solid and the liquid when interacting with bubbles. We can thus see the complicated phenomena of liquids mixing. The size of grid for the simulations shown in Figs. 2.29 and 2.30 was $128^3$. The average simulation time per frame was 120 s.

Figure 2.31 presents a dramatic scenario in which a liquid teapot and a rigid teapot are dropped into water one by one. First, it shows how they react when they are of the same type of substance, clear water. Second, the insoluble object drops, and when it reaches the bottom of the water, it turns into blue ink teapot. It slowly dissolves and mixes with the water. The grid for this simulation was $128^3$, and the average simulation time per frame was 100 s.

### *2.4.9 Conclusions*

This section has described a technique for modeling the flow of miscible multiphase fluids by improving the handling of interfacial properties and chemical reactions. In several experiments, we constructed naturalistic scenarios in which a solid body melts or liquids are mixed. These combinations of viscous fingering, chemical-based mass

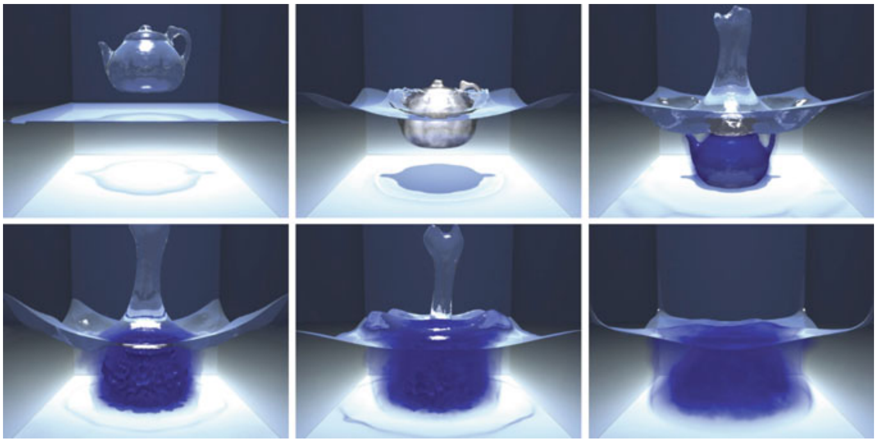


**Fig. 2.31** Water meets three types of teapots: clear water, steel, and *blue ink* (grid:128 × 128 × 128, particles: about 25,000)

transfer, and molecular forces are relatively easy to model with techniques familiar to the computer graphics community. In the future, it can be researched to simulate smaller scale features such as the filaments that appear when ink spreads out in water. And also, simulations for the liquid–solid reaction could be performed, based on accurate chemical laws such as solution and solidification with a state of chemical equilibrium.

## 2.5 Anisotropic Particle Level-Set Method for Multiphase Fluid

**Abstract** This section presents how to track the surface of a multiphase fluid more accurately by using the particle level-set method with anisotropic particles instead of spherical particles. While we use the weighted version of principal component analysis (WPCA) to construct the anisotropic particles, its computational cost is high. We adopt the distribution of particles from the directional derivative to generate the anisotropic particles. Compared to particle level-set method, this approach provides more details of surface, corrects numerical dissipation, and preserves the volume of the fluid. Furthermore, this section presents particle-based fluid simulations with surface reconstruction that uses anisotropic particles.

### 2.5.1 Introduction

Accurately capturing the interfaces of a multiphase fluid is a challenging problem in computer graphics. The level-set method within a Eulerian simulation can track the free surface of a liquid [16]. However, the standard grid-based semi-Lagrangian advection method only has first-order accuracy, so there is a large amount of numerical dissipation. Attempts to track the fluid interface more accurately have included the use of various triangle meshes or marker particles. Mesh-based surface tracking combines existing resampling methods with the use of convex hulls to connect surface features during topological changes [83, 87]. The particle level-set method corrects the level-set near the surface. Enright et al. [14] added Lagrangian particles to the level-set in order to represent surface details more accurately. To implement this method, marker particles are seeded near the surface and advected by a velocity which is tri-linearly interpolated at the particle position. Then the particles can be used to correct errors caused by numerical dissipation. To calculate the level-set on both sides of the fluid interface, the particles are considered as spheres, with radii determined by their distance from the surface. This method is widely used because it is very simple and resulting surface preserves the volume of the fluid.

This section proposes an anisotropic particle level-set method to achieve a more accurate fluid interface than the spherical particles. We create an anisotropic particle by considering distribution of particles. Though use of a weighted version of principal component analysis(WPCA) results the fluid surface accurately, the search for neighboring particles required to calculating singular value decomposition involves a high computational cost. So this section devises a new method in which directional derivative is used to generate the anisotropic particles. This reduces the computational costs but still allows the fluid interface to be tracked more accurately.

### 2.5.2 Related Work

The simulation of liquid using the Navier–Stokes equation has been researched for a long time. Foster and Metaxas [19] developed a three-dimensional Navier–Stokes method for fluid simulation. Stam [77] proposed a semi-Lagrangian integration scheme to simulate unconditionally stable fluids using a three-dimensional grid. Losasso et al. [47] utilized an adaptive octree structure to obtain a high resolution surface. Hong and Kim [27] considered surface tension between multiphase fluids using the ghost fluid method (GFM) to deal with discontinuities at the fluid interface. To represent the fluid surface more accurately, Enright et al. [14] proposed the particle level-set method. Then, they improved this method by introducing a semi-Lagrangian approach to track the surface more accurately and rapidly. Mihalef et al. [56] handled the dynamics of a liquid and its surface color. Ianniell and Mascio [33] tracked the interfaces by Lagrangian oriented particles in conjunction with a level-set. Advection of simulation is also for greater accuracy, while CIP [76] ensure high order accuracy. Furthermore, there are several hybrid approaches for bubble [22] and splash [28, 35]. Losasso et al. [52] used two-way coupled particle level-set and SPH [48] in order to simulate diffuse regions such as splashing. Kim et al. [38] simulate an ellipsoidal shape of an air bubble by a drag force on its upper surface. Wojtan et al. [88] were able to represent detailed fluid surfaces with thinner features using triangle meshes. Our anisotropic particle level-set approach has been inspired by anisotropic particle methods. Liu et al. [50] employed anisotropic kernels in the SPH simulation. Yu and Turk [90] formulated anisotropic smoothing kernels using the WPCA method. Jo et al. [34] simulated SPH-based fluids using anisotropic kernels formed by the particle velocities. We refer to works of Donia et al. [8], Zheng et al. [92], and Rahman and Murshed [68] for creation of anisotropic kernel. Donia et al. [8] propose a texture generation method by computing their movement of a motion distribution followed by the generation of image frames. Zheng et al. [92] detected a pattern by judging of eclipse and Support Vector Machines (SVM). Other interesting works include viscoelastic fluids [21], control methodology [83], vortex particle [75], and subgrid turbulence model [70].

### 2.5.3 Particle Level-Set Method (PLS)

We use the Navier–Stokes equation to simulate large volumes of liquid. The momentum conservation equation is

$$\mathbf{u}_t + (\mathbf{u}\cdot\nabla)\mathbf{u} + \frac{\nabla p}{\rho} = \mathbf{f} \tag{2.85}$$

and the mass conservation equation is

$$\nabla\cdot\mathbf{u} = 0 \tag{2.86}$$

where $\mathbf{u} = (u, v, w)$ is velocity, $p$ is pressure, and $\mathbf{f}$ is the sum of the external forces, including gravity and control forces. We use octree structures [47] for fast grid simulation, back and forth error compensation and correction (BFECC) [40] to achieve second-order accuracy in fluid volume preservation, and the particle level-set method [14] to represent complicated fluid surfaces.

First, marker particles are seeded in the surface region. The radius $r_p$ of a particle as follows:

$$r_p = \begin{cases} r_{max} & \text{if } s_p\phi(\mathbf{x}_p) > r_{max} \\ s_p\phi(\mathbf{x}_p) & \text{if } r_{min} \le s_p\phi(\mathbf{x}_p) \le r_{max} \\ r_{min} & \text{if } s_p\phi(\mathbf{x}_p) < r_{min} \end{cases} \tag{2.87}$$

where $s_p$ is the sign of the particle, $\phi(\mathbf{x}_p)$ is the implicit function, and $\mathbf{x}_p$ is the particle position. The minimum radius is $0.1\cdot\min(dx, dy, dz)$ and the maximum radius is

$0.5 \cdot \min(dx, dy, dz)$. In most of the examples presented later in this paper, 32 marker particles are used each cell. Then the level-set is integrated using Eq. (2.85), while the particles are advected with the interpolated velocity at their positions. The error correction scheme proposed by Enright et al. [14] uses spherical particles. A spherical implicit function $\phi_p(\mathbf{x})$ is determined by the marker particle radius, as follows:

$$\phi_p(\mathbf{x}) = s_p(r_p - |\mathbf{x} - \mathbf{x}_p|) \tag{2.88}$$

where $r_p$ is the particle radius and $\mathbf{x}$ is the position of the node. Then a new value of the level-set is determined by comparing the spherical implicit function $\phi_p(\mathbf{x})$ with the current level-set value. The positive and negative level-set values are then obtained as follows:

$$\phi^+ = \max_{\forall p \in \mathbf{E}^+}(\phi_p, \phi^+) \tag{2.89}$$

$$\phi^- = \max_{\forall p \in \mathbf{E}^-}(\phi_p, \phi^-) \tag{2.90}$$

where $\phi^+$ is the level-set value in the $\phi > 0$ region, $\phi^-$ is the level-set value in the $\phi < 0$ region, $\mathbf{E}^+$ is the set of escaped positive particles, and $\mathbf{E}^-$ is the set of escaped negative particles. More details on this are given elsewhere [13, 14].

## 2.5.4 *Anisotropic Particle Level-Set Method (APLS)*

We use anisotropic, instead of spherical, particles in the particle level-set method. We specify an anisotropic particle by its position, its three axes and its magnitudes along three axes.

### 2.5.4.1 Determining Particle Axes and Magnitudes Using WPCA

We first determine the axes of a particle from the distribution of neighboring particles by employing the WPCA Method [90]. We compute weighted mean $\mathbf{x}_i^w$ from the neighboring particle to construct the covariance matrix, as follows:

$$\mathbf{x}_i^w = \frac{\sum_j w_{ij}\mathbf{x}_j}{\sum_j w_{ij}} \min_{\forall p \in \mathbf{E}^-}(\phi_p, \phi^-) \tag{2.91}$$

The weight function $w_{ij}$ is calculated by using neighbor particles which have the same sign within radius $r_i$, as follows:

$$w_{ij} = \begin{cases} 1 - (||\mathbf{x}_i - \mathbf{x}_j||)^3 & \text{if} \quad ||\mathbf{x}_i - \mathbf{x}_j|| < r_i \\ 0 & \text{otherwise} \end{cases} \tag{2.92}$$

Then, the covariance matrix $\mathbf{C}_i$ is obtained as follows:

$$\mathbf{C}_i = \frac{\sum_j w_{ij}(\mathbf{x}_j - \mathbf{x}_i^w)(\mathbf{x}_j - \mathbf{x}_i^w)^T}{\sum_j w_{ij}} \tag{2.93}$$

For each marker particle, we perform a singular value decomposition of the covariance matrix $\mathbf{C}_i$ to obtain the eigenvectors and eigenvalues, which become the axes and magnitudes ($\sigma_1 < \sigma_2 < \sigma_3$) of the anisotropic particle.

#### 2.5.4.2 Error Correction Using Anisotropic Particles

Because we use anisotropic particles, we need to modify the error corrections scheme of the particle level-set method. We can obtain the radius of an anisotropic particle $r_p$ in any direction by solving Eq. (2.94). The local coordinate system of each particle transforms the position of the node. The local coordinate system is determined by three axes in Sect. 2.5.4.1. We can calculate $r_p$ for an arbitrary direction as follows:

$$r_p = \sqrt{\frac{1}{\frac{x_{lx}^2}{\sigma_1'^2} + \frac{x_{ly}^2}{\sigma_2'^2} + \frac{x_{lz}^2}{\sigma_3'^2}}} \tag{2.94}$$

where $\mathbf{x}_l = (x_{lx}, x_{ly}, x_{lz})$ is the position of node in the local coordinate system, $\sigma_1'$ is the shortest distance ($\phi_p$) to the surface, $\sigma_2' = (\sigma_2/\sigma_1)\phi_p$ and $\sigma_3' = (\sigma_3/\sigma_1)\phi_p$. So we apply this $r_p$ to Eq. (2.88) and obtain the new $\phi_p(\mathbf{x})$ using anisotropic particle. This approach improves the representation of the level-set surface, but it takes about 21 times longer than particle level-set method. This is because the data structures used in the anisotropic particle level-set method using WPCA are inappropriate for neighbor searching. We could use fewer anisotropic particles but then the resulting surface is to be worse. Section 2.5.4.3 introduces a new method to avoid this problem.

#### 2.5.4.3 Reducing of Computational Cost Using the Directional Derivative

If the number of anisotropic particles in APLS using WPCA decreases, the computational cost also decreases but the results would be of lower quality. Therefore, this section proposes a new method that has good performance in time consumption as well as improves representation of the surface. We generate an anisotropic particle using directional derivative instead of WPCA method. First, we discard the WPCA calculation. Second, we update the level-set using PLS. Third, we determine three axes of the anisotropic particle using directional derivative; the error correction module is computed in the anisotropic particle level-set method. More details are described in Algorithm 3.1.

**Algorithm 3.1 APLS Method**

```
1  for all leaf nodes do
2    (re)seed marker particles and set their radius
3  for all leaf nodes do
4    for all particles do
5      Time integration of marker particles
6  for all leaf nodes do
7    Time integration of the implicit function
8  for all leaf nodes do
9    for all particles do
10     delete and add marker particles
11     compute updated $\phi_p(\mathbf{x})$ using PLS
12 for all leaf nodes do
13   for all particles do
14     compute normal at the its position ($\nabla\phi$)
15     set minor axis ($\mathbf{e}_1$)
16     set major axis $\mathbf{e}_2$ using the directional derivative
17     set last axis $\mathbf{e}_3$= cross product($\mathbf{e}_1$, $\mathbf{e}_2$)
18 for all leaf nodes do
19   for all particles do
20     compute new $\phi_p(\mathbf{x})$ using the anisotropic particles
```

The minor axis $\mathbf{e}_1$ is determined from the gradient of $\phi$. The major axis is the axis with the minimum variation of level-set value. The direction of minimum variation is calculated using the directional derivative, as follows:

$$\nabla_{\mathbf{e}}\phi(\mathbf{x}) = \frac{\phi(\mathbf{x} + h\mathbf{e}) - \phi(\mathbf{x} - h\mathbf{e})}{2h} \tag{2.95}$$

where $\phi(\mathbf{x})$ is the level-set value at the position $\mathbf{x}$ and $\mathbf{e}$ is the direction that we seek, $||\mathbf{e}|| = 1$. In our example, $h$ is the particle radius. To find the minimum variation, we calculate the directional derivative iteratively using Eq. (2.95) on the plane, normal to $\mathbf{e}_1$. The direction that has the minimum variation is the major axis $\mathbf{e}_2$. The $\mathbf{e}_3$ axis is calculated by the cross product of the minor axis $\mathbf{e}_1$ and the major axis $\mathbf{e}_2$. We can introduce into Eq. (2.94) since we can obtain $\sigma_1$, $\sigma_2$ and $\sigma_3$ by calculating the directional derivative along the axes $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$ using Eq. (2.95). The value of $\sigma_1$, $\sigma_2$, and $\sigma_3$ are inversely proportional to the variation of the level-set in the corresponding direction. This simple calculation improves the performance of the anisotropic particle level-set method using directional derivative.

#### 2.5.4.4 Additional Trials

Our new method was applied to the surface reconstruction module of a particle-based fluid simulation using SPH [1, 59, 60]. Our simulations and surface reconstruction algorithms largely follow [90]. We only modify their method of setting the anisotropic kernels using WPCA.

The normal vector at each particle that is used to calculating the surface tension force is the minor axis of the anisotropic particle. We calculate the rate of density change for all neighboring particles. If a particle $j$ has the smallest rate of density change, we temporarily align the vector $\mathbf{x}_j - \mathbf{x}_i$ with the major axis. The direction of the third axis is the cross product of the minor and major axes. However, the major and minor axes must be at right align to each other. So we have to change the direction of the major axis to become the cross product of the direction of the minor axis and the third axis. The value of $\sigma_1$, $\sigma_2$ and $\sigma_3$ are inversely proportional to the rate of density change along the corresponding axes. This approach eliminates the computational time required to obtain three eigenvectors and three eigenvalues.

**Algorithm 3.2.1 Particle-based Fluid Simulation**

```
1   for all particles $i \in \mathbf{S}$ do
2     find neighbors $\mathbf{N}_i$
3   for all particles $i \in \mathbf{S}$ do
4     compute $\rho_i$, $p_i$
5   for all particles $i \in \mathbf{S}$ do
6     compute forces (gravity, pressure, viscosity, surface
    tension)
7   for all particles $i \in \mathbf{S}$ (for all about previous step) do
8     compute anisotropic kernel (compute three axes)
      $\rightarrow$ Algorithm 3.2.2
9     compute node density (for surface reconstruction)
10  for all particles $i \in \mathbf{S}$ do
11    compute new $\mathbf{v}_i$, $\mathbf{x}_i$
```

**Algorithm 3.2.2 Compute Anisotropic Particle**

```
1   major axis = normal vector of particles
2   for all particles $i \in \mathbf{N}_i$ do
3     compute rate of density change $i$, $j$ ($\Delta\rho_{ij}$)
4     if ($\Delta\rho_{min} > \Delta\rho_{ij}$)
5       $\Delta\rho_{min} = \Delta\rho_{ij}$
6       major axis = $\mathbf{x}_j - \mathbf{x}_i$
7     end if
8   the other axis = cross product(minor axis, major axis)
9   major axis = cross product(minor axis, the other axis)
```

### *2.5.5 Results*

Simulations were performed on an Intel Core i7 CPU running at 2.93 GHz, and rendered the fluid models by ray tracing. BFECC (back and forth error compensation and correction) was used for the advection, with an octree grid. The maximum depth of the octree is 7, so yielding a 128 × 128 × 128 grid. Monte Carlo integration was used to measure the volume of the fluid.

Figure 2.32 shows that the water ball bounced back from the surface. Figure 2.32a is the 21st frame with the PLS. It takes 0.7226 s to run the advection module with 32 particles, and the volume of fluid decreases by 0.00129. Figure 2.32b is the 21st frame with the APLS using WPCA. It takes 15.8472 s to run the advection module with 32 particles, and the volume of fluid decreases by 0.00123. Figure 2.32c is the 21st frame with the APLS using directional derivative. It takes 0.7477 s to run the advection module with 32 particles, and the volume of fluid decreases by 0.00107. Figure 2.32c shows complex water surface by adding trivial time.

Figure 2.33 shows that the water ball was dropped onto the fluid surface. Figure 2.33a is initial water ball drop and Fig. 2.33b is the 7th frame with the PLS. It takes 0.6172 s to run the advection module with 32 particles, and the volume of fluid decreases by 0.00027. Figure 2.33c is the 7th frame with the APLS using WPCA. It takes 14.3663 s to run the advection module with 32 particles, and the volume of fluid decreases by 0.00011. Figure 2.33d is the 7th frame with the APLS using directional derivative. It takes 0.6226 s to run the advection module with 32 particles, and the volume of fluid decreases by 0.00005. Figure 2.33d is similar to Fig. 2.33c for the volume, although the computational time of Fig. 2.33d is about 0.04333 times lower than the APLS with WPCA.

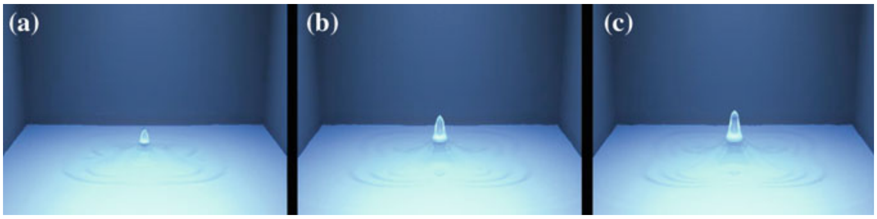


**Fig. 2.32** Water drop tower: **a** PLS **b** APLS with WPCA **c** APLS with directional derivative

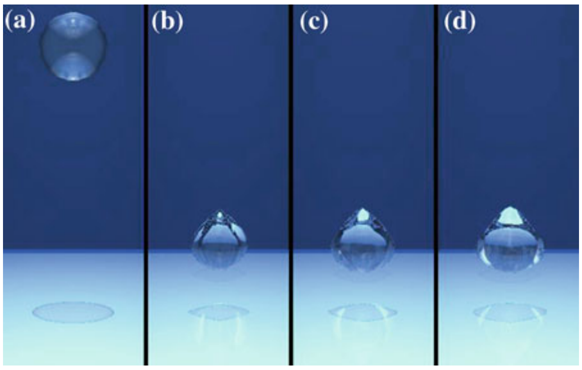


**Fig. 2.33** Water ball drop: **a** The initial water ball drop **b** PLS **c** APLS with WPCA **d** APLS with directional derivative

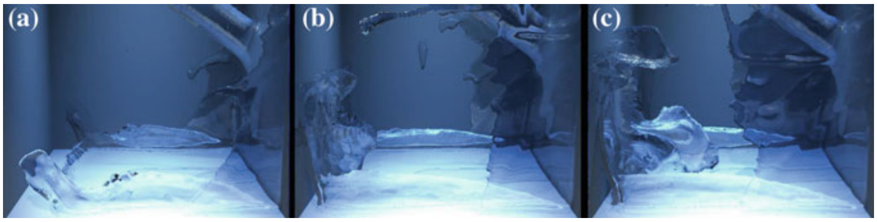


**Fig. 2.34** Water pouring into a tank: **a** 26th frame, **b** 43rd frame, **c** 60th frame

**Table 2.1** Volume and time-consuming for water ball drop

| Frame | PLS | APLS with WPCA | APLS with directional derivative |
|---|---|---|---|
| 1st | 1.15089 (0.6010 s) | 1.15089 (14.0895 s) | 1.15089 (0.6073 s) |
| 7th | 1.15062 (0.6172 s) | 1.15078 (14.3663 s) | 1.15084 (0.6226 s) |
| 21st | 1.1496 (0.7226 s) | 1.14966 (15.8472 s) | 1.14982 (0.7477 s) |
| 70th | 1.13582 (0.8449 s) | 1.15089 (18.5929 s) | 1.14146 (0.8896 s) |
| 100th | 1.12717 (0.0160 s) | 1.15089 (22.6637 s) | 1.13641 (1.0378 s) |

Figure 2.34 shows water pouring into a tank. The stretching feature of the liquid's surface is observed in this physical behavior and visual result. Figure 2.34a shows a water source and it was added in every frame. Figure 2.34b shows water colliding with a wall. The anisotropic particle level-set helps the water surface maintain sharp features. Next, Fig. 2.34c shows the complex surface of the water.

Table 2.1 shows how the volume of fluid in Figs. 2.32 and 2.33 is conserved. Compared to the first frame, the large amount of volume with the PLS in the 70th frame and the 100th frame is lost. But the volume of the fluid is conserved in the third and fourth column. In the 100th frame, it takes 1.0160 s to run the advection module with PLS and 1.0378 s to run the advection module with APLS using directional derivative. As a result, the PLS and the APLS with directional derivative have the similar time-consuming, but the volume of fluid with APLS using directional derivative is more conserved.

### 2.5.6 *Conclusion and Future Work*

This section presented the anisotropic particle level-set method that captures the interface of the multiphase fluid accurately. The anisotropic particle is created by particles' distribution. To get the three axes, the gradient of level-set value and directional derivative are used. As a result, this anisotropic particle level-set method provides more accurate simulation than spherical particle level-set method and faster than APLS with WPCA. However, there is still numerical dissipation into thin feature. Anisotropic escaped particles can be included to handle this problem. They act as splash where numerical dissipation occurred.

## References

1. Adams B, Pauly M, Keiser R, Guibas LJ (2007) Adaptively sampled particle fluids. In: Proceedings of ACM SIGGRAPH 2007. ACM transactions on graphics (TOG), vol 26(3), pp 481–487, July 2007
2. Brackbill JU, Kothe DB, Zemach C (1992) A continuum method for modeling surface tension. J Comput Phys 100(2):335–354
3. Becker M, Teschner M (2007) Weakly compressible SPH for free surface flows. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation, pp 209–217
4. Chorin AJ (1997) A numerical method for solving incompressible viscous flow problems. J Comput Phys 135(2):118–125
5. Carlson M, Mucha PJ, Van Horn II RB Turk G (2002) Melting and flowing. In: ACM SIGGRAPH symposium on computer animation, pp 167–174
6. Cleary PW, Pyo SH, Prakash M, Koo BK (2007) Bubbling and frothing liquids. In: Proceedings of ACM SIGGRAPH 2007. ACM transactions on graphics (TOG), vol 26(3), pp 971–976, July 2007
7. Desbrun M, Gascuel M-P (1996) Smoothed particles: a new paradigm for animating highly deformable bodies. In: Computer animation and simulation'96. pp 61–76
8. Donia T, Gabriel A, Serban, Andreea M, Rada M (2009) Textual entailment as a directional relation. J Res Pract Inf Technol 41(1)
9. de Sousa F, Mangiavacchi N, Nonato L, Castelo A, Tome M, Ferreira V, Cuminato J, Mckee S (2004) A front-tracking/front-capturing method for the simulation of 3d multi-fluid flows with free-surfaces. J Comput Phys 198(2):469–499
10. Durikovic R (2001) Animation of soap bubble dynamics, cluster formation and collision. Computer Graphics Forum (Eurographics 2001 Proceedings), vol 20(3), pp 67–76
11. Dyke MV (1982) An album of fluid motion. The Parabolic Press, Stanford
12. Enright D, Fedkiw R, Ferziger J, Mitchell I (2002) A hybrid particle level set method for improved interface capturing. J Comput Phys 183(1):83–116
13. Enright D, Losasso F, Fedkiw R (2005) A fast and accurate semi-lagrangian particle level set method. Comput Struct 83(6–7):479–490
14. Enright D, Marschner S, Fedkiw R (2002) Animation and rendering of complex water surfaces. In: Proceedings of ACM SIGGRAPH 2002. ACM transactions on graphics (TOG), vol 21(3), pp 736–744, July 2002
15. Fedkiw R, Aslam T, Merriman B, Osher S (1999) A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method). J Comput Phys 152(2):457–492
16. Foster N, Fedkiw R (2001) Practical animation of liquids. In: Proceedings of ACM SIGGRAPH, pp 23–30

17. Fournier A, Habibi, Poulin P (1998) Simulating the flow of liquid droplets. In: Graphics interface'98, June 1998
18. Fattal R, Lischinski D (2004) Target-driven smoke animation. In: Proceedings of Proceedings of ACM SIGGRAPH 2004. ACM transactions on graphics, vol 23(3), pp 441–448, August 2004
19. Foster N, Metaxas D (1996) Realistic animation of liquids. Graph Models Image Process 58(5):471–483
20. Fedkiw R, Stam J, Jensen HW (2001) visual simulation of smoke. In: Proceedings of SIGGRAPH, 15–22
21. Goktekin TG, Bargteil AW, O'brien JF (2004) A method for animating viscoelastic fluids. In: Proceedings of ACM SIGGRAPH 2004. ACM transactions on graphics, vol 23(3), pp 463–468, August 2004
22. Greenwood S, House D (2004) Better with bubbles: enhancing the visual realism of simulated fluid. In: ACM SIGGRAPH/Eurographics symposium on computer animation, pp 287–296
23. Gueyffier D, Li J, Nadim A, Scardoveli R, Zaleski S (1999) Volume-of-fluid interface tracking with smoothed surface stress method for three-dimensional flows. J Comput Phys 152(2):423–456
24. Hu X, Adams N (2006) A multi-phase SPH method for macroscopic and mesoscopic flows. J Comput Phys 213(2):844–861
25. Hong J-M, Kim C-H (2003) Animation of bubbles in liquid. In: Computer Graphics Forum (Eurographics 2003 Proceedings), vol 22(3), pp 253–262, Sept 2003
26. Hong J-M, Kim C-H (2004) Controlling fluid animation with geometric potential. Comput Animat Virtual Worlds 15(3–4):147–157
27. Hong J-M, Kim C-H (2005) Discontinuous fluids. In: Proceedings of ACM SIGGRAPH 2005. ACM transactions on graphics (TOG), vol 24(3), pp 915–920, July 2005
28. Hong J-M, Lee H-Y, Yoon J-C, Kim C-H (2008) Bubbles alive. In: Proceedings of ACM SIGGRAPH 2008. ACM transactions on graphics (TOG), vol 27(3), pp 48:1–48:4, Aug 2008
29. Hirt CW, Nichols BD (1981) Volume of fluid (VOF) method for the dynamics of free boundaries. J Comput Phys 39(1):201–255
30. Homsy GM (1987) Viscous fingering in porous media. Annu Rev Fluid Mech 19:271–311
31. Hong J-M, Shinar T, Fedkiw R (2007) Wrinkled flames and cellular patterns. In: Proceedings of ACM SIGGRAPH 2007. ACM transactions on graphics (TOG), vol 26(3), pp 471–476, July 2007
32. Harlow FH, Welch JE (1965) Numerical calculation of time-dependent viscous incompressible flow of fluid with free surfaces. Phys Fluids 8:2182–2189
33. Ianniello S, Mascio AD (2010) A self-adaptive oriented particles Level-Set method for tracking interfaces. J Comput Phys 229(4):1353–1380
34. Jo E-C, Kim D-Y, Song O-Y (2011) A new SPH fluid simulation method using ellipsoidal kernels. J Vis 14(4):371–379
35. Kim J, Cha D, Chang B, Koo B, Ihm I (2006) Practical animation of turbulent splashing water. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation, pp 335–344
36. Kang M, Fedkiw RP, Liu X-D (2000) A boundary condition capturing method for multiphase incompressible flow. J Sci Comput 15(3):323–360
37. Kass M, Miller G (1990) Rapid, stable fluid dynamics for computer graphics. In: Computer-Graphics (Proceedings of ACMSIGGRAPH'90), vol 24, pp 49–57
38. Kim P-R, Lee H-Y, Kim J-H, Kim C-H (2012) Controlling shapes of air bubbles in a multi-phase fluid simulation. Vis Comput 28(6–8):597–602
39. Kim B, Liu Y, Llamas I, Jiao X, Rossignac J (2007) Simulation of bubbles in foam with the volume control method. In: Proceedings of ACM SIGGRAPH 2007. ACM transactions on graphics (TOG), vol 26(3), pp 98:1–98:9, July 2007
40. Kim B, Liu Y, Llamas I, Rossignac J, Flowfixer (2005) Using BFECC for fluid simulation. In: Proceedings of Eurographics workshop on natural phenomena, pp 51–56

41. Kim B, Liu Y, Llamas I, Rossignac J (2007) Advections with significantly reduced dissipation and diffusion. IEEE Trans Vis Comput Graph 13(1):135–144
42. Kuck H, Vogelgsang C, Greiner G (2002) Simulation and rendering of liquid foams. Graph Interface
43. Kunimatsu A, Watanabe Y, Fujii H, Saito T, Hiwada K, Takahashi T, Ueki H (2001) Fast simulation and rendering techniques for fluid objects. Computer Graphics Forum (Proceedings of Eurographics 2001) 20(3):357–367
44. Lorensen WE, Cline HE (1987) Marching cubes: a high resolution 3D surface construction algorithm. In: Proceedings of ACM SIGGRAPH'87, vol 21(4), pp 163–169, July 1987
45. Liu X-D, Fedkiw RP, Kang M-J (2000) A boundary condition capturing method for poisson's equation on irregular domain. J Comput Phys 172(1):71–98
46. Losasso F, Fedkiw R, Osher S (2005) Spatially adaptive techniques for level set methods and incompressible flow. Comput Fluids
47. Losasso F, Gibou F, Fedkiw R (2004) Simulating water and smoke with an octree data structure. In: Proceedings of ACM SIGGRAPH 2004. ACM Transactions on Graphics (TOG), vol 23(3), pp 457–462, Aug 2004
48. Lee H-Y, Hong J-M, Kim c-h (2005) Interchangeable SPH and level set method in multiphase fluids. Vis Comput 25:713–718
49. Losasso F, Irving G, Guendelman E, Fedkiw R (2006) Melting and burning solids into liquids and gases. IEEE Trans Vis Comput Graph 12(3):343–352
50. Liu MB, Liu g r, Lam KY (2006) Adaptive smoothed particle hydrodynamics for high strain hydrodynamics with material strength. Shock Waves 15(1):21–29
51. Losasso f, Shinar t, Selle A, Fedkiw R (2006) Multiple interacting liquids. In: Proceedings of ACM SIGGRAPH 2006. ACM transactions on graphics (TOG), vol 25(3), pp 812–819, July 2006
52. Losasso f, Talton j, Kwatra n, Fedkiw R (2008) Two-way coupled SPH and particle level set fluid simulation. IEEE Trans Vis Comput Graph 14(4):797–804
53. Magnaudet J, Eames I (2000) The motion of high reynolds-number bubbles in inhomogeneous flows. Annu Rev Fluid Mech 32:659–708
54. Mcnamara A, Treuille A, Popovic Z, Stam J (2004) Fluid control using the adjoint method. In: Proceedings of ACM SIGGRAPH 2004. ACM transactions on graphics (TOG), vol 23(3), pp 449–456, Aug 2004
55. Mendelson HD (1967) The prediction of bubble terminal velocities from wave theory. Adv Chem Eng J 13(2):250–253
56. Mihalef V, Metaxas D, Sussman M (2007) Textured liquids based on the marker level set. Comput Graph Forum 26(3):457–466
57. Mihalef V, Unlusu B, Metaxas D, Sussman M, Hussaini MY (2006) Physics based boiling simulation. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation 1:317–324
58. Mullen P, Mckenzie A, Tong Y, Desbrun M,A (2007) Variational approach to eulerian geometry processing. In: Proceedings of ACM SIGGRAPH 2007. ACM transactions on graphics (TOG), vol 26(3), pp 66:1–66:10, July 2007
59. Müller M, Charypar D, Gross M (2003) Particle based fluid simulation for interactive applications. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation pp 154–159
60. Müller M, Solenthaler B, Keiser R, Gross M (2005) Particle-based fluid-fluid interaction. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation, pp 237–244
61. Nguyen D, Fedkiw R, Jensen H (2002) Physically based modeling and animation of fire. In: Proceedings of ACM SIGGRAPH 200. ACM transactions on graphics (TOG), vol 21(3), pp 721–728, July 2002
62. O'Brien J, Hodgins J (1995) Dynamic simulation of splashing fluids. Proc Comput Animat 95:198–205

63. Osher S, Fedkiw R (2002) Level set methods and dynamic implicit surfaces. Springer, New York
64. Park J, Kim Y, Wi D, Kang N, Shin SY, Noh J (2008) A unified handling of immiscible and miscible fluids. Comput Animat Virtual Worlds 19(3–4):455–467
65. Premoze S, Tasdizen T, Bigler J, Lefohn A, Whitaker R (2003) Particle-based simulation of fluids. In: Computer Graphics Forum (Eurographics Proceedings), vol 22, pp 401–410
66. Rasmussen N, Enright D, Nguyen D, Marino S, Sumner N, Geiger W, Hoon S, Fedkiw R (2004) Directable photorealistic liquids. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation, pp 193–202
67. Rider WJ, Kothe DB (April 1998) Reconstructing volume tracking. J Comput Phys 141(2):112–152
68. Rahman A, Manzur M (2008) Dynamic texture synthesis using motion distribution statistics. J Res Pract Inf Technol 40(2):129–148
69. SAAD Y (1996) Iterative methods for sparse linear systems. PWS Publishing
70. Schechter H, Bridson R (2008) Evolving sub-grid turbulence for smoke animation. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation, pp 1–7
71. Son G, Dhir VK, Ramanujapu N (1999) Dynamics and heat transfer associated with a single bubble during nucleate boiling on a horizontal surface. J Heat Transf 121(3):623–631
72. Shin S, Juric D (2002) Three-dimensional multiphase flow using a level contour reconstruction method for front tracking without connectivity. J Comput Phys 180(2):427–470
73. Shin S-H, KIM C-H (2007) Target-driven liquid animation with interfacial discontinuities. Comput Animat Virtual Worlds 18(4–5):447–453
74. Shew WL, Pinton J-F (2006) Dynamical model of bubble path instability. Phys Rev Lett 97:144508
75. Selle A, Rasmussen N, Fedkiw R (2005) A vortex particle method for smoke, water and explosions. In: Proceedings of ACM SIGGRAPH 2005. ACM transactions on graphics (TOG), vol 24(3), pp 910–914, July 2005
76. Song O-Y, Shin H-C, Ko H-S (2005) Stable but non-dissipative water. ACM Trans Graph 24(1):81–97
77. Stam J (1999) Stable fluids. In: Proceedings of ACM SIGGRAPH, pp 121–128
78. Sussman M (2003) A second order coupled level set and volume of-fluid method for computing growth and collapse of vapor bubbles. J Comput Phys 187(1):110–136
79. Torres DJ, Brackbill JU (2000) The point-set method: front-tracking without connectivity. J Comput Phys 165(2):620–644
80. Tryggvason G, Bunner B, Esmaeeli A, Juric D, Al-Rawashi N, Tauber W, Han J, Nas S, Jan Y-J (2001) A front tracking method for the computations of multiphase flow. J Comput Phys 169(2):708–759
81. Takahashi T, Fujii H, Kunimatsu A, Hiwada K, Saito T, Tanaka K, Ueki H (2003) Realistic animation of fluid with splash and foam. Computer Graphics Forum (In: Proceedings of Eurographics 2003), vol 22(3), pp 391–400, Sept 2003
82. Thüerey N (2007) Physically based animation of free surface flows with the lattice Boltzmann method. Ph.D. thesis, University of Erlangen-Nuremberg
83. Treuille A, Mcnamara A, Popović Z, Stam J (2003) Keyframe control of smoke simulations. In: Proceedings of ACM SIGGRAPH 2003. ACM transactions on graphics (TOG), vol 22(3), pp 716–723, July 2003
84. Thüerey N, Sadlo F, Schirm S, Müller-Fischer M, Gross, M (2007) Real-time simulations of bubbles and foam within a shallow water framework. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation, pp 191–198
85. Unverdi S, Tryggvason G (1992) A front tracking method for viscous, incompressible, multi-fluid flows. J Comput Phys 100(1):25–37
86. Wojtan C, Carlson M, Muycha PJ, Turk G (2007) Animating corrosion and erosion. Eurographics workshop on natural phenomena, pp 15–22
87. Wojtan C, Thürey N, Gross M, Turk G (2009) Deforming meshes that split and merge. ACM Trans Graph 28(3)

88. Wojtan C, Thürey N, Gross M, Turk G (2010) Physics-inspired topology changes for thin fluid features. In: Proceedings of ACM SIGGRAPH 2010. ACM transactions on graphics (TOG), vol 29(4), July 2010
89. Yu Y, Ho Jung, Cho H (1999) A new water droplet model using metaball in the gravitational field. Comput Graph 23(2):213–222
90. Yu J-H, Turk G (2010) Reconstructing surfaces of particle-based fluids using anisotropic kernels. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation, p 217
91. Zhu H, Liu X, Liu Y, Wu E (2006) Simulation of miscible binary mixtures based on lattice Boltzmann method: research articles. Comput Animat Virtual Worlds 17(3–4):403–410
92. Zheng Z, Yang J, Zhu Y (2006) Face detection and recognition using colour sequential images. J Res Pract Inf Technol 38(2)
93. http://developer.nvidia.com/

# Chapter 3
# Smoke

## 3.1 Animating Smoke with Dynamic Balance

**Abstract** This section proposes a numerical method for maintaining a dynamic rolling motion of animated gaseous phenomena, such as smoke, that avoids dissipation due to numerical error. It compensates for the errors induced by a semi-Lagrangian scheme using an error estimate for each time interval. And it develops a new advection term and perform vortex advection based on a vorticity confinement force. Example simulations show that this method is able to keep smoke features alive, even near the center of a vortex.

### *3.1.1 Introduction*

The simulation of natural phenomena such as smoke, fire, and water is a challenging problem in computer graphics. Gaseous phenomena are popularly used in movies and video games for special effects. An ideal computer graphics model of smoke is what is easy to use and generates stable results. To achieve this, we need to develop a scheme which is easy to implement and in which the numerical error caused by discretization is as small as possible.

Many researchers in the field of computer graphics have used physically based techniques to create a realistic gaseous motion that is self-perpetuating. We want to create a model that is close to realistic, but avoids dissipation, which is especially important to represent a feature of smoke such as a vortex. In previous work toward this goal, an artificial external force was used to amplify the existing effect of the simulation results. Although this achieves a plausible appearance, it leaks accuracy. This section introduces a new solution that fundamentally improves the accuracy of the simulation, without including much additional computation.

In computer graphics, fluids are simulated by updating a field of substance data, such as density, temperature, and velocity. Beyond some threshold conditions, a fluid simulation may produce unexpected results due to the occurrence of an unstable state. Our new method for animating smoke avoids this problem by using a *dynamic balance* to maintain the coherence of the field. To preserve the properties of the

C.-H. Kim et al., *Real-Time Visual Effects for Game Programming*,
Gaming Media and Social Effects, DOI 10.1007/978-981-287-487-0_3

initial flow, it is important that the simulation is achieved by gradual changes to the vector fields representing quantities such as position and velocity. Our approach includes compensation for losses in the energy of the velocity field by an advection step. We arrange for the substance field to move along a velocity field, which can be thought of as a framework of movement. Compensating for errors in the velocity field contributes to the coherence of the flow and creates a stable simulation. In this section, we aim to produce Karman vortex motions, as shown in Fig. 3.1, which are created by turbulent smoke moving over an obstacle. A separated vortex results from the difference in viscosity at the interface. Figure 3.1 shows the simulation of vortex street [39]. In simulating of this kind of smoke motion, it is important to maintain the momentum of the vortex [31]. Inspired by the work of Fedkiw et al. [7]. we focus on maintaining the vorticity, which is a feature of smoke that occurs within the velocity field. We represent a vortex as a substance field, and therefore the vorticity confinement force moves along the velocity field in the same way as density and temperature. In this way, we separate the vorticity field from the main velocity field and create a new advection term which conserves vorticity.

In the new advection step, we estimate the error during each time interval and compensate for it. This process reduces the numerical dissipation which necessarily results from the linear interpolation of a semi-Lagrangian scheme. The use of vortex advection allows us to generate a smoke model with a unique type of vortex, compared to previous smoke models, and implicitly maintain the momentum of the vortex, rather than requiring an external force to sustain it.

The specific contributions of this work are: (1) Improving the method for solving the differential equation for the advection step by using error compensation; and (2) allowing the smoke model to remain dynamic near the center of a vortex by the use of vortex advection. Our method is fully Eulerian and so it is simple and easy for implementation.

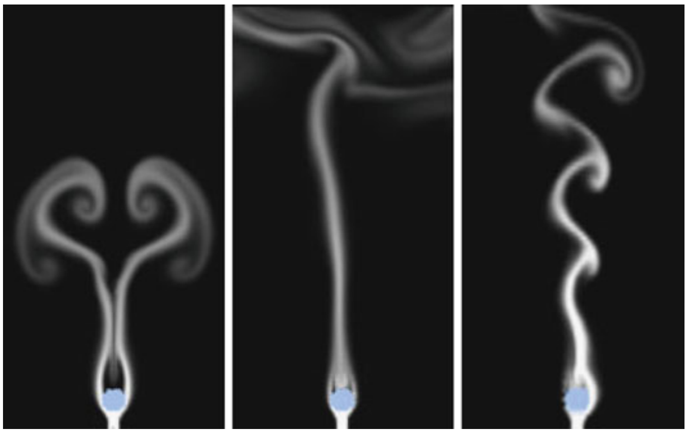


**Fig. 3.1** Karman vortex

Section 3.1.2 addresses related works. Section 3.1.3 describes our scheme on Computing Errors in the Advection Term and, in Sect. 3.1.4, on Vortex Advection Based on Vorticity Confinement. Section 3.1.5 describes the implementation of our method; then Sect. 3.1.6 presents experimental results and discussion. And we conclude in Sect. 3.1.7.

### *3.1.2 Related Work*

In the field of computer graphics, there has been a lot of research on simulating natural phenomena such as smoke, water, and fire. Foster and Metaxas [12] were the first to use the Navier-Stokes equation to simulate fluid flow. Recent results have received attention because they can automatically generate passive motion [6, 8, 24, 35]. These physics-based techniques are derived from the computational fluid dynamics (CFD) literature [1], and some of them have the advantage of providing higher precision by the use of lower-order polynomials. This is very helpful for simulating fluid flows which demonstrate complicated phenomena such as vortex and turbulence [13]. Fluid simulation can be achieved by computing an approximate solution to a partial differential equation (PDE) expressing a local function that represents the fluid flow. Physically based techniques allow the user to create the appearance of smoke automatically [6, 16, 38].

While these techniques are powerful, they have unavoidable errors associated with a numerical solution of a differential equation. For that reason, developing an adaptive method for reducing numerical errors is a significant problem [4, 19].

There are a lot of works related to this problem in computer graphics. Fattal and Lischinski [6] forced the fluid to form a user-defined shape, and they use a density error term to reverse the diffusion process so as to match a user-defined shape. Hong and Kim [16] maintained the density of smoke by keeping the total mass of particles constant during a simulation. These approaches are attractive, but they considered the density errors only.

Some research has focused more closely on velocity. Fedkiw et al. [7] kept alive small-scale features of smoke by applying a vorticity confinement term to the inviscid Euler equation. This allows a vortex to persist but, despite the exclusion of viscosity, they could not counteract severe dissipation of the smoke over time.

Other research has focused on maintaining a given property at its initial level. Dupont and Liu [5] used backward and forward forms of the Euler equation to determine the extent of the advection error. Their approach updates the level-set function over time to obtain the zero-set more accurately, and reduces the numerical error during topological changes. Our method is similar in that it estimates the amount of advection error and compensates for it during the simulation. Kim et al. [20] independently introduced BFECC to computer graphics recently. Song et al. [36] used CIP method to prevent dissipation of small-scale features of multiphase fluid. By the calculation of derivative values at the grid points, they obtained the less dissipative fluid motion at each time step. But this technique is computationally expensive compared to ours.

Whatever the underlying method, a major problem is reducing the dissipative error, which is the focus of this section. Because a vortex is one of the elements that makes smoke look realistic, we concentrate on handling the vortex effect. Recently, some research has focused on vortex effects as fluid features [2, 28, 33]. They used vortex particles based on the Lagrangian to alleviate dissipation within a method based on an Eulerian grid. On the other hand, we make good use of fully Eulerian method which is simple and easy to implement. To represent the effects of viscosity, we use the viscous Navier-Stokes equation for simulating fluid motion. We do not attempt to achieve all the features of smoke, but our results show that we can preserve a smoke vortex from the effects of numerical dissipation.

## 3.1.3 Computing Errors in the Advection Term

This section presents a method for error compensation using the Navier-Stokes equation.

### 3.1.3.1 The Equations of Flow

We assume a viscous gas, and therefore use the Navier-Stokes equation for simulating flow. We denote the velocity vector field as $\mathbf{u} = (u, v, w)$, and then the Navier-Stokes equation is:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{\nabla p}{\rho} + \nu \nabla^2 \mathbf{u} + \mathbf{f} \tag{3.1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{3.2}$$

where $\mathbf{f}$ accounts for external forces such as gravity and buoyancy, $p$ is the pressure in the velocity field, and $\rho$ is the density of the gas. Equations (3.1) and (3.2) model the conservation of momentum and of mass, respectively. They can be found in any standard text [14]. To make it easier to understand how these equations are solved, we consider a standard fluid simulation process:

$$M \circ A_\rho \circ P \circ D \circ A_\mathbf{v} \circ F$$

The term $F$ applies forces to the velocity field; $A_\mathbf{v}$ is the self-advection of the velocity vector field; $D$ is a diffusion step; $P$ projects the resulting field so that it is divergence-free; $A_\rho$ advects the density along the velocity vector field; and $M$ is a mass conservation step to counteract dissipation.

In the simulation, step $A_{\mathbf{v}}$ corresponds to the first term of right-hand side of Eq. (3.1):

$$A_{\mathbf{v}} = -(\mathbf{u} \cdot \nabla)\mathbf{u} \tag{3.3}$$

### 3.1.3.2 Error Compensation Scheme

To solve Eq. (3.3), which is nonlinear, we use a semi-Lagrangian scheme with error compensation that considers the time intervals before and after the advection step.

Let us denote some position in a smoke simulation by $\mathbf{x} = (x, y)$. To determine the velocity at time $t + 1$, we use the velocity at time $t$. We describe the state of a simulation using the Euler method at time $t$ as a function $\varphi_t(\mathbf{x})$. We have defined $\varphi$ as a function of both $t$ and $\mathbf{x}$, but this function depends not only on previous values but also on the subsequent state.

To calculate the forward update step, we describe $\mathscr{F}$, including the component $\varphi_t(\mathbf{x})$, which corresponds to the Euler equation at time $t$. This is used to determine the state of the simulation at time $t + 1$:

$$\tilde{\varphi}_{t+1}(\mathbf{x}) = \mathscr{F}(\varphi_t(\mathbf{x})) \tag{3.4}$$

And we represent the backward advection step as $\mathscr{B}$:

$$\hat{\varphi}_t(\mathbf{x}) = \mathscr{B}(\tilde{\varphi}_{t+1}(\mathbf{x})) \tag{3.5}$$

In an ideal situation, $\varphi_t(\mathbf{x})$ is equal to $\hat{\varphi}_t(\mathbf{x})$; but in practice, we cannot achieve this result because of numerical dissipation, as shown in Fig. 3.2a.

We want to minimize the error associated with the difference between $\varphi_t(\mathbf{x})$ and $\hat{\varphi}_t(\mathbf{x})$. This error $\mathbf{e}(vecx)$ can be found from the approximate solution of the Euler equation:

$$\mathbf{e}(\mathbf{x}) = \varphi_t(\mathbf{x}) - \hat{\varphi}_t(\mathbf{x}) \tag{3.6}$$

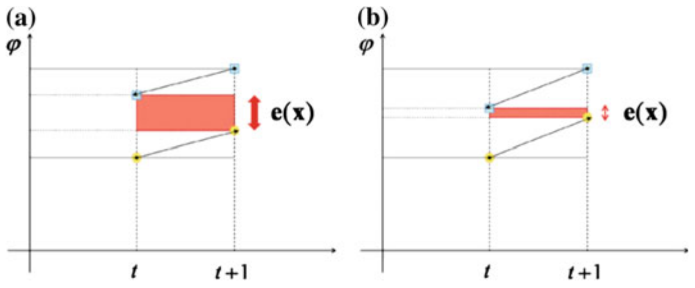


**Fig. 3.2** Estimated errors before and after error compensation: **a** before error compensation; **b** after error compensation

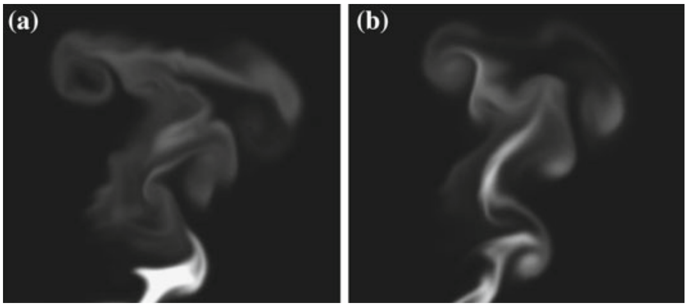


**Fig. 3.3** Simulated smoke: **a** without error compensation, leading to an irregularly scattered appearance; **b** with error compensation, showing a better approximation of the effect of the force on the field

We assume that this error occurs while moving back and forth in simulated time. We expect that the loss of energy in the forward advection step will be about $\frac{1}{2}\mathbf{e}(\mathbf{x})$. We use this observation to reduce the dissipation resulting from the numerical solution. We do this by injecting the missing energy from the velocity vector field before the advection step:

$$\varphi_t^{new}(\mathbf{x}) = \varphi_t(\mathbf{x}) + \frac{1}{2}\mathbf{e}(\mathbf{x}) \tag{3.7}$$

Now we use this value to determine the velocity at time $t + 1$. Figure 3.2b shows how this reduces the error after updating using $\varphi_t^{new}(\mathbf{x})$. This process can be applied to the density field in the same way. Our error compensation algorithm has the same accuracy as the second-order improved Euler scheme. This backward compensation can be replaced by forward correction because they are duals of each other. See Dupont and Liu [5] for more details.

Each image in Fig. 3.3 shows the plane $Z = 30$ at one moment in a 3D smoke simulation on a $200 \times 200 \times 60$ grid. The scattered appearance of the smoke in Fig. 3.3a results from numerical dissipation, rather than any change in momentum of the vortex resulting from the effect of viscosity. Figure 3.3b shows now the error compensation term reduces the dissipation of the fluid motion and allows characteristic smoke features to develop over time.

### *3.1.4 Vortex Advection Based on Vorticity Confinement*

Using vortex advection based on vorticity confinement, this section proposes modified equations for developing a fluid simulation with a continuous vortex.

#### 3.1.4.1 Vorticity Confinement

A vorticity confinement term generates forces that sustain a smoke feature such as a swirling motion. The motion of the smoke is represented by a velocity field $\mathbf{u}$. Let us denote vorticity as follows:

$$\omega = \nabla \times \mathbf{u} \tag{3.8}$$

where $\omega$ is the vorticity, which is the axis around which a velocity field spins. It has $x$ and $y$ components which can be computed as the curl of velocity. We can now obtain a normalized vector field $\mathbf{N}$:

$$\mathbf{N} = \frac{\nabla|\omega|}{|\nabla|\omega||} \tag{3.9}$$

From Eqs. (3.8) and (3.9), we can determine the direction and magnitude of the force required to spin a vector field forward through one time step. Then, we denote the vorticity confinement force as follows:

$$\mathbf{u_c} = \varepsilon h(\mathbf{N} \times \omega) \tag{3.10}$$

This provides a force which enforces the rolling feature of smoke. The constant $\varepsilon$ is used to control the confinement force.

#### 3.1.4.2 Vortex Advection Scheme

Two important properties of dynamic balance are: (i) It has an advection step with error compensation; and (ii) it uses vortex advection.

We assume that a vortex advects along a velocity vector field. We describe the evolution of the smoke feature flow in terms of the vorticity field, rather than in terms of the velocity field. We separate the vorticity field from the main velocity field and create a new advection term which conserves vorticity. To describe this, we substitute $\mathbf{u_c}$ for $\mathbf{u}$ in Eq. (3.3).
Then, the advection term associated with $\mathbf{u_c}$ is

$$A_\mathbf{c} = -(\mathbf{u_c} \cdot \nabla)\mathbf{u_c} \tag{3.11}$$

We add $A_\mathbf{c}$ to the velocity vector field $A_\mathbf{v}$ for the advection step, and use this term to create a new advection term $A_\mathbf{new}$:

$$A_\mathbf{new} = A_\mathbf{v} + A_\mathbf{c} \tag{3.12}$$

We apply the solution of this equation to the Navier-Stokes equation in a new advection step, and the modified fluid simulation process becomes:

$$M \circ A_\rho \circ P \circ D \circ A_\mathbf{new} \circ F$$

To achieve a dynamic balance of the velocity vector field, we advect the term $\mathbf{u_c}$ of Eq. (3.10) along the vector field with error compensation. Since the new simulation process is still based on the standard fluid flow equation, we can guarantee that this process is physically stable.

Results from the modified simulation scheme are shown in Results and Discussion.

## 3.1.5 Implementation

We will now describe the application of our method to a standard fluid simulation in which Eqs. (3.1) and (3.2) are implemented using a standard computer graphics approach [38].

### 3.1.5.1 Simulation Steps

The simulation process is represented as pseudo-code in Table 3.1.

We subdivide space into voxels and define values such as velocity and density in the center of each voxel. In the first stage, we add external forces such as gravity and buoyancy to the initial velocity field, then determine the vorticity confinement force, $\mathbf{v_c}$, from the velocity field. This makes the velocity field representing the smoke to spin. Before we apply the advection step to this vector field, we need to create an additional vector field which includes error compensation. This involves copying the vector field and then advecting the substance field along the compensated field, and using the result to update the vector field on which we are working.

**Table 3.1** Pseudo-code for fluid simulation

```
Pseudo-code
$\mathbf{f} \leftarrow$ FORCE($\mathbf{u}$)
$\mathbf{v} \leftarrow \mathbf{v} + \mathbf{f}$
$\mathbf{v_c} \leftarrow$ VORTICITY($\mathbf{v}$)
while ( SIMULATING )
{
    $\mathbf{v_{new}} \leftarrow$ COMPENSATE($\mathbf{v}$)
    $\mathbf{v_{c\_new}} \leftarrow$ COMPENSATE($\mathbf{v_c}$)
    $\mathbf{v_{new}} \leftarrow$ ADVECT($\mathbf{v}, \mathbf{v_{new}}$)
    $\mathbf{v_{c\_new}} \leftarrow$ ADVECT($\mathbf{v_c}, \mathbf{v_{c\_new}}$)
    $\mathbf{v} \leftarrow \mathbf{v_{new}}$
    $\mathbf{v_c} \leftarrow \mathbf{v_{c\_new}}$
}
$\mathbf{v} \leftarrow$ FORCE($\mathbf{v_c}$)
$\mathbf{v} \leftarrow$ DIFFUSE($\mathbf{v}$)
$\mathbf{v} \leftarrow$ PROJECT($\mathbf{v}$)
$\rho \leftarrow$ ADVECT($\mathbf{v}$,$\rho$)
$\rho \leftarrow$ PRESERVEMASS($\rho$)
return ($\mathbf{u}$)
```

Next, we add a vorticity field to the updated velocity field, which gives us the vector field to be used in the advection step. This is followed by a diffusion process which models the effect of viscosity. To get the pressures across the velocity field, we perform a projection using a Poisson solver and to solve a linear system, we use a conjugate gradient method [29]. Next, we advect the density field along an updated velocity field, and finally process that density field to ensure that mass is conserved.

At every time step, our simulator outputs a grid that contains the density, $\rho$, which is then used for rendering the smoke. The 2D and 3D simulation results were rendered by OpenGL and Mental Ray for MAYA.

### 3.1.6 Results and Discussion

Simulations were implemented in 2D and in 3D using the C programming language. All the experiments were performed on a Pentium IV CPU 3.2 GHz with 1 GB RAM running Windows, and a NVIDIA GeForce4 Ti 4400 graphics card with 128 MB of video RAM.

To simulate the features of smoke, a velocity field was set up proportional to density and a buoyancy field and a density source were applied as initial conditions. Figures 3.4, 3.5, and 3.6 show the difference made by the error compensation field under experimental conditions. Simulations are started with short of source to show the effect of the vortex clearly. Figure 3.4 shows results on a $200 \times 200$ grid generated using the standard flow equation without a vorticity confinement term. This simulation took less than 1 min. In Fig. 3.4, small-scale smoke features in a viscous fluid dissipate over time. In Fig. 3.5, the initial conditions are the same as those for Fig. 3.4, but a vorticity confinement term is added to the velocity field before the advection step. Figure 3.5 shows a better defined swirling motion than Fig. 3.4, but

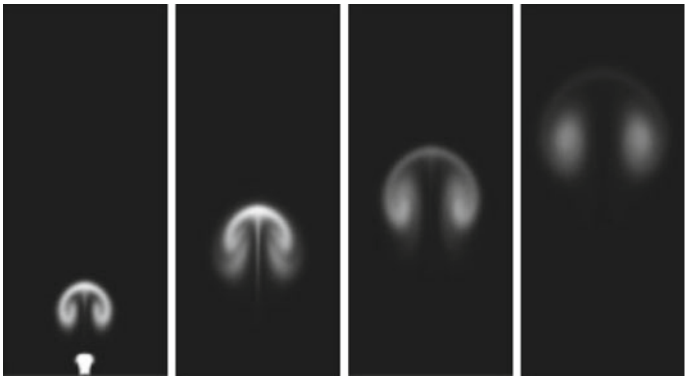


**Fig. 3.4** Simulation without a vorticity confinement term

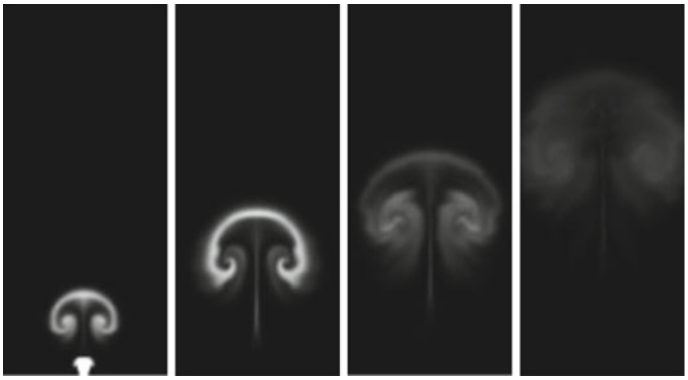


**Fig. 3.5** Simulation with a vorticity confinement term

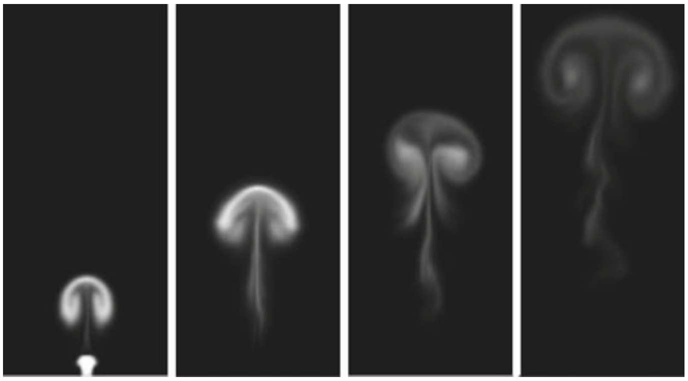


**Fig. 3.6** Simulation with both a vorticity confinement term and vortex advection

the vortex is lost quickly. Indeed, the spinning axis eventually disappears at a position near the center of the vortex, as can be seen in Fig. 3.5d.

Figure 3.6 shows how this problem was resolved using vortex advection. Now the smoke keeps spinning and motion features become visible near the center of the smoke. Notice that the smoke in Fig. 3.6d remains closer to the center of the vortex than in Fig. 3.5d. Even though Fig. 3.5c shows more swirling than Fig. 3.6c, it does not persist.

Figure 3.7 shows frames from another simulation on a $200 \times 200$ grid. Turbulent interaction between the smoke and solid objects creates a swirling motion. The top row of images was generated using an advection step with a vorticity confinement force and vortex advection disabled. This seems to be close to a natural behavior with scattering, but our aim is a rolling motion near the center of the vortex without scattering. The bottom row of images was generated using vortex advection based on a vorticity confinement force. This shows the properties of the smoke being maintained at a position near the center of the vortex. As the confinement force $\varepsilon$

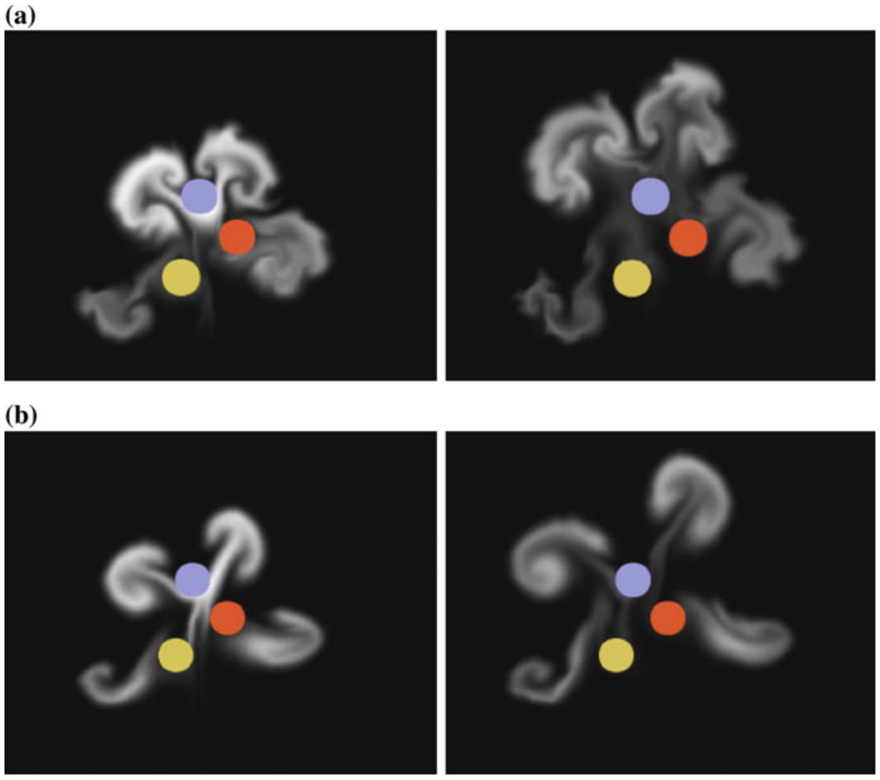


**Fig. 3.7** Smoke motion without (**a**) and with (**b**) vortex advection, created using a velocity field with error compensation. The separated vortex of **a** shows rolling motion which becomes scattered; but in **b** the center of the vortex is retained

increases, the difference between the two cases become clear, as the top row shows heavy scattering.

Figures 3.8 and 3.9 show smoke motion with an advection step with error compensation. These frames demonstrate how smoke, simulated on a $200 \times 200 \times 60$ grid using our method, retains its properties near the center of the vortex.

## 3.1.7 Conclusion

This section has proposed a new method for persistent modeling of the unique features of smoke such as vortices. Using the method, we have generated animations of rolling of smoke without scattering, as shown in Figs. 3.8 and 3.9. Many interesting simulations of real smoke effects [39] could be created using this technique. It will be effective for fluid simulation, in which there is an interaction between an object and

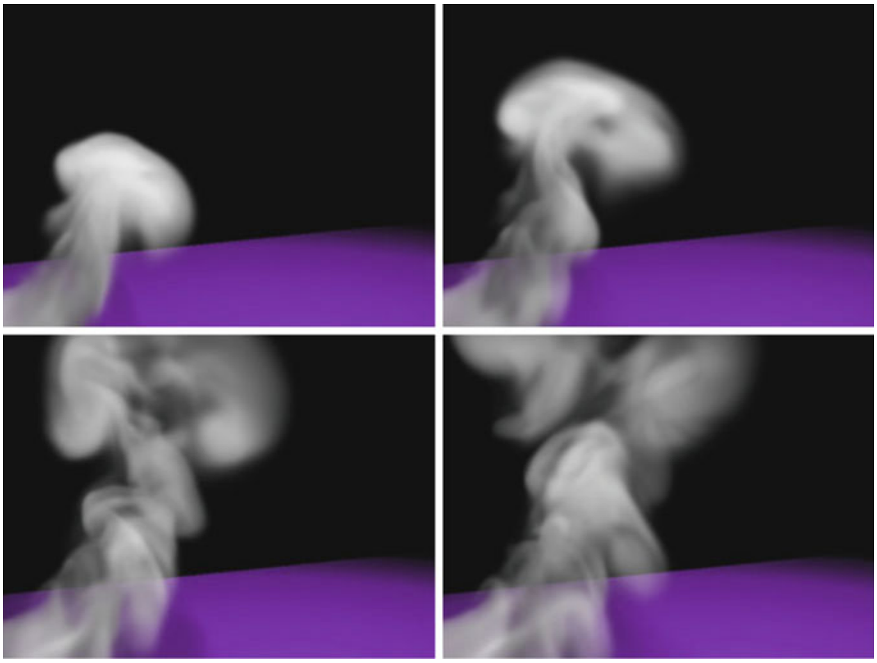


**Fig. 3.8** Smoke motion with vortex advection and error compensation

the fluid, and could also be used for liquids or multiphase fluids [15, 18]. Synergy with the particle method can be explored to preserve properties such as vorticity and density. And also, an animator for advanced effects with a GUI could be provided.

## 3.2 Procedural Synthesis Using Vortex Particle Method for Fluid Simulation

**Abstract** This section proposes a fast and effective technique to improve subgrid visual details of the grid based fluid simulation. It procedurally synthesizes the flow fields coming from the incompressible Navier-Stokes solver and the vorticity fields generated by vortex particle method for subgrid turbulence. This enables to efficiently animate smoke which is highly turbulent and swirling with small-scale details. Since this technique does not solve the linear system in high-resolution grids, it can perform fluid simulation more rapidly. The influence of turbulent and swirling effect to the fluid flow can be easily estimated.

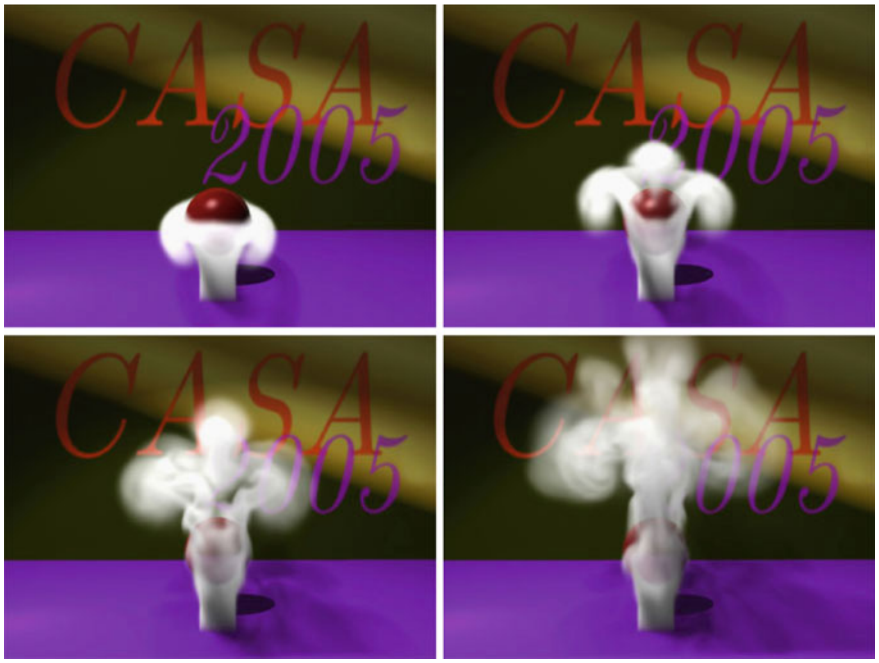


**Fig. 3.9** Rising smoke swirling over a sphere, while the vortex is preserved

### *3.2.1 Introduction*

As the use of computer graphics is increasing in movies, commercials, and other media, the work of animating fluids such as explosion and fire has become an important process in that area.

Recently, fluid dynamics technologies are applied to a range of special effects, yet achieving precise calculations is still a time-consuming process, which is why various technologies are being studied for better balance between productivity and the reality of graphics. Therefore, in the area of graphics, enhancing productivity while maintaining an adequate level of visual quality—even at the modest expense of physical/mathematical accuracy—presents a challenge to fluid animation technologies.

While the vortex particle method [33] is useful for generating turbulent effects, its drawback is that the grid is too coarse to accommodate the vorticity of all particles. Although it is possible to use high-resolution grids in simulation, it would require much more time and memory capacity. Recently introduced techniques [23, 26, 32] have improved details in grids by using noise.

This section proposes a new technique for improving details in grids in fluid simulation. This technique depicts the uncomplicated motion of fluid through flow fields calculated by solving the incompressible Navier-Stokes equations, and detailed mo tion of fluid through vorticity field calculated by the vortex particle method. This technique does not solve the linear system in high-resolution grids, thus it performs large-scale grid simulation efficiently. Another benefit is that it generates turbulent effects of explosion or rough motion of smoke as the vorticity of vortex particles is transferred to the high-resolution grid.

This approach has the following characteristics:

- A degree of resolution equivalent to the result of simulation using the $(k \times n)^3$ grid is achieved by solving fluid equations of the $n^3$ grid in a three-dimensional space. $k$ is a constant making desired resolution.
- Since the result can be quickly drawn through low-resolution simulation, if the result is satisfactory, a high-resolution result can be obtained without running simulation with adjusted parameters.
- Highly turbulent and swirling effects can be achieved that cannot be generated easily by using noise for simulating explosion, smoke, etc. since it is hard to anticipate how they will affect flow when using noise.
- Vortex particle carries the vorticity calculated from low-resolution and high-resolution vorticity field is created using the vorticity. This field uses the information from the base simulation.
- We can easily control the size of turbulence depending on the frequency by adjusting the particle radius.

### *3.2.2 Previous Work*

Stam and Fiume [37] proposed a method to depict gaseous phenomena through the ambient turbulence that is used to add new eddies by noise. Smoke simulation using the three-dimensional Navier-Stokes equations began with Foster and Metaxas [13]. Stam [38] presented a Semi-Lagrangian advection model which allows stable simulation with large time step. Fedkiw et al. [7] introduced a simulation model for generating a vorticity effect which is hard to depict in coarse grids. This vorticity confinement method reduced the loss of small-scale details and increased the amount of swirling motion. Felici and Drela [9–11] coupled an Eulerian and a Lagrangian Solver and tried to reduce numerical diffusion. Selle et al. [33] presented vortex particle method which allowed effective simulation by incorporating localized vorticity confinement in grid. It directly computes the velocity for particles by trilinear interpolation. It reduces the numerical loss because each vortex particle stores a vorticity value. But it is slow for high-resolution simulation. And Park and Kim [28] presented a Lagrangian method for gaseous phenomena simulation based on the vortex method. Hong and Kim [17] proposed the method for vortex advection based on vorticity confinement. Angelidis et al. [3] used the vortex filament methods based on the vorticity formulation of the Navier-Stokes equations. They defined a vorticity preserving flow field around a set of vortex primitives.

Previous research results were only useful in small-scale simulation due to the high computational cost. Although the use of fluid technologies is increasing, the improvement of computing environment still falls short of the demand of developers, which has led to a number of attempts to find ways to enhance the quality as well as reality of simulation. Neyret [27] increased the details of fluid by advecting texture to the velocity field. Rasmussen et al. [30] proposed an efficient method for large-scale fluid simulation, and Losasso et al. [24] introduced a method to reduce simulation time by limiting the refinement to the fluid surface part of the grid using octree data structure instead of using uniform grid. Another approach is to employ error correction schemes such as BFECC [21] or a Mac-Cormack method [34], or less dissipative advection schemes such as USCIP [22], to reduce the diffusion in the numerical method directly.

Recently, methods for adding further details in subgrid by using noise have been introduced. These methods are similar to our method's schematic outline, adding the small-scaled detail to the upsampled field gained by interpolation in coarse-grid simulation for making a new velocity field. Kim et al. [23] used wavelet method and synthesized missing turbulent flow components with band-limited wavelet noise. This helps to show small-scaled detail in a coarse grid. Schechter and Bridson [32] presented subgrid turbulence evolution model for fluid simulation. They tracked bands of turbulent energy using a simple linear model and created the turbulent velocity using flow noise. They added a predictor step to the usual time splitting of the incompressible Euler equation and corrected the additional vorticity dissipation due to time splitting of the pressure and advection. Narain et al. [26] introduced a technique coupling a procedural turbulence model with a numerical fluid solver. They used an energy function and a Lagrangian approach to advect noise when synthesizing an incompressible turbulent velocity field. They applied the method to simulation of liquids with free surfaces.

Turbulence can be edited easily using these methods because the turbulence is independent from the large-scale flow and able to be added like a post process. But, in our method, we do not include a noise and energy function but vortex particle method, i.e., vortex particle carries the vorticity calculated from low-resolution and the high-resolution vorticity field is created using the vorticity. This field uses the information from the base simulation while the noise addition is unrelated to the base simulation. In addition, our method is easy to implement and able to show highly turbulent effects such as an explosion or rocket smoke that is hard to be generated from a noise function.

### *3.2.3 High-Resolution Fluid Synthesis*

This section discusses the high-resolution method of synthesizing vortex particles in fluid simulation (Fig. 3.10). The notation used is as follows. Bold denotes a vector, and nonbold represents a scalar. Parameters used in low-resolution simulation are marked with lower-case letters and parameters used in high-resolution simulation are marked with upper-case letters, while $\mathbf{x}$ represents the location. $I(\mathbf{u}, \mathbf{X})$ is the

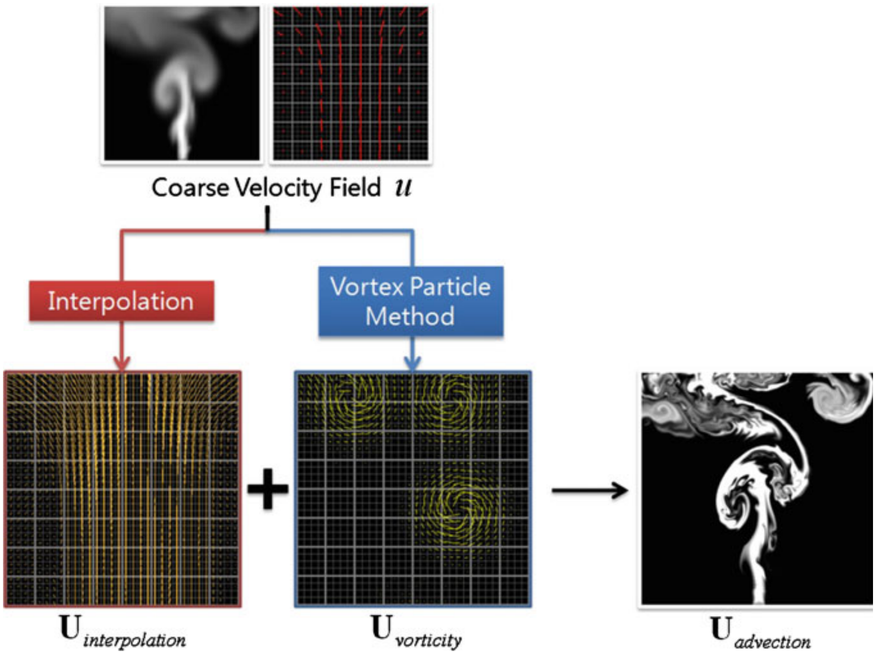


**Fig. 3.10** Overview of our method

function for interpolation of velocity field $\mathbf{u}$ by high-resolution location $\mathbf{X}$. $A(\mathbf{U}, D)$ is the function for advection of density $D$ by $\mathbf{U}$.

### 3.2.3.1 Base Fluid Simulation

We simulate the motion of fluid by solving the incompressible Navier-Stokes equations commonly used to animate fluid in graphics. The Navier-Stokes equations describing inviscid incompressible fluid motion are

$$\mathbf{u}_t + (\mathbf{u} \cdot \nabla)\mathbf{u} + \frac{\nabla p}{\rho} = \mathbf{f} \tag{3.13}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{3.14}$$

where $\mathbf{u} = (u, v, w)$ is the fluid's velocity, $p$ is the pressure, $\rho$ is the density, and $\mathbf{f}$ is the external forces such as gravity and vorticity confinement. Since upsampled velocity field depends on that of base simulation, using a vorticity confinement results in more dynamic flow than not using it. Since numerical methods of solving Eqs. (3.13) and (3.14) are well known, you can refer to [7, 38] for details.

#### 3.2.3.2 Saving Grid Vorticity to Vortex Particle

We use vortex particle method in this section. The Navier-Stokes equations can be put into vorticity form by taking the curl of Eq. (3.13) to obtain

$$\omega_t + (\mathbf{u} \cdot \nabla)\omega - (\omega \cdot \nabla)\mathbf{u} = \mu \nabla^2 \omega + \nabla \times \mathbf{f} \tag{3.15}$$

where $\omega = \nabla \times \mathbf{u}$, $\omega$ is vorticity. The vorticity advection term $(\mathbf{u} \cdot \nabla)\omega$ and the vortex stretching term $(\omega \cdot \nabla)\mathbf{u}$ are calculated and saved in the particles, thereby preserving the magnitude of vorticity in each step. We ignore the $\mu\nabla^2\omega$ and $\nabla \times \mathbf{f}$ terms like Selle et al. [33].

#### 3.2.3.3 Generating Vorticity Field

The vorticity of vortex particles obtained in Sect. 3.2.3.2 is transferred to high-resolution vorticity field $\mathbf{U}_{vorticity}$ which is different from Selle et al. [33]. We get the vorticity of a particle, $\omega_p^*(\mathbf{X}) = \xi_p(\mathbf{X} - \mathbf{X}_p)\omega_p$ by using the distribution kernel, $\xi_p(\mathbf{X} - \mathbf{X}_p)$ which is a function of the distance between the particle position and the high-resolution grid point. Then we calculate the sum of all contributions from all particles by the normalized location vector from grid point to vortex particle, $\mathbf{N}_p(\mathbf{X}) = \frac{\mathbf{X}_p - \mathbf{X}}{||\mathbf{X}_p - \mathbf{X}||}$ and the confinement force, $\mathbf{F}_p(\mathbf{X}) = \varepsilon_p(\mathbf{N}_p \times \omega_p^*)$ to update the vorticity field element which is in the influence of the particles.

**Discussion**: The vorticity field generated by above method is not divergence-free. To make vorticity field divergence-free, we can calculate curl of particle's vorticity taken by a ramp function instead of using $(\mathbf{N}_p \times \omega_p^*)$. Our result came from the method of Sect. 3.2.3.3. The nondivergence-free status in vorticity field at subgrid level is ignorable since the vorticity field is newly created at each step instead of being advected. Our result shows little difference as compared with divergence-free case and it operates a little faster.

#### 3.2.3.4 Blending Vorticity Field with Upsampled Velocity Field

We create a new field $\mathbf{U}_{advection}$ for the advection of density $D$, as follows:

$$\mathbf{U}_{advection}(\mathbf{X}, \mathbf{t}) = k\mathbf{U}_{vorticity}(\mathbf{X}, \mathbf{t}) + \mathbf{U}_{interpolation}(\mathbf{X}, \mathbf{t}) \tag{3.16}$$

where $k$ is a constant that controls influence of the vorticity field and $\mathbf{U}_{interpolation} = I(\mathbf{u}, \mathbf{X})$. We use $k = 1.0$ in our experiment and simple linear interpolation for $\mathbf{U}_{interpolation} = I(\mathbf{u}, \mathbf{X})$.

**Pseudo-code**

```
1 Advection($\mathbf{u}$)
2 Vorcity Confinement($\mathbf{u}$)
3 Projection($\mathbf{u}$)
4 $\mathbf{U}_{interpolation} = I(\mathbf{u}, \mathbf{X})$
5 $\mathbf{U}_{vorticity}$ = Vortex Particle Method($\mathbf{u}$)
6 $\mathbf{U}_{advection} = k\mathbf{U}_{vorticity} + \mathbf{U}_{interpolation}$
7 $A(\mathbf{U}_{advection}, \mathbf{D})$
```

### *3.2.4 Examples*

The following kernel function [25] is used to generate vorticity field from vortex particles.

$$\xi_p(\mathbf{X} - \mathbf{X}_p) = \begin{cases} (r^2 - (\mathbf{X} - \mathbf{X}_p)^2)^3 & 0 \le (\mathbf{X} - \mathbf{X}_p) \le r \\ 0 & \text{otherwise} \end{cases} \tag{3.17}$$

where $r$ is the radius of the particles, which determines the influence range when the force of the vortex particle is transferred to $\mathbf{U}_{vorticity}$. A particle radius is definedlong enough to cover about two to three $\mathbf{U}$ cells.

Figure 3.11 shows smoke rising from a spherical density source due to buoyancy forces. In this example, vortex particles are randomly seeded where the density is sourced. Vortex particle method (approximately 300 vortex particles for low-resolution field and 20,000 vortex particles for high-resolution field, respectively) and Vorticity Confinement were used for base simulation and the BFECC advection method was used for density and velocity advection. Figures 3.12 and 3.13 show the comparison between low-resolution and high-resolution simulations.

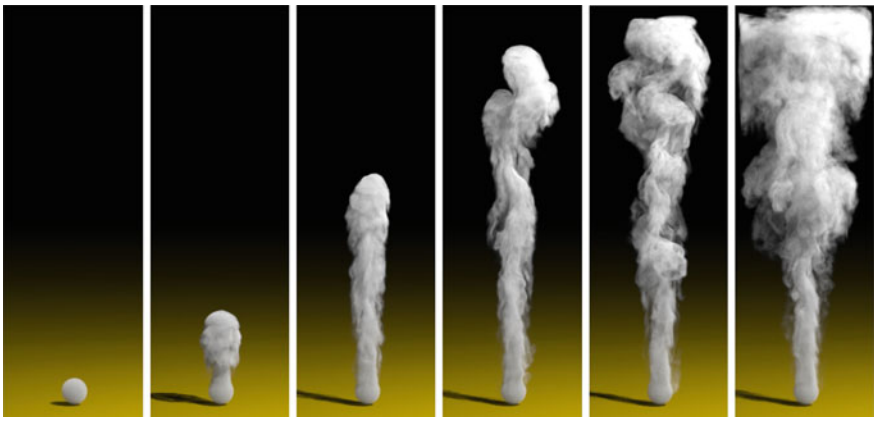


**Fig. 3.11** Rising smoke: Simulation sequence of synthesizing a $200 \times 600 \times 200$ resolution from a $50 \times 150 \times 50$ resolution by adding detail using a procedural vortex particle method

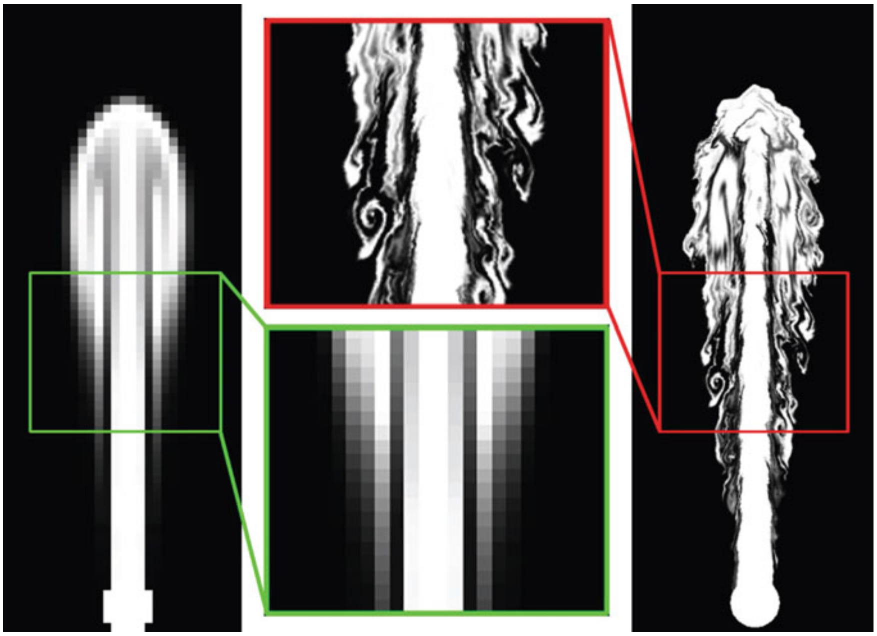


**Fig. 3.12** 2D simulation comparison: the low-resolution simulation (30 × 90) is shown in the *left*. An effective resolution of 480 × 1440 using our method is shown in the *right*

This method is prominently faster than an equivalent full-resolution simulation and performs around seven times faster than the full solver as shown in Table 3.2. All the experiments were performed on an Intel Quad Core CPU 2.4 GHz processor with 2 GB RAM.

**Limitations**: This method has following limitations. As we calculate the sum of all contributions from every vortex particle when generating vorticity field, vorticity force is added to velocity field in proportion to the number of vortex particles gathering together in the particle radius. And then strong vorticity force can cause unnaturalness in fluid's motion. This method cannot reproduce physically correct high-resolution simulation, and its boundary condition depends on low resolution. So there could be a velocity going into an obstacle at subgrid level.

**Table 3.2** Comparison of computation time: Performance comparison of our method with the traditional Navier-Stokes solver needed to generate the same visual detail

| | Our method | Traditional NS solver |
|---|---|---|
| Resolultion | 120 × 360 × 120 | 120 × 360 × 120 |
| Time per frame | 9.2 s | 59.0 s |

Our method used the result of base simulation (30 × 90 × 30, 0.58 s per frame)

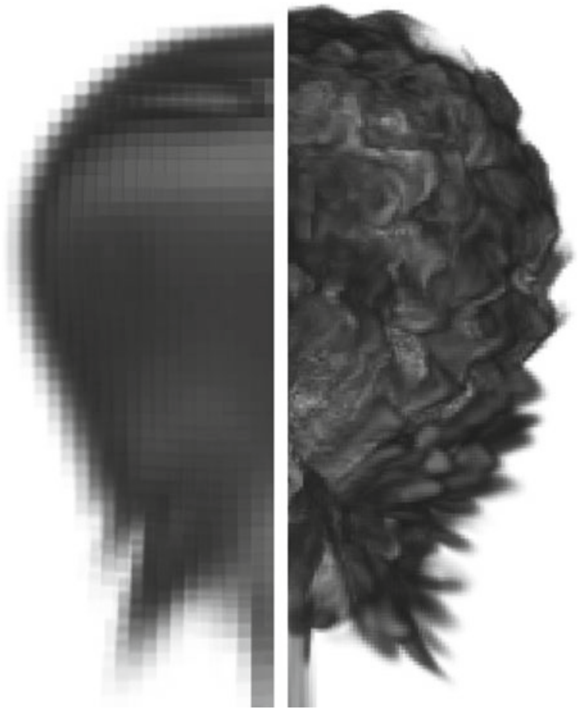


**Fig. 3.13** Explosion-like effect: The low-resolution simulation (50 × 100 × 50) is shown in the *left* half of the image. The high-resolution simulation (200 × 400 × 200) is in the *right*

### *3.2.5 Conclusions and Future Work*

This section proposed a synthesizing method using vortex particle method to improve subgrid visuals of fluid simulation. We modified the technique to apply the vortex particle method instead of using a noise function. We created flow field by solving the incompressible Navier-Stokes equations. It is upsampled through linear interpolation and blended with vorticity field which is calculated by vortex particle method. High-resolution results were achieved by advecting the density with using the calculated velocity field, and made it possible to simulate highly turbulent swirling motion of fluid.

We used vortex particle method. Compared with previous noise based work [23, 26], our work can show more subgrid details with high turbulences and swirls, and the generation of subgrid details can be expected more easily. Overall process of ours and theirs is similar but we have different approach in generating turbulence, so the computational time of our work is a little different from that of their work, and the computational time depends on the number of vortex particles and the radius of the particles.

In the future, our method could be applied to animation of liquids with free surfaces and make proper boundary conditions at the subgrid level for more precise simulation.

## References

1. Abbott MB (1989) Computational fluid dynamics. Wiley, New York
2. Angelidis A, Neyret F (2005) Simulation of smoke based on vortex filament primitives. In: Eurographics/ACM SIGGRAPH symposium on computer animation, pp 87–96
3. Angelidis A, Neyret F, Singh K, Nowrouzezahrai D (2006) A controllable, fast and stable basis for vortex based smoke simulation. In: Eurographics/SIGGRAPH symposium on computer animation, pp 25–32
4. Bruger A, Gustafsson B, Lotstedt P, Nilsson J (2005) High-order accurate solution of the incompressible Navier-Stokes equations. J Comput Phys 203(1):49–71
5. Dupont TF, Liu Y (2003) Back and forth error compensation and correction methods for removing errors induced by uneven gradients of the level set function. J Comput Phys 190(1):311–324
6. Fattal R, Lischinski D (2004) Target-driven smoke animation. ACM Trans Graph 23(3): 441–448
7. Fedkiw R, Stam J, Wann Jensen H (2001) Visual simulation of smoke. In: Proceedings of ACM SIGGRAPH, pp 15–22
8. Feldman BE, O'Brien JF, Arikan O (2003) Animating suspended particle explosions. ACM Trans Graph 22(3):708–715
9. Felici HM, Drela M (1990) Eulerian/Lagrangian solution of 3-d rotational flows. In: AIAA 21st fluid dynamics, plasma dynamics and lasers conference, 12 June 1990
10. Felici HM, Drela M (1993) Reduction of numerical diffusion in three-dimensional vortical flows using a coupled Eulerian/Lagrangian solution procedure. In: AIAA 24th fluid dynamics conference, 12 July 1993
11. Felici HM, Drela M (1995) An Eulerian/Lagrangian coupling procedure for three-dimensional vortical flows. AIAA J 33(1):48–55
12. Foster N, Metaxas D (1996) Realistic animation of liquids. Graph Models Image Process 58(5):471–483
13. Foster N, Metaxas D (1997) Modeling the motion of a hot, turbulent gas. In: Proceedings of SIGGRAPH, pp 181–188
14. Griebel M, Dornseifer T, Neunhoffer T (1988) Numerical simulation in fluid dynamics: a practical introduction. Cambridge University Press, Cambridge
15. Hong J-M, Kim C-H (2003) Animation of bubbles in liquid, computer graphics forum. In: Proceedings of Eurographics 2003, vol 22(3), pp 253–262, September 2003
16. Hong J-M, Kim C-H (2004) Controlling fluid animation with geometric potential. Comput Animat Virtual Worlds 15(3–4):147–157
17. Hong J-K, Kim C-H (2005) Animating smoke with dynamic balance: natural phenomena and special effects. Comput Animat Virtual Worlds 16(3–4):405–414
18. Hong J-M, Kim C-H (2005) Discontinuous fluids. ACM Trans Graph 24(3):915–920
19. Johnston H, Liu J-G (2004) Accurate, stable, and efficient Navier-Stokes solvers based on explicit treatment of the pressure term. J Comput Phys 199(1):221–259
20. Kim B-M, Liu Y, Llamas I, Rossignac J (2005) Flowfixer: using BFECC for fluid simulation. In: Eurographics workshop on natural phenomena, pp 51–56
21. Kim B, Liu Y, Llamas I, Rossignac J (2007) Advections with significantly reduced dissipation and diffusion. IEEE Trans Vis Comput Graph 13(1):135–144
22. Kim D, Song O-Y, Ko H-S (2008) A semi-Lagrangian CIP fluid solver without dimensional splitting. Comput Graph Forum 27(2):467–475
23. Kim T, Thurey N, James D, Gross M (2008) Wavelet turbulence for fluid simulation. ACM Trans Graph 27(3):1–6
24. Losasso F, Gibou F, Fedkiw R (2004) Simulating water and smoke with an octree data structure. ACM Trans Graph 23(3):457–462
25. Müller M, Charypar D, Gross M (2003) Particle-based fluid simulation for interactive applications. In: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on computer animation, pp 154–159

26. Narain R, Sewall J, Carlson M, Lin M (2008) Fast animation of turbulence using energy transport and procedural synthesis. ACM Trans Graph 27(5):1–8
27. Neyret F (2003) Advected textures. In: Proceedings of Eurographics/SIGGRAPH symposium on computer animation, pp 147–153
28. Park SI, Kim MJ (2005) Vortex fluid for gaseous phenomena. In: Eurographics/SIGGRAPH symposium on computer animation, pp 261–270
29. Press WH, Teukolsky SA, Vetterling WT, Flannery BP (1988) Numerical recipes in C. Cambridge University Press, Cambridge
30. Rasmussen N, Nguyen D, Geiger W, Fedkiw R (2003) Smoke simulation for large scale phenomena. ACM Trans Graph 22(3):703–707
31. Saffman PG (1992) Vortex dynamics. Cambridge University Press, Cambridge
32. Schechter H, Bridson R (2008) Evolving sub-grid turbulence for smoke animation. In: Eurographics/SIGGRAPH symposium on computer animation, pp 1–7
33. Selle A, Rasmussen N, Fedkiw R (2005) A vortex particle method for smoke, water and explosions. ACM Trans Graph 24(3):910–914
34. Selle A, Fedkiw R, Kim B, Liu Y, Rossignac J (2008) An unconditionally stable Maccormack method. J Sci Comput 35(2–3):350–371
35. Shi L, Yu Y (2005) Controllable smoke animation with guiding objects. ACM Trans Graph 24(1):140–164
36. Song O-Y, Shin H-C, Ko H-S (2005) Stable but non-dissipative water. ACM Trans Graph 24(1):81–97
37. Stam J, Fiume E (1993) Turbulent wind fields for gaseous phenomena. In: Proceedings of SIGGRAPH, pp 369–376
38. Stam J, Stable fluids. In: Proceedings of SIGGRAPH, vol 99, pp 121–128
39. Van Dyke M (1982) An album of fluid motion. The Parabolic Press, California

# Chapter 4
# Fire and Ice

## 4.1 Shrinkage, Wrinkling, and Ablation of Burning Cloth and Paper

**Abstract** The burning of a sheet of cellulose-based material, such as paper or cloth, involves uneven shrinkage which causes wrinkling. In this section, we simulate this geometrically complicated phenomenon by modeling the effects of heat transfer, shrinkage, and partial ablation on a thin shell. A strain limitation technique is applied to a two-layer structure of springs arranged as a body-centered square. Although this structure is overconstrained, convergence can be achieved using a new successive fast projection method. We also remesh the shells dynamically to deal with the topological changes that occur as regions burn away.

### *4.1.1 Introduction*

Heat changes the shape of solid objects by altering the material's physical and chemical state and properties. During combustion, a portion of material rapidly decomposes into gases. The residual solid is an ash which has different mass and density, causing shrinkage; as a result, the shell bends, crumples and tears, generating distinctive wrinkles.

Shells made of common combustible materials, such as cloth and paper, rarely stretch or compress in the plane of the shell, but easily bend. This property is called developability. When compressive forces are applied in the plane of a thin shell, it buckles but maintains its surface area. As a shell burns, mass is not lost evenly and the differential shrinkage in adjacent regions causes stresses, even though there is no external force. But the effect is the same: the shell curves to reach equilibrium.

Recent research on the simulation of burning shells has attempted to capture these complicated deformations. Losasso et al. [1] simulated melting and burning objects by tracing the changing surface of the remaining solid region. Their results

C.-H. Kim et al., *Real-Time Visual Effects for Game Programming*,
Gaming Media and Social Effects, DOI 10.1007/978-981-287-487-0_4

are only plausible superficially, because the deformations of burnt regions caused by shrinkage of the solid are not incorporated into their simulation. Later, Melek and Keyser [2] and Liu et al. [3] introduced deformation techniques to bend and crumple thin shells. However, their free-form deformation (FFD) method and the mass-spring systems produce simply curved shapes without the fine detail seen in reality, because these approaches do not fully enforce the inextensibility of thin shells.

This section proposes a method of simulating the bending, crumpling, and wrinkling of burning shells by integrating simulations of heat transfer and the structure of developable surface (Fig. 4.1). In order to produce fine wrinkles, it is necessary to model the internal dynamics of thin shells accurately. Inspired by the linear beam

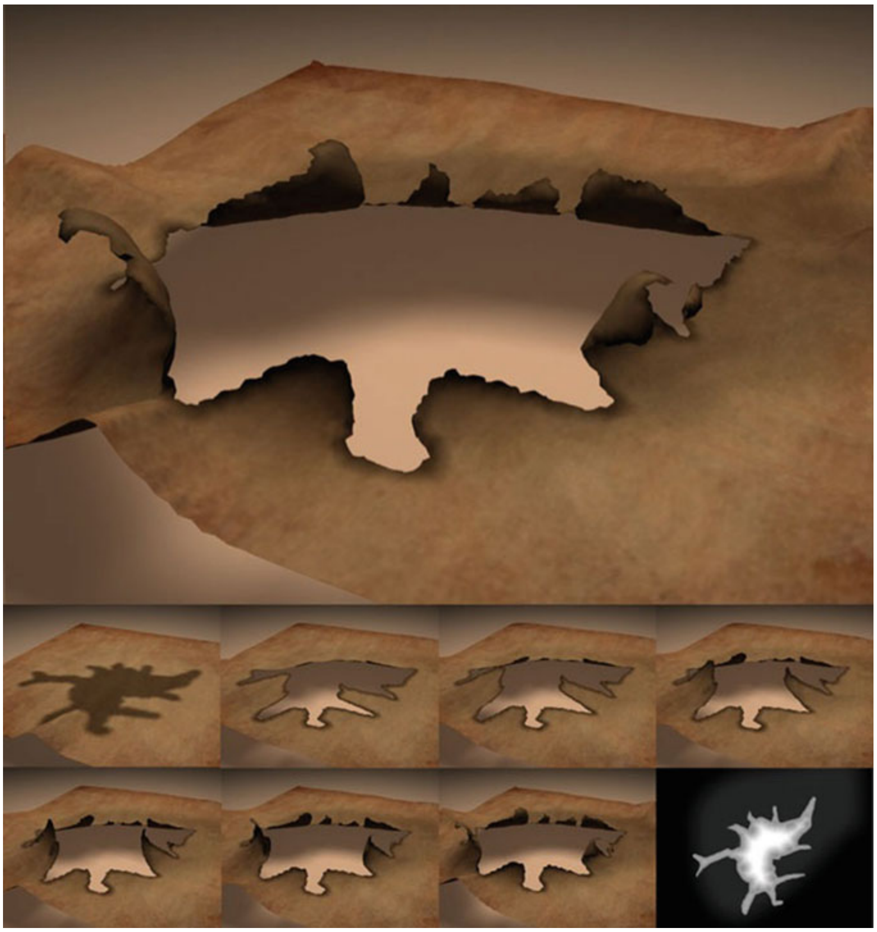


**Fig. 4.1** Simulation of burning paper. Simulated temperature is higher than the ignition point, and the rate of mass loss is weighted using the texture shown. As the mass of the paper falls, the paper crumples and wrinkles

geometry which is used to explain the bending of shells, this section proposes a double-layer shell model with finite thickness based on a body-centered square configuration of springs. This structure enables us to control the stretching and bending of thin shells by using the springs to limit the strain.

This requires a robust method for determining the extensions of a network of springs. The fast projection method is a powerful approach to limiting strain which is based on constrained Lagrangian mechanics. However, when applied to the shell structure above, fast projection diverges because the structure is overly constrained and the gradient of the spring constraints is linearly dependent. To avoid this, we break up the constraints into sets to satisfy convergence conditions and which we can project successively.

The extension of each spring is determined by considering the reduction in mass and density which occurs in regions which are burning because the temperature exceeds the ignition point. Different rates between adjacent regions, and the two layers of our shell model, buckle the shell. This approach is more physically plausible and produces more satisfyingly complicated wrinkles than existing methods, which simply relate bending to differences in temperature. To support this approach, this section also provides a remeshing technique to account for the topological changes produced by burning.

We will review related works in Sect. 4.1.2. Section 4.1.3 introduces the double-layered shell configuration. Section 4.1.4 presents the overview of heat transfer simulation process that is used to model shell deformation. Sections 4.1.5, 4.1.6, and 4.1.7 successively describe this heat transfer, the changes of material properties that occur during combustion, and the shell dynamics, respectively. Simulation results, details of the implementations, and a discussion of the limitations of our approach follow in Sect. 4.1.8. Finally, we conclude in Sect. 4.1.9.

### 4.1.2 Related Work

Focusing on the generation of wrinkles in burning shells, we will briefly review work on the deformation of solid objects related to heat transfer and developable shells.

Terzopoulos et al. [4] adopted particle-spring systems to discretize 3D volumes and model phase changes in melting solids by controlling the stiffness of springs. Carlson et al. [5] simulated the melting of solids, but they adjusted viscosity within a fluid dynamics simulation. Their grid-based representation straightforwardly handles fluid flows and topological changes. Losasso et al. [1] detected the boundary of the remaining solid in burning or melting by embedding these solids within a body-centered cubic lattice and then extracting the boundary of the solid by evaluating level-set values at lattice points.

Melek and Keyser [2] looked at the bending and crumpling of a burning object during combustion, and presented a deformation method to simulate it in real time. The burning object is encompassed by a low-resolution grid which is subjected to free-form deformations. Liu et al. [3] also deformed thin shells indirectly by modifying an enclosing lattice. However, they modeled the crumpling forces induced by differences in temperature in lattice points on either side of the shell instead of using geometric deformation.

A realistic simulation of thin shells is feasible if its internal dynamics can be accurately modeled. Early seminal works on cloth simulation [6, 7] regarded thin shells as elastic, but an elastic shell shrinks and loses detail when compressive forces are applied in its plane. Moreover, displacements from the rest shape cause an elastic shell to produce large restoring forces which reduce numerical stability. The more accurate implicit [6], semi-implicit [8], and BDF2 [9] integration methods have been proposed to improve convergence. Nevertheless, methods based on potential energy still produce unrealistically smooth results which motivate more aggressive methods of strain limitation.

Provot [10] limited the deformation rate of springs to 10 % of their rest lengths, and shrank locally elongated springs iteratively until they are within this limit. Müller et al. [11] enforced length constraints on springs by projecting two points into the line of action of a spring until they reach valid positions. Constraint-based methods have been successfully used to simulate quasi-inextensible cloth by enforcing global constraints. Enforcing implicit constraints [12] provide more stable convergence and allows the use of a larger time-step than the explicit method. Fast projection [13] is a linearized implicit integration which converges much more quickly than implicit methods, but still keeps strain under 0.1 %. English and Bridson [14] adopted fast projection and BDF2 to maintain the rest lengths of edges in a new nonconforming discretization which prevents spurious bending forces which imparts spurious stiffness to shells.

An explicit model of bending is indispensable in the realistic simulation of wrinkles in thin shells. Early cloth simulations [6, 8, 15] simply controlled bending forces in terms of the angles between adjacent faces. Thomaszweski and Wacker [16] introduced a physically more accurate nonlinear bending model which accounts for curvatures. Later, the linear [17], quadratic [18], and cubic polynomial [19] approximate bending models were introduced to improve performance and controllability. The bending models of Grinspun et al. [15] and Bridson et al. [8] use nonzero rest angles to preserve the curved and wrinkled rest shape of cloth while responding to collisions. The constraint-based approach [20] introduces hard angle constraints which allow a shell with sharp creases to be modeled.

## *4.1.3 The Shell Configuration*

Existing cloth simulations use independent stretching and bending models because thin shells are developable. However, the stretching forces caused by uneven shrinkage in the plane of shell generate wrinkles, motivating us to control all internal dynamics using a unified model.

This section first clarifies the principle of bending with the linear beam element, and then explains how both bending and stretching of thin shells can be managed by in-plane tensile and compressive forces. Then it proposes a particle-spring system in which a body-centered square defines two layers and finite thickness, based on the beam model.

#### 4.1.3.1 The Linear Beam Element

Thomaszewski and Wacker [16] explained the bending of thin shells in terms of the geometry of a simple beam (Fig. 4.2a) which corresponds to the classical beam element. This beam geometry has a neutral axis (green lines), top and bottom layers (blue lines), and a finite thickness spanned by normal lines (red lines). Based on the Kirchhoff–Love assumptions, the lengths of the neutral axis and the normal lines do not change, whereas the top and bottom layers stretch or shrink in response to the tensile and compressive stresses which occur when the neutral axis of the loaded geometry assumes a curved shape (Fig. 4.2c).

Applying the beam geometry to a shell, we see that the restoring forces in the plane of the shell correspond to bending forces exerted on the neutral axis of a beam. Stretched or compressed layers tend to be returned to their rest states by the restoring forces, the neutral axis restores flat. If a beam is extremely stiff, it tends to remain rigid, whereas flexible material such as cloth has a low stiffness. Therefore, it is possible to control inextensibility and bending stiffness by applying only stretching forces to the neutral axis and the top layer (we can disregard the bottom layer, since it is dependent on the top layer).

A combined model which accounts for both stretching and bending is more appropriate for generating realistic wrinkles on thin shells for two reasons: First, maintaining the rest curvature of a stiff shell or forcing that shell to bend require large forces which make the simulation unstable. The implicit method [6] offers a solution to this problem, but implicit integration of a nonlinear bending model is tricky because the gradient of the formula is difficult to calculate. Second, the bending of thin shells during burning is not caused by differences in temperature [2, 3], but by shrinkage caused by changes of mass and density. The use of bending model requires an additional formula to relate the amount of shrinkage to the curvature of the shell, whereas our model bends as implicitly one layer shrinks.

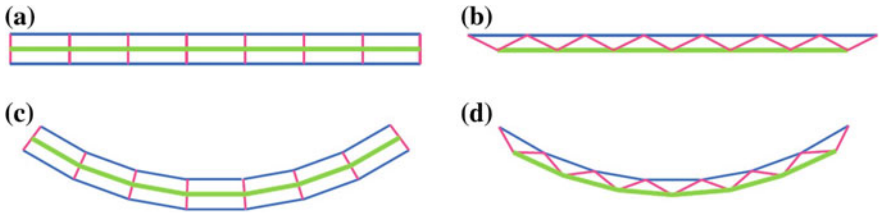


**Fig. 4.2** **a** The linear beam geometry, and **b** and our BCS shell, shown diagrammatically in 2D. The two layers and the normals connecting them correspond to the *blue*, *green*, and *red* springs, respectively. When the *top* layer shrinks and the *blue* springs contract, both **c** the beam geometry and **d** our shell *curve*

A thin shell with two layers is not an approach that has been generally applied to cloth simulation, but it has already been used in [2, 3] to model burning object. A thin shell can be embedded in a 3D lattice and then the lattice is deformed in response to differences in temperature between adjacent nodes in the structure. This approach involves the assumption that the bending of thin shells is caused by the temperature difference between the top and bottom layers.

#### 4.1.3.2 A Double-Layered Shell Based on the Body-Centered Square

The body-centered square (BCS) is two-dimensional analog of the BCC lattice, which is commonly found in stable molecular or crystal structures. Its uniform point distribution, symmetry, and stability have caused it to be used for many applications in computer graphics, including tetrahedralization [21], 3D volume deformations [21], and shell simulations [1].

The BCS consists of a set of rectangular cells, each of which has a lattice point at its center and four more points at its corners (Fig. 4.3a). We can create four types of spring which connect the lattice and corner points, and we can associate each type of spring with a color: Red, green, blue, and yellow (Fig. 4.3c), following Molino et al. [21]. The green and blue orthogonal springs, $\mathbf{S}_G$ and $\mathbf{S}_B$, correspond to the top and bottom surfaces of the linear beam model; the red intermediate springs, $\mathbf{S}_R$, stitch the two layers together diagonally to maintain adjacency and thickness, and the yellow springs, $\mathbf{S}_Y$, link the two diagonal corner points of a unit cell to control shearing. Since the lattice points and corner points are in the same plane, the resulting spring structure is still a 2D manifold mesh (Fig. 4.3b). Therefore, we offset the lattice points from the plane of the rest shell to create a shell of finite thickness (Fig. 4.3d).

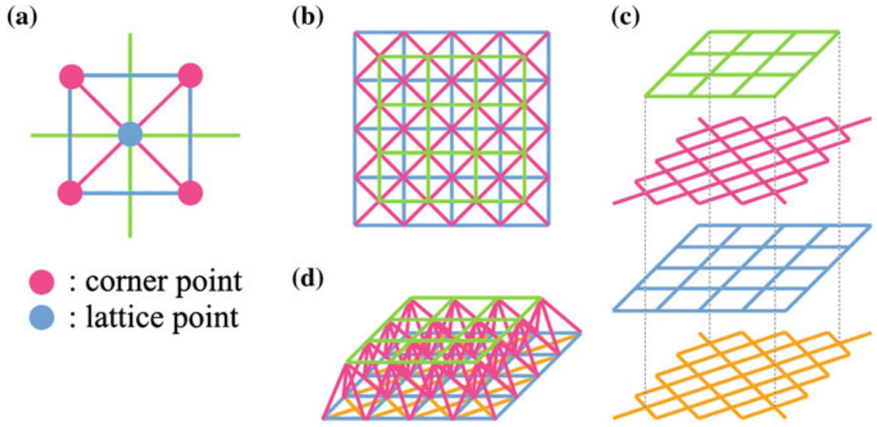


**Fig. 4.3** The double-layer shell configuration with thickness based on the body-centered *square*. The structure consists of four independent quadrilateral meshes: *red*, *blue*, *green*, and *yellow*

All the springs are used in modeling the dynamics of the shell, but only the quadrilateral mesh of blue springs is required for rendering. This mesh can easily be converted as an isotropic triangular mesh. However, because our shell has thickness, the lattice points are no longer in the same plane as the corner points. Therefore, we replace each lattice point with a virtual center point, which is placed at the average position of the four corner points, and update its position after computing the dynamics of the shell. The resulting triangular mesh is used for collision detections as well as rendering.

### 4.1.4 Simulation Overview

We construct a particle-spring system using the body-centered square structure. A shell consists of a set of particles $\mathbf{P}$, and four sets of springs $\mathbf{S}_R$, $\mathbf{S}_G$, $\mathbf{S}_B$, and $\mathbf{S}_Y$. A particle $p_i$ has physical attributes, including a position $\mathbf{x}_i$, a velocity $\mathbf{v}_i$, a mass $m_i$, and a temperature $T_i$ at its position. A spring $s_j$ connects two end particles $p_0$ and $p_1$. The length $l$ of a spring in the rest state is called the *target length*.

At each time-step $n$, the attributes of particles and springs are updated in the following order:

1. The temperatures $\mathbf{T}^n$ of particles are determined by simulating the heat transfer.
2. The masses $\mathbf{m}^n$ of particles at temperatures higher than the ignition point are reduced.
3. The particle-spring structure and the rendering mesh are remeshed to account for the deleted particles and springs.
4. The positions $\mathbf{x}^n$ and velocity $\mathbf{v}^n$ of particles are determined by simulating the shell dynamics and interactions with the environment.

### 4.1.5 Heat Transfer

A burning shell receives heat from its environment by convection and radiation, produces its own heat through combustion, and the heat is conducted through and along the shell. Incorporating fire simulations into our system could enhance the realism of burning scenes, since changing flame shapes produce natural variations in temperature across burning shells. However, we use simpler external heat sources, such as heat balls (Figs. 4.1 and 4.10) and heat textures (Fig. 4.4). We express the change in temperature produced by these heat sources as $\mathbf{T}^{ext}$. Additionally, when the temperature of particles exceeds the ignition point, the temperature rises quickly due to the reduced mass and the ratio of heat to mass, $\mu$. As a result of all these, the temperature of the particles can be expressed as follows:

$$\mathbf{T}^n = \mathbf{T}^{n-1} + \mathbf{T}^{ext} + \eta \Delta \mathbf{m}^{n-1}. \tag{4.1}$$

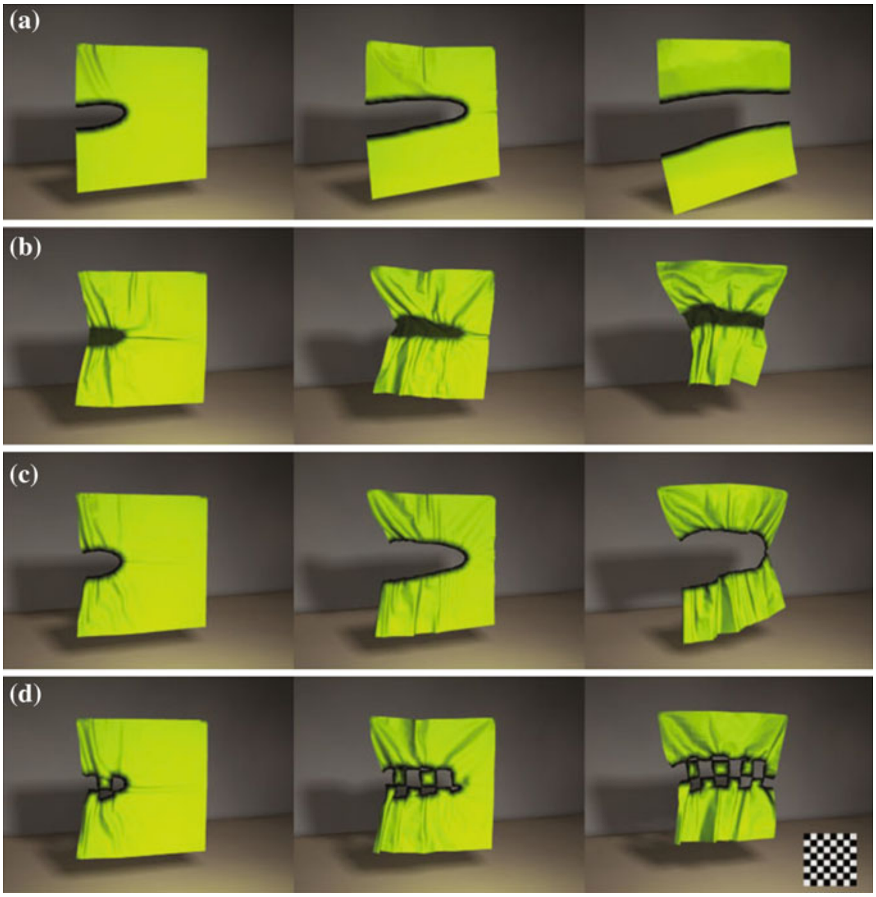


**Fig. 4.4** Comparisons of the results of various burning effects when a very hot sphere passes across thin shells: **a** boundary change only; **b** shrinkage; **c** boundary change and shrinkage; and **d** using a texture to adjust the rate of mass loss

We also simulate the conduction of heat over the shell. Physically, accurate diffusion can be obtained by solving the heat equation,

$$\frac{\partial \mathbf{T}^n}{\partial t} = \alpha \nabla^2, \tag{4.2}$$

where $\alpha$ is thermal diffusivity.

To solve the heat equation for our particle-spring structure, we use the method of Desbrun et al. [22], which approximates the Laplacian by umbrella operators. Since the spring lengths in our model are not identical and are changed by burning, we need to consider them as parameters in the heat equation. We use a scale-dependent umbrella operator which weights the Laplacian by the inverse of the distance $d_{ij}$

between two particles $p_i$ and $p_j$,

$$L(\mathbf{x}_i) = \frac{2}{E} \sum_{j \in N_1(i)} \frac{\mathbf{x}_j - \mathbf{x}_i}{|d_{ij}|}, \quad \text{where} \quad E = \sum_{j \in N_1(i)} |d_{ij}|, \tag{4.3}$$

and $N_1(i)$ are the 1-ring neighbors of particle $i$. This enables the uniform transfer of heat across irregular meshes. By using the implicit backward Euler method, we can have longer time-steps and higher thermal diffusivity, $\alpha$, in a stable simulation of heat diffusion. Best of all, the diffusion of heat is unrelated to the structure of the shell; whereas, the explicit method can only transfer heat from a particle to its neighbors.

### *4.1.6 Adjusting Target Lengths and Remeshing*

A burning shell shrinks and the boundary of the remaining regions changes. We model this phenomenon by shortening springs and updating the connectivity of the particle-spring system depending on the mass change.

During combustion, the initial mass $m_i^{\text{init}}$ of a particle $i$ decreases by chemical processes when the temperature $T_i^n$ is higher than the ignition point $T^{\text{ignition}}$. The reduced mass of the particle at time-step $n$ can be expressed as follows:

$$\Delta m_i^n = -\mu h m_i^{\text{init}} (T_i^n - T^{\text{ignition}}), \tag{4.4}$$

where $\mu$ is the rate at which the mass decreases.

The mass at which a particle starts to disappear is $m^{\text{loss}}$, and $m^{\text{min}}$ as the minimum mass that a particle can have. When $m^{\text{loss}}$ is close to $m_i^{\text{init}}$, then the shell burns out quickly; but if $m^{\text{loss}}$ is less than $m^{\text{min}}$, the shell wrinkles but remains intact. If the initial masses of two particles $p_0$ and $p_1$ connected by a spring $j$ are $m_0$ and $m_1$, and their relative masses during burning are $m_0' = m_0 - m^{\text{loss}}$ and $m_1' = m_0 - m^{\text{loss}}$, then we can clarify the states of their connecting spring as follows:

1. Shrunk: $m_0' \geq 0$ and $m_1' \geq 0$
2. Partly burnt: $m_0' m_1' < 0$
3. Burnt: $m_0' < 0$ and $m_1' < 0$ .

The shrinkage of a spring is proportional to the change in mass and density of the two particles which it connects. Since volume is equal to mass over density, the volume of a particle $p_i$ changes from $m_i^{\text{init}}/\rho_i^{\text{init}}$ to $m_i/\rho_i$, where $\rho_i^{\text{init}}$ and $\rho_i$ are the densities of the initial and burnt material. We can now calculate the proportional loss in volume of the ratio of a particle $\upsilon_i$ as follows:

$$\upsilon_i = \frac{m_i/\rho_i}{m_i^{\text{init}}/\rho_i^{\text{init}}} \quad (0 \leq \upsilon_i \leq 1). \tag{4.5}$$

If the proportional loss in volume of the two particles at the ends of spring $j$ is $\upsilon_0$ and $\upsilon_1$, and $l_j^{\text{init}}$ is the initial length, of the spring, then its new target length $l_j$ is determined as follows:

$$l_j = l_j^{\text{rest}}(\upsilon_0 + \upsilon_1)/2. \tag{4.6}$$

When particles are completely burnt out ($m'_i < 0$), they can be removed from the structure, and then we remesh the particle-spring system. Since the abrupt disappearance of particles would lead to aliasing during the rendering of our lattice-based shell, we interpolate smoothly varying boundaries for the remaining regions using the technique proposed by Losasso et al. [1]: Level-set values are maintained at lattice points and the structure is remeshed using marching triangles (Fig. 4.5).

A partly burnt spring ($m'_0 m'_1 < 0$) is divided into burnt and unburnt regions by an intermediate point $\mathbf{x}_{\text{mid}}$, determined by linear interpolation between $\mathbf{x}_0$ and $\mathbf{x}_1$:

$$\mathbf{x}_{\text{mid}} = \mathbf{x}_0(1-t) + \mathbf{x}_1 t, \quad \text{where} \quad t = \frac{m'_0}{m'_0 - m'_1}. \tag{4.7}$$

Figure 4.6 presents five examples which show how the state of a spring is changed by the masses of two particles that it converts. In cases (a) and (b), the spring only shrinks since both $m'_0$ and $m'_1$ are positive, and the proportional changes in length are 87.5 and 62.5 %. The springs in (c) and (d) have already been shorten as much as 57.5 and 50 % of their initial lengths, because they are partly burnt. In case (e), the spring is burnt out.

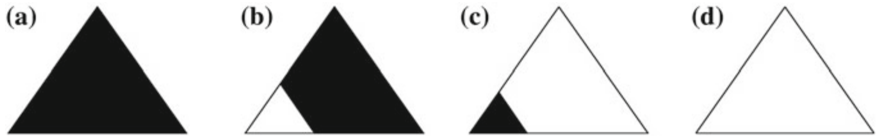


**Fig. 4.5** Four cases of a *triangle* in marching triangles, depending on the unburnt mass

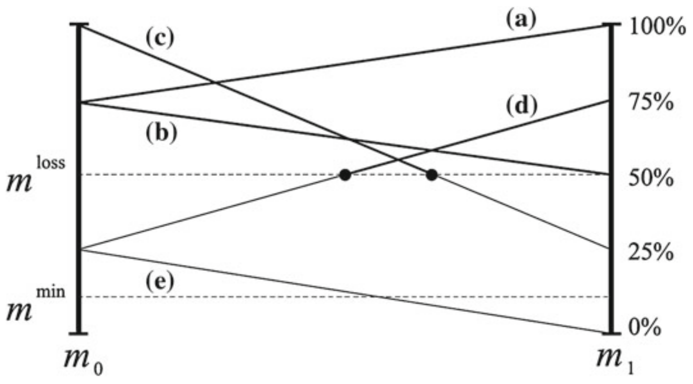


**Fig. 4.6** The states of springs with respect to $m'_0$ and $m'_1$

When the particles and springs have been updated, we remesh the particle-spring structure and the rendering mesh. Each triangle of the rendering mesh has one of four states, depending on the number of the particles remaining. If a triangle is left in the rendering mesh and there is at least one of the particles which connects positive relative mass, then we remesh the remaining particles using Marching triangles. If both triangles that share a spring disappear, we remove that spring from the system.

### 4.1.7 Shell Dynamics

Given the positions $\mathbf{x}^n$ and velocities $\mathbf{v}^n$ of particles at the beginning of time-step $n$, we first integrate the external forces $\mathbf{F}^{\text{ext}}$, such as gravity, using the forward Euler method:

$$\tilde{\mathbf{v}}^n = \mathbf{v}^n + h\mathbf{F}^{\text{ext}}, \tag{4.8}$$

$$\tilde{\mathbf{x}}^n = \mathbf{x}^n + \tilde{\mathbf{v}}^n. \tag{4.9}$$

Since our simulation of the internal dynamics, stretching, shearing, and bending of shell relies on the strain in the particle-spring systems, adopting a proper strain-limiting method is critical. We prefer the fast projection method [13] based on constrained Lagrangian mechanics, because it converges quickly to the target length.

If $\mathbf{x}_a$ and $\mathbf{x}_b$ are the two endpoints of a spring and $l$ is its target length, then a constraint that preserves the spring length and its gradient can be expressed as follows:

$$C(\mathbf{x}_a, \mathbf{x}_b) = \frac{||\mathbf{x}_a - \mathbf{x}_b||^2}{l} - l = 0 \tag{4.10}$$

$$\nabla C_{\mathbf{x}_a}(\mathbf{x}_a, \mathbf{x}_b) = \frac{2(\mathbf{x}_a - \mathbf{x}_b)}{l}. \tag{4.11}$$

Spring lengths that are changed by heat transfer may violate the spring constraints. Then the unconstrained positions $\tilde{\mathbf{x}}^n$ are moved to the positions $\hat{\mathbf{x}}^n$, which satisfy the constraint, by fast projection (this is explained in more detail in Sect. 4.1.7.1). The constrained velocity $\hat{\mathbf{v}}^n$ is obtained from the displacements during time-step $h$:

$$\hat{\mathbf{x}}^n = \text{fast\_projection}(\tilde{\mathbf{x}}^n), \tag{4.12}$$

$$\hat{\mathbf{v}}^n = \tilde{\mathbf{v}}^n + \frac{(\hat{\mathbf{x}}^n - \tilde{\mathbf{x}}^n)}{h}. \tag{4.13}$$

Finally, we correct the positions and velocities to avoid intersections with other objects and self-collision to obtain the final results:

$$\mathbf{x}^{n+1} = \text{position\_correction}(\hat{\mathbf{x}}^n), \tag{4.14}$$

$$\mathbf{v}^{n+1} = \text{velocity\_correction}(\hat{\mathbf{v}}^n). \tag{4.15}$$

Performing collision detection after fast projection could violate the constraints. But if the order of these processes is reverted, the shell could intersect other objects. Collisions with simple primitives, such as a sphere or plane, are detected at each projection step, and the positions and velocities of particles are corrected to avoid the collision.

#### 4.1.7.1 Successive Fast Projection

Fast projection [13] progressively projects points onto the closer manifold until they approach a constraint manifold $\mathbf{C}(\mathbf{x}) = 0$ within some threshold. At each projection step $j$, the unconstrained point $\mathbf{x}_j^n$ is moved by Newton's method in the direction of negative gradient of the constraint, which is expressed by the Lagrange multiplier $-\nabla\mathbf{C}(\mathbf{x}_j^n)\lambda_{j+1}$. The position of $\mathbf{x}_j^{n+1}$ after a projection step $j$ is determined by solving the following linear system with respect to $\lambda_{j+1}$:

$$(\nabla\mathbf{C}(\mathbf{x}_j)\nabla\mathbf{C}(\mathbf{x}_j)^T)\lambda_{j+1} = \mathbf{C}(\mathbf{x}_j). \tag{4.16}$$

Each displacement is then updated in turn:

$$\mathbf{x}_j^{n+1} = \mathbf{x}_j^n - \nabla\mathbf{C}(\mathbf{x}_j^n)\lambda_{j+1}. \tag{4.17}$$

We represent this projection process at step $j$ as a function of $\mathbf{S}$, which is the set of springs on which we wish to enforce the constraint

$$\mathbf{x}_{j+1}^n = \text{project}(\mathbf{x}_j^n, \mathbf{S}). \tag{4.18}$$

The numerical stability of the fast projection method mainly depends on the characteristics of the linear system (4.16). To guarantee convergence, the matrix in (4.16), a multiplication of the constraint gradient matrix and its transpose matrix, should be positive definite. The system will converge stably if the constraint gradients are linearly independent and the matrix is full ranked. The ratio of the number of constraints to the number of positional DOFs of the particles is an additional concern. A ratio of more than one prevents a solution. For example, the average ratios of a quadrilateral or triangular mesh are $2/3$ or 1, making these structures stable.

Unfortunately, the BCS does not meet these two requirements. First, the constraint gradients of the diagonal springs are linear combinations of those of the orthogonal springs which share the same particle, and so the system matrix suffers from rank deficiency. In addition, if $N_c$ is the total number of cells, the average sizes of the four constraint sets, $\mathbf{S}_R$, $\mathbf{S}_G$, $\mathbf{S}_B$ and $\mathbf{S}_Y$, are $4N_c$, $2N_c$, $2N_c$ and $2N_c$, and the number of DOFs of corner and lattice particles are $3N_c$ and $3N_c$, respectively. This makes the ratio of the number of constraints to the number of positional DOFs is 10:6, which is more than 1.

Our solution is to divide the constraints into subsets, each of which satisfies the convergence conditions, and solve them sequentially. We can create appropriate subsets by taking the springs of each color; the springs of each type constitute a network with the same topology as the orthogonal quadrilateral mesh used in previous techniques [13, 14]. There are four spring colors, giving us four constraints to apply successively. We achieve this using a fast projection step $j$ consisting of four subsequences. The points $\mathbf{x}_j$ $(=\mathbf{x}_j^n)$ input to the projection step $j$ are moved closer to the constraint $\mathbf{C}^R$ of the spring set $\mathbf{S}^R$. The resulting points $\mathbf{x}_j^R$ are then projected onto the manifold $\mathbf{C}^G$. This is the first of four projections:

$$\begin{aligned}
\mathbf{x}_j^R &= \text{project}(\mathbf{x}_j, \mathbf{S}^R),\\
\mathbf{x}_j^G &= \text{project}(\mathbf{x}_j^R, \mathbf{S}^G),\\
\mathbf{x}_j^B &= \text{project}(\mathbf{x}_j^G, \mathbf{S}^B),\\
\mathbf{x}_{j+1} &= \text{project}(\mathbf{x}_j^B, \mathbf{S}^Y).
\end{aligned}$$

This completes projection step $j$. The graphs in Fig. 4.7 show how the constraint errors $||\mathbf{C}(\mathbf{x})||$ change through successive projection steps proceeding. We see that the projection of one constraint set may increase the error of the other sets, but the sum of all the constraints always decreases.

It was experimented to see whether a point would converge at a stable location if two independent spring constraints are applied to it. Figure 4.8a shows how two types of springs, green, and blue, are connected to the particle p, and the other points are fixed. When the target lengths of the two springs are changed from 1 to 1.65 and

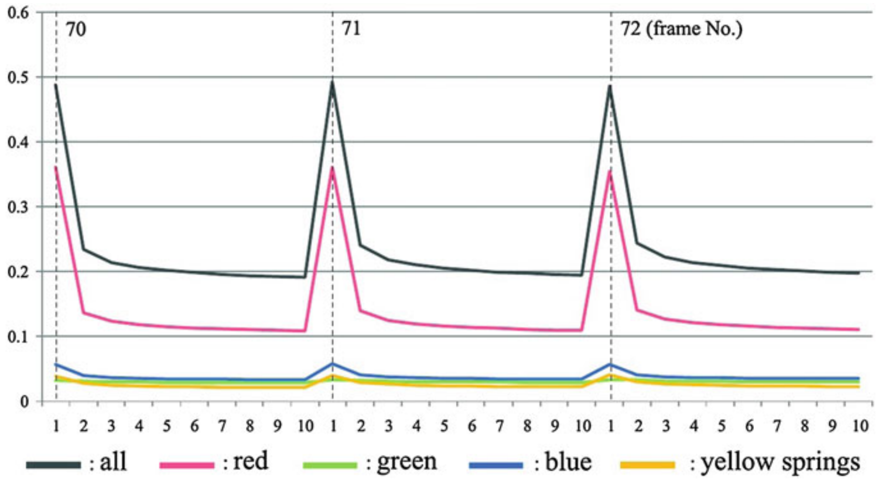


**Fig. 4.7** Errors in $||\mathbf{C}(\mathbf{x})||$ of all types of spring and their sum. The total error (*gray curve*) decreases during successive fast projection

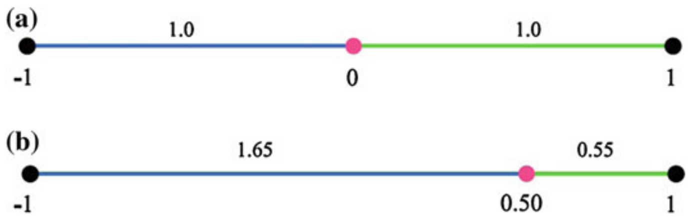


**Fig. 4.8** Convergence of two mutually exclusive springs which share a point, **a** before projections, **b** after projections

**Table 4.1** Positions of the unconstrained particle at each projection step $j$

| Projection step $j$ | Project ($\mathbf{S}^G$) | Project ($\mathbf{S}^B$) |
|---|---|---|
| Input | 0.0 | 0.0 |
| 1 | 0.4306 | 0.4401 |
| 2 | 0.5527 | 0.4954 |
| 3 | 0.5767 | 0.5038 |
| 4 | 0.5804 | 0.5050 |
| 5 | 0.5810 | 0.5052 (output) |

0.55 (Fig. 4.8b), they compete to assume their new lengths. Vibrations occur, but the system quickly converges to an equilibrium point (0.5), as shown in Table 4.1.

### 4.1.7.2 Weighted Constraints

The stiffness of an elastic body depends on its material properties. Even though the constraint $\mathbf{C}(\mathbf{x})$ does not use the material stiffness as a parameter, we can obtain the same effect by relaxing the spring constraints.

Each projection in (4.18) is equivalent to a step in Newton's method. As the slope of the gradient $\nabla\mathbf{C}(\mathbf{x}_j)$ increases, the variable $\mathbf{x}_j$ converges more slowly. If $w$ is the weight applied to a spring, we divide the displacement by the spring weight to produce a weighted displacement $-\nabla\mathbf{C}(\mathbf{x}_j^n)\lambda_{j+1}/w$, which leads to the modified linear system:

$$(\nabla\mathbf{C}(\mathbf{x}_j)\nabla\mathbf{C}(\mathbf{x}_j)^T)\lambda_{j+1} = \mathbf{W}\mathbf{C}(\mathbf{x}_j), \tag{4.19}$$

where $\mathbf{W}$ is a diagonal matrix which contains the weights of all springs. Figure 4.9 shows how we can adjust the weights of the blue and yellow springs to obtain a flexible shell. It is also possible to obtain a shell which has inhomogeneous bending stiffness by assigning different weights to the springs in each subset.

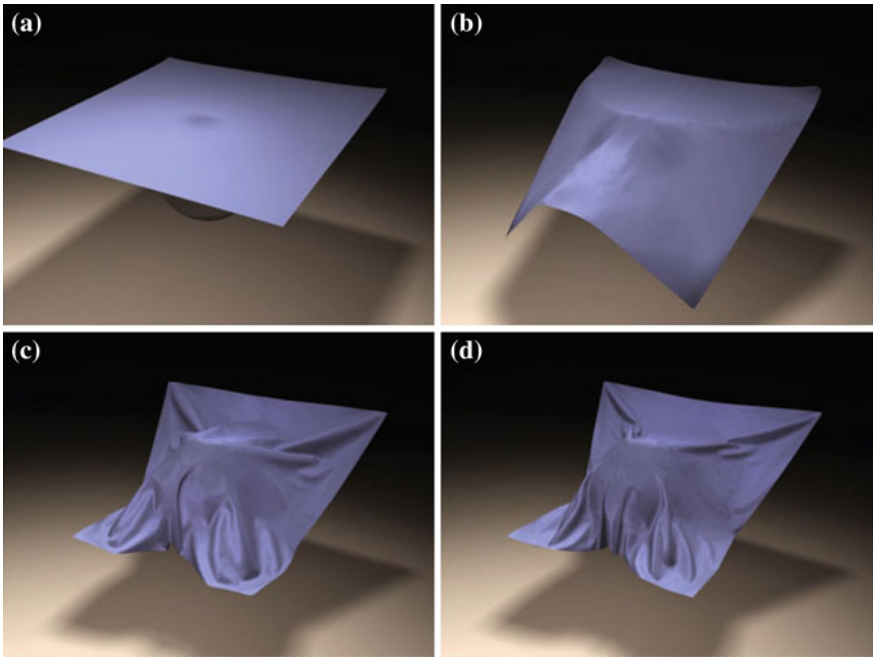


**Fig. 4.9** Different types of thin shells produced by adjusting the weights of springs and the thickness of the shell. The ratio of the thickness of each shell to the length of its longest side and the weight for *blue* and *yellow* springs are **a** 1:20 and 1.0, **b** 1:80 and 1.0, **c** 1:400 and 1.0, and **d** 1:400 and 0.1

### *4.1.8 Results*

The shell structure was tested using external heat sources. An initial temperature was assigned to a shell (Fig. 4.1) and then moving spheres of high temperature were introduced (Figs. 4.4 and 4.10). The burning process and its randomness were controlled by adjusting the rate of mass loss $\mu$, and the thermal diffusivity $\alpha$, by applying a texture. The texture colors $c \in [0, 1]$ were used at the $uv$ coordinates to weight the simulation parameters of each particle. The textures we used can be found with the simulation results.

Figures 4.1 and 4.11 show a simulation of burning paper. The initial temperature was set higher than the ignition point of the paper, and the rate of mass loss was varied using the shown texture. The partly burnt regions bend because of the loss of mass, but interior regions where the initial mass still remains also wrinkle due to the bending of the boundaries. In Fig. 4.4, a hot sphere is passed across a thin shell whose two corners are fixed. As a result, the shell separates into two parts. The same scene was simulated with the addition of topological changes and shrinkage. Figure 4.4a shows a shell which only changes its connectivity [1]. As we apply more effects, the simulation results become more detailed and realistic. Figure 4.4d, shows

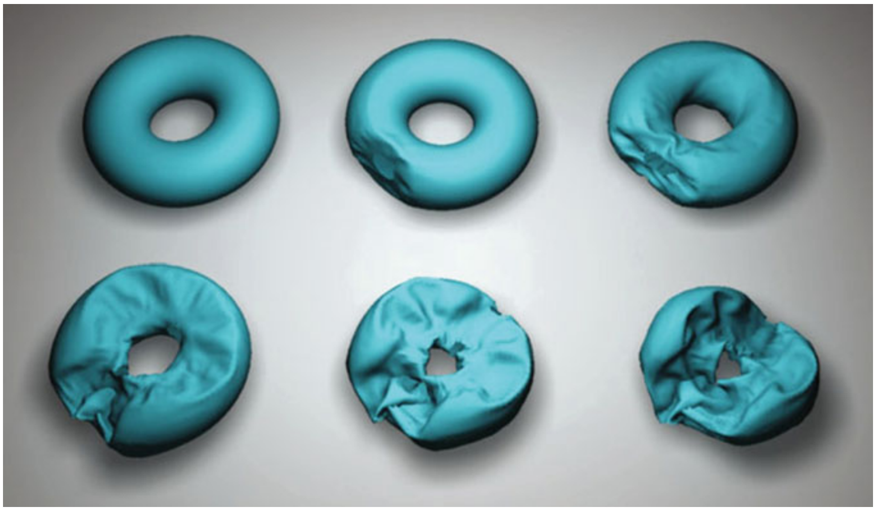


**Fig. 4.10** Snapshots of an animation of a burning torus

the results of changing the rate of mass loss $\mu$ using a texture to produce a tearing effect. A BCS-based shell can be derived from any quadrilateral mesh in Fig. 4.10, heat transfer and shrinkage of a toroidal shell were simulated.

Our strain-limiting method was verified by comparing results produced by a mass-spring model, based on Hooke's law, with those from successive fast projection. Figure 4.12a, b shows a torus falling on a sphere. The inner diameter of the torus is smaller than the sphere, and so the torus is supposed to remain on the sphere. However, the torus in Fig. 4.12b has passed over the sphere and fallen to the ground, because the mass-spring model cannot preserve the strains of springs, and allows the torus to stretch. Conversely, the torus simulated by our method maintains its rest shape and bounces against the sphere. In the second example, in Fig. 4.12c, d, the blue springs of the shell was made shrink by reducing the masses of the corner particles uniformly. Since the blue and green springs contract at different rates, realistic wrinkles appear in our simulation result (c). However, the shell in (d) lost its details, because the springs constrained by the masses and springs simply extend or compress.

#### 4.1.8.1 Computation Times

The simulation of shell dynamics takes most of the computation time, because it involves the solution of a linear system with the conjugate gradient method at each projection step $j$ and time-step $n$. The bottleneck in this solution process is the matrix-vector multiplication in the conjugate gradient method. To speed up the computation, we compress the matrix into the ELLPACK format [23], which separates values from

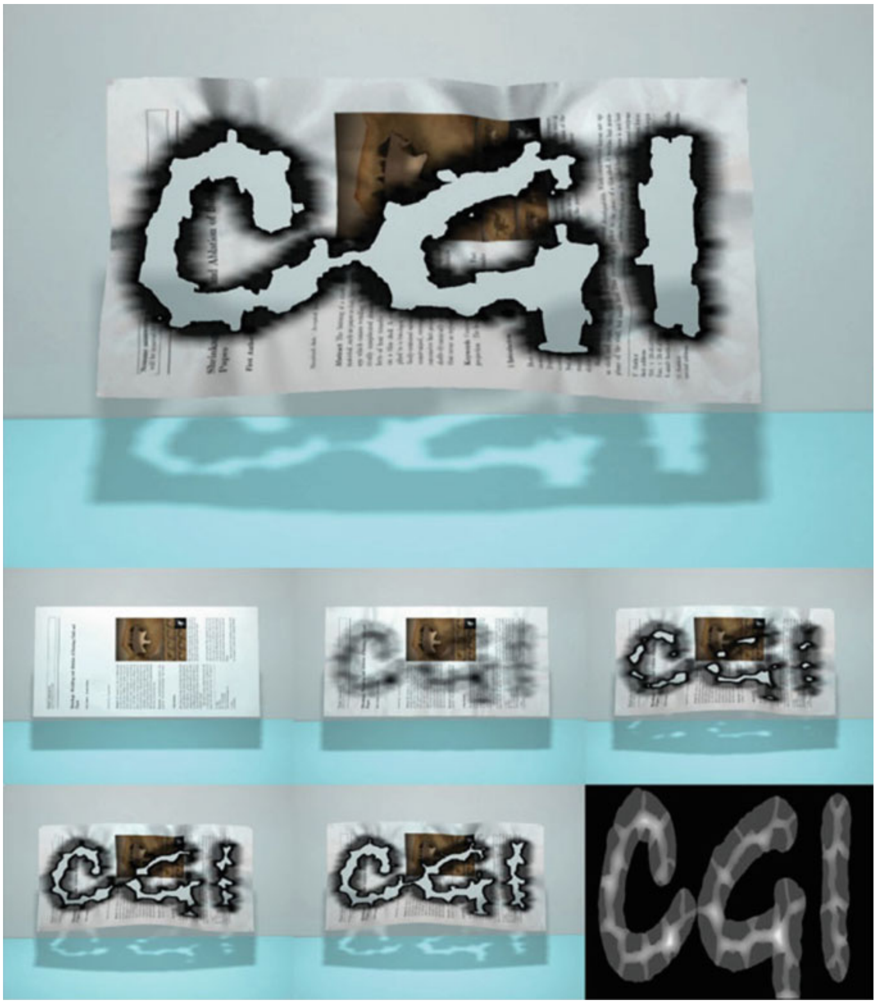


**Fig. 4.11** Simulation of burning paper

indices. This reduces the size of the matrix from $N_s^2$ to $N_s N_n$, where $N_s$ is the number of springs and $N_n$ is the maximum number of neighbors of a spring. Our shell structure can make good use of this format, because each spring set is a quadrilateral mesh and the number of neighbors of each spring is fixed ($N_s = 7$). Using the ELLPACK structure, the computation time is linearly proportional to the number of springs. Table 4.2 shows computation times against the number of springs.

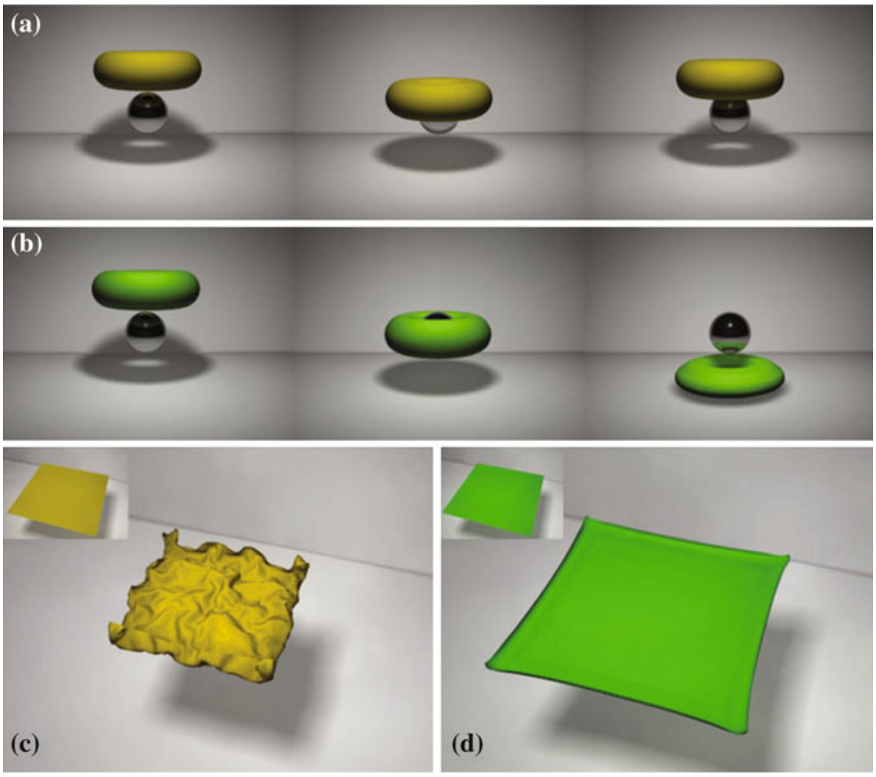


**Fig. 4.12** Our robust strain-limiting method preserves the rest shape of thin shells and the details of wrinkles produced by shrinkage. In (**a**) and (**b**), a torus has fallen on a *sphere* whose radius is 1.2 times bigger than *hole* in the torus. In (**b**), the torus simulated by the mass-spring system passes the sphere and contacts ground plane since it does not preserve the target length. In (**c**) and (**d**), the whole thin shell shrinks uniformly, but only the shell simulated by our method (**c**) creates complicated wrinkles, whereas the shell in (**d**) loses its details

### 4.1.8.2 Limitations

Successive fast projection is based upon the premise that each spring set is a stable quadrilateral structure. If a spring set contains points with more than six neighbors, then the simulation does not converge. Therefore, it is not straightforward to apply our method to adaptive BCS, BCC, or irregular meshes.

Ashes in burnt regions are easily torn and broken. We simulate this effect by reducing mass more quickly where we want the shell to break, as shown in Fig. 4.1. However, this phenomenon is not only caused by loss of mass. When two adjacent regions shrink at different rates, the region which shrinks more slowly prevents the other region from shrinking at its own speed. This creates tensile stresses in the more quickly shrinking region. If the tensile force is larger than the strength of the material

**Table 4.2** Computation times for thin shells with different numbers of springs

| (a) | (b) | (c) | (c)/(a) |
|---|---|---|---|
| 1,000 | 6.50 | 142.46 | 0.142 |
| 4,000 | 27.93 | 362.69 | 0.090 |
| 16,000 | 159.17 | 1268.00 | 0.079 |
| 25,000 | 211.67 | 1877.92 | 0.075 |
| 10,000 | 1045.02 | 7412.92 | 0.074 |

The time-step $h$, the maximum numbers of iterations, and the threshold of the fast projection are 0.002, 7, and $10^7$. The computation times and the number of springs are linearly proportional to each other. These tests were performed on the quad-core machine with 4 GB of memory. (a) The number of springs, (b) the computation time (ms) with convergence conditions, (c) the computation time (ms) without convergence conditions (simulations fully iterate), (d) the average computation time per spring ((c)/(a))

left in this region, the shell tears. However, we did not model this effect, and so our examples exhibit too many wrinkles and too few tears.

The proposed shell structure maintains its thickness using only red springs. Therefore, the rapid application of a large force will make the point on one layer move into the other layer. Since we do not explicitly correct the orientation of springs, the points in the other layer remain fixed because of the red springs producing sharp crease in the shell. This problem can be fixed by collision detection, but we do not apply this because it helps to create complicated wrinkles on the shell.

### *4.1.9 Conclusions*

In this section, we generated realistic fine wrinkles on a burning shell using a new shell structure and a method of strain limitation. A two-layer shell with thickness was created with a body-centered square (BCS) structure, and this shell wrinkled in response to stresses produced by the change in material properties that occur during burning. During successive fast projection, we dealt with subsets of the springs individually, allowing our overconstrained structure to converge. Remeshing burnt regions and setting parameters using textures improved the reality of our simulation.

## 4.2 Combustion Waves on the Point Set Surface

**Abstract** This section introduces a combustion model of heat transfer and fuel consumption for the propagation of a fire front on a point cloud surface. The heat transfer includes the heat advection by the airflow as well as diffusion, chemical reaction, and heat loss to generate complex, but controllable heat flows with a designed airflow velocity. For the stable heat advection, we solve a semi-Lagrangian method on point

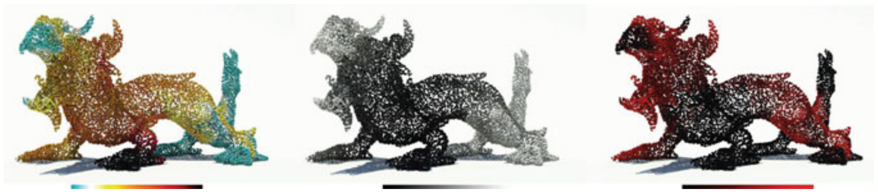


**Fig. 4.13** Simulation of combustion waves on the surface of a point cloud: Heat transfer, fuel consumption, and reaction zones from the *left*. The heat transfer model includes diffusion and advection with airflow velocity fields to generate complicated flows

samples using discrete exponential maps to trace the position from which the wind blows while preserving the geodesic distance. This section also proposes angular Voronoi weights for a discrete Laplace–Beltrami operator that shows better isotropic diffusion on the inhomogeneous distribution of point clouds than the cotangent or moving least-squares schemes. A diversity of burning scenarios are demonstrated by incorporating factors affecting the fire spreading such as buoyancy and object geometries in the airflow velocity fields, or by synthesizing patterns.

### *4.2.1 Introduction*

Combustion waves represent the propagation of reaction zones across combustible material. Finding these chemical reaction zones is central to the simulation of burning scenes: Heat and the gas fuels produced in the zone become the source of fire [24–26], and the burnt surface decomposes or deforms [2, 3, 27, 28].

The borders of a fire front are complicated, reflecting turbulent fire flows. Williams [29] noted that the spread of fire is influenced by many factors, including object geometry, airflow, buoyancy (for upward flows), and oxygen concentration. Most of these factors are intuitive; we know that fire spreads more rapidly upward and propagates in the direction of the wind.

Recent techniques [3, 27, 30, 31] for modeling the propagation of burning regions have largely been based on diffusion; but diffusion delivers heat energy evenly in all directions, making the fire front unrealistically smooths, regardless of environmental conditions or object geometries. Conversely, the modeling of convection requires the simulation of complicated flows of fluids [26, 32] such as air, heat, smoke and fire, and the exchange of heat and fuels between fluids and solids [1, 28, 30, 33]. Physics-based simulations of convection provide accurate and detailed representations of fluid flows. However, their high-computational cost prevents them from being used with a high-resolution mesh, because the complexity of the fluid motion depends on the resolution of the 3D grid.

We take a new approach, in which we calculate the heat flow directly from a 3D model, rather than coupling the model to a 3D grid to obtain the fluid velocity and

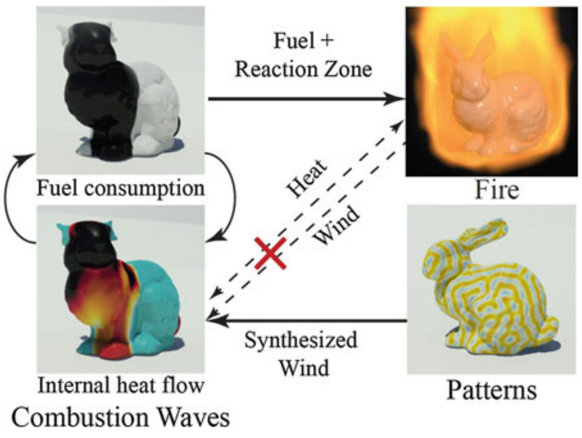


**Fig. 4.14** The simulation process: Instead of simulating convection using a grid, we synthesize fluid flow vectors and other factors which affect heat transfer. The output of our combustion model can be post-processed to produce a fire simulation

heat transfer (Figs. 4.13 and 4.14). Stam [34] simulated flows on arbitrary surfaces realistically by solving the equations of stable fluid motion [32] in a 2D texture space on the 3D surface. Other flow models have used triangular meshes [35], but we use a point cloud, which is more general, but obliges us to address issues caused by the lack of connectivity and the irregular distribution of points.

In this section, we mainly focus on the spread of a fire front on the surface of a burning object. We simulate the heat propagation using numerical models [36] of combustion waves, which integrate fuel consumption and heat transfer. The latter includes diffusion, chemical reaction, heat loss, and advection. Advection in particular is a key process in turbulent fire flows. This technique also allows users to design velocity fields with complicated patterns for artistic purpose. The contributions of this section can be summarized as follows:

- The introduction of an angular Voronoi weight for the Laplace–Beltrami operator for isotropic diffusion on an unorganized cloud of points;
- Modeling stable heat advection in point clouds by a semi-Lagrangian method with discrete exponential maps;
- Control of complicated heat flows by designing fluid velocities.

### 4.2.2 Related Work

In early work on fire spreading, the fire front was represented by a set of points which propagated across a polygonal surface, causing heat to be conducted within the burning object [37–40]. A multiscale fluid field [38] allows small-scale turbulence to be separated from large-scale flows. The lattice Boltzmann model (LBM) [40, 41] is used to generate fluid velocity fields to move flame particles around burning objects.

Physics-based techniques [30, 33] simulate the heat conduction within an object using the Laplace equation, while the motion of fluids such as air convection or fire are modeled by the stable fluids [32]. As fire propagates over solids, the burning object decomposes, shrinking, and deforming. The level-set method can be employed to track changes in the object's boundaries as fuel is consumed [1, 28, 31]. The changing shape of the burning object is modeled as free-form deformations [2, 3] or cloth simulations [27].

Regarding techniques for modeling fluids flowing over the surface of a 3D object on the discretization of that surface, Stam [34] modeled flows over arbitrary surfaces by solving the stable fluids [32] in curvilinear coordinates over Catmull–Clark surfaces; Um et al. [42] solved the Navier–Stokes equations in a 2D domain on a deformable 3D surface to model the absorption of water by a solid. However, these methods require additional analysis and computation to model the distortions and the boundary conditions linking parameterizations in 2D and 3D. To overcome these problems, Shi and Yu [35] directly simulated inviscid flows on a triangular mesh. They tracked points in the fluid as it moves over the triangular mesh to provide data to model semi-Lagrangian advection, and solved the discrete Poisson equation for incompressible fluids. Wang et al. [43] solved generalized shallow wave equations on a height field over the triangular mesh to model the phenomena associated with water moving over a surface. The LBM [44] method can model a fluid flowing over a point distribution, but it requires local point neighborhood information. Auer et al. [45] converted the surface flow problem into partial differential equations by embedding a 3D surface model in an adaptive grid using the closest point method.

Fluid flowing over deformable surfaces poses distinct problems. Neill et al. [46] simulated a fluid flowing over deforming and interacting surfaces using an efficient interpolation technique on a triangular mesh. Angst et al. [47] introduced an efficient algorithm to solve the shallow water equation on a deforming triangular mesh.

### 4.2.3 Combustion Model

Combustible material has properties such as a pyrolysis temperature $T_{\text{pyr}}$ and an ignition point $T_{\text{ign}}$, a diffusion coefficient $D$, a heat loss rate $h$, a heat production rate $\alpha$, and a fuel consumption rate $\beta$. If we consider cellulose, for which $T_{\text{pyr}} = 220\,\text{K}$ and $T_{\text{ign}} = 230\,\text{K}$ can make a useful simplification assuming that $T_{\text{pyr}}$ is the same as $T_{\text{ign}}$. The oxygen concentration also affects combustion, but we will assume that enough oxygen is always available.

The travel of combustion waves is governed by the equations for the conservation of heat $T$ and solid fuel $F$ [36]:

$$\frac{\partial T}{\partial t} = \underbrace{D\nabla^2 T}_{\text{diffusion}} + \underbrace{\mathbf{w}\cdot\nabla T}_{\text{advection}} + \underbrace{\alpha R(F,T)}_{\text{reaction}} - \underbrace{h(T - T_{\text{amb}})}_{\text{heat loss}} + \underbrace{T_{\text{ext}}}_{\text{external}}, \tag{4.20}$$

$$\frac{\partial F}{\partial t} = -\beta R(F,T), \tag{4.21}$$

$$R(F, T) = \begin{cases} Fe^{-1/(T - T_{\text{ign}})} & \text{if} \quad T > T_{\text{ign}} \\ 0 & \text{else} \end{cases}. \tag{4.22}$$

These equations describe the transfer of heat energy (Eq. 4.20), the loss of solid fuel (Eq. 4.21), and the Arrhenius equation for the chemical reaction rate (Eq. 4.22). The heat transfer model in Eq. (4.20) includes diffusion, airflow-guided advection, heat generated by reactions, heat loss, and heat absorbed from external sources. The process of burning from ignition to extinguishment can be modeled as follows:

**Pyrolysis and ignition**: Pyrolysis $\beta R(F, T)$ is the process by which a solid fuel is decomposed into ash, gas fuels, and other chemicals by absorbing heat energy. Ignition is an exothermic process $\alpha R(F, T)$ in which gas fuels are oxidized at temperatures above their ignition point. These chemical reactions start when the material's temperature reaches $T_{\text{pyr}}$ or $T_{\text{ign}}$, respectively, as it absorbs heat from external sources, at a temperature $T_{\text{ext}}$ such as an adjacent fire. We call the regions with $R(F, T) > 0$, *reaction zones* (Fig. 4.15c).

**Heat transfer**: Simplified heat models rely on thermal diffusion $\nabla^2 T$, but our combustion model also includes heat advection $\mathbf{w} \cdot \nabla T$ by fluid flow. This advection term makes the combustion waves chaotic, as found in real burning phenomena, rather than concentric. Because the temperature increases in the direction of the airflow $\mathbf{w} \cdot \nabla T > 0$ and decreases when the airflow is opposed $\mathbf{w} \cdot \nabla T < 0$, the heat follows the direction of the airflow.

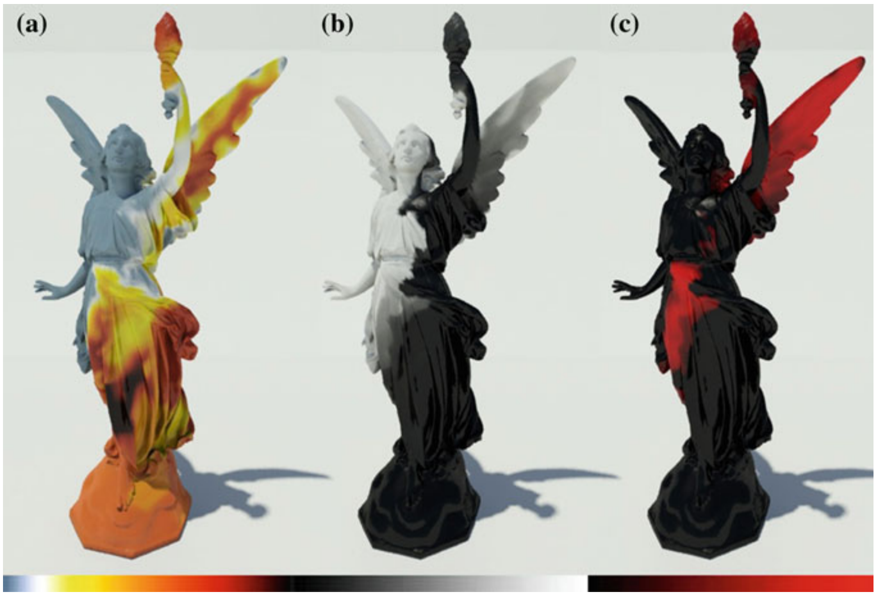


**Fig. 4.15** Characteristics of a combustion waves: **a** temperature, **b** solid fuel remaining, **c** and reaction zones

**Extinguishment**: When the solid fuel is completely consumed, $F = 0$, or the temperature drops through heat loss $-h(T - T_{\text{amb}})$, or heat transfer, so that $T < T_{\text{ign}}$. Then the chemical reactions cease, the fire is extinguished, and the reaction zone is empty, $R(F, T) = 0$.

We apply this combustion model to surfaces represented by a discrete-point clouds. The main problems in doing this are the diffusion and advection terms, which require a gradient. This is a partial derivative of position, and is distorted by irregular distribution of the points in the cloud. Stability of numerical methods is also an important issue.

We model diffusion on the surface of a point cloud using a Laplace–Beltrami operator with an angular Voronoi weight, and semi-Lagrangian advection on discrete exponential maps. This approach shifts the main numerical problem from that of determining gradients on the surface of a point cloud to one of local optimization and point tracking on a discrete-point geometry.

#### 4.2.3.1 Discrete Exponential Maps

We represent a 3D object as a set of discrete points $\{p_i\}$. A point $p_i$ has a position $\mathbf{x}_i$, a fuel mass $F_i$, a temperature $T_i$, an air flow $\mathbf{w}_i$, a normal $\mathbf{n}_i$, and an exponential map $e_i$. The 1-ring neighbor points $q_j \in N_1(p_i)$ are the $k$-nearest neighbors of $p_i$; this form of neighborhood computationally more efficient than $\varepsilon$-neighborhoods [48].

An exponential map $e$ is defined by a tangential plane, spanned by two basis vectors, $\mathbf{e_x}$ and $\mathbf{e_y}$, which are perpendicular to the normal $\mathbf{n}$ (Fig. 4.16a) at the position $\mathbf{x}$ of a point $p$. The two bases and the normal can be expressed as follows:

$$\mathbf{e_y} = \langle \mathbf{n} \times (\mathbf{x}_0 - \mathbf{x}) \rangle, \qquad \mathbf{e_x} = \langle \mathbf{e_y} \times \mathbf{n} \rangle, \tag{4.23}$$

$$\mathbf{n} = \left\langle \sum_{j \in N_1(p)} (\mathbf{x}_j - \mathbf{x}) \times (\mathbf{x}_{j+1} - \mathbf{x}) \right\rangle, \tag{4.24}$$

where $\mathbf{x}_j$ denotes the position of the $j$th neighbor point $q_i$ of $p$. The operator $\langle \mathbf{a} \rangle = \mathbf{a}/||\mathbf{a}||$ denotes the normalization of a vector $\mathbf{a}$.

The positions of neighbor points $\{\mathbf{x}_j\}$ in the local coordinates, which are $\{\bar{\mathbf{x}}_j\}$ (Fig. 4.16b), are found by projection onto the exponential map while preserving their

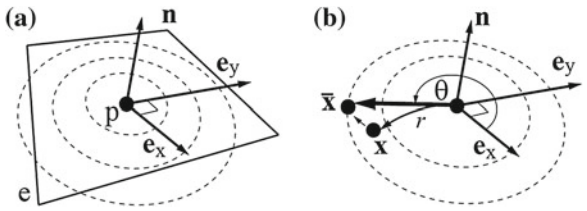


**Fig. 4.16** An exponential map e and **b** the projection of **x** to $\bar{\mathbf{x}}$ on **e**, while preserving the geodesic distance $r$–$p$

geodesic distance to $p$. The projected $\bar{\mathbf{x}}_j$ of $\mathbf{x}_j$ on the exponential map $e$ is found as follows:

$$\bar{\mathbf{x}}_j = \text{project}(e, \mathbf{x}_j) = \langle \mathbf{e_x}(\mathbf{e_x} \cdot \dot{\mathbf{x}}) + \mathbf{e_y}(\mathbf{e_y} \cdot \dot{\mathbf{x}}) \rangle ||\dot{\mathbf{x}}||, \tag{4.25}$$

where $\dot{\mathbf{x}} = \mathbf{x}_j - \mathbf{x}$. The point $\bar{\mathbf{x}}_j$ can be represented in the polar coordinates $(r_j, \theta_j)$, where:

$$r_j = ||\bar{\mathbf{x}}_j||, \tag{4.26}$$

$$\theta_j = \cos^{-1}\left(\frac{\mathbf{e_x} \cdot \bar{\mathbf{x}}_j}{||\bar{\mathbf{x}}||}\right) \text{sign}(\mathbf{e_y} \cdot \bar{\mathbf{x}}_j). \tag{4.27}$$

#### 4.2.3.2 Angular Voronoi Weights for Isotropic Diffusion

In isotropic diffusion, points at the same geodesic distance from a heat source receive the same amount of heat energy. This is easy to model on a uniform grid, but isotropic diffusion on irregular discrete points is more difficult, because the distribution and connectivity of the points distorts the gradient and the Laplacian.

We use a Laplace–Beltrami operator to simulate heat diffusion on a point cloud surface. The Laplacian $L(\mathbf{x}) = \nabla^2$ of a point $p_i$ can be approximated by the solution of a local optimization problem using a normalized weight function $W(\mathbf{x})$ for the positions of $p_i$ and its 1-ring neighbors $q_j \in N_1(p_i)$. We can represent a discrete Laplace–Beltrami operator in a general form as follows:

$$L(\mathbf{x}) = \sum_{j \in N_1(i)} W_j(\mathbf{x})(T_j - T_i). \tag{4.28}$$

The effect of a discrete Laplace operator depends on the weight function $W_j(\mathbf{x})$. For diffusion on a regular grid [32] or the umbrella operator [22], the weight function is $W_j(\mathbf{x}) = 1/\#N_1(p_i)$, where $\#N_1(p_i)$ is the number of 1-ring neighbors of $p_i$. However, when points are irregularly distributed, the underlying discrete-point geometry needs to be included in the weight function.

The positions of neighbor points on the exponential map have two degrees of freedom: (1) The distance $r$ from the center $p_i$ and (2) the angle $\theta$ between neighbors. The scale-independent Laplacian [22] incorporates the inverse of distance into its weight function:

$$W_j^{\text{SIL}}(\mathbf{x}) = \frac{2}{E}\frac{1}{||\mathbf{x}_j - \mathbf{x}_i||}, \qquad E = \sum_{j \in N_1(p_i)} ||\mathbf{x}_j - \mathbf{x}_i||. \tag{4.29}$$

It modifies the influence of each neighbor, depending on its distance $r$ from the center $p_i$.

In addition, the angle $\theta$ between neighbors also can be irregular. For example, when five points $q_i$ are the same distance from $p$, as shown in Fig. 4.17a, the

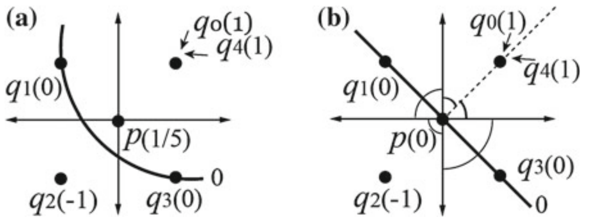


**Fig. 4.17** The values at $p$ obtained from a moving least-squares approximation are different when obtained from: **a** the Lagrangian or **b** Eulerian sampling of fire neighbors $q_j$ at the same distance from $p$. The weight function needs to incorporate the influence of each point for Eulerian sampling

weighted average value at $p$ obtained by a moving least-squares approximation is $1/5$. However, we consider that the value at $p$ should be zero in this configuration, so the weights applied to the neighbors of $p$ are changed from $1/5(1, 1, 1, 1, 1)$ to $1/4(1/2, 1, 1, 1, 1/2)$, as shown in Fig. 4.17b.

A cotangent weighting scheme [49] is widely used when discrete Laplace–Beltrami operators are applied to 2D manifold meshes

$$W_j^{\cot}(\mathbf{x}) = \frac{\cot \alpha_j + \cot \beta_j}{2}, \tag{4.30}$$

where $\alpha_j$ and $\beta_j$ are two opposite angles of two triangles which share the edge $e_j$ between $p_i$ and $q_j$ (Fig. 4.18b). A normalized version of the cotangent scheme [22] is also used:

$$W_j^{\mathrm{Ncot}}(\mathbf{x}) = \frac{1}{\sum_j (\cot \alpha_j + \cot \beta_j)} (\cot \alpha_j + \cot \beta_j). \tag{4.31}$$

This function weights the contribution of a neighbor point $q_j$ by the length of the edge of the Voronoi region that separates $q_j$ from $p$. Thus weights $W_j^{\mathrm{Ncot}}$ applied to neighbor points depend on their distribution.

The cotangent weight function produces isotropic diffusion on the anisotropic mesh in Fig. 4.19a, because it ignores points along the diagonals of the mesh by allocating them zero weights ($\cot(\pi/2) = 0$), that would otherwise cause

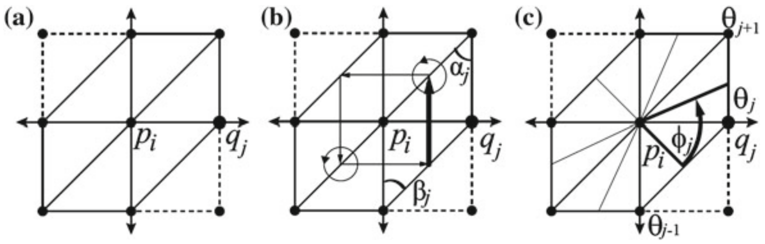
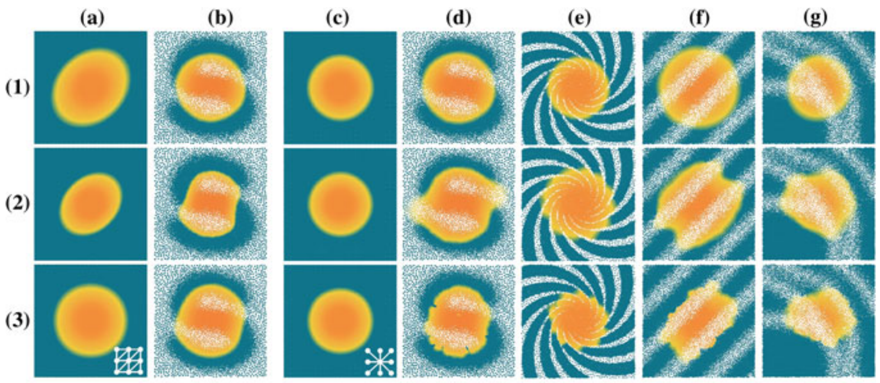


**Fig. 4.18** **a** The positions and scalar values of six neighbor particles of a particle $A$. The weights for the neighbor point $B$ are computed by **b** the cotangent scheme and **c** the angular Voronoi weight



| | (a) | | (b) | | (c) | | (d) | | (e) | | (f) | | (g) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.24 | 0.074 | 0.11 | **0.016** | 0.04 | **0.010** | 0.12 | **0.015** | 0.26 | **0.017** | 0.09 | **0.015** | 0.11 | **0.024** |
| 2 | 0.27 | 0.085 | 0.30 | 0.084 | 0.04 | **0.011** | 0.50 | 0.145 | 0.31 | 0.039 | 0.31 | 0.080 | 0.41 | 0.092 |
| 3 | 0.04 | **0.008** | 0.15 | 0.084 | 0.09 | **0.010** | 0.91 | 0.109 | 0.39 | 0.066 | 0.78 | 0.165 | 0.71 | 0.183 |

**Fig. 4.19** Comparison of diffusion results with (1–3) different weight functions and (**a**–**g**) distributions of points within a cloud. (1) The proposed angular Voronoi weight shows better isotropic diffusion on nonuniform distributions of point clouds, compared with (2) a MLS approximation and (3) the cotangent scheme. The pair of point clouds (**a**, **c**) and (**b**, **d**) have the same distributions, but in (a) and (b) the connectivity is computed by triangulation and in (**c**) and (**d**) by finding $k$-nearest neighbors ($k = 8$). The table shows the two errors: $\Delta d = (d_{max} - d_{min})/d_{mean}$ and $\varepsilon_d = |d_{max} - d_i|/(d_{mean} N_b)$ for each example

distortions. On point cloud surfaces, however, this function achieves poor diffusion (Fig. 4.19c–g), because $\cot(x)$ tends to $\pm\infty$ when two neighbors become too close or in the opposite directions.

Having seen the shortcomings of other weighting schemes, we use an angular Voronoi weight to achieve isotropic diffusion on point cloud surfaces. Our weight $W_j^{\mathrm{AV}}$ is a function of the average $\phi_j$ of adjacent angles of the edge between $p_i$ and $q_j$ (Fig. 4.18c):

$$W_j^{\mathrm{AV}}(\mathbf{x}) = \frac{\phi_j}{2\pi} = \frac{\theta_{j+1} - \theta_{j-1}}{4\pi}, \tag{4.32}$$

where $(r_j, \theta_j)$ are the polar coordinates of $q_j$ on an exponential map centered on $\mathbf{x}_i$, and $\phi_j = (\theta_{j+1} - \theta_j)/2 + (\theta_j - \theta_{j-1})/2$. We add $2\pi$ to $\phi_j$ if it is negative.

We can incorporate this angular Voronoi weight into the scale-independent Laplacian as follows:

$$L(\mathbf{x}) = \frac{2}{E} \sum_{j \in N_1(i)} \left( \frac{\theta_{j+1} - \theta_{j-1}}{||\mathbf{x}_j - \mathbf{x}_i||} \right) (T_j - T_i), \tag{4.33}$$

where $E = \sum_{j \in N_1(i)} (\theta_{j+1} - \theta_{j-1})/||\mathbf{x}_j - \mathbf{x}_i||$. This formulation reduces the distortion caused by the irregular distribution of neighbors in polar coordinates $(r, \theta)$.

The discrete Laplacian operator $L$ from Eq. (4.28) can be implicitly integrated [22]; and we can solve the resulting linear system using a conjugate gradient:

$$(\mathbf{I} - D\Delta t\mathbf{L})\mathbf{T}^{n+1} = \mathbf{T}^n, \tag{4.34}$$

where $\mathbf{T}$ is an $N^p \times 1$ vector of the temperatures of $N^p$ points, $\mathbf{L}$ is an $N^p \times N^p$ matrix that contains the weight values for the Laplacian operator in Eq. (4.33), and $\mathbf{I}$ is the identity matrix.

#### 4.2.3.3 Semi-Lagrangian Advection on Discrete Exponential Maps

The semi-Lagrangian method [32] has been widely used to simulate the advection of fluids; it is sample and stable with large time-steps. To approximate the quantity $f$ at the position $\mathbf{p}^{n+1}$ at the next time-step, this method traces the current velocity backward $-w^n \Delta t$ to obtain its position $\mathbf{q}^n$ at the current step, in which the quantity $f(\mathbf{q}^n)$ will flow from $\mathbf{q}^n$ into $\mathbf{p}^{n+1}$. Thus, determining the heat advection term $\mathbf{w} \cdot \nabla T$ is no longer gradient-related, but becomes a position tracking problem.

To calculate the advection term $\mathbf{w} \cdot \nabla T$ using a semi-Lagrangian method applied to discrete points, we adopt discrete exponential maps [50] to track the position $\mathbf{q}$ by projecting the airflow velocity $-\mathbf{w}^n \Delta t$ on the surface of the point cloud, while the geodesic distance between $\mathbf{p}$ and $\mathbf{q}$ remains constant.

**Projection of airflow**: We first obtain the projection $\bar{\mathbf{w}}$ of the position $\mathbf{p} - \mathbf{w}^n \Delta t$ on the exponential map of a point $p$. When we calculate the projection $\bar{\mathbf{w}}$ of the airflow velocity on the exponential map $e$, its magnitude can change because only the airflow in the tangential direction affects the point $p$ (Fig. 4.20a):

$$\bar{\mathbf{w}} = -\Delta t(\mathbf{e_x} \cdot \mathbf{w}^n + \mathbf{e_y} \cdot \mathbf{w}^n). \tag{4.35}$$

**Projections of discrete exponential maps**: Starting from $p$, we successively trace the closest point among the neighbors $\{q_j^k\}$ of a point $p^k$ to $\bar{\mathbf{w}}$ at each projection step $k$, until the distance from $p$ to $p^k$ approaches $||\bar{\mathbf{w}}||$ (Fig. 4.21). Since we project

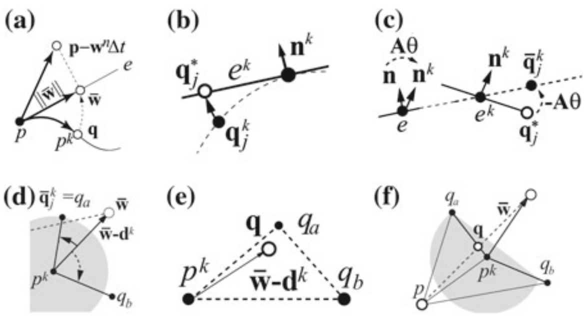


**Fig. 4.20** **a** Projection of airflow, **b** and **c** projections of discrete exponential maps, **d** advance to the next step, **e** and **f** termination conditions

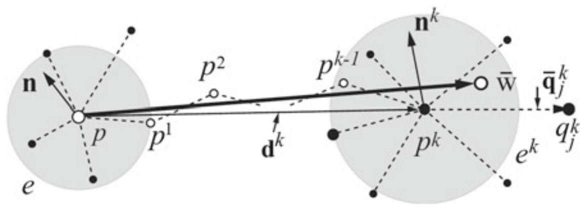


**Fig. 4.21** Point tracking of $\bar{\mathbf{w}}$ on discrete exponential maps

points on the local exponential map $e^k$, all positions are relative to the center point of $e^k$. Thus, we use $\mathbf{d}^k$ to represent the position of $p^k$ relative to the center $p$ of the exponential map $e$, and $\mathbf{d}^0 = \mathbf{0}$.

Given a point $p^k$ at projection step $k$, we project the positions $\{\mathbf{q}_j^k\}$ of neighbor points on the local exponential map $e^k$ to bring them on to its tangent plane (Fig. 4.20b):

$$\mathbf{q}_j^* = \text{project}(e^k, \mathbf{q}_j^k). \tag{4.36}$$

Since the normals of the two exponential maps $e$ and $e^k$ are different, we must factor in the rotation from $e$ to $e^k$ to project the neighbor points $\{\mathbf{q}_j^*\}$ on the tangential plane of $e$. A rotation from $e$ to $e^k$ is defined by an axis-angle $\mathbf{A}\theta$, where $\mathbf{A}$ and $\theta$ denote the axis and the angle between two normals $\mathbf{n}$ and $\mathbf{n}^k$. Then, $\mathbf{q}_j^*$ is transformed as follows (Fig. 4.20c):

$$\bar{\mathbf{q}}_j^k = \text{Rot2D}(\mathbf{q}_j^*, -\mathbf{A}\theta). \tag{4.37}$$

**Advance to the next step**: We find the closest neighbor $q_j$ to the airflow vector $\bar{\mathbf{w}} - \mathbf{d}^k$ on $e^k$ which maximizes (Fig. 4.20d)

$$q_a = \arg\max_j \left( \frac{\bar{\mathbf{q}}_j^k \cdot (\bar{\mathbf{w}} - \mathbf{d}^k)}{\|\bar{\mathbf{q}}_j^k\| \|\bar{\mathbf{w}} - \mathbf{d}^k\|} \right). \tag{4.38}$$

We set the closest neighbor $q_a$ to $p^{k+1}$ to advance the projection to the next step. The position of $p^{k+1}$ relative to $p$, which is $\mathbf{d}^{k+1}$, is updated by the position of the closest neighbor, so that $\mathbf{d}^{k+1} = \mathbf{d}^k + \bar{\mathbf{q}}_j^k$.

**Termination conditions**: We find $q_b$ to define a triangle $(p^k, q_a, q_b)$ which intersects with the vector $\bar{\mathbf{w}} - \mathbf{d}^k$:

$$q_b = \begin{cases} q_{j+1} & \text{if} \quad \bar{\mathbf{q}}_j^k \times (\bar{\mathbf{w}} - \mathbf{d}^k) > 0 \\ q_{j-1} & \text{else} \end{cases}, \tag{4.39}$$

where the indexes of neighbors are arranged in a counterclockwise direction.

Given the triangle, we can calculate the barycentric coordinates $\mathbf{b}_w(b_0, b_1, b_2)$ of $\bar{\mathbf{w}} - \mathbf{d}^k$ in the triangle $(p^k, q_a, q_b)$, whose local positions are $(\mathbf{0}, \mathbf{q}_a, \mathbf{q}_b)$ on $e^k$, to determine whether further projection is required. If $0 \leq b_i \leq 1$, we terminate the projections and approximate the quantity $f(\mathbf{q})$ by barycentric interpolation of the quantities at the triangle points (Fig. 4.20e).

Also, if the angle between $q_a$ and $q_b$ is larger than a threshold, or we have already visited $p^{k+1}$, projection stops, because $\bar{\mathbf{w}}$ is outside the surface boundary. We look for the intersection between the airflow velocity $\bar{\mathbf{w}}$ and the edges $\overline{p^k q_a}$ and $\overline{p^k q_b}$. If $(\bar{\mathbf{w}} \times \mathbf{d}^k) \cdot (\bar{\mathbf{w}} \times \mathbf{q}_a) < 0$, then the intersection point $\mathbf{q}$ is on $\overline{p^k q_a}$; otherwise, it is on $\overline{p^k q_b}$. If we refer $q_i$ to $q_a$ or $q_b$, the quantity $f(\mathbf{q})$ is approximated by a linear interpolation of $f(p^k)$ and $f(q_i)$ using the weight $w$, which is obtained as follows (Fig. 4.20f):

$$w = \frac{\|\bar{\mathbf{w}} \times \mathbf{d}^k\|}{\|\bar{\mathbf{w}} \times \mathbf{d}^k\| + \|\bar{\mathbf{w}} \times (\mathbf{d}^l + \mathbf{q}_i)\|}. \tag{4.40}$$

#### 4.2.3.4 Airflow Velocity Fields

The airflow velocities at discrete points are determined from environmental and internal factors, and artificial velocity fields, as described below.

**Nonadiabatic space**: The divergence-free constraint is an important property for general fluid simulation, but we do not force this constraint in our fire spreading model, since we consider that the simulation space to be nonadiabatic. For example, in a gust of wind, a burning object can suddenly burst into flames, thereby increasing the total heat energy within the object, whereas an opposite wind might extinguish combustion. In our simulation, therefore, the heat energy in a burning object flows upward by buoyancy and disappears at the sink of the velocity fields, located at the apex of the object. Nevertheless, if we wished to force the airflow to be adiabatic, a divergence-free vector field for point samples could be obtained from the Helmholtz-Hodge decomposition [51].

**Environmental factors**: In our simulation, buoyancy forces is simply modeled by a constant vector in the opposite direction to gravity:

$$\mathbf{w}^{buoy} = -c^{buoy}\mathbf{g}. \tag{4.41}$$

**Curl from scalar fields**: Bridson et al. [52] introduced a curl-noise method that generates turbulent velocities from noisy scalar fields. Given a set of scalar values $\{f(p_i)\}$ assigned to the discrete points $\{p_i\}$, we can calculate the curl $\mathbf{c}_i$ from gradient $\nabla f(p_i)$ of $p_i$, and rotate the gradient $\pi/2$ around the normal $\mathbf{n}_i$:

$$\nabla f(p_i) = \frac{1}{E} \sum_{j \in N_1(p_i)} \Big( W_j^{AV}(\mathbf{x})(f(p_j) - f(p_i))(\mathbf{x}_j - \mathbf{x}_i) \Big), \tag{4.42}$$

$$\mathbf{c}_i = \mathrm{Rot2D}\Big(\nabla f(p_i), \frac{\pi}{2}\mathbf{n}_i\Big), \tag{4.43}$$

where $E = \sum_{j \in N_1(p_i)} W_j^{AV}(\mathbf{x})$, which normalizes the weight function. Then, the airflow velocity obtained from the curl is:

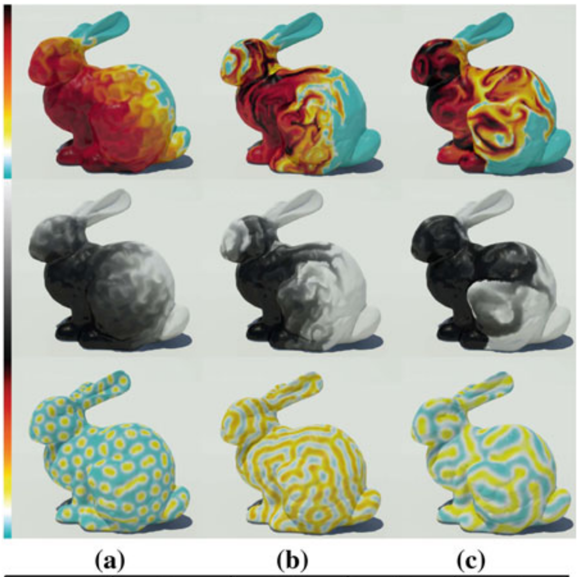
$$\mathbf{w}_i^{\text{curl}} = -c^{\text{curl}}\mathbf{c}_i. \tag{4.44}$$

Different flows can be generated using scalar fields created by formulas, textures, Perlin noise, or reaction–diffusion patterns. We use the Gray-Scott model [53] to produce the reaction–diffusion pattern as shown in Fig. 4.22.

**Geometry**: Airflow only affects a point in a direction which lies in the tangent plane of the surface of the point cloud. Therefore, the airflow velocity **w** obtained by summing all the factors is projected on the exponential map at the beginning of the advection step, using Eq. (4.35).

## *4.2.4 Results*

Most of the point cloud models used as examples were obtained from triangular meshes, which simplifies rendering. To visualize heat, fuel remaining, and reaction zones, a spectrum of colors was assigned to the attribute, and then the color was assigned corresponding to the value at each point. Then the triangular mesh was rendered with Mental Ray in 3ds Max. The random points shown in the examples



|  | $D_u$ | $D_v$ | $k$ | $F$ |
|---|---|---|---|---|
| **(a)** | 0.082 | 0.041 | 0.064 | 0.035 |
| **(b)** | 0.190 | 0.050 | 0.062 | 0.060 |
| **(c)** | 0.160 | 0.080 | 0.060 | 0.035 |

**Fig. 4.22** Parameters of the Gray-Scott model for advection with designed airflows using reaction–diffusion patterns

in Fig. 4.19 were generated using fast Poisson disk sampling [54] with adaptive distances, controlled by the intensity of a grayscale texture.

**Reaction zones**: Figure 4.13 shows the results produced by our combustion model on the surface of a point cloud; they show heat, fuel remaining, and reaction zones. Figure 4.15 explains some features of our combustion model. In previous techniques [1, 3, 28], reaction zones were considered as thin bands around the boundaries of burning regions which move in the direction of the gradient of the level-set, whereas reaction zones in our method can be of differing width, as shown on the chest and right wing of the Lucy model in Fig. 4.15c.

**Designing airflow**: As shown in Fig. 4.23a, the advection term can result in the fire front moving upward along the surface by buoyancy without any simulation in 3D. When the diffusion term is included, the heat spreads along the whole object surface by internal conduction. In Fig. 4.22, three different sets of parameters for the Gray-Scott model are used to generate the scalar fields for curl vectors. The resulting pattern of heat flow looks artificial, but might well be used for artistic purposes.

**Point tracking on discrete exponential maps**: To assess the accuracy of our semi-Lagrangian methods on discrete exponential maps, $q_j$ (red points) were projected regularly distributed around a center point $p$ on the Bunny model. The points were successfully projected on the curved surface (blue points), even on the sharp curves of the ears, as shown in Fig. 4.24.

**Isotropic diffusion**: Fig. 4.19 compares the results of diffusion by the cotangent weight, MLS, and the angular Voronoi weight on a 2D manifold mesh and on point clouds. The colors show the change in fuel $F$. A sphere-shaped external heat source is

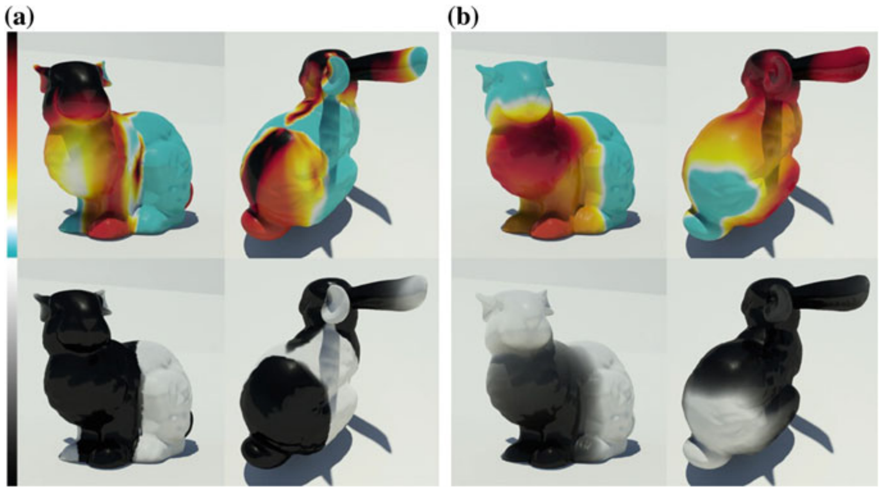


**Fig. 4.23** Changes in heat (*top*) and fuel mass (*bottom*) obtained from the combustion model: **a** advection by buoyancy only and **b** advection and diffusion simultaneously

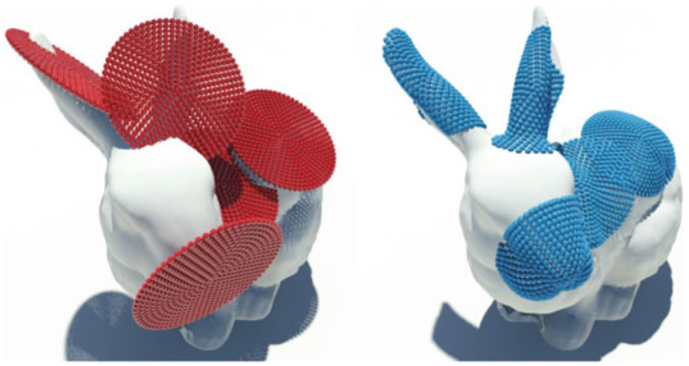


**Fig. 4.24** Regularly sampled points (*red*) on each exponential map are successfully projected onto the *curved* surface (*blue*) by our point tracking method

placed at the center and only diffusion and chemical reactions occur in the combustion model (Eq. 4.20).

On the triangular mesh, the angular Voronoi weights bias the boundaries of burnt regions in the direction of the edges. However, on point clouds, the boundary of the burnt regions modeled by angular Voronoi weights is nearly circular, whereas heat diffuses more slowly in dense areas using the MLS and the cotangent weight. We find that heat does not reach some points with the cotangent scheme, because the cotangent values tend to infinity as neighbors become too close.

The accuracy of isotropic diffusion was estimated using the distance $d_i$ between the center of the heat source and a point at the boundary of the burnt region in Fig. 4.19. The table below the figure shows the errors $\Delta d = (d_{\max} - d_{\min})/d_{\text{mean}}$ and $\varepsilon_d = \sum_i |d_{\max} - d_i|/(d_{\text{mean}} N_b)$ which are, respectively, deviations from circularity and from the actual circle around the center of the heat source. The distances $d_{\text{mean}}$, $d_{\min}$ and $d_{\max}$, $N_b$ denote the average, shortest, and largest values of $\{d_i\}$, and the number of boundary points. The errors $\Delta d$ and $\varepsilon_d$ are scaled by $1/d_{\text{mean}}$ for fair comparison. As the distribution of the points within a cloud becomes move ill-conditioned, the difference in errors between the proposed angular Voronoi weight and other methods become larger.

**Fire simulation by post-processing**: The results of our combustion model can be used as inputs to commercial CG tools for fire simulation. We use two textures for solid fuels and reaction zones. Each texture is used to change the color of smoldering regions, and to represent reaction zones, in which gas fuel and heat are being generated, respectively. In the solid fuel map (top-right in Fig. 4.25), the unburnt regions are dark. In the reaction zone map (bottom-right), we see that a line of fire separates burnt and unburnt material. Figure 4.26 shows a series of frames from an animation of burning cloth.

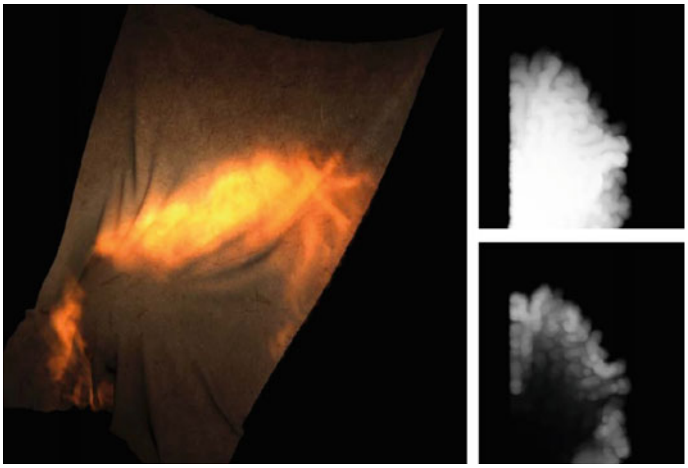


**Fig. 4.25** Two texture maps of solid fuel (*top*) and reaction zones (*bottom*) for the burning object (*left*)

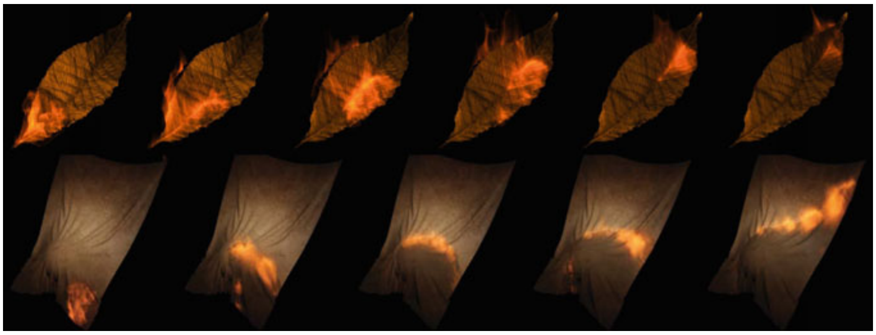


**Fig. 4.26** The simulation of fire by post-processing

**Computation times**: The combustion model was run on a multicore machine with an Intel Core i7-3660 processor and 16 GB of RAM. The simulation was parallelized by Open MP.

Table 4.3 shows computation times broken down between simulation steps. (A) When the simulation begins, (B) when the positions of points are updated, and (C) at each time-step, the attributes used for diffusion and advection are calculated. The most time-consuming operations are performed once in Step A. Step B only has to be performed once during initialization, if the point surface is stationary, but it must be calculated before Step C per frame if the geometry of the point cloud is changing. Figure 4.27 shows results obtained from our combustion model coupled with deformable surface.

**Table 4.3** Computation times (ms) for simulation steps

| Step | Model | Bunny | Lucy | Dragon |
|---|---|---|---|---|
| | No. of points | 34,835 | 50,014 | 99,932 |
| (A) | $k$-neighbors (8) | 724 | 2710 | 7274 |
| | Reaction–diffusion | 178 | 268 | 576 |
| | Sum | 902 | 2978 | 7850 |
| (B) | $\mathbf{n}$, $\mathbf{e}_x$, $\mathbf{e}_y$, $(r, \theta)$ | 154 | 130 | 402 |
| | $W^{AV}$ | 32 | 34 | 82 |
| | $W^{MLS}$ | 20 | 24 | 64 |
| | $W^{Ncot}$ | 94 | 104 | 282 |
| | Sum $(+W^{AV})$ | 186 | 164 | 484 |
| (C) | Chemical reaction | 11 | 11 | 42 |
| | Advection | 78 | 136 | 306 |
| | Diffusion | 34 | 162 | 514 |
| | Sum | 123 | 309 | 862 |

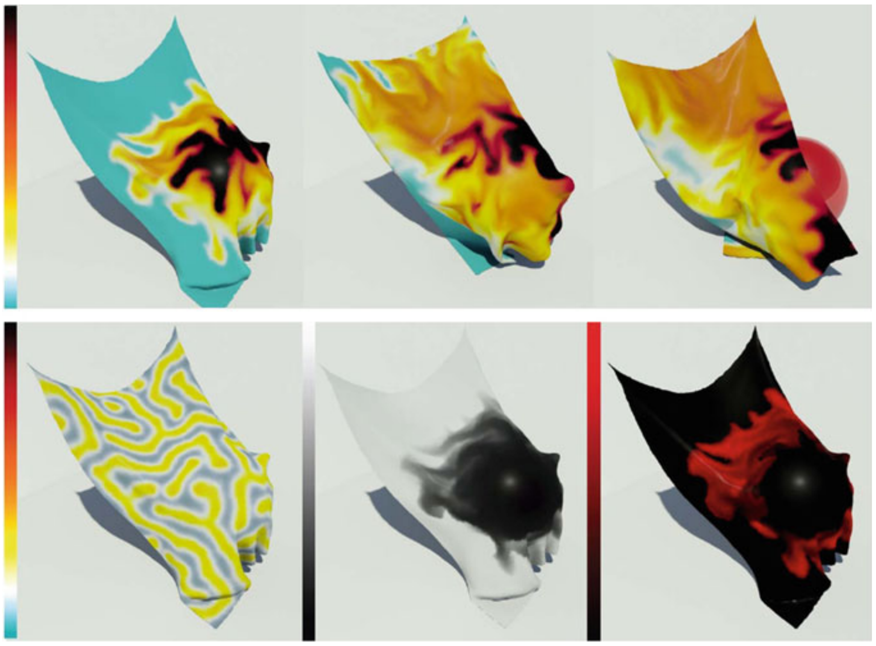


**Fig. 4.27** Combustion waves on draping cloth (*top*). Our combustion model can be applied to deformable bodies. The lower images show the corresponding pattern, fuel, and reaction zones of *top–left* image

### 4.2.5 *Conclusion*

We simulated combustion waves moving over point cloud surfaces. This model can generate a variety of realistic animations of spreading fire which compare favorably with existing diffusion-based methods. The angular Voronoi weight for a Laplace operator shows better isotropic diffusion than the cotangent scheme or a MLS on ill-conditioned point clouds. A semi-Lagrangian method on discrete exponential maps results in stable advection on curved Riemannian manifolds.

Our model of heat flow with designed airflows is not physically accurate, but it can generate complicated flows without the need for a 3D grid-based fluid simulation. It can be very suitable for creative use, because it allows designers to create a spreading fire in burning scenes by generating patterns or painting directly on the surface. Our combustion model can easily be integrated into the simulation of burning scenes by exchanging physical properties with 3D fluid simulations. Finally, because it is used on point clouds, our method can be applied to raw point data obtained from real-time capture systems such as 3D scanners.

## 4.3 A Particle-Grid Method for Opaque Ice Formation

**Abstract** This section presents a particle-grid method to simulate the generation of opaque ice which has air bubbles in it. Water temperature is diffused over a grid, and the exchange of dissolved air between ice and water particles is simulated. A particle is rendered as an air bubble if it has sufficient air. Otherwise, it is treated as a cloudy volume by distributing air into dissolved air field when the final state has been reached. In addition, the method includes a model in which heat transfer rate may change across the grid. Unlike previous models which could generate an ice volume of only fixed shapes, this approach uses signed distance function (SDF) to generate opaque ice volumes stored in containers of various geometric shapes and can render needle-shaped or egg-shaped bubbles.

### 4.3.1 *Introduction*

Ice found in daily life is often opaque due to air bubbles trapped inside while being frozen. However, existing techniques focused primarily on modeling of liquidification or solidification between ice and water. Rendering of ice found in movies often shows transparent and solid object with flat opaque texture with no volume (see Fig. 4.28). In order to render opaque ice volume, designers have to engage in tedious and time-consuming task of modeling behavior of air bubble particles inside of ice.

This section proposes a hybrid method to render opaque ice. We focuses mainly on behavior of water being frozen while stored in containers of various shapes. When rendered image do not suffer significantly in quality, we choose to ignore minor

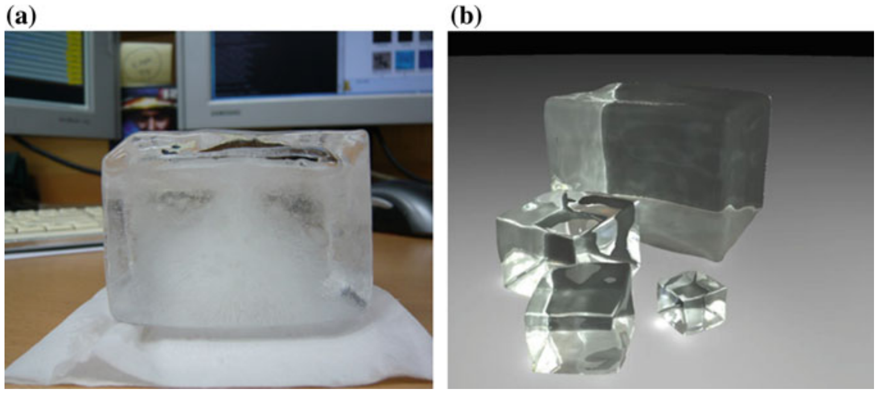


**Fig. 4.28** **a** Real ice and versus **b** transparent rendered ice

details to optimize the rendering process. We describe air bubbles found inside of an ice volume using both grid and particles. Amount of dissolved air found in ice defines the objects state. Ellipsoidal particles allow generation of either needle-shaped or egg-shaped bubbles depending on how quickly water is frozen. This model also includes an enhanced heat model with variable transfer rate.

### *4.3.2 Previous Work*

Phase changes in various types of fluids have been investigated by many researchers who developed techniques to capture melting or freezing process of solid or fluid objects [5, 55, 56]. However, they paid little attention on what happens when water is being frozen.

Kim et al. [57] developed a hybrid method for simulating frost over arbitrary surface by combining dynamic Lie algebra method [58], phase-field approach [59], and stable fluid simulation [32]. However, the technique focused only on generation of a thin frost layer, and it could not be applied on a volume of ice. Other papers addressed generation of an icicle [60, 61]. However, such techniques are suitable only for the generation of images in shapes similar to icicles or stalactites, and the generated icicles appeared transparent.

Hong et al. [62] proposed a method to simulate the movement and transformation of air bubbles inside of fluid object. It used grids to describe the fluid and particles for air bubbles, and it shows the merging and splitting of air bubbles in conjunction with a level-set method. However, this paper did not address phase changes taking place in water.

Madrazo et al. [63] simulated the generation of trapped air bubbles inside of an ice volume using only grid. Rendered results, whose qualities are largely dependent on the grid resolution, also appear monotonous because the method generated air bubbles only at the center of each grid square. Furthermore, the technique worked only on simulations of slowly freezing cubic ice, and bubbles lacked directional property. On the contrary, the technique in this section supports simulation of objects being frozen while stored in containers of various geometric shapes. Other advantages include (1) various attributes (e.g., direction and speed) can be modeled; and (2) air bubbles can be placed anywhere independent of the grid location.

Seipel and Nivfors [64] used textures to show opaque ice. The paper focused on real-time ice rendering including reflection and refraction. But, it used textures to render opaque area which is highly dependent on the view point and was limited to only convex geometries.

In order to accurately model directional movement of bubbles, we use ellipsoidal particle method as suggested by [65]. Jo et al. [66] improved the method to describe a smooth surface of fluid. In this section, we use particles with ellipsoidal kernel to accurately describe directional property of particles.

Carte [67] described the behavior of air bubbles trapped inside of an ice volume, but visualization has remained undone.

## 4.3.3 Overview

Realistic rendering of opaque ice requires deep understanding of properties of water as well as physical phenomenon that occur in water when temperature drops [68]. Generally, water contains various types of dissolved gases such as nitrogen, chlorine, and oxygen. These materials easily separate themselves from solvent as the temperature is lowered. When water kept in a container is frozen inside a freezer, the outer layer starts to freeze first because cold air first meets water at the containers surface. As the freezing ice layer becomes thick, dissolved air is repelled from the ice, thereby increasing the air volume toward the center of the container. The opaque region is created when there exists excess air inside the water volume and dissolved air can no longer be transferred externally.

Our method, illustrated in Fig. 4.29, consists of six steps. The initial and final steps, in green background, are performed on a grid, and all other steps, colored red involve transformations applied on particles. The grid is formed to model heat transfer through water and render particles as small cloudy air bubbles. Particles are used and classified each as water, frozen ice, or air bubble. As the amount of dissolved air around the center becomes high, dissolved air will be transformed either as air bubble particles or a group of small cloudy air bubbles.

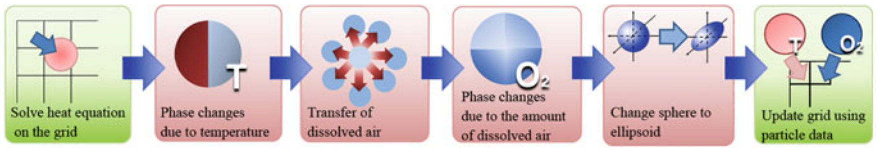

Solve heat equation on the grid
Phase changes due to temperature
T
Transfer of dissolved air
Phase changes due to the amount of dissolved air
O2
Change sphere to ellipsoid
Update grid using particle data
T
O2

**Fig. 4.29** System flow

The first step simulates heat transfer that occur across water and frozen ice using an enhanced equation over a grid. Then attributes of the particles and grid are updated in steps 2 through 5. Water particles update their temperature from the neighboring grid cells via interpolation. The amount of heat contained in a particle indicates whether it is frozen or not. Because ice cannot contain as much dissolved air as liquid water can; in the second step, we move repelled air from a frozen particle to nearby particles. Frozen particles search for nearby unfrozen particles to release dissolved air. Status of each water particle is updated next. Frozen particles are classified as a large air bubble or a group of small air bubbles depending on the amount of residual air. Next step computes the direction and shape of each particles depending on the freezing rate and the amount of dissolved air in it. The former follows that of ice being frozen while the latter is determined by how fast it is being frozen. Air volume is decided by the dissolved air the particle contains. The last step forms the dissolved air field by updating grid cells based on the amount of dissolved air contained in nearby frozen particles that are classified as groups of small cloudy air bubbles. This simulation is terminated when all water particles eventually become ice particles.

## *4.3.4 Simulation of Freezing Ice*

To model heat transfer across the freezing water, we solve a heat equation using finite difference method and Dirichlet boundary conditions. This equation can represent different rates of heat transfer depending on the status of water particles. Water particles fetch heat from the grid using trilinear interpolation.

Most of the heat transfer taking place in freezing water occurs by conduction through the solid frozen ice. During simulation, we ignore both heat transfer within water volume by convection and radiation heat transfer because we assume that (1) freezing occurs in a freezer where convection and radiation barely affects the freezing process, and (2) water is kept in a relatively small container.

### 4.3.4.1 Heat Transfer

Irregular patterns found in ice occur because the transfer rate of cold energy is different between liquid water and frozen ice.

Figure 4.30a, rendered using conventional heat equations, assumes the fixed heat transfer rate for all of the grid cells, thereby resulting in symmetric advection of heat. We enhance the heat equation to accommodate different heat transfer rates. Conventional heat transfer equation is

$$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = 0. \tag{4.45}$$

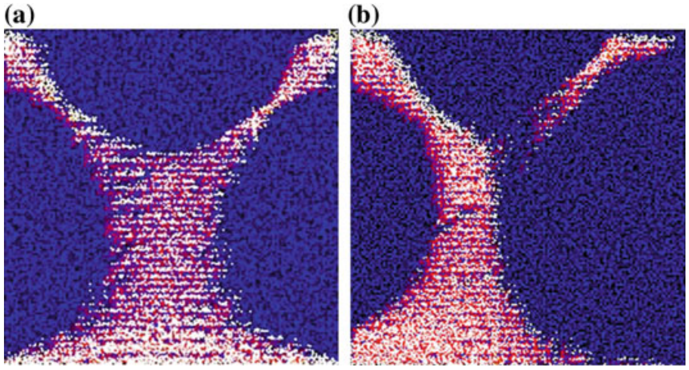


**Fig. 4.30** Comparison between **a** conventional heat equation and **b** our approach

In Eq. (4.45), $\alpha$ is the heat transfer constant, $u$ is temperature and $t$ is time, respectively. We enhance it to more accurately capture physical phenomenon

$$\alpha = (1 - W)\alpha_{water} + W\alpha_{ice}, \tag{4.46}$$

$$W = \frac{P_{ice}}{P_{ice} + P_{water}}. \tag{4.47}$$

$\alpha$ is the heat transfer rate modifier for a grid cell. $\alpha_{water}$ and $\alpha_{ice}$ are the heat transfer constants for water and ice, respectively. $W$ is the ratio on the number of ice particles to the total number of particles in the grid cell. To calculate $\alpha$, we check the status of particles inside the grid cell and adjust its value according to $W$. When there are fewer frozen particles than water particles, $\alpha$ will be assigned a higher value. Conversely, it will have a lower value when there are more frozen particles than water particles.

Figure 4.30b shows the rendered image based on revised heat transfer equation. Compared against the counterpart image, irregular distribution of heat transfer is visually apparent.

To update the particle status, we have to transfer heat contained within a grid cell to particles in the nearby cells. Amount of heat contained in each particle is computed using information obtained from weneighboring cells using trilinear interpolation. Each particle that has fetched heat from the grid will then be classified as either a water or an ice particle. When the temperature falls below 0, it becomes an ice particle.

### 4.3.4.2 Transfer of Dissolved Air

When a particle is frozen, it repels dissolved air toward nearby particles. Algorithm 5.1 is the pseudocode to model such behavior. It computes the distribution of dissolved air carried by a recently frozen ice particle $D_{P_i}$ into nearby water particles $D_{P_j}$. Amount of air transferred from particle $P_i$ to $P_j$ is computed by $\Delta D_{P_j}$:

$$\Delta D_{P_j} = \frac{\sum_i W_{ij} \Delta D_{P_i}}{\sum_i W_{ij}}, \tag{4.48}$$

where

$$\Delta D_{P_i} = \min\left(\frac{\sum_j W_{ij} D_{P_i}}{\sum_j W_{ij}}, D_{P_i}\right),$$

$$W_{ij} = \frac{1}{d_{ij}} \quad (d \le 1).$$

$\Delta D_{P_j}$ is determined by the distance between the particles $d_{ij}$, which adjusts the weight $W_i$ and the amount of the air particles $P_i$ can provide. Upon completion of this step, there exist ice particles containing excessive volume of air. Such air will separate itself out of ice and form an air bubble trapped inside. When dissolved air $D_{P_i}$ in the $i$th ice particle exceeds $Th_{air}$, it is rendered as an air bubble (Table 4.4). When the value remains between 0 and $Th_{air}$, air in the ice particles will be transferred to the dissolved air field and rendered as small, cloudy air bubbles. Should the threshold value be increased, bubbles become smaller and ice becomes increasingly foggy.

**Algorithm 5.1** Transfer of dissolved air

```
for All particles do
  if The phase the $i$th particle is ICE then
    for Every neighboring particle around the $i$th particle do
      if The phase of the $j$th neighbor of the $i$th particle is WATER then
        $D_{P_j} = D_{P_j} + \Delta D_{P_j}$ (Eq. 4.48)
        $D_{P_i} = D_{P_i} - \Delta D_{P_j}$
      end if
    end for
  end if
end for
```

#### 4.3.4.3 Generation of Directional Air Bubbles

As water transforms into ice, it transfers the dissolved air to nearby water particles. As the speed of the freezing process is faster than that of the air being transferred, air will become a bubble aligned to the direction of ice growth (See Fig. 4.31a). The volume depends on the amount of the dissolved air, and the shape is affected by the direction and speed of the freezing process. At faster freezing rate, bubbles will form a thinner and sharper shape. We use ellipsoidal particles to generate directional bubbles.

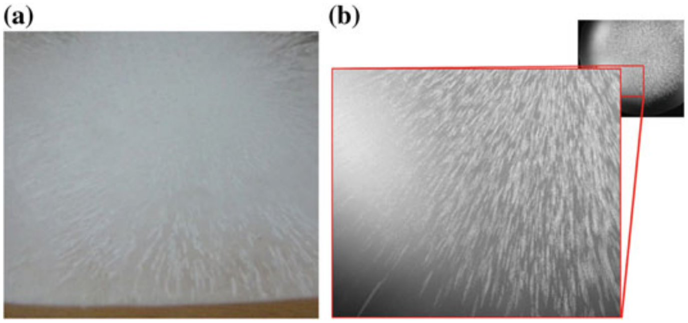


**Fig. 4.31** **a** Air bubbles shaped like needles appearing in the fast-frozen ice. **b** Our simulation of air bubble shapes like needles

**Table 4.4** Phase dependency on dissolved oxygen (DO)

| State | Condition | Description |
|---|---|---|
| DO field | $0 < D_{P_i} < Th_{air}$ | Stores DO in the grid |
| Air bubble | $Th_{air} \leq D_{P_i}$ | Forms an air bubble |

Exact process on how bubbles are created inside of an ice volume is still being studied [69]. In our method, bubble shape and volume are assumed to depend on the freezing speed and the amount of dissolved air [67]. Following equation is used to describe directional property of bubbles:

$$V = \frac{4\pi}{3} p^2(xz)q(y), \tag{4.49}$$

where $q = 1 + F$ and $p = \sqrt{1/q}$. The bubble ellipsoid has two scale factors, $q$ which is aligned to the direction of the growth of the along $y$ axis, $p$ for the rest. $V$ is the volume of air bubble which is calculated by a simple scalar multiplication of the amount of the dissolved air associated with the particle. $F$ is the freezing rate, which ranges from 0 to 1. Using the scale factors, we derive the following matrix for scaling.

$$\mathbf{S} = \begin{bmatrix} p & 0 & 0 \\ 0 & q & 0 \\ 0 & 0 & p \end{bmatrix} \tag{4.50}$$

Bubble is then aligned with the growth of ice. We generate a matrix which rotates y axis of the particle to match the direction of the growth. The particle will then be transformed with both matrices to produce a directional property (See Fig. 4.32).

Figure 4.31b shows a result of bubbles with directional property using ellipsoidal particles. We assume the same heat transfer rate over entire surface, and the result shows bubbles pointing inward.

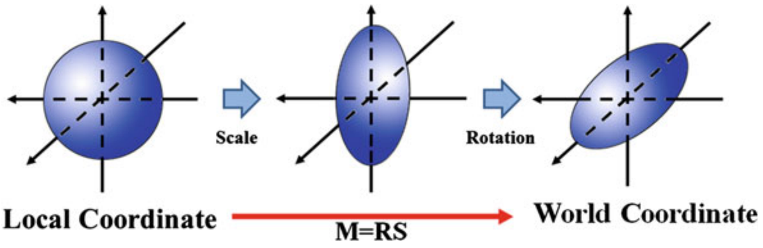


Fig. 4.32 Transformation between world coordinate and local coordinate

#### 4.3.4.4 Generation of the Dissolved Air Field

The center of an ice volume has foggy opaque region. Air bubbles further from the center tend to be larger, and become progressively smaller toward the center (Fig. 4.36a). Near the center, bubbles are sufficiently small that they appear like cloud. It is extremely time consuming to simulate every air bubble separately in this region. Therefore, we render a particle as an air bubble if it has sufficient air. Otherwise, we apply a volume-rendering technique using dissolved air values and generate cloudy volume image when the final state has been reached. We fill each cell of the dissolved air field by averaging the volume of dissolved air contained in ice particles within radius h as shown in the Fig. 4.33a. While such simplification would introduce a slight margin of error in computing the volume of dissolved air, it does not cause serious impact the quality of rendered image while simplifying computational complexity.

Figure 4.33b visualizes dissolved air field containing sufficient air in red and the air bubble particles in white. Our method can depict both large air bubbles and a group of small, cloudy air bubbles without requiring excessive numbers of particles.

Because we use both particles and grids to describe air bubbles, we can naturally display how air is distributed. Other techniques using only grids would result in

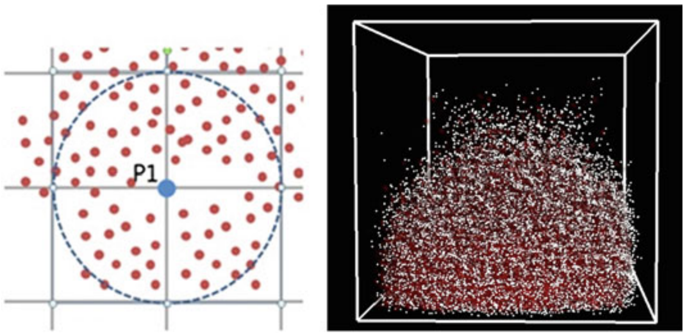


Fig. 4.33 Fetching the air around a grid cell

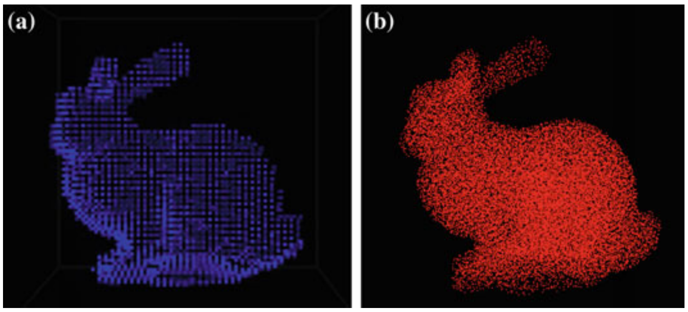


**Fig. 4.34** **a** The tracked model surface derived by Frisken's method. **b** The placement of the water particles

blocky formation of the bubbles. In addition, as demonstrated in Fig. 4.33b, our simulation can accurately depict air bubbles that are clustered or isolated.

#### 4.3.4.5 Freezing Simulations for Other Shapes

Past research mainly addressed simulation of cubic ice models. But, in reality, containers of arbitrary shape are widely used when freezing water. Our model used the SDF method suggested by Frisken [70] to simulate opaque ice of an arbitrary shape (Fig. 4.34).

Figure 4.35 shows the simulation results applied on various geometric shapes. Frozen particles are visualized in blue, the air bubbles in white, and dissolved air field in red, respectively. While conducting simulation using modified heat equation, it is assumed that the same amount of heat was transferred from outside. The figure captures the ice gradually freezing thicker toward the center.

### *4.3.5 Results and Discussion*

#### 4.3.5.1 Implementation and Results

The simulation was run on Intel Core2 2.66 GHz equipped with 6.0G RAM and NVidia GeForce 8800GTS graphic card. Ice surfaces were rendered by ray tracing method with photon mapping to show caustics. Typical rendering took about 5 min with 1024 × 768 resolution by Mental ray in Autodesk Maya 2010. During the rendering process, air bubbles were depicted as an ellipsoid, and dissolved air field was processed using volume rendering. Exact time depends on the number of particles processed. The cube model (Fig. 4.36b), with 295,000 particles, took about 10 min. Simpler models were processed in less than 2 min.

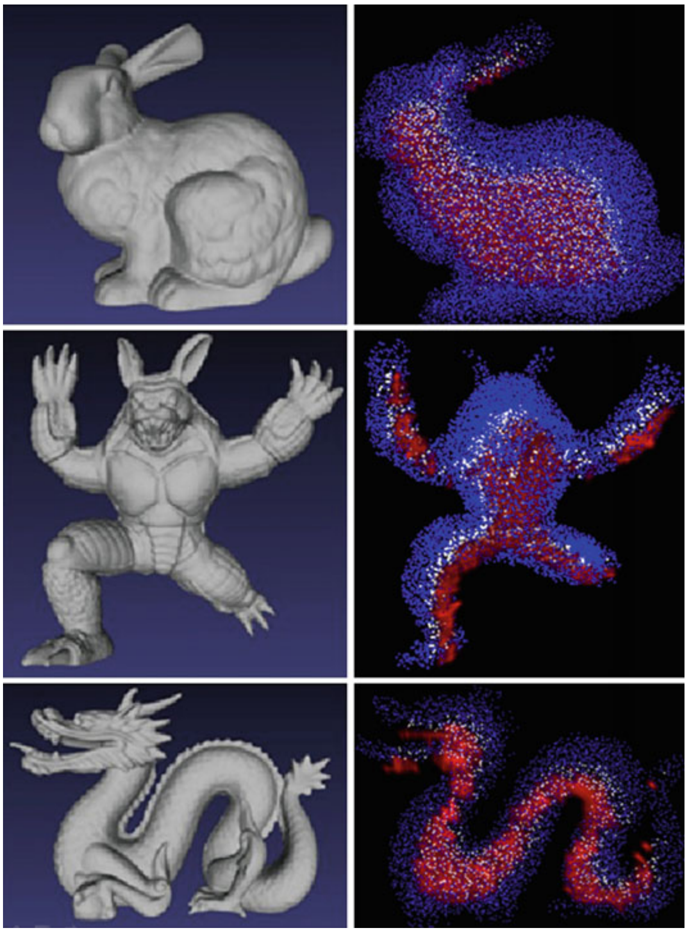


**Fig. 4.35** Simulation results for various geometries

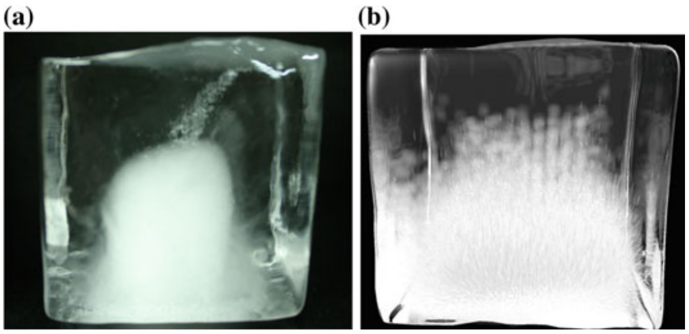


**Fig. 4.36** Comparison between **a** real ice and **b** our result

Particles were placed so as not to overlap each other in the beginning. Particles outside of the target geometry were then removed. To prevent large gaps between particles or excessive clusterization, it is made sure that particles maintain certain radius between each other. Same radius was also used for dissolved air distribution process. Initial temperature began at 10 °C.

Particles do not move during simulation since freezing water being frozen inside a static container will not exhibit drastic movement. Based on the similar assumption that water particles would exhibit little movement, grid was used instead of smoothed particle hydrodynamics (SPH) [56] to solve heat equation. SPH has an advantage of enabling more accurate image being generated when simulating body of fluid that move actively (e.g., river or fall). However, it is inefficient and does not deliver superior rendering quality when the subject is mostly stationary. It must be noted buoyancy simulation was not considered since bubbles are created only when trapped inside of the frozen volume.

Figure 4.36 compares our results against a photographic image of real ice. It is visually apparent that we achieved an impressive rendering quality which closely resembles that of a real object. Prior to taking the photo shown in Fig. 4.36a, water was first boiled and then placed in a freezer to reduce volume of dissolved air. In simulation, the rate of heat transfer on the bottom of the model was reduced to accurately reflect that the floor of a freezer delivers less heat than the air inside the freezer.

Figure 4.37 illustrates our approach in progressive phases. SDF was used to simulate freezing kept in a container of arbitrary shape and equal heat transfer rate was applied on all over the surface. 25,000 particles were used in the figure, and the number of air particles in the final image was 4,472. Visualization of only the air bubble particles cannot perfectly describe small cloudy air bubble groups. On the other hand, visualization of only the dissolved air field would result in a smoky image. Our result shows that realistic image of opaque ice was created using dissolved air field and air bubble particles.

All results were generated on $50 \times 50 \times 50$ grid. Figure 4.38 involved 85,650 particles, showing results viewed from the top and the side. Figure 4.39 shows the result of gargoyle statue model, and it convincingly demonstrates that our simulation works well on complex geometric shapes.

### 4.3.5.2 Limitation

The technique proposed in this section does not yet fully support accurate and step-by-step simulation of volume increase that occur during freezing process. Furthermore, physical phenomenon that occur during liquidification or solidification was outside the scope. Water is assumed to have no movement inside. In addition, supercooled water would exhibit a distinctively different behavior as it does not allow dissolved water to aggregate into larger bubbles. Such issues need to be addressed in future works.

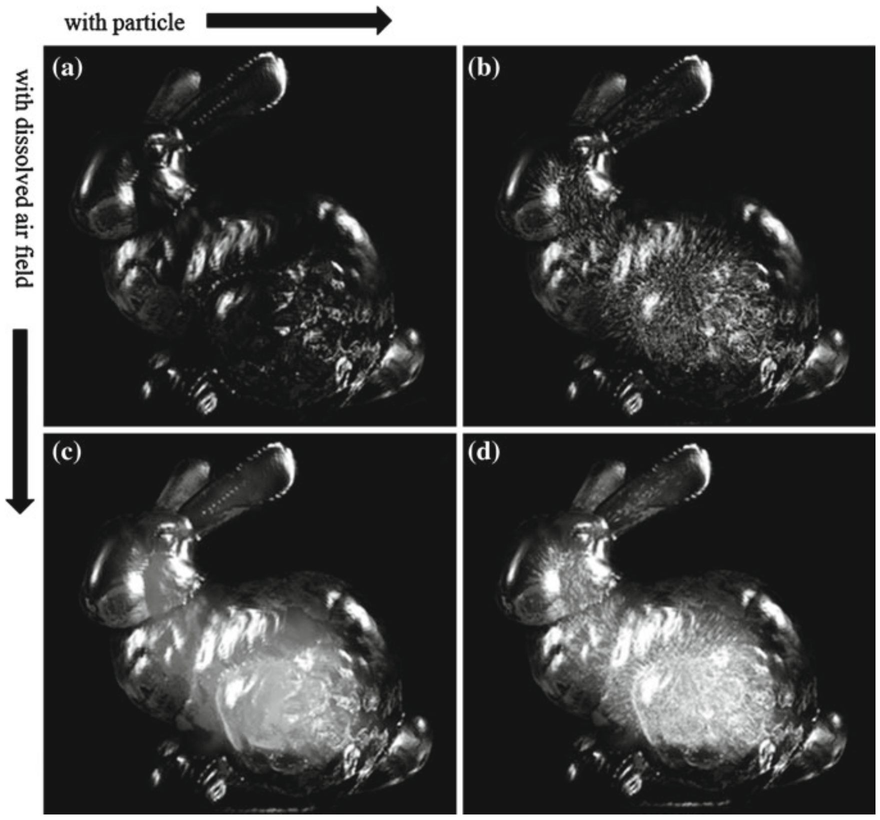


**Fig. 4.37** **a** Without any opaque area. **b** Using particle only. **c** Dissolved air field only. **d** Both particles and the field

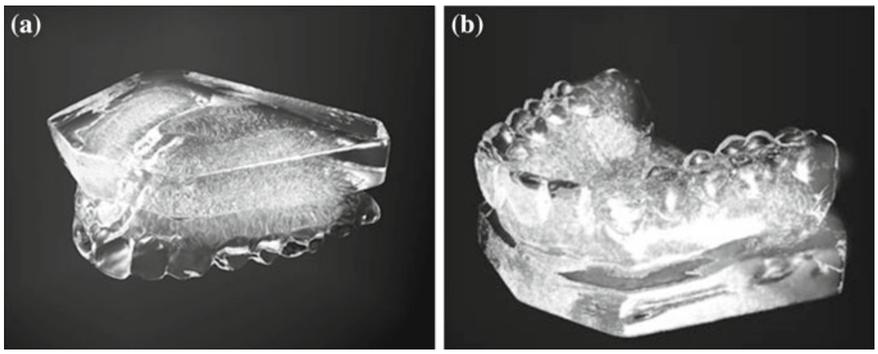


**Fig. 4.38** Teeth model: the numbers of water particles and produced air particles are 85,650 and 13,026

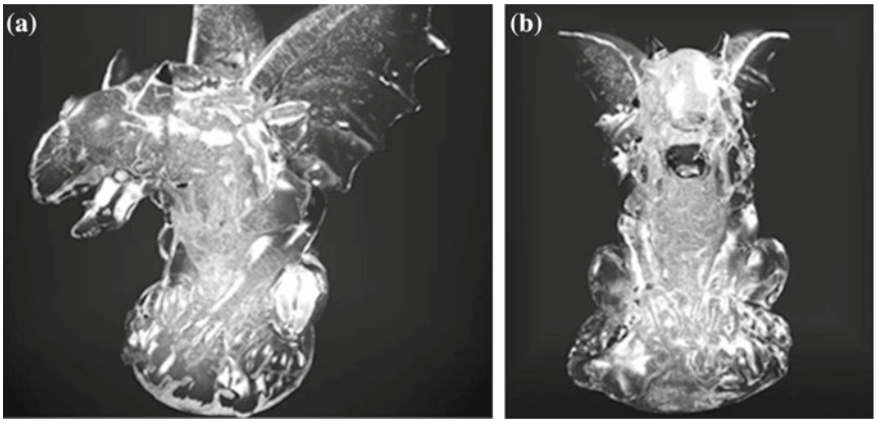


**Fig. 4.39** Gargoyle statue model: the numbers of water particles and produced air particles are 90,203, and 15,731

## *4.3.6 Conclusion and Future Work*

This section has presented an enhanced simulation method to generate realistic image of ice that contain air bubbles. Unlike previous methods, which worked well only on cubic models and produced only spherical bubbles, our method can simulate generation of opaque ice with arbitrary shape and directional bubbles. Moreover, it provides high-quality results even in a low-resolution grid. Our method supports a mechanism to model opaque region as a set of particles and dissolved air field, respectively. It handles both large air bubbles as well as small and cloudy air bubbles. By employing the dissolved air field, we developed relatively simple yet realistic enough opaque ice without a need to use excessive numbers of particles and memory. Enhanced heat equation allowed irregular pattern of freezing ice by accommodating different heat transfer rates for water and ice. While our heat transfer method is not fully accurate with respect to the law of physics, it produced visually convincing images.

The primary focus of this section is the generation of ice contained within a stationary container. Inclusion of equations on fluid dynamics would allow us to extend the model into freezing phenomenon that occur river or waterfall, which can be the next topic to work on.

## References

1. Losasso F, Irving G, Guendelman E, Fedkiw R (2006) Melting and burning solids into liquids and gases. IEEE Trans Vis Comput Graph 12(3):343–352
2. Melek Z, Keyser J (2007) Driving object deformations from internal physical processes. In: Proceedings of the 2007 ACM symposium on solid and physical modeling, pp 1–59

3. Liu S, Liu Q, An T, Sun J, Peng Q (2009) Physically based simulation of thin-shell objects' burning. Vis Comput Int J Comput Graph 25(5–7):687–696
4. Terzopoulos D, Platt J, Fleischer K (1989) Heating and melting deformable models (from goop to glop). Graph Interface 219:226
5. Carlson M, Mucha PJ, Van Horn RB III, Turk G (2002) Melting and flowing. In: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on computer animation, pp 167–174
6. Baraff D, Witkin A (1998) Large steps in cloth simulation. In: SIGGRAPH'98: proceedings of the 25th annual conference on computer graphics and interactive techniques, pp 43–54
7. Terzopoulos D, Platt J, Barr A, Fleischer K (1987) Elastically deformable models. In: Proceedings of the 14th annual conference on computer graphics and interactive techniques, pp 205–214
8. Bridson R, Marino S, Fedkiw R (2003) Simulation of clothing with folds and wrinkles. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation, pp 28–36
9. Choi KJ, Ko HS (2002) Stable but responsive cloth. ACM Trans Graph 21(3):604–611
10. Provot X (1995) Deformation constraints in a mass-spring model to describe rigid cloth behavior. Graph Interface 147:154
11. Müller M, Heidelberger B, Hennix M, Ratcliff J (2007) Position based dynamics. Vis Commun Image Represent 18(2):109–118
12. Hong M, Choi M, Jung S, Welch S (2005) Effective constrained dynamic simulation using implicit constraint enforcement. In: International conference on robotics and automation, pp 4520–4525
13. Goldenthal R, Harmon D, Fattal R, Bercovier M, Grinspun E (2007) Efficient simulation of inextensible cloth. In: Proceedings of ACM SIGGRAPH 2007, ACM transactions on graphics (TOG), vol. 26(3): Article No. 49
14. English E, Bridson R (2008) Animating developable surfaces using nonconforming elements. ACM Trans Graph 27(3):1–5
15. Grinspun E, Hirani AN, Desbrun M, Schröder P (2003) Discrete shells. In: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on computer animation, SCA'03, pp 62–67
16. Thomaszewski B, Wacker M (2006) Bending models for thin flexible objects. In: WSCG short communication proceedings 9
17. Volino P, Magnenat-Thalmann N (2006) Simple linear bending stiffness in particle systems. In: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on computer animation, pp 101–105
18. Bergou M, Wardetzky M, Harmon D, Zorin D, Grinspun E (2006) A quadratic bending model for inextensible surfaces. In: Proceedings of the fourth Eurographics symposium on geometry processing, SGP'06, pp 227–230
19. Garg A, Grinspun E, Wardetzky M, Zorin D (2007) Cubic shells. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation, SCA'07, pp 91–98
20. Choi MH, Hong M, Welch S (2004) Modeling and simulation of sharp creases. In: SIGGRAPH'04: ACM SIGGRAPH 2004 sketches, p 95
21. Molino N, Bridson R, Teran J, Fedkiw R (2003) A crystalline, red green strategy for meshing highly deformable objects with tetrahedral. In: IMR, pp 103–114
22. Desbrun M, Meyer M, Schröder P, Barr AH (1999) Implicit fairing of irregular meshes using diffusion and curvature flow. In: Proceedings of the 26th annual conference on computer graphics and interactive techniques, SIGGRAPH'99, New York, pp 317–324
23. Bell N, Garland M (2008) Efficient sparse matrix-vector multiplication on Cuda, NVIDIA Technical report NVR-2008-004
24. Hong J-M, Shinar T, Fedkiw R (2007) Wrinkled flames and cellular patterns. In: Proceedings of ACM. ACM transactions on graphics (TOG), vol 26(3), pp 47:1–47:6
25. Min K, Metaxas D (2007) A combustion-based technique for fire animation and visualization. Vis Comput: Int J Comput Graph 23(9):679–687
26. Nguyen DQ, Fedkiw R, Jensen HW (2002) Physically based modeling and animation of fire. In: Proceedings of ACM SIGGRAPH 2002. ACM transactions on graphics (TOG), vol 21(3), pp 721–728

27. Jeong S, Kim T-H, Kim C-H (2011) Shrinkage, wrinkling and ablation of burning cloth and paper. Vis Comput: Int J Comput Graph—CGI'2011 27(6–8):417–427
28. Zhu J, Chang Y, Wu E (2011) Realistic, fast, and controllable simulation of solid combustion. Comput Animat Virtual Worlds 22(23):125–132
29. Williams F (1977) Mechanisms of fire spread. Int Symp Combust 16(1):1281–294
30. Melek Z, Keyser J (2003) Interactive simulation of burning objects. In: Proceedings of the 11th Pacific conference on computer graphics and applications, PG'03, pp 462–466
31. Melek Z, Keyser J (2005) Multi-representation interaction for physically based modeling. In: Proceedings of the 2005 ACM symposium on solid and physical modeling, SPM'05, pp 187–196
32. STAM J (1999) Stable fluids. In: Proceedings of the 26th annual conference on computer graphics and interactive techniques, pp 121–128
33. Ishikawa T, Miyazaki R, Dobashi Y, Nishita T (2005) Visual simulation of spreading fire. In: Proceedings of the NICOGRAPH, international, pp 43–48
34. STAM J (2003) Flows on surfaces of arbitrary topology. In: Proceedings of ACM SIGGRAPH 2003. ACM transactions on graphics (TOG), vol 22(3), pp 724–731
35. Shi L, Yu Y (2004) Inviscid and incompressible fluid simulation on triangle meshes. In: Computer animation and virtual worlds—special issue: the very best papers from CASA 2004, vol 15(34), pp 173–181
36. Graham-Eagle J, Schult D (2001) The effect of wind on combustion waves with reactant depletion. Proc R Soc Lond Ser A: Math Phys Eng Sci 457:2397–2417
37. Beaudoin P, Paquet S, Poulin P (2001) Realistic and controllable fire simulation. Proc Graph, Interface159–166
38. Lee H, Kim L, Meyer M, Desbrun M (2001) Meshes on fire. In: Proceedings of the Eurographics workshop on computer animation and simulation, pp 75–84
39. Perry CH, Picard RW (1994) Synthesizing flames and their spreading. In: Proceedings of the fifth Eurographics workshop on animation and simulation, pp 1–14
40. Zhao Y, Wei X, Fan Z, Kaufman A, Qin H (2003) Voxels on fire. In: Proceedings of the 14th IEEE visualization 2003, p 36
41. Wei X, Li W, Mueller K, Kaufman A (2002) Simulating fire with texture splats. In: Proceedings of the conference on visualization'02, pp 227–235
42. Um K, Kim T-Y, Kwon Y, Han J (2013) Porous deformable shell simulation with surface water flow and saturation. Comput Animat Virtual Worlds 24(34):247–254
43. Wang H, Miller G, Turk G (2007) Solving general shallow wave equations on surfaces. In: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on computer animation, pp 229–238
44. Fan Z, Zhao Y, Kaufman A, He Y (2005) Adapted unstructured LBM for flow simulation on curved surfaces. In: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on computer animation, SCA'05, pp 245–254
45. Auer S, Macdonald CB, Treib M, Schneider J, Westermann R (2008) Real-time fluid effects on surfaces using the closest point method. Comput Graph Forum 31(6):1909–1923
46. Neill P, Metoyer R, Zhang E (2007) Fluid flow on interacting deformable surfaces. In: SIGGRAPH'07 ACM SIGGRAPH 2007 posters. Article no. 57
47. Angst R, Thuerey N, Botsch M, Gross M (2008) Robust and efficient wave simulations on deforming meshes. Comput Graph Forum 27(7):1895–1900
48. Belkin M, Niyogi P (2003) Laplacian eigenmaps for dimensionality reduction and data representation. Neural Comput 15(6):1373–1396
49. Pinkall U, Polthier K (1993) Computing discrete minimal surfaces and their conjugates. Exp Math 2(1):15–36
50. Schmidt R, Grimm C, Wyvill B (2006) Interactive decal compositing with discrete exponential maps. In: Proceedings of ACM SIGGRAPH 2006. ACM transactions on graphics (TOG), vol 25(3), pp 605–613
51. Petronetto F, Paiva A, Lage M, Tavares G, Lopes H, Lewiner T (2010) Meshless Helmholtz-Hodge decomposition. IEEE Trans Vis Comput Graph 16(2):338–349

52. Bridson R, Houriham J, Nordenstam M (2007) Curl-noise for procedural fluid flow. ACM Trans Graph 26(3):46:1–46:3
53. Pearson JE (1993) Complex patterns in a simple system. Science 261:189–192
54. Bridson R (2007) Fast poisson disk sampling in arbitrary dimensions. In: ACM SIGGRAPH 2007, sketche. Article no. 22
55. Fujisawa M, Miura KT (2007) Animation of ice melting phenomenon based on thermodynamics with thermal radiation. In: Proceedings of the 5th international conference on computer graphics and interactive techniques in Australia and Southeast Asia, pp 249–256
56. Iwasaki K, Uchida H, Dobashi Y, Nishita T (2010) Fast particle-based visual simulation of ice melting. Comput Graph Forum 29(7):2215–2223
57. Kim T, Henson M, Lin MC (2004) A hybrid algorithm for modeling ice formation. In: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on computer animation, SCA'04, Eurographics association, pp 305–314
58. Witten TA, Sander LM (1981) Diffusion-limited aggregation, a kinetic critical phenomenon. Phys Rev Lett 47:1400–1403
59. Ma S, Grinstein G, Mazenko G (1986) Directions in condensed matter physics: memorial volume in honor of Shang-Keng Ma., World Scientific Series on Directions in Condensed Matter PhysicsWorld Scientific, Singapore
60. Kim T, Adalsteinsson D, Lin MC (2006) Modeling ice dynamics as a thin-film Stefan problem. In: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on computer animation, SCA'06, Eurographics association, pp 167–176
61. Kharitonsky D, Gonczarowski J (1993) A physically based model for icicle growth. Vis Comput 10(2):88–100
62. Hong J-M, Lee H-Y, Yoon J-C, Kim C-H (2008) Bubbles alive. In: Proceedings of ACM SIGGRAPH 2008. ACM transactions on Graph (TOG), vol 27(3), pp 48:1–48:4
63. Madrazo C, Tsuchiya T, Sawano H, Koyanagi K (2009) Air bubbles in ice by simulating freezing phenomenon. J Soc Art Sci 8(1):35–42
64. Seipel S, Nivfors A (2007) Real-time rendering of ice. In: Proceedings of the ninth IASTED international conference on computer graphics and imaging, CGIM'07, pp 60–66
65. Owen JM, Villumsen JV, Shapiro PR, Martel H (1998) Adaptive smoothed particle hydrodynamics: methodology. Astrophys J Suppl Ser II 116(2):155–209
66. Jo E, Kim D, Song O-Y (2011) A new SPH fluid simulation method using ellipsoidal kernels. J Vis 14(4):371–379
67. Carte AE (1961) Air bubbles in ice. Proc Phys Soc 77(3):757–769
68. Monk P (2004) Physical chemistry: understanding our chemical world. Wiley, Chichester
69. Forbes G (2007) Texture and bubble size measurements for modelling concentrate grade in flotation froth systems
70. Frisken SF, Perry RN, Rockwood AP, Jones TR (2000) Adaptively sampled distance fields: a general representation of shape for computer graphics. In: Proceedings of the 27th annual conference on computer graphics and interactive techniques, pp 249–254

# Chapter 5
# Fluid Interaction

## 5.1 Solid-Fluid Collision Detection

### *5.1.1 Fast Coupling for Particle-Based Simulation with Motions*

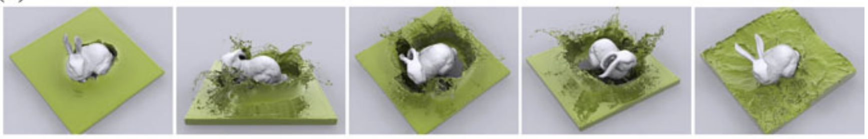
**Abstract** This section proposes a direct motion tree (DMT) method to speedup the particle-solid coupling process in fluid simulations through prediction of the particle motion path. The DMT method is adaptively constructed, and each node stores the signed distance to the object, as well as a list of particles that are likely to collide with that. The possibility of collision is determined based on the particles predicted motion path, derived from the current position and velocity. So we can robustly deal with the tunneling problem that the previous approaches suffer from when there are particles moving quickly in comparison to the width of leaf node. To extract collision-candidate particles (called DMT particles in this section) and follow the constraint of Courant-Friedrichs-Lewy (CFL) condition for them, we introduce a new particle transfer method. Particles are transferred to child nodes, based on the intersecting patterns between the nodes and the motion path of each particle. As a result, the DMT particles are robustly found and the leaking particles are barely shown in simulations. In addition, a new parallel hashing scheme is also introduced into the signed distance value calculation process, which significantly improves the performance of the entire particle simulation. Experimental results show the improved performance and robustness over the previous approaches.

#### 5.1.1.1 Introduction

The use of computer graphics in films is growing rapidly. However, a technique that still requires a lot of research is the modeling of rigid or deformable bodies interacting with various fluids. Collision detection is a key process in modeling interactions between multiple objects. While Lagrangian methods work well with

C.-H. Kim et al., *Real-Time Visual Effects for Game Programming*,
Gaming Media and Social Effects, DOI 10.1007/978-981-287-487-0_5

**(a)**



**(b)**

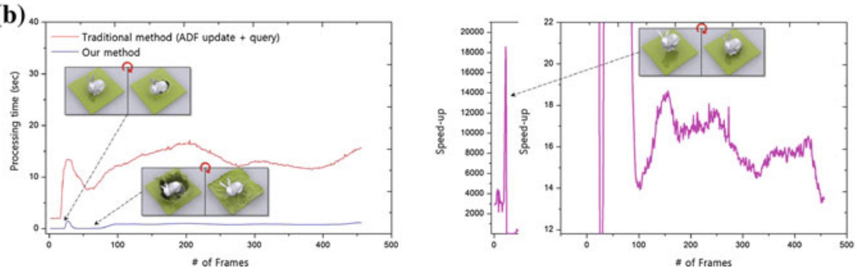


**Fig. 5.1** A falling bunny interacting with fluids of 3,687,560 particles. Our direct motion tree (DMT, *blue curve*) method accelerates collision detection for arbitrary scenes with any rigid/deformable objects. **b** On average, DMT shows about ×17 speedup over an adaptively sampled distance field (ADF, *red curve*) [7] (*Red circular arrow* represents the scene transition)

simple objects, an enormous number of particles are required to model collisions of complex objects such as cloth, hair, or fluids. But its computational requirement grows with the number of primitives, motivating the development of more efficient culling methods.

The bounding volume hierarchy (BVH) is an efficient culling method for rigid bodies. But if an object is deformable, its BVH structure must be updated at 40 every time-step, which has a serious impact on performance. This issue is addressed by recent techniques in which spherical approximations are combined with BVH to simulate particle-based fluids, as well as rigid-body interactions. Another method is to use a signed distance field (SDF) to simulate the interaction of both rigid objects and deformable bodies, such as cloth [7, 23].

This section proposes a direct motion tree (DMT) method for efficient culling in simulating the motion of particles (see Fig. 5.1). Unlike previous BVH-based culling methods for polygonal models, this technique is designed to improve the speed of a Lagrangian particle simulation. Generally, Lagrangian approaches require a greater number of particles than the number of polygons involved in mesh-based simulations, and it becomes a serious bottleneck. Because this DMT investigates only the particles with a high possibility of collision, its performance is much higher than previous ADF-based methods. The contributions of this method are as follows:

- **Fast and robust collision detection**: To handle the enormous number of particles efficiently, DMT has a structure of adaptively subdivided spaces such as the BVH or ADF. In previous BVH/ADF approaches, tunneling problems—where many colliding particles are not detected—sometimes occur when they only use the leaf nodes. This is because the magnitudes of their velocities exceed the grid size of a leaf node. Unlike the previous approaches, our method shows very robust collision detection, because it only finds surface particles by considering the motion of the particles as well as the location of the particles. Furthermore, the performance is way above those of previous approaches, since the number of particles near the surface is very small compared to that of entire particles.

- **Parallel hash-based ADF**: A new hash-based parallel data structure is introduced which is suitable for both rigid and deformable bodies, based on the adaptively sampled distance field (ADF) presented by Frisken [22].
- **Efficiency of representation**: Because this method is performed on just one simulation field which includes multiple objects, it is easier to represent fluid interactions.

#### 5.1.1.2 Related Work

Collision detection methods have been studied in fields such as computer graphics, robotics, and computational geometry. Traditional culling techniques can be classified as either discrete or continuous approaches, depending on the timing of the collision detection.

Discrete Collision Detection (DCD) finds the collisions that occur during time-steps of a fixed length, at which point BVH approaches are generally applied. There are many variations of DCD, some of which consider the topological properties of the objects. For example, convex hull tree and Voronoi Marching methods are designed to detect collisions between convex polygons [16, 17]. Due to the inherent discreteness of DCD, it is difficult to find consecutive collisions that occur in the middle of consecutive time-steps.

Continuous Collision Detection (CCD) was developed to find close sequential collisions. Various CCD techniques include:

– *Algebraic equation solution* provides a simple vector representation of the collision space [56].
– *Swept volume* constructs the volume swept by a moving-object over consecutive time-steps, to find the collision between that volume and other objects [4].
– *Adaptive bisection* detects changes in the configuration of primitives over a certain time interval by determining distances between neighboring primitives. It repeatedly subdivides two consecutive time-steps until it converges to the moment of collision [57, 66].

Most CCD techniques are not fast enough for real-time applications, prompting the development of other methods to improve the efficiency of collision detection.

Bounding Volume Hierarchy (BVH) techniques provide efficient culling that uses a bounding volume with a simpler topology than the actual object [27]. The bounding volumes may be spheres, axis-aligned bounding boxes, object-oriented bounding boxes, $k$-DOPs (discrete oriented polytopes), etc. However, the BVH approach requires a lengthy preprocessing step for the hierarchy construction, and when the models are deformable, the hierarchy refit is repeatedly incurred at every time-step.

Spherical approximation, which generates surface spheres for approximating triangular meshes, has been widely used in collision detection [15]. It is also used to model the objects interacting with water sheets [2, 3], while others used it to model the interactions between rigid-bodies and particle-based fluids [9, 59]. One criticism of spherical approximation has been that they are very slow due to the enormous number of spheres (see Fig. 5.2b). To solve this problem, the inner sphere tree method is proposed, in which a shape is approximated by a tree of adaptively sized spheres [55, 73]. However, reconstructing this tree would take too long to allow this method to be used for simulating deformable bodies. To enhance the performance of collision detection, spheres are created only near the surface of an object (see Fig. 5.2c), needless to approximate the interior of the object. As a result, the number of spheres will be greatly reduced, leading to a significant improvement in the performance of collision detection. But this approach may fail if particles move too quickly, since some 'leaking' particles penetrate the surface whose interior has not been approximated by spheres.

Signed Distance Field (SDF) contains the distances from every cell in a grid covering the scene to the nearest object surface (see Fig. 5.3). Because SDF makes it easy to perform Boolean operations for 3D shapes, it is widely used in applications

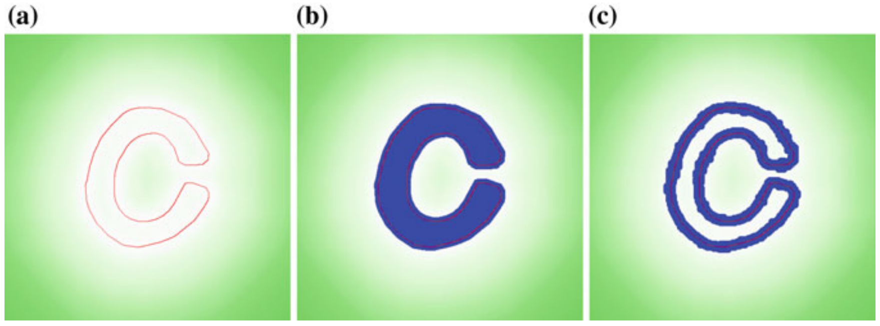


**Fig. 5.2** Spherical approximation: **a** mesh contour, **b** all spheres, **c** surface spheres

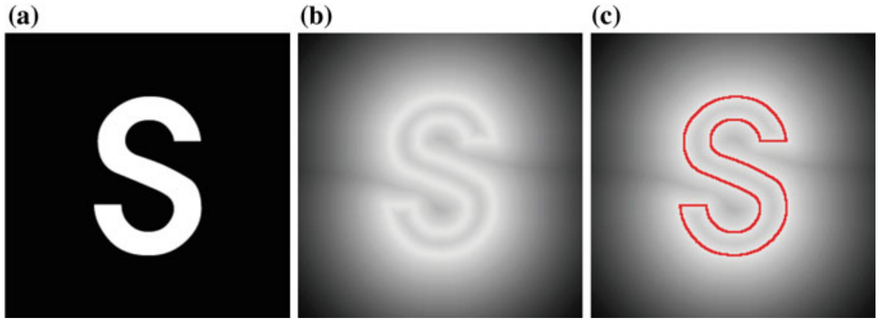


**Fig. 5.3** A 2D illustration of SDF: **a** original image, **b** distance field, **c** outline + **b**

such as computer-aided designs, computer graphics, and rapid prototyping. Applying SDF approaches to physics simulations with millions of particles incurs a lot of computation overhead, especially when investigating whether each particle collides with the surface.

There has been much effort put into accelerating SDF computations for complex polygonal models, rather than just particles. Mauch and Mauch constructed a triangular net which contained prisms, cones, and wedges, which represented the nearest faces, vertices, edges, and objects [48]. Points in the same zone of the net share the same nearest surface primitive. The efficiency of this approach can be improved by tuning various parameters, but determining the sign of the distance values remains a problem. Sigg et al. proposed a prism scan algorithm which modifies Mauchs method by constructing Voronoi sites for various types of vertices such as convexes, concaves, saddles, or planes [64]. In their method, a net structure is constructed only at the faces of the object, after which a subsequent scan conversion is performed on the graphics hardware. While this reduces the computation time, the sign of the distance value for a certain point is sometimes miscalculated when its nearest primitive is a vertex.

Additionally, errors can occur in determining normal vectors from SDF, because it contains no information about the surface topology. To collect the signs for distance values, Baerentzen and Aanaes introduced the concept of an angle-weighted pseudo-normal, which is associated with the vertices and edges of objects [5]. It is obtained by calculating the weighted average of the normal vectors of all triangles connected to each target vertex. Guéziec improved the computation of the distance fields using a hierarchical triangular mesh of objects [28]. He also introduced related traversal algorithms to reduce the time for constructing the SDF. Sud et al. determined the Voronoi regions of the primitives from which objects are constructed [62]. Subsequently, they created a distance field for each slice of a grid, and used the Voronoi regions to speedup the search for the primitive nearest to each grid point. Lefebvre and Hoppe also exploited spatial coherence by using an adaptive primal tree [44]. Huang et al. introduced methods that measured the complete distance fields, which represent distances accurately [33]. Houston et al., on the other hand, also proposed an RLE (Run Length Encoding) method [35] which computed sparse level sets by encoding the regions using RLE with respect to their distance only in the narrow band. Nielsen and Museth [53] proposed a new structure named the dynamic tubular grid (DT-grid). Finally, Frisken et al. used adaptively sampled distance fields (ADF), which have a sampling resolution that varies with the amount of local surface details [22]. The sampled distances are then stored in an octree structure (see Fig. 5.4). We utilized this method to create an SDF.

#### 5.1.1.3 Parallel SDF Framework

It is time-consuming to construct an SDF for models with large numbers of primitives. Likewise, computing SDF at every time-step on nonlinearly deformed objects requires a large computational overhead. To reduce this cost, this section introduces

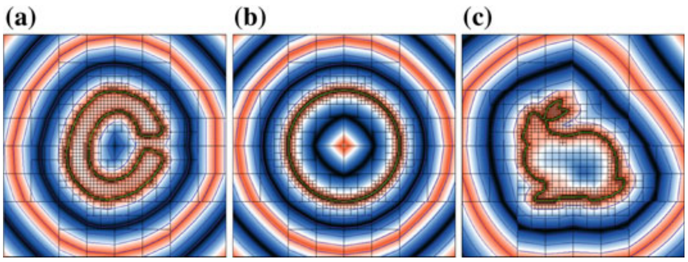


**Fig. 5.4** ADF construction examples (*green line*: zero-contours): **a** C glyph, **b** sphere, **c** Stanford bunny

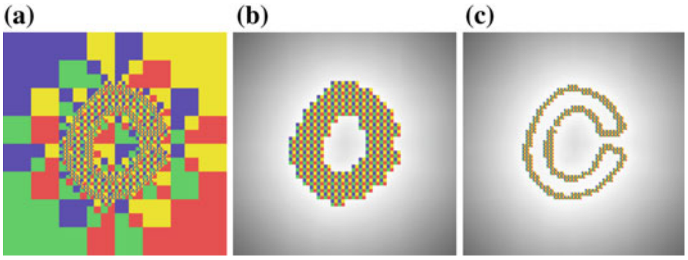


**Fig. 5.5** Multithreaded ADF construction. The same thread at each level has the same color: **a** entire structure, **b** ADF at the level 6, **c** ADF at the level 7

a novel method, which uses parallel computation to construct and maintain an ADF in conjunction with a hash table for querying its distance values [7].

## ADF Construction with Parallel Hashing

To apply parallel computation to the ADF calculation, we handle the nodes at the same level by multithreading (see Fig. 5.5). Each ADF cell has a corresponding thread index, which are reused at higher levels of the ADF (see Fig. 5.5a).

When performing the ray-casting for the subdivision, the same operation is repeated for some nodes, slowing the construction of the ADF. To reduce this redundancy, we used a hash table to access nodes quickly.

**Algorithm 5.1:** HASH KEY($c$)

```
Input: Corner $c$ of each node
$length \leftarrow 1.0/2^{level}$
$size \leftarrow length + 1$
// calculate coordinate index $x, y, z$
$i, j, k \leftarrow c$.getCoordinateIndex()
$key \leftarrow i \times size \times size + j \times size + k$
return($key$)
```

**Table 5.1** Comparison of ADF construction time with and without our hashing scheme

| Method | Depth | Timing (sec) |
|---|---|---|
| Without hashing | 4 | 3.27 |
| | 6 | 15.25 |
| | 8 | 86.50 |
| With hashing | 4 | 1.61 |
| | 6 | 5.69 |
| | 8 | 36.05 |

Each cell in an ADF is always subdivided into eight subcells of equal size. Thus, a unique key for each ADF cell can be calculated by encoding its coordinate indices (see Algorithm 6.1). The cell containing a query location can then be retrieved reliably and efficiently from the resulting hash table. Level is the depth of the ADF tree and $i$, $j$, and $k$ are the coordinate indices of the current node. As shown in Fig. 5.5, all nodes at the same level can be processed in parallel, and the creation of the hash table can also be performed in parallel.

Teschner et al. also proposed an optimized spatial hashing technique [71], but they assigned a hash key to each primitive, which creates a long delay if the number of primitives increases. On the other hand, our method evenly subdivides the space into an octree and creates a hash key for each node. As a result, the number of primitives has a smaller influence on performance. Table 5.1 shows that parallel ADF construction using a hash table is about 11 times faster than the serial ADF.

#### 5.1.1.4 Construction of Direct Motion Tree

##### Leaking Particles by Tunneling Problem

Leaking particles are a well-known problem with particle-based techniques, including spherical approximation and SDF. The method of spherical approximation can take a long time, since too many unnecessary spheres are constructed inside the object (see Fig. 5.2b). Performance is increased if spheres are created only near the surface of objects (see Fig. 5.2c). However, when simulating scenes that change rapidly, some particles end up penetrating the object surface without their collision with the surface spheres being accurately detected.

If a collision detection is performed for all particles in the SDF approach, none will leak. However, it is slow to query all the particles, because only those near the object surfaces usually collide with an object. If we handle only particles near the surfaces, the leaking particle problem can also occur under the aforementioned rapidly changing scenes during spherical approximation. Figure 5.6 shows the leaking particles when only the surface particles are considered.

Figure 5.7a shows that spherical approximation has the same problem. Because some particles penetrate inside the base stone, only a little volume of water remains.

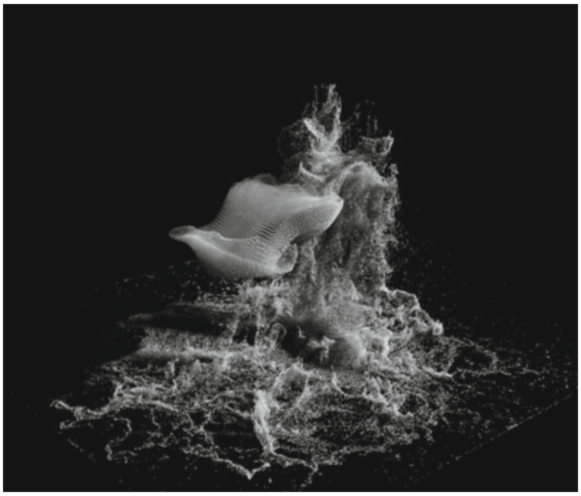


**Fig. 5.6** Leaking particles in an SDF-based simulation

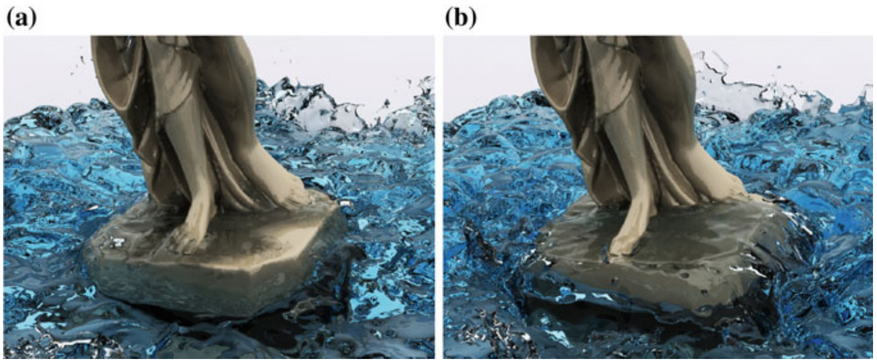


**Fig. 5.7** Comparison of leaking particles: **a** spherical approximation, **b** DMT

We address this problem by integrating the predicted motion path of particles into the collision detection framework, not just the particle positions.

## Coupling

**One-way coupling**: To simulate the interaction between the particles and the object, we use the modified version of [7]. Consider a collision between a particle $p$ traveling at velocity $\mathbf{v}_p$ and a target object with its own level-set data. At the location of $p$, we use the local level-set value $\phi$, the outward normal $n = \nabla\phi/|\nabla\phi|$, and the local velocity $\mathbf{v}$ of the target object.

Equation (5.1) predicts how the current level-set value of the particle $p$ will change at the next time-step. This equation includes a term that accounts for the relative motion between the particle and the object in the normal direction. If $\phi_{t+1}$ is positive, then the particle will still be outside the object and nothing else will be required. Otherwise, a collision will occur, at which point the time must be calculated. By setting $\phi_{t+1} = 0$ in Eq. (5.1), we obtain Eq. (5.2), which gives us the time of collision $\Delta\tau$.

$$\phi_{t+1} = \phi_t + \Delta\tau(\mathbf{v}_p - \mathbf{v}_s) \cdot n \tag{5.1}$$

$$\begin{gathered} \phi_t + \Delta\tau(\mathbf{v}_p - \mathbf{v}_s) \cdot n = 0 \\ \rightarrow \Delta\tau = \frac{-\phi_t}{(\mathbf{v}_p - \mathbf{v}_s) \cdot n} \end{gathered} \tag{5.2}$$

Figure 5.8 provides an example of interactions between cloth and a rigid-body, which shows a lot of detail. To make the collision response, the method of Bridson et al. is used that is based on the level-set data [7].

**Two-way coupling**: To perform two-way coupling with a deformable body, each DMT node also stores the shortest distance to an object surface and the identifier of the nearest primitive, as well as the indices of all particles passing through that node. Information about the nearest primitive makes the two-way coupling faster when processing the interactions of particles which are near an object surface. Our method for collision responses is based on that of Bridson et al.

Equations (5.3) and (5.4) determine the adjusted positions of vertices on the colliding faces [7]. For the particle-triangle case, where the member vertices of face **xyz** with barycentric coordinates $w_1 w_2 w_3$ are interacting with a particle $p$, we use linear interpolation to determine the velocity of a particle in the interior of a triangular face.

$$\tilde{I} = \frac{2c}{1 + w_1^2 + w_2^2 + w_3^2} \tag{5.3}$$

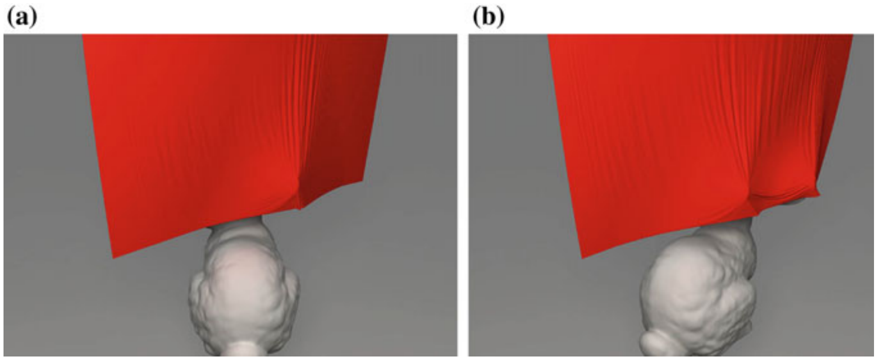


**Fig. 5.8** Interaction between the cloth and a rigid body

$$\mathbf{v}_i^{new} = \frac{\sum_{p=0}^{\Omega}\left(\mathbf{v}_i + w_i^p \frac{\tilde{I}}{m_i}\hat{n}\right)}{\Omega} \tag{5.4}$$

In Eq. (5.4), $i$ is the index of a vertex of the target face, and $\tilde{I}$ is the linearly interpolated impulse of magnitude in the direction $\hat{n}$, normal to the face. $c$ is a parameter which controls the flow of the coupling simulation. $\Omega$ is the number of particles inside the triangle. The new velocity $\mathbf{v}_i^{new}$ is obtained by interpolating the velocities of these particles, using Eq. (5.4).

### 5.1.1.5 Implementation and Applications

Our direct motion tree can be applied to a number of graphics tasks. We compare this method to spherical approximation and ADF in four different scenarios : coupling with rigid and deformable bodies, splash particles in a moving object, and cloth falling on a rigid body. Simulations used an Intel i7-2600k 3.40 GHz CPU with 16 GB of memory, and an NVIDIA GeForce GTX 580 graphic card.

#### Coupling of Fluid with Rigid and Deformable Bodies

Applying DMT to rigid and deformable bodies must be done in different ways. The distance field associated with a deformable body is updated at every time-step, so the particle transfer is performed during the modification of DMT. On the other hand, the distance field is constant for a rigid body, so it might be an overhead if the particle transfer is done by re-traversing the entire tree at every time-step.

To reduce this overhead, the phase region method is introduced. The phase region is simply divided into interior and exterior parts of an object (see Fig. 5.9b). All particles belong to the exterior region at the current time-step. By approximating the positions of particles at the next time-step through the utilizing their current positions and velocities, we are able to quickly find them near the object surface.

Only the particles that may belong to the interior region at the next time-step are tested through accurate collision detection. Figure 5.10 shows the result of coupling the particle-based fluid simulation with a rigid body. Our DMT accelerated the computation efficiency approximately 31 times faster than the method of [7], and 2,075 times faster than spherical approximation. Using spherical approximation, it is easy to find the sphere nearest to each particle, but the computational overhead in the entire process is large because of the need to process interactions between particles and spheres.

We use the finite element method (FEM) to model the deformation of an object. Figure 5.1 shows the result of coupling a particle-based fluid to a deformable body. The results for the spherical approximation method are not presented because they were too slow. On average, DMT ran about 17 times faster than ADF. As seen from the spikes in the plots in Fig. 5.1a, the processing time becomes slightly longer due to the increase in the number of DMT particles when the bunny collides with the

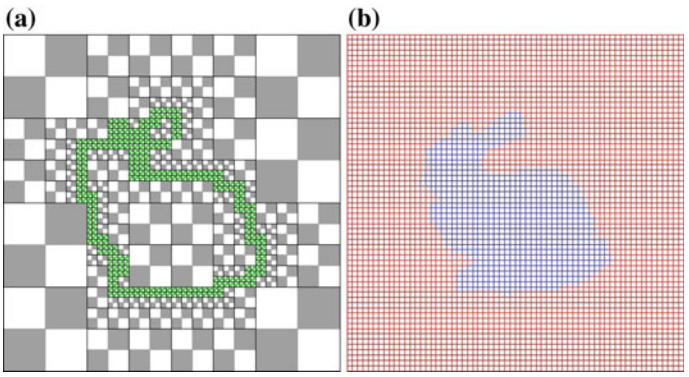


**Fig. 5.9** DMT structure on rigid-body: **a** DMT nodes (*green*), **b** phase region (*blue* interior, *red* exterior)

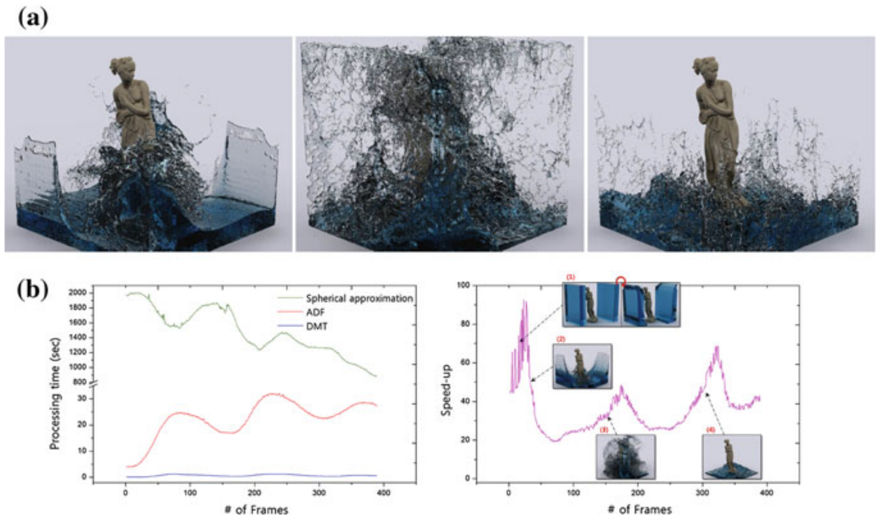


**Fig. 5.10** Coupling of fluid with a rigid body: **a** performance, **b** DMT speedup over ADF

liquid surfaces. Because the number of extracted DMT particles directly affects the entire performance, our performance exceeds that of the previous ADF method that considers all the particles by traversing the adaptive tree. The improvement of the performance is shown very well in Fig. 5.1b.

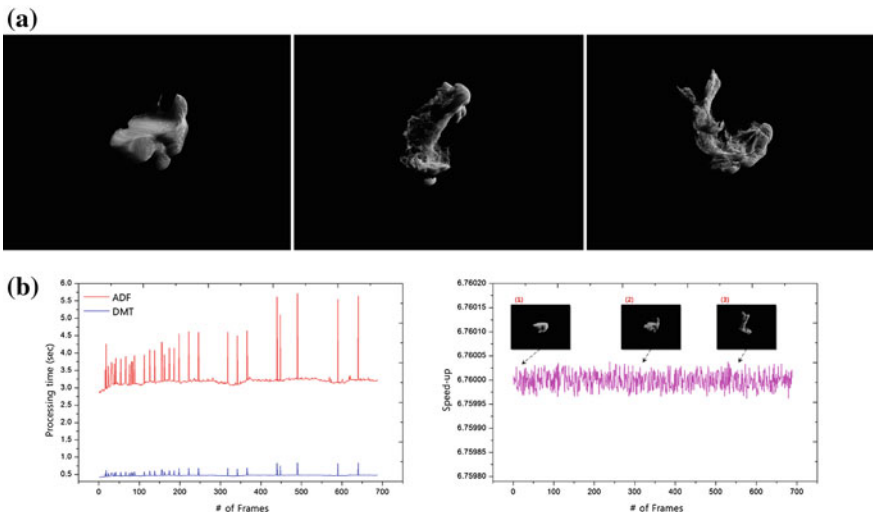


**Fig. 5.11** Splash particles in rotating boundary: **a** performance, **b** DMT speedup over ADF

## Splash Particle in Rotating Boundary

Simulating a fluid with a rotating boundary is known to be computationally very expensive [32]. Figure 5.11 shows an example of this scenario in which a bunny-shaped transparent boundary continuously rotates around its *z*-axis. The motion of the particles is not stabilized, as it is when simulating deformable bodies, which results in excessive particle–surface interactions. DMT showed to be approximately 5 times faster than ADF, as well as no signs of leaking particles. The number of particles considered in collision detection was also greatly reduced. The particles are leaked around the object boundary. On the contrary, DMT robustly finds the particles with a high possibility of collision in interior nodes by considering the particle motion.

Figure 5.11 shows dynamically interacting particles inside a rotating object. The number of extracted DMT particles is kept in a certain range, because the surface–particle collision starts from the beginning of the simulation and the colliding appearance has no big change during the entire simulation (see subcaptions (1), (2), and (3) in Fig. 5.11b). Even in a dynamic scene like this, our method shows faster performance than the ADF-based approach.

## Coupling Cloth Falling on a Rigid Body

Figure 5.12 shows the particles involved in the interaction between the cloth and a rigid body. Red particles represent the DMT particles which can be collided with the object. Unlike the fluid simulation, many cloth particles are classified as DMT particles due to the draping effect (see subcaptions (2) and (3) in Fig. 5.13b). This

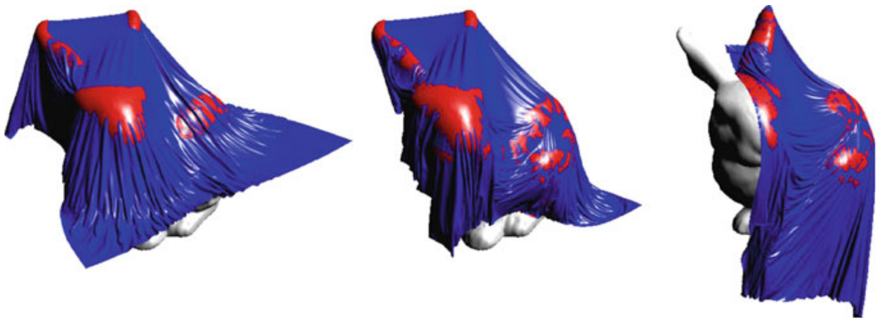


Fig. 5.12 Visualizing DMT particles (*red*)

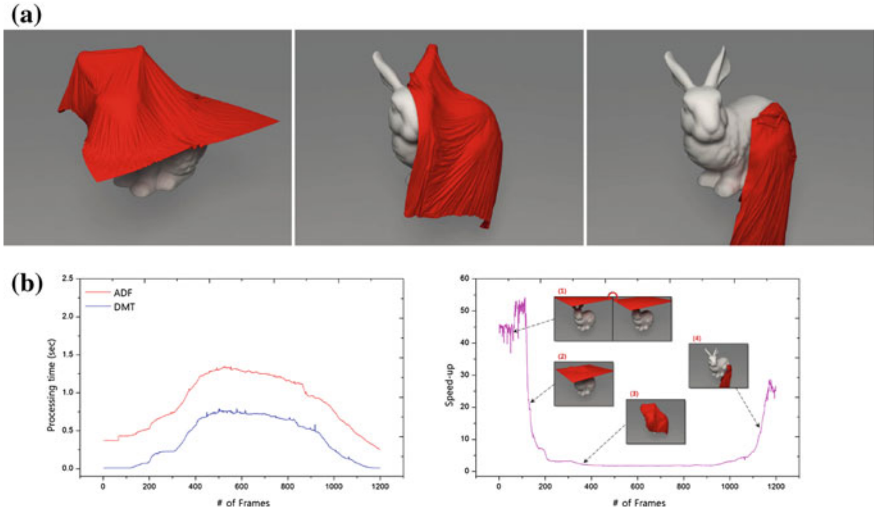


**Fig. 5.13** Coupling cloth falling on a rigid body: **a** performance, **b** DMT speedup over ADF

slows down the performance of DMT processing, but DMT still shows better efficiency than the previous methods.

Contrary to general particle-based fluid simulation, many particles stay near the object surface in cloth simulation due to the draping effect, which reduces the improvement by DMT. As shown in the results in Fig. 5.13b, our performance over the ADF is not so big, but still about 2 times faster on average.The Constrained Lagrangian Mechanics (CLM) by [26] was used for the cloth simulation.

#### 5.1.1.6 Conclusion

The DMT speedsup collision detection by investigating only the particles in its leaf nodes. The problem of leaking particles is significantly reduced by considering the predicted motions of particles at the particle transfer stage. Additionally, the construction of the DMT is accelerated by employing a new parallel hashing technique. Experimental results demonstrated improved performance and robustness. The parallel DMT construction algorithm can be converted into a framework for a GPU.

## 5.2 Controlling Fluid Animation

### *5.2.1 Interchangeable SPH and Level-Set Method in Multiple Fluids*

**Abstract** In grid-based fluid simulation, subgrid-scale fluid is difficult to represent realistically. In this section, we describe such small-scale details effectively, even on a coarse grid by using escaped particles. These particles, when simulated through smooth particle hydrodynamics, allow the illustration of dynamic and realistic fluids. Particles in SPH have a force for them to be merged within the range of each particle's radius. However, the SPH approach does not address the objective of only the part indescribable on a grid being portrayed through the particle method, decreasing the accuracy of the simulation. Integrated particles which are able to be described by level-set method are also likely to end up simulated by particles. To address this problem, this section introduces a method through which the indefinable part of a grid is denoted in particles, while level-set method is used to describe the particles merged on the grid.

#### 5.2.1.1 Introduction

Fluids research in the computer graphics community has largely been focused on the precision and realistic visual quality of simulations. Advection of simulation is also indispensable for greater accuracy, while BFECC [41] and CIP [67, 68] ensure second order accuracy. This section adopts the BFECC method to improve simulation accuracy and implements simulation using the particle level-set method [18, 20] to detect smooth surfaces. Even with both methods applied, Eulerian grid-based simulation, which is based on Navier–Stokes equations, continues to result in dissipation. Currently, a great deal of research on this problem is underway. One outstanding example, and a primary motive for this section, is [34]. The combination of particle and grid-based methods boosts realistic visual attributes and, most notably, plays an important role in depicting splashing, foam, and bubbles. Generally, the particles remaining outside the level-set surface after the correction of the particle level-set method are utilized. Meanwhile, this research is being conducted to improve authenticity through the addition of SPH-based physical properties. In this section, we also perform SPH-adopting simulation for the remaining escaped particles. Then, particles and grid are attempted two-way coupling using physical dynamics. Also, vorticity confinement method and cohesive force are added to SPH particle to harmonize grid-based simulation and particle simulation. Cohesive force refers to a force through which particles draw one another together. This force attracts neighboring particles and eventually engenders particle clustering. Prior research has adopted a hybrid approach for particle and grid-based simulation in order to exhibit the indefinable part on a grid. Therefore, we will focus on method to improve the accuracy of simulation. To this end, we utilize SPH particles to display the indefinable on the grid scale while small-scale features and particles which are clustered and become large enough to be depicted via the grid method are converted to level set on a grid.

#### 5.2.1.2 Previous Work

Foster and Metaxas [21] made three-dimensional fluid simulation based on Navier–Stokes equations and Stam [68] suggested a semi-Lagrangian integration scheme and introduced unconditionally stable fluid. Authors of [18, 20] adopted particle and level set and pioneered tracking complicated water surfaces. In [29], the volume of fluid method shows multiphase fluid simulation of surface tension between water and air bubble. Subsequently, Hong and Kim [31] suggest a numerical method that emphasizes a discontinuous interface among different fluids. Desbrun and Gascuel [14] focus on the SPH method to handle viscous fluids and Müller et al. [49] propose an interactive method, an underpinning of SPH in the simulation of water, and represent a multiphase SPH method to describe fluids with different compositions [51]. Cleary et al. [13] treat the collision motion of foam and bubbles on a complicated surface realistically. Recently, great effort is being made to highlight the details of an underlying subgrid, facilitated by the hybridization of a Eulerian grid and Lagrangian particle. Greenwood and House [25] portray small-scale features of a subgrid on a grid-based simulation by adding the particle level-set method [18], which uses the escaped marker particle from water or air. Kim et al. [37] also depicted splash by combining SPH method with escaped particles from particle level set. Losasso et al. two-way coupled particle level set and SPH [47] in order to simulate diffuse regions such as splashing. Finally, Hong et al. [34], the fundamental foundation of this section, utilized SPH air bubbles on a coarse grid to create lively air bubbles with small-scale features.

### 5.2.1.3 Fluid Simulation

#### Grid-Based Fluid Simulation

The Navier–Stokes equation provides the simulation of incompressible fluid, and preserves mass and momentum.

$$\mathbf{u}_t = -(\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p/\rho + \mathbf{f} \tag{5.5}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{5.6}$$

where $\mathbf{u}$ is the velocity, $\rho$ is the density, and $\mathbf{f}$ is external force (such as gravity, buoyancy etc.). Equations (5.5) and (5.6) are solved using Chorin's method [12]. We divide Eq. (5.5) into two equations by intermediate velocity $\mathbf{u}^*$.

$$\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} = -(\mathbf{u}^n \cdot \nabla)\mathbf{u}^n + \mathbf{f} \tag{5.7}$$

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^*}{\Delta t} = -\frac{\nabla p}{\rho} \tag{5.8}$$

To obtain $\mathbf{u}^*$, first, we should define $-(\mathbf{u}^n \cdot \nabla)\mathbf{u}^n$, the advection term, from the semi-Lagrangian method [68], and the rest of the equation, excluding pressure, is added. Next, we make Eq. (5.8) Poisson's equation in Eq. (5.9) and yield the pressure profile, based on the divergence-free condition from Eq. (5.6).

$$\nabla^2 p = \frac{\rho}{\Delta t}\nabla \cdot \mathbf{u}^* \tag{5.9}$$

Substitute pressure value in Eq. (5.10) to generate the velocity of the next step.

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \frac{\Delta t}{\rho}\nabla p \tag{5.10}$$

We use the octree [43] focused on the interface. BFECC (Back and forth Error Compensation and Correction) method reduces easily volume loss; so we implement the BFECC method to satisfy second-order accuracy. Furthermore, smooth surface from complicated water surface are found through the particle level-set method [18, 20].

#### SPH Fluid Simulation

SPH is a technique to interpolate the particle system. The equation to obtain $A(\mathbf{x}_i)$ at a particle $\mathbf{x}_i$ is as follows:

$$A(\mathbf{x}_i) = \sum_j m_j \frac{A_j}{\rho_j} W(\mathbf{x}_{ij}, h) \tag{5.11}$$

where $\rho$ is the density of particle $i$; $W(\mathbf{x}_{ij}, h)$ is a smoothing kernel with core radius $h$. $m_j$ is the mass of particle $j$. The acceleration of particle $i$ is calculated by dividing particle force $\mathbf{f}_i$ with density. Acceleration and velocity are used to yield velocity and position.

$$\mathbf{a}_i = \frac{d\mathbf{v}_i}{dt} = \frac{\mathbf{f}_i}{\rho} \tag{5.12}$$

Adams et al. suggest the adaptive sampling algorithms [1]. In order to describe a variety of bubbles we use the radius adaptive. Pressure force is provided as in the following equation.

$$\mathbf{f}_{ij}^{pressure} = -V_i V_j (P_i + P_j)(\nabla W(\mathbf{x}_{ij}, r_i) + \nabla W(\mathbf{x}_{ij}, r_j))/2 \tag{5.13}$$

where the volume $V_i$ is $m_i/\rho_i$, $r$ is the radius, the mass $m_i$ is proportional to $r_i^3$, $\mathbf{x}_{ij} = \mathbf{x}_j - \mathbf{x}_i$, and the pressure $P_i = k\rho_i$ with a control parameter $k$. SPH requires the calculation of the viscosity term but since we have implemented a grid-based simulation we leave out viscous force. External force is introduced in the next section.

#### 5.2.1.4 Method to Make SPH and Level Set Interchangeable

##### SPH Bubble Particles in the Subgrid Dynamic

Our hybrid simulation uses the particle level-set method. When the value of the $\phi$ value of marker particles is 0 or less than 0, water is corrected, while more than 0 requires air. However, when marker particles in the particle level-set method are left on the opposite side after correction for level set, they are replaced with escaped particles. Then, in SPH-adjusted simulation method, escaped particles are used to illustrate small-scale features of an underlying subgrid dynamically. Buoyancy is determined to make it proportional to each particle volume. Authors of [13, 50] used drag force and lift force in the next equation.

$$\mathbf{f}_i^{drag} = -k_{drag} r_i^2 |\mathbf{v}_i - \mathbf{u}_i|(\mathbf{v}_i - \mathbf{u}_i) \tag{5.14}$$

$$\mathbf{f}_i^{lift} = -k_{lift} V_i (\mathbf{v}_i - \mathbf{u}_i) \times \omega_i \tag{5.15}$$

where $\mathbf{u}_i$ and $\omega_i = \nabla \times \mathbf{u}_i$ are the liquid's velocity and vorticity, which are interpolated from the value of the particle's position in the grid; $\mathbf{v}_i$ is the particle's velocity. The coefficients $k_{drag}$ and $k_{lift}$ are given in 6000 and 200. Then, the lift force is used to make the path of air bubbles unstable. Additionally, Hong et al. [34] proposes SPH vorticity confinement.

$$\mathbf{f}_{ij}^{vorticity} = \varepsilon \left( \mathbf{N} \times \frac{\omega}{|\omega|} \right) \rho_i. \tag{5.16}$$

$\omega = \nabla \times \mathbf{v}$ denotes vorticity at the mass center of two SPH particles. Suppose that mass center is $\mathbf{p}^*$ and vorticity location vector means $\mathbf{n} = \mathbf{p}^* - \mathbf{p}_i$. Normalizing n becomes $\mathbf{N} = \mathbf{n}/|\mathbf{n}|$. Meanwhile, the greater density ratio of water than that of air induces air bubbles to be merged immediately.

$$\mathbf{f}_{ij}^{attraction} = k_{attraction} W_{attraction}(\mathbf{x}_{ij}, r_i + r_j)\rho_i. \tag{5.17}$$

Becker and Teschner [8] introduces cohesive attraction force to produce the cluster of air bubbles and shows the physically plausible phenomenon of SPH particles. Attraction force influencing air bubbles to be integrated enhances the $\rho$ of SPH particles which in turn induces rising high and eventually generates natural bubble turbulence motion in an unstable path when Eq. (5.11) pushes adjacent particles and maintains the surface tension of SPH particles.

### Large-Scale SPH Particle Back to Grid-Based Level Set

We displayed small-scaled bubble motion based on Lagrangian SPH particles in a grid-based simulation to detect detailed features of the subgrid. In fact, the cohesion force SPH particles contain enables particles to be merged, yielding high density and creating air bubbles large enough to be depicted in a grid. Though the creation of SPH particles was originally intended to describe details of a subgrid, the integration of SPH particles larger than subgrid size reduces the simulation accuracy of SPH particles. When the size of SPH particles outgrows that of the subgrid, we turn to grid-based level set.

$$\phi_i^{temp} = \sum_j m V_j W(\mathbf{x}_{ij}, r_j) \tag{5.18}$$

denotes the level-set value of a temporary $i$ node on a grid: $m$ mass coefficient: $V_j$ the volume of particle $j$. When an SPH particle exists on the $i$ node, it attains temporary value by using the particle on the designated node and recursively checking neighboring nodes and calculating the node value if present. When these nodes are described by grid size, temporary node value is changed into level-set value on the grid. In [47] when an SPH particle has a significantly high density, the SPH particle acts as a marker particle of particle level set and back to the level set. However, we render the level-set model by SPH particle even when our SPH particle has no marker particle of level set in order for the grid and particle method to faithfully fulfill each intended function: SPH particles in the subgrid method and level set in the grid-based method.

### 5.2.1.5 Result

These simulations were performed on an Intel Core 2 CPU 3.0 GHz and rendered by ray-tracing. The simulation adopted particle level set and BFECC method on a max level 7 octree grid and the radius of particles ranged from 0.3 to 0.8 of a leaf cell.

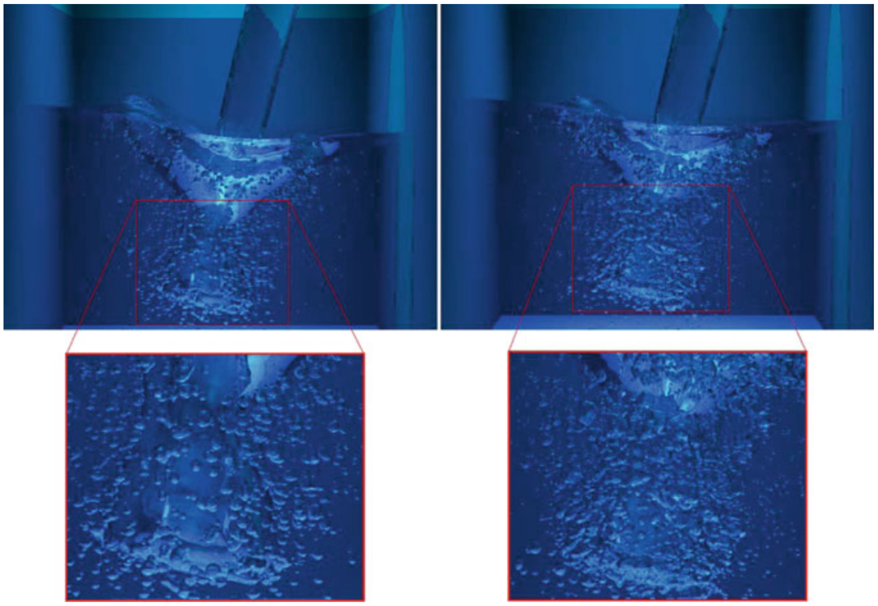


**Fig. 5.14** Example of pouring water. All the bubbles are simulated as SPH particles using [34] method on the *left* while *right* includes our method of describing large air bubbles in level set and smaller ones as SPH particles

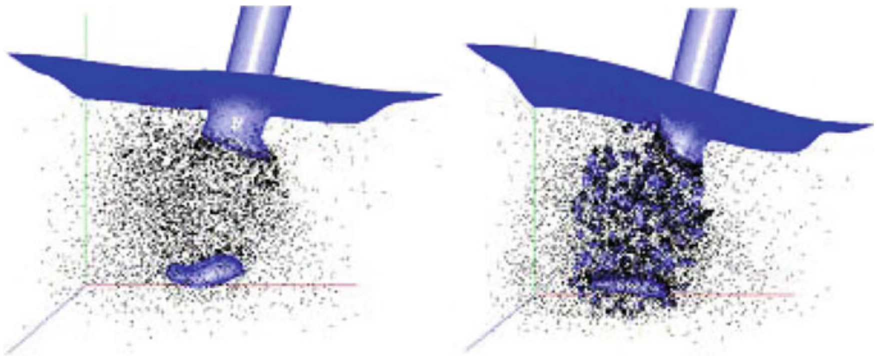


**Fig. 5.15** An example of pouring water. The *left* denotes SPH particles, showing the motion of the subgrid. Meanwhile, the *right* displays air bubbles by converting merged SPH particles into level set. *Dots* denote SPH particles

Figures 5.14 and 5.15 demonstrate a pouring water simulation. When air bubbles are merged, the left simulation renders particles intact while the right simulation converts larger air bubbles to level set and renders them. Physical dynamics were applied to SPH particles in the subgrid-based simulation and SPH bubble particles above grid size were converted into level set to ensure more accurate simulation. In Fig. 5.16,

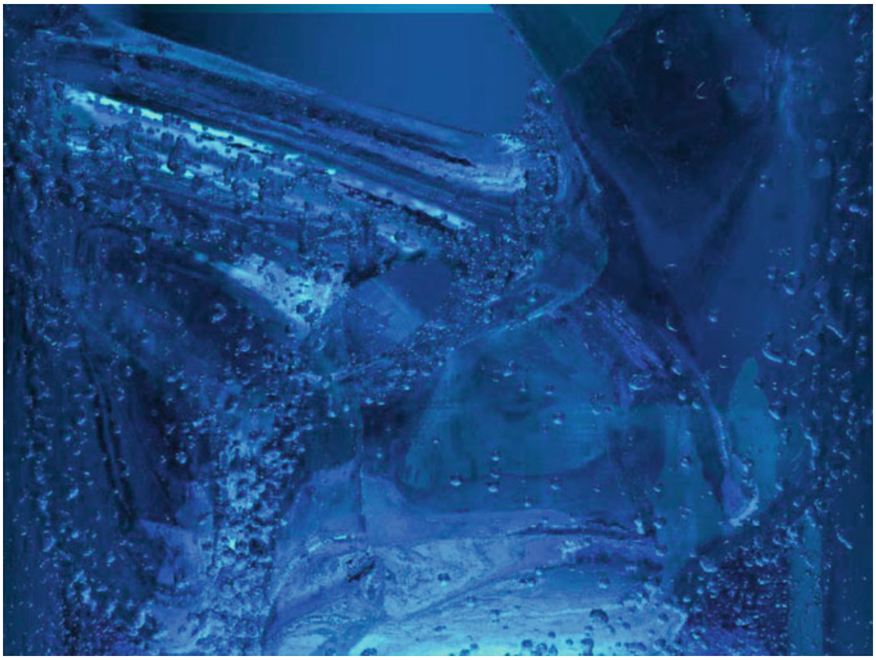


**Fig. 5.16** Pouring water in an empty box generates numerous particles. Air bubbles on the *right* side of the figure are described in level set from merged particles

pouring water on the wall of an empty box creates strong turbulence. In this state, so many escaped particles are formed that the process of transforming SPH particles to level set is clearly shown. The number of SPH particles produced in the simulation shown in Fig. 5.16 stands at 39,435 and a successful simulation is performed to show the natural motion of air bubbles from SPH particles to level set without significant computational cost.

#### 5.2.1.6 Conclusion

In a grid-based fluid simulation using Lagrangian particles, we realistically displayed small-scale details of water and air bubbles which are indefinable on a subgrid. SPH method and physical dynamics facilitated this process. Moreover, we raised the accuracy of simulation by transforming SPH particles larger than subgrid into level set, in which merged particles become large enough to be depicted on a grid. This hybrid fluid simulation more precisely and naturally illustrates water and air bubbles in multiphase fluid.

## *5.2.2 Simulation of Swirling Bubbly Water Using Bubble Particles*

**Abstract** The effect of surface tension is dynamically and realistically represented within a multiphase fluid simulation. Air bubbles are seeded with 'bubble particles', which move randomly. These molecule-like movements modify the surface of the air bubbles and generate turbulence in the water. The surface tension between air bubble and water, determined by the composition of the water, remains constant regardless of the size of the bubble, while external forces cause unstable fluid motion as the surface tension strives to remain constant, bubbles split and merge. The bubble particles can also compute for the numerical dissipation usually experienced in grid-based fluid simulations, by restoring the lost volume of individual bubbles. The realistic tearing of bubble surfaces is shown in a range of examples.

### 5.2.2.1 Introduction

Recent fluid animation techniques based on computational fluid dynamics (CFD) have achieved representation of fluid motion which are highly realistic, especially small-scale effects. The appearance of small-scale details such as splashes and bubbles can be enhanced by combining the use of particles with a grid-based simulation. This section is about the simulation of single bubbles tearing into multiple bubbles within a multiphase fluid. When air is injected from below into a container of water, it forms several bubbles which rise to the surface of the water. Simultaneously, turbulence can be observed in the water. Figure 5.17 (left) shows a rather simplistic simulation of a bubble rising through calm water. In Fig. 5.17 (right), in contrast, turbulence was simulated through surface tension, gravity, pressure and other external forces, producing a more complicated and realistic effect. This phenomenon of bubbles breaking up occurs because of a range of different forces that act at the interface between air bubbles and water. To produce a turbulence effect at the bubble interface, we seed bubbles with particles which move randomly, like molecules, and push the level set that represents the wall of a bubble, thereby tearing it into several small bubbles. This method is similar to the vortex particle method in which a dissipated vortex force is added to grid simulations. As well as tearing bubbles, our approach causes the bubbles to spiral under the lift and drag forces produced by the particle dynamics. This section also introduces a new fluid simulation in which Eulerian and Lagrangian methods are combined. The movement of the particles allows animators to create a variety of fluid motions by changing the curvature of the bubble walls.

### 5.2.2.2 Previous Work

Foster and Metaxas [21] were the first to study grid-based 3D fluid simulations using the Navier–Stokes equation; Stam [68] proposed a semi-Lagrangian integration scheme for simulating an unconditionally stable fluid. Other authors [18, 20]

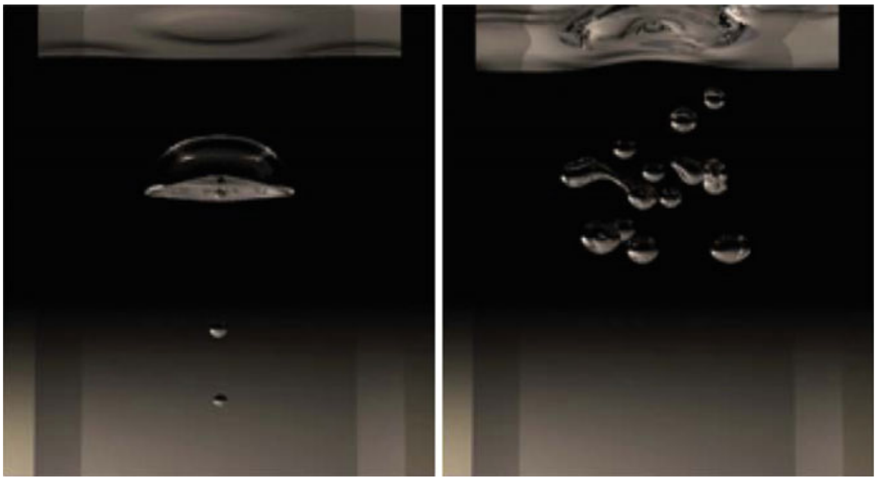


**Fig. 5.17** An air bubble rising in calm water (*left*); Air bubbles spiraling upwards (*right*)

tracked a smooth water surface using a level set and particles. Takahashi et al. [70] introduced a multiphase simulation in which gas and liquid could be modeled simultaneously; Hong and Kim [29] also simulated multiphase fluids, focusing on the surface tension between bubbles and the liquid. Other multiphase fluid simulations include numerical methods to model surface tension and viscosity changes at interfaces [31], the simulation of a fluids with many components (such as water, air, and fuel) [46], inhomogeneous flow [52], and the small-scale features of multiphase fluids [67]. Shi and Yu [69] controlled a liquid animation using a feedback force; Shin and Kim [60] controlled air bubble in a similar way. Fluid simulations using SPH (smoothed particle hydrodynamics) have been based on the particle method [51]; the hybrid technique using a grid with particles models small-scale droplets and bubbles [25, 37]. Fedkiw et al. [24] introduced vorticity confinement to model the small-scale rolling features characteristic of smoke. Selle et al. presented the vortex particle method [65] to model highly turbulent effects such as explosions or rough water. Hong et al. developed a simulation of bubbly water, in which the sub-grid details were improved by incorporating SPH into a coarse grid [34]. Losasso et al. also improved this technique by coupling a model of dense water volumes to the diffuse regions [47]. These papers represented lively bubble and splash using SPH particles in grid simulation, but in this section we try to simulate turbulent bubbly water using bubble particles.

#### 5.2.2.3 Fluid Simulation

The Navier–Stokes equation provides the simulation of incompressible fluid and preserves mass and momentum.

$$\mathbf{u}_t + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p/\rho = \mathbf{f} \tag{5.19}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{5.20}$$

where $\rho$ is the density of the fluid, and $\mathbf{f}$ is external force which includes gravity and control forces. This equation can be solved in two steps. First, intermediate velocity $\mathbf{u}^*$, when pressure is not considered, is solved.

$$\mathbf{u}^* = \mathbf{u} + \Delta t\,(-(\mathbf{u} \cdot \nabla)\mathbf{u} + \mathbf{f}) \tag{5.21}$$

Since $\nabla \cdot \mathbf{u}$ should be zero, the pressure profile is determined by solving Eq. (5.23); and we get the final velocity profile by Eq. (5.22).

$$\mathbf{u} = \mathbf{u}^* - \frac{\Delta t}{\rho}\nabla p \tag{5.22}$$

$$\nabla \cdot \mathbf{u} = \nabla \cdot \mathbf{u}^* - \frac{\Delta t}{\rho}\nabla^2 p = 0 \tag{5.23}$$

We use an octree [43], to focuses on the interface between the air and water. The particle level-set method [18, 20] makes this interface a smooth surface (Fig. 5.18).

#### 5.2.2.4 Bubble Particle Method

Water has tension that minimizes the surface in a free surface; thus air bubbles in water tend to become spherical. However, fluids are also subject to external forces such as gravity, pressure, and turbulence that can tear air bubbles into pieces. We use the seeding of bubble particles to provide a more realistic simulation of bubbles spiraling upward (Fig. 5.19).

**Generation of Bubble Particles**

We create bubble particles that move like molecules as follows:

(1) Search for air bubbles cell by cell; when a cell with an air bubble is detected, an index is assigned to that cell. If the same air bubble is also found in neighboring cells, the same index is given to those cells. The volume of the air bubble is then calculated. In the case of a cell completely full of air, the volume is the cube of the cell's length. When a cell includes the interface between air and water, the volume is calculated by Monte Carlo integration.
(2) Continue this procedure until no air bubble remains without an index.
(3) Find a leaf cell in the octree which includes a bubble particle, and assign the index of that cell to that particle.
(4) If there is no bubble particle in an air bubble, new bubble particles are seeded within that bubble.

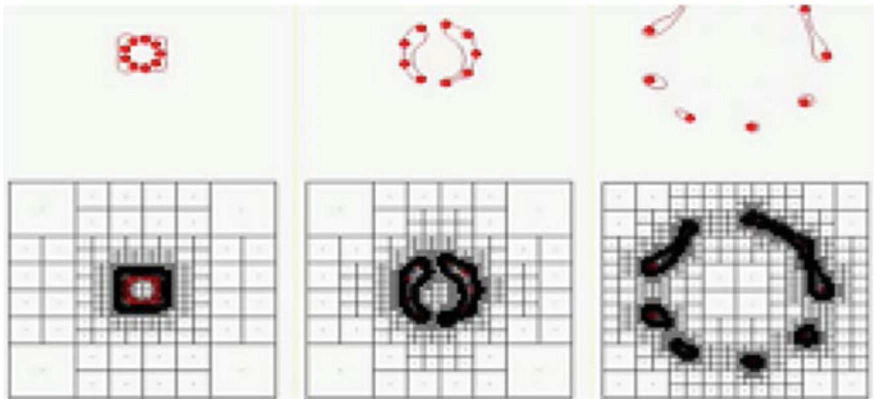


**Fig. 5.18** Level-set changes in evolving bubble particles (in two dimensions)

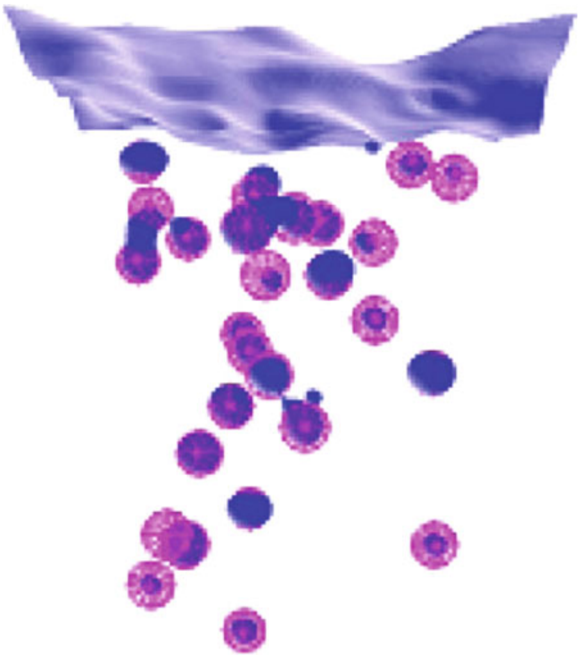


**Fig. 5.19** Bubble particles are seeded in every air bubble; the *red spheres* in this three dimensional example are bubble particles

(5) If bubble particles with different indices are found within one air bubble, it means two air bubbles have merged. So the bubble particles should be given the same index and the volume of the air bubble should be recalculated. When bubble particles with the same index are found in different air bubbles, it means the air bubble has split. In this case the bubble particles should be given another index and the volume should be recalculated as well.

### Movement of a Bubble Particle

A bubble particle moves in an air bubble with a random path like a molecule exhibiting Brownian motion. Bubble particle is given an initial velocity and its subsequent movement is irregular. We model this irregular motion as follows:

$$\mathbf{u}_{bp}^{n+1} = \mathbf{u}_{bp}^{n} + k\nabla\phi, \qquad \phi \le r_{bp} \tag{5.24}$$

where $\mathbf{u}_{bp}^{n}$ is a bubble particle's velocity in the current time-step; $\mathbf{u}_{bp}^{n+1}$ is the bubble particle's velocity in the next time-step; $k$ is the coefficient of the degree to which the bubble particle is pushed toward a higher gradient of $\phi$, so the bubble particle bounces against the interface between the air and water, evolving like a molecule. $\phi$ is the level-set value of a bubble particle's position. $r_{bp}$ is a bubble particle's radius. If a bubble particle is near the interface between water and bubble, that is, the level-set value of the bubble particle's position is almost zero, the bubble particle turns in the direction of the level set's gradient of its position, causing it to move as a molecule does. Authors of [13, 50] simulated using drag force ($\mathbf{f}^{drag} = \frac{1}{2}\rho\pi r^2|\mathbf{v}^{particle} - \mathbf{u}^{liquid}|(\mathbf{v}^{particle} - \mathbf{u}^{liquid})$) and lift force ($f^{lift} = -\rho V(\mathbf{v}^{particle} - \mathbf{u}^{liquid}) \times \nabla \times \mathbf{u}$). The drag force offsets the particle's velocity by the velocity of liquid; the lift force generates vortex force around the particle's position. These forces cause the spiraling movement of an air bubble, but in their simulations, the air bubble was made of a particle. This can be seen in the interaction between the air bubble and water. In contrast, in this section, air bubbles move upward, while swirling and changing level set, which makes the simulation appear more realistic.

### The Effect of a Bubble Particle on the Surface of an Air Bubble

A bubble particle affects the velocity of its neighbors in the grid, so the level set of the fluid and the surface of air bubbles change. Consequently, air bubbles split while spiraling. The neighboring velocity's affection equation by bubble particle's velocity is:

$$\mathbf{u}_{nb}^{n+1} = \mathbf{u}_{nb}^{n} + s\mathbf{u}_{bp}^{n}, \qquad |\mathbf{P}_{nb} - \mathbf{P}_{bp}| \le r_{bp} \tag{5.25}$$

where $\mathbf{u}_{np}^{n+1}$ is a neighboring node's new velocity, $\mathbf{u}_{nb}^{n}$ is the neighboring node's current velocity, $s$ is a coefficient of a neighboring node's velocity pushed by a bubble particle, $\mathbf{u}_{bp}^{n}$ is a bubble particle's velocity. $\mathbf{P}_{nb}$ is a neighboring node's position, and $\mathbf{P}_{bp}$ is a bubble particle's position. This process should be performed before the projection step. The bubble surface is changed by updating neighboring node's velocities; the bubble tears into several smaller bubbles and rise in a zig zag manner. The vortex particle method [65] also attempted to make turbulence, but it did not present bubbles tearing at their rims. We assist bubble particles to generate Rayleigh instability phenomenon; air bubbles appear alive without the lift and drag forces of a particle system.

#### 5.2.2.5 Correcting Air Bubble Volumes

In grid-based fluid simulations, advection causes the dissipation of air bubble volumes; the larger the size of a cell in the grid and the time-step, the bigger this dissipation becomes. When an air bubble moves up that is smaller than the size of a cell, it disappears, in most cases, by numerical dissipation before reaching the surface. BFECC and volume control method [40] have been studied in order to significantly reduce this numerical dissipation. We also correct bubbles' volume with a similar method. We got bubble volume in Sect. 5.2.2.4, and assumed $V_i^n$ is the volume of index $i$ at $n_{th}$ time-step. If there is no dissipation of an air bubble, $V_i^n = V_i^{n-1}$; if there is dissipation, when $x_i^n$ is the volume dissipated, $x_i^n = V_i^n - V_i^{n-1}$. The volume dissipated should be compensated for in the air bubbles; divergence value $c_i^n = -k_p x_i^n$, where $k_p$ is the bubble loss compensation coefficient. This divergence value $c_i^n$ is calculated in the projection step.

In a cell without bubble particles

$$\nabla^2 p = \frac{\rho}{\Delta t}(\nabla \cdot \mathbf{u}^*) \tag{5.26}$$

and in a cell that contains bubble particles

$$\nabla^2 p = \frac{\rho}{\Delta t}(\nabla \cdot \mathbf{u}^* - c_i^n) \tag{5.27}$$

When the projection step is calculated as above, the air bubble with bubble particles is compensated for its loss of volume. Although this compensation solution is similar to that of [40], they differ in the following ways: first whereas [40] compensated for the loss of whole fluid, our simulation focuses on air bubbles, only compensating for the volume loss of air bubbles; second, our method also checks the mergers and the splits of air bubbles by bubble particles. This is its strength.

#### 5.2.2.6 Result

Simulations were performed on a PC with an Intel Core2 CPU 3.0, and rendered by ray tracing. In Fig. 5.20, we compare bubble particle method with base simulation, vorticity confinement method, and vortex particle method. The bubble particle method shows swirling bubbly water while other simulations fail to present tearing at the rim of air bubbles. The extra computational cost incurred by using bubble particle was about 10 % overhead. Figure 5.21 shows air bubbles interacting with a rigid sphere; In Fig. 5.22, bubble particles, seeded in air bubbles, change the level-set value by moving, thereby creating lively fluid motions in calm water.

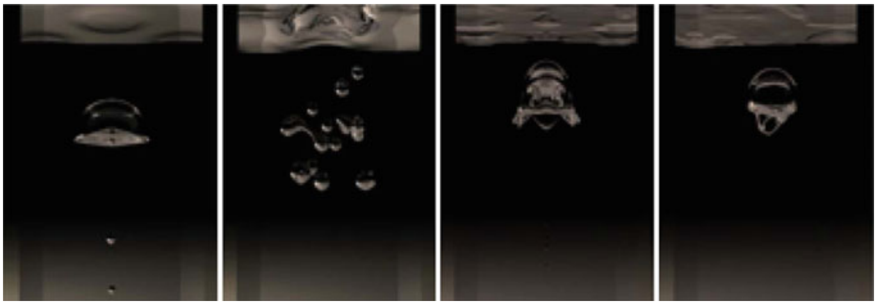


**Fig. 5.20** Images from *left* to *right*: base simulation, bubble particle method, vorticity confinement method, and vortex particle method

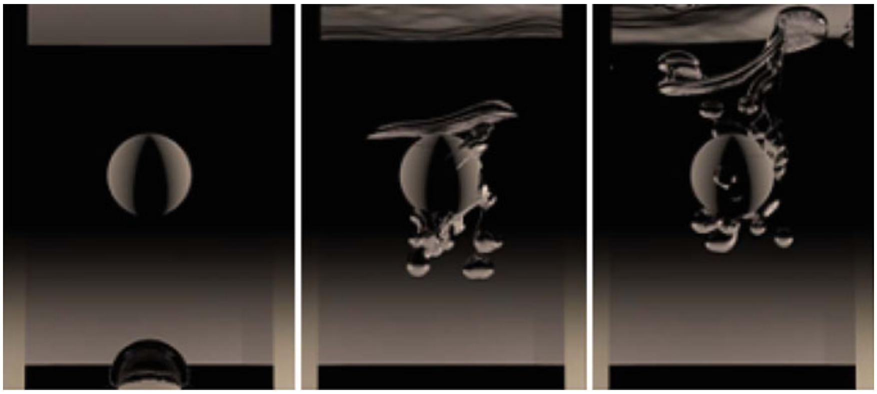


**Fig. 5.21** Rising air bubbles with bubble particles interacting with a rigid *sphere*

#### 5.2.2.7 Conclusion

We seeded Lagrangian bubble particles in air bubbles and caused them to move like molecules in order to create turbulence and consequently simulate realistic fluid in a grid-based fluid simulation. Large air bubbles tear into several small bubbles; small bubbles zig zag. When the Rayleigh instability phenomenon appears in this simulation, the flow field must be considered in the future work. We removed the dissipation of air bubbles by making bubble particles check the volumes of air bubbles. In conclusion, the contributions of this section is that it enables animators to control various fluid motions by creating turbulence with bubble particles and correcting the volume of air bubbles (Table 5.2).

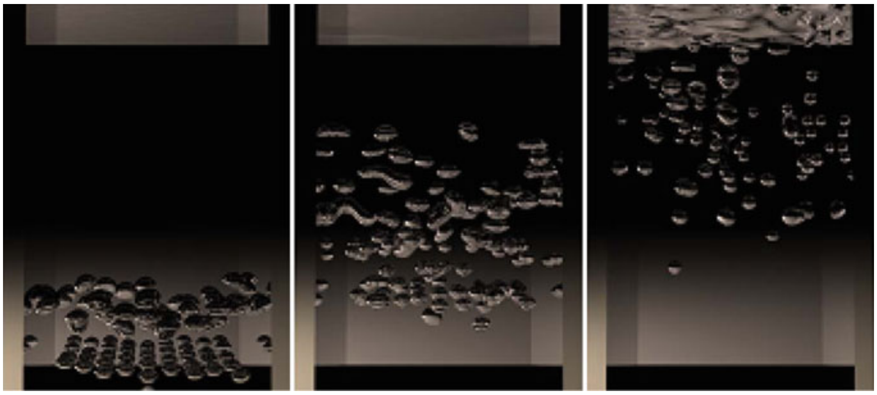


**Fig. 5.22** Bubbles zigzagging as they rise

**Table 5.2** Comparison of simulation time bubble particle method with base simulation, vorticity confinement, and vortex particle method in 128 × 256 × 128 grid simulation

| Base simulation (sec/frame) | Bubble particle method (sec/frame) | Vorticity confinement (sec/frame) | Vortex particle (sec/frame) |
|---|---|---|---|
| 82.1 | 91.7 | 85.2 | 88.4 |

## *5.2.3 Controlling Shapes of Air Bubbles in a Multiphase Fluid Simulation*

**Abstract** Controlling shapes is a challenging problem in a multiphase fluid simulation. Bubble particles enable the details of air bubbles to be represented within a simulation based on a Euler grid. We control the target shapes of bubbles by the gradient vectors of the signed distance field and attraction forces associated with control particles. This hybrid approach enables to simulate physically plausible movements of bubbles while preserving the details of a target shape. Furthermore, we control the paths of moving bubbles using user-defined curves and the shape of an air bubbles by drag force. An accurate model of the drag force near the fluid surface means that bubbles have realistic ellipsoidal shapes.

### 5.2.3.1 Introduction

In recent SF movies, physics-based fluid simulation has been used to produce effects which are hard to distinguish from reality. In the computer game industry, too, real-time fluid simulation methods have become a common ingredient in plausible virtual worlds. Target-driven fluid simulation techniques are required to produce these special effects; but it remains challenging to control fluid flow while preserving its physical characteristics.

In this section, we adopt the smoothed particle hydrodynamics (SPH) technique based on grid-based multiphase fluid simulation [34]. Particles are simulated by solving the Navier–Stokes equation, and the target shapes of air bubbles are controlled by external forces; as a result, our method captures small-scale details which are ignored by existing target-driven grid-based methods, which allow features of the target shape to disappear due to numerical dissipation. Conversely, particle-based approaches are fast and preserve features of the target shape but have difficulty in simulating multiphase fluids. By combining grid-based and particle-based techniques, we can simulate multiphase fluids while preserving target shapes.

A hybrid method of Eulerian and Lagrangian fluid simulations was introduced by Greenwood and House [25]. They simulated air bubbles as particles escaping from the particle level set [18], whereas Hong et al. [34] simulated escaped particles by SPH. By modeling an air bubble as a combination of several particles interacting with each other within the SPH vorticity confinement, it is possible to obtain bubbles with dynamically deforming shapes. However, this approach is still not adequate to represent real bubbles accurately, because the drag forces applied to each particle in an air bubble are almost the same, and so the bubble stays spherical. In reality, a rising bubble deforms into ellipsoid or a mushroom shape because a large drag force acts on its upper surface. Unlike Hong et al., our computation of the drag force applied to the particles near the bubble surface takes account of the overall shape of the air bubble to produce more realistic results.

We simulate the complicated behavior of fluids and air bubbles by the interaction between grid-based liquids and SPH-based air bubbles, subject to a vorticity force and a cohesive force. In addition, this section proposes techniques to control the flow of air bubbles using curves and the shape of bubbles using target shapes. The resulting fluid movements are stable without artifacts.

#### 5.2.3.2 Related Work

Stam [68] proposed a semi-Lagrangian integration scheme to simulate unconditionally stable fluids using a three-dimensional grid. Enright et al. [18] contributed to a particle level-set method to detect the fluid surface accurately using marker particles. Losasso et al. [43] utilized an adaptive octree structure to obtain a high-resolution surface. Kim et al. proposed volume control method and regional level-set method for multiphase fluid simulation [38, 40]. Hong and Kim [31] considered surface tension between multiphase fluids using the ghost fluid method (GFM) to deal with discontinuities at the fluid surface. Robinson-Mosher et al. introduced interactions between rigid bodies and a fluid [54]. Hong and Kim [30] and Shin and Kim [60] controlled the shape of a fluid using a gradient-based non-linear method, and Shi and Yu [69] introduced a target-driven method to force fluids to follow a rapidly deforming target shape.

Desbrun and Gascuel [14] used SPH to animate elastic bodies. Müller et al. [49] contributed to the simulation and rendering of liquid using the SPH. Müller et al. [49] devised a particle system to animate elastic and plastic objects with specified material properties. Adams et al. [1] proposed adaptively-sized SPH particles. Thürey et al. [72] introduced a technique to control liquid, which preserves its details using high-pass and low-pass filters.

Fluid simulation techniques can be divided into grid-based Eulerian and particle-based Lagrangian methods; and hybrid techniques combine these two methods. Greenwood and House [25] represented air bubbles positively as particles escaping from a particle level set. However, since they treat an air bubble as a sphere, the results are unrealistic. But in later work [45, 47] air bubbles and water drops are simulated using SPH, and their realism is complete. Kim et al. [42] represented the flow of dispersed bubbles using a stochastic solver, but their method still has a limited ability to deal with the collision, merger and overlapping of particles. Rasmussen et al. [54] used a hybrid method to control fluids for photorealistic effects.

Additionally, we control the path and shape of air bubbles. The application of [11] allows animators to control fluids intuitively. The path is used to define the location of fluid flow. The direction of drag is by definition opposite to the velocity. A bubble has been observed in experiments that the short axis of the ellipsoidal bubble is always aligned with the bubble velocity vector [63]. Jo et al. [36] proposed a new SPH simulation method that utilizes ellipsoidal kernels instead of spherical kernels.

#### 5.2.3.3 Fluid Simulation

We use the Navier–Stokes equation to simulate large volumes of water. The momentum conservation equation is

$$\mathbf{u}_t + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p/\rho = \mathbf{f}, \tag{5.28}$$

and the mass conservation equation is

$$\nabla \cdot \mathbf{u} = 0, \tag{5.29}$$

where $\mathbf{u} = (u, v, w)$ is velocity, $p$ is pressure, $\rho$ is density, and $\mathbf{f}$ is the sum of the external forces, including gravity and control forces. We use the particle level-set method [18] to represent complicated fluid surfaces, and octree structure [43] for fast grid simulation, and back and forth error compensation and correction (BFECC) [41] to achieve second-order accuracy in preserving the fluid volume. We employ the adaptive SPH technique [1] to simulate small-scale air bubbles. The equation for the pressure force acting on an air bubble is

$$\mathbf{f}_{ij}^{\text{pressure}} = -V_i V_j (P_i + P_j)(\nabla W(\mathbf{x}_{ij}, r_i) + \nabla W(\mathbf{x}_{ij}, r_j))/2, \tag{5.30}$$

where the volume $V_i$ is $m_i/\rho_i$, $r$ is the radius, the mass $m_i$ is proportional to $r_i^3$ and $\mathbf{x}_{ij} = \mathbf{x}_j - \mathbf{x}_i$; the pressure $P_i = k\rho_i$, where $k$ is a control parameter. Using an Eulerian grid to model water and Lagrangian SPH to model air bubbles, we employ the SPH vorticity confinement and cohesive forces due to Hong et al. [34] to improve the realism of the air bubbles.

### 5.2.3.4 Control of Air Bubbles

#### Controlling Target Shapes of Air Bubbles

In this section, we control the shape of air bubbles using hybrid method. To attract air bubbles to a target shape, we create control particles in regions which have negative level-set values in the region of the signed distance field corresponding to the target region [10]. The control force $\mathbf{f}_{\text{ctl}}$ applied to an air particle is determined by the negative gradient vector of the signed distance field $-\nabla\phi_{\text{shape}}$ and the attraction force $\mathbf{f}_a$, as follows:

$$\mathbf{f}_{\text{ctl}} = -\alpha\nabla\phi_{\text{shape}} + (1-\alpha)\mathbf{f}_a, \tag{5.31}$$

$$\alpha = \begin{cases} k_{\text{ctl}}\phi_p & \phi_p > 0, \\ 0 & \phi_p \leq 0, \end{cases} \tag{5.32}$$

where $\phi_p$ is the level-set value at the position of an air particle $p$, and $\alpha$ is linearly proportional to the distance between $p$ and the surface of the target shape. If the level-set value $\phi_p$ is negative, $\alpha = 0$ and the control particle only applies the attraction force $\mathbf{f}_a$. We model the effect of $\mathbf{f}_a$ on an air particle, which makes it move toward control particles in a similar way to Thürey et al. [72]:

$$\mathbf{f}_a = \sum_i \beta_i \frac{\mathbf{c}_i - \mathbf{p}}{\|\mathbf{c}_i - \mathbf{p}\|} W(d, h), \tag{5.33}$$

where $\mathbf{c}_i$ is the position of the control particle $i$, $\mathbf{p}$ is the position of the air particle, and $W$ is the kernel function of the control particle; $d$ is the distance between $\mathbf{c}_i$ and $\mathbf{p}$, $h$ is the kernel size, and $\beta_i$ is a coefficient which is inversely proportional to the density of air particles at the control particle $i$. Therefore, if there are already enough air particles around the control particle, further air particles are only weakly attracted. In Fig. 5.23, the bubble particle moves into the target shape.

#### The Drag Force at a Bubble Surface

Since the density of water is 800 times that of air, the force applied to water by air is insignificant. Conversely, water induces buoyancy, drag, and left forces on bubbles. For example, in the left-hand image in Fig. 5.24, a drag force is applied to the upper portion of the air bubble by buoyancy. Ellingsen and Risso [19] observed that an air bubble injected into water retains its ellipsoidal shape while it rises, implying that the drag force only acts on the upper portion of the air bubble in Fig. 5.25. Although the interaction between water and air bubbles occurs at the upper surface, the resulting force affects other particles by solving the SPH. Therefore, we do not need to calculate

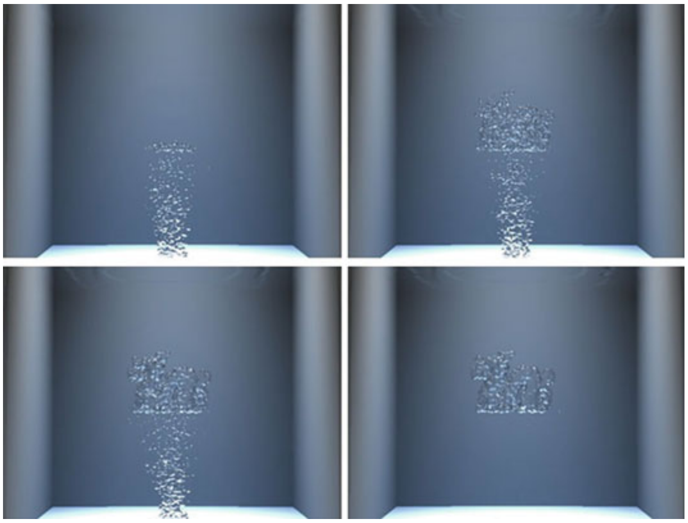


Fig. 5.23 An animation of the varying shape of air bubbles

the drag forces of particles which are located in lower portion of air bubble. Hong et al. [34] and Cleary et al. [13] computed the interaction between water and air bubbles at each SPH particle. Then, as they use in the right-hand image in Fig. 5.24, bubbles are influenced by the drag force which is almost omnidirectional, and the shape of a bubble is spherical. Conversely, we model the drag force $\mathbf{f}_i^{\text{drag}}$ acting on an air particle $i$ taking account of the position and direction of the bubble, so that

$$\mathbf{f}_i^{\text{drag}} = -k_{\text{drag}} r_i^2 |\mathbf{v}_{\oplus} - \mathbf{u}_{\oplus}|(\mathbf{v}_{\oplus} - \mathbf{u}_{\oplus}), \tag{5.34}$$

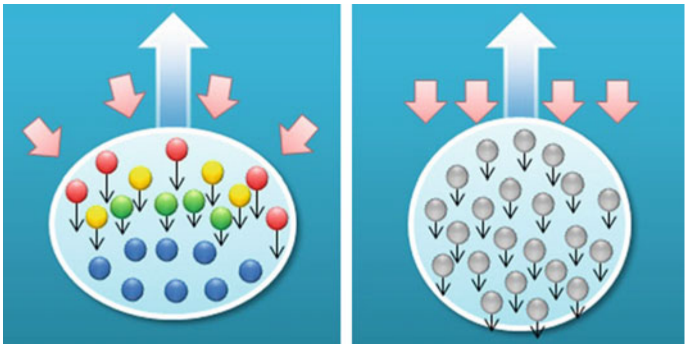


Fig. 5.24 The allocation of the drag force to air using particles in our method (*left*) and that of Hong et al. (*right*) [34]

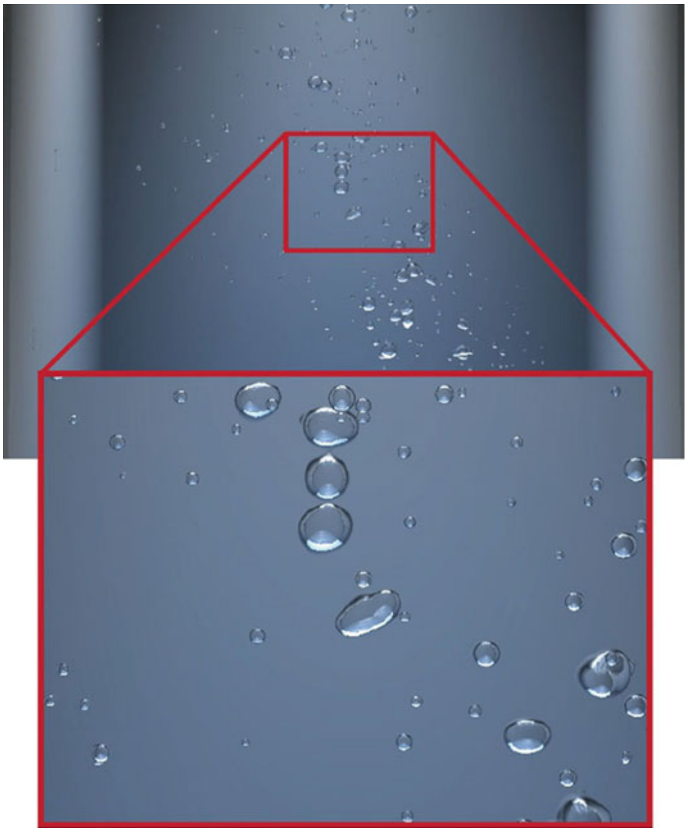


**Fig. 5.25** Ellipsoidal shape of air bubbles using our proposed drag force

where $\mathbf{p}_i$ is the position of an air particle $i$, $\mathbf{p}_\oplus$ is the center of the bubble and $\mathbf{p}_\oplus = \frac{1}{N}\sum_i^n \mathbf{p}_i$; $\mathbf{v}_\oplus$ is the velocity of $\mathbf{p}_\oplus$, and $\mathbf{u}_\oplus$ is the velocity on the grid of the fluid at position $\mathbf{p}_\oplus$. $N$ is the number of particles in one bubble. Since our formation of the drag force only affects the upper surface of a bubble, $k_{\text{drag}}$ of the drag force is as follows:

$$k_{\text{drag}} = \begin{cases} \eta\gamma|\mathbf{p}_i - \mathbf{p}_\oplus| & \gamma > 0, \\ 0 & \gamma \leq 0, \end{cases} \tag{5.35}$$

where $\gamma = (\mathbf{p}_i - \mathbf{p}_\oplus) \cdot (\mathbf{v}_i - \mathbf{u}_\oplus)$.

If $\gamma$ is negative, the particle is not influenced by the drag force; $k_{\text{drag}}$ is zero. $\eta$ is the coefficient of $k_{\text{drag}}$. Therefore, as shown in the left-hand image in Fig. 5.24, air particles (red > yellow > green) near the upper surface of a bubble experience strong drag force; but these forces barely affect particles (blue) in the interior of the bubble or near its lower surface.

### Control of Air Bubble Flow

We provide users with artistic control over the flow of air bubbles by allowing them to specify a curve $\mathbf{f}(t)$. $L_1$ and $L_2$ are other curves paralleled to a curve $\mathbf{f}(t)$. If the position of bubble particle $\mathbf{p}_n$ is on the curve $L_n$, the $\mathbf{f}_{\text{curve}}$ is as follows:

$$\mathbf{f}_{\text{curve}} = -\mu \frac{\mathbf{g}'(t)}{|\mathbf{g}'(t)|} \tag{5.36}$$

where $\mathbf{g}(t) = \mathbf{f}(t)\mathbf{v}_n$, $\mathbf{v}_n$ is translation vector for curve $L_n$ and $\mu$ is coefficient of $\mathbf{f}_{\text{curve}}$. This causes bubble to follow paths approximately parallel to the user-defined curve.

### 5.2.3.5 Result

Simulations were performed on an Intel Core i7 CPU running at 2.93 GHz, and rendered the fluid models by ray-tracing.

In Fig. 5.23, the air bubbles gather to form a target dragon model. The flow of the bubbles is controlled by the sum of the gradient vectors of the signed distance field of the target model and the attraction force of the control particles. Although water was simulated with a coarse grid, air particles move freely toward the control particles, which are independent of the grid. Air particles were added at each frame, and the number of particles at the last scene in Fig. 5.23 is 13,815.

The shape and path of air bubbles were controlled in a multiphase fluid simulation to demonstrate user-controlled motions. Figure 5.26 shows air bubbles controlled by user defined curves in Fig. 5.27. The bubbles appear to rise naturally, but the external force associated with the curves makes them follow its paths. The left-hand, middle, and right-hand images in Fig. 5.26 are the 31st, 58th, and 253rd frames of the simulation, and the numbers of particles in the simulation at each frame are 390, 676, and 1,030, respectively.

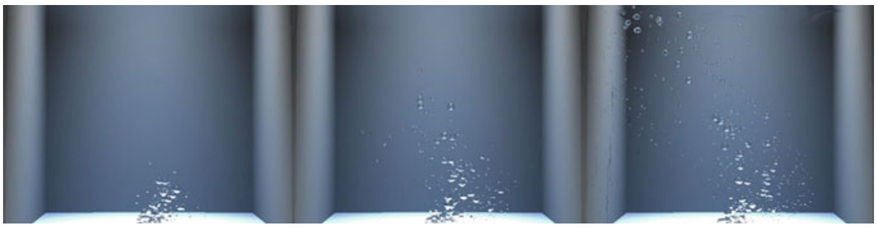


**Fig. 5.26** User control of the flow of bubbles: the 31st frame, 58th frame, and 253rd frame of a movie

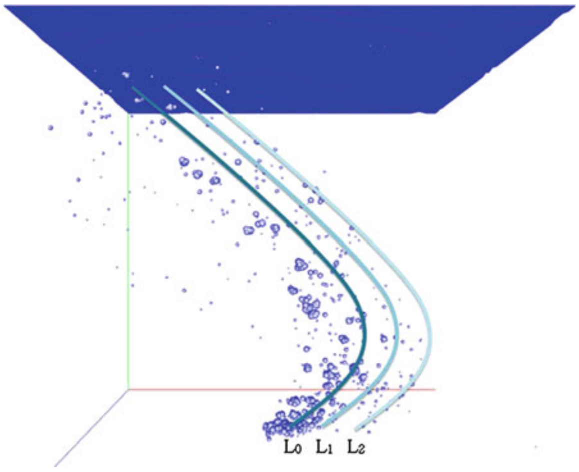


**Fig. 5.27** The path of bubbles controlled by user-defined *curves*

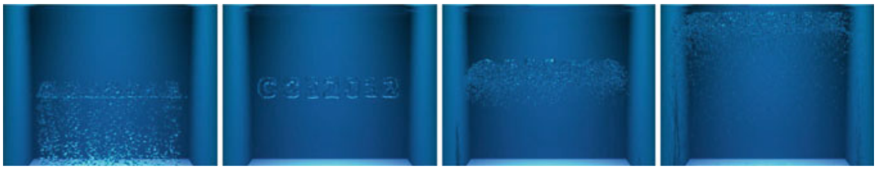


**Fig. 5.28** The simulation of dispersed air bubbles after tracing the target shape CGI 2012

In the example of Fig. 5.28, the control force $\mathbf{f}_{\text{ctl}}$ in Eq. (5.31) drives the air bubbles to form the characters 'CGI 2012'. In the third image, the bubbles disperse as all control forces are removed, and finally they move buoyantly upward.

### 5.2.3.6 Conclusion

This section has introduced a method to control the shape and flow of air bubbles in a multiphase fluid simulation. Using simple functions such as a curve or a three-dimensional mesh, users can control the motion of the bubbles. We derive parallel curves from the simple curve so that bubbles which are not located on the curve move on parallel paths. To control the shape of the bubbles, we combine the gradient vectors of the signed distance field of the target shape and the attraction forces associated with the control particles. Despite the provision of user control, the simulation results look physically plausible. Since the drag force only affects the upper part of the surface of a bubble, the bubbles maintain their ellipsoidal shape as they rise.

## References

1. Adams B, Pauly M, Keiser R, Guibas LJ (2007) Adaptively sampled particle fluids. In: Proceedings of ACM SIGGRAPH 2007. ACM Transactions on Graph (TOG), vol 26(3), pp 481–487
2. Ando R, Tsuruno R (2011) A particle-based method for preserving fluid sheets. In: Proceedings of the 2011 ACM SIGGRAPH/Eurographics symposium on computer animation, pp 7–16
3. Ando R, Thürey N, Tsuruno R (2012) Preserving fluid sheets with adaptively sampled anisotropic particles. IEEE Trans Vis Comput Graph 18(8):1202–1214
4. Abdel-Malek K, Yang Y, Blackmore D, Joy K (2006) Swept volumes: foundations perspectives and applications. Int J Shape Model 12(1):87–127
5. Baerentzen J, Aanaes H (2005) Signed distance computation using the angle weighted pseudo-normal. IEEE Trans Vis Comput Graph 11(3):243–253
6. Bridson R, Fedkiw R, Anderson J (2002) Robust treatment of collisions contact and friction for cloth animation. ACM Trans Graph (TOG) 21(3):594–603
7. Bridson R, Marino S, Fedkiw R (2003) Simulation of clothing with folds and wrinkles. In: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on computer animation, pp 28–36
8. Becker M, Teschner M (2007) Weakly compressible SPH for free surface flows. In: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on computer animation, pp 209–217
9. Becker M, Tessendorf H, Teschner M (2009) Direct forcing for Lagrangian rigid-fluid coupling. IEEE Trans Vis Comput Graph 15(3):493–503
10. Bell N, Yu Y, Mucha PJ (2005) Particle-based simulation of granular materials. In: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on computer animation (SCA 05), pp 77–86
11. Coleman P (2002) Motion control for fluid animation: flow along a control path. Technical report, Ohio State University, Undergraduate thesis
12. Chorin AJ (1967) A numerical method for solving incompressible viscous flow problems. J Comput Phys 2(1):12–16
13. Cleary PW, Pyo SH, Prakash M, Koo BK (2007) Bubbling and frothing liquids. ACM Trans Graph (TOG) 26(3):971–976
14. Desbrun M, Gascuel M-P (1996) Smoothed particles: a new paradigm for animating highly deformable bodies. In: Proceedings of the Eurographics workshop on computer animation and simulation, pp 61–76
15. Erleben K, Dohlmann H (2007) GPU Gems 3:611–632
16. Ehmann S, Lin M (2000) Accelerated proximity queries between convex polyhedra by multi-level voronoi marching. In: Proceedings of IEEE/RSJ international conference, vol 3, pp 2101–2106
17. Ehmann SA, Lin MC (2001) Accurate and fast proximity queries between polyhedra using convex surface decomposition. Comput Graph Forum 20(3):500–510
18. Enright D, Marschner S, Fedkiw R (2002) Animation and rendering of complex water surfaces. ACM Trans Graph (TOG) 21(3):736–744
19. Ellingsen K, Risso F (2001) On the rise of an ellipsoidal bubble in water: oscillatory paths and liquid-induced velocity. J Fluid Mech 440:235–268
20. Foster N, Fedkiw R (2001) Practical animation of liquids. In: Proceedings of the 28th annual conference on computer graphics and interactive techniques, pp 23–30
21. Foster N, Metaxas D (1996) Realistic animation of liquids. Graph Models Image Process 58(5):471–483
22. Frisken SF, Perry RN, Rockwood AP, Jones TR (2000) Adaptively sampled distance fields: a general representation of shape for computer graphics. In: Proceedings of the 27th annual conference on computer graphics and interactive techniques, pp 249–254
23. Fuhrmann A, Sobottka G, Grob C (2003) Distance fields for rapid collision detection in physically based modeling. In: Proceedings of GraphiCon, pp 58–65

24. Fedkiw R, Stam J, Jensen HW (2001) Visual simulation of smoke. In: Proceedings of the 28th annual conference on computer graphics and interactive techniques, pp 15–22
25. Greenwood ST, House DH (2004) Better with bubbles: enhancing the visual realism of simulated fluid. In: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on computer animation, pp 287–296
26. Goldenthal R, Harmon D, Fattal R, Bercovier M, Grinspun E (2007) Efficient simulation of inextensible cloth. ACM Trans Graph (TOG) 26(3):49
27. Gottschalk S, Lin MC, Manocha D (1996) OBBTree: a hierarchical structure for rapid interference detection. In: Proceedings of the 23rd annual conference on computer graphics and interactive techniques, pp 171–180
28. Guéziec A (2001) Meshsweeper: dynamic point-to-polygonal mesh distance and applications. IEEE Trans Vis Comput Graph 7(1):47–61
29. Hong J-M, Kim C-H (2003) Animation of bubbles in liquid. Comput Graph Forum 22(3): 253–262
30. Hong J-M, Kim C-H (2004) Controlling fluid animation with geometric potential. Comput Animat Virtual Worlds 15(3–4):147–157
31. Hong J-M, Kim C-H (2005) Discontinuous fluids. ACM Trans Graph (TOG) 24(3):915–920
32. Harada T, Koshizuka S, Kawaguchi Y (2007) Improvement in the boundary conditions of smoothed particle hydrodynamics. Comput Graph Geom 9(3):2–15
33. Huang J, Li Y, Crawfis R, Lu SC, Liou SY (2001) A complete distance field representation. In: Proceedings of the conference on visualization'01, pp 247–254
34. Hong J-M, Lee H-Y, Yoon J-C, Kim C-H (2008) Bubbles alive. ACM Trans Graph (TOG) 27(3):48
35. Houston B, Wiebe M, Batty C (2004) RLE sparse level sets. In: ACM SIGGRAPH 2004 sketches, pp 137–137
36. Jo E, Kim D, Song O-Y (2011) A new SPH fluid simulation method using ellipsoidal kernels. J Vis 14(4):371–379
37. Kim J, Cha D, Chang B, Koo B, Ihm I (2006) Practical animation of turbulent splashing water. In: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on computer animation, pp 335–344
38. Kim B (2010) Multi-phase fluid simulations using regional level sets. ACM Trans Graph (TOG) 29(6):175
39. Kim D, Ko H-S (2007) Eulerian motion blur. In: Eurographics workshop on natural phenomena, pp 39–46
40. Kim B, Liu Y, Llamas I, Jiao X, Rossignac J (2007) Simulation of bubbles in foam with the volume control method. ACM Trans Graph (TOG) 26(3):98
41. Kim B, Liu Y, Llamas I, Rossignac J (2005) Flowflxer: using BFECC for fluid simulation. In: proceedings of the first Eurographics conference on natural phenomena, pp 51–56
42. Kim D, Song O-Y, Ko H-S (2010) A practical simulation of dispersed bubble flow. ACM Trans Graph (TOG) 29(4):70
43. Losasso F, Gibou F, Fedkiw R (2004) Simulating water and smoke with an octree data structure. ACM Trans Graph (TOG) 23(3):457–462
44. Lefebvre S, Hoppe H (2007) Compressed random access trees for spatially coherent data. In: Proceedings of the 18th Eurographics conference on rendering techniques, pp 339–349
45. Lee H-Y, Hong J-M, Kim C-H (2009) Interchangeable sph and level set method in multiphase fluids. Vis Comput 25(5–87):713–718
46. Losasso F, Shinar T, Selle A, Fedkiw R (2006) Multiple interacting liquids. ACM Trans Graph (TOG) 25(3):812–819
47. Losasso F, Talton J, Kwatra N, Fedkiw R (2008) Two-way coupled SPH and particle level set fluid simulation. IEEE Trans Vis Comput Graph 14(4):797–804
48. Mauch S (2003) Efficient algorithms for solving static Hamilton-Jacobi equations. Ph.D. thesis, California Institute of Technology
49. Müller M, Charypar D, Gross M (2003) Particle-based fluid simulation for interactive applications. In: Proceedings of 2003 ACM SIGGRAPH symposium on computer animation, pp 154–159

50. Magnaudet J, Eames I (2000) The motion of high-Reynolds number bubbles in inhomogeneous flow. Annu Rev Fluid Mech 32(1):659–708
51. Müller M, Solenthaler B, Keiser R, Gross M (2005) Particle-based fluid-fluid interaction. In: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on computer animation, pp 237–244
52. Mihalef V, Unlusu B, Metaxas D, Sussman M, Hussaini MY (2006) Physics based boiling simulation. In: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on computer animation, pp 317–324
53. Nielsen MB, Museth K (2006) Dynamic tubular grid: an efficient data structure and algorithms for high resolution level sets. J Sci Comput 26(3):261–299
54. Rasmussen N, Enright D, Nguyen D, Marino S, Sumner N, Geiger W, Hoon S, Fedkiw R (2004) Directable photorealistic liquids. In: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on computer animation, pp 193–202
55. Rene W, Gabriel Z (2011) Inner sphere trees and their application to collision detection. Springer
56. Redon S, Keddar A, Coquillart S (2000) An algebraic solution to the problem of collision detection for rigid polyhedral objects. In: Proceeding of IEEE conference on robotics and automation, vol 4
57. Redon S, Kheddar A, Coquillart S (2000) Fast continuous collision detection between rigid bodies. Comput Graph Forum 21(3):279–287
58. Robinson-Mosher A, Shinar T, Gretarsson J, Su J, Fedkiw R (2008) Two-way coupling of fluids to rigid and deformable solids and shells. ACM Trans Graph (TOG) 27(3):46
59. Sun H, Han J (2010) Particle-based realistic simulation of fluid/solid interaction. Comput Animat Virtual Worlds 21(6):589–595
60. Shin S-H, Kim C-H (2007) Target-driven liquid animation with interfacial discontinuities. Comput Animat Virtual Worlds 18(4–5):447–453
61. Selle A, Lentine M, Fedkiw R (2008) A mass spring model for hair simulation. ACM Trans Graph (TOG) 27(3):64
62. Sud A, Otaduy MA, Manocha D (2004) Difi: Fast 3d distance field computation using graphics hardware. Comput Graph Forum 23(3):557–566
63. Shew WL, Poncet S, Pinton JF (2006) Viscoelastic effects on the dynamics of a rising bubble. J Stat Mech: Theory Exp 2006(01):P01009
64. Sigg C, Peikert R, Gross M (2003) Signed distance transform using graphics hardware. In: IEEE visualization, pp 83–90
65. Selle A, Rasmussen N, Fedkiw R (2005) A vortex particle method for smoke water and explosions. ACM Trans Graph (TOG) 24(3):910–914
66. Schwarzer F, Saha M, Latombe J-C (2004) Exact collision checking of robot paths. Algorithmic foundations of robotics V, pp 25–42
67. Song O, Shin H, Ko H-S (2005) Stable but nondissipative water. ACM Trans Graph (TOG) 24(1):81–97
68. Stam J (1999) Stable fluids. In: Proceedings of the 26th annual conference on computer graphics and interactive techniques, pp 121–128
69. Shi L, Yu Y (2005) Taming liquids for rapidly changing targets. In: Proceedings of ACM SIGGRAPH/Eurographics symposium on computer animation, pp 229–236
70. Takahashi T, Fujii H, Kunimatsu A, Hiwada K, Saito T, Tanaka K, Ueki H (2003) Realistic animation of fluid with splash and foam. Comput Graph Forum 22(3):391–400
71. Teschner M, Heidelberger B, Mueller M, Pomeranets D, Gross M (2003) Optimized spatial hashing for collision detection of deformable objects. In: Proceedings of vision modeling visualization, pp 47–54
72. Thürey N, Keiser R, Pauly M, Rüde U (2009) Detail-preserving fluid control. Graph Models 71(6):221–228
73. Weller R, Zachmann G (2009) Inner sphere trees for proximity and penetration queries. Robotics: science and systems, vol 2. MIT Press, Cambridge

# Chapter 6
# Real-Time Visual Effects Programming

**Abstract** This chapter shows the essential part to implement real-time visual effects. An application such as a game should run in real time to interact with users. For smooth interaction, its rendering speed should be more than 30 FPS (33 ms per frame). Due to recent technological advances computation power becomes very high, but it is almost impossible to obtain real-time visual effects using CPU-based computation yet. To overcome this shortcoming, CUDA SDK from NVIDIA utilizes idle resources of the graphic card for computation and it leads to maximizing operational efficiency and enhancing computation power. This chapter presents an example of visual effects programming with CUDA SDK. The programming codes in this chapter help to implement and directly control a simple 3D fluid simulation. For stability of the simulation, the most up-to-date method, "Position-based Fluid" and the screen space rendering method are implemented. Both methods optimize the usage of GPUs and shaders.

## 6.1 GPU Programming: CUDA

Most high-quality visual effects are generated through physically based simulations that involve very time-consuming and complex processes. Recently, to overcome this disadvantage, GPUs have been used to reduce the computation time. There are a number of ways to use GPUs, such as DirectX, OpenCL, and Compute Unified Device Architecture (CUDA) programming. This chapter describes how to design and implement simulations using CUDA.

### *6.1.1 Introduction to CUDA*

CUDA provides parallel computing and general-purpose GPU techniques that allow programs to run on GPUs using C-like and industry-standard programming languages. The preparation for CUDA programming simple involves installing a GPU driver and the NVIDIA architecture. It is easy for beginners to set up a CUDA development environment and develop a CUDA program.

Additionally, because CUDA provides both low- and high-level APIs, there is no restriction on the design of CUDA programming. Compared with traditional general-purpose GPUs, the use of CUDA graphics APIs has the following advantages:

- Random access: Data can be read from any location in the GPU memory.
- Shared memory: CUDA can divide a task and then assign it to multiple threads in the high-speed shared memory area (16 or 48 KB). This memory can be used for user-management cash memory, and this technique provides faster computation than using a texture look-up table.
- The read/write process on the GPU is faster than on the CPU.
- Integer computation and bitwise operations are supported, including an integer-type texture look-up table.

Despite its many advantages, the following limitations must be understood before CUDA programs can be designed:

- CUDA does not support texture rendering.
- The recursive function is not supported.
- There is a bottleneck in bus bandwidth between the GPU and CPU, which may cause some time delay.
- Optimum performance is only achieved when at least 32 threads are running at the same time and the total number of threads is in the thousands. If 32 threads are running along the same path, branches in the CUDA program do not significantly impact performance.
- CUDA programs only run on NVIDIA graphics devices such as GeForce, Quadro, Terga, and Tesla.

### *6.1.2 Installation and Setup*

In early versions of CUDA, the installation and setup were very complicated. Since version 5.0, the CUDA software development kit (SDK) installer sets up the Microsoft Visual Studio environment directly.

First, we examine the steps involved in creating and setting up a CUDA project. A new project is created by selecting File ⇒ New ⇒ Project, as shown in Fig. 6.1.

If the CUDA SDK installation is successful, NVIDIA items will be created in the Project Templates folder (Fig. 6.2). Select a project template that has the same CUDA version as your own.

Projects created by the above steps will have a "kernel.cu" file. "*.cu" is a unique CUDA filename extension, and functions used in the GPU architecture are defined in such files. Their basic structure is similar to "*.c" or "*.cpp" files, because the host (CPU level) function may also be defined there. For header files, the "*.cuh" extension is used.

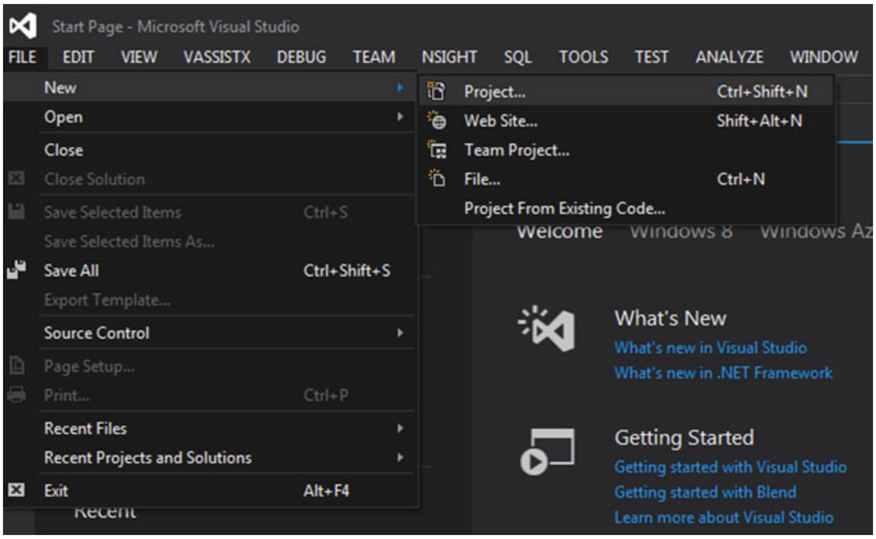
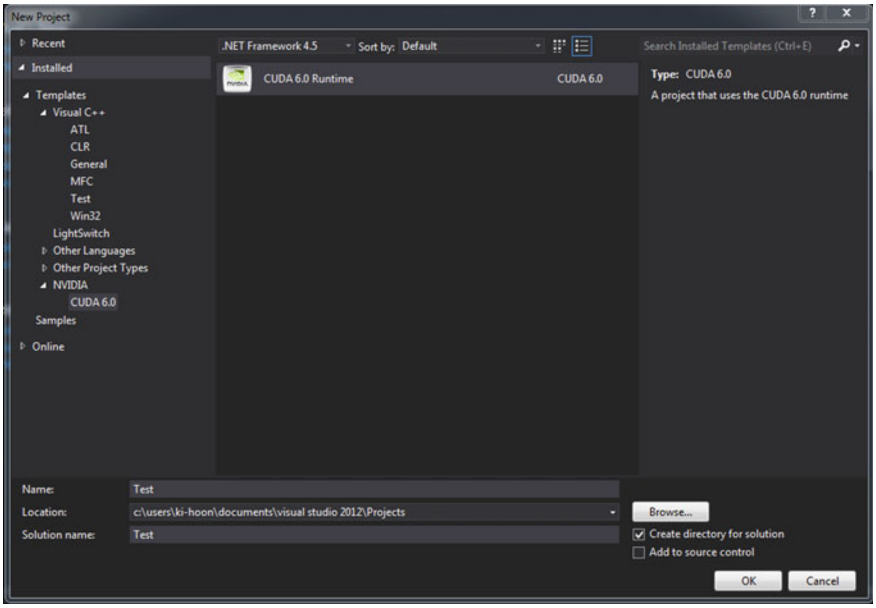

Start Page - Microsoft Visual Studio
FILE EDIT VIEW VASSISTX DEBUG TEAM NSIGHT SQL TOOLS TEST ANALYZE WINDOW
New
Open
Close
Close Solution
Save Selected Items Ctrl+S
Save Selected Items As...
Save All Ctrl+Shift+S
Export Template...
Source Control
Page Setup...
Print... Ctrl+P
Recent Files
Recent Projects and Solutions
Exit Alt+F4
Project... Ctrl+Shift+N
Web Site... Shift+Alt+N
Team Project...
File... Ctrl+N
Project From Existing Code...
Welcome Windows 8 Windows Az
What's New
What's new in Visual Studio
What's new in .NET Framework
Getting Started
Getting started with Visual Studio
Getting started with Blend
Learn more about Visual Studio
Recent

**Fig. 6.1** Creating a new project

If the CUDA SDK is installed correctly, the compiled project should look like that shown in Fig. 6.3. When examining files in the project, you may find unfamiliar



.NET Framework 4.5 Sort by: Default

CUDA 6.0 Runtime CUDA 6.0

Type: CUDA 6.0

A project that uses the CUDA 6.0 runtime

Name: Test

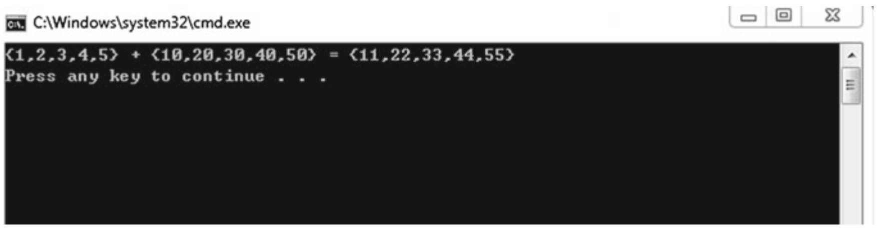
Location: c:\users\ki-hoon\documents\visual studio 2012\Projects

Solution name: Test

[✓] Create directory for solution

[ ] Add to source control

**Fig. 6.2** Project templates



```
{1,2,3,4,5} + {10,20,30,40,50} = {11,22,33,44,55}
Press any key to continue . . .
```

**Fig. 6.3** Example of a compile result

code (`__global__`) that is different from the well-known C language. Details of such code is addressed in Sect. 6.1.3.

### 6.1.3 *Structure of CUDA Functions*

CUDA has three types of function: Host, global, and device. These are classified according to where they are to be called and executed. Host functions are called and executed in the host (CPU), and correspond to general functions in C. Global functions are called in the host but executed in the device (GPU). The return type of global function must be void, and all parallel algorithms are programmed in this function. Device functions are called and executed in the device, and have no return type restriction. Recursive modules cannot be used in the global and device functions.

The prefixes `__host__`, `__global__`, and `__device__` are used to denote the definition of these functions. Note that `__host__` can be omitted and used at the same time as `__device__`. For example, `__global__ void addKernel (int *c, const int *a, const int *b)` is a global function and `int main()` is a host function.

## 6.2 Real-Time Fluid Programming

### 6.2.1 *Position-Based Fluid*

This section introduces an example fluid simulation using CUDA. Traditional SPH uses small time-steps to guarantee the stability of the simulation. Even when the generated simulation has a high frame rate, the animation is shown in slow motion. To solve this problem, various techniques have been proposed. A typical technique is PCI SPH [2], which improves the stability of a simulation using an iterative solver. However, a bottleneck may occur in the GPU program, because PCI SPH uses adaptive iteration. Therefore, this section introduces CUDA programming for a "Point-Based

Fluid" [1] instead of PCI SPH. This method optimizes the use of multiple threads by nonadaptive iteration, and is appropriate for stable real-time simulations.

### *6.2.2 Data Structure for Vectors*

For 3D simulations, some data structure is needed for the 3D vectors. However, we develop a data structure for 4D vectors, because the simulation and rendering processes are performed on a GPU, which is well suited to 4D vectors. We use the header file "`Vector_type.h`" provided by CUDA SDK for 4D vectors.

### *6.2.3 Hash Table*

In particle-based simulations, the most important issue is to search neighboring particles. Because it takes too much time to search the entire set of particles, we need only search adjacent particles. A hash table is used to determine the nearest neighbor. First, we divide the simulation space into a grid, and then compute the location of grid cells that contain particles using the particle positions in world coordinates. This leaves the problem of searching the adjacent grid cells. To solve this problem, we propose the "Building the Grid using Sorting" method, which rearranges particles according to the indices of their respective cells. The `thrust` function is used for the sorting operation, where `dGridParticleHash` is the index of a cell including a particle, `dGridParticleIndex` is the previous particle index in the array, and `numParticles` is the number of particles to be sorted.

```
thrust::sort_by_key(thrust::device_ptr<uint>(dGridParticleHash),
      thrust::device_ptr<uint>(dGridParticleHash + numParticles),
                 thrust::device_ptr<uint>(dGridParticleIndex));
```

The `sort_by_key` function returns the sorted list of `key(dGridParticle Hash)` values. Because only the indices have been sorted, a further step is required to sort the data according to the sorted indices.

```
int originalIndex = gridParticleIndex[index];
sortedData[index] = originalData[originalIndex];
```

Using a hash table, we want to sort the data as described above and identify a data structure that can access both the particle index and grid index during fluid simulations. To build such a data structure, we store the first and last particle indices in a grid as flags for the range of the grid index of sorted data.

```
    hash = gridParticleHash[index];
    sharedHash[threadIdx.x+1] = hash;
    if (index > 0 && threadIdx.x == 0) {
       sharedHash[0] = gridParticleHash[index-1];
    }
    if (index == 0 || hash != sharedHash[threadIdx.x]) {
       cellStart[hash] = index;
       if (index > 0)
          cellEnd[sharedHash[threadIdx.x]] = index;
       if (index == numParticles - 1)
          cellEnd[hash] = index + 1;
    }
```

The following code shows that hash keys are updated using the shared GPU memory. When a grid index is different from that in the neighboring memory, the difference is stored in the shared memory. Incorrect results will be produced if threads simultaneously access the memory. We use the `__syncthreads` function to synchronize threads and prevent such simultaneous access.

```
__global__ void reorderDataAndFindCellStartD(uint *cellStart,
            uint *cellEnd, float4 *sortedPos,
            float4 *sortedVel, uint *gridParticleHash,
            uint *gridParticleIndex, float4 *oldPos,
            float4 *oldVel, uint numParticles)
{
   extern __shared__ uint sharedHash[];
   uint index = __umul24(blockIdx.x, blockDim.x) + threadIdx.x;
   uint hash;
   if (index < numParticles) {
      hash = gridParticleHash[index];
      sharedHash[threadIdx.x+1] = hash;
      if (index > 0 && threadIdx.x == 0)
         sharedHash[0] = gridParticleHash[index-1];
   }
   __syncthreads();
   if (index < numParticles) {
      if (index == 0 || hash != sharedHash[threadIdx.x]) {
          cellStart[hash] = index;
         if (index > 0)
            cellEnd[sharedHash[threadIdx.x]] = index;
      }
      if (index == numParticles - 1)
         cellEnd[hash] = index + 1;
      uint originalIndex = gridParticleIndex[index];
```

```
                float4 pos = oldPos[originalIndex];
                float4 vel = oldVel[originalIndex];
                sortedPos[index] = pos;
                sortedVel[index] = vel;
            }
        }
```

Particle and grid indices can be mutually accessed using the `cellStart`, `cellEnd`, and `gridParticleHash` variables. Because the access is performed via the sorted data, neighboring memory can be accessed very quickly. If the final update uses old data, continuous simulation is possible without any change of indices.

### 6.2.4 Simulation Programming

We have completed the preparations necessary for a CUDA fluid simulation. This section describes the implementation of a particle-based simulation using CUDA. One advantage of particle-based simulations is that rapid computation can be performed by assigning one thread per particle. The basic design is as follows:

```
__global__ void globalFunction(float4 *oldPos, float4 *oldVel,
             float4 *sortedPos, float4 *sortedVel,
             uint *gridParticleIndex, uint *cellStart,
             uint *cellEnd, uint numParticles)
{
   uint index = __mul24(blockIdx.x, blockDim.x) + threadIdx.x;
   uint originalIndex = gridParticleIndex[index];
   if (index >= numParticles) return;
   float4 pos = sortedPos[index];
   float4 vel = sortedVel[index];
   float4 oldParticlePos = oldPos[originalIndex];
   int3 gridPos = calcGridPos(oldParticlePos);
   resultStructure output = make_result(0.0f);
   for (int z=-1; z<=1; z++) {
      for (int y=-1; y<=1; y++) {
         for (int x=-1; x<=1; x++) {
            int3 neighbourPos = gridPos + make_int3(x, y, z);
            output += gridCalculate(neighbourPos, index, pos,
               vel, sortedPos, sortedVel, cellStart, cellEnd);
         }
      }
   }
}
```

The `calGridPos` function returns a grid index as a 3D integer vector, which is computed from the particle position in world coordinates. Using nested `for` statements, the 3D index of adjacent cells is stored in the variable `neighbourPos`. Particles in neighboring cells are searched with the variable `neighbourPos`, and then the dynamics between particles are computed. The `gridCalculate` function uses data from the updated hash table as follows:

```
__device__ resultStructure gridCalculate(int3 gridPos, uint index,
                    float4 pos, float4 vel,
                    float4 *oldPos, float4 *oldVel,
                    uint *cellStart, uint *cellEnd)
{
   uint gridHash = calcGridHash(gridPos);
   uint startIndex = cellStart[gridHash];
   resultStructure output = make_result(0.0f);
   if (startIndex != 0xffffffff) {
      // iterate over particles in this cell
      uint endIndex = cellEnd[gridHash];
      for (uint j=startIndex; j<endIndex; j++) {
         float4 pos2 = oldPos[j];
         float4 vel2 = oldVel[j];
         output += neighborParticleCalculate(pos, pos2,
                 vel, vel2);
      }
   }
   return output;
}
```

To update the physical properties of a particle, it is first necessary to find a particle's grid index. Because the input argument `gridPos` is a 3D index, we must convert this into a 1D index. This conversion process is responsible for the `calcGridHash` function. With the converted 1D index, we access the hash table created in the previous step, and then obtain the value of `startIndex` from the `cellStart` array. If the value of `startIndex` is equal to `0xffffffff`, there is no particle in that cell, and the update process can be skipped. Otherwise, numbers from `startIndex` to `endIndex` become the indices of the particles in the grid. With these indices, we obtain data on neighboring particles and update the pressure and position. Finally, the updated results are returned.

Inner functions and variables can be changed as required, and properties such as the density and velocity can be updated via user-controllable variables. Note that "old data" should be updated, not "sorted data." Otherwise, unexpected results may occur because of the simultaneous access and update of data.

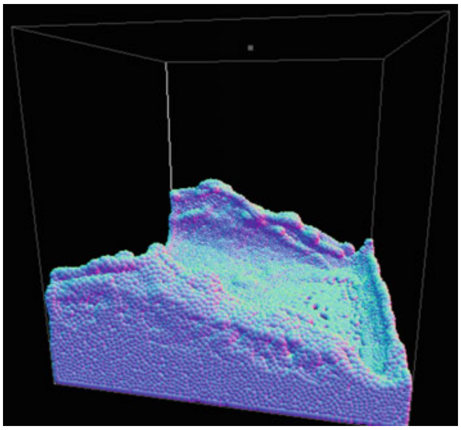


**Fig. 6.4** Rendering with a sphere model

### 6.2.5 Visualization Programming

This section introduces a visualization technique for the fluid simulations implemented in the previous section. Figure 6.4 shows the drawing of spheres for particle data.

In Fig. 6.4, the motion of each particle is similar to the movement of water, but it looks like a granular simulation rather than a water simulation. We can create a more plausible visualization through additional rendering tasks. Among the many ways of visualizing fluid simulations, we describe the screen space rendering method, which produces a more heuristic result using on-screen rendering information.

First, a normal map is needed. This is easily created using the fragment shader as follows:

```
vec3 N;
N.xy = gl_TexCoord[0].xy*vec2(2.0, -2.0) + vec2(-1.0, 1.0);
float mag = dot(N.xy, N.xy);
if (mag > 1.0) discard;
N.z = sqrt(1.0-mag);
N *= 0.5f;
N += vec3(0.5);
gl_FragColor = vec4(N, 1.0f);
```

Because a normal vector is stored as an RGB value in a normal map, $x$, $y$, and $z$ coordinates between $+1$ and $-1$ should be converted into color values of between 0 and 1.

After creating a normal map, we apply a smoothing step to the normal map and add lighting effects. The above processes produce a final result similar to that in Fig. 6.5.

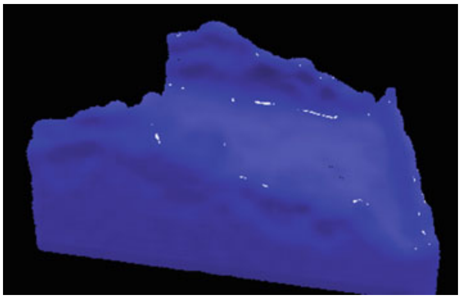


**Fig. 6.5** Visualization of fluid simulation

## References

1. Macklin M, Müller M (2013) Position based fluids. ACM Trans Graph 32(4):104
2. Solenthaler B, Pajarola R (2009) Predictive-corrective incompressible SPH. In: Proceedings of ACM SIGGRAPH 2009. ACM transactions on graphics (TOG), vol 28(3), p 40, August 2009

# Appendix A
# Data Structure for SDF

## A.1 Level-Set Method

In everyday life, we constantly interact with complex and beautiful surfaces and deformations. One of the best examples is our interaction with water, which is so natural to us that we hardly acknowledge the intricate ways in which it merges, forms, and breaks up. To a large degree, the ambition of CG is to simulate and reproduce the appearance and dynamic behavior of phenomena in the world. Simulations are particularly useful when the effect of a certain phenomenon cannot be easily reproduced by artists or animators.

For this task, CG unites several different scientific disciplines. For instance, to achieve computational techniques capable of reproducing the behavior of water, we must resolve theory and practice from the disciplines of mathematics and physics. Many ideas are also retooled from fields such as engineering. A level set is a mathematical construction that captures a dynamic implicit surface that possesses the properties required to represent complex surface deformations, such as those of water. The adjective ‘dynamic refers to the ability of the surface to change over time, and ‘implicit refers to how the surface is represented. The level-set method was introduced by Osher and Sethian [4] in 1988 as a method for tracking interfaces (i.e. surfaces) in computational physics. Since then, considerable efforts have been made to develop more accurate and robust numerical methods for solving the equations that govern the dynamic behavior of a level set. Level sets have also been used in the surface representation of several areas spanning multiple fields. In CG and computer vision, in particular, these areas include (but are certainly not limited to) fluid simulations of water and fire, collision detection in particle simulations, geometric modeling, shape metamorphosis, and the segmentation of volumetric datasets. Examples of the applications mentioned above can be found in Figs. A.1 and A.2. Further examples of level-set applications in image processing, computer vision, and computational physics can be found in the book by Osher and Paragios [3], to which we refer interested readers.

C.-H. Kim et al., *Real-Time Visual Effects for Game Programming*,
Gaming Media and Social Effects, DOI 10.1007/978-981-287-487-0

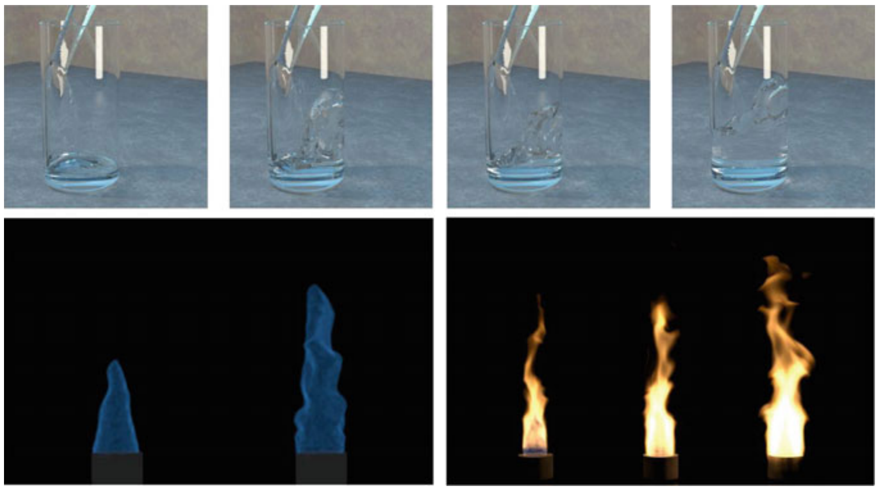


**Fig. A.1** *Top* Even a mundane act such as pouring a glass of water results in highly complex and detailed behavior. The water surface is represented by a level set. *Bottom* A physically based simulation of fire. The blue core of the flame is represented using a level set

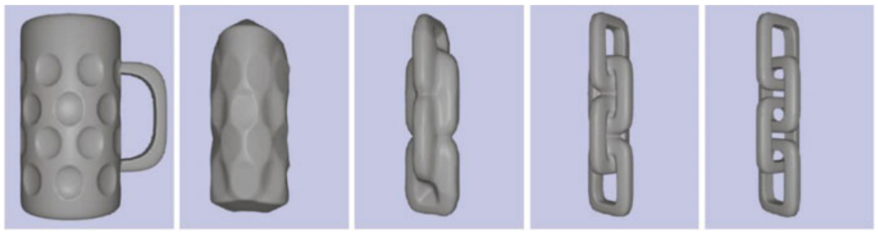


**Fig. A.2** An example of shape metamorphosis in which a beer mug morphs into a four-link chain. The level set gracefully handles topological changes during the metamorphosis

Level sets provide a number of unique advantages over other surface representations. A level set cannot self-intersect, i.e. the surface cannot cross over itself, which is a simple consequence of the definition of level sets. Furthermore, complex topological changes are handled automatically by the underlying mathematics. These properties are not shared by explicit representations, such as triangle meshes, which are currently the most widely used surface representation in CG. Finally, numerical schemes for the dynamics of level sets are relatively simple to employ. These unique features are all utilized in the applications mentioned above. For example, fluid surfaces such as water often undergo complex topological and geometrical changes as they merge and pinch off in elaborate ways. These changes are handled automatically by employing level sets as the underlying surface representation (see Fig. A.1). In shape metamorphosis, topological changes are handled gracefully when utilizing level sets (see Fig. A.2). Geometric modeling with traditional surface representations, such as triangle meshes, often results in self-intersecting geometry due to the inherent properties of the common workflows. This is clearly undesirable in final models that are to be used for physically based simulations or physical prototyping. Level sets can be used in the modeling process to avoid this problem. Various surface editing operators that are somewhat complex in triangle meshes can be easily applied with level sets. These surface editing operators can be used to repair digital models scanned from real-world geometry. Using physical models to create characters, for instance, is a very popular technique in many feature film productions, and therefore the ability to repair the scanned geometry is very useful.

The dynamic nature of level sets rests on a solid and rather advanced mathematical and numerical framework that can be elegantly generalized to any number of dimensions. However, as we are concerned with CG, we will mostly work in three dimensions. Therefore, we restrict ourselves to descriptions in three dimensions that are reduced to one or two dimensions when it serves the exposition. We first focus on describing the basics of the level-set method, and then look at extensions and improvements. In the next section, we briefly describe the ideas behind implicit surfaces and contrast them with explicit surface representations. We then discuss the theory behind the dynamics of the level-set method, and look closely at the equations governing the movement. Finally, we describe numerical schemes used to implement the level-set theory on a computer.

## A.2 Implicit Surface

Implicit surfaces and their properties are generally well understood in mathematical terms, and come in many distinct forms in CG, each associated with its own theory and unique properties. An explicit surface representation specifies the points on the surface. Expressed mathematically, an explicit surface representation provides a map between a parameter space and the points on the surface. In CG, there are several examples of explicit surfaces. They are typically sampled (i.e. discrete) representations in the sense that they specify a finite number of possible points along with information on their connectivity and how to interpolate the surface between them. Triangle meshes, point-based representations, NURBS(Non-Uniform Rational B-Spline), and subdivision surfaces are examples of such representations.

An implicit surface representation instead specifies a surface as the isocontour of a scalar function. Mathematically, given a scalar embedding function $\phi : \mathbf{R}^3 \rightarrow \mathbf{R}$, an implicit surface is represented as the pre-image $\phi^{-1}(k)$ of some scalar $k$. In other words, the surface consists of the set of points $\mathbf{x}$ for $\mathbf{R}^3$ at which $\phi(\mathbf{x}) = k$. Usually, and without loss of generality, we restrict our attention to the zero isocontour $k = 0$. Because a two-dimensional surface is defined by a three-dimensional embedding function, the implicit surface has co-dimension one.

Spheres and circles are easy to describe implicitly. A circle of radius $r$, for example, is given by the expression $\phi(x, y) = \sqrt{x^2 + y^2} - r = 0$. The left side of Fig. A.3 illustrates this example, while an explicit representation of the same circle is given

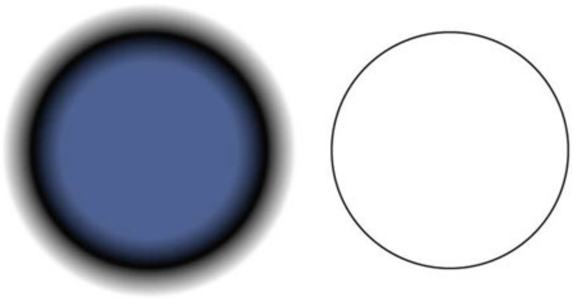


**Fig. A.3** *Left* Circle implicitly represented as the zero isocontour of the embedding function $\phi(x, y) = \sqrt{x^2 + y^2} - r$. The color has been clamped to get a noticeable color gradient near the interface. *Right* Circle explicitly represented as $(\cos\theta, \sin\theta)$ with $\theta \in [0, 2\pi]$

to the right. Note that an implicit surface representation does not directly specify the points on the surface. Instead, it queries whether or not a given point lies on the surface. This may make implicit representations seem inferior to explicit ones, and there are some applications within CG where this is true. However, very powerful tools are readily available using the implicit representation.

The level-set method only deals with closed implicit surfaces. This means that the surface must partition $\mathbf{R}^3$ into clearly defined interior and exterior regions, denoted as $\Omega^-$ and $\Omega^+$, respectively. We will assume that the implicit surface is given by the zero isocontour, and that the embedding function maps points in the interior region to negative values, with points in the exterior region assigned to positive values. This can be seen on the left of Fig. A.3, where the blue color is the interior region of negative values and the exterior region is shown in white.

At this point, we can identify two important advantages of level sets. First, for a given point in space, we can determine whether it is inside or outside the surface simply by evaluating the embedding function at that point and looking at the sign. For explicit representations, such as meshes, this kind of query is more complicated, and the result is ambiguous if the mesh contains holes or self-intersections. Second, a level set cannot contain self-intersections. This stems from the simple fact that an implicit surface is represented by a single-valued embedding function. Thus, any point in $\mathbf{R}^3$ cannot have both a negative and a positive sign at the same time.

Implicit surfaces are often not represented analytically, because no analytical expression is available or known for a given surface. Instead, the embedding function is sampled on a grid, as shown on the left in Fig. A.4. A sampling of an explicit representation is shown on the right. Sampling an implicit representation on a fixed grid is referred to as an Eulerian representation, because it captures the interface rather than tracking it. Sampling an explicit representation is often referred to as a Lagrangian representation. Note that grid points in the Eulerian representation remain fixed during any deformation. It is changes to the scalar values of the sampled embedding function that cause the surface to move. In a purely Lagrangian

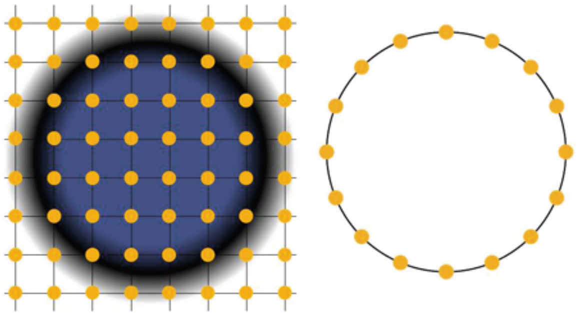


**Fig. A.4** *Left* Circle implicitly represented as the zero isocontour of the embedding function $\phi(x, y) = \sqrt{x^2 + y^2} - r$ sampled on a dense uniform grid. The rather coarse sampling serves to illustrate the principle of an Eulerian representation. *Right* Circle explicitly represented by points connected by line segments. Again, a rather coarse sampling has been used to illustrate the principle of a Lagrangian representation

representation, it is the sample points that move throughout the deformation. Various grids have been suggested, and the dense uniform grid of Fig. A.4 is one of the most common.

Placing the implicit surface in an embedding function gives access to a powerful differential toolbox. In particular, the surface normal pointing outwards can be computed directly from the normalized gradient

$$\mathbf{N} = \frac{\Delta\phi}{|\Delta\phi|}, \tag{A.1}$$

while the mean curvature in three dimensions is given by

$$k = \frac{1}{2}\nabla \cdot \mathbf{N}, \tag{A.2}$$

where $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)$ is a differential operator. Quantities such as the surface and volume integrals can be computed in a similar manner, as explained by Osher and Fedkiw [2]. From a theoretical viewpoint, the embedding function is immaterial as long as it is Lipschitz continuous. Even so, one particular class of functions has proved very useful in CG and level-set simulations. The SDF assigns the shortest distance to the surface to each point in $\mathbf{R}^3$. This distance is multiplied by $1/2$ for points in the interior region. The expression given previously for the implicit representation of a circle is, in fact, an SDF. Many operations and formulas can be simplified as a result of the properties of the SDF. Besides being able to immediately distinguish whether a point is in the interior or exterior region, we are now also given the shortest distance to the surface.

## A.3 Expressing Implicit Surfaces in the Sample Implementation

This section presents two algorithms that approximate the contour of an implicit surface with polygons, and describes a specific software implementation (Fig. A.5). This implementation (for both algorithms) is closely tied to how an implicit surface is actually described to the software. Therefore, it is important to first describe the classes that define an implicit surface. An implicit surface is defined by a derivative of the class Isosurface. This class is the interface for implementing the field function $d = f(x, y, z)$ discussed above, providing a pure virtual function Isosurface::fDensity() that computes the field function. To gain better performance, fDensity() can compute any number of points, starting at a point specified by three parameters $(x_0, y_0, z_0)$ and progressing towards the $+z$ axis according to a delta value specified by the parameter $dz$. As an example, an implementation of the field function of a sphere using Isosurface proceeds as follows:

```
class SphereIsosurface : public Isosurface
{
public:
    SphereIsosurface(float rad) : _radius(rad)
    {
        Vector center;  // (0,0,0)
        Vector v1 = center - _radius;
        Vector v2 = center + _radius;
        Isosurface::addBoundingBox(BoundingBox(v1, v2));
    }

    void SphereIsosurface::fDensity(
        float x0, float y0, float z0,
        float dz, int num_points, float *densities)
    {
        for (int i = 0; i < num_points; ++i) {
            float sqr_dist = x0 * x0 + y0 * y0 + z0 * z0;
            float sqr_rad = _radius * _radius;
            float d = sqr_dist - sqr_rad;
            densities[i] = d;

            z0 += dz;
        }
    }

protected:
    float _radius;
};
```

The constructor of SphereIsosurface computes an axis-aligned bounding box containing the sphere and reports it using Isosurface::addBoundingBox(). It is not easy to determine the bounding box just by looking at the field function. A few algorithms have been proposed to address this problem, but they are not perfect: for example, they may not work correctly when the field function describes volumes that are not connected. In many cases, Isosurface is derived to implement solid primitives, such as a sphere, box, or torus, whose bounding box can be easily computed. Therefore, it is reasonable to expect the implementation of an Isosurface to report its own bounding box.

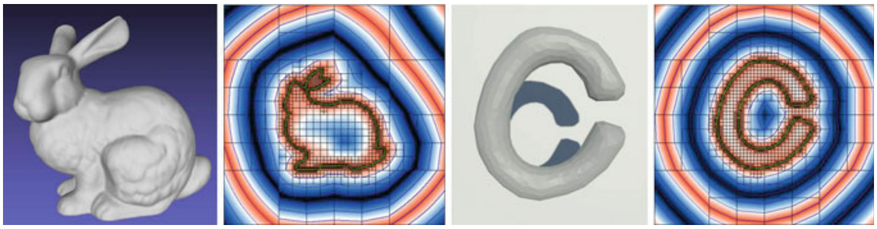


**Fig. A.5** A result of SDF using ADF [1]

The implementation of Isosurface also permits the transformation of the volume. The Matrix class implements a $4 \times 4$ transformation matrix that can represent any combination of translation, rotation, scaling, and shearing operations at a point (or an object made up of a collection of points) in 3D space. The Matrix class is also compatible with the way OpenGL deals with matrices. The Transform class, which inherits Matrix without adding any new data members, contains a function Transform::glMultMatrix() that is simply a wrapper around the OpenGL function glMultMatrixf().

A transformation matrix can be applied to the implicit surface using Isosurface::setTransform(). Consider a sphere of radius 1 that has been translated on the $x$-axis and is now centered at $(-5, 0, 0)$. Evaluating the field function for the point at $(0, 0, 0)$ should return a result indicating that the point is outside the volume. In this case, SphereIsosurface::fDensity(), as presented above, will evaluate the result incorrectly, as it does not take the translation into account. However, the definition can be easily revised to account for any kind of transformation:

```
void SphereIsosurface::fDensity(
    float x0, float y0, float z0,
    float dz, int num_points, float *densities)
{
    for (int i = 0; i < num_points; ++i) {
        float xt, yt, zt;
        inverted_matrix.transform(x0, y0, z0, &xt, &yt, &zt);

        float sqr_dist = xt * xt + yt * yt + zt * zt;
        float sqr_rad = _radius * _radius;
        float d = sqr_dist - sqr_rad;
        densities[i] = d;

        z0 += dz;
    }
}
```

The revised version first transforms the specified point given by $(x_0, y_0, z_0)$ according to the inverted matrix of the transformation applied to the implicit surface. When no transformation has been applied, the transformation matrix is the identity matrix (and so is its inverse), resulting in $xt = x_0$, $yt = y_0$, $zt = z_0$. To revisit the last example, where a sphere is translated by $(5, 0, 0)$, the inverted matrix would translate the point by $(+5, 0, 0)$, resulting in $xt = (x_0 + 5)$, $yt = y_0$, $zt = z_0$.

Thus, evaluating the field function at (0, 0, 0) would correctly return a result indicating that the point is outside the volume, whereas evaluating the point at $(-5, 0, 0)$ would correctly indicate that the point is inside the volume.

## References

1. Frisken SF, Perry RN, Rockwood AP, Jones TR (2000) Adaptively sampled distance fields: a general representation of shape for computer graphics. In: Proceedings of SIGGRAPH 2000, pp 249–254
2. Osher SJ, Fedkiw RP (2002) Level Set Methods and Dynamic Implicit Surfaces, Springer, Oct 2002. ISBN 0387954821
3. Osher S, Paragios N (2003) Geometric level set methods in imaging vision and graphics, Springer. ISBN 0387954880
4. Osher S, Sethian JA (1988) Fronts propagating with curvature-dependent speed: algorithms based on Hamilton–Jacobi formulations. J Comput Phys 79:12–49

# Appendix B
# Surface Tracking

## B.1 Marching Cubes

The Marching cubes algorithm generates a 3D polygonal mesh from a 3D scalar field. (In the 2D case, there is a Marching squares algorithm to generate a 2D polygon from 2D data.) A scalar field represents not only density values in a grid structure, but also Boolean values in simpler models. A 3D polygonal mesh is the surface of the 3D scalar field we wish to track.

For example, medical data can be converted to a 3D scalar field representing the density measurement of an organ, and this can then be represented by a 3D mesh using Marching cubes. This mesh refers to the surface of an organ. In fluid simulations, the density of the fluid is recorded at every time step, and a 3D polygonal surface is then generated using the Marching cubes algorithm. This mesh describes the surface of the fluid.

The Marching cubes algorithm is used in many areas, but this chapter describes its application in fluid simulations.

### *B.1.1 Basics*

To use the Marching cubes algorithm in fluid simulations, the simulation space must first be approximated to a grid. Every grid point contains information, such as a Boolean to denote whether it is inside or outside the fluid and scalar values describing the density of the fluid at that position.

The simplest form of the Marching cubes algorithm uses Boolean values stored at each grid point. In the case of a particle-based simulation, if the simpler model is used, it is easy to implement the Marching cubes algorithm: we simply refer to a hash table and generate a 3D mesh without any complicated computations. In the case of grid-based simulations, the same grid resolution is used for both the fluid simulation and Marching cubes algorithm. A 3D polygonal surface is generated with Boolean values of 'true to denote that the density in a grid cell is positive (which

C.-H. Kim et al., *Real-Time Visual Effects for Game Programming*,
Gaming Media and Social Effects, DOI 10.1007/978-981-287-487-0

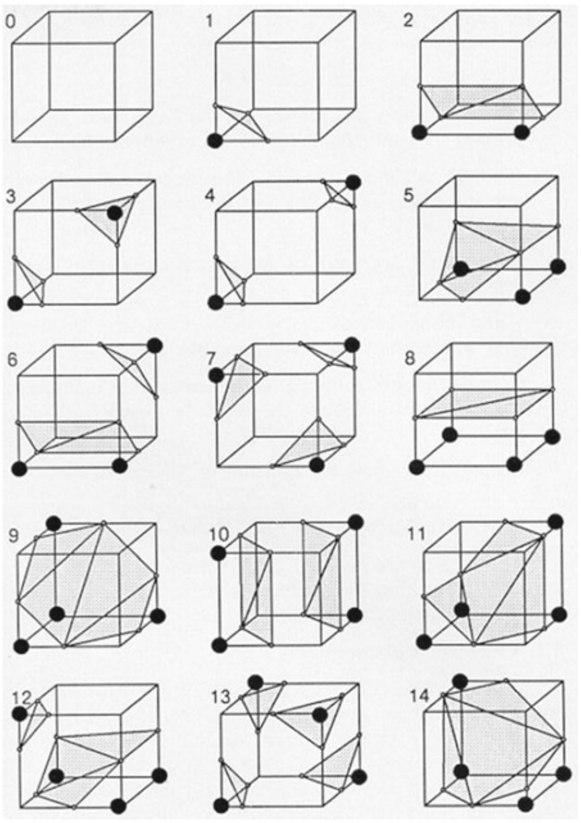


**Fig. B.1** Fifteen cube configurations [1]

means that particles exist in that cell). However, in both cases, it is noticeable that aliasing occurs in the generated 3D mesh, and its quality is not particularly good.

To obtain higher-quality results, we use the fluid density to generate a 3D polygonal mesh. It is possible to produce a more natural and smooth surface when representing large volumes in a high-density grid cell and small volumes in a low-density grid cell. This method will be described in detail in the next section.

The Marching cubes algorithm proceeds one grid cell at a time, and each grid cell has eight grid points. A total of 256 ($2^8$) possible configurations within a cell are precalculated, and these are finally reduced to 15 configurations by removing cases of rotation and symmetry.

In Fig. B.1, the big black dots are inside the fluid and small white points are vertexes of the generated 3D polygonal mesh. It is possible to obtain a more natural mesh by adjusting the position of these white points, as described in more detail in Sect. B.1.3.

The 256 cases are implemented with a table in the header file, and every grid cell references this table. One or more triangles are generated in each grid cell, and these triangles give the surface of the fluid being simulated.

### *B.1.2 Density*

In fluid simulations, the Marching cubes algorithm uses the density field of the fluid to generate a more natural surface.

- **Grid-based (Eulerian):** If the grid resolution is the same for both the simulation and the Marching cubes algorithm, the density is used as it is. If the resolutions are different, the density is interpolated prior to being used.
- **Particle-based (Lagrangian):** Because the density is stored in a particle, it is necessary to recalculate and store the density in grid points to implement Marching cubes. Generally, there are two methods. First, we can add the particle density to the grid point density of a cell containing the particle. The other method places a kernel on a particle and adds its density to that of the grid points within the kernel with a weight that is inversely proportional to the distance between the particle and the grid point. In both methods, the density of every grid point is initialized to 0 and then calculated.

The density stored using the above method is interpolated to produce a more natural polygonal mesh for the fluid.

### *B.1.3 Interpolation Method*

An interpolation technique must be introduced to generate a smooth curved surface for the fluid. This technique can be applied to grid cells that contain both fluid and air. That is, the interpolation function determines where to put the vertexes of an isosurface on edges that have one endpoint inside the fluid and the other endpoint outside the fluid. The threshold is the criterion of setting the vertex position. It is very important to find an appropriate threshold value, because the appearance of a polygonal mesh representing a fluid depends on this threshold value. Figure B.2 shows the position of a vertex according to the threshold value.

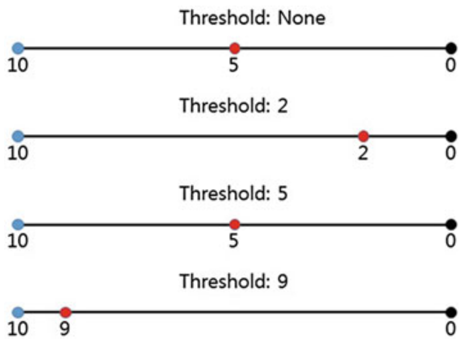


**Fig. B.2** The interpolated position of a vertex

A smoother curved surface is generated when interpolation is applied with a suitable threshold value than when interpolation is not used.

## Reference

1. Lorensen WE, Cline HE (1987) Marching cubes: a high resolution 3d surface construction algorithm. ACM Comput Graph 21(4):163–169

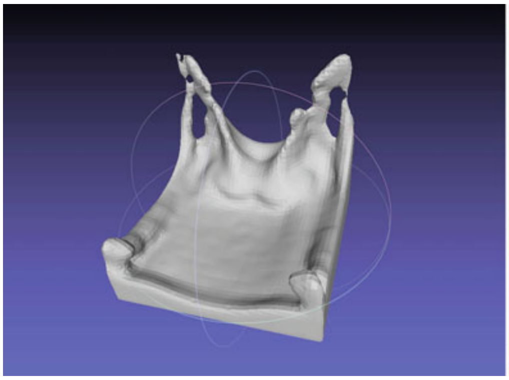
# Appendix C
# Rendering

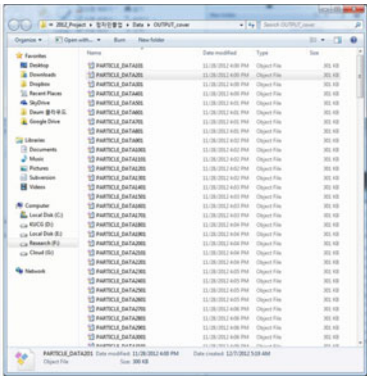
## C.1 3ds Max

The 3ds Max software provides an integrated rendering production system, including two global illumination algorithms called "ray tracing" and "radiosity". 3ds Max allows specular, transparent, and special lighting effects to be added using the radiosity, scanline, or ray tracing renderer at a specific time. Therefore, it is possible to correct inaccurate ambient light and produce a more realistic rendering result. Because 3ds Max integrates techniques from fast and interactive lighting up to realistic global rendering, such as radiosity and ray tracing, it can be used to visualize the full range of phenomena.

This chapter looks at the high-quality rendering of fluid simulation data using 3ds Max.

1. A polygonal model generated using the Marching cubes algorithm is exported to an OBJ file format, and the sequence of polygonal meshes for all frames in the fluid simulation is successively exported and stored.





2012_Project ▸ [illegible] ▸ Data ▸ OUTPUT_cover

Search OUTPUT_cover

Organize ▾ Open with... ▾ Burn New folder

Favorites: Desktop, Downloads, Dropbox, Recent Places, SkyDrive, Daum 클라우드, Google Drive

Libraries: Documents, Music, Pictures, Subversion, Videos

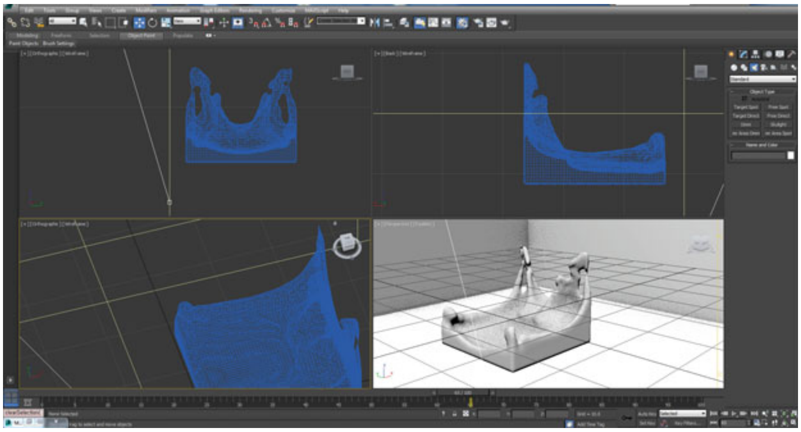
Computer: Local Disk (C:), KUCG (D:), Local Disk (E:), Research (F:), Cloud (G:)

Network

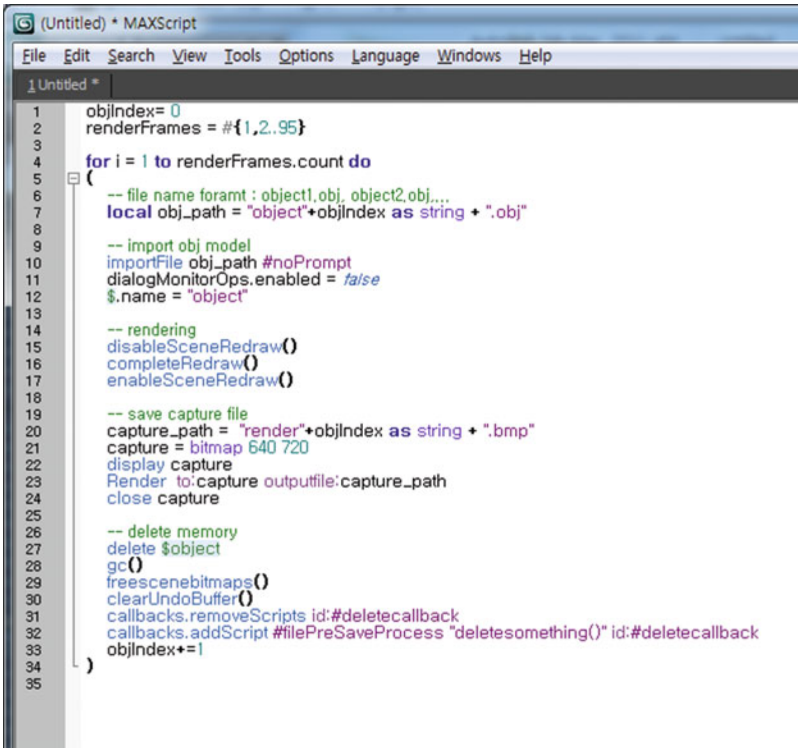
| Name | Date modified | Type | Size |
|---|---|---|---|
| PARTICLE_DATA101 | 11/28/2012 4:00 PM | Object File | 301 KB |
| PARTICLE_DATA201 | 11/28/2012 4:00 PM | Object File | 301 KB |
| PARTICLE_DATA301 | 11/28/2012 4:00 PM | Object File | 301 KB |
| PARTICLE_DATA401 | 11/28/2012 4:00 PM | Object File | 301 KB |
| PARTICLE_DATA501 | 11/28/2012 4:01 PM | Object File | 301 KB |
| PARTICLE_DATA601 | 11/28/2012 4:01 PM | Object File | 301 KB |
| PARTICLE_DATA701 | 11/28/2012 4:01 PM | Object File | 301 KB |
| PARTICLE_DATA801 | 11/28/2012 4:01 PM | Object File | 301 KB |
| PARTICLE_DATA901 | 11/28/2012 4:02 PM | Object File | 301 KB |
| PARTICLE_DATA1001 | 11/28/2012 4:02 PM | Object File | 301 KB |
| PARTICLE_DATA1101 | 11/28/2012 4:02 PM | Object File | 301 KB |
| PARTICLE_DATA1201 | 11/28/2012 4:02 PM | Object File | 301 KB |
| PARTICLE_DATA1301 | 11/28/2012 4:02 PM | Object File | 301 KB |
| PARTICLE_DATA1401 | 11/28/2012 4:03 PM | Object File | 301 KB |
| PARTICLE_DATA1501 | 11/28/2012 4:03 PM | Object File | 301 KB |
| PARTICLE_DATA1601 | 11/28/2012 4:03 PM | Object File | 301 KB |
| PARTICLE_DATA1701 | 11/28/2012 4:03 PM | Object File | 301 KB |
| PARTICLE_DATA1801 | 11/28/2012 4:04 PM | Object File | 301 KB |
| PARTICLE_DATA1901 | 11/28/2012 4:04 PM | Object File | 301 KB |
| PARTICLE_DATA2001 | 11/28/2012 4:04 PM | Object File | 301 KB |
| PARTICLE_DATA2101 | 11/28/2012 4:04 PM | Object File | 301 KB |
| PARTICLE_DATA2201 | 11/28/2012 4:04 PM | Object File | 301 KB |
| PARTICLE_DATA2301 | 11/28/2012 4:05 PM | Object File | 301 KB |
| PARTICLE_DATA2401 | 11/28/2012 4:05 PM | Object File | 301 KB |
| PARTICLE_DATA2501 | 11/28/2012 4:05 PM | Object File | 301 KB |
| PARTICLE_DATA2601 | 11/28/2012 4:05 PM | Object File | 301 KB |
| PARTICLE_DATA2701 | 11/28/2012 4:06 PM | Object File | 301 KB |
| PARTICLE_DATA2801 | 11/28/2012 4:06 PM | Object File | 301 KB |
| PARTICLE_DATA2901 | 11/28/2012 4:06 PM | Object File | 301 KB |
| PARTICLE_DATA3001 | 11/28/2012 4:06 PM | Object File | 301 KB |

PARTICLE_DATA201 Date modified: 11/28/2012 4:00 PM Date created: 12/7/2012 5:10 AM
Object File Size: 300 KB

2. An OBJ file is imported into 3ds Max; all OBJ files can be imported sequentially using a script.

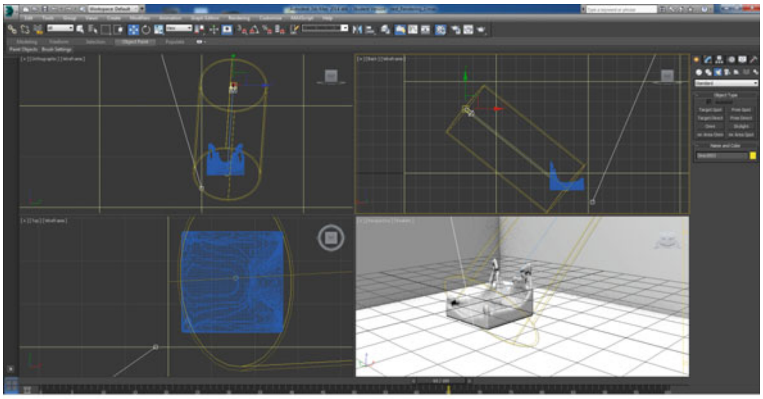


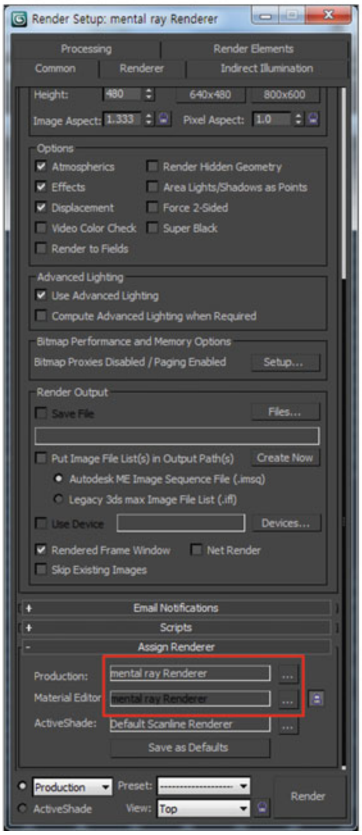
The following shows a 3ds Max script to import a series of OBJ files, render them, and store the rendered scenes in BMP files.



```
objIndex= 0
renderFrames = #{1,2..95}

for i = 1 to renderFrames.count do
(
    -- file name foramt : object1.obj, object2.obj,...
    local obj_path = "object"+objIndex as string + ".obj"

    -- import obj model
    importFile obj_path #noPrompt
    dialogMonitorOps.enabled = false
    $.name = "object"

    -- rendering
    disableSceneRedraw()
    completeRedraw()
    enableSceneRedraw()

    -- save capture file
    capture_path =  "render"+objIndex as string + ".bmp"
    capture = bitmap 640 720
    display capture
    Render  to:capture outputfile:capture_path
    close capture

    -- delete memory
    delete $object
    gc()
    freescenebitmaps()
    clearUndoBuffer()
    callbacks.removeScripts id:#deletecallback
    callbacks.addScript #filePreSaveProcess "deletesomething()" id:#deletecallback
    objIndex+=1
)
```

3. Scene configuration in 3ds Max is set for fluid rendering. Light and camera settings should be set, and a primitive can be added to the scene using 3ds Max modeling tools. In this process, it is possible to apply the ray tracing or radiosity rendering techniques.





Render Setup: mental ray Renderer

Processing | Render Elements

Common | Renderer | Indirect Illumination

Height: 480 | 640x480 | 800x600

Image Aspect: 1.333 | Pixel Aspect: 1.0

Options
- [✓] Atmospherics
- [ ] Render Hidden Geometry
- [✓] Effects
- [ ] Area Lights/Shadows as Points
- [✓] Displacement
- [ ] Force 2-Sided
- [ ] Video Color Check
- [ ] Super Black
- [ ] Render to Fields

Advanced Lighting
- [✓] Use Advanced Lighting
- [ ] Compute Advanced Lighting when Required

Bitmap Performance and Memory Options

Bitmap Proxies Disabled / Paging Enabled | Setup...

Render Output
- [ ] Save File | Files...
- [ ] Put Image File List(s) in Output Path(s) | Create Now
- ● Autodesk ME Image Sequence File (.imsq)
- ○ Legacy 3ds max Image File List (.ifl)
- [ ] Use Device | Devices...
- [✓] Rendered Frame Window
- [ ] Net Render
- [ ] Skip Existing Images

+ Email Notifications

+ Scripts

- Assign Renderer

Production: mental ray Renderer

Material Editor: mental ray Renderer

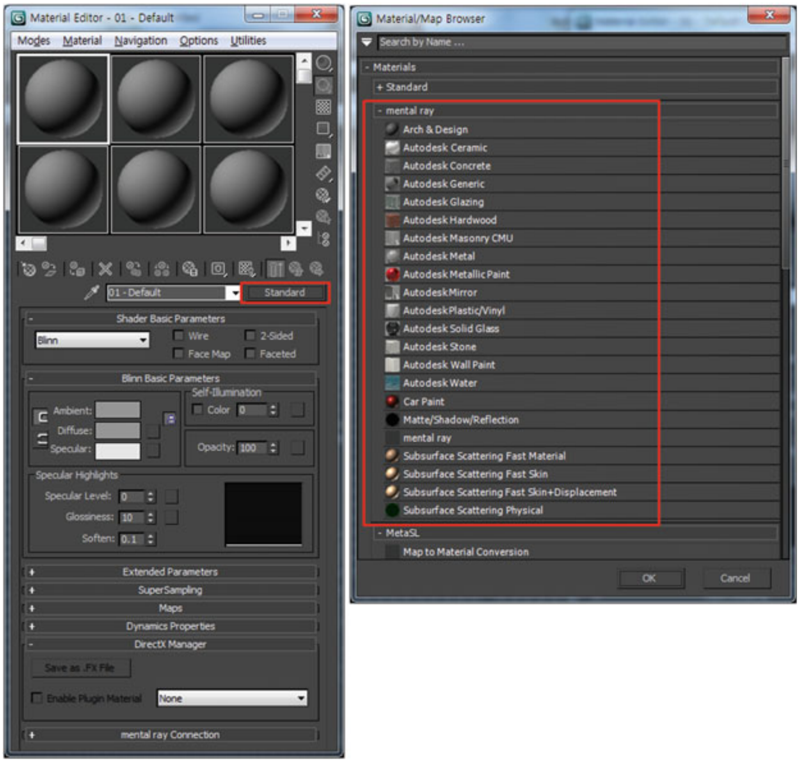
ActiveShade: Default Scanline Renderer

Save as Defaults

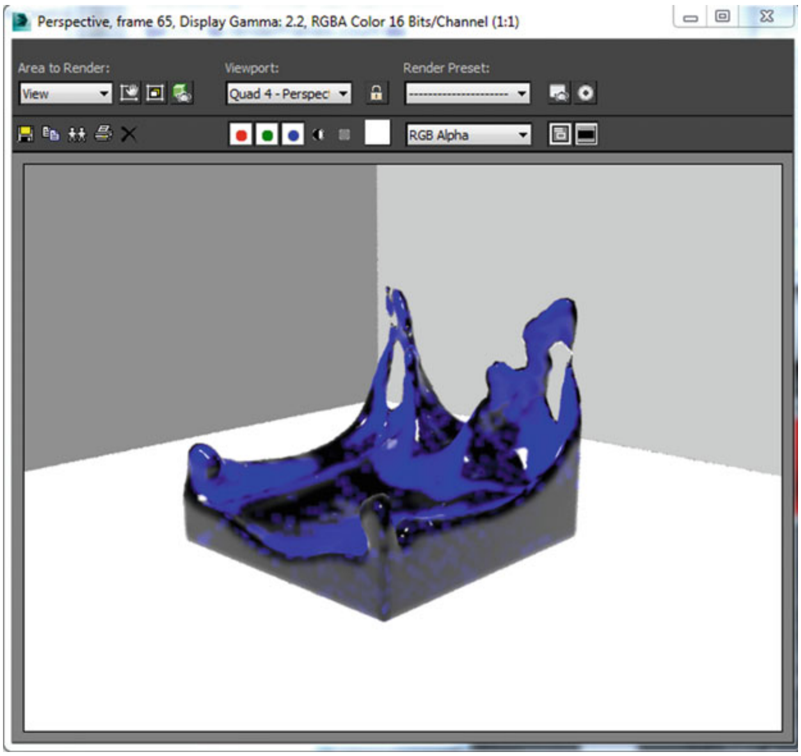
● Production | Preset: ------------------ | Render

○ ActiveShade | View: Top

The above screenshot shows a dialog box for rendering setup in 3ds Max. To render a fluid, users assign the ‘Production’ and ‘Material Editor’ types in the ‘Assign Renderer’ box by changing the default scanline renderer to the mental ray renderer. The following shows the material editor, where one of various mental ray materials can be chosen.



4. When the scene has been configured, 3ds Max performs the rendering process. The rendering result can be saved as an image file, and a script allows the generation of consecutive images for all simulation frames. Finally, a moving fluid simulation is completed by sequentially connecting all images.


Perspective, frame 65, Display Gamma: 2.2, RGBA Color 16 Bits/Channel (1:1)
Area to Render:
View
Viewport:
Quad 4 - Perspec
Render Preset:
RGB Alpha