Master Thesis

Application of the Semi-Lagrangian Method for Visualization of Near-body Hydrodynamics and Enhancement of Amorphous Effects

by

Truong Duc Thang

HT071012A

Supervised by

Dr. Anthony Fang Chee Hung

Dr. Golam Ashraf

Submitted to

Department of Computer Science

School of Computing

National University of Singapore

November 2011

Abstract

Efficient numerical techniques developed in the field of computer graphics are able to simulate compellingly realistic simulations of interactions between solids and fluids. In this work, we apply one of these techniques, the Semi-Lagrangian Stable Fluids method coupled with a new model of interaction between fluid and general semi-rigid 3D meshes, to biomechanical hydrodynamics visualization. The near-body surface dynamics provide meaningful information for rendering visuals that are intuitive to the streamlined flow characteristics surrounding the body. Our results show the visualization of active and passive resistive forces on the body in a video-based capture of an immersed dolphin kick. We also applied the resultant fluid field in a novel application where procedural fire trails of a muscle-shape body was directed by its near-body fluid dynamics. The techniques we employed are less used for engineering applications due to errors which are inherent outcomes of systemic approximations in its formulation. The errors, manifesting as excessive damping, are expected to change the nature of steady-state fields and nullify the typical engineering static flow analysis. On the other hand, near-body solid/fluids interaction are affected to a lesser degree since errors originating from dissipation are more severe when accumulated over time. Although it is generally not possible to numerically validate unsteady, high-velocity vector fields, our investigations show that near-body solid/fluid dynamics converges with respect to parametric refinements. Though far from a correctness proof, the numerical convergence of near-body quantities suggests the applicability of the method to certain classes of analysis and visualization where near-body characteristics are of greater concern.

Acknowledgements

I would like to thank Dr Golam Ashraf for his valuable guidance and support for completing the thesis as well as his new ideas on how to improve upon my existing work. Being accepted by him as one of his students at the most difficult time of my research was a great encouragement for me.

I would also like to thank Dr Anthony Fang for giving me the inspiration to start up with this idea. His guidance, although shortly, had led to the foundation of this work.

This thesis would also not be possible without the help of friends and family who have always given me valuable feedbacks along the way and I would like to take this opportunity to thank them.

Contents

Abstract **ii**

Acknowledgements **iii**

List of Figures **vii**

1 Introduction **1**

1.1 Numerical dissipation problem 4

1.2 Thesis objective and contribution 5

1.3 Results summary 6

1.4 Thesis organization 7

2 Background **8**

2.1 Fluid simulation and animation for computer graphics 8

2.2 Fluid and object interaction 10

2.3 Fluid simulation and application in engineering and biomechanics . 12

2.4 Navier-Stoke equations 13

3 Methodology and Implementation 17

3.1 Components and Approach 17

3.2 3D Model voxelization 18

3.2.1 Angle weighted pseudo-normal 22

3.3 Fluid simulator 23

3.3.1 Breaking down the Navier-Stoke equations 23

3.3.2 Grid structure for fluid simulation 29

3.3.3 Implementing the add force step 30

3.3.4 Implement the advection step 31

3.3.5 Implement the diffuse step 32

3.3.6 Implement the project step 36

3.3.7 Enforce the fluid volume boundary 37

3.3.8 Incorporate the object movement 37

4 Results 43

4.1 Visualization of near-body hydro-dynamics 43

4.1.1 Data acquisition 44

4.1.2 Near body hydrodynamics visualization results 46

4.1.3 Tuning of simulation parameters 50

4.2 Application on fire enhancement 51

4.2.1 Real time voxelization 54

4.2.2 Parameters passing to GPU 55

4.3 Fluid simulation and general semi-rigid object integrations 57

5 Conclusions and Discussions **60**

5.1 Publication 61

Bibliography **62**

List of Figures

1.1 Near-body characteristic visualization using flow fields obtained from a direct, Semi-Lagrangian simulation. The image shows instantaneous pressure field on the upper-body surface during a dolphin kick motion. The pressure values are normalized and color-coded using the HSV color bar shown. The red tone indicates high pressure and the blue tone indicates tangential surface pressure flow. 2

2.1 2D simulation results (upper) and 3D simulation result (lower) of Jos Stam's method. Pictures from Stam (2003) 10

3.1 Algorithm summary 18

3.2 Velocity of a boundary voxel is approximated as the velocity of its closest point on the mesh, which is interpolated from the velocities of the vertices of the containing triangle. 21

3.3 The normal at the vertex is changed if the triangle is divided into multiple smaller triangles. The averaged normal is biased toward the direction of the smaller triangles' normals. 22

3.4 The angle weighted normal. 23

3.5 The sequence of steps for the simulator in each time steps. 25

3.6 The new velocity at $\vec{x}$ is interpolated from the old velocity surrounding $\vec{x}_S$. The points surrounding the position $\vec{x}_S$ are the centres of the surrounding cells. 27

3.7 The grid structure for the fluid simulator and the voxelization of object models. 30

4.1 The underwater movements of the human swimmer are recorded using a pair of conventional cameras in water-tight enclosures and firmly anchored to the wall of the swimming pool. (LEFT) Camera calibrations performed using a customized static frame which provides eight markers whose pairwise distances are know. The coordinates of the calibration markers are used to derive the intrinsic and extrinsic parameters of the cameras. (RIGHT) The swimmer is outfitted with a Lycra-made,full-body suit with customized fabric markers. The markers partially assist the tracking process, but due to noise and image degrading factors, tracking of markers is presently a semi-automatic, manually-guided process. 45

4.2 Surface pressure variations on the frontal shank region in a downward phase of the dolphin kick cycle. The proximal-to-distal pressure variations are depicted as continuous color tone variations on the shaded surface. 47

4.3 Time-lapse images showing one complete dolphin-kick cycle(1/15 seconds per frame; viewed left to right, top to bottom), illustrating near-body path-lines that traces the characteristic flow fields surrounding a moving boundary surface. Longer paths indicate faster motion at the associated surface point. The path tracing technique was adapted from the particle back tracing method used in the advection algorithm. 48

4.4 Rapid dispersion of suspended particles in the dolphin kick motion. Viewing order: top to bottom,, left to right. 49

4.5 Tuning grid resolution based on the convergence of total pressure of 100 cells. 51

4.6 Volume filling (bottom) and path tracing (top) fire effects. Image from (Shijun, 2010) 52

4.7 The fire trail created and animated not using the fluid velocity field (TOP) and using the fluid velocity field (BOTTOM). Notice how the fire trails in the are different. In the bottom of the figure, the trail's particles are more coherent and move together. In the top part, the fire trail's particles stay separately and does not move. This can be observed more clearly in the accompanying videos. . . . 53

4.8 The creation of the muscle shape. Figure from (Keng Siang and Ashraf, 2007) 54

4.9 Calculation of distance using discretized action curve and its problem. 55

4.10 Two overlapped discretization. The second discretization is shown in thick lines connecting the red dots. 56

4.11 A virtual movement of the cone inside the fluid volume 57

4.12 A virtual snake move inside the fluid volume 58

4.13 A comparison between simulation (TOP) and real life example (BOTTOM) of a fluid flow passing by a black cube. 59

Chapter 1

Introduction

Simulation of fluid and its interaction with submerged objects are well known to be a challenging problem in the field of computer graphics and engineering. The difficulties come from the intricate underlying physics which requires huge amount of computational resource and time. However, fluid simulation is still required in many applications in visualization, special effects, games, and engineering such as car building. It is even found in sport where there is a need for analysis and understanding fluid flows in swimming.

Given the demand, there has been much research done in the field of fluid simulation. Through out decades of development, there are two major schools of techniques were created for fluid simulation. The more classical ones were developed in the field Computational Fluid Dynamics (CFD) for engineering and the more recent techniques were developed for CFD usages in computer graphics. In order to explain the purpose and motivation of this thesis, it is useful to briefly explain the difference between these two schools of fluid simulation techniques and their usages.

Computational Fluid Dynamics (Ferziger and Peric, 1999) is a well established field for analysing various flow problems. Among its many branches, the analytical and empirical study of the coupling between erratic, free-moving, and unstructured deformable bodies within an incompressible fluid medium, which is the focus of

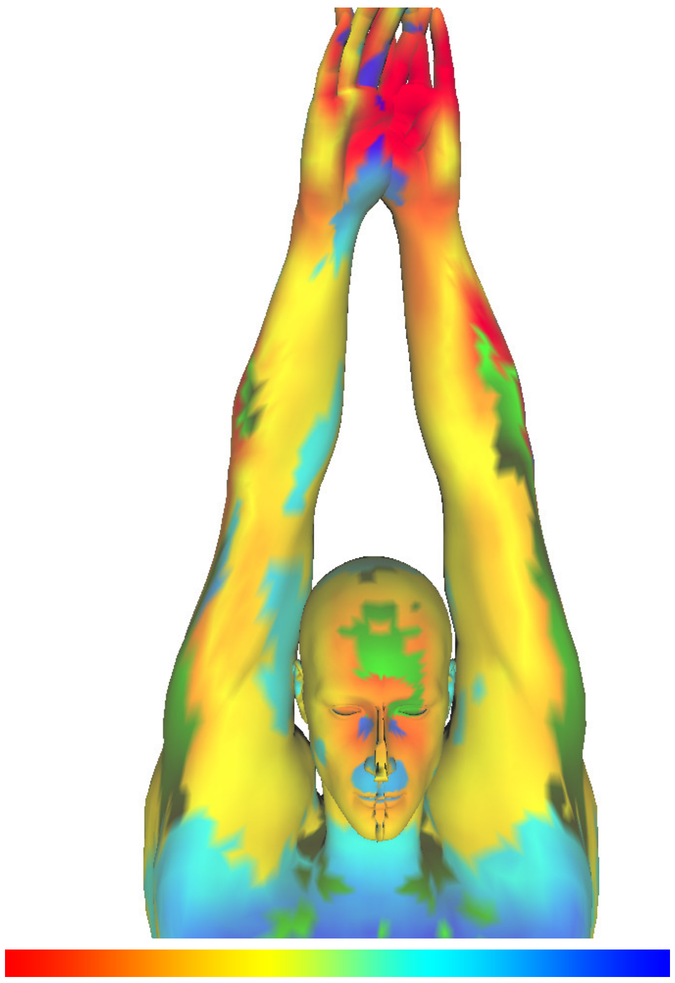

Figure 1.1: *Near-body characteristic visualization using flow fields obtained from a direct, Semi-Lagrangian simulation. The image shows instantaneous pressure field on the upper-body surface during a dolphin kick motion. The pressure values are normalized and color-coded using the HSV color bar shown. The red tone indicates high pressure and the blue tone indicates tangential surface pressure flow.*

our work, is widely acknowledged to be a highly complex problem to which no known verifiable solution exists for the general case (Negrao, 1995).

Typically, the nature and the objective of the simulation have direct implications on the choice of methodology, accuracies, and stability of the numerical process. As such, classical CFD literature contains an abundance of techniques that are applicable to only specialized classes of flow problems. Indiscriminate use of generic CFD software, other than on the most trivial problems, often churns results whose accuracies are questionable, possibly meaningless, and cannot be validated in today's state-of-the-art of computational fluid dynamics. Generally, substantial knowledge in the theories of fluid dynamics and the many limitations and caveats of simulation software are necessary in determining the choice or applicability of any solution method.

On the other hand, in computer graphics research, techniques have emerged that were able to create stunningly realistic animations of fluid phenomenon with complexities far beyond the typical engineering CFD experiment. Furthermore, these simulations are conducted at a fraction of the computational costs while engineering CFD harness super-computers in the order of hours/days of computation time, the typical graphics-based technique can be deployed on general-purpose computers.

This apparent gap between the two classes of CFD developments comes as no surprise. The motivation and background underlying the developments within each research community is largely different. The Semi-Lagrangian Stable Fluids method (Stam, 1999) and its various derivatives, commonly used for the synthesis of visual effects, are purported approximations of fluid dynamics. Nevertheless, as the method retains its physical basis through the Navier-Stokes equations, the simulations that it produces often appear realistic and convincingly plausible. The typical engineering techniques, such as discretization methods or turbulence models, on the other hand, are less approximated and rely on carefully discretization of the problems at high resolution. These techniques are therefore much slower and complex than computer graphics' methods. They often need special treatment for each specific problem domain, for otherwise, the results are unreliable and the techniques can easily become unstable.

Our work focuses on leveraging the speed and the ability to simulate complex scenes when applying Computer Graphics' fluid simulation techniques to applications that are not easily achievable using classical CFD techniques. However, because there is no complete verification method existed for complex CFD problems, and because of the approximation nature of most simulation techniques, we must be careful in the choice of application as well as must have a proper understanding of the possible limitations and expected errors when using fast CG fluid simulation techniques. Only under this proper understanding, can we make an informed choice of applications and make just assessment of the outcome of this work.

1.1 Numerical dissipation problem

Numerical dissipation, a generic term referring to fluid momentum that are lost or unaccounted for, is the main problem with the Semi-Lagrangian method. The errors manifest as large, implausible damping of velocities in the fluid field. Such dissipation is a direct result of the approximation mechanism, where advection flows are traced and interpolated at a finite number of grid points. Unfortunately, there is presently no known method to quantify the magnitudes of these inaccuracies. The lack of analytical solutions, and the fact that these highly-unsteady simulations are not typical of Engineering experiments, further prevents validation or comparisons between results. It should be noted however that such lack of solutions in classical CFD is not due to a lack of interest, but rather that the problems are generally considered to be overwhelmingly difficult with today's technology. Thus, when assessing the soundness and applicability of any CFD method for solving any fluid dynamics problem, an informed rationalization coupled with an understanding of the systemic limitations is required.

Velocity dissipation are first-order numerical errors whose severity increases over time. As a result, the fluid medium in a Semi-Lagrangian simulation tends to appear more viscous than it were. Effects such as vortex shedding, trails, and other flow phenomenon are visibly dampened. For many engineering simulations, the steady-state presence of vortexes are of primary concern because it is the basis for evaluating the performance or feasibility of an engineered design. In other applications, however, steady-state outcome is less critical and dissipation errors are arguably more tolerable. For instance, in simulations where near-body characteristics are of primary interest, the presence of vortexes that are located at a distance is of little consequence. And in simulations where the volume of fluid is constantly moving forward, shedded fluid trails are of little concern. In these scenarios, the artificial dampening, which effectively waters down these effects, may pose a lesser influence on the validity and usability of the results.

Under no circumstances, however, can a claim of correctness be made (and such is neither the purpose nor conclusion of this work). After all, all numerical simulations make assumptions and ours is no different. However, given the physical basis of the formulation, coupled with an understanding of the nature in the expected errors, we postulate that the applicability of some of these recent CFD developments in computer graphics may extend beyond its traditional realm.

1.2 Thesis objective and contribution

There are three main objectives for our work. The first objective is to develop a method for generic simulation and interaction model between a fluid volume and a general 3D deformable mesh. The method should be able to work with captured movement of a 3D human model or any other 3D mesh. The second objective of our work is to apply the simulation and interaction result to visualize the near body hydrodynamics information, which is useful in improving the swimmer's performance. The third objective is to extend this simulation and interaction scheme into other novel applications, such as improvement of fire animation.

In summary, our contributions are as following:

- We proposed a new interaction model between fluid and general 3D meshes.
- Our interaction model is not dependent on the dynamics of the 3D object. The 3D model can freely deform in any way and our interaction model will take care of the deformation effects on the fluid. This is extremely useful in the case where we need to investigate the interaction between the swimming person and fluid. In this situation, we only able to capture the swimmer movement without any knowledge of how the movement or the forces required to produce the movement.

- Our work is the first to investigate the possibilities to apply fast and approximated simulation methods developed by graphics community into everyday life application, particularly, in the visualization of resistive forces on swimmer body. This application might be useful for swimming coaches who desire to find way to improve swimmer performance, but do not have any knowledge nor the time of how to setup and run a fluid simulation problem in engineering setup.
- We also proposed to use this interaction model in other applications such as fire trails enhancement. This kind of application show that our method can be applicable in the game setting where approximated interactions between game characters and fluids is needed.

1.3 Results summary

Our work adapts the Semi-Lagrangian Stable Fluids simulation method of (Stam, 1999) to the biomechanical visualization of hydrodynamics surrounding the body surface of a swimmer. The adaptation can also be used genericly to simulate the effect of a moving 3D semi-rigid object on a fluid volume. Figure 1.1 shows a still image from the simulation. The motion involves several cycles of full-submerged dolphin-kicks that were digitized from a series of video frames. The shaded surface encodes the pressure flow field on the surface of the body. The animated visuals depict smooth and intuitive pressure variations that are meaningful to the understanding of biomechanical hydrodynamics.

Further more, we also demonstrate the general applicability of the near-body dynamics fluid field through its usage in a novel application. A hypothetical muscular shape rigid body was filled with procedural fire animation and moved within a closed fluid volume. The interaction between the fluid and the muscle shape disturb the velocity field, which, in turn, directs and enhances the movement of the fire trails and gives them a natural look.

Although procedural fire is a fast method for generating fire animation, it lacks a necessary physical basis to be interactive with surrounding environment. With the introduction of the fluid velocity field, we have helped create a new level of interaction. Among many possibilities, we can create novel situation where the fire can interact with fluid medium such as water, which is not possible in real life or add in a windy air flow that can affect the movement of the fire particles' shapes.

1.4 Thesis organization

The rest of this thesis is organized as follows: chapter 2 briefly discuss Navier-Stoke equations, which are the mathematical background of our work, and the related background literature; chapter 3 describes the technical details and our contribution to the fluid/surface interaction model; chapter 4 describes the visualization of the results by applying the simulation on a motion capture sequence of a scanned human geometry. We also describe the fire interaction example and fluid's interaction with hypothetical objects. We conclude the thesis with a discussion of our approach and some of its limitations in chapter 5.

Chapter 2

Background

2.1 Fluid simulation and animation for computer graphics

Fluid simulation in computer graphics has a long standing development history of nearly two decades. The techniques which are used to generate fluid animation in computer graphics range from kinematics method to dynamics method in both 2D and 3D.

Among the earliest techniques, (Reeves, 1983) and (Sims, 1990) used particle systems to simulate fuzzy objects like water. Here, primitive particles are assigned kinematic attributes such as velocities, positions, colours and are randomly introduced into the particle systems to simulate fluid particles. Although not following an accurate physical process, these methods produced acceptable visual results. The complexity of fluid flows was greatly enhanced with the introduction of random turbulences in (Stam and Eugene, 1993). However, these turbulences did not have the physical basis, they were just random turbulence vector fields varying over space and time and were generated using Fourier synthesis. Because lacking of physical background, these early methods could only produce visual results without additional interactions with external forces and objects.

The set of Navier-Stoke equations is the mathematical model that describes the general fluid movement. By incorporating these equations the researchers were able to produce more compelling and accurate animations of fluid or fluid like phenomena.

Early applications of the Navier-Stoke equations were first implemented in 2D. In (Yaeger et al., 1986) the equations were used to solve for the vorticity movements that resemble the planet Jupiter's surface. Similarly, in (Gamito et al., 1995) also used vorticity coupled with a Poisson solver on a velocity field to simulate the fluid movement on 2D. The vorticity movements were modelled as the movement of particles in the velocity fields and their positions were integrated forward in time. Later, in (Chen et al., 1997) a height field was derived from the pressure term which is obtained by solving a 2D version of the Navier Stoke equations. The height field was then used to animate the water surface. These methods were however not stable.

3D animation of fluid was first developed in (Foster and Metaxas, 1997) by finite differencing the Navier-Stoke equations with an explicit time solver. This method is however not stable in its nature. Even with small time step or high resolution grid, the finite differencing numerical scheme will eventually blow up and the simulation has to be restarted.

To solve the instability problem, (Stam, 1999) introduced a semi-Lagrangian approach. This method essentially borrows the Lagrangian view point to solve the advection part of the Navier Stoke equations using back-tracking of imaginary particles through the fluid velocity field. This alleviates the need for solving the advection equation with finite differencing scheme and makes the method unconditionally stable. The stability is obtained from the fact that the new velocity values cannot exceed the maximum velocity of the original velocity field. Beside the stability advantage, this method was also the first to introduce the operator splitting scheme to the graphics community. This scheme helps breaking down complex partial differential equations (PDE) into manageable pieces which can be solved or approximated using specialized techniques. The major problem with this semi-Lagrangian method is numerical dissipation where artificial viscosity is introduced into the solution through numerical error from interpolations at grid points. This causes the velocity field to be dampened and the total energy is not

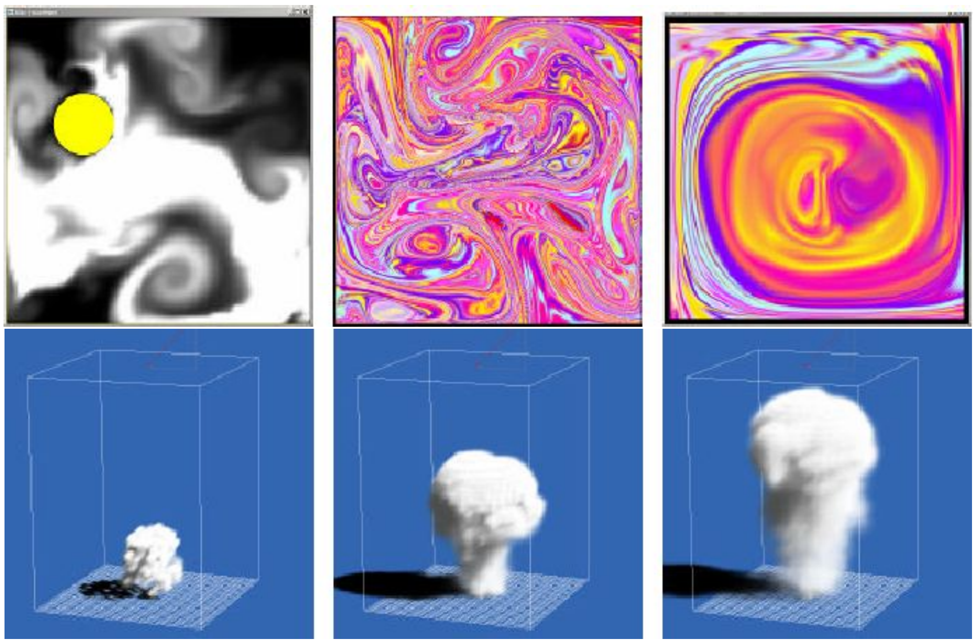

Figure 2.1: *2D simulation results (upper) and 3D simulation result (lower) of Jos Stam's method. Pictures from Stam (2003)*

conserved. Some results of this method are shown in figure 2.1.

Based on the semi-Lagrangian method many variations have been developed to simulate the fluid movement, to capture the fluid surface and the fluid interaction with the surrounding environment. Since our work only concerns with the fluid movement and interaction with submerged objects but not the fluid surface, in the next section we will review some prominent work related to this area.

2.2 Fluid and object interaction

Several solutions have been proposed to model the fluid and object interactions. For example, some authors (Foster and Metaxas, 1997; Stam, 1999) treated rigid bodies as boundary condition and set the fluid velocities around the object's boundary. This velocity fixing is often sufficient to coerce fluid flows to go around the object.

In (Foster and Fedkiw, 2001) improved the technique by letting the fluid to flow freely around a polygonal rigid object along the tangents. The improvement was also adopted by others e.g. (Enright et al., 2002).

Carlson (Carlson et al., 2004) introduced the rigid-fluid method as a rigorous way to ensure the motion consistency between interacting fluid and solids. The method, which accounts for two-way coupling between fluids and solids, enforces a rigid motion constraint on cells that are occupied by the same rigid body. The result is that the rigid cells are simply fluids with very high viscosity, and a deformation operator is used to restrict their movements. Interestingly, substances that are less-solid, such as melting wax, can also be modelled with a similar approach. It is not trivial and not clear of how to represent 3D deforming triangle meshes using this representation of solid.

Olivier (Arash et al., 2003) introduced a different way to couple fluid with solids. In his work, he bridged the Eulerian fluid simulation using velocity field with the Lagrangian rigid body representation through introduction of marker in fluid and the spring mass system in the rigid body. Although this representation leads directly to an interaction model using impulse force calculation, it is not suitable for representing complex non-deforming rigid object.

(Müller et al., 2004) also introduced an interaction model between fluid and deformable objects based on triangular mesh. This model based on several boundary conditions and interaction between the triangles and the fluid particles. For example, there are no-slip and non-penetration condition at the boundary of the object. This work however is only applicable to fluid simulated with Smooth Particle Hydrodynamics (SPH).

(Klingner et al., 2006) introduced a different model of fluid animation in which fluid is not simulated in a fix grid but on a tetrahedral mesh which changes based on the boundary of the simulated environment, including the boundary of the moving object. The generation of the tetrahedral mesh is however a large overhead in addition to the fluid simulation. Beside, it is not clear how to refine the mesh in order to achieve desirable near body hydrodynamics visualization results.

(Chentanez et al., 2006) presented a method of two way coupling between deformable elastic object and fluid through the combination of the fluid simulation's projection step and elasticity simulation's integration step into one single equation. This method show a physically correct interaction model. However, it is not suitable for our project since its require both the object and the fluid to be simulated and it only works with elastic objects. For general captured models and motions, there is not an existing method to estimate the physical characteristics that decide the motions, hence we can not come up with a single physics equation for both the captured motion and the fluid simulation.

The processing power of GPU has also been utilized in fluid simulation. (Harada et al., 2007) used GPU to simulate the two way coupling of fluids and rigid bodies. They used particle based fluid simulation and particle system for rigid bodies. Although the method works fast it only works with rigid bodies, it is unclear how to employ the method with general mesh based object. However, GPU is also a promising area that can greatly enhance the grid based fluid simulation speed.

The difference of our method is that we only focus on one way coupling between object movement and the surrounding fluid modelled on a 3D spatial grid. This is specially suitable with the investigation and visualization of the fluid effects produced by predefined 3D object movement, in contrast to two ways coupling methods. Our interaction model is simple, fast, and general and can handle semi-rigid objects. Its results are also shown to be useful in simulating and visualizing near-body hydrodynamics information.

2.3 Fluid simulation and application in engineering and biomechanics

Computational fluid dynamics within Engineering, Applied Mathematics, and other branches of sciences have taken on substantially different paths. Due to differences in levels of tolerance in accuracies, the methods that are favoured also differ between the communities. Suffice to mention, however, the desire to synthesize meaningful visuals which depict natural phenomena is not unique to computer graphics. The ability to visualize, understand,and optimize the dynamics of a swimmer has long been something that swimming coaches and sports scientists desired.

In the context of biomechanical hydrodynamics visualization, a few attempts have been made using methods of classical CFD. Some of these works deal with external fluid phenomenon of simpler biological mechanisms (Mittal, 2004), fluid flows internal to biological bodies (Mittal et al., 2004), while others work with scanned human bodies in steady states (Hyman, 2004).

2.4 Navier-Stoke equations

Our work deals with incompressible fluid simulation and is entirely governed by the famous Navier-Stoke equations. Because of their importance, in this section, we will briefly explain the derivation of these equations to help the reader understand the underlying physical concepts as well as the methods to solve these equations. For a more complete explanation of these equations in the computer graphics context, we refer the reader to (Bridson and Müller-Fischer, 2007).

The Navier-Stoke equations are a set of partial different equations (PDE) that hold through out the simulated fluid volume. They are written as:

$$\frac{\partial \vec{u}}{\partial t} = \vec{g} - \vec{u} \cdot \nabla \vec{u} - \frac{1}{\rho} \nabla p + \upsilon \nabla \cdot \nabla \vec{u} \tag{2.1a}$$

$$\nabla \cdot \vec{u} = 0 \tag{2.1b}$$

The symbols used in these equations are explained as below:

- $\vec{u}$: is used to represent the velocity field of the fluid volume.
- p : stands for the pressure.
- ρ : stands the fluid density.
- $\vec{g}$: stands for the gravitational acceleration. Note that, in application that additional control is required, control forces can be added on top of the

gravity. And hence, $\vec{g}$ is more generally called "**body forces**" because it is applied through out the fluid volume.

- υ : stands for the fluid's kinematic viscosity.
- ∇ : stands for the gradient operator that is commonly used in calculus.
- The dot $\cdot$: stands for the dot product operation.

The first differential equation (2.1a) is sometimes called the "momentum equation". Although looking quite complex, it is actually derived from the fundamental Newton's second principal which govern how an object accelerates given the force acting on it: $\vec{F} = m\vec{a}$.

To derive the equation (2.1a), we start with a model of the fluid movement. The most common way of modelling the fluid flow is using a particle system, that is to think of the fluid as consisting of very small fluid blobs. Each blob has a mass m and a volume V and a velocity $\vec{u}$. In order to integrate the system over time, we need to figure out the forces acting on a fluid blob. Then the equation $\vec{F} = m\vec{a}$ tells us how the fluid blob accelerates. The acceleration of the fluid blob is the time derivative of its velocity:

$$\vec{a} = \frac{D\vec{u}}{Dt} \tag{2.2}$$

The forces that act on the fluid blob then can be related to its velocity:

$$\vec{F} = m\frac{D\vec{u}}{Dt} \tag{2.3}$$

The most obvious force acting on the blob is gravity $m\vec{g}$. Other part of the fluid also exerts forces on the blob. First, there is the fluid pressure force. The high pressure region will push against the lower pressure region. We are however, not interested in the pressure itself but the difference of pressure between the regions. The simplest way to measure this difference is to use the gradient operator: ∇p. We need to integrate this over the fluid blob volume to get the pressure force:

$$\vec{F}_{pressure} = -V\nabla p \tag{2.4}$$

V is the volume of the fluid blob. The negative sign is from the fact that the force will point away from high pressure region, toward to lower pressure region. Here we use the gradient operator ∇ to measure the spatial differences in the pressure field. The discretization of this operator will be shown in later section.

The second type of force that exerts on the fluid blob from other part of the fluid is the viscous force. Just like frictional force, viscous force is caused by the fluid blobs moving at different velocities rubbing against each other. The force tries to minimize the difference in velocities of nearby fluid blobs. We can use the differential Laplacian operator $\nabla \cdot \nabla$ to measure the difference of the velocity of the fluid blob with the surrounding. To get the viscous force, we multiply it with the dynamic viscosity coefficient and integrate over the fluid blob's volume.

$$\vec{F}_{viscosity} = V\mu\nabla \cdot \nabla\vec{u} \tag{2.5}$$

Putting everything together, we have:

$$m\frac{D\vec{u}}{Dt} = m\vec{g} - V\nabla p + V\mu\nabla \cdot \nabla\vec{u} \tag{2.6}$$

Divide both side with V we have

$$\rho\frac{D\vec{u}}{Dt} = \rho\vec{g} - \nabla p + \mu\nabla \cdot \nabla\vec{u} \tag{2.7}$$

We divide both side by the density ρ to get:

$$\frac{D\vec{u}}{Dt} = \vec{g} - \frac{1}{\rho}\nabla p + \upsilon\nabla \cdot \nabla\vec{u} \tag{2.8}$$

υ is the kinematic viscosity equal to $\frac{\mu}{\rho}$.

We almost arrive at the momentum equation (2.1a). In equation (2.8) we use the material derivative, the reason for this is that velocity of a fluid particle depends on both time and its position. By definition material derivative of a quantity can be expanded using the total derivative as:

$$\begin{aligned}\frac{Dq}{Dt} &= \frac{d}{dt}q(t,\vec{x}) \\ &= \frac{\partial q}{\partial t} + \frac{\partial q}{\partial \vec{x}} \cdot \frac{d\vec{x}}{dt} \\ &= \frac{\partial q}{\partial t} + \nabla q \cdot \vec{u}\end{aligned} \tag{2.9}$$

Substitute the material derivative in (2.9) into (2.8) we have our momentum equation:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} = \vec{g} - \frac{1}{\rho}\nabla p + \upsilon \nabla \cdot \nabla \vec{u} \tag{2.10}$$

The second equation in the Navier Stoke system, equation (2.1b) is sometimes called the diversion free condition or incompressibility condition of the fluid. The reason for this condition is that we assume the fluid keeps constant volume at all time. While in reality, fluid might change its volume, but that will only noticeable under extreme circumstances such as incredibly high pressure or in phenomena such as explosion. In our case, we want to model the interaction between a moving semi-rigid object and fluid under normal condition, hence, the compressibility of the fluid can be ignored.

The assumption of incompressibility and constant volume of the fluid lead to the fact that the amount of fluid that flow into and out of a constant fluid region must be equal. Hence, the total rate of change of fluid velocity in all direction of the fluid blob must be zero. The total rate of change of velocity can be measured using the divergence operator (hence the name divergence free condition):

$$\begin{aligned}\nabla \cdot \vec{u} &= \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z} \\ &= 0\end{aligned} \tag{2.11}$$

In this section, we have briefly presented the derivation of Navier-Stoke equations. The method to solve these equations numerically to obtain a fluid simulator will be discussed in section 3.3.1.

Chapter 3

Methodology and Implementation

3.1 Components and Approach

Our goal is to examine the interaction between a moving semi-rigid 3D object and then visualize the near-body hydrodynamics of the interaction. In this section we will go through the components and the key steps in our algorithm.

The first important component of our algorithm is the 3D object, which is modelled as a series of fixed number of key-frames. Its movement is obtained when the key frames are played, the time step between each key frame is known. Each key frame is a 3D triangle mesh, whose normals at vertices and faces points outward from the mesh. We use triangle mesh for generality purpose but any type of mesh such as procedural mesh as in section 4.2 can also be used as long as the mesh can be voxelized and the velocity of boundary voxels can be calculated in some ways.

The second component is the fluid simulator, which runs on a 3D simulation grid. The simulator's role is to update the velocity field on the simulation grid. At each instance, the simulator takes in a key frame, voxelizes it, calculates the boundary voxels' velocities, incorporates the velocities into the simulation grid and integrates the simulation grid over time to calculate the new velocities at all other grid points. The simulation time step should match the time step between the object's key frames, otherwise, we choose to interpolate the object's key frames to have matching time steps.

The whole process is summarized in figure 3.1:

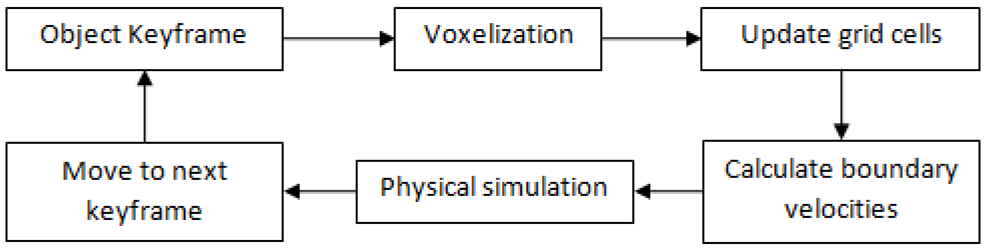

Figure 3.1: *Algorithm summary*

Since our objective is to visualize the near-body hydrodynamics characteristics given the object's movement, hence, the movement can be considered unchanged and defined by fixed key frames. Thus, the voxelization, and calculation of boundary cells' velocities can be considered preprocessing steps. For real time application where the object movement is unpredictable, the object meshes and its boundary voxels' velocities must be voxelized and calculated online.

3.2 3D Model voxelization

Any general object constructed from several 3D triangular mesh key frames can be voxelized. The models we are interested in this project are laser scanned models of human-bodies. We assume the motion comprises of a fixed number of key frames, as is the case with conventional motion capture. To generate the model's motion, we bind the external geometry of the model to a skeletal structure, whose movement deforms the skin as guided by a pre-defined set of deformation weights (Wang and Phillips, 2002). As a result, each key frame of the model is set up to have the same number of vertices and faces, which are well corresponded between key frames. This is an important property of our 3D model as it facilitates a simple calculation method for boundary voxels' velocities.

The voxelization step essentially classifies the space around the geometry of each key frame at the same resolution as fluid simulation grid's resolution. When the geometry is situated in the fluid simulation grid, each spatial grid cell will be classified as an interior, boundary or exterior cell. Due to the discretization of the space, the grid cells cannot faithfully reconstruct the geometry volume. As a result, a boundary cell might not lie on the actually geometry surface, hence we need a clear definition of what it means by an exterior cell, interior cell and a boundary cell:

- Exterior cells are cells whose center points lie completely outside of the geometry.
- Interior cells are cells whose center points lie completely inside the geometry surface and are not adjacent to any exterior cell.
- Boundary cells are cells whose center points lie on the geometry's triangles, or lie inside the geometry but are adjacent to at least one exterior cell.

This definition makes sure that there is not any whole in the voxelized volume hence preventing the fluid to flow through the object.

Several methods exist for voxelizing a general polyhedral models, of which we employed the method similar to (Baerentzen and Aanaes, 2005).

The first step is to classify whether a grid point lies inside or outside of the key frame's geometry. Algorithm 1 contains the pseudo-code for this step. Basically, for each of the grid points, we find the closest point to it on the geometry, and then use the dot product of the vector between the two points and the normal at the closest point to decide whether the grid point lie inside or outside of the geometry.

We should note that in Algorithm 1 the normal $\vec{n}$ is not the real normal of the geometry, but rather the pseudo-normal calculated as weighted average of the geometry faces' normals. The calculation of this pseudo-normal gives a more accurate representation of the geometry's vertices' normals than traditional average

Algorithm 1 Classifying the grid points

```
for each grid point $P$ do
    $closestPoint$ = NULL
    for each triangle $T$ of the geometry do
        $P1 \leftarrow$ Closest point on $T$ to $P$
        if $closestPoint$ = NULL or distance($P$,$P1$) $<$ distance($P$, $closestPoint$) then
            $closestPoint \leftarrow P1$
        end if
        $\vec{n} \leftarrow$ Geometry normal at $closestPoint$
        $\vec{x} \leftarrow$ vector from $P$ to $closestPoint$
        if $\vec{n} \cdot \vec{x} \geq 0$ then
            $insideSet \leftarrow P$
        else
            $outsideSet \leftarrow P$
        end if
    end for
end for
```

normals. More detailed explanation for the use of the pseudo-normal is given in section 3.2.1.

After classification of the inside voxels and outside voxels, the boundary voxels can be easily found. We simply iterate through all the inside voxels. The ones that have at least one exterior voxel adjacent to it will be considered the boundary voxels.

After all the key frames have been voxelized, the next step is to calculate the boundary voxels' velocities. Since we do not have the correspondences of the boundary voxels between two key frame, we choose to approximate the velocities through the vertex velocities of the actual mesh. In figure 3.2, we calculate the velocity of the closest point P' of boundary voxel P and assign that velocity to voxel P. The velocity of P' is calculated as the interpolation of the velocities of the three vertices of the triangle containing P'. If it is an edge or vertex that P' falls on, then the interpolation will be from two vertices and one vertex respectively. The velocity of a vertex is calculated through finite differencing the position of

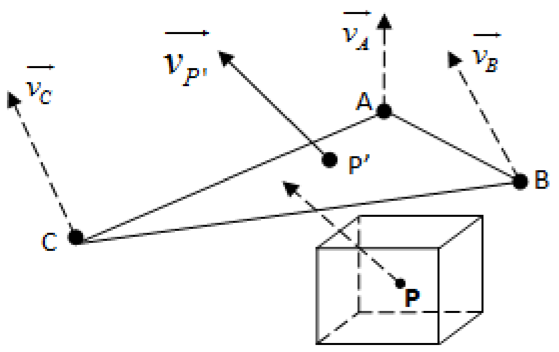

Figure 3.2: *Velocity of a boundary voxel is approximated as the velocity of its closest point on the mesh, which is interpolated from the velocities of the vertices of the containing triangle.*

that vertex and the corresponding vertex in the next key frame.

3.2.1 Angle weighted pseudo-normal

As has been mentioned earlier, we used the angle-weighted normals instead of average normals for a point on the geometry surface. This is a more accurate measurement of the true object's shape as it can avoid bias when a vertex has many incident faces, as seen in figure 3.3. Hence, we can avoid false classification of exterior point as interior point when using algorithm 1, as in the case of point S in figure 3.3.

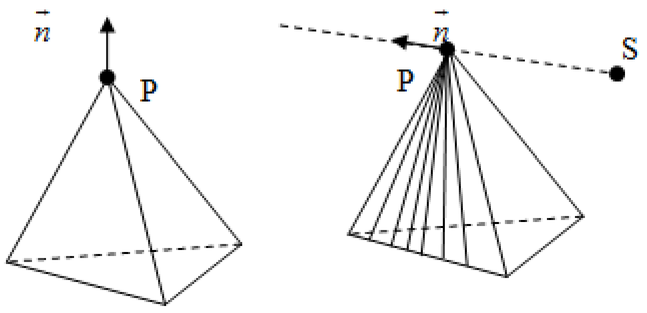

Figure 3.3: *The normal at the vertex is changed if the triangle is divided into multiple smaller triangles. The averaged normal is biased toward the direction of the smaller triangles' normals.*

The formulation of angle-weighted normal is straight forward:

$$\vec{n} = \frac{\sum_{i=1}^{n} \alpha_i \vec{n_i}}{\sum_{i=1}^{n} \alpha_i} \tag{3.1}$$

Where as α_i is the incident angle on the *ith* face that contains the point P at which we are calculating the normal. This is illustrated in figure 3.4. When the point P falls on exactly one face, the formula degenerate into exactly the normal of the face:

$$\vec{n} = \frac{2\pi \vec{n}_f}{2\pi} = \vec{n}_f \tag{3.2}$$

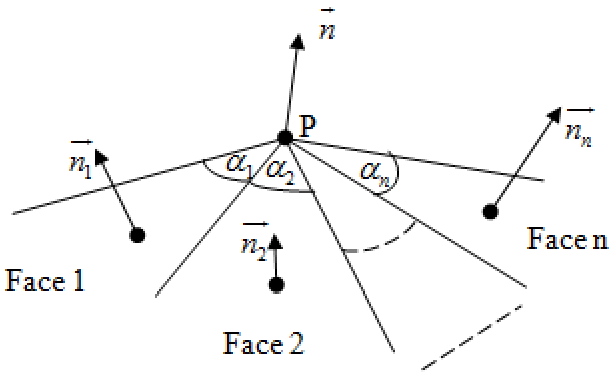

Figure 3.4: *The angle weighted normal.*

3.3 Fluid simulator

The fluid simulator is the most complicated component of our system. We build the simulator by numerically solve the Navier-Stoke equations (2.1). At each time step, the solver will integrate overtime to find new velocity value for the fluid field. Beside the normal steps needed for the velocity field integration, we need to incorporate one extra step to treat the velocities at the object boundary cells.

In section 3.3.1, we will break down the Navier-Stoke equations to see the required steps for the fluid simulation and in section 3.3.2 we will go into the detailed implementation of the simulator and also how to implement the extra step for incorporation of the object's movement.

3.3.1 Breaking down the Navier-Stoke equations

The momentum equation (2.1a) is quite complicated at the first glance. There are four terms in this equation, namely, the advection term $-u \cdot \nabla u$, the diffusion term $v(\nabla \cdot \nabla u)$, the pressure term $-\frac{1}{\rho}\nabla p$ and the acceleration term due to external forces, and lumped under $\vec{g}$. There are multiple ways of solving the momentum equation, but we will employ the operator splitting scheme which works particularly well and very suitable for graphics application. The motivation for using this scheme is that we can divide the equation into smaller, more manageable pieces and solve each piece with the special numerical method that is suitable for it. We should note that it is possible to solve all the terms at once, however, it will force us to deal with the complexity of the whole equation and the algorithm might not be as efficient as the employment of a special algorithm for each of the term.

Similar to Forward Euler method for solving differential equation numerically, the splitting scheme is also a first order accurate method. To understand this method, let us look at a simple example to illustrate how the splitting scheme works. Suppose we want to solve a simple ordinary differential equation:

$$\frac{dq}{dt} = f(q) + g(q) \tag{3.3}$$

We want to obtain the new value of q^{n+1} after a small period of time Δt from the current value q^n. The splitting scheme says that we can do this through Forward Euler method in two step. First, we solve for an intermediate value q_1:

$$q_1 = q^n + \Delta t f(q^n) \tag{3.4}$$

And second, we solve for the value q^{n+1} from q_1:

$$q^{n+1} = q_1 + \Delta t g(q_1) \tag{3.5}$$

Solving the equation 3.3 into steps, we have effectively divided a more complicated problem into two simpler independent problems where we might have special numerical methods that can solve them more efficiently. The two simpler differential equations are:

$$\frac{dq}{dt} = f(q) \tag{3.6a}$$
$$\frac{dq}{dt} = g(q) \tag{3.6b}$$

Applying this scheme to our momentum equation we can solve the equation through solving four independent differential equations independently and sequentially.

The four equations are:

$$\frac{\partial \vec{u}}{\partial t} = \vec{g} \tag{3.7a}$$
$$\frac{\partial \vec{u}}{\partial t} = -u \cdot \nabla \vec{u} \tag{3.7b}$$
$$\frac{\partial \vec{u}}{\partial t} = \upsilon \nabla \cdot \nabla \vec{u} \tag{3.7c}$$
$$\frac{\partial \vec{u}}{\partial t} = -\frac{1}{\rho} \nabla p \quad s.t. \quad \nabla \cdot \vec{u} = 0 \tag{3.7d}$$

At each time step, the simulator would solve these four equations sequentially, as shown in figure 3.5. For each step, the input will be the velocity field of the previous step. The easiest step in the above process is the add force step. This

$$\begin{tikzcd} u^n \arrow[r, "\text{Add force}"] & u_1 \arrow[r, "\text{Advection}"] & u_2 \arrow[r, "\text{Diffusion}"] & u_3 \arrow[r, "\text{Projection}"] & u^{n+1} \end{tikzcd}$$

Figure 3.5: *The sequence of steps for the simulator in each time steps.*

step is used to add in the gravity force or the control force to direct the fluid flow. We assume that the animator force can be stored in a force field similar to the velocity field, and the force at the position $\vec{x}$ will be $F(x)$. The velocity field after applying this step will be:

$$\vec{u_1}(x) = \vec{F}(x)\Delta t \tag{3.8}$$

The advection is responsible for swirling motion and the turbulences of the fluid. A disturbance somewhere in the fluid is transported to another position in the fluid volume by its own movements. This is a difficult term to solve and is the cause of the instability in many methods. Intuitively, we can solve this with the Forward Euler method. In one dimension we can write the equation as following:

$$\begin{aligned} \frac{\partial u}{\partial t} &= -u \nabla u \\ &= -u \frac{\partial u}{\partial x} \end{aligned} \tag{3.9}$$

Converting 3.9 into discretized form, we have:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = -u\frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} \tag{3.10}$$

Here i is the index of the ith cell along the discretized dimension x. u_i and u_{i+1} are the velocities at two adjacent cells. Rearrange this equation, we have an explicit formula for the velocity field in the next time step in one dimension (this can easily be extended to three dimensions):

$$u_i^{n+1} = u_i^n - \Delta t u_i^n \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} \tag{3.11}$$

This seems to be a really simple formula, there are unfortunately serious problems with this Forward Euler method. Firstly, the Forward Euler has been proven to be **unconditionally unstable** for the discretization of the spatial derivative ∇u. What it means is that if we keep integrating the velocity field over time, no matter how small the time step is, the integration will inevitably introduce some errors and eventually blow up the simulation as the error is accumulated over time. Secondly, the very high frequency components of the solution will erroneously register as having zero or near zero value spatial derivative (Bridson and Müller-Fischer, 2007). Hence, this component will not be integrated and changed much more slower than other low frequency component. This will introduce serious errors to the simulation and affect the visualization of the result.

Fortunately, there is a method that borrows the Lagrangian view of the fluid. Using this view, fluid is just a set of small particles that are moving around. Because of this, the velocity of a point $\vec{x}$ is actually just the old velocity of the particle that end up at that position at the current time. Applying this reasoning to the velocity field we can get the semi-Lagrangian method, used by (Stam, 1999). What we want is to figure out the velocity at a grid position $\vec{x}$. Imagine that there is a virtual particle that flow along with the fluid and end up at the position $\vec{x}$. The starting position of the particle would be:

$$\vec{x}_S = \vec{x} - \Delta t \vec{u}_1(\vec{x}) \tag{3.12}$$

The value we want to know is the new velocity at position $\vec{x}$, $\vec{u}_2(\vec{x})$. This value will be exactly the current velocity at the starting position of the virtual particle ending up at x which is $\vec{u}_1(\vec{x}_S)$. But what if the position $\vec{x}_S$ is not on the grid? The answer is that we can interpolate the velocity at $\vec{x}_S$ from the surrounding velocities value. For simplicity, we use tri-linear interpolation of the surrounding voxels' velocities. More accurate interpolations scheme are available. The advection step is summarized in figure 3.6.

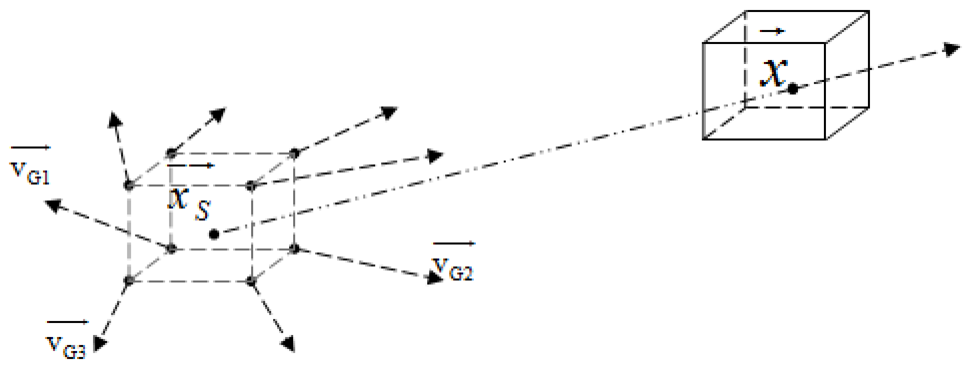

Figure 3.6: *The new velocity at $\vec{x}$ is interpolated from the old velocity surrounding $\vec{x}_S$. The points surrounding the position $\vec{x}_S$ are the centres of the surrounding cells.*

And the final formula for u_2 is:

$$\vec{u}_2(\vec{x}) = \vec{u}_1(trackback(\vec{u}_1, \vec{x})) \tag{3.13}$$

Where $trackback(\vec{u}_1, \vec{x})$ is the function that gives the tracked back position from $\vec{x}$ in the velocity field $\vec{u}_1$.

Under careful numerical analysis, the interpolation made by this advection step results a problem known as numerical dissipation (Bridson and Müller-Fischer, 2007). This means that instead of solving the original PDE equation 3.7b, we actually solved a modified PDE of the form:

$$\frac{\partial \vec{u}}{\partial t} = -\vec{u} \cdot \nabla \vec{u} + \vec{u} \Delta x \nabla^2 \vec{u} \tag{3.14}$$

The extra term is a viscosity like term, which makes the fluid appear more viscous than normal, and gradually over time the fluid loses its energy. There are methods to reduce the effect of this problem. The details of these methods can be found in (Bridson and Müller-Fischer, 2007).

The third step to solve the Navier-Stoke equations is the diffusion step, which accounts for the change in velocity field due to viscosity of the fluid. This step is equivalent to the diffusion equation:

$$\frac{\partial \vec{u}_3}{\partial t} = \upsilon \nabla \cdot \nabla \vec{u}_2 \tag{3.15}$$

Similar to the advection equation, we can solve this directly using the Forward Euler method and use finite differencing to discretize the equation. However, doing it that way can cause the instability for the simulation process. The problem is that when the viscosity υ or the time step is large, the effect of diffusion might be spread over a range of several cells. The direct finite differencing on the other hand, only account for the diffusion of adjacent cells. To go around this problem, we once again borrow the tracking back idea from the advection step. Instead of integrating forward from the current velocity field $\vec{u}_2$, we choose to find $\vec{u}_3$ that when **diffuse backward** we obtain $\vec{u}_2$ the equation for this process is as follow:

$$\vec{u}_3 - \Delta t \upsilon \nabla \cdot \nabla \vec{u}_3 = \vec{u}_2 \tag{3.16}$$

Discretizing the Laplacian operator in the above equation results in a system of linear equation, which can be solve effectively using existing iterative methods. Our implementation for this diffusion step will be presented in section 3.3.2.

The final pressure step requires to find the pressure of the fluid. But what is pressure and how to calculate the pressure field? Because pressure is the push of one water region against another region, it actually relates to the incompressibility of the fluid. The more fluid flows into an infinitesimal volume the more pressure it creates to push fluid out of the volume on the other side. So we can consider pressure as forces that ensures the incompressibility of the fluid volume. Mathematically we can relate the velocity and the pressure through the incompressibility equation,

$$\begin{aligned}\frac{\partial \vec{u}^{n+1}}{\partial t} &= -\frac{1}{\rho}\nabla p \qquad .s.t \quad \nabla \cdot \vec{u}^{n+1} = 0 \\ \vec{u}^{n+1} &= \vec{u}_3 - \Delta t \frac{1}{\rho}\nabla p \qquad .s.t \quad \nabla \cdot \vec{u}^{n+1} = 0\end{aligned} \tag{3.17}$$

Multiply each side of the equation (3.17) with $\nabla\cdot$ we have:

$$\nabla \cdot \vec{u}^{n+1} = \nabla \cdot \vec{u}_3 - \nabla \cdot \Delta t \frac{1}{\rho}\nabla p \qquad .s.t \quad \nabla \cdot \vec{u}^{n+1} = 0 \tag{3.18}$$

Substitute the divergence free constraint we have:

$$\begin{aligned} 0 &= \nabla \cdot \vec{u}_3 - \nabla \cdot \Delta t \frac{1}{\rho}\nabla p \\ \nabla \cdot \vec{u}_3 &= \Delta t \frac{1}{\rho} \nabla \cdot \nabla p \end{aligned} \tag{3.19}$$

From (3.19) we can calculate the pressure by discretize the $\nabla\cdot$ and $\nabla \cdot \nabla$ operator. We will not discuss the discretization of these operators, cause a complete reference can be found in (Bridson and Müller-Fischer, 2007). Substitute the pressure we calculated in (3.19) into (3.17) we can calculate the new velocity field $\vec{u}^{n+1}$. This concludes one cycle of the fluid simulator.

We have gone through the steps to evolve the fluid velocity field from the current value $\vec{u}^n$ to the new value $\vec{u}^{n+1}$. The next section 3.3.2 discusses the implementation details of the fluid simulator and the incorporation of the object movements.

3.3.2 Grid structure for fluid simulation

Before going into the details of the implementation, we should first define the grid structure that our whole system would run on. The grid structure is used to represent the fluid state as well as for voxelization of our models. The visualization of the grid is shown in figure 3.7. The velocity field is a 3D grid of known size $S1$x$S2$x$S3$ in dimension x, y, z respectively. The voxels are of equal size h. For each voxel, we define a velocity vector $\vec{v}$ at the center and a pressure value p. In our implementation, we represent the velocity field is represented with three dimensional float C++ arrays. We have one array for each component x, y or

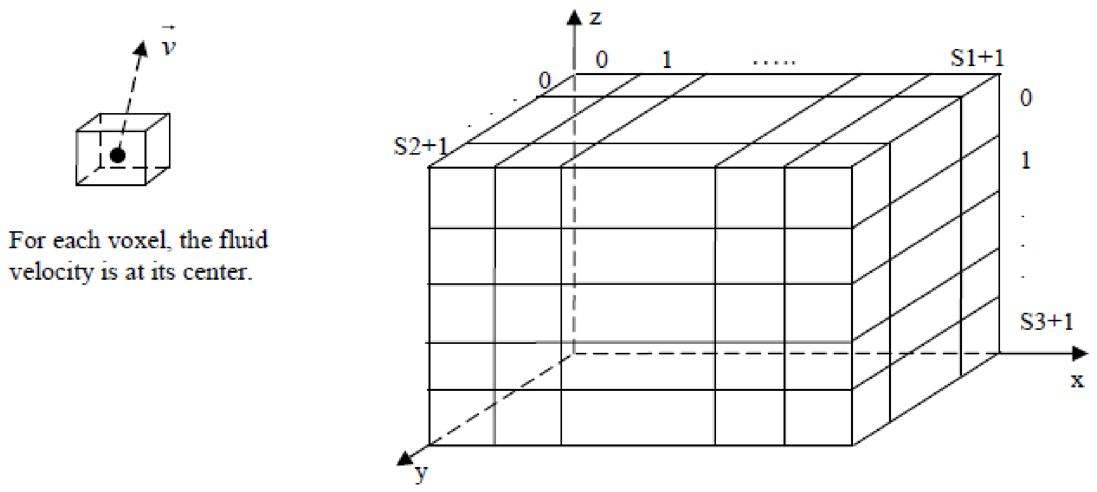

Figure 3.7: *The grid structure for the fluid simulator and the voxelization of object models.*

z of the velocities and one array to store pressures in the voxels. We also need to handle the boundary of the grid. For convenience, we pad each array with 2 extra units in each dimension, therefore the each array will be of dimension $(S1+2)$x$(S2+2)$x$(S3+2)$. With this padding, we can avoid lengthy conditional when access the cells at position 0 or position $S1$+1 or $S2$+1 or $S3$+1 of the grid. These edge cells also contain the boundary values that establish the boundary conditions of the simulation.

3.3.3 Implementing the add force step

We assume the accelerations we want to add into the velocity field is stored in 3D arrays in a similar fashion with the storage of the velocity fields. Each component of the acceleration in x or y or z dimension is stored in a 3D float array. The algorithm for add force step is listed in algorithm 2. This algorithm only add one component of the velocity field with one component of the force. To complete the add force step, we repeat this procedure with all three components of the velocity field.

For additional convenience, the user can also add the force at specific position in

Algorithm 2 The add force procedure.

```
1: procedure AddForce(Array3D $u$,Array3D $f$, float $timeStep$)    ▷ Add the acceleration in $f$ into velocity field $u$
2:     for $i = 1$ to $S1$ do
3:         for $j = 1$ to $S2$ do
4:             for $k = 1$ to $S3$ do
5:                 $u[i][j][k]$ += $f[i][j][k]$*$timeStep$;
6:             end for
7:         end for
8:     end for
9: end procedure
```

the velocity field with the algorithm 3

Algorithm 3 The add force procedure.

```
1: procedure AddForce(Array3D $u$,int $x$, int $y$, int $z$, float $forceValue$, float $timeStep$)
2:     $u[x][y][z]$ += $forceValue$*$timeStep$;
3: end procedure
```

3.3.4 Implement the advection step

The implementation of the advection step implements equation (3.7b). It follows exactly the simple backtracking method described earlier. As has been shown in figure 3.6 the new velocity at point P is calculated as the linear interpolated velocity at point P' which is tracked back from P over one time step. We use tri-linear interpolation of the 8 cells surrounding the point P'. In the case that P' lies outside the grid's boundary, we choose to clamp the indexes and get the velocity that is nearest to P' as the interpolant. We treat each component of the velocity field separately and advect each component independently.

Suppose P' has position (x, y, z) on the grid. Let (i, j, k) be the integer such that $i \leq x \leq i + 1$, $j \leq y \leq j + 1$, $k \leq z \leq k + 1$ then the eight cells that surround

P' are at positions: (i, j, k), $(i+1, j, k)$, $(i, j+1, k)$, $(i, j, k+1)$, $(i+1, j+1, k)$, $(i+1, j, k+1)$, $(i, j+1, k+1)$, $(i+1, j+1, k+1)$.

Then the advection can be done as in algorithm 4. In this algorithm, we passed in the quantity we wanted to advect as old quantity *oq* and stored the advected result as new quantity in *nq*. The old quantity can be the current velocity field component, or an array storing any quantity such as heat of the fluid. We also passed in the current velocity field in *ux*, *uy*, *uz*. We also specified the type of quantity in *oq*. The variable name *dimension* signals that most of the time, *oq* is the velocity components in different dimensions. If $dimension = 1$, we are advecting the x component of the velocity and 2 for the y dimension, and 3 for the z dimension. If $dimension = 0$, we are not advecting the velocity but another non-velocity value. At the end of the algorithm, we had to take care of the boundary values of the advected result. Here we choose to set the boundary values to zeros because the fluid cannot flow at the boundary. The routine *SetBoundaryZero* will be discussed in the separated section 3.3.7.

3.3.5 Implement the diffuse step

The diffusion step is responsible for propagating fluid velocity from one voxel to another voxels due to effect of viscosity. A voxel can exchange velocity with its six adjacent neighbours, two in each direction. The amount of exchange is governed by equation (3.15). However, as discussed earlier, due to stability problem, we will use equation (3.16). The equation is rewritten here for reading convenience:

$$\vec{u}_3 - \Delta t v \nabla \cdot \nabla \vec{u}_3 = \vec{u}_2$$

Algorithm 4 The advection procedure.

```
procedure ADVECT(Array3D $oq$,Array3D $nq$, Array3D ux, Array3D uy, Array3D uz, Integer dimension)
    $factor = timeStep/cellSize$
    for $i = 1$ to $S1$ do
        for $j = 1$ to $S2$ do
            for $k = 1$ to $S3$ do
                $x = i$ - $ux[i][j][k]$*$factor$
                $y = j$ - $uy[i][j][k]$*$factor$
                $z = k$ - $uz[i][j][k]$*$factor$

                $i1$ = floor($x$), $i2 = i1 + 1$
                $j1$ = floor($y$), $j2 = j1 + 1$
                $k1$ = floor($z$), $k2 = k1 + 1$

                $r1 = x$ - $i1$, $r2 = 1$ - $r1$
                $t1 = y$ - $j1$, $t2 = 1$ - $t1$
                $s1 = z$ - $k1$, $s2 = 1$ - $s1$

                $nq[i][j][k] =$
                        $r1$ * ($t1$ * ($oq$[i1][j1][k1] * $s1$ + $oq$[i1][j1][k2] * $s2$) +
                                $t2$ * ($oq$[i1][j2][k1] * $s1$ + $oq$[i1][j2][k2] * $s2$)) +
                        $r2$ * ($t1$ * ($oq$[i2][j1][k1] * $s1$ + $oq$[i2][j1][k2] * $s2$) +
                                $t2$ * ($oq$[i2][j2][k1] * $s1$ + $oq$[i2][j2][k2] * $s2$))

                SetBoundaryZero(nq, dimension)
            end for
        end for
    end for
end procedure
```

We need to discretize the Lapalacian operator $\nabla \cdot \nabla$ in order to implement the equation:

$$\begin{aligned}\nabla \cdot \nabla \vec{u}_3 =& \frac{\partial^2 \vec{u}_3}{\partial x^2} + \frac{\partial^2 \vec{u}_3}{\partial y^2} + \frac{\partial^2 \vec{u}_3}{\partial z^2} \\ =& \frac{\vec{u}_3(i+1,j,k) + \vec{u}_3(i-1,j,k) - 2\vec{u}_3(i,j,k)}{\Delta x^2} + \\ & \frac{\vec{u}_3(i,j+1,k) + \vec{u}_3(i,j-1,k) - 2\vec{u}_3(i,j,k)}{\Delta y^2} + \\ & \frac{\vec{u}_3(i,j,k-1) + \vec{u}_3(i,j,k-1) - 2\vec{u}_3(i,j,k)}{\Delta z^2}\end{aligned} \tag{3.20}$$

Because we are using square grid, $\Delta x = \Delta y = \Delta z = h$. Substitute this discretization into equation (3.16) we have an equation relate the velocities of the voxels in the new velocity field $\vec{u}_3$ to those of the old velocity field $\vec{u}_2$.

$$\begin{aligned}\vec{u}_2(i,j,k) =& \vec{u}_3 - \Delta t v (\frac{\vec{u}_3(i+1,j,k) + \vec{u}_3(i-1,j,k) - 2\vec{u}_3(i,j,k)}{h^2} + \\ & \frac{\vec{u}_3(i,j+1,k) + \vec{u}_3(i,j-1,k) - 2\vec{u}_3(i,j,k)}{h^2} + \\ & \frac{\vec{u}_3(i,j,k-1) + \vec{u}_3(i,j,k-1) - 2\vec{u}_3(i,j,k)}{h^2})\end{aligned} \tag{3.21}$$

Each voxel of the grid give us an equation like (3.21). These equations construct a sparse linear equation system and can be solved efficiently using a Gauss-Seidel relaxation iterative solver. The pseudo-code for this is listed in Algorithm 5 Similar to the advect algorithm, *oq* is the array that store the current quantity we want to advect and *nq* is the place where we store the advected result. In our experience, we use 20 iterations to solve the linear system with acceptable accuracy. The number of iteration can be tuned for better accuracy. One intuitive way to tune this parameter is that it should grow with the size of the grid. The larger the grid the more iterations we need to get to the convergence of the solution. In some implementations, instead of using a fix number of iterations we can measure the amount of change between iterations to decide when to stop the algorithm.

Algorithm 5 The diffuse procedure.

```
1: procedure DIFFUSE(Array3D $oq$,Array3D $nq$, Integer $dimension$, Float $visocsityCoefficient$)
2:     $factor = visocsityCoefficient*timeStep*cellSize$
3:     for $round = 1$ to 20 do
4:         for $i = 1$ to $S1$ do
5:             for $j = 1$ to $S2$ do
6:                 for $k = 1$ to $S3$ do
7:                     if $round = 0$ then
8:                         $nq[i][j][k] = (oq[i][j][k] +$
9:                                 $factor*(oq[i-1][j][k] + oq[i+1][j][k] +$
10:                                        $oq[i][j+1][k] + oq[i][j-1][k] +$
11:                                        $oq[i][j][k+1] + oq[i][j][k-1])$
12:                                $)/(1+6*factor)$
13:                    else
14:                        $nq[i][j][k] = (oq[i][j][k] +$
15:                                $factor*(nq[i-1][j][k] + nq[i+1][j][k] +$
16:                                        $nq[i][j+1][k] + nq[i][j-1][k] +$
17:                                        $nq[i][j][k+1] + nq[i][j][k-1])$
18:                                $)/(1+6*factor)$
19:                    end if
20:                end for
21:            end for
22:        end for
23:    end for
24:    SetBoundaryZero($nq$, $dimension$)
25: end procedure
```

3.3.6 Implement the project step

This step ensures the divergence free condition of the fluid field. We will use equation (3.7d) to implement this step. As seen in equation (3.17) we can calculate the next velocity field $\vec{u}^{n+1}$ from the $\vec{u}_3$ by subtracting from it the gradient of the pressure field. Equation (3.19) gives us a way to calculate the pressure field.

$$\nabla \cdot \vec{u}_3 = \Delta t \frac{1}{\rho} \nabla \cdot \nabla p$$

In section 3.3.5 we have discussed how to discretize the Laplacian operator $\nabla \cdot \nabla$. Now we just need to discretize the operator $\nabla\cdot$ and then we can use the iterative Gauss Seidel solver similar to the one in 3.3.5 to solve for pressure field p.

The discretization of the left hand side is as following:

$$\begin{aligned}
\nabla \cdot \vec{u}_3 &= \frac{\partial(\vec{u}_3)_y}{\partial y} + \frac{\partial(\vec{u}_3)_y}{\partial y} + \frac{\partial(\vec{u}_3)_z}{\partial z} \\
&= \frac{(\vec{u}_3(i+1,j,k))_x - (\vec{u}_3(i-1,j,k))_x}{2\Delta x} + \\
&\quad \frac{(\vec{u}_3(i,j+1,k))_y - (\vec{u}_3(i,j-1,k))_y}{2\Delta y} + \\
&\quad \frac{(\vec{u}_3(i,j,k+1))_z - (\vec{u}_3(i,j,k-1))_z}{2\Delta z}
\end{aligned} \tag{3.22}$$

where $\Delta x = \Delta y = \Delta z = h$.

After discretizing the Laplacian operator on the right hand side of equation (3.19) and solving the linear system we can easily substitute the pressure field into equation (3.17) to calculate the final velocity field $\vec{u}^{n+1}$.

$$\vec{u}^{n+1} = \vec{u}_3 - \Delta t \frac{1}{\rho} \nabla p$$

The gradient of the pressure field can be easily discretized:

$$\begin{aligned}\nabla p(i,j,k) &= (\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y}, \frac{\partial p}{\partial z}) \\ &= (\frac{p(i+1,j,k)-p(i-1,j,k)}{2\Delta x}, \\ &\quad \frac{p(i,j+1,k)-p(i,j-1,k)}{2\Delta y}, \\ &\quad \frac{p(i,j,k+1)-p(i,j,k-1)}{2\Delta z})\end{aligned} \tag{3.23}$$

where $\Delta x = \Delta y = \Delta z = h$.

Now, we have gathered all the ingredients needed to implement the project step. The pseudo-code for this step is listed in algorithm 6 and 7.

3.3.7 Enforce the fluid volume boundary

In our implementation, we assumed that the fluid is moving within the containment of a rigid wall. There are many ways to enforce this constraint. One can set fluid velocity at the boundary voxels such that the fluid can only flow along the rigid wall but not flow into the rigid wall. This means that at the left and right walls, the x component of the fluid velocity is set to zero. On the other hand, one can make the fluid bounce back when it hit the wall. This means that we set the velocity at the boundary voxels to be of the opposite sign to the velocity in the adjacent voxels. Each method of boundary handling will create different effect on the fluid volume. Intuitively, we often observe that fluid flow along hard wall of the container instead of bouncing back, hence we choose the first method in our implementation. The pseudo-code is listed in algorithm 8.

3.3.8 Incorporate the object movement

We have seen how to implement the fluid solver. Our last step in the implementation is the incorporation of the object movement into the fluid volume. Here, we have to especially care about the fluid velocity at the boundary voxels of the

Algorithm 6 The project procedure.

```
procedure PROJECT(Array3D $ux$,Array3D $uy$, Array3D $uz$)
    Array3D $p$, $d$                    ▷ Store the pressure and divergence calculations
    for $i = 1$ to $S1$ do
        for $j = 1$ to $S2$ do
            for $k = 1$ to $S3$ do
                $d[i][j][k] = (ux[i+1][j][k]$ - $ux[i-1][j][k]$ +
                              $uy[i][j+1][k]$ - $uy[i][j-1][k]$ +
                              $uz[i][j][k+1]$ - $uz[i][j][k-1])/(2*cellSize)$
                $p[i][j][k] = 0$
            end for
        end for
    end for
    $f = cellSize*cellSize*fluidDensity/timeStep$
    for $round = 1$ to $20$ do
        for $i = 1$ to $S1$ do
            for $j = 1$ to $S2$ do
                for $k = 1$ to $S3$ do
                    $p[i][j][k] = (p[i+1][j][k] + p[i-1][j][k]$ +
                                  $p[i][j+1][k] + p[i][j-1][k]$ +
                                  $p[i][j][k+1] + p[i][j][k-1]$ - $f*d[i][j][k])/6$
                end for
            end for
        end for
    end for
```

Algorithm 7 The project procedure (continue).

```
    for $i$ = 1 to $S1$ do
        for $j$ = 1 to $S2$ do
            for $k$ = 1 to $S3$ do
                $ux[i][j][k]$ -= 0.5*($p[i+1][j][k]$ - $p[i-1][j][k]$)$/cellSize$
                $uy[i][j][k]$ -= 0.5*($p[i][j+1][k]$ - $p[i][j-1][k]$)$/cellSize$
                $uz[i][j][k]$ -= 0.5*($p[i][j][k+1]$ - $p[i][j][k-1]$)$/cellSize$
            end for
        end for
    end for
    SetBoundaryZero($ux$, 1)
    SetBoundaryZero($uy$, 2)
    SetBoundaryZero($uz$, 3)
end procedure
```

voxelized object model. One hard constraint we need to maintain is that the fluid cannot flow into the object body. Our idea to implement this step is that we calculate the local interactions of the fluid layer at the object boundary voxels and the propagation of disturbance will be automatically taken care of by the fluid simulator.

For any given boundary voxel on any given key frame we define following four variables:

- $\vec{v}_f$: the current fluid velocity at the voxel.
- $\vec{v}_o$: the object velocity at the voxel. This velocity is calculated as explained in section 3.2
- $\vec{n}$: the normal of the object boundary at the voxel. This is the approximated angle weighted normal as explained in section 3.2.1. The normal is pointing out of the object surface.
- υ : the viscosity coefficient that control how viscous the fluid is when flow along the object surface.

Algorithm 8 Setting the boundary velocity routine.

```
1: procedure SetBoundaryZero(Array3D $u$, Integer $dimension$)
2:     if $dimension = 1$ OR $dimension = 0$ then
3:         for $j = 0$ to $S2+1$ do
4:             for $k = 0$ to $S3+1$ do $u[0][j][k] = 0$ $u[S1+1][j][k] = 0$
5:             end for
6:         end for
7:     end if
8:     if $dimension = 2$ OR $dimension = 0$ then
9:         for $i = 0$ to $S1+1$ do
10:            for $k = 0$ to $S3+1$ do $u[i][0][k] = 0$ $u[i][S2+1][k] = 0$
11:            end for
12:        end for
13:    end if
14:    if $dimension = 3$ OR $dimension = 0$ then
15:        for $i = 0$ to $S1+1$ do
16:            for $j = 0$ to $S2+1$ do $u[i][j][0] = 0$ $u[i][j][S3+1] = 0$
17:            end for
18:        end for
19:    end if
20: end procedure
```

The fluid must not flow through the object boundary and into the body hence the relative object fluid velocity must be greater or equal to zero when projected on the normal vector. Let $\vec{v}_{fn}$ and $\vec{v}_{on}$ be the velocity of the fluid and the boundary at the same voxel along the normal vector direction. We have:

$$\begin{aligned} \vec{v}_{fn} &= (\vec{v}_f \cdot \vec{n})\vec{n} \\ \vec{v}_{on} &= (\vec{v}_o \cdot \vec{n})\vec{n} \end{aligned} \tag{3.24}$$

The relative velocity along $\vec{n}$ is $\vec{v}_{rn} = \vec{v}_{fn} - \vec{v}_{on}$. If $\vec{v}_{rn}$ is in the same direction with $\vec{n}$ then the fluid is flowing away from the object and at the same time being pushed by the object. Hence we reassign the fluid velocity at the voxel as:

$$\vec{v}_{fn} = \vec{v}_{fn} + \vec{v}_{on} \tag{3.25}$$

If $\vec{v}_{rn}$ is in the opposite direction with $\vec{n}$ then the fluid is flowing against the object, but it cannot go through hence the fluid velocity is reassigned such that it is exactly the object velocity at the voxel:

$$\vec{v}_{fn} = \vec{v}_{on} \tag{3.26}$$

Grab the two equations together we calculate the normal component of the fluid velocity at boundary voxel as following:

$$\vec{v}_{fn} = \begin{cases} \vec{v}_{fn} + \vec{v}_{on} & \text{if} \vec{v}_{rn}\vec{n} > 0 \\ \vec{v}_{on} & \text{if} \vec{v}_{rn}\vec{n} \leq 0 \end{cases} \tag{3.27}$$

The tangent component of the fluid velocity is also affected by the object movement due to the friction between the surface and the fluid. We use the Newtonian fluid equation to address this frictional force. The Newtonian fluid equation is:

$$\tau = -v\frac{du}{dx} \tag{3.28}$$

Here, τ is the shear stress between the surfaces. du is the difference in the velocities between the two surface and dx is the height of the fluid layer. Let the tangent components of the fluid velocity and the object velocity to be:

$$\begin{aligned}\vec{v}_{ft} &= \vec{v}_f - \vec{v}_{fn} \\ \vec{v}_{ot} &= \vec{v}_o - \vec{v}_{on}\end{aligned} \tag{3.29}$$

Then according to the Newtonian fluid equation, the new tangent component of the fluid velocity is:

$$\vec{v}_{ft} = \vec{v}_{ft} - v\frac{\vec{v}_{ft} - \vec{v}_{ot}}{h} \tag{3.30}$$

After calculating the tangent and the normal components of the fluid velocity at the boundary voxels, we simply add up the two to get the final fluid velocity:

$$\vec{v}_f = \vec{v}_{ft} + \vec{v}_{fn} \tag{3.31}$$

This modification of the fluid velocity at the object boundary voxels is done before the projection step to make sure the changes are propagated to other part of the fluid volume.

Chapter 4

Results

In this chapter, we will present the simulation and visualization results obtained from our program. In particular, we will presented three types of results:

- Visualizations of the near body hydro-dynamics. These results demonstrate our ability to obtain useful information and visualize it from the dynamics simulation. This is the main focus of our work.
- The application of our method to the enhancement of procedural fire trail. This application demonstrates the potential of our method in applications that require real time interaction between fluid and other scene objects, e.g. fire.
- The application of our method for general 3D semi-rigid objects. This application demonstrates the generality of our method, in which, the interaction model is applicable for almost any situation, whether the objects are real life captured models or just hypothetical models for experimental purposes.

4.1 Visualization of near-body hydro-dynamics

Apart from other application that will be presented in later sections, this work's main focus is to visualize the near body characteristics obtained from the fluid flow fields. While it is possible to apply to any motion, it will be more meaningful and useful when we apply this work to real life situation. We chose to apply our simulation and interaction models to the motion capture data of a human swimmer. The ultimate goal of our simulations is to understand and identify the surface hydrodynamics that support or limit the performance of the human athlete.

4.1.1 Data acquisition

We will only briefly discuss the underwater motion capture process, whose details befit its own paper and are secondary in the context of this work. Our setup involves two underwater cameras that are firmly anchored against the side-wall of a swimming pool. The pair of cameras are placed 3 meters apart, with a slight tilt towards the center of the capture volume. A set of four buoyant and four sunken markers were used to provide the camera calibration information. The most challenging aspect of the underwater capture and calibration is the non-uniform attenuation of light intensities. As such, our cameras have wide field of views which require special care in accounting for severe radial distortion in the video images. We calibrate the cameras using an implementation of the Direct Linear Transform (Fitzgibbon, 2001), which accounts for radial lens distortion and require only pairwise point relationships. We assume the light conducting environment to be homogeneous (i.e., underwater) and hence, refraction is not a problem in our case. Figure 4.1 shows frames of the underwater calibration setup and a swimmer's motion. To avoid errors that could be introduced in the digitization process, we restrict ourselves to swimming motions that are planar to the direction of whole-body motion, which is generally true of a submerged dolphin-kick movement. This self-imposed constraint is solely to simplify digitization of the capture process; all other parts of this work are fully functional in 3D.

After recovering the 3-D marker motion, we reconstruct the joint angles by fitting the moving points to a skeleton rigged, laser-scanned geometry. The skinned geometry deforms according to the joint kinematics that is obtained from the motion capture data. Both the geometry and kinematics are then immersed into the computational fluids simulator to estimate the hydro dynamical characteristics

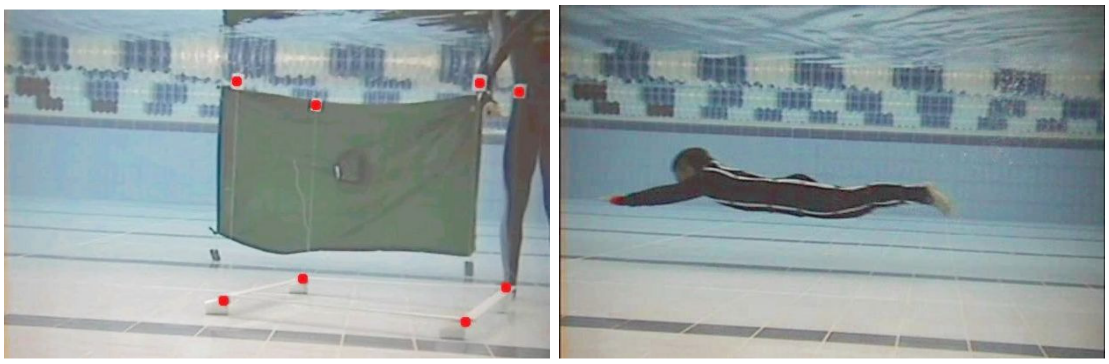

Figure 4.1: *The underwater movements of the human swimmer are recorded using a pair of conventional cameras in water-tight enclosures and firmly anchored to the wall of the swimming pool. (LEFT) Camera calibrations performed using a customized static frame which provides eight markers whose pairwise distances are know. The coordinates of the calibration markers are used to derive the intrinsic and extrinsic parameters of the cameras. (RIGHT) The swimmer is outfitted with a Lycra-made,full-body suit with customized fabric markers. The markers partially assist the tracking process, but due to noise and image degrading factors, tracking of markers is presently a semi-automatic, manually-guided process.*

of the swimmer's motion.

4.1.2 Near body hydrodynamics visualization results

Visualization of hydrodynamic characteristics around body surfaces is one of our objectives. An advantage in using the Semi-Lagrangian method is that an abundance of numerical information is readily available to illustrate spatio-temporal near-body characteristics. A direct visualization of the pressure field provides meaningful information about the experience of the performer during a cycle of the movement.

Figure 4.2 shows a series of time lapsed images depicting pressure variations on the lower leg as it performs a highly-intense downward "whip". The color-coded regions clearly shows that as the shank (red) is moving down, the thigh segment has already initiated an upward cycle (blue). This surface characteristics obtained from the simulation model is consistent with principles of sequential kinetic transfer (Kreighbaum and Barthels, 1996), which advocates a smooth proximal-to-distal successive joint rotations in order to maximize forward thrust (Maglischo, 2003).

One of the traditional ways to study near-body hydrodynamics is through the use of specially-made tuft-suits. These full-body suits are made with fabrics that are covered with short, hair-like strands. An underwater video recording of the swimmer's movements exposes anomalies where surface flows are either not streamlined or are introducing excessive resistance. The main disadvantage of this method is that the stranded suit itself may introduce excessive resistance and drag, which may result in performance degradation. Consequently, the dynamics of the recording may not be representative of the swimmer's peak performance.

By making use of the velocity flow field, we can display particle traces backward in time. This achieves effects similar to tuft-suits, without the burden of the outfit. Figure 4.3 shows a time-lapse sequence that incorporates particle traces from the body surfaces. These traces are obtained directly from the fluid field using the Semi-Lagrangian particle tracing backward in time. The lengths of the traces can be varied to achieve different visuals. To illustrate that our simulator does

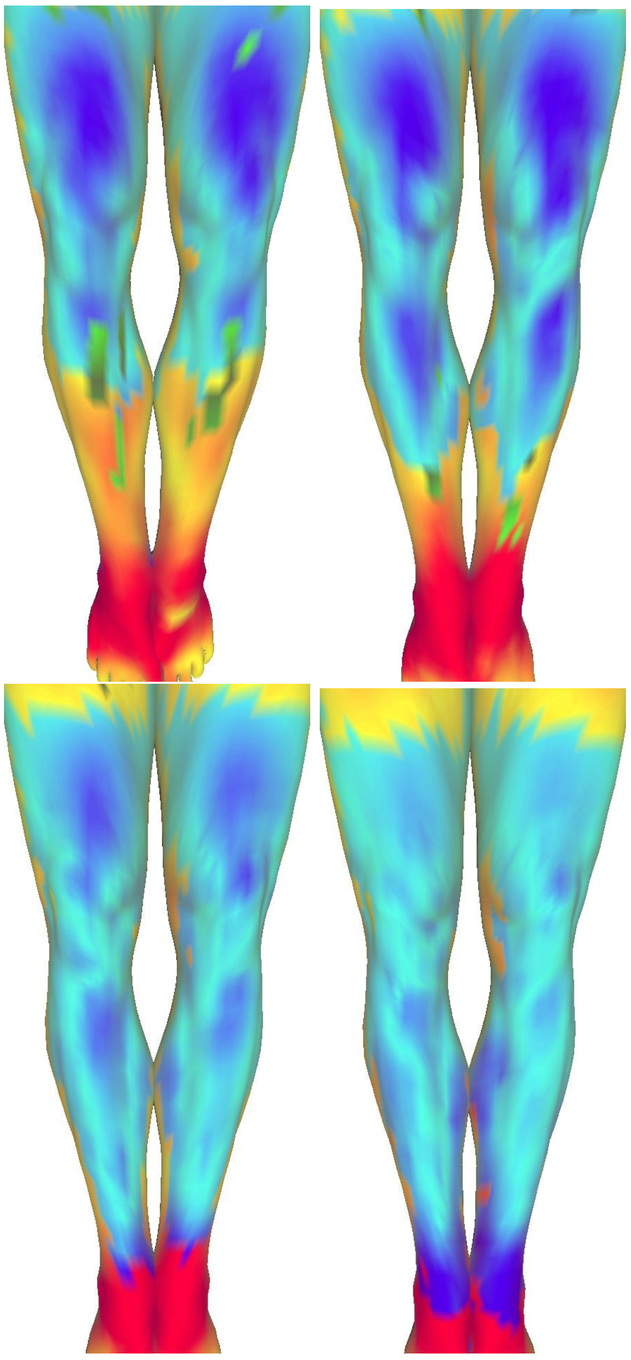

Figure 4.2: *Surface pressure variations on the frontal shank region in a downward phase of the dolphin kick cycle. The proximal-to-distal pressure variations are depicted as continuous color tone variations on the shaded surface.*

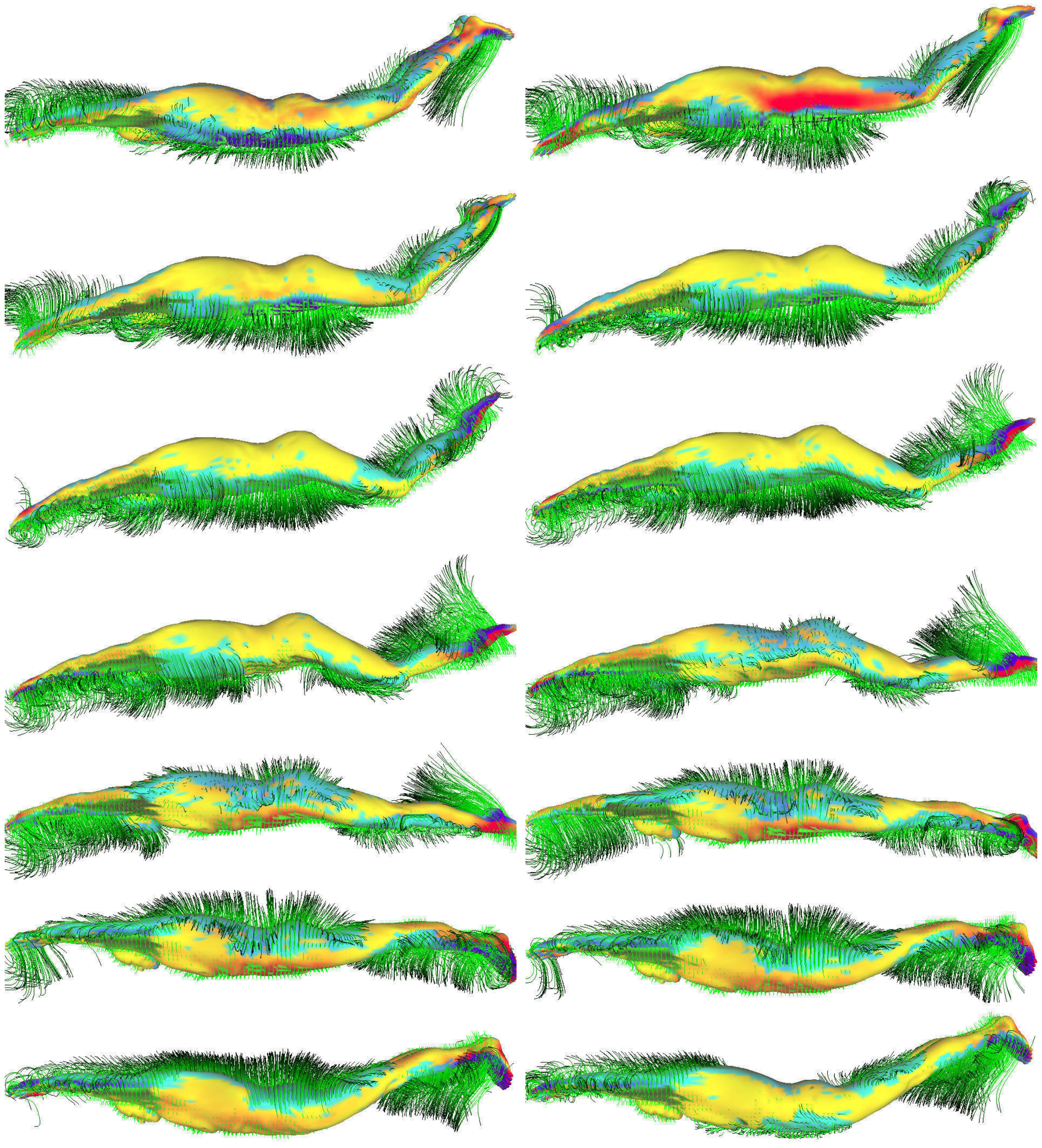

Figure 4.3: *Time-lapse images showing one complete dolphin-kick cycle(1/15 seconds per frame; viewed left to right, top to bottom), illustrating near-body path-lines that traces the characteristic flow fields surrounding a moving boundary surface. Longer paths indicate faster motion at the associated surface point. The path tracing technique was adapted from the particle back tracing method used in the advection algorithm.*

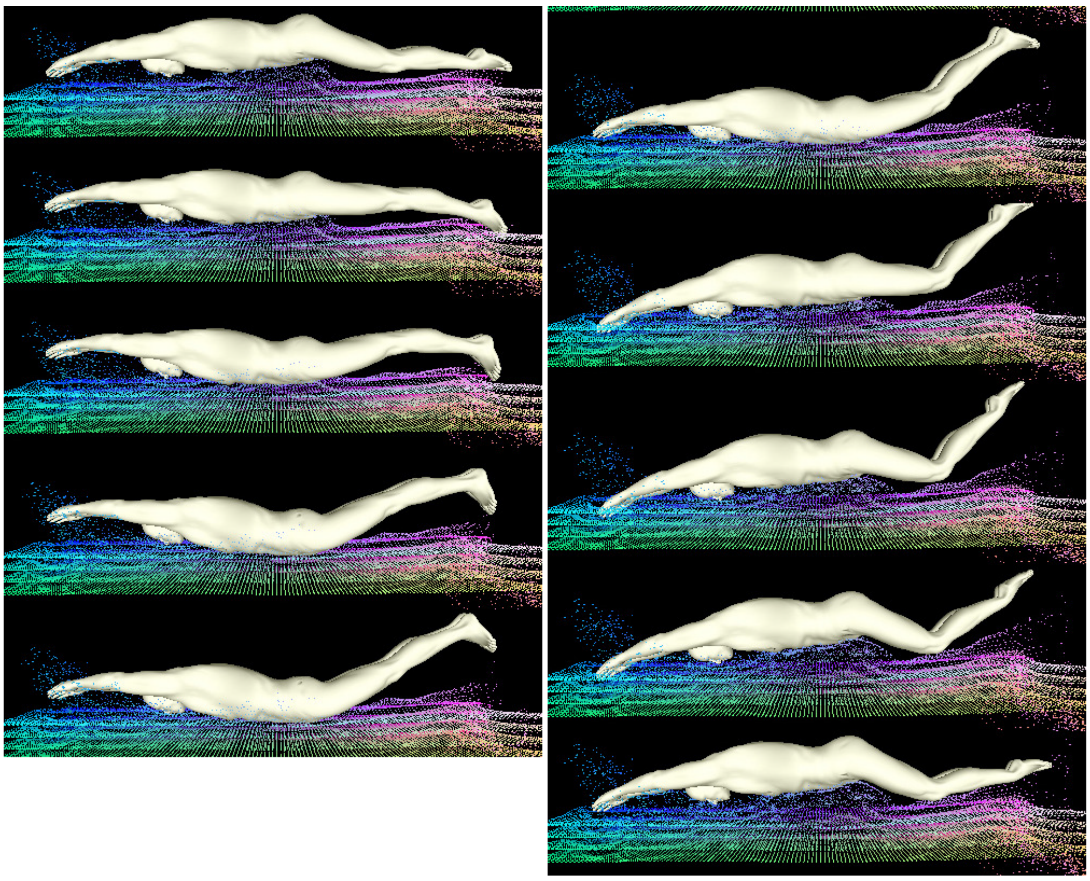

Figure 4.4: *Rapid dispersion of suspended particles in the dolphin kick motion. Viewing order: top to bottom,, left to right.*

indeed simulate fully three-dimensional volumetric fluid flow, and not just body surface effects, Figure 4.4 shows a simulation where planes of massless particles were placed at regular intervals. The results of this simulation, though not the near-body dynamics that we have focused on in this paper, is visually similar to those that have obtained by others in the past e.g. (Foster and Metaxas, 1997; Foster and Fedkiw, 2001; Enright et al., 2002; Carlson et al., 2004).

4.1.3 Tuning of simulation parameters

Various simulation parameters are required in the setting up for a fluid dynamic simulation. These include integration time step, mesh densities, and volumetric resolution. Generally, changing any parameter will result in different simulation and numerical quantities. Further more In traditional computational fluid dynamics simulation, parameters are required to be tuned carefully to avoid errors and instability of the simulation. As such, we realized the importance of the parameter tuning step.

Our strategy to obtain acceptable tuning values is as follows. We identify the fundamental quantity which we are interested in. Typically, this quantity is an integral of some dynamical quantity over the boundary surface. Next, through a systematic process of parameter refinement, we identify the parameter values where the value converges within some thresholds.

To illustrate our approach, the graph in figure 4.5 shows the convergence properties of the body surface pressure integral (Y-axis) against the simulation fluid cell size (X-axis) (or alternatively, the resolution of the volume). These plots of near-body quantities must show a convergent trend, for otherwise the Semi-Lagrangian method cannot be considered a viable solution to the problem. From our experience, plots of parameter against body surface integrals does indeed converge. We therefore analyse each plot, and determine sensible values for use in the simulation. The values used must lie in the convergent zones of the collected quantities.

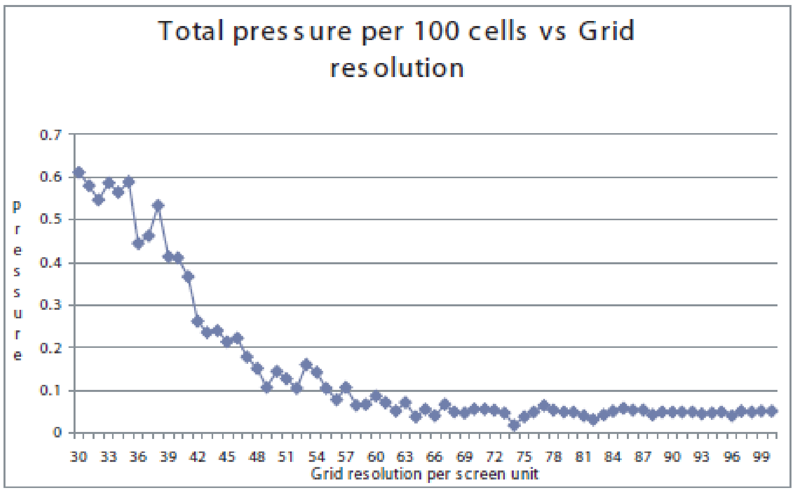

Figure 4.5: *Tuning grid resolution based on the convergence of total pressure of 100 cells.*

4.2 Application on fire enhancement

We also wanted to illustrate that the fluid velocity field and near body quantities such as surface pressure can be useful in many applications. Although through out our discussions, we focus on submerge swimming human, our method is also applicable also for air and other objects.

Fire has long been one of interested effects in many areas such as in games and movies. In (Shijun, 2010), the author has created procedural fire which approximate real fire effect with particles of tear drop shape. Although it is not real simulation of physical processes which create real fire, this approach can create compelling looking animation of fire effect at the speed of hundreds of frames per second. Further more, this approach can create novel, unreal effects such as volume filling or path tracing fire, e.g. figure 4.6.

We have proposed to enhance these effects by extending them with fluid velocity field. With this extension, we submerge the fire filled muscle-shape volume in the fluid simulator just as our swimming human. When the muscle interacts with and affects the fluid volume, we use the hydrodynamics characteristic in the velocity field to move the left behind fire particles and create interesting and natural moving

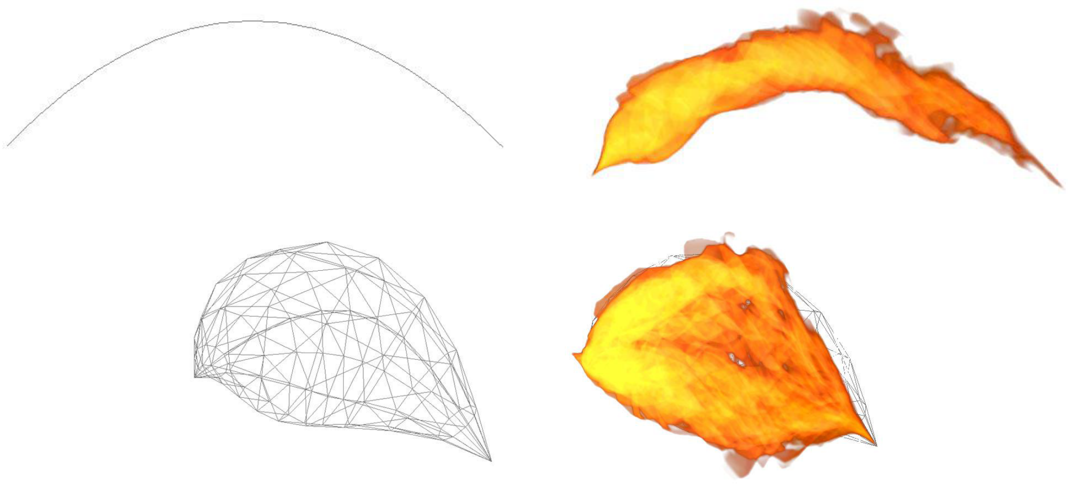

Figure 4.6: *Volume filling (bottom) and path tracing (top) fire effects. Image from (Shijun, 2010)*

fire trail. We compare the fire trails in the case we use and don't use the fluid velocity field to see the effect of our approach as shown in figure 4.7.

As seen in the figure, when using the hydrodynamics, we are able to obtain a natural looking fire trail. The fire particles in the fire trail exhibit fluid movement (see the bottom of figure 4.7). When not using the hydrodynamics feature, the fire particles in the trail stay in place and do not move as a coherent entity (see the top of figure 4.7). The differences are shown more clearly in the accompanying videos. This is one of many possible enhancements we could come up from the combination of a fluid field and a procedural fire implementation. We look forward to creating more interactions between them. Here, we have already been able to create fire trail for a muscle shape volume, it will be interesting to extend this to a novel moving human, which comprises of multiple muscles.

One problem of this application is that the volume movements are not known before hand, and hence it not possible to preprocess the voxelization. Fortunately, the muscle is modelled procedurally and we can devise a simple voxelization strategy for this kind of shape, which is explained briefly in 4.2.1.

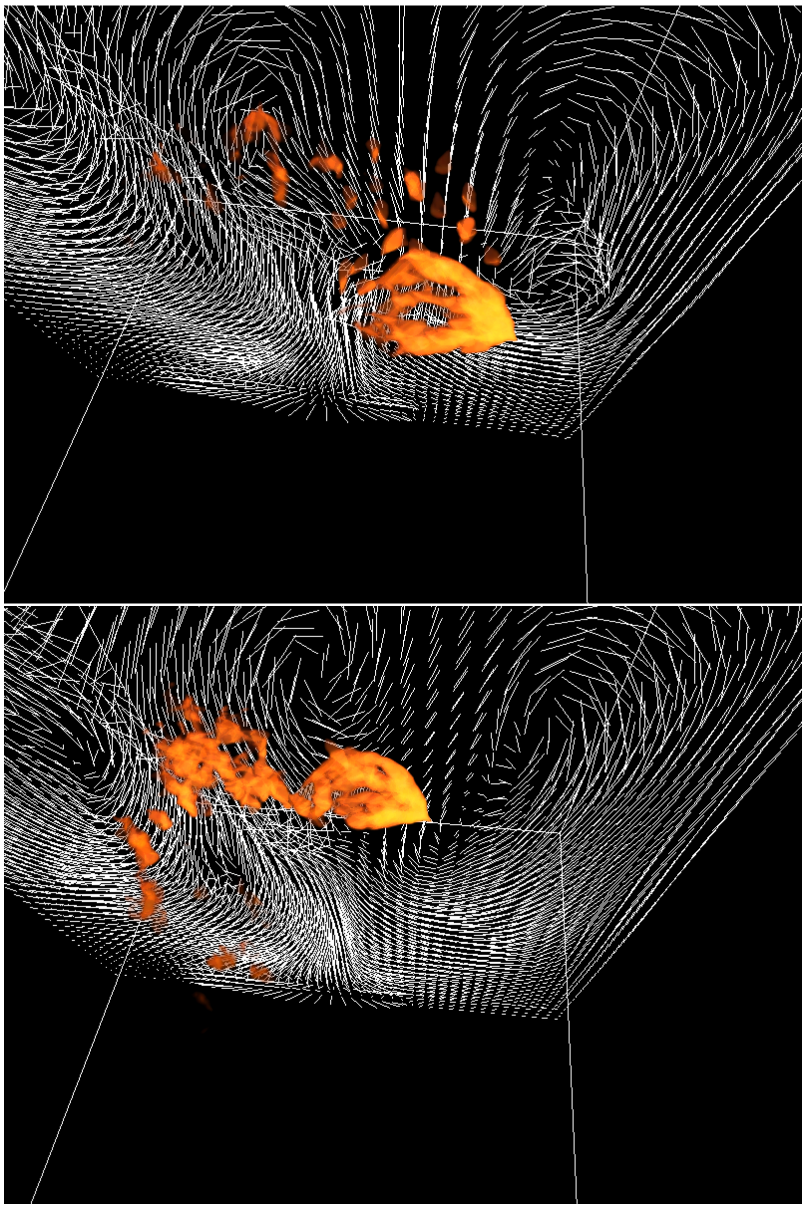

Figure 4.7: *The fire trail created and animated not using the fluid velocity field (TOP) and using the fluid velocity field (BOTTOM). Notice how the fire trails in the are different. In the bottom of the figure, the trail's particles are more coherent and move together. In the top part, the fire trail's particles stay separately and do not move. This can be observed more clearly in the accompanying videos.*

4.2.1 Real time voxelization

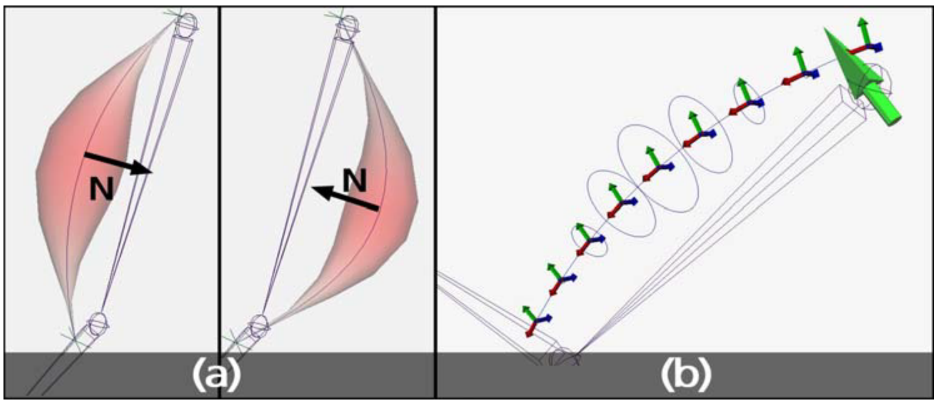

Figure 4.8: *The creation of the muscle shape. Figure from (Keng Siang and Ashraf, 2007)*

The muscle is modelled as in figure 4.8a. At the center there is a action curve (the black curve). This action curve is actually a bezier curve defined by three control points. Denote the action curve as $c(t)$. The muscle volume is defined such that any cross section of the muscle at point t_1 on $c(t)$ and perpendicular to $c(t)$ is a circular disk with radius r_1 dependent on t_1:

$$r_1 = Ssin^T(t_1\pi) \tag{4.1}$$

The constant S and T are parameters to control the bulge size and taper of the muscle shape (Keng Siang and Ashraf, 2007). t_1 is assumed to be in the range $(0, 1)$.

To voxelize the shape defined in this way, we can avoid using its triangle mesh to do voxelization because the number of triangles might be large. Instead, we exploit the implicit discrete representation of the muscle shape. We iterate through the voxels, and for each voxel we decide whether it is outside or inside the volume by calculating the distance from the voxel to the action curve. If the distance is smaller than the radius at the projection point of the voxel on the curve, then the voxel is inside the volume, else, it is out side.

Calculating the distance from the voxel to the bezier curve is not a fast operation.

We would need to calculate the projection of the voxel on the curve, which requires us to solve a polynomial equation of degree 3. We instead choose to discretize the curve into segments to facilitate fast distance calculation.

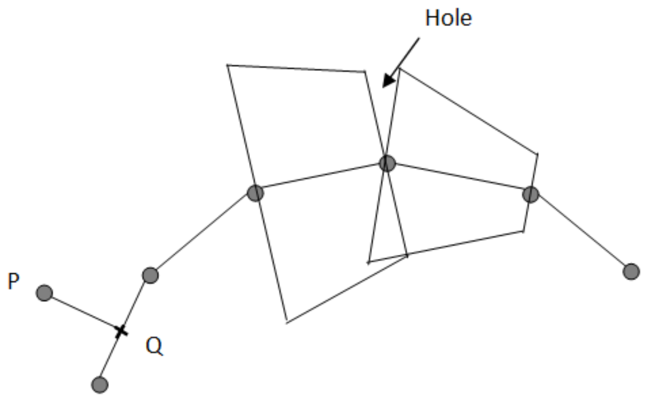

Figure 4.9: *Calculation of distance using discretized action curve and its problem.*

In figure 4.9, a voxel at point P can be classified as inside or outside by first find the segment that contains its projection Q. If the distance between P and Q is smaller than the shape radius at Q then P is inside, otherwise it is outside. However, this strategy will create holes, as depicted in figure 4.9, because the voxels in the hole areas do not have their projects on any segment. To solve this situation, we choose to overlap the discretization with another one to cover the hole areas. The arrangement of the two discretizations are shown in figure 4.10. The holes created by the first discretization will be covered by the second discretization of the curve.

4.2.2 Parameters passing to GPU

Our fire animation utilize the GPU to speed up the deformations of thousands of fire particles. For each particle, the deformation is controlled by several parameters such as the life of the particles, the deformation frequency, the positions of the particle on the emitter (Shijun, 2010), the fluid velocity at the particle position

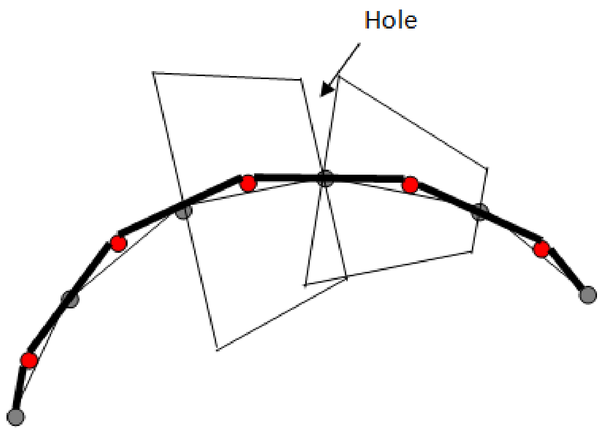

Figure 4.10: *Two overlapped discretization. The second discretization is shown in thick lines connecting the red dots.*

and the advected position calculated from the fluid field. Since each particle is a tear drop triangular mesh with many vertices, passing these parameters along with each of the vertex will be expensive and inefficient. Instead, we chose to pass the parameters separately from the vertices using DirectX instance stream and pass the vertices geometric information in a vertex stream. With proper vertex definition, the GPU hardware will combine these two streams and pass to the vertex and fragment shader.

Our original vertex definition in C++ is as following:

```
const D3DVERTEXELEMENT9 TEARDROPVERTEX::Decl[] =
{
 { 0, 0,  D3DDECLTYPE_FLOAT4, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_POSITION, 0},
 { 0, 16, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_NORMAL,   0},
 { 0, 28, D3DDECLTYPE_FLOAT4, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TEXCOORD, 0},
 { 1, 0,  D3DDECLTYPE_FLOAT4, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TEXCOORD, 1},
 { 1, 16, D3DDECLTYPE_FLOAT4, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TEXCOORD, 2},
 { 1, 32, D3DDECLTYPE_FLOAT4, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TEXCOORD, 3},
 { 1, 48, D3DDECLTYPE_FLOAT4, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TEXCOORD, 4},
 { 1, 64, D3DDECLTYPE_FLOAT4, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TEXCOORD, 5},
 { 1, 80, D3DDECLTYPE_FLOAT4, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TEXCOORD, 6},
 D3DDECL_END()
};
```

The first three fields of the definition are vertex based data (stream 0) and the rest is instance based data (stream 1). With this vertex definition, we were able

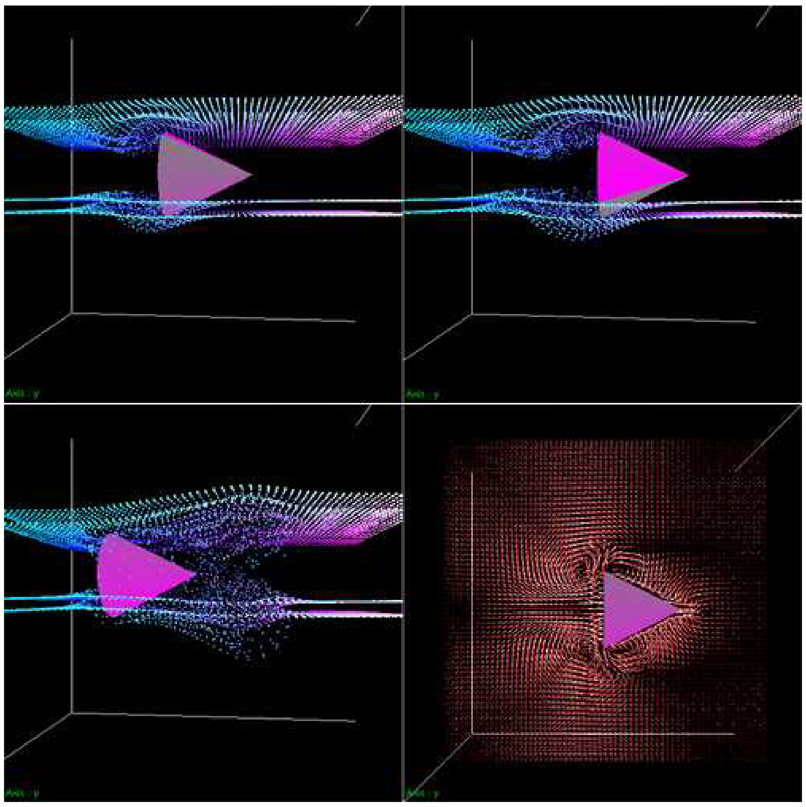
Figure 4.11: *A virtual movement of the cone inside the fluid volume*

to reduce significantly the amount of data passed to the GPU.

4.3 Fluid simulation and general semi-rigid object integrations

To illustrate that our method can use with any type of objects and movements we have introduced several submergence of novel objects into the fluid volume. Since the verification of erratic free flow simulation is generally not numerically achievable we choose to compare our simulation with one real life common example to show the viability of our simulation.

In figure 4.11, we show a cone moved in the fluid volume. It is generally difficult to

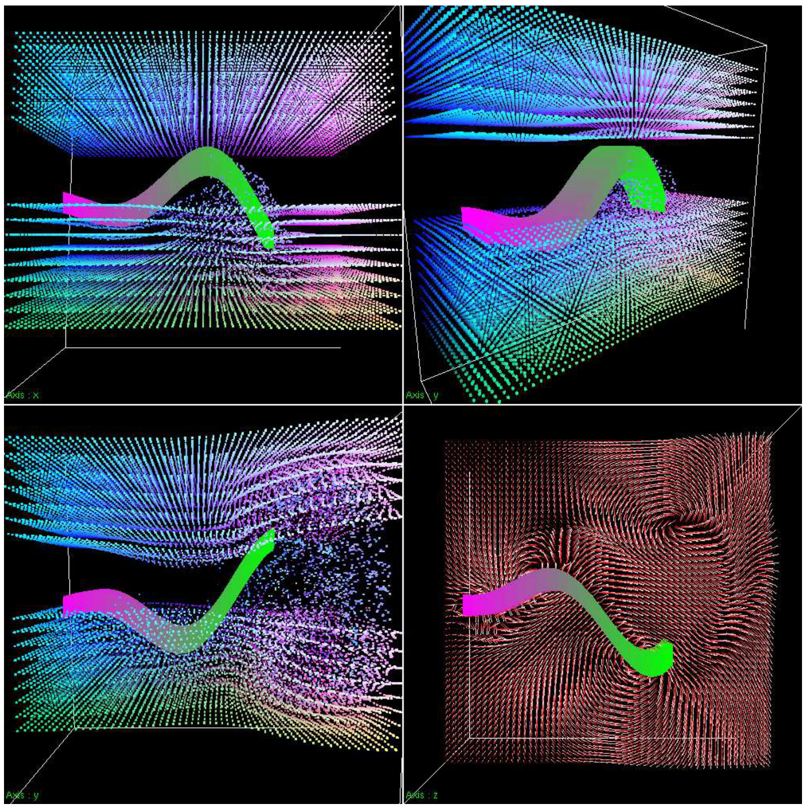

Figure 4.12: *A virtual snake move inside the fluid volume*

setup this kind of movement and capture it, hence our method have the potential of replacing real capture facility in this kind of novel object movement submerged in a fluid volume. In this example, the colourful markers are massless particles whose movements are exactly identical to the fluid and is animated with the velocity field. Their purpose is to give the visualization of the real fluid. The bottom right of the figure 4.11 we show the velocity field. In this picture we can see the swirling and bending of the fluid layer under the effect of the cone's movement.

In figure 4.12 we introduced a virtual snake which is almost impossible to capture in real life. The snake move within the fluid volume and splash the fluid around its movement. To illustrate the viability of our method, we compare with a real life example. In top part of figure 4.13 we show a stream of fluid flow pass a stationary cube in a similar fashion as the real life example in the bottom part. We compare the wake region in both pictures and see that there are common vortice patterns in the fluid flow. Vortices are important features of fluid flow. Although the two pictures are not exactly the same due to different experiment conditions, the common pattern has shown that our method has its potential and its application is possible, at least in the field of computer graphics where speed and visualization are favoured over exact physical correctness.

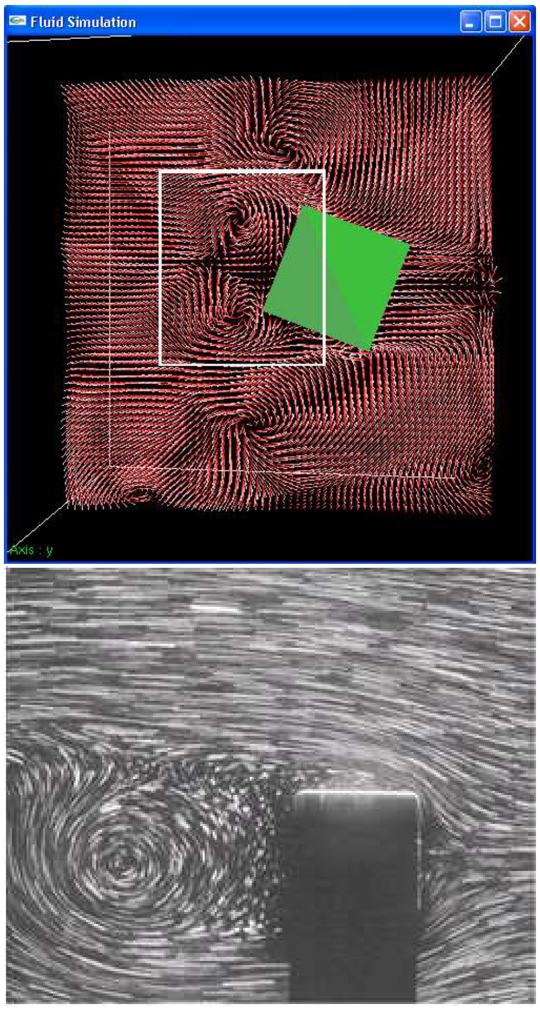

Figure 4.13: *A comparison between simulation (TOP) and real life example (BOTTOM) of a fluid flow passing by a black cube.*

Chapter 5

Conclusions and Discussions

In this paper, we have presented a method to apply Semi-Lagrangian fluid simulation technique developed for computer graphics to visualization of near body hydrodynamics of a submerged swimmer or any other submerged semi-rigid 3D object. Our method is general and fast and can be applied in many applications. Visualizing nearbody hydrodynamics characteristics is the focus of our work. The visualizations are meaningful and potential aids for traditional facilities required in coaching activities to analyze the performance of swimmers.

Optimal human performance, however, is of primary interests to the fields of biomechanics, sports and exercise science; the limits of human performance are of primary concern in the field of physiology; the basis of human behavior is fundamental drive in neuro-psychology. Indeed, the science of human movements is multi-faceted, multi-disciplinary, and the cross-applicability of techniques has become an increasingly common trend.

The thrust of this work involves applying a visual-driven approximation of a highly-nonlinear phenomenon to the biomechanical visualization of swimming dynamics. The suggestion of crossing a technique that was originally developed for computer graphics effects and games, with an application which many would regard a more serious branch of science require considerable persuasion and justification on our part.

There are several points of view in which the approach may be deemed acceptable, and we categorically list them as follows:

- The current state of classical CFD, though more rigorous and analytical by nature, is nowhere near an ablility to render simulations of such complexity.
- Current general-purpose fluid dynamics software are highly-specialized, difficult to use, and several orders of magnitude higher in computational costs.
- The computational cost typical of classical CFD methods largely overwhelms the facilities available to a typical institution, consumer group, or an individual (e.g.a swimming coach). It is hard to imagine that such software can be adopted by a wider audience any time soon.
- Dissipation errors are more severe at a distance away from the body. Here, we are only concerned with the near-body dynamics which lessens the effects of dissipation errors. Thus, our method is more applicable in daily life analysis setting such as in improving swimmers' performances and not in engineering-based analysis which requires accurate simulation in any part of the fluid flow. The domains of our method and engineering based methods are, therefore, different.
- Empirical plots of near-body quantities, such as the integral of surface pressure, converges with respect refinements of simulation parameters. This suggests possibilities of fine-tuning the simulation parameters to obtain values that are acceptable to various degrees of user tolerance.

5.1 Publication

Part of the work in this thesis was published in Pacific Graphics 2007 (Truong et al., 2007). The near body interaction between fire and the surrounding fluid has been included in a fire simulation paper that is currently under submission to The Visual Computer.

Bibliography

O. E. Arash, O. Gnevaux, A. Habibi, and J. michel Dischler. Simulating fluid-solid interaction. In *in Graphics Interface*, pages 31–38, 2003.

J. A. Baerentzen and H. Aanaes. Signed distance computation using the angle weighted pseudonormal. *IEEE Transactions on Visualization and Computer Graphics*, 11:243–253, May 2005. ISSN 1077-2626.

R. Bridson and M. Müller-Fischer. Fluid simulation: Siggraph 2007 course notes. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, pages 1–81, New York, NY, USA, 2007. ACM.

M. Carlson, P. J. Mucha, and G. Turk. Rigid fluid: animating the interplay between rigid bodies and fluid. *ACM Trans. Graph.*, 23:377–384, August 2004. ISSN 0730-0301.

J. X. Chen, N. da Vitoria Lobo, C. E. Hughes, and J. M. Moshell. Real-time fluid simulation in a dynamic virtual environment, 1997.

N. Chentanez, T. G. Goktekin, B. E. Feldman, and J. F. O'Brien. Simultaneous coupling of fluids and deformable bodies. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '06, pages 83–89, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

D. Enright, S. Marschner, and R. Fedkiw. Animation and rendering of complex water surfaces. *ACM Trans. Graph.*, 21:736–744, July 2002. ISSN 0730-0301.

J. H. Ferziger and M. Peric. *Computational Methods for Fluid Dynamics.* Springer, Berlin, 1999.

A. W. Fitzgibbon. Simultaneous linear estimation of multiple view geometry and lens distortion, 2001.

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

N. Foster and D. Metaxas. Modeling the motion of a hot, turbulent gas. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 181–188, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

M. N. Gamito, P. F. Lopes, and M. R. Gomes. Two-dimensional simulation of gaseous phenomena using vortex particles. In *In Proceedings of the 6th Eurographics Workshop on Computer Animation and Simulation*, pages 3–15. Springer-Verlag, 1995.

T. Harada, M. Tanaka, S. Koshizuka, and Y. Kawaguchi. Realtime coupling of fluid and rigid bodies. 2007.

J. Hyman. Stroke science. 2004.

L. Keng Siang and G. Ashraf. Simplified muscle dynamics for appealing real time skin deformation. 2007.

B. M. Klingner, B. E. Feldman, N. Chentanez, and J. F. O'Brien. Fluid animation with dynamic meshes. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 820–825, New York, NY, USA, 2006. ACM.

E. Kreighbaum and K. Barthels. *Biomechanics: a qualitative approach for studying human movement.* Allyn and Bacon, 1996.

E. W. Maglischo. *Swimming fastest.* Human Kinetics Publishers, 3 edition, 2003.

R. Mittal. Computational modeling in biohydrodynamics:trends, challenges, and recent advances. 2004.

R. Mittal, F. Najjar, R. Byrganhalli, V. Seshadri, and H. Singh. Fluid mechanics. 2004.

M. Müller, S. Schirm, M. Teschner, B. Heidelberger, and M. Gross. Interaction of fluids with deformable solids: Research articles. *Comput. Animat. Virtual Worlds*, 15:159–171, July 2004. ISSN 1546-4261.

C. Negrao. Conflation of computational fluid dynamics and building thermal simulation, 1995.

W. T. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, 2:359–376, 1983.

H. Shijun. Amorphous effect in characters, 2010.

K. Sims. Particle animation and rendering using data parallel computation. In *Computer Graphics*, pages 405–413, 1990.

J. Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

J. Stam. Real-time fluid dynamics for games, 2003.

J. Stam and F. Eugene. Turbulent wind fields for gaseous phenomena. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 369–376, New York, NY, USA, 1993. ACM.

D.-T. Truong, Y.-Y. Chow, and A. Fang. Visualization and simulation of near-body hydrodynamics using the semi-lagrangian fluid simulation method. In *Computer Graphics and Applications, 2007. PG '07. 15th Pacific Conference on*, pages 219 –228, 29 2007-nov. 2 2007.

X. C. Wang and C. Phillips. Multi-weight enveloping: least-squares approximation techniques for skin animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '02, pages 129–138, New York, NY, USA, 2002. ACM.

L. Yaeger, C. Upson, and R. Myers. Combining physical and visual simulation, creation of the planet jupiter for the film 2010. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '86, pages 85–93, New York, NY, USA, 1986. ACM.