uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Javier Adolfo Mora-Cano
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.C.S.
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Hapto-Visual Representation of Three Dimensional Incompressible Flows

TITRE DE LA THÈSE / TITLE OF THESIS

Wonsook Lee
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Ali Arya

Robert Laganière

Gary W. Slater
Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Hapto-Visual Representation of Three Dimensional Incompressible Flows

Javier Mora

Thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

In partial fulfillment of the requirements for the degree

Master of Computer Science

Ottawa-Carleton Institute for Computer Science

School of Information Technology and Engineering

University of Ottawa

Ottawa, Ontario, Canada

January 2008

Your file Votre référence
ISBN: 978-0-494-41675-4
Our file Notre référence
ISBN: 978-0-494-41675-4

Abstract

Imagine a videogame in which you impersonate a wizard who needs to create a potion by stirring substances in a cauldron. Through a desktop haptic probe, you are able to stir and feel how the fluid changes its viscosity, velocity and other properties. So far solid or deformable objects have been experimented for haptic-tactile feedback. In this thesis we innovate by devising techniques that enable the haptical rendering of shape-less objects, such as fluids. We achieved the real-time 3D fluid simulation of multiple substances based on the Navier-Stokes equation and coupled it with a discreet mass-spring particle system representing its deformable surface. We overcame the challenges that arise during the integration of both haptics and graphics workspaces, the free-view visualization of 3D fluid volume, and the rendering of haptic forces. Our system is flexible to accommodate different kinds of fluids, such as liquid and smoke, to be co-simulated.

Acknowledgements

Numerous people deserve recognition for their parts in my Master's studies. First of all, I would like to express special thanks to my thesis supervisor, Prof. Won-Sook Lee, for her guidance, support and encouragement to complete my master studies. I extend my thanks to my research colleagues at the DISCOVER LAB for their cooperation and discussion during this work. *The University of Ottawa* provided me with funding during my master studies, for which I am also very grateful and appreciative. Special thanks go to my family and friends, for their endless understanding, encouragement and support. I would also like to thank *Fuel Industries*, for offering me a flexible schedule that let me balance my professional and academic life.

Table of Contents

List of Figures

List of Tables

Glossary of Terms

2D Two Dimensional

3D Three Dimensional

DOF Degrees of Freedom

HIP Haptic Interface Point

HLA High Level Architecture

PC Personal Computer

VR Virtual Reality

FPS Frames per Second

I/O Input and Output

Chapter 1 Introduction

1.1 Motivation

A fluid, such as liquid and gas, is a substance which deforms continuously under a shear stress. Examples of fluid phenomena are gas, smoke, wind, ocean waves, waves induced by ships or simply pouring of a glass of water. As simple and ordinary these phenomena may seem, as complex and difficult it is to simulate them.

Automating the animation of fluids is important as it would be extremely hard to animate fluids by hand due to the facts that surface is changing very quickly and it contains lots of small details. The reason for the complexity of fluid behavior is the complex interplay of various phenomena such as convection, diffusion, turbulence and surface tension. Animating fluids is extremely complex and CPU intensive animation technique available. With the computational power of next generation game consoles and PC's more and more physical effects – so far only seen in feature films – are now entering the realm of real-time applications such as three-dimensional computer games. There is, however, still a big gap between pre-computing a physical effect for many hours on a farm of workstations and simulating the effect in real-time, typically at 30 _ 60 frames per second on a single console.

Meanwhile, *Haptics*, which refers to the technology which stimulates the users' sense of touch, has been increasing in popularity because of the powerful enhancements that it brings to the human-computer interaction experience. Haptics allow users to literally touch and feel characteristics about computer-generated objects such as texture, roughness, viscosity, elasticity, and many other properties. The human tactile and kinesthetic senses are stimulated through computer-controlled forces which convey to the users a sense of natural

feel about a virtual or remote environment. Such interaction exposes characteristics about the object which cannot be easily purveyed using graphical or acoustic displays. Unlike these other graphical or acoustic displays, haptic devices offer a bi-directional human-machine interface, providing both input and output for other senses as well. As outlined by Hayward et al. [HAC*04], a distinguishing feature of haptic interfaces is that they respond to human gestures by generating tactile and kinesthetic cues creating a simultaneous exchange of information between the machine and user.

So far solid or deformable objects have been experimented for haptic-tactile feedback. Our motivation is to produce a system that brings human-computer interaction to real-time fluid animations, so that users can appreciate and feel the properties of a fluid simulation via a haptic interface (see Figure 1). Our haptic interaction with a pool of water is the world first experiment. This integration could benefit a wide-spread of applications in the following areas.

TRAINING OF TECHNICAL EQUIPMENT

In combination with audio and video displays, haptics technology may be used to train people for tasks requiring hand-eye coordination, such as ship docking maneuvers. In the scope of our research, hapto-fluid integration may be used to guide in virtual fishing tasks, or paddling through a small current flow. It may also be used as assistive technology for the blind or visually impaired.

APPLIED SCIENCE RESEARCH

Hapto-fluid integration may be used to aid in testing fluid dynamics. Some science fields such as Astronomy, Chemical Engineering, Oil Engineering, and Fluid Mechanics could benefit from this research. Astronomers could modify and test different fluid simulation parameters such as gravity changes from various scenarios. Engineers could

model different reactions that could happen when mixing substances of different chemical properties. Oil Engineers could study and physically feel the viscosity of oils, among others.

MEDICINE APPLICATIONS

Haptics allows surgeons, for instance, to feel realistic touch sensation when training to perform laparoscopic surgery [SRI00]. The Laparoscopy VRTM [IMM07] is a hapto-visual display designed especially for this task. Incorporating haptics as part of surgical simulation is an application of notable importance because it has been shown to increase the overall success rate of surgeons during training [HWS02]. The touch models used by these applications need to be able to accurately purvey tactile and kinesthetic information to the surgeon about the medical probe as it comes in contact with virtual human organs (e.g., during slicing, cutting, suturing, puncturing, poking or interfering with a blood flow). Otherwise, the surgeon may apply too much force, causing tissue trauma, or too little force, failing to perform a procedure correctly. In the scope of our research, hapto-fluid integration may be used to analyze the motion of hearts ventricles, or to simulate the blood flow in a patient's cardiovascular system to enhance the realism of the haptic training.

THE VIDEOGAME INDUSTRY

The videogame industry can also serve from this technology [GSK07][AML*06][MNS04]. Haptics would allow players to feel the physical properties of in-game objects, adding an extra level of interaction that traditional interface devices do not offer. Nintendo's recent Wii™ games [NINTENDO] are an example of the industry's interest for higher interactive applications. HaptiCast [AML*06], a haptic game developed by the thesis author and colleagues, investigated the use of haptic metaphors in 3D first-person shooter style video games. Players are given a series of haptically enabled wands which offer various methods of interacting with the game world. The *lift* wand allows a player to pick up and feel the weight of any object in the game world. The *lob* wand acts like a haptic

slingshot, and the *blast/bolt* wands shoot fiery recoil blasts. Response to the game was extremely positive from users who played the game, including Professor Perlin [PER85], who commented on the potential to see haptic technology integrated into future videogames. Haptics can increase the realism of game simulations and thus increase their overall entertainment value.

In the scope of our research, hapto-fluid integration may be used to enable game scenarios that involve fluids. For instance, a witch flying a broom on a stormy sky could be exposed to wind flow. A wizard stirring a cauldron could feel the viscosity of the mixture while creating a magic potion.

1.2 Problem Statement

This thesis addresses the problem of developing a system that brings human-computer interaction to real-time fluid animations, so that users can *see* and *feel* the simulation properties of a fluid, a shape-less object, via a haptic interface. In order to solve this problem, we focus on two main issues:

- Real-time fluid animation: how to stably represent a fluid simulation in real-time and render it on the screen at an acceptable frame rate of approximately 30 frames per second.
- Haptic interaction with the fluid: how to haptically render the simulated shape-less fluid to the user. The haptic probe must also interact with the fluid surface and be able to modify the current flow generated by the simulation.

In addition, we also discuss

- Haptic gesture recognition as an interactive application for haptic games.

1.3 Proposed Solution

This section briefs the solution to the problem stated in the previous section. The proposed solution consists of the following techniques:

- Use computational fluid dynamics to model a three-dimensional fluid flow.
- Couple the fluid with a deformable surface which deforms with the haptic touch.
- Use the output of the fluid simulation to represent a variety of haptic feedback modes such as the rendering of surface tension, viscosity, and current flow. In viceversa, the device inserts back forces that modify the fluid flow.

More specifically, our solution includes the following more technical approaches:

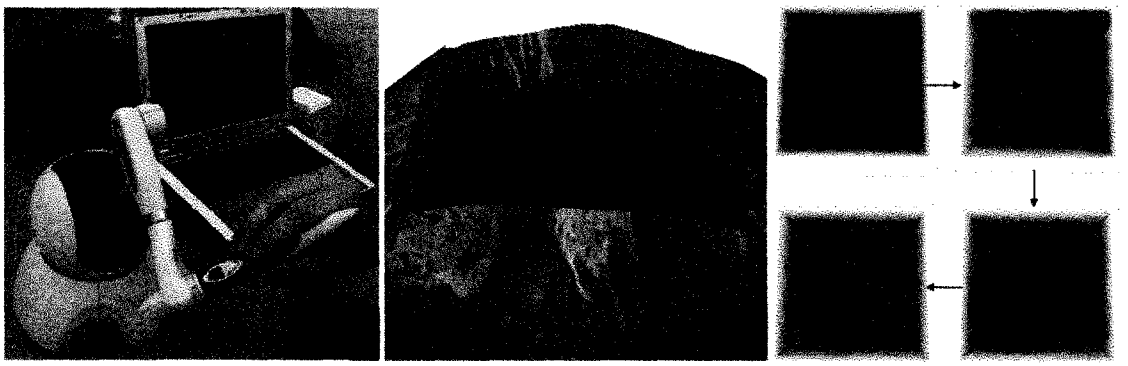
- Match haptic and graphic coordinates through a mapping function to achieve the coordination of hapto-visual I/O and the maximization of the workspace area.
- Use a three-dimensional grid structure to represent the state of the fluid simulation and model the flow motion based on the Navier-Stokes equations.
- Calculate forces that originate from the flow equations of the pool of fluid and render them to the haptic device.
- Use a deformable and discreet mass-spring particle system to represent the fluid on the surface. The surface deforms with the touch of the haptic probe and propagates the applied deformation to the fluid represented on the 3D grid.
- To achieve volume rendering, slice the 3D grid into planes that are perpendicular to the line of camera sight. Apply gradual alpha transparency to the slices and sort them based on their distance to the camera. Continuously construct a 3D texture based on the fluid simulation data, and proceed to texture the slices with this information.

- Based on the workspace boundaries, the surface deformation, the velocity field and density values of the fluid simulation, render a set of haptic effects in order to achieve the kinesthetic feeling of a fluid in motion.

Publications based on the work from this thesis appear at the end of this document.

1.4 Document Organization and System Overview

Figure 2 shows the high level architecture (HLA) for our system. It displays the two main stages of our system and illustrates the data flow for generating hapto-visual 3D flows from the fluid simulation. These stages are namely the *fluid processing*, and the *hapto-visual rendering of the simulation*. The fluid processing stage computes the fluid simulation and the grid deformation at each time-step. The hapto-visual rendering stage consists of presenting the output to the user via the haptic device, through haptic rendering, and via the monitor, through graphic rendering. Detailed discussion of these stages and their components occur in the ensuing chapters.



(a) (b) (c)

Figure 1: (a) 3D fluid interaction with haptic probe with force feedback in laptop environment. (b) Deforming the fluid surface and generating surface tension force feedback. (c) Side view of the inner fluid simulation with the green haptic probe inside. A red and a green substance are added into the simulation and the user stirs with the haptic device until they mix together in the fluid.

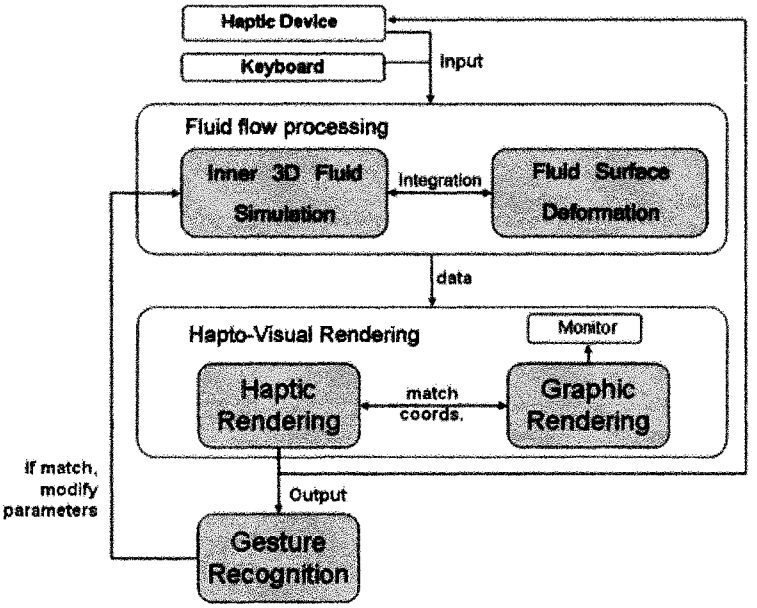


Figure 2: HLA of our system.

In Chapter 2 of this thesis we present a literature review on haptics as well as on real-time fluid animations. In Chapter 3 we present the fluid processing stage of the system, which includes the solution of the flow equations, a description of the surface deformation process and the integration between graphics and haptics workspaces. Chapter 4 discloses the hapto-visual rendering algorithm along with the presentation of the haptic force calculations. As an application in computer games, a haptic gesture recognition module is described in chapter 5. We present conclusions in chapter 6.

Chapter 2 State of the Art

In this chapter, we discuss literature review on fluid simulation and haptic rendering. Section 2.1 introduces some of the approaches to model fluids and section 2.2 describes techniques to render them on the screen. Section 2.3 presents previous research on haptic rendering. Finally, section 2.4 discloses previous work on Haptic-Fluid integration and how it compares with our approach.

2.1 Fluid Modeling

A fluid modeling approach depends on the specific application. For example when simulating a fountain, a waterfall, or a jet of water then some sort of particle system would be the most efficient implementation, while when displaying a flame or smoke with more flow realism we would like to use fluid flow tracking approaches, such as Computational Fluid Dynamics (CFD).

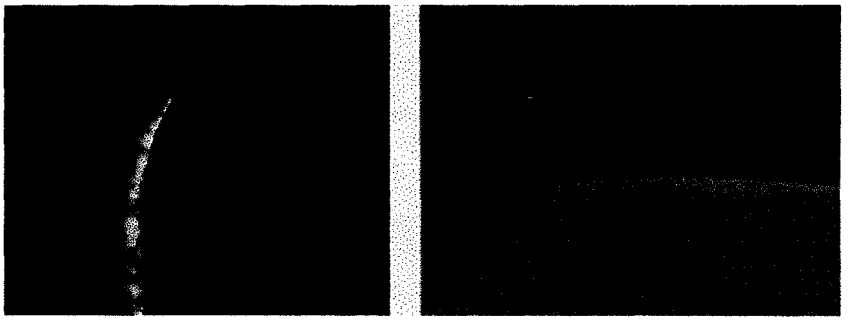


Figure 3: The "genesis effect" particle system from Star Trek II [REE83].

Particle systems, such as the one shown in Figure 3, simulate certain fuzzy phenomena which are controlled by an emitter. The emitter acts as the source of the particles, and its location in 3D space determines where they are generated. The emitter has

attached to it a set of particle behavior parameters. These parameters can include the spawning rate (how many particles are generated per unit of time), the particles' initial velocity vector (the direction they are emitted upon creation), particle lifetime (the length of time each individual particle exists before disappearing), particle color, and many more. These systems don't have smooth well-defined surfaces and are non-rigid objects [REE83].

Computational Fluid Dynamics [LL75], on the other hand, is a computer-based mathematical modeling tool that incorporates the solution of the fundamental equations of fluid flow, the Navier-Stokes equations, and other allied equations. Computers are used to perform the millions of calculations required to simulate the interaction of fluids and gases with the complex surfaces used in engineering. However, even with simplified equations and high-speed supercomputers, only approximate solutions can be achieved in many cases.

The following sections present different approaches to model fluids. If a volumetric fluid flow needs to be tracked in time, then CFD approaches presented in section 2.1.1 may be best suited to target the problem. On the other hand, if the interest is to directly simulate the physical effect of fluids or their surfaces instead of simulating the causes of it, other approaches presented in section 2.1.2 may be used. Some of the approaches presented in both sections may be combined together to offer more accurate simulations.

2.1.1 Fluid Flow Tracking Approaches

When volumetric fluids are in movement, they generate fluid flows which may be perceived as vector fields. Computational fluid dynamics (CFD) is one of the branches of fluid mechanics that uses numerical methods and algorithms to solve and analyze problems that involve fluid flows. The fundamental bases of any CFD problem are the Navier-Stokes

equations, which define any single-phase fluid flow. When we think about a continuum (like a fluid or a deformable solid) moving, there are two main approaches to track this motion, the Lagrangian viewpoint and the Eulerian viewpoint. Both viewpoints are useful when trying to solve the Navier-Stokes equations. The next sub-sections explain each of these viewpoints, the theory behind the fluid equations, and a review of related work in this area.

2.1.1.1 Navier-Stokes Equation

The Navier-Stokes differential equations are a set of partial differential equations (PDEs), which describe the motion of viscous incompressible fluids. This means that all properties of the fluid are described exclusively by its viscosity and density. These equations are based on the assumption that fluids can be described as a collection of particles. The terms describe the forces acting on a particle, derived by observing the behavior in a unit cube around the particle. This is done under the assumption that the fluid inside this unit cube behaves uniformly. To better understand the tracking of flow motion, we now present a more comprehensive definition of the Navier-Stokes equations.

The Navier-Stokes equations were derived from Newton's second law of motion (see Figure 4), which states that:

$$\vec{f} = m \cdot \vec{a}$$

Figure 4: Newton's Second Law of Motion

where m is mass, $\vec{a}$ is acceleration, and $\vec{f}$ is force. They describe the changes in a velocity field, i.e. the acceleration of the fluid, as a sum of the forces acting on the fluid – including forces introduced by the fluids own movement. In a compact vector notation the Navier-Stokes equations are presented as:

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

$$\nabla \cdot u = 0. \qquad (2)$$

The four terms on the right-hand side of the Navier-Stokes equation represent accelerations. The first term represents external forces, the second term is advection, the third is diffusion, and the last one is pressure. In these equations t is time, $\vec{u}$ is the velocity, ρ is the density of the fluid, $\vec{f}$ is the force field, ν is the kinematic viscosity of the fluid, ∇ is the gradient operator and p is pressure. We will examine each of them in turn.

External Forces

The first term in the Navier-Stokes equations represents external forces and is given by the following expression:

$$\frac{1}{\rho}\vec{f} = \frac{1}{\rho}(f^x, f^y, f^z)^T \qquad (3)$$

where the force field $\vec{f} = (f^x, f^y, f^z)^T$ is the sum of all external forces working on the fluid, and ρ is the density of the fluid, which describes the mass of a unit cube of fluid. Because $\vec{u}$ is a vector quantity, there are four equations and four unknowns: *u, v, w,* and *p.*

Advection

The second term in the Navier-Stokes equations represents advection and represents the force of the *fluid motion working on itself.* This can be thought of as the molecules in a fluid bouncing into each other. If one molecule bumps into another molecule, the other molecule is affected and will start moving. The contribution of advection is described by:

$$-(\vec{u}\cdot\nabla)\vec{u} = -\begin{pmatrix} u^x\frac{\partial u^x}{\partial x} & u^y\frac{\partial u^x}{\partial y} & u^z\frac{\partial u^x}{\partial z} \\ u^x\frac{\partial u^y}{\partial x} & u^y\frac{\partial u^y}{\partial y} & u^z\frac{\partial u^y}{\partial z} \\ u^x\frac{\partial u^z}{\partial x} & u^y\frac{\partial u^z}{\partial y} & u^z\frac{\partial u^z}{\partial z} \end{pmatrix} \qquad (4)$$

where ∇ is the gradient operator, and $\vec{u} = (u^x, u^y, u^z)^T$ is the velocity.

Diffusion

Diffusion occurs when part of the fluid passes by an obstacle, or another part of the fluid with a different velocity. The fluid is slowed down and vortices appear. The contribution of diffusion is described by the term:

$$\nu\nabla^2\vec{u} = \nu\begin{pmatrix} \frac{\partial^2 u^x}{\partial x^2} + \frac{\partial^2 u^x}{\partial y^2} + \frac{\partial^2 u^x}{\partial z^2} \\ \frac{\partial^2 u^y}{\partial x^2} + \frac{\partial^2 u^y}{\partial y^2} + \frac{\partial^2 u^y}{\partial z^2} \\ \frac{\partial^2 u^z}{\partial x^2} + \frac{\partial^2 u^z}{\partial y^2} + \frac{\partial^2 u^z}{\partial z^2} \end{pmatrix} \qquad (5)$$

where ν is the kinematic viscosity of the fluid, which describes the "thickness" of the fluid. Hence, diffusion is also referred to as the *effect of viscosity*. Some fluids are "thicker" than others. For example, molasses and maple syrup flow slowly, but alcohol flows quickly. We say that thick fluids have a high viscosity. Viscosity is a measure of how resistive a fluid is to flow. This resistance results in diffusion of the momentum (and therefore velocity), so the third term is called the diffusion term.

Pressure

Fluid moving in and out of the observed unit cube causes the pressure to change. Differences in pressure between the unit cube and its surroundings affect the velocity as described given by

$$-\frac{1}{\rho}\nabla\vec{p} = -\frac{1}{\rho}(\frac{\partial p}{\partial x},\frac{\partial p}{\partial y},\frac{\partial p}{\partial z})^T, \qquad (6)$$

where ρ is still the density of the fluid and $\vec{p}$ is the pressure. Because the molecules of a fluid can move around each other, they tend to "squish" and "slosh". When force is applied to a fluid, it does not instantly propagate through the entire volume. Instead, the molecules close to the force push on those farther away, and pressure builds up. Because pressure is force per unit area, any pressure in the fluid naturally leads to acceleration. (Think of Newton's second law, $\vec{f} = m \cdot \vec{a}$). The fourth term, called the pressure term, represents this acceleration.

Incompressibility

If the relative speeds within a flow are low enough, thermodynamic effects and density changes due to changes in pressure become negligible. If density is constant and mass is conserved so is volume. Essentially what goes into a differential volume must exit it simultaneously. Coupling this equation with conservation of momentum makes the system fully determined, without need of the energy equation or an equation of state, and yields extremely efficient simulations. This condition is expressed mathematically as the divergence of velocity is zero ($\nabla \cdot u = 0$).

There have been two main simulation approaches to track the fluid motion described by the Navier-Stokes equations: the Lagrangian viewpoint and the Eulerian viewpoint.

2.1.1.2 Lagrangian Schemes

The Lagrangian approach [LL75] (named after the French mathematician Lagrange) treats the continuum just like a particle system (see Figure 5). Particle systems are

essentially a mathematical formalism used to describe phenomena that are complex, dynamic and highly parallel with small individual components. Particle systems are context insensitive, meaning they can be used to model very different situations. The particle system is a tool, but does not imply a specific use-case in itself. By changing some of its inner components, it can be suited to many problems.

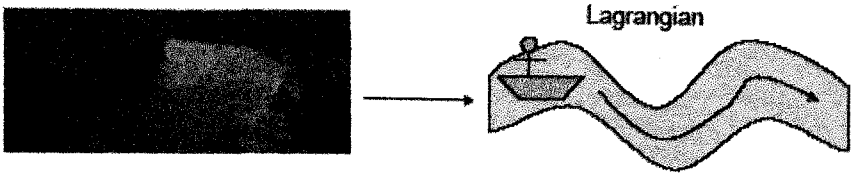


Figure 5: In a Lagrangian approach, we follow a parcel of fluid as it moves. An example would be to measure water temperatures from a moving boat.

In the Lagrangian approach, each point in the fluid or solid is labeled as a separate particle, with a position ~x and a velocity ~u. One could even think of each particle as being one molecule of the fluid. Solids are almost always simulated in a Lagrangian way, with a discrete set of particles usually connected up in a mesh. Examples include work by Muller et al. [MCG03] and Clavet et al. [CBP05] based on Smoothed Particle Hydrodynamics [MON92][OH95]. Reconstructing a smooth surface from the particles remains challenging, though recent work on point based level sets by Corbett [COR05] is promising. Vortex particle and vortex filament methods [AN05][ANS*06] for the simulation of smoke also fall into this category.

2.1.1.3 Eulerian Grid-based Approach

The Eulerian approach [LL75] (named after the Swiss mathematician Euler) takes a different tactic, that's usually used for fluids. Instead of tracking each particle, we instead

look at fixed points in space and see how the fluid quantities (such as density, velocity, temperature, etc.) measured at those points change in time (see Figure 6).

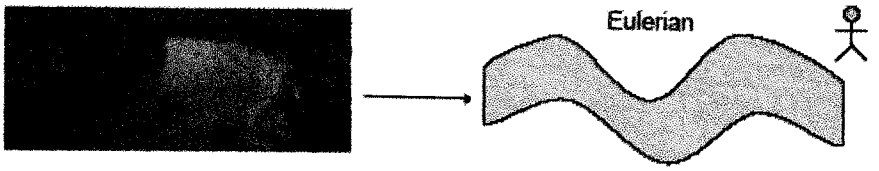


Figure 6: An Eulerian approach looks at fixed points in space and see how the fluid quantities measured at those points change in time.

This approach uses discrete function values throughout the computational domain, on a regular volumetric grid. Foster and Metaxas [FM96] began the current research thrust in Eulerian fluid simulations for animation. Their method has been used for liquid, smoke [FSJ01] and sand simulation [ZB05]. Their technique has been steadily improved with advancements in implicit surface tracking [EMF02][FF01], unconditionally stable time integration [STA99], adaptive grid construction [LGF04], and vorticity preservation [SRF05]. For high quality smooth surfaces, Eulerian methods using level sets are typically used. Here the surface is implicitly represented by the zero level set of a surface function, which is usually the signed distance from the surface. This surface function is then advected by the velocity field defined on the grid (often using particles to avoid mass conservation errors in advection [EMF02][BFA02][HNB*06]). The main drawback of level set implicit surfaces is their susceptibility to numerical dissipation, and inability to maintain areas of high curvature.

2.1.1.4 Literature Review on Flow Tracking Approaches

The following chronologically summarizes a variety of publications that have contributed to the field by using eulerian, lagrangian and hybrid approaches. If we

chronologically look at the evolution of pioneering research in the computational modeling of incompressible flows, we can go back to the 1950's. The primitive variable approach favored in computer graphics was pioneered by Harlow and Welch who developed the Marker and Cell method (MAC) in 1965 [HW65], Chorin who developed the artificial compressibility method in 1967 [CHO67], Patankar and Spalding who developed the semi-implicit method for pressure linked equations (SIMPLE) in 1972 [PS72], and Issa who developed the pressure implicit with splitting of operators (PISA) method in 1985 [ISS85]. The computer graphics industry has primarily focused on the work of Harlow and Welch and the use of MAC grids.

Foster and Metaxas [FM96] describe one of the first methods for simulating the full 3D Navier-Stokes equations for computer graphics. The method is based on explicit methods and only stable for small time steps. In fact, the time step must uphold the *Courant-Friedrich-Levy (CFL)* condition [CFL67]. Foster and Metaxas [FM97] give an extended version of the 1996 algorithm including the forces of thermal buoyancy. This means that temperature is represented in the centre of each grid cell, defining a discrete temperature field equivalently to pressure. The temperature field is advected and diffused using the same method as with velocity. Stam [STA99] extends Foster and Metaxas [FM96] with the aim of making the solver unconditionally stable. This method is further explained in Chapter 3 of this thesis. Fedkiw et al [FSJ01] give a method for simulating smoke based on Stam's method. Under the assumption that the effects of viscosity are negligible in gases, when simulated on a coarse grid, the diffusion term of the Navier-Stokes equations are left out, leaving the incompressible Euler equations:

$$\frac{\partial \vec{u}}{\partial t} = -(\vec{u} \cdot \nabla)\vec{u} - \nabla p + \vec{f} \qquad (7)$$

$$\nabla \cdot \vec{u} = 0, \qquad (8)$$

where $\vec{u}$ is the velocity, p is the pressure, and f is the external forces. In order to make up for the coarse grid representations, *vorticity confinement* is used to add the small scale detail that has been damped out by numerical dissipation back into the velocity field. This is further explained in the following chapter.

Stam [STA03] presents a simple and rapid implementation of the Stam [STA99] fluid dynamics solver for game engines with the Navier-Stokes equations. The algorithm presented in Stam's papers presents only 2D implementation. In this thesis, we expand his implementation to 3D so that we can integrate it with a 3D haptic interface.

Losasso et al. [LGF04] present a method for simulating water and smoke on an unrestricted octree data structure exploiting mesh refinement techniques to capture the small scale visual detail. The paper proposes a new technique for discretizing the Poisson equation on this octree grid with fast solution methods such as preconditioned conjugate gradients. Harris [HAR03] presents a physically-based, visually-realistic cloud simulation suitable for interactive applications such as flight simulators. Clouds in the system are modeled using partial differential equations describing fluid motion, thermodynamic processes, buoyant forces, and water phase transitions.

Guendelman et al. [GSL*05] present a novel method for solid/fluid coupling that can treat infinitesimally thin solids modeled by a lower dimensional triangulated surface. Since classical solid/fluid coupling algorithms rasterizing the solid body onto the fluid grid, an entirely new approach is required to treat thin objects that do not contain an interior region. Selle et al. [SRF05] introduce a new hybrid technique that makes synergistic use of Lagrangian vortex particle methods and Eulerian grid based methods to overcome the

weaknesses of both. The approach uses vorticity confinement itself to couple these two methods together and generates highly turbulent effects unachievable by standard grid based methods, and show applications to smoke, water and explosion simulations.

Starting with the previously mentioned work of Foster and Metaxas in 1996 [FM96], the amount of graphics research in modeling the Navier-Stokes with incompressible flow has targeted different areas of research, including semi-Lagrangian advection [STA99], vorticity confinement [FSJ01] [NFJ02], surface tracking [EMF02] [LTK*07][EFF*02][BGO*06], surface tension [LGF04], surface flow [STA03*], splashing surface waves [YHD07][WYL*05][KDB*06], dynamic meshes [KFC*06], fire [NFJ02][STA00], viscosity [CMT04][REN*04], visco-elastic modeling [GBO04], coupling with rigid and deformable solids [GSL*05], smooth-particle hydrodynamics [MSK*05] [PTB*03], multiple fluid flow [LSS*06], irregular boundaries [BBB07], and of course, sand [BYM05] [ZB05] and explosions [FOA03] [NF99] [RNG*03] [TOT*03]. However, these publications do not take into account any pertinent consideration to the integration of haptics for the generation of force feedback.

2.1.2 Other Approaches

If the interest is to directly simulate the physical effect of fluids or their surfaces instead of simulating the causes of it, other approaches may be used. Some of these approaches, among others, may be combined with CFD mechanisms to improve the accuracy of their simulation.

2.1.2.1 Procedural Fluid

A procedural method animates a physical effect directly instead of simulating the cause of it [FR86][HNC02]. A common way to procedurally generate the surface of a lake or ocean, for instance, is to superimpose sine waves of a variety of amplitudes and directions (See Figure 7). There are no limits to creativity when it comes to devising procedural animation techniques. Everything goes as long as the method achieves the desired visual effect. An advantage of procedural animation is its controllability, an important feature in games. The main disadvantage of simulating water procedurally is the difficulty of correctly getting the interaction of the water with the boundary or with immersed bodies.

Procedural methods for animating turbulent fluid are often preferred over simulation, both for speed and for the degree of animator control. Perlin [PER85] used noise as a method to mimic and model ocean waves by stochastically perturbing surface normals (bump mapping) according to a superposition of randomly distributed spherical wavefront sources. Bridson et al. [BHN07] offer an extremely simple approach to efficiently generating turbulent velocity fields based on Perlin noise, with a formula that is exactly incompressible (necessary for the characteristic look of everyday fluids), exactly respects solid boundaries (not allowing fluid to flow through arbitrarily-specified surfaces), and whose amplitude can be modulated in space as desired.

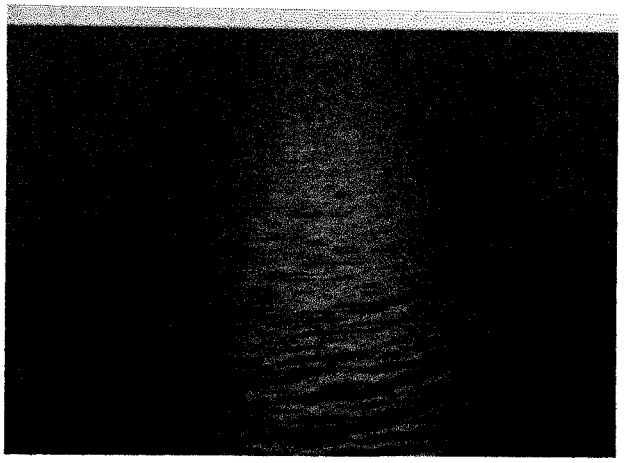


Figure 7: Using a procedural waver model, Hinsinger et al. restrict computations to the visible part of the ocean surface [HNC02].

2.1.2.2 HeightField Approximations

If we would only be interested in the animation of the two dimensional surface of a lake or ocean, it would be an overkill to simulate the entire three dimensional body of water. In this case, we could just only represent the surface as a two dimensional function or height-field and animate it using the 2D wave equation. This simplification can make the simulation orders of magnitude faster. The downside of this approach is that a function or height-field can only represent one height value at each location on the plane. This makes the simulation of breaking waves impossible. Kass and Miller linearize the shallow water equations to simulate liquids [KM90]. The simplifications do not, however, capture the interesting rotational motions characteristic of fluids.

A height field is a function of two variables that return the height for a given point in two-dimensional space. The equation below shows how a height field can be used to displace a plane in three-dimensional space.

$$p_{heightfield}(s,t) = p_{plane}(s,t) + f_{HF}(s,t) \cdot N_{plane} \qquad (9)$$

Using a height-field as a representation for a water surface does have its restrictions compared to a general surface. As the height field only represent a single height-value for each point it is impossible to have one part of the surface that overlaps another. Figure 9 shows an example of a height field in Maya and Figure 8 shows a two-dimensional equivalent of the limitation where the right image cannot represent the part of the curve that overlaps.

The height field has many advantages as it is easy to use and it is easy to find a data structure that is appropriate for storing it. It is often stored as a texture, which is called height-map. The process of rendering a surface with a height-map is usually called displacement-mapping as the original geometry (*Pplane*) is displaced by the amount defined in the height map.

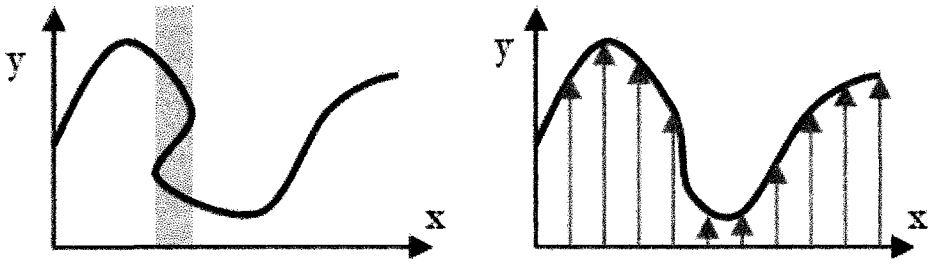


Figure 8: Illustration of the limitation that applies to height fields (2D equivalent). Left:(x,y)=(fx[s],fy[s]). Right:(x,y) = (x, f[x])

Figure 9: Fluid effects in Maya using 2D height field [AUTODESK].

2.1.2.3 Implicit Modeling for Deformable Objects

Soft Objects

We normally consider clay, skin, and water soft and concrete hard. However, almost all objects will exhibit some degree of softness in extreme conditions. Soft objects are those everyday objects that deform significantly in response to their normal environment. Examples include cushions and plasticine among others. In some cases, viscous fluids may be modeled using implicit techniques. Among the first to present a soft object model were Wyvill et al. [WMW86] who represent a plasticine-like soft object using a combination of an implicit function and a particle system. A simple approach is to use an explicit mass-spring system, representing the object as a lattice of mass elements linked with springs.

Terzopoulos and Fleischer [TF88] use a physically based approach to simulate a variety of deformable behaviors. They model objects using configurations of elastic, viscous, and plastic units. They achieve such effects as sculpting with lumps of plasticine, tearing leaves of paper, and pushing holes through fabric sheets. The models of Terzopoulos

and Fleischer store the elastic properties of materials in a stiffness matrix. This is difficult and expensive to compute. James and Pai [JP99] also use linear elasticity methods to model plasticine-like deformable objects, but they use a boundary element (BEM) solver and can achieve interactive real-time behavior by exploiting temporal coherence. Mass-spring systems, in which a lattice of mass elements linked with springs represents an object, provide an easier model of an elastic material. Miller [MIL88] uses this approach to animate the soft bodies of snakes and worms. He sends compression waves down the mass-spring system to produce locomotion of the animals.

The surface of a highly flexible substance should conform to the shapes of the objects it presses against. Cani-Gascuel and Desbrun [CD97] use implicit surfaces to achieve such an effect. In this model, skeletal primitives are enclosed in implicit surfaces. The skeleton layer is inelastic and gives the soft object's large-scale motion and shape. The implicit layer is elastic and gives the exact surface shape. The model can handle collisions between multiple soft objects and allows soft-object fusion. The model also provides solutions to problems associated with implicit surfaces, particularly volume variation and incorrect blending.

Nixon and Lobb [NL02] present a physically based model for animating soft objects. The model consists of two components: an elastic surface and a compressible fluid. As shown in Figure 10, the surface is represented as a mass spring system and the fluid is modeled using finite difference approximations to the Navier-Stokes equations of fluid flow.

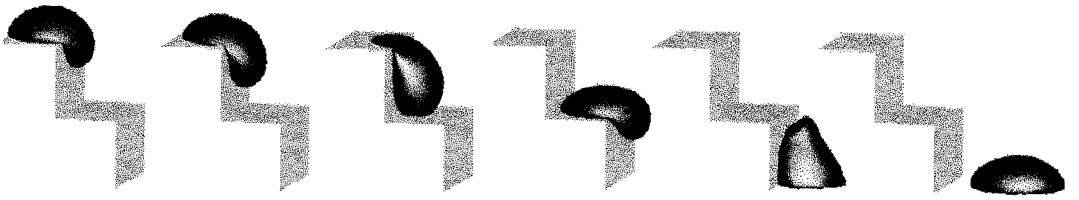


Figure 10: Closed elastic membranes filled with viscous compressible fluid. [NL02]

Metaballs

If a particle system is used for the simulation, then a fluid can also be modeled using spheres or metaballs in the positions of the particles, giving the impression of moving lava blobs (see Figure 11). Metaballs, in computer graphics terms, are organic-looking n-dimensional objects. This implicit modeling technique was invented by Jim Blinn in the early 1980s [BLI82]. Each metaball, or blobby object, is defined as a function in n-dimensions. A thresholding value is also chosen, to define a solid volume. Then,

$$\sum_{i=0}^{n} metaball_i(x, y, z) <= threshold \qquad (10)$$

represents whether the volume enclosed by the surface defined by n metaballs is filled at (x,y,z) or not. When seeking a more efficient fall off function, several qualities are desired:

- Finite Support: a function with finite support goes to zero at a maximum radius. When evaluating the metaball field, any points beyond their maximum radius from the sample point can be ignored. A hierarchical culling system can thus ensure only the closest metaballs will need to be evaluated regardless of the total number in the field.
- Smoothness: because the isosurface is the result of adding the fields together, its smoothness is dependent on the smoothness of the fall off curves.

A simple generalization of metaballs is to apply the fall off curve to distance-from-lines or distance-from-surfaces. The two most common rendering approaches for three dimensional metaballs are brute force raycasting and the marching cubes algorithm, both explained in section 2.2.

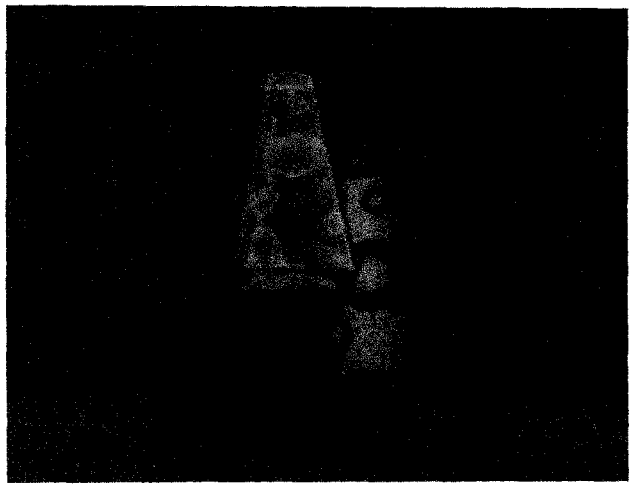


Figure 11: Metaballs in lava lamp [FAR07].

2.1.3 Discussion on Fluid Modeling

We would like to model a volumetric pool of fluid that generates current flows when in movement. The fluid surface should also be modeled so that it can deform with the touch of a haptic device. Therefore, procedural techniques presented in section 2.1.2 are not appropriate as they do not take into account the causes of inner fluid flow motion. In addition, implicit modeling may compress the fluid and generate discontinuous blobby effects. Since we need to track a fluid's flow field, CFD methods explained in section 2.1.1 are then more appropriate. An interesting CFD approach is Stam's [STA03] presentation of

a simple and rapid implementation of a fluid dynamics solver for game engines with the Navier-Stokes equations. However, his algorithm presents only a 2D implementation and does not take into account any incorporation of fluid surface deformations. Height-field approximations were previously presented as an approach to represent a fluid's surface. However, as the height field only represents a single height-value for each point it is impossible to have one part of the surface overlapping another part. In this thesis, we use a mass-spring particle system to account for surface deformations and we further explain this in chapter 3. In addition, we expand Stam's implementation to 3D so that we can integrate it with a 3D haptic interface and receive haptic force feedback in real-time. The next section presents different approaches to render fluids to the screen.

2.2 Fluid Rendering

Simulating fluids is one part of the equation, but rendering them to the screen is another significant issue. A three-dimensional fluid simulation produces volumetric data. This data is organized in the same manner as a two-dimensional image, but generalized in three dimensions, essentially a three-dimensional grid. When this grid is discussed in a rendering context, the individual cells are often referred to as voxels, a generalization of the word pixel. For a fluid simulation, the grid to display on the screen is a grid where each cell contains the density of the fluid at that location in space. This representation can also be applied to medical data, for example, where each cell would contain a density, the type of tissue, the amount of blood, etc. Even though volumetric data is very similar to standard two-dimensional data, it is much more difficult to display due to the fact that they are generally much larger, and much less adapted to computer display hardware. Most 3D

rendering techniques concentrate on rendering 2D polygons in 3D space [BB01][WV91] [CKH*99][OPL*01][LC87]. Volumetric data generates a large number of 3D voxels which need to be rendered on the screen. We are faced with a choice of using the graphics card to display data to which it is not very well adapted, or using the CPU to execute a more appropriate algorithm.

There are two principal techniques that are used for volumetric rendering, with some variants. One technique is a Raytracing algorithm which simulates the physics of light within the volumetric data, and the other technique consists of searching for surfaces within the data, which gives two-dimensional objects that the computer can display using traditional 3D rendering techniques. The following sub-sections explain these concepts in addition to other approaches such as shear-warp rendering and off-line rendering.

2.2.1 Raytracing

The physics of light suggests the simulation of light rays to produce the final image. In the real world, light passes within an object and then enters the eye or camera. In a virtual world the same principle is used, but only light rays that actually touch the camera are examined. This is done by tracing the rays in the opposite direction, from the camera to the object. A virtual screen is placed in front of the volumetric data, and rays of light are cast from the camera towards the virtual screen, and from there into the data. Samples are taken as the ray traverses the data, which allows for the calculation of the final color at that location on the virtual screen. This technique is used to draw fog in games, and for medical data [CKH*99] [BB01] [EYS*94]. The main idea taken from this approach is that cells that are further away from the camera should be rendered first on the screen. For instance, if

alpha-blending is enabled, the color of an incoming cell will blend with the color of existing cells that are behind it. This will be further explained in Chapter 4. This method includes some variants, mainly voxel rendering, splatting, shear-warp rendering and slices in 3D textures.

2.2.1.1 Voxel Rendering

This technique consists of drawing each point of data in the 3D cube directly on the screen by using one or several polygons or points in 3D space. As shown in Figure 12, a way to implement this is to draw the voxels directly on the screen with a simple 2D square. For each voxel in the cube, the computer draws a square in the same position, facing towards the camera, and with the appropriate color. Often, the data to be displayed is very sparse, where most voxels are empty. For this type of data, the algorithm can be optimized not to draw the squares which correspond to a voxel whose value is less than a certain threshold. This technique is very easy to implement and reasonably fast for small quantities of data, or very sparse data sets, but it can become extremely slow. This variant is actually a mathematical simplification of the Raytracing technique, although its implementation is completely different and much less complicated than a true Raytracing algorithm.

A team at Mitsubishi [OPL*01] developed architecture for a graphics card which can render relatively large quantities of volumetric data with low-cost hardware. This architecture uses a data storage layout which allows a large number of graphics processors to simultaneously render different pieces of the same data. This allows for a high degree of parallelism and therefore high-speed rendering, up to 30 frames per second for a 2563 cube. Mitsubishi's architecture is based on the voxel rendering technique [OPL*01]. However, we

are more interested in a technique that can perform well on a single computer without parallel processing.

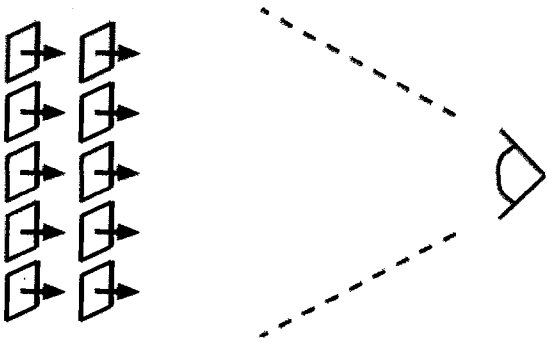


Figure 12: Voxels are drawn directly on the screen

2.2.1.2 Splatting

Ray casting works from the image space to the object space (volume dataset), thus it is called a backward projection method. Another way of achieving volume rendering is to try to reconstruct the image from the object space to the image space, by computing for every element in the dataset its contribution to the image. Several such techniques have been developed [DCH88][WG91][WES91]. In splatting, the final image is generated by computing for each voxel in the volume dataset its contribution to the final image. The algorithm works by virtually "throwing" the voxels onto the image plane. In this process every voxel in the object space leaves a footprint in the image space that will represent the object. The computation is processed by virtually "peeling" the object space in slices, and by accumulating the result in the image plane.

Formally the process consists of reconstructing the signal that represents the original object, sampling it and computing the image from the re-sampled signal. This reconstruction is done in steps, one voxel at a time. For each voxel, the algorithm calculates its contribution

to the final image, its footprint, and then it accumulates that footprint in the image plane buffer. The computation can take place in back-to-front or front-to-back order. The footprint is in fact the reconstruction kernel and its computation is the key to the accuracy of the algorithm. Westover [WES91] proves that the footprint does not depend on the spatial position of voxel itself, thus he is able to use a lookup table to approximate the footprint. During computation, the algorithm needs to multiply the footprint with the color of the voxel. However, this technique can become slow depending on the quantity of data.

2.2.1.3 Shear-Warp Rendering

A new approach to volume rendering was popularized by Philippe Lacroute and Marc Levoy, and described in the paper "Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation" [LL94]. In this technique, the viewing transformation is transformed such that the nearest face of the volume becomes axis aligned with an off-screen image buffer with a fixed scale of voxels to pixels. The volume is then rendered into this buffer using the far more favorable memory alignment and fixed scaling and blending factors. Once all slices of the volume have been rendered, the buffer is then warped into the desired orientation and scaled in the displayed image.

An overhead is expected due to the storing of multiple copies of the volume for the ability to have near axis aligned volumes. This technique is relatively fast at the expense of less accurate sampling and potentially worse image quality compared to other ray casting variants.

2.2.1.4 Slices in 3D Textures

It is possible to accelerate volumetric rendering by using graphic cards that have support for 3D textures. The volumetric data is loaded onto the video card as a 3D texture. The cube to display is cut into slices which are positioned to face the camera, as shown in Figure 13. Each slice shows a piece of the 3D cube. When the slices are drawn in a back-to-front order, the user sees the cube displayed on his screen. By varying the number of slices, it is possible to change the level of detail in order to have more detailed images, or a more fluid display. This technique takes advantage of the graphics hardware and therefore can be much faster than the direct voxel technique. However, it is also limited by the abilities of the graphics hardware. Like the voxels technique, this technique is a mathematical simplification of Raytracing, but with an extremely different implementation. This technique uses a dedicated graphics card which is present in most modern computers, which allows the main processor to be used for other tasks while the graphics card performs the rendering [FRA95].

However, this method requires the volumetric data to be loaded into the graphics card's memory, which is generally significantly smaller than the computer's main memory. For dynamic data, which is our case, this fact means that the computer is forced to reload the data onto the graphics card after each change in the data (timestep). Since the bandwidth to the graphics card is relatively small compared to the bandwidth to main memory, this can create a bottleneck for data updates. In addition, simulations often produce data in a format which is not directly supported by the graphics card, requiring an expensive conversion for every timestep update.

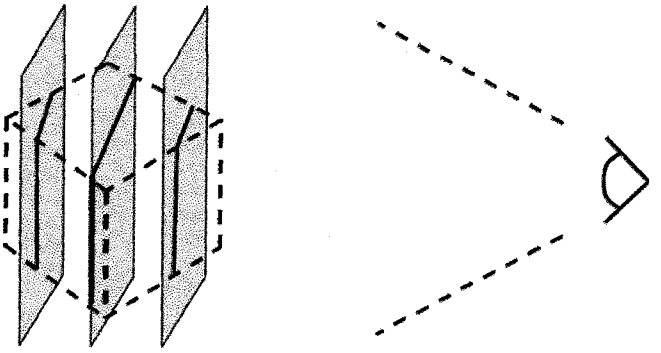


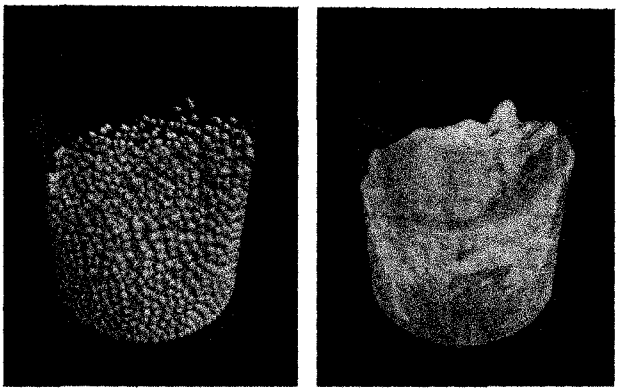
Figure 13: Volumetric rendering using 3D textured slices

2.2.2 Surface Reconstruction Techniques

The other principal technique is surface reconstruction. We define regions inside the volumetric cube, and the computer searches for surfaces between the regions. For example, if the cube contains a sphere with cell values of 1 on the interior and 0 on the exterior, the two regions could be defined as value < 1 and value >= 1.

The surface reconstruction algorithm would then construct the surface of the sphere using the values present in the cube. A fluid simulation could define a region where the density is greater than a certain value, and this technique would then search for iso-surfaces where the density is equal to this value. Medical data would define regions such as the brain, the skull, arteries, etc., and search for the surfaces between them. After defining the regions, the surface reconstruction algorithm analyzes the data and creates the surfaces. The standard surface reconstruction algorithm is *Marching Cubes* [LC87]. This algorithm examines the data in groups of eight, with each group forming a small cube. For each corner of the cube, the algorithm decides whether it is on the inside or the outside by using the provided region definitions. Taking into account all possible rotations and reflections, there are only fifteen

possible combinations, each one with a unique set of polygons which the algorithm adds to the surface under construction. Muller et al. [MCG03] show a swirl in a glass induced by a rotational force field. Part (a) of Figure 14 shows the particles and part (c) shows the iso-surface triangulated via marching cubes.



(a) (b)

Figure 14: (a) Glass particle swirl. (b) Iso-surface created via marching cubes. [MCG03]

Surface reconstruction techniques are fairly slow, and in general too slow to be used in real-time. For static data, the algorithm only has to be applied one time and the algorithm's speed is not a large problem. Once the algorithm has been applied, the computer works directly with the resulting 2D surfaces [LC87].

2.2.3 Off-line Rendering Tools

A simulation engine may also be ported to high-end 3D animation tools in the form of plug-ins. These software tools offer high quality fluid rendering modules to process off-line data. Some popular software packages include among others:

- Maya Unlimited: The Maya Fluid tool is a computational fluid dynamics toolset that brings a huge range of atmospheric, pyrotechnic, viscous liquid, and open water effects to Maya.

- Blender: It is a free, open-source, 3D studio for animation, modeling, rendering, and texturing offering a feature set comparable to high end and mid range 3D animation suites. It is developed under the GPL and runs on many platforms including Windows, OS X, Linux, BSD, Sun, and Irix as well as some handheld PC platforms.
- Houdini 9: Its solver lets users simulate liquids, gas and sand. The integration between a fluid simulation with other solvers such as the Rigid Body solver is possible. There is also a fluid solver available for Houdini's particle operators (POPs) to help add fluid-type motion to particle simulations.

However, for our implementation, we are interested in methods that allow the rendering of fluids in real-time.

2.2.4 Discussion on Fluid Rendering Approaches

Although the set of particles animated by a typical simulation technique move, as a whole, like a fluid, it often does not look like a fluid if rendered directly. Typically, the number of particles in a set is too small to give the appearance of a continuous surface of a fluid. From previous sections we have seen that some techniques could render particles as metaballs. However, they give the surface of the fluid a lumpy appearance, similar to rice pudding, which is unacceptable for representing fluids such as water. Another rendering technique creates an isosurface from the set of particles. The isosurface creates a smooth, spatially-continuous fluid surface; however, this surface is temporally discontinuous due to frame-to-frame noise. As a result, the motion of the isosurface appears jerky and discontinuous.

Another discussed technique creates a level set representing the fluid surface from the set of particles. The values of the level set are adjusted for each frame of animation from the velocities of the set of particles, resulting in a fluid surface that is spatially continuous. However, the resulting fluid appears sterile and artificial. Increasing the number of particles in the fluid simulation may allow for a more realistic rendering; however, this greatly increases the computational burden of the simulation. Additionally, level sets would only render the surface of the fluid but would not tackle the problem of volume rendering a three-dimensional fluid below the surface. Raytracing variants are reasonably fast for sparse data sets in volume rendering, but may become extremely slow when dealing with a large quantity of data.

It is therefore desirable to have a system and method for rendering a three-dimensional fluid without overtaking the computational burden of the fluid simulation. Chapter 4 explains in further detail our implemented rendering approach in which we use three-dimensional textures to represent the state of the fluid and then bind those textures to camera-aligned alpha-blended slices. The following section presents previous work in Haptics along with a discussion on haptic rendering approaches.

2.3 Haptic Rendering

Haptics is an area which is receiving increasing attention in research. Much of the work has been done to create perceptually plausible and distinct haptic models which mimic the physical properties of solid objects, or objects that have shape. Much less research has tackled the modeling and rendering of shapeless objects such as fluids; and therefore, this is something we try to address in this thesis.

Physical modeling in haptic refers to the techniques used to haptically represent an object, how tactile and kinesthetic cues are displayed to the user. This is partially dependent on what kind of application is intended. The algorithms used to model and synthesize haptic cues are collectively called *haptic rendering*. Similarly to how *graphic rendering* is used to display visual aspects of a computer generated object or environment, haptic rendering displays aspects which are touch-based. Many phenomena, physical or otherwise, may be rendered using haptic displays. For example, users of the technology may use force-feedback to explore the shape of an object in a 3D virtual environment; a graphical user interface (GUI) application may use haptic cues to purvey information about interaction with on-screen components, such as when a button is pushed or a radio box is checked; or the sensation felt as a needle punctures skin may be simulated by a medical application. Rendering of these phenomena is often done based on models which represent physical attributes of objects in the application environment, and several notable attributes are stiffness, damping, and roughness.

Stiffness defines the elastic resistance of an object to deformation by some force, modeled as being proportional to a displacement value $(e.g., \vec{f} = -k_s \vec{x})$. This property is

often used to control how hard or soft an object feels, and stiffness coefficients are common among many haptic rendering algorithms.

Damping controls the rate of deformation due to some force and is often modeled as being proportional to velocity ($e.g., \vec{f} = -k_d\vec{v}$). This property is related to the viscosity felt due to interaction with an object or environment.

Roughness is a measure of the small-scale variations in a surface and is related to friction and texture. Unlike stiffness and damping, roughness is a property valid only at the surface of an object. Assigning values to these and other characteristics can sometimes be done trivially or through artistic efforts. However, it often requires careful inspection of the real-world object being rendered. Estimation of physical properties based on real-world interaction behavior can be difficult and cumbersome.

For the class of haptic devices supported by the OpenHaptics toolkit [SENSABLE], forces are typically used to either resist or assist motion (i.e. force feedback). There are a variety of ways to compute the forces that are displayed by the haptic device. Some of the most interesting force interactions come from considering the position of the device *end-effector* (the end of the kinematic chain of the device you hold in your hand) and its relationship to objects in a virtual environment. When zero force is being rendered the motion of the device end-effector should feel relatively free and weightless. As the user moves the device's end-effector around the virtual environment, the haptics rendering loop commands forces at a very high rate (1000 times per second is a typical value) that impedes the end-effector from penetrating surfaces [BAS07]. This allows the user to effectively feel the shape of objects in a virtual environment.

Haptic rendering algorithms for force-based displays are variations on a common theme. Quite often, a penalty-based approach is taken to render haptic models [SB97][BAS07]. At a basic level, a haptic device generates force-feedback based on the position of the probe's end-effector and the Haptic Interface Point (HIP). The difference between the HIP and the end-effector simply refers to the level of abstraction that they represent. The end-effector is the virtual proxy of the system, while the HIP is the tip of the haptic probe. These two positions are initially the same, but as the player manipulates the haptic device, the HIP might traverse a collision surface. A force is then rendered at the haptic device which is directly proportional to the vector (times the stiffness scalar) between the device's end-effector and the position of the HIP. In Figure 15, the HIP's position has penetrated a static obstacle (e.g. the baton has touched a wall of the bowl). Since the end-effector cannot move to the HIP's position, a spring force is displayed at the haptic device and the users can feel a collision response.

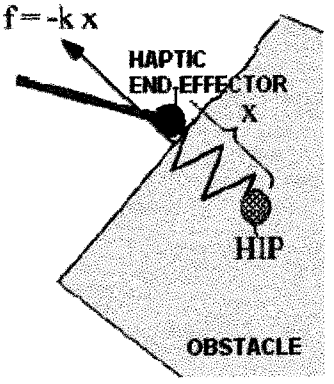


Figure 15: Penalty-based haptic rendering.

The way in which forces are computed can vary to produce different effects. For example, the forces can make an object surface feel hard, soft, rough, slick, sticky, etc.

Furthermore, the forces generated by the haptics rendering can be used to produce an ambient effect. For instance, inertia, and viscosity are common ways to modify the otherwise free space motion of the user in the environment. Another common use of forces in a virtual environment is to provide guidance by constraining the user's motion while the user is selecting an object or performing a manipulation. Further background on haptic rendering and haptics in general can be found in some additional articles [HBS99][MRF*96]. The following sub-sections present a categorization of haptic devices, previous work on rigid body and elastic contacts, volumetric bodies, and a discussion on haptic rendering approaches.

2.3.1 Devices

Haptic devices have varying complexities, and can move in different ways. Force feedback devices are sometimes described by their Degrees of Freedom (DOF). A Degree of Freedom refers to a direction of movement. Common Degrees of Freedom include right-left movement (X), up-down movement (Y), forwards-backwards movement (Z), roll (rotation about the Z axis), pitch (rotation about the X axis), and yaw (rotation about the Y axis). Degrees of Freedom can refer both to how a device keeps track of position, and how a device outputs forces. A mouse, for example, is a 2 DOF input device – it keeps track of position in the right-left, and the forward-backward. A joystick is also a 2 DOF device, but its Degrees of Freedom are different (it rotates forwards-backward, and right-left). A force feedback joystick is a 2 DOF device with force feedback. It both tracks 2 DOF and gives simple forces in 2 DOF. The following list presents different approaches available to achieve various forms of haptic information.

- **Exoskeletons and Stationary Devices:** The term exoskeleton refers to the hard outer shell that exists on many insects and other creatures. In a technical sense the word refers to a system that covers the user or that the user has to wear. Current haptic devices that are classified as exoskeletons are large and immobile systems that the user must attach himself or herself to, as shown on Figure 16 (Left). The benefit of exoskeleton devices is that their large size and immobile nature allow for the generation of large and varied force information without strict size or weight constraints. Their main disadvantages are large size, heavy weight, and high cost. Also the calibration to adapt to different sized users is an issue.

Figure 16: Left: Exoskeleton device [VRAC]. Right: CyberGlove [IMMERSION]

- **Gloves and Wearable devices:** These devices are smaller exoskeleton-like devices that often, but not always, take the form of a glove (Figure 16 Right). One major benefit of a wearable system is that the user can move more naturally without being weighed down by a large exoskeleton or other immobile device. The drawback of the wearable systems is that since weight and size of the devices are still a concern the systems will have a more limited set of capabilities. Also the calibration to adapt to different sized users is an issue.

- **Point-sources and Specific Task Devices:** This is a class of devices that are very specialized in either their technology or their application. Devices that provide any type of

force information that may be required can be complicated and even extremely difficult to develop. Designing a device to perform a single type of task restricts the application of that device to a much smaller number of functions. However, it allows the designer to focus the device to perform its task extremely well. These specific task devices have two general forms - single point of interface devices and specific task devices.

An example of a single point of interface device is the Phantom from Sensable Technologies™. The Phantom has either a handle or a fingertip interface with an armature that provides up to six degrees of freedom, to a single point. The Phantom family of devices are all 6-DOF input, based on an articulated armature design. The Desktop and Omni (Figure 17 Right) models provide 3-DOF force feedback, while the Premium-A model is capable of full 6-DOF haptic feedback (force and torque). The novel Novint Falcon provides 3-DOF Input/Output at an affordable price (Figure 17 Left).

Our research focuses primarily on models which are suitable for stylus-based haptic interfaces (e.g., [SENSABLE]). So in our system, the user interacts with the environment using an Omni Phantom device although display on other devices, such as Cyber-Gloves, may be possible by sending appropriate forces to each output sensor (e.g., [PH03]). We leave it up to the application developer to ensure that any vibrotactile and force cues rendered by fluid interaction are compatible with the limits of the haptic device.

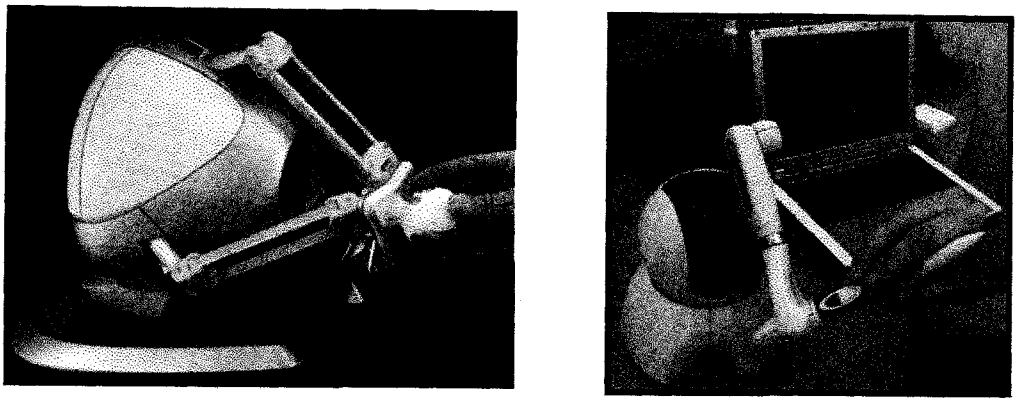


Figure 17: Left: The Novint Falcon [NOVINT]. Right: The Sensable Phantom Omni [SENSABLE].

- **Locomotive Interfaces:** Locomotion interfaces are movement of force restriction devices that, in a confined space, simulate unrestrained human mobility such as walking and running for virtual reality (Figure 18). Locomotion interfaces overcome limitations of using joysticks for maneuvering or whole-body motion platforms, in which the user is seated and does not expend energy, and of room environments, where only short distances can be traversed.

Figure 18: Treadport, Sarcos Inc. [SARCOS].

- **Vibrotactile Feedback devices:** These are vibrating gamepads and force-feedback joysticks which offer limited interactivity and display simple haptic effects to the user

(Figure 19 Left). For example, open-loop vibrotactile feedback and predefined force feedback signals. These devices complement other peripherals such as keyboards, mice, or trackers to enhance the user's interaction experience.

Figure 19: Left: Sidewinder Force feedback Joystick, Microsoft [MICROSOFT]. Right: Tactile Display [TFC07]

- **Tactile displays:** These devices stimulate the skin to generate the sensations of contact (Figure 19 Right). The skin responds to several types of physical sensations; such as vibrations, small-scale shape or pressure distribution, and temperature sensations. Tactile feedback can also be used to produce a symbol, like Braille, or simply a sensation that indicates some condition or surface texture.

2.3.2 Haptic Forces and Textures

Much effort has been applied to the problem of haptic display of rigid body contact as well as compliant elastic contact. Some progress has also been made in haptic rendering of textured surfaces [OJS*04] [SP96]. However, haptic simulation of the interaction of shapeless volumetric objects such as fluids is still a very novel field of research.

Rigid Body and Elastic Contacts

Some research in this field has been applied in the context of tissue modeling for surgical simulation [AD03]. Several techniques have been proposed for integrating force feedback with real-time virtual environments to enhance the user's ability to perform interaction tasks [CB94][MS94][SBM95]. Ruspini et al. [RKK97] presented a haptic interface library "HL" that uses a virtual proxy and a multi-level control system to effectively display forces using 3-DOF haptic devices. Hollerbach et al. [HCT*97] [NNH*98] described a haptic display system for contact and manipulation in the CAD design of mechanical assemblies, and Thompson et al. [TJC97] have presented a system for direct haptic rendering of sculptured models.

Durbeck et al. [DMW*98] have described a system for enhancing scientific visualization of vector fields with haptic feedback. The force is computed by using a point-probe model and a simple vector-to-force magnitude mapping function. Unlike our method, it does not compute actual aerodynamic or hydrodynamic forces, it does not support volumetric probes, and the forces were not physically-based. More recently, Lawrence et al. presented a haptic display technique using a 5-DOF force feedback device for shocks and vortices in hypersonic CFD datasets [LLP*00].

McNeely et al. [MPT99] presented a method for the haptic rendering of complex virtual environments by voxelizing objects. Otaduy and Lin [OL03] presented a simplification algorithm for faster collision queries with sufficient accuracy for the haptic rendering. These methods realize realistic haptic interactions with virtual worlds. However, they cannot handle forces by objects interacting with fluids.

Volumetric Bodies

Gibson [GIB95] proposed an algorithm for object manipulation including haptic interaction with volumetric objects and physically-realistic modeling of object interactions. The algorithms presented by Avila and Sobierajski [AS96] rely on interactive force feedback and rendering to allow a user to quickly explore and modify volumetric scenes. Essentially, most present techniques for direct haptic rendering of volumetric data follow a general method, where the force display is defined as a vector valued function, or is transformed to one [AS96][GIB95][IN93]. From this function, force feedback is generated by the data around the probe and from the velocity of the tip.

2.3.3 Discussion on Haptic Rendering Approaches

The amount of literature regarding haptic technology and rendering has increased substantially in recent years. In the context of gaming, experimental haptic games such as HaptiCast [AML*96] – our system - , and Haptic Battle Pong [MNS04] have been generating brainstorming ideas for assessing haptic effects in game design. Nilsson and Aamisepp [NA03] explain the relevance of incorporating haptics in a 3D game engine and a plug-in for Crystal Space [CRYSTAL] was developed to demonstrate this integration successfully. However, haptic interaction in the context of 3D gaming was not well explored by their project, nor it addressed any fluid haptic feedback.

Other efforts [CBM*05] to combine haptic and graphical rendering are ambitious, but they are just concerned with the introduction of a general haptic library without targeting specific feedback needed for fluids. Dachille et al. [DQK01] present interesting haptic

sculpting tools to expedite the deformation of B-Spline surfaces with haptic feedback and constraints, but they also do not explore any feedback integration with fluid simulations.

Foster and Metaxas [FM96] used a form of the hydrostatic force equations to create only visual animations of rigid body objects in their offline Navier-Stokes simulation. Hydrostatics ignores the dynamic effects of fluid flow on the objects. Their simulation also did not account for the effect of the objects on the fluid.

Tu [TU96] computed forces by using a boundary integral of the relative velocity between a fish's fin surface and surrounding fluid for simulating how swimming motions propel a fish. Like [FM96] this method apparently does not take the influence of the immersed surface on the fluid, since at the actual surface of an immersed object, the physical boundary conditions demand that the velocity of the fluid relative to the surface is always zero, which would mean the force should always be zero as well with this method.

Gosline et al. [GSY04] used Finite Element methods to simulate fluid pockets enclosed in an elastic body. However, they focus on the relation between the volume and pressure of a fluid cavity using a specially designed haptic device and the simulation is limited to 2D. Ristow [RIS99] [RIS00] has created a number of graphical offline simulations of spherical and elliptical particles falling through fluids using accurate force computations based on the fluid stress tensor. However, forces were not targeted to a real-time haptic application. Our proposed method uses a different numerical procedure to compute the force. A distinction is that our "particle" is an actively controlled haptic probe rather than a passively simulated object. In contrast to these methods, we use a real-time fluid simulation to generate the force feedback to drive a haptic display, allowing the user to both feel and influence the fluid at the same time.

2.4 Haptic-Fluid Integration

There are two interesting related projects that address the integration of haptics and fluids [DSH*06] [BL04]. However, they employ different application-dependent approaches that do not target the problem we are trying to solve. We present them in more detail in the following subsections.

2.4.1 A Fluid Resistance Map Method

Dobashi and his team [DSH*06] created a model that approximates real-world forces acting on a fishing rod or kayak paddle by doing part of the math in advance of the simulation: the forces associated with different water velocities and different positions for the paddle or fishing lure were pre-calculated and saved in the database. In addition, their simulation is based on a much larger setup, including two projection screens and large haptic equipment (see Figure 20).

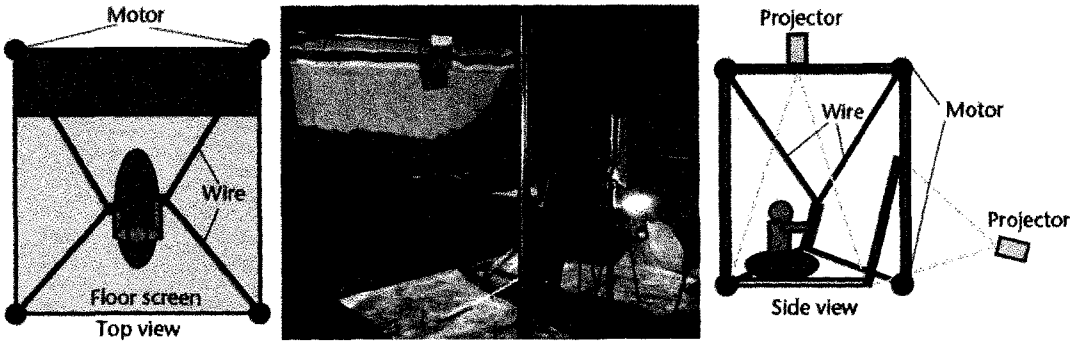


Figure 20: Dobashi's et al. large simulation setup [DSH*06].

In contrast to this, our intention is to enable the rendering of real-time fluid calculations on a personal computer or a laptop with a low-end desktop haptic device. To cope with these resource limitations, we take a different approach in simulating real-time 3D

fluids by using a grid-based numerical method free from timestep restrictions and rendering real-time forces based on the density and velocity fields of the simulation. Section 4.2 presents a table that summarizes the differences between our approaches.

2.4.2 Physically-Based Model for Interactive Digital Painting

Baxter and Lin [BL04] demonstrated a method for integrating force feedback with interactive fluid simulations. They succeeded in a real-time haptic display of fluids in two dimensions by directly solving Navier-Stokes equations. Similarly to us, they calculate the feedback forces due to fluids based on physical phenomenon; however, they did not explore volumetric interactions in three-dimensions and therefore ignored 3D related issues such as volume rendering and haptic interaction in a volumetric space. In addition, an object controlled by the user is discretized by using grids. The discretized shape changes from frame to frame when the object moves. This causes undesirable noise in the resulting force.

They adapt their fluid-haptic feedback method for use in a painting application that enables artists to feel the viscosity of the paint as they make brush strokes using the haptic stylus on a flat surface (see Figure 21). However, their application is a mere translation of a two-dimensional plane in a three-dimensional space, which does not reflect a real-time volumetric interaction with a fluid in motion. In this thesis, we focus on the force feedback that results from the interaction with a pool of moving fluid.

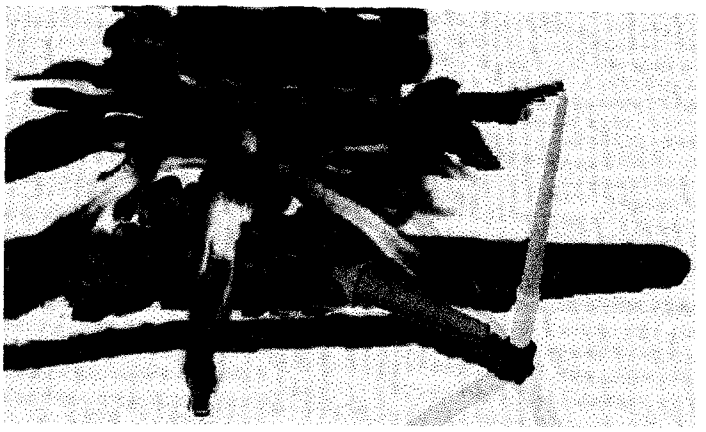


Figure 21: Screen capture of Baxter and Lin painting application [BL04].

In contrast to their approach, we explore haptic ambient forces to represent differences in fluid densities. An ambient force is a global strength effect that surrounds the haptic probe, regardless of collision with any surface. In addition, we adapt our method to be integrated with a spring-net deformable surface. As the fluid surface has a stronger tension than the inner fluid, this spring-net surface permits the calculation of force feedback generated by surface tension, enabling users to feel the surface waves and ripples in a 3D perspective. The intended result is to generate a fluid-like haptic effect. When users touch the fluid surface through the haptic interface, they can perceive the resulting surface deformations. When they stir the fluid, they can see changes in the fluid's density and velocity and simultaneously feel the resulting force in real-time. The force felt depends on the velocity, direction, density and viscosity properties. Section 4.2 presents a table that summarizes the differences between our approaches.

The next chapter presents the fluid processing stage of the system, which includes the solution of the flow equations, a description of the surface deformation process and the integration between graphics and haptics workspaces.

Chapter 3 Fluid Flow Processing Stage

In this research, we would like to achieve a stable three-dimensional fluid simulation algorithm, so that we can get enough information to later render them to the user, graphically and haptically. The simulation should also account for input from the haptic device and run in real-time. In this chapter we present the fluid flow processing stage of the system, which includes the solution of the flow equations, the extension of the simulation to three dimensions, the simulation of multiple substances, a description of the surface deformation process, and the integration between graphics and haptics workspaces.

3.1 Real-time Fluid Simulation

In order to make a simulation algorithm suitable for real-time interactive applications, it needs to have certain characteristics:

- Inexpensive computations: the application must run at a constant rate of 30 – 60 frames per second. From the total time available for a single frame, the physically-based simulation only gets a fraction besides other tasks such as graphic and haptic rendering. Hence, low computational complexity remains one of the most important constraints.
- Low memory consumption: while in off-line computations, people simply buy as much memory as they need to simulate the desired effect, the memory on a single PC is limited and needs to be shared among different tasks very similar to the computation time. Therefore memory efficiency of the method also needs to get special attention.

- Stability: In off-line simulations adaptive time-stepping can be used in situations where stability problems arise. In contrast, a real-time application runs at a fixed frame rate. The simulation method must, therefore, be stable for the given time step size no matter what happens. External forces and the movement of boundaries can get almost arbitrarily high. A real-time simulation method needs to be able to cope with all those situations and must remain stable under all circumstances.
- Plausibility: Of course it is not possible to reduce the computational and spatial complexity so drastically and increase the stability so significantly without some trade off with the quality of the result. Therefore, what we require from a real-time simulation method is visual plausibility, not necessarily scenes that are undistinguishable from the real world.

Jos Stam [STA99] [STA03] proposed a two-dimensional fluid simulation based on real fluid physics, but whose speed is adequate for simulation on a standard PC. Our approach departs from Jos Stam's previous work and from Fedkiw's et al. [FSJ01] but we further extend the simulation to three-dimensions in order to take advantage of the open haptic workspace. For the sake of clarity we will start to describe a fluid living in two dimensions. The extension to three dimensions is described in the following section. We model density as a set of particles (centers of grid cells) that move through a velocity field using Eulerian Grid-based approach. The fundamentals of this equation were explained in chapter 2 and we now proceed to explain its solution.

Jos Stam's main contribution was to propose an unconditionally-stable, semi-implicit advection scheme for the real-time solution of the Navier-Stokes equation. Our contribution is not the solution of the Navier-Stokes equation itself, but how Stam's approach can be expanded to couple a 3D fluid flow with a deformable surface and haptic interaction in real-time.

3.1.1 Solving the Equations

Analytical solutions of the Navier-Stokes equations can only be found for a few simple physical problems. Nevertheless, it is possible to use numerical integration techniques to solve them incrementally. The aim of this project is to display the evolution of the flow over time, so an incremental numerical solution is sufficient.

The first step in the algorithm is to transform the Navier-Stokes equations into a simplified form that one can use in a numerical solution [STA99]. Recall from section 2.1.1.1 that the four terms on the right-hand side of the Navier-Stokes equation represent accelerations. The fourth term, pressure, says that the velocity moves along a pressure gradient. This term will actually disappear because it creates divergence. Divergence means that the velocity would always flow out of high-pressure cells, which is illegal in our constant-density universe, so Stam's simplification [STA99] omits this term in his presentation.

The following section describes a transformation that leads to a straightforward algorithm. Very often density is not constant (e.g. in smoke simulations), and we want density to move around too. Stam [STA99] introduces a *density equation* that we add to our

system to simulate density movement, and it looks very much like the *velocity equation*, as follows:

$$\textit{Velocity Movement: } \frac{\partial \vec{u}}{\partial t} = -(\vec{u} \cdot \nabla)\vec{u} + \nu \nabla^2 \vec{u} + \vec{f} \qquad (11)$$

$$\textit{Density Movement: } \frac{\partial \rho}{\partial t} = -(\vec{u} \cdot \nabla)\rho + k\nabla^2 \rho + S \qquad (12)$$

where u is the vector-valued velocity at a point, t is time, ν is the kinematic viscosity of the fluid, f represents the external force, _ represents its density, S represents the external substance that is being added to the fluid, k represents a constant at which density tends to diffuse, "·" denotes a dot product between vectors, and "∇" denotes the vector of spatial partial derivatives. The density equation says that (a) density moves along the velocity field, (b) density tends to diffuse according to some constant k, and (c) the user can add or remove density S anywhere he/she wants. The two similar representations of Navier-Stokes equations indicate that both velocity and density fields could be solved in a similar fashion.

For a computer simulation, it is necessary to create a discrete representation of the fluid. Our simulation places the fluid in a grid where each cell contains a velocity and a density, as shown in Figure 22.

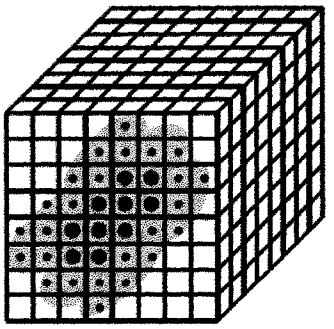


Figure 22: Organization of a fluid simulation's volumetric data

Initially, both the velocity and the density are assumed to be constant in each grid cell and we usually display their values at the cell center. In practice we allocate two arrays for both the density and the velocity of size, *size= (N+2)*(N+2) (In 3D it would be size= (NX+2)*(NY+2)*(NZ+2))*. We prefer to use single dimensional arrays over double ones for efficiency purposes in our implementation. The movement of the fluid is determined by several simulation steps carried out on the velocity and density. First we will show how to solve the density equation. This will explain the different components of the solver. Subsequently we will transfer these ideas to the harder problem of simulating velocity fields.

The basic structure of the solver is as follows. We start with some initial state for the velocity and the density and then update its values according to events happening in the environment such as the interactions with the haptic device. In the prototype, we let the user apply forces and add density sources with the haptic probe.

The fluid flow can then be visualized by tracking substance particles which simply move through the fluid as done by Foster and Metaxas [FM96]. As an example, we implemented a haptic game where a user uses a haptic probe to stir a pool of fluid in a bowl and also inject multiple substances into the fluid. The scenario has two components; (i) fluid animation - a magician mix various substances in the liquid bowl and (ii) haptic motion recognition (it will be discussed in Chapter 5) - when the substances and the mixing gestures are correct, the liquid turns to a magic potion with vapor. In our game example, the forces could come from the movement of the haptic device while preparing a magic potion, while the density sources could represent the different substantial ingredients that are needed in the potion recipe.

The simulation is therefore a set of snapshots of the velocity and density grids. We assume that the time spacing between the snapshots is given by the fixed variable *dt* in the remainder of this document.

Moving Densities

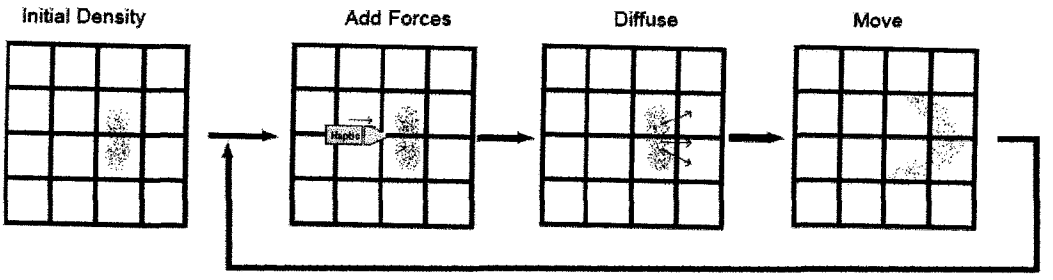


Figure 23: At every time-step we resolve the three terms appearing on the right hand side of the density equation.

As mentioned above we will first describe the solver for a density field moving through a fixed velocity field that does not change over time. The density equation states that the changes in density over a single time step are due to three causes. These causes are the three terms on the right hand side of the equal sign in the equation. The first term says that the density should follow the velocity field, the second states that the density may diffuse at a certain rate and the third term says that the density may increase or decrease depending on the addition or removal of substances by the user. The solver will resolve these terms in the reverse order as they appear in the equation as shown in Figure 23. We start from an initial density and then repeatedly resolve these three terms over each time step.

The first term is easy to implement. We assume that the sources for a given frame are provided in an array *s[]*. This array is filled in by some part of the game engine which

detects sources of density. In our prototype, it is filled with a substance that emerges from the stirring baton when the user presses a button on the haptic probe, as shown in Figure 25.

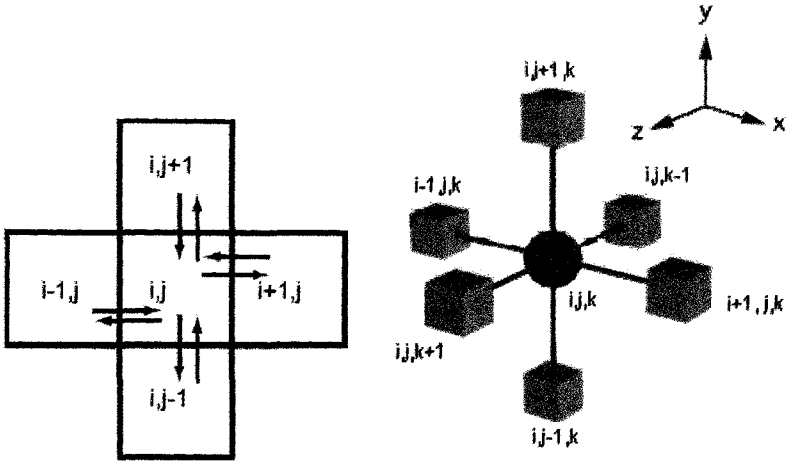


Figure 24: Through diffusion each cell exchanges density with its direct neighbors. Left: 2D. Right: 3D.

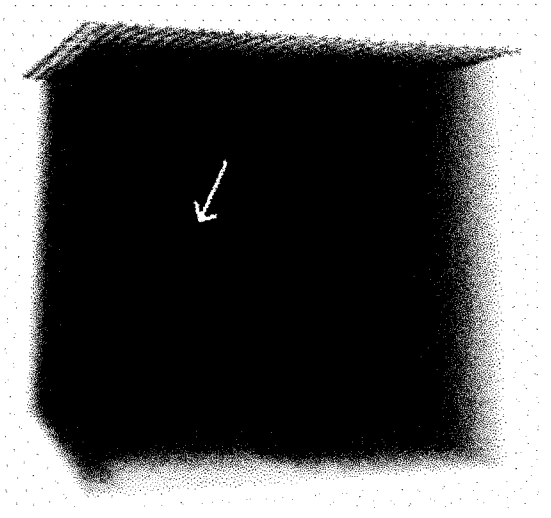


Figure 25: Substance is inserted into the simulation through the tip of the haptic device. Device is moving in the direction of the arrow introducing a force.

The second step accounts for possible diffusion at a rate *diff* (given as a parameter), when *diff* > *0* the density and the velocity will spread across the grid cells. We first consider what happens at a single grid cell. In this case we assume that the cell exchanges densities

only with its four direct neighbors as shown in Figure 24. The cell's density will decrease by losing density to its neighbors, but will also increase due to densities flowing in from the neighbors, which results in a net difference of

$$d_0[i-1,j]+ d_0[i+1,j]+ d_0[i,j-1]+ d_0[i,j+1]-4* d_0[i,j]$$

where d_0 is the density value at a specific grid cell. A possible implementation of a diffusion solver then simply computes these exchanges at every grid cell and adds them to the existing values. Although the diffusion routine might seem attractive at first, it unfortunately does not work. For large diffusion rates a the density values start to oscillate, become negative and finally diverge, making the simulation collapse. This behavior is a general problem that plagues unstable behavior. For these reasons we consider a stable method for the diffusion step. The basic idea behind this stable method is to find the densities which when diffused backward in time yield the densities we started with. In code:

$$d_0[i,j] = d_1[i,j] - a*(d_1[i-1,j]+d_1[i+1,j]+d_1[i,j-1]+d_1[i,j+1]-4*d_1[i,j]);$$

where d_0 is the density we started with and d_1 is the next density value of a grid cell. This is a linear system for the unknowns $d_1[i, j]$. We could build the matrix for this linear system and then call a standard matrix inversion routine. However, this is overkill for this problem because the matrix is very sparse: only very few of its elements are non-zero. Consequently we can use a simpler iterative technique to invert the matrix. The simplest iterative solver which works well in practice is Gauss-Seidel relaxation [COL00]. This relaxation attempts to increase the convergence rate by using values computed for the k_{th} iteration in subsequent computations within the k_{th} iteration. The general iteration formula is:

$$x_i^k = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^k - \sum_{j=i+1}^{n} a_{ij}x_j^{k-1}\right] \qquad (13)$$

which assume we compute the x_i in natural order (i.e. sequentially). Elimination methods such as Gaussian elimination are prone to round off errors for a large set of equations. Iterative methods, such as Gauss-Seidel method, allow the user the control of the round-off error. Also if the physics of the problem are well known for faster convergence, initial guesses needed in iterative methods can be made more judiciously. The benefit of this version of the diffusion solver is that it is almost as simple as the unstable one, but can handle any values for *diff, dt,* or *N*. Regardless of how big these values are the simulation will not blow up.

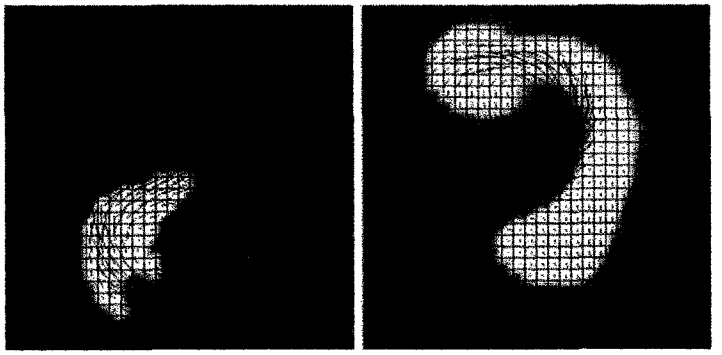


Figure 26: The advection step moves the density through a static velocity field. The red short line segments are the current and the white regions contain high density.

The final step in the density solver forces the density to follow a given velocity field. Refer to Figure 26. Again we want a technique which is stable and does not blow up. Similarly to the diffusion step we could set up a linear system and solve it using Gauss-Seidel relaxation. However, the resulting linear equations would now depend on the velocity, making it trickier to solve. Fortunately, there is an alternative which is more effective. The key idea behind this technique is that moving densities would be easy to solve

if the density were modeled as a set of particles. In this case we would simply have to trace the particles though the velocity field. For example, we could pretend that each grid cell's center is a particle and trace it through the velocity field as shown in Figure 27(b). The problem is that we then have to convert these particles back to grid values. How to properly do that is not necessarily obvious. A better method is to find the particles which over a single time step end up exactly at the grid cell's centers as shown in Figure 27(c). The amount of density that these particles carry is simply obtained by linearly interpolating the density at their starting location with the closest neighbors. This suggests the following update procedure for the density. Start with two grids: one that contains the density values from the previous time step and one that will contain the new values. For each grid cell of the latter we trace the cell's center position backwards through the velocity field. We then linearly interpolate from the grid of previous density values and assign this value to the current grid cell. So, instead of moving the cell centers forward in time (Figure 27(b)) through the velocity field shown in (Figure 27(a)), we look for the particles which end up exactly at the cell centers by tracing backwards in time from the cell centers (Figure 27(c)). At time zero, when the system starts, we construct and pass an original vector field to be used by the system. This initial vector field, which is our base case, consists of zero component vectors resembling an undisturbed vector field. When the user introduces forces via the haptic device on a particular grid cell, the read haptic-force values are added into the appropriate vector components of that cell. The resulting vector field is the one used for the advection process already explained in this section. These advection steps are grouped together into an *advect()* routine.

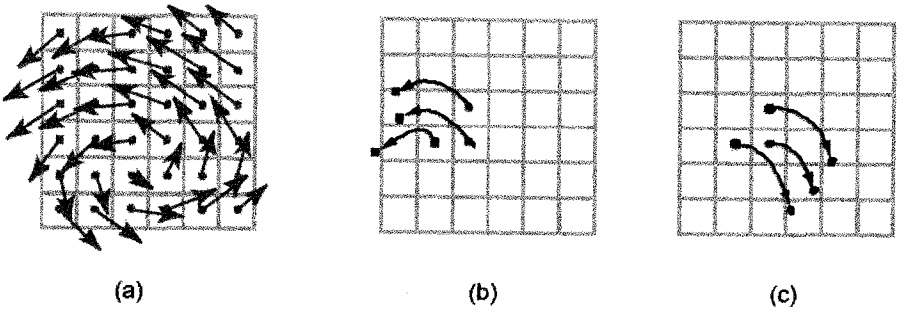


Figure 27: Basic idea behind the advection step.

The idea of tracing back and interpolating goes back to the work by Courant et al. [CIR52]. It has since then been rediscovered by many researchers in various fields and is generally classified under the heading of "Semi-Lagrangian" techniques. The back-tracing method is the main reason for making the interactive fluid stable and efficient, but it is also the main reason for causing not high-level accuracy and unrealistic visual effects. However the visual effects are still quite acceptable for our goals. This completes our description of the density solver. All of these steps can conveniently be grouped together into a single routine.

Evolving Velocities

Once again consider the Navier-Stokes equations presented earlier. In the light of what we now know about the density solver we can interpret the velocity equation as saying that the velocity over a time step changes due to three causes: the addition of forces, viscous diffusion and self-advection. Self-advection may seem obscure but we can simply interpret it as the fact that the velocity field is moved along itself. More importantly we can reuse the routines that we developed for the density solver and apply them to update the velocity field. In most cases we simply had to duplicate the calls for each component of the velocity field. There is, however, a routine called *project()* which forces the velocity to be mass conserving. This is an important property of real fluids which should be enforced. Visually it

forces the flow to have many vortices which produce realistic flows. It is therefore an important part of the solver.

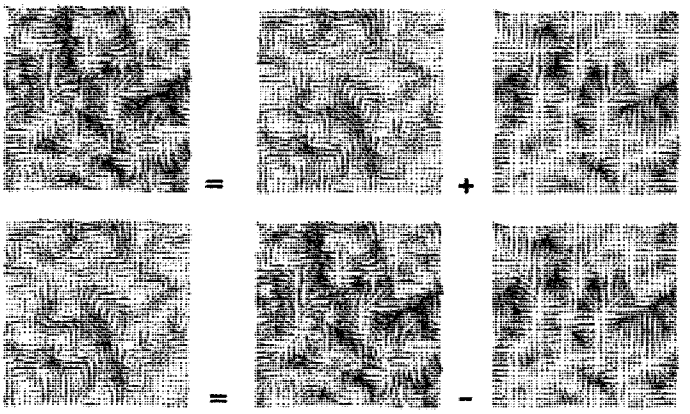


Figure 28: Using Hodge decomposition to obtain an incompressible field [STA01]. Up left most image shows velocity fields, up-middle image shows mass conserving fied and the up-right one shows gradient field.

After the steps preceding the *project()* routine the velocity field seldom conserves mass. The idea is to make it mass conserving in the last step. To achieve this we use a result from pure mathematics called the *Hodge decomposition*: every velocity field is the sum of a mass conserving field and a gradient field. This result is illustrated in Figure 28 (top). Notice how the mass conserving field has nice vortices, typically the type of field we would like to have. On the other hand the gradient field shown in the upper right corner of Figure 28 is the worst possible case: the flow at some points either points all outward or inward. In fact the gradient field indicates the direction of steepest descent of some height function. Imagine a terrain with hills and valleys with an arrow at every point pointing in the direction of steepest descent. Computing the gradient is then equivalent to computing a height field. Once we have this height field we can subtract its gradient from our velocity field to get a mass conserving one as shown in Figure 28 (bottom). We will not go into the mathematical details, but will simply state that computing the height field involves the solution of some

linear system called a Poisson equation. This system is sparse and we can re-use the Gauss-Seidel relaxation routine developed for the density diffusion step to solve it.

We call the *project*() routine twice in the implementation. We do this because the *advect()* routine behaves more accurately when the velocity field is mass conserving. In conclusion, the *project()* routine refers to the process of *pressure projection*, which is not any kind of geometrical projection, but which represents the routine for finding the best divergence-free field to match a vector field by subtracting the gradient of pressure, as previously explained through the Hodge decomposition. A further discussion on pressure projection can be found in Bridson et al. 2007 [BM07].

Boundary Conditions

Something we have left out up to now is the treatment of the boundary, namely the purpose of the *set_bnd()* routine. We assume that the fluid is contained in a box with solid walls: no flow should exit the walls. This simply means that the horizontal component of the velocity should be zero on the vertical walls, while the vertical component of the velocity should be zero on the horizontal walls. For the density and other fields considered in the code we simply assume continuity.

Other boundary conditions are of course possible. For example, we could assume that the fluid wraps around itself: a flow that exits one wall simply reenters the opposite one. Another possibility is to have a fixed velocity on some parts of the boundary to simulate an inflow like that found in a wind tunnel. A simple way of implementing internal boundaries is to allocate a Boolean grid which indicates which cells are occupied by an object or not. Then we simply have to add some code to the *set_bnd()* routine to fill in values for the occupied cells from the values of their direct neighbors. This simple procedure will work if

an object is at least two grid cells thick, since otherwise some values might leak through the boundary. However, we believe that it is not worth the additional effort, since the visual improvement is minor.

The current solver suffers from what is called "numerical dissipation": the fluids dampen faster than they should in reality. This is in essence what makes the algorithms stable. Recently Fedkiw et al. [FSJ01] proposed a technique called "vorticity confinement" which re-injects the lost energy due to dissipation back into the fluid, through a force which encourages the flow to exhibit small scale vorticity. This technique works well for the simulation of smoke for example.

3.1.2 Simulation Extension to 3D

The most important quantity to represent in a fluid simulation is the velocity of the fluid, because velocity determines how the fluid moves itself and the things that are in it. In the context of this thesis, we define the velocity vector field of a fluid on a three-dimensional *Cartesian* grid such that for every discrete position $\vec{x} = (x, y, z)$, there is an associated velocity at time t, $\vec{u}(\vec{x}, t) = (u(\vec{x}, t), v(\vec{x}, t), w(\vec{x}, t))$. The key to fluid simulation is to take steps in time, and at each time step, correctly determine the current velocity field. This is done by solving the Navier-Stokes equations for incompressible flow as explained previously. Once velocity field is acquired, one can use it to move objects, smoke, cloud water concentrations, and other quantities that can be displayed in applications.

The explained simulation algorithm was presented as a two-dimensional simulation, but all of the techniques can be applied to a three-dimensional simulation. In terms of implementation, some functions need to be extended as follows:

- It is necessary to add a third term in every function that manipulates the 3D grid.
- We need to change all the loops and data structures to accommodate the third dimension within the simulation. This means that call frequencies change from 2^2 = 4 in 2D to 2^3 = 8 calls in 3D.
- Interpolation is changed to a trilinear interpolation.
- The number of cell's neighbours increase from 4 to 6, as shown in Figure 24.
- The cell's density will decrease by losing density to its neighbors, but will also increase due to densities flowing in from the neighbors, which results in a net difference of:

$$d_0[i,j,k] = (d_1[i,j,k] + a*(d_1[i-1,j,k] + d_1[i+1,j,k] + d_1[i,j-1,k] + d_1[i,j+1,k] + d_1[i,j,k-1] + d_1[i,j,k+1]))/(1+6*a);$$

  where d_0 is the density we started with, d_1 is the next density value of a grid cell, $a=dt*diff*N*N*N$, and N is the number of grid cells.
- The *set boundaries* routine, *set_bnd()*, needs to further check for a velocity component of zero on the walls of the new axis.

A two-dimensional simulation produces data which can be easily transformed into pixels that are then displayed directly on the screen. However, a three-dimensional simulation creates volumetric data which is difficult to display. Displaying these results requires volumetric rendering techniques. Chapter 4 will explore this issue in more detail.

3.1.3 User Interaction and External Forces

Forces will set the fluid into motion while substances will inject densities into the environment. The Sensable Omni [SENSABLE] provides a robust API that allows the

programmer to read input parameters of the haptic device, such as velocity and position of the probe. Therefore, we can read the velocity vector that describes the probe motion and use that information to inject a force into the simulation. We use the applied force values to update the simulation of both the inner fluid and its surface in the following manner.

A grid cell has array information about velocity *u[] v[] w[]* and also density *d[]*. For example, Figure 29 shows a force being applied at grid cell *xyz [5, 2, 10]* in the direction of the velocity vector *uvw(0.2, 0.8, 0.5)*. To integrate this into the solver, we go through the following steps which account for the force term of equation 3:

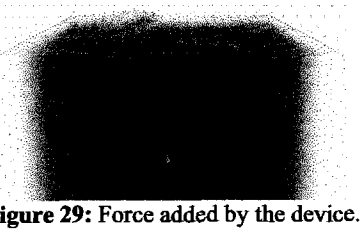


Figure 29: Force added by the device.

1. Calculate the 3D grid-cell in which the HIP is positioned, using the hapto-visual mapping function explained in section 3.3.
2. Read the velocity vector of the device.
3. Insert force constant at the grid-cell in the direction of the velocity vector (a value for each velocity component in x, y, z). For instance:

   u[5,2,10] = *force**0.2;

   v[5,2,10] = *force**0.8;

   w[5,2,10] = *force**0.5;

   where *force* can be one(1) or a scaled value imposed by the programmer. The solver uses this updated array of values for the computation of the next time-step.
4. The solver finds the subsequent timestep of this force insertion.

At the surface, force is used to deform the mass-spring system. In Figure 35 we can see that the force is proportional to distance *y*, times the stiffness scalar *k*. When the probe touches the surface, we find the closest mass-spring particle and add the force to that particle, generating a new ordinary differential equation which is solved by the RK4 method explained in section 3.2.1.

The input force is the distance *y* between the device's end-effector and the position of the *HIP*. Stiffness *k* is given by the elasticity of the surface (*k=0* is a full elastic surface, while *k=1* is a rigid non-elastic surface). Other mass-spring particle system parameters are: gravity, drag, spring-stiffness and damping-constant. However, spring-stiffness *k* is the one used to calculate haptic force value as shown in the figure. The other parameters are used to simulate the graphic deformation of the system and how quickly it can restore itself.

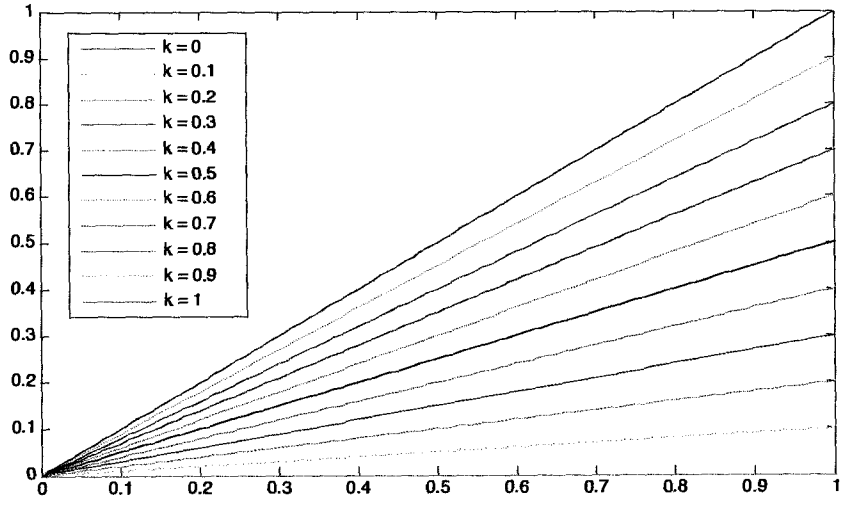


Figure 30: Output force *f* with respect to input force (distance *y*), in terms of surface stiffness *k*.

Figure 30 shows the amount of force feedback felt, f, with respect to the distance, y, between the device's end-effector and the position of the HIP. The stiffness parameter, k, is presented at different values to show how it affects the force felt. A surface with high elasticity (low k values) is able to easily deform resulting in low y and f values. On the other hand, a more rigid surface can produce higher force feedback as it is less likely to deform resulting in large y distance values.

3.1.4 Fluid Simulation of Multiple Substances

A substance represents an entity with given properties (e.g. density, color, viscosity) that enters the base fluid simulation. Our system allows for the combination of multiple substances on top of the base fluid. Therefore, different calculation grids are maintained for each substance, as each of them have their own characteristics and colors. The resulting rendered force is a weighted combination of each substance's grid involved in the mix. Figure 31 shows a screenshot of an initial red substance which is later mixed with a denser green substance. The result is a yellowish blend which combines the contributed haptic properties of both sources.

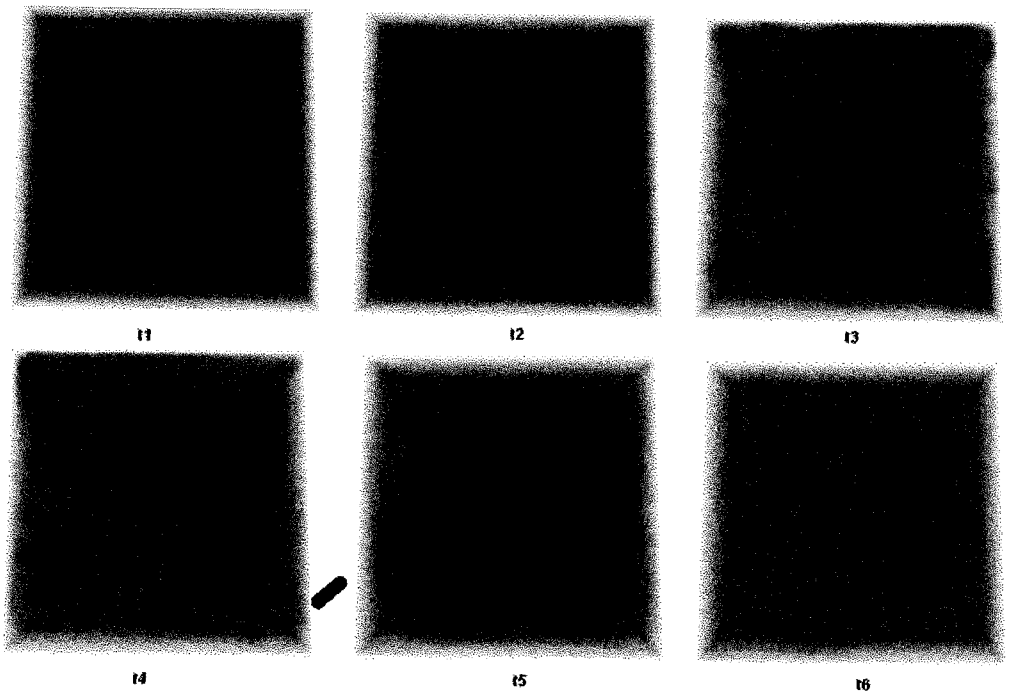


Figure 31: Simulation of multiple substances. A red and a green substance are added into the simulation (t1-t2). The user stirs with the haptic device until they mix together in the fluid (t3-t6).

This feature allows the user to perceive the dynamics of mixing substances with different properties into one simulation. The haptic rendering effect at a particular coordinate in space results from adding properties of each substance at that particular location.

3.2 Surface Deformation

There are two main reasons to overlay a surface on top of the 3D fluid:

(i) Graphical rendering purposes: one thing that should be noted is that although particles are more accurately moved through space, they alone could not be used to represent a dynamic surface such as water. The problem is that as the simulation is updated, the distribution of particles can become very uneven, resulting in unresolved areas where there simply aren't

enough particles to properly represent a surface. The second problem inherent in a particle only method is that a surface still needs to be defined and doing so based on particles is non trivial.

(ii) Haptic force-feedback purposes: As the fluid surface has a stronger tension than the inner fluid, this spring-net surface permits the calculation of force feedback generated by surface tension, enabling users to feel the surface waves and ripples in a 3D perspective.

Therefore, we use a mass-spring model to represent the deformable surface. In section 2 we said that Nixon and Lobb [NL02] have used explicit mass-spring system, as shown in Figure 10, but they used the system to represent closed elastic membranes filled with viscous compressible fluids. A mass-spring system is also used by Miller [MIL88] to animate the soft bodies of snakes and worms, but it was not considered in the context of fluid simulations of incompressible flows. Other methods such as Height-Field approximations only represent a single height-value for each point and it is impossible to have one part of the surface overlapping another. In addition, level set surfaces are susceptible to numerical dissipation, and are unable to maintain areas of high curvature. There are two reasons for choosing a mass-spring model instead of other computational models. The first is computational speed as the mass-spring model can be computed in real-time. The second reason is that mass-spring facilitates the interaction handling between the fluid and the surface, as they are both discreet implementations that can match the same grid resolution.

The major drawback of mass-spring model is that its smoothness depends on its grid size. The more particles the smoother, but the more it will increase computational time. However, by choosing an acceptable grid size we were able to achieve plausible looking

deformations. Section 3.2.3 includes a table that clearly shows the maximum resolution supported on our system.

3.2.1 Deformation Process

Our three-dimensional deformable surface is designed based on a discreet mass-spring particle system [NMK*05]. Springs have adjustable stiffness and damping constraints that represent different surfaces. The surface deforms with the touch of a haptic probe and gives back the resulting forces to the users. By modeling the surface of the fluid as a deformable surface, we are able to simulate the waves and ripples generated by the touchable interaction via the haptic interface. The main disadvantage is that the fluid surface will not be able to tear apart, spill over a container, nor generate splashes, but it will still permit a visual representation of a wavy surface based on user interaction as drafted on Figure 32.

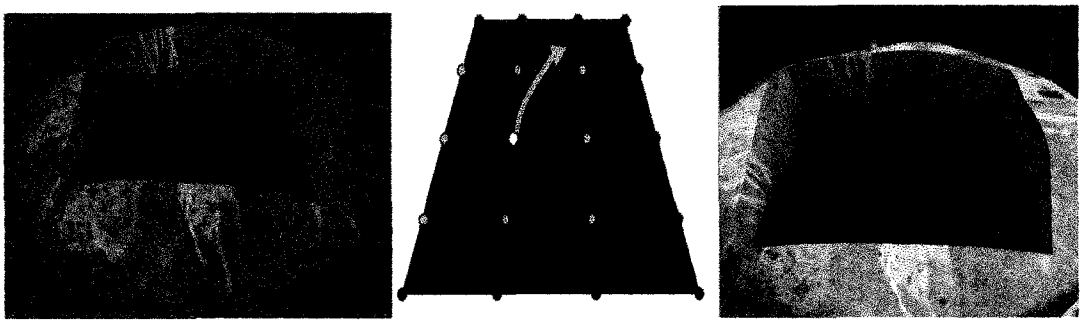


Figure 32: Draft of a deformable surface. Weighted particles move according to user's haptic interaction.

The deformable surface uses the classical fourth-order Runge–Kutta (RK4) method [COL00] to solve the ordinary differential equations (ODE) formed by the applied forces and the constrained spring-network of particles. We saw from section 3.1.3 and Figure 35

that a spring force is generated at deformation, which is proportional and opposite in direction to the penetration of the haptic interface point (HIP) beneath the surface (y), such that force values, f, are calculated by $\vec{f} = -ky$. In addition, a damping (friction) force resists the motion and it is proportional to the velocity $\vec{f} = -bv$ where b is the damping coefficient and v is the velocity. Therefore, the total deformation force is $\vec{f} = -ky - bv$. Combining with Newton's law of motion $f = m.a$, and the definition of acceleration as the second derivative of position a = y" we have the differential equation:

$$m\,y'' = \_k\,y \_ b\,v \qquad (14)$$

$$y'' = \_\,k/m\,y \_\,b/m\,y' \qquad (15)$$

This is the equation of motion, defining what happens to the deformable surface over time. In order to use RK4, we convert the second order differential equation (15) into a set of first order differential equations. For this we can write the acceleration as the first derivative of velocity: y" = v'.

$$y' = v \qquad (16)$$

$$v' = \_\,k/m\,y \_\,b/m\,v \qquad (17)$$

which is the form we need to use RK4. To begin the simulation, we initialize the two variables y,v for their value at time t=0. We then use the RK4 algorithm [COL00] to calculate the values of y,v after a short time interval, and this continues indefinitely. In summary, the RK4 algorithm is as follows. Let an initial value problem be specified as:

$$y' = f(t,y), y(t_0) = y_0 \qquad (18)$$

Then the RK4 problem is given by the following equation:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \qquad (19)$$

where

$$k_1 = f(t_n, y_n) \tag{20}$$

$$k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \tag{21}$$

$$k_3 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \tag{22}$$

$$k_4 = f(t_n + h, y_n + hk_3) \tag{23}$$

The parameters have meaning as follows:

- k_1 is the slope at the beginning of the interval;
- k_2 is the slope at the midpoint of the interval, using slope k_1 to determine the value of y at the point $t_n + h/2$ using Euler's method;
- k_3 is again the slope at the midpoint, but now using the slope k_2 to determine the y-value;
- k_4 is the slope at the end of the interval, with its y-value determined using k_3.

Thus, the next value (y_{n+1}) is determined by the present value (y_n) plus the product of the size of the interval (h) and an estimated slope. The slope is a weighted average of slopes. In averaging the four slopes, greater weight is given to the slopes at the midpoint:

$$slope = \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \tag{24}$$

The RK4 method is a fourth-order method, meaning that the error per step is on the order of h^5, while the total accumulated error has order h^4. Note that the above formulas are valid for both scalar- and vector-valued functions (i.e., y can be a vector and f an operator). In order to assimilate and get closer to a real-life interaction, we communicate the deformable surface deformation to all the fluid grid layers that are positioned beneath the surface.

However, these layers will damp the deformation based on their distance to the surface. Layers that are farther away from the surface will exhibit a lower deformation than layers that are closer to the fluid surface. The next section describes this technique.

3.2.2 Damping Layers

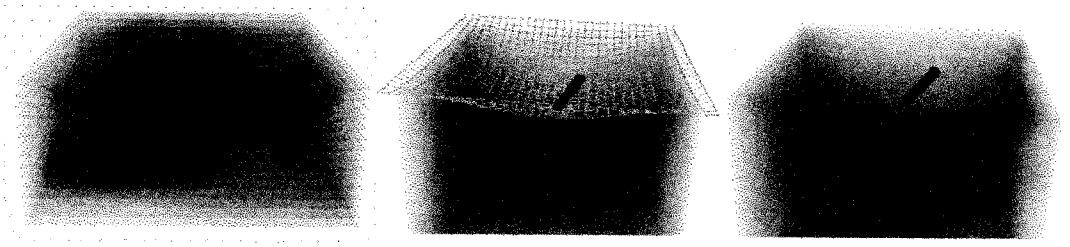


Figure 33: Damped Layers.

In order to interconnect the surface with the fluid and retain the deformation within the bowl, the applied energy is damped when it travels through the inner grid layers. A layer is each of the depth rows in a three-dimensional grid, starting from the surface and finishing at the bottom of the bowl. Damping refers to the progressive reduction or suppression of oscillation in a device or system. Layers that are farther away from the surface will exhibit a lower deformation than layers that are closer to the fluid surface. The deformable surface matches the resolution size of a grid layer. This enables a straight-forward coupling between surface and fluid particles.

In order to achieve this effect, as shown in Figure 33, we need to store the initial position of the particles when the surface has not yet been deformed. We only need to do this once, when the system starts. At each scene rendering update, we compute the distance vector between the stored initial state and the new state of the deformation. Each layer is assigned a deformation scaling factor based on their properties, including their depth or distance from the surface. These scaling factors follow an exponential sequence. For each particle on a layer:

$$p_1 = p_0 + S * \vec{d} \qquad (25)$$

where p_1 is the final position of a particle, p_0 was the original position of the particle when the system started, S is the deformable scaling factor for the layer in which the particle exists, and $\vec{d}$ is the distance vector between the stored original position and the projected position if no damping were introduced. The projected position is obtained from the RK4 algorithm described above. We will then use this new acquired information to generate our texture values at the moment of rendering. This is further explained in chapter 4.

3.2.3 Resolution Trade-Off

In order to represent a fluid, space (either 2D or 3D) is divided into as many cells as necessary to ensure sufficient precision or resolution in the simulation. Each cell contains values which represent the fluid (density and velocity).

However, there is a trade-off between fluid accuracy and performance of the system. The more cells used in the simulation, the more accurate the fluid's representation becomes, but the more computations are needed to calculate the state of the simulation. In our system, a PC with an AMD Opteron™ Processor 252, 2.59 GHz, 2GB in RAM, and an NVIDIA Quadro FX 3400/4400 card, we reached a stable simulation with a grid of NxNxN, where N=15.

Table 1: System resolution trade-off.

N	NUMBER OF GRID CELLS (NxNxN)	SIMULATION TIME (FPS)
30	27000	~ 8
25	15625	~ 12
20	8000	~ 20
15	3375	~ 32

This was the maximum setup permitted to support a reasonable stable and fast real-time simulation of approx. 32 FPS. If we further increase the size, the deformable surface would perform too slowly for real-time purposes. If we further lower the size, it may perform faster but the deformations would not look smooth and the fluid flow would become less realistic. Table 1 summarizes these findings.

3.3 Coupling Graphics and Haptics Workspaces

Typically, haptic devices are not employed in isolation. They are most often used to enhance the user experience in conjunction with a virtual 3D graphical environment. In order to enable haptic interaction, all objects modeled in the graphic workspace also need to be modeled on the haptic workspace. In order for users to feel what they actually see, the position of these 3D models needs to match and correspond on the scene. The first issue to consider when combining haptics with 3D graphics is that the refresh rate for displaying forces on the haptic device is more than an order of magnitude higher than the refresh rate necessary for displaying images on the screen. This difference stems from the psycho-physics of human perception. Typically, a graphics application will refresh the contents of the frame buffer approximately 30-60 times a second in order to give the human eye the impression of continuous motion on the screen. However, a haptic application will refresh the forces rendered by the haptic device at approximately 1000 times a second in order to give the kinesthetic sense of stiff contact [BAS07]. If the frame rate of a graphics application is run at a rate lower than 30 Hz, the user may perceive discontinuities in an animation such that it no longer appears visually smooth. Similarly, the user may perceive force discontinuities and a loss in fidelity when the haptic device is refreshed at a rate below

1000 Hz. As a result, haptics and graphics rendering are typically performed concurrently in separate threads so that each rendering loop can run at its respective refresh rate.

On this matter, the number of deformable surface particles was adjusted to maintain the stability of the system as explained in the previous section. This is a very important point to keep in mind, especially in an application where more than one object is moving on the screen or being felt by the haptic simulation at the same time. The HLAPI [SENSABLE] allows you to specify geometry for haptic rendering in the same thread and at the same rate as graphics. It takes care of the 1000 Hz haptics updates for you so that you do not have to implement any haptic rendering in the 1000 Hz haptics thread or implement state synchronization between haptics and graphics threads.

A haptic device is a natural interface for a dynamic simulation because it allows the user to provide both input to the simulation in the form of forces, positions, velocity, etc. as well as receive force output from the simulation. In our fluid system, we want the tip of the visual baton to correspond with the Haptic Interface Point (HIP), so that the surface can be deformed at the right contact point, substances can be added at the right spot, and forces can be rendered at the correct location. We are interested in rendering immediate force feedback while maintaining acceptable visual simulation effects.

A dynamic simulation with a haptic device requires special treatment, however. First a dynamic simulation works by integrating forces applied to bodies. When dealing with a position controlled impedance style haptic device, such as the kind currently supported by the OpenHaptics toolkit [SENSABLE], forces are not directly available as input. Additionally, the mechanical properties and digital nature of the haptic device make it challenging to directly incorporate as part of the simulation. Combining a haptic device with

a dynamic simulation tends to be much more approachable and stable if a *virtual coupling* technique is used. Virtual coupling introduces a layer of indirection between the mechanical device and the simulation.

Since there are differences between the graphic workspace and the haptic workspace, the integration between graphic and haptic workspaces is required. The graphic workspace and the haptic workspace have different boundary limitations and coordinate systems. Therefore, the point of intersection between the haptic probe and the fluid surface is different in both workspaces. The 3D cursor represents the position of the probe as well as the user's point of interaction. The fluid surface grid size is defined as an NxNxN cube, and its coordinates range from [0...N-1] in the X, Y and Z axis. If we want to know which part of the fluid surface was touched, we need to convert the haptic coordinates into those of the fluid grid. We want to take advantage of the ample haptic workspace to match the positional limitations of our fluid grid, as depicted in Figure 34.

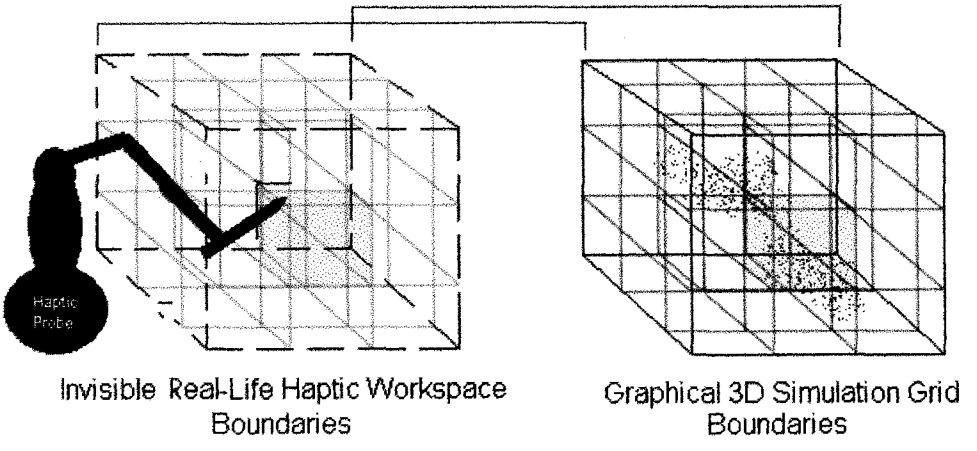


Figure 34: Matching boundaries between Workspaces.

To match both workspaces, we first determine the boundaries of the haptic workspace by moving the probe around and tracing out the virtual proxy position at each update. We can then record the positions at which we want our simulation to be bounded to. We then use a mapping function that converts any of the possible haptic virtual proxy positions to a place in the three-dimensional grid (range from [0...N-1] in the X, Y and Z axis):

$$P_{Gx} = (((P_{Hx} + Max_{Hx})/(Max_{Hx}-Min_{Hx}))*Nx; \quad (26)$$

$$P_{Gy} = (((P_{Hy} + Max_{Hy})/(Max_{Hy}-Min_{Hy}))*Ny; \quad (27)$$

$$P_{Gz} = (((P_{Hz} + Max_{Hz})/(Max_{Hz}-Min_{Hz}))*Nz; \quad (28)$$

where P_G is the grid position we want to find, P_H is the position of the haptic virtual proxy we need to convert, Max_H and Min_H are the maximum and minimum positions that the virtual proxy can have, and N is the grid resolution (number of grid cells) at a particular axis. Based on these conversions, the graphic workspace and haptic workspace are integrated precisely and it keeps performance in real-time. Figure 25 demonstrates the correspondence between graphic and haptic coordinates. The HIP position is mapped in order to determine the cell at which the substance enters the simulation.

In the case of haptics, the sense of feeling something can be improved dramatically by providing a visual representation of the contact. The trick is to provide the correct visual. For instance, one common mistake with haptics is to haptically render contact with a rigid virtual object yet visually display the device cursor penetrating its surface. The illusion of contacting a rigid virtual object can be made significantly more believable if the cursor is never displayed violating the contact. In most cases, this is simply a matter of displaying the

constrained proxy instead of the device position. Haptic cues can also be used to reinforce visual cues. For instance, it is common for selection of an object to be preceded or accompanied by highlighting of the object. An appropriate haptic cue can make that highlighting even more pronounced by providing a gravity well or a localized friction sensation. In the next chapter we present our approach to haptically and graphically render the fluid simulation that was acquired through the mechanisms explained in this chapter.

Chapter 4 Hapto-Visual Rendering Stage

In this research, we would like to achieve a haptically and visually plausible fluid rendering of our simulation. It is therefore desirable to have a system and method for rendering a three-dimensional fluid without overtaking the computational burden of the fluid simulation. In this chapter we present the hapto-visual rendering stage of the system, which includes descriptions of our graphical rendering and our haptic rendering, and a discussion on how to bind both approaches together.

4.1 Volumetric Haptic Rendering – Force Feedback

The force feedback calculation is based on the equations of an incompressible Navier-Stokes fluid simulation explained in previous chapters, which enables the generation of forces for use with haptic devices. The bowl can also be touched through the haptic interface, giving the player a sense of boundary limitations for the interaction. In contrast with conventional haptic systems, our reaction force feedback originates from two main sources:

(i) Deformable surface – that accounts for elastic forces as well as surface tension.

(ii) Fluid simulation – provides values for viscosity, density, velocity, and inertia.

The fluid surface is deformed as the haptic probe pushes through it. As explained in section 3.2, this deformation is gradually transmitted and damped to the lower layers of the inner fluid based on their depth and fluid density. After a certain pop-through force threshold, the probe is able to penetrate the surface and interact with the inner 3D fluid.

Once inside the fluid, the sense of viscosity can be rendered in any direction of interaction as the probe moves on the three-dimensional scene.

In order to better appreciate the quality of haptic rendering, our interface permits that the user is able to mix and toggle between different force rendering modes; (i) deformable-surface mode (ii) viscosity mode (iii) flow mode and (iv) flow-resistant mode. If the user is feeling the surface and then pushes the probe inside the fluid, the force mode is changed automatically. Each of these force modes may be enabled or disabled with toggle keyboard buttons according to the user's preferences. These force modes focus on particular aspects of the force feedback:

The deformable-surface mode enables the user to feel the ripples on the fluid surface. The deformable surface is haptically rendered following the penalty-based method shown on Figure 15 and Figure 35. The applied force deforms the mass-spring system and generates surface oscillations. If enough force higher than the surface tension is applied to the surface, a pop-through threshold lets the probe go through it and access the inner fluid simulation. If not enough force is applied and the user lets loose of the stylus, the probe will float over the deformed surface simulating the idle wavy state of the surface.

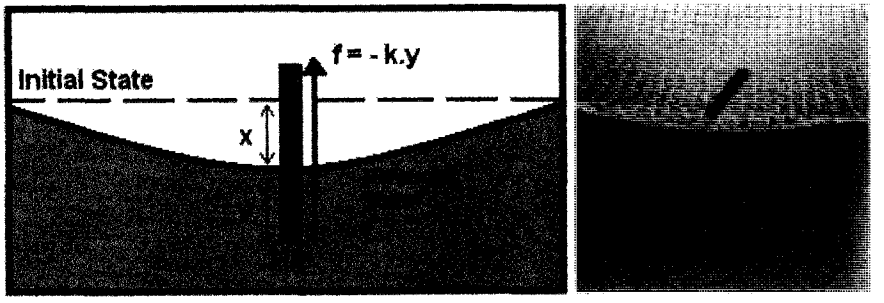


Figure 35: Haptic Collision Response.

This approach to surface haptic rendering uses a spring constant, k, and a displacement value, y, to generate a spring force which is proportional and opposite in direction to the penetration of the haptic interface point (HIP) beneath the surface, such that force values, f, are calculated by:

$$\vec{f} = -ky \qquad (29)$$

The viscosity mode challenges the user to move around thick fluid. Elastic spring forces are controlled by stiffness properties that are particular to rigid surfaces, like the walls of a bowl. However, when the probe enters an area of fluid, the force felt is that of a viscous force rather than a spring force. The fluid does not actually repel the probe, but just slows down its stirring movement, according to the density grid computed at the time. The more substance a grid cell contains, the harder it is to stir through it. This ambient force makes use of the following commands:

```
hlEffectd(HL_EFFECT_PROPERTY_MAGNITUDE, var...);
hlEffectd(HL_EFFECT_PROPERTY_GAIN, var...);
hlTriggerEffect(HL_EFFECT_VISCOUS);
```

The viscous force is based on the current velocity of the haptic device and is calculated to resist the motion of the haptic device. Specifically the force is calculated using the equation [LL75]:

$$f = -k.V \qquad (30)$$

where f is the spring force, V is the velocity and k is the gain. For simplification, in the name of interactivity, the gain is proportional to the concentration of substance contained in the grid cell at which the probe is positioned. For instance, in Figure 34, the position of the HIP is matched to a cell in the fluid simulation. The amount of density in that cell will determine the value of k. We can then infer, from this example, that it would be easier to move the

probe around the corner cells of the grid than in the middle cells of the grid. The magnitude of the effect force is capped at the value of the effect property HL_EFFECT_MAGNITUDE. The gain is specified by the property HL_EFFECT_PROPERTY_GAIN. Higher gains will cause these effects to generate larger forces.

The flow mode guides the haptic probe, hence the user's hand, through the velocity field (Figure 36) so that the user perceives the formed currents. To implement this, we repeatedly disable and enable HL_PROXY_RESOLUTION in order to *setPosition()* and *readPosition()* of the haptic device.

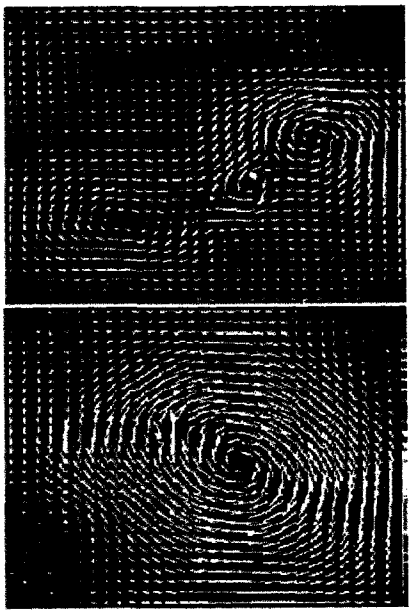


Figure 36: Fluid flow vector field.

We treat the *HIP* as an extra particle in the simulation. This lets us know the specific velocity vector that the particle exhibits at a particular grid cell position. We can then

instruct the haptic probe to accordingly render a force following the *u, v, w* components of that vector. This force approximation makes the assumption that the velocity field can be characterized by single vectors, so these methods can only generate forces, and not torques.

The flow-resistant mode enables the user to modify the velocity field by applying forces that resist the current flow. As shown in Figure 37, the effect in this case is an inertia effect, which simulates a small point mass being dragged around by the haptic device. If the fluid's velocity field is running to the left and the user tries to stir to the right, for instance, a higher force feedback will be felt until the velocity field has adapted itself to the new input forces.

Figure 37: The current flow influences the movement of the haptic probe.

In this case, we follow Newton's second law of motion, which states that $\vec{f} = m \cdot \vec{a}$. The acceleration a is in the direction of the force and proportional to its strength, and is also inversely proportional to the mass being moved. In our implementation we approach this as:

$$\begin{aligned} S_f &= P_{KS} * (H_{pos} - P_{pos}); \\ D_f &= -P_{KD} * P_v \\ I_f &= S_f + D_f \end{aligned} \qquad (31)$$

where H_{pos} and P_{pos} are the positions of the *HIP* and the point mass, S_f is spring force, D_f is damper force, I_f is inertia force, P_{KS} is stiffness coefficient of point mass, P_{KD} is damping coefficient of point mass, and P_v is velocity of point mass. We then perform a simple Euler integration of the point mass state, where a is acceleration and d_t is delta time:

$$
\begin{aligned}
a &= I_f / P_m; \\
P_v &= P_v + (a * d_t); \\
P_{pos} &= P_{pos} + (P_v * d_t);
\end{aligned}
\qquad (32)
$$

We can then instruct the haptic probe to add the components of the inertiaForce vector to the rendering force. This implementation makes use of a callback HL_CALLBACK_COMPUTEFORCE which executes at every timestep and allows the user to modify current forces based on any custom force that the programmer would like to implement.

As a result, the system serves as an experimental framework to analyze haptic experiments for fluid simulations. Some drawbacks of this proposed approach include limitations on the force values that can be rendered, tradeoffs on real-life physics, and restrictions on the grid size of the simulation.

4.2 Free-View Volumetric Visualization

A three-dimensional fluid simulation produces volumetric data. Volumetric rendering is a technique that is very costly in terms of processor time, and which also requires a great deal of memory to store the data used. In order to visualize the fluid simulation, different approaches were considered for the rendering process, taking into account the limitations of OpenGL rendering capabilities. This section presents some of our experiments such as voxel raytracing and semi-transparent grid rendering that were not adequate enough for our goals. We then devised a variant of camera-aligned slices to achieve our objective.

Voxel Raytracing variant

Similar to the method explained in section 2.1.1 Voxel Rendering fills the scene occupied by the 3D grid by rendering a considerable amount of points. The amount of color for each point is determined by a linear combination of the density values of each of their grid-cell neighbors. However, this is not a good approach as the rendering appears dusty and discontinuous. In addition, it reduces performance because of the computations needed for rendering each point.

Semi-transparent grid rendering

We render smaller alpha transparent cubes that are sorted based on their distance to the camera. The density value of a grid vertex is used as a cue for its color. The higher the density, the brighter the color is. Each small cube face interpolates the color of the four vertices that form it. The result, as shown in Figure 38, allows the user to perceive the simulation in any three-dimensional angle. However, OpenGL does not support a direct interface for rendering translucent (partially opaque) primitives. Therefore, transparency effects may be created with the blend feature and carefully ordering the primitive data. When using depth buffering in an application, the order in which primitives are rendered is important. Fully opaque primitives need to be rendered first, followed by partially opaque primitives in back-to-front order. If objects are not rendered in this order, the primitives, which would otherwise be visible through a partially opaque primitive, might lose the depth test entirely, as shown in Figure 39.

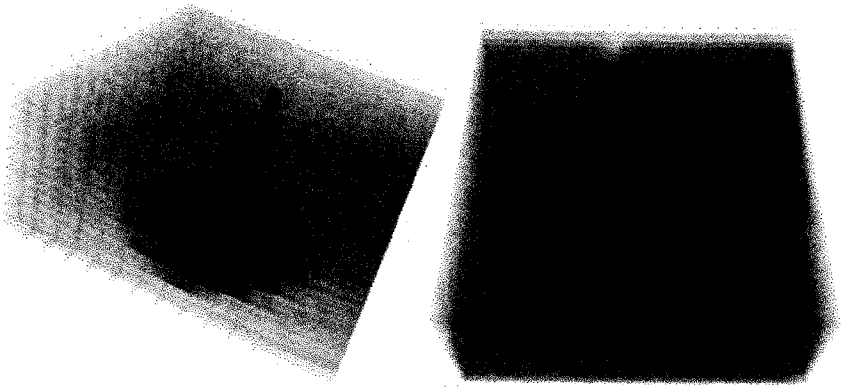


Figure 38: Smaller alpha transparent cubes are sorted based on their distance to the camera. Each small cube face interpolates the color of the 4 vertices that form it.

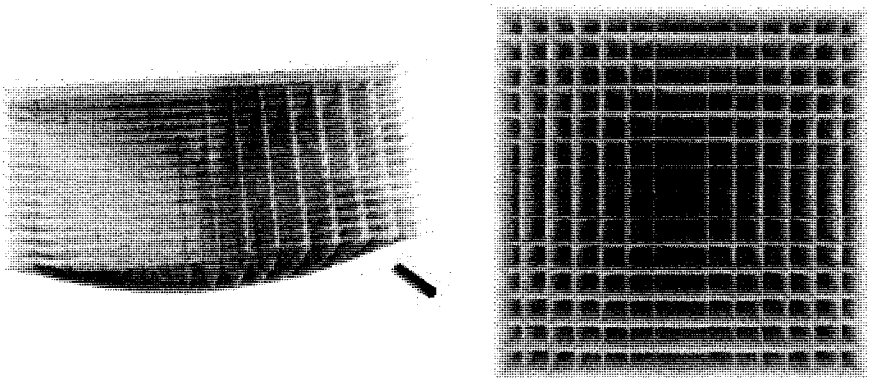


Figure 39: Rendering artifacts produced by alpha-blending when faces are not sorted by distance to the camera.

In order to properly sort faces, we need to port grid coordinates from the "local space" to a "world space". To do this, we need to obtain OpenGL's *ModelView* matrix, and multiply each vertex by the transformations present in this matrix. The *ModelView* Matrix, as shown in Figure 40, is present in OpenGl for the purpose of transforming vertices into eye coordinates. OpenGl then multiplies these coordinates with the Projection Matrix to obtain the 2D screen coordinates of the objects represented in the scene.

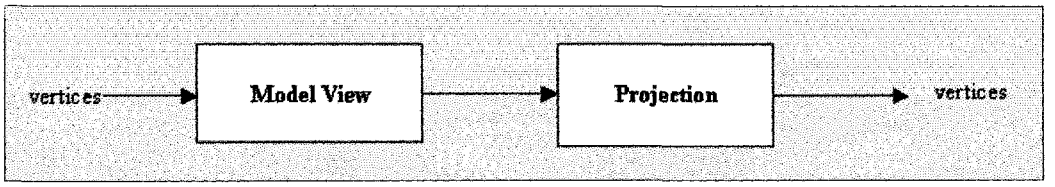


Figure 40: OpenGL's transformation matrices.

We can obtain the corresponding matrix by invoking *glGetDoublev(GL_MODELVIEW_MATRIX, mvm);* and storing it in *GLdouble mvm[16].* Let us take into account that *RotationFromMatrix(mvm)* will return the fields m[0], m[1], m[2], m[4], m[5], m[6], m[8], m[9], m[10], from the ModelView Matrix[16]; while, *TranslationFromMatrix(mvm)* will return the fields $[m[12], m[13], m[14]]$ from the same matrix. For each particle we can then get its world-space coordinate with the following formula:

*WorldCoord=RotationFromMatrix(mvm)*ObjectCoord +TranslationFromMatrix(mvm);*

The following summarizes this experimental rendering algorithm:

1. Describe Vertex Connections: At system start, we store each vertex position in an array and we specify for each vertex the number of faces it is connected to. There are four types of vertices in our cubical grid:

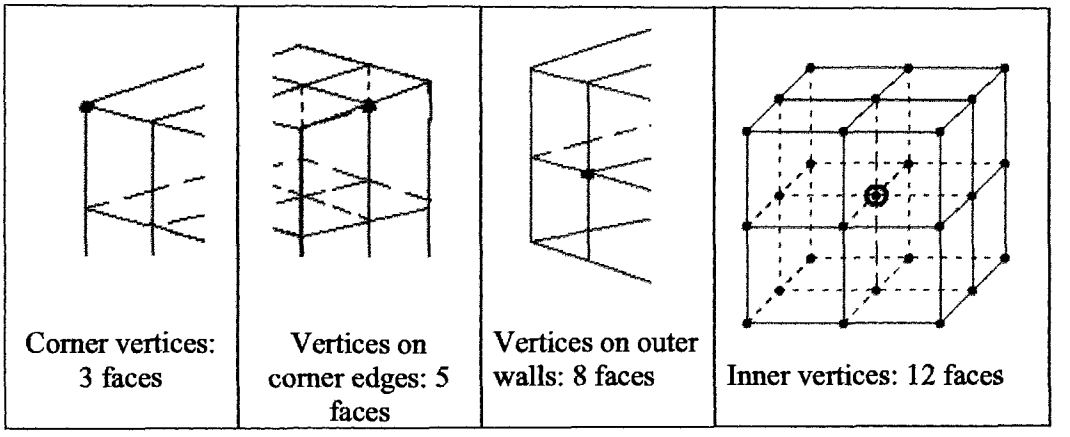


Figure 41: Types of Vertices in a Cubical Grid.

2. Convert each vertex position to "world-space" coordinates using the formula presented above.
3. Sort vertices by their current "depth" coordinate.
4. For each sorted vertex, from back-to-front:
    a. Draw the faces that are connected to the vertex (see Figure 41).
    b. To avoid flickering, flag each drawn face so that it does not get rendered a second time.

The result is a semi-transparent grid in which the fluid simulation is shown. Colors are interpolated in-between faces giving a smooth visualization of the fluid flow. However, this approach still shows the walls of the three-dimensional grid, as shown in Figure 38 (Left). Therefore, it is not suitable for our graphical rendering purposes as we would like to have a smoother looking fluid. As a result, we devise a camera-aligned deformed texture rendering approach in order to achieve our smooth graphical rendering.

CAMERA-ALIGNED DEFORMED TEXTURE RENDERING

This approach consists of slicing the 3D grid into planes that are parallel to the camera view (see Figure 43). We apply gradual alpha transparency to the slices and sort them based on their distance to the camera.

To get our planes, we first find the cube vertex that is closest to the viewer. Our slices are then defined by the plane equation $a*x + b*y + c*z + d = 0$ where (a,b,c) is the plane normal given by the view direction; and d is the parameter along the view direction. The first d is given by inserting the previously found vertex into the plane equation. For each subsequent slice, we get the intersection points of all cube edges with the given plane that lie within the cube. We sort these points to get a convex polygon and draw it on the screen. The result will be a set of camera-aligned planes that will be used to bind a 3D texture representing the state of our fluid simulation.

As shown in Figure 42, a 3D texture is a series of (width * height * depth * bytes per texel) bytes where width, height, and depths are powers of 2. It may be thought of a series of two dimension textures where an extra parameter (depth, the R texture coordinate) specifies which 2D texture will be used. Notably, like other texture coordinates, the R-coordinate is not required to be an integer, the (closest 2 texels) ^ (3 dimensions) = 8 texels will be accounted for in the calculation based on the selected filter scheme.

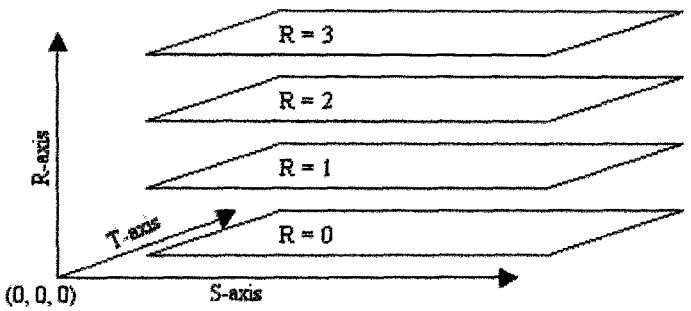


Figure 42: Architecture of a 3D texture.

3D texturing is not part of the standard OpenGL libraries and has to be loaded at run-time. Fortunately, "glext.h" defines a pointer type called PFNGLTEXIMAGE3DPROC specifically for this purpose. After setting up 3D texturing we can use the glTexImage3D function to create our textures. Similar to creating a 2D texture object, the steps to creating an OpenGL 3d texture object are as follows:

1. Load or generate the texels.
2. Get a name for the texture from OpenGL.
3. Bind to this name.
4. Specify the wrapping, filtering, and any other parameters.
5. Finally, call glTexImage3D.

At every time-step we construct a 3D texture based on the fluid simulation data, and proceed to texture the slices with this information. The texture color is based on the density of the fluid grid cells. The higher the density, the brighter the color is. The texture alpha channel is also adjusted based on density values during the simulation. The lower the density, the more transparent the color is. However, we also need to account for surface deformations. Therefore, we modify this algorithm in order to simultaneously visualize surface changes. As the haptic probe injects forces that deform our surface, the applied energy is damped when it travels through the inner grid layers. As shown in Figure 43, we

compute these changes in deformation, according to the algorithm described in section 3.2.2 and alter our textures accordingly. We then render the surface haptically to enable user interaction, and visually show the deformations.

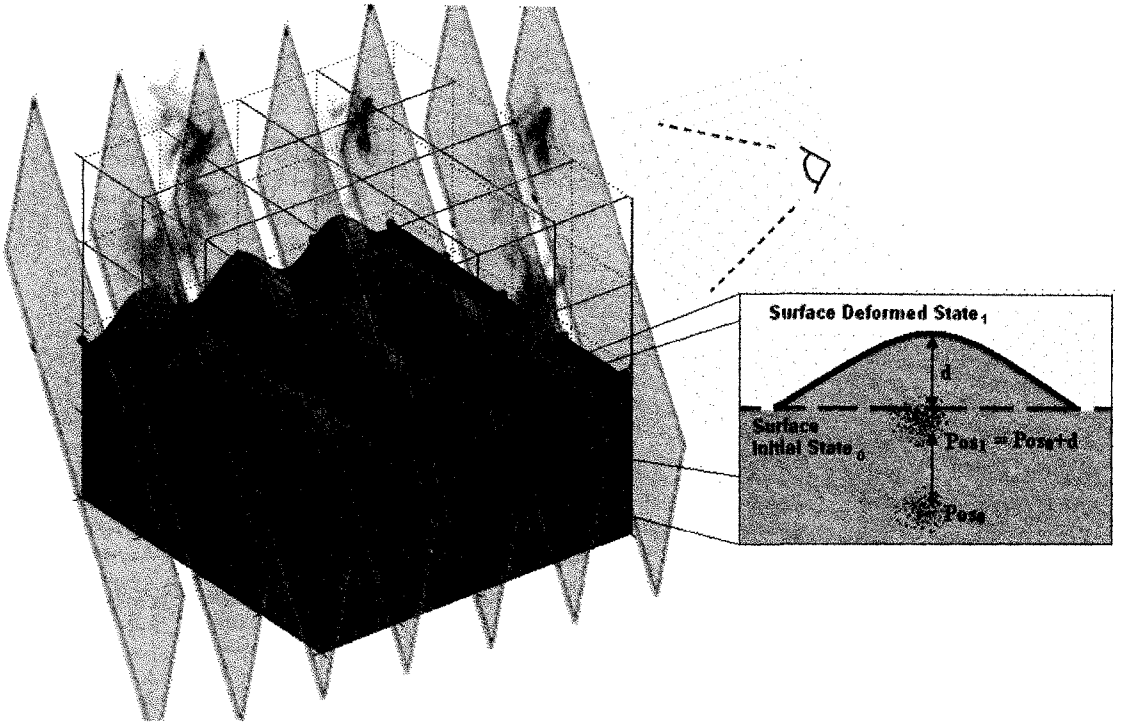
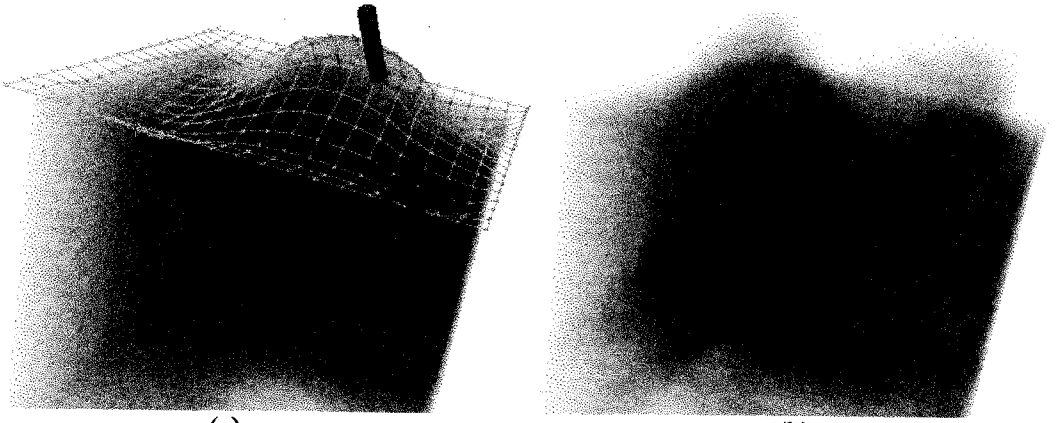


Figure 43: Texture represents changes in fluid caused by surface deformations.

As shown in Figure 44 (left), a keyboard button lets us toggle the visualization of the deformed mass-spring system. Notice that a damped fluid deformation follows the surface oscillations in Figure 44 (right). As the user rotates the scene, the slices are re-aligned so the user is able to perceive the simulation in any three-dimensional angle.



(a) (b)

Figure 44: Samples of our rendering approach results. (a) With surface guide. (b) Without surface guide.

The previous two sections presented our approach to render the fluid simulation in both visually and haptically manners. To summarize as an example, the following table presents a contrast between our presented approach and that of Dobashi's and his team [DSH*06], mentioned in section 2.4.

Table 2: Comparison with Dobashi's Approach.

Dobashi et al. Approach	**Our Approach**
Forces are pre-computed and stored in a Database. Pre-computing is biased as it is hard to store and predict every possible outcome of a simulated fluid.	Forces are computed in real-time based on a grid-based numerical fluid simulation. Therefore, output forces correspond to real-time calculations.
Surface represented by HeightField approach. It only permits one height value at each location on the plane.	Surface represented by a deformable surface, which permits multiple height values at a location on the plane.
Intended to be used with large haptic setup equipment only available to research labs.	Intended to be used on personal computers or laptops with desktop haptic devices available on the mass market.
Applications need to be accustomed to their large and specific VR setup.	Ease of setup is beneficial for more general and marketable desktop applications such as video-games.

In addition, the following table also summarizes the differences between our presented approach and that of Baxter's and Lin's [BL04].

Table 3: Comparison with Baxter and Lin's Approach.

Baxter and Lin's Approach	Our Approach
Real-time haptic display of fluids in *two dimensions* by directly solving Navier-Stokes equations.	Real-time haptic display of fluids in *three dimensions* by solving Navier-Stokes equations.
Method does not explore volumetric haptic interaction, nor addresses 3D issues like volumetric rendering.	We explore volumetric rendering of fluids and their haptic interaction in a volumetric space.
The HIP is modeled as a discretized object by using grids, causing undesirable noise in the resulting force.	The HIP is treated as an extra particle in the simulation which offers more control over the rendering of forces.
Their painting example presents blobs of viscous fluid on a two-dimensional plane in a three-dimensional space. It does not account for surface waves.	Our videogame example presents a fluid in motion in volumetric space along with its deformable surface, for a more fluid-immersive experience.
Their application only renders forces that result from brushing paint blobs on a surface. It does not render fluid currents.	We additionally explore haptic ambient forces to represent differences in fluid densities across a volumetric fluid flow.

The next section discusses the coupling between both routines and how the application needs to handle their integration.

4.3 Integration of Hapto-Visual Rendering Routines

In order to combine our haptic and graphic rendering approaches, we need to integrate functionality from C++, OpenGL®, OpenHaptics, and a 3Ds Loader Module to load the 3D mesh of our stone bowl [AUTODESK]. OpenHaptics, introduced by Sensable

[SENSABLE], is the main SDK used to code Haptics in our system. The OpenHaptics toolkit includes the Haptic Device API (HDAPI), the Haptic Library API (HLAPI), Utilities, and the PHANTOM Device Drivers (PDD).

The HDAPI provides low-level access to the haptic device, enables haptics programmers to render forces directly, offers control over configuring the runtime behavior of the drivers, and provides convenient utility features and debugging aids. The HLAPI provides high-level haptic rendering and is designed to be familiar to OpenGL API programmers. It allows significant reuse of existing OpenGL code and greatly simplifies synchronization of the haptics and graphics threads. The PHANTOM Device Drivers support all currently shipping PHANTOM devices.

HLAPI follows traditional graphics techniques such as those found in the OpenGL API. Adding haptics to an object is a fairly trivial process that resembles the model used to represent the object graphically. Tactile properties, such a stiffness and friction, are similarly abstracted to materials. The HLAPI also provides event handling for ease of integration into applications.

For example, when using HDAPI, creating a haptic/graphics sphere involves writing the graphics code and creating scheduler callbacks for handling the sphere forces. When using HLAPI, the process involves creating a graphics sphere then calling *hlBeginShape()* with the desired haptic shape type when drawing the graphics sphere. A typical HLAPI program has the structure of Figure 45.

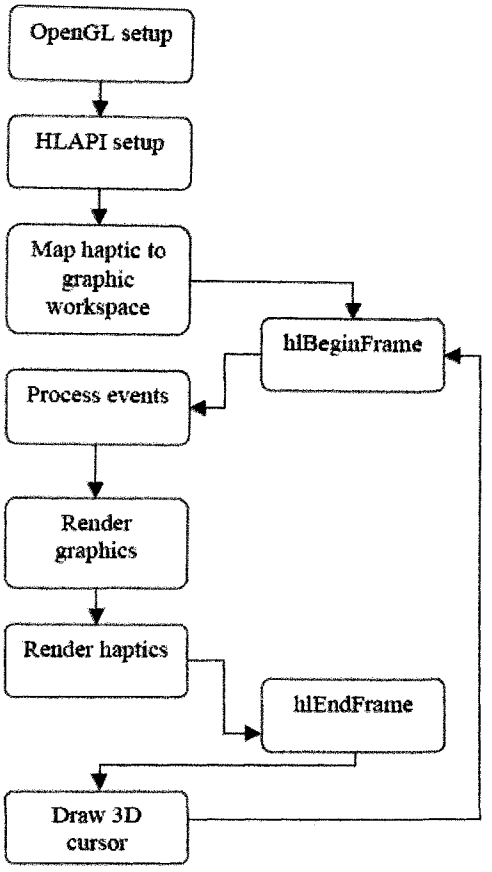


Figure 45: HLAPI Program Structure.

First, the program sets up OpenGL by creating a graphics rendering context and tying it to a window. Then it initializes the HLAPI by creating a haptics rendering context and tying it to a haptic device. Then the program specifies how the physical coordinates of the haptic device should be mapped into the coordinate space used by the graphics. This mapping is used by the HLAPI to map geometry specified in the graphics space to the physical workspace of the haptic device, as explained in the previous chapter. Next, the application renders the scene graphics using OpenGL. Then the program processes any events generated by the haptics rendering engine such as contact with a shape or a click of the stylus button. Then the haptics are rendered, usually by executing nearly the same code

as for rendering the graphics, but capturing the geometry as a depth or feedback buffer shape. In addition to rendering scene geometry, a 3D cursor is rendered at the proxy position reported by the HLAPI. Finally, the rendering loop continues by rendering the graphics again. Forces can also be visually appreciated by looking at the color of the baton during the simulation. These colors are dynamically modified according to the rendered forces. Users are able to associate the physical force feeling with the visual cue: a green shaded baton for light forces, a yellow color for moderate forces, and red for strong forces. In a typical program, *hlBeginFrame()* is called at the top of the rendering loop, so that any objects in the scene that depend on the haptic device or proxy state have the most current data. *hlEndFrame()* is called at the end of the rendering loop to flush the changes to the haptic rendering engine at the same time that the graphics are flushed so that two will be in synch. At the start of the haptic frame, *hlBeginFrame()* samples the current haptic rendering state from the haptic rendering thread. *hlEndFrame()* will commit the rendered haptic frame and will synchronously resolve any dynamic changes by updating the proxy position. The following is a summary table of the main differences between HD and HL.

Table 4: Main differences between HDAPI and HLAPI.

HD	HL
• A low-level foundational layer for haptics rendering • Enabling to send force and read ADC manually. • Modifying the rate of servo loop	• Designed for high-level haptics scene rendering. • Built on top of the HDAPI • Providing a higher level control of haptics than HDAPI. • Designed for ease of use

4.4 Validation

We conducted a study to validate our results and get various users perspectives on our system. Ten adults participated in the study. Most were students at our university, from Master and Ph.D programs. A few had technical backgrounds of haptics: two had heard of the concept of haptics but had never used a device before, six had used haptic devices before but had no programming knowledge of them, and two had programmed with haptics before. Of the 10 participants, 2 had never heard of fluid simulations before, 7 knew about them but had no knowledge of their techniques and 1 was familiar with common simulation methods. All the participants were fairly comfortable with using computers and were able to follow the assigned tasks thoroughly. The participants were presented with ten tasks to perform in our application. They were also presented with a list of different keyboard commands that they could use at any time to toggle between the different functionalities of the system:

Left-click mouse and drag around to change camera viewpoint
'ESC' or 'Q' – Quit Program
'1' – add substance type 1 (red)
'2' – add substance type 2 (green)
'P' – toggles pop-through surface (apply enough force to enter inner fluid)
'A' – toggles bowl visibility
'S' – toggles visibility of deformable surface
'D' – toggles haptic rendering of density values
'F' – toggles haptic rendering of fluid flow (current)
'G' – toggles recognition of gestures

After a task was performed, they rate the hapto-visual experience from 1 to 5; 1 being the lowest and 5 the highest rating. In addition they could give open comments on their experience and offer ideas for future features. The following table summarizes the results of the survey.

Table 5: Summary of survey findings.

Task\Rating	1	2	3	4	5	AVG.
• Colliding with the walls of the cauldron.	0	0	1	3	6	4.5
• Colliding with the deformable surface.	0	0	2	3	5	4.3
• Feeling the surface ripples.	0	0	3	5	2	3.9
• Feeling the viscosity of the fluid.	0	0	0	5	5	4.5
• Feeling the current generated by the fluid flow.	0	0	4	1	5	4.1
• Visualizing the mixture of multiple substances.	0	0	0	3	7	4.7
• Visualizing the motion of the fluid.	0	0	0	4	6	4.6
• Performing haptic gesture recognition.	0	0	0	3	7	4.7
• Experiencing fluid parameter changes when recognition is successful.	0	0	1	3	6	4.5
• Overall Application as an interactive tool to haptically and visually represent a fluid simulation of multiple substances.	0	0	0	6	4	4.4

A first-time haptic user was surprised by how much more satisfying and engaging a user interaction can be when more than one sense is involved. The audience commented on how well the combination of fluid modeling and force feedback made for a realistic interaction with the scene. The highest average score was related to the visualization of multiple substances being mixed and the ability of the system to recognize their gestures. The lowest average score was related to the feeling of surface ripples. However, we think that this was mainly due to the participant not introducing enough forces that could generate ripples and oscillations. If the fluid surface were to be at rest, there would be no ripples or oscillations to feel and not many users generated them in a large amount to feel them clearly. Differences in participant answers may also be due to the system's lack of usability guides such as GUI menus and ease of navigation. Overall, the participants enjoyed feeling the physical characteristics of the fluid simulation in real-time and we determined that this added a dimension of interaction that traditional interfaces do not offer.

Chapter 5 Enhanced User Interaction Application – Gesture Recognition

Figure 46: Game Scenario

This chapter describes how this haptic interactive system can be enhanced by integrating it with haptic gesture recognition. If a gesture is recognized, different simulation parameters can be changed automatically within the same system. This is not our main research for this thesis, but it is a further step to show the potential of the haptic-game interaction where a user's motion and behavior may serve as input for a controlled virtual world.

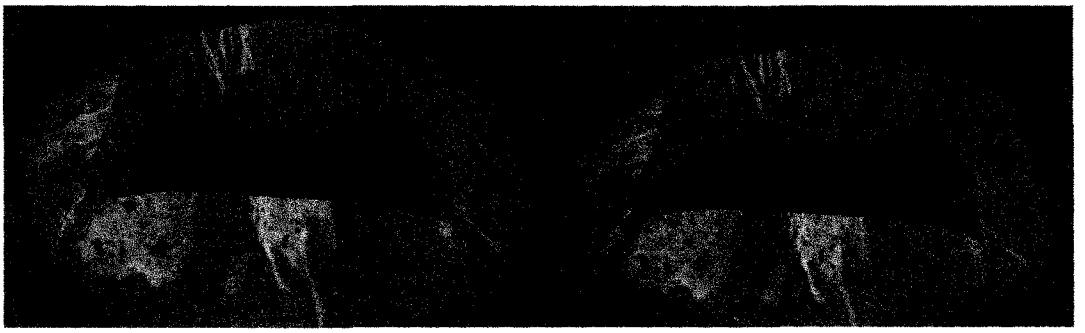


Figure 47: Creation of the master gesture and the comparable gesture.

A possible application can be a game situation in which the player impersonates the role of a witch, as shown in Figure 47. Following a specific recipe, a magic potion needs to be created. It would require the right ingredients, mixed at the right moment, with the proper stirring movements and force. Once the player succeeds, the system is able to trigger customized modifications to the properties of the fluid simulation and the deformable surface. The fluid might change color, viscosity and elasticity parameters among other characteristics. In addition, it also enables to trigger another fluid body simulation such as

smoke buoyancy on top of the liquid. The decision might be made through a haptic motion recognition module as one of possible ways that will allow game developers to take full advantage of the high degree-of-freedom input capabilities of modern haptic devices.

Haptic devices provide more valuable parameters (force, torque, velocity, etc.) than conventional graphics users interfaces. It does allow us not only to recognize 3D coordinates, but also to use force-feedback data as extractable features. These parameters are used to raise the recognition rate of user motions. For instance, a harsh circular movement will be recognized differently than a gentle circular movement. Even though these are both circular movements, different forces were applied. Haptic biometric behavioral applications [OE05] show the importance of force and torque for the purpose of recognition. We present how to recognize a few simple figures, also known as gestures, which would trigger the right potion spell, for instance, three consecutive circular motions or the shape of a star. This would reduce the complexity of the task, and therefore it would be more feasible for the recognition to be performed in real-time, parallel to the haptics and graphics fluid simulations.

The recognition module has two main steps; (i) train and (ii) test. For each step, we perform feature extraction and classification, which are used for similarity measurement. The system is prepared in advance with a train set as a off-line process and whenever a user performs a test motion, the similarity is measured between the train motion and the test motion to decide if it is the correct motion or not. Our implementation follows Dopertchouk's concepts of motion recognition [DOP07] and is organized in three major steps: Creation and storage of the master gesture templates as a train set, normalization of

the test strokes as feature extraction process, and recognition of the test motion shapes as classification process.

5.1 Creation and Storage of the Master Gesture Templates

The gesture templates are recorded from predefined sample haptic inputs. We read the proxy position of the haptic device and store the 4D coordinates as a sequence of points in the workspace, mainly x, y, z and force data (see Figure 47). When players stir the potion mix, some of the gesture shapes may be different in size, speed, or position. Even thought the shape results might look similar to the naked eye, these shapes would look like completely unrelated sets of coordinates to the computer. Therefore, we need to normalize the captured strokes, as shown in Figure 48.

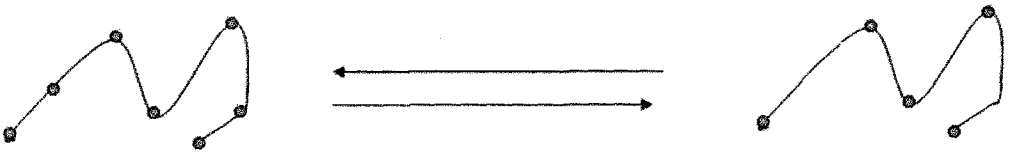


Figure 48: Gestures must be normalized in order to have unbiased comparisons.

In our system, the user is able to record a train gesture by pressing the white button of the haptic device. As shown in Figure 47 (left), the recorded gesture will appear in green, at the center of the screen.

5.2 Normalization of the Strokes

First, we need to scale the gesture to a predetermined size (e.g. 1 unit). We do this by finding the length and dimensions of the gesture, and dividing its coordinates by the scaling

factor. Second, we need to put the individual points in the gesture at a uniform distance. We do this through a dynamic time warping algorithm [MR81]. Since we are interested in the geometric shape in this specific application, it would be irrelevant to know how fast or slow the gesture was drawn. Finally, we need to center the gesture at the origin of the coordinate system through a translation matrix, as shown in Figure 49.

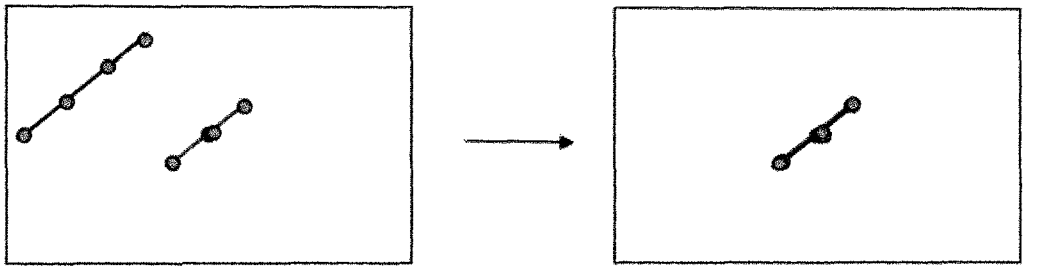


Figure 49: Center the gesture at the origin of the coordinate system.

5.3 Recognition of the Test Motion Shapes

We compare two different approaches for the haptic recognition of the gestures: a neural network, and a simpler linear algebra method.

5.3.1 Neural Network

Simple shape recognition was performed through the implementation of a neural network-based recognition engine, using an approach similar to others [BOU07] [MT91], whom also provide good introductions to neural networks. A similar feature extraction procedure was used. However, haptic proxy positions were converted to directional vectors (e.g. Right: 1,0,0; 1,0,0; ...). A back-propagation algorithm was used to train the neural network with a few basic shapes, run as many epochs and find the minimum sum-of-squares error (SSE) [BIS95] constraint. However, this method proved to be cumbersome to perform

in respect to the additional marginal benefit that we would get in the recognition phase. It would be more time-consuming to integrate a new predefined gesture into the system, as the network would need to be retrained.

5.3.2 Dot Product

A simpler linear algebra method, dot product between two vectors, also provide the measure the similarity between train vector and test vector. Since both our gesture templates and captured strokes have the same number of points after normalization, we model our gestures as normalized vectors. These are 4XN dimensional vectors, where N is the number of points in the gesture. Using this technique, if you compare two normalized vectors that are exactly the same, the result will be one. The result will be a value slightly less than one for vectors that point in more-or-less the same direction, and the result will be a low number for vectors that point in different directions. This dot product approached worked well for simple shape matching and it did not slow down any of the haptic fluid nor the deformable surface computations.

5.4 Recognition Results

From a set of three basic motions (e.g., circle, V shape, and S shape), this module was able to reach recognition rates above 95% for both recognition approaches. In our game scenario, the dot product approach seemed effective enough to recognize potion shapes. Neural networks also provided acceptable gesture recognition rates, but the time allocated for network retraining is cumbersome and tedious for gaming purposes.

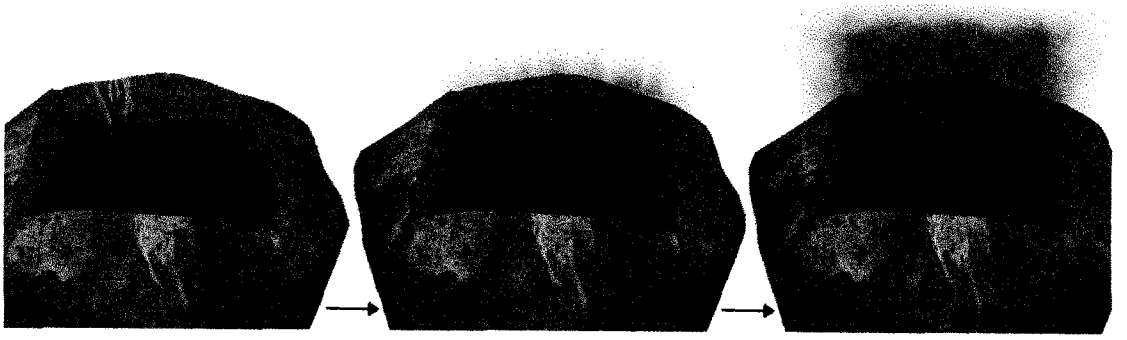


Figure 50: Recognition triggers changes in fluid parameters (e.g., fluid viscosity increase) plus appearance of buoyant smoke.

SIMULATION AFTER RECOGNITION

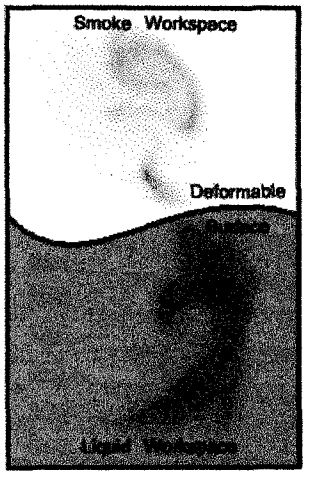


Figure 51: Different fluid workspaces.

Once the gesture has been successfully recognized, we can change the parameters of the simulation on-the-go (see Figure 50). We can make the fluid more viscous, change its colors, and make the deformable surface less elastic so it resembles thick clay. We also trigger the simulation of smoke above the surface as our system is flexible enough to accommodate multiple fluids as long as the speed permits real-time calculation.

The smoke in the air is a separate fluid simulation that takes buoyancy into account. Buoyancy is the upward force on an object produced by the surrounding gas. The smoke also follows the same rendering approach as the fluid, creating a separate 3D texture and binding it to camera-aligned slices. As shown in Figure 51, both liquid and smoke workspaces are combined with the deformable surface in between them. When we trigger the gas simulation, grey substances are added to the lower layer of the smoke workspace and an upward velocity continuously raises them in the air simulating buoyancy. This velocity

can be represented as: $v_y = v_y + d * b_k$, where v_y is the current velocity component in the *up* direction, d is the amount of grey substance and b_k is the buoyancy constant. We are able to change the simulation parameters to represent different scenarios. In our example application, the liquid workspace shows a high viscosity of 0.6 while the smoke shows a low viscosity of 0.1 with a buoyancy constant of 0.2. The result in this example scenario is a thick movement of fluids under the surface, and a light movement of smoke above the surface. The source location of smoke density is randomly generated on the lowest layer of the smoke workspace near the surface. This is beneficial as it generates a nice visual effect in which smoke is sparse across the airspace of the system, as shown in Figure 52.

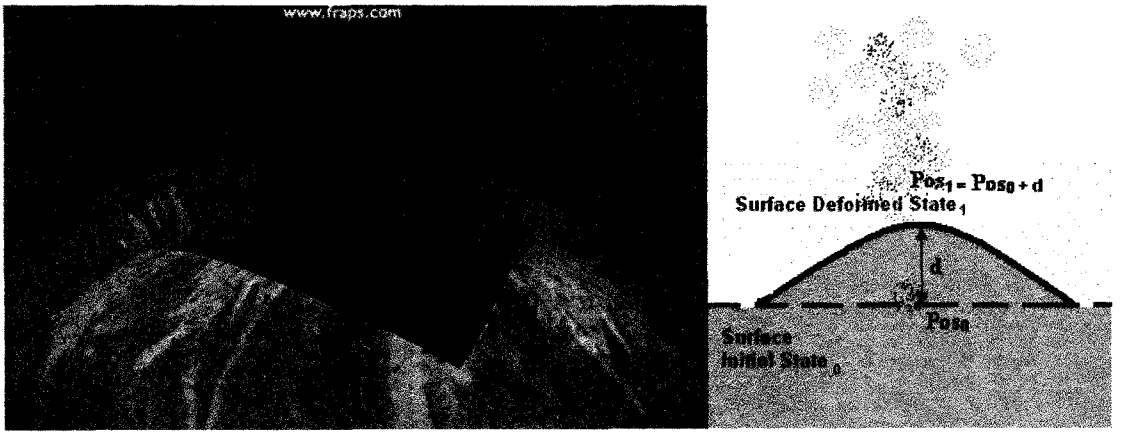


Figure 52: Smoke above the surface.

The smoke workspace relates to the fluid workspace in that they share the surface deformation information. As shown in Figure 52 and in Figure 53, the smoke source location is also translated based on the surface deformation information in the same manner we explained for the inner fluid. In this manner we can think of the deformable surface as the border between the fluid workspace and the smoke workspace. Figure 53 (left) shows the different workspaces being integrated together. While the deformable surface sits on top of the liquid workspace, the smoke surface sits on top of the deformable surface. Initially, as

there are no deformations, the smoke source comes from the surface at rest. Once deformations are introduced by the haptic device, the source gets translated based on the damped surface oscillations, as perceived on the right of Figure 53.

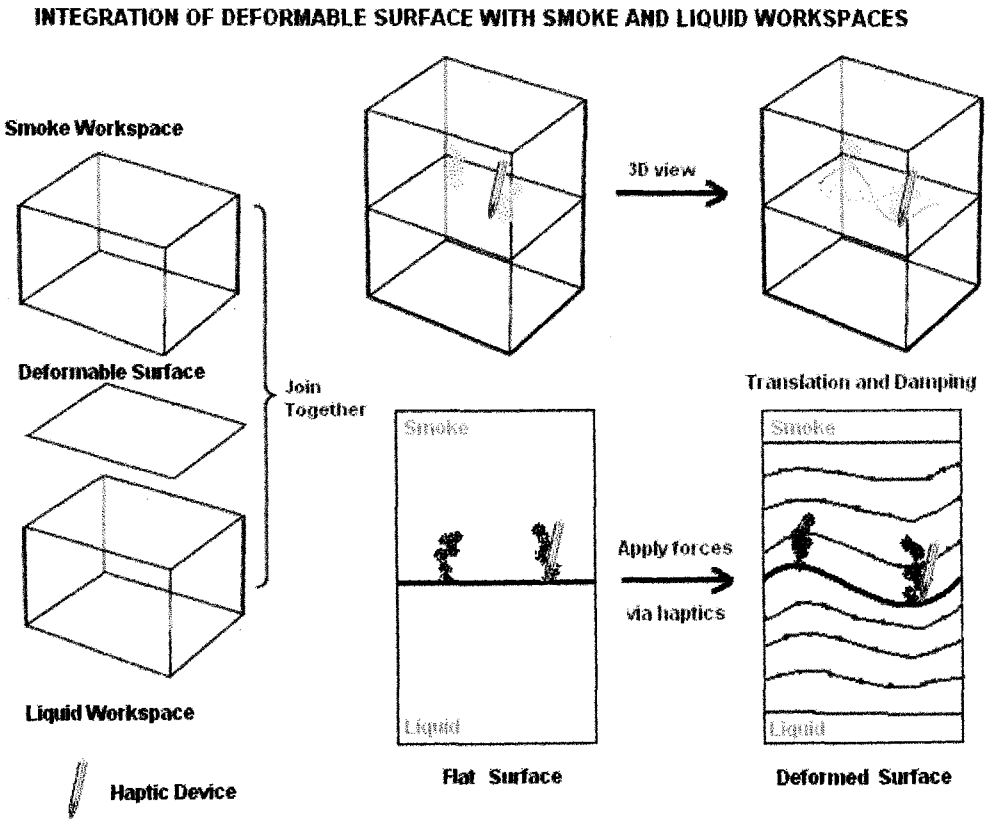


Figure 53: Surface integration with fluid and smoke workspaces.

Chapter 6 Conclusions

We have presented a novel human-computer system based on haptic-fluid interaction. Based on the Navier-Stokes equation, the system simulates three-dimensional fluid flow in real-time. Different calculation grids are maintained to enable the mixture of multiple substances. A discreet mass-spring particle system represents the deformable surface. In order to interconnect the surface with the fluid and retain the deformation within the bowl, the applied energy is damped when it travels through the inner grid layers. Both graphics and haptics workspaces are coupled in order to match the hapto-visual interaction. The system haptically renders physical forces generated by the stirring movements and fluid density changes. In viceversa, the device inserts back forces that modify the fluid flow. In addition, we have presented a volumetric rendering approach to couple the three-dimensional fluid with its deformed surface. We made use of lighting, blending, and shading effects to appreciate the animated fluid ripples. As an example application, we have demonstrated a game scenario combined with haptic gesture recognition along with multiple fluids co-simulations and automatic fluid parameter updates. Finally, our user-study showed haptic-interaction with virtual fluid pioneering a new attractive area.

This chapter concludes the research documented in the preceding chapters. Section 6.1 lists the contributions made by this thesis. Section 6.2 discusses the limitations of the proposed methodology, and Section 6.3 suggests future research.

6.1 Contributions

The following contributions were made in this thesis:

- Developed a real-time 3D fluid simulation.
- Developed a haptic user interface with fluid as a way to enhance Human-Computer interactions in a wide-range of applications.
- Developed techniques to haptically render the properties of a fluid in motion by representing forces that originate from the Navier-Stokes equation.
- Showed a volumetric rendering approach to couple a three-dimensional fluid with its deformed surface.
- Developed a technique to simulate the mixture of multiple substances into a fluid simulation.

6.2 Discussion

The proposed scheme has many benefits, including the ability to see and feel the properties of a three-dimensional fluid simulation in real-time. However, the maximum amount of forces that desktop haptic devices are capable of rendering may not reach the intensity of those found in the real world. As a consequence, forces need to be scaled and results become approximations of real world parameter values. Secondly, the system complexity should be simplified in order to keep up with real-time requirements. This results in less accurate rendering as we trade off accuracy for real-time speed. Additionally, grid size resolution also exhibits a trade-off between rendering quality and computational complexity. Therefore, resolution should be dynamically selected in a way for the system to provide users with an acceptable rendering, in which hardware-dependent computational

time fits within frame time. This offers better rendering quality for users with more powerful hardware systems; however, it is considered to be a limitation for users working in less powerful machines.

6.3 Future Research

The research documented in this thesis lays the groundwork for many areas of future research. Many topics may be pursued to extend this work.

- Haptic gesture recognition: The orientation and workspace of the Phantom series of haptic devices allow the users to make natural, human gestures using a stylus. In the future, it would be interesting to keep exploring the haptic gesture recognition module of the project to produce more various effects on the fluid within game scenarios. In our wizard story, even the idea of waving a haptic stylus through the air in order to cast spells is appealing in that it makes the player feel as if they really are wizards.
- Construction of breakable fluid surfaces: It is to generate splashes and fluid spills for more dynamic fluid simulation. Alternate non-grid algorithms could be used such as smoothed particle hydrodynamics (SPH).
- Fluid interaction with higher boundary treatments.
- GUI: Based on the feedback received on our user study, it would be beneficial to implement a more user-friendly UI in order to enhance the navigation of hapto-fluid applications and facilitate the user interaction with the system.

References

[AD03] Ayache N., Delingette H.: Surgery Simulation and Soft Tissue Modeling. *Proceedings of International Symposium IS4TM'03*, 2003.

[AML*06] Andrews S., Mora J., Lang J., Lee W.S: HaptiCast: A Physically- Based 3D Game with Haptic Feedback. *In Proceedings of FuturePlay*, 2006.

[AN05] Angelidis A., Neyret F.: Simulation of smoke based on vortex filament primitives. *In SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 87–96, New York, USA, 2005.

[AND95] Anderson J., JR.: Computational Fluid Dynamics, *McGraw-Hill* (USA), 1995.

[ANS*06] Angelidis A., Neyret F., Singh K., Nowrouzezahrai D.: A controllable, fast and stable basis for vortex based smoke simulation. *In ACM-SIGGRAPH/EG Symposium on Computer Animation (SCA)*, 2006.

[AS96] Avila R. S., Sobierajski L. M.: A haptic interaction method for volume visualization. *Proceedings of Visualization'96*, pages 197–204, 1996.

[AUTODESK] Autodesk, Available at http://autodesk.com.

[BAS07] Basdogan, C.: Haptic Rendering Tutorial, Available at http://network.ku.edu.tr/~cbasdogan/Tutorials/haptic_tutorial.html, 2007.

[BB01] Baker D., Boyd C.: Volumetric Rendering in Real-time. *Proceedings of the 2001 Game Developers Conference*, 2001.

[BBB07] Batty C., Bertails F., Bridson R.: A fast variational framework for accurate solid-fluid coupling. *In SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 100, New York, NY, USA, ACM Press, 2007.

[BFA02] Bridson R., Fedkiw R., Anderson J.: Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics (SIGGRAPH'02)*, 2002.

[BM07] Bridson R., Muller-Fischer M.: Fluid Simulation. *In proceedings of ACM SIGGRAPH 2007* Course Notes, 2007.

[BGO*06] Bargteil A., Goktekin T., O'brien J., Strain J.: A semi-lagrangian contouring method for fluid simulation. *ACM Trans. Graph.*, 25(1):19–38, 2006.

[BHN07] Bridson R., Houriham J., Nordenstam M.: Curl-noise for procedural fluid flow. *In Proceedings of the 2007 SIGGRAPH Conference.* Volume 26, Issue 3, 2007.

[BIS95] Bishop C.: Neural Networks for Pattern Recognition, *Oxford Press*, USA, 1995.

[BL04] Baxter W., Lin M.C.: Haptic Interaction with Fluid Media. *Proceedings of Graphics Interface*. Vol. 62, pp.81-88, Canada, 2004.

[BLI82] Blinn J.: A Generalization of Algebraic Surface Drawing. *In proceedings of ACM Transactions on Graphics*, Vol. 1, No. 3, pp. 235-256, 1982.

[BOU07] Boukreev K.: Mouse gestures recognition. Available at http://codeproject.com/cpp/gestureapp.asp, 2007.

[BYM05] Bell N., Yu Y., Mucha P.: Particle-based simulation of granular materials. *In SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation,* pages 77–86, New York, NY, USA, ACM Press, 2005.

[CB94] Colgate J. E., Brown J. M.: Factors affecting the z-width of a haptic display. *IEEE Conference on Robotics and Automation*, pages 3205–3210, 1994.

[CBM*05] Conti F., Barbagli F., Morris D., Sewell C.: CHAI: An Open-Source Library for the Rapid Development of Haptic Scenes. Paper presented at *IEEE World Haptics*, Italy, 2005.

[CBP05] Clavet S., Beaudoin P., Poulin P.: Particle-based visco-elastic fluid simulation. *In Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, pages 219–228, 2005.

[CD97] Cani-Gascuel M., Desbrun M.: Animation of Deformable Models Using Implicit Surfaces. *IEEE Trans. Visualization and Computer Graphics*, vol. 3, pp. 39-40, 1997.

[CFL67] Courant R., Friedrichs K., Lewy H.: On the partial difference equations of mathematical physics, *IBM Journal*, pp. 215-234, 1967.

[CHO67] Chorin A: A numerical method for solving incompressible viscous flow problems. *J. Comp. Phys.*, 2:12–26, 1967.

[CHUN02] Chung T.: Computational Fluid Dynamics, *Cambridge Univ. Press*, 2002.

[CIR52] Courant R., Isaacson R., Rees M.: On the Solution of Nonlinear Hyperbolic Differential Equations by Finite Differences, *Communication on Pure and Applied Mathematics*, 5, 243-255, 1952.

[CKH*99] Calhoun P., Kuszyk B., Heath D., Carley J., Fishman E.: Three-dimensional Volume Rendering of Spiral CT Data: Theory and Method. *Radiographics*, 19:745-764. Available at http://radiographics.rsnajnls.org/cgi/content/full/19/3/745, 1999.

[CMT04] Carlson M., Mucha P., Turk G.: Rigid fluid: animating the interplay between rigid bodies and fluid. *In SIGGRAPH '04: ACM SIGGRAPH 2004 Papers,* pages 377–384, New York, NY, USA, ACM Press, 2004.

[COL00] Collins G.W, II: Fundamental Numerical Methods and Data Analysis. NASA Astrophysics Data System (ADS) Virtual Library, 2000.

[COR05] Corbett R.: Point-based level sets and progress towards unorganized particle-based fluids. *Master's thesis, University of British Columbia*, 2005.

[CRYSTAL] Crystal Space 3D, Available at http://www.crystalspace3d.org.

[DCH88] Debrin, R., Carpenter, L., Hanrahan, P.: Volume Rendering, *Computer Graphics,* Vol. 22(4), 1988.

[DMW*98] Durbeck L., Macias N., Weinstein D., Johnson C., Hollerbach J.: SCIRun haptic display for scientific visualization. *Phantom Users Group Meetings*, 1998.

[DOP07] Dopertchouk, O.: Recognition of Handwritten Gestures, Available at http://gamedev.net/reference/articles, 2007.

[DQK01] Dachille F., Qin H., Kaufman A.: Novel haptics-based interface and sculpting system for physics-based geometric design. *Computer-Aided Design*, Vol. 33, pp.403-420, 2001.

[DSH*06] Dobashi Y., Sato M., Hasegawa S., Yamamoto T., Kato M., Nishita T.: A Fluid Resistance Map Method for Real-time Haptic Interaction with Fluids. *Proceedings of ACM VRST'06*. pp. 91-99, 2006.

[EFF*02] Enright D., Fedkiw R., Ferziger J., Mitchell I.: A hybrid particle level set method for improved interface capturing. *J. Comput. Phys.*, 183(1):83–116, 2002.

[EMF02] Enright D., Marschner S., Fedkiw R.: Animation and rendering of complex water surfaces. *In SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 736–744, New York, NY, USA, ACM Press, 2002.

[EYS*94] Ebert D., Yagel R., Scott J., Kurzion Y.: Volume Rendering Methods for Computational Fluid Dynamics Visualization. *IEEE Visualization*, 1994.

[FAR07] Farrell D.: Lava lamp using Bryce Metaballs, *M.Sc school project for Northwestern University*, 2007.

[FF01] Foster N., Fedkiw R.: Practical animation of liquids. *In Proc. SIGGRAPH*, pages 23–30, 2001.

[FM96] Foster N. and Metaxas D.: Realistic animation of liquids. *Graph. Models Image Process.*, 58(5):471–483, 1996.

[FM97] Foster N., Metaxas D.: Modeling the Motion of a Hot, Turbulent Gas. In *Computer Graphics Proceedings, Annual Conference Series,* p.181–188, 1997.

[FOA03] Feldman B., O'Brien J., Arikan O.: Animating suspended particle explosions. *In SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 708–715, New York, NY, USA, ACM Press, 2003.

[FR86] Fournier A., Reeves W.: A simple model of ocean waves, *SIGGRAPH 86*, pages 75–84, 1986.

[FRA95] Fraser R., SGI Computer Systems Advanced Systems Division: Interactive Volume Rendering Using Advanced Graphics Architectures. Available at http://web.archive.org/web/20001022092548/http://www.sgi.com/Technology/volume/VolumeRendering.html.

[FSJ01] Fedkiw R., Stam J., Jensen H.: Visual simulation of smoke. *In SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 15–22, New York, NY, USA, ACM Press, 2001.

[GBO04] Goktekin T., Bargteil A., O'Brien J.: A method for animating visco-elastic fluids. *In SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 463–468, New York, NY, USA, ACM Press, 2004.

[GIB95] Gibson S.: Beyond volume rendering: Visualization, haptic exploration, and physical modeling of element-based objects. In *Proc. Eurographics workshop on Visualization in Scientific Computing*, pages 10–24, 1995.

[GSK07] Gourishankar V., Srimathveeravalli G., Kesavadas T.: Hapstick: A high-fidelity haptic simulation for billiards. *In Proc. of 2nd Joint EuroHaptics Conference and*

Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, pp. 494-500, 2007.

[GSL*05] Guendelman E., Selle A., Losasso F., Fedkiw R.: Coupling water and smoke to thin deformable and rigid shells. *In SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 973–981, New York, NY, USA, ACM Press, 2005.

[GSY04] Gosline A., Salcudean S., Yan J: Haptic Simulation of Linear Elastic Media with Fluid Pockets. In Proc. 12th International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS'04), pages 266-271. 2004.

[HAC*04] Hayward V., Astley O.R., Cruz-Hernandez M., Grant D., Robles-De-la-Torre G.: Haptic interfaces and devices. In *Sensor Review*, pp. 16-29, 2004.

[HAR03] Harris J,: Real-time cloud simulation and rendering. *Chapel Hill: The University of North Carolina,* 2003.

[HBS99] Ho C.H., Basdogan C., Srinivasan M.: Efficient point-based rendering techniques for haptic display of virtual objects. Teleoperators and Virtual Environments, pp.477-491, 1999.

[HCT*97] Hollerbach J., Cohen E., Thompson W., Freier R., Johnson D., Nahvi A., Nelson D., Thompson T., Jacobsen S.: Haptic interfacing for virtual prototyping of mechanical cad designs. In *ASME Design for Manufacturing Symposium*, 1997.

[HNB*06] Houston B., Nielsen M., Batty C., Nilsson O., Museth K.: Hierarchical RLE level set: A compact and versatile deformable surface representation. *ACM Trans. Graph.*, 25(1):151–175, 2006.

[HNC02] Hinsinger D., Neyret F., Cani M.P.: Interactive animation of ocean waves. *In Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, pages 161–166, 2002.

[HW65] Harlow F., Welch J: Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Phys. Fluids*, 8:2182– 2189, 1965.

[HWS02] Howe R.D., Wagner C.R., Stylopoulos N.: The role of force feedback in surgery: analysis of blunt dissection. *In Proc. of 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pp. 68-74, 2002.

[IMM07] Immersion Corporation: Surgical simulator, the laparoscopy VR virtual reality system. Available at http://www.immersion.com/medical/products/laparoscopy, 2007.

[IMMERSION] Immersion Corporation, Cyberforce, Available at http://immersion.com

[IN93] Iwata H., Noma N.: Volume haptization. *Proc. of IEEE VRAIS*, p. 16–23, 1993.

[ISS85] Issa R.: Solution of the implicitly discretized fluid flow equations by operator splitting. *In J. Comp. Phys.*, 62:40–65, 1985.

[JP99] James D., Pai D.: ArtDefo—Accurate Real Time Deformable Objects. *Computer Graphics (Proc. Siggraph 99),* ACM Press, New York, pp 65-72, 1999.

[KDB*06] Kim J., Cha D., Chang B., Koo B., Ihm I.: Practical Animation of Turbulent Splashing Water. *In Proceedings of Eurographics '06*, 2006.

[KFC*06] Klingner B., Feldman B., Chentanez N., O'Brien J.: Fluid animation with dynamic meshes. *In Proceedings of ACM SIGGRAPH 2006*, 2006.

[KM90] Kass M., Miller G.: Rapid, Stable Fluid Dynamics for Computer Graphics. *ACM Computer Graphics (SIGGRAPH '90)*, 24(4):49–57, 1990.

[LC87] Lorensen W., Cline H.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics (Proceedings of SIGGRAPH '87)*, Vol. 21, No. 4, pp. 163-169, 1987.

[LGF04] Losasso F., Gibou F., Fedkiw R.: Simulating water and smoke with an octree data structure. *In SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 457–462, New York, NY, USA, ACM Press, 2004.

[LIN07] Lingrand D.: The Marching Cubes Tutorial. Available at http://www.polytech.unice.fr/~lingrand/MarchingCubes/algo.html, 2007.

[LL75] Landau L., Lifschitz E.: Fluid Mechanics. *Pergamon Press*, 1975.

[LL94] Lacroute P., Levoy M.: Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation, *Proc. SIGGRAPH'94*, p.451-458, 1994.

[LLP*00] Lawrence D., Lee C., Pao L., Novoselov R.: Shock and vortex visualization using a combined visual-haptic interface. *IEEE Visualization*, pp.131-137, 2000.

[LSS*06] Losasso F., Shinar T., Selle A., Fedkiw R.: Multiple interacting liquids. *ACM Trans. Graph.*, 25(3):812–819, 2006.

[LTK*07] Losasso F., Talton J., Kwatra N., Fedkiw R.: Two-way Coupled SPH and Particle Level Set Fluid. *In Press: EEE TVCG Submission*, 2007.

[MCG03] Müller M., Charypar D., Gross M.: Particle-based fluid simulation for interactive applications. *In Proc. Eurographics Symp. Comp. Anim.*, pp.54–159, 2003.

[MICROSOFT] Microsoft Corporation. Available at http://www.microsoft.com.

[MIL88] Miller G.: The Motion Dynamics of Snakes and Worms. *ACM Computer Graphics*, vol. 22, no. 4, pp. 169-177, 1988.

[MNS04] Morris D., Neel J., Salisbury K.: Haptic Battle Pong: High-Degree-of-Freedom Haptics in a Multiplayer Gaming Environment. In Experimental Gameplay Workshop, *GDC*, 2004.

[MON92] Monaghan J.: Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 1992.

[MPT99] Mcneely, W. A., Puterbaugh, K. D., Troy, J. J.: Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling, In *Proceedings of ACM SIGGRAPH 99*, Annual Conference Series, 401 - 408, 1999.

[MR81] Myers C.S., Rabiner L.R.: A comparative study of several dynamic time-warping algorithms for connected word recognition. *The Bell System Technical Journal*, 60(7):1389-1409, 1981.

[MRF*96] Mark W., Randolph M., Finch J., Verth R., Taylor II.: Adding Force Feedback to Graphics Systems: Issues and Solutions. *SIGGRAPH'96*, pp.447-452, 1996.

[MS94] Massie T. M., Salisbury J. K.: The phantom haptic interface: A device for probing virtual objects. *Proc. of ASME Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 1:295–301, 1994.

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

[MT91] Murakami K., Taguchi, H.: Gesture Recognition using Recurrent Neural Networks. *In Proceedings: Conference on Human Factors in Computing Systems*, pp.237 - 242, 1991.

[NA03] Nilsson D., Aamisepp H.: Haptic hardware support in a 3D game engine. *Master thesis, Department of Computer Science, Lund University*, 2003.

[NF99] Neff M., Fiume E..: A visual model for blast waves and fracture. *In Proceedings of the 1999 conference on Graphics interface'99*, pages 193–202, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 1999.

[NFJ02] Nguyen D., Fedkiw R., Jensen H.: Physically based modeling and animation of fire. *In SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 721–728, New York, NY, USA, 2002.

[NINTENDO] Nintendo Wii, Available at http://www.wii.com.

[NL02] Nixon D., Lobb R.: A Fluid-Based Soft-Object Model. *In Proceedings of IEEE Computer Graphics and Applications*, Vol. 22, Issue 4, p. 68 – 75, 2002.

[NMK*05] Nealen A., Muller M., Keiser R., Boxerman E., Carlson M.: Physically based deformable models in computer graphics. *Eurographics State of the Art Report*, 2005.

[NNH*98] Nahvi A., Nelson D., Hollerbach J., Johnson D.: Haptic manipulation of virtual mechanisms from mechanical CAD designs. In *Proc. of IEEE Conference on Robotics and Automation*, pages 375–380, 1998.

[NOVINT] Novint Technologies. Available at: http://home.novint.com.

[OE05] Orozco M., El Saddik A.: Haptic: The New Biometrics-embedded Media to Recognizing and Quantifying Human Patterns. *In proceedings of 13th Annual ACM International Conference on Multimedia* (ACMMM 2005), Singapore, 2005.

[OH95] J. O'Brien and J. Hodgins, Dynamic simulation of splashing fluids, *In Computer Animation 95*, pages 198–205, 1995.

[OJS*04] Otaduy Miguel A., Jain Nitin, Sud Avneesh, Lin Ming C.: Haptic rendering of interaction between textured objects. *Technical Report TR-04-07, University of North Carolina at Chapel Hill*, 2004.

[OL03] Otaduy, M. A., Lin, M. C.: Sensation Preserving Simplification for Haptic Rendering. *Proceedings of SIGGRAPH'03, 23*, 3, 2003.

[OPL*01] Osborne R., Pfister H., Lauer H., McKenzie N., Gibson S., Hiatt W., Ohkami T., Mitsubishi Electric Research Lab: EM-Cube: An Architecture for Low-Cost Real-Time Volume Rendering. *IEEE symposium on Volume visualization*, pp.31-38, 2001.

[PER85] Perlin K.: An Image Synthesizer. *Computer Graphics*, 19(3), pp.287-296, 1985.

[PH03] Pasquero J., Hayward V.: Stress: A practical tactile display system with one millimeter spatial resolution and 700 Hz refresh rate. In *Proc. of Eurohaptics*, 2003.

[PS72] Patankar S., Spalding D.: A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *Int. J. Heat Mass Transfer*, 15:1787–1806, 1972.

[PTB*03] Premoze S., Tasdizen T., Bigler J., Lefohn A., Whitaker R.: Particle based simulation of fluids. *In Comp. Graph. Forum (Eurographics Proc.),* volume 22, pages 401–410, New York, NY, USA, ACM Press, 2003.

[REE83] Reeves William T.: Particle Systems - A Technique for Modeling a Class of Fuzzy Objects. *In Computer Graphics (SIGGRAPH 83)* 17:3 pp. 359-376, 1983.

[REN*04] Rasmussen N., Enright D., Nguyen D., Marino S., Sumner N., Geiger W., Hoon S., Fedkiw R..: Directable photorealistic liquids. *In Proceedings of ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp.193–202, 2004.

[RIS00] Ristow Gerald H.: Tumbling motion of elliptical particles in viscous two-dimensional fluids. *International Journal of Modern Physics C*, 11(0), 2000.

[RIS99] Ristow Gerald H.: Particles moving in spatially bounded, viscous fluids. *Computer Physics Communications*, pages 43–52, 1999.

[RKK97] Ruspini D.C., Kolarov K., Khatib O.: The haptic display of complex graphical environments. *Proc. of ACM SIGGRAPH*, pages 345–352, 1997.

[RNG*03] Rasmussen N., Nguyen D., Geiger W., Fedkiw R..: Smoke simulation for large scale phenomena. *In SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 703–707, New York, NY, USA, ACM Press, 2003.

[SARCOS] Sarcos Inc., Available at http://www.sarcos.com.

[SB97] Srinivasan, M., Basdogan, C.: Haptics in Virtual Environments: Taxonomy, Research Status, and Challenges. *Computer and Graphics*, 21(4), pp.393-404, 1997.

[SBM*95] Salisbury K., Brock D., Massie T., Swarup N, Zilles C.: Haptic rendering: Programming touch interaction with virtual objects. *Proc. of 1995 ACM Symposium on Interactive 3D Graphics*, pages 123–130, 1995.

[SENSABLE] SensAble Technologies Inc. Available at http://www.sensable.com.

[SP96] Siira J., Pai Dinesh K.: Haptic texturing – a stochastic approach. *Proceedings of the Int'l Conf. on Robotics and Automation*, 1996.

[SRF05] Selle A., Rasmussen N., Fedkiw R.: A vortex particle method for smoke, water and explosions. ACM Trans. Graph., 24(3):910–914, 2005.

[SRI00] Srikanth R.K. Mishra S.: GENIE - an haptic interface for simulation of laparoscopic surgery. *Proc. of International Conference on Intelligent Robots and Systems*, pp. 714-719, 2000.

[STA00] Stam J.: Interacting with smoke and fire in real time. In: *Communications of the ACM*, Volume 43, Issue 7, pp.76-83, 2000.

[STA01] Stam J.: A Simple Fluid Solver based on the FFT, Journal of Graphics Tools, Volume 6, Number 2, pp. 43-52, 2001.

[STA03*] Stam J.: Flows on surfaces of arbitrary topology. *In SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 724–731, New York, NY, USA, ACM Press, 2003.

[STA03] Stam, J.: Real-Time Fluid Dynamics for Games. In *Proceedings of the Game Developer Conference*, 2003.

[STA99] Stam J.: Stable fluids. *In SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co, 1999.

[TF88] Terzopoulos D., Fleischer K.: Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture. *ACM Computer Graphics*, vol. 22, no. 4, pp. 269-278, 1988.

[TFC07] Taylor M. A., Ferber A. R., Colgate J. E.: Assessing the Efficacy of Variable Compliance Tactile Displays. *Second Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems* (WHC'07). pp. 427-432, 2007.

[TJC97] Thompson T.V., Johnson D., Cohen E.: Direct haptic rendering of sculptured models. *Proc.of ACM Interactive 3D Graphics*, pages 167–176,1997.

[TOT*03] Takeshita D., Ota S., Tamura M., Fujimoto T., Muraoka K., Chiba N.: Particle-based visual simulation of explosive flames. *In PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, page 482, Washington, DC, USA, IEEE Computer Society, 2003.

[TU96] Tu X.: Artificial Animals for Computer Animation: Biomechanics, Locomotion, Perception, and Behavior. *PhD thesis, University of Toronto*, 1996.

[VRAC] Virtual Reality Applications Center (VRAC), Iowa State University, Available at http://www.vrac.iastate.edu/research/robotics/magnetic/index.html.

[WES91] Westover, L.: Splatting: A Parallel, Feed-Forward Volume Rendering Algorithm. *PhD Dissertation*, 1991.

[WG91] Wilhelms J., Gelder A.: A Coherent Projection Approach for Direct Volume Rendering, *Computer Graphics*, Vol. 25(4), 1991.

[WMW86] Wyvill G., McPhetters C., Wyvill B.: Data Structure for Soft Objects. *In proceedings of the Visual Computer*, Vol. 2, pp. 227-234, 1986.

[WV91] Wilhelms J., Van Gelder A.: A Coherent Projection Approach for Direct Volume Rendering. *In Computer Graphics*, 35(4):275-284, 1991.

[WYL*05] Wu J., Yang T., Luo B., Pozrikidis C.: Fluid kinematics on a deformable surface. *In J. Fluid Mech.*, vol. 541, pp. 371–381, 2005.

[YHD07] Yuksel C., House D., Keyser J.: Wave Particles. *In Proceedings of ACM SIGGRAPH'07*, 2007.

[ZB05] Zhu Y., Bridson R.: Animating sand as a fluid. *ACM Trans. Graph.*, 24(3):965–972, 2005.

Appendix A: Solver Code

This section is devoted to provide programming fragments, which were used to solve various problems mentioned in the main chapters. We show snippets of code for different routines such as indexing macros, adding a substance to the density, diffusing, advecting, projecting, and advancing the density and the velocity for a time-step.

- Indexing macro: the array elements are referenced using the following macro:

```
#define IX(i,j) ((i)+(N+2)*(j)) in 2D, or
#define IX(i,j,k) ((i) + ((N)+2)*(j) + ((N)+2)*((N)+2)*(k)) in 3D.
```

For example cell (i,j,k) of the horizontal component of the velocity is given by the entry u[IX(i,j,k)]. We also assume that the physical length of each side of the grid is one so that the grid spacing is given by h=1/N.

- Addition of the substance to the density.

```
void add_source ( int N, float * x, float * s, float dt )
{
      int i, size=(N+2)*(N+2)*(N+2);
      for ( i=0 ; i<size ; i++ ) x[i] += dt*s[i];
}
```

- Diffusion: It uses a simple iterative solver which works well in practice.

```
void diffuse (int N,int b, float * x, float * x0, float diff, float dt)
{
      int i, j, k, l;
      float a=dt*diff*N*N*N;
      for (l=0; l<20; l++)
      {
            for (k=1; k<=N; k++)
            {
                  for (j=1; j<=N; j++)
                  {
                        for (i=1; i<=N; i++)
                        {
                        x[IX(i,j,k)] = (x0[IX(i,j,k)] + a*(
                              x[IX(i-1,j,k)]+x[IX(i+1,j,k)]+
                              x[IX(i,j-1,k)]+x[IX(i,j+1,k)]+
```

```
                                    x[IX(i,j,k-1)]+x[IX(i,j,k+1)]))/(1+6*a);
                        }
                    }
                }
                set_bnd(N,b,x);
        }

}
```

- **Advection. We use a simple linear back-trace.**

```
void advect (int N, int b, float * a0, float * a1, float * u,float * v,
            float * w, float dt )
{

        int i, j, k, i0, j0, k0, i1, j1, k1;
        float sx0, sx1, sy0, sy1, sz0, sz1, v0, v1;
        float xx, yy, zz, dt0;
        dt0 = dt*N;
        for (k=1; k<=N; k++)
        {
                for (j=1; j<=N; j++)
                {
                        for (i=1; i<=N; i++)
                        {
                        xx = i-dt0*u[IX(i,j,k)];
                        yy = j-dt0*v[IX(i,j,k)];
                        zz = k-dt0*w[IX(i,j,k)];
                        if (xx<0.5) xx=0.5f;
                        if (xx>N+0.5) xx=N+0.5f; i0=(int)xx; i1=i0+1;
                        if (yy<0.5) yy=0.5f;
                        if (yy>N+0.5) yy=N+0.5f; j0=(int)yy; j1=j0+1;
                        if (zz<0.5) zz=0.5f;
                        if (zz>N+0.5) zz=N+0.5f; k0=(int)zz; k1=k0+1;
                        sx1 = xx-i0; sx0 = 1-sx1;
                        sy1 = yy-j0; sy0 = 1-sy1;
                        sz1 = zz-k0; sz0 = 1-sz1;
                        v0 = sx0*(sy0*a1[IX(i0,j0,k0)]+sy1*a1[IX(i0,j1,k0)])+
                        sx1*(sy0*a1[IX(i1,j0,k0)]+ sy1*a1[IX(i1,j1,k0)]);
                        v1  =  sx0*(sy0*a1[IX(i0,j0,k1)]+sy1*a1[IX(i0,j1,k1)])+
                        sx1*(sy0*a1[IX(i1,j0,k1)]+sy1*a1[IX(i1,j1,k1)]);
                        a0[IX(i,j,k)] = sz0*v0 + sz1*v1;
                        }
                }
        }
        set_bnd(N,b,a0);
}
```

- **The projection step:**

```
void project ( int N, float * u, float * v, float * w, float * p, float *
div )
{
        int i, j, k, l;
        float h;
        h = 1.0f/N;
```

```
        for (k=1; k<=N; k++) {
                for (j=1; j<=N; j++) {
                        for (i=1; i<=N; i++) {
                                div[IX(i,j,k)] = -h*(
                                        u[IX(i+1,j,k)]-u[IX(i-1,j,k)]+
                                        v[IX(i,j+1,k)]-v[IX(i,j-1,k)]+
                                        w[IX(i,j,k+1)]-w[IX(i,j,k-1)])/3;
                                p[IX(i,j,k)] = 0;
                        }
                }
        }
        set_bnd(N,0,div); set_bnd(N,0,p);
        for (l=0; l<20; l++)
        {
                for (k=1; k<=N; k++) {
                        for (j=1; j<=N; j++) {
                                for (i=1; i<=N; i++) {
                                        p[IX(i,j,k)] = (div[IX(i,j,k)]+
                                                p[IX(i-1,j,k)]+p[IX(i+1,j,k)]+
                                                p[IX(i,j-1,k)]+p[IX(i,j+1,k)]+
                                                p[IX(i,j,k-1)]+p[IX(i,j,k+1)])/6;
                                }
                        }
                }
                set_bnd(N,0,p);
        }
        for (k=1; k<=N; k++) {
                for (j=1; j<=N; j++) {
                        for (i=1; i<=N; i++) {
                        u[IX(i,j,k)] -= (p[IX(i+1,j,k)]-p[IX(i-1,j,k)])/3/h;
                        v[IX(i,j,k)] -= (p[IX(i,j+1,k)]-p[IX(i,j-1,k)])/3/h;
                        w[IX(i,j,k)] -= (p[IX(i,j,k+1)]-p[IX(i,j,k-1)])/3/h;
                        }
                }
        }
        set_bnd(N,1,u); set_bnd(N,2,v);

}
```

- **Advance density in a time-step: We assume here that the substance densities are initially contained in the x0 array.**

```
void dens_step ( int N, float * x, float * x0, float * u, float * v, float
* w, float diff, float dt )
{
        add_source ( N, x, x0, dt );
        SWAP ( x0, x ); diffuse ( N, 0, x, x0, diff, dt );
        SWAP ( x0, x ); advect ( N, 0, x, x0, u, v, w, dt );

}
```

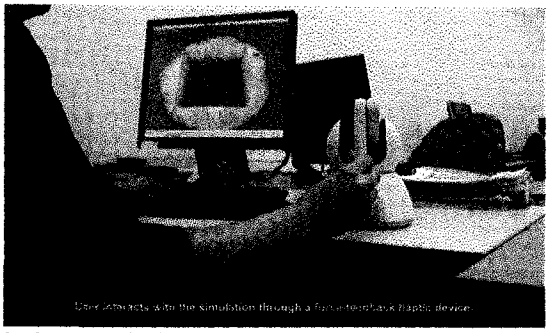
where SWAP is a macro that swaps the two array pointers:

```
#define SWAP(x0,x) {float * tmp=x0;x0=x;x=tmp;}
```

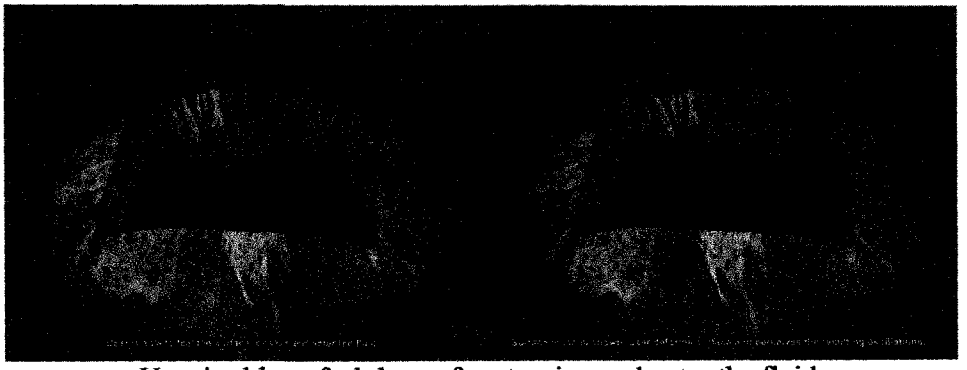
- Advance velocity for a time-step: Assuming that the force field is stored in arrays *u0, v0 and w0*, we have the following:

```
void vel_step ( int N, float * u, float * v, float * w, float * u0, float
* v0, float * w0, float * x, float * x0, float visc, float dt, bool air )
{
      __isAir = air;

      add_source ( N, u, u0, dt );
      add_source ( N, v, v0, dt );
      add_source ( N, w, w0, dt );

      if(air==true) add_buoyancy(dt,v,x);

      SWAP ( u0, u );
      SWAP ( v0, v );
      SWAP ( w0, w );
      diffuse ( N, 1, u, u0, visc, dt );
      diffuse ( N, 2, v, v0, visc, dt );
      diffuse ( N, 3, w, w0, visc, dt );
      project ( N, u, v, w, u0, v0 );
      SWAP ( u0, u );
      SWAP ( v0, v );
      SWAP ( w0, w );
      advect ( N, 1, u, u0, u0, v0, w0, dt );
      advect ( N, 2, v, v0, u0, v0, w0, dt );
      advect ( N, 3, w, w0, u0, v0, w0, dt );
      project ( N, u, v, w, u0, v0 );

}
```

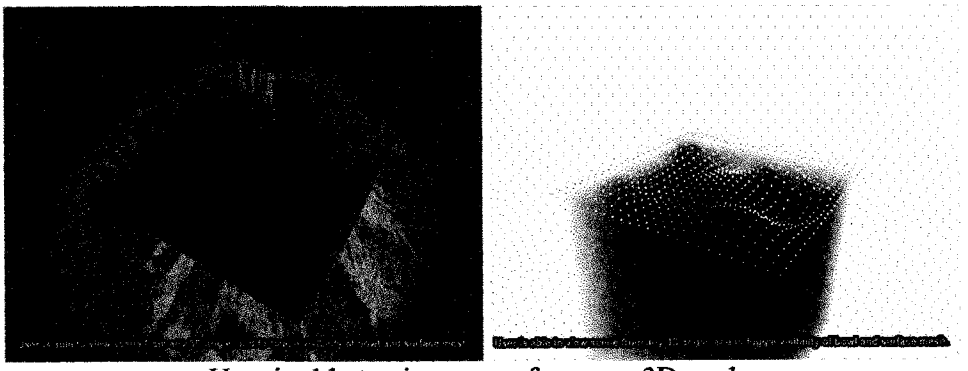
Appendix B: System Video Storyboard



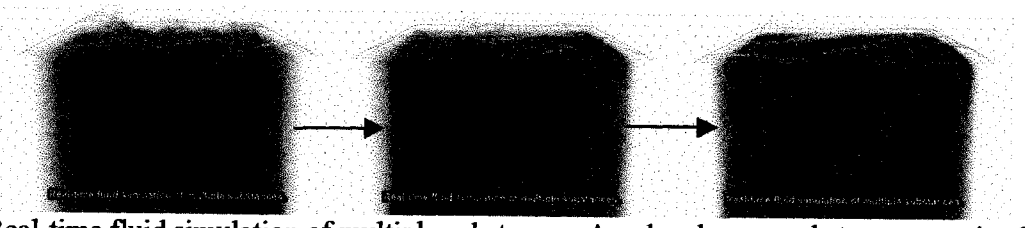
User interacts with the simulation through an Omni [SENSABLE] haptic device.



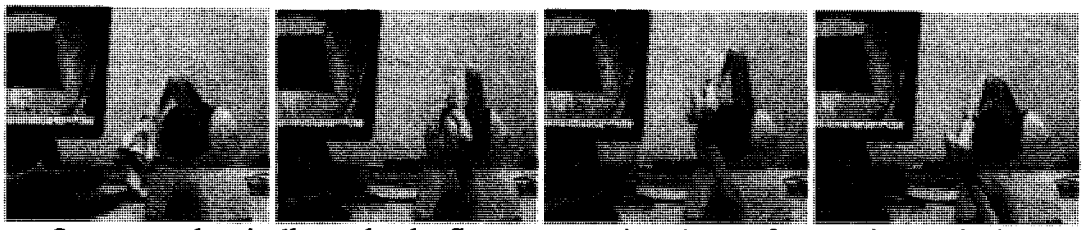
User is able to feel the surface tension and enter the fluid.
Screenshot shows surface deformations and mass-spring mesh visibility.



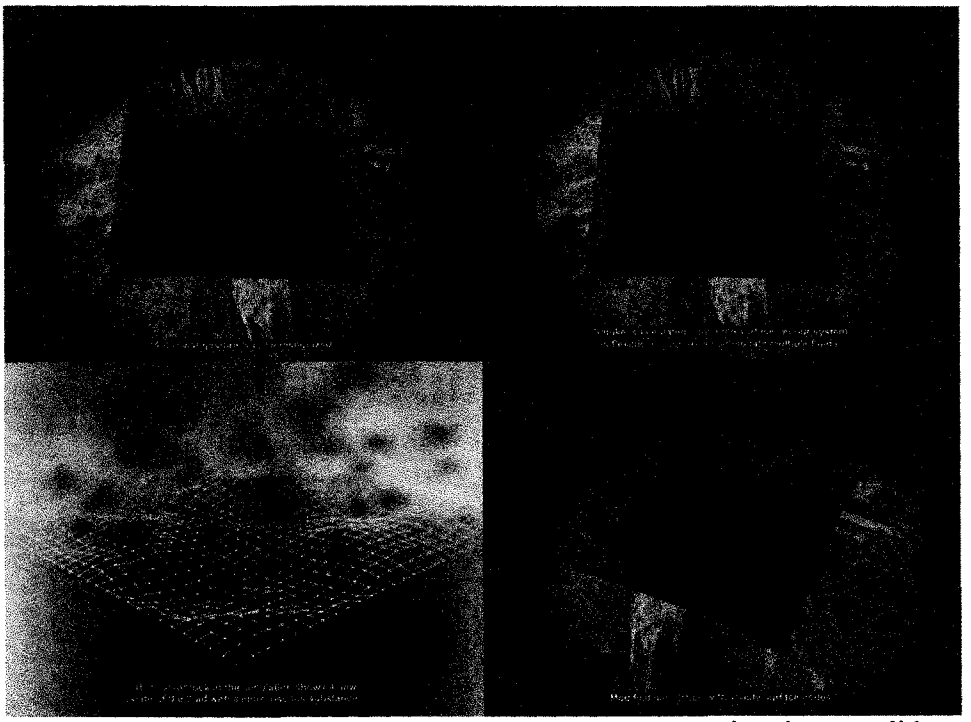
User is able to view scene from any 3D angle,
and to toggle visibility of bowl and surface mesh.



Real-time fluid simulation of multiple substances. A red and green substance are mixed together generating orange blends.



System can haptically render the flow current, viscosity, surface tension, and other properties at the surface and at any cell inside the fluid. Screenshots show the haptic device following the circular motion of the current flow without applying human force.



A circular gesture is performed by the user and the system recognizes it as a valid motion. Smoke is simulated above the surface.

Related publications by the author

1. Javier Mora and Won-Sook Lee, "Real-Time 3D Fluid Interaction with a Haptic User Interface", in the *Proceedings of the IEEE Symposium on 3D User Interfaces (3DUI 2008)*, Reno – Nevada, March 2008.
2. Javier Mora and Won-Sook Lee, "Real-Time Fluid Interaction with a Haptic Device", in the *Proceedings of the IEEE International Workshop on Haptic Audio Visual Environments and their Applications (HAVE 2007)*, October 2007.
3. Sheldon Andrews, Javier Mora, Jochen Lang and Won-Sook Lee, "HaptiCast: A Physically-Based 3D Game with Haptic Feedback", *in the Proceedings of FuturePlay 2006 (to be in ACM library), London -ON.* [Runner-up prize for Future Game Technology and Design].

Other publications by the author

1. Javier Mora, Won-Sook Lee, Gilles Comeau, Shervin Shirmohammadi and Abdulmotaleb El Saddik, "Assisted Piano Pedagogy through 3D Visualization of Piano Playing", in the *Proceedings of the IEEE International Workshop on Haptic Audio Visual Environments and their Applications (HAVE 2006)*, 2006.
2. Javier Mora, Won-Sook Lee and Gilles Comeau, "3D Visual Feedback in Learning of Piano Posture", in the *Proceedings of Edutainment 2007, in Lectures Notes in Computer Science, CUHK*, Hong Kong, 11-13 June 2007.