REAL-TIME SMOKE SIMULATION

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Eren Algan
December, 2010

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Bülent Özgüç (Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Tolga Çapın

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Tolga Can

Approved for the Institute of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Institute

ABSTRACT

REAL-TIME SMOKE SIMULATION

Eren Algan
M.S. in Computer Engineering
Supervisor: Prof. Dr. Bülent Özgüç
December, 2010

Realistic simulation of fluid-like behaviour is an important and challenging problem in computer graphics. Huge and increasing amount of animations has made this phenomena even more important. Although many scientists provided solutions regarding this issue, recently, the need of fast and easy implemented fluid simulations has directed researches to focus on quick and stable solutions.

This thesis presents an unconditionally stable, easy implemented real-time smoke simulation, solving Navier-Stokes equations with Lagrangian and implicit methods. The study focuses on the comprehension of fluid dynamics as much as the solution, by providing background information about Navier-Stokes equations, how they are derived and used. While the proposed solution is applied only to create a simulation for smoke like behaviour, it is highly adaptive for other fluids as well. One important aspect of the simulation is being suitable for 2 and 3 dimensions, giving the flexibility to the animator to choose in between.

Keywords: Navier-Stokes equations, simulation, physics-based modeling, animation of fluids, stable solvers, gaseous phenomena.

ÖZET

GERÇEK ZAMANLI DUMAN SİMÜLASYONU

Eren Algan
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Bülent Özgüç
Aralık, 2010

Bilgisayar grafiği alanında, akışkanların davranışlarının simülasyonu önemli bir problemdir. Çok miktarda bulunan ve artmaya devam eden animasyonlar, bu fenomeni daha önemli bir problem haline getirmiştir. Bir çok araştırmacı, bu konuyla ilgili çözümler sunmuşlardır. Fakat, son zamanlarda, hızlı ve kolay programabilir akışkan simülasyonlarına olan ihtiyacın artması, araştırmacıları daha çabuk ve istikrarlı çözümler bulmaya itmiştir.

Bu tez araştırmasında, Navier-Stokes denklemlerini Langrange ve örtük metodlar kullanarak çözen, kolay programlanabilir, gerçek zamanlı bir duman simülasyonu sunulmaktadır. Araştırma, çözümün kendisine odaklandığı gibi, Navier-Stokes denklemlerini ve bu denklemlerin nasıl derive edildiklerini anlatarak akışkan mekaniği ile ilgili de kapsamlı bir bilgi sunmaktadır. Çözüm duman simülasyonu için verilmesine rağmen, diğer akışkanlar için de uygulanabilir. Simülasyonun diğer bir özelliği ise kullanıcının isteğine göre 2 ve 3 boyutlu uzaya uyumlu olabilmesidir.

Anahtar sözcükler: Navier-Stokes denklemleri, simülasyon, fiziksel modelleme, akışkan animasyonu, istikrarlı çözücüler, gaz fenomenler.

Acknowledgement

I would like to express my sincere gratitude to my supervisor Prof. Dr. Bülent Özgüç for his instructive comments, suggestions, support and encouragement during this thesis work.

I am grateful to my jury members, and Asst. Prof. Dr. Tolga Çapın, and Asst. Prof. Dr. Tolga Can for reading and reviewing this thesis.

Special thanks to my family for their support during my education and their never lasting efforts on me.

The very intuition of this work can be summarized by the famous quote of Sir Isaac Newton: "If I have seen further, it is by standing on the shoulders of giants." Everyone would agree that we, as researchers, are never creating from scratch. Each one of us develops future intellectual pursuits by understanding the research and works created by notable thinkers of the past. Every single one of the researchers that I quoted and got helped throughout the thesis can be considered as my giants. While I am quite happy to work on such a topic, develop and finish a quite successful system, I am deeply thankful for such an extravagant help by those giants.

If I have seen further, it is by standing on the shoulders of giants.

Sir Isaac Newton

Everything should be made as simple as possible, but not simpler.

Albert Einstein

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Modeling of natural phenomena such as smoke is a challenging problem in computer graphics. This is not surprising since the motion of gases is highly complex. Furthermore, modeling fluid behavior is of great importance for a broad range of areas from animation to engineering. On one side, due to the increase in the amount of animations in the film industry, there is a high demand to convincingly mimic the appearance of fluids. On the other side, it is of primary concern to simulate fluid-like behavior physically correct in engineering applications. Nevertheless, both majors require this phenomenon to be modeled effectively, as well as fast and easy.

Many scientists have been working on this problem and there is a consensus among scientists that *Navier-Stokes* [5] equations are a very good model for fluid flow [20]. Although the existence and uniqueness of classical solutions of the 3-D Navier-Stokes equations is still an open mathematical problem and is one of the Clay Institute's Millenium Problems [3], in 2-D, existence and uniqueness of regular solutions for all time have been shown by Jean Leray in 1933 [6]. Anyhow, these equations are used in 3-D in fluid flow as well as other fields.

"In the early 1960s, Fromm, Harlow and Welch at the Los Alamos Scientific Laboratory developed several algorithms based on the Navier-Stokes equations for simulating fluid flow on electronic computers and the field of Computational Fluid Dynamics (CFD) was born [13, 14, 15]. The field has advanced rapidly in the intervening period as computer power has grown exponentially, and while many problems remain unsolved, computational fluid dynamics now encompasses a vast array of different techniques.

The first ones to model the motion of smoke by simulating the equations of fluid dynamics directly were Kajiya and Von Herzen [16] in 1984. Unfortunately computer power at the time limited them to working at a low level of detail and the simulations were extremely slow.

Little progress was made until Foster and Metaxas [12] created an animation tool for gaseous volumes which solved a simplified version of the Navier-Stokes equations in three dimensions over a regular voxel grid. They used customized equations, which modeled only the processes responsible for creating visually interesting elements of the flow, ignoring other elements necessary for a strict scientific simulation. This typifies the approach used in computer graphics where a gaseous volume can normally be assumed to be incompressible and homogenous without detriment to the visual effect.

There were however limitations, most notably in the methods for solving the fluid equations where an explicit integration scheme based on that developed by Harlow and Welch [15] was used. In the explicit (forward Euler) scheme, values at the new time are directly calculated from previous values. While this method is straightforward to code, it introduces the constraint that the time step must be kept small to ensure stability, and hence severely limits the speed of simulation, which can be achieved. If the time step is too large, instability occurs when small-scale local oscillations in the variables resonate and dominate the solution, causing the volume to blow up.

Jos Stam overcame the problems associated with instability by using a semi-Lagrangian and implicit integration scheme which is unconditionally stable for any time step, hence allowing a much quicker simulation. The implicit (backward

Euler) scheme solves the equations by iteratively modifying an initial trial solution until it converges to within a specified range. The simulations are further improved by advecting texture coordinates through the volume along with the density, thus creating the effect of a complex, detailed flow even on low-resolution grids. By using a simple hardware renderer alongside his fluid solver, Stam was able to produce real time interactive simulations on low-resolution grids [20].

Despite being fast however, Stam's simulations lacked many of the small-scale vortices and swirling motions that make fluid flow so interesting. The semi-Lagrangian method used to approximate the fluid flow suffers from excessive numerical dissipation causing small-scale vortices to dampen and vanish too quickly.

This problem was partially addressed by Stam, Fedkiw and Jensen [10] who borrowed another technique from the CFD literature, called vorticity confinement. Energy lost from the system due to numerical dissipation is reinjected using a forcing term which increases the vorticity of the flow and keeps small-scale eddies alive. The addition of vorticity confinement incurs only a small computational cost and the simulation remains stable as long as the magnitude of the forcing term remains below a certain threshold. They combined the solver with a photon map renderer to produce stunning animations of smoke, although none of the simulations were real time and some of the more complex effects took over one minute to render each frame.

Despite the success of Eulerian grid based methods, they are still unsuitable for creating many types of effects. The introduction of vorticity confinement by Fedkiw et al [10] went some way to addressing the problems of numerical dissipation inherent in the semi-Lagrangian integration scheme, but was limited to only increasing the magnitude of pre-existing grid vorticity. For highly turbulent effects the grid resolutions used are typically too coarse to adequately capture the necessary level of detail in the flow, and vorticity confinement is not as effective. For this reason, while grid based methods can be very effective for small scale and wispy smoke effects, they are generally not capable of capturing large scale or violent phenomena such as explosions.

Another area of CFD literature, which has recently begun attracting attention is the Lattice Boltzmann Method (LBM) which uses cellular automata to describe the flow. The LBM dispenses with the difficult Navier-Stokes Equations and instead uses a lattice of cells, which update their attributes at each time step by relatively straightforward linear and local rules. The LBM effectively provides a simple model of microscopic flow in the volume, which is found to accurately simulate the Navier-Stokes equations at the macroscopic level.

A major advantage of the LBM is its ability to handle complex moving boundaries. Li, Zan and Wei [25] produce 2D GPU simulations of flow fields around various complex boundaries, including a jellyfish whose body deforms in a swimming motion as it moves through the field. Li et al [18] again use the LBM to produce 3D animations of smoke and steam which interact with stationary and slow moving solid object boundaries. However, in order to achieve real time simulations, they were forced into using very low-resolution lattices and rendering the volume using textured splats. As a result, much of the intricate physics-based motion in the volumes is lost, and the resulting simulations do not offer significant visual improvement over similar effects animated using faster non-physics based techniques.

The LBM is a relatively new area of interest in computer graphics, and it remains to be seen whether it is a viable solution for real time fluid simulation in computer games. Although the linear rules which model the flow at the macroscopic level are straightforward, this advantage is offset somewhat by the need to represent detailed flows with many thousands or even millions of cells.

It is important to emphasize that there are also non-physics techniques such as particle systems and solid spaces.

Particle systems are an extremely versatile technique that can be applied to the simulation of smoke and other gaseous phenomena. A particle system models an amorphous volume as a collection of primitive particles, which are born into the system and over time can move and change form before eventually expiring from the system. Animation of the overall volume is achieved using functions to control each particle individually.

An alternative to particle systems which has achieved notable results is the idea of solid spaces developed by Ebert [9, 19], and similar to Perlins work on hypertextures [19]. The idea is similar to solid texturing, where an object such as a stone or marble figure is carved out of a procedurally produced three-dimensional colour space. In his work on solid spaces, Ebert uses stochastic noise and turbulence functions to define a three-dimensional turbulence space. The geometry of the gas is then modeled using volume density functions, which map each point in the volume in world space to a corresponding point in turbulence space, apply a turbulence function, and return a value which corresponds to the density of the volume at that point. The value returned by the turbulence function is then shaped using basic mathematical functions to create different effects depending on the type of gas required. For example, applying a power function to the value will produce greater contrast and definition in the density of the volume. Shaping the gas using sine waves, which are used in solid texturing to produce marble effects, creates veins in the volume.“

Although there are some successful results of both techniques, since they are non-physics approaches, we do not consider them as suitable models for smoke simulation.

In this study, we are proposing an easy implemented physics-based, real-time Jos Stam like smoke simulation, solving Navier-Stokes equations with Lagrangian and implicit methods. The main contribution of this study is implementing a real-time smoke simulation with vorticity confinement on Jos Stam's stable fluids. Unlike the solver presented in [21] which computes results based on the Fast Fourier Transform, this implementation uses a sparse linear solver that can function under arbitrary boundary conditions. For completeness, we implemented a simple iterative solver directly, just as in [22]. Furthermore, this study is a explanatory source for the basics of computational fluid dynamics and focusing on smoke simulation.

1.2 Outline of the Thesis

The thesis is organized as follows:

- **Chapter 2** describes the basics of fluid dynamics giving an intuition for those who are unfamiliar with the topic.
- **Chapter 3** presents our smoke simulation along with how these basics were used. It describes the system in detail, including the main loop of the simulation.
- **Chapter 4** gives the details of the implementation and shows a few screen shots along with some descriptive tables.
- **Chapter 5** concludes the study including special thanks to important scholars of history.
- **Appendix A** is an appendix giving fundamental information about vector calculus, numerical methods and derivations. There is also a small section, providing a brief history of Navier-Stokes equations.

Chapter 2

Fluid Dynamics: The Basics

From the air we breath, and water we shower, to the fire we see on a match, fluids are everywhere in our lives. It is a beautiful phenomena, which attracts the attention of the researchers for years. There is a general agreement among scientists that Navier-Stokes equations, the very fundamental equations governing this phenomena, are a very good model for fluid flow [20].

In this section, rudimentary information about Navier-Stokes equations will be provided.

2.1 Navier-Stokes Equations

Navier-Stokes equations, named after Cladue-Louis Navier and George Gabriel Stokes, are used to describe the fluid like behavior. The equations are:

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

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

where

The letter $\vec{u}$ is the *velocity* of the fluid. It represents the components of the velocity field, (u, v, w), in three-dimensional space.

The letter ρ is the *density* of the fluid. Density can be described as the degree of compactness of the fluid. While, it is 1000 kg/m^3 for water, for air it is 1.3 kg/m^3 [4]. It can be formulated as the mass of fluid, m, divided by the volume of the fluid, V, e.g. m/V.

p stands for *pressure*. Pressure is the force per unit area applied perpendicular to any object. Mathematically representing:

$$p = \frac{F}{A}$$

where p is the pressure, F is the force, and A is the area.

$\vec{g}$ is *gravity*. It is (0, -9.81, 0) m/s^2 in three dimensions. However in this study, it is used as a general term representing *body forces*.

The letter ν is *kinematic viscosity*, measuring how much the fluid resists deforming while it flows. While fluids with low viscosity, such as water, are less resistant to flow, fluids with high viscosity, such as honey, are very resistant. All real fluids have some resistance to stress, but a fluid which has no resistance is known as an ideal or inviscid fluid [7].

Although equations 2.1 and 2.2 appear very complicated at first, they become pretty easy after breaking them into parts.

2.1.1 Conservation of Momentum

The first equation 2.1 is called the momentum equation. The equation, per se, is nothing but the Newton's second law, conservation of momentum, $\vec{F} = m\vec{a}$. It basically expresses how the fluid accelerates when a force acts on it.

Let's assume that each particle in a fluid would have a mass m, a volume V, and a velocity $\vec{u}$. Acceleration can be denoted as $\vec{a}$, and is the material derivative of the velocity over time.

$$\vec{a} \equiv \frac{D\vec{u}}{Dt}$$

which yields Newton's equation to become

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

Now is a good time to investigate what forces are acting on the fluid. Let's start with dividing them into two major parts. The first one is the outer force, the forces acting on a fluid, and the other one is the fluid forces, force existing with the interaction of particles with each other.

The outer force is simply gravity: m$\vec{g}$

One of the fluid forces is pressure. Pressure, as defined earlier, is the force per unit area and is expressed as the letter p. In this context, we can say that pressure is the force that keeps the fluid at constant volume.

It is a fact that "high pressure regions push on lower pressure regions" [8] and it is significant to state that what really important is the net force on a particle. Hence, "if the pressure is equal in every direction, then there is going to be a net force of zero" [8] yielding to no acceleration due to pressure.

The derivation of pressure is given in [8]. We can just skip to the conclusion and say that "the way to measure the imbalance in pressure at a position of the particle is to take the negative gradient of pressure" [8], e.g. -∇p. As an approximation, we need to multiply this by the volume V, yielding to $-V\nabla p$.

The other fluid force is viscosity. Viscosity is the resistant force of fluid trying to minimize the velocity differences between fluid particles. It can also be defined as the thickness of fluid. While fluids like honey are thick, high viscosious fluids, fluids like water are thin, low viscosious fluids. As defined in Chapter 5, the "differential operator that measures how far a quantity is from the average around" [8] is the Laplacian. Hence, taking the Laplacian of the velocity field and multiplying it with the volume (approximation of integrating over volume) and dynamic viscosity coefficient (η), we have $V\eta\nabla.\nabla\vec{u}$.

Putting them altogether, we have:

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

In order to get rid of the approximation errors, we need to take the limit of this equation with particle number and blob size constraints. The number of particles shall go to infinity while the size of each blob goes to zero. To do so, we need to divide the equation by the volume and then take the limit, because the mass m and the volume V go to zero. We know that m/V is the density ρ, and we have

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

After dividing the equation with density and arranging items, we get

$$\frac{D\vec{u}}{Dt} + \frac{1}{\rho}\nabla p = \vec{g} + \frac{\eta}{\rho}\nabla\cdot\nabla\vec{u}$$

Furthermore, we define the kinematic viscosity as $\nu = \eta/\rho$, and we have

$$\frac{D\vec{u}}{Dt} + \frac{1}{\rho}\nabla p = \vec{g} + \nu\nabla\cdot\nabla\vec{u} \tag{2.3}$$

This equation is really similar to 2.1. In order to turn this equation to the original equation, we need to understand the material derivative and the difference between Eulerian and Lagrangian viewpoints.

2.1.2 Material Derivative, Eulerian and Lagrangian Viewpoints

Basically, there are two ways of tracking a continuous move (like fluid move), Langrangian and Eulerian. While the former one is named after the French mathematician Lagrange, the latter one is named after the Swiss mathematician Euler.

In the Lagrangian viewpoint, the system is composed of particles and the continuum is observed through a particle. Each particle in the fluid has a position field and a velocity field. "Most of the solids are simulated in a Lagrangian way, with a discrete set of particles usually connected up in a mesh" [8].

"The Eulerian approach follows another strategy, which is more usable for fluids" [8]. In this approach there are no particles but constant points, which we

might call grids. Continuum is observed through these constant points, calculating the change in density, velocity, temperature, etc..

"A very good example to understand the difference between these two viewpoints is to consider a weather report. In the Lagrangian approach, an observer stands in a balloon floating around and calculating the temperature of each point he passes through. On the other hand, in the Eulerian method, the observer is on the ground measuring the temperature of the air flowing past at that constant point" [8].

It can be said that numerically, the Lagrangian approach corresponds to a particle system, while the Eulerian approach to fixed grid system. Although, the Eulerian approach seems complicated, there are few reasons that make it more suitable for fluids [8]:

- "It is easier to analytically work with the spatial derivatives like the pressure gradient and viscosity term in the Eulerian approach" [8].
- "It is easier to numerically approximate those spatial derivatives on a fixed Eulerian mesh than on moving particles" [8].

Material derivative is a link between Eulerian and Lagrangian viewpoints. Let's consider a particle (Lagrangian) having a position $\vec{x}$ and velocity $\vec{u}$. Let's say that each particle has a scalar quantity (density, velocity, temperature, etc.), q, and define a function $q(t, \vec{x})$ telling us the value of q at time t for the particle at position $\vec{x}$. This is obviously an Eulerian variable because it is a function of space, not of particles. "How fast is q changing for that particle at position $\vec{x}$ is an Lagrangian question and can be found by the Chain Rule:" [8]

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}q(t,\vec{x}) &= \frac{\partial q}{\partial t} + \nabla q \cdot \frac{\mathrm{d}\vec{x}}{\mathrm{d}t} \\ &= \frac{\partial q}{\partial t} + \nabla q \cdot \vec{u} \\ &\equiv \frac{Dq}{Dt} \\ \frac{Dq}{Dt} &= \frac{\partial q}{\partial t} + u\frac{\partial q}{\partial x} + v\frac{\partial q}{\partial y} + w\frac{\partial q}{\partial z}\end{aligned} \tag{2.4}$$

Equation 2.4 is the mathematical definition of material derivative. "The first term, $\partial q/\partial t$, is an Eulerian measurement, and it basically states that how fast q is changing at that fixed point in space. The second one, $\nabla q.\vec{u}$, corrects how much of that change is due to differences in the fluid flowing past" [8].

Returning back to the equation:

$$\frac{D\vec{u}}{Dt} + \frac{1}{\rho}\nabla p = \vec{g} + \nu\nabla \cdot \nabla\vec{u}$$

and applying material derivative of $\vec{u}$ as follows:

$$\frac{D\vec{u}}{Dt} = \begin{bmatrix} Du/Dt \\ Dv/Dt \\ Dw/Dt \end{bmatrix} = \begin{bmatrix} \partial u/\partial t + \vec{u}.\nabla u \\ \partial v/\partial t + \vec{u}.\nabla v \\ \partial w/\partial t + \vec{u}.\nabla w \end{bmatrix} = \frac{\partial\vec{u}}{\partial t} + \vec{u}\cdot\nabla\vec{u} \tag{2.5}$$

changing $D\vec{u}/Dt$ in equation 2.3 with 2.5, we get:

$$\frac{\partial\vec{u}}{\partial t} + \vec{u}\cdot\nabla\vec{u} + \frac{1}{\rho}\nabla p = \vec{g} + \nu\nabla\cdot\nabla\vec{u}$$

which is exactly same as 2.1.

2.1.3 Incompresibility Condition

Although most people think that liquids do not change their volumes, in fact, they do. Otherwise, we would not be able to hear under water. Nevertheless, the important thing is that fluids do not change their volume very much and this is the phenomena that equation 2.2 bases on. Scientists treat both liquids and gases as *imcompressible*, which basically means that fluids' volume does not change. Let's investigate this phenomena in a mathematical point of view.

Assuming we have a chunk of fluid, "the volume of the fluid is Ω and its boundary surface is $\partial\Omega$. In order to measure how fast the volume of this chunk of fluid is changing, we need to integrate the normal component of its velocity around the boundary:" [8]

$$\frac{\mathrm{d}}{\mathrm{d}t}volume(\Omega) = \iint_{\partial\Omega}\vec{u}\cdot\hat{n}$$

Hence, we assume that the volume is constant for an incompressible fluid; the rate of change should be zero.

$$\iint_{\partial\Omega} \vec{u} \cdot \hat{n} = 0$$

By using the divergence theorem, this equation can be changed to a volume integral.

$$\iiint_{\Omega} \nabla \cdot \vec{u} = 0$$

"The only function that integrates to zero independent of the volume of integration is zero itself" [8]. So, the integrand should be zero:

$$\nabla \cdot \vec{u} = 0$$

This equation is the second part of Navier-Stokes equations (2.2) and called *imcompresibility condition.*

Chapter 3

Smoke

This study focuses on smoke as a visual fluid phenomenon. Smoke, repeatedly used throughout the study, can basically be defined as a hot gas in a surrounding medium.

3.1 Proposed System

Having said many rudimentary things about fluids, now we can focus on smoke, smoke itself. It is important to emphasize here that our system follows a similar principle of Jos Stam's stable fluids [20]. Unlike the solver presented in [20] which computes results based on the Fast Fourier Transform, our implementation uses a sparse linear solver. Let's remember the equations 2.1 and 2.2, and manipulate them a little bit to obtain our stable fluid.

Remember that we have,

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + \nu \nabla \cdot \nabla \vec{u}$$

$$\nabla \cdot \vec{u} = 0$$

Let's change the order of the items in the momentum equation to have an

equation for the velocity field, e.g.:

$$\frac{\partial \vec{u}}{\partial t} = -\vec{u} \cdot \nabla \vec{u} - \frac{1}{\rho} \nabla p + \nu \nabla \cdot \nabla \vec{u} + \vec{g} \tag{3.1}$$

The second equation remains the same,

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

In an ideal fluid, these equations create a relation between the acceleration of fluid and the gradient of pressure. A single equation for the velocity can be achieved by combining 3.1 and 3.2 [17]. Helmholtz-Hodge Decomposition (see Chapter 5 for the details of Helmholtz-Hodge Decomposition) states that any vector field, $\vec{w}$, can be decomposed into the form:

$$\vec{w} = \vec{u} + \nabla q \tag{3.3}$$

where $\vec{u}$ is divergence free: $\nabla \vec{u} = 0$ and q is a scalar field. Any vector field can be described as the sum of a mass conserving field and a gradient field. This outcome allows us to define an operator $\mathbf{D}$ which projects any vector field w onto its divergence free part $\vec{u} = \mathbf{D}\vec{w}$. This operator can be implicitly defined by multiplying both sides of the equation 3.3 by a divergence operator (See Chapter 5 for the details of divergence operator):

$$\nabla \cdot \vec{w} = \nabla^2 q$$

This equation is a Poisson equation (See Chapter 5 for the details of Poisson equation) for the scalar field q. A solution to this equation is used to compute the projection $\vec{u}$:

$$\vec{u} = \mathbf{D}\vec{w} = \vec{w} - \nabla q$$

If we apply this projection operator on both sides of 3.1, we get a single equation for velocity field:

$$\frac{\partial \vec{u}}{\partial t} = -\vec{u} \cdot \nabla \vec{u} + \nu \nabla \cdot \nabla \vec{u} + \vec{g}$$

Since the gravity is not the only outer force, let's change the $\vec{g}$ term to a general force term f governing gravity, vorticity confinement and buoyancy.

$$\frac{\partial \vec{u}}{\partial t} = -\vec{u} \cdot \nabla \vec{u} + \nu \nabla \cdot \nabla \vec{u} + f \tag{3.4}$$

This equation is the basis of our smoke simulation and models how the velocity of a gas changes over time depending on advection ($\vec{u} \cdot \nabla \vec{u}$), external forces ($f$), and diffusion ($\nu \nabla \cdot \nabla \vec{u}$). It is solved with an initial state $u_0 = u(x, 0)$ by marching through the time with a time step Δt. Assuming to start from time t, we are trying to find a solution for time $t + \Delta t$. So, we start from $u(x, t)$ and we are trying to find a solution for $u(x, t + \Delta t)$. The transformation from $u(x, t)$ to $u(x, t + \Delta t)$ is acquired by a combination of the steps adding force, advection, diffusion and projection. The same principle is applied for solving density as well. The solution is detailed in the following subsections.

The general system can be described by the following pseudo code given in Algorithm 1:

Algorithm 1: Main loop for the proposed system.

Input: N/A
Output: N/A

```
1 initialize variables;
2 for each frame do
3     solve velocity;
4     solve density;
5     display
6 end
```

Figure 3.1 illustrates how the velocity is solved:

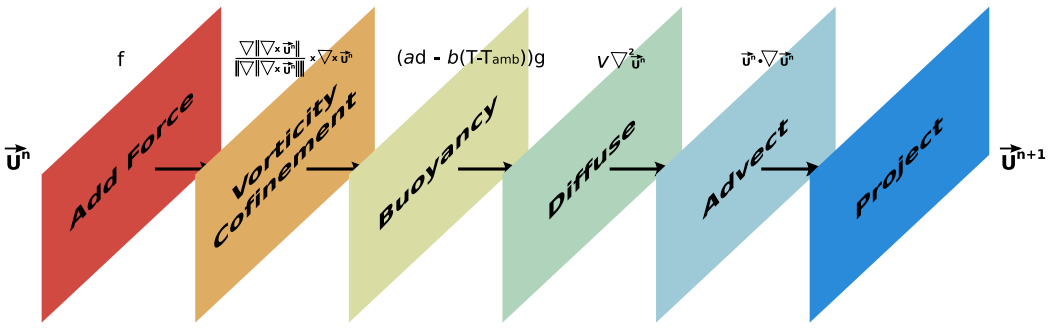


Figure 3.1: Velocity Solver

Figure 3.2 shows how the density is solved:

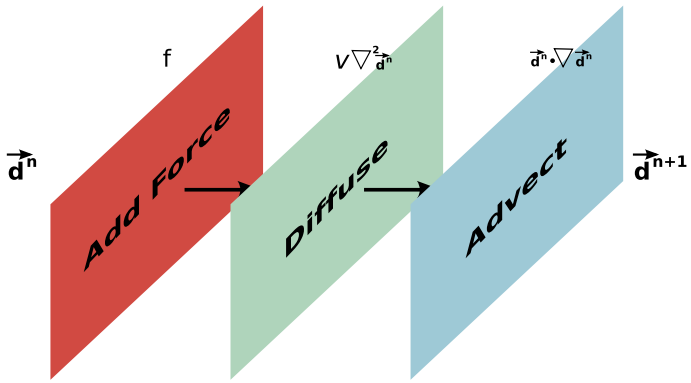


Figure 3.2: Density Solver

3.1.1 Thermal Buoyancy

In physics, buoyancy is an upward acting force, caused by fluid pressure [2]. Archimedes of Syracuse, the founder of buoyancy principle, states that "any object, wholly or partially immersed in a fluid, is buoyed up by a force equal to the weight of the fluid displaced by the object" [2].

In order to calculate this force, we need two main variables: Temperature, T and smoke concentration, d. It is important to emphasize that if a hot gaseous element is surrounded by cooler elements, the gas will rise (or move against gravity in cases of interest to us).

Calculate the thermal buoyancy force as follows:

$$F_{buoy} = (\alpha d - \beta\,(T - T_{amb}))\vec{g}$$

where $\vec{g}$ is the gravity. The constants α and β are positive with appropriate (physically meaningful) units. T is the temperature at the current cell, T_{amb} is the average temperature of the fluid grid. The density, d, provides a mass that counteracts the buoyancy force. In a simplified implementation, we say that the temperature is synonymous with density (since smoke is hot) because there are no other heat sources. So, we might just use the density field instead of a new, separate temperature field in our implementation.

3.1.2 Grid Structure

In this study, since we use an Eulerian approach instead of a Lagrangian one, we need a grid structure to represent our system. The grid is an $n \times n$ square grid in 2D or an $n \times n \times n$ cube in 3D. The velocity field is defined at the center of each cell as shown in Figure 3.3. Although, in previous studies the velocity was defined at the boundary of the cells [12], cell-centered cell is more advantageous since it is straightforward to implement. For simplicity and efficiency, we allocate two grids for each component of the velocity and scalar fields. At each time step, one grid holds the solution from the previous step and swapped and cleared accordingly.

3.1.3 Turbulence, Vorticity and Vorticity Confinement

In this study, some methods are used just aiming to capture more of the finescale swirly motion characteristic of turbulence (vorticity). This strategy is far from a scientific examination of turbulence, and in fact scientific work on the subject tends to concentrate on averaging over the details of turbulent velocity fields whereas we want to acquire those details as cheaply as possible, although they fall short of accuracy.

The smoke is drawn into adjacent regions of greater velocity (or lower pressure) resulting a swirling-like motion, a characteristic of turbulent flow, called vorticity. Vorticity, $\vec{\omega}$, can be calculated as follows:

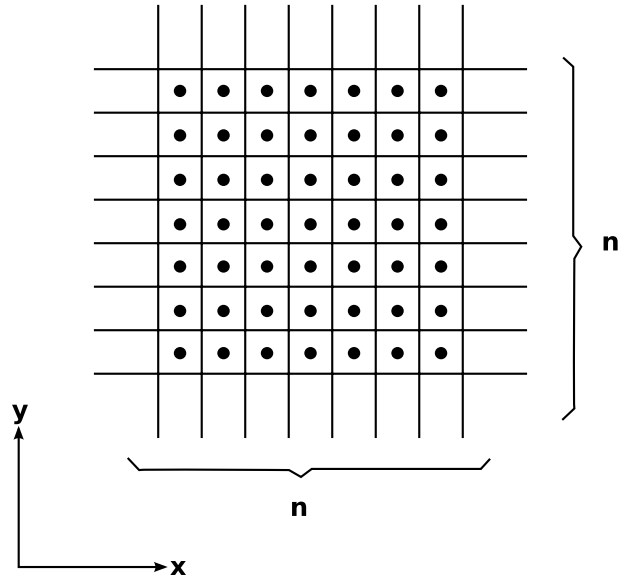
$$\vec{\omega} = \nabla \times \vec{u}$$

What it represents is, basically, how much the velocity field rotates around a point (See Chapter 5 for more information about curl operator). In three dimensions:

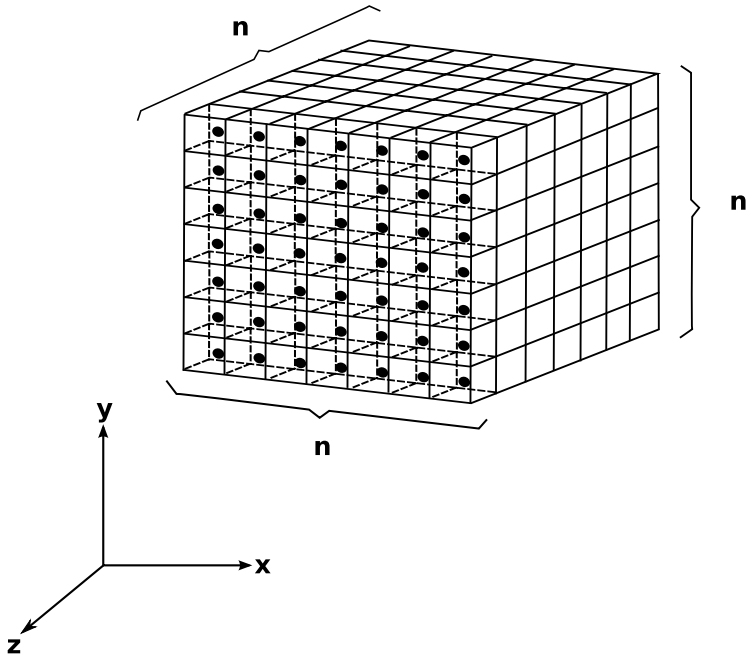
$$\vec{\omega} = \nabla \times \vec{u} = \left(\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}, \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}, \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \right)$$

In two dimensions, it reduces to a scalar value:

$$\vec{\omega} = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$$



(a) 2D grid structure



(b) 3D grid structure

Figure 3.3: Grid Structure

However in the calculation of *vorticity equation* (not shown here), the flow blows and some of the vorticity is lost. In order to add some of the vorticity back, a new technique, vorticity confinement is introduced.

The vorticity confinement technique was developed by Steinhoff and Underhill [24] and is a modification of Navier-Stokes equations by a term that tries to preserve vorticity. Fedkiw et al. [23] introduced this technique to computer graphics. The underlying idea is to detect where vortices are located and add a body force to boost the rotational motion around each vortex or cell. The calculation of vorticy confinement is given below:

$$F_{vortCof} = \vec{N} \times \vec{\omega}$$

where $\vec{\omega}$ is the curl and N is the vector pointing to the vortex (or cell) center, e.g.:

$$\vec{N} = \frac{\nabla \|\vec{\omega}\|}{\|\nabla \|\vec{\omega}\|\|}$$

Here are some frames of a smoke video of which I personally shot. Smoke source is a cigarette and it is in a surrounding medium. There is a light source at the top as well. You may see the examples of vorticity confinement in Figures 3.5 and bouyancy and diffusion effects in 3.4 and 3.5.

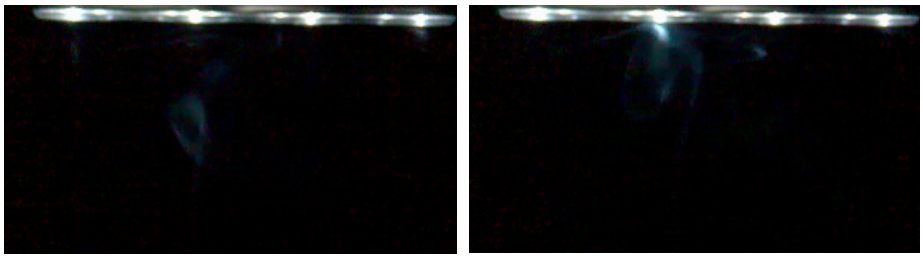


Figure 3.4: Two frames of a real-life smoke video

3.1.4 Time Steps

It is of great importance to choose the right time step size in creating animations. The underlying idea of this primary concern is to assure that the numerical

Figure 3.5: Another two frames of a real-life smoke video

method is stable. While the Eulerian approaches are conditionally stable, "the semi-Lagrangian approach is unconditionally stable, e.g. no matter how big the time step is, the approach never blows up" [8].

The approach never blows up because the new value is always interpolated from old values. Since the interpolation techniques (linear, bilinear, trilinear) always produce "values that lie between the values interpolated from" [8], the interpolated value always stays bounded.

However, in practice it is suggested by Fedkiw et. al. that "an appropriate strategy is to limit time step (Δt) so that the furthest a particle trajectory is traced is five grid cell widths:

$$\Delta t \leq \frac{5\Delta t}{u_{max}}$$

where u_{max} is an estimate of the maximum velocity in the fluid, e.g. the maximum velocity in that current cell" [11].

In this study, the time step is a variable that can be changed by the user. It is set to 0.2 as default.

3.1.5 Add Force-Source

This step is all about solving the f term in the equation 3.4. There are some methods to solve it, such as forward Euler and Runge-Kutta (see Chapter 5 for details). In this study, we used forward Euler, defined as:

$$\dot{u}(x) = \ddot{u}(x) + \Delta t f(x,t)$$

This step can also be called as adding source because what we are trying to do here is to add velocity, density sources, as well as add in vorticity confinement and buoyancy forces. As illustrated in Figure 3.1 and 3.2, this step is called while adding sources or after the calculation of buoyancy and vorticity confinement. Pseudo code for this step is given in Algorithm 2:

Algorithm 2: Adding source algorithm.

```
Input: input array
Output: output array
for each input array element do
    output array element = Δt× input array element;
end
```

3.1.6 Semi-Lagrangian Advection

Advection (modeled as $\vec{u} \cdot \nabla \vec{u}$ in equation 3.4) is the transfer of heat or matter by the flow of a fluid. In the case of smoke, the density and velocity fields are advected by using the Semi-Lagrangian scheme. Semi-Lagrangian advection was introduced to graphics by Jos Stam [20]. The method uses a regular (Eulerian) grid, just like finite difference methods. The idea is to calculate the point where a parcel is originated at every time step. An interpolation scheme is then utilized to estimate the value of the dependent variable at the grid points surrounding the point where the particle originated. The following procedure illustrates how the velocity field is advected. Just like velocity fieled, the same principle is applicable for the density field.

Let's assume that a hypothetical particle moves from position $\vec{x}_P$ to $\vec{x}_G$ with an old velocity q_P^n. When that hypothetical particle ends up at the position $\vec{x}_G$, it will have an velocity of q_G^{n+1}. The question is to figure out q_G^{n+1} (see Figure 3.6). Mathematically, the imaginary "particle moves according to the simple ordinary differential equation:" [8]

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{u}(\vec{x})$$

and the time step is Δt. To find the particle's start point, we can use the forward

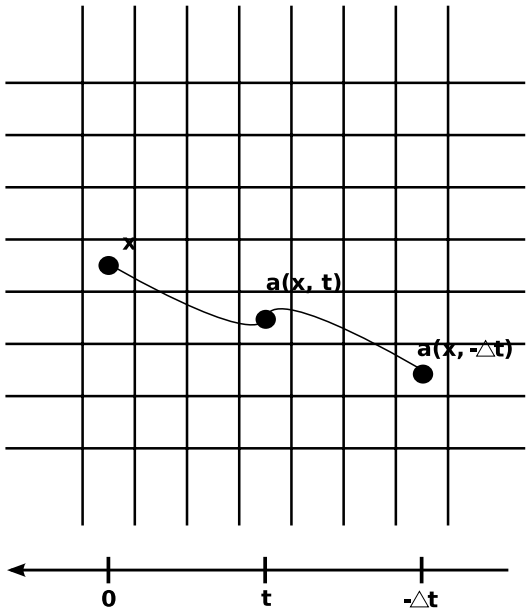


Figure 3.6: Advection: Tracing back in time

Euler method:

$$\vec{x}_P = \vec{x}_G - \Delta t \vec{u}(\vec{x})$$

It is important to bear in mind that there is no particle around, it is purely hypothetical. Since we are using Langrangian method to do an Eulerian calculation, the method is called semi-Langrangian advection.

We know that the imaginary particle has started this journey in the position $\vec{x}_P$. In order to calculate the velocity value of that particle $\vec{x}_P$, we need to interpolate that value from nearby grid points. To do so, bilinear (for 2D) or trilinear (for 3D) interpolation can be used. Let's describe the process for one dimension (using linear interpolation) to have an insight.

For the grid point x_i that lies on the interval $[x_j, x_{j+1}]$, the imaginary particle is traced back to $x_P = x_i - \Delta t u$ and $\alpha = (x_P - x_i)\Delta x$. α represent "the fraction of the interval the point lands in" [8]. So, applying the linear interpolation, we have:

$$q_P^n = (1 - \alpha) q_j^n + \alpha q_{j+1}^n$$

So, as a result the velocity field is interpolated from the nearby grid points. As stated earlier, the same principle is used to calculate density advection. To be

more informative, let us give the pseudo algorithm of the advection function (Algorithm 3):

Algorithm 3: Advection algorithm.

Input: Previous timestep array

Output: output array

1 Start with an input array from the previous timestep and an output array **for** *all grid cells* **do**

2 Trace the cell's center position backwards through the velocity field;

3 Interpolate the value from the grid of the previous timestep;

4 Assign this value to the current grid cell;

5 **end**

3.1.7 Diffusion

Diffusion is the intermingling of substances by the natural movement of their particles. It is modeled as $\nu \nabla \cdot \nabla \vec{u}$ in Navier-Stokes equation, where ν is the kinematic viscosity, $\nabla \cdot \nabla$ or ∇^2 is the Laplacian operator (see Chapter 5 for details). The equation is in fact:

$$\frac{\partial \vec{u}}{\partial t} = \nu \nabla^2 u = \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right)$$

To solve this equation, we might use a finite difference approximation, which is:

$$\begin{aligned} \frac{d_{x,y,z}^{t+\Delta t} - d_{x,y,z}^{t}}{\Delta t} &= \nu \times \left\{ \frac{d_{x-1,y,z}^{t+\Delta t} - 2d_{x,y,z}^{t+\Delta t} + d_{x+1,y,z}^{t+\Delta t}}{\Delta x^2} \right. \\ &+ \frac{d_{x,y-1,z}^{t+\Delta t} - 2d_{x,y,z}^{t+\Delta t} + d_{x,y+1,z}^{t+\Delta t}}{\Delta y^2} \\ &+ \left. \frac{d_{x,y,z-1}^{t+\Delta t} - 2d_{x,y,z}^{t+\Delta t} + d_{x,y,z+1}^{t+\Delta t}}{\Delta z^2} \right\} \end{aligned}$$

However, this approach blows up for large time steps. In order to achieve unconditional stability, we need to solve this equation going backwards in time.

Let the grid spacing be $(\Delta x, \Delta y, \Delta z)$ in 3D. By re-arranging this expression, we isolate the terms to time t on the left side and terms to time $t + \Delta t$:

$$d^t_{x,y,z} = (1+6\beta)d^{t+\Delta t}_{x,y,z} - \beta\left(d^{t+\Delta t}_{x-1,y,z} + d^{t+\Delta t}_{x+1,y,z} + d^{t+\Delta t}_{x,y-1,z} + d^{t+\Delta t}_{x,y+1,z} + d^{t+\Delta t}_{x,y,z-1} + d^{t+\Delta t}_{x,y,z+1}\right)$$

where

$$\beta = \frac{\nu \Delta t}{\Delta x^2}, \Delta x = \Delta y = \Delta z$$

In order to solve this, we must arrange this into a system of linear equations:

$$A \cdot d^{t+\Delta t} = d^t$$

where we order the elements of the grid in (x, y, z) lexicographic order.

We need to solve for $d^{t+\Delta t}$ and each row in A looks like:

$$[\cdots - \beta \cdots - \beta \cdots - \beta \cdots (1 + 6\beta) \cdots - \beta \cdots - \beta \cdots - \beta \cdots]$$

The linear system of equations:

$$A \cdot d^{t+\Delta t} = d^t$$

is sparse, symmetric and positive definite. This means that the system can be solved efficiently using iterative techniques such as Gauss-Seidel relaxation method (see chapter 5 for details).

3.1.8 Projection

This is the last step of our simulation. It is called projection because in linear algebra, "projection is a special type of linear operator such that if you apply it twice, you get the same result as applying it once" [8]. For instance, a matrix D is a projection if $D^2 = D$. Pysically, our transformation from $u(x, t)$ to $u(x, t + \Delta t)$ is a linear projection as well.

The aim is to use projection to make the velocity a mass conserving, incompressible field. This is achieved through a Helmholtz-Hodge Decomposition (see Chapter 5 for details). First we calculate the divergence field of our velocity using

the mean finite difference approach, and then apply the linear solver to compute the Poisson equation and obtain a "height" field. At the end, we subtract the gradient of this field to obtain our mass conserving velocity field.

Projection will "subtract off the pressure gradient from the intermediate velocity field $\vec{u}$" [8]. Let's describe it mathematically:

Helmholtz-Hodge Decomposition states that any vector field can be uniquely decomposed into the form:

$$\vec{u} = \vec{u}^{n+1} + \Delta t \frac{1}{\rho} \nabla p \tag{3.5}$$

where $\vec{u}^{n+1}$ is divergence free, e.g. $\nabla \cdot \vec{u}^{n+1} = 0$. We can define a projection operator by taking the divergence of both side of the equation 3.5.

We will get:

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

We know that $\nabla \cdot \vec{u}^{n+1}$ is 0. So we have,

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

It is basically a Poisson equation of the form, $\nabla^2 \varphi = f$.

We can discretize the divergence on our grid (3D) using mean finite difference approach as follows:

$$(\nabla \cdot \vec{u})_{i,j,k} \approx \frac{u_{i+1,j,k} - u_{i-1,j,k}}{\Delta x} + \frac{v_{i,j+1,k} - v_{i,j-1,k}}{\Delta x} + \frac{w_{i,j,k+1} - w_{i,j,k-1}}{\Delta x}$$

and

$$\begin{aligned} \frac{1}{\Delta x} \times \{ \quad & \left(u_{i+1,j,k} - \Delta t \tfrac{1}{\rho} \tfrac{p_{i+1,j,k} - p_{i,j,k}}{\Delta x}\right) - \left(u_{i-1,j,k} - \Delta t \tfrac{1}{\rho} \tfrac{p_{i,j,k} - p_{i-1,j,k}}{\Delta x}\right) \\ + \quad & \left(v_{i,j+1,k} - \Delta t \tfrac{1}{\rho} \tfrac{p_{i,j+1,k} - p_{i,j,k}}{\Delta x}\right) - \left(v_{i,j-1,k} - \Delta t \tfrac{1}{\rho} \tfrac{p_{i,j,k} - p_{i,j-1,k}}{\Delta x}\right) \\ + \quad & \left(w_{i,j,k+1} - \Delta t \tfrac{1}{\rho} \tfrac{p_{i,j,k+1} - p_{i,j,k}}{\Delta x}\right) - \left(w_{i,j,k-1} - \Delta t \tfrac{1}{\rho} \tfrac{p_{i,j,k} - p_{i,j,k-1}}{\Delta x}\right) \\ \} = 0 \end{aligned}$$

which leads to:

$$\begin{aligned} \frac{\Delta t}{\rho} & \left(\frac{6p_{i,j,k}-p_{i+1,j,k}-p_{i,j+1,k}-p_{i,j,k+1}-p_{i-1,j,k}-p_{i,j-1,k}-p_{i,j,k-1}}{\Delta x^2}\right) \\ = & -\left(\frac{u_{i+1,j,k}-u_{i-1,j,k}}{\Delta x}+\frac{v_{i,j+1,k}-v_{i,j-1,k}}{\Delta x}+\frac{w_{i,j,k+1}-w_{i,j,k-1}}{\Delta x}\right) \end{aligned}$$

This is a large system of linear equations and it needs to be solved using a linear solver. Assuming that the system is into the matrix form $Ap = b$, with unknown p, we solve this system again with Gauss-Seidel method. Our aim is to find $\vec{u}^{n+1}$. We already know $\vec{u}$, we solve the linear system and acquire p. As a last step, we subtract the gradient of p to obtain our mass conserving velocity field, e.g.:

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

3.1.9 Boundary Conditions

Getting the boundary conditions correct is the most important thing in numerical simulations. In this study, we can define the boundary as a solid wall surrounding the smoke, e.g. we are releasing smoke in a closed container. Hence we are releasing it in a closed container, we are assuming that "smoke cannot flow into the solid or out of it" [8]. So, mathematically "the normal component of velocity field has to be zero:" [8]

$$\vec{u} \cdot \hat{n} = 0$$

Chapter 4

Implementation & Results

The underlying idea of this chapter is to give intuition to the readers about how the theoretical information given throughout the study is applied for smoke simulation.

XCode developer tool and MacBook Pro with 2.4 GHz Inter Core 2 Due processor, 2 GB 667 MHz DDR2 SDRAM, 256 MB NVIDIA GeForce 8600M GT graphics card have been used during the implementation of this study.

First of all, the simulation in 2D is implemented on both Java and C++. While in Java, applet is used for display purposes, in C++ OpenGL is used. Although no benchmark tests are applied for the comparison of Java and C++, it is obvious that C++ is faster than Java whilst using the simulation. The main reason of the obvious result is the different techniques used in displaying. Eventually, Java applets are slower than OpenGL Api running on C++. Nevertheless, implementation was easier and much faster in Java than C++.

For 3D simulation, C++ is used for the implementation. OpenGL with SDL library and X Window System library is used for display purposes. While SDL library serves for mouse interaction, Xlib provides the window management. Both libraries are widely used in the industry nowadays.

The algorithm below is common for both two and three-dimensional space.

The main loop of the simulation is given in Algorithm 4:

Algorithm 4: Main loop for the proposed system.

Input: N/A

Output: N/A

```
1 initialize variables;
2 for each frame do
3     solve velocity;
4     solve density;
5     display
6 end
```

There are basically three important parts of this main loop. The first one is solving velocity of which the details are given in Chapter 3. The second one is solving density of which the details are again given in Chapter 3. The last one is display where the results are shown in this chapter. To be more descriptive, here is given the pseudo codes of both solving velocity and density.

Velocity is solved by using Algorithm 5:

Algorithm 5: Velocity solving algorithm

Input: N/A

Output: N/A

```
1  add source/force for each dimension of velocity;
2  solve vorticity confinement for velocity;
3  add source/force for each dimension of velocity;
4  solve buoyancy;
5  add buoyancy force to y dimension of velocity;
6  swap arrays for economical memory use;
7  calculate diffusion for velocity;
8  project;
9  swap arrays for economical memory use;
10 advect for all dimensions;
11 project;
12 for all elements of velocity do
13     clear old values of velocity dimensions;
14 end
```

Density is solved by using Algorithm 6:

Algorithm 6: Density solving algorithm

Input: N/A

Output: N/A

```
1  add source/force for density;
2  swap array for economical memory use;
3  calculate diffusion for density;
4  swap array for economical memory use;
5  advect;
6  for all elements of density do
7      clear old values of density;
8  end
```

Secondly, there are basically two clusters of results in this study. The first one is 2D results, showing how the system works with different grid size and time

step in two-dimensional space (Figures 4.4 - Figures 4.10). The second cluster is 3D results, again showing how the system works with different grid size in three-dimensional space (Figures 4.11 - Figures 4.14). You may also take a look at how many frames are created per second according to the grid size in Figures 4.1, 4.2, 4.3, Tables 4.1, 4.2, 4.3. It is obvious from the tables that there is a rapid fall down with the increase in grid size.

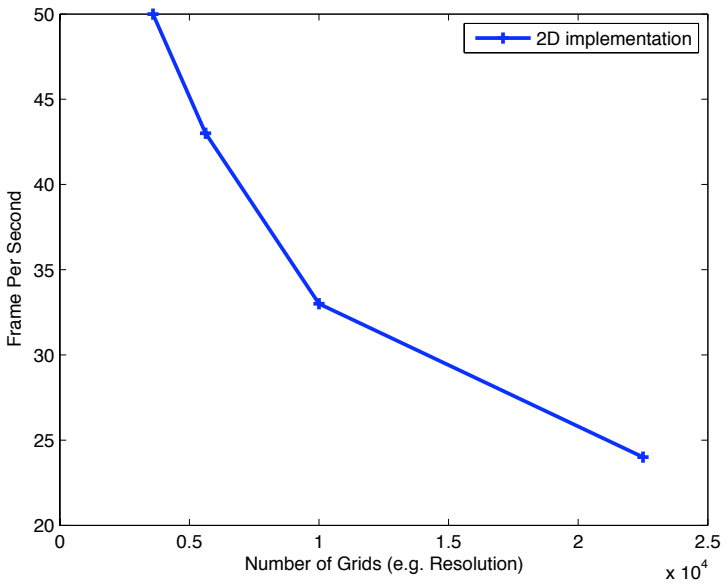


Figure 4.1: 2D implementation Resolution vs. FPS graph

Resolution	Frame Per Second in 2D
60x60	50
75x75	43
100x100	33
150x150	24

Table 4.1: Grid size vs. fps for 2D

Although it is obvious from the screenshots, it is important to say that 2D results are more realistic than 3D results. There might be few reasons to that conclusion. The most important reason is to make the simulation real time for 2D and 3D. Of course, having a three dimensional space makes it slower to simulate

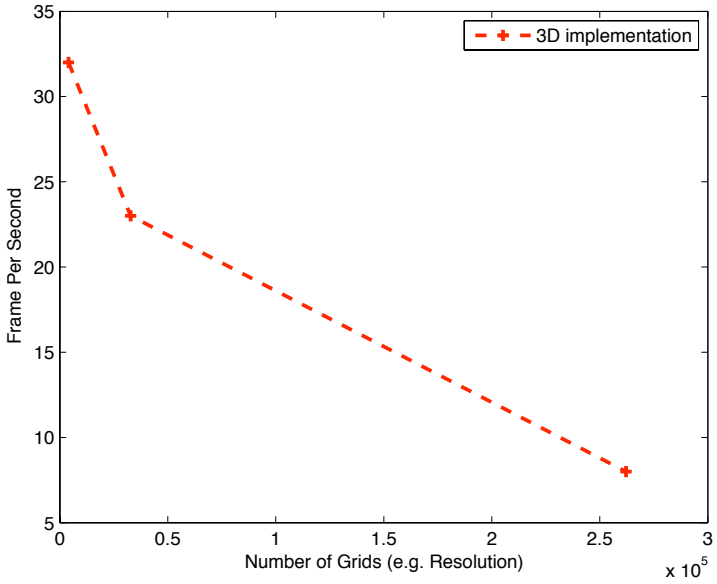


Figure 4.2: 3D implementation Resolution vs. FPS graph

Resolution	Frame Per Second in 3D
32x32x32	23
64x64x64	8

Table 4.2: Grid size vs. fps for 3D

and display. To make it faster, few approximations were introduced to some calculations. After some improvements on 3D system, such as increasing iteration number and correcting a bug in the implementation of how density was solved, the smoke is modelled more correct and realistic. The results are given in Figures 4.4 - 4.14:

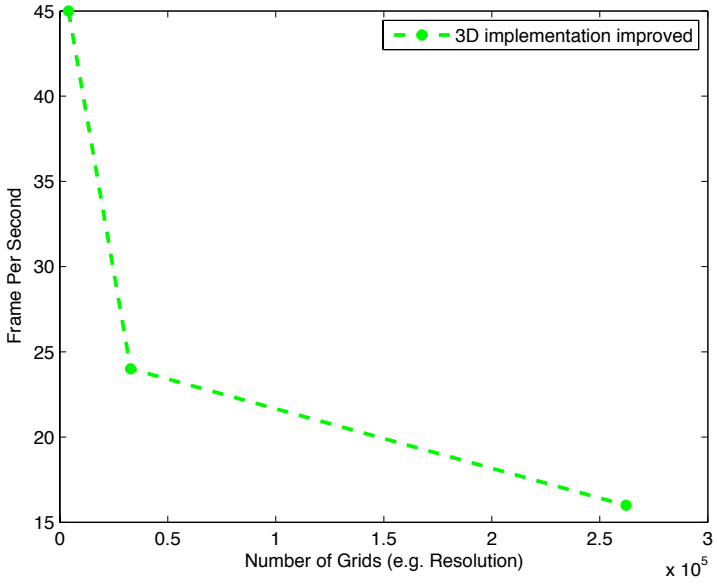


Figure 4.3: 3D implementation Resolution vs. FPS graph improved

Resolution	Frame Per Second in 3D (improved)
32x32x32	24
64x64x64	16

Table 4.3: Grid size vs. fps for 3D (improved)

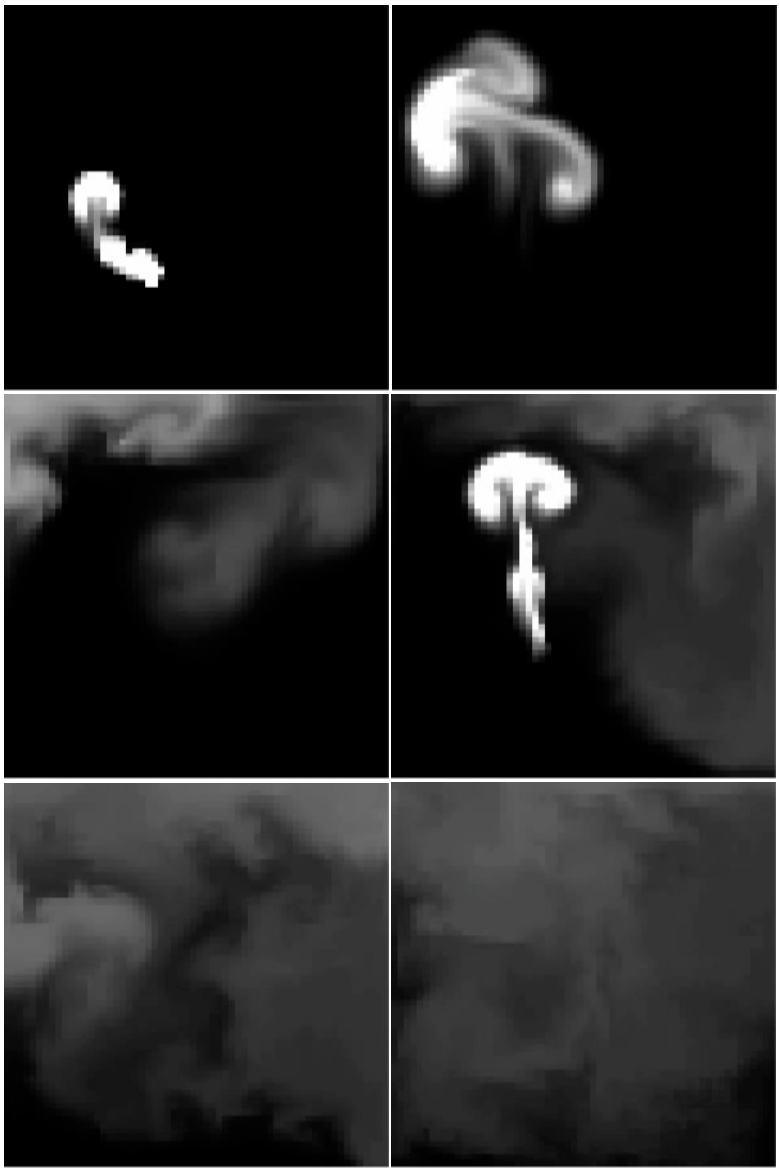


Figure 4.4: Smoke Simulation in 2D with time step 0.2 and grid size 60x60

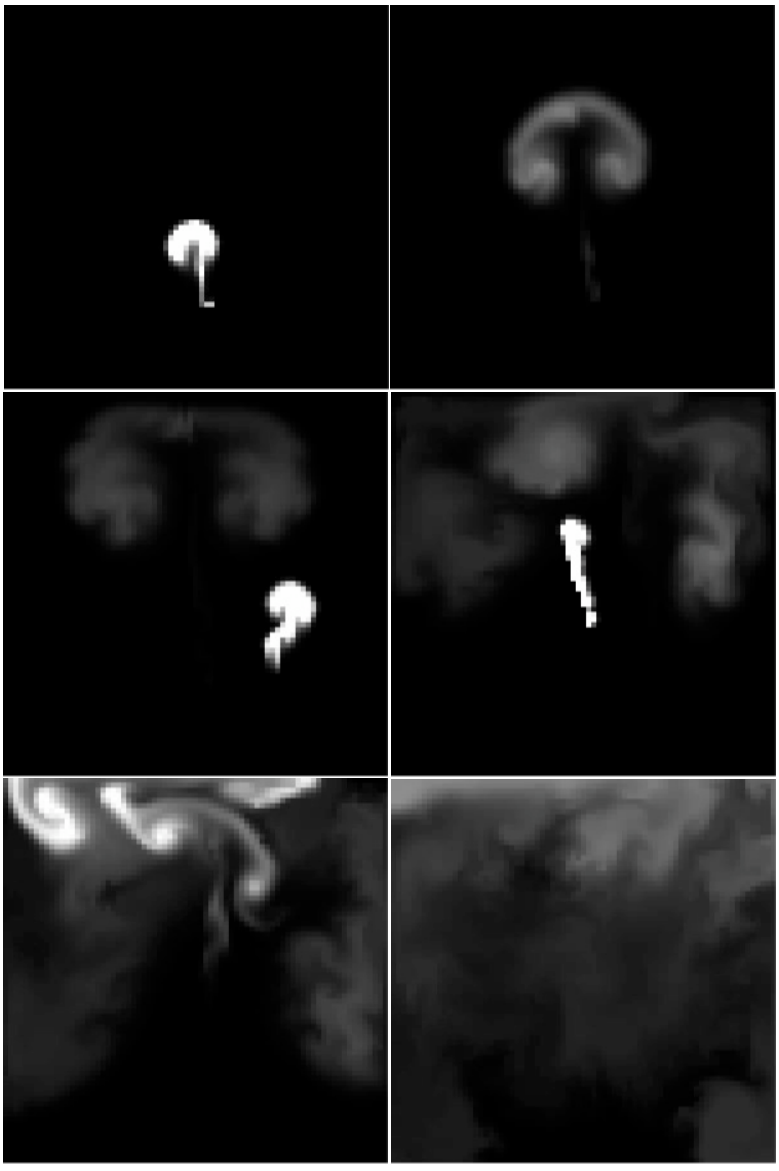


Figure 4.5: Smoke Simulation in 2D with time step 0.2 and grid size 75x75

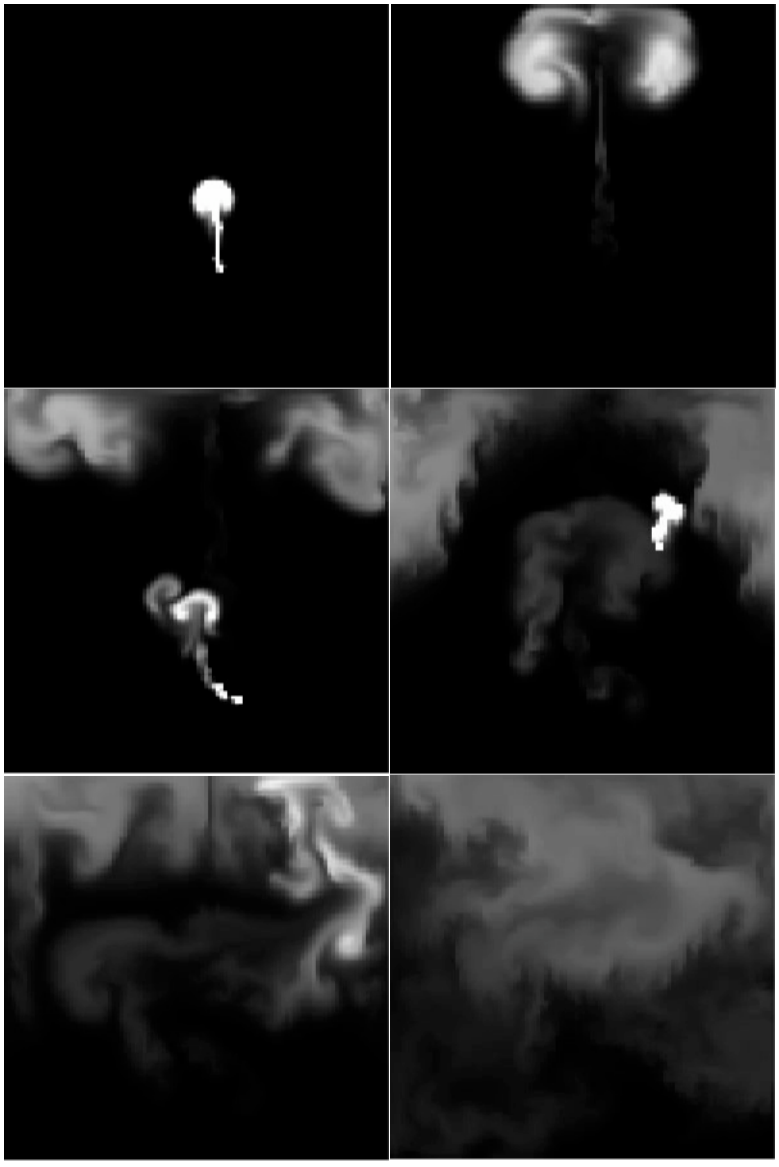


Figure 4.6: Smoke Simulation in 2D with time step 0.1 and grid size 100x100

Figure 4.7: Smoke Simulation in 2D with time step 0.2 and grid size 100x100

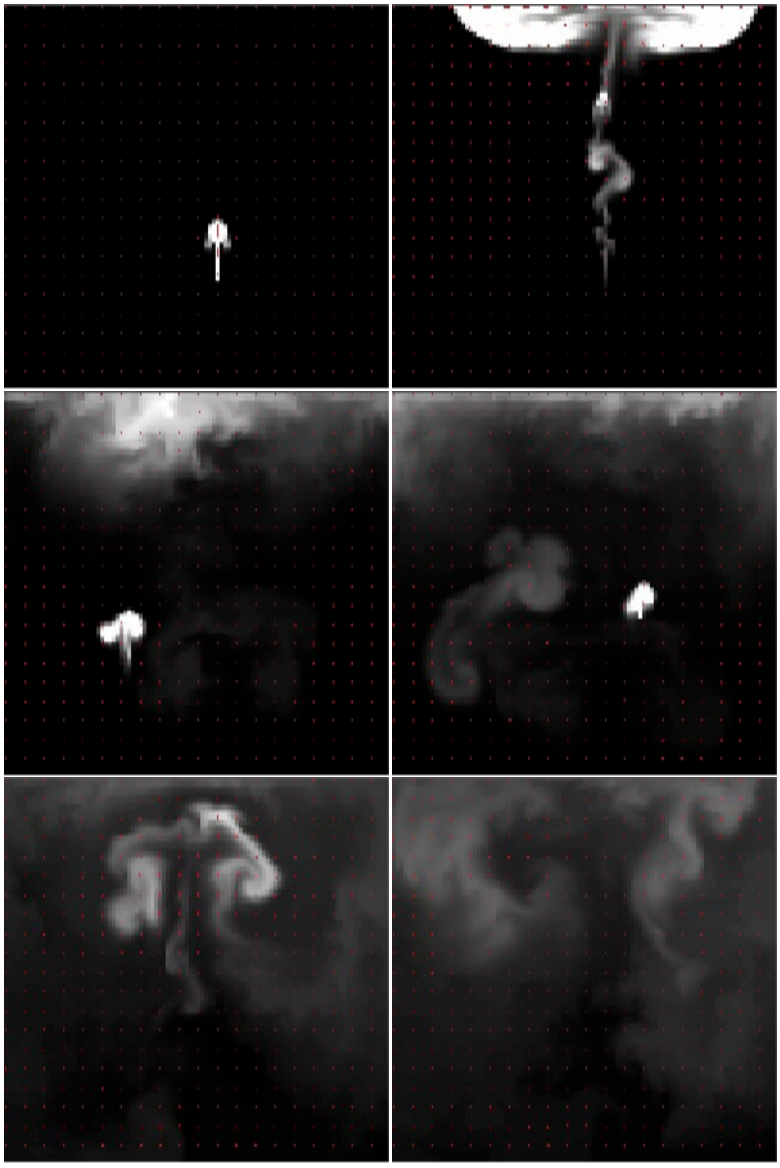


Figure 4.8: Smoke Simulation in 2D with time step 0.2 and grid size 100x100 with visible velocity field

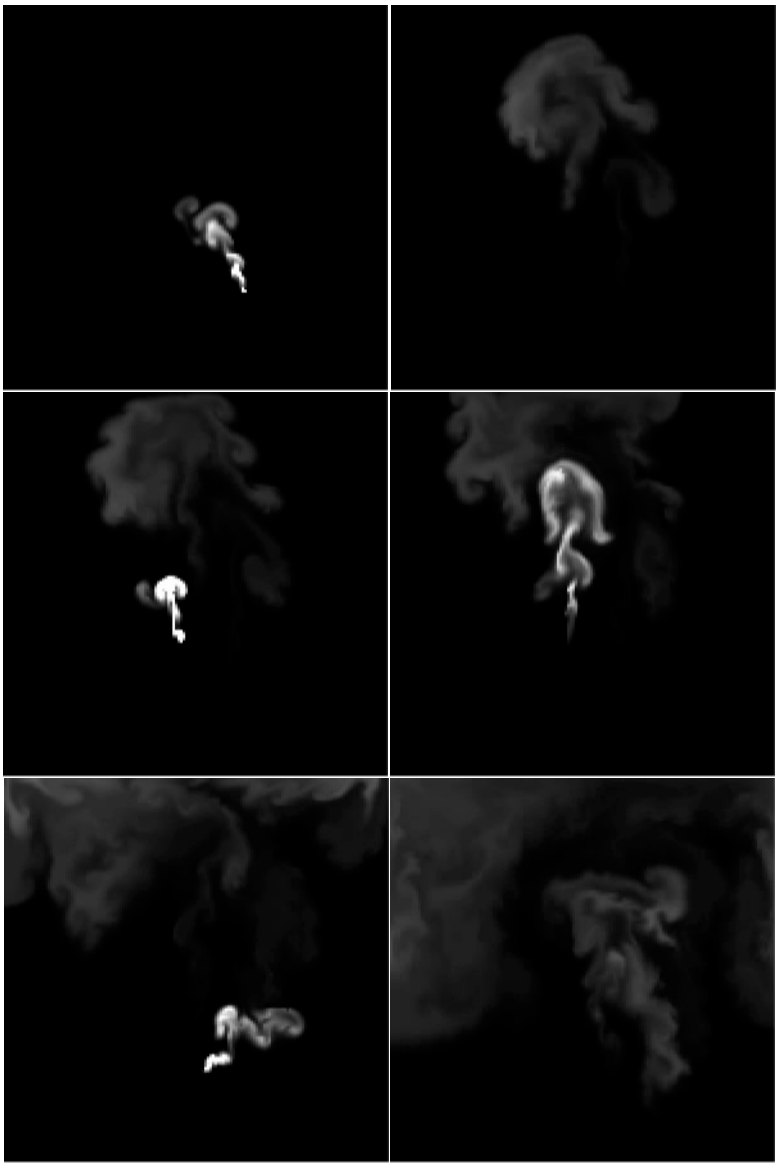


Figure 4.9: Smoke Simulation in 2D with time step 0.2 and grid size 150x150

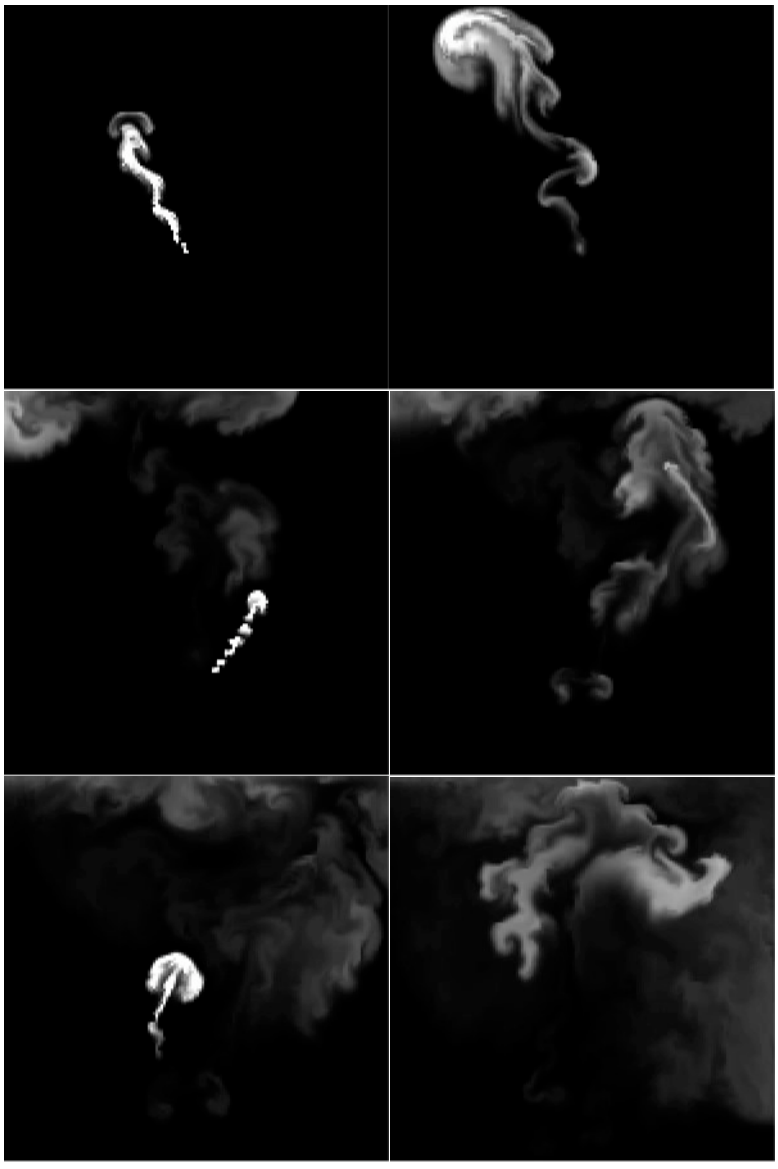


Figure 4.10: Smoke Simulation in 2D with time step 0.4 and grid size 150x150

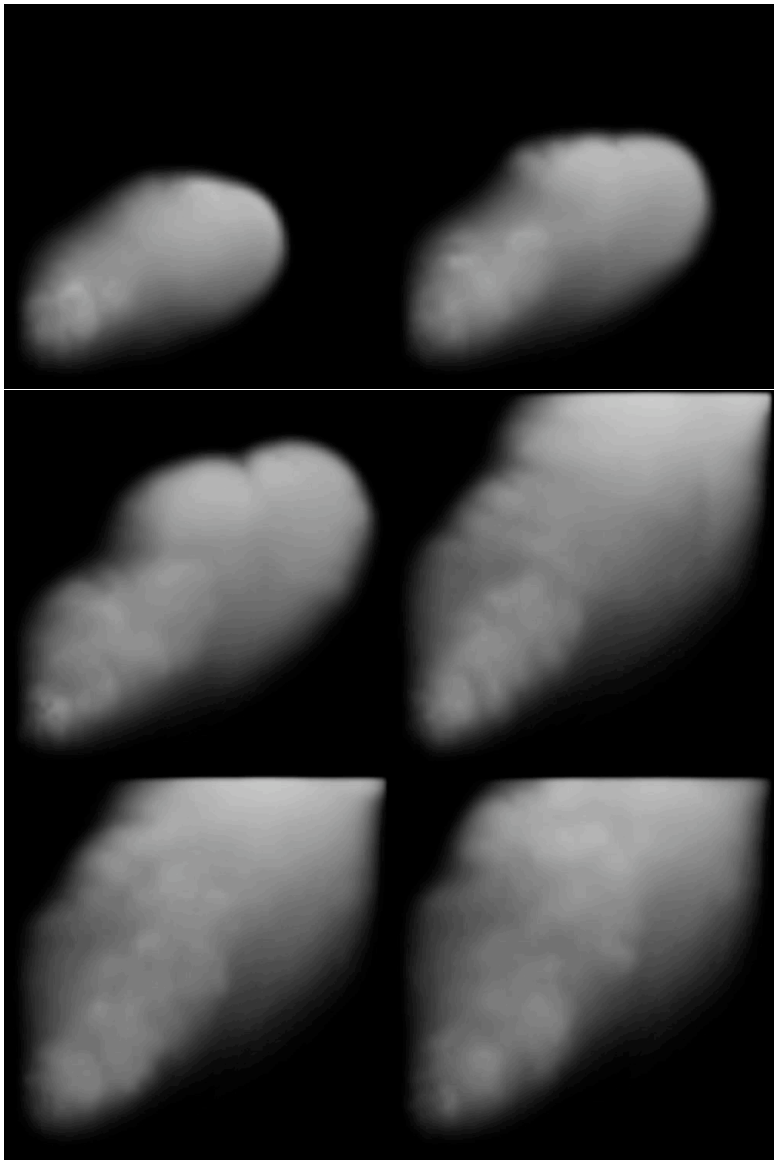


Figure 4.11: Smoke Simulation in 3D grid size 64x64x64

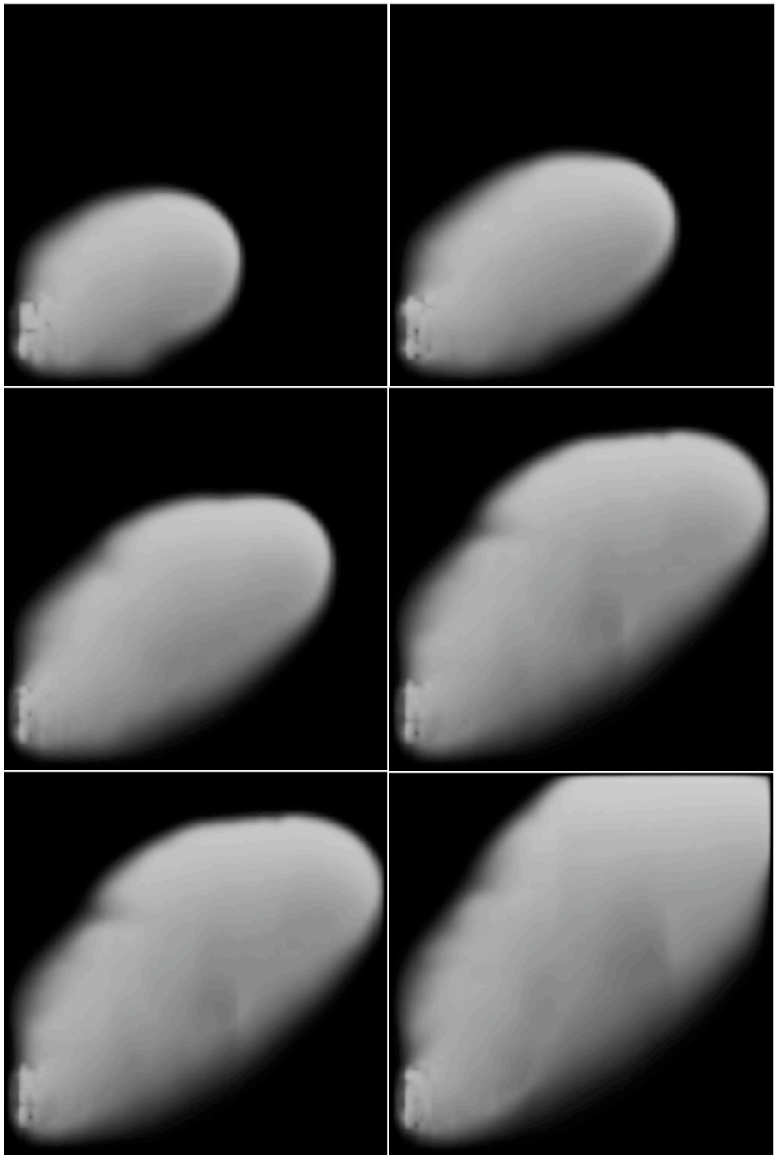


Figure 4.12: Smoke Simulation in 3D grid size 32x32x32

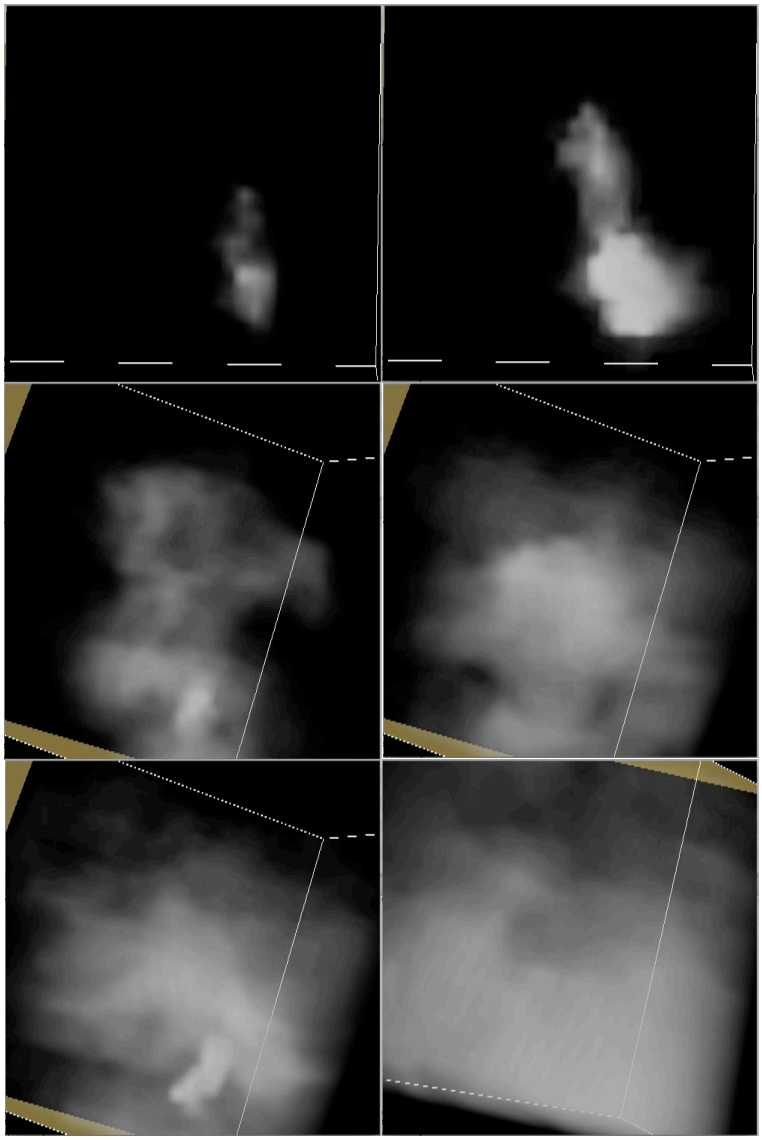


Figure 4.13: Smoke Simulation in 3D grid size 64x64x64, improved

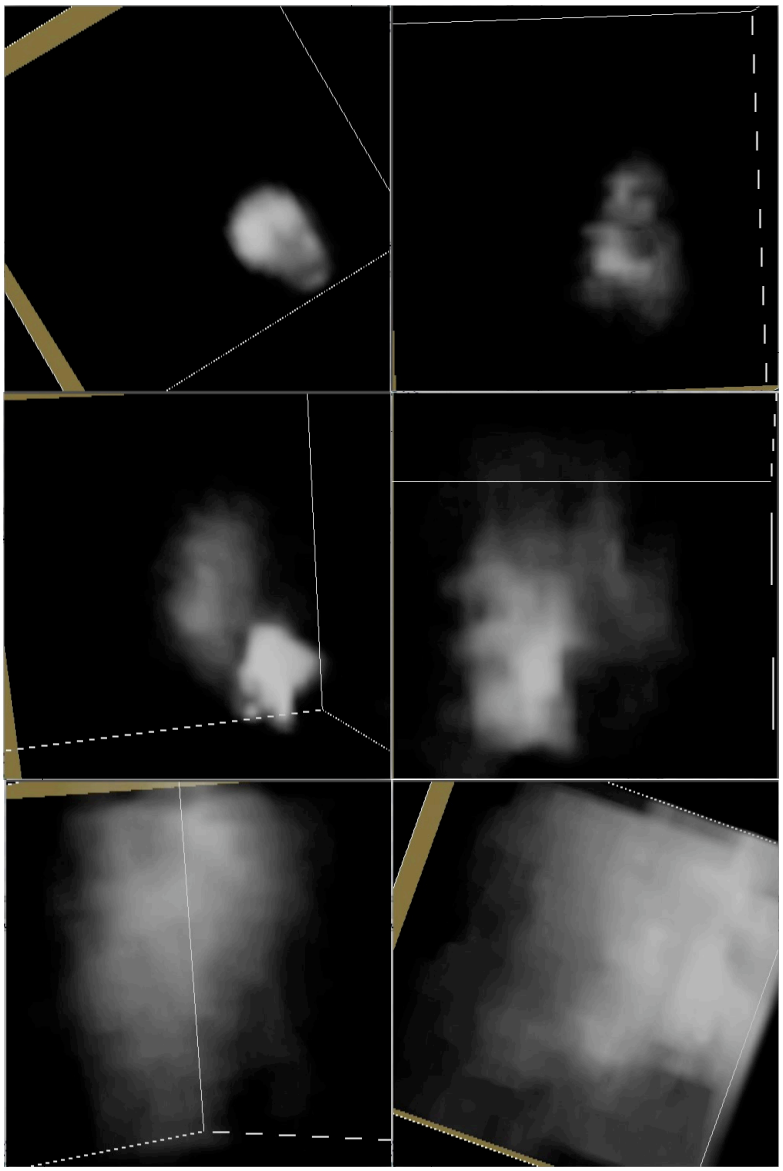


Figure 4.14: Smoke Simulation in 3D grid size 32x32x32, improved

Chapter 5

Conclusions

Modeling of natural phenomena such as smoke is a challenging problem in computer graphics. This is not surprising since the motion of gases is highly complex. Furthermore, modeling fluid behavior is of great importance for a broad range of areas from animation to engineering. On one side, due to the increase in the amount of animations in the film industry, there is a high demand to convincingly mimic the appearance of fluids. On the other side, it is of primary concern to simulate fluid-like behavior physically correct in engineering applications. Nevertheless, both majors require this phenomenon to be modeled effectively, as well as fast and easy.

In this thesis work, we propose an unconditionally stable, easy implemented real-time physics-based smoke simulation with vorticity confinement, solving Navier-Stokes equations with Lagrangian and implicit methods. The study is based on Jos Stam's Stable Fluids [20] and Fedkiw et al.'s Visual Simulation of Smoke [23]. While the solution is detailed throughout the research, the work also focuses on the comprehension of fluid dynamics by providing background information about Navier-Stokes equations, how they are derived and used. One of the most important property of the proposed solution is its ability of adaptation to the other fluids as well as smoke. Another aspect is its modification to 2 and 3 dimensional space, satisfying the animator needs. The results shows that the implementation is simple yet quite powerful, whilst giving the sense of smoke to

the user.

One weak point, so to speak, of our system is the linear solver used in solving the equation in the form of $Ax = b$. As mentioned before, we used preconditioned conjugate gradient method, an iterative relaxation technique, Gauss-Seidel, in order to solve a sparse, symmetric and positive definite system. However, iterative methods, by their nature, suffer from approximations. In order to apply the method, one has to start from an initial guess to find successive approximations to the solution. Though, we think that, it has its own advantages and is a good choice for real-time systems.

Although 2D results are impressive and above satisfactory, 3D results seem inadequate. The most important reason of this conclusion is that of approximation errors. So as to have a real-time system in 3D, we had to introduce some approximations on 2D system. The main approximation is in the linear solver where number of iterations had to be reduced in order to stay in the boundaries of a real-time system. Another reason of dissatisfactory 3D results is that of avoiding expensive calculations. Using forward Eularian approach for solving the f term in the Navier-Stokes equation is cheap (computation-wise) but arouses unreality. On the other hand, more expensive methods such as fourth (or more) order Runge-Kutta, would slowdown the process but increase reality. This discussion would diverge to the main decision point in computer technology: the trade off between computation power and speed, where; would you like your application to be fast and unrealistic or slow but realistic?

2D system works 50 frames per second for 60x60 grid, where as 3D system works 8 frames per second for 64x64x64 grid. “Today, 24p (refers to a video format that operates at 24 frames per second) formats are being increasingly used for aesthetic reasons in image acquisition, delivering film-like motion characteristics” [1]. Eventually, our 3D system is slow for human use.

Moreover, our future extensions will focus on how to improve the effectiveness and performance of 3D results and we plan to implement a renderer for good visualization purposes.

Appendix A

Background: Basic Math and History

This chapter provides background information about the mathematical terminology used through the study and also gives a brief historical information about Navier-Stokes equations for those who are interested.

A.1 Gradient

The gradient operator, denoted as ∇, takes a function as an input and returns vector. It simply "takes all the spatial partial derivatives of the input function" [8]. Gradient of a scalar function (e.g. density) is a vector that points in the direction of maximal change.

In two dimensions,

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$$

In three dimensions,

$$\nabla f(x, y, z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right)$$

A.2 Divergence

The divergence operator, denoted as $\nabla\cdot$, takes a vector field and returns a scalar field. "It basically says how much the vector is converging or diverging at any point" [8].

In two dimensions,

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

In three dimensions,

$$\nabla \cdot \vec{u} = \nabla \cdot (u, v, w) = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}$$

A.3 Curl

The curl operator, denoted as $\nabla\times$, takes a vector and returns a scalar value in two dimension, a vector in three dimensions. "It calculates how much a vector field rotates around a point" [8].

In two dimensions,

$$\nabla \times \vec{u} = \nabla \times (u, v) = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$$

In three dimensions,

$$\nabla \times \vec{u} = \nabla \times (u, v, w) = \left(\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}, \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}, \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}\right)$$

A.4 Laplacian

The Laplacian is defined "as the divergence of the gradient" [8] of a scalar function. It is mostly denoted as $\nabla \cdot \nabla$. It measures how far a quantity is from the average around it.

In two dimensions,

$$\nabla \cdot \nabla f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

In three dimensions,

$$\nabla \cdot \nabla f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2}$$

A.5 Time Integration

Solving differential equations in time generally revolves around the same finite approach. For example, to solve the differential equation

$$\frac{\partial q}{\partial t} = f(q)$$

with initial conditions q(0) = q^0, we can approximate q at discrete times t^n, with q^n = q(t^n). The time step Δt is $\Delta t = t^{n+1} - t^n$. The process of time integration is determining the approximate values $q^1, q^2, q^3, ...$ in sequence.

There are few methods used in time integration and one of the easiest methods is forward Euler.

$$q^{n+1} = q^n + \Delta t f(q^n)$$

One of the most effective methods is Runge-Kutta.

Second-order Runge-Kutta is

$$q^{n+\frac{1}{2}} = q^n + \frac{1}{2}\Delta t f(q^n) q^{n+1} = q^n + \Delta t f(q^{n+\frac{1}{2}})$$

Third-order Runge-Kutta is

$$k_1 = f(q^n)$$

$$k_2 = f(q^n + \frac{1}{2}\Delta t k_1)$$

$$k_3 = f(q^n + \frac{3}{4}\Delta t k_2)$$

$$q^{n+1} = q^n + \frac{2}{9}\Delta t k_1 + \frac{3}{9}\Delta t k_2 + \frac{4}{9}\Delta t k_3$$

A.6 Gauss-Seidel

Gauss-Seidel is an iterative method used to solve a linear system of equations. For a given square system in the form of

$$Ax = b$$

where there are n linear equations with unknown x.

$$A = \begin{bmatrix} a_{11}a_{12}\cdots a_{1n} \\ a_{21}a_{22}\cdots a_{2n} \\ \vdots\vdots\ddots\vdots \\ a_{n1}a_{n2}\cdots a_{nn} \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Then A can be decomposed into a lower triangular component L, and a strictly upper triangular component U:

$$A = L_* + U$$

where

$$L_* = \begin{bmatrix} a_{11}0\cdots 0 \\ a_{21}a_{22}\cdots 0 \\ \vdots\vdots\ddots\vdots \\ a_{n1}a_{n2}\cdots a_{nn} \end{bmatrix}, U = \begin{bmatrix} 0a_{12}\cdots a_{1n} \\ 00\cdots a_{2n} \\ \vdots\vdots\ddots\vdots \\ 00\cdots 0 \end{bmatrix}$$

The system of linear equations can be written as:

$$L_*x = b - Ux$$

The Gauss-Seidel method is an iterative technique that solves the left hand side of this expression for x, using previous value for x on the right hand side. Analytically, this may be written as:

$$x^{k+1} = L_*^{-1}(b - Ux^{(k)})$$

A.7 Helmholtz-Hodge Decomposition

Fundamental theorem of vector calculus states that a velocity field V defined on some domain D can be uniquely decomposed in the form

$$V = u + \nabla p$$

where u is a divergent free velocity field and p is the pressure in the fluid.

A.8 Poisson Equation

Poisson Equation, named after French mathematician Simeon-Denis Poisson, is a partial differential equation used in many areas. The equation is defined as:

$$\nabla^2 \varphi = f$$

In three dimensions, it is

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \right) \varphi(x, y, z) = f(x, y, z)$$

Another form of Poisson equation is Laplace equation where $f = 0$, e.g:

$$\nabla^2 \varphi = 0$$

A.9 Brief History of Navier-Stokes Equations

Claude Louis Marie Henri Naviers name is associated with the famous Navier-Stokes equations that govern motion of a viscous fluid. He derived the Navier-Stokes equations in a paper in 1822. His derivation was however based on a molecular theory of attraction and repulsion between neighboring molecules. Euler had already derived the equations for an ideal fluid in 1755, which did not include the effects of viscosity. Navier did not recognize the physical significance of viscosity and attributed the viscosity coefficient to be a function of molecular spacing.

The equations of motion were rederived by Cauchy in 1828 and by Poisson in 1829. In 1843 Barre de Saint-Venant published a derivation of the equations that applied to both laminar and turbulent flows. However the other person whose name is attached with Navier is the Irish mathematician-physicist George Gabriel Stokes. In 1845 he published a derivation of the equations in a manner that is currently understood [5].

Bibliography

[1] 24p format, available at `http://en.wikipedia.org/wiki/24p`, 2010.

[2] Buoyancy definition, available at `http://en.wikipedia.org/wiki/Buoyancy`, 2010.

[3] Clay mathematics institute, the millennium problems, available at `http://www.claymath.org/millennium/Navier-Stokes_Equations/`, 2010.

[4] Density definition, available at `http://en.wikipedia.org/wiki/Density`, 2010.

[5] History of navier-stokes equations, available at `http://www.cfd-online.com/Wiki/Navier-Stokes_equations`, 2010.

[6] Navier-stokes equations, available at `http://en.wikipedia.org/wiki/Navier%E2%80%93Stokes_equations`, 2010.

[7] Viscosity definition, available at `http://en.wikipedia.org/wiki/Viscosity`, 2010.

[8] R. Bridson. *Fluid Simulation for Computer Graphics*. Ak Peters, Wellessey, Massachusetts, second edition, 2008.

[9] D. S. Ebert. Solid spaces: A unified approach to describing object attributes, 1996.

[10] J. S. R. Fedkiw and H. W. Jensen. Visual simulation of smoke. In *In SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer*

Graphics and Interactive Techniques, pages 15–22. ACM New York, NY, USA, 2001.

[11] N. Foster and R. Fedkiw. Practical animation of fluids. In *In SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 23–30. New York: ACM, 2001.

[12] N. Foster and D. Metaxas. Modeling the motion of a hot, turbulent gas. In *In SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 181–188. Reading, MA: Addision Wesley, 1997.

[13] J. Fromm. A method for computing non-steady, incompressible viscous fluid flows. In *Los Alamos Scientific Laboratory Report*, pages LA–2910, May 1963.

[14] J. Fromm and F. Harlow. Numerical solution of the problem of vortex street development. *Physics of Fluids*, 6:975–982, 1963.

[15] F. Harlow and J.Welch. Numerical calculation of timedependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 8:2182–2188, 1965.

[16] J. T. Kajiya and B. P. von Herzen. Ray tracing volume densities. In *In SIGGRAPH '84: Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, page 165174. ACM New York, NY, USA, July 1984.

[17] A. J. C. J. E. Marsden. *A Mathematical Introduction to Fluid Mechanics*. Springer-Verlag, New York, third edition, 2000.

[18] W. L. X. W. K. Mueller and A. Kaufman. The lattice-boltzmann method for simulating gaseous phenomena. *IEEE Transactions on Visualization and Computer Graphics archive*, 10:164–176, 2004.

[19] D. E. F. K. M. D. P. K. Perlin and S. Worley. *Texturing and Modelling: A Procedural Approach*. Morgan Kauffman, New York, second edition, 2003.

[20] J. Stam. Stable fluids. In *In SIGGRAPH '99: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, pages 121–128. New York: ACM, 1999.

[21] J. Stam. A simple fluid solver based on the fft. *Journal of Graphics Tools*, 6(1):43–52, 2001.

[22] J. Stam. Real-time fluid dynamics for games. In *Proceedings of the Game Developer Conference*, March 2003.

[23] R. F. J. Stam and H. W. Jensen. Visiual simulation of smoke. In *In SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 15–22. New York: ACM, 2001.

[24] J. Steinhoff and D. Underhill. Modification of the euler equations for "vorticity confinement": Application to the computation of interacting vortex rings. *Physics of Fluids*, 6:2738–2744, 1994.

[25] W. L. Z. F. X. Wei and A. Kaufman. Gpu-based flow simulation with complex boundaries. pages 747–764. Addison-Wesley, 2005.